Efficient Power and Thermal Management using Fine-grain Architectural Approaches in Multicores

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Juan Manuel Cebrián

Advisors

Juan Luis Aragón

Stefanos Kaxiras

Departamento de Ingeniería y Tecnología de Computadores

Facultad de Informática

Universidad de Murcia

June 2011

Diseño de Mecanismos de Grano Fino para la Gestión Eficiente de Consumo y Temperatura en Procesadores Multinúcleo

Memoria que presenta para optar al título de Doctor en Informática

Juan Manuel Cebrián González

Dirigida por los Doctores

Juan Luis Aragón Alcaraz
Stefanos Kaxiras

Departamento de Ingeniería y Tecnología de Computadores
Facultad de Informática
Universidad de Murcia

Junio de 2011

To my family and friends.
To the memory of my grandparents.
To my dear María.

UNIVERSIDAD
DE MURCIA

DEPARTAMENTO DE INGENIERÍA Y TECNOLOGÍA DE COMPUTADORES

D. Juan Luis Aragón Alcaraz, Profesor Titular de Universidad del Área de Arquitectura y Tecnología de Computadores en el Departamento de Ingeniería y Tecnología de Computadores de la Universidad de Murcia (España)

y

D. Stefanos Kaxiras, Profesor Titular de Universidad del Área de Arquitectura y Tecnología de Computadores en el Departamento de Ingeniería y Tecnología de Computadores de la Universidad de Uppsala (Suecia)

AUTORIZAN:

La presentación de la Tesis Doctoral titulada «*Diseño de Mecanismos de Grano Fino para la Gestión Eficiente de Consumo y Temperatura en Procesadores Multinúcleo*», realizada por D. Juan Manuel Cebrián González, bajo su inmediata dirección y supervisión, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia.

En Murcia, a 8 de Abril de 2011.

Fdo: Dr. Juan Luis Aragón Alcaraz

Fdo: Dr. Stefanos Kaxiras

UNIVERSIDAD DE
MURCIA

D. Antonio Fernando Gómez Skarmeta, Catedrático de Universidad del Área de Ingeniería Telemática y presidente de la Comisión Académica del Programa de Postgrado de Tecnologías de la Información y Telemática Avanzadas de la Universidad de Murcia, INFORMA:

Que la Tesis Doctoral titulada «*Diseño de Mecanismos de Grano Fino para la Gestión Eficiente de Consumo y Temperatura en Procesadores Multinúcleo*», ha sido realizada por D. Juan Manuel Cebrián González, bajo la inmediata dirección y supervisión de D. Juan Luis Aragón Alcaraz y de D. Stefanos Kaxiras, y que la Comisión Académica ha dado su conformidad para que sea presentada ante la Comisión de Doctorado.

En Murcia, a 8 de Abril de 2011.

Fdo: Dr. Antonio Fernando Gómez Skarmeta

Abstract

Thermal and power related issues are common in most modern microprocessors. We are not only talking about servers, but also mobile devices, desktop computers, laptop, GPUs, APUs, etc. For many years microprocessor design has been (and is) limited by power dissipation and temperature. Many studies refer to these key factors as the "Power Wall". Moreover, after 10 years of focusing on increasing the operating frequency of the transistors, engineers switched to increasing the number of processing cores on the same die promoting throughput and leaving individual core performance aside. Even today, this "power wall" is still limiting the number of cores that can be placed on the same die.
We can distinguish two main components in power dissipation: dynamic power, which is proportional to usage (every time we access a structure) and static power (or leakage), that is derived from gate leakage and subthreshold leakage currents that flow even when the transistor is not in use. Dynamic Voltage and Frequency Scaling (DVFS) and Clock Gating have been the most common and effective techniques to contain energy consumption. Unluckily, DVFS becomes less effective with every new generation of transistors, because it mainly relies on decreasing supply voltage and, in order to maintain the transistor's switching speed, threshold voltage must be reduced at the same rate. This reduction increases leakage power exponentially and when the building technology process drops below 65nm, leakage power becomes an important part of the total power dissipated by the processor.
Thermal and power related problems are usually detected by a Dynamic Thermal Management (DTM) mechanism and solved by limiting the amount of power the processor / core can consume (by establishing a power budget). However, most modern microprocessors do not have the means to accurately measure power in real time. Power is estimated based on performance counters over periods of thousands of cycles. In this Thesis we introduce the concept of Power Tokens. With Power Tokens we can measure energy at a higher resolution by adding to the base energy consumption of the instruction (i.e., all regular accesses to structures done by that instruction) a dynamic component that depends on the time it spends on the pipeline. Total energy can be estimated as the addition of the individual energy of all the instructions inside the processor's pipeline.
We analyze the performance, power, energy and accuracy impact of different power saving mechanisms to discover that individual mechanisms are not suitable to accurately match the established power budget with a reasonable performance penalty. In order to deal

with this problem we propose several microarchitecture-level techniques that dynamically combine individual techniques to obtain a more accurate power budget matching with a reasonable performance penalty. Our techniques for the single-core scenario include: power token throttling (PTT) a token-based approach that keeps track of the current power being dissipated by the processor and stalls fetch if the next instruction to be executed will make the processor go over the power budget; a basic block-level mechanism (BBLM), that keeps track of the energy consumed the last time the processor executed a basic block and uses that energy translated into tokens to select between different power-saving microarchitectural techniques; and finally a two-level approach where DVFS acts as a coarse-grain technique to lower the average power dissipation towards the power budget, while microarchitectural techniques focus on removing power spikes.

However, when evaluating these mechanisms in the CMP field we discovered that legacy techniques work properly for thread-independent and multi-programmed workloads, but not for parallel workloads. In the latter we noticed that synchronization points alter the optimal execution of the code, so it is no longer dictated by each individual core, but by the whole CMP. To enhance power saving mechanisms in the CMP field we introduce Power Token Balancing (PTB). This mechanism balances the power dissipated among the different cores using a power token-based approach while optimizing the energy efficiency. We can use power (seen as tokens) from non-critical threads for the benefit of critical threads.

Finally, and also in the multi-core scenario, we analyze the use of microarchitectural power budget techniques to reduce peak temperature in 3D die-stacked architectures. In particular, we introduce Token3D, a power balancing technique that takes into account temperature and layout information to balance the available per-core power along with other power optimizations for 3D designs. We also analyze a wide range of floorplans looking for the optimal temperature configuration. In addition, we also develop some optimizations for vertical 3D designs and a custom floorplan design that places hotspot structures in upper layers of the 3D stack and cooler structures in inner layers.

Agradecimientos

Para empezar me gustaría agradecer a mis directores de Tesis, Juan Luis Aragón, José Manuel García Carrasco y Stefanos Kaxiras por su apoyo y dedicación durante el desarrollo de la misma. Gracias a ellos he podido madurar (un poco) y aprender nuevas cosas que no estaban a mi alcance durante la carrera.

También me gustaría agradecer el apoyo y ayuda de todos los miembros del departamento de Arquitectura y Tecnología de Computadores y en especial a mis compañeros más cercanos, Alberto, Ana, Antonio, Chema, Dani, Epi, Ginés, Kenneth, José Luis, Ricardo, Rubén y Varadan. Sin su ayuda y compañía hace mucho que hubiera abandonado mis estudios de doctorado.

Quisiera además agradecer a las personas que me acogieron durante mis estancias en Grecia y Suecia, en especial a Pavlos Petoumenos, Nikos Nikoleris y Vasileios Spiliopoulos por hacerme sentir como en casa.

Gracias a mis padres, Juan y Lola, a mi hermana Beatriz, y a mis amigos más cercanos por mostrar tanta paciencia conmigo en esos días donde el estrés podía con la razón y no les trataba como se merecen. Para finalizar, agradecer a la última persona que se ha incorporado a mi vida, y que en poco tiempo se ha convertido en la más importante, mi querida María. Espero que podamos recuperar parte del tiempo que he invertido en este trabajo para estar juntos.

Índice

Abstract **XI**

Agradecimientos **XIII**

Lista de Figuras **XXI**

Lista de Tablas **XXV**

Resumen **XXVII**

1. Introducción **1**

2. Definición del Problema y Metodología de Simulación **7**

3. Mecanismos de Control de Consumo **35**

4. Adaptando el Consumo en Procesadores Mononúcleo **67**

5. Adaptando el Consumo en Procesadores Multinúcleo **85**

6. Adaptando el Consumo en Procesadores Multinúcleo Multicapa **111**

7. Conclusiones y Trabajo Futuro **129**

Bibliografía **135**

Contents

Abstract xi

Agradecimientos xiii

Resumen xxvii

0.1. Introducción . . . xxvii
0.2. Contribuciones . . . xxx
0.3. Control de Consumo en Procesadores Mononúcleo . . . xxxii
0.4. Control de Consumo en Procesadores Multinúcleo . . . xxxv
0.5. Control de Consumo en Procesadores Multinúcleo 3D . . . xxxvii
0.6. Conclusiones . . . xxxix

1. Introduction 1

1.1. Motivation . . . 1
1.2. Contributions . . . 3
1.3. Organization . . . 4

2. Problem Statement and Simulation Methodology 7

2.1. Introduction . . . 7
2.2. Power Tokens . . . 10
2.3. Importance of Accuracy . . . 11
2.4. Simulators . . . 12
2.5. Power Models . . . 15
2.5.1. Dynamic Power Models . . . 15
2.5.2. Leakage Power Models . . . 18
2.6. Temperature Models . . . 21
2.7. Benchmarks . . . 23
2.8. Benchmark Thermal Profiles and Per-Structure Power Distribution . . . 29
2.9. Performance vs Power-Efficiency in a Multi-Core Scenario . . . 33

3. Power Saving Mechanisms 35

3.1. Dynamic Power Control Mechanisms . . . 35

3.1.1. Dynamic Voltage and Frequency Scaling . . . 36
3.1.2. Pipeline Throttling . . . 39
3.1.3. Critical Path . . . 40
3.1.4. Hybrid Approaches . . . 42
3.1.5. Timeline Analysis of Power Dissipation . . . 44
3.2. Leakage Control Mechanisms . . . 44
3.2.1. Value Predictors: A Case Study for Leakage Reduction . . . 46
3.2.2. Problem Overview: Generational Behaviour in Value Prediction Structures . . . 47
3.2.3. Techniques for Reducing Leakage in Value Predictors . . . 49
3.2.4. Experimental Results . . . 55
3.3. Conclusions . . . 63

4. Single-Core Power Budget Matching 67
4.1. Introduction . . . 67
4.2. Power-Saving Microarchitectural Techniques . . . 70
4.2.1. Reactive Techniques . . . 70
4.2.2. Predictive Techniques . . . 70
4.3. Experimental Results . . . 75
4.3.1. Simulation Methodology . . . 75
4.3.2. A Power Budget of What? (100% Usage $\neq$ 100% Power) . . . 76
4.3.3. Cycles Over PB and Area Distribution . . . 77
4.3.4. Efficiency on Matching a Power Budget . . . 77
4.3.5. Preventive Switch-Off and Predictive Switch-on . . . 80
4.3.6. Sensitivity Study . . . 81
4.4. Conclusions . . . 81

5. Multi-Core Power Budget Matching 85
5.1. Introduction . . . 85
5.2. Background and Related Work . . . 88
5.2.1. CMP-specific Power Control Mechanisms . . . 88
5.3. Enforcing a Power Budget in CMPs . . . 89
5.3.1. Simulation Environment . . . 90
5.3.2. Matching a Power Budget in a CMP Running Parallel Workloads . 91
5.3.3. Analysis on the Power Dissipated in Spinning . . . 93
5.3.4. Power Token Balancing (PTB) . . . 94
5.3.5. Reusing Wasted Power to Reduce Energy: Nitro . . . 97
5.4. Experimental Results . . . 99
5.4.1. Efficiency of Power Token Balancing (PTB) . . . 99
5.4.2. Dynamic Policy Selector . . . 102

5.4.3. Relaxing PTB to be More Energy-Efficient . . . 103
5.4.4. The Importance of Accuracy . . . 104
5.4.5. Temperature Analysis . . . 105
5.4.6. Nitro Energy and Performance Analysis . . . 107
5.5. Conclusions . . . 108

6. 3D Die-Stacked Power Budget Matching - Token3D 111
6.1. Introduction . . . 111
6.2. Background and Related Work . . . 114
6.2.1. Power and Thermal Control in Microprocessors . . . 114
6.2.2. Towards the Third Dimension . . . 115
6.3. Thermal Control in 3D Die-Stacked Processors . . . 120
6.3.1. Token3D: Balancing Temperature on 3D Die-Staked Designs . . . 120
6.3.2. Token3D Implementation Details . . . 120
6.4. Experimental Results . . . 121
6.4.1. Simulation Environment . . . 121
6.4.2. Effects of Token3D on Peak Temperature . . . 123
6.4.3. Further Temperature Optimizations . . . 125
6.5. Conclusions . . . 127

7. Conclusions and Future Ways 129
7.1. Conclusions . . . 129
7.2. Future Work . . . 133

Bibliography 135

List of Figures

1. Ejemplo de la métrica AoPB. XXIX
2. Disipación de potencia por ciclo para un límite de consumo de 50 %. . . . XXXIII
3. AoPB relativa para distintos límites de consumo. XXXIV
4. Energía normalizada. XXXIV
5. Energía Normalizada (arriba) y AoPB (abajo) para un CMP de 16 núcleos con un límite de consumo de 50 %. XXXVI
6. Posibles distribuciones de los núcleos en capas. XXXVIII
7. Temperatura pico para PTB, Token3D y caso base para diferentes organizaciones y configuraciones de núcleos. XXXIX

2.1. Evolution of the thermal design power for AMD microprocessors. 8
2.2. Evolution of the thermal design power for Intel microprocessors. 9
2.3. Example of the Area over Power Budget (AoPB) metric. Shadowed areas represent the energy consumed over the target power budget. 12
2.4. High-level CMOS Inverter Diagram. 19
2.5. High-k vs regular transistor. 20
2.6. Packing components (left) and 3x3 grid thermal model (right) as described in HotSpot [88]. 22
2.7. Core Configuration 30
2.8. Power breakdown (%) of the simulated core respect to a total dynamic power of $7.6W$. 30
2.9. Thermal profiles for the SPLASH-2 benchmark suite. 31
2.10. Thermal profiles for the PARSEC 2.1 benchmark suite. 31
2.11. Thermal profiles for the SPECINT2000 benchmark suite. 32
2.12. Performance speedup (left) and energy (right) for the studied parallel benchmarks. 33
2.13. Energy delay product for the studied parallel benchmarks. 34

3.1. Cycles over PB and slowdown for DCR-based throttling (power budget = 50%). 40
3.2. Cycles over PB and slowdown for JRS-based throttling (power budget = 50%). 40

3.3. DVFS and critical path effects on power dissipation at a cycle level. 41
3.4. Instruction criticality analysis approach for a power budget of 50%. 42
3.5. Per-cycle power dissipation for a power budget of 50%. 43
3.6. Standard deviation effect for one window ($500\mu secs$). 45
3.7. Value Predictor Analysis. 48
3.8. Temporal behavior of a value predictor entry. 49
3.9. Fraction of time VP entries spend in dead state (SpecInt2000). 49
3.10. AVPD mechanism workflow. 52
3.11. Average speedup for the static decay and drowsy schemes for 10 KB VPs. 56
3.12. STP performance degradation (top) and leakage energy savings (bottom). 57
3.13. FCM performance degradation (top) and leakage energy savings (bottom). 57
3.14. DFCM performance degradation (top) and leakage energy savings (bottom). 59
3.15. Static drowsy scheme for the DFCM (top), STP (middle) and FCM (bottom) value predictors. 60
3.16. Static decay scheme for a 10 KB DFCM value predictor. 62
3.17. STP (top), FCM (middle) and DFCM (bottom) value predictor leakage energy savings. 63

4.1. Base vs DVFS exploration window power dissipation for the "go" benchmark. 71
4.2. Detailed per-cycle power dissipation for the "go" benchmark. 72
4.3. Overview of the preventive switch-off and predictive switch-on mechanisms. 75
4.4. Area and Cycles over PB distributions 77
4.5. Relative cycles over PB for different power budgets. 78
4.6. Relative area over PB for different power budgets. 79
4.7. Normalized energy consumption. 79
4.8. Relative cycles over PB for different power budgets. 80
4.9. Relative area over PB for different power budgets.. 80
4.10. Normalized energy for all the evaluated approaches along with preventive switch-off and predictive switch-on. 81
4.11. Sensitivity analysis. 82

5.1. Normalized Energy (top) and AoPB (bottom) for a 16-core CMP with a power budget of 50%. 92
5.2. Execution time breakdown for a varying number of cores. 94
5.3. Normalized spinlock power for a varying number of cores. 94
5.4. Power Token Balancing motivation (not real numbers). 95
5.5. Per-cycle power behavior of a spinning core. 95
5.6. Power Token Balancing example in the case of a barrier (using the *ToAll* policy). 96
5.7. PTB implementation diagram for a 4-core CMP. 98

5.8. Normalized energy (top) and area over the power budget (bottom) for a varying number of cores and PTB policies. . . . 100
5.9. Detailed energy (top) and AoPB (bottom) for a 16-core CMP with the ToAll PTB policy. . . . 100
5.10. Detailed energy (top) and AoPB (bottom) for a 16-core CMP with the ToOne PTB policy. . . . 101
5.11. Detailed energy (top) and AoPB (bottom) for a 16-core CMP using the dynamic policy selector. . . . 102
5.12. Detailed Performance for a 16-core CMP using the dynamic policy selector. 102
5.13. Normalized energy (top) and area over the power budget (bottom) for a varying number of cores and PTB policies. . . . 104
5.14. Core floorplan. . . . 106
5.15. Average and peak temp. of a 16-core CMP. . . . 106
5.16. Minimum base vs peak PTB temperature of a 16-core CMP. . . . 106
5.17. Normalized per-structure peak (left) and average (right) temperature analysis. . . . 107
5.18. Normalized per-benchmark peak (left) and average (right) temperature analysis. . . . 107
5.19. Performance improvement and energy reduction for a 16-core CMP using Nitro. . . . 108

6.1. Range of a wire in a single clock cycle (Source: Real World Technologies). 116
6.2. 130nm process generation with 6 layers of copper interconnect (Source: Intel). . . . 116
6.3. Yield impact of wafer to wafer bonding. . . . 119
6.4. Core distribution along the layers. . . . 120
6.5. Core Configuration . . . 122
6.6. Peak temperature for PTB, Token3D and the base case for different floorplans and core configurations. . . . 123
6.7. Per structure peak temperature (top) and performance (bottom) of a 4-layer 16-core CMP using the mirror floorplan. . . . 125
6.8. Peak temperature of the instruction window (top) and ALUs (bottom) for a varying number of cores. . . . 126

List of Tables

2.1. Equations for Wattch power models. $C_{diff}C_{gate}C_{metal}$ represent diffusion, transistor gate and metal wire capacitances. 16
2.2. K_{design} parameters for typical circuits. 18
2.3. Leakage power models for different processor structures. 18
2.4. Normalized power distribution for SPLASH-2 32
2.5. Normalized power distribution for PARSEC 33
2.6. Normalized power distribution for SPECINT2000 33

3.1. Core configuration. 36

4.1. Core configuration. 76

5.1. Core configuration. 90
5.2. Evaluated benchmarks and input working sets. 90

6.1. Evaluated benchmarks and input working sets. 121

Resumen

0.1. Introducción

En la última década los ingenieros informáticos se han enfrentado a profundos cambios en el modo en que se diseñan y fabrican los microprocesadores. Los nuevos procesadores no solo deben ser más rápidos que los anteriores, también deben ser factibles en términos de energía y disipación térmica, sobre todo en dispositivos que trabajan con baterías. Los problemas relacionados con consumo y temperatura son muy comunes en estos procesadores. No solo estamos hablando de servidores, sino también de dispositivos móviles, ordenadores de sobremesa, portátiles, tarjetas gráficas, etc. Durante años el diseño de microprocesadores ha estado (y está) limitado por la disipación de potencia y la temperatura. Muchos de los trabajos relacionados se refieren a estos factores clave con el término de "Power Wall". También cabe destacar que después de diez años en los que el diseño de procesadores se centraba en aumentar la frecuencia de funcionamiento de los transistores, los ingenieros han decidido cambiar de estrategia y pasar a aumentar el número de núcleos dentro del mismo procesador, favoreciendo la productividad total del procesador sobre el aumento de rendimiento local de cada núcleo. A pesar de esto, el denominado "power wall" sigue presente, limitando el número máximo de núcleos que podemos colocar dentro del procesador.

Como cualquier otro dispositivo electrónico, los procesadores tienen dos componentes principales de disipación de potencia, la disipación de potencia dinámica, que es directamente proporcional al uso (cada vez que accedemos a una estructura dentro del procesador) y la potencia estática (o potencia de fuga), que proviene de las corrientes de fuga que fluyen por el circuito incluso cuando los transistores no están siendo utilizados. "Dynamic Voltage and Frequency Scaling" (DVFS), "Dynamic Frequency Scaling" (DFS) y "Clock Gating" son tres de los mecanismos más comunes para controlar el consumo energético. Desafortunadamente, DVFS se hace cada vez menos efectivo con cada nueva escala de integración de transistores. DVFS depende de la reducción del voltaje que alimenta al circuito para reducir el consumo $P_d \approx V_{DD}^2 \cdot f$, pero, para mantener la velocidad de transición entre estados de un transistor debemos modificar el voltaje umbral de igual forma. La pega es que esta reducción del voltaje umbral produce un incremento exponencial del consumo estático, el cual supone una importante porción del consumo total para tecnologías por debajo de los 65nm, incluso con la introducción de

los transistores tipo high-k [47]. DFS es similar a DVFS, pero solo escala la frecuencia del transistor. Finalmente, "Clock Gating" bloquea la señal de reloj que alimenta el circuito para prevenir que este se active y disipe potencia.

Los problemas de temperatura y consumo mencionados anteriormente son habitualmente detectados y tratados por un mecanismo de gestión dinámica de temperatura o "Dynamic Thermal Management" (DTM). La política más común suele ser limitar la cantidad de energía que el procesador o núcleos del procesador pueden consumir (estableciendo un límite de consumo). En cualquier caso, la mayoría de los procesadores actuales no disponen de los medios para medir el consumo actual en tiempo real. En realidad el consumo se estima basándose en información proporcionada por los contadores de rendimiento a lo largo de miles de ciclos. En esta Tesis presentamos los denominados "Power Tokens". Utilizando los "Power Tokens" podemos estimar la energía con mayor precisión, sumando al consumo base de cada instrucción (que depende de las estructuras del procesador a las que accede) una componente dinámica que depende del tiempo que dicha instrucción permanece en el "pipeline". El consumo total del procesador se puede estimar sumando los consumos individuales de cada una de las instrucciones que son ejecutadas en el procesador.

Además, hay ocasiones en las que los requisitos de consumo del procesador exceden las características del dispositivo donde se quiere utilizar. En la mayoría de los casos no nos podemos permitir diseñar un nuevo procesador para adaptarse a dichas características ya que resulta muy caro. El problema es aún peor si las restricciones de consumo son temporales, y tras un periodo de bajo consumo queremos restablecer el comportamiento original del procesador (p. ej. sobrecalentamiento). Imaginemos, por ejemplo, un "cluster" de computación conectado a uno o más dispositivos de alimentación auxiliares. Si se produce un corte en el suministro eléctrico, los procesadores continuarán trabajando a máxima potencia consumiendo rápidamente la batería de los dispositivos de alimentación auxiliares, apagando los ordenadores cuando esté apunto de acabarse para mantener la consistencia del sistema, perdiendo todo el trabajo realizado. Si los procesadores no están realizando un trabajo crítico podríamos limitar el consumo de los mismos, aumentando así la autonomía de los dispositivos aguantando, con suerte, hasta que se restablezca el suministro eléctrico. Otra situación en la que la capacidad de limitar el consumo de los procesadores podría resultar útil sería un centro de computación que comparte el suministro eléctrico entre nodos de computación y sistema de refrigeración. En el peor de los casos (verano a medio día) se podría limitar el consumo de los procesadores para enfriar la sala, disponiendo a su vez los dispositivos de refrigeración de una mayor porción de energía para realizar su trabajo. Surge por lo tanto una nueva necesidad, la de ser capaces de adaptar el consumo de un microprocesador ante una limitación externa, aunque ello conlleve una degradación del rendimiento.

Es importante destacar la importancia de la precisión a la hora de adaptarse a un límite de consumo impuesto. Para garantizar una desviación mínima del límite de consumo preestablecido introduciremos una nueva métrica, el área sobre el límite de consumo

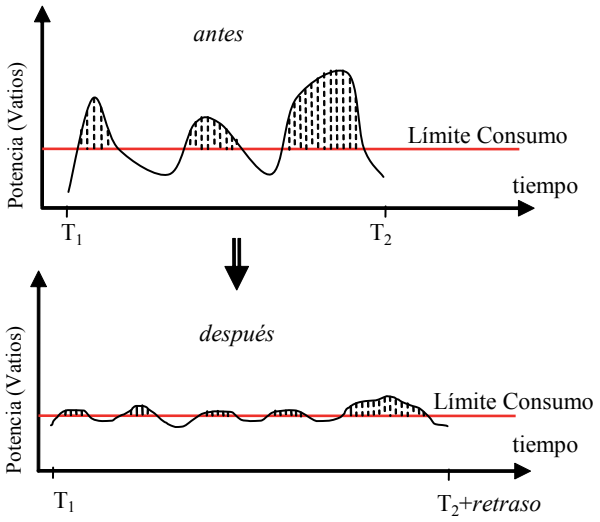


Figura 1: Ejemplo de la métrica AoPB.

o "Area over Power Budget" (AoPB), que corresponde al consumo (en julios) entre el límite de consumo preestablecido y la curva de consumo dinámica de cada núcleo (zona sombreada en la Figura 1). Cuanto menor sea esta área, mayor será la precisión del mecanismo.

En esta Tesis analizamos el rendimiento, disipación de potencia, consumo energético y precisión de diferentes mecanismos de reducción de consumo extraídos de la literatura. Tras este análisis descubrimos que dichos mecanismos no son suficientemente buenos para adaptarse a un límite de consumo preestablecido con una penalización de rendimiento razonable. Para solucionar este problema proponemos diversas técnicas a nivel de microarquitectura que combinan de manera dinámica varios mecanismos de reducción de consumo para obtener una aproximación al límite de consumo mucho más precisa con una penalización de rendimiento mínima. Nuestros mecanismos para procesadores mononúcleo son los siguientes: "Power Token Throttling" (PTT), un mecanismo basado en "tokens" que monitoriza el consumo en tiempo real del procesador y deja de introducir instrucciones en el "pipeline" si la siguiente instrucción a ser ejecutada hace que el procesador exceda el límite de consumo; "Basic Block Level Mechanism" (BBLM), el cual monitoriza el consumo del procesador a nivel de bloque básico y selecciona de manera predictiva el mecanismo de reducción de consumo que mejor se adapte al consumo medio durante la ejecución de dicho bloque básico; y, finalmente, un mecanismo de dos niveles que utiliza DVFS como un mecanismo de grano grueso para reducir el consumo medio del procesador y dejarlo lo más próximo al límite de consumo para a continuación utilizar mecanismos de microarquitectura con el fin de eliminar los numerosos picos de consumo del procesador.

Cuando evaluamos los mecanismos mencionados en el párrafo anterior en un entorno multinúcleo o "Chip Multiprocessor" (CMP) descubrimos que dichos mecanismos solo funcionan correctamente para aplicaciones con hilos independientes o cargas multiprogramadas, pero no para cargas paralelas. Los puntos de sincronización característicos de

dichas cargas alteran la ejecución óptima del código, de tal forma que el tiempo de ejecución de una aplicación no viene dictado por el rendimiento individual de cada núcleo, sino de la combinación de los distintos núcleos del procesador. Sacrificar rendimiento para reducir el consumo de un núcleo (hilo) de manera individual puede suponer un retraso en todos los núcleos (hilos) del procesador debido a un futuro punto de sincronización (cerrojo/barrera), degradando el rendimiento global de la aplicación. Para mejorar los mecanismos de control de consumo en el área de los procesadores multinúcleo presentamos "Power Token Balancing" (PTB). Este mecanismo balancea de manera eficiente la potencia disipada entre los distintos núcleos utilizando un sistema basado en "tokens". Podemos utilizar la potencia sobrante (vista como "tokens") de los hilos no críticos para beneficiar a los hilos críticos.
El uso de los procesadores multinúcleo permite un incremento del rendimiento de la aplicación explotando el paralelismo de la misma, pero tiene dos importantes limitaciones: el ancho de banda de las comunicaciones externas al procesador y la latencia de comunicación entre los núcleos. Los procesadores multinúcleo 3D suponen un gran avance para solventar estas limitaciones apilando varias capas de procesadores unas sobre otras. El principal problema de este diseño es que al aumentar la densidad del empaquetado también se aumenta la densidad de potencia, lo que se traduce en mayores problemas térmicos. Para finalizar, y también en el área de los procesadores multinúcleo, hemos analizado cómo los mecanismos de control de consumo afectan a la temperatura de los nuevos procesadores multinúcleo 3D. En concreto presentamos Token3D, un mecanismo de balanceo de consumo que tiene en cuenta información acerca de la temperatura de las distintas estructuras del procesador así como de su organización dentro del circuito electrónico para balancear la potencia disponible entre los distintos núcleos del procesador. También se realiza un amplio análisis de distintas organizaciones de estructuras con el fin de optimizar la temperatura total del procesador, a la vez que se presenta una organización de estructuras personalizada que optimiza la temperatura del núcleo situando estructuras calientes en capas externas de la pila 3D y estructuras frías en las capas más internas del procesador 3D.

0.2. Contribuciones

Las principales contribuciones de esta Tesis han sido publicadas en [23, 24, 25, 26, 27] y se resumen a continuación:

- Introducimos el concepto de "Power Tokens". En la mayoría de los microprocesadores modernos no existe ninguna forma de medir el consumo en tiempo real. Las mediciones de consumo no son más que meras aproximaciones basadas en los contadores de rendimiento internos del procesador que se realizan cada miles de ciclos. Cuando nos disponemos a utilizar una técnica de microarquitectura que trabaja a nivel de ciclo surge la necesidad de disponer de un sistema de medición que trabaje a dicho nivel, por eso proponemos los "Power Tokens". Su funcionamiento se basa

en aproximar el consumo de una instrucción como la suma de su consumo base (debido a los accesos a las distintas estructuras usadas por la instrucción) más una componente variable que depende del tiempo que la instrucción permanezca en el "pipeline" del procesador. La línea de procesadores Sandy Bridge de Intel incluye una serie de registros internos donde se almacena información acerca del consumo de diversas estructuras del procesador, lo que en cierta forma consolida las decisiones que tomamos hace tres años a la hora de diseñar los "Power Tokens".

- Realizamos un análisis en términos de rendimiento, disipación de potencia y consumo energético de diferentes mecanismos de reducción de consumo en procesadores mononúcleo. Con el fin de gestionar la disipación de potencia dinámica analizamos: DVFS, que modifica tanto el voltaje como la frecuencia de funcionamiento del procesador para reducir el consumo; "Pipeline Throttling", que reduce la cantidad de instrucciones emitidas por el procesador para reducir el consumo; "Instruction Criticality", que retrasa la ejecución de instrucciones de ciclos sobre el límite de consumo a ciclos por debajo del límite de consumo; y "Confidence Estimation", que permite reducir el nivel de especulación del procesador en los caminos de ejecución de baja confianza (aquellos que corresponden a saltos que no pueden ser predichos de manera precisa). Para gestionar la disipación de potencia estática ("leakage") analizaremos dos tipos de técnicas: técnicas basadas en "decay", que desconectan entradas de las estructuras tipo "array" del procesador reduciendo su consumo estático a cero pero perdiendo su contenido (técnicas sin conservación del estado); y técnicas basadas en "drowsy", que reducen el voltaje de determinadas entradas de una estructura, y, aunque no reducen tanto el consumo como las técnicas de "decay", son capaces de recuperar el estado de las entradas de la estructura (técnicas con conservación del estado). Ambos tipos de técnicas son evaluadas utilizando predictores de valor como un ejemplo típico de estructura regular.

- También en el ámbito de los procesadores mononúcleo analizamos los efectos de distintas técnicas de reducción de consumo publicadas en la literatura intentando adaptar el consumo del procesador a un límite externo. Tras este análisis descubrimos que estos mecanismos no son suficientemente buenos para adaptarse a un límite de consumo preestablecido con una penalización de rendimiento razonable. Para solucionar este problema proponemos diversas técnicas híbridas a nivel de microarquitectura que combinan de manera dinámica varios mecanismos de reducción de consumo para obtener una aproximación al límite de consumo mucho más precisa con una penalización de rendimiento mínima. Nuestros mecanismos para procesadores mononúcleo son los siguientes: "Power Token Throttling" (PTT), un mecanismo basado en "tokens" que monitoriza el consumo en tiempo real del procesador y deja de introducir instrucciones en el "pipeline" si la siguiente instrucción a ser ejecutada hace que el procesador exceda el límite de consumo; "Basic Block Level Mechanism" (BBLM), el cual monitoriza el consumo del procesador a nivel de bloque básico y selecciona de manera predictiva el mecanismo de reducción de

consumo que mejor se adapte al consumo medio durante la ejecución de dicho bloque básico; y finalmente un mecanismo de dos niveles que utiliza DVFS como un mecanismo de grano grueso para reducir el consumo medio del procesador lo más próximo al límite de consumo para a continuación utilizar mecanismos de microarquitectura con el fin de eliminar los numerosos picos de consumo del procesador.

- En el ámbito de los microprocesadores multinúcleo presentamos "Power Token Balancing" (PTB), un mecanismo que tiene como objetivo el adaptarse a un límite de consumo de manera eficiente, balanceando la potencia disipada entre los distintos núcleos utilizando un sistema basado en "tokens". Podemos utilizar la potencia sobrante (vista como "tokens") de los hilos no críticos de ejecución para beneficiar a los hilos críticos. PTB no afecta al rendimiento ni al consumo de las cargas multiprogramadas ni de las aplicaciones secuenciales y puede ser utilizado además como un mecanismo de detección de esperas activas.

- También en el ámbito de los procesadores multinúcleo analizamos cómo los mecanismos de control de consumo afectan a la temperatura de los nuevos procesadores multinúcleo 3D. En concreto presentamos Token3D, un mecanismo de balanceo de consumo que tiene en cuenta información acerca de la temperatura de las distintas estructuras del procesador así como de su organización dentro del circuito electrónico para balancear la potencia disponible entre los distintos núcleos del procesador. También se realiza un amplio análisis de distintas organizaciones de estructuras con el fin de optimizar la temperatura total del procesador, a la vez que se presenta una organización de estructuras personalizada que optimiza la temperatura del núcleo, situando estructuras calientes en capas externas de la pila 3D y estructuras frías en las capas más internas del procesador 3D.

0.3. Control de Consumo en Procesadores Mononúcleo

Los procesadores actuales se enfrentan constantemente a problemas relacionados con consumo y temperatura, los cuales suelen solventarse aplicando un límite de consumo al procesador. El escalado dinámico de voltaje y frecuencia o "Dynamic Voltage and Frequency Scaling" (DVFS) ha supuesto un gran avance que permite a los procesadores adaptarse a un límite de consumo impuesto. A pesar de esto, DVFS está empezado a resultar cada día menos efectivo, debido a la creciente importancia del consumo estático sobre el dinámico.

En esta sección proponemos el uso de técnicas de microarquitectura para adaptar de manera precisa el consumo del procesador ante un límite externo de manera eficiente. Tras un análisis individual de las técnicas más comúnmente utilizadas en la literatura para controlar el consumo en microprocesadores (Figura 2) concluimos que ninguna de ellas es capaz de adaptarse al límite de consumo impuesto con una precisión y degradación de rendimiento razonables.

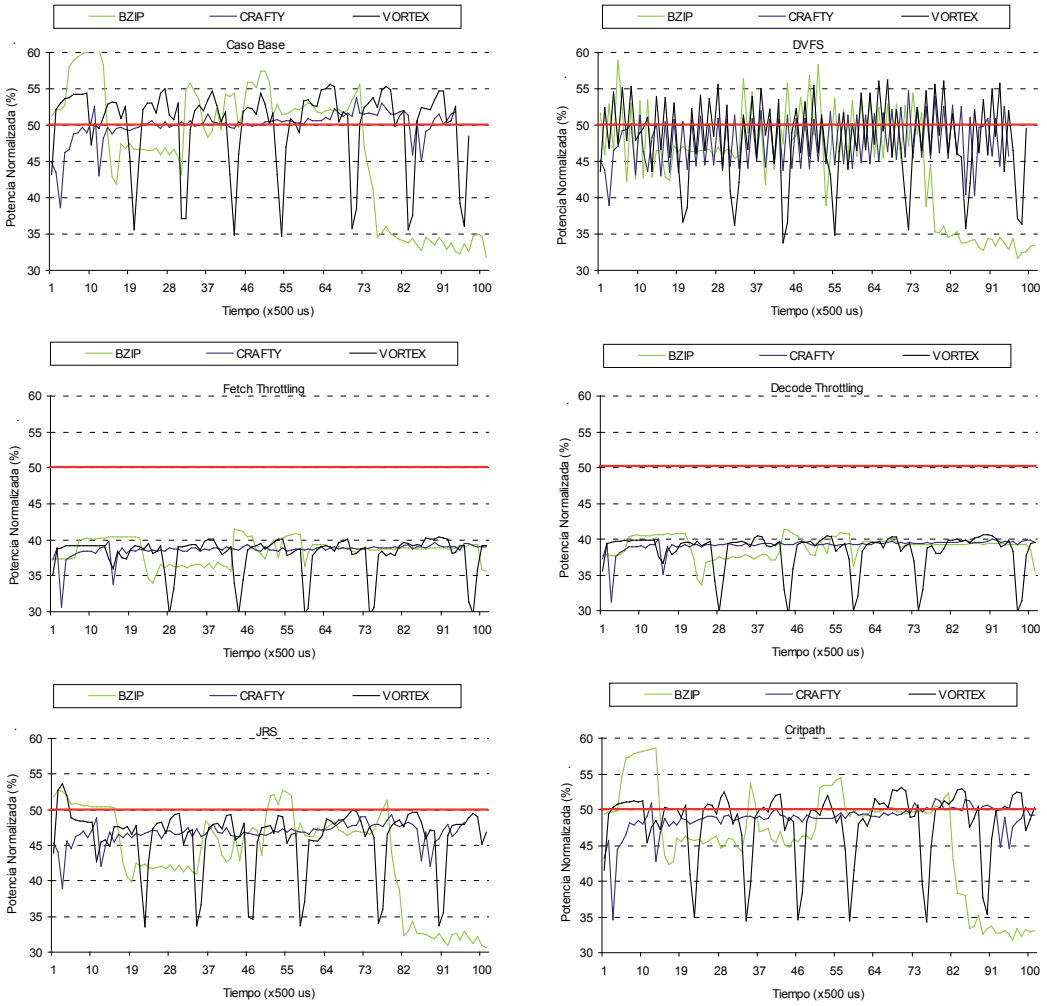


Figura 2: Disipación de potencia por ciclo para un límite de consumo de 50 %.

Para mejorar estas características proponemos tres mecanismos: "Power Token Throttling" (PTT), "Basic Block Level Manager" (BBLM) y un mecanismo adaptativo de dos niveles (DVFS+BBLM). PTT es un mecanismo basado en información de consumo en forma de "Power Tokens" que monitoriza constantemente el consumo actual del procesador y previene la inserción de nuevas instrucciones en el procesador si la siguiente instrucción a ejecutarse hace que el procesador supere su límite de consumo. BBLM es un mecanismo predictivo de control de consumo que almacena información acerca de la energía consumida por un determinado bloque básico (o cadena de bloques básicos). BBLM selecciona, entre diferentes mecanismos de control de consumo, el más apropiado para adaptarse al límite de consumo basándose en un histórico de consumo del bloque básico a ejecutar por el procesador. Para finalizar proponemos un mecanismo de dos niveles que utiliza DVFS como un mecanismo de grano grueso para aproximar el consumo medio del procesador al límite de consumo para a continuación utilizar técnicas de microarquitectura con el fin de eliminar los numerosos picos de consumo restantes.

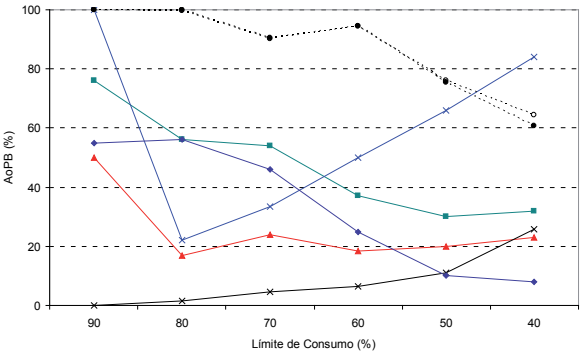


Figura 3: AoPB relativa para distintos límites de consumo.

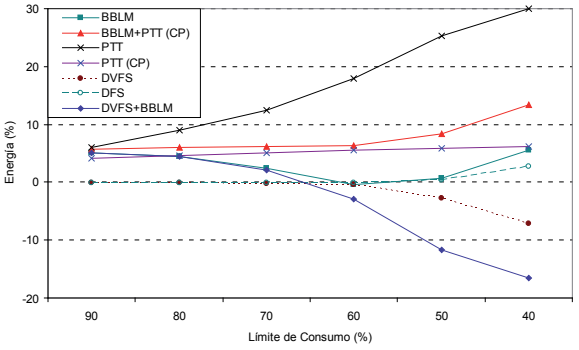


Figura 4: Energía normalizada.

Las técnicas utilizadas por BBLM intentarán:

1. Localizar instrucciones que no sean críticas en ciclos en los que el procesador está sobre el límite de consumo y retrasarlas a ciclos en los que el procesador esté bajo el límite de consumo. Hay que tener en cuenta que instrucciones no críticas inicialmente pueden volverse críticas si se retrasan demasiado.

2. Se ha demostrado que aproximadamente el 30 % del consumo de un procesador viene dado por la ejecución de instrucciones que pertenecen a caminos incorrectos [4, 66]. Por lo tanto, podemos reducir el número de instrucciones ejecutadas especulativamente por el procesador cuando nos encontramos sobre el límite de consumo basándonos en información acerca de la confianza que tiene un estimador sobre el resultado de la predicción del salto.

3. Por último, cuando las dos políticas anteriores no son capaces de reducir el consumo por debajo del límite impuesto, podemos reducir el número de instrucciones que ejecuta el procesador cada ciclo.

La Figura 4 muestra la energía normalizada para los distintos mecanismos propuestos (y diversas combinaciones de los mismos) y varios límites de consumo. Los límites de consumo están expresados como un porcentaje referente al consumo pico del procesador medido en tiempo de ejecución. Podemos observar que el mecanismo de dos niveles

(DVFS+BBLM) es el más eficiente en términos de energía para todos los límites de consumo estudiados, y especialmente para los límites más restrictivos. El resto de técnicas microarquitecturales (salvo PTT) muestran un aumento de la energía total consumida entre un 4 % y un 10 %. BBLM muestra unos números de energía similares a DFS hasta un límite de consumo del 50 % siendo cuatro veces más preciso (reduce cuatro veces más el AoPB). Además, dado que estamos trabajando bajo un límite de consumo, la Figura 3 muestra cómo de precisos son los distintos mecanismos estudiados, ya que muestra el AoPB de los mismos cuando intentan adaptarse a un límite de consumo. Se puede observar que todas las técnicas de microarquitectura son mucho más precisas que DVFS y DFS cuando intentan adaptarse al límite de consumo. PTT sufre un aumento de la energía total consumida ya que enlentece tanto el procesador que provoca que la energía adicional consumida por mantener el procesador activo supere los ahorros obtenidos por los mecanismos de reducción de consumo.

Cuando trabajamos con límites de consumo restrictivos (<60 %), nuestro mecanismo de dos niveles (DVFS+BBLM) es un 4 % más eficiente en términos de energía que DVFS y también tres veces más preciso. Para un límite de consumo extremo de 40 % del consumo pico original del procesador, nuestro mecanismo de dos niveles mejora tanto en términos de área (solo un 10 % del área permanece sobre el límite de consumo) como en energía, siendo un 11 % más eficiente energéticamente que DVFS. En términos generales DVFS y DFS son mecanismos de grano grueso incapaces de eliminar los numerosos picos de consumo, lo que reduce considerablemente su precisión.

0.4. Control de Consumo en Procesadores Multinúcleo

El mercado actual está dominado por procesadores que emplean arquitecturas multinúcleo (CMPs), las cuales muestran un comportamiento diferente dependiendo de la aplicación que se ejecuta sobre ellas (paralela, multiprogramada, secuencial). Es posible utilizar mecanismos de control de consumo heredados de procesadores mononúcleo en CMPs que ejecutan o bien aplicaciones secuenciales o bien cargas con hilos de ejecución independientes. El problema aparece cuando intentamos ejecutar aplicaciones paralelas de memoria compartida, donde al penalizar el rendimiento de un único núcleo (hilo) para ahorrar energía podemos penalizar de manera inintencionada el rendimiento del resto de núcleos, debido a puntos de sincronización (cerrojos/barreras), ralentizando la ejecución global del programa.

El aumento significativo del número de núcleos de los CMPs está causando graves problemas de temperatura y consumo, los cuales suelen controlarse mediante el uso de límites de consumo sobre el procesador o sobre un subconjunto de núcleos del mismo. En esta Tesis realizamos un análisis de cómo se comportan diferentes técnicas propuestas en la literatura a la hora de adaptarse a un límite de consumo en un CMP. La Figura 5 nos muestra cómo las técnicas heredadas de procesadores mononúcleo no son capaces de adaptarse correctamente al límite de consumo cuando ejecutan cargas paralelas (>45 %

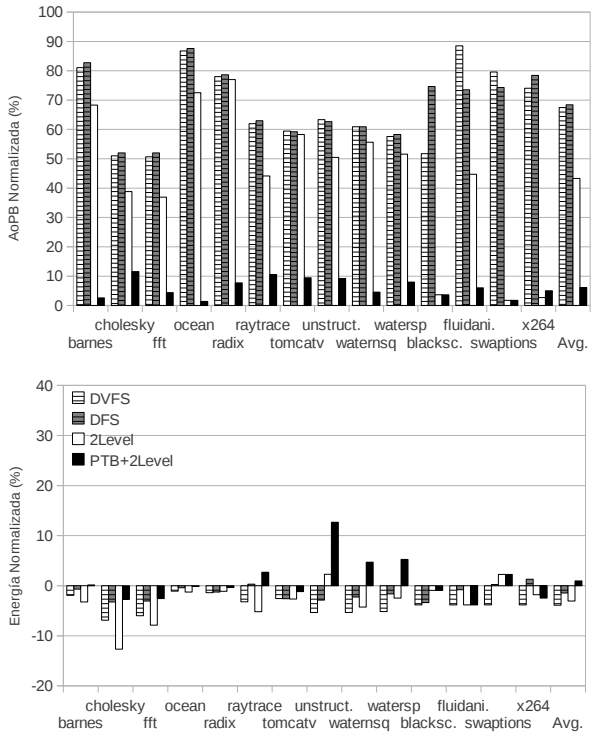


Figura 5: Energía Normalizada (arriba) y AoPB (abajo) para un CMP de 16 núcleos con un límite de consumo de 50 %.

AoPB cuando debería estar por debajo del 10), pero sí para cargas multiprogramadas o con hilos de ejecución independientes. Esto se debe a que los mecanismos tratan de adaptar el consumo individual de cada núcleo de manera independiente en un entorno donde existen dependencias entre los hilos de ejecución. Para solucionar este problema proponemos "Power Token Balancing" (PTB), que tiene como objetivo adaptar el consumo de un CMP a un límite impuesto de manera eficiente balanceando la energía consumida por los distintos núcleos del procesador mediante una serie de políticas basadas en "Power Tokens". PTB utiliza el consumo de los núcleos por debajo del límite de consumo (visto como "tokens" o cupones) para gestionar la energía de los núcleos sobre el límite de consumo. PTB se ejecuta de manera transparente para las aplicaciones de hilos independientes y cargas multiprogramadas. Los resultados experimentales (Figura 5) muestran cómo PTB se adapta de manera mucho más precisa al límite de consumo (la energía total consumida sobre el límite de consumo, AoPB, se reduce a un 8 % para un procesador de 16 núcleos) que DVFS con un incremento del consumo energético de un 3 % para un límite de consumo de un 50 % del consumo pico dinámico del procesador. Además, si relajamos las condiciones de precisión de PTB, podemos conseguir reducciones en la energía total consumida similares a las obtenidas por DVFS con una precisión tres veces superior (<20 % AoPB).

Decidimos analizar también los efectos en la temperatura del procesador usando números realistas de consumo y configuraciones de disipador/ventilador. Los resultados obtenidos

muestran cómo PTB no solo balancea la temperatura entre los distintos núcleos, sino que también permite reducir la temperatura media del procesador entre un 28 y un 30 %. Finalmente comentar que en este área hemos diseñado un mecanismo denominado "Nitro", inspirado en la idea del KERS de la Fórmula Uno, con el fin de recuperar energía cuando estás malgastándola y reutilizarla para acelerar en otro lugar. "Nitro" aumenta de manera dinámica la frecuencia de funcionamiento del núcleo que accede a una sección crítica (delimitada por cerrojos) para liberar dichos cerrojos lo más rápidamente posible, con el fin de permitir a otro núcleo (hilo) acceder a dicha sección crítica. El mecanismo asegura que este aumento de la frecuencia se puede realizar de manera segura ya que solo se aplica mientras exista energía disponible (no utilizada) por núcleos ociosos ("idle") o en estado de espera activa ("spinning"). "Nitro" es capaz de reducir el tiempo de ejecución de las aplicaciones con gran contención de cerrojos en más de un 5 % aumentando la frecuencia en menos de un 1 % del tiempo de ejecución.

0.5. Control de Consumo en Procesadores Multinúcleo 3D

El uso de los procesadores multinúcleo supone un incremento del rendimiento de una aplicación explotando el paralelismo de la misma, pero tiene dos importantes limitaciones: el ancho de banda de las comunicaciones externas al procesador y la latencia de comunicación entre los núcleos. Las arquitecturas 3D, presentadas por Souri en [89], apilan varias capas circuitos (p. ej., núcleos, memoria) estableciendo conexiones verticales entre ellas. Una consecuencia directa de este diseño es que se reduce la latencia y los costes energéticos de comunicación entre núcleos, a la vez que se aumenta la densidad del empaquetamiento del procesador en función del número de capas disponibles. A pesar de las numerosas ventajas de los diseños tridimensionales, también existen diversos retos que los diseñadores deben superar. Para empezar, el aumento de la densidad del empaquetamiento lleva asociado un aumento de la densidad de disipación de potencia del procesador, lo que se transforma en problemas de temperatura. También debemos tener en cuenta que el número de posibles diseños del circuito aumenta conforme aumenta el número de capas, a la vez que el coste de analizar las distintas posibilidades para optimizar los diseños tridimensionales. Finalmente debemos tener en cuenta que los costes y la complejidad de verificación de los procesadores también aumenta con el número de capas.

Así pues, y con el fin de controlar la temperatura del procesador, lo cual resulta de especial interés en los procesadores tridimensionales ya que este tipo de tecnologías ponen al límite las posibilidades térmicas de los materiales, necesitamos mecanismos mucho más precisos a la hora de controlar la disipación de potencia y la energía consumida por los distintos núcleos.

En este área realizamos tres contribuciones importantes. Primero, analizamos los efectos de los distintos mecanismos de control de consumo comentados en secciones previas para controlar la temperatura del procesador 3D. Basándonos en este análisis diseñamos

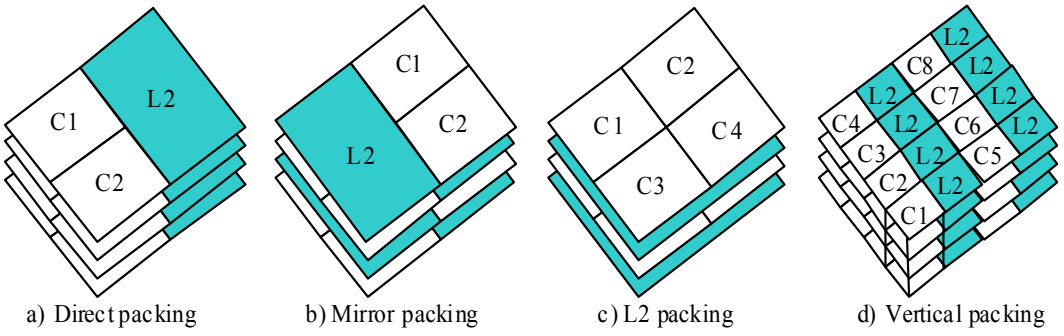


Figura 6: Posibles distribuciones de los núcleos en capas.

"Token3D", un mecanismo de balanceo de consumo basado en PTB que tiene en cuenta información de organización de estructuras y temperatura para balancear la potencia disponible entre los distintos núcleos y capas.

Segundo, realizamos un análisis de un amplio rango de organizaciones de los elementos de procesamiento y cachés de segundo nivel intentando optimizar nuestro diseño para una temperatura mínima teniendo en cuenta tanto el consumo estático como el dinámico. Existen numerosas formas de organizar los núcleos dentro de las capas del procesador 3D, las más importantes se muestran en la Figura 6. Se pueden identificar dos tendencias bien definidas; construir los núcleos de manera horizontal o de manera vertical. Los diseños horizontales (a-c) son los más comunes en la literatura, ya que son los más fáciles de implementar y validar. "Direct packing" es el diseño más sencillo, ya que capas con distribuciones de núcleos idénticas se apilan unas encima de otras para formar el procesador 3D. "Mirror packing" es una modificación muy simple de "direct packing" en el cual la mitad de las capas giradas 180 grados. Esta modificación supone una reducción considerable de la temperatura del procesador ya que las cachés de segundo nivel de las capas pares hacen de disipador de las impares y viceversa. Finalmente, "L2 packing" intercala capas que contienen núcleos con capas que solo contienen memoria (cachés de segundo o tercer nivel), lo cual reduce aún más la temperatura pero aumenta la complejidad de producción y de testeo. Por otro lado, los diseños verticales (Figura 6-d), presentados por Puttaswamy *et al.* en [79], ofrecen una reducción considerable de latencia y consumo sobre los diseños horizontales al proporcionar acceso paralelo a una misma estructura.

Y tercero, hemos desarrollado diversos mecanismos específicos de control de consumo para diseños de núcleos 3D así como organizaciones híbridas que mezclan diseños horizontales y verticales.

La Figura 7 muestra la temperatura pico de las diferentes configuraciones de procesadores 3D variando el número de núcleos (de 4 a 16). La temperatura "idle" corresponde a la temperatura media de los núcleos procesador en estado "idle"[1]. La barra correspondiente a "Token3D" muestra el aumento de temperatura desde la temperatura "idle" hasta la temperatura alcanzada por el procesador mientras se ejecutaban distintas aplicaciones

[1]Cuando el procesador está ocioso (solo está ejecutando el sistema operativo).

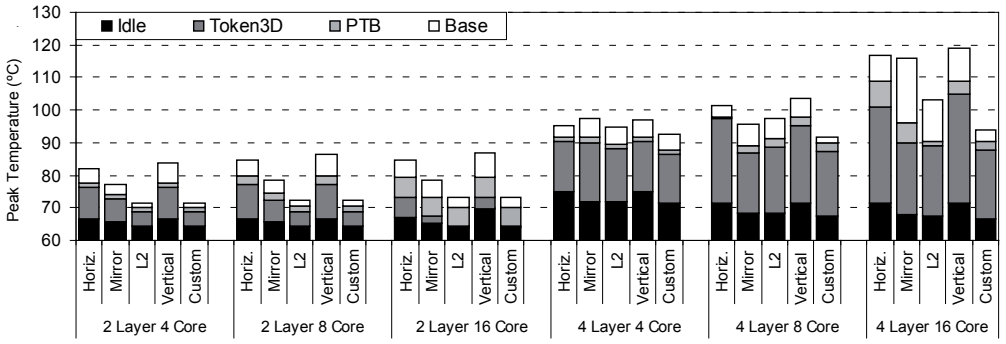


Figura 7: Temperatura pico para PTB, Token3D y caso base para diferentes organizaciones y configuraciones de núcleos.

(ver Sección 2.7) con "Token3D" activo. La barra "PTB" es similar a "Token3D" pero utilizando el mecanismo original de "Power Token Balancing" presentado en la sección anterior. Las organizaciones de núcleos estudiadas son: "Horizontal" (Figura 6.a), "Mirror" (Figura 6.b), "L2" (Figura 6.c), "Vertical" (Figura 6.d) y "Custom". Esta última se corresponde a una nueva configuración híbrida que hemos diseñado la cual sitúa las estructuras más calientes en capas externas del procesador 3D, facilitando su enfriamiento, y las estructuras más frías en las capas más internas. La Figura 7 muestra claramente que las configuraciones "L2" y "Custom" son las más apropiadas para reducir la temperatura del procesador. Esto se debe a que ambos diseños sitúan la caché de segundo nivel en las capas más internas, y dicha estructura es una de las más frías del procesador, incluso teniendo en cuenta el consumo estático de la misma. Esta organización deja a las estructuras más calientes en capas cercanas al disipador y pueden enfriarse más fácilmente.

En términos generales, tanto "PTB" como "Token3D" son capaces de reducir la temperatura del procesador entre 2ºC y 26ºC dependiendo de la organización de estructuras que estemos utilizando. "Token3D" es siempre entre 1 y 3 grados mejor que "PTB". Hay que tener siempre en cuenta que cuanto más nos aproximamos a la temperatura "idle", más difícil resulta reducir la temperatura del procesador sin penalizar el rendimiento.

0.6. Conclusiones

La disipación de potencia, la energía y la temperatura han sido, durante los últimos años, los factores determinantes del diseño de microprocesadores. Con la brusca parada en el aumento de la frecuencia de funcionamiento de los procesadores, los ingenieros se han centrado en aumentar el número de núcleos dentro del procesador, en vez de aumentar el rendimiento individual de cada uno de ellos. Hoy en día, los procesadores multinúcleo (CMPs) son el diseño más elegido en un amplio rango de dispositivos: dispositivos móviles, ordenadores de sobremesa, portátiles, servidores, tarjetas gráficas, etc.

Podemos identificar dos componentes principales en la disipación de potencia, la dinámica y la estática. La disipación de potencia dinámica de una estructura es directamente

proporcional al uso que se hace de la misma (debido a la carga y descarga de los transistores). Por otro lado, la disipación de potencia estática viene dada por las corrientes de fuga que fluyen por el transistor aunque éste no esté siendo utilizado. La disipación de potencia estática aumenta con cada nueva escala de transistores, siendo especialmente grande en estructuras de tipo matricial de gran tamaño (cachés o tablas de predicción). Para tecnologías actuales (32nm), incluso con las corrientes de fuga de la puerta del transistor bajo control mediante el uso de dieléctricos *high-k*, la potencia estática supone una parte importante de la disipación de potencia total del procesador.

Quizá lo primero que debemos preguntarnos es: ¿Cómo medimos el consumo?. Hasta hace muy poco (la familia de procesadores "Sandy Bridge" de Intel), las estimaciones de consumo se realizaban basándose en los contadores de rendimiento del procesador, y se hacían cada miles de ciclos. En esta Tesis queremos utilizar mecanismos de grano fino que trabajan a nivel de ciclo para controlar el consumo del procesador, así que necesitamos un sistema de medición con mayor resolución. La primera contribución importante de esta Tesis es la introducción de los "Power Tokens". Mediante el uso de "Power Tokens" podemos aproximar de manera simple y precisa la potencia disipada por una instrucción en la etapa de confirmación sumando a su consumo base (acceso a estructuras que siempre utiliza) una componente variable que depende del tiempo que la instrucción permanece en el "pipeline" del procesador. La disipación de potencia total del procesador se puede estimar sumando los "power tokens" de todas las instrucciones que están actualmente en el "pipeline".

Mediante el uso de los "power tokens" hemos estudiado diferentes mecanismos de control de temperatura en microprocesadores, tanto a nivel de circuito como a nivel de microarquitectura. DVFS es un mecanismo a nivel de circuito que aprovecha la relación entre el consumo dinámico y el voltaje y frecuencia del transistor ($P_d \approx V_{DD}^2 \cdot f$) para ahorrar energía. El impacto en el rendimiento de DVFS sobre aplicaciones secuenciales o con cargas multiprogramadas viene dado únicamente por la reducción en la frecuencia. Además, la penalización en el rendimiento puede ser incluso menor si se trata de aplicaciones que hacen un uso intensivo de la memoria, ya que solo estamos escalando la frecuencia de funcionamiento del procesador, el tiempo de acceso a memoria se ve reducido. También hemos evaluado diversas técnicas de microarquitectura para reducir el consumo tales como "Critical Path Prediction", "Branch Confidence Estimation" o "Pipeline Throttling". Mientras que DVFS es bastante inestable a la hora de intentar adaptarse a un límite de consumo, todas las técnicas de microarquitectura son capaces de reducir el consumo de manera uniforme (sin muchos picos de consumo). Lamentablemente, la degradación de rendimiento de estas técnicas es bastante alta.

Durante toda la Tesis nos hemos centrado principalmente en la componente dinámica de la disipación de potencia, ya que a día de hoy es la principal fuente de disipación de potencia del procesador. En cualquier caso y, como se ha mencionado anteriormente, la componente estática está ganando importancia con cada nueva escala de integración de transistores. En nuestro caso, para el procesador que hemos modelado la componen-

te estática supone un 30 % de la disipación total del procesador. Por lo cual es muy importante tenerla en cuenta en las estimaciones de consumo y temperatura.

No se puede discutir la importancia que tiene el consumo energético en los nuevos diseños de procesadores, aún así en esta Tesis queríamos ir un paso más allá. Es bien sabido que los procesadores de propósito general se pueden utilizar en todo tipo de dispositivos los cuales suelen tener diferentes requisitos energéticos. En algunos casos sería interesante ser capaz de definir un límite en el consumo del procesador el cual nos permitiera adaptar el consumo del procesador a los requisitos del dispositivo. Además, no parece razonable diseñar el sistema de refrigeración de un procesador para el peor caso, ya que es muy improbable que el procesador alcance semejante temperatura. Por el contrario si somos capaces de crear un sistema de limitación de consumo en el procesador podríamos permitirle enfriarse si supera una determinada temperatura.

Durante nuestro análisis de distintas técnicas de reducción de consumo hemos descubierto que ninguna de ellas era capaz de adaptarse de manera precisa a un límite de consumo. Además, estos mecanismos funcionan de una manera reactiva, una vez se supera el límite de consumo, por lo que resulta imposible adaptarse a dicho límite de forma precisa. Por el contrario, podemos utilizar información histórica sobre el consumo del procesador para activar los mecanismos de manera predictiva, y aproximarnos mejor al límite de consumo. En este sentido proponemos dos técnicas, "Power Token Throttling" (PTT) y "Basic Block Level Manager" (BBLM). PTT es un mecanismo basado en "tokens" que monitoriza el consumo actual del procesador y bloquea la ejecución de nuevas instrucciones si superamos el límite de consumo. Este mecanismo es bastante agresivo (alta degradación del rendimiento), pero es extremadamente preciso a la hora de adaptarse al límite de consumo. Por otro lado, BBLM utiliza información histórica de consumo a nivel de bloque básico (expresada como "power tokens") para determinar qué tipo de mecanismo debe utilizar en función de cómo de lejos está el procesador del límite de consumo. Para mejorar aún más la precisión y el ahorro energético de estos mecanismos proponemos un mecanismo híbrido de dos niveles que combina las técnicas de microarquitectura con DVFS para sacar partido de sus principales cualidades. DVFS actúa como un mecanismo de grano grueso aproximando la media de consumo del procesador hacia el límite de consumo, mientras las técnicas de microarquitectura se centran en eliminar los numerosos picos de consumo que DVFS es incapaz de ver. Este mecanismo híbrido es capaz de superar a DVFS tanto en energía (11 % menos de energía) como en precisión (6 veces más preciso).

El siguiente paso lógico era migrar todos estos mecanismos a un entorno multinúcleo (CMP). Cuando analizamos dichos mecanismos en un CMP ejecutando cargas paralelas descubrimos que no eran capaces de adaptarse de manera precisa al límite de consumo, debido a las dependencias entre hilos de ejecución y a los puntos de sincronización. La degradación de rendimiento de DVFS sobre cargas paralelas varía dependiendo de cómo lo apliquemos. Si lo aplicamos a nivel de núcleo y retrasamos a un hilo crítico observaremos una degradación de rendimiento global debido a los puntos de sincronización. Si somos

capaces de balancear la frecuencia de ejecución de tal forma que ahorremos energía en hilos no críticos podemos reducir el consumo sin afectar al rendimiento. Por otro lado, si aplicamos DVFS a todos los núcleos del procesador simultáneamente estaremos en un caso similar al de los procesadores mononúcleo, con menor degradación de rendimiento para aplicaciones con uso intensivo de memoria. Necesitamos un mecanismo de control global que sea capaz de adaptarse a estas características.

Por ello presentamos "Power Token Balancing" (PTB). PTB balancea de manera dinámica el consumo de los distintos núcleos para asegurarse que el consumo total del procesador permanece por debajo del límite impuesto. Nuestro mecanismo monitoriza el consumo de los núcleos por debajo del límite de consumo y transfiere su excedente energético a los núcleos que están por encima del límite de consumo, por lo que dichos núcleos no tienen por qué restringir su funcionamiento para adaptarse a su límite de consumo local. Por lo tanto, PTB constituye un mecanismo de grano fino que garantiza una gran precisión con una mínima desviación del límite de consumo impuesto, lo que resulta crucial si estamos intentando optimizar los costes del empaquetamiento o aumentar el número de núcleos de nuestro procesador manteniendo un mismo TDP[2]. De todos modos se puede sacrificar parte de esa precisión para aumentar la eficiencia energética del mecanismo.

Como efecto colateral de la alta precisión a la hora de adaptarse al límite de consumo, PTB tiene otro beneficio añadido: una temperatura mucho más estable durante la ejecución del programa. Para las aplicaciones analizadas en esta Tesis, PTB es capaz de reducir las temperaturas pico y media del procesador en torno al 27-30 %. PTB también permite balancear la temperatura entre los distintos núcleos, reduciendo la temperatura del núcleo más caliente e igualándola a la del núcleo más frío del CMP. Esta reducción de temperatura no solo reduce la disipación de potencia estática, también reduce la tasa de fallos del procesador.

En el proceso de escalado de las tecnologías de fabricación la redes de interconexión no han seguido la misma tendencia que los transistores, convirtiéndose en un factor que limita tanto el rendimiento como el consumo. Una solución intuitiva para reducir la longitud de las líneas de datos de la red de interconexión consiste en apilar capas de procesadores unas sobre otras, en lugar de usar una distribución bidimensional. Este tipo de procesadores 3D prometen solventar los problemas de ancho de banda y latencia de comunicaciones de los procesadores. La principal desventaja es que al aumentar la densidad de núcleos por superficie también estamos aumentando la densidad de potencia y, en consecuencia, la temperatura del procesador. Existen mecanismos para controlar la temperatura del procesador y prevenir el sobrecalentamiento, reduciendo el consumo. Lamentablemente ni DVFS ni la migración de procesos (las técnicas más comunes para reducir el consumo y temperatura) ofrecen soluciones precisas para adaptarse a estos límites de consumo, por eso decidimos evaluar nuestros mecanismos de control de consumo en procesadores multinúcleo 3D.

En este escenario PTB es capaz de reducir la temperatura media entre 2ºC y 20ºC de-

[2]Thermal Design Power.

pendiendo de cómo estén organizadas las estructuras del procesador. Para las estructuras más calientes, PTB es capaz de reducir su temperatura en un 40 % para un procesador de 16 núcleos y 4 capas. También proponemos "Token3D", una nueva política de distribución de energía para PTB que tiene en cuenta información de temperatura y mapeado de estructuras a la hora de balancear la energía del procesador, dando prioridad a los procesadores más fríos sobre los calientes. Esta nueva política mejora a PTB, reduciendo la temperatura del procesador un 3 % adicional. Para finalizar, hemos diseñado una organización híbrida que mezcla diseños horizontales y verticales de procesadores 3D, tratando de minimizar la temperatura. En este diseño las estructuras más calientes del núcleo se sitúan en las capas más externas del procesador, dejando las estructuras frías en las capas más internas del procesador. Dicha organización es capaz de reducir la temperatura del núcleo un 10 % sobre el mejor diseño horizontal y un 85 % sobre los diseños verticales.

Chapter 1

Introduction

Summary: In Chapter 1 we present a brief introduction to this Thesis. We will disclose two of the key design limitations in modern microprocessors, power dissipation, and the direct problem related to it, overheating. Both power and thermal related issues will be the primary objective that motivates this Thesis. This chapter also summarizes the main contributions of this Thesis and presents the overall organization of the document.

1.1. Motivation

In the last decade computer engineers have faced changes in the way microprocessors are designed. New microprocessors do not only need to be faster than the previous generation, but also be feasible in terms of energy consumption and thermal dissipation, especially in battery operated devices. With the megahertz race over, engineers have focused on increasing the number of processing cores available on the same die. Nowadays, chip multiprocessors (CMPs) are the new standard design for a wide range of microprocessors: mobile devices (in the near future almost every smartphone will be governed by a multicore CPU), desktop computers, laptop, servers, GPUs, APUs, etc. These microprocessors face constant thermal and power related problems during their everyday use.

Like any other transistor-based electronic device, microprocessors have two sources of power dissipation, dynamic power dissipation and static power dissipation (or leakage). Dynamic power dissipation is proportional to usage (every time we access a structure), due to the constant charge and discharge of transistors. On the other hand, static power dissipation is derived from gate leakage and subthreshold leakage currents that flow even when the transistor is not in use. As process technology advances toward deep submicron, the static power component becomes a serious problem, especially for large on-chip array structures such as caches or prediction tables. The leakage component is something that many studies do not take into consideration when dealing with temperature, but it cannot

be ignored. For current technologies (under 32nm), even with gate leakage under control by using high-k dielectrics, subthreshold leakage has a great impact in the total power dissipated by processors [47]. Moreover, leakage depends on temperature as well, so it is crucial to add a leakage-temperature loop to update leakage measurements in real time depending on the core/structure's temperature.

Thermal and power related issues are usually solved by applying a power budget to the processor/core, usually triggered by a Dynamic Thermal Management (DTM) mechanism. Dynamic Voltage and Frequency Scaling (DVFS) has been an effective technique that allowed microprocessors to match a predefined power budget. However, the continuous shrinking in the building process technology will eventually make leakage power the main source of power dissipation. In that scenario DVFS becomes less effective. DVFS mainly relies on decreasing supply voltage, however, if we want to maintain the transistor's switching speed, threshold voltage must be also lowered. Nevertheless, decreasing the threshold voltage increases leakage power dissipation exponentially. In a CMP scenario it is possible to use legacy (i.e., single-core) power saving techniques if the CMP runs either sequential applications or independent multithreaded workloads. However, new challenges arise when running parallel shared-memory applications. In the later case, sacrificing some performance in a single core (thread) in order to be more energy-efficient might unintentionally delay the rest of cores (threads) due to synchronization points (locks/barriers), therefore, degrading the performance of the whole application.

The use of CMPs to increase performance by exploiting parallelism has two additional drawbacks: off-chip bandwidth and communication latency between cores. 3D die-stacked processors are a recent design trend aimed at overcoming these drawbacks by stacking multiple device layers. However, the increase in packing density of these designs leads to an increase in power density, which translates into more thermal problems.

Moreover, there are times when power requirements of a microprocessor exceed the available power of the target device. In most cases it is not possible to design a new processor to match up whatever power requirements, because it would be too expensive. The problem gets even worse if the power restrictions are temporary (i.e. overheating), and after that period of power restrictions we want to reestablish the regular working mode of the processor. Imagine a computation cluster connected to one or more UPS units to protect from power failures. If there is a power cut, all processors will continue working at full speed consuming all of the UPS batteries quickly, and then switching the computers off when the battery is about to run out and, consequently, losing all the work on fly. During the power failure (many times they are of limited duration), if the processors are not doing critical work, it might be better to extend the UPS battery duration at the expense of degrading the performance, than to lose all the work done. Another situation where setting a power budget could be useful is a computing centre that shares a power supply among all kind of electric devices (i.e., computers, lights, air conditioning, etc.). In a worst case scenario (e.g., in summer at mid-day with all the computers working at full speed), if we integrate some kind of power budget management into the processors, we

could decrease the power of all processors, lowering the ambient temperature and having more power for the air conditioning. In this way, we could design the power capacity of the computing centre for the average case, reducing its cost. A new need arises in microprocessor designs, to include a mechanism that allows the processor to match a temporal power constraint, even at the cost of performance degradation.

1.2. Contributions

The main contributions of this Thesis have been published in [23, 24, 25, 26, 27] and are listed below:

- We introduce the concept of Power Tokens. In most modern microprocessors there is no way to accurately measure power in real time. Power measurements are just approximations based on performance counters done over periods of thousands of cycles. When using microarchitectural techniques that work at a cycle level we need some way to estimate power at a more fine-grain level, that is why we propose the Power-Token approach. The rationale behind is to calculate the energy consumed by an instruction at commit stage by adding the base energy consumption of the instruction (i.e., all regular accesses to structures done by that instruction) plus a dynamic component that depends on the time it spends on the pipeline. The new Sandy Bridge processor from Intel has some machine specific registers (MSRs) that provide per structure power information. This new feature validates that the decisions we made three years ago when we introduced the Power Tokens where in the right direction.

- We analyze performance, power and energy impact of different power saving mechanisms, initially, in a single-core scenario. In order to manage dynamic power dissipation we will evaluate: DVFS, that modifies both voltage and frequency in order to save power; Pipeline Throttling, that reduces the amount of instructions fetched by the processor in order to save power; Instruction Criticality, that delays the execution of non-critical instructions from cycles over the power budget to cycles under the power budget; and Confidence Estimation, that reduces the amount of speculation in low-confidence paths (the ones that correspond to branches that cannot be accurately predicted). For managing static power dissipation (leakage) we will analyze the effects of two techniques: decay-based techniques, that switch-off entries of an structure, reducing it's leakage to zero but losing its contents (non-state preserving techniques); and drowsy-based techniques, that lower a structure's voltage, obtaining less power savings but being able to restore the state of the structure in a reasonable time (state preserving techniques). Both state and non-state preserving techniques will be evaluated in the context of value predictors, as a case study of a regular array-based structure.

- Also in the single-core scenario we analyze the effects of different legacy power

saving techniques when trying to match a predefined power budget and discovered that these techniques are not suitable to perform that task with reasonable accuracy and performance penalties. To deal with this problem this Thesis proposes the use of new hybrid microarchitectural techniques to accurately match a power constraint while maximizing the energy-efficiency of the processor. In this context we introduce Power Token Throttling (PTT), a token-based approach that keeps track of the current power being dissipated by the processor, measured as tokens, and stalls the fetch stage if the next instruction to be fetched and executed will make the processor go over the power budget. We also introduce and evaluate a basic block-level mechanism, that keeps track of the power dissipated the last time the processor executed a basic block and uses that power translated into tokens to select between different power-saving microarchitectural techniques. These techniques are orthogonal to DVFS so they can be simultaneously applied. We will also introduce a two-level approach where DVFS acts as a coarse-grain technique to lower the average power dissipation towards the power budget, while microarchitectural techniques focus on removing the numerous power spikes.

- In the multi-core scenario we introduce Power Token Balancing (PTB), a mechanism aimed at accurately matching an external power constraint by balancing the power dissipated among the different cores using a Power Token-based approach while optimizing the energy efficiency. We can use power (seen as tokens or coupons) from non-critical threads for the benefit of critical threads. PTB runs transparent for thread independent / multiprogrammed workloads and can also be used as a spinlock detector based on power patterns.

- Also in the multi-core scenario we analyze the use of microarchitectural power budget techniques to reduce peak temperature in 3D die-stacked architectures. In particular, we introduce Token3D, a power balancing technique that takes into account temperature and layout information to balance the available per-core power along with other power optimizations for 3D designs. We also analyze a wide range of floorplans looking for the optimal temperature configuration. Finally, we present some optimizations for vertical 3D designs and a custom floorplan design that places high temperature structures in upper layers of the 3D stack and cool structures in inner layers to reduce overall peak temperature.

1.3. Organization

This Thesis is organized as follows:

- **Chapter 2: Problem statement and simulation methodology**. In this chapter we explain the importance of power dissipation and temperature in microprocessor designs. We propose a mechanism to estimate power in real-time (Power

Tokens), that will be used to control power and study the effects on both performance and temperature of power saving mechanisms. This chapter also analyzes the simulators, the benchmarks and the different power and temperature models we have evaluated.

- **Chapter 3: Power saving mechanisms**. In this chapter we will analyze the effects on power (static and dynamic), energy and performance of different previously proposed power saving mechanisms for the single-core scenario. The studied dynamic power saving mechanisms are based on DVFS, instruction criticality, branch confidence estimation and fetch throttling. As a case study we will analyze some specific mechanisms to control leakage power in cache-like structures, more specifically in Value Predictors.

- **Chapter 4: Single-core power budget matching**. In this chapter we recall the concept of power budget, why it is important and how to properly enforce it. In the previous chapter we analyzed the effects of power control mechanisms on performance and total cycles and energy over the power budget. This analysis showed that most of these mechanisms have problems when matching the target power constraint on their own, so it is not well respected. In order to solve this problem we propose three mechanisms: Power Token Throttling (PTT), Basic Block Level Manager (BBLM) and a two-level approach (DVFS+BBLM). PTT is a token-based approach that keeps track of the current power being dissipated by the processor, measured as tokens, and stalls fetch if the next instruction to be executed will make the processor go over the power budget. BBLM is a predictive power-control mechanism that stores information about the energy consumption of a basic block (or chain of basic blocks). BBLM will select from different architecture-level power saving mechanisms (more or less aggressive) based on the energy this basic block consumed the last time it was executed. Finally, the two-level approach will use DVFS as a coarse-grain approach to lower the average power dissipation towards the power budget along with BBLM to remove the remaining power spikes.

- **Chapter 5: CMP power budget matching**. In this chapter we adapt and analyze the proposed mechanisms from Chapter 4 in a CMP scenario. After an initial analysis we discovered that the proposed mechanisms work properly for both multiprogrammed and multiple instances of sequential applications, but are unable to match the target power budget when running parallel applications because of dependencies between threads. We believe that a global mechanism is required to properly adapt energy consumption of each core of the CMP to accurately match the target power budget with minimal performance impact, and thus, we propose PTB. PTB is based on a centralized structure, the power token balancer, that receives power information from all the cores under the power budget, measured as tokens, and gives those tokens to cores over the power budget, so they can continue execution without applying any power control mechanism, ensuring that the global

power constraint is satisfied.

- **Chapter 6: 3D die-stacked power budget matching - Token3D**. In this chapter we introduce and study a new design field, 3D die-stacked cores. Although 3D die-stacked designs reduce the latency and power gap between the interconnection network and the CMP cores, it has some important drawbacks. First, the increase in packing density also leads to an increase in power density that eventually translates into thermal problems. Second, a deeper design space exploration of different floorplan configurations is essential to take advantage of these emerging 3D technologies. Third, chip verification complexity increases with the number of layers. In order to minimize thermal related problems we port all the previously proposed mechanisms from chapters 4 and 5 to a 3D die-stacked scenario and study the effects they have on performance, energy and temperature. In addition we study new floorplan designs and other specific optimizations based on instruction window resizing and ALU selection.

- **Chapter 7: Conclusions and future ways**. In this chapter we discuss the conclusions to our work and some future lines that can be followed in the energy-efficient processor design field.

Chapter 2

Problem Statement and Simulation Methodology

Summary:

In this chapter we discuss the importance of energy consumption and temperature in microprocessor designs. At first we propose a mechanism to estimate power in real-time (Power Tokens), that will be used to control power and study the effects on both performance and temperature of power saving mechanisms. We reason about the importance of accuracy when matching the power budget, and propose the use of a new metric, Area Over Power Budget (AoPB), that will help us estimate the accuracy of the studied power saving mechanisms. We also introduce the different power and temperature models, the simulators used and the studied benchmarks during this Thesis.

2.1. Introduction

As device size shrinks and operating frequencies rise, power dissipation and thermal output have become the main limiting factors in microprocessor design. From 1986 to 2002, performance increased at a rate of 52% per year. This trend slowed down in 2002 and performance only increased by around 6% each year, until now [74]. This rapid increase of performance was known as the "Megahertz Race", due to the continuous increase in the clock rates of microprocessors. As we will see later, there is a direct relation between clock rate and power dissipation/temperature, and processors rapidly hit what is known as the "Power Wall". Between 2002 and 2005 microprocessor designers spent many resources (transistors) to build complex structures that only achieved minimal performance increments. The amount of available transistors on the die due to feature shrinking was much higher than the performance architects could get from them. Moreover, if they tried to simply increase the clock rate of so many transistors, the heat dissipated by the core was so high that it will immediately burn out. This slow intra-core ILP and long communication latencies in wiredelays in large wide-issue superscalar processors lead designers to use those transistors to build more cores inside the same die rather than to

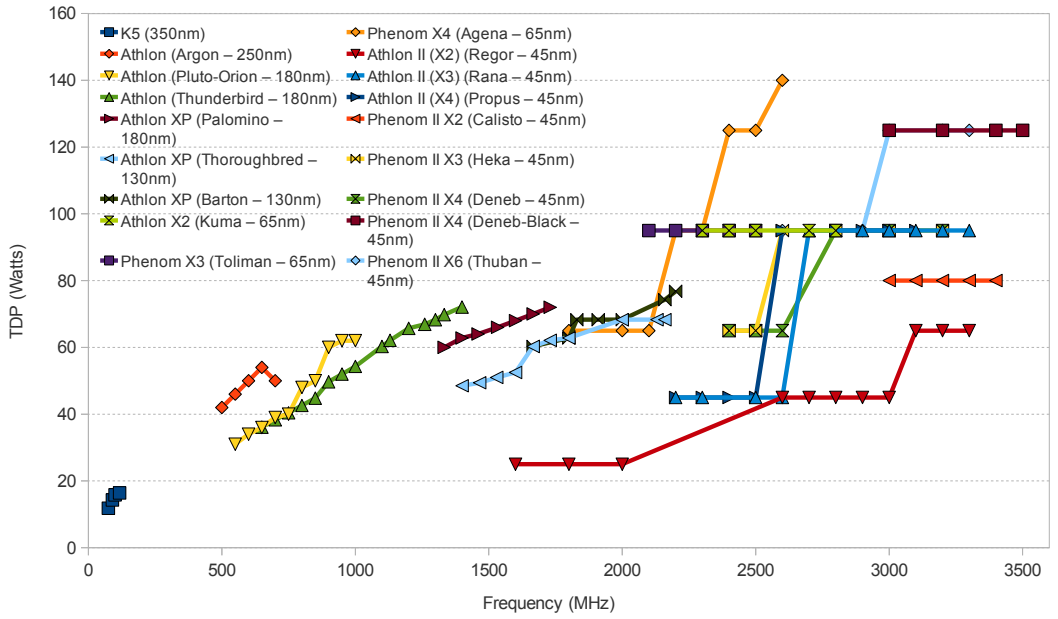


Figure 2.1: Evolution of the thermal design power for AMD microprocessors.

"increase" the per-core performance. To prevent the cores from overheating, frequencies got stalled in the range of the 2-3Ghz. Nowadays, even if the supply voltage is lowered with each new generation of transistors to reduce dynamic power, the continuous increase of leakage power and the additional power dissipation from complex interconnection networks in chip multiprocessors (CMPs) plus the increasing on-die memory size leads to a global increase on the total power and energy consumed by the processor. Therefore, we can state that the power wall is also limiting the amount of cores per processor.

The thermal design power (TDP), sometimes called thermal design point, represents the maximum amount of power the cooling system in a computer is required to dissipate. It can be achieved by using an active cooling method such as a fan or any of the three passive cooling methods: convection, thermal radiation or conduction. Typically, a combination of methods is used. The TDP is typically not the higher (peak) power the chip could ever draw, but rather the maximum power that it would draw when running real applications. This ensures that the computer will be able to handle, essentially, all applications without exceeding its thermal envelope or requiring a cooling system for the maximum theoretical power, which would cost more. Figures 2.1 and 2.2 show the TDP evolution for different families of AMD and Intel microprocessors, respectively. We can clearly see a dependence on the cooling requirements as the frequency is increased, due to the direct dependency between power dissipation and the processor's voltage and frequency ($P = CV_{DD}^2 Af$). Although the per core TDP is lowered after the P4 and the Athlon XP for each generation of microprocessors (currently around 10W per core), the increasing amount on the number of cores inside the CMP leads to a total TDP that sometimes goes over 125W, for those processors with high overclocking capabilities (*Extreme Edition* from Intel or *Black Edition* from AMD).

In some designs the TDP is under-estimated and in some real applications (typically

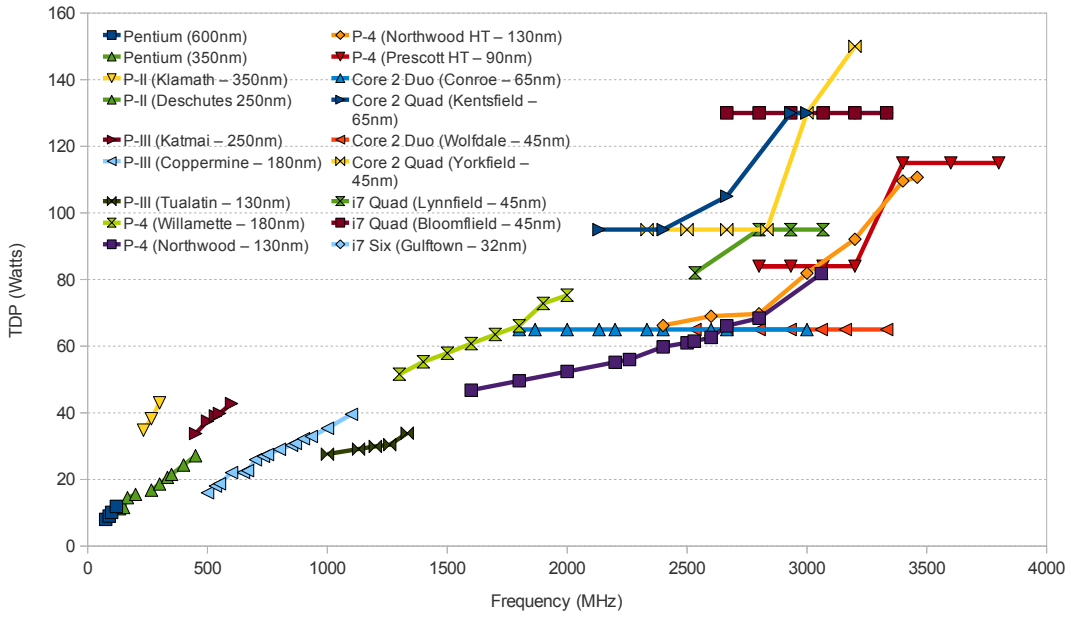


Figure 2.2: Evolution of the thermal design power for Intel microprocessors.

strenuous, such as video encoding or games) the CPU exceeds the TDP. In this case, the CPU will either cause a system failure (a "therm-trip") or will throttle its speed down using a DTM policy. Most modern CPUs will only cause a therm-trip on a catastrophic cooling failure such as a stuck fan or a loose heatsink. Moreover, in most modern processors TDPs are often specified for families of processors, with the low-end models usually using significantly less power than those at the high end of the family.

As mentioned in the previous chapter, one possible way to continue increasing the number of cores inside a CMP avoiding the interconnection latency and offchip bandwidth problems is the use of 3D die-stacked designs. Temperature management gets even more difficult in this scenario. It is not feasible to stack several processor dies on top of each other with a large number of cores per layer using current process technologies (32nm) due to thermal problems [13][28][79]. However, some companies like Toshiba are putting lots of efforts to build 3D DRAM, as memory chips are far cooler than regular processor chips. There is also the possibility that in the near future microprocessors will include some internal memory layers to act as an additional memory hierarchy level.

Dynamic Thermal Management (DTM) is a mechanism that reduces the processor energy consumption (and therefore performance) during critical time intervals so it can cool down. One way to achieve this goal is to set a power budget to the processor. This processor's power budget is not only useful to control power and temperature but also to adapt to external power constraints. There are situations where device power constraints are more restrictive than the power needs of a processor at full speed. In most of the cases we cannot afford to design a new processor to meet whatever power constraint because is too expensive. Therefore, being able to set a power budget to the processor could help designers to reuse existent hardware in new devices. The problem grows when, as usual, power constraints are transitory and after some period of low power dissipation we want

to restore the processor's regular behavior.
The main goal of this Thesis is to provide architecture-level mechanisms that allow the microprocessor to adapt to a predefined power constraint in an energy efficient way, while being as accurate as possible. This minimal deviation from the target power budget will translate in minimal variations of the processor temperature during runtime. Minimizing the spatial gradient (difference between temperatures within the die) will ensure a more reliable microprocessor and reduce the fault ratio. However, there are several requirements to achieve this goal:

- Provide the chip infrastructure to account power in real time.
- Design a metric that measures accuracy when matching a predefined power budget.
- Reduce speculation to save power from wrong path instructions when exceeding the power budget.
- Prioritize critical instructions, i.e., delay non-critical instructions from cycles over the power budget to cycles under the power budget.
- In a CMP scenario, efficiently balance the available power between the different cores to benefit critical threads.
- If the processor is still over the power budget, use a more aggressive power saving mechanism, like pipeline throttling or dynamic voltage and frequency scaling (DVFS).

We will address all these points in this Thesis, along with some floorplan organization/optimizations to reduce temperature in different scenarios.

2.2. Power Tokens

Up until recently (Intel Sandy Bridge processors) there was no way to accurately measure power dissipation in real time. Power measurements are just approximations based on performance counters done over periods of thousands of cycles. When using microarchitectural techniques that work at a very fine granularity (from several to a few tens of cycles) we need some way to estimate power at this fine granularity. That is why we propose the Power-Token approach, which calculates the power dissipated by an instruction at commit stage by adding the base power dissipation of the instruction (i.e., all regular accesses to structures done by that instruction) plus a dynamic component that depends on the time it spends on the pipeline. The dynamic component corresponds to the combined RUU[1] wakeup-matching logic power that we divide between all the active instructions in the RUU. Therefore, in order to work with power-token units in a simple way, we define a power-token unit as the joules dissipated by one instruction staying in the RUU for one cycle.

[1] Register Update Unit.

We calculated the base power-tokens of every instruction type using the SPECINT2000 benchmark suite. Once we had the base power for all the possible instructions, we used a K-mean algorithm to group instructions with similar base power dissipation. We analyzed independent instructions and several grouping, that ranged from 32 groups to 4 groups. Simulated results showed that having just 8 groups of instructions is accurate enough for the Power-Token approach to properly work with a deviation lower than 1% (compared to accounting for the actual energy consumption as provided by the evaluated simulator, HotLeakage).

In this way, the number of power-tokens dissipated by an instruction will be calculated as the addition of its base power-tokens plus the amount of cycles it spends in the RUU. The implementation of the Power-Token approach is done by means of an 8K-entry history table[2] (Power-Token History Table - PTHT), accessed by PC, which stores the power cost (in power-token units) of each instruction's previous execution. This table introduces a measured power overhead of around 0.5% which is accounted for in our results. We also need five 16 bit adders for accounting the power tokens in our modeled processor (four wide processor plus one for adding the total power) and a register to store the current power dissipated in the processor (0,025% power overhead). When an instruction commits, the PTHT is updated with the number of power-tokens dissipated. Finally, the overall processor power dissipation (in power-token units) can be easily estimated in a given cycle based on the instructions that are traversing the pipeline without using performance counters by simply accumulating the power-tokens (as provided by the PTHT) of each instruction being fetched. Our evaluated cores implement a three-stage fetch unit (Figure 2.7) so there is enough time to estimate this power.

2.3. Importance of Accuracy

This Thesis is focused on adapting the processor's power dissipation to a target power constraint, with minimal deviation from the target power budget while minimizing energy increase and performance slowdown.

If we want to ensure minimal deviation from the target power budget we need to introduce a metric that enables us to measure how close we are from the power constraint. We define the metric Area over the Power Budget (AoPB) as the amount of energy (joules) between the power budget and each core dynamic power curve, represented by shadowed areas in Figure 2.3. The lower the area (energy) the more accurate the analyzed technique is (note that the ideal AoPB is zero).

Lets us provide an example that tries to illustrate the importance of the accuracy on matching a predefined power budget. Imagine that we want to increase the number of cores in a CMP while maintaining the same TDP (this is a common practice done by microprocessor manufacturers in the past 5 years, as depicted in Figures 2.1 and 2.2). For a 16-core CMP with a 100W TDP, each core would use 6.25W (for simplicity let us

[2] 8 bits per entry.

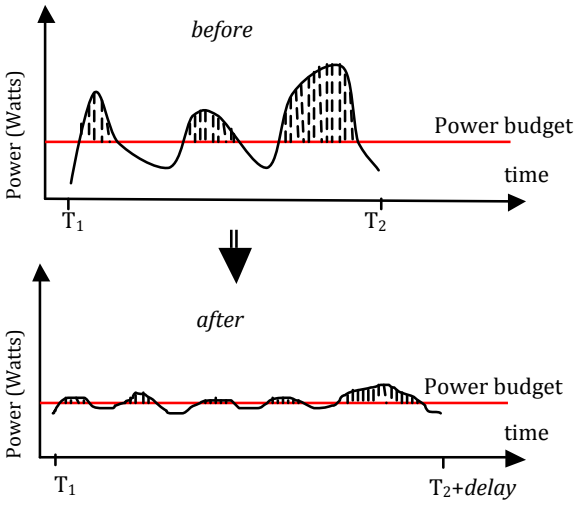


Figure 2.3: Example of the Area over Power Budget (AoPB) metric. Shadowed areas represent the energy consumed over the target power budget.

ignore the interconnection network). If we set a power budget of 50% we could ideally duplicate the number of cores in that CMP with the same TDP (up to 32 cores, each one consuming an average of 3.125W). But for this ideal case a perfect accuracy on matching the power budget is needed.

As we will see later on this Thesis, standard DVFS incurs in a energy deviation of 65% over the power budget (the AoPB metric). Therefore, with a 65% deviation each core dissipation raises to 3.125*1.65=5.15W, and, for a 100W TDP, we can put a maximum of 100/5.15=19 cores inside the CMP. Using a two-level approach (without PTB - Chapter 4) the deviation is reduced to 40%. This gives us a potential average power dissipation of 3.125*1.40=4.375W per core, so we can put 100/4.375=22 cores in the CMP with the same TDP. Finally, when using the non-relaxed PTB (Chapter 5) approach the deviation is reduced below 10%, that gives us a potential average power dissipation of 3.125*1.1=3.4375W per core, so we can put 100/3.4375=29 cores inside our CMP. Therefore, thanks to the extra cores (we could go from 16 cores in the original CMP design to 29 cores) we can perfectly overcome the 3% performance degradation that results from using our proposed PTB mechanism if the application is parallel enough to use these extra cores.

2.4. Simulators

During the past five years we have been researching on different proposals that involved working in various environments, from single-core microprocessors to 3D die-stacked multiprocessors. This section summarizes all the simulators used in the development of this Thesis.

- **SimpleScalar**. SimpleScalar [5] is an open source computer architecture simulator developed by Todd Austin while he was a PhD student at the University of

Wisconsin-Madison. This system software infrastructure is used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Using the SimpleScalar tools, users can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. The SimpleScalar tools are used widely for research and instruction, for example, in 2000 more than one third of all papers published in top computer architecture conferences used the SimpleScalar tools to evaluate their designs. In addition to simulators, the SimpleScalar tool set includes performance visualization tools, statistical analysis resources, and debug and verification infrastructure.

- **Wattch**. Wattch [18] is a framework for analyzing and optimizing microprocessor power dissipation at the architecture-level. Wattch is 1000X or more faster than existing layout-level power tools, and yet maintains accuracy within 10% of their estimates as verified using industry tools on leading-edge designs. Wattch can be seen as a complement to existing lower-level tools; it allows architects to explore and cull the design space early on, using faster, higher-level tools. It also opens up the field of power-efficient computing to a wider range of researchers by providing a power evaluation methodology within the portable and familiar SimpleScalar framework.

- **HotLeakage**. HotLeakage [100] is a software model of leakage based on BSIM3 technology data that is publicly available on the web. It is computationally very simple and can easily be integrated into popular power-performance simulators like Wattch. It can also be easily extended to accommodate other technology models and can be used to model leakage in a variety of structures (not just caches, which have been the focus of a lot of prior work). HotLeakage extends the Butts-Sohi model [19] and corrects several important sources of inaccuracy. This model is called HotLeakage, because it includes the exponential effects of temperature on leakage. Temperature effects are important, because leakage current depends exponentially on temperature. In fact, HotLeakage also includes the heretofore unmodeled effects of supply voltage, gate leakage, and parameter variations. HotLeakage has circuit-level accuracy because the parameters are derived from transistor-level simulation (Cadence tools). Yet like the Butts and Sohi model, simplicity is maintained by deriving the necessary circuit-level model for individual cells, like memory cells or decoder circuits, and then taking advantage of the regularity of major structures to develop abstract models that can be expressed in simple formulas similar to the Butts-Sohi model.

- **CACTI**. CACTI [90] is an integrated cache and memory access time, cycle time,

area, leakage, and dynamic power model. By integrating all these models together, users can have confidence that tradeoffs between time, power, and area are all based on the same assumptions and, hence, are mutually consistent. CACTI is intended for use by computer architects to better understand the performance tradeoffs inherent in memory system organizations.

- **Simics**. Simics [65] is a full-system simulator used to run unchanged production binaries of the target hardware at high-performance speeds. Simics was originally developed by the Swedish Institute of Computer Science (SICS), and then spun off to Virtutech for commercial development in 1998. Simics can simulate systems such as Alpha, x86-64, IA-64, ARM, MIPS (32- and 64-bit), MSP430, PowerPC (32- and 64-bit), POWER, SPARC-V8 and V9, and x86 CPUs. Many operating systems have been run on various varieties of the simulated hardware, including MS-DOS, Windows, VxWorks, OSE, Solaris, FreeBSD, Linux, QNX, and RTEMS. The NetBSD AMD64 port was initially developed using Simics before the public release of the chip. The purpose of simulation in Simics is often to develop software for a particular type of embedded hardware, using Simics as a virtual platform.

- **GEMS (Opal + Ruby)**. The Wisconsin Multifacet Project [67] presented the open-source General Execution-driven Multiprocessor Simulator (GEMS). GEMS is a set of modules for Virtutech Simics that enables detailed simulation of multiprocessor systems, including CMPs. GEMS leverages the potential of Virtutech Simics to simulate a Sparc multiprocessor system. This enables the simulation of commercial software such as database systems running on the Solaris operating system. By off-loading the correctness requirement to Simics, GEMS "timing-first" simulation can focus on accurate performance modeling rather than correctness details.

  - The Opal module provides a detailed out-of-order processor model. Opal is flexible and highly configurable to model different branch predictors, issue-widths, execution resources, etc.
  - The Ruby module provides a detailed memory system simulator. It can simulate a wide variety of memory hierarchies and systems ranging from broadcast-based SMP to a hierarchical directory-based Multiple-CMP system.

- **McPAT**. McPAT (Multicore Power, Area, and Timing) [56] is an integrated power, area, and timing modeling framework for multithreaded, multicore, and manycore architectures. It supports comprehensive early stage design space exploration for multicore and manycore processor configurations ranging from 90nm to 22nm and beyond. McPAT includes models for the components of a complete chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, and integrated memory controllers. McPAT models timing, area,

and dynamic, short-circuit, and leakage power for each of the device types forecast in the ITRS roadmap including bulk CMOS, SOI, and double-gate transistors. McPAT has a flexible XML interface to facilitate its use with different performance simulators.

- **HotSpot**. HotSpot [88] is an accurate and fast thermal model suitable for use in architectural studies. It is based on an equivalent circuit of thermal resistances and capacitances that correspond to microarchitecture blocks and essential aspects of the thermal package. The model has been validated using finite element simulation. HotSpot has a simple set of interfaces and hence can be integrated with most power-performance simulators like Wattch or Hotleakage. The foremost advantage of HotSpot is that it is compatible with the kinds of power/performance models used in the computer-architecture community, requiring no detailed design or synthesis description. HotSpot makes it possible to study thermal evolution over long periods of real, full-length applications.

2.5. Power Models

The previously described simulators implement different power models to account both dynamic and static power dissipation of the microprocessor. In this section we want to discuss the basics of these power models.

2.5.1. Dynamic Power Models

Dynamic power has been the dominant component in total power dissipation for many years. Dynamic power can be approximated by the following formula:

$$P = CV_{DD}^2 Af \tag{2.1}$$

Where C is the load capacitance, V_{DD} is the supply voltage, A is the activity factor and f is the operating frequency [47]. Each of these components is described in more detail below.

Capacitance (C). Load capacitance largely depends on the wire lengths of on-chip structures. There are several ways designers can influence this metric. The introduction of 3D die-stacked architectures is likely to reduce average wire length both inter-structure/core if we adopt a vertical design (where structures are split across layers) or inter-core if we adopt a typical horizontal design.

Supply Voltage (V_{DD}). Supply voltage is the power input of the transistor. It drops with every new generation of transistor, and is one of the key components in power-aware designs because of its quadratic influence on dynamic power.

Activity Factor (A). The activity factor is a number between 0 and 1 that refers to how often wires actually transition from 0 to 1 or 1 to 0. While the clock signal obviously switches at its full frequency, most other wires in the design have activity factors below

1. Strategies such as clock gating are used to save power by reducing activity factors during idle periods. Clock gating adds an AND gate to the clock signal that powers an specific unit or structure with a control signal. If the control signal is on, the unit will be clocked as expected. If the unit is unneeded for a cycle or more, the control signal can be set to 0, in which case the unit will not be clocked.

Frequency (f). The clock frequency has a fundamental and far-reaching impact on power dissipation. Not only does clock frequency directly influence power dissipation, but it also indirectly shapes power by its effect on supply voltage. Typically, maintaining higher clock frequencies may require maintaining a higher supply voltage. Thus, the combined $V_{DD}^2 f$ portion of the dynamic power equation has a cubic impact on power dissipation. Strategies such as DVFS can be applied on periods of low performance (i.e. memory-bound or latency-tolerant regions of code) and reduce voltage and frequency accordingly.

At a high level, dynamic power models can be divided into analytical and empirical techniques. Analytical techniques seek to express power behavior in terms of equations parametrized by model size or other characteristics. Empirical techniques, in contrast, have focused on predicting the behavior of one possible chip design by appropriately scaling per-module power behaviors observed for some other measured chip design.

Both capacitance and activity factor are expressions where the architect has some high-level understanding and control, even though the ultimate details are dependent on the particulars of the circuit design chosen. The activity factor is related both to the application program being executed (both its data patterns and control) and to some circuit design choices. For example, for circuits that pre-charge and discharge on every cycle (i.e., double-ended array bitlines) an activity factor of 1 is used. For wires that represent data buses, the activity factor can be chosen based on knowledge of the I/O statistics in the data set being studied. Capacitance, like activity factor, depends in part on circuit design choices. Even in relatively regular array structures, the aspect ratio, number of wire routing layers, or other layout choices can influence capacitance. Dynamic power models extracted from [18] are summarized in Table 2.1.

Table 2.1: Equations for Wattch power models. $C_{diff} C_{gate} C_{metal}$ represent diffusion, transistor gate and metal wire capacitances.

Regfile Wordline Capacitance	$C_{diff}(WordLineDriver) + C_{gate}(CellAccess)$ $\cdot NumBitLines + C_{metal} \cdot WordLineLength$
Regfile Bitline Capacitance	$C_{diff}(PreCharge) + C_{diff}(CellAccess)$ $\cdot NumWordLines + C_{metal} \cdot BitLineLength$
CAM Tagline Capacitance	$C_{gate}(CompareEn) \cdot NumberTags$ $+C_{diff}(CompareDriver) + C_{metal} \cdot TagLineLength$
CAM Matchline Capacitance	$2 \cdot C_{diff}(CompareEn) \cdot TagSize$ $+C_{diff}(MatchPrecharge)$ $+C_{diff}(MatchOR) + C_{metal} \cdot MatchLineLength$
ResultBus Capacitance	$0.5 \cdot C_{metal} \cdot (NumALU \cdot AluHeight)$ $+C_{metal} \cdot RegfileHeight$

Now we will give a description of how Wattch approximates the power dissipation of the different structures.

Array Structures. Array structures power models are parameterized based on the number of rows (entries), columns (width of each entry), and the number of read/write ports. These parameters affect the size and number of decoders, the number of wordlines and the number of bitlines. In addition, these models also use the specified parameters to estimate the length of the pre-decode wires as well as the lengths of the array's wordlines and bitlines. For array structures the power model is divided in the following stages: decoder, wordline drive, bitline discharge and output drive (or sense amplifier). Wordline drive and bitline discharge are the bulk of the power dissipation in array structures. Equations for wordline and bitline capacitances are shown in Table 2.1.
Modeling the power dissipation of wordlines and bitlines requires estimating the total capacitance on both of these lines. The capacitance of the wordlines include three main components, diffusion capacitance of the wordline driver, the gate capacitance of the cell access transistor times the number of bitlines and the capacitance of the wordline's metal wire.
The bitline capacitance is computed similarly. The total capacitance is equal to the diffusion capacitance of the pre-charge transistor, the diffusion capacitance of the cell access transistor multiplied by the number of word lines, and the metal capacitance of the bitline. These models allow the use of both single-ended or double-ended bitlines. Wattch assumes that register file array structures use single-ended bitlines while cache array structures use double-ended bitlines.
Multiple ports on the array structure will increase the power dissipation in three ways. First, there will be more capacitance on the wordlines because each additional port requires an additional transistor connection. Second, each additional port requires up to two additional bitlines, which must precharge/evaluate on every cycle. Finally, each core cell becomes larger which leads to longer word and bitlines, incurring additional wire capacitance.
Transistor sizing plays an important role in the amount of capacitance within the various structures. Generally, transistors in array structures are kept relatively small to reduce the area. In their model, certain critical transistors are automatically sized based on the model parameters to achieve reasonable delays. Moreover, they rely on CACTI tool to determine delay-optimal cache hardware configurations and squarify array-based structures.

Content-addressable memory (CAM) structures. CAM power models are very similar to the ones proposed for array structures. However, in the CAM structures taglines and matchlines are modeled instead of bitlines and wordlines. Equations for tagline and matchline capacitances are also shown in Table 2.1. Parameters of the CAM structures include: number of rows (tags) that depends on the instruction window size,

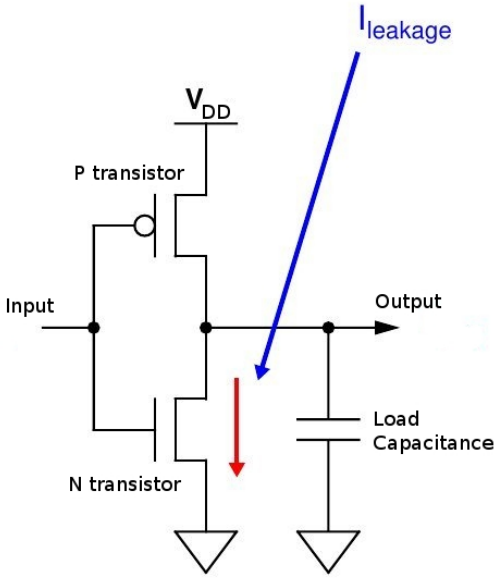


Figure 2.4: High-level CMOS Inverter Diagram.

columns (bits per tag to match) that is dependent on both the issue/commit width of the machine and the physical register tag size and finally the ports on the CAM.

Table 2.2: K_{design} parameters for typical circuits.

Circuit	*N*	K_{design}	*Notes*
D flip flops	22/bit	1.4	Edge-triggered FF
D latch	10/bit	2.0	Transparent latch
2-input Mux	2/bit/input	1.9	+1.2/input over 2
6T RAM cell	6/bit	1.2	1 RW port
CAM cell	13/bit	1.7	1 RW + 1 CAM ports
Static logic	2/gate input	11	Depends on speed and load

2.5.2. Leakage Power Models

While dynamic power dissipation represented the predominant factor in CMOS power dissipation for many years, leakage power has been increasingly prominent in recent technologies (Figure 2.4). Representing roughly 20-36% or more of the power dissipation in current designs, its proportion is expected to increase in the future. Leakage power can come from several sources including gate leakage and sub-threshold leakage.
Butts and Sohi [19] created a leakage power model based on BSIM3v3.2 MOSFET transistor model:

$$I_{Dsub} = I_{s0} \cdot \frac{W}{L} \cdot \left(1 - e^{\frac{-V_{DS}}{v_t}}\right) \cdot e^{\frac{V_{GS}-V_T-V_{off}}{n \cdot v_t}} \tag{2.2}$$

In this equation I_{Dsub} is the sub-threshold drain current, V_{DS} is the voltage across the drain and the source, V_{GS}, the voltage across the gate and the source terminal. V_{off} is an empirically determined model parameter and v_t is a physical parameter proportional to temperature (exponential). The term n encapsulates various device parameters. The term I_{s0} depends on transistor geometry width W and length L. They simplified this formula for a single device in off state ($V_{GS} = 0$), where $\left(1 - e^{\frac{-V_{DS}}{v_t}}\right)$ is approximately 1, since $V_{DS} = V_{DD} >> V_T$, leaving the equation in:

$$I_{Dsub} = I_{s0} \cdot \frac{W}{L} \cdot e^{\frac{-V_T - V_{off}}{n \cdot v_t}} \tag{2.3}$$

and abstracted the terms $I_{s0} \cdot e^{\frac{-V_T - V_{off}}{n \cdot v_t}}$ as I'_{leak}, or leakage current, that depends on the transistor speed.
From this formula, P_{leak} can be expressed as:

$$P_{leak} = V_{DD} \cdot N \cdot k_{design} \cdot I'_{leak} \tag{2.4}$$

Where V_{DD} represents the supply voltage, N represents the number of transistors, I_{leak} the leakage current and the k_{design} parameter represents the degree of "stacking" seen by transistors in different types of circuit designs (i.e., array structures, static logic, etc). The key insight here is that many detailed aspects of the circuit design choices can be abstracted into k_{design} factor seen in these equations. The HotLeakage simulator [100] stands on Butts/Sohi analytics to provide a simulation package for leakage power. Some examples of k_{design} parameters can be seen in table 2.2, whereas table 2.3 summarizes how the leakage power from different processor structures is modeled.

Gate Leakage

Gate leakage (also known as gate oxide leakage) grew 100-fold from the 130nm technology

Table 2.3: Leakage power models for different processor structures.

Hardware Structure	*Model Type*
Instruction Cache	Cache Array (2x bitlines)
Wakeup Logic	CAM
Issue Selection Logic	Complex combinational
Instruction Window	Array/CAM
Branch Predictor	Cache Array (2x bitlines)
Register File	Array (1x bitline)
Translation Lookaside Buffer	Array / CAM
Load/Store Queue	Array / CAM
Data Cache	Cache Array (2x bitlines)
Integer Functional Units	Complex Combinational
FP Funcional Units	Complex Combinational
Global Clock	Clock

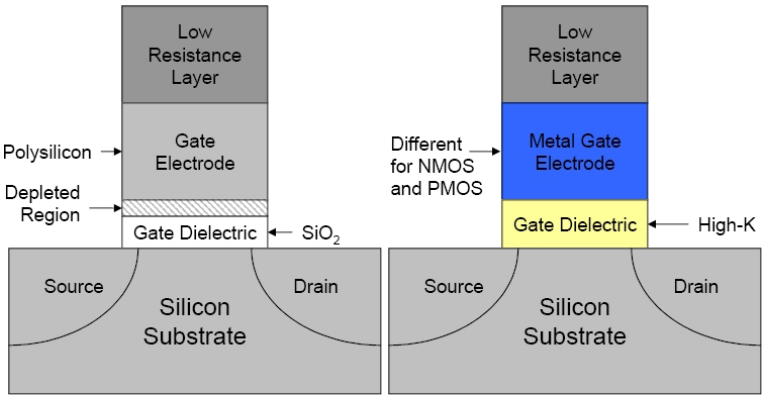


Figure 2.5: High-k vs regular transistor.

(2001) to the 90nm technology (2003) [15]. Major semiconductor companies are switching to "high-k" (Figure 2.5) dielectrics in their building process technologies to alleviate this problem [15]. The reason for gate leakage is the direct tunneling of electrons through the gate insulator (commonly silicon dioxide, SiO_2) that separates the gate terminal from the transistor channel. The thickness, T_{ox}, of the gate SiO_2 insulator must also be scaled along with other dimensions of the transistor to allow the gate's electric field to effectively control the conductance of the channel. The problem is that, when the gate insulator becomes very thin, quantum mechanics allow electrons to tunnel across it and appear at the other side. When the insulation layer is thick the probability of tunneling across is virtually non-existent. As the insulation layer becomes thinner, tunneling becomes stronger. It is amazing to realize that gate oxide thickness has scaled from 100nm to just 1.2nm in 90 nm -and 65nm- technologies. This corresponds to a thickness of just 4-5 atoms [15]. The result is an uncontrollable, exponential, increase in gate leakage.
Gate leakage is weakly dependent on temperature but strongly dependent on insulator thickness and on the gate-to-source (V_{GS}) or gate-to-drain (V_{GD}) biases seen by the device. Without the V_{GS} or V_{GD} biases the necessary electric field to cause the electrons to tunnel across the gate is absent. Since the supply voltage (V_{DD}) determines the magnitude of V_{GS} and V_{GD}, scaling V_{DD} reduces gate leakage. There is also a weaker dependence of gate leakage on V_{DS}, the voltage across the drain and source that ties gate leakage to the state of a circuit [80]. The most promising remedy for gate leakage, and the one that is currently in use is the latest generation 32nm technologies, they insulate the gate using high-k dielectric materials instead of the more common SiO_2 oxide material[3]. The increased thickness significantly reduces the tunneling effect but at the same time does not compromise the ability of the gate to control the channel. In other words, performance is not compromised. Architecturally, gate leakage has not been given the same attention as sub-threshold leakage. The trade-off, as with sub-threshold leakage, is one of speed vs. power. In sub-micron technologies, sub-threshold and gate leakage are the cost we have to pay for the increased speed afforded by scaling. Scaling the supply

[3] A thicker insulating layer of a high-k material can be as good as a thin layer of a low-k material.

voltage attempts to curb an increase in dynamic power while at the same time inflates to enormous scales sub-threshold and gate leakage. This explains why static power share increases over the dynamic power component with every new process generation.
For the most part gate leakage is considered as an additional leakage component and the hope is that process-level solutions will address the problem. The HotLeakage simulator, mentioned in section 2.4, takes gate leakage into account, thus giving a more accurate picture for the benefits of various techniques that target sub-threshold leakage.

Sub-threshold Leakage

Sub-threshold leakage power represents the power dissipated by a transistor whose gate is intended to be off. While our idealized view of transistors is that they operate as switches, the reality is that the relationship between current and voltage is analog and shows a non-zero amount of current even for supply voltages lower than the threshold voltage (V_T) at which the transistor is viewed as switched "on". This modest current for V_{DD} less than V_T is referred to as the sub-threshold current. The power dissipation resulting from this current is referred to as the sub-threshold leakage power, because the transistor appears to leak charge to ground. Sub-threshold leakage power is given by the following simplified equation.

$$P = V_{DD}(ke^{-qV_T/(ak_aT)}) \tag{2.5}$$

In this equation, V_{DD} refers to the supply voltage, while V_T refers to the threshold voltage. The exponential link between leakage power and threshold voltage is immediately obvious. Lowering the threshold voltage brings a tremendous increase in leakage power. Unfortunately, lowering the threshold voltage is what we have to do to maintain the switching speed in the face of lower supply voltages. Temperature, T, is also an important factor in the equation: leakage power depends exponentially on temperature. The remaining parameters, (q, a and k_a) summarize logic design and fabrication characteristics.

2.6. Temperature Models

For temperature models we decided to integrate HotSpot [88] (version 5.0) in our simulation environment. In this section, we will explain how this tool performs an approximation of temperature based on power and time.
Figure 2.6-left shows a typical modern single-chip package [73]. Heat generated from the active silicon device layer is conducted through the silicon bulk to the thermal interface material, heat spreader and heat sink, then convectively removed to the ambient. In addition to this primary heat transfer path, a secondary heat flow path uses conduction through the interconnect layer, I/O pads, ceramic substrate, leads/balls to the printed circuit board. HotSpot method can model all these layers and both heat flow

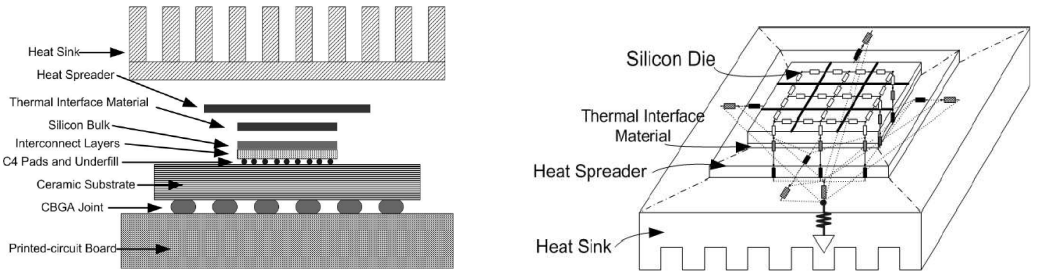


Figure 2.6: Packing components (left) and 3x3 grid thermal model (right) as described in HotSpot [88].

paths. It also considers lateral heat flow within each layer to achieve greater accuracy of temperature estimation.

In HotSpot modeling method each layer is first divided into a number of blocks. For example, the silicon bulk layer can be divided into an irregular set of blocks, according to architecture-level functional units, or into a regular set of grid cells at lower or finer granularity, depending on what the design requires. An example of regular grid primary heat transfer path with 3x3 grid cells on the silicon layer is showed in Figure 2.6-right. In order to improve accuracy, the thermal interface material and the center part of the heat spreader that is right under the interface material are also divided into the same number of grid cells as the silicon die. The remaining outside part of the heat spreader is divided into four trapezoidal blocks. The heat sink is divided into five blocks: one corresponding to the area right under the heat spreader and four trapezoids for the periphery. Each block or grid cell maps to a node in the thermal circuit. In addition to that, each cell in each layer has one vertical thermal resistance and several lateral resistances, which model vertical heat transfer to the layers above and below, and lateral heat spreading/constriction within the layer itself to the neighboring blocks, respectively. The vertical resistance values are calculated directly based on the 3 shape and material properties using:

$$R_{vertical} = \frac{t}{k \cdot A} \tag{2.6}$$

Where t is the thickness of that layer, k is the thermal conductivity of the material of that layer, and A is the cross-sectional area of the block. Calculating the lateral thermal resistances is slightly more complicated because heat spreading and constriction must be accounted for. Basically, the lateral thermal resistance on one side of a cell can be considered as the spreading/constriction thermal resistance of the neighboring part within a layer to that specific block. Details of lateral thermal resistance derivation and formulae can be found in [54] and [88].

For layers that have surfaces interfacing with the ambient (i.e. the boundaries) their model assumes that each surface (or part of a surface) has a constant heat transfer

coefficient h. The corresponding thermal resistance is then calculated as:

$$R_{convection} = \frac{1}{h \cdot A} \tag{2.7}$$

Where A is the surface area. Strictly speaking, these convection thermal resistances are not part of the compact thermal model, because they include information about the environment. If the environment changes (i.e., the boundary conditions change) these convection resistances also change. On the other hand, for a particular design, the values of all the other internal thermal resistances will not change if the compact thermal model is BCI[4].

From the above description, it is worth noting that more package layers can be easily included in the models, due to the structured assembly nature of our method. Therefore, this approach can also accommodate emerging packaging schemes such as stacked chip-scale packaging (SCP) [2] and 3D integration [7].

2.7. Benchmarks

In order to analyze performance, power and temperature of the different proposals of this Thesis we decided to use different benchmark suites, as we perform analysis in both the single-core and multi-core fields. For evaluation purposes in the single-core scenario we use the integer version of the SPEC (Standard Performance Evaluation Corporation) 2000 benchmark suite [38]. For the multi-core scenario we use the Splash-2 [94] and Parsec 2.1 [12] benchmark suites plus some other commonly used benchmarks. In this section describe the individual benchmarks and the next section performs a power/temperature analysis for those benchmarks that could be ported and compiled in our second simulation environment (GEMS - Opal+Ruby running Solaris 10).

SPECINT2000 Suite

- *256.bzip2.* Bzip2 compresses files using the Burrows-Wheeler block-sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors. Bzip2 is built on top of libbzip2, a flexible library for handling compressed data in the bzip2 format. The implemented version is based on Julian Seward's bzip2 version 0.1. The only difference between bzip2 0.1 and SPECINT2000 bzip2 is that SPEC's version of bzip2 performs no file I/O other than reading the input. All compression and decompression happens entirely in memory. This is to help isolate the work done to only the CPU and memory subsystem.

- *186.crafty.* Crafty is a high-performance computer chess program that is designed around a 64-bit word. It runs on 32 bit machines using the "long long" (or similar, as

[4]Boundary Condition Independent

_int64 in Microsoft C) data type. It is primarily an integer code, with a significant number of logical operations such as and, or, exclusive or and shift. It can be configured to run a reproducible set of searches to evaluate the integer/branch prediction/pipelining facilities of a processor.

- *252.eon.* Eon is a probabilistic ray tracer based on Kajiya's 1986 ACM SIGGRAPH conference paper. It sends a number of 3D lines (rays) into a 3D polygonal model. Intersections between the lines and the polygons are computed, and new lines are generated to compute light incident at these intersection points. The final result of the computation is an image as seen by camera. The computational demands of the program are much like a traditional deterministic ray tracer as described in basic computer graphics texts, but it has less memory coherence because many of the random rays generated in the same part of the code traverse very different parts of 3D space.

- *254.gap.* Gap is a system for computational discrete algebra, with particular emphasis on computational group theory. It provides a programming language, a library of thousands of functions implementing algebraic algorithms written in the gap language as well as large data libraries of algebraic objects. See also the overview and the description of the mathematical capabilities. Gap is used in research and teaching for studying groups and their representations, rings, vector spaces, algebras, combinatorial structures, and more.

- *176.gcc.* Based on gcc Version 2.7.2.2. This benchmark generates code for a Motorola 88100 processor. It runs as a compiler with many of its optimization flags enabled. 176.gcc has had its inlining heuristics altered slightly, so as to inline more code than would be typical on a Unix system in 1997. It is expected that this effect will be more typical of compiler usage in 2002. This was done so that 176.gcc would spend more time analyzing it's source code inputs, and use more memory. Without this effect, 176.gcc would have done less analysis, and needed more input workloads to achieve the run times required for SPECINT2000.

- *164.gzip.* gzip (GNU zip) is a popular data compression program written by Jean-Loup Gailly for the GNU project. It uses Lempel-Ziv coding (LZ77) as its compression algorithm. SPEC's version of gzip performs no file I/O other than reading the input. All compression and decompression happens entirely in memory. This is to help isolate the work done to just the CPU and the memory subsystem.

- *181.mcf.* A benchmark derived from a program used for single-depot vehicle scheduling in public mass transportation. The program is written in C, the benchmark version uses almost exclusively integer arithmetic.

  The program is designed for the solution of single-depot vehicle scheduling subproblems occurring in the planning process of public transportation companies. It considers one single depot and a homogeneous vehicle fleet. Based on a line plan

and service frequencies, so-called timetabled trips with fixed departure/arrival locations and times are derived. Each of this timetabled trip has to be serviced by exactly one vehicle. The links between these trips are so-called dead-head trips. In addition, there are pull-out and pull-in trips for leaving and entering the depot.

- *197.parser.* The link grammar parser is a syntactic parser of English, based on link grammar, an original theory of English syntax. Given a sentence, the system assigns to it a syntactic structure, which consists of set of labeled links connecting pairs of words. The parser has a dictionary of about 60000 word forms. It has coverage of a wide variety of syntactic constructions, including many rare and idiomatic ones. The parser is robust; it is able to skip over portions of the sentence that it cannot understand, and assign some structure to the rest of the sentence. It is able to handle unknown vocabulary, and make intelligent guesses from context about the syntactic categories of unknown words.

- *300.twolf.* The TimberWolfSC placement and global routing package is used in the process of creating the lithography artwork needed for the production of microchips. Specifically, it determines the placement and global connections for groups of transistors (known as standard cells) which constitute the microchip. The placement problem is a permutation. Therefore, a simple or brute force exploration of the state space would take an execution time proportional to the factorial of the input size. For problems as small as 70 cells, a brute force algorithm would take longer than the age of the universe on the world's fastest computer. Instead, the TimberWolfSC program uses simulated annealing as a heuristic to find very good solutions for the row-based standard cell design style. In this design style, transistors are grouped together to form standard cells. These standard cells are placed in rows so that all cells of a row may share power and ground connections by abutment. The simulated annealing algorithm has found the best known solutions to a large group of placement problems. The global router which follows the placement step interconnects the microchip design. It utilizes a constructive algorithm followed by iterative improvement.

- *255.vortex.* Vortex is a single-user object-oriented database transaction benchmark which exercises a system kernel coded in integer C. The Vortex benchmark is a derivative of a full OODBMS that has been customized to conform to SPEC-CINT2000 guidelines. The benchmark 255.vortex is a subset of a full object oriented database program called Vortex. (Vortex stands for "Virtual Object Runtime EXpository.")

  Transactions to and from the database are translated though a schema. (A schema provides the necessary information to generate the mapping of the internally stored data block to a model viewable in the context of the application.)

- *175.vpr.* Vpr is a placement and routing program; it automatically implements a

technology-mapped circuit (i.e. a netlist, or hypergraph, composed of FPGA logic blocks and I/O pads and their required connections) in a Field-Programmable Gate Array (FPGA) chip. Vpr is an example of an integrated circuit computer-aided design program, and algorithmically it belongs to the combinatorial optimization class of programs.

Placement consists of determining which logic block and which I/O pad within the FPGA should implement each of the functions required by the circuit. The goal is to place pieces of logic which are connected (i.e. must communicate) close together in order to minimize the amount of wiring required and to maximize the circuit speed. This is basically a slot assignment problem – assign every logic block function required by the circuit and every I/O function required by the circuit to a logic block or I/O pad in the FPGA, such that speed and wire-minimization goals are met. Vpr uses simulated annealing to place the circuit. An initial random placement is repeatedly modified through local perturbations in order to increase the quality of the placement, in a method similar to the way metals are slowly cooled to produce strong objects.

Vpr uses a variation of Dijkstra's algorithm in its innermost routing loop in order to connect the terminals of a net (signal) together. Congestion detection and avoidance features run "on top" of this innermost algorithm to resolve contention between different circuit signals over the limited interconnect resources in the FPGA.

Splash-2 suite

- *Barnes.* The barnes application simulates the interaction of a system of bodies (galaxies or particles, for example) in three dimensions over a number of time steps, using the Barnes-Hut hierarchical N-body method. Each body is modeled as a point mass and exerts forces on all other bodies in the system. To speed up the interbody force calculations, groups of bodies that are sufficiently far away are abstracted as point masses. In order to facilitate this clustering, physical space is divided recursively, forming an octree. The tree representation of space has to be traversed once for each body and rebuilt after each time step to account for the movement of bodies.

  The main data structure in barnes is the tree itself, which is implemented as an array of bodies and an array of space cells that are linked together. Bodies are assigned to processors at the beginning of each time step in a partitioning phase. Each processor calculates the forces exerted on its own subset of bodies. The bodies are then moved under the influence of those forces. Finally, the tree is regenerated for the next time step. There are several barriers for separating different phases of the computation and successive time steps. Some phases require exclusive access to tree cells and a set of distributed locks is used for this purpose. The communication patterns are dependent on the particle distribution and are quite irregular. No

attempt is made at intelligent distribution of body data in main memory, since this is difficult at page granularity and not very important to performance.

- *Cholesky.* The blocked sparse cholesky factorization kernel factors a sparse matrix into the product of a lower triangular matrix and its transpose. It is similar in structure and partitioning to the LU factorization kernel, but has two major differences: (i) it operates on sparse matrices, which have a larger communication to computation ratio for comparable problem sizes, and (ii) it is not globally synchronized between steps.

- *FFT.* The fft kernel is a complex one-dimensional version of the radix $\sqrt{x}$ six-step fft algorithm, which is optimized to minimize interprocessor communication. The dataset consists of the n complex data points to be transformed, and another n complex data points referred to as the roots of unity. Both sets of data are organized as $\sqrt{x}$ x $\sqrt{x}$ matrices partitioned so that every processor is assigned a contiguous set of rows which are allocated in its local memory.

- *Ocean.* The ocean application studies large-scale ocean movements based on eddy and boundary currents. The algorithm simulates a cuboidal basin using discretized circulation model that takes into account wind stress from atmospheric effects and the friction with ocean floor and walls. The algorithm performs the simulation for many time steps until the eddies and mean ocean flow attain a mutual balance. The work performed every time step essentially involves setting up and solving a set of spatial partial differential equations. For this purpose, the algorithm discretizes the continuous functions by second-order finite-differencing. After that it sets up the resulting difference equations on two-dimensional fixed-size grids representing horizontal cross-sections of the ocean basin. Finally it solves these equations using a red-back Gauss-Seidel multigrid equation solver. Each task performs the computational steps on the section of the grids that it owns, regularly communicating with other processes.

- *Radix.* The radix program sorts a series of integers, called keys, using the popular radix sorting method. The algorithm is iterative, performing one iteration for each radix digit of the keys. In each iteration, a processor passes over its assigned keys and generates a local histogram. The local histograms are then accumulated into a global histogram. Finally, each processor uses the global histogram to permute its keys into a new array for the next iteration. This permutation step requires all-to-all communication. The permutation is inherently a sender determined one, so keys are communicated through writes rather than reads.

- *Raytrace.* This application renders a three-dimensional scene using ray tracing. A hierarchical uniform grid is used to represent the scene, and early ray termination is implemented. A ray is traced through each pixel in the image plane and it produces other rays as it strikes the objects of the scene, resulting in a tree of rays

per pixel. The image is partitioned among processors in contiguous blocks of pixel groups, and distributed task queues are used with task stealing. The data accesses are highly unpredictable in this application. Synchronization in raytrace is done by using locks. This benchmark is characterized for having very short critical sections and very high contention.

- *Water-nsq.* The water-nsq application performs an N-body molecular dynamics simulation of the forces and potentials in a system of water molecules. It is used to predict some of the physical properties of water in the liquid state. Molecules are statically split among the processors and the main data structure in water-nsq is a large array of records that is used to store the state of each molecule. At each time step, the processors calculate the interaction of the atoms within each molecule and the interaction of the molecules with one another. For each molecule, the owning processor calculates the interactions with only half of the molecules ahead of it in the array. Since the forces between the molecules are symmetric, each pair-wise interaction between molecules is thus considered only once. The state associated with the molecules is then updated. Although some portions of the molecule state are modified at each interaction, others are only changed between time steps.

- *Water-sp.* This application solves the same problem as water-nsq, but uses a more efficient algorithm. It imposes a uniform 3-D grid of cells on the problem domain, and uses an O(n) algorithm which is more efficient than water-nsq for large numbers of molecules. The advantage of the grid of cells is that processors which own a cell need only to look at neighboring cells to find molecules that might be within the cutoff radius of molecules in the box it owns. The movement of molecules into and out of the cells causes cell lists to be updated, resulting in communication.

Parsec 2.1 suite

- *Blackscholes.* This application is an Intel RMS benchmark. It calculates the prices for a portfolio of European options analytically with the Black-Scholes partial differential equation (PDE). There is no closed-form expression for the Black-Scholes equation and as such it must be computed numerically.

- *Fluidanimate.* This Intel RMS application uses an extension of the Smoothed Particle Hydrodynamics (SPH) method to simulate an incompressible fluid for interactive animation purposes. It was included in the PARSEC benchmark suite because of the increasing significance of physics simulations for animations.

- *Swaptions.* The application is an Intel RMS workload which uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of swaptions. Swaptions employs Monte Carlo (MC) simulation to compute the prices.

- *x264.* This application is an H.264/AVC (Advanced Video Coding) video encoder. H.264 describes the lossy compression of a video stream and is also part of ISO/IEC

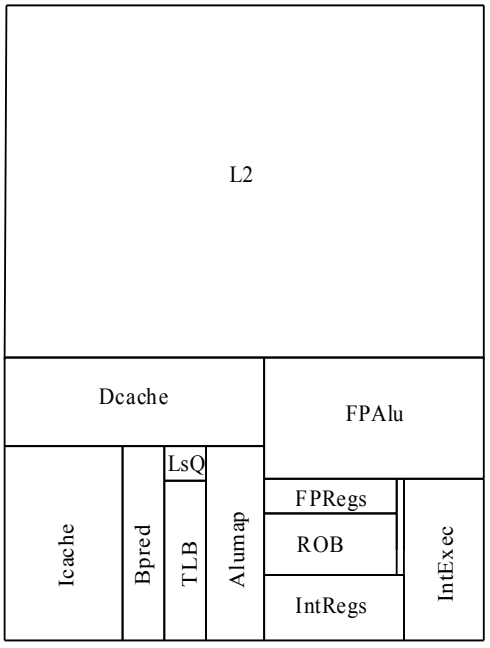
MPEG-4. The flexibility and wide range of application of the H.264 standard and its ubiquity in next-generation video systems are the reasons for the inclusion of x264 in the PARSEC benchmark suite.

Other benchmarks

- *Unstructured.* Unstructured is a computational fluid dynamics application that uses an unstructured mesh to model a physical structure, such as an airplane wing or body. The mesh is represented by nodes, edges that connect two nodes, and faces that connect three or four nodes. The mesh is static, so its connectivity does not change. The mesh is partitioned spatially among different processors using a recursive coordinate bisection partitioner. The computation contains a series of loops that iterate over nodes, edges and faces. Most communication occurs along the edges and faces of the mesh.
- *Tomcatv.* Parallel version of the SPECCFP95 - 101.tomcatv benchmark. Tomcatv is a vectorized mesh generation program part of Prof. W. Gentzsch's benchmark suite.

2.8. Benchmark Thermal Profiles and Per-Structure Power Distribution

In this section we analyze the thermal profiles for all the benchmark suites used in this Thesis running on a single core. In addition we will also show power usage of the different structures normalized to the total power usage of each benchmark, as well as a comparison with the processor peak power dissipation as measured by McPAT.
We simulate a 14-stage, out-of-order core. Figure 2.7 shows the core configuration and floorplan used to obtain temperature and power numbers in this section. For a N-core CMP the core floorplan will be replicated N times. Power and area numbers for this core configuration were obtained through the McPAT framework. Figure 2.8 depicts the power distribution of the different core structures provided by McPAT and normalized to the total (peak) power of the core ($7.6W$). We then input these power numbers into GEMS-Opal to obtain preliminary average power numbers of each structure using a modified version of the Wattch implementation included in GEMS-Opal (minor bugfixes). Using these average power numbers we build the input files for HotSpot - Hotfloorplanner that will generate the core floorplan. In addition to the power numbers, HotSpot also needs per-structure area information (provided by McPAT) and structure communication needs (we used Alpha 21264 structure dependences as our input for Hotfloorplanner). To estimate temperatures we integrated HotSpot, that uses as inputs the generated floorplan and the dynamic and leakage power provided by McPAT, into GEMS-Opal. We also implemented a temperature/leakage power loop inside the GEMS-Opal simulator, as there is a direct relationship between both terms.



(a) Core Floorplan

Processor Core	
Process Technology:	32 nanometers
Frequency:	3000 Mhz
V_{DD}:	0.9 V
Instruction Window	128 RUU + 64 IW
Load Store Queue	64 Entries
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	6 Int Alu; 2 Int Mult
	4 FP Alu; 4 FP Mult
Branch Predictor:	16bit Gshare
Memory Hierarchy	
Coherence Prot.:	MOESI
Memory Latency:	300
L1 I-cache:	64KB, 2-way, 1 cycle lat.
L1 D-cache:	64KB, 2-way, 1 cycle lat.
L2 cache:	1MB/core, 4-way, unified
	12 cycle latency
TLB:	256 entries
Network Parameters	
Topology:	2D mesh
Link Latency:	4 cycles
Flit size:	4 bytes
Link Bandwidth:	1 flit / cycle

(b) Core Structures

Figure 2.7: Core Configuration

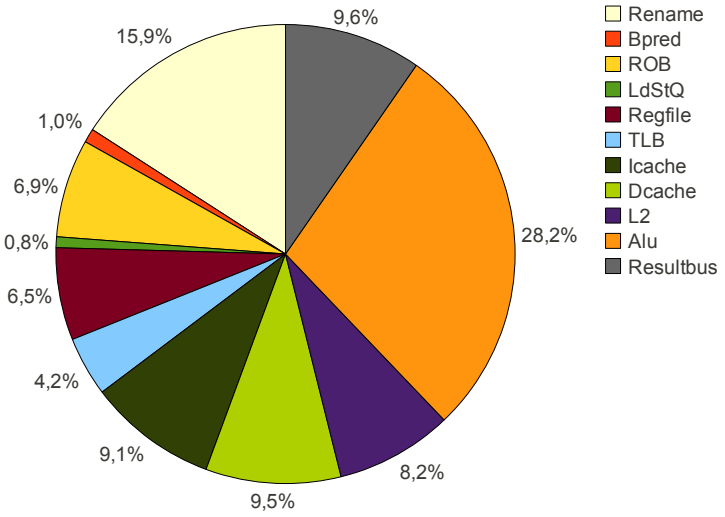


Figure 2.8: Power breakdown (%) of the simulated core respect to a total dynamic power of 7.6W.

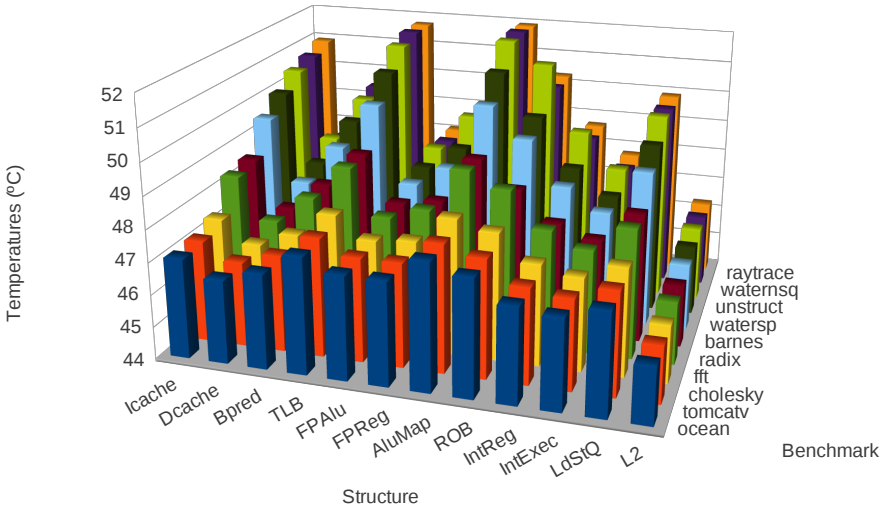


Figure 2.9: Thermal profiles for the SPLASH-2 benchmark suite.

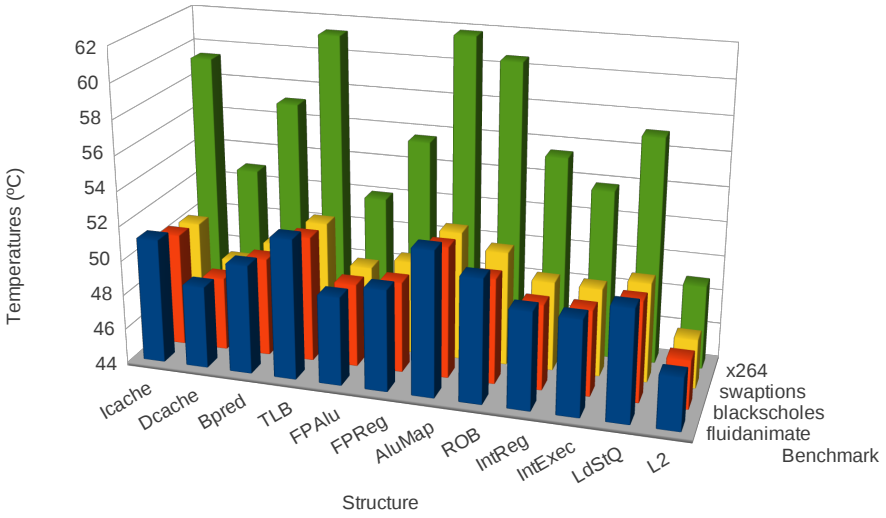


Figure 2.10: Thermal profiles for the PARSEC 2.1 benchmark suite.

Figure 2.9 shows per-structure temperatures for the SPLASH 2 suite whereas Figure 2.10 depicts the same information for the benchmarks from the PARSEC 2.1 suite and, finally, Figure 2.11 shows the analogous information for the SPECINT2000 suite. We can clearly identify the major core hotspots: TLB (Translation Lookaside Buffer), Icache (Instruction Cache), Alumap (Rename logic), ALUs (Arithmetic Logic Unit), ROB (Reorder buffer) and LdStQ (Load Store Queue). Power utilization (and thus temperature) has a high variability between benchmarks. For example, we can see temperature variations of 12°C between Ocean (47°C) and x264 (59°C). These temperature variations are due to the fact that some of the benchmarks do not use all of the processors resources, even though they have good scalability as we increase the number of cores of the CMP (as we will see in Section 2.9).

If we want to gain some insight of how each benchmark uses the core resources we need

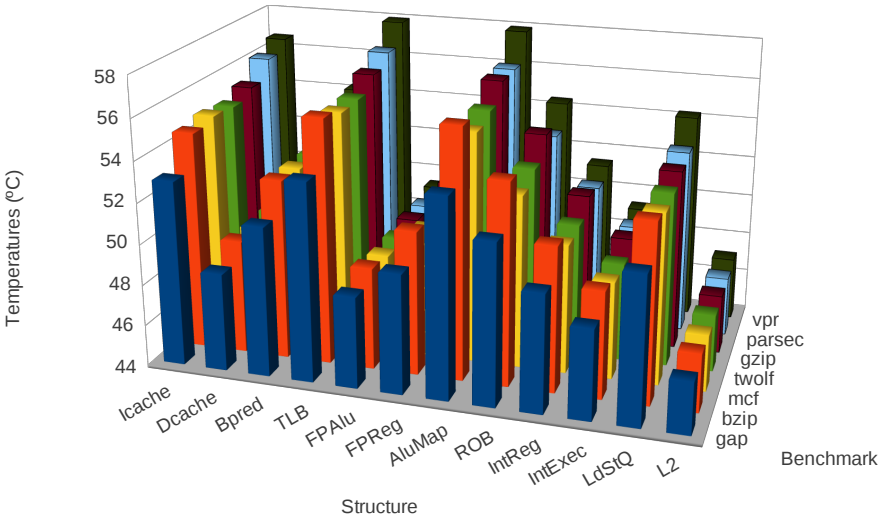


Figure 2.11: Thermal profiles for the SPECINT2000 benchmark suite.

to take a look to the power numbers. Tables 2.4, 2.5 and 2.6 show per-structure power usage for the three benchmark suites, respectively. Structure power is normalized to the total power dissipated by each benchmark. We also include the total per-cycle power usage of the benchmark and the fraction of power this represents from the peak dynamic power provided by McPAT. According to these numbers, most of the power dissipated by the core is done by the Icache, ROB, Rename Logic, ALUs and TLB, that match the temperature hotspots of the chip seen in Figures 2.9 to 2.11. There is also another hotspot, the TLB, that does not consume a big part of the total power. The TLB has a really small area (as it can be seen in Figure 6.5b), so it is difficult for this structure to transfer heat to the heatsink.

Table 2.4: Normalized power distribution for SPLASH-2

Structure	*barnes*	*cholesky*	*fft*	*ocean*	*radix*	*raytrace*	*tomcatv*	*unstruc*	*waternsq*	*watersp*
Rename	18.9%	16.1%	17.6%	16.1%	17.8%	19.4%	15.6%	18.8%	20.3%	19.7%
Bpred	0.4%	0.7%	0.5%	0.8%	0.6%	0.7%	0.7%	0.3%	0.7%	0.8%
ROB	12.3%	11.7%	12.9%	10.4%	10.1%	11.4%	10.9%	15%	11.7%	12%
LdStQ	0.4%	0.7%	0.5%	0.8%	0.6%	0.7%	0.7%	0.7%	0.3%	0.4%
Regfile	6.1%	5.1%	5.8%	5.6%	5.3%	6.1%	5.4%	6.3%	6.2%	6%
TLB	11.3%	8.8%	10%	8%	10.7%	12.2%	8.5%	11.2%	12.9%	12%
Icache	23.7%	16.9%	20.5%	15.3%	22.6%	25.9%	15.6%	22.5%	25.1%	24%
Dcache	5.6%	8%	5.8%	8%	5.9%	6.1%	8.5%	7.1%	5.1%	6%
L2	2.8%	4.4%	3.5%	4.8%	3.5%	2.2%	4.6%	2.2%	2.3%	2.5%
Alu	17.5%	26.4%	21.7%	29%	22%	14.5%	28.1%	15%	14.5%	15.8%
Resultbus	0.4%	0.7%	0.5%	0.8%	0.6%	0.3%	0.7%	0.3%	0.3%	0.4%
Total (Watts)	2.1	1.35	1.69	1.23	1.67	2.61	1.27	2.65	2.54	2.32
% of peak	27.7%	17.8%	22.3%	16.3%	22.1%	34.4%	16.8%	34.9%	33.5%	30.6%

Table 2.5: Normalized power distribution for PARSEC

Structure	*blackscholes*	*fluidanimate*	*swaptions*	*x264*
Rename	19.9%	19%	18.8%	19.8%
Bpred	0.4%	0.4%	0.4%	1%
ROB	9.7%	11.7%	12.3%	17%
LdStQ	0.4%	0.8%	0.4%	0.5%
Regfile	6%	6%	5.9%	6.3%
TLB	11.5%	12.1%	11.4%	12.7%
Icache	25.4%	24.7%	23.3%	27.1%
Dcache	5.5%	6.4%	6.4%	6%
L2	2.7%	2.4%	2.7%	1%
Alu	17.5%	15.7%	17.4%	7.8%
Resultbus	0.4%	0.4%	0.4%	0.3%
Total (Watts)	2.1	2.4	2.1	5.6
% of peak	28.4%	32.4%	28.6%	74.1%

Table 2.6: Normalized power distribution for SPECINT2000

Structure	*bzip*	*gap*	*gzip*	*mcf*	*parsec*	*twolf*	*vpr*
Rename	20.2%	19%	20.4%	19.1%	19.6%	19.7%	20.3%
Bpred	0.7%	0.6%	0.7%	0.7%	0.7%	0.7%	0.6%
ROB	13.4%	12.3%	13.3%	11.1%	11.2%	12.1%	12%
LdStQ	0.4%	0.6%	0.4%	0.5%	0.4%	0.5%	0.6%
Regfile	6.3%	6%	6.1%	5.6%	5.8%	6%	6.1%
TLB	13.2%	12.9%	13.3%	13.9%	14.4%	13.4%	13.9%
Icache	28.1%	29%	29%	31.2%	32%	29.8%	29.6%
Dcache	5.1%	5.4%	4.5%	5.4%	4.4%	5.5%	6.5%
L2	1.4%	1.8%	1.4%	1.5%	1.4%	1.5%	1.2%
Alu	10.5%	11.7%	10.2%	10.3%	9.5%	10.1%	8.4%
Resultbus	0.2%	0.3%	0.2%	0.2%	0.2%	0.2%	0.2%
Total (Watts)	4	3.3	4.1	3.8	4.2	3.9	4.7
% of peak	53.8%	43.5%	55.2%	50.9%	56.3%	51.9%	62%

2.9. Performance vs Power-Efficiency in a Multi-Core Scenario

In this section we show a brief study of the scalability of the different applications from SPLASH-2 and PARSEC 2.1 in terms of both performance and energy consumption. Figure 2.12-left shows the performance speedup for different applications as we increase the number of cores from 2 to 16. We can clearly see that FFT benchmark does not properly scale after 2 cores and saturates at 8 cores. Something similar happens with x264 at 4 cores. The rest of the studied benchmarks have a sustained speedup progres-

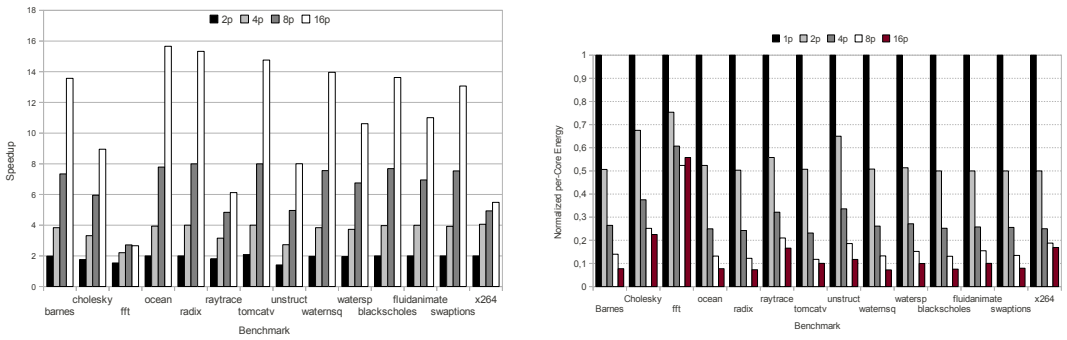


Figure 2.12: Performance speedup (left) and energy (right) for the studied parallel benchmarks.

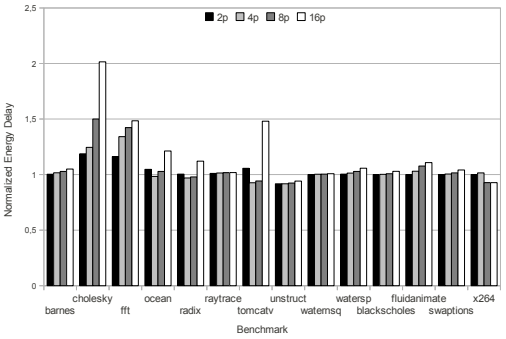


Figure 2.13: Energy delay product for the studied parallel benchmarks.

sion as we increase the number of cores in the CMP. However, if we multiply the speedup obtained by the energy consumed[5] to run the benchmark (2.13) things change a little bit. For x264 benchmark, even if the benchmark does not scale properly in terms of performance, the energy consumed by the cores is reduced. This means that individual cores have less work to do, and regular clock gating is enough to prevent useless power utilization. On the other hand, cholesky benchmark, although it shows a good speedup progression, individual cores are wasting more power resources than the speedup they provide, probably because they are executing useless instructions (wrong path instructions). The same thing happens to tomcatv benchmark, that has an increase of power dissipation in useless instructions of almost 20% for 16 cores. For the rest of benchmarks there is almost no power overhead when scaling the number of cores.

[5]This is known as Energy Delay product.

Chapter 3

Power Saving Mechanisms

Summary:

Nowadays, when we think about the design of embedded microprocessors we always think in energy consumption and power dissipation, especially in the case of battery-operated devices. Now the trend has moved to the high performance domain where operating costs and thermal constraints are becoming a serious problem. There are two sources of power dissipation: dynamic and static power. For several generations, static power (or leakage) has been just a small fraction of the overall energy consumption in microprocessors, and it was not considered a major concern [48, 50]. As cited in the previous chapter, the continued CMOS scaling allows a lower supply voltage which has the positive effect of reducing the dynamic power component. However, using smaller geometries leads to use lower threshold voltages which has the additional effect of increasing leakage power exponentially. This results in static power beginning to gain importance in the overall power dissipation as process technology drops below 65 nm [16, 34, 50].

In this chapter we introduce some of the most common power control mechanisms that include: Dynamic Voltage and Frequency Scaling, Branch Confidence Estimation, Pipeline Throttling enabled through Decode Commit Ratio (DCR), Critical Path Prediction for dynamic power reduction and, finally, Decay and Drowsy mechanisms for static power reduction with examples of usage in caches and value predictors.

3.1. Dynamic Power Control Mechanisms

Dynamic power has been the main source of power dissipation in microprocessors until very recently. This power dissipation comes from the equation $P_d \approx V_{DD}^2 \cdot f$. A straightforward approach to control dynamic power is to lower the supply voltage V_{DD}. This has been done on every downscaling of the transistor technology, going down from around 1.8V at 180nm to 1.2V at 32nm.

However, another dependence exists between a transistor's maximum switching speed and the supply voltage: $\delta \approx 1/(V_{DD} - V_T)^\alpha$, with $\alpha > 1$. This means that, to keep a constant

switching speed while lowering the supply voltage the threshold voltage (V_T) must be lowered accordingly. However, lowering V_T affects reliability and increases leakage power dissipation.

In this section we study different power saving mechanisms to control dynamic power. These mechanisms include both circuit-level mechanisms (like DVFS) and some microarchitectural techniques. Note that, for circuit-level mechanisms the performance impact is much lower than for microarchitectural mechanisms, because part of the power reduction comes from voltage variations while for microarchitectural mechanisms every power reduction has a direct impact on performance (unless they only affect instructions from the wrong path of execution that have no impact on performance). Techniques will be applied to a single-core processor in this analysis (the processor configuration can be seen in Table 3.1). Some of these power saving mechanisms are usually triggered by a DTM mechanism instead of being active all the time. For our experimental analysis in next sections we assume a power-based triggering mechanism that enables power saving mechanisms when the processor exceeds an specific target power budget. We will use a restrictive power budget of 50% of the peak dynamic power of the processor to force a high usage of the different power saving mechanisms.

Table 3.1: Core configuration.

Processor Core	
Process Technology:	65 nanometers
Frequency:	3000 Mhz
Instruction Window	128 RUU, 64 LSQ
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	8 Int Alu; 2 Int Mult
	8 FP Alu; 2 FP Mult
Memory Hierarchy	
L1 I-cache:	64KB, 2-way
L1 D-cache:	64KB, 2-way
L2 cache:	2MB, 4-way, unified
Memory:	300 cycle latency
	2 memports
TLB:	256 entries
Other Information	
Branch Predictor:	16bit Gshare
Branch and Value Pred.:	2 ports

3.1.1. Dynamic Voltage and Frequency Scaling

This technique has been widely used since the early 90's [62] offering a great promise to reduce energy consumption in microprocessors. DVFS relies on the fact that dynamic power dissipation depends on both voltage and frequency $P_D \approx V_{DD}^2 \cdot f$, and it dynamically scales these terms to save dynamic power [85, 87, 96]. Kaxiras *et al.* [47] describe the key design issues for DVFS.

1. Operation level: There are three different levels where decisions can be made, exploiting slack.
   - System-level: Detect idle periods (low system load) and scale voltage/frequency for the whole processor.
   - Program/phase-level: Decisions are taken according to the program/phase behavior, exploiting instruction slack at this level.
   - Hardware-level: This level tries to exploit the slack hidden in hardware operation (similar to idle time at system level).
2. DVFS settings: Some designs allow software to adjust a register that encodes the desired voltage and frequency settings while in other cases choices are made dynamically by hardware mechanisms.
3. Hardware granularity to control DVFS: Most work on DVFS focuses on cases where the entire processor operates at the same voltage and frequency but is asynchronous with the outside world (memory). In cases where the processor is stalled due to memory dependencies DVFS can be applied without significant impact on performance.
4. Implementation characteristics of DVFS: Delay on switching between (voltage/frequency) modes. Can the processor continue working when switching?
5. DVFS on multiple-core processors: In a multicore scenario executing parallel applications, reducing clock frequency to one core may impact other dependent threads that are waiting for a result to be produced by the delayed thread.

Finally, one of the major drawbacks of DVFS has been the slow off-chip voltage regulators that lack the ability to adjust to different voltages at small time scales (0.016mV/ns according to [95]).
In the recent years, researchers and designers have moved to CMP architectures as a way of maintaining performance scaling while staying within tight power constraints [32, 40]. This trend, coupled with diverse workloads found in modern systems, motivates the need for fast per-core DVFS control. Kim *et al.* [51] recently proposed the use of on-chip regulators to achieve fast transition speeds of 30-50mV/ns. This solves one of the major problems of DVFS but still has some limitations. As the building process goes into deep submicron, the margin between V_{DD} (supply voltage) and V_T (threshold voltage) is reduced, and as this margin decreases, the processor's reliability is reduced (among other undesirable effects). Moreover, as cited before, the transistor's delay (switching speed) depends on: $\delta \approx 1/(V_{DD} - V_T)^{\alpha}$, with $\alpha > 1$. This means that we can lower V_{DD} for DVFS as long as we keep the margin between V_{DD} and V_T (i.e., V_T must be lowered accordingly) so we can obtain the desired speed increase derived from technology scaling. However, the counterpart of reducing V_T is twofold: a) leakage power increases as it exponentially depends on V_T (equation 2.5), which makes leakage an important

source of power dissipation as the process technology scales below 65nm [34, 48, 50]; and b) processor reliability is further reduced.

DVFS - MCD (multiple clock domains)

DVFS-MCD is an improved version of DVFS that divides the processor into different clock domains, typically four, and applies different pairs of voltage-frequencies according to the program needs. The most common domains are:

- Front end + L1 Icache
- Integer Units
- FP Units
- Load-Store Units + L1 Dcache + L2 cache

The processor's front-end frequency is set statically because it is too complex to adjust dynamically [85]. All the other domains modify their voltage and frequency at runtime according to the program needs. Buffers are placed between domains for synchronization purposes. The analysis of the program needs can be done either offline using profiling [64, 85] or online. By taking into account information about the buffers occupancy we can determine at runtime each domain needs and modify frequencies accordingly [43, 95]. If the buffer is empty (the consumer is too fast or the producer is too slow) we can modify their frequencies depending on what we pursue: performance or lower energy consumption. We can do the opposite if the buffer fills up.

DVFS - CMP (Chip multiprocessors)

If we apply DVFS in a CMP at a processor level we are reducing the frequency and voltage for all the cores in the CMP, and that leads to a high performance degradation. It is more useful to have independent voltage regulators for each core and reduce the frequency of the cores that do less useful work [32].
Juang *et al.* [43] proposed the use of distributed DVFS that dynamically identifies execution threads in the critical path (threads that take more time to reach a synchronization point). Once these "critical" threads are identified they move them to a core that works at full speed while the rest of the cores run at a lower speed executing non-critical threads. This way all the threads in the processor reach the synchronization point at the same time. They assume full parallel programs with synchronization points that behave uniformly during program execution and critical threads that will always be critical.
Isci *et al.* [40] introduce a mechanism that uses DVFS in a CMP to match a predefined power budget, assuming that we have mechanisms to measure power and performance in real-time. Their MaxBIPs approach calculates the best combination of core speeds to maximize throughput at runtime depending on the different program needs and behavior. However, this mechanism does not work properly for memory-intensive benchmarks.

3.1.2. Pipeline Throttling

Pipeline Throttling is a technique that reduces the amount of in-flight instructions in the pipeline to reduce power dissipation [4, 8, 66]. Pipeline Throttling can be applied at different stages, producing different effects on both power and performance.

- Instruction flow control: These techniques try to estimate the amount of ILP in the processor by tracking the instructions traversing the pipeline. Authors in [8] propose the decode/commit ratio (DCR) heuristic for estimating the processor's current ILP. We can take advantage of this information and either stop or slow down the front-end for a small number of cycles to reduce power dissipation.
- Confidence estimation: Confidence estimators try to add some additional information to branch predictors so that the processor can check how good a prediction is and act accordingly. JRS [41] is one of the most cost-effective confidence estimators. It uses a direct-mapped table accessed by the program counter where it stores how many consecutive hits there are for a branch. When the counter exceeds a threshold, the branch is considered confident.

For this thesis we are interested in the determining the capabilities of the different techniques to adapt to power constraints (that can be measured either by the number of cycles over the power budget or by the area over the power budget). In addition we want to know how much power we can save and what performance penalty these techniques have on the processor.
Pipeline throttling techniques rely on reducing the instruction flow inside the pipeline to reduce power dissipation. In this section we experimentally analyze how the Pipeline Throttling technique behaves individually in terms of both the number of cycles over the power budget and performance degradation for the JRS and DCR approaches. For the DCR approach (Figure 3.1) we show the throttling effect when: a) the technique is always enabled (DCR); b) throttling is done only when both the power budget is exceeded and the DCR criteria is matched (DCR + >PB); and c) throttling is done only when the power budget is exceeded (>PB). While pipeline throttling is active, we divide both the fetch and decode bandwidth by X (note that we are evaluating a 4-wide issue processor, so X can vary from 0 to 4). As mentioned before, in order to have a better understanding of each technique capabilities (accuracy, total dynamic power reduction and performance degradation) we will use a restrictive power budget of 50% of the peak dynamic power of the processor.
Figures 3.1-left and 3.2-left show the amount of throttling in the x axis as follows: the base case corresponds to bar 1; bars 2 and 4 correspond to a throttling of 1/2 and 1/4 of its original capacity, respectively; finally, bar 0 means a "full-stop" of the fetch unit. All the techniques are applied as long as the trigger condition lasts (e.g., JRS labels a branch as non-confident; or the DCR condition is met[1] and the power budget is exceeded)

[1] There are three times more committed than decoded instructions.

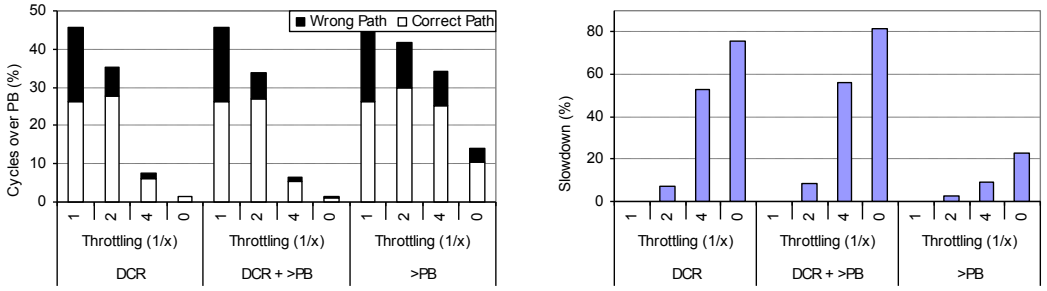


Figure 3.1: Cycles over PB and slowdown for DCR-based throttling (power budget = 50%).

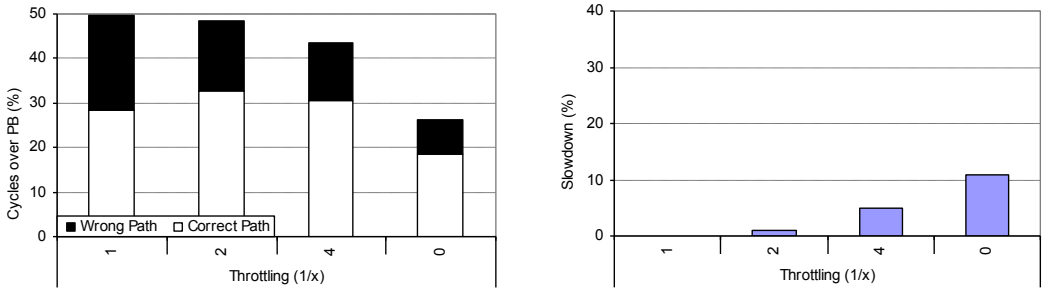


Figure 3.2: Cycles over PB and slowdown for JRS-based throttling (power budget = 50%).

plus 3 cycles. The evaluated confidence estimator is a modified version of JRS (Figure 3.2) with a 64K-entry table where each entry contains a 2-bit saturating counter. The confidence estimator, thus, has a size of 16KB with a power overhead of around 0.3%, also accounted in our results. When a branch is labeled as non-confident, we either stop or slow down the front-end of the processor.

One of the most interesting things we can observe in Figures 3.1 and 3.2 is that the major reduction in the cycles over power budget comes from cycles that belong to a mispredicted path, as intended (microarchitectural techniques can only save power from instructions from the wrong path of execution, not from "real" instructions). However, none of these techniques alone is good enough to match the required power budget of 50% unless performance is highly degraded (e.g., bars 4 and 0 in Figure 3.1). Therefore, under high power restrictions, the performance degradation of Pipeline Throttling mechanisms makes them worsen than DVFS. However, under less restrictive power constraints, Pipeline Throttling mechanisms are able to match the target power budget with minimal performance degradation, saving power from instructions than belong to a mispredicted path of execution.

3.1.3. Critical Path

Data dependencies are one of the main bottlenecks in high performance processors: dependency chains limit the performance of the machine leaving most of the processor structures and logic idle [92]. These chains of dependent instructions are known as the

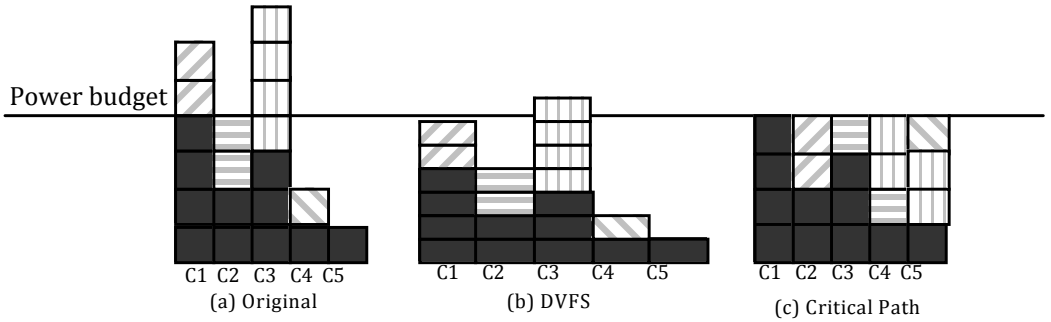


Figure 3.3: DVFS and critical path effects on power dissipation at a cycle level.

critical path of the code. A processor's performance is determined by how fast it can execute the critical path, not by how fast it can execute all of the code.

If we were able to distinguish between instructions that belong to the critical path of the program, we could either accelerate their execution to improve the machine's performance, or slow down non-critical instructions to reduce power and temperature. Authors in [92] proposed a critical path predictor that is able to predict critical instructions. However, the amount of cycles a non-critical instruction can be delayed without becoming critical is really small [22].

When working in a power-constrained scenario where a power budget is defined, a critical path predictor can be used to locate non-critical instructions in cycles over the power budget in order to move them to cycles under the power budget [26, 27]. As mentioned previously, the more cycles an instruction is delayed, the more chances that it becomes part of the critical path, so instructions are only delayed up to 8 cycles from their original execution cycle. As in [92], we use an 8K-entry table indexed by PC. Each entry has a 6-bit saturating counter that is incremented by 8 if the instruction belongs to the critical path and is decremented by 1 otherwise. This 6KB table (340 times smaller than the L2 cache) introduces a power overhead of around 0.5% which is also accounted in our results. From the proposed policies in [92], we have chosen the "QOld" policy for our implementation because of its simplicity and cost-effectiveness. Each cycle, QOld policy marks as critical the oldest instruction in the instruction queue, unless it is ready. When the instruction becomes ready it is marked as non-critical.

In order to gain some insight of the differences between circuit-level and microarchitectural techniques, Figure 3.3 illustrates how instructions behave inside the pipeline for the original instruction flow, DVFS and critical path for several cycles (Ci represents cycle i). Solid boxes correspond to critical instructions while the rest represent non-critical instructions. If we set a power budget like the one in Figure 3.3, DVFS will slow down all the instructions (actually, DVFS increases cycle length) to match the required budget at the expense of increasing the execution time. On the other hand, by using instruction criticality information, as shown in Figure 3.3-(c), we locate non-critical instructions, so they can be efficiently delayed to cycles under the budget.

For a simple implementation we only add a single bit in the critical path predictor table,

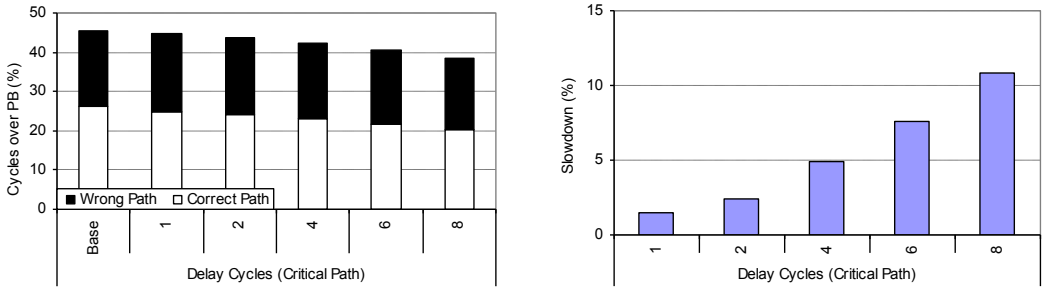


Figure 3.4: Instruction criticality analysis approach for a power budget of 50%.

along with the information about the criticality of an instruction, to mark if the last time it was executed the power budget was exceeded. Therefore, if the predictor detects a non-critical instruction previously executed in a cycle over the budget, then our proposed mechanism will try to delay its execution to a cycle under the power budget. The longer it is delayed, the more chances for the instruction to become critical. Once a delayed instruction becomes critical we will experience a significant performance degradation.
Again, we will use a restrictive power budget of 50% to see how much we can benefit from instruction reordering. Results for the instruction criticality analysis can be seen in Figure 3.4. The "delay cycles" (x axis) represent the maximum number of cycles a non-critical instruction can be delayed. Results are far worsen than the ones from pipeline throttling. First of all we must note that, for the base case, almost half of the execution cycles are over the power budget (45%). Note also that our critical path predictor relies on finding holes where it can delay instructions from cycles over the budget. Therefore, the more cycles a program is over the power budget, the harder it is to balance the instructions so they can all execute under the budget. Second, in our implementation, the critical path predictor has no information about the power cost of each instruction neither how much power over the budget was dissipated in each cycle. Even if the mechanism had that information it would still need some inter-instruction communication in order to know if enough instructions have been delayed to be under the power budget. To keep it simple, this first approach delays as many non-critical instructions as found when the power budget is exceeded. In the next chapters we will enhance these simple mechanisms in order to increase their accuracy and their associated energy penalty when matching the power budget.

3.1.4. Hybrid Approaches

There are several proposals that try to merge both DVFS and microarchitectural techniques in a two-level mechanism to benefit from both coarse and fine-grain mechanisms. Sasanka *et al.* propose the use of DVS and some microarchitectural techniques to specifically reduce the energy consumption in real time video applications [83]. Their selected microarchitectural techniques try to reduce the power of functional units and the instruction window. Moreover, they don't use clock gating in their proposal, and the studied

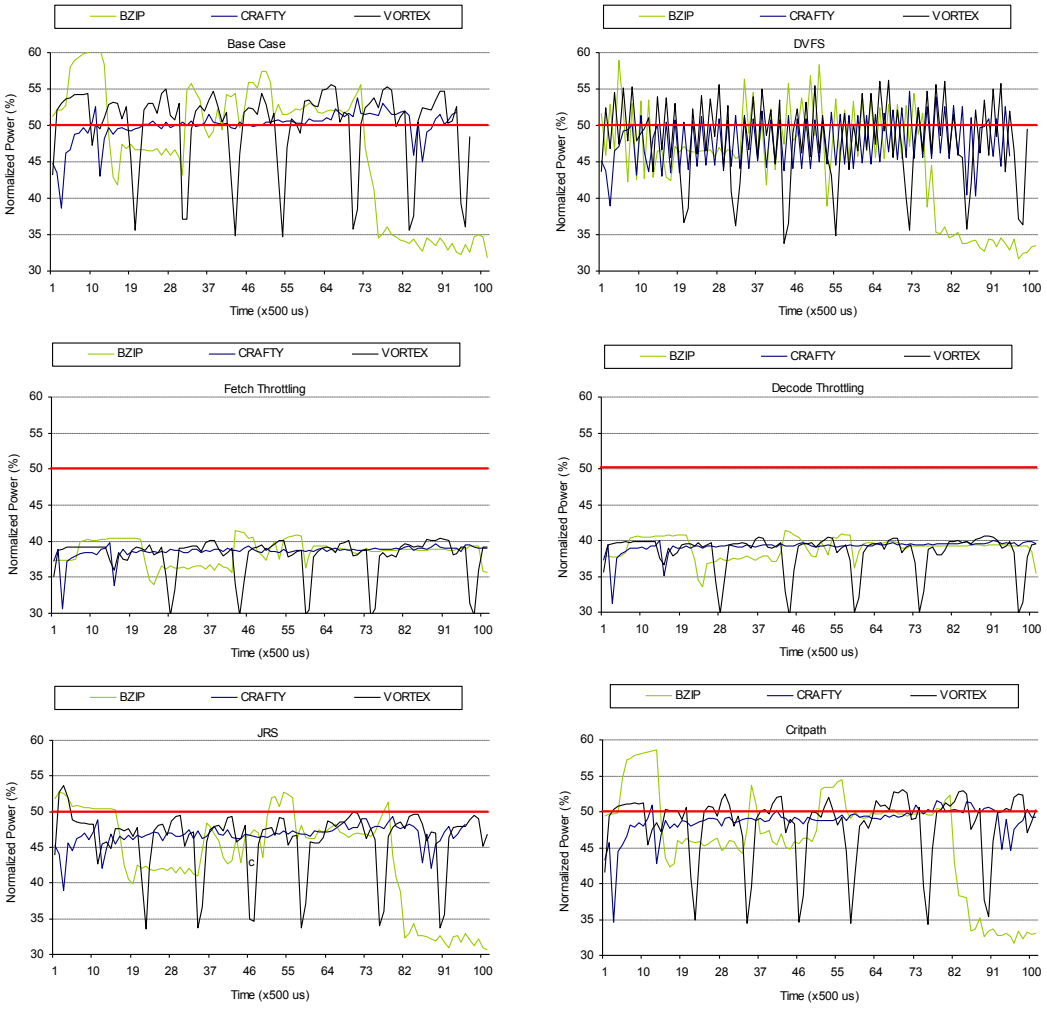


Figure 3.5: Per-cycle power dissipation for a power budget of 50%.

benchmarks have special properties that their selected microarchitectural techniques can take advantage of. In [26] we propose a more generic and adaptive hybrid mechanism, as it does not depend on profiling. This proposal is described in more detail in Chapter 4, but in essence what it proposes is the use of power-token information to select between different power saving mechanism depending on how far we are over the power budget. Although we do not implement any specific microarchitectural technique to reduce the consumption of the instruction window and functional units, the use of clock gating prevents these structures to consume when they are underused. Winter *et al.* [93] propose the use of a two-level approach that merges DVFS and thread migration to reduce temperature in SMT processors. More recently, Siddiqua *et al.* [86] propose the combination of DVFS with microarchitectural techniques to reduce temperature in the context of improving processor reliability.

3.1.5. Timeline Analysis of Power Dissipation

In the previous sections we have introduced some of the most common dynamic power saving mechanisms. Our primary goal in this Thesis is to accurately match a predefined power budget while minimizing the energy penalty. We don't know a priori if this power constraint will be far or close to the peak power dissipation of the processor, so we need to know how much these techniques can lower the power towards the budget, and how can we merge them to obtain further power reductions. As in previous sections we will use an aggressive power budget of 50% of the peak power dissipation as a challenging scenario for the power saving mechanisms to reduce power down to the budget.

This section shows a timeline analysis of the studied techniques for a given power budget of 50% (Figure 3.5) (we only show three benchmarks for the sake of visibility). The power dissipation of each technique is represented relative to the maximum power dissipation, so 50 means 50% of the peak power of the processor using clock gating. As not all benchmarks last the same, we only show the first 100 windows of 500 microseconds each. The critpath configuration shown is for a delay of 8 cycles. All fetch, decode and JRS configurations shown are enabled when the processor is over the power budget (red line in the figure) and produce a full stop of the front-end. In addition, we show a non-blocking DVFS mechanism (it can continue execution during power mode changes) that can work in two modes: the first one produces a 15% power reduction with 5% performance degradation and the second mode reduces power by 35% and performance by 10%.

With the information provided by Figure 3.5 we can classify the different mechanisms depending on how close they are to the power budget. Critical Path is the power saving mechanism that reduces the least power of all the studied mechanisms, followed by JRS. These two mechanisms are the ones with the least power reduction but also the ones with minimal performance impact on the processor. Meanwhile, DVFS constantly changes between power modes trying to get as close as possible to the power budget, showing low accuracy because of the long exploration and activation windows.

Note that both Fetch and Decode throttling (the most restrictive approaches) also have a predefined power budget of 50% but are stuck at 40%. This is not a coding mistake. Note that the graphs plot the average power dissipation on intervals of $500\mu s$, that is, for our processor configuration (3 Ghz), 1500000 instructions. Both mechanisms are applied at cycle level, and only during cycles that are over the PB. That makes the standard deviation from the average power quite high in these windows. When we reduce the power dissipation in the cycles that are over the power budget, the new average for the window is reduced. Figure 3.6 shows this effect graphically.

3.2. Leakage Control Mechanisms

As it was previously stated, leakage power is becoming an important source of power dissipation in modern microprocessor designs. Fighting to reduce leakage has been an

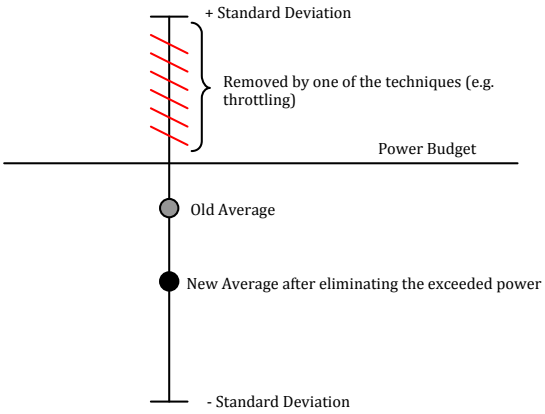


Figure 3.6: Standard deviation effect for one window ($500\mu secs$).

important topic in microprocessor development in the last few years. The techniques to deal with leakage have been categorized into non-state preserving and state-preserving [19, 58, 99, 10].

In the non-state preserving scenario, Powell *et al.* [76] proposed $gated - V_{DD}$ as a procedure to restrain leakage power by gating off the supply voltage of cells. This non-state preserving technique, known as decay, reduces the leakage power drastically at the expense of losing the cell's contents. Decay techniques must be applied very carefully since the information loss can result in an increase of the dynamic power. Kaxiras *et al.* [46] successfully applied decay techniques to individual cache lines in order to reduce leakage in cache structures (67% of static energy consumption can be saved with minimal performance loss). This technique has also been applied to conditional branch predictors and BTB structures [39, 42].

Conversely, the state preserving scenario is lead by drowsy techniques which try to reduce leakage without losing the cell's contents. Drowsy caches [33] use different supply voltages according to the state of each cache line. The lines in drowsy mode use a low-voltage level that allows a leakage reduction while retaining the data, but requiring a high voltage level to access it again. Waking up from the drowsy state is similar to a pseudo-cache miss incurring in some penalty cycles (about 7 cycles) according to [58]. Of course, the leakage power savings of this mechanism are lower than the decay ones, but the additional dynamic energy consumption due to the information loss is also decreased. Flautner *et al.* [33] showed that a drowsy cache, which is putting to sleep all cache blocks periodically, achieves 54% leakage power savings with negligible performance degradation (about 1%). The authors in [58] confronted the state and non-state preserving techniques for caches and showed that, for a fast L2 cache (5-8 latency cycles), decay techniques are superior in terms of both performance degradation and energy savings compared to drowsy ones.

As an example of an adaptive decay mechanism suited for caches, Zhou *et al.* [101] proposed an adaptive time based mechanism that dynamically disables cache lines in order to reduce leakage power dissipation. The mechanism takes advantage of the cache tag array, which is never switched off, to track if there are many induced cache misses in

order to adapt the length of the decay interval accordingly.
Finally, quasi-static four-transistor (4T) memory cells are an alternative to traditional decay techniques. These cells are approximately as fast as 6T SRAM cells, but do not have connections to the supply voltage (V_{DD}). Conversely, the 4T cells are charged upon each access whether read or write, and slowly leak the charge over time until, eventually, the values stored are lost. Authors in [42] applied decay techniques to branch predictors by using 4T cells, removing some of the drawbacks of $gated-V_{DD}$ transistors, since any access to a 4T cell automatically reactivates it, whereas reactivating a 6T cell from the "sleep" mode is somehow more complex, requiring extra hardware involved in gating the supply voltage.
In the next sections we will evaluate both state and non-state preserving techniques for an specific case of study, "Reducing leakage power in value predictors". We will propose several mechanisms to maximize leakage energy reduction with minimal performance penalty including Static Value Prediction Decay (SVPD) [23], Adaptive Value Prediction Decay (AVPD) [24] and a power-performance drowsy approach for Value Predictors (VP) [25]. These mechanisms are able to dramatically reduce the leakage energy of traditional value predictors with negligible impact on neither prediction accuracy nor processor performance. The proposed decay-based techniques dynamically locate VP entries that have not been accessed for a noticeable amount of time and switch them off to prevent leakage. On the other hand, the drowsy-based approach will locate these entries and lower the supply voltage, retaining the data but requiring five additional cycles to access it again.

3.2.1. Value Predictors: A Case Study for Leakage Reduction

Data dependences are one of the key factors that limit performance in current high-performance superscalar microprocessors. Previous works suggested that value prediction can overcome the limits imposed by data dependencies [35, 36, 60, 76]. Value prediction is a technique that predicts the output of an instruction as soon as it is fetched allowing subsequent instructions that depend on that result to execute using the predicted value. These instructions must be validated once the actual values are computed. VP has also been successfully used to perform early load retirements in high performance processors [52]. However, the use of value prediction techniques has not been widespread, despite the speedup provided (15% on average as reported in [21]), mainly due to complexity-delay issues. Note that unlike other prediction structures, such as branch predictors, the access time in VPs is not crucial. First of all, the predicted value is not needed until the instruction has reached its issue stage, and second, current high performance processors typically implement deeper pipelines (14 stages or more) which effectively hide the VP latency.
However, the use of VP structures incurs in additional dynamic and static power dissipation. The continuous access to the prediction tables in almost each clock cycle may result in a thermal hot spot, increasing the leakage power of the structure, as it also

happens in caches and branch predictors. In modern high performance processors, due to high operating temperatures, it is crucial to reduce leakage in every possible structure. Although the VP is a small structure compared to a L2 cache, if we let it overheat (likely, as it is accessed frequently and resides quite close to the core) without any precaution to regulate its leakage, the negative effects can be quite serious. Small hot structures can leak more than larger but cooler ones and we cannot afford to allow leakage even in the smallest structures.

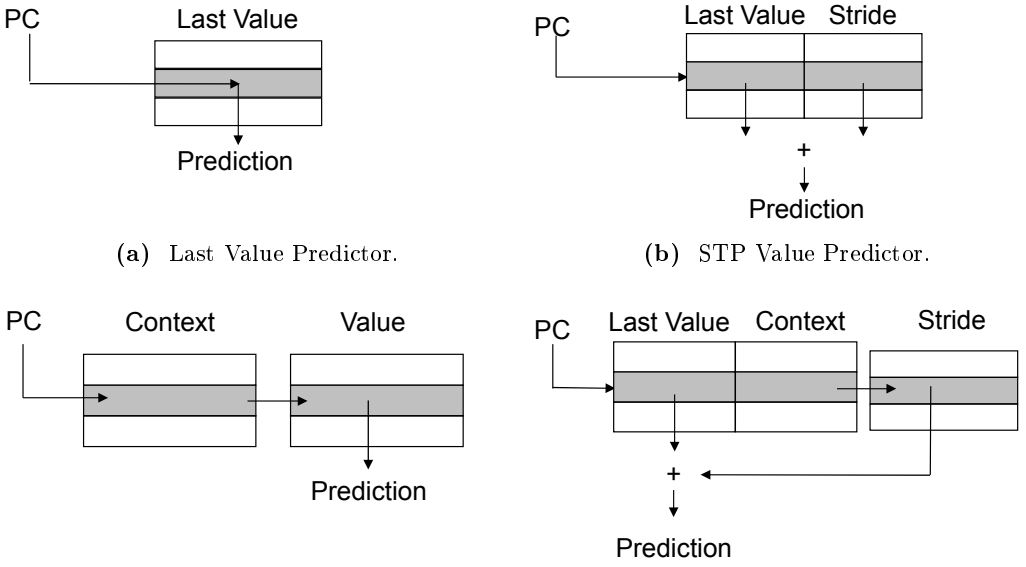
We will now introduce the most common value predictors and their particularities. The last value predictor (Figure 3.7a) was introduced by Lipasti *et al.* [60]. This is the most basic prediction mechanism and, basically, it assumes that the next value produced by an instruction will be the same as the previous one. A generalization of the last value predictor leads to the stride value predictor (STP). Introduced by Gabbay *et al.* [35], STP (Figure 3.7b) uses the last value produced by an instruction plus a stride pattern. In a stride pattern, the difference between two consecutive values is a constant. The next predicted value is computed by adding the last value to the stride.

The finite context method value predictor (FCM - Figure 3.7c), introduced by Sazeides *et al.* [84], uses the history of recent values, called the context, to determine the next value. This is implemented by using two-level prediction tables. The first level stores the recent history of the instructions outputs (VHT). The second level stores, for each possible context, the value which is most likely to follow that pattern (VPT). The value is predicted by using the program counter to access the VHT table and, according to the context hash function, the VPT table is accessed to get the predicted value.

Finally, the differential finite context method value predictor (DFCM - Figure 3.7d) introduced by Goeman *et al.* [36], joins the two previous predictors in one structure. DFCM works like FCM (two-level prediction tables), but it stores as context the differences between the outputs instead of the outputs themselves plus the last output of the instruction. In this way, DFCM can predict stride patterns using less storage space than FCM by adding the last value to the stride associated to the context. For non-stride patterns, DFCM works just like the FCM predictor.

3.2.2. Problem Overview: Generational Behaviour in Value Prediction Structures

As commented before, there are two sources of power dissipation in all processor structures, dynamic and static power. In value prediction, the dynamic component is produced by the repeated capacitance charge and discharge on the transistor gate outputs and strongly depends on the utilization of the VP tables: the more use the more consumption. The output values of instructions can be predicted at different demanding levels: the most aggressive utilization predicts the output for all instructions traversing the pipeline (this is the worst case scenario for power reduction techniques, and it will be the one used in the rest of this study). Other approaches restrict the use of the value predictor to just a fraction of instructions such as long-latency instructions, load instruc-



(a) Last Value Predictor.

(b) STP Value Predictor.

(c) FCM Value Predictor.

(d) DFCM Value Predictor.

Figure 3.7: Value Predictor Analysis.

tions that miss in the data caches, instructions that belong to the critical path, or just to predict the effective address for memory disambiguation. Therefore, restricting the VP utilization to just a fraction of selected instructions effectively reduces the dynamic power component of this structure. However, the static power component is still there: the VP structure leaks regardless of activity, especially through subthreshold leakage currents and gate leakage currents that flow even when transistors are nominally off, with increasing leakage loss for finer process technologies. For this reason, this proposal focuses on reducing the VP structure's static power component. Value predictors share many similarities with caches as they are both array structures. As happened in [46], looking at the value predictor entries behavior we can divide the stream of accesses into generations. A generation is defined as the period of time an entry is accessed by the same instruction. Entries have an initial usage time (known as live time) followed by a period of no utilization (known as dead time) before they are accessed by a different instruction, as shown in Figure 3.8. An entry's live time will be the period of time the entry is accessed by the same PC and its dead time will be the period of time between the last access with a specific PC and the first access with a new one. In order to evaluate the generational behavior and the utilization of the VP entries, Figure 3.9 shows the fraction of time each entry remains in the dead state[2] for the whole SPECint2000 benchmark suite as a function of VP size. This way we can determine if disabling all the entries during the dead times will produce enough savings to justify the mechanism. We can observe that the three evaluated value predictors -Stride, FCM and DFCM- present

[2]This fraction of time can be measured as the ratio: *total dead time* / (*total live time* + *total dead time*).

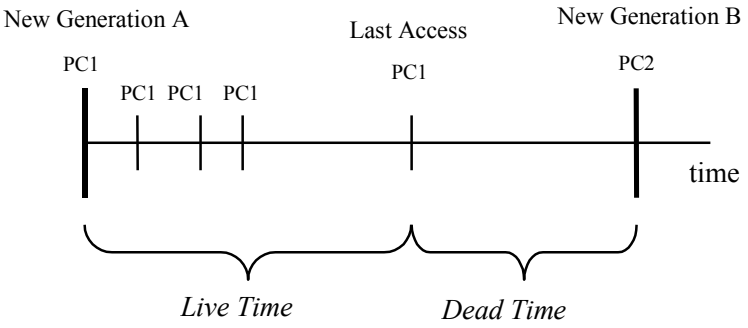


Figure 3.8: Temporal behavior of a value predictor entry.

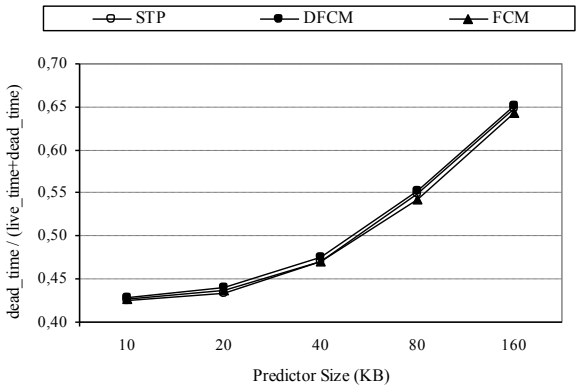


Figure 3.9: Fraction of time VP entries spend in dead state (SpecInt2000).

a similar utilization regardless of their size. For sizes around 20 KB, the average fraction of dead time is 43%, and for predictor sizes around 40 KB, the average fraction of time the entries spend in their dead state is 47%. Hence, if we were able to take advantage of these dead times by detecting them and shutting the entries off, we could reduce the leakage energy of the value predictor structure by one half on average. However, it is important to note that this is not an upper bound on the leakage energy savings that could be achieved by decaying VP entries. Long periods of inactive live time could also be detected to early shut the entry off in order to obtain further leakage savings, at the expense of slightly reducing the VP accuracy and processor performance, as we will show in the following sections.

3.2.3. Techniques for Reducing Leakage in Value Predictors

3.2.3.1. Non-State Preserving Scheme: Static Decay

We will begin introducing the non-state preserving techniques. Static decay is a technique that tries to locate unused VP entries in order to switch them off [23]. To detect if an entry is unused a decay interval is established, i.e. the number of cycles we should wait before shutting an entry off if there are no instructions accessing it. If we choose short decay intervals we will increase the energy savings, but we can degrade processor performance,

as we can disable entries during their live time, losing their contents. On the other hand, choosing long decay intervals will decrease the chances of disabling entries during live time, but we will lose some potential energy savings from dead times. Therefore, we need to track the accesses to each VP entry in order to detect if a particular entry is accessed very frequently or, conversely, the entry has been unused for a long period of time, probably entering into a dead state. For the static decay scheme it is crucial to explore a wide range of decay intervals to precisely detect the dead states while, at the same time, not degrading the VP accuracy. Ideally, the best static decay interval is the one that minimizes the performance impact of prematurely disabling a VP entry.

The static decay mechanism will be implemented by means of a hierarchical counter composed of a global decay counter and a two-bit saturated gray-code counter on each value predictor entry[3]. The local counters will only be incremented when the global counter reaches zero. Any access to an entry will reset its local counter. When a local counter reaches its upper limit it means that the entry has been unused for a decay interval, and can be shut off, reducing its leakage power dissipation to almost zero. As mentioned previously, the access time to the VP structure is not crucial but, in our study, we will use a moderate access time of 5 cycles.

To prevent VP entries from leaking we will use $gated - V_{DD}$ transistors [76]. These "sleep" transistors are inserted between the ground (or supply) and the cells of each VP entry, which reduces their leakage in several orders of magnitude and can be considered negligible. An alternative to using $gated - V_{DD}$ transistors consists of using quasi-static 4T transistors in the VP array, although similar leakage savings would be expected [42].

The length of the decay interval is controlled by the global decay counter. If we set the global decay counter to a low value then the VP entries may be disabled prematurely and leakage will be reduced drastically, at the cost of reducing the hit rate of the predictor. On the other hand, if we increase the global counter too much (a long decay interval), the leakage energy savings will not be as high as their potential.

Regarding the utilization of VPs, throughout the chapter we are predicting the output values for all instructions traversing the pipeline. However, it is important to note that this aggressive prediction scheme does not benefit a decay mechanism (either static or adaptive) since they are based on locating unused predictor entries. The more we use the VP structures, the less chances to detect unused VP entries and the less leakage energy savings obtained from a decaying mechanism.

There are several overheads that must be considered when performing the leakage energy savings evaluation. The first overhead component takes into account the extra dynamic and static power that results from the additional hardware (a global decay interval counter as well as the two-bit local counters[4] per VP entry [24]). The second

[3] Using a hierarchical counter is more power-efficient since it allows accessing the local counters at a much coarser level. Accessing the local counters each cycle would be prohibitive because of the power overhead.

[4] The dynamic and static power overhead of all 2-bit local counters has been measured to be less than 2% of the total VP structure.

overhead component comes from the induced VP misses (when a VP entry is prematurely disabled) that increases the execution time. These extra cycles that the program is running will also lead to additional static and dynamic power dissipation. Note that this second overhead is highly destructive since each extra cycle accounts for the overall processor dynamic and static power and can easily cancel the VP leakage energy savings provided by the decay scheme.

3.2.3.2. Non-State Preserving Scheme: Adaptive Value Prediction Decay

Adaptive Value Prediction Decay (AVPD) is, like Static Decay, a time-based mechanism that analyzes the VP tables to detect unused entries. If an entry is unused for a long period of time, it probably means that it has entered in a dead state, and we should turn it off. As we will see in the results section, the decay interval depends on the application running in the processor and even on the code section being executed. During program execution there are sections of code where the VP usually hits or fails its predictions (correct and wrong predictions appear clustered depending on the program phase). We can also find program sections where the number of VP entries being accessed is abnormally low, and even identify instructions whose optimal decay interval is different from others. Therefore, higher leakage energy savings could be obtained (compared to statically setting the decay interval) if we are able to dynamically adapt the decay interval to the program needs. The AVPD mechanism will use, basically, the same implementation as the static approach. The mechanism uses a hierarchical counter composed of a global counter and a two-bit saturated gray-code counter for each individual value predictor entry (local counters) to implement the decay interval. The AVPD mechanism considers that each VP entry can be in one of the following three states, as shown in Figure 3.10: enabled (both data and the local counter are enabled), partially disabled (data is shut off but the local counter is enabled) or disabled (both data and the local counter are shut off).

AVPD uses two additional global counters that account for: a) the number of partially disabled entries (entries that change from the enable state to the partially disabled state) within the previous decay interval; and b) the number of re-enabled entries (entries that change from the partially disabled state to the enabled state) within the current decay interval. After a number of cycles equal to the average live time[5], a re-activation ratio is calculated as the number re-enabled entries over the number of partially disabled entries. As we cannot identify the instruction accessing the value predictor (because it is implemented as a non-tagged table), it is not possible to determine if there is a real generational change or if we disabled the entry during its live time. Therefore, we will use a time-based approach. We will add an intermediate state between the enabled and disabled states and account, during a short period of time, if there are many entries that are re-enabled (come back from the intermediate state to enable state). If that happens,

[5]The static decay experiments showed that the average live time is around 400 cycles for the three evaluated VPs.

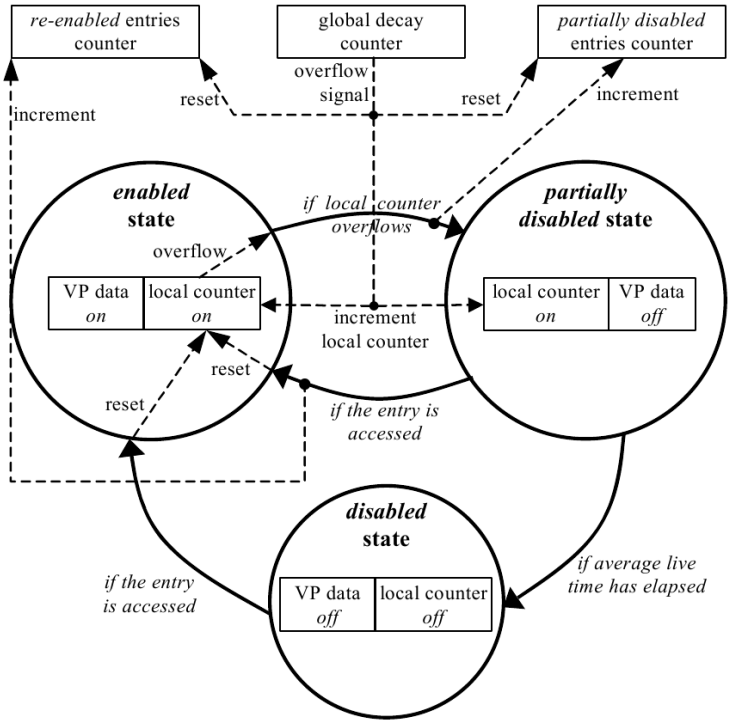


Figure 3.10: AVPD mechanism workflow.

the decay interval is too short. If there aren't many re-enabled entries we can try to lower the decay interval to increase savings. A positive effect of AVPD compared to the original cache decay mechanism is that prematurely disabling a VP entry is not as harmful as disabling a cache line: losing the contents of the cache line always leads to an extra access to L2 cache or memory to retrieve the lost information incurring in extra execution cycles. However, losing the contents of a VP entry might result -or not- in a value misprediction on the next access to that entry but this is exactly what would happen if we had a real generation change (which is a very common situation and one of the major limitations in traditional non-tagged VPs, where the huge number of destructive interferences dramatically shortens the generational replacement).

In addition, AVPD uses two pre-defined threshold values (*increasing threshold* and *decreasing threshold*) in order to determine whether the length of the current decay interval is correct: i.e., if the current decay interval makes VP entries to decay during their live time (prematurely) or during their dead time (as expected). Therefore, if the re-activation ratio is higher than the *increasing threshold* then the current decay window is too short and it is doubled since there are many entries being disabled prematurely. On the other hand, if the re-activation ratio is lower than the *decreasing threshold*, the current decay window is too long and it is halved since we are shutting entries off too late, losing opportunities to reduce the VP leakage. In order to make the AVPD mechanism easier to implement we will use power-of-two decay intervals. VP entries are shut off, preventing them from leaking, by using $gated - V_{DD}$ transistors [76].

The AVPD mechanism works as follows (see Figure 3.10): each cycle the global decay

counter is incremented by one and, when it overflows, the local counters of all VP entries in either enabled or partially disabled states are incremented. However, an access to a VP entry will result on an immediate reset of its local counter. In addition:

- For those entries in the enabled state (both VP data and the local counter are enabled): If the entry remains unused for a long time, its local counter will eventually overflow and the entry will change to the partially disabled state. The number of partially disabled entries is incremented.
- For those entries in the partially disabled state (VP data is shut off whereas the local counter is enabled): If the entry is not accessed within the average live time then it will be changed to the disabled state and the local counter will also be shut off. However, an access to a partially disabled entry will change it to the enabled state, increasing the number of re-enabled entries.
- For those entries in the disabled state (both VP data and the local counter are shut off): Any access to the entry will change it to the enabled state.

Regarding the pre-defined values used for the *increasing* and *decreasing thresholds*, it is important to note that setting the *decreasing threshold* to low values will reassure AVPD that there are few re-enabled entries before lowering the decay interval, resulting in a more conservative policy. On the other hand, setting the *decreasing threshold* to high values will make AVPD to reduce the decay interval more frequently, resulting in a more aggressive policy. Analogously, setting the *increasing threshold* to low values means that AVPD will increase the decay interval even if there are few re-enabled entries; whereas setting the *increasing threshold* to high values will make AVPD to wait until it has a great fraction of re-activations before increasing the decay interval. In Section 3.2.4.5 we evaluate the leakage-efficiency of the AVPD mechanism for different *increasing* and *decreasing thresholds*.

As for this static approach, we can split the power overhead of the AVPD mechanism into three main components. The first component is associated to the dynamic and static power derived from the two-bit local counters inserted into every predictor entry (same overhead as for the static decay scheme). The second component comes from the three global counters: one is part of the hierarchical decay interval counter (also appears in the static decay scheme) and the other two counters are specific of the adaptive decay scheme. The third component is derived from the induced VP misses (when a VP entry is prematurely disabled) that increase program execution time. These extra cycles that the program is running will also lead to additional static and dynamic power dissipation. It is important to note that AVPD is virtually neither introducing additional power overhead nor complexity when compared to the static decay scheme (just the additional two global counters whose power overhead has been conveniently modeled into the AVPD power model).

3.2.3.3. State Preserving Scheme: Drowsy

The drowsy technique aims to reduce leakage power dissipation while preserving the contents of a cell by switching between two working modes, low-power and high-power. While the cell is in low-power mode, the information is preserved, but it cannot be accessed. In order to access the cell again it must be reinstated into the high-power mode. A drowsy scheme can be configured according to the following parameters [33]:

- Update window size: Specifies if the amount of cycles to wait before turning entries into drowsy mode can be varied.
- Simple or Noaccess policy: "Simple" means that all entries are turned into drowsy mode after a number of cycles. "Noaccess" puts to drowsy mode only the entries that have not been accessed in a number of cycles.
- Awake or drowsy tags: put tags into drowsy mode or not (affects latency).

According to [33], the simple policy with a window size of 8000 cycles comes very close to the behavior of the no-access policy with a window size of 2000. They choose a simple policy with a window size of 4000 cycles for their tests, as it reaches a reasonable compromise between simplicity of implementation, power savings and performance. In our case, it is not fair to compare the decay and drowsy approaches using the simple policy since the decay approach will delete all the information of the structure every decay interval (data loss will mean an access to L2 or memory in decay, but only a few cycles for drowsy to re-enable the entry). Instead, we will then use an update window-noaccess policy[6]. In our case, drowsy will be used exactly as decay, at an entry level. The leakage power dissipation of a transistor in drowsy mode is measured to be 15% of the original leakage (as estimated in [33]), something significant compared with the almost 0% of the decay scheme.

There are several ways of implementing this two power-level technique (ABC-MTCMOS, DVS, etc). The Dynamic Voltage Scaling (DVS) [33] mechanism allows structures to dynamically change their speed and voltage while operating, increasing their energy efficiency. If we refer to memory structures, it is possible to reduce their supply voltage while retaining the data. Scaling the voltage of the cell to 0.5 times the supply voltage can maintain the cell's state. This reduction of the operating voltage allows us to reduce the leakage currents, and thus the leakage power of the memory cell. DVS must be controlled by a voltage scheduler to dynamically adjust the processor speed and voltage during execution. Voltage schedulers analyze the state and context of the system in order to predict future workload of the processor, increasing the complexity of scheduling.

Auto-Backgate-Controlled MT-CMOS is a mechanism that dynamically increases the transistors threshold voltages when going to "sleep" mode by raising the transistors body voltage. This higher V_T reduces the leakage current without losing the cells contents when going to "sleep" mode but offsets the total leakage power savings. This technique also

[6] The awake/drowsy tags policy cannot be evaluated because the evaluated VPs do not have tags.

requires high energy to switch between states increasing the time needed to transition. DVS is faster, easier to implement, and obtains more power reduction than ABC (and will be used in our results) but depends on the process technology and is more sensible to SEU[7] noise.

3.2.4. Experimental Results

3.2.4.1. Simulation Methodology

To evaluate the energy-efficiency of SVPD, AVPD and drowsy techniques we have used the SPECint2000 benchmark suite. All benchmarks were compiled with maximum optimizations (-O4 -fast) by the Compaq Alpha compiler and they were run using a modified version of HotLeakage power-performance simulator [100] that includes the dynamic and static power model for the evaluated Value Predictors (Stride, FCM and DFCM) as well as the power overhead associated to SVPD, AVPD and drowsy techniques. All benchmarks were run to completion using the reduced input data set (test). The VP access latency has been set to 5 cycles for the three evaluated VPs.

We use the same processor configuration than for the dynamic power analysis (Table 3.1). The STP value predictor has 76 bits per entry: Value (64) + stride (8) + confidence (2) + decay counter (2). FCM has a variable number of bits per entry on the first level table: First level decay counter (2) + history bits for second level table (variable) and 68 bits per entry on the second level table: Value (64) + confidence (2) + second level decay counter (2). DFCM has 68 bits per entry + a variable amount of bits on the first level table: Value (64) + first level decay counter (2) + confidence bits (2) + history bits for second level table (variable) and 12 bits on the second level table: Confidence (2) + decay counter (2) + stride (8). The leakage related parameters have been taken from the Alpha 21264 processor provided with the HotLeakage simulator suite using a process technology of 65 nanometers.

3.2.4.2. Leakage-Efficiency of State and Non-State Preserving Mechanisms

As we discussed earlier, predicting the output values for all instructions traversing the pipeline does not benefit any of the proposed mechanisms, neither static nor adaptive, since they are based on locating unused predictor entries. The more demanding use of the VP structures, the less opportunities to detect unused VP entries. Figure 3.11 shows the differences in speedup for both static decay and drowsy with decay intervals from 256 Kcycles to just 64 cycles in order to better understand the effects of prematurely deactivating a VP entry. We must never forget that traditional value predictors (with no power-saving techniques) can provide significant speedups (13% for a 10 KB DFCM). Looking into the performance degradation caused by static decaying, we can notice that for FCM and DFCM predictors there is no IPC degradation until 256-cycle decay intervals. For the STP predictor, there is a negligible IPC degradation (less than 1%) for

[7]Single Event Upset.

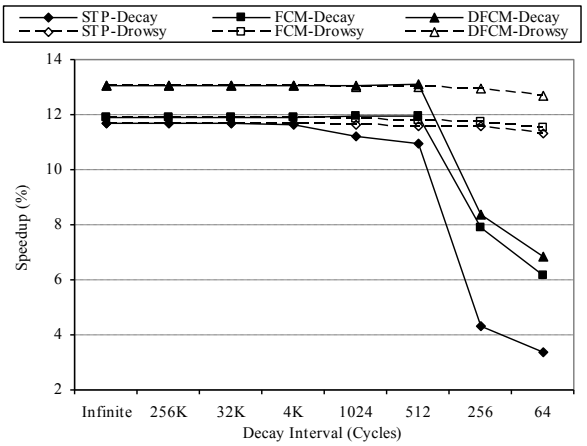


Figure 3.11: Average speedup for the static decay and drowsy schemes for 10 KB VPs.

1024 and 512-cycle decay intervals. For 256-cycle (and smaller) intervals the performance degradation is not tolerable. We measured the VP entries average live time to be about 400 cycles. Therefore, if the decay interval exceeds that critical point then the speedup provided will fall apart.

On the other hand, the drowsy technique "normalizes" the optimal decay interval for all predictors to 256 cycles, behind the average live time, but close enough to it to maintain the speedups provided by the value predictor. Note that drowsy does not lose the entries contents, which makes drowsy not to reduce the performance as quickly as decay. But this does not make drowsy more energy-efficient, as performance is only an element of the energy metric, and the power reduction provided by drowsy is lower than that of decay.

3.2.4.3. Static Decay for VPs

In this section we perform an evaluation on the energy-efficiency of the static decay scheme for value predictors with varying sizes (sizes are not power-of-two numbers because of the extra 2-bit counters per entry) and for several decay interval windows: 64, 256, 512, 1024, 4K, 32K and 256K cycles. For each evaluated VP we report the IPC degradation as we reduce the decay interval and the corresponding leakage energy[8] savings for the VP structure. Overall leakage energy savings are not presented due to HotLeakage limitations that only provide static power models for regular array structures such as caches, predictors, and the register file.

Figure 3.12-top shows the performance degradation of the STP value predictor for different sizes (SpecInt2000 average). Looking into the performance degradation caused by static decay we can notice that it is degraded as expected, due to the data loss of prematurely deactivating VP entries that still are in a live state. In particular, there is a slight IPC degradation (around 1%) for 1024- and 512-cycle decay intervals. However,

[8]Recall that the performance degradation is also included in the energy metrics ($Leakage_{Energy} = Leakage_{Power} \cdot Delay$).

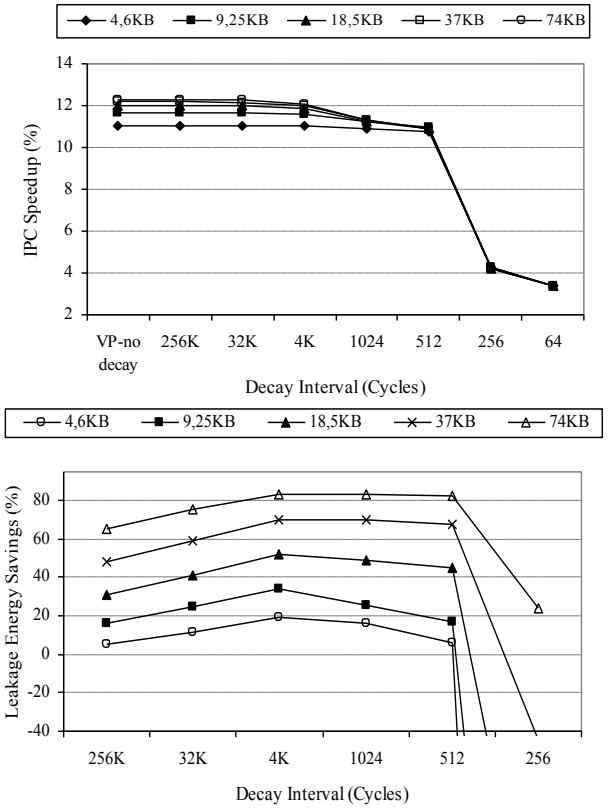


Figure 3.12: STP performance degradation (top) and leakage energy savings (bottom).

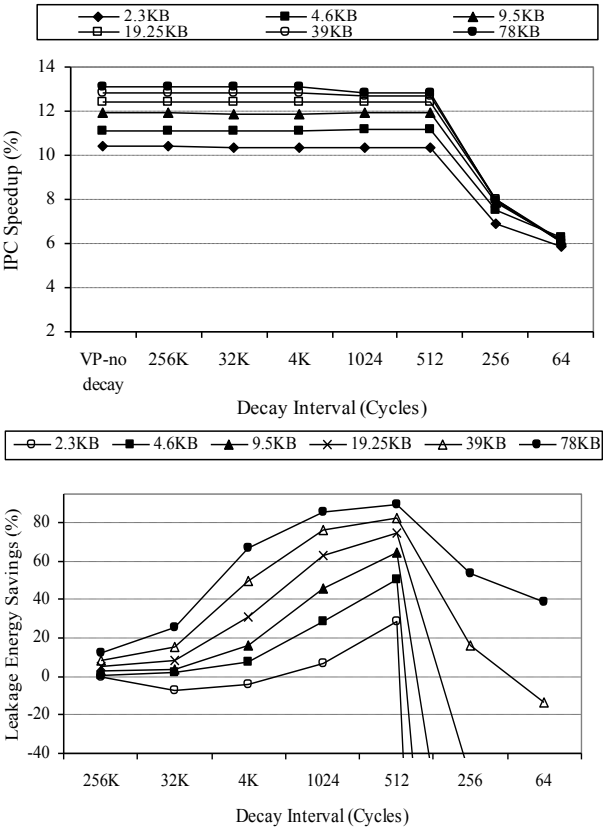


Figure 3.13: FCM performance degradation (top) and leakage energy savings (bottom).

due to early deactivation of entries and the induced extra execution cycles for 256-cycle and smaller decay intervals, the performance loss is intolerable.

Figure 3.12-bottom shows the average leakage energy savings of the STP predictor. As expected from the IPC degradation, for very small decay intervals (64 and 256 cycles), the early deactivation of entries results in no leakage energy savings at all due to the induced extra execution cycles that completely cancel whatever leakage power savings provided by the decay mechanism. However, for 1024 cycles and, particularly, for a 4K-cycle decay interval, the proposed VP decay approach obtains 52% average leakage energy savings when considering a medium size (20 KB) predictor.

Figure 3.13 show the performance degradation and the average leakage energy savings of the FCM value predictor. Since FCM is a two-level predictor (with the relevant data stored in the second level table) we will disable both levels independently. We can notice a similar behavior to the STP predictor for the very small decay intervals (64 and 256 cycles), again with negative leakage energy savings due to the early deactivation of entries and the induced extra execution cycles. On the other hand, for very big decay intervals (32 K and 256 K-cycles), the overhead is almost zero, but we obtain very small leakage energy savings (Figure 3.13-bottom) since there are almost no deactivations of VP entries. However, as we reduce the decay interval length, there is an increase in leakage savings with a maximum in the 512-cycle interval. For this decay interval, a 20 KB FCM predictor obtains average leakage energy savings of 75%, showing the benefits of Value Prediction Decay.

Figure 3.14 show the performance degradation and the average leakage energy savings of the DFCM value predictor. We can notice that the DFCM predictor behaves very close to FCM. However, for big decay intervals, DFCM obtains better energy savings (Figure 3.14-bottom) due to a positive side effect when shutting entries off, that results in a reduction of destructive interferences, this effect is explained as follows. Imagine that we have entries in the first level table with different predictions that point to different second level table entries. If the instruction that these entries are predicting changes, there is "trash" on the tables from the previous prediction that will interfere during some cycles with the new information until the predictor gets stable, whereas if we decay, all entries will reset, so they will always behave the same. Again, as we reduce the decay interval length, there is an increase in leakage energy savings with a maximum in the 512-cycle interval. In this case, for a predictor size around 20 KB we obtain average leakage energy savings of 65%. For both FCM and DFCM predictors, the best energy savings are obtained for a decay interval within the 512-cycle range, unlike data caches where the best decay intervals are within the 8-Kcycle range [46].

3.2.4.4. Drowsy for VPs

This section shows the results of the drowsy technique applied to DFCM, STP and FCM value predictors. In order to access an entry in drowsy state we add 5 additional latency cycles to the original 5 latency cycles of the VP structure. Figure 3.15-top shows

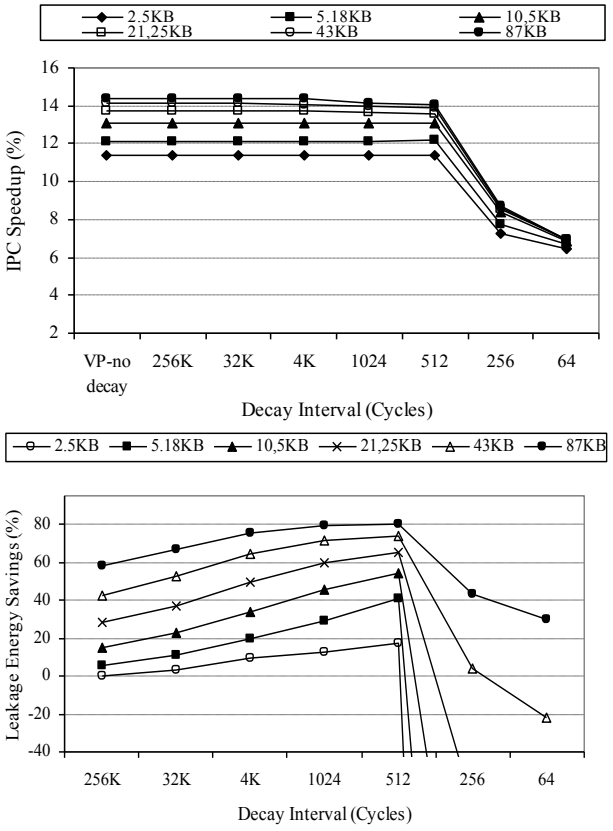


Figure 3.14: DFCM performance degradation (top) and leakage energy savings (bottom).

the average energy savings for a DFCM predictor using the drowsy technique applied statically. Compared to Figure 3.14b-bottom, the best drowsy configuration obtains less leakage energy savings than the best static decay. Despite the fact that drowsy shortens the best decay interval from 512 to 256 cycles, the performance degradation and the additional overhead (15% extra consumption) makes drowsy to lose against static decay for DFCM. For a predictor size around 10 KB, drowsy obtains average leakage energy savings of 44% while for a size around 20 KB, the average leakage energy savings are 57%.

In the STP predictor case (Figures 3.15-middle and 3.12-bottom), drowsy behaves better than decay. As we can see in Figure 3.11, the speedup degradation in the static decay for STP predictor begins in 4K cycles but, for drowsy, it starts at 256 cycles. This reduction of the decay interval and the low IPC degradation makes energy savings greater in drowsy than in decay.

As happened previously with the DFCM predictor, the FCM predictor behaves better for the static decay scheme than for drowsy (Figures 3.13-bottom and 3.15-bottom). As we can see in Figure 3.15-bottom, the best decay interval depends on the predictor's size. For small predictor sizes, the best decay interval is 256 cycles, and, for anything greater than 10 KB, the best decay interval is 512 cycles. This is due to the extra penalty cycles in the FCM predictor for sizes bigger than 10 KB. For a predictor size around 10 KB, the drowsy scheme obtains average leakage energy savings of 51% while, for a size about

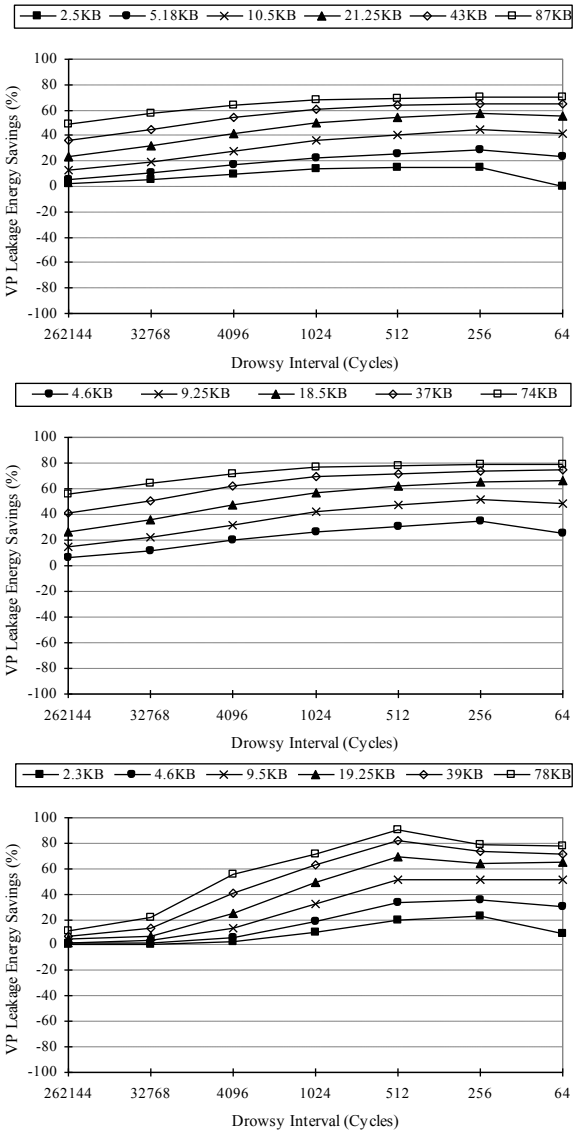


Figure 3.15: Static drowsy scheme for the DFCM (top), STP (middle) and FCM (bottom) value predictors.

20 KB, it obtains 64% energy savings.

3.2.4.5. Adaptive Decay for VPs

Using a static decay interval means that we must choose that decay interval carefully in order to maximize the leakage savings. Figure 3.16 shows the leakage energy savings, per benchmark, for a 10 KB DFCM value predictor using the static decay technique. In some cases, the best static decay interval may differ between applications (as it can be seen in Figure 3.16, where the best static decay interval is 4 Kcycles for some benchmarks -mcf and gap- and 512 cycles for the rest). However, even when using profiling techniques in order to determine the best decay interval per application, there is no guarantee that the best leakage savings are obtained, since the static decay approach cannot capture variations within an application. This second effect is important in the case of value

prediction structures since right and wrong predictions appear clustered depending on the program phase. Therefore, an adaptive decay scheme can dynamically choose decay intervals at run-time to more precisely match the generational behavior of prediction tables entries.

This section presents the leakage-efficiency evaluation of the proposed AVPD mechanism for the Stride, FCM and DFCM predictors, compared to the best configuration of both static decay and drowsy schemes. Each subfigure in Figure 3.17 shows the average VP leakage energy savings for some representative configurations of the adaptive mechanism as well as the best static decay (512-cycle decay interval according to section 3.2.4.3) and the best static drowsy configuration (256-cycle interval according to section 3.2.4.4) for comparison purposes.

For the evaluation of AVPD we carried out a comprehensive set of experiments for many configurations using different *decreasing* and *increasing threshold* values. In this Thesis we only present the most representative configurations:

- Configuration 00/100 (*decreasing threshold* set to 0% / *increasing threshold* set to 100%): this is the most conservative policy since AVPD will only try to decrease the decay interval if none of the entries are re-activated; and it will only try to increase the decay interval when all the entries are re-activated. It works pretty well for all the studied predictors as it does not take any risks when changing the decay interval.

- Configuration 50/50: this is the most aggressive configuration as it continuously keeps changing the decay interval, increasing or decreasing the decay interval according to the re-activation ratio.

- Configurations 40/60 and 70/100: these are the best ones we have found for the different predictors. The 40/60 is also aggressive but works well with the Stride predictor, as it balances long decay intervals with short ones. The 70/100 configuration trends to shorten the decay interval whenever is possible, only raising it when all decayed entries are re-activated.

Figure 3.17-bottom shows the average leakage energy savings for the DFCM predictor and the cited adaptive configurations as well as the best static and drowsy decay intervals (512 and 256 cycles respectively). As it can be seen, for DFCM the best adaptive configuration is 70/100. This configuration surpasses the best static decay and drowsy schemes for all evaluated predictor sizes. For an average size of 10.5 KB, AVPD obtains 64% average leakage energy savings versus 55% of the static scheme and 44% of drowsy. For the smaller size of 5 KB, the difference between the adaptive and static schemes is even more evident: AVPD provides additional average leakage energy savings of 14% respect to the static scheme (AVPD obtains 55% and the static scheme just 41%) and 26% respect the drowsy scheme (55% versus 29%). It can be observed that, as size grows, the differences between the adaptive and static schemes disappear, both obtaining 80%

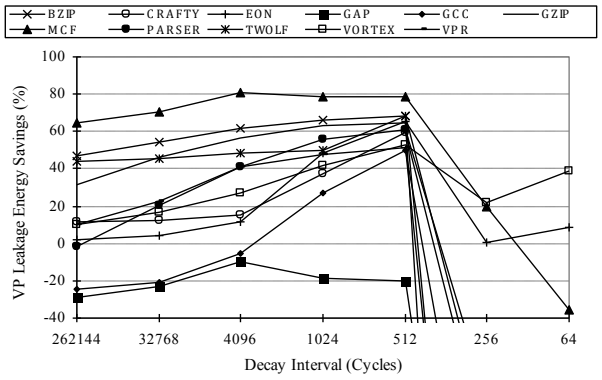


Figure 3.16: Static decay scheme for a 10 KB DFCM value predictor.

average leakage energy savings for a size of 87 KB. In such large predictors there is no need for an adaptive scheme, because there are very low generational changes, and they can be easily identified by the static scheme. The 70/100 configuration is the best one we have found since it tends to reduce the decay interval towards its lower limit (256 cycles). In general, we have seen that whatever configuration that tends to shorten the decay interval will perform well with DFCM, but constant changes of the decay interval will result in a loss of net leakage energy savings.

Figure 3.17-top shows the average leakage energy savings for the STP predictor. As cited in section 3.2.3.2, the AVPD mechanism tries to decrease the decay interval in order to reduce the leakage energy. The STP predictor is especially susceptible to these trials of reducing the decay interval since a big interval reduction degrades the STP accuracy enough (as shown in Figure 3.11) to make the power overhead due to the induced extra cycles equal to the power savings provided by AVPD. This makes the adaptive scheme to behave similarly to the static scheme. The STP predictor works better with configurations that change the decay interval quickly, like 50/50 or 40/60, because configurations that trend to shorten the decay interval (like 70/100) decrease the predictor's accuracy too much, making the overhead much greater than the provided energy savings. On the other hand, as said in section 3.2.4.4, drowsy dominates over decay approaches for the STP due to the reduction of the decay interval from 4K cycles to 256 (without almost no performance penalty). This reduction increases power savings more than the drowsy costs, without any major effects on performance, making the energy savings larger for drowsy than decay schemes.

Finally, figure 3.17-middle shows the average leakage energy savings for the FCM predictor. This value predictor behaves very similarly to DFCM, with the same best configuration of 70/100, but obtaining even greater leakage energy savings. In addition, the differences compared to the best static decay and the best drowsy schemes are also higher. For a predictor size of 4.6 KB, the static decay approach obtains 50% leakage energy savings whereas the adaptive scheme obtains 74% (an additional 24%). Drowsy only obtains 35% average leakage energy savings for that size. For bigger sizes, the difference between the static and adaptive schemes keeps lowering until it converges to the

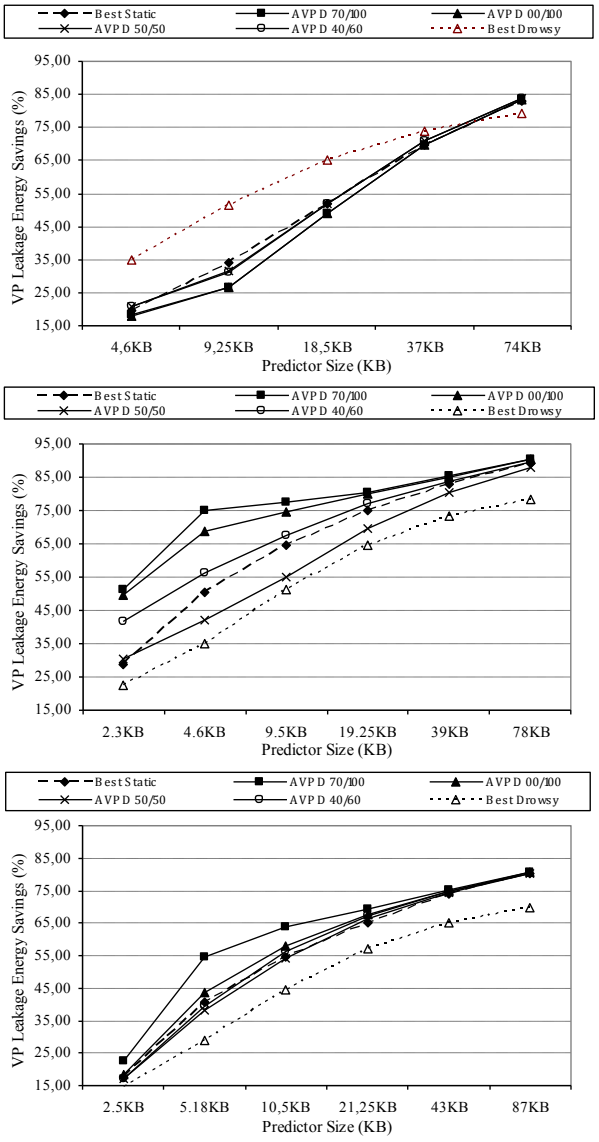


Figure 3.17: STP (top), FCM (middle) and DFCM (bottom) value predictor leakage energy savings.

same leakage energy savings for big predictor sizes (close to 90% savings for a size of 78 KB), but still perform better than the static drowsy. If we focus on moderated FCM sizes (around 10 KB), the best static decay scheme obtains 64% average leakage energy savings, while the best drowsy obtains 51%, and the AVPD obtains 77% (13% and 28% of additional savings). Note that FCM, like DFCM, performs well with any other configuration that tends to decrease the decay interval, due to the negligible impact on its accuracy.

3.3. Conclusions

Power dissipation, energy consumption and thermal output have been, for the past 8 years, the main limiting factors in microprocessor design. This chapter evaluates several mechanisms that are able to reduce both dynamic and static power dissipation in

microprocessors.

Dynamic power is nowadays the main source of power dissipation in microprocessors. The studied power saving mechanisms allow the reduction of this power dissipation at both circuit level (DVFS) and microarchitecture level. DVFS is a circuit level mechanism that uses the relation between dynamic power and both voltage and frequency ($P_d \approx V_{DD}^2 \cdot f$) to save power. The impact on performance for sequential or multi-programmed applications comes from the frequency reduction performed by DVFS. Performance impact can be lower in memory intensive applications because we only modify the processor frequency and not the memory frequency. Therefore, offchip memory latency is reduced as we lower the processor frequency. The performance impact of DVFS in CMPs running parallel workloads can vary depending on how we apply DVFS. If we apply DVFS at a core level and delay a critical thread then we will observe a performance degradation on the overall application because of synchronization points. If we are able to balance the frequency of the different execution threads we can save power without any performance impact. On the other hand, if we chose to use DVFS at a chip level instead of at a core level, the CMP will behave as the sequential case, with lower performance impact in memory intensive applications. DVFS is best suited for reducing the average energy consumption towards a target power budget, but it is too coarse-grained to remove all power spikes. This is due to the fact that DVFS is based on average energy consumption of exploration windows and is applied during long time intervals to recoup for DVFS overheads.

In this chapter we also evaluate some microarchitectural techniques in terms of power and performance. These techniques can be organized from least to most restrictive as: critical path estimator, branch confidence estimators (JRS) and pipeline throttling. While DVFS is quite unstable in terms of power (Figure 3.5), all the microarchitectural mechanisms reduce power dissipation smoothly (without major power spikes). However, the performance impact of these mechanisms is quite high. For pipeline throttling we can reduce the number of cycles over the power budget (CoPB) from 50% to 25-30% with a performance slowdown of 8-10%. Moreover, there is still a gap between the reduction on cycles over the power budget and the perfect power budget matching. In order to increase the accuracy when matching this power budget we could design a hybrid mechanism that uses DVFS to lower the average power towards the power budget (coarse-grain approach) and then use microarchitectural techniques to remove power spikes (fine-grain approach). In addition, all the studied mechanisms are triggered once the power dissipation goes over the budget, we could optimize the way we enable the different mechanisms by using some historic information. We will further analyze these approaches in the next chapter.

Finally, to study static power saving mechanisms we have performed a case study to reduce the average leakage energy of traditional value predictors with negligible impact on neither prediction accuracy nor processor performance. This makes Value Prediction a low-power approach to increase performance in energy-efficient microprocessor designs. Note that it is important to reduce leakage in any possible structure, despite its size,

as smaller and hotter structures can leak more than larger but cooler ones. We have evaluated both state and non-state preserving techniques for reducing leakage power in value predictors including Static Value Prediction Decay (SVPD) [23], Adaptive Value Prediction Decay (AVPD) [24] and a drowsy approach. SVPD and AVPD are mechanisms able to dramatically reduce the leakage energy of traditional value predictors with negligible impact on performance. These techniques dynamically locate VP entries that have not been accessed for a noticeable amount of time and switch them off to prevent leakage. On the other hand, the drowsy approach will locate these entries and lower the supply voltage, retaining the data but requiring additional cycles to access it again.

This case study shows that the AVPD approach (with just slight modifications and virtually no extra hardware compared to the static decay scheme) is able to beat both static decay and drowsy for the most precise value predictors (FCM and DFCM) while the STP predictor behaves better with the drowsy approach. The reason is that the STP predictor is more sensitive to losing the entries contents than FCM and DFCM, since the STP has all the information stored in only one table while the other are two-level predictors. Experimental results show that, for the DFCM value predictor and considering an average predictor size of 10 KB, the leakage energy savings obtained by AVPD surpass the static approach by 14% and the drowsy approach by 24%, on average. We can conclude that, as long as the data contents of the structure are easy to restore, the decay approach works better than the drowsy approach.

Chapter 4

Single-Core Power Budget Matching

Summary: In this chapter we recall the concept of power budget, why it is important and how to properly enforce it. In the previous chapter we analyzed the effects of power control mechanisms on performance and total cycles and energy over the power budget. This analysis showed that most of the studied mechanisms have problems when matching the target power constraint on their own, so it is not well respected. In order to solve this problem we propose three mechanisms: Power Token Throttling (PTT), Basic Block Level Manager (BBLM) and a two-level approach (DVFS+BBLM). PTT is a token-based approach that keeps track of the current power being dissipated by the processor, measured as tokens, and stalls fetch if the next instruction to be executed will make the processor go over the power budget. BBLM is a predictive power-control mechanism that stores information about the energy consumption of a basic block (or chain of basic blocks). BBLM will select from different architecture-level power saving mechanisms (more or less aggressive) based on the energy this basic block consumed the last time it was executed. Finally, the two-level approach will use DVFS as a coarse-grain approach to lower the average power dissipation towards the power budget along with BBLM to remove the remaining power spikes.

4.1. Introduction

In the past few years computer architects have faced an important design change. Individual core performance is saturating and processor designs are moving to multi-core approaches looking for throughput in addition to a reduction in application execution time. In order to reduce packaging costs, processors are generally designed for average energy consumption and thermal constraints (it is improbable that a processor uses all of its resources at once) and face special cases with both power saving and thermal management mechanisms.

As cited in Chapter 2, Dynamic Thermal Management (DTM) is a mechanism that reduces the processor power dissipation (and therefore performance) during critical time

intervals so it can cool down. One way to achieve this goal is to set a power budget to the processor. This processor's power budget is not only useful to control the processor power and temperature but also to adapt to external (off-chip) power constraints.

There are situations where device power requirements are much lower than the power needs of the processor at full speed. In most of the cases we cannot afford to design a new processor to meet whatever power constraint because it is too expensive. Therefore, being able to set a power budget to the processor could help designers to reuse existent hardware in new devices. The problem grows when, as usual, power constraints are transitory and after some period of low performance to save power we want all the processor's performance back. For example, imagine we have a computation cluster connected to one or more UPS units to protect it from power failures. If there is a power cut, all processors will continue working at full speed consuming all of the UPS batteries quickly. After some time it will switch the computers off when the batteries are close to run out and, consequently, losing all the work on fly. During the power failure (many times they are of limited duration) it might be more interesting to extend the UPS battery duration at the expense of degrading some performance, than to lose all the work done.

Another situation where setting a power budget could be useful is the case of a computing centre that shares a power supply among all kind of electric devices (i.e., computers, lights, air conditioning, etc.). In a worst case scenario (e.g., in summer at mid-day with all the computers working at full speed), if we integrate some kind of power budget management into the processors, we could set a power budget to lower the ambient temperature also having more power for the air conditioning. In this way, we could design the power capacity of the computing centre for the average case, reducing its cost.

As seen previously, one way to make the processor's power go under a power budget is Dynamic Voltage and Frequency Scaling (DVFS) [62, 85, 87, 96]. DVFS relies on modifications in both voltage and frequency to reduce the processor's dynamic power, as dynamic power depends on both voltage (quadratically) and frequency (linearly). DVFS has been implemented in many commercial processors and is commonly used by DTM techniques [32]. The major advantage of DVFS is its precision for estimating the final energy consumption and performance degradation associated with the voltage and frequency reduction. However, DVFS has important drawbacks:

- Long transition times between power modes.
- Long exploration and use windows (in order to amortize DVFS overheads), making it difficult to adapt precisely to the program behavior.
- When activated, DVFS affects all instructions regardless of their usefulness in the program. Therefore, it cannot exploit situations such as instruction slack, instruction criticality, or low confidence on the predicted path.
- As process technology shrinks, reducing voltage (V_{DD} and V_T) to lower dynamic power becomes impractical since leakage exponentially depends on V_T (see section

3.1.1) which will turn DVFS into just DFS (Dynamic Frequency Scaling) for deep submicron designs. However, DFS is not so energy-efficient since it seriously hurts performance.

This chapter studies the use of fine-grain microarchitectural power-saving techniques to accurately match a predefined power budget. The studied mechanisms will capture and store information about the processor energy consumption, either at a cycle level (Power-Token Throttling) or at a basic block level (Basic Block Level Manager). We can decide whether the next instruction/basic block can be executed by checking both the past energy consumption and how far the processor is from the power budget. In this chapter we also propose an efficient two-level approach that firstly applies DVFS as a coarse-grain approach to lower the average power and, secondly, uses microarchitectural techniques to remove the numerous power spikes. One of the benefits of using microarchitectural techniques is that they can be applied at a cycle level, only in those cycles where the processor exceeds the power budget. The peculiarities of each microarchitectural technique were detailed in section 3.1 but, in general terms, these fine-grain techniques:

1. Locate non-critical instructions in cycles exceeding the power budget and delay them to cycles under the budget. Note that non-critical instructions can become critical if they are delayed for a long time.

2. Previous studies have shown that 30% of the overall processor power comes from wrong path instructions [4, 66]. Therefore, we can reduce the number of low-confident speculative instructions in the pipeline when the processor is exceeding the power budget by using a confidence estimator.

3. We can throttle the pipeline at different stages when the two previous policies are not enough to put the processor under the required power budget.

In addition we also propose and evaluate the use of two additional mechanisms that help microarchitectural techniques to match the predefined power budget by analyzing the power curve trend: *preventive switch-off* and *predictive switch-on*. *Preventive switch-off* calculates the difference in power between the last N cycles and predicts if the processor will exceed the power budget based in the previous cycle power plus the calculated difference; whereas *predictive switch-on* does the opposite: calculates differences to disable techniques before getting under the power budget, hoping that the decreasing power trend will be enough to lead the processor under the budget. Experimental results show that the use of power-saving microarchitectural techniques is more energy efficient as well as more accurate (i.e., less area over the power budget, see Section 2.3 for details) than DVFS for driving the processor under the required power budget.
The rest of the chapter is organized as follows. Section 4.2 shows a first analysis of the individual techniques and motivates the need for a hybrid approach to match the power budget. Section 4.3 describes our simulation methodology and reports the main experimental results. Finally, Section 4.4 shows our concluding remarks.

4.2. Power-Saving Microarchitectural Techniques

When reducing power dissipation under a power budget there is something that should be kept in mind: there is going to be performance degradation. This is justified by the fact that there is a strong need to match the imposed power constraint. Our main goal is to reduce power dissipation to match an imposed power budget in an energy-efficient way. To achieve that goal we first need to study how the different microarchitectural techniques behave independently, in order to design an adaptive mechanism that takes advantage of each of their peculiarities. As evaluation metrics we will measure both the fraction of cycles exceeding the power budget over the total execution cycles as well as the induced performance degradation for the whole SPECint2000 (see Section 4.3.1 for details about the processor configuration).

4.2.1. Reactive Techniques

Reactive techniques check the processor's power at a cycle-level. If the current power exceeds the required budget, the processor applies a certain microarchitectural technique to reduce its power consumption. The major concern about reactive techniques is that they must be applied during enough cycles in order to achieve its low-power effect. If in a given cycle the processor goes over the power budget, we do not know a priori how long this situation will last. Furthermore, reactive techniques are not able to remove all the cycles the processor spends over the power budget since we activate them once the processor power dissipation is over the budget, unless we use a predictive approach. Examples of dynamic power reduction reactive techniques are: DVFS, Pipeline Throttling and Instruction Reordering and were introduced in section 3.1.

4.2.2. Predictive Techniques

As cited previously, reactive techniques are not good enough to accurately match a predefined power budget, basically because they rely on local information about power dissipation, and have no knowledge about past or future trends in power dissipation. Predictive techniques, on the other hand, will capture and store information about the processor power dissipation, either at a cycle level (Power-Token Throttling) or at a basic block level (Basic Block Level Manager), in order to decide whether the next instruction/basic block can be executed based on its previous power. For predictive techniques to work we need a way to measure power dissipation at a cycle level.

4.2.2.1. Power-Tokens

Until recently (Intel's Sandy Bridge power MSRs) there was no way to accurately measure energy consumption in real time. Energy measurements were just approximations based on performance counters done over periods of thousands of cycles. When using microarchitectural techniques that work at a cycle level we need some way to estimate

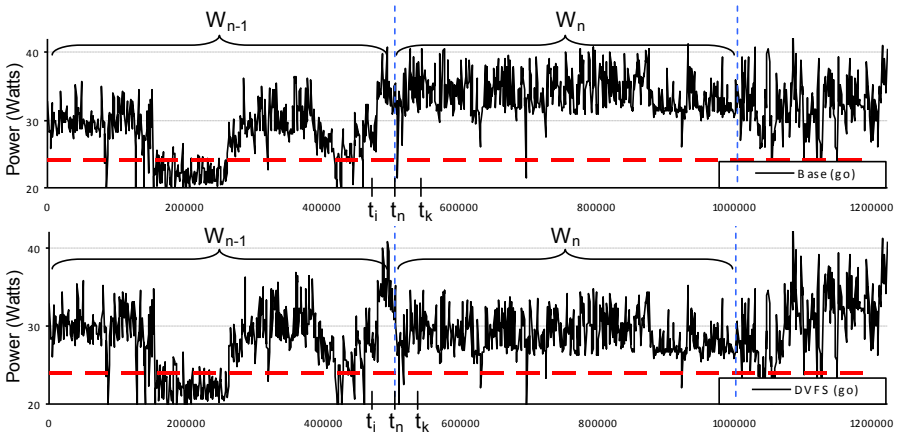


Figure 4.1: Base vs DVFS exploration window power dissipation for the "go" benchmark.

power at a cycle level. Our proposal uses a Power-Token approach to dynamically estimate power dissipation. Power Tokens are described in detail in section 2.2. Essentially, Power Tokens estimate the processor's dynamic power in a given cycle based on instruction power dissipation information (measured as token units).

4.2.2.2. Power-Token Throttling (PTT)

This technique follows a simple premise: if there is power left to burn, let the instruction enter the pipeline, otherwise, stall fetch. Basically, it estimates the dynamic power of the instructions inside the pipeline in a given cycle by means of accounting for the power-tokens they consume. When the total power of all the instructions inside the processor exceeds the power budget, the fetch stage is stalled until instructions commit and leave the pipeline, releasing their power-tokens. PTT allows the introduction of new instructions into the pipeline when the current total power goes under the power budget. This is the most accurate technique but at the same time the least energy-efficient since it incurs in serious performance loss. We also evaluate a modified version of the Power-Token Throttling approach that uses a Critical Path Predictor (labelled as PTT-CP in Section 4.3) to detect critical instructions. In this way, even if we are over the power budget, we allow these instructions to continue their execution. This version is less aggressive than the original PTT.

First of all, in order to gain some insight on the power dissipation behavior DVFS, Figure 4.1 shows the average power dissipation for the "go" benchmark of the base processor compared to the same processor using DVFS. The dashed horizontal line represents the required power budget (set to 24W, which corresponds to a 50% power budget over the peak power of 48W, see Section 4.3.2 for details). The used DVFS implementation (as in [40]) calculates the average power dissipation over an exploration window of 500Kcycles, and modifies the voltage and frequency accordingly (from a set of pairs voltage/frequency modes) to match the desired power budget. For exploration window W_{n-1} DVFS calculates its average power dissipation and changes to the power mode that is closest to

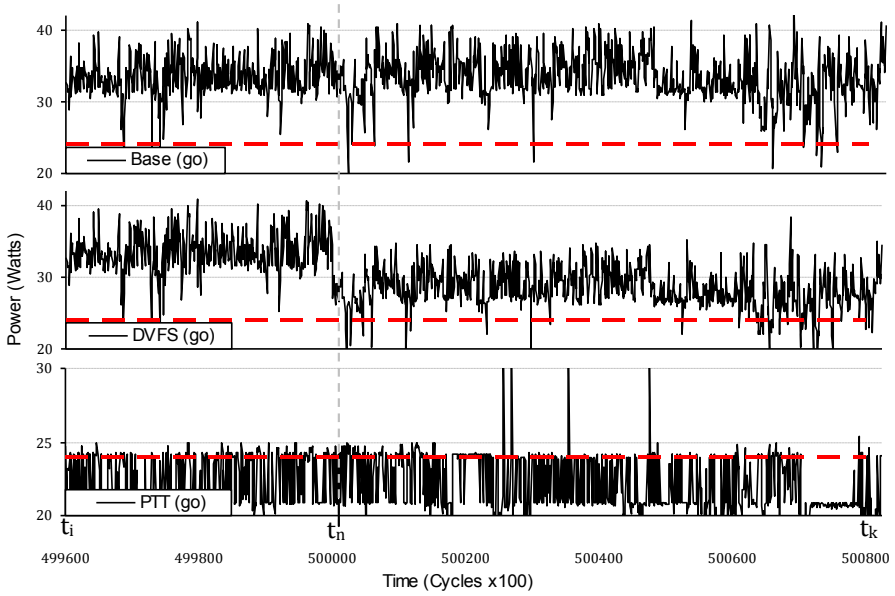


Figure 4.2: Detailed per-cycle power dissipation for the "go" benchmark.

the power budget. This power mode will be used during exploration window W_n. In this example, during W_n the average power dissipation of the base case increases (Figure 4.1-top), which leaves the average power dissipation of the DVFS processor still over the power budget (even though a lower power mode was selected; Figure 4.1-bottom). This entails that the amount of power over the budget is indeed lower when using DVFS but its accuracy for matching the power budget is questionable.

Figure 4.2 is a more detailed representation of the power behavior for the "go" benchmark (this figure shows a zoom-in from time t_i to t_k in Figure 4.1). For the sake of visibility we only plot now the power information for a reduced time interval of 120Kcycles. As said before, for the DVFS case (Figure 4.2-middle), after t_n the selected power mode is the one closest to the average power dissipation of the previous window W_{n-1} (as seen in Figure 4.1-bottom). However, because during window W_{n-1} the average power was higher than the actual power after point t_n, the DVFS selected power mode is still not able drive the processor under the power budget.

On the other hand, the Power-Token Throttling approach (Figure 4.2-bottom), accurately follows the power budget at a cycle level, as it knows in advance how much power (in tokens) the next instruction consumes. If we cannot afford to execute it, we wait until a committed instruction leaves the pipeline and there is enough power left to burn in the new instruction. Spikes in the PTT plot are due to branches that are left to enter the pipeline in order to discover eventual mispredictions as soon as possible (otherwise, performance is seriously hurt).

4.2.2.3. Basic Block Level Manager (BBLM)

This second technique keeps track of historical information about power-tokens consumed by a basic block[1] as well as the power left to exceed the power budget in order to decide what technique should be applied to reduce power dissipation. The available techniques are (from less to more aggressive): a) none, b) critical path, c) confidence estimation throttling, or d) decode-commit ratio throttling. Every time the processor decodes a branch it estimates the energy consumed by the previous basic block, measured as the addition of the power-tokens of every instruction that belongs to that basic block. This power is stored in the branch predictor entry of the branch that points to the start of that basic block (not the one that we have just decoded). The overhead of this mechanism consists in 9 additional bits per branch predictor entry plus a register and a 16-bit adder to store the current basic block power which corresponds to a negligible 0.3% power increase in the total processor power, accounted for in our results. Therefore, every time a branch is predicted we also obtain information about the subsequent basic block power, length, etc, and we estimate how far from the power budget we will get if we completely execute that basic block. Depending on how far the estimated power is from the budget, we select a different power saving technique. In addition, when updating the branch predictor's saturating counter, we also update the power-tokens consumed by the subsequent basic block (that corresponds to the next predicted path). Please note that, although it is very difficult that a basic block will always consume the same exact amount of energy, this is not really relevant for our mechanisms, as long as the selected technique is the same for a specific basic block (which is actually the common case). We define two threshold values, X and Y for BBLM to decide what technique should be applied. Therefore, for a fraction of power over the budget lower than X, BBLM will select the first technique (i.e., critical path, which is the least aggressive); from X to Y, BBLM will select the second technique (i.e., confidence estimation throttling); and for a power over the budget greater than Y, BBLM will select the third technique (i.e., decode-commit ratio throttling, which is the most aggressive). Similarly, techniques are disabled progressively, in reverse order, once we get under the power budget. We performed an experimental study to determine the best values for these thresholds (best power budget matching with minimal performance degradation), and discovered that for the current processor configuration and used benchmarks the best thresholds are X=15 and Y=65.

4.2.2.4. Two-level Approach

Cycle-level microarchitectural techniques have some disadvantages compared with DVFS. In DVFS only a third part of the power reduction comes from performance degradation $P \approx V_{DD}^2 * f$ whereas for microarchitectural techniques every change has a direct impact in performance. All instructions from the correct path of execution consume a constant amount of energy, so only removing instructions from wrong path of execution

[1] Assumed in this Thesis as a stream of instructions between two branches.

or reordering instructions will allow the processor to go towards a power budget with minimal performance impact.

When the current power dissipation is far from the budget microarchitectural techniques do not work efficiently, as we will see in Section 4.3, since there are few chances to reduce power dissipation without degrading performance. On the other hand, DVFS is extremely inaccurate when there are power spikes, because the influence of power spikes on the average power dissipation from the sampling window is negligible. If we only use microarchitectural techniques we may not get close to the power budget, or the performance degradation will be high. Moreover, it has been proved that benchmarks exhibit different program phases with different average power dissipation. If we are able to detect these phases we could use only microarchitectural techniques in phases close to the power budget and a coarse-grain approach (DVFS) in phases far from the power budget. Therefore, our proposal consists of a two-level approach: first we apply DVFS to take coarse-grain decisions about power dissipation, and secondly we apply microarchitectural techniques for fine-grain decisions (mainly for removing the numerous power spikes). This two-level approach increases the DVFS accuracy for matching a power budget while at the same time does not need all the DVFS power modes to reach a power budget. By using this two-level approach only the least aggressive DVFS power modes are enough to accurately match the predefined power budget. It is important to note that, as discussed in section 3.1.1, when the process technology goes below 65nm the reduction on V_T will be limited by both reliability and leakage power. At this point, it may not be a matter of not using the extreme power saving modes, but implementing those power modes will be infeasible.

4.2.2.5. Preventive Switch-Off and Predictive Switch-On

Reactive techniques rely on current information to decide whether or not to apply some microarchitectural techniques to reduce power dissipation. The base decision mechanism looks at the current cycle power dissipation to detect if the processor is over the power budget, and then applies whatever mechanism the processor implements to reduce dynamic power. One possible way to improve this could be adding some intelligence to the process. If the power is continuously increasing from cycle to cycle we may predict that the next cycle will continue rising. If that is true, we can enable the power saving mechanisms preventing the processor from exceeding the power budget. The preventive switch-off approach calculates and stores the differences between per-cycle power dissipation and predicts if power will be over the budget the next cycle by adding the current cycle power plus the difference in power between the last two cycles (Figure 4.3- points A and B).

Moreover, this mechanism is also useful when we work with predictive techniques (as PTT and BBLM). In BBLM we have an idea of how far from the power budget the current basic block is, but have no idea on how power is distributed among cycles. With preventive switch-off we can add some additional information about per-cycle power trends and

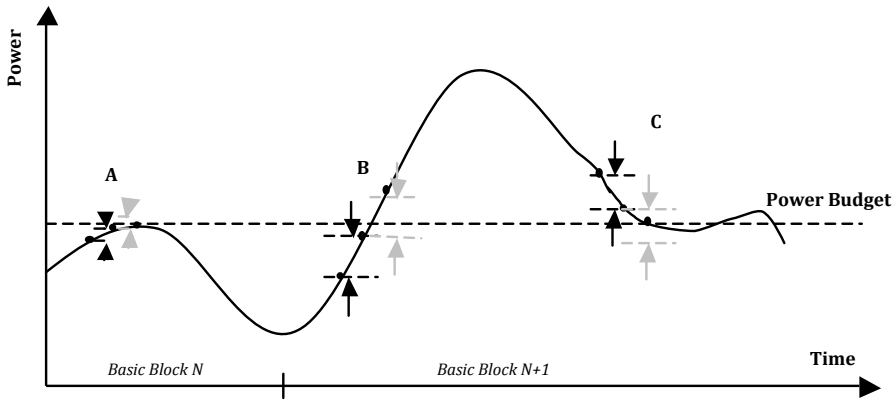


Figure 4.3: Overview of the preventive switch-off and predictive switch-on mechanisms.

help the selected microarchitectural mechanism in its task. However, preventive switch-off may lead to false positives if we prematurely enable microarchitectural techniques. For example in Figure 4.3-A the current cycle power plus the difference in power between the last two cycles will lead the next cycle over the power budget and thus we should enable power-saving techniques to prevent that. In this specific case this decision is wrong, as power will start to descend in the next few cycles. This may not happen with predictive techniques as they have additional information about future power behavior of the basic block. If BBLM is enabled in Figure 4.3-A, the mechanism will choose not to do anything prematurely as the average power of the basic block is under the power budget.

Analogously, we have the predictive switch-on approach. When microarchitectural mechanisms are enabled power begins to descend until the processor gets under the power budget, and then power-saving mechanisms are disabled. Usually, microarchitectural mechanisms have some hysteresis in which power continues going down. If we disable the power-saving technique in advance, before getting under the power budget, the hysteresis period should be enough to lead the processor under the power budget. The predictive switch-on mechanism works like preventive switch-off but when power is decreasing. The mechanism calculates the power differences between cycles and estimates if the current power minus the delta power is under the budget in order to disable microarchitectural techniques (as seen in Figure 4.3-C). Note that false positives in this mechanism are less harmful than for preventive switch-off, because only "accuracy" is reduced for matching the power budget, but not performance.

4.3. Experimental Results

4.3.1. Simulation Methodology

To evaluate the energy-efficiency of both DVFS and microarchitectural techniques we have used the SPECint2000 benchmark suite (refer to section 2.7 for further details). All benchmarks were compiled with maximum optimizations (-O4 -fast) by the Compaq

Alpha compiler and they were run using a modified version of Simplescalar simulator using power models provided by Wattch/HotLeakage [100] that includes the dynamic power models for the evaluated microarchitectural approaches as well as their associated power overhead. All benchmarks were run to completion using the reduced input data set (test). Table 4.1 shows the configuration of the simulated processor.

Table 4.1: Core configuration.

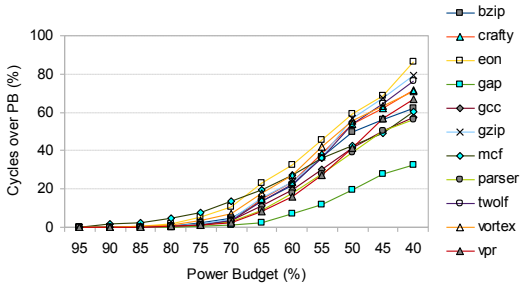
Processor Core	
Process Technology:	32 nanometers
Frequency:	3000 Mhz
V_{DD}:	0.9 V
Instruction Window	128 RUU + 64 IW
Load Store Queue	64 Entries
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	6 Int Alu; 2 Int Mult
	4 FP Alu; 4 FP Mult
Branch Predictor:	16bit Gshare
Memory Hierarchy	
Coherence Prot.:	MOESI
Memory Latency:	300
L1 I-cache:	64KB, 2-way, 1 cycle lat.
L1 D-cache:	64KB, 2-way, 1 cycle lat.
L2 cache:	1MB/core, 4-way, unified
	12 cycle latency
TLB:	256 entries
Network Parameters	
Topology:	2D mesh
Link Latency:	4 cycles
Flit size:	4 bytes
Link Bandwidth:	1 flit / cycle

4.3.2. A Power Budget of What? (100% Usage $\neq$ 100% Power)

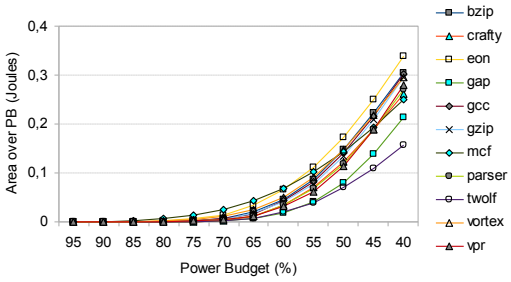
When we think about the maximum power dissipation of a processor we think about the processor using all of its resources at once (i.e., the peak dissipation of a processor), and that barely happens in common applications or even scientific applications. Our base processor has a peak power dissipation of 75 Watts as provided by the Wattch power simulator. When a circuit-level technique such as clock gating is enabled, the average power dissipation for the SPECint2000 drops to 25 Watts with a peak power dissipation of 48 Watts for our simulated processor and benchmark suite. We will use 48 Watts of peak power[2] as our reference power (i.e., 100% power budget), which corresponds to the power dissipation of the processor before applying any power-saving technique.

Next sections show the simulation results for the SPECint2000 benchmark suite for: a) fraction of cycles over the power budget (Cycles over PB); b) total power exceeded over the budget (AoPB - see section 2.3); and c) normalized energy. We use the AoPB metric instead of the average power since the standard deviation of the per-cycle power dissipation is quite high. The processor has periods of high power dissipation hidden by periods of low power dissipation (branch mispredictions, cache misses, etc.), therefore, the average power is not a good metric for what is really happening inside the processor.

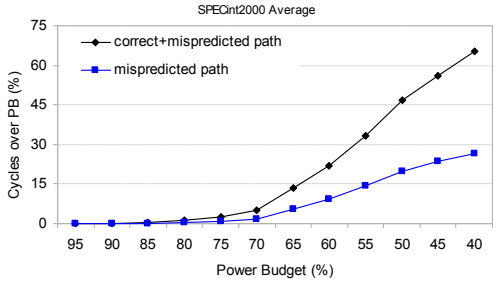
[2]Using the Wattch's clock-gating style cc3, which scales power linearly with unit usage. Inactive units still dissipate 10% of its maximum power.



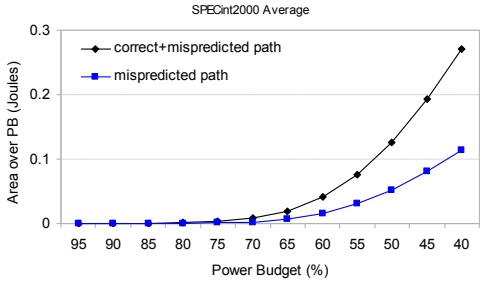
(a) Per Benchmark Cycle over PB.



(b) Per Benchmark Area over PB.



(c) Average Cycle over PB.



(d) Average Area over PB.

Figure 4.4: Area and Cycles over PB distributions

4.3.3. Cycles Over PB and Area Distribution

Figure 4.4 shows both the fraction of cycles over power budget and the area over the power budget for different budgets and benchmarks before applying any power saving technique. Evaluated PBs range from 95% of the original peak power to 40%. As we can see in Figures 4.4-top, both the amount of cycles and the area the processor spends over the power budget is almost negligible for high budgets (95%-70%), due to the effects of clock gating on power dissipation, but being highly noticeable for power budgets under 65%. As explained before, DVFS will be unable to find power spikes for low power budgets, as it works with the average power dissipation of its long exploration windows. On the other hand, microarchitectural techniques will detect these spikes and remove them whenever possible. Figures 4.4-bottom show the cycles and area over the power budget distinguishing between power/cycles from correct path of execution as well as mispredicted paths. We can see how almost half of the power/cycles over the power budget come from instructions from the wrong path of execution, those are the ones we are most interested in, as microarchitectural techniques heavily rely on them to reduce power towards the budget with low performance degradation.

4.3.4. Efficiency on Matching a Power Budget

The main objective of this chapter is the proposal of accurate energy-efficient techniques for matching a power budget, and thus this subsection evaluates the energy-efficiency of the proposed microarchitectural techniques as well as their accuracy for matching an

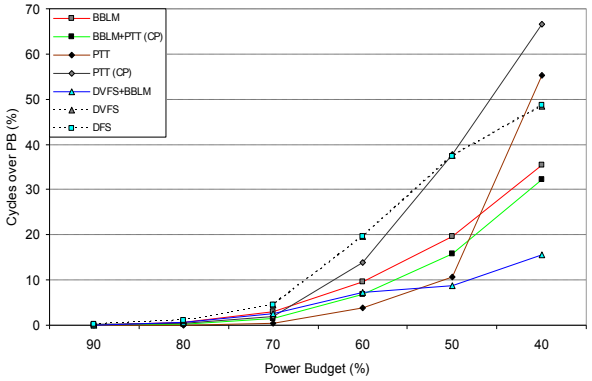


Figure 4.5: Relative cycles over PB for different power budgets.

imposed power budget. We report results for the following techniques: DVFS, PTT, BBLM (with and without using Critical Path information), and the two-level approach (DVFS+BBLM). Our DVFS approach initially calculates the average power dissipated during an exploration window. If the mechanism finds out that the average power dissipation is over the budget, DVFS changes to a pair of voltage and frequency values from a set of predefined working modes in order to reduce the average power dissipation. DVFS uses a exploration window of 500Kcycles with a very fast transition time between modes set to 50 mV/ns (as in [51]), so it takes only 3 cycles to switch from one mode to another[3]. This implementation is similar to the one proposed in [40] and discussed in section 3.1.1. The evaluated working modes are the following:

- DVFS: Uses five power modes (100% V_{DD}, 100% f), (95% V_{DD}, 95% f), (90% V_{DD}, 90% f), (90% V_{DD}, 75% f), and (90% V_{DD}, 65% f).
- DVFS+BBLM: The power modes are reduced to (100% V_{DD}, 100% f), (95% V_{DD}, 95% f), and (90% V_{DD}, 90% f).
- DFS: Only scales frequency as needed (V_{DD} remains unchanged).

The studied BBLM uses the configuration parameters and techniques proposed in Section 4.2.2.3. As explained before, the selected thresholds are X=15 and Y=65. Critical path (CP) is used as the first technique, JRS-throttling as the second technique, and DCR-throttling as the third one.

Figure 4.7 shows the normalized energy consumption for the different techniques and power budgets. As we can see, the two-level approach (DVFS+BBLM) is the most energy-efficient technique for all the studied power budgets, and especially for very restrictive power budgets. The rest of microarchitectural techniques (except PTT) as well as DFS show a low energy degradation (between 4% and 10%). BBLM shows similar energy-efficiency than DFS up to a power budget of 50% while reducing four times more

[3]For HotLeakage V_{DD} at 65nm is 1V, so each 5% reduction in voltage translates into 50mV. That means it will take 1ns to switch between modes. As the processor runs at 3Ghz, it will take 3 cycles to change between power modes.

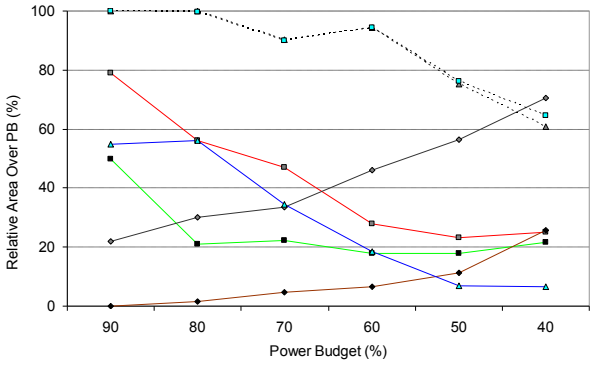


Figure 4.6: Relative area over PB for different power budgets.

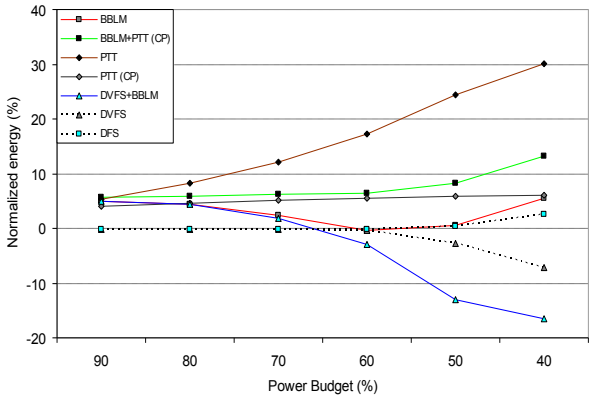


Figure 4.7: Normalized energy consumption.

the AoPB. Moreover, as we are working under an imposed power budget, Figures 4.5 and 4.6 show how accurate each technique is when trying to match the power constraint, as we plot both the relative cycles and area over the power budget. It can be observed that all microarchitectural techniques are far more accurate than DVFS when adapting to the imposed power budget. For power budgets between 90% and 70%, BBLM+PTT (CP) is the best approach: 25% of area over PB is left after applying this technique with a total energy increase of only 6% (Figure 4.7). PTT shows an increasing energy trend because it throttles the pipeline so much that makes the overall energy increase to offset the savings provided by the throttling mechanisms.

When we move to more restrictive power budgets (<60%), our two-level (DVFS+BBLM) approach is 4% more energy-efficient than DVFS while the accuracy of the former is three times better (25% of relative AoPB is left by the two-level approach whereas DVFS barely reduces the area to a 90% of the original). For an extreme power budget of 40%, our two-level approach gets even better: only 10% of relative AoPB is left, in contrast with the 60% reported by DVFS. In addition, the two-level approach for this extreme power budget is 11% more energy-efficient than DVFS and 20% than DFS. In general terms, DVFS and DFS are coarse-grain approaches unable to remove the numerous power spikes making them far less accurate than microarchitectural techniques.

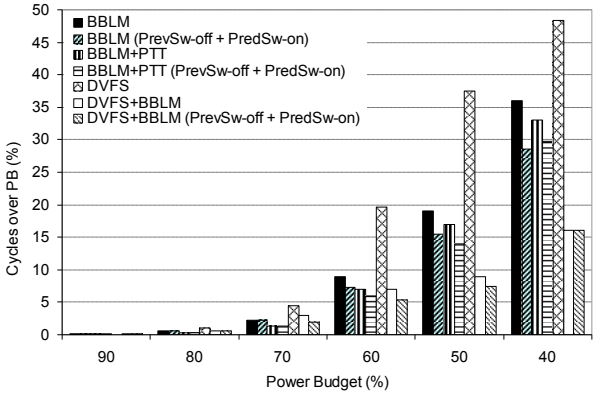


Figure 4.8: Relative cycles over PB for different power budgets.

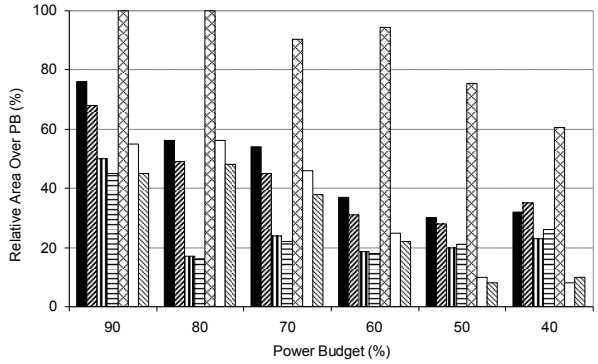


Figure 4.9: Relative area over PB for different power budgets..

4.3.5. Preventive Switch-Off and Predictive Switch-on

In this section we show some experimental results for both the preventive switch-off and predictive switch-on mechanisms when simultaneously applied with both BBLM and the proposed two-level approach (DVFS+BBLM).

Figures 4.8 and 4.9 show a comparison between the original techniques and the ones that include the preventive switch-off and predictive switch-on mechanisms in terms of cycles and area over the power budget. For power budgets between 90% and 60% there is a reduction in both area and cycles over the power budget of 3-4% for all the studied mechanisms. For power budgets of 50% and 40% there is something curious happening in the processor. Although there are less cycles over the power budget when preventive switch-off is applied, the total AoPB increases. This is due to the fact that preventive switch-off and predictive switch-on approaches give more false positives with restrictive power budgets. In addition, as most of the cycles are over the power budget, preventive switch-off has few chances to discover rising power trends.

When preventive switch-off and predictive switch-on work together with BBLM and the two-level approach, energy numbers get better (1-2%) as shown in Figure 4.10, especially for power budgets between 90% and 60%. For a power budget of 50% the difference is minimal, and for 40% techniques are less energy efficient, again, due to false positives.

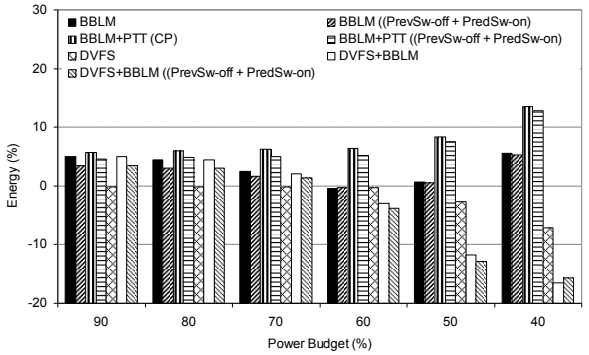


Figure 4.10: Normalized energy for all the evaluated approaches along with preventive switch-off and predictive switch-on.

Summarizing, preventive switch-off and predictive switch-on techniques usually help with both accuracy and energy for power budgets between 90% and 50%, but become less effective with restrictive power budgets.

4.3.6. Sensitivity Study

In this section we explore different processor configurations for some of the proposed techniques (BBLM, DVFS and the two-level approach), varying pipeline length and the decode/issue width. With this study we want to ensure that, even if the processor complexity is either reduced or increased, the proposed techniques behave regularly. Please note that the microarchitectural techniques have not been optimized for the new processor configurations. Figure 4.11 shows the cycles over the power budget (CoPB), area over the power budget (AoPB) and energy numbers for 5 different processor configurations. The three figures at the left show results for different decoding/issue width sizes using a 22-stage pipeline. For decoding/issue widths of 2 and 4 with this new pipeline, results in area and energy are very close to the 14-stage pipeline. On the other hand, a more complex 8-issue width processor obtains relatively bad results in energy numbers. Still, the microarchitectural techniques and the two-level approach are more accurate than DVFS. With a 14-stage pipeline and issue width sizes of 2 and 8, energy numbers behave reasonably well, but the area numbers are a little bit higher because of the lack of optimization parameters for the different techniques.

4.4. Conclusions

Current general purpose microprocessors can be used in several kinds of gadgets that usually have different power requirements. In some scenarios being able to define a maximum power budget for the processor can help the reuse of previous designs in new devices. This can be especially useful if our only options are either to switch-off the device or to reduce the power dissipation to match the power constraint. Note that, in many cases, the shut-off option is not even viable (e.g., for critical systems).

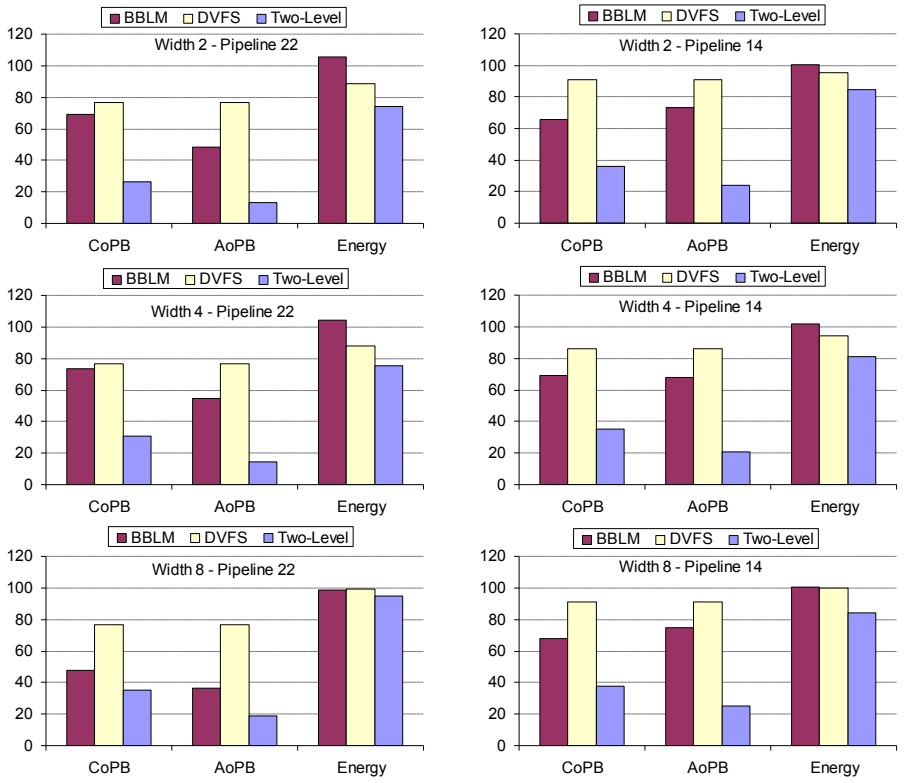


Figure 4.11: Sensitivity analysis.

We cannot design the thermal envelopment of a processor for the worst case scenario, because production price highly increases and the processor barely ever reaches its peak temperature. However, we can set a power budget to the processor, limiting its power and temperature.

When microarchitectural techniques are used we obtain a more precise power budget matching while maximizing the processor's energy efficiency. We propose two adaptive techniques: Power Token Throttling (PTT) and Basic Block Level Manager (BBLM). PTT is a token-based approach that keeps track of the current power being dissipated by the processor, measured as tokens, and stalls fetch if the next instruction to be executed will make the processor go over the power budget. This is a very aggressive mechanism that obtains an accurate power budget matching but has a huge performance degradation. On the other hand, BBLM uses basic block energy consumption history (translated into power-tokens) in order to determine the best power saving technique for the current and near-future processor power dissipation. BBLM optimizes the use of microarchitectural techniques to minimize performance impact while removing most of the power spikes. However, these mechanisms by themselves are not enough to match restrictive power budgets because of their high performance degradation.

We have then proposed a two-level approach that combines both microarchitectural techniques and DVFS to take advantage of their best qualities. DVFS acts as a coarse-grain technique to lower the average power dissipation while microarchitectural techniques re-

move all the power spikes efficiently. The two-level approach (BBLM+DVFS) is able to beat DVFS in both energy efficiency (up to 11% more energy efficient) and area exceeded over the power budget (up to 6 times less area).

The preventive switch-off and predictive switch-on techniques are orthogonal to both DVFS and microarchitectural techniques and help them to reduce energy consumption and increase accuracy for power budgets between 90% and 60%, becoming less effective for restrictive power budgets due to false positives.

In the next chapter, we are moving all these mechanisms to a CMP scenario where synchronization mechanisms can destroy all the power savings achieved by local power saving mechanisms and global CMP mechanisms are needed to match the predefined power budget.

Chapter 5

Multi-Core Power Budget Matching

Summary: In this chapter we analyze some common techniques to match a predefined power budget ported from the single-core scenario into a CMP scenario. We discovered that legacy techniques work properly for thread-independent and multi-programmed workloads, but not for parallel workloads. In order to solve this problem we propose *Power Token Balancing* (PTB) aimed at accurately matching an external power constraint by balancing the power dissipated among the different cores using a power token-based approach while optimizing the energy efficiency. PTB uses power (seen as tokens) from cores under the power budget to match power constraints on cores over the power budget. Experimental results show that PTB matches more accurately a predefined power budget than DVFS with minimal energy increase.

We also introduce a mechanism named "Nitro", inspired by the idea of Formula One's KERS. Nitro will overclock the core that enters a critical section (delimited by locks) in order to free the lock as soon as possible and let another core enter the critical section. The mechanism ensures that the overclocking can be safely done as long as there is power left to burn from idle or spinning cores. We conclude the work with an analysis of the thermal effects of PTB in different CMP configurations using realistic power numbers and heatsink/fan configurations. Results show how PTB not only balances temperature between the different cores but also allows a reduction of 28-30% of both average and peak temperatures.

5.1. Introduction

Chip multiprocessors (CMPs) are the new standard in processor design for a wide range of devices, from small mobile devices to computation clusters. These architectures exhibit some peculiarities in terms of power and performance compared to a single-core processor. For instance, if we focus on power, when the number of processing cores is doubled the power dissipation is also approximately doubled. Fortunately, technology scaling trends help designers to keep the dynamic power increase partially under control on each new generation. However, the complexity of the interconnection network and

caches increases when more cores are incorporated into the die resulting in higher power dissipation. When a CMP runs a parallel multithreaded program where threads have dependencies (i.e., synchronization), if a traditional power saving mechanism such as DVFS is independently applied to a single core it can affect the rest of the threads in the next synchronization point. This may slow down the whole program execution and increase the overall energy consumption. If we want to avoid this situation global information is required to reduce power in the threads that are not in the critical path of execution (e.g., threads that arrive earlier to the synchronization points) [40][20][57][55][11].

On the other hand, there are also thermal problems. The continuous increase in the number of cores on a CMP forces designers to develop more and more complex floorplans. At some point hotspots may appear in inner cores, decreasing the reliability of the processor or even causing major damage to the die. A straightforward way to reduce temperature is to set a *power budget* to the processor [40][26]. This processor's power budget can be used either to satisfy external power constraints (i.e., power cuts or shared power supply), to increase the number of cores in a CMP maintaining the same TDP (*Thermal Design Power*), or to even reuse an existent processor design with a cheaper thermal package.

As cited in the previous chapters, *Dynamic Voltage and Frequency Scaling* (DVFS) [32][62][87] is a well known mechanism to make the processor's power converge to a given power budget. DVFS is based on fact that dynamic power depends on both voltage (quadratically) and frequency (linearly), so, if we lower any of these terms, we obtain a power reduction. However, DVFS exhibits some important drawbacks: a) long transition times between power modes [87]; b) long exploration windows in order to compensate DVFS overheads; and c) when activated DVFS affects all instructions within a thread/-core regardless of their usefulness to the forward-progress of a program.

In the CMP field we can find many specific proposals to match a predefined power budget such as [40][70][82], but these proposals are only suitable for CMPs running multiple single-threaded (or multi-programmed) applications, and have not been tested with parallel workloads. Moreover, they do not perform any accuracy analysis on the budget requirements. There are other works aimed at reducing the power wasted when cores wait at synchronization points, either putting cores to sleep [57] or trying to make all cores reach the synchronization point simultaneously (e.g., *meeting points* [20] or *thrifty barriers* [55]). However, these mechanisms are not suitable for matching a power budget on their own, the main goal of this Thesis. In order to overcome those limitations we proposed the use of fine-grain microarchitectural power-saving techniques to accurately match a predefined power budget in a single-core scenario in Chapter 4. However, when applied in a CMP scenario, these techniques have a great impact on power and performance due to synchronization points.

To address the shortcomings of previous proposals we propose *Power Token Balancing* (PTB), a mechanism that balances the CMP power dissipation by efficiently distributing the available power among the cores. When we set a global power budget to a CMP,

local power budgets are applied to all running cores. Without any global mechanism the power would be just equally split between the cores as we will show later. PTB globally manages power dissipation so cores that are under their local power budget give away their remaining allotment of power (up to the local budget) to cores over the budget so they can continue execution without performance degradation while ensuring that the global power budget is not exceeded. PTB is based on the power imbalance that exists between cores due to cache misses, ROB stalls, pipeline stalls, etc. In addition, PTB transparently benefits from thread's busy-waiting synchronization in a very "lightweight" way. When a core is waiting in a barrier it naturally reduces its power dissipation. PTB allows its spare power tokens to be given to other cores doing useful work (i.e., critical threads). The same applies to locks: a core that enters a critical section receives extra power tokens from other cores waiting on spinlocks. Thanks to the extra power tokens its local power budget is less restrictive and the core can leave the critical section faster. Furthermore, when several cores enter a spinning state in a lock-delimited critical section they need to wait until the core currently in the critical section leaves it. If we are working in a power-constrained scenario using PTB, the core in the critical section will receive extra power from all spinning cores, and most likely will run at full speed because the additional power will prevent the core to restrain itself to match the power budget. However, some times, even at full speed, we still have power left to burn. To reuse this remaining power we introduce Nitro. Inspired by the same idea as the new Formula One's KERS system (i.e., save power when breaking and use it to accelerate), Nitro reuses power to overclock the core that enters the critical section. During this time the core will run faster than usual, leaving the critical section earlier. This mechanism allows to obtain speedups with minimal overclocking run-time while ensuring that overclocking can be safely done since it is achieved by reusing power from cores under their local power budget. This approach can be used alone in any CMP but, as our major focus is a power-constrained scenario, we use Nitro as part of our global power balancing mechanism for matching a power budget (PTB).

At this point we want to summarize the main contributions of this chapter:

- Introduction of the *Power Token Balancing* (PTB) approach.
  - Its main goal is to make the per-cycle power dissipation go below a certain power budget while maximizing accuracy and not to reduce overall energy.
  - It is designed (but not limited) to work in a CMP scenario running parallel workloads.
  - It can identify critical threads faster than [20][57][55], (the critical thread can change during execution) since it relies on cycle-level information increasing its adaptability to the application behavior.
  - It is a fine-grain approach unlike [20][57][55][82], because it relies on actual real-time information and not time/power estimations and it can identify crit-

ical threads faster than other approaches due to the use of cycle-level information.

- It is able to reduce both the average power dissipation and the average/-peak chip temperature due to its precision on matching the predefined power budget.

- Accuracy and performance analysis of previous proposals which are compared with PTB.
- Temperature analysis of different core and floorplan configurations is performed for the PTB mechanism.
- Introduction of a dynamic overclocking mechanism, Nitro, which selectively overclocks cores that enter a critical section (delimited by locks) as long as the rest of the cores do not exceed the predefined power budget.

The rest of the chapter is organized as follows. Section 5.2 provides some background on power-saving techniques for both single-core processors and CMPs. Section 5.3 describes our simulation methodology and shows a first analysis on the individual techniques and motivates the need for CMP-specific approaches to match the power budget. Section 5.4 reports the main experimental results. Finally, Section 5.5 shows our concluding remarks.

5.2. Background and Related Work

5.2.1. CMP-specific Power Control Mechanisms

CMP architectures exhibit some peculiarities when running parallel workloads, especially in terms of performance and power. In such workloads threads must periodically synchronize (e.g., for communication purposes) and any delay introduced in one of the threads may end up delaying the whole application.

5.2.1.1. Saving Power from Spinning

One of the main sources of power dissipation in CMPs running parallel workloads is "spinning" or "busy waiting". When a core is trying to acquire a lock or waiting in a barrier it enters in a spinning state that may become an important source of useless power dissipation. In order to detect spinning, initial approaches used source code or binary instrumentation but that requires recompilation and might be infeasible for certain situations. Li *et al.* [57] proposed a real-time hardware mechanism to detect processors in spinning state. Their mechanism checks the machine's state between instructions that cause a *backward control transfer* (BCT), usually a branch or a jump instruction. If the machine's state remains the same between several BCTs, the processor has entered a spinning state. They also propose scaling frequency for processors in a spinning state

assuming that they can wake up a processor. However, this mechanism does not provide precise power management and cannot be applied outside locks/barriers.

5.2.1.2. Dynamic Voltage and Frequency Scaling in CMPs

In 2004, Li *et al.* proposed *thrifty barriers* [55]. This DVFS-based mechanism reduces power dissipation in CMPs by estimating the per-core time interval between synchronization points and disabling or using DVFS in cores that get to the synchronization point. They approximate the wakeup time by the time the slowest thread takes to get to the synchronization point. If the sleep/wakeup takes more time than they can save then the technique is not used. Later on, in 2006, Isci *et al.* [40] proposed a chip-level dynamic power management for CMPs just focusing on single-threaded programs while Sartori *et al.* [82] extended this work to reduce peak power in a distributed way. These mechanisms selectively change between several DVFS power modes for the different cores maximizing throughput under certain power constraints. Unfortunately, as these proposals rely on the use of performance counters and/or time estimation, they only work properly for multi-programmed or single-threaded applications simultaneously running on the different cores of the CMP. This is because for parallel workloads performance counters (and time-based estimations) are almost useless for relating performance and power (due to synchronization points). A spinning core may have a high IPC by doing nothing but spinning. In other words, synchronization points may increase global execution time although local core performance counters show a performance increase. Moreover, none of the mentioned techniques provide any accuracy analysis when matching an imposed power budget.

In 2008, Cai *et al.* proposed *meeting points* [20] which locates critical threads in parallel regions and uses DVFS to reduce energy consumption of non-critical threads. They propose two approaches: thread delaying, that slows down the fastest thread to ensure that all the threads get to the synchronization point (meeting point) at the same time; and thread balancing, that gives priority to the critical thread when accessing resources in a 2-way SMT processor. They achieve substantial energy reduction as long as the critical thread can be identified.

In the commercial area, Intel's i7 turbo mode shuts off idle cores, reducing their voltage to zero rather than just lowering the power provided to them. Not having as many cores that produce heat will allow other cores to use more power, increasing the performance of those cores while still not exceeding the maximum TDP of the processor. This is useful when running sequential or low-parallel applications. However, for parallel workloads, overclocking two cores does not necessarily mean a performance improvement due to memory dependences and synchronization points.

5.3. Enforcing a Power Budget in CMPs

Most of the previous proposals in power control focus on global power or energy reduction. However, sometimes we require a precise core-level power/energy control during regular usage of the processor due to temporal power or thermal constraints. This way we can reuse existent hardware in a different scenario it was originally intended for, increase the number of computation cores reusing a TDP, etc. In this chapter we introduce a mechanism to restrain the power dissipation so that the processor can accurately match an imposed power budget in an energy-efficient way. To achieve this goal we first detect program points where power can be saved without harming performance (e.g., spin-locks, wrong execution paths, cache misses, etc.) and reduce it; second, we balance the power between the cores; and finally (when nothing else can be done), we reduce the power locally even at the cost of degrading performance (by means of DVFS and/or microarchitectural techniques). This section will present our simulation environment, analyze why legacy power control mechanisms are unable to match a predefined power budget in CMPs and introduce *Power Token Balancing*, a power-control mechanism that balances power between cores to control global power usage.

Table 5.1: Core configuration.

Processor Core	
Process Technology:	32 nanometers
Frequency:	3000 Mhz
V_{DD}:	0.9 V
Instruction Window	128 RUU + 64 IW
Load Store Queue	64 Entries
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	6 Int Alu; 2 Int Mult
	4 FP Alu; 4 FP Mult
Branch Predictor:	16bit Gshare
Memory Hierarchy	
Coherence Prot.:	MOESI
Memory Latency:	300
L1 I-cache:	64KB, 2-way, 1 cycle lat.
L1 D-cache:	64KB, 2-way, 1 cycle lat.
L2 cache:	1MB/core, 4-way, unified
	12 cycle latency
TLB:	256 entries
Network Parameters	
Topology:	2D mesh
Link Latency:	4 cycles
Flit size:	4 bytes
Link Bandwidth:	1 flit / cycle

Table 5.2: Evaluated benchmarks and input working sets.

	Benchmark	*Size*	*Benchmark*	*Size*
SPLASH-2	Barnes	8192 bodies, 4 time steps	Raytrace	Teapot
	Cholesky	tk16.0	Water-NSQ	512 molecules, 4 time steps
	FFT	256K complex doubles	Water-SP	512 molecules, 4 time steps
	Ocean	258x258 ocean	Tomcatv	256 elements, 5 iterations
	Radix	1M keys, 1024 radix	Unstructured	Mesh.2K, 5 time steps
PARSEC	Blackscholes	simsmall	Swaptions	simsmall
	Fluidanimate	simsmall	x264	simsmall

5.3.1. Simulation Environment

For evaluating the proposed approaches we have used the Virtutech Simics platform [65] extended with Wisconsin GEMS v2.1 [67]. GEMS provides both detailed memory simulation through a module called Ruby and a cycle-level pipeline simulation through a module called Opal. We have extended both Opal and Ruby with all the studied mechanisms that will be explained later. The simulated system is a homogeneous CMP consisting of a number of replicated cores connected by a switched 2D-mesh direct network. Table 5.1 shows the most relevant parameters of the simulated system. Power scaling factors for a 32nm technology were obtained from McPAT [56]. To evaluate the performance and power dissipation of the different mechanisms we used scientific applications from the SPLASH-2 benchmark suite in addition to some PARSEC applications (the ones that finished execution in less than 3 days in our cluster). Results have been extracted from the parallel phase of each benchmark. Benchmark sizes are specified in Table 5.2.

We will provide overall CMP energy consumption along with the *accuracy* of each evaluated technique on matching a predefined global power budget in the simulation results. To measure each technique's accuracy we use the metric *Area over the Power Budget* (AoPB) defined in Section 2.3. This metric measures the amount of energy (in joules) between the power budget and each core dynamic power. The lower the area (energy) the more accurate the technique (remember that the ideal AoPB is zero). Performance results are shown for the dynamic approach (see section 5.4.2).

5.3.2. Matching a Power Budget in a CMP Running Parallel Workloads

Once we have a mechanism to measure power in a core, the next step is to analyze how different power saving mechanisms behave under power constraints. Initially, we will adapt and tune the proposed techniques in Chapter 4 to a CMP scenario. The evaluated techniques are:

- DVFS with five power modes (Voltage, Frequency): (100% V_{DD}, 100% f); (95% V_{DD}, 95% f); (90% V_{DD}, 90% f); (90% V_{DD}, 75% f); and (90% V_{DD}, 65% f).
- DFS: Similar to a) but only scaling down frequency. I.e., V_{DD} remains 100% in all cases.
- Two-Level (DVFS+BBLM): As proposed in the previous chapter, this 2-level approach uses DVFS to lower the average power dissipation towards the power budget and then uses different microarchitectural techniques to remove the remaining power spikes.

Note that these techniques were designed for the single-core scenario and, hence, they are applied at the core-level instead of at the CMP-level. Therefore, the first step should

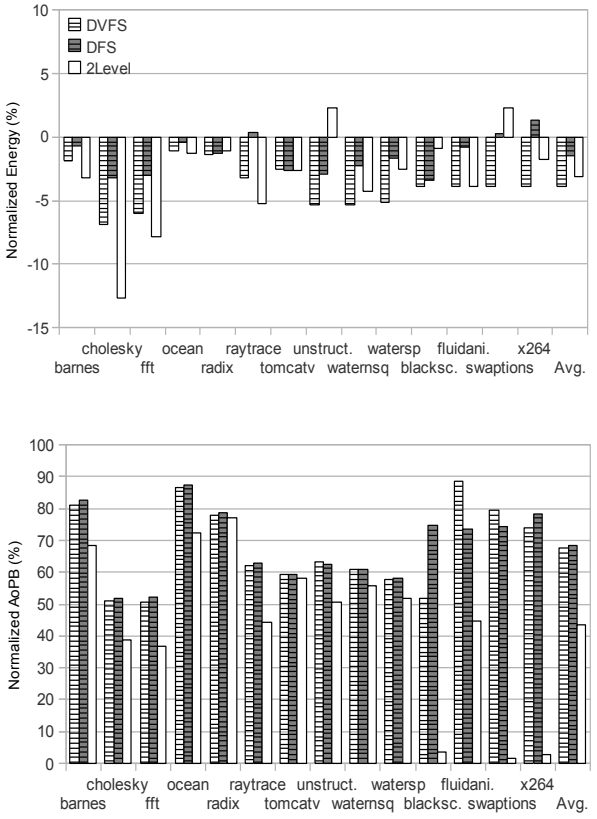


Figure 5.1: Normalized Energy (top) and AoPB (bottom) for a 16-core CMP with a power budget of 50%.

be to decide how to split the available power for the whole CMP (as determined by the global power budget) among the individual cores. An initial and straightforward implementation is to equally split the available power among all cores. In this case, power budget techniques will be locally applied to a particular core under two conditions:

1. The whole CMP is over the global power budget:

$$\sum Core_i Power > GlobalPowerBudget$$

2. A particular core is over its local power budget:

$$Core_i Power > GlobalPowerBudget/NumberOfCores$$

To analyze whether the single-core mechanisms work properly in the CMP scenario we will apply the above power matching techniques (DVFS, DFS, 2level) to a 16-core CMP (results for 2, 4 and 8 cores have been omitted for the sake of visibility) for the SPLASH-2 benchmark suite and some PARSEC benchmarks with a global power budget set to 50% of the original processor peak power dissipation using clock gating. It is important to note that we have selected Kim's implementation [51] as a best case scenario for DVFS with a fast transition time of 30-50 mV/ns. Using a slower and more realistic DVFS

will mean that microarchitecture-level techniques (used in the 2-level experiment) will become even more accurate and energy-efficient than DVFS.

In Figure 5.1 we can see the normalized energy and area over the power budget (AoPB) with respect to a base case where no power-control mechanisms are used to match the global power budget. If we take a look at the energy numbers we can notice that all the evaluated techniques behave accordingly with the reported numbers in Chapter 4 for the single-core scenario. In benchmarks like Cholesky, the 2-level approach is able to reduce energy by almost 13%. In terms of performance, the average degradation is under 1% for the studied benchmarks. However, differences arise when looking at the accuracy metric (AoPB). Although there are particular benchmarks that report a reduced AoPB (depending on the evaluated technique - for example Blackscholes, Swaptions and x264 from PARSEC), the average AoPB is still very high. We obtained an average of 45% AoPB, which is far from the average 10% AoPB obtained for the single-core scenario in the previous chapter. Moreover, for benchmarks like Ocean and Radix, the AoPB is especially high, around 70-80%, which means that the global power budget constraint is not properly respected.

There is a key difference between single-threaded applications and parallel workloads: synchronization points. In parallel workloads we noticed that synchronization points alter the optimal execution of the code, so it is no longer dictated by each individual core, but by the whole CMP. These syncronization points are messing up the AoPB results. There are some benchmarks with low AoPB in Figure 5.1-bottom, these benchmarks have no lock/barrier contention as we will see in Figure 5.2 because they do not have synchronization points, so single-core mechanisms work properly for them. This initial analysis shows that previous mechanisms for managing power under temporary power constraints are not suitable for a CMP scenario when using this initial distribution policy that equally splits the power among cores.

5.3.3. Analysis on the Power Dissipated in Spinning

The primary goal of this chapter is to accurately match an imposed power budget in an energy-efficient way while having in mind the peculiarities of CMP processors running parallel workloads. In this case, it is important to focus on places where power can be saved without harming performance such as synchronization points.

Figure 5.2 shows an analysis on the time spent by a CMP with a varying number of cores (from 2 to 16) either spinning or performing useful work. Each bar shows the fraction of time spent in lock acquisition, lock release, barriers, and useful computation (busy). As expected, the time each application wastes in spinning grows linearly with the number of cores. Some applications (Unstructured/Fluidanimate) spend a significant time in Lock-Acq and Lock-Rel states (contended locks) while others (Cholesky/Blackscholes/Swaptions/x264), in contrast, have no lock/barrier contention. While Cholesly's behavior is due to a well balanced code, the other three benchmarks only synchronize at the end of the code.

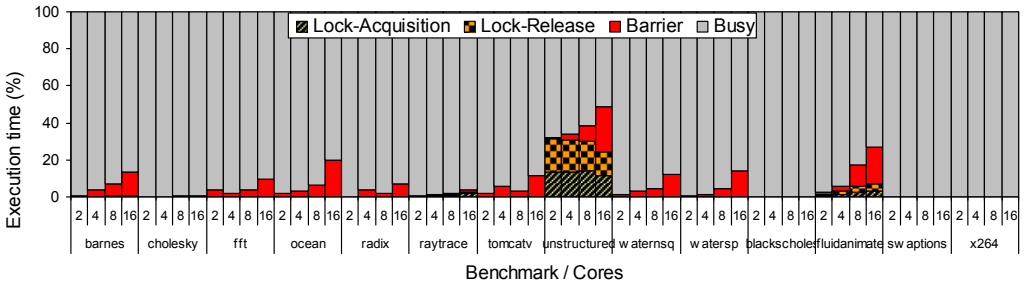


Figure 5.2: Execution time breakdown for a varying number of cores.

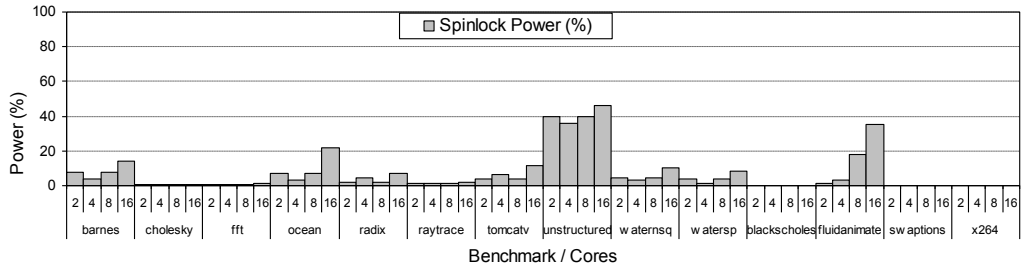


Figure 5.3: Normalized spinlock power for a varying number of cores.

In Figure 5.3 we can see the fraction of power wasted in spinning states, normalized to the total power dissipated by the original processor. As explained in Section 5.2.1, this wasted power can be reduced by detecting spinning and slowing down/stalling the cores. The Power-Token approach can be used to indirectly detect spinning states as we will describe in the next section. This spinlock power is close to a 10% on average for a 16-core processor running all the studied benchmarks. In any case, this potential power savings due to spinning are not enough to accurately match a restrictive power budget (e.g., 50% of the peak power) since a) it is a small amount (10%); and b) spinning is located in very specific points over time while we are aimed at meeting the power budget constraint as long as it lasts. Therefore, we need a more generic approach that could benefit from other wasteful situations such as mispredictions events.

5.3.4. Power Token Balancing (PTB)

5.3.4.1. PTB Motivation and Fundamentals

Imagine a power-constrained scenario with a global power budget that we need to satisfy and local power budgets that individual cores try to match. Now that we can account power at a cycle level by using power tokens, we can find out situations where power imbalance exists among the cores of the CMP. Once detected we can balance their power and minimize performance degradation, and that is what PTB tries to achieve. Figure 5.4 shows an example where the total power dissipated by the CMP is over the global power budget, but some cores are under their power budget share. In this example we assume a 4-core CMP and global power budget of 40W, with a simple implementation that equally splits power between cores, so each core has a local power budget of 10W.

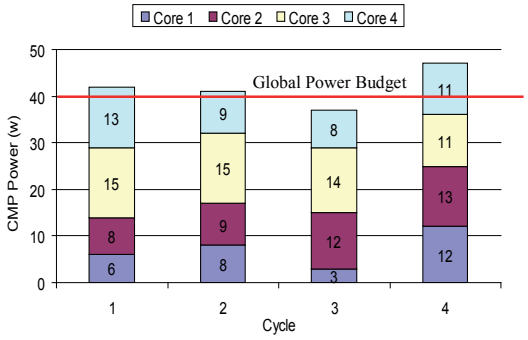


Figure 5.4: Power Token Balancing motivation (not real numbers).

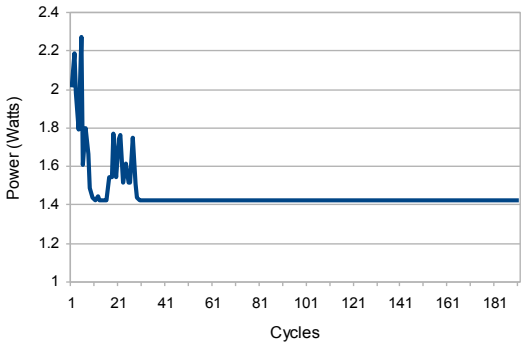


Figure 5.5: Per-cycle power behavior of a spinning core.

We can notice that cycles 1, 2 and 4 are over the global power budget (40W), so we enable the local power-saving mechanisms. In cycle 1 no power-control mechanisms are applied to cores 1 and 2, since they are under their local power budget (10W). On the other hand, cores 3 and 4 need to use their local power-saving mechanisms to match their local power budget. However, if those cores run critical execution threads and we slow them down, we could potentially harm performance and energy in a future synchronization point. The same happens in cycle 2 with core 3. In cycle 3, even though there are cores exceeding their local power budget, no mechanism is applied as the global CMP power is under the budget. Finally, in cycle 4, all cores exceed their local power budget (so does the CMP exceeding the global one), and hence local mechanisms are applied to all cores. However, if we were able to tell cores 3&4 in cycles 1 and 2 that their respective local power budgets are less restrictive than 10W (since cores 1&2 have some power left, 4+2 W in cycle 1; 2+1 W in cycle 2) then the effects on the performance should be less harmful.

In PTB each individual core will count, at a cycle level, the number of power tokens it has consumed from the available local power tokens. In a given cycle, if a core still has available power tokens and the CMP is over the global power budget then the core offers its spare tokens to the PTB load-balancer. Tokens are used as a currency to account for power, so it is important to note that they are neither sent nor received. In PTB cores just send the number of spare tokens. Analogously, cores over their local power budget will receive extra tokens from the PTB load-balancer which will prevent them to enable a power-saving technique (that can reduce performance) as long as the global power budget constraint is met. The PTB load-balancer calculates every cycle the overall available power tokens based on the spare tokens that cores have for each cycle. Therefore, PTB is not a loan/refund mechanism since a core can reuse power from others but there is no need to give it back. In PTB we define two power distribution policies that will be discussed later: a) give tokens to the most power-hungry core (*ToOne* policy); or b) equally distribute the extra tokens among all cores over the power budget (*ToAll* policy). PTB also exhibits two inherent features that allow "transparent" optimizations without any specific mechanism: 1) indirect spinning detection, and 2) an automatic

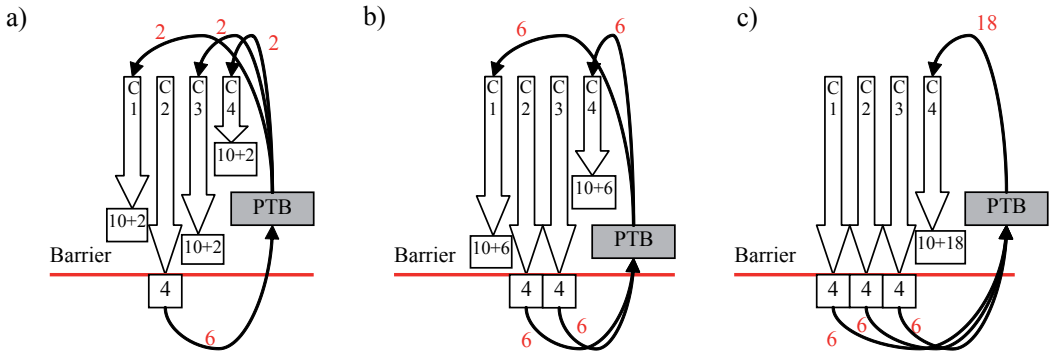


Figure 5.6: Power Token Balancing example in the case of a barrier (using the *ToAll* policy).

priority system for non-spinning threads.

Figures 5.5 help us to illustrate how PTB could be used to detect spinning. When a core enters a spinning state, the dynamic power follows the behavior shown in Figure 5.5. In this Figure we can see an initial power peak due to useful computation. If the spinning state lasts enough, the pipeline empties and power goes down and stabilizes (cycle >35 in Figure 5.5) to an amount that is usually under the budget. We can assume then that the core is spinning. Note that spinning is just a particular case of power imbalance, so our mechanism will benefit from it but that is not the only case. Remember that PTB knows nothing about locks, barriers, mispredictions, etc, it just balances power.

For illustrating purposes and continuing with the spinning example, Figure 5.6 shows how PTB works in the case of a barrier (using the *ToAll* policy). For this example let us assume there are four cores (C1 to C4) with local power budgets set to 10 tokens and that when spinning a core consumes 4 tokens. As cited before, a spinning core gives its spare tokens to the PTB load-balancer. Figure 5.6-a shows that core 2 reaches the barrier and transfers 6 tokens to the load-balancer. Now the rest of cores have more available power left to burn until they get to the synchronization point (in our example, cores 1, 3, 4 receive 2 extra tokens each from the load-balancer, raising their local budget to 12 tokens). When any other core (e.g., core 3 in Figure 5.6-b) reaches to the barrier it also gives 6 spare tokens to the PTB load-balancer which allows cores 1&4 to use the 6+6 extra tokens from cores 2&3, raising their local budget to 16. Finally, Figure 5.6-c shows core 1 spinning in the barrier and giving its 6 spare tokens to the PTB load-balancer which prevents the last core (C4) to be slowed down as it can use all the spare tokens. Again, note that PTB does not explicitly distinguish between barrier- or lock-spinning. PTB basically detects power imbalance among cores and will benefit from any misprediction event (e.g., a cache miss or a mispredicted branch), not only from spinning states.

5.3.4.2. PTB Implementation Details

The implementation of *Power Token Balancing* is based on a centralized structure called PTB load-balancer. This structure receives the number of spare power tokens from

all cores under their local power budget and splits them among the cores exceeding it (intending not to trigger any power-saving mechanism for the exceeding cores which would result in a performance degradation). Balancing is done at a cycle level, so tokens from previous cycles are not stored in the balancer. To exchange token information we need to build communication wires between the cores and the PTB load-balancer, as depicted in Figure 5.7. We will use 4 wires for sending and 4 wires for receiving the number of tokens per core; this limits the amount of given/received tokens but makes the mechanism more power-efficient. Note that these wires are used to send the amount of spare tokens, not the tokens themselves (tokens are used as a currency to account for power). All these wires will be placed on a different layer of that of the interconnection network.

To estimate latency delays of the communication wires we used Xilinx ISE for a processor running at 3GHz without buffers as a reference to calculate the logic delay of the circuit. We removed the delay caused by both pins and routing, making the logic delay almost equivalent to the delay of a circuit in an ASIC implementation. For a 4-core CMP delays are: one cycle for sending the number of spare tokens, one for processing tokens and one for sending the number of spare tokens back to the cores over the power budget. For an 8-core processor, wire delay increases to 2 cycles, so it will take a total of 5 cycles to send and receive the number of spare tokens to/from the PTB load-balancer. For a 16-core CMP the mechanism needs 4 cycles for receiving the tokens, 2 cycles for processing and 4 cycles for sending the tokens to the cores over the power budget, according to Xilinx ISE. When a core gives away tokens it sets a more restrictive power budget to ensure it won't dissipate power until tokens reach its destination. The power dissipation of the PTB mechanism plus the communication wires has been estimated using Xilinx XPower Analyzer with the same configuration as the delay latency, increasing the average application power dissipation by just 1%, which is also accounted in the experimental results presented in the next section.

Problems might arise as we increase the number of processing cores, and thus, the PTB load-balancer communication and processing latencies. However, for the analyzed number of cores and latencies, experimental results show significant improvements in terms of temperature, energy and accuracy on matching the power budget, even with a pessimistic 10-cycle delay for sending/receiving the number of spare tokens from other cores. Nevertheless, one approach to make PTB more scalable (>32 cores) consists of clustering the PTB load-balancer into groups of 8 or 16 cores and replicate the structure as needed. Results in next section will show that such a group of cores (8 or 16) is enough for PTB to efficiently balance power and accurately match the imposed power budget.

5.3.5. Reusing Wasted Power to Reduce Energy: Nitro

This idea is inspired by the Formula One's KERS (*Kinetic Energy Recovery System*) mechanism, also known as regenerative brake. As mentioned before, when running parallel workloads on a CMP, speeding up or slowing down a specific core may not vary the

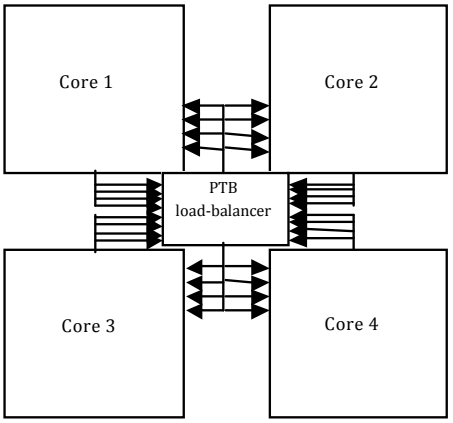


Figure 5.7: PTB implementation diagram for a 4-core CMP.

final program execution time due to synchronization points. For example, overclocking a core may not lead to any performance improvement. The key point in a CMP running a parallel application is that, in general, it is more crucial *when* you apply the mechanism than the mechanism itself. The idea behind Nitro is quite simple: save power when a core does not need it (e.g., while spinning) and reuse it when it becomes really useful (e.g., critical threads/sections). Nitro differs from other spinning-based mechanisms [20][57][55] in two things. First, all these mechanisms are meant to reduce energy consumption before/while spinning, whereas Nitro tries to reuse this energy somewhere else. Second, all these mechanisms usually try to exploit the barrier synchronization mechanism, while Nitro benefits from locks and barriers. In general terms, Nitro approach may resemble the i7 turbo mode but in a finer-grain. However, Nitro is designed for a parallel workload scenario, looking for code sections that we know for sure will benefit from local overclocking.

As mentioned in Section 5.2.1.1, spinning states can be detected either by hardware or by static instructions introduced by the programmer. In any case, those sections can be identified and, by using the power-token approach, we can approximate the amount of power spent in spinning (depending on how long a core stays spinning). However, a small structure to account for total power-tokens saved is still needed. Nitro works as follows: once a lock is detected, the processor that gets access to the lock after the contention period (lock acquisition) is overclocked, as long as we have power-tokens left to overclock and for as long as the critical section lasts. We will assume the proposed DVFS by Kim *et al.* [51] which is able to quickly switch between power modes at speeds of 30-50mV/ns. The overclocked core will run at a 15% faster rate than the base frequency. Of course, sometimes there are not enough cycles in the critical section to take advantage of Nitro, so we need some kind of mechanism to estimate the duration of the critical section and only apply Nitro if it lasts for long enough. For this purpose we use the spinning predictor proposed in [55], based on the PC to predict a critical section duration. The overclocking will also last for a short period of time, so the processor will still have time to recover after the overclocking. Nitro can also be applied to speed up the remaining threads while

others wait in a barrier, but we did not implement this feature.

5.3.5.1. Nitro Outside Locks

Nitro was designed to benefit from wasted power from spinning cores, saving that power and reusing it somewhere else. We decided to focus our analysis in lock-delimited code sections, because there were many previous works that focused on reusing or balancing power from spinning cores in barriers [20][57][55][82]. However, with minimal modifications on the PTB mechanism, Nitro can be used to speed up execution in both barriers and locks. If we take a look to the example from Figure 5.6-c, once cores 1 2 and 3 get to the barrier, the PTB load-balancer receives all their spare tokens, and gives them to core 4. As there is only one core left, it probably won't even use any power saving mechanism, either because the total power dissipated by the whole CMP will be under the global budget (one of the requirements from Section 5.3.2) or because the extra spare tokens from the PTB load-balancer prevent the usage of local power saving mechanisms. Under these conditions we could overclock core 4, the last one to get to the barrier, in order to reach the synchronization point faster, and thus reduce global execution time. The overclocking should only be done as long as we have power tokens left from the PTB load-balancer.

More specifically, if PTB is in *ToOne* mode, Nitro will overclock the most power hungry core (note that this core is not necessarily the latest to get to the synchronization point), speeding it up as long as we have tokens left from spinning cores. On the other hand, if PTB is in *ToAll* mode, power tokens will be equally divided between all the cores over the budget, and, eventually, all these cores will receive enough tokens to overclock themselves, speeding up more and more as cores reach the synchronization point. We did not implement Nitro outside locks because of time restrictions, but we are wiling to evaluate the potential of this mechanism in our future work.

5.4. Experimental Results

5.4.1. Efficiency of Power Token Balancing (PTB)

In this section we perform an analysis of the PTB mechanism with the previously defined power-token distribution policies: *ToAll* (that shares the power-tokens among all the cores over their local power budget) and *ToOne* (that gives all the spare power-tokens to the core that needs them the most). The selected global power budget will be 50% of the peak power dissipation[1] with clock gating and a varying number of processing cores in the CMP (2 to 16 cores). Results are normalized to a base case without power-control mechanisms to match the global power budget.

Figure 5.8-top shows the energy consumption for the evaluated techniques (enumerated

[1]We only report results for a 50% power budget for the sake of visibility. For less restrictive power budgets PTB also works properly.

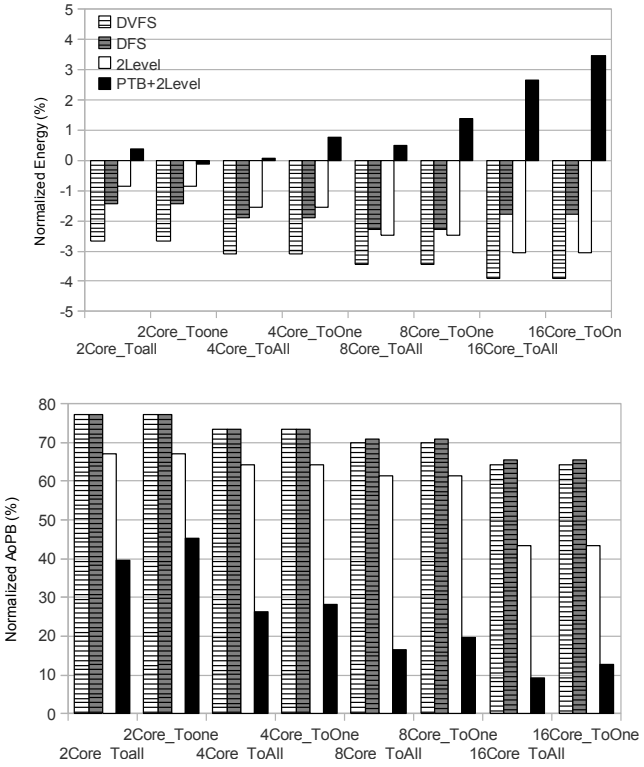


Figure 5.8: Normalized energy (top) and area over the power budget (bottom) for a varying number of cores and PTB policies.

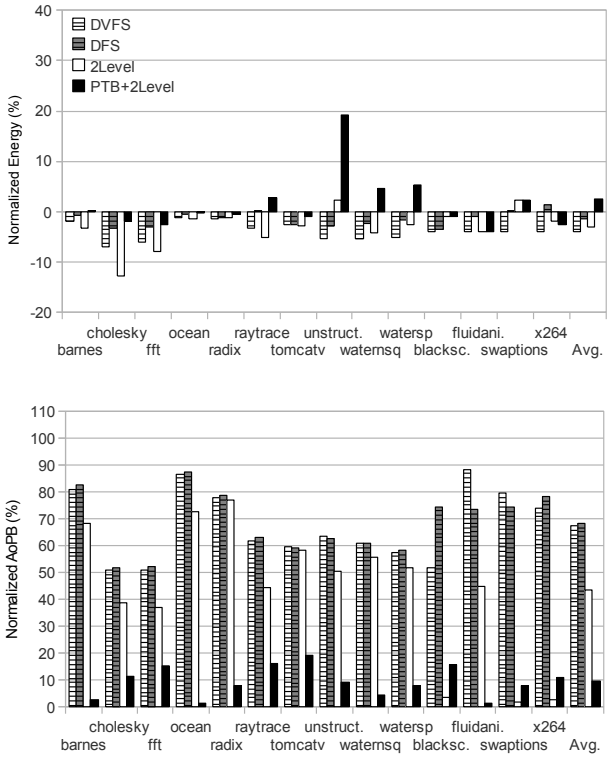


Figure 5.9: Detailed energy (top) and AoPB (bottom) for a 16-core CMP with the ToAll PTB policy.

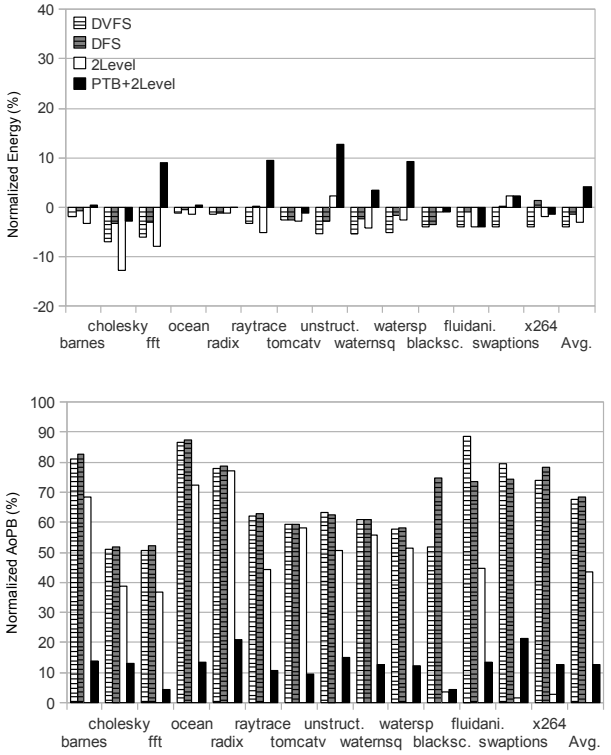


Figure 5.10: Detailed energy (top) and AoPB (bottom) for a 16-core CMP with the ToOne PTB policy.

in Section 5.3.2) using different combinations of core number/policy whereas Figure 5.8-bottom shows the AoPB metric. We can observe that when using the proposed PTB mechanism, area numbers go back to the reported numbers in the previous chapter for the single-core scenario: average 10% of AoPB for a 16-core CMP when the PTB+2level technique is used (although energy numbers are not as good as in the single-core scenario). In a 16-core CMP, DVFS and DFS are unable to lower the AoPB below 65% while PTB+2level reduces the average area to just 8%, getting close to the ideal AoPB of zero, with only 3% more energy consumed. It can also be observed that the accuracy on matching the power budget increases (i.e., AoPB decreases) with the number of cores, because we have more chances of receiving tokens from other cores. A more detailed analysis (Figures 5.9 and 5.10) shows that there are benchmarks, like Unstructured, where energy increases when using power saving techniques (mainly due to sync points - see Figures 5.2 and 5.3). Unstructured has many thread dependences and slowing down a core causes a great impact on performance. On the other hand, benchmarks like Barnes and Ocean, that reported very high AoPB of 70% for the initial power-distribution implementation discussed in Section 5.3.2 (Figure 5.1), now offer an AoPB of just 2% thanks to the efficient power distribution among cores performed by PTB. However, in some benchmarks the extra accuracy on matching the global power budget comes at the cost of higher energy consumption than that of DVFS alone. DVFS shows an average energy reduction of 6% whereas PTB increases the energy in 3%. Note, however, that this energy increase can be turned into energy savings if we relax the accuracy constraint

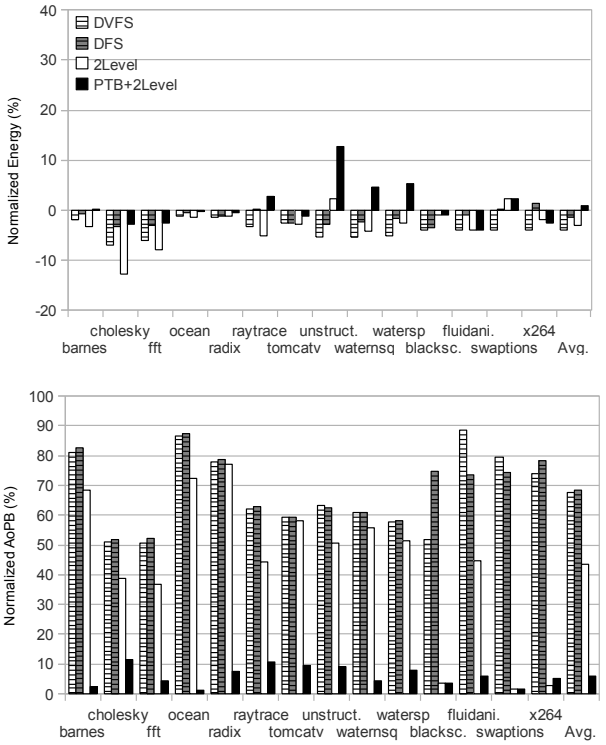


Figure 5.11: Detailed energy (top) and AoPB (bottom) for a 16-core CMP using the dynamic policy selector.

of PTB, as we will show in section 5.4.3.

If we compare area and energy results for both power-token distribution policies (*ToAll* and *ToOne*) we can see that, on average, the former works better than the later. Benchmarks like Unstructured and Waternsq work better when the extra power is given to a single core rather than to all cores. In these benchmarks, threads have an unbalanced workload and spend a significant fraction of their time spinning on locks; therefore, they benefit from giving the extra power (priority) to threads that enter in a critical section (i.e., the *ToOne* policy) so it will finish faster and release the lock.

5.4.2. Dynamic Policy Selector

When working with barriers, the *ToAll* policy will split power tokens from cores already waiting in the barrier among the remaining cores, speeding them all in order to get to the barrier as soon as possible whereas the *ToOne* policy will only benefit one core, that will get faster to the barrier, but we still have to wait for the rest of cores. On the other hand, when a core is spinning in a lock and gets access to a critical section, giving all the tokens to this core will benefit the overall program execution, as the core that enters the critical section can finish this section faster and release the lock. Results in the previous section showed that the *ToAll* policy is best suited for applications with many barriers in the code whereas the *ToOne* policy works better for applications with high lock contention. Therefore, in order to enhance the PTB power balancing mechanism we have included a dynamic selector for the power sharing policy (either *ToOne* or *ToAll*).

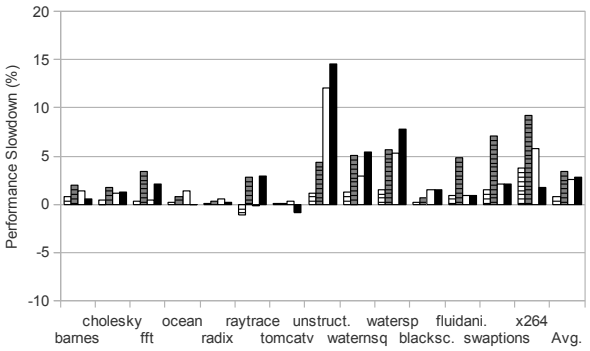


Figure 5.12: Detailed Performance for a 16-core CMP using the dynamic policy selector.

This selector will change the policy depending on the current state of the spinning cores. If the spinning is taking place to access a lock, the mechanism will use the *ToOne* policy. If the spinning is taking place in a barrier (or there is no spinning) the PTB mechanism will use the *ToAll* policy.

Figure 5.11 shows how this dynamic policy selection approach obtains the best results for the evaluated techniques in terms of both area and energy metrics. We can also observe in Figure 5.12 that PTB is really close to DVFS (around 2%) in terms of performance. Moreover, PTB has the extra benefit of being far more accurate on matching the imposed power budget than DVFS or DFS approaches. Normalized energy goes down to 2%, 1% less than the static *ToAll* and 3% less than the static *ToOne*. Accuracy is improved in 3% compared with the static *ToAll* and 5% compared with the static *ToOne* policy. In terms of performance, Unstructured is the application that is more affected by the microarchitectural power-control mechanisms.

Note that this dynamic policy selector, for the presented results, is assisted by actual application-specific information although pure indirect dynamic detection of the type of spinning is possible (and practical) via heuristics: e.g., monitoring the number of cores that stop spinning simultaneously via their power token consumption, run-time instruction analysis or other techniques similar to those described in [57]. For the sake of clarity, we only report on the first approach which does not entail any additional energy cost for the classification of spinning to barrier- or lock-spinning.

5.4.3. Relaxing PTB to be More Energy-Efficient

Up to this point we have focused on a PTB mechanism that optimizes the accuracy on matching the given power budget. As explained before, this kind of optimization hurts performance and, therefore, increases overall energy consumption. However, if we relax the accuracy constraint, PTB can also achieve positive energy savings since power-saving mechanisms would be applied in a less restrictive way, not affecting performance that much. In order to analyze this new focus, Figure 5.13 shows how PTB behaves when optimizing for energy-efficiency instead of just accuracy for several relaxed area thresholds (+10%, +20%, +30%, etc). These relaxed area thresholds are used to delay

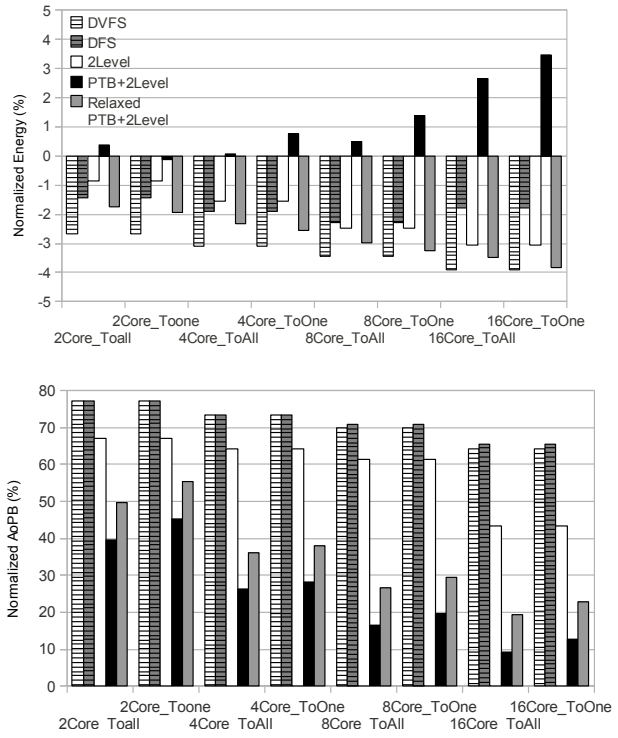


Figure 5.13: Normalized energy (top) and area over the power budget (bottom) for a varying number of cores and PTB policies.

triggering a power-saving mechanism when the global/local power budgets are exceeded. Note that the original PTB triggers the power-saving mechanisms *immediately* after detecting that the current power budget was exceeded.

If we take for example a 16-core CMP and relax the AoPB metric allowing it to be 20% above the power budget, PTB obtains an average energy reduction of 4% (Figure 5.13-top) similar to that obtained by DVFS, and still being far more accurate than DVFS (as seen in Figure 5.13-bottom) on matching the power budget. Of course, better energy savings could be achieved for the same 16-core CMP if we relax the area constraint even more. Finally, if we use PTB as a spinlock detector and we disable the spinning cores to save power we could further increase the energy savings.

5.4.4. The Importance of Accuracy

In this section we will brush up the example introduced in Chapter 2, that illustrates the importance of the accuracy on matching a predefined power budget by increasing the number of cores of a CMP maintaining the same TDP (Thermal Design Power). Let us assume that we want to increase the number of cores in a CMP maintaining the same TDP. For a 16-core CMP with a 100W TDP each core would use 6.25W (for simplicity let us ignore the interconnection network). If we set a power budget of 50% we could ideally duplicate the number of cores in that CMP with the same TDP (up to 32 cores, each one dissipating an average of 3.125W). But for this ideal case a perfect accuracy on matching the power budget is needed.

According to the previous results, DVFS incurs in an energy deviation of 65% over the power budget (the AoPB metric). Therefore, with such 65% deviation each core dissipation raises to 3.125*1.65=5.15W, meaning that for a 100W TDP we can put a maximum of 100/5.15=19 cores inside the CMP. Using a regular 2level approach (without PTB) the deviation is reduced to 40% that gives us an average power dissipation of 3.125*1.40=4.375W per core, so we can put 100/4.375=22 cores in the CMP with the same TDP. Finally, when using the non-relaxed PTB approach the error is reduced below 10%, that gives us a potential average power dissipation of 3.125*1.1=3.4375W per core, so we could put 100/3.4375=29 cores inside our CMP. Therefore, thanks to the extra cores (we could go from 16 cores in the original CMP design without PTB to 29 cores) we can perfectly overcome the 3% performance degradation that results from using our proposed PTB mechanism if the application is parallel enough to use these extra cores.

5.4.5. Temperature Analysis

Thermal hotspots increase cooling costs and have a negative impact on reliability and performance. The significant increase in cooling costs requires designs for temperature margins lower than the worst-case. When we reduce the per-cycle power dissipation of an application we can consequently reduce the CMP temperature over time. Moreover, leakage power is exponentially dependent on temperature and an incremental feedback loop exists between temperature and leakage, which may turn small structures into hotspots and potentially damage the circuit. High temperatures also adversely affect performance, as the effective operating speed of transistors decreases as they heat up. In this section we will analyze the per-structure and per-benchmark temperature for the base case, PTB and DVFS.

Temperature numbers were obtained by introducing the HotSpot 5.0 [88] thermal models into Opal and building our tiled CMP by replicating N times our custom floorplan (depicted in Figure 5.14), where N is the number of cores. More specifically, we have modeled both leakage (through McPAT [56]) and the leakage/temperature loop in Opal, so leakage will be updated on every Hotspot exploration window (10K cycles). Leakage power is translated into power tokens and updated according to the formula $L_{new} = L_{Base} \cdot e^{Leak_{\beta} \cdot (T_{current} - T_{base})}$ (Chapter 2) where $Leak_{\beta}$ depends on technology scaling factor and is provided by HotSpot 5.0, L_{new} is the updated leakage, L_{Base} is the base leakage (obtained using McPAT thermal models), $T_{current}$ is the current temperature and T_{base} is the base temperature. Once leakage is updated, it is translated back to *power tokens*. Another important parameter is the cooling system. The regular thermal resistance of a cooling system ranges from 0.25 K/W for the all-copper fan model at the highest speed setting, to 0.33 K/W for the copper/aluminum variety at the lowest setting. In this work we model a real-world Zalman CNPS7700-Cu heatsink with 0.25 K/W thermal resistance and an area of 3.268 cm2 (136mm side).

Figure 5.15 shows both average and peak (maximum) temperatures before using PTB for the studied benchmarks running on a 16-core CMP along with their corresponding

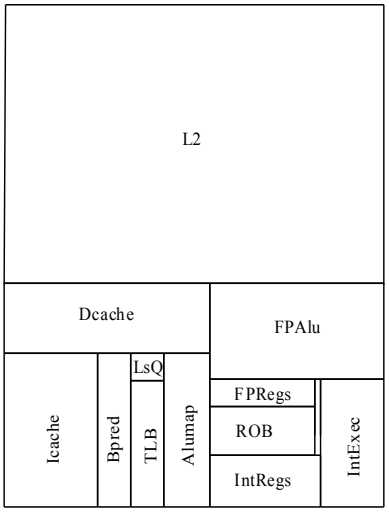


Figure 5.14: Core floorplan.

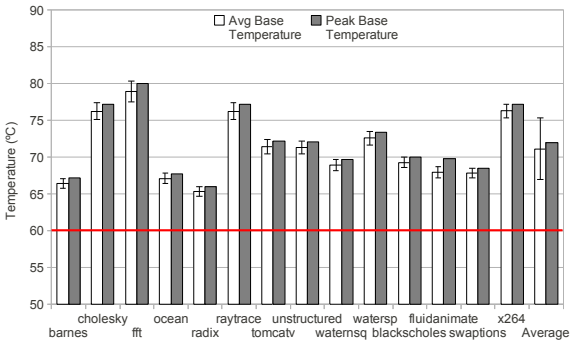


Figure 5.15: Average and peak temp. of a 16-core CMP.

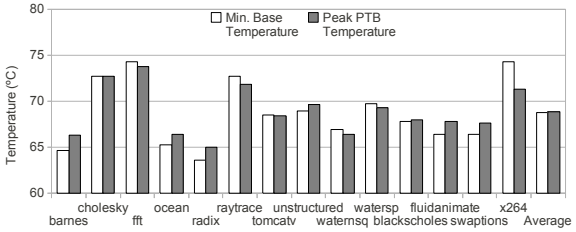


Figure 5.16: Minimum base vs peak PTB temperature of a 16-core CMP.

standard deviation. We define "idle" temperature as the temperature of the whole CMP in idle state (i.e., only the operating system is running). Therefore, the maximum temperature reduction any power saving mechanism can achieve will vary between the base peak/average temperature of the CMP and the idle temperature. For the studied 16-core CMP the idle temperature reported by McPAT is around 60°C (red line in Figure 5.15). In Figure 5.15 we also see that the average temperature is 72°C for all the evaluated benchmarks, therefore, the maximum temperature reduction we could ideally aspire is, on average, 12°C.

Figure 5.16 shows a comparison between the minimum temperature of the CMP without PTB (coolest core) against the peak temperature of the CMP when using PTB (hottest core). This initial study shows how PTB is able to balance temperature between the cores, lowering the peak temperature of the hottest core of the CMP to almost equal the temperature of the coolest core in the base CMP (without PTB). This is the benefit we expected from the balancing and highly accurate power budget matching our PTB mechanism provides, that ensures minimal deviation from the target power budget and, therefore, temperature.

For a more detailed analysis, Figure 5.17 shows the per-structure peak and average temperatures for a 16-core CMP for the base case, DVFS and PTB mechanisms. Temperatures are normalized against the maximum temperature gain (the difference between the peak/average temperature and the idle temperature we cited before). We can see how PTB and DVFS are able to reduce the temperature of all the internal structures of the core. However, PTB almost doubles the temperature reduction of DVFS, due to the extra accuracy when matching the target power budget. In particular, PTB obtains

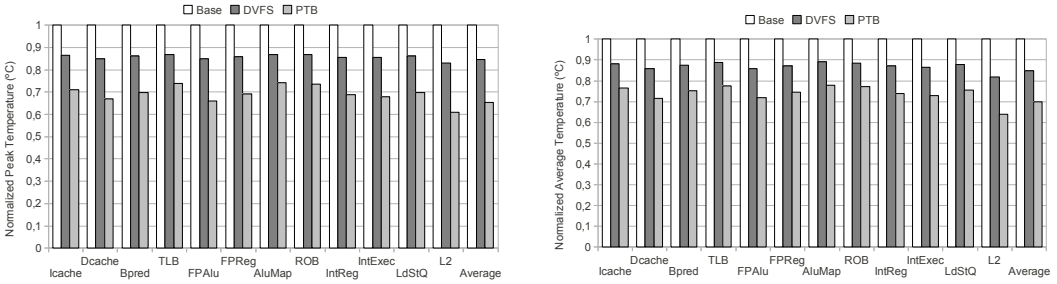


Figure 5.17: Normalized per-structure peak (left) and average (right) temperature analysis.

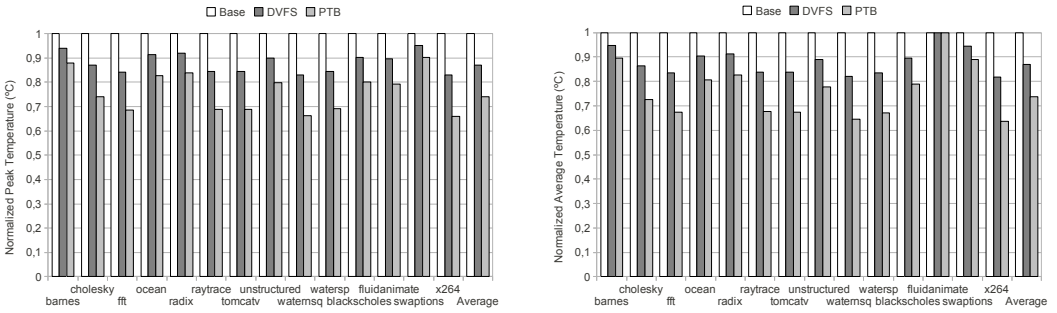


Figure 5.18: Normalized per-benchmark peak (left) and average (right) temperature analysis.

an average reduction of the peak temperature of all structures of 35% in addition to an average reduction of the average temperature of 30% for the evaluated 16-core CMP. On the other hand, the per-benchmark temperature reduction achieved by PTB follows the same trend, as it can be observed in Figure 5.18. We can see that, except for fluidanimate, there exists a temperature reduction in all of the studied benchmarks, for both peak and average temperature during the benchmark execution. The peak and average temperature reductions provided by the use of PTB are again close to 27% on average, almost doubling the temperature reduction provided by DVFS.

5.4.6. Nitro Energy and Performance Analysis

Finally, we have evaluated Nitro for a varying number of cores (from 2 to 16) running the SPLASH-2 benchmark suite and some benchmarks from the PARSEC 2.1 suite. Figure 5.19 shows the performance improvement and the normalized energy reduction for the different benchmarks. As expected, the benchmark that benefits the most from Nitro is Unstructured, which is the one that has the most lock contention from the set of studied benchmarks. Note, however, that this mechanism does not cause a heavy impact on energy in the rest of studied benchmarks. For Unstructured, the number of overclocked cycles represent just a 0.7% of the total simulation cycles, and that is enough to reduce the energy consumption of this benchmark by 3%. However, both the SPLASH-2 benchmark suite and the studied PARSEC benchmarks are quite optimized and contention periods are kept as low as possible, especially the ones related to locks, but this is not always the case in parallel applications such as for commercial and server

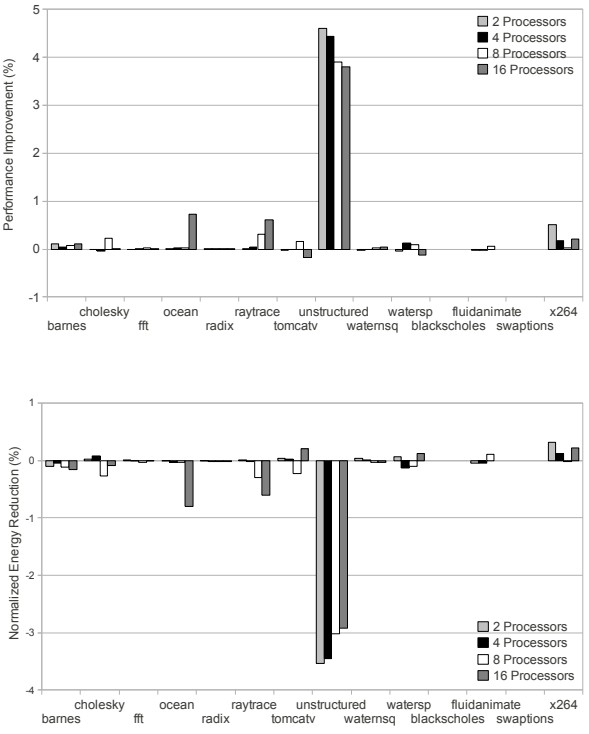


Figure 5.19: Performance improvement and energy reduction for a 16-core CMP using Nitro.

workloads (or any parallel application not coded by highly experienced programmers). Nevertheless, results obtained by the Nitro approach are encouraging since they show that, when contention for lock acquisition and release exists, our proposal can improve energy and increase performance. We are looking forward to analyze some commercial and server workloads or greater datasets to validate these results. Moreover, the extension of Nitro to work with both barrier and lock information could achieve even further energy savings. In section 5.3.3 we saw that many benchmarks spend part of their execution time waiting in barriers. This fraction of time increases as we increase the number of cores. If we detect cores in spinning state and use that wasted energy (approximately 10% according to Figure 5.3) to speed up cores doing useful work we could achieve greater energy savings. We did not implement this feature because of time limitations but we are wiling to analyze it's potential energy savings.

5.5. Conclusions

Design complexity and verification of microprocessors is increasingly costly. Some companies cannot afford the design of custom processors for their products (especially for cost-sensitive consumer handheld devices and gadgets) and have to rely on existing processors that may not meet their power requirements. In other scenarios it might be useful to increase the number of cores on a CMP maintaining the same thermal envelopment. Moreover, thermal envelop designs cannot be done for the worst case, because production costs are raised. Being able to set a power budget to the processor can be helpful in

these cases.

When analyzing CMPs running parallel workloads, previously proposed power managing mechanisms fail to accurately adapt to temporary power constraints, due to thread dependences and synchronization points. Moreover, power saved just from spinlocks is not enough to match an aggressive power budget because it is too local and too low. A global control mechanism is needed to match these design peculiarities. In this chapter we have proposed *Power Token Balancing* (PTB), a mechanism that dynamically balances power among the different cores to ensure that the whole processor accurately matches a predefined and global power budget. Our proposed mechanism accounts for unused power from cores that are under the power budget (translated into power-tokens) and passes that power to cores over the power budget, hence, not having to slow them down to match their local power budget constraint.

PTB is a fine-grain mechanism that ensures maximum accuracy with minimal standard deviation from the power budget, which is crucial if you want to optimize packaging costs or to increase the number of cores in a CMP with the same TDP. However, the accuracy constraint can be relaxed in order for PTB to be more energy-efficient by means of applying the power-saving mechanisms in a less restrictive way, and therefore, not affecting performance too much.

Experimental results have shown that PTB is able to accurately match the global power budget with an AoPB of just 8% for a 16-core CMP with a negligible energy increase (3%) while DVFS fails to match the power budget precisely, resulting in a high AoPB of around 65%. Furthermore, when considering a relaxed PTB approach that allows being 20% above the power budget (still far from the 65% AoPB obtained by DVFS), PTB obtains the same energy reduction as DVFS for the 16-core CMP. Of course, better energy savings can be achieved if the area constraint is further relaxed.

As a side effect of this accurate power budget matching, the use of PTB provides another interesting benefit: a more stable temperature over execution time. For the studied benchmarks we can obtain a 27-30% peak and average temperature reduction, that also applies to the individual structures. PTB is also able to balance temperature between cores, reducing the peak temperature of the hottest core making it equal to the temperature of the coldest core in the base CMP design. This temperature reduction not only reduces leakage power but also increases reliability and can result in reduced packing costs.

Finally, we have proposed Nitro, a mechanism based on the idea of Formula One's KERS. This technique saves power when not needed (e.g., from threads spinning in locks) and uses that power to overclock other cores that are executing critical sections of the program (e.g., code delimited by locks). This mechanism is suited for applications with contended locks and does not degrade performance for the rest of applications. Unfortunately, both the SPLASH-2 and the PARSEC 2.1 benchmark suites are optimized to reduce contention, and only one benchmark exhibit high lock contention periods. However, other regular programs with more coarse-grain locks will provide higher improvements.

Moreover, we can expect greater energy savings if we extend Nitro to work with barriers, speeding up the working cores using the energy from the spinning cores.

Chapter 6

3D Die-Stacked Power Budget Matching - Token3D

Summary: Nowadays, chip multiprocessors (CMPs) are the new standard design for a wide range of microprocessors: mobile devices (in the near future almost every smartphone will be governed by a CMP), desktop computers, laptop, servers, GPUs, APUs, etc. This new way of increasing performance by exploiting parallelism has two major drawbacks: off-chip bandwidth and communication latency between cores. 3D die-stacked processors are a recent design trend aimed at overcoming these drawbacks by stacking multiple device layers. However, the increase in packing density also leads to an increase in power density, which translates into thermal problems. Different proposals can be found in the literature to face these thermal problems such as dynamic thermal management (DTM), dynamic voltage and frequency scaling (DVFS), thread migration, etc. In this chapter we propose the use of microarchitectural power budget techniques to reduce peak temperature. In particular, we first introduce Token3D, a new power balancing policy that takes into account temperature and layout information to balance the available per core power along other power optimizations for 3D designs. And second, we analyze a wide range of floorplans looking for the optimal temperature configuration. Experimental results show a reduction of the peak temperature of 2-26°C depending on the selected floorplan.

6.1. Introduction

With the global market dominated by chip multiprocessors and the GHz race over, designers look for ways to increase productivity by increasing the number of available processing cores inside the CMP. The shrinking of transistor's feature size allows the integration of more cores, as the per-core power dissipation decreases with each new generation. However, interconnects have not followed the same scaling trend as transistors,

becoming a limiting factor in both performance and power dissipation. One intuitive solution to reduce wirelength of the interconnection network is to stack structures on top of each other, instead of using a traditional planar distribution.

Introduced by Souri *et al.* in [89], 3D architectures stack together multiple device layers (i.e., cores, memory) with direct vertical interconnects through them (inter-wafer vias or die-to-die vias). A direct consequence of this design is the reduction on the communication delays and power costs between different cores, as well as an increase in packing density that depends on the number of available layers. However, despite of the great benefits of 3D integration, there are several challenges that designers have to face. First, the increase in packing density also leads to an increase in power density that eventually translates into thermal problems. Second, a deeper design space exploration of different floorplan configurations is essential to take advantage of these emerging 3D technologies. Third, chip verification complexity increases with the number of layers.

To face the first challenge there are several proposals that come from the 2D field:

- Dynamic Voltage and Frequency Scaling (DVFS) to reduce power dissipation, and thus temperature. DVFS-based approaches can be applied either to the whole 3D chip or only to cores that show thermal problems (usually cores away from the edges of the 3D chip) [40, 62, 82].
- Task/thread migration to move execution threads from internal to external cores whenever possible, or reschedule memory intensive threads to internal cores and CPU intensive threads to external cores [28, 29, 102].

These mechanisms are usually triggered by a Dynamic Thermal Management (DTM) scheme, so whenever a core exceeds a certain temperature, power control or task migration mechanisms take place inside the CMP. However, these mechanisms are not perfect. As explained in previous chapters, DVFS is a coarse-grain mechanism usually triggered by the operating system with very long transition times between power modes that leads to a high variability in temperature. On the other hand, task migration, despite the fact that it can be applied at a finer granularity (i.e., faster) than DVFS, has the additional overhead of warming up both the cache and the pipeline of the target core. Moreover, none of these mechanisms affects leakage power. Leakage (or static power) is something that many studies do not take into consideration when dealing with temperature, but it cannot be ignored (since it depends quadratically on temperature). For current technologies (32nm and below), even with gate leakage under control by using high-k dielectrics, subthreshold leakage has a great impact in the total energy consumed by processors. Furthermore, leakage depends on temperature, so it is crucial to add a leakage-temperature loop to update leakage dissipation in real time depending on the core/structure's temperature.

Therefore, in order to accurately control peak temperature, which is of special interest in 3D-stacked processors as this integration technology exasperates thermal problems, a much tighter control is necessary to restraint the power dissipation of the different cores.

In Chapters 4 and 5 we proposed the use of a hybrid mechanism to match a predefined power budget [27]. This mechanism accurately matches a power budget and ensures minimal deviation from the target power and the corresponding temperature, by first using DVFS to lower the average power dissipation towards the power budget and then removing power spikes by using microarchitectural mechanisms (e.g., pipeline throttling, confidence estimation on branches, critical path prediction, etc).
In this chapter we make three major contributions. First, we analyze the effects of cycle-level accurate power control mechanisms to control peak temperature in 3D die-stacked processors. Based on this analysis we propose Token3D, a power balancing mechanism based on PTB that takes into account temperature and layout information when balancing power among cores and layers. Second, we analyze a wide range of floorplan configurations looking for the optimal temperature configuration taking into account both dynamic and leakage power (as well as the leakage-temperature loop). And third we include some specific power control mechanisms for vertical 3D floorplans. Experimental results show a reduction of the peak temperature of 2-26°C depending on the selected floorplan when including cycle-level power control mechanisms into the 3D die-stacked design. Summarizing, the main contributions of the present chapter are the following:

- Reducing the peak temperature through power control mechanisms:
  - Implementation and analysis of power balancing mechanisms on 3D die-stacked architectures to minimize hotspots.
  - Introduction of a new policy to balance power among cores, Token3D. This policy will use layout and temperature information to distribute the available power among the different cores and layers, giving more work to cool cores and cores close to edges than to internal cores.
- Temperature analysis of the main 3D design choices:
  - Analysis of different 3D floorplan designs using accurate area, power (both static and dynamic) and heatsink information.
  - Analysis of the effects of ROB resizing [75] on temperature for vertical designs.
  - Temperature analysis when using ALUs with different physical properties (energy-efficient vs. low latency ALUs) on the same layout.
  - Implementation and analysis of a hybrid floorplan design (vertical+horizontal).

The rest of this chapter is organized as follows. Section 6.2 provides some background on power-saving techniques for CMPs and 3D die-stacked multicores. Section 6.3 describes the proposed Token3D approach. Section 6.4 describes our simulation methodology and shows the main experimental results. Finally, Section 6.5 shows our concluding remarks.

6.2. Background and Related Work

In this section we will introduce the main power and thermal control mechanisms as well as an overview on 3D die-stacked processors along with the different floorplan design choices.

6.2.1. Power and Thermal Control in Microprocessors

6.2.1.1. Dynamic Thermal Management (DTM)

As mentioned before, temperature is the main drawback in 3D die-stacked designs. In 2001, Brooks and Martonosi [17] introduced Dynamic Thermal Management (DTM) mechanisms in microprocessors. In that work they explore performance trade-offs between different DTM mechanisms trying to tune up the thermal profile at runtime. Thread migration [88], fetch throttling [29], clock gating or distributed dynamic voltage scaling [77] are techniques that can be used by DTM mechanisms. For the thermal management of 3D die-stacked processors, most of the prior work has addressed design stage optimizations, such as thermal-aware floorplaning (as in [37]). In [102], the authors evaluate several policies for task migration and DVS specifically designed for 3D architectures. Something similar is done in [28], where the authors explore a wide range of different floorplan configurations using clock gating, DVFS and task migration to lower peak temperature.
However, both thread migration and DVFS-based approaches exhibit really low accuracy when matching a target power budget, and thus a high deviation from the target temperature. So the designers have two choices, either to increase the power constraint to ensure the target temperature or to use a more accurate way to match the desired (if needed) power budget and temperature. In order to do this we first need a way to measure power accurately, because up until now power was estimated by using performance counters, although the new Intel Sandy Bridge processors include some MSRs (machine specific registers) that can be used to retrieve power monitoring information from different processor structures. We will use Power-Tokens to deal with this problem.

6.2.1.2. Hybrid Power Control Approaches

In Chapter 4 we introduced a two-level approach that firstly applies DVFS as a coarse-grain approach to reduce power dissipation towards a predefined power budget, and secondly chooses between different microarchitectural techniques to remove the remaining and numerous power spikes [26]. The second-level mechanism depends on how far the processor is over the power budget in order to select the most appropriate microarchitectural technique. Winter *et al.* also proposed the use of a two-level approach that merges DVFS with thread migration to reduce temperature in SMT processors [93]. Their selected microarchitectural techniques try to reduce the power of functional units and the instruction window by means of profiling.

However, previous approaches failed to match the target power budget when considering the execution of parallel workloads in a CMP processor. Very recently, we have proposed Power Token Balancing (PTB) [27] as described in Chapter 5. This mechanism will balance the power between the different cores of a 2D CMP to ensure a given power constraint (or budget) with minimal energy and performance degradation. Based in power token accounting, this proposal uses a PTB load-balancer as a centralized structure that receives and sends power information (measured as power tokens) from cores under the power budget to cores over the power budget. Tokens are used as a currency to account for power, so it is important to note that they are neither sent nor received, cores just send the number of spare tokens. PTB will benefit from any power unbalance between cores. Note also that task migration mechanisms are orthogonal to PTB and can be applied together for further temperature reductions.

6.2.2. Towards the Third Dimension

While transistor switching speed has continued to improve by roughly a third with each new manufacturing process, the interconnection networks have comparatively slowed down in performance [69]. There have been improvements in interconnection networks, like switching from aluminum to copper or developing better insulating materials between layers to reduce parasitic capacitance [71]. However, the overall wire performance is getting worse with each new generation of transistors.

Latency of on-chip wires can be estimated as the product of their resistance and capacitance, namely, the RC delay. A wire's RC propagation delay depends quadratically on its length (i.e., a wire that is twice longer might have an RC delay four times larger). As process feature sizes shrink, the capacitance of wires slightly decreases. However, the cross-section of the wire is cut in half, which doubles the resistance, effectively doubling the propagation delay. This would triple the relative difference between global wire and transistor performance every process generation, which leads to a reduction on the chip area a signal can travel per clock cycle, as illustrated in Figure 6.1 [1].

One possible solution to mitigate this problem is the insertion of buffers and/or flip-flops to split a long wire into segments, boosting the signal, as it was done in the Pentium 4 design. Because of the quadratic relation between the length of a wire and it's delay, splitting a wire into two equal sub-segments reduces the total wire latency by half, although the buffer itself introduces a small delay. However, buffers and flip-flops dissipate additional power. Moreover, the number of buffers needed to build the interconnection network grows exponentially with each new generation of transistors, making this solution unfeasible as a long term solution.

Another commonly used method to reduce the interconnection network delay is to add additional metal layers to the interconnect stack [69], as shown in Figure 6.2. Metal layers are deposited one at a time on top of the silicon substrate. Thicker and shorter wires reduce interconnection delay. These highest quality wires are used to distribute the global clock and power to the functional unit blocks. The upper metal layers generally

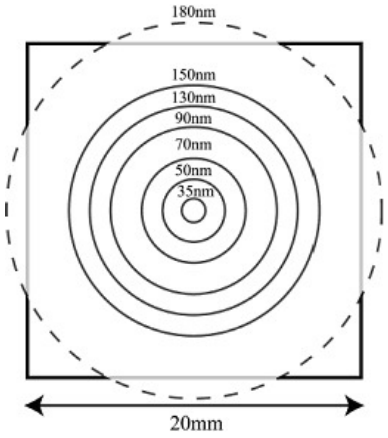


Figure 6.1: Range of a wire in a single clock cycle (Source: Real World Technologies).

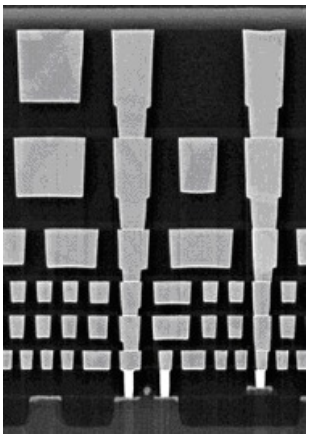


Figure 6.2: 130nm process generation with 6 layers of copper interconnect (Source: Intel).

keep their dimensions and spacing between inherited wires from the previous transistor generation. As a result, the latency of the global wires is kept constant between generations. However, the addition of new metal wires increases the total energy consumption, and more importantly, it requires additional processing steps on the wafers. The extra steps plus the consequent reduction on yields (or number of good die per wafer) due to the increased defect rate, increases the overall production costs. Moreover, the addition of more metal layers is still not enough to reduce the growing gap between transistor and global interconnect performance. The continuous shrinking on transistor's size is making chips to be unnecessarily larger to accommodate the global interconnect.

6.2.2.1. Building a 3D Die-Stacked Processor

As seen in the previous section, most modern circuits are already three dimensional in nature. There are various interconnection metal layers stacked over the silicon layer, yet the transistors remain in a planar configuration. 3D die-stacked cores propose the use of multiple layers of silicon substrate, each one containing transistors, arranged in a stacked configuration, one on top of the other. In these cores the interconnect wires not only run on top of each substrate as in conventional chips, but also vertically through layers [6], as shown in Figure 6.2.
As mentioned before, the main benefits of 3D designs include:

1. Increased packing density and smaller footprint. Cost reduction is a consequence of this, as fewer pins are needed per chip to communicate with nearby chips or with the motherboard, simplifying packaging.

2. Increased performance due to reduced interconnect delay.

3. Overal energy reduction due to the shorten in total wire length.

4. Open possibilities to build mixed-technology chips.

While all these advantages are significant, the three dimensional integration also has its drawbacks. As mentioned before, the overall energy consumption is reduced, however, power density increases in some parts of the 3D integrated circuit (especially in lower layers of the 3D stack). Without major investments in the design and simulation of the processor, the operating temperatures could reach unacceptable levels, decreasing the device's reliability and requiring expensive cooling solutions.
Summarizing, in order to build a 3D die-stacked processor we need to decide two things: how we build and put together the different layers and how we establish the communication between them. There are two main approaches to build the layers [91]: the *bottom-up* and the *top-down* approaches.
In the (*bottom-up*) approach, the front-end processing is repeated on a single wafer to create multiple silicon layers before building interconnects among them (like interconnect metal layers). An example of this approach is Multilayer Buried Structures (MLBS) technology [98][44].
The second approach (*top-down*) creates each layer of the 3D stack separately using conventional techniques, and then assembles them using wafer-bonding technologies [81]. There are several possible implementation technologies currently being considered:

- The face-to-face (F2F) organization. In particular, this organization provides the greatest die-to-die via density [14][30][72]. F2F bonding consists of depositing "stub" vias on to the top-level metal of each die as if another metal layer were to be implemented. The two dies are then placed face-to-face such that each stub from the first die is directly pressed against the corresponding stub on the second die.

- The face-to-back (F2B) organization. In this technology, die-to-die vias must be etched through the back-side (device-side) of one of the dies [31][81]. The etching process has less resolution than the "stub" union. As a result, the effective density of the vias in F2B process is lower than the F2F process. Furthermore, the etching process requires the die-to-die via to pass through the device layer, which may seriously disrupt the crystal structure of a strained-silicon process. The main advantage of F2B bonding is that it can be repeated for an arbitrary number dies to stack them together [78].

If we decide to bound at a wafer level (wafer-to-wafer bonding) we can obtain high production throughput. On the other hand, die-do-die techniques can provided finer layout alignment. After bonding the layers we need to establish communications between them. There are various vertical interconnect technologies that have been explored, including wire bonded, microbump, contactless (capacitive or inductive), and through-via vertical interconnect. A comparison in terms of vertical density and practical limits can be found in [97, 102]. However, the most promising vertical interconnect strategies involve through die vias, particularly through silicon vias, which promise the highest vertical interconnect density.

If we compare MLBS technology to wafer-bonding integration, the former requires more changes to the manufacture process steps than the later. Another key difference that can influence the building process is the size of the vertical vias which provide connections between the different layers. In wafer bonding, the dimension of the 3D vias is not expected to scale at the same rate as feature size because wafer-to-wafer alignment tolerance [31]. Current dimensions of 3D via sizes vary from $1\mu m$ by $1\mu m$ to $10\mu m$ by $10\mu m$ [9]. On the other hand, MLBS provides more flexibility in the inter-layer connection because it can potentially scale down with feature size due to the use of local wires for connection [98]. Availability of such technologies makes it possible to partition the cache at the granularity of individual cache cells [45]. However, wafer bonding requires fewer changes in the manufacturing process and is more popular in industry [68][53] than MLBS technology. Therefore, the integration approach we adopt in this study is the wafer-bonding technology.

Finally, the latency to drive a signal from one die another also affects how to partition a processor across multiple dies. For F2F stacking in a 65nm technology, the total length of the die-to-die via to connect the two die varies from $< 5\mu m$ to $30\mu m$ [30]. This resolution allows, for example, to split a 6T memory cell across two layers at a wordline or bitline level [79]. Moreover, in the same paper, Puttaswamy estimates the inter-layer communication latency to be in the order of one FO4. As current technologies allow to perform from 9 to 12 FO4 per cycle, the inter-layer communication delay is not critical. However, there are other papers that claim that inter-layer communication takes as long as an off-chip memory access [97].

6.2.2.2. 3D Bonding and Yield

The final 3D chip fabrication issue is manufacturing yield (number of good die per wafer) and testing. As mentioned before, the 3D stack can be assembled at the wafer scale, before testing [59], or at the die level (after testing).

Wafer-to-wafer bonding of many layers has a negative effect on yield, since the more wafers are stacked together, the more likely the whole chip stack is to be ruined by one defective layer. If we assume equal size layers, with a die yield of 80%, and we stack two of these layers together, the resulting yield will be at 64%. However, wafer to wafer bonding can have a positive effect on yield if designers use 3D integration to reduce the size of each individual die. Specifically, breaking a planar die into multiple smaller pieces will increase the yield of the smaller dice, since more candidates can fit in a given wafer. Because there are more candidate dice per wafer, the number of defects (which should be roughly constant), will effect a smaller percentage of the overall number of candidate dices. The benefits of additional stacks decreases and will eventually reduce yield because the attachment process is not perfect.

On the other hand, die-to-die bonding good quality stacks dice together, ensuring high yield levels before attaching the layers. The manufacturing throughput is reduced since individual dies are processed rather than complete wafers. However, the yields are higher

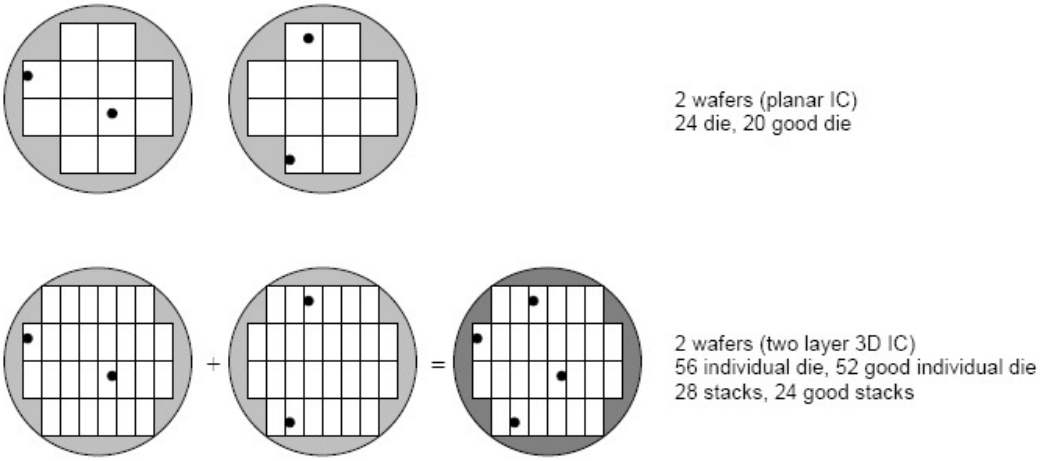


Figure 6.3: Yield impact of wafer to wafer bonding.

than wafer-to-wafer bonding. In the example in Figure 6.3, a two layer stack with 2 defects per wafer produced 24 out of 28 good stacks, for wafer-to-wafer bonding. In the same scenario, die-to-die bonding would have better yields, 26 out of 28 good stacks.

6.2.2.3. 3D Integration Technology

From the previously introduced technologies, wafer-to-wafer bonding appears to be the most promising approach [13] and there are many recent publications that have chosen this type of 3D stacking technology [49, 61, 63]. Therefore, this is the integration approach we are going to follow in this chapter.

Now there are multiple choices on how cores are distributed along the different layers, which are shown in Figure 6.4. We can clearly identify two trends; either build the cores vertical or horizontal. Horizontal distributions (a-c) are the most common choices in literature, as they are easier to implement and validate. Direct packing is the most straight-forward design. In this distribution, identical floorplans are stacked together to form the 3D processor. Mirror packing is a slight modification to direct packing, that turns half of the layers 180 degrees. This simple modification reduces temperature because the L2 of the even layers acts as a heatsink for the odd ones and vice versa. Finally, L2 packing interleaves core layers with L2 layers further reducing temperature, but increases the design complexity and testing. On the other hand, vertical designs (Figure 6.4-d), introduced by Puttaswamy *et al.* in [79], offer improved latency and power reduction compared to horizontal designs. However, they supposed an inter-layer communication latency to be in the order of one FO4, and current technologies can do 9-12 FO4 in one cycle. Therefore, in their proposal inter-layer communication could be done in less than one cycle while other papers claim that inter-layer communication takes as long as an off-chip memory access [97]. Furthermore, vertical designs require really accurate layer alignment to match a structure split in different layers, and that is far from the current technology status. However, as a possible future implementation of 3D die-stacked processors we also evaluate these floorplans in this chapter, and for

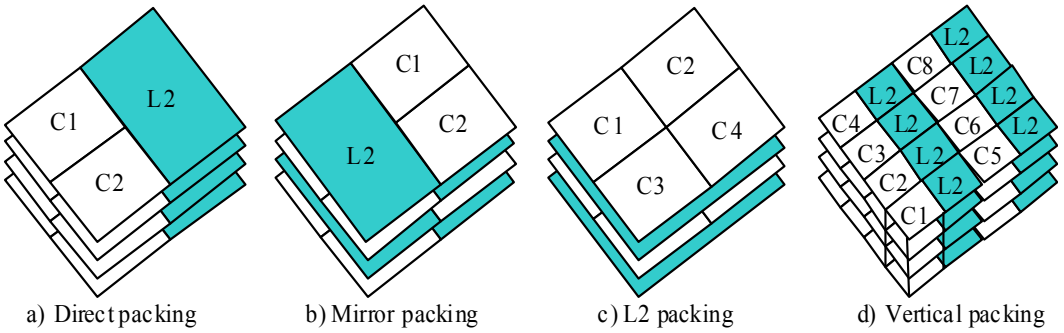


Figure 6.4: Core distribution along the layers.

comparative purposes, we also assume one FO4 interconnection delay for our evaluation of vertical designs (10μm length wires between layers).

6.3. Thermal Control in 3D Die-Stacked Processors

6.3.1. Token3D: Balancing Temperature on 3D Die-Staked Designs

As proposed in Chapter 5, Power Token Balancing (PTB) is a global balancing mechanism to restrain power dissipation up to a preset power budget [27]. One of the main goals of this chapter is to analyze the effects of the original PTB approach in 3D die-stacked architectures. We will also propose a novel policy, Token3D, aimed at distributing the power among cores and/or dies that are over their local power budget. Token3D will give priority to cooler cores, usually located close to the edges/surface of the 3D stack. By prioritizing those cores, Token3D balances not only power but also temperature, as cool cores will work more than the rest of cores, balancing the global CMP temperature. Once a cool core gets to a synchronization point or to a low computation phase (i.e., low IPC due to a misprediction event) it will naturally cool down again, acting like a heatsink to hotter cores located beneath it in the 3D stack.

6.3.2. Token3D Implementation Details

Token3D is a new policy on how PTB splits the available power tokens, given by cores under the power budget to the PTB load-balancer, among the cores that are over the power budget (details about power tokens and the PTB approach are covered in sections 2.2 and 5.3.4). Basically, Token3D will create N buckets, where N represents the amount of layers of our 3D die-stacked processor. Then the PTB load-balancer will place the coolest core in bucket one and will distribute the rest of the cores between the available buckets in increments of 5% in temperature. So, cores that have a difference between 0 and 5% in temperature with respect to the coolest core will be placed in the same bucket; cores between 5% and 10% will be placed on the next bucket; and so on until N. Note that this process does not need to be done at a cycle level, as temperature does not change so quickly. In our case, this process is performed every 100K-cycles. For example,

in a four layer 3D-stacked processor, if the coolest core has an average temperature of 70°C, bucket one will hold cores with temperatures between 70°C and 73.5°C, bucket two will hold cores with temperate between 73.5°C and 77°C, bucket three 77°C to 80.5°C and bucket four any core over 80.5°C.

Once we have identified the cores that are over the power budget (those that did not provide any tokens to the PTB load-balancer), the load balancer will distribute the power tokens between the active buckets (i.e., the buckets that have cores over the power budget) in an iterative way, giving extra tokens depending on the bucket the core is in. For a 4-layer design, the bucket that holds the hottest core will have a x1 multiplier on the number of received tokens, while the coolest bucket will have a x4 multiplier on the amount of received tokens. For example, if buckets 1, 2 and 3 are active (being 1 the one that holds the coolest cores), all the cores will receive one token, cores in buckets 2 and 1 will receive a second token and, finally, cores in bucket 1 will receive a third token. If there are any power tokens left, we repeat the process.

6.4. Experimental Results

In this section we will evaluate both the original PTB and the novel Token3D approaches as mechanisms to control temperature in a 3D die-stacked CMP. In addition, we will evaluate some specific optimizations for a vertical design that uses a custom floorplan where hotspot structures have been placed in the upper (cooler) layers whereas cooler structures are placed in lower layers. We will also analyze the different floorplan organizations in order to minimize peak temperature in the 3D die-stacked architecture. For our evaluation the selected power budget is 50% of the original power dissipation of the processor. We will use this restrictive power budget to force a high usage of the different power saving mechanisms.

Table 6.1: Evaluated benchmarks and input working sets.

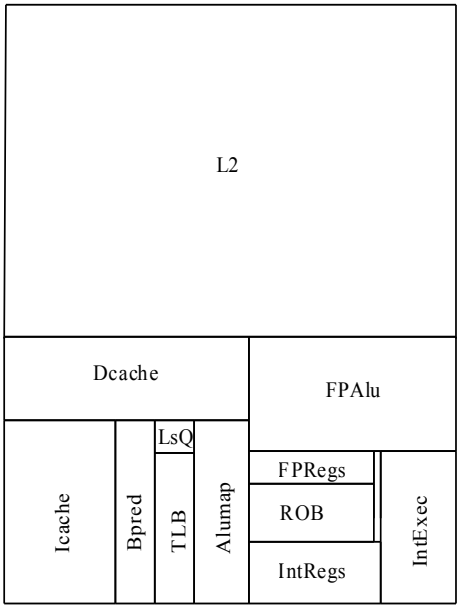
	Benchmark	*Size*	*Benchmark*	*Size*
SPLASH-2	Barnes	8192 bodies, 4 time steps	Raytrace	Teapot
	Cholesky	tk16.0	Water-NSQ	512 molecules, 4 time steps
	FFT	256K complex doubles	Water-SP	512 molecules, 4 time steps
	Ocean	258x258 ocean	Tomcatv	256 elements, 5 iterations
	Radix	1M keys, 1024 radix	Unstructured	Mesh.2K, 5 time steps
PARSEC	Blackscholes	simsmall	Swaptions	simsmall
	Fluidanimate	simsmall	x264	simsmall

6.4.1. Simulation Environment

For evaluating the proposed approaches we have used the Virtutech Simics platform extended with Wisconsin GEMS v2.1 [67]. As cited in Chapter 2, GEMS provides both detailed memory simulation through a module called Ruby and a cycle-level pipeline simulation through a module called Opal. We have extended both Opal and Ruby with all the studied mechanisms that will be explained next. The simulated system is a

Processor Core	
Process Technology:	32 nanometers
Frequency:	3000 Mhz
V_{DD}:	0.9 V
Instruction Window	128 RUU + 64 IW
Load Store Queue	64 Entries
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	6 Int Alu; 2 Int Mult
	4 FP Alu; 4 FP Mult
Branch Predictor:	16bit Gshare
Memory Hierarchy	
Coherence Prot.:	MOESI
Memory Latency:	300
L1 I-cache:	64KB, 2-way, 1 cycle lat.
L1 D-cache:	64KB, 2-way, 1 cycle lat.
L2 cache:	1MB/core, 4-way, unified
	12 cycle latency
TLB:	256 entries
Network Parameters	
Topology:	2D mesh
Link Latency:	4 cycles
Flit size:	4 bytes
Link Bandwidth:	1 flit / cycle

(a) Core Structures



(b) Core Floorplan

Figure 6.5: Core Configuration

homogeneous CMP consisting of a number of replicated cores connected by a switched 2D-mesh direct network. Table 6.5a shows the most relevant parameters of the simulated system. Power scaling factors for a 32nm technology were obtained from McPAT [56]. To evaluate the performance and power dissipation of the different mechanisms we used scientific applications from the SPLASH-2 benchmark suite in addition to some PARSEC applications (the ones that finished execution in less than 5 days in our cluster). Results have been extracted from the parallel phase of each benchmark. Benchmark sizes are specified in Table 6.1.

3D thermal modeling can be accomplished using an automated model that forms the RC circuit for given grid dimensions. For this work we have ported HotSpot 5.0 [88] thermal models into Opal and have built our tiled CMP by replicating N times our customized floorplan, where N is the number of cores. Figure 6.5b shows the base floorplan design we have chosen. This floorplan was obtained from Hotfloorplaner (provided by the Hotspot 5.0). Our resulting CMP will be composed of a varying number of these cores (from 2 to 16). As cited before, we will assume an interconnection delay between layers of one FO4 (10μm length wires, as in [79]).

Moreover, thermal hotspots increase cooling costs and have a negative impact on reliability and performance. The significant increase in cooling costs requires designs for temperature margins lower than the worst-case. Leakage power is exponentially dependent on temperature, and an incremental feedback loop exists between temperature and leakage, which may turn small structures into hotspots and potentially damage the circuit. High temperatures also adversely affect performance, as the effective operating

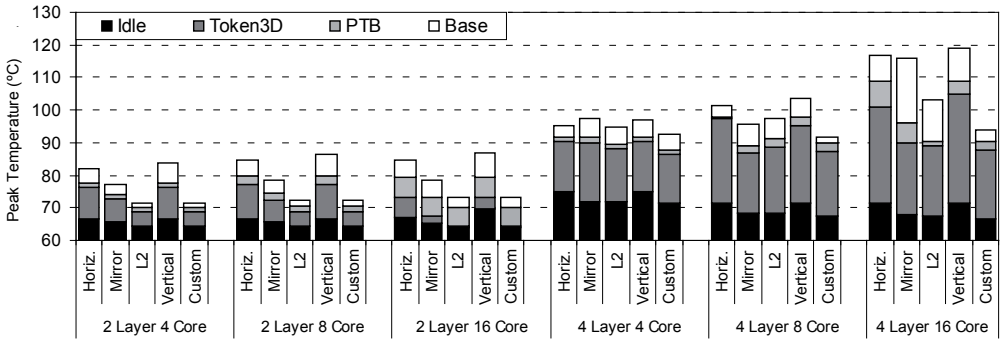


Figure 6.6: Peak temperature for PTB, Token3D and the base case for different floorplans and core configurations.

speed of transistors decreases as they heat up. In this chapter we model both leakage (through McPAT) and the leakage/temperature loop in Opal, so leakage will be updated on every Hotspot exploration window (10K cycles). Leakage power is translated into power tokens and updated according to the formula:

$$L_{new} = L_{Base} * e^{Leak_{\beta}*(T_{Current}-T_{Base})}$$

Where $Leak_{\beta}$ depends on technology scaling factor and is provided by HotSpot 5.0, L_{new} is the updated leakage, L_{Base} is the base leakage (obtained using McPAT), $T_{Current}$ is the current temperature and T_{Base} is the base temperature. Once leakage is updated, it is translated back to power tokens. Another important parameter is the cooling system. The regular thermal resistance of a cooling system ranges from 0.25 K/W for the all-copper fan model at the highest speed setting (very good), to 0.33 K/W for the copper/aluminum variety at the lowest setting. In this work we model a real-world Zalman CNPS7700-Cu heatsink with 0.25 K/W thermal resistance and an area of 3.268 cm2 (136mm side).

6.4.2. Effects of Token3D on Peak Temperature

Figure 6.6 shows the peak temperature for different floorplan configurations and a varying number of cores (from 4 to 16) using stacked bars. The reported "idle" temperature corresponds to the average idle temperature of the cores[1]. The Token3D bar corresponds to the temperature increase from "idle" temperature reached when executing the studied benchmarks running the new Token3D policy. The PTB bar corresponds to the original Power Token Balancing approach as proposed in the previous chapter. Finally, the base bar corresponds to the baseline configuration where no power/temperature control mechanisms are used. The studied floorplans are those described in Section 6.2.2.3: Horizontal (Figure 6.4.a), Mirror (Figure 6.4.b), L2 (Figure 6.4.c), Vertical (Figure 6.4.d) and Custom. As cited before, this last floorplan corresponds to a new configuration that

[1]We define "idle" temperature as the temperature of the whole CMP in idle state (i.e., only the operating system is running).

places hotspots into upper layers of the 3D stack, giving more chances for them to cool down, and will be further discussed later in the next subsection. In Figure 6.6 we can clearly see that both L2 and Custom are the best designs to reduce peak temperature of the processor. This is due to the fact that both designs place the L2 in lower layers, and, as it can be seen in Figure 6.7, the L2 is the coolest structure within a core, even though we are accounting for leakage to calculate temperature. This placement leaves hotspots close to the surface and hot structures can cool down easily. We can also see that even a simple change in the floorplan such as mirroring each core between layers gives substantial peak temperature reduction (5-6°C) compared with the horizontal design.

When considering the vertical design we can observe a higher peak temperature than the horizontal one. This vertical design was introduced in [79] by Puttaswamy *et al.* along with a dynamic power saving mechanism, *Thermal Herding*, that disables layers at runtime, depending on the number of bits used by the different instructions. This vertical design assumes each structure is vertically implemented across all layers. In our evaluation of this vertical design the area occupied by each structure and its power dissipation is divided by the number of available layers, but we do not disable any layer, to isolate our proposed power control mechanisms from the benefits obtained by *Thermal Herding*. For instance, in a 4-layer vertical design the implemented thermal model calculates the temperature of a structure in layer i by considering one fourth of its original power and area, however, the fraction of that structure is stacked on top of another equal portion of the same structure, with all portions simultaneously accessed, and therefore, increasing temperature. Note, however, that the use of *Thermal Herding* and its ability to disable unused layers for the vertical design is orthogonal to the use of our proposed PTB and Token3D approaches.

When it comes to the studied power control mechanisms both the original PTB and Token3D are able to reduce peak temperature by 2-26°C depending on the floorplan configuration. Token3D is always 1-3% better than the original PTB balancing mechanism. We must also note that, as we get closer to the idle temperature, any temperature reduction comes at a higher performance degradation.

Figure 6.7-top shows a more detailed analysis on the effects of both PTB and Token3D in the peak temperature of the different core structures. We selected the most typical configuration for 3D die-stacked cores (Mirror, Figure 6.4.b) and a 4-layer 16-core CMP for this per-structure temperature analysis. As cited before, PTB and Token3D are evaluated with a preset power budget of 50% of the original average power dissipation. For comparison purposes we also evaluate DVFS trying to match the same target power budget of 50%. Figure 6.7-top helps us to locate our design hotspots (I-cache, TLB, Branch Predictor, Load Store Queue) and see how both cycle-level power control mechanisms are able to reduce peak temperature by 20-36%. For example, the I-cache goes from 150°C down to 110°C, 30°C less than DVFS. We can also see that, on average for this selected design, Token3D is 5-6°C better than regular PTB. It is also important to note that our cycle-level mechanisms are able to reduce all hotspots peak temperature and put them

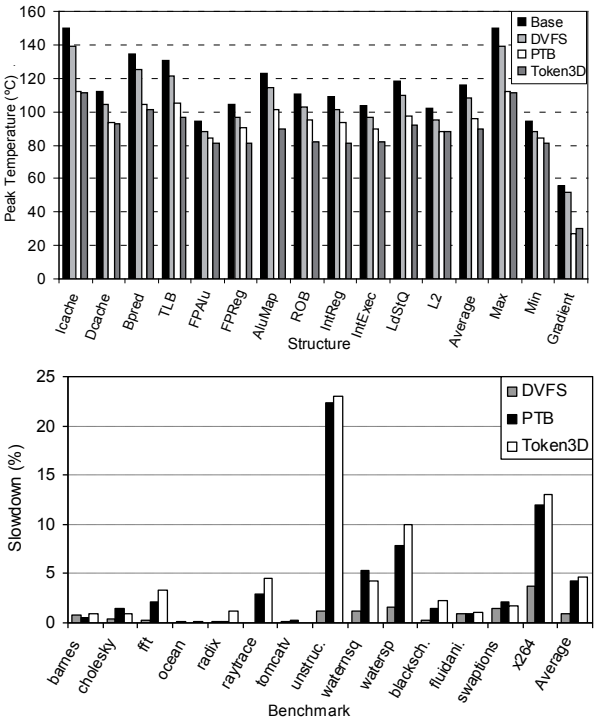


Figure 6.7: Per structure peak temperature (top) and performance (bottom) of a 4-layer 16-core CMP using the mirror floorplan.

close to the average core temperature. This last result is especially interesting on 3D architectures, as they exacerbate thermal problems and a much tighter power control is necessary. This is the benefit we expected from the highly accurate power budget matching provided by our mechanisms, that ensures minimal deviation from the target power budget and, therefore, temperature. In Figure 6.7-top we also show the spatial gradient (temperature difference between the hottest and coolest structure of the core). Reducing spatial gradients is important because they can cause clock skew and impact circuit delay [3]. In particular, both PTB and Token3D are able to reduce this gradient by more than 50%, from 50°C to 22°C. In terms of performance degradation (Figure 6.7-bottom), regular PTB behaves slightly better than Token3D, as power is equally divided between all cores and they can get to the next synchronization point more evenly, while Token3D will unbalance cores and make them wait at the synchronization point more time.

6.4.3. Further Temperature Optimizations

In addition to the introduction of Token3D we also wanted to perform some optimizations for the vertical 3D die-stacked layout. More specifically, we will analyze the effects on peak temperature of MLP-based instruction window (ROB) resizing [75] and ALU selection based on instruction criticality (from ALUs placed on different layers) while varying the number of cores. We selected these structures because they are well known hotspots, but have not being well optimized in the 3D die-stacked field.

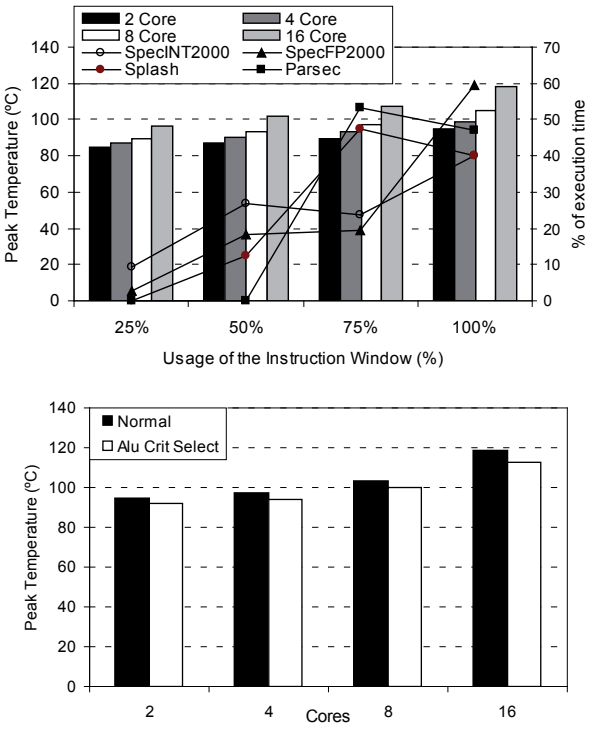


Figure 6.8: Peak temperature of the instruction window (top) and ALUs (bottom) for a varying number of cores.

Figure 6.8 shows the effects on peak temperature of different instruction window (IW) sizes for a 4-layer vertical core design (Figure 6.4.d). Each core has a 128-entry IW that is equally distributed across the different layers in the vertical design, (as we are working with 4 layers, each layer has 32 entries). Entries are disabled by layer, so we disable entries in groups of 32. In order to decide the current IW size we use a dynamic MLP-based IW resizing mechanism as proposed in [75]. In Figure 6.8-top, we also show the distribution of the average IW size for different benchmark suites (represented with lines). This average window size highly varies between benchmarks, as memory-bound benchmarks require many IW entries to bring more data simultaneously from memory, while CPU-bound applications do not need that many entries. Therefore, instead of just showing the peak temperature reduction of the average benchmarks (bars in Figure 6.8-top) we decided to do a design exploration of the peak temperature based on the IW size. For example, Parsec benchmarks use 0% of the time 25% and 50% of the IW, 55% of the time use a 75% of the IW (12°C reduction) and 45% of the time use the whole IW. This benchmark suite barely gets any temperature benefit from dynamic IW resizing. On the other hand SPECINT2000 benckmarks spend 20% of the time using only half of the IW, and 10% of the time using 25% of the IW. In this benchmark suite we can obtain temperature reductions of 20-22°C for a 4-layer 16-core CMP by dynamically resizing the IW. Moreover, as the usage of the IW dependes on the benchmark characteristics there are some CPU-bound benchmarks that spend most of the time using only 25-50% of the IW. We can obtain huge overal temperature reductions in those cases. Note also

that the IW is just one of the multiple hotspots of the core, and we cannot expect greater temperature reductions as the area of the IW is only a 2.5% of the total area of the core. When working with vertical designs we can think of having different types of ALUs placed into different layers: fast (and hot) ALUs placed on upper layers for critical instructions plus slower power-saving ALUs placed in lower layers. As our core design includes an instruction criticality predictor we can use this information to decide where we want to send a specific instruction. Figure 6.8-bottom shows the effect on peak core temperature having half of the ALUs placed in layers 2-3 (upper layers) and half of the ALUs placed in layers 0-1 (lower layers). The ALUs in the lower layers dissipate 25% of the original power dissipation but are also 25% slower than the original ALUs. Results show a peak temperature reduction of 3-5°C for this configuration. This small temperature reduction is due to the fact that in our core design ALUs are not a hotspot (as it can be seen in Figure 6.7-top: IntExec and FPAlu structures) for the studied benchmarks, and thus, their temperature contribution has almost no impact on the average peak temperature of the processor. However, we can expect better results with other CPU-bound applications where ALUs really become a hotspot. There are other possible policies to decide where to send an ALU instructions, like temperature or layout information about the available ALUs. We implemented these policies in the simulator but didn't had time to do a deep study of the different possibilities.

Finally, we want to introduce a custom floorplan design that merges both vertical and horizontal designs in order to minimize hotspots. This design is an extension of the L2 design (Figure 6.4.c) for a 4-layer core. Based on the information provided by Figure 6.7-top we can separate cool from hot structures and place them in different layers. Hot structures are placed in the top layer (Bpred, Icache, Alumap, TLB, LdStQ, IntReg and ROB), which is the closest to the heatsink, and it's easier to cool down. The second layer consists of the rest of structures except the L2. We placed the Dcache under the Icache as the Icache is our hottests structure and the Dcache was below the average core temperature. Finally, the last two layers hold the L2 cache and memory controllers. This custom design has the additional advantage of reducing inter-layer communication when bringing data from memory, as memory controllers and the L2 are placed close to the socket. As we can see in Figure 6.6 (last bar on each group), this new design is able to reduce peak temperature by almost 12°C over the best horizontal design for a 4-layer 16-core processor. It is clear that intra-layer (global) floorplaning has thermal advantages over stacking previous layout designs, and it may be necessary if we want to move to 3D die-stacked microprocessors. However, this floorplan optimization requires assembly before in-deep testing, and that may increase production costs.

6.5. Conclusions

3D die-stacked integration offers a great promise to increase scalability of CMPs by reducing both bandwidth and communication latency problems. However, the increase on

core density leads to an increase in temperature and hotspots in these designs. Moreover, as building process scales down below 32nm, leakage becomes an important source of power dissipation and, as it increases exponentially with temperature, causes a power/temperature loop that affects to 3D die-stacked processors negatively. To control temperature, regular DTM mechanisms detect overheating in any of the temperature sensors and trigger a power control mechanism to limit power dissipation and cool the processor down. However, neither DVFS nor task migration (the most frequently used mechanisms) offer accurate ways to match a desired target power budget.

Power tokens and Power Token Balancing (PTB) were introduced in Chapters 2, 4 and 5 as an accurate way to account for power and match a power constraint with minimal performance degradation by balancing power among the different cores of a 2D CMP. In this chapter we evaluate these mechanisms in a new design scenario, 3D die-stacked processors. In this scenario PTB is able to reduce average peak temperature by 2-20^{o}C depending on the selected floorplan. For specific hotspot structures (i.e., instruction cache) PTB can reduce peak temperature by almost 40% in a 4-layer 16-core CMP. In addition, we have proposed Token3D, a novel policy that takes into account temperature and layout information when balancing power, giving priority to cool cores over hot ones. This new policy enhances PTB by providing an additional 3% temperature reduction over the original PTB approach. Also note that task migration is orthogonal to PTB and can be applied simultaneously for further temperature reductions.

To conclude this work we have also extended 3D die-stacked vertical designs with additional power control mechanisms. First, we enabled instruction window resizing based on MLP. CPU-intensive applications are highly dependent on cache, but do not show performance degradation if the instruction window is reduced. On the other hand, memory-intensive applications require big instruction windows to locate loads and stores and take advantage of MLP. Based on these properties we extended previous vertical designs with adaptive instruction window resizing. Second, we split ALUs in different groups, low latency and high latency ALUs. Low latency ALUs dissipate more power and should be placed in upper layers of the 3D design, on the other hand, high latency ALUs are more energy-friendly and can be placed in lower layers of the 3D stack, lowering the chances of becoming a potential hotspot. An instruction criticality predictor was used to decide where an instruction should be placed, either in a fast but expensive unit or in a slower but energy-efficient unit.

Finally, we explored a custom 3D design that merges both vertical and horizontal designs trying to minimize hotspots. In this design hot processor structures are placed in upper layers while cool structures are placed in lower layers. This design is able to reduce peak temperature by an additional 10% over the best horizontal design and 85% over the vertical design.

Chapter 7

Conclusions and Future Ways

Summary: In this chapter we discuss the conclusions to our work and outline some future research lines that can be followed in the energy-efficient processor design field.

7.1. Conclusions

Power dissipation, energy consumption and thermal output (known as the "Power Wall") have been, for the past years, the key limiting factors in microprocessor design. With the megahertz race over, engineers have focused on increasing the available number of processing cores on the same die. Nowadays, chip multiprocessors (CMPs) are the new standard design for a wide range of microprocessors: mobile devices (in the near future almost every smartphone will be governed by a multicore CPU), desktop computers, laptop, servers, GPUs, APUs, etc. All these microprocessors face constant thermal and power related problems during their everyday use.

We can distinguish two components in power dissipation, dynamic and static power dissipation. Dynamic power dissipation is proportional to usage (every time we access a structure), due to the constant charge and discharge of transistors. On the other hand, static power dissipation (or leakage), is derived from gate leakage and subthreshold leakage currents that flow even when a transistor is not in use. As process technology advances toward deep submicron, the static power component becomes a serious problem, especially for large on-chip array structures such as caches or prediction tables. For current technologies (under 32nm), even with gate leakage under control by using high-k dielectrics, subthreshold leakage has a great impact in the total power dissipated by processors. Moreover, leakage depends on temperature as well, so it is crucial to consider the leakage-temperature loop when measuring leakage.

Perhaps the first question we should address is: how do we measure power? Up until recently (Intel Sandy Bridge processors) power measurements were made as mere approximations based on performance counters done over periods of thousands of cycles.

In this Thesis we wanted to work with microarchitectural techniques that work at a very fine granularity (from several to a few tens of cycles) so we needed some way to estimate power at this fine granularity. The first major contribution of this Thesis were the *Power Tokens*. With *Power Tokens* we can easily estimate the total power dissipated by an instruction at commit stage by adding the base power dissipation of the instruction (i.e., all regular accesses to structures done by that instruction) plus a dynamic component that depends on the time it spends on the pipeline. The total power dissipated by the processor in a given cycle can be thus estimated as the addition of the individual power tokens from all the instructions inside the pipeline.

Using the power tokens we have studied different power saving mechanisms to reduce energy consumption in microprocessors, both at the circuit and microarchitecture levels. DVFS is a circuit level mechanism that uses the relation between dynamic power and both voltage and frequency ($P_d \approx V_{DD}^2 \cdot f$) to save power. The impact on performance for sequential or multi-programmed applications comes from the frequency reduction performed by DVFS. Performance impact can be lower in memory-intensive applications because we only modify the processor frequency but not the memory frequency. Therefore, offchip memory latency is reduced as we lower the processor frequency. We have also evaluated some microarchitectural techniques in terms of power and performance such as critical path prediction, branch confidence estimation and pipeline throttling. While DVFS is quite unstable in terms of power, all the microarchitectural mechanisms reduce power dissipation smoothly (without major power spikes). However, the performance impact of these mechanisms is higher. In addition, to study static power saving mechanisms we performed a case study to reduce the average leakage energy of traditional value predictors with negligible impact on neither prediction accuracy nor processor performance. We have evaluated both state and non-state preserving techniques for reducing leakage power in value predictors including Static Value Prediction Decay (SVPD), Adaptive Value Prediction Decay (AVPD) and a drowsy approach.

We have mainly focused on the dynamic component of power, as nowadays it is the main source of power consumption in microprocessors. However, as mentioned previously, leakage power is growing in importance (in our processor design peak leakage power was measured to be a 30% of the peak dynamic power dissipation) so it is also important to account for leakage power and implement leakage power saving mechanisms to further reduce energy consumption and temperature.

There is no discussion about the importance of power dissipation in current designs, but in this Thesis we wanted to go a step further. It is well known that general purpose microprocessors can be used in several kinds of gadgets that usually have different power requirements. In some scenarios being able to define a maximum power budget for the processor can help the reuse of previous designs in new devices. This can be especially useful if our only options are either to switch-off the device or to reduce the power consumption to match the power constraint. Note that, in many cases, the shut-off option is not even viable (e.g., for critical systems). We cannot design the thermal

envelopment of a processor for the worst case scenario, because production price highly increases and the processor barely ever reaches its peak temperature. However, we can set a power budget to the processor, limiting its power and temperature.

When we analyzed the different power saving mechanisms trying to match a predefined power budget we discovered that there was still a gap between the reduction on cycles over the power budget and the perfect power budget matching. In addition, simple power mechanisms are triggered once the power dissipation goes over the budget. We can optimize the way we enable the different mechanisms by using some historic information. We propose two adaptive techniques: Power Token Throttling (PTT) and Basic Block Level Manager (BBLM). PTT is a token-based approach that keeps track of the current power being consumed by the processor, measured as tokens, and stalls fetch if the next instruction to be executed will make the processor go over the power budget. This is a very aggressive mechanism that obtains an accurate power budget matching but has a huge performance degradation. On the other hand, BBLM uses basic block power consumption history (translated into power-tokens) in order to determine the best power saving technique for the current and near-future processor power dissipation. BBLM optimizes the use of microarchitectural techniques to minimize performance impact while removing most of the numerous power spikes. However, these mechanisms by themselves are not enough to match restrictive power budgets because of their high performance degradation.

For that purpose we have proposed a two-level approach that combines both microarchitectural techniques and DVFS to take advantage of their best qualities. DVFS acts as a coarse-grain technique to lower the average power dissipation while microarchitectural techniques remove all the power spikes efficiently. The two-level approach (BBLM + DVFS) is able to beat DVFS in both energy efficiency (up to 11% more energy efficient) and area exceeded over the power budget (up to 6 times less area).

The next logical step was to move all these mechanisms to a chip multiprocessor (CMP) scenario. When analyzing CMPs running parallel workloads, previously proposed power managing mechanisms fail to accurately adapt to temporary power constraints, due to thread dependences and synchronization points. Moreover, power saved just from spin-locks is not enough to match an aggressive power budget because it is too local and too low. The performance impact of DVFS in CMPs running parallel workloads can vary depending on how we apply DVFS. If we apply DVFS at a core level and delay a critical thread then we will observe a performance degradation on the overall application because of synchronization points. If we are able to balance the frequency of the different execution threads we can save power without any performance impact. On the other hand, if we chose to use DVFS at a chip level, instead of at a core level, the CMP will behave as the sequential case, with lower performance impact in memory intensive applications. A global control mechanism is needed to match these design peculiarities.

To overcome these limitations we proposed *Power Token Balancing* (PTB). PTB dynamically balances power among the different cores to ensure that the whole processor

accurately matches a predefined and global power budget. Our proposed mechanism accounts for unused power from cores that are under the power budget (translated into power-tokens) and passes that power to cores over the power budget, hence, not having to slow them down to match their local power budget constraint. PTB is a fine-grain mechanism that ensures maximum accuracy with minimal standard deviation from the power budget, which is crucial if you want to optimize packaging costs or to increase the number of cores in a CMP with the same TDP. However, the accuracy constraint can be relaxed in order for PTB to be more energy-efficient by means of applying the power-saving mechanisms in a less restrictive way, and therefore, not affecting performance too much.

As a side effect of this accurate power budget matching, the use of PTB provides another interesting benefit: a more stable temperature over execution time. For the studied benchmarks we can obtain a 27-30% peak and average temperature reduction, that also applies to the individual structures. PTB is also able to balance temperature between cores, reducing the peak temperature of the hottest core making it equal to the temperature of the coldest core in the base CMP design. This temperature reduction not only reduces leakage power but also increases reliability and can result in reduced packing costs.

In the process scaling, interconnects have not followed the same trend as transistors, becoming a limiting factor in both performance and power. One intuitive solution to reduce the wirelength of the interconnection network is to stack structures on top of each other, instead of using a traditional planar distribution. 3D die-stacked integration offers a great promise to increase scalability of CMPs by reducing both bandwidth and communication latency problems. However, the increase in core density leads to an increase in temperature and hotspots in these designs. To control temperature, regular DTM mechanisms detect overheating in any of the temperature sensors and trigger a power control mechanism to limit power consumption and cool the processor down. However, neither DVFS nor task migration (the most frequently used mechanisms) offer accurate ways to match this target power budget. We decided to evaluate our proposed mechanisms in a new design scenario, 3D die-stacked processors.

In this scenario PTB is able to reduce average peak temperature by 2-20°C depending on the selected floorplan. For specific hotspot structures (i.e., instruction cache) PTB can reduce peak temperature by almost 40% in a 4-layer 16-core CMP. In addition, we have proposed Token3D, a policy that takes into account temperature and layout information when balancing power, giving priority to cool cores over hot ones. This new policy enhances PTB by providing an additional 3% temperature reduction over the original PTB approach.

To conclude this Thesis we have also extended 3D die-stacked vertical designs with additional power control mechanisms. First, we enabled instruction window resizing based on MLP. CPU-intensive applications are highly dependent on cache, but do not show performance degradation if the instruction window is reduced. On the other hand,

memory-intensive applications require big instruction windows to locate loads and stores and take advantage of MLP. Based on these properties we extended previous vertical designs with adaptive instruction window resizing. Second, we split ALUs in two different groups: low latency and high latency ALUs. Low latency ALUs consume more power and should be placed in upper layers of the 3D design, on the other hand, high latency ALUs are more energy-friendly and can be placed in lower layers of the 3D stack, lowering the chances of becoming a potential hotspot. An instruction criticality predictor was used to decide where an instruction should be placed, either in a fast but expensive unit or in a slower but energy-efficient unit. Finally, we also explored a custom 3D design that merges both vertical and horizontal designs trying to minimize hotspots. In this design hot processor structures are placed in upper layers while cool structures are placed in lower layers. This design is able to reduce peak temperature by an additional 10% over the best horizontal design and 85% over the vertical design.

7.2. Future Work

The results presented in this Thesis open a number of interesting research topics that can be further developed in the future. The most interesting are detailed as follows:

- The use of power tokens as a spinlock detector. One of the indirect benefits of using power tokens is the potential ability to detect *spinning* or *busy waiting* of the cores inside a CMP. As we discussed in Section 5.3.3, when a core enters a spinning state, the dynamic power has an initial power peak due to useful computation. If the spinning state lasts enough, the pipeline empties and power goes down and stabilizes to an amount that is usually under the budget. We can assume then that the core is spinning. If we could accurately identify these spinning states we could increase our power savings by stalling the spinning core and giving all its power tokens to the PTB load-balancer.

- The use of the Nitro approach to speed up cores to get earlier to a barrier. As mentioned in Section 5.3.5.1, there are times when cores receive more power tokens that they can use. Under these conditions we could overclock the cores that receive tokens that they cannot use in order to reach to the next synchronization point faster, and thus reduce global execution time. The overclocking should only be done as long as we have power tokens left from the PTB load-balancer to prevent overheating because of exceeding our TDP.

- Better mechanisms to balance power in 3D Cores. We did many modifications to the GEMS-Opal simulator to model 3D die-stacked processors that allow us to vertically and horizontally split all the structures inside the core (even simultaneously). However, we didn't had time to do an in depth exploration of the different combinations of vertical and horizontal designs to optimize energy and temperature. We believe there is room for improvement in 3D floorplan designs and 3D

structure optimizations to further reduce temperature in these chips.

- Power in Embedded / Low-Power devices. During the past five years we have focused on superscalar out-of-order single-core or CMP processors, but never took a look at in-order / low-power processors to check if our proposed mechanisms are feasible in that scenario.
- Power in GPUs. In the past few years GPUs have gained much importance in both the academia and the global market. However, most of the work done with GPUs is performance-oriented, and not power-oriented. We believe GPUs are an open field of possibilities and would like to focus on this area in the near future.

Bibliography

[1] 3d integration: A revolution in design. website: http://www.realworldtech.com/page.cfm?articleid=rwt050207213241.

[2] Intel's stacked chip scale packaging. website: http://www.intel.com/research/silicon/mobilepackaging.htm.

[3] A. H. Ajami, K. Banerjee, and M. Pedram. Modeling and analysis of nonuniform substrate temperature effects on global ulsi interconnects. 24(6):849–861, 2005.

[4] J. L. Aragon, J. Gonzalez, and A. Gonzalez. Power-aware control speculation through selective throttling. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, pages 103–112, 2003.

[5] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35:59–67, February 2002.

[6] J. Baliga. Chips go vertical. *IEEE Spectr.*, 41:43–47, March 2004.

[7] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat. 3-d ics: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. In *Proceedings of the IEEE*, pages 602–633, 2001.

[8] A. Baniasadi and A. Moshovos. Instruction flow-based front-end throttling for power-aware high-performance processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 16–21, 2001.

[9] K. Bernstein. Introduction to 3d integration. In *International Solid State Circuits Conference Tutorial*, 2006.

[10] R. Bhargava and L. K. John. Latency and energy aware value prediction for high-frequency processors. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, pages 45–56, New York, NY, USA, 2002. ACM.

[11] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 290–301, New York, NY, USA, 2009. ACM.

[12] C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[13] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *Proceedings of the 39th Annual International Symposium on Microarchitecture*, pages 469–479, 2006.

[14] B. Black, D. W. Nelson, C. Webb, and N. Samra. 3d processing technology and its impact on ia32 microprocessors. In *Proceedings of the IEEE International Conference on Computer Design*, pages 316–318, Washington, DC, USA, 2004. IEEE Computer Society.

[15] M. Bohr, R. Chau, T. Ghani, and K. Mistry. The high-k solution. *IEEE Spectrum*, 44(10):29–35, Oct. 2007.

[16] S. Borkar. Design challenges of technology scaling. 19(4):23–29, 1999.

[17] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pages 171–182, 2001.

[18] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, 2000.

[19] J. A. Butts and G. S. Sohi. A static power model for architects. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 191–201, 2000.

[20] Q. Cai, J. Gonzalez, R. Rakvic, G. Magklis, P. Chaparro, and A. Gonzalez. Meeting points: Using thread criticality to adapt multicore hardware to parallel regions. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 240–249, 2008.

[21] B. Calder, G. Reinman, and D. M. Tullsen. Selective value prediction. In *Proceedings of the 26th International Symposium on Computer Architecture*, pages 64–74, 1999.

[22] J. Casmira and D. Grunwald. Dynamic instruction scheduling slack. In *Proceedings of the KoolChips Workshop*, 2000.

[23] J. M. Cebrian, J. L. Aragon, and J. M. Garcia. Leakage energy reduction in value predictors through static decay. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, pages 1–7, 2007.

[24] J. M. Cebrian, J. L. Aragon, J. M. Garcia, and S. Kaxiras. Adaptive vp decay: Making value predictors leakage-efficient designs for high performance processors. In *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, pages 113–122, New York, NY, USA, 2007. ACM.

[25] J. M. Cebrian, J. L. Aragon, J. M. Garcia, and S. Kaxiras. Leakage-efficient design of value predictors through state and non-state preserving techniques. *J. Supercomputing*, 55:28–50, January 2011.

[26] J. M. Cebrian, J. L. Aragon, J. M. Garcia, P. Petoumenos, and S. Kaxiras. Efficient microarchitecture policies for accurately adapting to power constraints. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, pages 1–12, 2009.

[27] J. M. Cebrian, J. L. Aragon, and S. Kaxiras. Power token balancing: Adapting cmps to power constraints for parallel multithreaded workloads. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, 2011.

[28] A. K. Coskun, J. L. Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici. Dynamic thermal management in 3d multicore architectures. In *Proceedings of the International Design, Automation & Test in Europe Conference & Exhibition*, pages 1410–1415, 2009.

[29] A. K. Coskun, T. T. Rosing, K. A. Whisnant, and K. C. Gross. Static and dynamic temperature-aware scheduling for multiprocessor socs. 16(9):1127–1140, 2008.

[30] S. Das, A. Chandrakasan, and R. Reif. Three-dimensional integrated circuits: Performance, design methodology, and cad tools. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, ISVLSI '03, pages 13–, Washington, DC, USA, 2003. IEEE Computer Society.

[31] S. Das, A. Fan, K.-N. Chen, C. S. Tan, N. Checka, and R. Reif. Technology, performance and computer-aided design of three-dimensional integrated circuits. In *Proceedings of the 2004 International Symposium on Physical Design*, ISPD '04, pages 108–115, New York, NY, USA, 2004. ACM.

[32] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the 33rd International Symposium on Computer Architecture*, pages 78–88, 2006.

[33] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 148–157, 2002.

[34] M. J. Flynn and P. Hung. Microprocessor design issues: Thoughts on the road ahead. 25(3):16–31, 2005.

[35] F. Gabbay and A. Meldenson. Speculative execution based on value prediction. Technical report, EE Department TR 1080, Technion - Israel Institue of Technology, 1996.

[36] B. Goeman, H. Vandierendonck, and K. de Bosschere. Differential fcm: Increasing value prediction accuracy by improving table usage efficiency. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pages 207–216, 2001.

[37] M. B. Healy, M. Vittes, M. Ekpanyapong, C. S. Ballapuram, S. K. Lim, H.-H. S. Lee, and G. H. Loh. Multiobjective microarchitectural floorplanning for 2-d and 3-d ics. *IEEE Transactions on CAD of Integrated Circuits and Systems*, pages 38–52, 2007.

[38] J. L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33:28–35, July 2000.

[39] Z. Hu, P. Juang, K. Skadron, D. Clark, and M. Martonosi. Applying decay strategies to branch predictors for leakage energy savings. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 442–445, 2002.

[40] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, 2006.

[41] E. Jacobsen, E. Rotenberg, and J. E. Smith. Assigning confidence to conditional branch predictions. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 142–152, 1996.

[42] P. Juang, P. Diodato, S. Kaxiras, K. Skadron, Z. Hu, M. Martonosi, and D. W. Clark. Implementing decay techniques using 4t quasi-static memory cells. *Computer Architecture Letters*, 1(1), 2002.

[43] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark. Coordinated, distributed, formal energy management of chip multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 127–130, 2005.

[44] S. M. Jung, J. Jang, W. Cho, J. Moon, K. Kwak, B. Choi, B. Hwang, H. Lim, J. Jeong, J. Kim, and K. Kim. The revolutionary and truly 3-dimentional 25f2 sram technology with the smallest s3 cell, 0.16um2 and sstff for ultra high density sram. In *VLSI Technology Digest of Technical Papers*, 2004.

[45] Y. H. Kang, S. M. Jung, J. H. Jang, J. H. Moon, W. S. Cho, C. D. Yeo, K. H. Kwak, B. H. Choi, B. J. Hwang, W. R. Jung, S. J. Kim, J. H. Kim, J. H. Na, H. Lim,

J. H. Jeong, and K. Kim. Fabrication and characteristics of novel load pmos sstft (stacked single-crystal thin film transistor) for 3-dimentional sram memory cell. In *Proceedings of the IEEE Silicon-on-Insulator Conference*, 2004.

[46] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, 2001.

[47] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008.

[48] A. e. a. Keshavarzi. Intrinsic iddq: Origins, reduction, and applications in deep sub-low-power cmos ic's. In *Proceedings of the IEEE International Test Conference*, 1997.

[49] T. Kgil, A. Saidi, N. Binkert, R. Dreslinski, S. Reinhardt, K. Flautner, and T. Mudge. Picoserver: Using 3d stacking technology to enable a compact energy efficient chip multiprocessor. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 117–128, 2006.

[50] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003.

[51] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134, 2008.

[52] N. Kirman, M. Kirman, M. Chaudhuri, and J. F. Martinez. Checkpointed early load retirement. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 16–27, 2005.

[53] K. W. Lee, T. Nakqmura, T. Ono, Y. Yamada, T. Mozukusa, H. Hashimoto, K. T. Park, K. H., and N. Koyanag. Three-dimensional shared memory fabricated using wafer stacking technology. In *International Electron Devides Meeting. Technical Digest*, 2000.

[54] S. Lee, S. S. V. Au, and K. P. Moran. Constriction/ spreading resistance model for electronics packaging. In *Proceedings of the ASME/JSME Thermal Engineering Conference*, 1995.

[55] J. Li, J. F. Martinez, and M. C. Huang. The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pages 14–23, 2004.

[56] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42th International Symposium on Microarchitecture*, pages 469–480, 2009.

[57] T. Li, A. R. Lebeck, and D. J. Sorin. Spin detection hardware for improved management of multithreaded systems. 17(6):508–521, 2006.

[58] Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, M. Stan, and K. Skadron. State-preserving vs. non-state-preserving leakage control in caches. In *Proceedings of the International Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 22–27, 2004.

[59] P. Lindner, V. Dragoi, T. Glinsner, C. Schaefer, and R. Islam. *3D Interconnect Through Aligned Wafer Level Bonding*, pages 1439–43 BN – 0 7803 7430 4. IEEE, 2002.

[60] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. In *Proceedings of the 7th International Vonference on Architectural Support for Programming Languages and Operating Systems*, pages 138–147, 1996.

[61] G. L. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood, and K. Banerjee. A thermally-aware performance analysis of vertically integrated (3-d) processor-memory hierarchy. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pages 991–996, 2006.

[62] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey. A voltage reduction technique for digital systems. In *Proceedings of the 37th IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 238–239, 1990.

[63] N. Madan and R. Balasubramonian. Leveraging 3d technology for improved reliability. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 223–235, 2007.

[64] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 14–25, 2003.

[65] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[66] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, 1998.

[67] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Computer Architecture News*, 33:2005, 2005.

[68] J. Mayega, O. Erdogan, P. M. Belemjian, K. Zhou, J. F. McDonald, and R. P. Kraft. 3d direct vertical interconnect microprocessors test vehicle. In *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, GLSVLSI '03, pages 141–146, New York, NY, USA, 2003. ACM.

[69] J. D. Meindl. Interconnect opportunities for gigascale integration. In *IEEE Micro, vol. 23, no. 3, pp. 28-35,*, 2003.

[70] K. Meng, R. Joseph, R. P. Dick, and L. Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 177–186, New York, NY, USA, 2008. ACM.

[71] N. Nagaraj, T. Bonifield, A. Singh, R. Griesmer, and P. Balsara. Interconnect modeling for copper/low-k technologies. In *Proceedings of the 17th International Conference on VLSI Design*, VLSID '04, pages 425–, Washington, DC, USA, 2004. IEEE Computer Society.

[72] D. Nelson, C. Webb, D. McCauley, K. Raol, J. Rupley, J. DeVale, and B. Black. A 3d interconnect methodology applied to ia32-class architectures for performance improvements through rc mitigation. In *Proceedings of the 21st International VLSI Multilevel Interconnection Conference*, 2004.

[73] J. Parry, H. Rosten, and G. B. Kromann. The development of component-level thermal compact models of a c4/cbga interconnect technology: The motorola powerpc 603 and powerpc 604 risc microproceesors. *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, 21:104–112, 1998.

[74] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2008.

[75] P. Petoumenos, G. Psychou, S. Kaxiras, J. Cebrian Gonzalez, and J. Aragon. Mlp-aware instruction queue resizing: The key to power-efficient performance. In C. Müller-Schloer, W. Karl, and S. Yehia, editors, *Architecture of Computing Systems*, volume 5974 of *Lecture Notes in Computer Science*, pages 113–125. Springer Berlin / Heidelberg, 2010.

[76] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceed-*

ings of the International Symposium on Low Power Electronics and Design, pages 90–95, 2000.

[77] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-run: Leveraging smt and cmp to manage power density through the operating system. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 260–270, 2004.

[78] K. Puttaswamy and G. H. Loh. Implementing caches in a 3d technology for high performance processors. In *Proceedings of the 2005 International Conference on Computer Design*, ICCD '05, pages 525–532, Washington, DC, USA, 2005. IEEE Computer Society.

[79] K. Puttaswamy and G. H. Loh. Thermal herding: Microarchitecture techniques for controlling hotspots in high-performance 3d-integrated processors. In *Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture*, pages 193–204, 2007.

[80] R. M. Rao, J. L. Burns, A. Devgan, and R. B. Brown. Efficient techniques for gate leakage estimation. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, ISLPED '03, pages 100–103, New York, NY, USA, 2003. ACM.

[81] R. Reif, A. Fan, K.-N. Chen, and S. Das. Fabrication technologies for three-dimensional integrated circuits. In *Proceedings of the International Quality Electronic Design Symposium*, pages 33–37, 2002.

[82] J. Sartori and R. Kumar. Distributed peak power management for many-core architectures. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 1556–1559, 2009.

[83] R. Sasanka, C. J. Hughes, and S. V. Adve. Joint local and global hardware adaptations for energy. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS-X, pages 144–155, New York, NY, USA, 2002. ACM.

[84] Y. Sazeides and J. E. Smith. The predictability of data values. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 248–258, 1997.

[85] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the 8th International High-Performance Computer Architecture Symposium*, pages 29–40, 2002.

[86] T. Siddiqua and S. Gurumurthi. Balancing soft error coverage with lifetime reliability in redundantly multithreaded processors. In *Proceedings of the IEEE*

International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009.

[87] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. de Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings on Design Automation Conference*, pages 524–529, 2001.

[88] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Computer Architecture Symposium*, pages 2–13, 2003.

[89] S. J. Souri, K. Banerjee, A. Mehrotra, and K. C. Saraswat. Multiple si layer ics: Motivation, performance analysis and design implications. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 213–220, New York, NY, USA, 2000. ACM.

[90] S. Thoziyoor and N. Muralimanohar. Cacti 5.1. Technical report, 2008.

[91] A. W. Topol, D. C. La Tulipe, Jr., L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and M. Ieong. Three-dimensional integrated circuits. *IBM J. Res. Dev.*, 50:491–506, July 2006.

[92] E. Tune, D. Liang, D. M. Tullsen, and B. Calder. Dynamic prediction of critical path instructions. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pages 185–195, 2001.

[93] J. A. Winter and D. H. Albonesi. Addressing thermal nonuniformity in smt workloads. *ACM Transactions on Architecture and Code Optimization*, 5:4:1–4:28, May 2008.

[94] S. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, June 1995.

[95] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 248–259, 2004.

[96] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 178–189, 2005.

[97] Y. Xie, G. H. Loh, B. Black, and K. Bernstein. Design space exploration for 3d architectures. *J. Emerg. Technol. Comput. Syst.*, 2:65–103, April 2006.

[98] L. Xue, C. C. Liu, and S. Tiwari. Multi-layers with buried structures (mlbs): An approach to three-dimensional integration. In *Proceedings on IEEE International SOI Conference*, pages 117–118, 2001.

[99] S. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pages 147–157, 2001.

[100] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical report, 2003.

[101] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 61–70, 2001.

[102] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph. Three-dimensional chip-multiprocessor run-time thermal management. 27(8):1479–1492, 2008.