Performance and Power Optimizations in Chip Multiprocessors for Throughput-Aware Computation

DISSERTATION
Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain

June 2013

BY

AUGUSTO J. VEGA

Thesis Advisors:
Alex Ramírez
Mateo Valero

Per Chiara
"Il tuo sorriso si espande
come una farfalla"

Abstract

The so-called "power (or power density) wall" has caused core frequency (and single-thread performance) to slow down, giving rise to the era of multi-core/multi-thread processors. For example, the IBM POWER4 processor [97], released in 2001, incorporated two single-thread cores into the same chip. In 2010, IBM released the POWER7 processor [59] with eight 4-thread cores in the same chip, for a total capacity of 32 execution contexts. The ever increasing number of cores and threads gives rise to new opportunities and challenges for software and hardware architects. At software level, applications can benefit from the abundant number of execution contexts to boost throughput. But this challenges programmers to create highly-parallel applications and operating systems capable of scheduling them correctly. At hardware level, the increasing core and thread count puts pressure on the memory interface, because memory bandwidth grows at a slower pace — phenomenon known as the "bandwidth (or memory) wall". In addition to memory bandwidth issues, chip power consumption rises due to manufacturers' difficulty to lower operating voltages sufficiently every processor generation. This thesis presents innovations to improve bandwidth and power consumption in chip multiprocessors (CMPs) for throughput-aware computation: a bandwidth-optimized last-level cache (LLC), a bandwidth-optimized vector register file, and a power/performance-aware thread placement heuristic.

In contrast to state-of-the-art LLC designs, our organization avoids data replication and, hence, does not require keeping data coherent. Instead, the address space is statically distributed all over the LLC (in a fine-grained interleaving fashion). The absence of data replication increases the cache

effective capacity, which results in better hit rates and higher bandwidth compared to a coherent LLC. We use double buffering to hide the extra access latency due to the lack of data replication.

The proposed vector register file is composed of thousands of registers and organized as an aggregation of banks. We leverage such organization to attach small special-function *local computation elements* (LCEs) to each bank. This approach —referred to as the *processor-in-regfile* (PIR) strategy— overcomes the limited number of register file ports. Because each LCE is a SIMD computation element and all of them can proceed concurrently, the PIR strategy constitutes a highly-parallel super-wide-SIMD device (ideal for throughput-aware computation).

Finally, we present a heuristic to reduce chip power consumption by dynamically placing software (application) threads across hardware (physical) threads. The heuristic gathers chip-level power and performance information at runtime to infer characteristics of the applications being executed. For example, if an application's threads share data, the heuristic may decide to place them in fewer cores to favor inter-thread data sharing and communication. In such case, the number of active cores decreases, which is a good opportunity to switch off the unused cores to save power.

It is increasingly harder to find bulletproof (micro-)architectural solutions for the bandwidth and power scalability limitations in CMPs. Consequently, we think that architects should attack those problems from different flanks simultaneously, with complementary innovations. This thesis contributes with a battery of solutions to alleviate those problems in the context of throughput-aware computation: 1) proposing a bandwidth-optimized LLC; 2) proposing a bandwidth-optimized register file organization; and 3) proposing a simple technique to improve power-performance efficiency.

Acknowledgments

There is a day that certainly changed my life. It was in 1990. I was ten years old when, one day at school, a teacher put in front of me a computer for the first time. It was a Commodore 64. It happened in a remote Argentinian village a long time ago. That day I will never forget.

After that *first time*, some people throughout my life helped me understand that, if working with computers was my dream, I had to do my best to pursue it. My parents, Miryam and Carlos, are the first ones in that "list". In the *Spring Day* of 1991 they got rid of a small motorcycle to buy me a *Personal Computer XT*. Seven years after I got my first computer, one day in 1998, they told me: *"look, we have no wealth to bequeath you; but we can give you one thing: a university career."* I then moved to Buenos Aires where I met Professor José Luis Hamkalo, a computer architecture researcher who opened me the doors of his lab for the first time. I then realized that I wanted *to do* processor architectures.

In 2007 I moved again, this time to cross the ocean. I started my Ph.D. at Universitat Politècnica de Catalunya, working as a Resident Student at the Barcelona Supercomputing Center. I want to especially thank Professors Alex Ramírez and Mateo Valero, my Ph.D. advisors, for giving me the opportunity to work with such a world-class group of researchers. Thanks!

I joined the IBM T.J. Watson Research Center in 2010, where I met the two most influential people in my professional life: Doctors Pradip Bose and Alper Buyuktosunoglu. With them, I discovered the *happiness* side of doing research, which is what really matters at the end of the day. With them, I finally became *a researcher*.

There is one special person around who my life revolves. Her name is

Chiara and she is my wife. Her support has been my source of strength since we met in Barcelona. And I hope I made sense to her life too. I dedicate this work to her.

I also want to thank Alex Rico and all the guys from BSC and UPC with whom I was fortunate to share unforgettable moments. Thanks to my sister too, María Inés, for those beautiful days of *mate y biscochitos* in Buenos Aires. You know how much I love you.

I could continue saying "thanks" to everyone who did their bit. To all of them, thanks for helping me to be a better person.

New York City
Spring 2013

"Opportunity is missed by most people
because it is dressed in overalls and looks like work."
Thomas A. Edison

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The well-known limits to transistor miniaturization gave rise to a variety of multi-threaded and multi-core processors. Since early 2000s, the number of threads in a core and the number of cores in a chip have been growing steadily as an answer to the concerns about Moore's Law sustainability. This trend crosses different design domains, from general-purpose processors, to graphics processing units (GPUs), to hybrid processors. In the *general-purpose computation domain*, the IBM POWER7 processor [59, 104] incorporates eight cores in the same chip, each one with four hardware threads. Intel's Xeon E5 processor family [49], based on the Sandy Bridge-EN and Sandy Bridge-EP architecture, has up to six cores, two threads each. In the *GPU domain*, we can mention the NVIDIA GeForce GTX 580 GPU [74] (based on the Fermi micro-architecture [76]), with 512 CUDA cores in the same chip. More recently, NVIDIA has introduced the Tesla K-series GPU accelerator family based on the NVIDIA Kepler architecture [77] with up to 2688 CUDA cores. In the *hybrid computation domain* —i.e., designs combining general-purpose cores, with GPUs and/or other types of accelerators—, AMD released in 2011 the Llano [16] variant of the Fusion accelerated processor unit (APU) [4]. Llano integrates four x86 cores and a Radeon-based GPU into the same chip. The IBM PowerEN processor [17, 37, 57] is composed of 16 PowerPC-based cores, four threads each, plus six hardware accelerators. We also have to mention the IBM Cell/B.E. processor [58] with eight Syner-

gistic Processing Elements (SPEs) attached to a POWER-based processor, intended for multimedia and vector processing applications.

There are two ways to exploit the growing number of execution contexts (i.e., physical threads and cores) in current chip multiprocessors (CMPs): an application can be parallelized using a parallel programming models (e.g. POSIX threads [20], OpenMP [22] or CUDA [75], among others), or multiple non-parallelizable applications can be executed at the same time, one per thread or per core. Throughout this thesis, we refer to the former model as *multi-threaded execution*, and to the latter as *multi-programmed execution*. Also, whenever we refer to physical threads at core level, we mean Simultaneous Multi-Threading (SMT) [103].

Regardless of the parallel execution model, threads and/or programs share CMP's resources, which poses new architectural challenges. This dissertation tackles two of those challenges, which we consider key in the context of throughput-aware computation:

- **Memory bandwidth** – how to efficiently share the available memory bandwidth between physical threads and cores in the chip.
- **Power consumption** – how to manage the chip power budget in order to achieve power-proportional computation [8].

In this thesis, we employ the term *throughput-aware computation* to refer to scenarios with abundant parallelism, either in the form of multiple threads or multiple single-threaded applications running at the same time.

1.1 Motivation

In the last years, new designs were introduced in the growing domain of chip multiprocessors (CMPs). We provided examples of such designs in the previous section, categorized as: *general-purpose computation* (IBM POWER7, Intel Xeon E5), *graphical computation* (NVIDIA's Fermi and Kepler-based GPUs) and *hybrid computation* (AMD Llano, IBM PowerEN, IBM Cell/B.E.). The number of execution contexts (threads and cores) in CMPs has been

steadily growing to mimic Moore's Law. Today, we can find designs with tens or hundreds of execution contexts: e.g. 32 threads in IBM POWER7 (grouped in eight 4-way SMT cores) or 512 threads in NVIDIA GeForce GTX 580. This enables unprecedented levels of parallelism, which is ideal for throughput-aware computation [40, 73]. Examples of throughput-aware applications are: real-time computer graphics and video processing, signal processing, medical-image analysis, molecular dynamics, astrophysical simulation, and gene sequencing [40]. The most important characteristic of these applications is that they can be decomposed into data blocks for massively parallel processing.

The memory system in throughput-aware designs can be organized based on two models: *hardware-managed coherent caches* and *software-managed streaming memory* [64]. In the latter case, the processor usually incorporates per-core local memories and transfers data to and from those memories using *direct memory accesses* (DMAs). DMA transfers overlap computation to hide latency —technique known as *double buffering* [23, 89]. The double buffering technique requires large-enough local (*scratchpad*) memories to keep both, the data being processed and the data being prefetched from the memory system to be processed next. As an example, each Synergistic Processing Element (SPE) in the Cell/B.E. architecture [58] has a 256-KB local memory. A SPE-level DMA engine transfers data between this memory and main memory using high-bandwidth DMAs.

The double buffering technique is an effective way to tolerate high memory latencies. But the increment in the number of cores in current CMPs moved the problem from the *latency* side to the *bandwidth* side. It is not enough anymore to feed a core with data quickly. It is also mandatory to feed *many* cores. The memory bandwidth became the key limiter for performance scalability. For example, with only eight SPEs, the Cell/B.E. already had to resort to a high bandwidth Rambus XDR memory system to deliver 25.6 GB/s [58]. The POWER7, with eight 4-way SMT cores, incorporates two four-channel DDR3 memory controllers, delivering up to 100 GB/s [92]. The NVIDIA GeForce GTX 580 GPU feeds its 512 threads with a 192.4 GB/s DDR5 memory system [74].

The underlying problem about memory bandwidth is that it grows much slower than the number of cores. Bandwidth scalability is mainly limited by chip pin count and power consumption. Adopting large on-chip memories and caches can reduce memory pressure due to temporal reuse of data. However, off-chip memory pressure is heavily exacerbated by the increasing number of cores and threads [42], particularly true for highly parallel applications, to the point where *just* large caches are not the solution anymore. Based on this motivation, we focus on two key parts of the memory system: the last-level cache and the register file.

In the last-level cache (LLC), we leverage the benefits of the software-managed streaming memory model with DMA transfers. The main idea is to avoid data replication (and coherence requirements) in the LLC. Instead, cores are allowed to access remote cores' caches. Figure 1.1 shows an illustrative CMP with 32 cores grouped in four clusters. Cores in each cluster share a local LLC block, and can also access other clusters' LLCs. With this strategy, each core "sees" a much larger LLC, which is beneficial for bandwidth. We use double buffering to hide the extra latency to access remote blocks. The LLC organization proposed in this thesis is presented in Chapter 3.

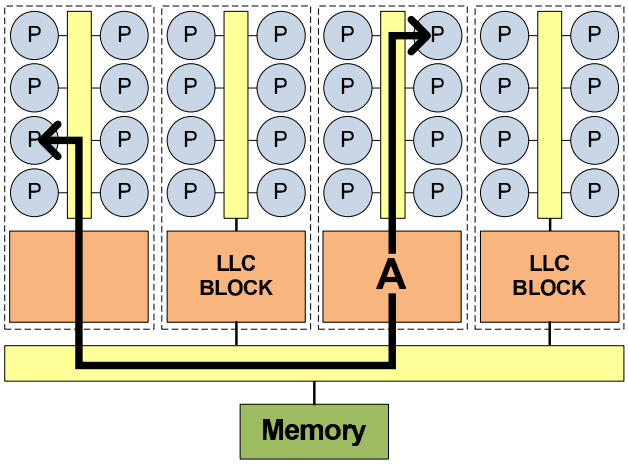


Figure 1.1: Illustrative 32-core CMP, with the proposed LLC organization. Eight cores per cluster share a local LLC block, and can access remote blocks (double buffering hides the extra latency). Data is not replicated in the LLC.

To further reduce the pressure on the memory system, we adopt a very-large vector register file. The goal is to keep data as much as possible in the register file to reduce accesses to the cache hierarchy. The register file considered in this thesis, which has thousands of vector registers, is presented by Derby et al. in [29] in the context of an in-line accelerator. In this thesis, we implement such organization with multiple banks, which provides a twofold benefit: it keeps wire propagation delay under control, and allows to exploit local computation in each bank. As explained in Chapter 4, this processing capability at register file level is implemented with embedded, small Single Instruction, Multiple Data (SIMD) *local computation elements* (LCEs) attached to each bank. Figure 1.2 shows an 8-bank register file with one LCE per bank.

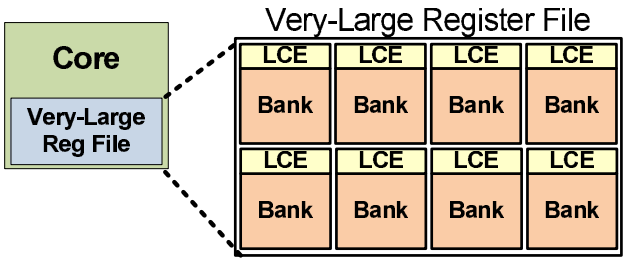


Figure 1.2: Proposed very-large register file, in this case with eight banks and one LCE per bank. Each LCE provides SIMD computation support to its attached bank.

In addition to memory bandwidth, chip power consumption is the other key obstacle to CMP performance scalability. The "power wall" issue is so critical that has given rise to a new research field in microprocessor architectures: to minimize transistor under-utilization (*dark silicon*) [33]. The underlying problem about dark silicon is that the available energy/power budget in today's CMPs prevents having 100% of the chip powered on all the time. For example, at 22nm, 21% of a chip must be powered off due to power consumption constraints [33]. Therefore, it is crucial to understand what parts of the chip can be powered off during execution, in order to reduce energy/power consumption, with minimal performance impact. Even on highly-parallel applications, it is possible to identify computation phases

in which not all chip resources are exercised. Figure 1.3 presents the software thread count frequency for twelve PARSEC multi-threaded applications [12] executed with 32 software threads and *native* inputs on an IBM POWER7 processor. As can be observed, multi-threaded applications spend significant time with fewer threads than the specified thread count (just Swaptions and Vips execute with 32 threads most of the time). In such scenarios, we may decide to power off unused resources (e.g. cores) whenever possible to save energy and power. In this thesis we propose and evaluate a thread consolidation heuristic (TCH) which optimizes processor power-performance efficiency based on applications characteristics. TCH —which is implemented as a simple closed-loop control algorithm at operating system level— gets runtime chip-level power and performance information to infer characteristics of the applications being executed. Based on this information, the heuristic decides how to distribute software (application-level) threads across hardware (physical) threads and cores in a CMP to optimize power-performance efficiency. We describe the implementation details of this heuristic in Chapter 5.

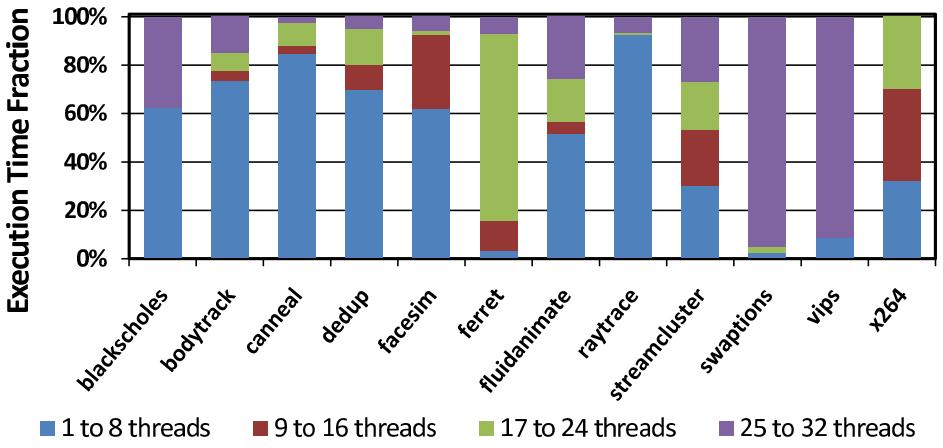


Figure 1.3: Software thread count histogram for twelve multi-threaded PARSEC applications. Applications are executed with 32 software threads to fully populate the underlying POWER7 processor. Even in that case, applications are not able to exploit all the available hardware threads.

1.2 Thesis Statement

Performance scalability of current CMPs is mainly constrained by memory bandwidth and chip power consumption.

Memory systems (including register files, caches, interconnections and memory controllers) are optimized for latency, which is supported by data replication and coherence. We argue that adopting latency-optimized memory systems in the domain of throughput-aware computation requires a thorough analysis. Throughput-aware computation is mostly bandwidth bound, instead of latency bound.

Chip power consumption is the other key factor that limits the amount of logic that can turned on in a modern CMP. We argue that power management techniques have to be incorporated into the chip and/or system software (e.g. operating system kernel). The goal is to attain power-proportional computation, by power gating unused chip components (e.g. functional units, cores, memory controllers, etc.).

The main goal of this dissertation is to improve CMP performance scalability in the context of throughput-aware computation, by tackling the memory bandwidth and power consumption issues. This goal is achieved with new bandwidth-optimized last-level cache and register file organizations, and a heuristic for chip power reduction.

1.3 Contributions

In this thesis we evaluate optimizations to improve performance and power consumption of CMPs under throughput-aware workloads. We next summarize the main contributions of this thesis (an overview is shown in Figure 1.4).

1.3.1 Bandwidth-Optimized Last-Level Cache

A re-design of the last-level cache (LLC) targeting throughput-aware computation. We present a bandwidth-optimized LLC organization that is suitable for throughput-aware computation on CMPs. Its most important character-

istics are the following:

- We avoid data replication to improve effective capacity. Since data is not replicated in the cache, there is no need to deal with coherence issues.
- We statically distribute the address space across cache blocks in a fine-grained interleaving fashion.

The benefit of our LLC organization is twofold: its larger effective capacity delivers higher bandwidth (more data is kept in the LLC, which results in fewer off-chip memory accesses), and the fine-grained address space interleaving allows multiple transfers to proceed in parallel. These benefits come at the expense of locality, due to data being spread all over the available storage. While this trade-off is harmful for latency-aware computation, it is beneficial for throughput-aware workloads.

1.3.2 Bandwidth-Optimized Register File

A very-large register file that significantly cuts down the number of memory accesses. It is conceived as an aggregation of banks to keep wire propagation delay under control. Such organization unveils an additional optimization opportunity: SIMD computation support embedded into the register file. This *processor-in-regfile* (PIR) strategy is implemented as small special-function *local computation elements* (LCEs) attached to each bank. This approach overcomes the limited number of register file ports. Each LCE is a SIMD computation element, and all of them can proceed concurrently. Therefore, the PIR strategy constitutes a highly-parallel super-wide-SIMD approach, ideal for throughput-aware computation.

1.3.3 Power Management Techniques for CMPs

A heuristic that improves power-performance efficiency in CMPs by dynamically placing parallel applications across physical threads and cores. The heuristic —which is implemented as a simple closed-loop control algorithm

at operating system level— gets runtime chip-level power and performance information to infer characteristics of the applications being executed. For example, if an application's threads share data, the heuristic may decide to place them in fewer cores (at higher SMT level) to favor inter-thread data sharing and communication. In such case, the number of active cores decreases, which is a good opportunity to switch off the unused cores to save power.

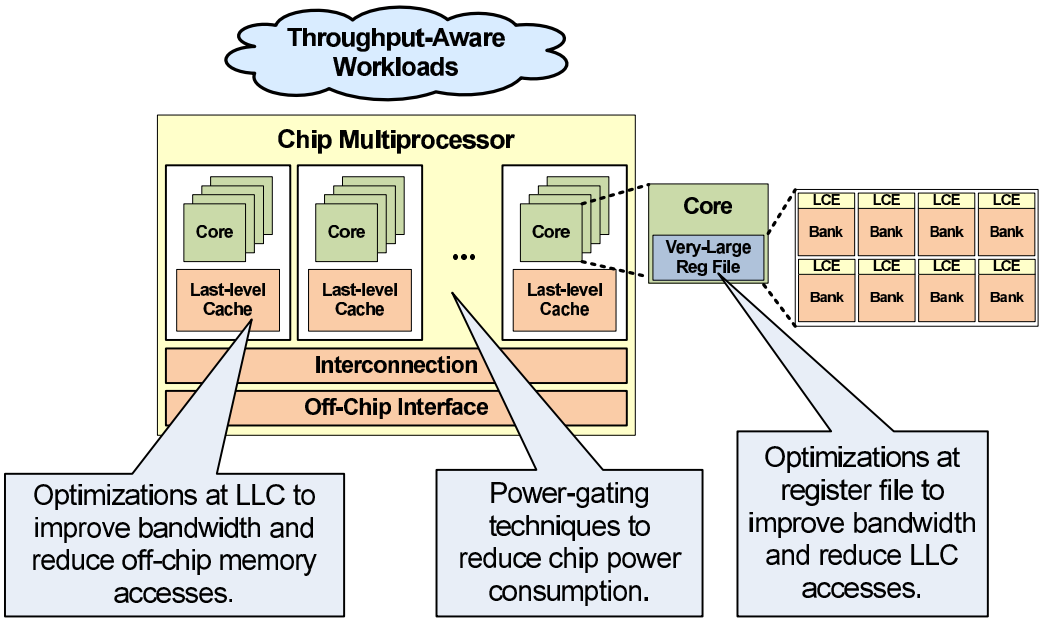


Figure 1.4: Overview of the optimizations considered in this thesis.

1.4 Summary and Concluding Remarks

This thesis evaluates optimizations to improve memory bandwidth and chip power consumption in the context of CMPs for throughput-aware computation.

Throughput-aware applications differ from traditional latency-aware applications in that they can be decomposed into data blocks for massively parallel processing. This abundant amount of parallelism increases the mem-

ory bandwidth pressure. Consequently, the adoption of caches optimized for latency in the domain of throughput-aware computation has to be done thoroughly: in throughput-aware scenarios, the memory system has to be designed to tackle the "bandwidth wall" instead of the "latency wall". The bandwidth wall, and its implications on CMP performance scalability, motivates us to propose a last-level cache (LLC) and a register file organizations aimed for throughput-aware computation.

The proposed LLC is composed of a set of independent blocks but neither multiple copies nor block migration between them is allowed. Instead, data is statically interleaved across LLC blocks in a fine-grained fashion. This approach delivers more bandwidth at the expense of higher access latency —we use double buffering to hide this extra latency.

The proposed register file is a very large storage with thousands of registers, conceived as an aggregation of banks. We attach a small SIMD local computation elements (LCEs) to each bank. This *processor-in-regfile* (PIR) strategy overcomes the limited number of register file ports: all the LCEs can proceed concurrently, reading/writing registers from/to its attached register file bank. The PIR strategy and the large register file capacity (tens of kilobytes) enable a novel computation model, suitable for throughput-aware domains: substantial performance gains can be obtained by loading relatively large blocks of data into the register file (e.g. a cache-line at a time), operating on the entire block of data, keeping intermediate results in the register file, and storing the final results to memory a cache-line at a time. The proposed register file organization provides further advantage when the produced result is used as the input to another function, a common scenario in throughput-aware computation. For instance, in graphics processing, a JPEG compressor applies a chain of functions to the input image. With the proposed register file organization, the entire image (or a large part of it) resides in the register file while the LCEs apply the different functions *in situ*.

In addition to the bandwidth-optimized LLC and register file, we also propose a heuristic that optimizes CMP power-performance efficiency. The heuristic —which is implemented as a simple closed-loop control algorithm

at operating system level— gets runtime chip-level power and performance information to decide which is the most efficient way to distribute software threads across hardware threads and cores. The goal is to place software threads in as few cores as possible to reduce power consumption on unused cores, with minimal performance impact.

In conjunction, the techniques presented in this thesis (either for bandwidth and power optimization) are intended to alleviate CMP performance scalability limitations in the context of throughput-aware computation.

Chapter 2

Methodology

This chapter describes the tools and methodology adopted throughout the development of this thesis. This includes analytical models, simulators, real hardware platforms, applications and other resources. As it was explained in Chapter 1, this thesis focuses on architectural innovations at three different CMP components: the last-level cache (LLC), the register file and core-level power gating —the presentation of the methodology is organized following that same structure.

2.1 Last-Level Cache Methodology

We model the LLC organizations as part of TaskSim, a trace-driven cycle-accurate CMP simulator [83, 84]. The author of this thesis was part of the Heterogeneous Computer Architecture group at Barcelona Supercomputing Center, the team that developed TaskSim. We present TaskSim in Section 2.1.1.

We use CACTI to estimate the access latency of the LLCs modeled in TaskSim. CACTI is a model for cache access time, area and power developed by HP Labs [102]. We present CACTI in Section 2.1.2.

We adopt a set of parallel scientific kernels to generate the traces to use in TaskSim. We present the details of these applications and the trace generation process in Section 2.1.3.

2.1.1 TaskSim

TaskSim is a trace-driven cycle-accurate modular simulator [83, 84] which targets large-scale multi-core architectures. TaskSim's most important characteristic is its ability to accurately model large-scale CMPs: tens or hundreds of cores, interconnections, complete cache hierarchy and memory system. This ability relies on the use of a task-level abstraction for the internal simulation of cores. This means that TaskSim does not capture intra-core micro-architectural behavior. The rest of the components (interconnections, cache hierarchy and memory controllers) are realistically modeled at cycle level. The lack of micro-architectural core details in the model is irrelevant to accurately evaluate the LLCs, while it also provides fast simulation.

Figure 2.1 presents a high-level diagram of the CMP components that can be modeled using TaskSim. The simulator implements the different components as *modules* which are interconnected through *ports*. Our case study, shown in Figure 2.1, is a Cell/B.E.-like architecture with 32 processing elements (workers), each one composed of a CPU (core), a local memory and a DMA engine. Processing elements group together in clusters and share a local LLC block. A global network interconnects clusters and a set of memory controllers. In addition, a master processor is responsible for assigning tasks to processing elements, which resembles the master-worker execution model in the IBM Cell/B.E. processor [58]. We leverage TaskSim modularity to model a variety of configurations, with different numbers of processing elements and LLC blocks.

Table 2.1 summarizes the main architectural parameters adopted for the LLC evaluation of Chapter 3. The number of processing elements simulated are 16, 32, 64 and 128 in clusters of eight (i.e. 2, 4, 8 and 16 clusters, respectively). The LLC is partitioned in one block per cluster and modeled as embedded DRAM (eDRAM) [5, 55, 56]. The LLC sizes simulated are 8, 16, 32 and 64 MB. The access to LLC data is based on two schemes: *interleaved* and *non-interleaved*. The former does not replicate data in the LLC —instead, the address space is interleaved across blocks. The latter corresponds to a traditional coherence-based organization where data can be replicated across

LLC blocks. Chapter 3 presents the evaluation of these two LLC schemes. We model global and in-cluster interconnections as 8-byte/cycle full-crossbar networks, and the memory interface as four 25.6-GB/s memory controllers (for a 102.4-GB/s total bandwidth).

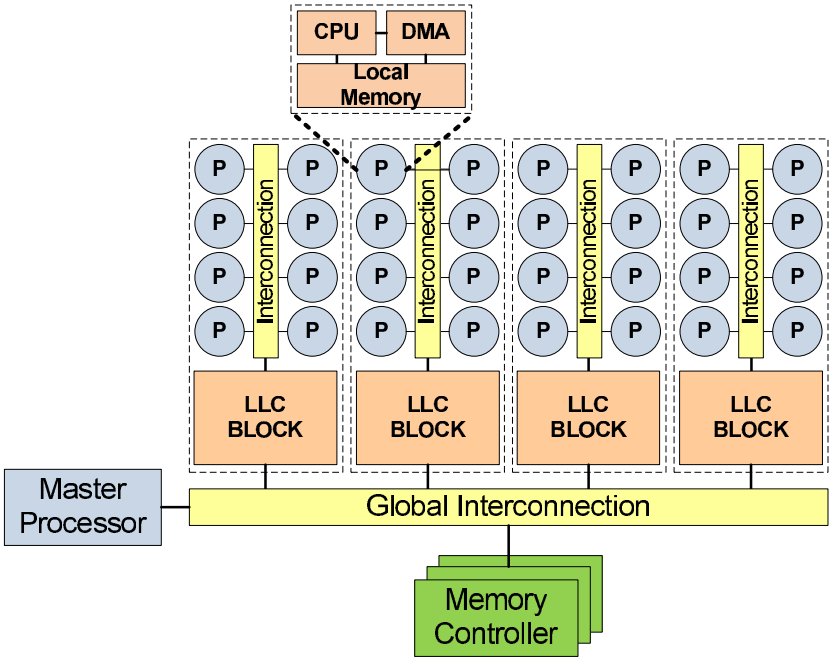


Figure 2.1: Illustrative CMP architecture simulated in TaskSim. In this example, processing elements are clustered in groups of eight. Each processing element includes a CPU (core), a local memory and a DMA engine. There is a LLC block per cluster, and all clusters are interconnected through a global network. A set of memory controllers are also modeled.

2.1.2 CACTI

CACTI is a tool capable of finding optimal cache configurations in terms of access time, along with its power and area characteristics [102]. In this thesis, we adopt CACTI version 5.3 to determine the access latency corresponding to the different LLC configurations evaluated. Table 2.2 summarizes some of the most relevant CACTI input parameters chosen. Because we are interested in the access latency of one LLC block, the cache size parameter corresponds to

Component	Configuration
Processing elements	16, 32, 64 and 128 processing elements (eight per cluster) **Core frequency:** 3.2 GHz **Local memory size:** 256 KB (one local memory per core) **DMA engine:** up to 16 concurrent DMAs, 128-byte DMA packages
Last-level cache	4-way set associative, 45nm eDRAM, 128-byte lines **Size:** 8, 16, 32 and 64 MB **Blocks:** 2, 4, 8 and 16 (one LLC block per cluster) **Ports:** 2 read/write ports per LLC block **Data placement policies:** interleaved and with replication **MSHR:** 64 entries each (one independent MSHR per LLC block)
Interconnections	8 bytes/cycle bandwidth, full-crossbar network
Memory system	4 memory controllers at 25.6 GB/s each (102.4 GB/s total)

Table 2.1: Simulated architectural parameters.

the size of the block and not the entire LLC. This size varies as a consequence of the difference LLC sizes and number of blocks explored, which are listed in Table 2.1. A LLC block is composed of one or more physical banks. In this thesis we assume a physical bank size of 512 KB. Therefore, the number of banks depends on the LLC block size. The smallest LLC block size simulated is 512 KB, which is implemented with just one physical bank. The largest LLC block size is 32 MB, which is implemented with 64 banks. To model eDRAM-based LLCs, the data and tag array cell type adopted is logic process based DRAM (LP-DRAM) [101].

Parameter	Value
Cache sizes (LLC block sizes)	512 KB to 32 MB
Line size	128 bytes
Associativity	4
Banks	1 to 64
Technology Node	45 nm
Read/Write Ports	2
RAM cell/transistor type in data array	LP-DRAM
RAM cell/transistor type in tag array	LP-DRAM
Interconnect projection type	Conservative

Table 2.2: CACTI input parameters used to estimate LLC block access latency.

2.1.3 Applications

In TaskSim, the master and worker CPU modules are fed with traces of scientific applications written using a task-based programming model. The traces are sequences of tasks that the master processor schedules on worker processors. The LLC evaluation presented in Chapter 3 is performed using six parallel scientific kernels: Check-LU, Cholesky, FFT-3D, K-means, K-NN and MatMul. These benchmarks were written in the Cell/B.E. variant of the StarSs [9] programming model. The traces collected from these benchmarks contain the information about the required computation time for different phases in the processors as well as the inter-processor communications through DMA transfers. Table 2.3 shows a summary of the main characteristics of each benchmark: number of tasks, average task run time, memory footprint, and estimated bandwidth required per task. The bandwidth estimate is obtained from the average task data size and run time.

2.2 Register File Methodology

Due to TaskSim's lack of support for core-level micro-architecture modeling, we have to resort to other tools to study the proposed register file. This part of the thesis is conducted at the IBM T.J. Watson Research Laborato-

Kernel	Number of tasks	Average task run time	Problem size	Estimated BW per task
Check-LU	54814	45.7 μs	256 MB	1.11 GB/s
Cholesky	357760	28.0 μs	512 MB	1.68 GB/s
FFT-3D	32768	13.9 μs	128 MB	3.27 GB/s
K-means	335872	30.7 μs	195 MB	1.56 GB/s
K-NN	800768	7.9 μs	36 MB	0.49 GB/s
MatMul	262144	25.8 μs	192 MB	1.42 GB/s

Table 2.3: Simulated benchmarks.

ries, during the internship that the author carries out in the Reliability- and Power-Aware Microarchitectures group. Due to confidentiality restrictions, configuration values and results corresponding to this study are normalized. Still, these disclosure-related issues do not affect the value of the study nor depreciate the importance of the conclusions presented in this thesis.

2.2.1 In-Line Accelerator and Register File Model

As we explain in Chapter 4, the very-large vector register file proposed in this thesis is studied in the context of an in-line accelerator for the IBM PowerEN processor [17, 37, 57]. Figure 2.2 presents an architectural outline of such in-line accelerator and its interaction with the A2 processor, the PowerEN's constituent core.

Execution in the accelerator proceeds in two stages. First, the map registers (MRs) are accessed to determine which registers will be read or written. MRs constitute a mechanism for indirect access to the register file using operand-associated mappings [29], because 5-bit operands are not enough to index 2048 registers. Vector execution then proceeds in the second stage. The fetch engine can enqueue two instructions per cycle into the MR issue queue inside the accelerator. After issuing and reading their input MRs, map management instructions are executed immediately. Other instructions proceed through the select pointers stage, which determines the actual registers read/written, based on the pointers in the MR values read from the register file. Dependency analysis then determines which prior instructions, if any,

each vector instruction depends on. Instructions are then enqueued a second time to wait for vector register dependencies and an appropriate execution pipeline. Once instructions issue, they read their input values and execute. Register renaming is impractical for registers in the very-large register file because the 2048 architected registers would require a highly-ported rename table with 2048 entries. Instead, the in-line accelerator employs a *future file*. The future file holds vector register values until the producer commits. After commit, future file entries are spilled to the register file, which holds only committed architected state. Instructions determine the future file entry of prior in-flight instructions that produce needed input values. After issue, instructions read their inputs from the future file, the bypass network or the register file as needed.

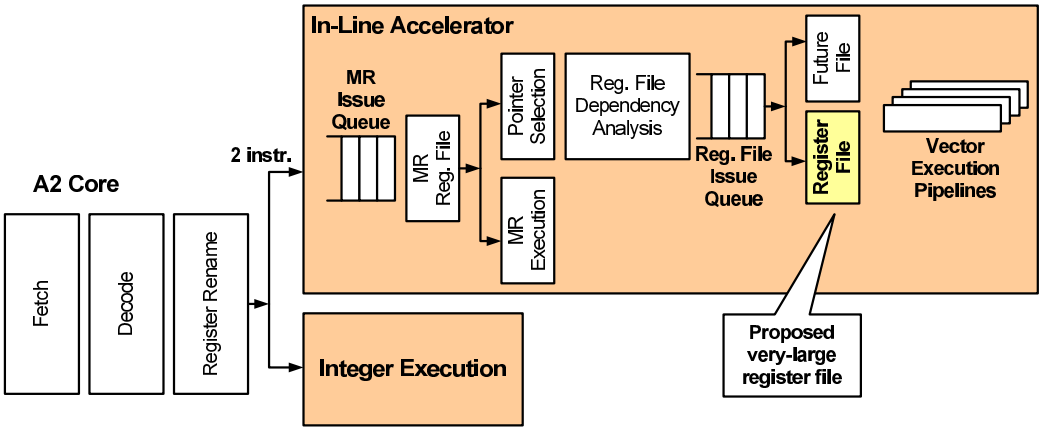


Figure 2.2: Micro-architecture model of the in-line accelerator (source: [28])

2.2.2 Performance Evaluation

To study the performance of the proposed register file in the context of an A2 processor, we use an IBM-internal trace-driven cycle-accurate simulator. This tool models the most important A2 core micro-architectural details, as well as first level instruction and data caches, and second level cache. The simulator is augmented to incorporate the in-line accelerator, of which the very-large register file is a constituent part.

We use the simulator to evaluate the register file performance for a single A2 core. Table 2.4 summarizes the main architectural parameters adopted for this study. Although the original A2 design is 4-way SMT, in this thesis we model a single-threaded core because so are the studied applications. Among the core execution pipelines, the in-line accelerator is a VMX-like 32-byte-wide SIMD unit. The register file, which is part of the accelerator, consists of 2048 32-byte vector registers together with an indirection mechanism for addressing them dynamically. We discuss the register file features in Chapter 4. In our model, we adopt perfect (i.e. "infinite") L1 and L2 caches. Due to its large capacity, the register file is able to keep input and output data from beginning to end of the computation. Therefore, L1 and L2 caches do not influence the evaluation.

Component	**Configuration**
A2-like core	Single-threaded†, 64-bit PowerPC-based architecture (†original A2 processor is 4-way SMT) **Core frequency:** 2.3 GHz **Issue width:** 2 **Execution mode:** out-of-order **Execution units:** branch unit, two fixed-point units, load/store unit and in-line accelerator
Cache hierarchy	Perfect L1 and L2 caches
In-line accelerator	32-byte-wide SIMD unit with fixed-point and single-precision floating-point arithmetic support **Execution mode:** in-order **Register file:** 2048 256-bit vector registers (64 KB total), eight 8-KB banks, 4RD/1WR ports per bank

Table 2.4: Simulated A2 core architectural parameters.

2.2.3 Applications

The benefits of the proposed register file are evaluated in the context of applications for wireless base stations. We select two classes of algorithms of particular importance for base stations, namely FFT and Turbo Decoding,

because they are the most heavily used and perhaps the most significant consumers of processor cycles.

The adopted FFT implementation is based on Pease's method [93], which is explicitly optimized for parallel computation. The evaluated FFT sizes are 512, 1024 and 2048 complex points, with 16 bits each for the real and imaginary parts.

The Turbo Decoding algorithm incorporates two constituent decoders, interleaver, and de-interleaver in a feedback loop, with the decoders implementing the BCJR algorithm [7]. The input to the decoder is a bit stream (*codeword*) with two parity bits per each data bit (1/3 rate encoding). We evaluated a 6144-element codeword because this is the maximum length specified by the 3GPP Long Term Evolution (LTE) wireless standard [1].

2.2.4 Area and Power Evaluation

Register file area and power consumption are analytically estimated based on information about PowerEN 45nm SOI-CMOS technology and register file cell type [31, 57].

2.3 Power Management Methodology

In this part of the thesis we study the power implications of running multi-threaded workloads on cores with varying SMT levels. This evaluation is done in the context of a real platform (an IBM BladeCenter PS701 system), which is explained in Section 2.3.1. We use multi-threaded applications from the PARSEC 2.1 benchmark suite [12], which are presented in Section 2.3.4.

2.3.1 Experimental Platform

The set-up system used for the experiments is an IBM BladeCenter PS701 machine. The system has one IBM POWER7 processor running at 3.0 GHz and 32 GB of DDR3 SDRAM running at 800 MHz. The POWER7 chip multi-processor [59] is composed of eight processor cores, each one capable

of four-way SMT operation. The eight cores can provide 32 concurrently executing threads. Each core has access to a private 256-KB L2 cache and to a local 4-MB L3 region. Eight L3 regions constitute a 32-MB on-chip L3 cache (local L3 regions provide low-latency access to the cores). In addition to its private L2 and local L3 caches, a core can also obtain data from other cores' L2 and L3 caches [92].

The platform runs RHEL 5.7 OS with Linux kernel version 3.0.1. Each PARSEC benchmark is executed as a single running workload to analyze its performance and power characteristics. During each run, we explicitly define the correspondence between software and hardware threads using the `taskset` command, which is a Linux tool to set processes' CPU affinity [91].

Our experimental system consists of an IBM BladeCenter PS701 system (Figure 2.3), where the proposed thread consolidation heuristic (TCH) executes and actuates at OS level. TCH obtains chip-level power measurements using the IBM Automated Measurement of Systems for Temperature and Energy Reporting software [35, 63]. The software connects to the EnergyScale microcontroller to download real-time power readings of the POWER7 processor under evaluation. In addition, TCH obtains hardware events information from processor counters available in the POWER7 chip [32, 69], which are discussed in Section 2.3.3. With all this information, TCH can make decisions and perform actuations (i.e. to consolidate or unconsolidate threads) at runtime.

Whenever TCH decides to consolidate threads, cores that are left unused are switched to *nap* idle state to reduce power consumption. The nap mode in POWER7 deactivates instruction fetch and execution and turns off all clocks to the execution engines in the core, but it still keeps L2 and L3 caches coherent [35, 36].

2.3.2 IBM Automated Measurement of Systems for Temperature and Energy Reporting

The IBM BladeCenter PS701 system used in our experiments incorporates firmware which runs on a dedicated microcontroller to provide energy man-

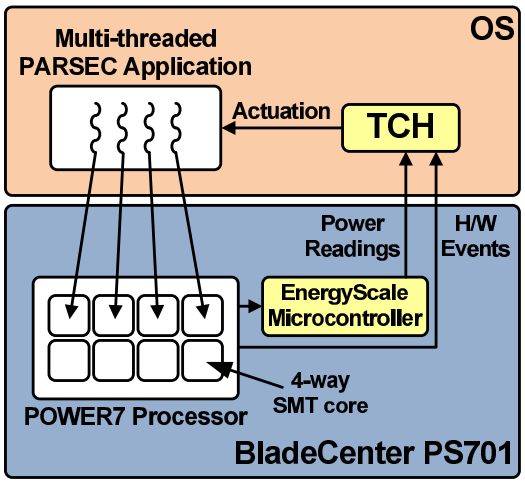


Figure 2.3: Experimental system for the evaluation of the proposed thread consolidation heuristic (TCH).

agement support [35, 63]. This subsystem, which is known as the EnergyScale firmware and microcontroller, senses the system at runtime providing a mechanism to control it based on user directives. The heuristic presented in this thesis leverages this energy management capability to collect chip power consumption information at runtime. The EnergyScale microcontroller is accessed through an interface known as the IBM Automated Measurement of Systems for Temperature and Energy Reporting software.

2.3.3 POWER7 Performance Monitoring Unit

The POWER7 processor incorporates a built-in performance monitoring unit (PMU) which makes possible to measure runtime performance through a set of six thread-level performance monitoring counters [32, 69]. POWER7 PMU provides an extensive list of more than 500 performance events that can be measured in the chip, such as cache miss rates, unit utilization, thread balance, hazard conditions, translation related misses, stall analysis, instruction mix, cache behavior and memory latency, among others. The six available counters can be programmed to "count" different event combinations. In this thesis, we make use of just three events, which are listed in Table 2.5. The heuristic presented in this thesis employs events `PM_RUN_INST_CMPL` and

PM_RUN_CYC to characterize performance as instructions per cycle (IPC), and event PM_L1_DCACHE_RELOAD_VALID to detect execution phase transitions.

Event	Description
PM_RUN_INST_CMPL	Number of instructions completed, gated by the run latch.
PM_RUN_CYC	Processor cycles gated by the run latch. Operating systems use the run latch to indicate when they are doing useful work. The run latch is typically cleared in the OS idle loop. Gating by the run latch filters out the idle loop.
PM_L1_DCACHE_RELOAD_VALID	The L1 data cache has been reloaded for demand loads, reported once per cache line.

Table 2.5: POWER7 performance events measured by the proposed heuristic.

2.3.4 Applications

We adopt the PARSEC 2.1 benchmark suite [12] to study the use of the optimum combination of core-wise SMT level and number of active cores as a knob to achieve a desired power-performance efficiency. PARSEC applications include a representative set of shared-memory parallel programs for chip-multiprocessors. Parallelism is supported by either POSIX threads (pthreads) [20], OpenMP [22] or Intel Threading Building Blocks (TBB) [50]. In this thesis, we adopt the POSIX threads version. Table 2.6 presents a description of the applications used in our experiments (we exclude *freqmine* due to its lack of support for POSIX threads). All the executions are done using *native* input sets — the largest ones provided in PARSEC and which resemble real program inputs most closely.

Program	**Domain**	**Description**
blackscholes	Financial	Option pricing kernel that uses the Black-Scholes partial differential equation
bodytrack	Computer Vision	Tracking of a person's body
canneal	Engineering	Simulated cache-aware annealing kernel which optimizes the routing cost of a chip design
dedup	Enterprise Storage	Next-generation compression kernel which employs data deduplication
facesim	Animation	Simulation of the motions of a human face for animation purposes
ferret	Similarity Search	Content similarity search server
fluidanimate	Animation	Fluid dynamics simulation for animation purposes
raytrace	Rendering	Real-time raytracing
streamcluster	Data Mining	On-line clustering
swaptions	Financial Analysis	Pricing of a portfolio of swaptions with the Heath-Jarrow-Morton framework
vips	Media Processing	Image processing application
x264	Media Processing	H.264 video encoding application

Table 2.6: Adopted PARSEC benchmarks (source: [12]).

Chapter 3

Last-Level Cache Design

This chapter presents a last-level cache (LLC) organization conscientiously designed for throughput-aware computation in chip multiprocessors (CMPs). As shown in Figure 3.1, the proposed LLC is divided into multiple independent *blocks*. Cores are grouped in clusters, and each cluster contains a LLC block connected to its local network. The most important characteristic of the LLC is the lack of data replication. Instead, the address space is interleaved in a fine-grained fashion across blocks. In this way, effective capacity of the LLC is significantly improved, which boosts memory bandwidth. In addition, since data is not replicated in the cache, there is no need to deal with data coherence.

The key insight of our proposal is that a fine-grain partitioning of the address space enables higher bandwidth, since multiple transfers can proceed in parallel, at the expense of locality, due to data being spread all over the available storage. While this trade-off is harmful for latency-aware architectures, it is beneficial for throughput-aware computation in CMPs.

In this chapter, we compare our proposal against a memory system optimized for latency. Both alternatives are evaluated on a throughput-aware CMP with a master-worker execution model and support for DMA transfers. The architecture is composed of processing elements (cores) grouped in clusters interconnected through a global network.

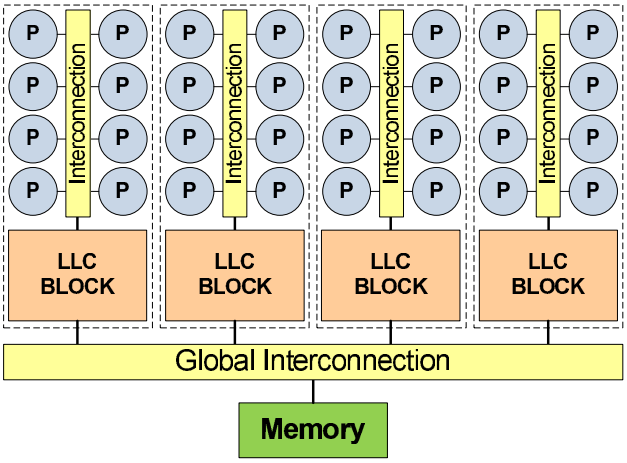


Figure 3.1: Illustrative 32-core CMP, with the proposed LLC organization. In this particular example, eight cores per cluster share a local LLC block. Because data is not replicated in the LLC, cores can also access remote blocks.

3.1 Latency- vs. Bandwidth-Optimized Last-Level Caches

The increment in the number of cores in current CMPs poses additional stress to the memory system [6]. The cache hierarchy plays a key role by providing low-latency access to the data, by means of data replication and coherence mechanisms. In particular, the last-level cache (LLC) helps to cut down the number of accesses to off-chip memory. For those reasons, we can find that recent throughput-oriented designs have adopted caches to alleviate the pressure imposed to the memory interface, whereas previous models in the same family did not. For example, the NVIDIA Fermi architecture [73, 76, 79] organizes its 512 cores in 16 clusters of 32 cores each. Inside each cluster there is a local memory that can act as an L1 cache. Unlike previous NVIDIA products (like the G80 and GT200), all clusters share a 768-KB L2 cache to capture temporal data reuse, and reduce off-chip memory traffic. Other designs were conceived with large LLCs from scratch. For example, the IBM PowerEN processor [17, 37, 57], a hybrid architecture for network and server processing, incorporates four 2-MB eDRAM L2 caches. In all those

designs, however, such caches are traditional organizations from the domain of latency-aware computation, optimized for latency rather than bandwidth.

From our point of view, the adoption of caches optimized for latency in the domain of throughput-aware computation has to be done thoroughly, and its different needs have to be taken into account. Latency-aware computation is latency bound. Instead, throughput-aware computation is bandwidth bound. To show this effect, we evaluate a CMP with a master-worker execution model and support for DMA transfers. The architecture is composed of worker processors (cores) grouped in clusters. Figure 3.2 shows a configuration with 32 cores. CMPs with 16, 64 and 128 cores were also considered. Besides the worker processors, there are a small number of high-performance master processors (not shown in the figure), which spawn tasks to be executed by workers.

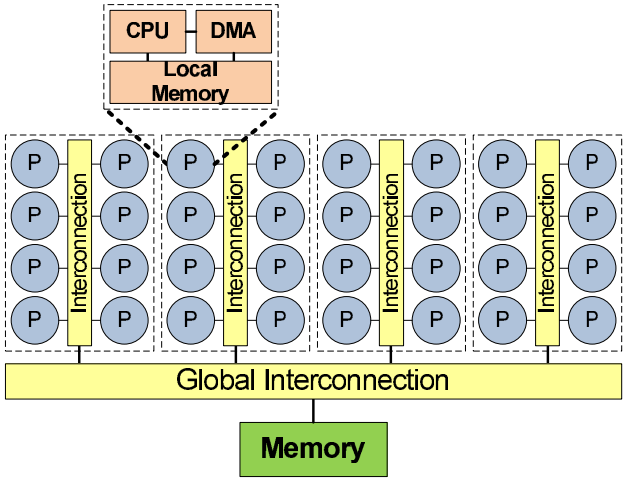


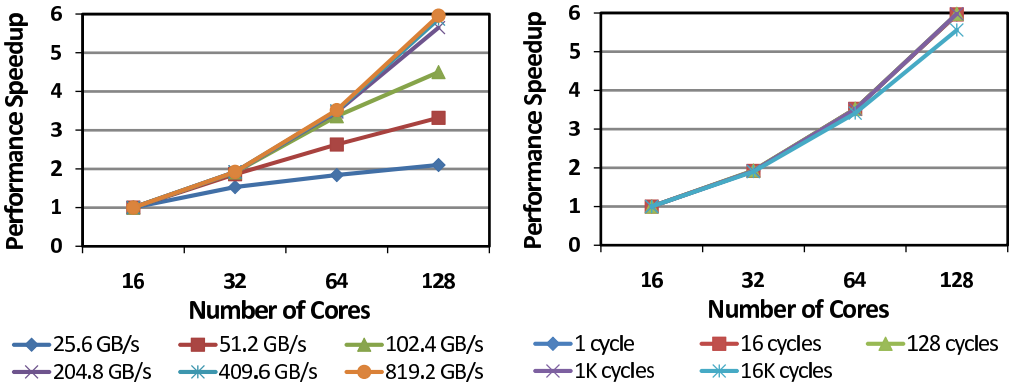
Figure 3.2: Evaluated DMA-based CMP architecture (LLC is not shown for sake of simplicity).

In this architecture, each worker consists of a low-power in-order CPU, a local memory and a DMA controller. The worker CPU can only access its local memory, which is used for both code and data. All external memory accesses are managed through a programmable DMA controller. Workers program the DMA controller to fetch the required input data for a task. In the same way, when the worker finishes the execution of a task, it programs

its DMA controller again to write back the task output data. After task execution, the worker notifies its availability to the master processor in order to receive a new task to be executed. This DMA-based design allows the architecture to fully support double buffering without any interference between execution and data prefetching of subsequent tasks. We use six parallel scientific kernels: Check-LU, Cholesky, FFT-3D, K-means, K-NN and MatMul. Figure 3.3(a) shows the impact on performance (total execution time) as a function of the number of cores, averaged for all considered kernels. In this case, we assume an ideal 0-cycle latency memory system. As it can be observed, the bandwidth provided by the memory system (chart series) has a very significant impact on the execution time. For instance, in the case of a memory system providing 25.6 GB/s, increasing the number of cores from 16 to 128 improves performance by 2.1×. However, if the memory system provides at least 204.8 GB/s, performance is improved by close to a factor of 6. Figure 3.3(b) also shows the impact on performance as a function of the number of cores, but considering different latencies (chart series), and assuming a high-bandwidth memory system. In this case, because memory bandwidth is large enough, the latency of the memory system has not impact on performance. Therefore, in throughput-aware scenarios, the memory system has to be designed to tackle the "bandwidth wall", instead of the "latency wall".

As a result, increasing latency in throughput-aware CMPs with DMA-based memory systems has little impact on performance. This is a key observation that hints potential optimizations in the cache hierarchy. For example, it may be not necessary to adopt cache coherence mechanisms with data replication in the LLC (usually found in latency-aware CMPs). Avoiding data coherence and replication reduces the LLC design time and complexity, and increases its effective capacity. Although processing elements should spend more time to get data which is not replicated in their local LLC blocks, this extra latency could be hidden by overlapping computation and DMA transfers.

In addition to the bandwidth wall problem, the number of pins to access off-chip memory is not growing at the same pace as the bandwidth require-



(a) Ideal 0-cycle latency memory system. (b) 819.2-GB/s high-bandwidth memory system.

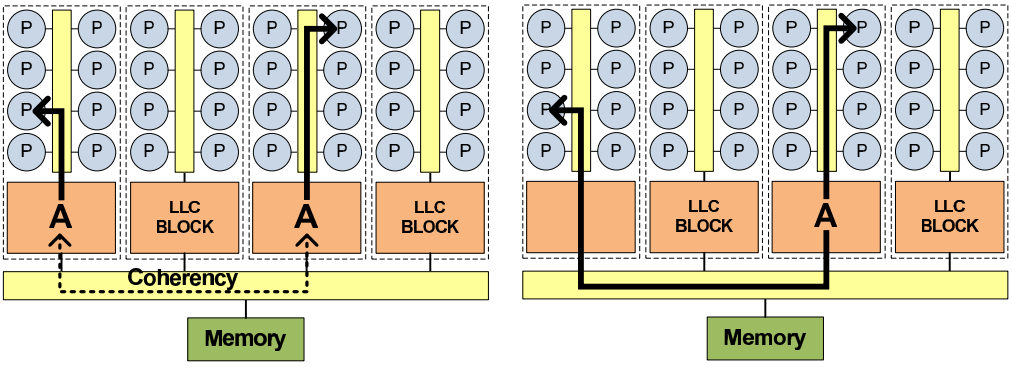
Figure 3.3: Performance impact as a function of the number of cores in a DMA-based CMP. Due to the huge amount of parallelism in the applications, memory access latency can be hidden, and data bandwidth becomes the limiting factor. Results are averaged for all considered kernels. Performance is measured as total execution time.

ments. According to ITRS projections [54], the pin count just grows about 10% per year. Even employing high-performance memory systems, that pin count increase is not enough to satisfy such bandwidth demands.

To overcome the *bandwidth* and *pin-count scaling* walls just mentioned, it becomes necessary to adopt caches in throughput-aware CMPs to filter off-chip memory traffic. In this sense, many points arise regarding the organization of the memory system in such scenarios:

- With data replication to optimize access latency (Figure 3.4(a)) vs. without data replication to optimize capacity and bandwidth (Figure 3.4(b)).
- If data is replicated across LLC blocks, the impact of the data coherence mechanism.
- If data is not replicated, how data should be distributed in the LLC (address space interleaving granularity), and what is its impact.
- Features from latency-aware cache systems that should (or should not)

be preserved when moving to the throughput-aware computation domain.



(a) Latency-aware organization. (b) Bandwidth-aware organization.

Figure 3.4: Evaluated last-level cache alternatives.

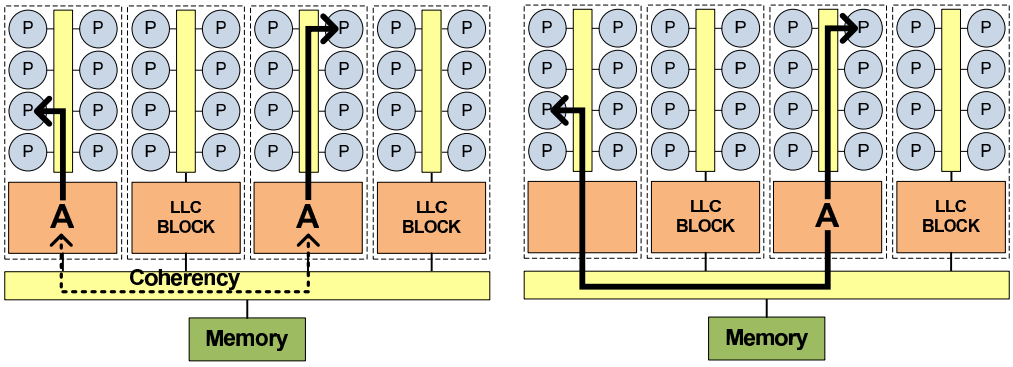
Those questions motivate us to present a cache hierarchy organization conscientiously designed for throughput-aware CMPs.

3.2 Last-level Cache Organization

As stated in Section 3.1, there is a trend to adopt latency-aware cache-based memory systems into the throughput-aware computation domain to alleviate the pressure on bandwidth requirements. Although such CMP architectures can benefit from caches, we should take into account that traditional cache systems are designed to optimize access latency rather than bandwidth. For that reason, in this work we evaluate two memory system alternatives in the context of throughput-aware computation: one optimized for access latency and another optimized for bandwidth and capacity (our proposal).

In this architecture, the LLC is divided into multiple blocks. Each cluster contains a cache block connected to its local network, as shown in Figure 3.5. Each cache block has its own independent Miss Status Holding Register (MSHR) and two full duplex ports (one for processor requests and responses, and the other for memory requests and responses). For the scheme

with data replication, we model an invalidation-based cache coherence protocol: when a CPU modifies a cache line, all the copies on other caches are invalidated. Cache access time has been determined using CACTI [102].



(a) Latency-aware organization. (b) Bandwidth-aware organization.

Figure 3.5: Evaluated last-level cache alternatives.

In the scheme optimized for latency, the LLC is composed of independent blocks and data can be replicated across them. For instance, in the example shown in Figure 3.5(a), data "A" is replicated and always accessed locally. In this way, cores have nearby copies of data in order to reduce the latency to access the cache. However, having multiple copies of data diminishes the effective capacity of the cache and makes it necessary to implement a coherence mechanism.

In the scheme optimized for bandwidth, the LLC is also composed of a set of independent blocks but neither multiple copies nor block migration between them is allowed. In the example shown in Figure 3.5(b), there is a single instance of data "A" and some cores have to access it remotely, thus paying an additional latency. In this scenario, data is statically placed using bit indexing: a set of bits in the data address is used to choose unambiguously the cache block where the data is placed. A detailed analysis of this static-placement technique is presented in Section 3.2.1.

3.2.1 Interleaving Granularity

In our proposal, the address space is partitioned between LLC blocks based on the data address. More specifically, for an N-block LLC, we employ $log_2(N)$ bits to determine the destination block (Figure 3.6). Taking the $log_2(N)$ least significant address bits, data placement is interleaved at a fine granularity across blocks. On the other end, taking the $log_2(N)$ most significant bits, the placement is interleaved with coarse granularity. Also, any point in between can be chosen.

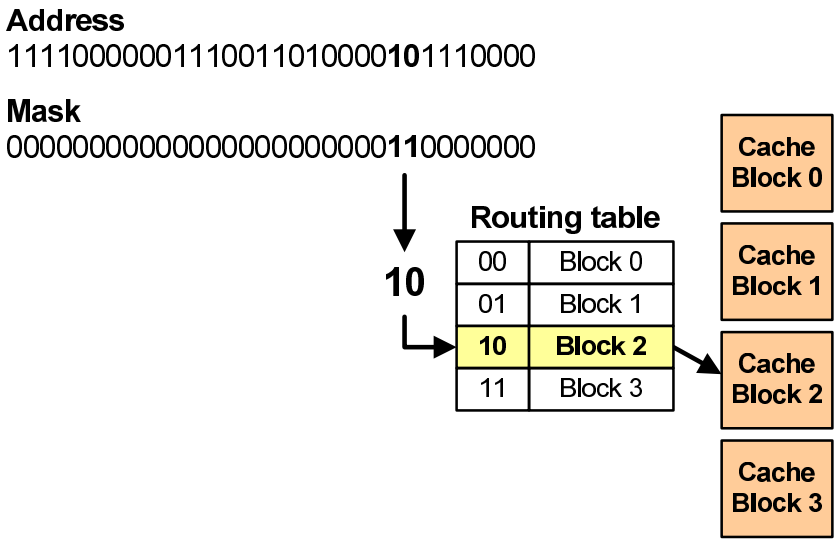


Figure 3.6: Example of the mechanism employed in our LLC to map a particular memory address into a LLC block.

Figure 3.7 shows the impact on bandwidth and performance (total execution time) attained by the worker processors as a function of the interleaving granularity, averaged for all evaluated applications. Interleaving granularities range from 128 bytes (the baseline) to 32 KB. In each chart, 16, 32, 64 and 128 cores are plotted in a 64-MB LLC configuration. We can observe that the finest-grained interleaving (128 bytes) is the best choice for the considered numbers of cores. In particular, for 128 cores there is an improvement of 10% on bandwidth and 4% on performance using a 128-byte interleaving with respect to the typically used 4-KB interleaving, and

34% on bandwidth and 13% on performance when compared against a 32-KB interleaving. Fine-grained interleaving shows the highest DMA bandwidth and performance because it allows highly-balanced access to the cache blocks. With coarse-grained interleaving, a cache block could be stressed by all processors simultaneously during short periods of time, thus becoming a bottleneck.

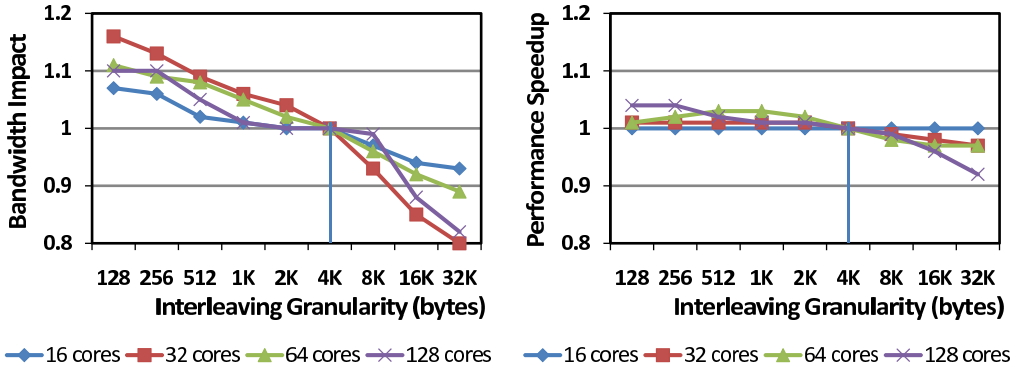


Figure 3.7: Impact on (a) bandwidth and (b) performance of last-level cache interleaving granularity. Results are normalized to the 4-KB interleaving case. Performance is measured as total execution time.

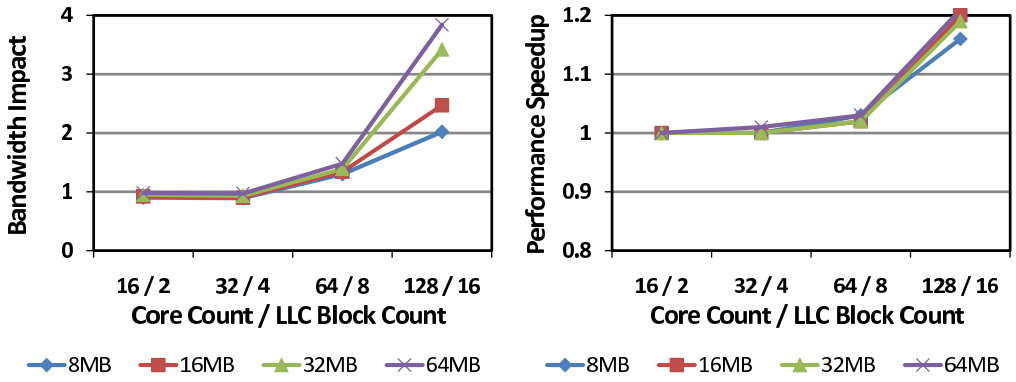
For the rest of this chapter, we adopt the 128-byte fine-grained granularity to interleave the address-space across cache blocks because it shows the highest DMA bandwidth and performance.

3.2.2 Partitioned vs. With-Replication Scheme

In this section, we compare a 128-byte interleaved partitioned LLC (the one that provides the highest DMA bandwidth) against a latency-aware LLC with data replication.

Figure 3.8(a) shows the relative bandwidth for the partitioned LLC with respect to the organization with data replication, as a function of the number of cores (workers). As it can be observed, the bandwidth improvement for the scenario with partitioned LLC significantly increases with the number of cores: for a 64-MB LLC and 64 workers, the bandwidth is 1.5× better, while

for 128 workers it is almost 4× better. This is because there are (potentially) more data replication and coherence invalidations in the LLC optimized for latency, as the number of LLC blocks increases. Figure 3.8(b) shows the performance speedup (i.e. total execution time reduction) for the partitioned LLC with respect to the organization with data replication, for the same experiment. In this case, both alternatives present similar performance for 64 workers or less. But for 128 workers, the scenario with partitioned LLC shows an improvement near 20% with respect to the LLC with data replication. We expect this gap to continue growing for larger numbers of workers, because bandwidth (and not latency) becomes more and more critical as the number of cores increases.



(a) Relative bandwidth.

(b) Performance speedup.

Figure 3.8: Partitioned LLC vs. an organization with data replication. For a particular number of workers (each point in X-axis), results are with respect to the organization with data replication. Performance is measured as total execution time.

As expected, the partitioned LLC scheme is better for throughput-aware CMPs. The advantage of replicated caches is an increased locality and, hence, a lower access latency. However, as we have shown in Figure 3.3(b), latency is irrelevant in throughput-aware scenarios. In such cases, the partitioned scheme provides higher bandwidth and better off-chip filtering. Figure 3.9 shows the hit rate for both alternatives, adopting a 64-MB LLC. As it can be observed, the partitioned LLC outperforms the scheme with data replication. Moreover, the hit rate changes slightly as the number of cores increases,

while in the LLC with replication the hit rate is significantly affected by the number of cores. This is due to the fact that, as the number of cores increases, there are more LLC blocks and, hence, more data replication. Furthermore, in the scheme with replication, a core may invalidate a cache line that may potentially be useful for other cores. In the same figure, the hit rate on invalidated lines is plotted on top of each bar for the LLC with replication. The number of hits on invalidated lines still present in the LLC is not negligible — it can be around 6 $\sim$ 7% of the accesses in some cases (16, 32 and 64 cores).

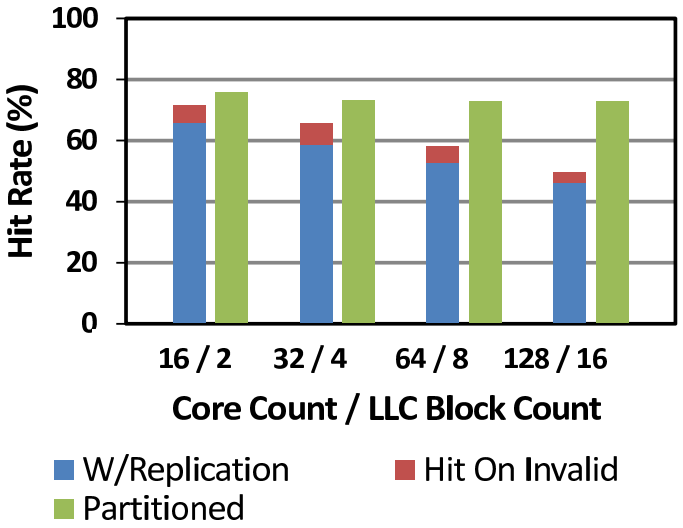
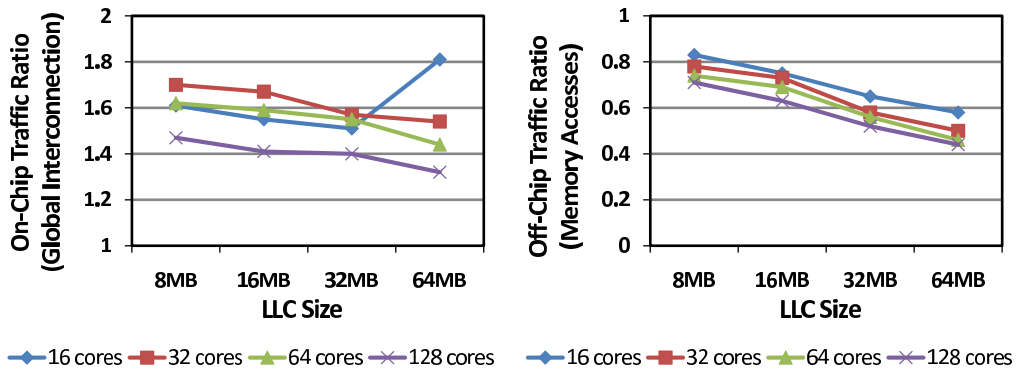


Figure 3.9: Hit rate as a function of the number of cores, averaged for all considered kernels. Results correspond to a LLC with data replication and data coherence ("W/Replication") and the proposed partitioned LLC ("Partitioned").

At first glance, the high amount of traffic in the global interconnection appears as the main drawback of a partitioned LLC with fine-grained interleaving: every LLC access from a core will evenly access all the LLC blocks, the local one (in its own cluster) and the remote ones. To evaluate this issue, we measure the number of packets in the global interconnection for both LLC alternatives. For the partitioned LLC, traffic is composed of accesses to remote LLC blocks (due to the partitioning) and requests to memory.

For the LLC with replication, besides memory requests, we also consider coherence traffic (accesses to remote shared lines and invalidation requests). Figure 3.10(a) shows the traffic ratio for the partitioned LLC with respect to the LLC with replication, as a function of the LLC size. As shown in the chart, the traffic increase for the partitioned LLC is not as high as we could expect if we take into account the benefits obtained in bandwidth and performance. For instance, in the scenario with a 64-MB LLC and 128 cores, the traffic in the global interconnection is 32% higher, but the partitioned cache provides 3.8× improvement on bandwidth and 1.2× improvement on performance. Likewise remarkable is the trend: the traffic ratio decreases as the cache size or number of cores grow, that is what we expect to happen in future CMPs.



(a) On-chip traffic (global interconnection). (b) Off-chip traffic (memory accesses).

Figure 3.10: Partitioned LLC vs. an organization with data replication. For a particular LLC size (each point in X-axis), results are with respect to the organization with data replication.

One of the most important characteristics of our proposal is the off-chip traffic filtering due to a better use of the LLC capacity. Figure 3.10(b) shows the off-chip traffic ratio for the partitioned LLC with respect to the LLC with replication for the same experiment. As it can be observed, the number of off-chip accesses is substantially reduced and this cut is more noticeable as the number of cores and LLC size are increased (expected trend for future CMPs). This is a very desirable effect because off-chip memory access is one

of the main sources of system power consumption.

3.2.3 No-Write-Allocate Optimization

As previously mentioned, caches for latency-aware computation are designed to minimize access latency. That means that some optimizations implemented in such organizations do not necessarily perform well on throughput-aware domains. For instance, write allocation is intended for latency reduction when a write access misses the cache. However, in DMA-based CMPs, DMA accesses are usually split into requests that are the same size as cache lines. In such scenario, if a write access misses the cache, allocating the line from memory has no impact because the line will be rewritten completely. Deactivating the write-allocate policy in a DMA-based CMP decreases off-chip traffic. Figure 3.11 shows the extra number of memory accesses when the write-allocate policy is activated. In some cases, the additional off-chip traffic could be significant: for a 128-core CMP with a 8-MB LLC, the traffic to memory is almost 10% higher when write allocation is employed. When the LLC size is increased, the additional off-chip traffic decreases because most of the accesses hit in the cache — but even for a 64-MB LLC, it is still over 2%. This is extra traffic that can be eliminated by just deactivating an optimization that works in latency-aware scenarios but not in throughput-aware designs.

3.3 Off-chip Memory Organization

To improve the off-chip memory bandwidth, we adopt an approach similar to the one used for the LLC: to access the off-chip memory, we employ more than one memory controllers, with several channels each, and the address space is interleaved across channels and memory controllers. Although this is not a novel technique [19, 94, 99], in this work we analyze what is the optimal interleaving granularity to obtain the highest bandwidth.

In the scenario evaluated in this section, we consider a 64-MB LLC with fine-grained (128 bytes) interleaving granularity with one LLC block per

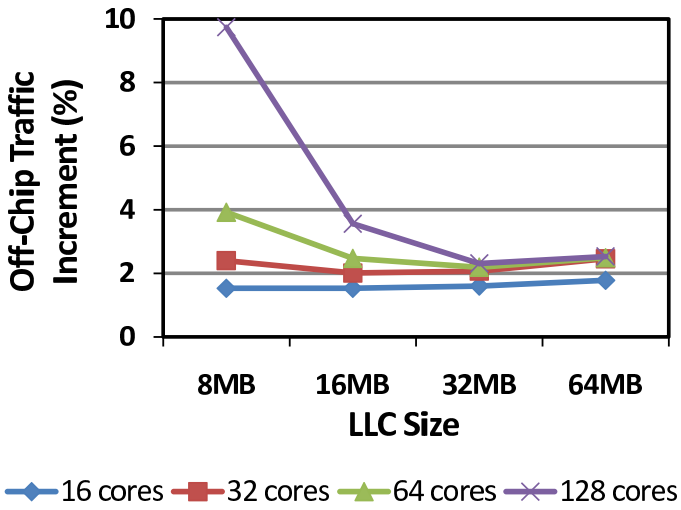
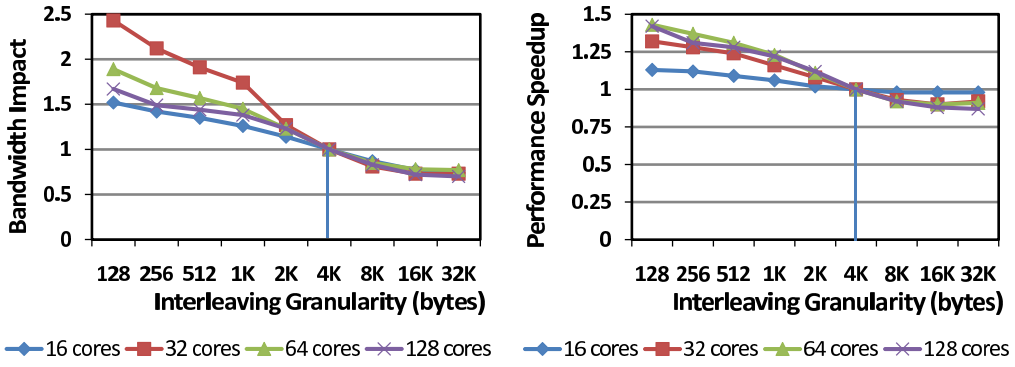


Figure 3.11: Increment in the number of off-chip accesses when write allocation is enabled. For a particular LLC size (each point in X-axis), results are with respect to the organization with write-allocation disabled.

cluster. The configurations evaluated cover scenarios with 16, 32, 64 and 128 cores (workers), and 2, 4, 8 and 16 LLC blocks. Figure 3.12(a) shows the normalized DMA bandwidth obtained by the cores as the memory interleaving granularity is increased from 128 bytes to 32 KB (baseline: 4 KB). As it can be observed for a 128-core CMP, the 128-byte interleaving provides 67% more bandwidth than the 4-KB interleaving, which is the most commonly used in current memory systems; and 238% more bandwidth when compared against the 32-KB one. Similar results can be seen for performance in Figure 3.12(b). Once again, the finest-grained interleaving granularity presents the best result with a 42% performance improvement (i.e. total execution time reduction) than the 4-KB interleaving, and a 63% performance improvement compared to the 32-KB one.

Similarly to what was observed for last-level caches in Section 3.2, fine-grained interleaving also provides the highest DMA bandwidth when employed to access off-chip memory, because it allows highly-balanced access to memory controllers and channels. Therefore, we conclude from our experiments that a complete memory system thoroughly optimized at both levels,



(a) (b)

Figure 3.12: Impact on (a) bandwidth and (b) performance of memory interleaving granularity. Results are normalized to the 4-KB interleaving case. A 64-MB LLC with 128-byte interleaving is considered. Performance is measured as total execution time.

LLC and memory, provides significant improvements on bandwidth and performance for throughput-aware CMPs rather than traditional latency-aware memory organizations.

3.4 Summary and Concluding Remarks

Throughput-aware CMPs have rapidly become an important specimen in the multicore ecosystem. In such designs, the increase in the number of cores pushes up the bandwidth demand to access off-chip memory. For that reason, recent throughput-oriented architectures have adopted caches to alleviate the pressure imposed to the memory interface. But those caches are conceived to improve latency, not bandwidth.

In this chapter, we presented a re-design of the memory system targeting throughput-aware computation. Instead of a traditional latency-aware cache, we proposed to spread the address space using fine-grained interleaving all over a shared non-coherent LLC. In this way, on-chip storage is optimally used, with no need to keep coherence. On the memory side, we also proposed the use of interleaving across DRAMs but with a much finer granularity than usual page-size approaches. All these optimizations synergistically provide

significant improvements on bandwidth and performance. For a CMP with 128 cores and a 64-MB LLC, our proposal shows a 3.8× improvement on bandwidth due to the LLC organization, and an additional 1.7× improvement due to the DRAM organization (128-byte instead of 4-KB interleaving), which stack together for a total 6.4× bandwidth improvement. For performance, our proposal shows a 1.2× improvement due to the LLC organization, and an additional 1.4× improvement due to the DRAM organization, which stack together for a total 1.7× performance improvement (i.e. total execution time reduction).

The trend is also remarkable: bandwidth and performance improvements become more significant with the increase in the number of cores or LLC size. The higher the system size (in terms of number of cores and LLC size), the greater the number of blocks the LLC has to be split into. In a latency-aware LLC, that means more data replication and less efficient use of effective capacity, while in our proposal the effective capacity is not affected by replication. Moreover, cores can evenly access all blocks in our LLC organization.

Additionally, we found the write-allocate optimization from latency-aware caches not to be worth applying on DMA-based CMPs. In such cases, write cache misses usually are the same size as the cache line. Therefore, allocating the missing line from memory not only does not help, but it adds extra off-chip traffic. In the proposed organization, the write-allocate optimization is deactivated. For 128 cores, our results show close to 10% higher off-chip traffic when the write-allocate policy is employed. When the LLC size is increased, the additional off-chip traffic decreases because most of the accesses hit in the cache — but even for a 64-MB LLC, it is still over 2%.

Chapter 4

Register File Design

DMA-based CMPs (as the one evaluated in the previous chapter) move data from the cache hierarchy into each processor's local memory by means of DMA commands. Once in its local memory, a processor moves data into the register file to satisfy on-the-fly instructions' input operands. From our point of view, the adoption of local memories in these scenarios can be avoided by significantly enlarging the architected register file (e.g. thousands of registers). This approach presents the following advantages:

- It reduces the cost associated with data movement between the local memory and the register file. Instead, a processor directly brings data from the cache hierarchy into the register file, as in a traditional design, and sends back results to the cache hierarchy once data is not needed anymore. If the register file is large enough, a processor can move large blocks of input data into the register file, which is a desirable feature in data-hungry throughput-aware applications.

- From a programming model perspective, register files are easier to use with respect to software-managed streaming memories. In a register file, conventional load/store instructions move data between the register file and the cache hierarchy. Coherence is managed by hardware, and there is no need for software to have knowledge of the precise location of the data being accessed. On the other hand, a software-managed streaming memory requires special instructions to move data between

the cache hierarchy and the local memories, and coherence (if present) is explicitly managed at software level.

- Data placement and manipulation is also more flexible in a register file than in a local memory, because it can be supported by regular instructions in the Instruction Set Architecture (ISA). This is another desirable feature because throughput-aware applications usually present phases during which data has to be shuffled or interleaved in some particular way.

In addition to these benefits, a very-large register file can also be used for *double-buffered* computation. This technique —usually implemented with DMA engines— is key to hide the memory access latency by overlapping computation and data transferring. To enable double buffering, DMA-based systems resort to large local memories to keep both, the data being processed and the data being prefetched from the memory system to be processed next. Double buffering can still be implemented with the adoption of a very-large register file. The evaluation of a very-large register file as a means to use double buffering in DMA-based CMPs is beyond the scope of this thesis. Instead, we study the virtues of the proposed register file organization in the context of load-store CMP architecture.

Based on these observations, in this chapter we study the possibility of significantly re-architecting the user-addressable register file organization, as well as the impact of a very-large vector register file in the context of throughput-aware computation. The register file considered in this thesis was presented by Derby et al. in [29] in the context of an in-line accelerator. In this thesis, we propose to implement such organization with multiple banks to keep wire propagation delay under control and exploit local computation in each bank. As we explain in Section 4.5, this processing capability at register file level is implemented with small, SIMD computation elements attached to each bank.

Throughout this chapter, we study the benefits of the proposed very-large register file in the context of wireless base stations for the third generation (3G) and fourth generation (4G) wireless telecommunication standards.

These applications are throughput demanding and heavily bandwidth bound which make them suitable to study the kind of optimizations presented in this thesis. The approach involves exploiting the throughput computation capabilities of the IBM PowerEN processor [17, 37, 57] (a multicore, massively multithreaded platform) augmented with a layer of in-line universal acceleration support. These in-line accelerators are incorporated within the cores and include the proposed register file with embedded SIMD support.

4.1 Wireless Base Stations

In wireless networks, *base stations* play a key role as an intermediary between mobile devices and wired networks. With the rapid proliferation of smartphones, tablets and other mobile broadband devices, base stations are responsible for operating on large amounts of traffic at high speed rates (in the order of 1Gbps for 4G). Therefore, it is crucial to provide base stations with enough computation resources to satisfy such demand on speed and throughput, in the face of the new coming standards.

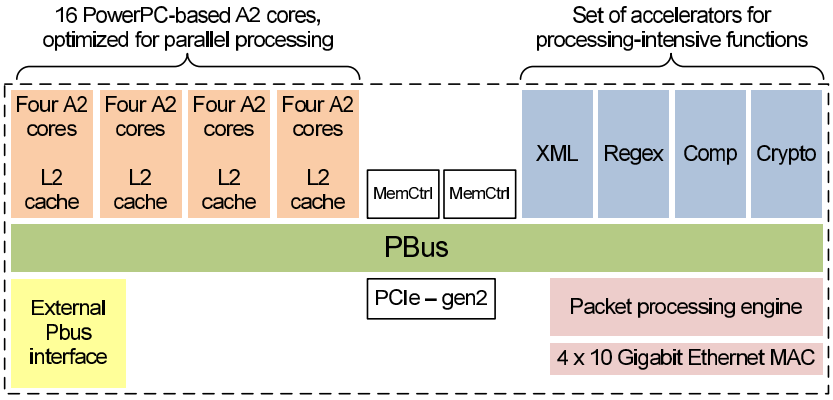
Since the release of the PowerEN processor in 2010, IBM is playing an important role in the development of suitable hardware to meet the computation requirements in the network processing domain. In this chapter, we analyze the potential benefits that the PowerEN processor could provide for wireless communication systems, particularly base stations. We leverage features from PowerEN such as massively multithreaded design and high-bandwidth networking interfaces. Furthermore, we replace the bus-attached special-function accelerators by *in-line universal* accelerators. Instead of being connected to the bus, the in-line accelerators are incorporated within all the cores or a selected subset thereof. The benefits of this approach are twofold. The in-line accelerator can be used as a traditional SIMD unit incorporated in a general-purpose processor, avoiding the communication overhead when employing off-line (bus-attached) accelerators. In addition, from a programming model perspective, in-line accelerators are easier to use because the instructions that drive the accelerator are part of the same stream that drives the host processor.

In the context of a single core, in this thesis we focus on the in-line vector-based accelerator (VBA) design, which builds on the *indirect* VMX (iVMX) architecture [29]. A key constituent of the proposed VBA is the Vector String Register File (VSRF). It is a very-large register file which helps to significantly reduce the number of memory access instructions. To keep wire propagation delay under control, we conceive the VSRF as an aggregation of banks. Such organization unveils an additional optimization opportunity: SIMD computation support embedded into the register file. This *processor-in-regfile* (PIR) strategy is implemented as small special-function *local computation elements* (LCEs) attached to each bank. With this approach, the limited number of register file ports is overcome. Each LCE is a SIMD computation element, and all of them can proceed concurrently. Therefore, the PIR strategy constitutes a highly-parallel super-wide-SIMD device, ideal for throughput-aware computation. In order to target a broader spectrum of throughput-aware applications in base stations, we also analyze the feasibility of the PIR architecture implementation based on reconfigurable LCEs.

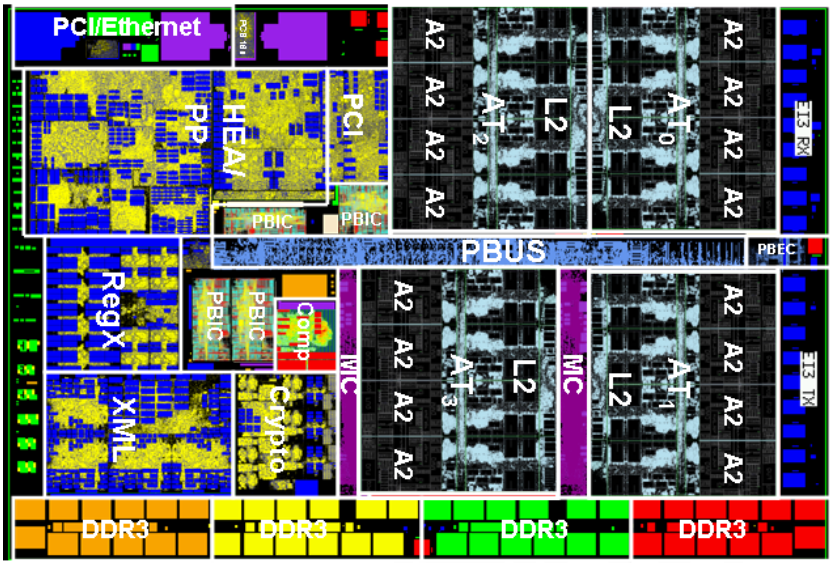
4.2 IBM PowerEN Processor

The PowerEN architecture targets highly-parallel applications, providing acceleration for network domains. As shown in Figures 4.1(a) and 4.1(b), a PowerEN chip is composed of 16 PowerPC-based A2 cores (4-way SMT each) grouped into four clusters, with a shared 2-MB L2 cache in each cluster. Additionally, there are four bus-attached hardware accelerators for XML, pattern-matching, compression/decompression and cryptography. The cores and accelerators are connected through an interconnection fabric (PBus), which is also responsible for data coherence among the L2 caches. As announced in 2010 [57], the PowerEN processor is a 2.3GHz, 45nm SOI chip with a die size of 428 mm^2.

At first glance, we would consider adding new hardware accelerators attached to the PBus to target applications such as FFT and Turbo Decoding. In the physical layer in an LTE base station, the signal going through the downlink or uplink chains is processed with a sequence of functions (e.g.



(a) Block diagram (source: [37]).



(b) Die photo (source: [57]).

Figure 4.1: IBM PowerEN processor.

FFT → channel estimation → Turbo Decoding → CRC checking). As we discuss in Section 4.4.1, a programming model based on in-line accelerators would be more suitable for base stations to avoid the coordination and data movement between successive functions. Therefore, we consider replacing the bus-attached, globally-shared accelerators by *in-line* accelerators incorporated within all the (or a selected subset of) A2 cores.

The envisaged design for the in-line accelerator would be such that it could be used to compute different functions (e.g. FFT, Turbo Decoding, etc).

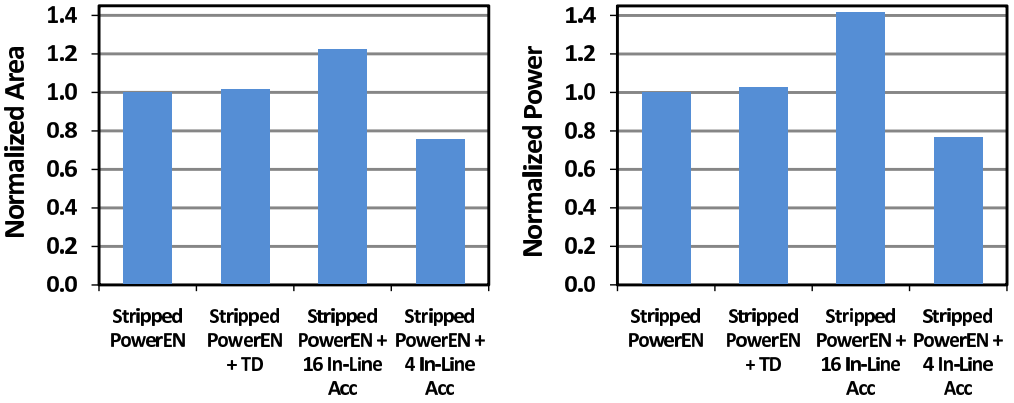
This approach, if deemed attractive from an area, performance and energy efficiency viewpoint, ostensibly also reduces development complexity in that the multiple accelerator design efforts can be merged into a single (universal) accelerator design. The code for such in-line accelerator can be written using a completely traditional programming model, with data resident in memory and loaded to the VSRF as needed.

4.3 Area and Power Implications

In this section, we analyze the impact in terms of area and power consumption due to the adoption of in-line universal accelerators. The PowerEN chip has a die size of 428 mm^2. We first remove the four hardware accelerators (XML, pattern-matching, compression/decompression and cryptography) because they are not required in the base station computation domain ("Stripped PowerEN" configuration in Figure 4.2(a)). By connecting just one Turbo Decoding accelerator to the PBus ("Stripped PowerEN + TD" configuration), system-on-chip (SoC) area increases by 1.5%, based on an optimistic estimate for 45nm. On the other hand, by incorporating 16 in-line universal accelerators ("Stripped PowerEN + 16 In-Line Acc" configuration), SoC area increases by 22%. However, the latter configuration implies 16 times more acceleration capability than the one with just one Turbo Decoding accelerator connected to the PBus. Moreover, the configuration based on in-line accelerators constitutes a fully-programmable, scalable design for a much broader range of applications.

Based on a preliminary assessment for the LTE standard, the complete digital baseband for a single sector supporting a 4x4 multiple-input multiple-output (MIMO) antenna configuration in a 20 MHz channel can be implemented with just two to three A2 cores, each with an in-line accelerator attached [28]. Such configuration, which is suitable for pico and femto base stations, presents an area that is almost 40% smaller than the one with 16 in-line accelerators ("Stripped PowerEN + 4 In-Line Acc" in Figure 4.2(a)).

Figure 4.2(b) presents a comparison in terms of power consumption for the same four configurations. Connecting one Turbo Decoding accelerator to



(a) Area comparison (b) Power comparison

Figure 4.2: Normalized area and power comparison for four particular configurations: a PowerEN without bus-attached accelerators ("Stripped PowerEN"), a stripped PowerEN with a bus-attached Turbo Decoding accelerator ("Stripped PowerEN + TD"), a stripped PowerEN with 16 in-line universal accelerators ("Stripped PowerEN + 16 In-Line Acc") and a stripped PowerEN with 4 in-line universal accelerators to target pico and femto base stations ("Stripped PowerEN + 4 In-Line Acc").

the bus ('Stripped PowerEN + TD") increases power consumption by 2.6% with respect to the baseline stripped PowerEN SoC. By incorporating 16 in-line universal accelerators ("Stripped PowerEN + 16 In-Line Acc"), SoC power increases by 25%, but providing 16 times more fully-programmable acceleration capability than the one with just one Turbo Decoding accelerator. Finally, the configuration suitable for pico and femto base stations ("Stripped PowerEN + 4 In-Line Acc") presents a 46% reduction in power consumption compared to the one with 16 in-line accelerators.

The comparison presented above is conservative. Area and power consumption can be further reduced when memory bandwidth is also taken into account. To feed 16 A2 cores, the original PowerEN chip resorts to two independent DDR3 memory controllers, each one with two independent channels [57]. However, the configuration "Stripped PowerEN + 4 In-Line Acc" proposed for base stations incorporates just one-fourth the number of A2 cores. Even more, the in-line accelerators within those cores help to reduce the memory bandwidth pressure. Based on this observation, we may expect

memory bandwidth demands to be one-fourth (or less) of the bandwidth required in the original PowerEN chip. Therefore, we can do without one of the two DDR3 memory controllers. In this case, the "Stripped PowerEN + 4 In-Line Acc" configuration area is 44% smaller and power consumption is 51% less than the "Stripped PowerEN + 16 In-Line Acc" configuration.

From the analysis done in this section, it is possible to infer that the PowerEN architecture has a great potential to target base stations. By adapting it for pico and femtocells, with four cores and four in-line universal accelerators, significant savings in area and power consumption can be achieved.

4.4 Universal In-Line Accelerator

The vector-based accelerator (VBA) presented in this chapter is an auxiliary processing unit attached to an A2 core, built upon the *indirect* VMX (iVMX) architecture [29]. It takes and executes instructions fetched and passed to it by the core, and provides essentially the same computational facilities as VMX. The accelerator includes an extremely large register file, the VSRF, which can be accessed using an indirection mechanism based on register mappings. In this chapter, we consider a 2048-entry VSRF implementation. Each register is 256-bit wide with corresponding subword parallelism (e.g. 16-wide for 16-bit halfwords, 8-wide for 32-bit fullwords). The VBA incorporates low-latency gather operations to operate on the register file data. These operations permit access to up to eight data elements at arbitrary locations in the VSRF with a single *gather* instruction.

In [85], Rico et al. present the virtues of an in-line accelerator with a VSRF, compared to a standard VMX baseline implementation. Applications benefit from the VSRF size to load large amounts of data at once, operate locally for a long period and store the results at the end. VMX codes, instead, require to load and store data much more frequently due to the limited amount of registers. Thus, the adoption of a very-large register file leads to significantly less vector load and vector store instructions compared to having a regular 32-entry register file. Moreover, if the result is to be used as the input to another function, there is no need to store the output into

the cache hierarchy.

As shown in [85], the reduction in instruction counts in the VBA+VSRF case and its CPI improvements over VMX result in significant speedups (up to 80% reduction in execution time). On the other hand, power consumption in the VBA+VSRF case is higher due to the larger area and additional logic for indirection. However, the lower CPI and the reduced off-chip traffic lead in most of the cases to a better *energy per instruction* consumption compared to VMX. This fact, in conjunction with the significant code reduction, results in very large energy savings. Taking into account that electricity consumption of telecommunications infrastructure grows 16% per year, and that 80% of such energy is just consumed by base stations [34], energy efficiency is a highly desirable attribute for any SoC design for base stations.

4.4.1 Programming Model

The VBA can be programmed using a completely traditional programming model, with memory-resident data loaded and stored as needed. However, this approach underutilizes the VSRF. Substantial performance gains can be obtained by loading relatively large blocks of data into the VSRF a cache-line at a time, operating on the entire block of data, keeping intermediate results in the VSRF, and storing the final results to memory a cache-line at a time. The VBA provides further advantage when the produced result is used as the input to another function. For instance, it can pass the FFT output into subcarrier demapping in an LTE uplink receiver. In this case there is no need to store the output of the FFT; the next function is merely given a pointer to its input within the VSRF.

In a SoC with bus-attached accelerators, cores offload tasks to those hardware accelerators, with frequent coordination and data movement between them. With the in-line acceleration model enabled by the VBA, a sequence of tasks is implemented in software in a single A2 core with VBA. Data remains local (ideally in the VSRF and if necessary in the local L2 cache) and coordination is handled through normal program flow.

Two additional points with respect to the programming model are worth

noting. First, it remains a load/store model, with load and store instructions used to move data in and out of a vector unit's VSRF. Given the PowerEN's hardware-managed coherence, there is no need for software to have knowledge of the precise location of a block of data to be accessed. And, second, the use of the VBA enables solutions that are highly scalable, since each VBA can provide acceleration for any of the desired functions, depending on the code it executes.

4.4.2 Why Such a Large Register File?

The VSRF contains more storage than the L1 data cache found in most processors. A natural question is why not use the VSRF silicon to augment the cache storage hierarchy "seen" by the CPU. First of all, the VSRF is in fact a register file, in that it can supply the contents of up to four on-the-fly instruction operands per cycle. Its access latency is completely hidden by pipelining and bypassing. Instead, an L1 cache has fewer ports and higher access latency.

The VSRF also provides fixed access latency for reading or writing a register. This is crucial for computation at physical layer in base stations. As 3GPP specifies for the LTE standard [3], a radio frame is received every 10 ms, partitioned into ten subframes of 1 ms each, and each subframe partitioned into two 0.5 ms slots. In each slot, seven orthogonal frequency-division multiplexing (OFDM) symbols are received, each one containing a little over 2000 complex samples. Each received symbol has to be processed on time, to avoid significant performance degradation and situations where recovery may be extremely difficult (if not impossible). A design with a larger cache instead of the VSRF, would be more prone to fall behind the processing deadline, e.g. because of a burst of unexpected cache misses.

To enable application optimizations, the VSRF provides much more data placement flexibility with respect to a cache. This feature gives to the programmer the means to split data for parallel processing, as it will be shown in Sections 4.4.3 and 4.5.1. Moreover, data in a cache is at risk to being evicted while it is still needed.

In the next section, we discuss a radix-8 FFT implementation which exploits the large capacity, low-latency access and flexible data placement of the VSRF.

4.4.3 Case Study: Fast Fourier Transform

Cooley-Turkey FFT algorithms [25] are used extensively in LTE. These algorithms are commonly implemented using fixed-point arithmetic, with 16 bits each for the real and imaginary parts of the data array. With interleaved real and imaginary parts at 16 bits each, the FFT algorithm "sees" the VBA as providing an 8-wide SIMD. There is a natural affinity between the radix-8 FFT and an 8-wide SIMD, in that eight radix-8 butterflies can be executed in parallel in place throughout the FFT, with just one step of shuffling the data array with base-8-digit-reversed indexing. Moreover, a very simple and clean implementation of the shuffling is possible given the capabilities of the VBA.

Considering a 512-point FFT, the radix-8 algorithm computes $log_8(512) = 3$ stages, each one with eight 8-wide radix-8 butterflies. The data array occupies 64 vector registers. A decimation-in-time (DIT) implementation on the proposed in-line accelerator proceeds as follows:

1. The data array is accessed in sequential fashion for the first stage, which requires no twiddle factors. The first stage requires about 120 cycles.

2. Groups of eight vectors are transposed in the VSRF, using a sequence of gather instructions. Each 8x8 transpose requires 8 cycles.

3. The shuffled intermediate data array from step 2 is the input to the second stage. The access pattern for the array completes the base-8-digit-reversed indexing. This access pattern is implemented by construction and updating of appropriate sets of pointers in map registers. The second stage, including multiplication by twiddle factors and the necessary map management, completes in about 160 cycles. The data array at the output of this stage is in the VSRF in natural order.

4. The third stage completes the FFT. Including multiplication by twiddle factors and the appropriate map management, it completes in about 160 cycles.

The complete 512-point FFT executes in about 550 cycles. This assumes that the data and twiddle arrays are already in the VSRF at the outset and that the transformed data array remains in the VSRF at the end. The overhead to load the data and twiddles and to store the result may increase the cycle count by 15%. However, in the LTE layer 1, FFTs represent one step (or a set of steps) in a sequence of functions applied to the baseband signals. With the in-line programming model supported by the VBA and the very-large register file, it is realistic to consider the FFT requiring neither loads from memory nor stores to memory.

The implementation presented here is based on Pease's method [93]. This algorithm is explicitly optimized for parallel FFT computation and requires just one step of data shuffling. This observation is particularly important, because FFT scalability is usually limited by the data shuffling cost, which increases significantly with the FFT size. By adopting Pease's method, the scaling of processing time from 512 points to 1024 and 2048 points (sizes commonly found in current base stations) is very close to $N \times log_2(N)$. For instance, a 2048-point fixed-point FFT executes in about 2500 cycles in the VBA; i.e. 1884 millions of samples per second (Msps) at 2.3GHz. In comparison to the state-of-the-art digital signal processor (DSP) solutions [38, 100], our simulation-based throughput estimation is 1.5 to 3.5 times higher, for the same clock frequency. Even more, the design proposed constitutes a fully-programmable, scalable design for a much broader range of applications than DSPs.

4.5 Processor-in-Regfile Strategy

In the context of throughput-aware applications, exploiting the large amount of parallelism would require moving a lot of data from the register file to the computation resources and vice versa. In that case, the available register

file ports would not be enough to keep the A2 core busy. To alleviate the pressure on the register file interface, in this thesis we propose to embed part of the computation resources into the register file. We denote this strategy as *processor-in-regfile* (PIR) due to its analogy with the *processor-in-memory* (PIM) approach [80]. Our PIR proposal is intended to exploit local computation in each bank as much as possible. Taking advantage of the available parallelism, an application is partitioned into smaller parallel problems, whose working sets fit in each bank. In this way, the pressure on the register file interface (read/write ports) is significantly reduced. The embedded logic, referred to as *local computation elements* (LCEs), is attached to each bank and provides SIMD support, as shown in Figure 4.3.

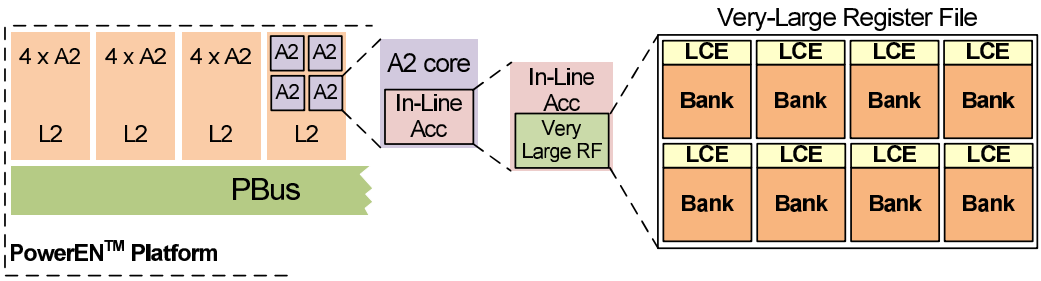


Figure 4.3: Organization of the 8-bank register file with 8 embedded LCEs.

Assuming a 64-KB register file with 8 LCEs and 4RD/1WR ports per bank, a scenario where all the LCEs can proceed in parallel is equivalent to a register file with 8 × 4RD/1WR ports and a SIMD width 8 times the LCE SIMD width.

In Figure 4.3 we assume that each LCE is attached (operates on) a particular bank in the register file. However, this could not always be the case. Part of the potential of the PIR strategy resides in the innumerable ways to organize the embedded LCEs. Depending on the targeted market, we could be interested in exploiting throughput by attaching one LCE to each bank, as shown in Figure 4.3. In some other scenarios, we could target power and/or area savings, by embedding fewer LCEs, each one sharing two or more banks. Moreover, those scenarios could coexist in the same design, in such a way that each configuration is employed depending on the particular needs of the

application being executed. The innumerable ways to organize the embedded LCEs in the VSRF makes our proposed PIR strategy a key enabler for highly-flexible workload-optimized designs for base stations and, in general, for throughout-aware computation. The flexibility and benefits provided by the PIR strategy will be next assessed based on a Turbo Decoding application for base stations.

4.5.1 Case Study: Turbo Decoding

A Turbo Code is a class of forward error-correction code that allows channel throughput levels very close to the channel capacity [10]. A Turbo Decoder, as the one considered in this section, incorporates two constituent decoders, interleaver, and de-interleaver in a feedback loop, with the decoders implementing the BCJR algorithm [7]. The input to the decoder is a bit stream (*codeword*) with two parity bits per each data bit (1/3 rate encoding).

Even though decoding a Turbo Code is a sequential process, to take advantage of the abundant parallelism available in our PIR architecture, the codeword is divided into as many sub-blocks as the number of banks in the VSRF (similar to [45]). In this way, each LCE decodes the sub-block stored in its associated bank, and all the LCEs can proceed in parallel. The decoding process performed by each LCE involves the computation of forward recursion probabilities (α values), backward recursion probabilities (β values), and extrinsic probabilities.

The organization adopted for Turbo Decoding consists of eight banks and four LCEs, in such a way that each LCE is shared by two neighboring banks, as shown in Figure 4.4. The benefits of this sharing are twofold: it saves area and enables the latency through the two-cycle LCE pipeline to be covered. Each LCE is fed by its even bank on even cycles and by its odd bank on odd cycles. Figure 4.4 also shows the internal LCE structure required for the forward recursion computation. Each LCE incorporates 1-byte multiplexers, saturated adders/subtractors and maximum selectors. This logic is organized in a two-stage two-cycle pipeline, with an intermediate buffer between stages. The same logic is used for backward recursion computation.

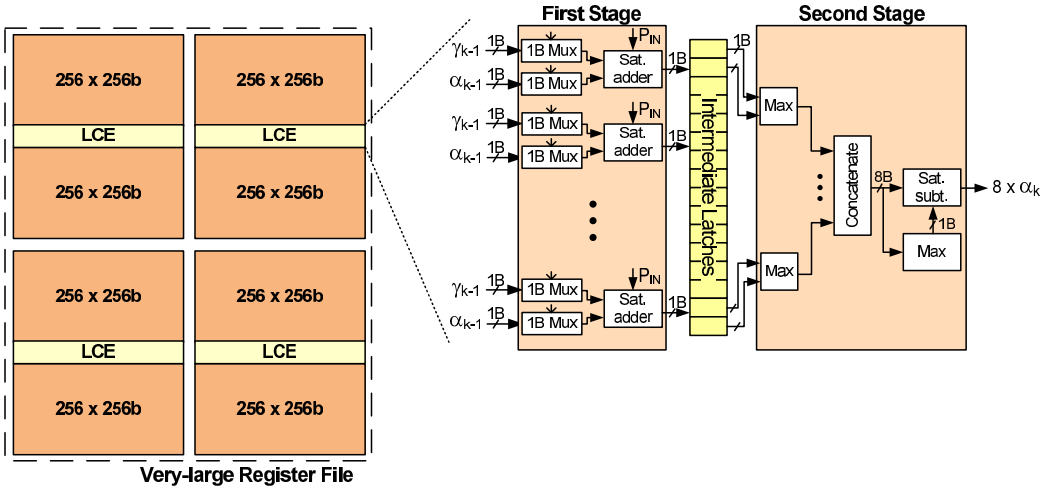


Figure 4.4: 8-bank 4-LCE register file organization for Turbo Decoding. Each LCE decodes the sub-blocks stored in its attached banks. The LCE is organized in a two-stage two-cycle pipeline, with an intermediate latch-based buffer between stages. The use of the two stages is multiplexed in time to hide the latency. Each "1B Mux" selects a 1-byte α or γ value from one of the 32-byte input registers. The output of the second stage is a set of eight 1-byte α values corresponding to time step k.

In our approach, the 6144-element codeword (the maximum length specified in LTE) is split into eight 768-element sub-blocks. Each sub-block is assigned to one particular bank. Making use of their attached LCEs, all the banks proceed concurrently to compute: first, the 768 time steps of forward recursion probabilities (α values); second, the 768 time steps of backward recursion probabilities (β values); and finally, the 768 extrinsic probabilities.

After the decoding phase, all the statistical information generated (extrinsic probabilities) is shuffled (interleaved), based on mapping information that is also stored in the VSRF. The interleaving, which relies on the *gathering* support provided by the VBA, implies data movement between banks and cannot be parallelized. Even in that case, the interleaving phase benefits from the low wire latency and flexible data placement provided by the VSRF. Once this data reordering is finished, a new decoding phase begins.

Figure 4.5 shows the performance estimation for the VBA with logic embedded into the VSRF ("VBA+PIR" scenario). The VSRF has 64 KB of

capacity, with 8 banks and 4RD/1WR ports. As it can be observed, the VBA+PIR scenario presents significant performance improvements: 72% reduction on execution time compared to the VMX baseline, and 43% reduction compared to the VBA case.

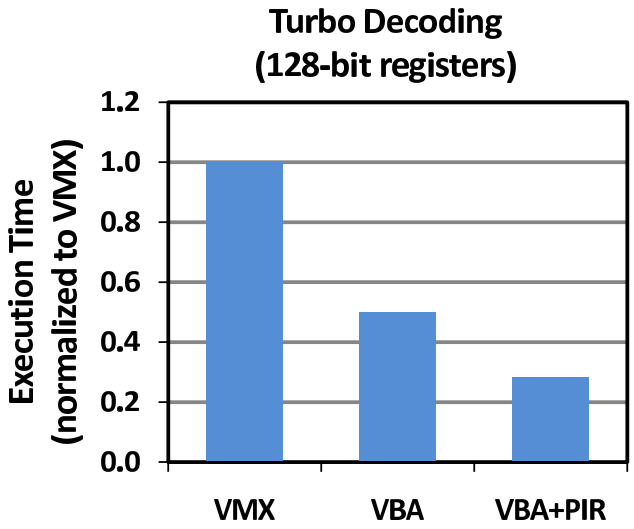


Figure 4.5: Normalized execution time to decode a 6144-bit codeword in six iterations. "VMX" is a scenario with a 32-entry vector register file, "VBA" incorporates a 2048-entry VSRF and in "VBA+PIR", four LCEs are embedded into the VSRF. In all the cases, 256-bit vector registers are considered.

For a maximum-length codeword (6144 bits), the throughput of our Turbo Decoding implementation is 230 Mbps for a single A2 core with VBA. Decoding in a base station is performed in the uplink, for which 3GPP specifies data rates of up to 75 Mbps (LTE) [1] and 500 Mbps (LTE-Advanced) [2]. We can satisfy the LTE requirements by using one A2 core with VBA. For LTE-Advanced, the requirements can be met with two to three A2 cores with VBA proceeding in parallel.

In comparison to the existing solutions, the 230 Mbps throughput estimation is similar or slightly lower than two state-of-the-art DSPs [38, 100], for the same clock frequency. However, as it was mentioned for the FFT case in Section 4.4.3, this performance is attained with a fully-programmable, scalable design for a much broader range of applications than DSPs.

4.6 Opportunities and Challenges

4.6.1 Reconfigurable Processor-in-Regfile Architecture

A PowerEN chip with in-line universal accelerators constitutes by itself a general-purpose, high-throughput platform that meets the high computation demand in base stations. In the Turbo Decoding example in Section 4.5.1 we show that such computation capability is further improved when small special-function computation elements are embedded into the register file. In the example, each LCE is highly-optimized to perform a particular function (decoding a codeword sub-block). Even better would be a PIR-based design with highly-optimized LCEs, while still targeting a broad spectrum of throughput-aware applications. Based on that motivation, in this section we briefly discuss *rPIR*, a PIR-based architecture built upon reconfigurable LCEs.

In the envisaged rPIR organization, the reconfigurable LCEs attached to the banks incorporate enough resources to accelerate different throughput-aware applications. Those resources are basic building blocks (multiplexers, adders, multipliers, etc.) which can be combined in different ways to perform different types of computation. We call this approach "reconfigurable" because such combination of the building blocks can be changed by modifying the interconnection between them.

To define the type and amount of building blocks to incorporate in the reconfigurable LCEs for a particular market, we analyze a representative suite of applications for such market. For illustrative purposes, we just consider FFT and Turbo Decoding for base stations in the following analysis. This evaluation determines the operations common to both applications: type and size of additions, type and size of multiplications, type of multiplexers, among others. Those building blocks are incorporated into each LCE, and interconnected through a reconfigurable crossbar-like network. To further illustrate it, we consider the radix-8 DIT FFT implementation from Section 4.4.3 and the Turbo Decoding implementation from Section 4.5.1.

The FFT implementation presented in Section 4.4.3 is based on Pease's

algorithm for parallel computation [93]. By leveraging the natural affinity between the radix-8 FFT and our 8-wide SIMD design, eight radix-8 butterflies can be executed in parallel in place (i.e., *locally* in each bank) throughout the FFT. Each LCE is an 8-wide SIMD unit, which takes two input 256-bit vector registers. Each vector register holds eight complex values. Therefore, to simultaneously add two vector registers, we include 16 2-byte adders, each one operating on real or imaginary parts (strictly speaking, we implement the 16 2-byte adders with 32 1-byte adders). In addition, we need 32 2-byte multiplexers to select real or imaginary parts from the two input registers. For the butterfly multiplications, we rely on the complex multiplier in the VBA due to complexity of integrating it in the LCE. Table 4.1 summarizes the required logic to compute the FFT butterfly additions locally in each LCE.

The LCE implementation for Turbo Decoding shown in Figure 4.4 comprises 16 1-byte multiplexers to select the input γ values, 16 1-byte multiplexers to select the input P_{IN} probabilities, 16 1-byte multiplexers to select the input α values, 8 1-byte multiplexers for the implementation of the saturated subtractor, 16 saturated byte adders for the first LCE stage, 8 saturated byte adders for the implementation of the saturated subtractor, 8 1-byte maximum selectors and one 8-byte maximum selector. Table 4.1 summarizes the required logic for Turbo Decoding.

The building blocks listed in Table 4.1 are incorporated into each LCE as "facilities" and interconnected through a reconfigurable crossbar-like network. The internal LCE organization for this particular scenario is shown in Figure 4.6. The interconnect network behaves as a full crossbar, and can take any element (byte, half-word, word or double-word) from any input or temporal buffer, and feed it to any input in any computation facility. This flexibility, the key benefit of hardware reconfigurability in our proposal, can be controlled with a set of bits (top "Select" signal in Figure 4.6). Configuring a particular functionality at run time requires that we specify the group of control bits to be set in each clock cycle. All the bits associated with a particular configuration are stored in a look-up table ("Config. LUT" box in Figure 4.6).

	Radix-8 FFT	Turbo Decoding
1-byte muxes	-	56
2-byte muxes	32	-
1-byte adders	32	24
1-byte max selectors	-	8
8-byte max selectors	-	1

Table 4.1: Required logic for radix-8 FFT and Turbo Decoding computation at LCE level.

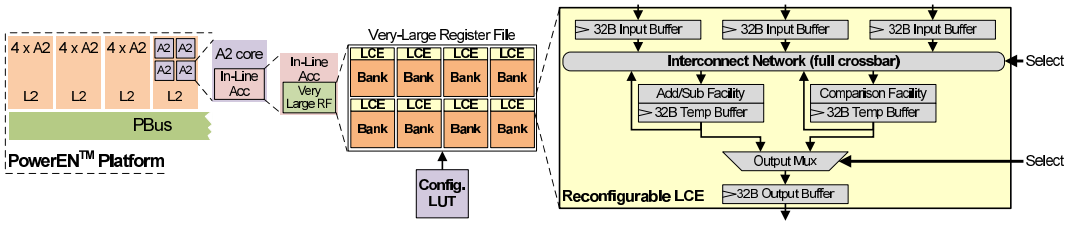


Figure 4.6: Internal organization of a reconfigurable LCE, in one particular envisaged design.

The rPIR approach presented in this section constitutes a trade-off between highly-optimized special-function designs and general-purpose processors. On one hand, rPIR allows special-function computation to accelerate parts of applications (e.g. butterfly additions in the FFT), but limited to the building blocks that can be included into the LCEs. On the other hand, the resources included into the LCEs can be re-used to target different applications, making an efficient use of the area.

4.6.2 Challenges

Many issues remain open regarding the VBA and the PIR/rPIR approaches. These challenges are:

- **Compilation support for VBA, PIR and rPIR** — The large register file proposed in this chapter poses an additional challenge to the programmer. It is not possible to address thousands of registers by encoding them with just five bits, as it happens with conventional Instruction Set Architectures. Instead, registers are accessed through an indirection mechanism based on register mappings [29]. Therefore, it

would be important to provide the programmer with compiler support to deal with the thousands of available data registers plus the register mappings. In addition, the PIR and rPIR proposals also require that the VBA ISA be extended with extra instructions to drive the embedded LCEs and to set different configurations in the rPIR scenario.

- **Use in multi-threaded environments** — Although the workloads evaluated in this chapter are single-threaded, we also envisage a design where each thread operates upon a particular set of banks and LCEs. For instance, assuming an 8-bank 8-LCE register file in a 2-way SMT scenario, each thread may be devoted to operate on half of the register file. This configuration could allow, for instance, two independent applications (e.g., FFT and Turbo Decoding) to proceed in parallel.

- **Other application domains** — In addition to the advantages for base stations, the PIR strategy may provide potential benefits for other types of applications. Just to mention a few examples, parallelizable sorting, search algorithms and sparse matrix computation could exploit the bank-based VSRF organization with embedded LCEs. The AA-Sort sorting algorithm [47] is an example of an application that one can port. In the original version, AA-Sort first divides the input data array into blocks that fit into the cache of the processor. Each block is then sorted independently, and all of them are finally merged to produce the ordered data array. In a VSRF scenario with embedded LCEs, one can apply a similar strategy. We first divide the input data array into blocks, and each block is assigned to one VSRF bank. All the LCEs proceed in parallel to sort the block in its attached bank. Finally, all the blocks are merged taking advantage of the low-latency gather operations provided by the VBA. We expect significant performance results with this strategy.

4.7 Summary and Concluding Remarks

Mobile networks and communication systems are currently moving from the third generation (3G) toward the fourth generation (4G) standardization. This evolution is facing new challenges regarding the huge amount of acceleration and throughput computation required by base stations. In this chapter, we expound on the potential benefits that the PowerEN processor could provide for wireless communication systems, particularly base stations. In such context, we present potential extensions to exploit the throughput computation capabilities of the PowerEN processor. These modifications involve replacing the bus-attached special-function accelerators with a layer of *in-line*, *universal*, throughput acceleration support, incorporated within the A2 cores. In order to highlight the benefits of such an approach, we present an evaluation of a radix-8 FFT algorithm, which is one of the dominant applications in the context of base stations.

We analyze the impact in terms of area and power consumption due to the adoption of in-line universal accelerators. As we show, significant savings in area and power consumption can be achieved by adapting the PowerEN architecture for pico and femtocells.

The in-line accelerator incorporates a bank-based very-large register file, with embedded SIMD support (*processor-in-regfile* or PIR). The PIR strategy is key to overcoming the limited number of register file ports, and constitutes a highly-parallel super-wide-SIMD device, ideal for throughput-aware computation. For a particular Turbo Decoding implementation, we show how the PIR strategy fits the computation requirements for LTE base stations. In this case, our proposal shows significant performance advantages over a traditional VMX-based design as well as a baseline VBA-based design.

Our design presents throughput improvements ranging from 1.5 to 3.5 times for the FFT computation, and similar or slightly lower for Turbo Decoding, compared to existing DSPs. This performance is attained with just one design (the VBA), which can be fully programmed to accelerate other types of applications. Instead, DSPs rely on hardware accelerators, which are special-purpose designs targeted to tackle particular application types.

Chapter 5

Power Management

The emergence of high-end multi-core chips in the early 2000s is not just a consequence of single-thread performance limitations. It is also a result of the unprecedented power consumption levels being faced by single-core processors in the late 1990s. In the beginning, power consumption concerns were mostly linked to thermal issues: power hungry chips demand more sophisticated (and, therefore, more expensive) cooling mechanisms. With time, energy bills also became part of those concerns, mostly associated to data centers power and energy consumption. For example, total data center power consumption accounted for between 1.7% and 2.2% of total electricity use in the U.S. in 2010 [62]. As a result, processor design and manufacturing since the early 2000s was not driven just by performance, but also constrained by strict power budgets. This phenomenon is usually referred to as the "power wall".

The rationale behind the power wall has its origins in 1974, when Robert Dennard, Fritz Gaensslen, Hwa-Nien Yu, V. Leo Rideout, Ernest Bassous and Andre LeBlanc, from the IBM T. J. Watson Research Center, postulated the scaling rules of metal-oxide-semiconductor field-effect transistors (MOSFETs) [27]. One key assumption of the Dennard's scaling rule is that operating voltage V and current I should scale proportionally to the linear dimensions of the transistor in order to keep power consumption ($V \times I$) proportional to the transistor area (A). But manufacturers were not able to

lower operating voltages sufficiently, and power density ($V \times I/A$) kept growing until it reached the power wall. As a result, frequency scaling slowed down and industry shifted to multicore designs to cope with single-thread performance limitations.

The power wall has fundamentally changed the way modern processors are conceived, particularly in the realm of mobile applications. Multiple simpler cores were incorporated into the same chip instead of just one fat, power-hungry core. The multi-core philosophy not only helps to overcome single-thread performance limitations, but it is also an important strategy to reduce power density and distribute temperature more uniformly within the chip. Processors became more aware of power consumption, with additional on-chip "intelligence" for power management. Two popular dynamic power reduction techniques are *clock gating* and *dynamic voltage and frequency scaling* (DVFS). Clock gating consists of disconnecting the clock network —which is responsible for a significant portion of chip power— from unused circuit blocks. DVFS allows to dynamically reduce supply voltage V_{DD} and/or frequency which can result in significant power reductions. Even if these mechanisms are effective in reducing dynamic power consumption, still static power accounts for a large fraction of today's chip power. *Power gating* is a circuit-level technique capable of eliminating both dynamic and static power by cutting off the power supply to a logic macro. The application of power gating, however, is not trivial. There is a considerable latency associated with switching a macro on and off. Therefore, it is crucial to identify (and even, generate) the right opportunities for the actuation of the power gating knob. In this chapter, we focus on strategies to leverage the software-hardware interaction to power gate chip components as much as possible with minimal performance impact.

The incorporation of multiple threads and cores into a single chip significantly affects the hardware-software interaction. The operating system (OS), compilers and, ultimately, the programmer are now aware of the existence of multiple processing elements. The allocation of *software threads* across available *hardware threads* is a software-level responsibility, with little or no hardware control (Figure 5.1). In most cases, programmers rely on

OS schedulers to allocate software threads across hardware threads. Sometimes, programmers can also explicitly establish preferred thread allocations by setting *affinities* between software and hardware threads [15].

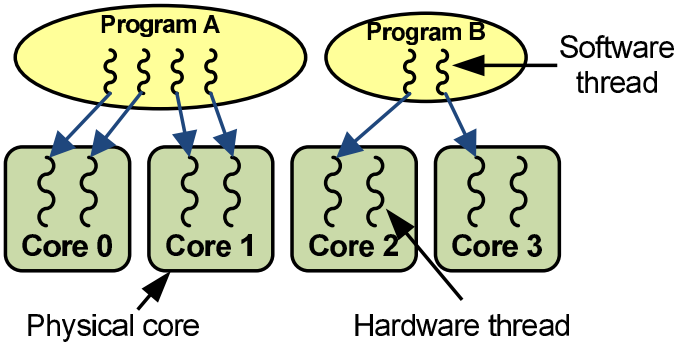


Figure 5.1: Allocation of six software threads (four belonging to program "A" and two to program "B") across eight hardware threads, in a 4-core 2-way SMT CMP. From a program standpoint, hardware threads are "seen" as eight logical cores. In most cases, the operating system maps software threads into hardware threads and cores.

The way software threads are assigned to available hardware threads has performance and power consumption implications. Under multi-threaded workloads, it is not always evident what is the best thread allocation policy to improve performance. It is neither evident how such allocation impacts the power consumption. To illustrate the thread allocation implications for performance and power consumption, we execute a multi-threaded application in a POWER7 processor with eight 4-way SMT cores in the same chip (i.e. 32 available hardware threads). The evaluated application is Dedup (from the PARSEC benchmark suite [12]), which is executed with eight software threads. The platform runs Linux OS with kernel version 3.0.1. Software threads are pinned to specific hardware threads by setting CPU affinities. In one scenario, each software thread is pinned to a different core (configuration "8x1": eight cores with one software thread each). In another scenario, two software threads are pinned per core (configuration "4x2": four cores with two software threads each). Configuration "8x1" mimics what the OS scheduler usually strives to do: to place different software threads in different cores to minimize inter-thread conflicts and maximize performance.

Figure 5.2 presents a comparison between these two scenarios ("8x1" and "4x2"), in terms of execution time, chip power consumption and accesses to remote caches in the same chip. In POWER7, each core can effectively access shared data located in remote L2 and L3 caches through the coherence fabric. While execution time is increased by just 5% when configuration "4x2" is adopted, chip power consumption is cut down by slightly more than 20%. The reason for such small performance degradation when the number of cores is halved relies on the significant inter-thread data sharing present in this application. When software threads are executed closer (sharing the same cache hierarchy), the number of accesses to remote cache regions in the chip decreases significantly.

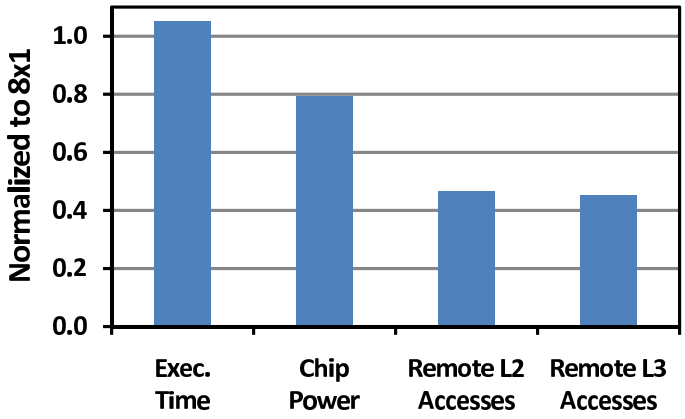


Figure 5.2: Dedup application with eight software threads executed in POWER7. In one configuration, each software thread is pinned to a different core ("8x1": eight cores with one software thread each), while in other configuration two software threads are pinned per core ("4x2": four cores with two software threads each). Figure presents execution time, chip power consumption and accesses to remote cores' caches. The results are normalized to the "8x1" configuration.

In the Dedup example presented, software threads are statically pinned to hardware threads. In other cases, multi-threaded applications may not benefit from statically pinning software threads to fewer cores. It strongly depends on data sharing degree and working set sizes. It is also determined by the number of software threads in an application: the larger the number of software threads, the more the performance is degraded when the number

of cores is reduced. Therefore, we cannot statically confine multi-threaded applications to fewer cores and always expect better power-performance efficiency. Instead, we want to dynamically find those *sweet spots* (particular software thread placements which maximize power-performance efficiency). Our goal is to build and evaluate a heuristic capable of finding the best power-performance execution points at runtime.

In this chapter we use the term *thread consolidation* (or TC) to refer to the method of using fewer cores at higher SMT levels. In the Dedup example presented before, software threads are consolidated from the "8x1" configuration (one software thread per core) to the "4x2" configuration (two software threads per core). The heuristic presented in this chapter is based on thread consolidation. Its objective is to detect (during an application's execution) when it is possible to consolidate threads with minimal (or zero) impact on performance. Similarly, if threads placed in the same core face high inter-thread conflicts (e.g. in the L1 cache), the heuristic unconsolidates them to mitigate the situation.

The key contributions of this chapter are:

- We present a *thread consolidation heuristic* (TCH) capable of maximizing power-performance efficiency at runtime for multi-threaded applications. The heuristic makes use of the TC technique to place software threads across hardware threads, with power-performance as its objective function. TCH is an extremely simple heuristic, which just requires access to three hardware event counters and on-line chip power consumption information. TCH does not require any kind of off-line pre-processing and performs very lightweight computation to make TC-related decisions at runtime. These characteristics make TCH very suitable for being implemented either at system software level or at chip level.

- We implement and evaluate TCH in a real POWER7-based system. Therefore, results presented in this work are not based on any simplifying assumptions and include all possible overheads associated with

thread consolidation and unconsolidation. TCH efficiency is also evaluated for different software thread counts.

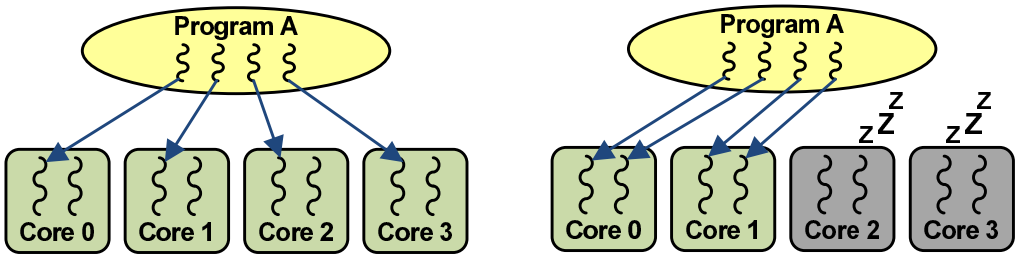
- We study an additional approach to benefit from TC: *per-core power gating* (PCPG). When PCPG is adopted, cores that remain unused after TC are powered off to further minimize power consumption. We show that the synergy between TC and the PCPG technique can provide significant power-performance efficiency improvements.

5.1 Thread Mapping and Thread Consolidation

In this work, the placement of software threads across hardware threads and cores is referred to as *thread mapping*. Assuming a multi-threaded application with N software threads executed on a CMP with C cores and S SMT threads per core ($T = C \times S$ hardware threads total), there are multiple possible mappings. In particular, in this work we assume symmetric placement of software threads across cores (i.e. software threads are evenly distributed across cores). In such scenario, multiple possible mappings exist if $N \leq T/2$. For example, the possible mappings when an 8-thread application ($N = 8$) is executed in a POWER7 processor ($C = 8$, $S = 4$ and $T = 32$) are the following: eight cores with one thread each ("8x1"), four cores with two threads each ("4x2") or two cores with four threads each ("2x4"). For an application with 16 software threads ($N = 16$), they can be placed in eight cores with two threads each ("8x2") or in four cores with four threads each ("4x4"). Figure 5.3(a) presents a scenario where a 4-thread application executes in an illustrative CMP with four 2-way SMT cores. Each software thread is assigned to a different core ("4x1" mapping).

Given a particular symmetric thread mapping, *thread consolidation* halves the number of assigned cores while doubling their SMT level. Thread consolidation results in a new thread mapping. For example, after applying thread consolidation to the mapping "4x1" in Figure 5.3(a), the new mapping "2x2" takes place, shown in Figure 5.3(b). In this work we benefit from

unused cores to reduce chip power consumption (e.g. by power gating them).



(a) "4x1" mapping (b) "2x2" mapping

Figure 5.3: Placement of four software threads belonging to a multi-threaded application across four 2-way SMT cores. In 5.3(a), each software thread is assigned to a different core ("4x1" mapping).In 5.3(b), two software threads are assigned per core after consolidation. Unused cores can be leveraged to either save power or boost throughput.

As mentioned before, in this work we assume symmetric placement of software threads across cores. When non-symmetric placements are considered, many other mappings are also possible. For example, the four software threads shown in Figure 5.3 can also be placed in two cores with one thread each plus one core with two threads ("2x1 + 1x2"). Throughout this chapter we focus just on symmetric mappings, due to the following reasons:

1. With symmetric mappings, thread consolidation is natural. For example, consolidation from "4x1" to "2x2" means that, for the same number of software threads, the number of cores is halved and the SMT level per core is doubled. With asymmetric mappings, on the other hand, thread consolidation becomes a blur.

2. Additionally, for the kind of multi-threaded applications considered in this work, software threads are usually assigned the same amount of work. With asymmetric mappings, cores with more software threads will take longer to complete (e.g. due to higher inter-thread cache contention), while cores with fewer threads will have to wait for them. In that scenario, the whole execution will slow down, which will provide no benefit for the less loaded cores.

5.2 Thread Consolidation Efficiency (TCE)

The objective of this work is to build and evaluate a heuristic capable of finding the most power-performance efficient thread mappings at runtime. Therefore, it is required to have a metric to quantify such efficiency. In this section we present the *thread consolidation efficiency* (TCE) metric, which provides insights about the "quality" of thread mappings, and constitutes the basis of the proposed heuristic. TCE is defined as follows:

$$TCE = \frac{Perf(mapping\_A)/Perf(mapping\_B)}{Power(mapping\_A)/Power(mapping\_B)}$$

where *Perf* and *Power* are the application execution performance and power consumption, respectively, for a particular thread mapping. The *TCE* metric quantifies the power-performance efficiency when the thread mapping of an application is changed from *mapping_B* to *mapping_A*. It is important to note that the TCE metric can be used either with consolidations (e.g., from $mapping\_B = 8{\times}1$ to $mapping\_A = 4{\times}2$) or with unconsolidations (e.g., from $mapping\_B = 4{\times}2$ to $mapping\_A = 8{\times}1$). If $TCE > 1$, the new mapping (*mapping_A*) provides larger power-performance efficiency than the previous one (*mapping_B*). If $TCE < 1$, the new mapping is less power-performance efficient than the previous one. If $TCE = 1$, both mappings perform equally in terms of power-performance efficiency. Ideally, we are interested in actions (consolidations or unconsolidations) with *TCE* values larger than one.

In this work, *Perf* is represented by throughput (i.e. the total number of instructions completed by all the threads per cycle), and *Power* is represented by chip power consumption. In this case, TCE becomes essentially *energy per instruction of mapping "A"* divided by *energy per instruction of mapping "B"*.

The use of the TCE metric can be illustrated with the Dedup application presented at the beginning of this chapter. As shown in Figure 5.2, performance is degraded by 5% and chip power is reduced by 21% when the

application is executed with mapping "4x2" compared to mapping "8x1". TCE for "4x2" compared to "8x1" is calculated as follows:

$$TCE = \frac{Perf(4\times2)/Perf(8\times1)}{Power(4\times2)/Power(8\times1)} = \frac{0.95}{0.79} = 1.20$$

This means that for the Dedup example presented at the beginning of this chapter, mapping "4x2" is 1.20× more power-performance efficient than mapping "8x1". In this example, performance is slightly hurt due to consolidation. In other cases, however, performance may be significantly degraded and, even so, TCE may be larger than one. For example, let's assume that consolidation reduces chip power consumption by 50% and, at the same time, degrades performance by 40%. This results in $TCE = 0.60/0.50 = 1.20$. Even if the new mapping is more power-performance efficient, we will not accept such a performance degradation. To cope with this aspect of the TCE metric, the heuristic presented in Section 5.4 is adjusted to favor performance over power reduction.

5.3 Thread Consolidation Opportunities and Value

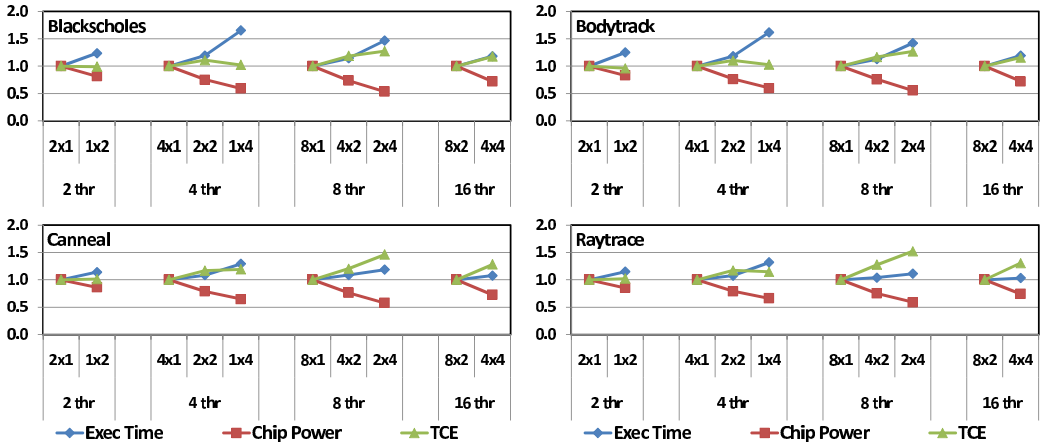
In this section, we assess the value of thread consolidation (TC) for power-performance efficiency, from a static perspective. *Static* means that a particular software-hardware thread mapping is set during the onset of the application's execution and kept until the end.

We present the results corresponding to four PARSEC applications out of twelve evaluated: Blackscholes, Bodytrack, Canneal and Raytrace. These applications are representative in terms of thread consolidation *friendliness*. In some cases, performance degradation due to consolidation is significantly smaller than power saving. We referred to them as *TC-friendly* applications. In contrast, *TC-unfriendly* applications are heavily affected in terms of performance when thread consolidation is applied. Each application is ex-

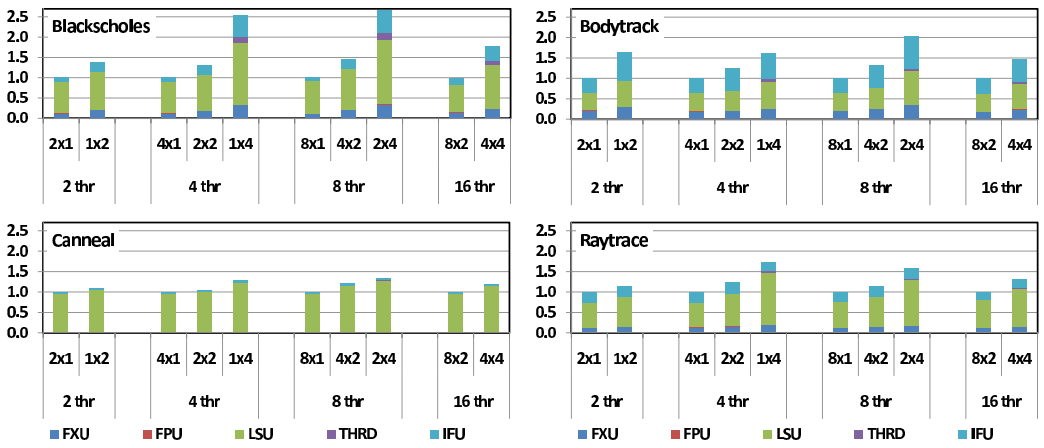
ecuted with different numbers of software threads: 2, 4, 8 and 16. Software threads are mapped to hardware threads considering all possible combinations between cores and SMT levels. For example, for two software threads, applications are executed on two 1-thread cores ("2x1") and on one 2-thread core ("1x2"). Figure 5.4(a) presents the execution time, chip power and thread consolidation efficiency metric (TCE) for Blackscholes, Bodytrack, Canneal and Raytrace. In all cases, the execution time increases when fewer cores at higher SMT levels are used (*thread consolidation*). This is mainly due to the competition for shared resources among threads in each core. For example, the more threads at core level, the higher the pressure on the cache hierarchy is. But at the same time, chip power decreases in all cases. Thread consolidation results in trade-offs between power reduction and how much performance we are ready to trade in exchange.

To better understand the performance-power trade-off, we consider the TCE metric presented in Section 5.2. When two mappings are compared, a TCE value larger than one means better power-performance efficiency. In the results presented in Figure 5.4(a), applications executed in more consolidated configurations show better power-performance efficiency, with a few rare exceptions (e.g. Bodytrack with two threads). This trend is observed for all PARSEC applications. In some cases, better power-performance efficiencies come at the expense of significant performance degradation. For example, Blackscholes with four threads and "1x4" mapping increases execution time by 65% compared to the "4x1" mapping. A similar performance degradation is observed for Bodytrack with four threads. In other cases, performance degradation for more consolidated configurations is considerably small. For example, Canneal with 16 threads and mapping "4x4" increases execution time by 8% compared to the "8x2" mapping, and cuts chip power consumption by 28%. Similarly, Raytrace with eight threads and mapping "4x2" increases execution time by just 4% compared to the "8x1" mapping, with 25% chip power reduction. In general, we observe that some applications are more friendly to TC (e.g. Canneal and Raytrace), while others benefit less from TC (e.g. Blackscholes and Bodytrack).

To better understand the reasons of performance degradation when threads



(a) Execution time, chip power and thread consolidation efficiency (TCE)



(b) CPI stall analysis

Figure 5.4: Static mapping analysis for four multi-threaded PARSEC applications on POWER7. Each application is executed with 2, 4, 8 and 16 software threads. For each software thread count, all possible mappings are considered. In each group, the results are normalized to the first configuration.

are consolidated, we perform a CPI stall analysis based on hardware events information. This stack is composed of the amount of cycles an instruction is stalled for completion due to: execution in the fixed-point unit ("FXU"), execution in the floating-point unit ("FPU"), load/store unit long-latency events ("LSU"), thread conflicts ("THRD") and instruction-fetch stalls ("IFU").

Figure 5.4(b) presents the stall CPI stack for the same applications and mappings discussed before. The two most important reasons for instruction stalling are long-latency events in the load/store unit (e.g. L1, L2 and L3 cache misses) and instruction-fetch stalls (e.g. branch mispredictions). Both are exacerbated when threads are consolidated because the larger the number of threads in the same core, the higher the inter-thread interference in the branch prediction unit and caches. Even though this interference may determine the success of TCH, it is important to note that in this section we are analyzing *static* thread mappings. Instead, TCH can set mappings *dynamically* in order to exploit *phases* of application behavior [53, 90] where inter-thread interference is minimal and, hence, friendly to TC.

5.3.1 Fully Populated Scenarios

In Section 5.1 we commented that, in order to have options for TC, the number of software threads N has to be less than or equal to $T/2$, where T is the total number of hardware threads. For example, T is equal to 32 in a POWER7 processor. If at any particular moment there are 32 software threads, just one mapping is possible ("8x4"). For this reason, in Section 5.3 we showed results for software thread counts less than or equal to 16. This may lead to the perception that TC does not have value in fully populated scenarios. However, when a multi-threaded application is analyzed from a dynamic point of view, we observe that the number of software threads is not constant throughout its execution. Figure 1.3 presents the software thread count frequency for twelve PARSEC multi-threaded applications executed with 32 software threads and *native* inputs on an IBM POWER7 processor. The figure shows that multi-threaded applications spend significant amounts of time with fewer threads than the specified thread count (just Swaptions and Vips execute with 32 threads most of the time). This brief analysis shows significant potential for TC, even when applications are launched with as many software threads as the number of available hardware threads. The heuristic proposed in this work can detect execution phases when $N \leq T/2$ and consequently apply TC to maximize power-performance efficiency.

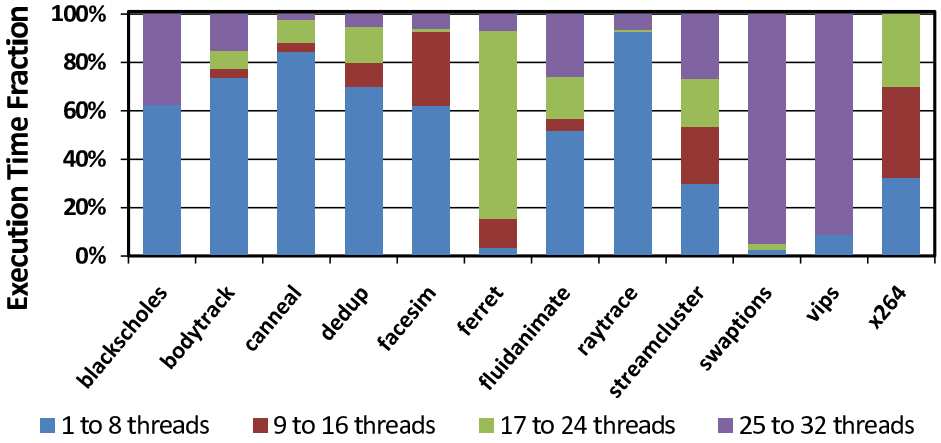


Figure 5.5: Software thread count histogram for twelve multi-threaded PARSEC applications. Applications are executed with 32 software threads to fully populate the underlying POWER7 processor. Even in that case, applications are not able to exploit all the available hardware threads.

5.4 Thread Consolidation Heuristic (TCH)

The ultimate goal of this work is to benefit from TC at runtime to minimize power consumption with minimal (or zero) impact on performance. In this section we present a *thread consolidation heuristic* (TCH) capable of dynamically maximizing power-performance efficiency for multi-threaded applications. During an application's execution, TCH strives to find the most efficient mappings by means of consolidations and unconsolidations. After any decision, TCH computes the TCE metric to evaluate the quality of the last action. In case of adverse TCE values, TCH may decide to undo the last action to recover the previous (more efficient) thread mapping.

TCH is an extremely simple closed-loop control algorithm (Figure 5.6). It is triggered every T milliseconds, but makes decisions if it is enabled (*step 1*). TCH is enabled only during stable computation phases, which is determined by sampling the application-level L1 miss count, as explained in Section 5.4.1.

Before making any consolidation/unconsolidation decision, TCH chooses the right *thread bucket* (*step 2*). Based on the current software thread count N, a thread bucket is defined as the minimum number of hardware threads

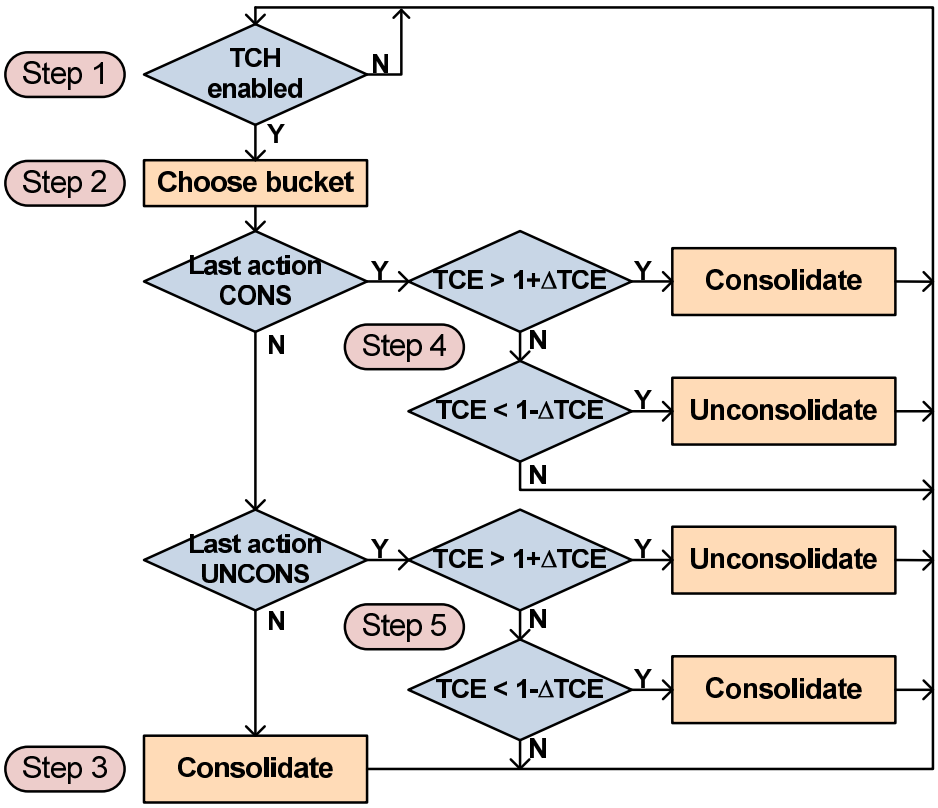


Figure 5.6: Thread consolidation heuristic (TCH).

(power of 2) required to support the current number of software threads. For example, if the current number of software threads is $N = 4$, the bucket to be used is 4, or if $N = 13$, the bucket is 16. By choosing the right bucket, TCH knows what are the possible thread mappings it can "play" with. All possible thread buckets in POWER7 are: 1, 2, 4, 8, 16 and 32 threads. Once the bucket is chosen, TCH sets the most unconsolidated mapping for that particular bucket to not harm application performance. Throughout the example, we will refer to this mapping as $mapping(i)$.

The first TCH action is consolidation (*step 3*). Just after a bucket selection, there is no power/performance history because the previous bucket may have a completely different power/performance footprint. Hence, TCH begins judging the effects of consolidation on power-performance efficiency. We refer to this new mapping as $mapping(i+1)$. To compare $mapping(i+1)$

versus $mapping(i)$, TCE is then computed and analyzed (*step 4*). If $TCE > 1 + \Delta TCE$, TCH assumes that the application is traversing a consolidation-friendly phase, and further consolidates threads (if possible). If, instead, $TCE < 1 - \Delta TCE$, TCH considers that $mapping(i + 1)$ is less power-performance efficient than $mapping(i)$, and goes back to $mapping(i)$. TCH continues in this way, analyzing the TCE of the last action, and making a new decision based on that (*steps 4* and *5*). Every time $TCE \geq 1 - \Delta TCE$ and $TCE \leq 1 + \Delta TCE$, TCH makes no decision.

5.4.1 TCH Adjustment Knobs

TCH efficiency can be adjusted based on four configuration parameters:

- **Monitoring interval (T)**: TCH is triggered (and power/performance values are read) every T milliseconds.

- **History length (H)**: TCH considers the last H power/performance samples to represent performance and power consumption for the current thread mapping. If $H = 1$, TCH takes just the most recent sample to represent performance and power consumption of the current mapping. If $H > 1$, the average of the last H samples is used to represent the current mapping. The smaller the H value, the faster TCH can make decisions. However, too small history values can lead to wrong decisions due to lack of information to characterize the current mapping. On the other hand, large H values can help to smooth application behavior, reducing the risk of wrong decisions due to power and/or performance outliers.

- **L1 miss count threshold (K)**: TCH determines if it is enabled by computing the average and standard deviation of the last H L1 miss count samples: $L1miss_{avg}$ and $L1miss_{stdev}$, respectively. If $L1miss_{stdev} \leq K \times L1miss_{avg}$, TCH assumes that the application is traversing a stable computation phase and, therefore, it is safe to make decisions. It prevents TCH from making decisions during phase transitions, which are wrong in most cases. Too small K values will keep TCH silent most

Parameter	Value	Range Explored
T	1000ms	Smaller values are not explored due to infrastructure limitations to collect power readings at finer granularities.
H	3	1, 2, 3, 4, 5
K	0.4	0.0, 0.1, 0.2, 0.4, 0.6, 0.8
ΔTCE	0.2	0.0, 0.1, 0.2, 0.4, 0.6, 0.8

Table 5.1: Selected TCH configuration parameters and ranges explored.

of the time. On the other hand, too large K values may lead to TCH decisions even during phase changes.

- **TC sensitivity (ΔTCE)**: TCH computes the TCE of the last action to determine if it has to be emphasized or undo. We prevent TCH from performing an excessive number of re-mappings (which in some cases may provide negligible benefits) by using a ΔTCE value whenever TCE is evaluated. The last action is considered beneficial if $TCE > 1+ \Delta TCE$ or detrimental if $TCE < 1 - \Delta TCE$. Too small ΔTCE values will result in too many re-mappings, while too large ΔTCE values will make it harder to find a more efficient mapping or undo a bad one. The current TCH implementation is adjusted to favor performance over power reduction. This means that the ΔTCE parameter is employed just during consolidation actions. The goal of this asymmetry is to create more "resistance" toward consolidation to minimize performance degradation. In other words, TCH will decide to consolidate if the power-performance benefit is really significant.

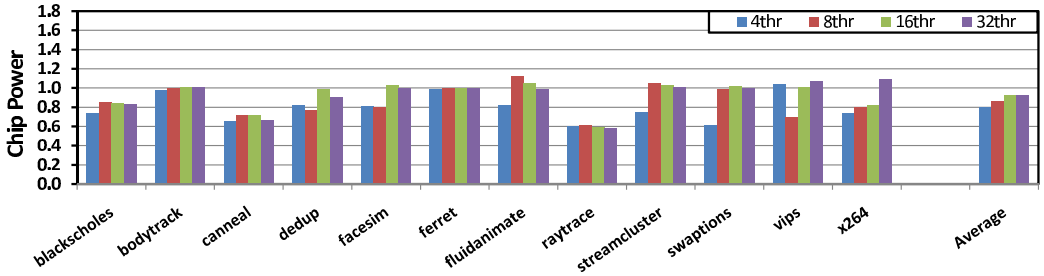
We explore all possible combinations of the TCH adjustment parameters to determine the set which maximizes the power-performance efficiency averaged across all the applications. The set of parameters used for the experiments of Section 5.5 are listed in Table 5.1.

5.5 TCH Evaluation

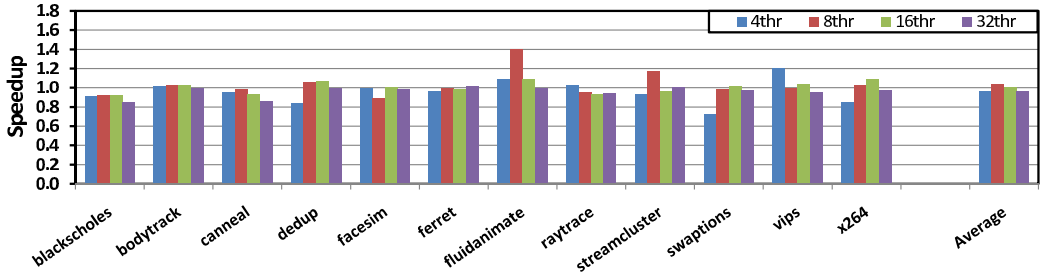
This section presents the results of the proposed TCH. It is evaluated in the context of the IBM BladeCenter PS701 system which is described in Section 2.3. All PARSEC applications are executed with 4, 8, 16 and 32 software threads. The baseline configuration consists of the default Completely Fair Scheduler (CFS) incorporated into the Linux kernel since version 2.6.23 [71]. CFS attempts to maximize CPU utilization by load-balancing the threads across available cores.

Figure 5.7(a) presents chip power consumption of PARSEC applications when threads are placed by TCH compared to the case when applications are scheduled by the Linux kernel. As it is observed, TCH significantly cuts down chip power consumption by exploiting TC. The reductions averaged across all the applications are: 21% (4 threads), 14% (8 threads), 8% (16 threads) and 7% (32 threads). In some cases, savings are very significant (e.g. up to 42% in the case of Raytrace). As expected, the larger the number of software threads, the smaller the reduction on chip power consumption, due to fewer TC opportunities when more threads are running. Even under fully populated scenarios (i.e. 32 threads), TC shows a non-negligible 7% power saving, due to the arguments discussed in Section 5.3.1.

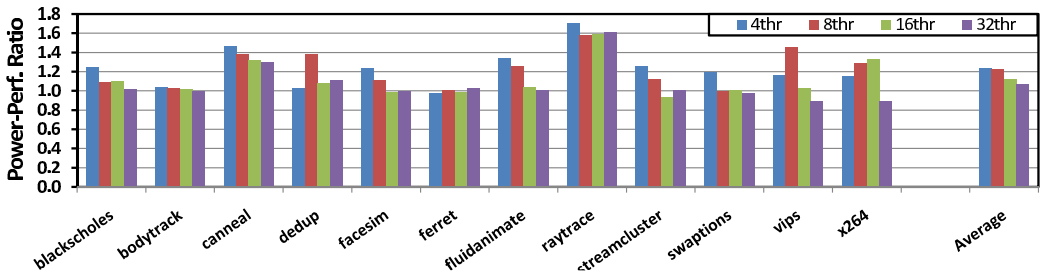
The "trick" behind chip power reduction lies in the fact that TCH tries to consolidate threads into fewer cores as long as it implies low performance impact. As shown in Section 5.3 for static thread placements, TC usually affects performance negatively. On top of that, TC also has its own overhead, because it implies software thread migrations across cores. Due to those reasons, we may expect performance degradations when TCH is applied. Figure 5.7(b) presents the speedup when applications are executed with TCH in comparison to the default Linux scheduler. In most applications, performance degradation is below 8%. Swaptions with 4 threads is one exception, with a 27% performance degradation. But more surprising is the fact that TCH can also improve performance compared to the Linux scheduler, significantly in some cases: 41% boost for Fluidanimate with 8 threads, 20% for Vips with 4 threads, and 17% for Streamcluster with 8 threads. Av-



(a) Chip power consumption



(b) Speedup in execution time



(c) Power-performance ratio: $\frac{speedup}{chip\ power\ reduction}$

Figure 5.7: TCH vs. the default Linux scheduler in an IBM BladeCenter PS701 system, in terms of chip power consumption (Figure 5.7(a)) and performance (Figure 5.7(b)). Figure 5.7(c) presents the power-performance ratio: values larger than one mean better power-performance efficiency. The results correspond to all PARSEC applications executed with 4, 8, 16 and 32 software threads.

eraged across all the applications, TCH performance is degraded by just 4% for the 4- and 32-thread scenarios, it is not degraded for 16 threads, and it is improved by 4% for 8 threads.

The benefits of TCH are better expressed when both chip power consumption and performance are taken into account. Figure 5.7(c) shows the

power-performance ratio, computed as speedup divided by chip power reduction. A ratio larger than one means a better power-performance efficiency, either because (1) power reduction is larger than performance degradation or (2) power increase is smaller than performance boost. Swaptions with 4 threads is an example of the former, while Fluidanimate with 8 threads is an example of the latter. As shown in the figure, power-performance ratios are larger than one in most cases, being up to 70% for Raytrace with 4 threads. Averaged across all the applications, TCH improves power-performance efficiency by 23% for 4 threads, 22% for 8 threads, 12% for 16 threads and 7% for 32 threads.

5.5.1 Analysis of Scenarios

This section analyzes particular scenarios from the set of results presented in previous section. In advance, we may expect TCH to lower power consumption at the expense of performance degradation. However, Figure 5.7 shows some exceptions:

- As discussed in Section 5.5, the Completely Fair Scheduler (CFS) tries to assign computationally intensive threads to different cores. Even if the number of software threads is smaller than or equal to the number of physical cores, it does not necessarily mean that each thread in a PARSEC application will run in a different core. For example, OS processes or other applications can be running on cores which, from the scheduler perspective, are *in use*. Scheduling decisions in such scenario may result in multiple threads from the PARSEC application being placed in the same core. But even if no other OS processes or applications are running in the processor, multiple threads can be placed together. For example, in the case of Fluidanimate with eight threads, we observe up to nine threads running together during some short periods: the main application plus eight *children* [1]. From the

[1] Child threads are created by the main application to parallelize the computation. The main application thread is suspended most of the time while children are in execution. Therefore, we should avoid assigning a dedicated core to the main thread.

scheduler standpoint, there are nine threads to schedule, with high chance of having two children assigned to the same core in the 8-core POWER7 processor. This explains the higher TCH performance for Fluidanimate with eight threads: CFS may assign multiple children in the same core whereas TCH will begin placing them one per core. With just 4 threads, both CFS and TCH place one thread per core. And with 16 and 32 threads, CFS and TCH necessarily put multiple children per core.

- Some other scenarios show important chip power reductions just for a particular number of threads. This is the case of Streamcluster with 4 threads, Swaptions with 4 threads and Vips with 8 threads. In these cases, TCH was able to find a highly efficient thread placement at the very beginning of the execution and adhered to it until the end. For the other thread counts, TCH had to perform some exploration at the beginning (in terms of consolidations and unconsolidations) to find an efficient thread mapping. The effectiveness of quickly finding the right mappings depends in part on the system load in the moment TCH is executed. Therefore, this kind of results are not rare.

- X264 with 32 threads is also worth commenting on because it does not present any chip power benefit. In this case, the number of software threads is larger than 16 throughout the whole execution, preventing TCH to apply TC.

To illustrate TCH "in action", Figure 5.8 shows a snapshot of 40 seconds of execution corresponding to X264 with eight threads. Top figure presents the number of active cores when X264 is executed under both TCH and Linux scheduler supervision. TCH begins assigning one software thread per core. Around second 5 (labeled as "1" in the figure), TCH decides to consolidate the eight threads into four cores ("4x2"). This action results in $TCE = 1.60$, reason for which TCH decides to further consolidate threads into just two cores (label "2"). This also results in a more efficient thread placement ($TCE = 1.26$). However, for the current thread bucket (8 threads), the

"2x4" mapping is the most consolidated possible placement. The reason TCH decides to fully consolidate threads is because X264 is a very communication intensive application [13]. Therefore, consolidation benefit is twofold: it helps performance by keeping threads as close as possible and it reduces chip power consumption because just a few cores are needed. Around second 24, performance starts dropping off, due to the nature of the X264 application. TCH detects this situation and immediately decides to unconsolidate (label "3")[2]. Even if this action stops performance degradation, its benefit is not significant enough to further unconsolidate. Around second 36, performance starts increasing more than chip power consumption, given rise to a large TCE. This means that the application is traversing an unconsolidation friendly phase, leading TCH to further unconsolidate threads (label "4"). In contrast, the default Linux scheduler keeps using most of the cores all the time. Not only the scheduler placement is not beneficial for performance, but it also demands more chip power by keeping the eight cores active most of the time. As it is shown in this simple example, TCH can find more efficient thread placements at runtime just relying on the TCE metric, without affecting performance.

5.6 Per-Core Power Gating

One of the most significant components of power consumption on today's complementary metal-oxide-semiconductor (CMOS) chips is *leakage power* (also referred to as *static power*). This is the power that a transistor dissipates when it is powered-on, even if it is in a stable logic state. As a consequence of Dennard's scaling rule, chip supply voltage has scaled down every processor generation. Along with supply voltage, threshold voltage has also been pushed down. Leakage current (and hence power) grows exponentially as the threshold voltage reduces. As a result, leakage power in today's chips accounts for a large fraction of total chip power. One effec-

[2]The delay between the moment performance starts dropping off (second 24) and TCH unconsolidation decision (second 27) is due to the $H = 3$ samples averaged to represent the current state.

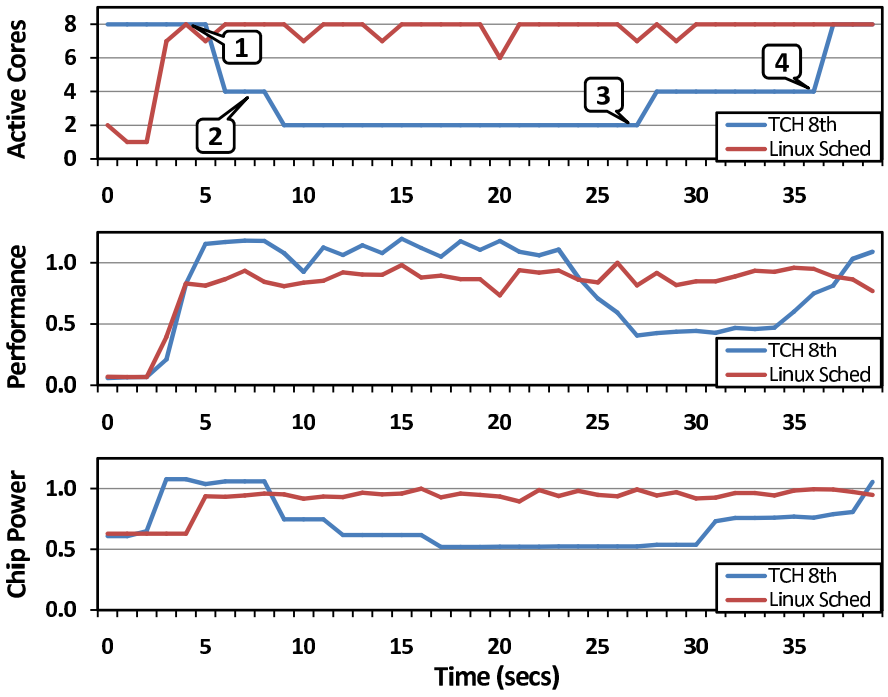
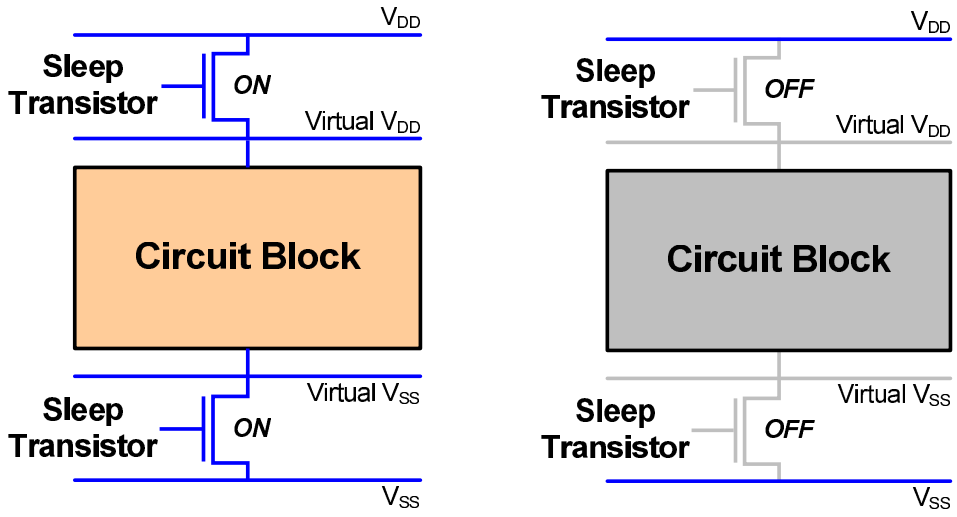


Figure 5.8: Number of active cores (top), performance (middle) and chip power consumption (bottom) when X264 is executed under both TCH and Linux scheduler supervision. Performance and chip power curves are normalized to the maximum value in the Linux scheduler case.

tive approach to cope with the increase of leakage power is *power gating*, a circuit-level technique that allows to cut off the power supply to a circuit block. Power gating is intended to completely turn off an idle circuit block, in order to virtually eliminate its leakage power. As it is shown in Figure 5.9, it is implemented with the help of a sleep transistor ("switch") that is inserted as a series header or footer device in the V_{DD}-to-Ground circuit path that includes the targeted circuit block [46, 78].

The power gating technique can be extensively adopted across the chip. At coarse granularity, large logic macros (cores, caches, on-chip controllers, etc.) can be completely power gated. At fine granularity, even functional units can benefit from this technique. Power gating, however, does not come for free. Its implementation requires large transistors for the header and footer switches. It implies an impact on the area consumed by the sleep



(a) Active mode (sleep transistor on)

(b) Idle mode (sleep transistor off)

Figure 5.9: Power gating operation. In Figure 5.9(a), header and footer sleep transistors are *on* and, therefore, the circuit block is active. In Figure 5.9(b), sleep transistors are *off* and the circuit block is power gated.

transistors as well as the latency to drive them. As a consequence, the applicability of the power gating technique is not trivial and requires the identification of long enough idle periods to hide the sleep transistors latency.

In particular, per-core power gating (PCPG) [65] is becoming an increasingly common knob in today's microprocessors [16, 18, 48]. Nevertheless, how to make the most use of PCPG is still an open question. For example, actuating the PCPG knob every time a core becomes idle may lead to negative power/performance benefits if the core idleness period is not long enough. Evidently, processes and software threads have to be scheduled accordingly across cores to generate opportunities beneficial for PCPG, with minimal impact on performance. TCH's ultimate goal is consolidation of software threads in as few cores as possible with minimal (or not at all) performance degradation. It is capable of detecting execution periods during which applications can execute in fewer cores with negligible performance degradation. For all these reasons, we believe that TCH in conjunction with PCPG can result in important power savings. In this section, we propose to adopt PCPG in order to switch off the cores that are left unused after thread

consolidation. The goal is to reduce or even eliminate the power consumption which takes place in the unused cores that is not related to the workload under evaluation (*workload-independent power*).

The particular IBM BladeCenter PS701 system utilized for the experiments does not support PCPG. To assess PCPG benefits in the context of TCH, we build a simple empirical model to estimate the workload independent power consumption of an idle core chiplet in POWER7 ($P_{chiplet\_idle}$). This is the power consumed by a core chiplet (core, L2 and L3 caches) when it is idling with the OS polling for work. Workload independent power consumption can be almost completely eliminated when the core chiplet is power gated. To estimate $P_{chiplet\_idle}$, we measure the POWER7 chip power consumption for different numbers of on-line idle cores as it is shown in Figure 5.10. The rest of the cores in each configuration are switched to *nap* mode and cannot be used by the OS. A linear regression across all measured points reveals that when a core chiplet enters nap mode, its power is reduced ΔP_{nap}. Based on information about which chiplet components are quiesced during nap [35], the chiplet power consumption breakdown [106] and ΔP_{nap}, we can then estimate $P_{chiplet\_idle}$.

There is an overhead associated with the power gating technique. In terms of latency, it includes the cost of driving the sleep transistor at circuit level as well as the activities incurred by the OS to prepare the core either for powering it down or up. Because the particular IBM BladeCenter PS701 system used in this work does not support PCPG, we estimate the power gating overhead based on the projections presented in [35]. The projected power gating overhead adopted in our experiments is 20ms.

We then feed the per-core-chiplet workload independent power estimation ($P_{chiplet\_idle}$) and power gating overhead to TCH and re-evaluate it for the same configurations considered in Section 5.5 (4, 8, 16 and 32 software threads for all PARSEC applications). Figure 5.11 presents the power-performance ratios for both TCH and TCH+PCPG. TCH results are the same shown in Section 5.5 and are included here for the sake of comparison. Baseline is the default Linux scheduler. As expected, the synergy between TC and the PCPG technique provides significant improvements in

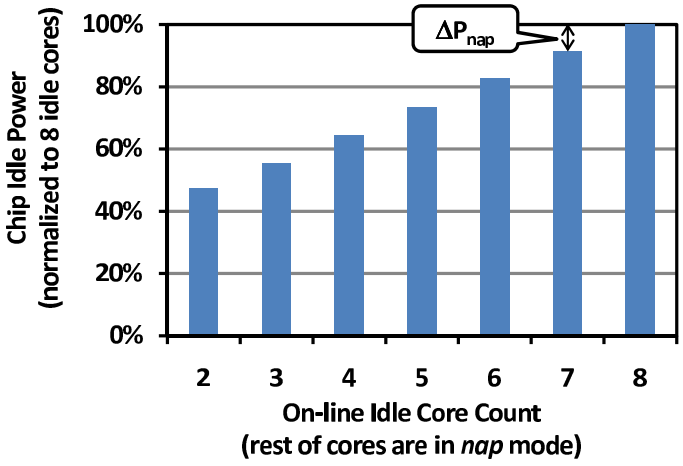


Figure 5.10: POWER7 chip power consumption for different numbers of on-line idle cores, normalized to the eight on-line cores case. Off-line cores remain in *nap* idle state, which deactivates instruction fetch and execution and turns off all clocks to the execution engines in the core, but it still keeps L2 and L3 caches coherent [35].

power-performance efficiency. Averaged across all the applications, power-performance efficiency is improved by 2.1×, 1.6×, 1.4× and 1.3× for 4, 8, 16 and 32 threads, respectively. These improvement factors are significantly larger than the already important benefits of solo TCH, being steeply large in some cases (e.g. Canneal or Raytrace). It is due to the nature of such applications, which spend significant amounts of time with very few active software threads. In those cases, TCH is able to aggressively reduce power consumption by applying PCPG. It is also worth mentioning that the fewer the number of software threads, the larger the power-performance efficiency benefits. With fewer threads in execution, TCH has more opportunities to power gate a larger number of cores. However, even when PARSEC applications are executed with 32 threads, the power-performance efficiency is 1.3× larger on average than the baseline.

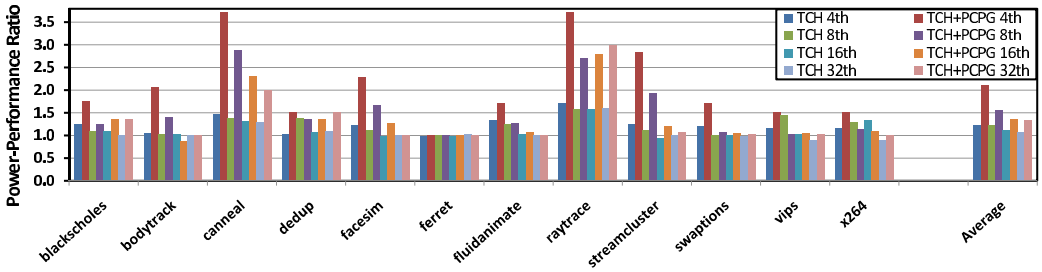


Figure 5.11: TCH+PCPG vs. the default Linux scheduler in an IBM BladeCenter PS701 system, in terms of power-performance efficiency (values larger than one mean that chip power reduction is larger than performance degradation). The results correspond to all PARSEC applications executed with 4, 8, 16 and 32 software threads. Per-core-chiplet power reduction and PCPG latency are based on estimations.

5.7 Summary and Concluding Remarks

In this chapter we investigate the impact of thread placement on power-performance efficiency of SMT-enabled CMP architectures. The first observation that arises from the execution of multi-threaded applications in a real SMT-enabled CMP (POWER7) is that the software-hardware thread mapping does affect performance and power consumption, resulting in different trade-offs. Against conventional wisdom, placing software threads across cores as much separated as possible is not always the best approach. Due to particular characteristics of an application, there are execution periods during which threads need to be closer (located in fewer cores) to favor inter-thread data sharing. In addition to the performance benefits when threads are consolidated, it is also possible to save power consumption on unused cores. As a result, there are great opportunities for power-performance efficiency improvement during an application's execution.

In this work we present a *thread consolidation heuristic* (TCH) capable of maximizing power-performance efficiency at runtime for multi-threaded applications. The heuristic makes use of thread consolidation and unconsolidation to place software threads across hardware threads, with power-performance as its objective function. TCH is an extremely simple heuristic, which relies on a few hardware event counters. TCH does not require any

kind of off-line pre-processing and performs very lightweight computation to make thread placement decisions at runtime.

We implemented and evaluated TCH in a real POWER7-based system. Results show chip power reductions of up to 21% (averaged across applications) compared to the default Linux thread scheduling policy, with performance degradations below 8% in most cases. In the presence of power-gating, TCH can improve power-performance efficiency by a factor of up to 2.1 with respect to the OS scheduler. Our work shows that intelligent thread placement exposes the potential for significant boost in power-efficiency in SMT-enabled CMP architectures. This becomes more evident as the number of cores and threads keep growing, which makes the software-hardware cooperation essential for power benefits.

Chapter 6

State of the Art

This chapter provides an overview of the prior art and current technologies related to power-performance efficient, throughput-aware multi-core chips. We focus on works from academia as well as products from industry, and we contrast them with our approaches to emphasize the novelty of the ideas presented in this dissertation. This chapter is organized following the thesis structure: we focus, first, on bandwidth-optimized last-level cache designs; second, on register file organizations for throughput-aware computation; and finally, on chip-level power management techniques and optimizations.

6.1 Bandwidth-Optimized Last-Level Caches

In the context of multi-core processors for throughput-aware computation, the existence of many (tens or hundreds) physical threads and cores puts a dramatic pressure on memory bandwidth. *Scalability* is the fundamental problem behind memory bandwidth: it grows at a much slower pace than the number of cores in the chip, limited by chip pin count and power consumption. Consequently, it is not surprising that a significant part of academia and industry research in the last decade was aimed to tackle that problem, known as the "bandwidth wall".

6.1.1 Commercial Products

Even with just two single-threaded cores, the IBM POWER4 processor [97] released in 2001 already incorporated a memory system highly optimized for bandwidth, with a 100 GB/s shared L2 cache and a 10 GB/s memory interface [30]. The evolution of IBM's POWER processor family has given rise to POWER7, a 4-way SMT, 8-core design released in 2010 [59, 104]. To feed its 32 physical threads, POWER7's memory interface is composed of two four-channel DDR3 controllers to deliver up to 100 GB/s [92]. A key innovation in POWER7 is the incorporation of a large *on-chip* 32-MB L3 cache (LLC). It is composed of eight 4-MB "regions", each one tightly coupled to a core to provide local fast access, as it is shown in Figure 6.1. Accesses that miss the L2 cache go to the 4-MB local L3 region. If the local L3 region is also missed, the other seven L3 regions are accessed through a coherence fabric. In addition, a cache line can be casted out from one L3 region to another. In other words, the eight regions constitute an effectively shared LLC in POWER7. Cache lines can also be cloned between L3 regions when multiple cores are actively sharing them in order to reduce access latency.

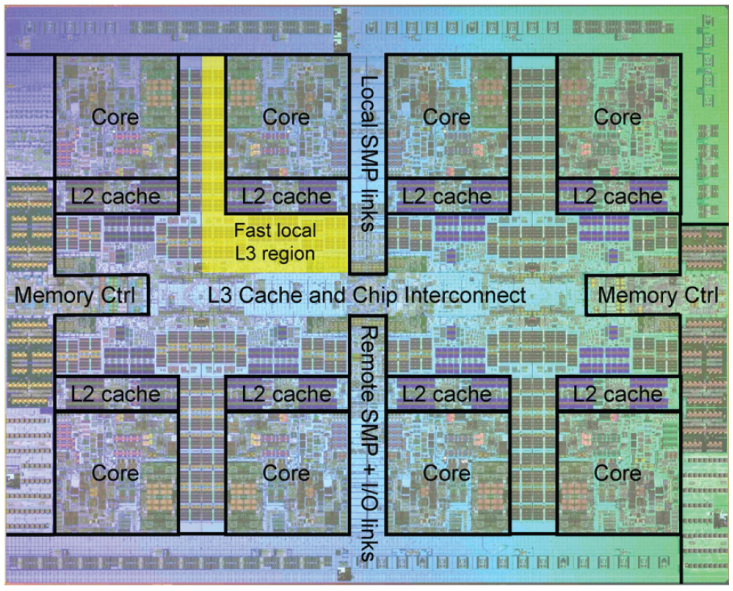


Figure 6.1: IBM POWER7 die photo (source: [92]).

The Intel Xeon processor also employs a shared large L3 cache to feed its eight 2-way SMT cores [87]. Similar to POWER7, Xeon's LLC is also partitioned in eight regions ("slices", in Intel's jargon). It is not clear, though, if some degree of data replication is allowed in the LLC as it happens in POWER7. In contrast to POWER7, the memory address space is spread across LLC slices based on a hash function, which gives Xeon cores a holistic view of the entire LLC [51].

Similar to POWER7 and Xeon LLC organizations, the design presented in this dissertation is also composed of "regions" or "slices" (referred to as "blocks"). In contrast to POWER7 and Xeon, our LLC is highly optimized for bandwidth instead of latency. For example, our design avoids data replication between blocks to maximize the effective cache capacity, even at the expense of higher access time. The double-buffering mechanism is leveraged to hide the additional latency to access remote blocks. Another key difference is coherence. In our case, the memory system is simpler, composed of *non-coherent* core-level local memories and just one cache level (the LLC). Similar to Xeon LLC, the address space is also spread across blocks in our organization. However, we interleave addresses instead of using a hash function[1]. Table 6.1 presents a summary of the most important differences between the LLC presented in this thesis and the ones found in POWER7 and Xeon commercial products.

Parameter	**IBM POWER7**	**Intel Xeon**	**Proposed LLC**
Size (MB)	32	24	8, 16, 32 and 64
Blocks	8 (regions)	8 (slices)	2, 4, 8 and 16
Optimize	Latency	Latency	Bandwidth
Coherent	Yes	Yes	No
Shared	Yes (through coherence fabric)	Yes (hash function)	Yes (address interleaving)

Table 6.1: Comparison between POWER7 LLC, Xeon LLC and the organization presented in this thesis.

[1]The address interleaving scheme adopted in our design can also be considered as a special case of a hash function.

6.1.2 Research Projects

There are several research projects which approach the bandwidth wall problem. Rogers *et al.* [86] present an analytical model to study memory bandwidth as a bottleneck for performance scalability in CMPs. Starting from a baseline 8-core CMP, performance scalability is analyzed for the next four chip generations, according to Moore's Law. To overcome bandwidth limitations, authors consider different memory traffic reduction techniques: cache compression, DRAM and 3D-stacked caches, link compression and sectored caches, among others. While authors adopt an analytical model, we perform a much more detailed analysis of the LLC based on cycle-accurate simulations. For a given LLC area and size, we emphasize the ways to make an optimum use of them (for instance, by interleaving the address space in a fine-grained manner across LLC blocks). The conclusions we obtain are usually not explicit for analytical models.

Liu *et al.* [66] present an analytical model to study memory bandwidth partitioning and its interaction with LLC partitioning in CMPs. Bandwidth partitioning is implemented using a *token bucket* algorithm. Each thread sends off-chip requests to a bucket. A *token generator* distributes tokens between buckets with rates proportional to the fractions allocated to different threads. An off-chip request can leave the bucket as far as there exists a corresponding token. While Liu *et al.* model general-purpose CMPs, we consider throughput-aware scenarios. In their work, threads are implemented as independent applications: there is neither data sharing among them nor coherence traffic. In contrast, we consider that data sharing is a more realistic picture in current chip multiprocessors as well as cache coherence traffic. Furthermore, while Liu *et al.* model a small 4-core CMP, we consider larger scenarios with up to 128 cores.

Hardavellas *et al.* [44] propose *Reactive NUCA*, a mechanism for LLC data placement in CMPs, targeting both latency and capacity. The design relies on the classification of different cache access patterns in server and multi-programmed applications: shared data is placed at fixed address-interleaved LLC blocks, private data is kept in the LLC block closest to the requester, and

data with a certain sharing degree is replicated across groups of slices. In their work, Hardavellas *et al.* consider server and multi-programmed applications, while we concentrate on throughput-aware workloads that can benefit from streaming memory systems to tolerate high latencies. In that sense, we put our attention in the memory bandwidth, because this is the bottleneck that limits the performance of current throughput-aware CMPs. Because latency is not a problem in our case, our approach is indeed simpler and it does not depend neither on the operating system nor on the applications characteristics to achieve significant bandwidth improvements and optimal capacity use. Furthermore, we also consider larger systems with tens or hundreds of processors.

Zhao *et al.* [105] present a hybrid LLC for CMPs where part of the cache is shared among all cores and, hence, optimized for capacity. Each core has also a private portion optimized for latency, along with a directory cache to locate data lines in remote shared portions. Each core begins looking for data in its private portion. In case of a hit, data is retrieved with low latency. In case of a miss, the search continues in the directory first and then in the shared part. Zhao *et al.*'s design is optimized for both latency and capacity, with the expense of additional area for directories. In our work, however, we tackle bandwidth and capacity instead of latency, without any additional cost.

Kelm *et al.* [60] propose the Rigel accelerator architecture, which can support over a thousand cores. Rigel groups processors in clusters, and cores within a cluster share a common cache. Clusters are connected and grouped within a tile, and all tiles are attached to a global LLC. Coherence between cluster caches is software managed, and supported via specialized synchronization structures. The Rigel architecture closely relates to the architecture we consider in our work. In Rigel, however, data replication is allowed across cluster caches, which makes it necessary to support cache coherence. It is implemented at software level, adding responsibility to the application. In our architecture there is no need to keep coherence, which means no coherence traffic and no need to deal with coherence issues.

6.2 Throughput-Aware Register Files

As it was discussed in previous section, there are several commercial products and research projects which tackle the bandwidth wall by optimizing the on-chip cache hierarchy, with special attention to the last-level cache. In some other works, the emphasis was on the off-chip memory interface and associated bandwidth partitioning or management ideas. There are also architectures that incorporate local memories close to the cores, as in the Cell/B.E chip [58]. Regardless of the adopted strategy, all those approaches are intended to keep large amounts of data as close as possible to the processing units.

In such prior work, there has been no attention paid to the possibility of significantly re-architecting the user-addressable register file organization, even though it constitutes the closest data storage to the processing logic in terms of access time. Consequently, in this section we discuss some relevant products and research works in the realm of throughput-aware computation, even if they do not present significant innovations at register file level. In particular, we focus on commercial designs for base station applications and we present a qualitative comparison with graphics processing units (GPUs).

6.2.1 Commercial Products

The base station processor market is unquestionably driven by the continuously increasing demand on smartphones (expected to exceed 700 million units by 2015) and cellular-enabled tablets [43]. A steady increase in the number of deployed macro and micro base stations is expected, with a peak at slightly over 1 million units per year in 2012. This will be followed by a slight decline in macro and micro base stations, but it will be accompanied by an explosive growth for smaller pico and femto base stations, exceeding 7.8 million units deployed per year by 2014 [95].

To target pico and femto base stations, TI has announced KeyStone TCI6612 and TCI6614 processors [43]. Those designs include a Cortex-A8 CPU, C66x DSPs (two in the TCI6612 and four in the TCI6614) and a set of hardware accelerators. The accelerators are intended to help CPU and

DSP cores to offload tasks such as FFTs, Viterbi decoding and Turbo encoding/decoding.

Freescale has presented Qonverge PSC9132, a system-on-chip (SoC) that integrates two PowerPC e500 CPUs with two StarCore DSPs in the same chip [39]. The PSC9132 supports the performance requirements of the 3GPP Long Term Evolution (LTE) wireless standard [1], for a 20 MHz single sector, by handling 150 Mbps downlink and 75 Mbps uplink rates. This is half the maximum data rate supported by its direct competitors, the TI KeyStone TCI6612 and TCI6614 processors (300 Mbps downlink, and 150 Mbps uplink).

Mindspeed, another chip maker in the base station market, presented the Transcede 4000/4020 SoCs in 2010. These SoCs incorporate two Cortex-A9 CPUs, ten CEVA® DSPs and ten DSP accelerators. The Transcede architecture, as announced by Mindspeed, is conceived to support base stations ranging in size from macrocells to picocells [70].

Picochip, a pioneer company in the market of small base stations (femtocells), has developed a family of devices for residential and small-business users. The picoXcell PC323 SoC, the most advanced femtocell solution in that family, implements a physical-layer NodeB (base station), including an ARM11 processor, a cryptographic engine, high-speed accelerators, and peripherals to support the requirements of the 3GPP Evolved High-Speed Packet Access (HSPA+) wireless standard [81].

From the solutions presented in this section, mainly conceived for the niche of small base stations, it is possible to infer that the trend is toward designs with two key characteristics. First, the new solutions should integrate CPUs, hardware accelerators and DSPs in the same chip. And, second, they have to be flexible-enough (programmable) to handle the ecosystem of cellular protocols available today. The solution that we present in this thesis, based on the PowerEN processor, shares both. Moreover, the in-line acceleration engine included in the PowerEN cores is universal (and fully programmable), suitable to handle the current and future protocols, with minimum (or no) impact on hardware re-design. In this regard, the design presented enables the signal processing as well as higher-layer functions and

IT-oriented functions to be run on a common computing platform with a single architecture, a single toolset, and a single programming model.

Throughput-Aware Register Files vs. Graphics Processing Units

The *processor-in-regfile* (PIR) strategy presented in Chapter 4 resembles the *streaming multiprocessor* (SM) architecture found in modern graphics processing units (GPUs) [76, 77]: multiple *simple* processing elements operating in a SIMD fashion on large amounts of data. However, there are features that make the very-large register file with embedded logic more appealing than GPUs for certain applications. For example, the Fast Fourier Transform and Turbo Decoding applications (discussed in Sections 4.4.3 and 4.5.1, respectively) benefit from the register file's low wire latency and conventional data movement instructions during the data shuffling parts, while the communication cost between GPU threads is usually higher. This is because GPUs are well-optimized for embarrassingly parallel computation where parallel tasks execute more or less independently, without communication. In addition, even though the proposed register file incorporates specialized embedded logic, it is part of a general-purpose core and can be still used for general-purpose computation. This flexibility is not present in GPUs, which are special-purpose computation engines. Table 6.2 presents a list of their most relevant similarities and differences.

6.2.2 Research Projects

The register file organization presented in this thesis is based on Derby *et al.*'s work [29]. Authors propose VICTORIA, an *auxiliary processing unit* (APU) which connects to a host core to provide in-line acceleration. VICTORIA incorporates a very-large vector register file with enough capacity to hold sizable intermediate results. This helps to reduce the negative effects of limited memory bandwidth and high memory latency. To access such amount of registers, VICTORIA builds on the *indirect* VMX (iVMX) architecture, which provides support for indirect access to the very-large register file using operand-associated mappings. As we mentioned before, the register file

	PIR-based Register File	**GPU Streaming Multiprocessor**
Processing element count	$8 \sim 16$	32 [76], 192 [77]
Storage size	64-KB register file	64-KB shared memory
Programmability	relatively simple (in the end, it is a register file)	more complex, it requires software level support
Execution model	lockstep execution (all LCEs execute the same operations at the same time)	independent execution threads
Data movement cost	relatively cheap, it benefits from the low wire latency to move data between banks	costly, not well suited for applications that demand communication between threads
Application domain	general-purpose computation as well as highly parallel computation	embarrassingly parallel computation
Parallelism level	Large	Huge

Table 6.2: A comparative summary of a register file with embedded SIMD support and a GPU streaming multiprocessor.

and in-line accelerator presented in Chapter 4 are based on VICTORIA. The most important innovation in our case is that our register file is partitioned into multiple banks with embedded logic attached to each one. In addition, we widen SIMD support from 16 to 32 bytes. Multiple banks with embedded logic and wider SIMD make our design effective for throughput-aware domains.

Dally *et al.* [26] present an efficient low-power microprocessor (ELM), intended to provide the energy-efficiency of application-specific integrated circuits (ASICs), but with the flexibility of programmable processors. In their work, the authors identify the main overheads in programmable pro-

cessors that make them inadequate for embedded applications. Their main argument is that, in a programmable processor, data and instructions are supplied in a inefficient way: e.g., for a 10-pJ arithmetic operation, the processor spends 70 pJ on instruction supply and 47 pJ on data supply. One technique adopted in ELM to improve energy efficiency is the placement of a small, four-entry operand register file (ORF) on each arithmetic/logic unit (ALU) input. This technique provides energy savings of up to 13× with respect to the use of a conventional general register file. Similar to Dally *et al.*'s work, our register file organization is also intended to provide acceleration capabilities of specialized hardware, but with the flexibility of programmable processors. However, we consider that our approach has a significant advantage in ease of programming and use. First, because it works as a conventional load/store architecture. And second, because data movement between banks can be performed with conventional register manipulation instructions. On the other hand, ELM requires that the compiler takes additional responsibility coordinating the movement of data between ORFs and ALUs, among other tasks.

Khailany *et al.* [61] introduce Storm-1, a stream processor SoC designed to meet the demands for embedded signal processing. The Storm-1 processor contains two CPU cores for running the main application threads, a set of integrated I/Os for embedded systems, and a data-parallel unit (DPU) that runs kernels using a stream-processing execution model. In the DPU, sixteen data-parallel lanes combine to deliver high performance. Each lane has a 16-KB lane register file (LRF) and ten VLIW function units. The function units' inputs are fed by dedicated 16-word 1-read 1-write operand register files (ORFs). This distributed ORF architecture enables more scalability than a traditional unified register file. The area and power of the ORFs scale linearly with the number of function units, whereas a multiported register file scales quadratically with the number of ports. In our case, the quadratic grow of area and power is mitigated by using multiple banks for the register file implementation. In addition, we consider that our design is easier to program and use due to the reasons mentioned before (load/store model and conventional register manipulation instructions).

6.3 Chip-Level Power Management

The failure of the Dennard's scaling rule and the power density increase not only gave rise to multi-core chips, as it was discussed in Chapter 5, but also to more power-aware designs. In the last decade, chip power reduction has been placed on the same level of importance as performance optimization. Today, most CMP designs incorporate some level of power management at chip level. Two popular techniques to reduce dynamic power are *clock gating* and *dynamic voltage and frequency scaling* (DVFS), while *power gating* is becoming an increasingly adopted knob to eliminate both dynamic and static power. In this section we present CMP architectures with on-chip power management, as well as research projects which relate to this dissertation.

6.3.1 Commercial Products

The IBM POWER7 processor implements both clock gating and DVFS [35]. In addition, it provides a variety of sensors to measure the environment and workloads under which the chip is operating. At system level, the EnergyScale firmware and microcontroller presented in Section 2.3.2 accesses those sensors to control POWER7 behavior at runtime. In other words, EnergyScale adapts the processor to the changing thermal conditions and workloads necessities. Its ultimate goal is to improve power-performance efficiency by either reducing power consumption while maintaining the same amount of performance or by increasing performance at the same power level. POWER7 also includes an on-chip controller which adjust voltage either automatically or based on directives from the EnergyScale microcontroller. Frequency can also be adjusted automatically in a per-core basis. Under maximum performance conditions, frequency is set at its maximum value, which is known as the *turbo mode*.

In the Intel Xeon processor, power management is controlled by an on-chip power control unit (PCU) [88]. The PCU receives the output of core-level voltage and temperature sensors, and dynamically sets appropriate voltage and frequency values accordingly (DVFS). The core voltage is variable from 0.85 to 1.1 V. The highest voltage is used for the turbo mode, while

the lowest voltage is used when all cores are active. An important feature of Xeon is LLC and per-core power gating (PCPG), which is also managed by the PCU.

The AMD Llano accelerated processor unit (APU) [16] is also worth commenting on. In contrast to IBM POWER7 and Intel Xeon, Llano is not a general-purpose processor but a SoC which combines general processor execution as well as graphics processing in the same die. Similar to POWER7 and Xeon, it also incorporates an on-chip power management controller (PMC), which optimizes power-performance efficiency across the different APU components. Power management in Llano is supported by both DVFS and power gating. The PMC can dynamically adjust voltage and frequency based on workloads performance and units activity. Power gating is extensively applied across the different components. For example, each core and its associated L2 cache can can be power gated individually, as well as the graphics unit.

In this dissertation we do not propose any on-chip power management innovation. Instead, the thread consolidation heuristic (TCH) presented in Chapter 5 is implemented at system software level. The reason for which we present the IBM POWER7, Intel Xeon and AMD Llano processors in this section is because our heuristic can benefit extensively from the power management capabilities offered by these designs. In particular, we implement and evaluate TCH on top of POWER7. However, the heuristic can be extended to, and provide power-performance benefits in, other CMP processors with per-core DVFS and/or power gating capabilities.

6.3.2 Research Projects

The idea of thread consolidation in SMT-enabled CMPs is discussed in few prior works. Among them, the most closely related to ours is by Cochran *et al.* [24]. They propose to *pack* software threads onto a variable number of cores to fit a given power budget, in conjunction with dynamic voltage/frequency scaling (DVFS). The work examines different thread packing and DVFS configurations to maximize performance within variable power

caps, but it does not take into account the actual software-hardware thread mapping. In contrast, we show that asymmetries between logical cores (usually "seen" as *uniform* units) significantly affect performance and power consumption. Compared to their technique, the thread consolidation heuristic (TCH) proposed in this thesis is much simpler and does not require any kind of off-line analysis. In addition, we also consider power gating idle cores, because it is a more aggressive power saving technique compared to DVFS.

Tam *et al.* [96] propose a mechanism for thread clustering based on data sharing patterns. It is implemented at OS kernel level with information from hardware event counters. This work closely relates to ours in the methodology they use, which is also based on dynamic analysis of processor counters. However, they just tackle performance improvement while we also consider power reduction.

Gomaa *et al.* [41] propose a technique to cope with chip overheating by leveraging SMT in CMPs. Threads are distributed across cores to maximize heat generation in each core. When a core reaches its critical temperature, threads are migrated to other non-heated cores to allow cooling.

Rangan *et al.* [82] present *thread motion*, a technique capable of fine-grained power management based on thread migration between cores with different voltage/frequency (VF) settings.

The use of hardware event counters in the context of multi-threaded applications is also leveraged in prior studies. In addition to Tam *et al.*'s work, Bhattacharjee *et al.* [11] also propose the use of processor counters to dynamically predict *thread criticality*. A critical thread is the slowest thread in an application, which limits its performance. They propose to exploit thread criticality prediction for load balancing and energy saving purposes.

Regarding power management techniques, it is worth mentioning Isci *et al.*'s work [52]. They propose the adoption of a global power manager in the context of CMPs which senses per-core power and performance information at runtime. Based on variations in application behavior, the power manager sets particular per-core power levels to fit a power budget.

Madan *et al.* [67] present a study of two basic PCPG heuristics and their potential flaws. The heuristics are aimed to reduce power consumption of

idle cores. The paper also analyzes possible "holes" (which may produce negative power savings) and proposes a guard mechanism to prevent them.

Also related to robustness of power management in multi-core processors, Bose *et al.* [14] provides a broad overview of the potential holes and introduce the idea of guarded power management.

Power gating in the context of multi-core processors is also tackled by Musoll [72]. This work considers CMPs where cores are grouped in clusters. In this scenario, a cluster can be power gated when all its cores are idle. The author proposes a load-balancing mechanism to distribute load across clusters and across cores within a cluster. The goal is to reduce overall power consumption and avoid hotspots with minimal performance degradation.

Teodorescu *et al.* [98] present scheduling algorithms to benefit from existing within-die variations in CMPs. The objective of this work is to maximize throughput at a given power budget.

Meisner *et al.* [68] propose PowerNap, a power management technique to reduce the power consumed by idle components in a server. PowerNap is aimed to minimize the power consumed by an idle server, as well as its transition time in response to instantaneous load.

To meet a global power budget in a CMP, Cebrian *et al.* [21] propose the use of *power tokens*. Cores exchange tokens in order to balance the CMP power consumption (e.g. a core under its local budget can cede its remaining tokens to cores over the budget).

It is important to note that most of the works referenced in this section are based on simulations. In contrast, in this thesis we implement and evaluate the proposed heuristic in a real system.

6.4 Summary and Concluding Remarks

This chapter presents an overview of the related work in the context of power and performance optimizations for throughput-aware computation. We compare the LLC designs of two commercial products (the IBM POWER7 and Intel Xeon processors) against the one proposed in this thesis. We also discuss academic projects which tackle the bandwidth wall problem. Some of

those works present new LLC designs while others focus on models to study the memory bandwidth restrictions in CMPs.

We find no commercial processors with significant innovations in the register file for throughput-aware computation. For such reason and due to the context where we evaluate our design, we limit our discussion to SoCs for base station applications. By comparing our in-line accelerator (with a very-large register file and embedded logic) against base station processors, we want to emphasize that our solution can handle the ecosystem of cellular protocols available today with a fully-programmable approach. In contrast, most base station processors today achieve the throughput demanded by the current standards by means of specialized hardware accelerators. We also comment on two research projects, the efficient low-power microprocessor (ELM) and the Storm-1 processor, which adopt operand register files (ORFs) to provide more scalability than a traditional register file.

Finally, we analyze the most relevant innovations for chip-level power management in three state-of-the-art CMPs: IBM POWER7, Intel Xeon and AMD Llano. Even if our work about power management takes place at system software level, the proposed heuristic leverages chip-level power management capabilities to generate power-performance efficiency benefits. For example, thread consolidation can aggressively reduced chip power consumption by power gating unused cores. We also discuss a variety of research projects which closely relate to this dissertation. They cover the topics of thread placement and clustering, use of hardware event counters for performance optimization, and power management techniques.

Chapter 7

Publications

This chapter includes the publications and intellectual property generated as the result of the research done during this thesis. Each section groups publications according to which part of the thesis they relate to. Within each section, publications are presented in reverse chronological order.

7.1 Last-Level Cache Design

- **Augusto Vega**, Felipe Cabarcas, Alex Ramírez, Mateo Valero. "Breaking the Bandwidth Wall in Chip Multiprocessors." In *Proceedings of the Eleventh International Conference on Embedded Computer Systems: Architectures, MOdeling, and Simulation (SAMOS 2011)*. Samos (Greece). July 2011.

- **Augusto Vega**, Alejandro Rico, Felipe Cabarcas, Alex Ramírez, Mateo Valero. "Comparing Last-level Cache Designs for CMP Architectures." In *Proceedings of the Second International Forum on Next-Generation Multicore/Manycore Technologies (IFMT 2010)*. Saint-Malo (France). June 2010.

7.2 Register File Design

- Dheeraj Sreedhar, Jeff Derby, **Augusto Vega**, Brian Rogers, Charles Johnson, Robert Montoye. "Processor Architecture for Software Implementation of Multi-sector G-Rake Receivers for HSUPA Wireless Infrastructure." In *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013)*. Vancouver (Canada). May 2013.

- **Augusto Vega**, Pradip Bose, Alper Buyuktosunoglu, Jeff Derby, Michele Franceschini, Charles Johnson, Robert Montoye. "Architectural Perspectives of Future Wireless Base Stations based on the IBM PowerEN Processor." In *Proceedings of the 18th International Symposium on High-Performance Computer Architecture (HPCA 2012)*. New Orleans (USA). February 2012.

- Jeff Derby, Timothy Heil, Michele Franceschini, Anil Krishna, Robert Montoye, Dheeraj Sreedhar, **Augusto Vega**, Hangu Yeo, Charles Johnson. "Vector-Based Acceleration in the IBM PowerEN Processor To Enable Software-Defined Radio." In *Proceedings of the 2011 Software Defined Radio Technical Forum (SDR 2011)*. Washington DC (USA). November 2011.

- **Augusto Vega**, Alper Buyuktosunoglu, Michele Franceschini, Robert Montoye, Jeff Derby, Pradip Bose. "Local Computation Logic Embedded in a Register File to Accelerate Programs." Publication number US20130046955 A1. Filed 08/17/2011. Published 02/21/2013.

7.3 Power Management

- **Augusto Vega**, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, Srinivasan Ramani. "Crank It Up or Dial It Down: Coordinated Multiprocessor Frequency and Folding Control." *Under review for the 46th International Symposium on Microarchitecture (MICRO 2013)*. Davis, California (USA). December 2013.

- **Augusto Vega**, Alper Buyuktosunoglu, Pradip Bose. "SMT-Centric Power-Aware Thread Placement in Chip Multiprocessors." In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT 2013)*. Edinburgh (Scotland). September 2013. *Accepted May 2013 — to appear.*

- **Augusto Vega**, Alper Buyuktosunoglu, Pradip Bose, Kyung Ryu, Bryan Rosenburg. "Method and System of Thread Consolidation and Core Folding for Power Savings in an Accelerator-Enabled Computational Node." Patent application filed December 2012. Patent pending.

- Priyanka Tembey, **Augusto Vega**, Alper Buyuktosunoglu, Dilma Da Silva, Pradip Bose. "SMT switch: Software Mechanisms for Power Shifting." In *IEEE Computer Architecture Letters* (accepted August 2012 – to appear).

- **Augusto Vega**, Pradip Bose, Alper Buyuktosunoglu. "Power-Aware Thread Placement in SMT/CMP Architectures." In *Proceedings of the Fourth Workshop on Energy Efficient Design (WEED 2012)*. Portland (USA). June 2012.

- Pradip Bose, Alper Buyuktosunoglu, John Darringer, Meeta Gupta, Michael Healy, Hans Jacobson, Indira Nair, Jude Rivers, Jeonghee Shin, **Augusto Vega**, Alan Weger. "Power Management of Multi-Core Chips: Challenges and Pitfalls." In *Proceedings of the 2012 Conference on Design, Automation and Test in Europe (DATE 2012)*. Dresden (Germany). March 2012.

7.4 Methodology and Tools

- Alejandro Rico, Felipe Cabarcas, Carlos Villavieja, Milan Pavlovic, **Augusto Vega**, Yoav Etsion, Alex Ramirez, Mateo Valero. "On the Simulation of Large-scale Architectures Using Multiple Application Abstraction Levels". In *Proceedings of the 7th International Confer-*

ence on High-Performance and Embedded Architectures and Compilers (HIPEAC 2012). Paris (France). January 2012.

- Veerle Desmet, Sylvain Girbal, Alex Ramírez, Olivier Temam, **Augusto Vega**. "ArchExplorer for Automatic Design Space Exploration." In *IEEE Micro Special Issue: European Multicore Processing Projects.* September/October 2010.

- Alejandro Rico, Felipe Cabarcas, Antonio Quesada, Milan Pavlovic, **Augusto Vega**, Carlos Villavieja, Yoav Etsion, Alex Ramírez. "Scalable Simulation of Decoupled Accelerator Architectures." In *Technical Report UPC-DAC-RR-2010-14.* Universitat Politècnica de Catalunya. June 2010.

7.5 Other Publications

- **Augusto Vega**. "Performance, Power and Thermal Modeling in 3D Die-Stacking Architectures." *Master Thesis. Department of Computer Architecture, Universitat Politècnica de Catalunya.* Barcelona (Spain). January 2009.

- **Augusto Vega**, Alex Ramírez, Mateo Valero. "3D Die-Stacking Architectures: State of the Art." In *Advanced Computer Architecture and Compilation for Embedded Systems (ACACES 2008).* L'Aquila (Italy), July 2008.

Chapter 8

Conclusions

This thesis presents three complementary innovations to tackle *memory bandwidth* and *power consumption* constraints in chip multiprocessors (CMPs) for throughput-aware computation. Both memory bandwidth and chip power consumption are key limiting factors for CMP performance scalability. Memory bandwidth is exacerbated every processor generation with the growing number of cores in the chip. Power consumption increases as a consequence of manufacturers' difficulty to lower operating voltages sufficiently to follow Dennard's scaling rule.

First, we present a bandwidth-optimized last-level cache (LLC) which is suitable for throughput-aware computation in CMPs. The proposed LLC avoids data replication to improve its effective capacity and, therefore, boost memory bandwidth. We leverage the benefits of software-managed streaming memory with direct memory access (DMA) transfers to hide the extra access latency that arises from the lack of data replication. Secondly, we present a novel bank-based vector register file with thousands of registers. Due to its size, data is kept as much as possible in the register file during computation, which further reduces the pressure on the memory system. We leverage the bank-based organization to exploit local computation in each bank., with embedded *per-bank* Single Instruction, Multiple Data (SIMD) *local computation elements* (LCEs). The design is promising in terms of throughput, power and area reduction, when it is evaluated in the context of applica-

tions for small base stations. Finally, we present a simple heuristic to reduce CMPs' power consumption. The heuristic works at system software level (e.g. the operating system kernel) by dynamically placing software threads across physical threads and cores to maximize power-performance efficiency. Its goal is to place software threads in as few cores as possible with minimal performance impact, and to power gate the cores that remain unused. We show important power-performance efficiency improvements when the proposed heuristic is adopted instead of the default Linux thread scheduling policy. Next, we summarize in more detail the work presented in this thesis.

8.1 Bandwidth-Optimized Last-Level Cache

The first contribution of this thesis is a novel last-level cache (LLC) organization, which is suitable for throughput-aware computation in CMPs. The LLC is divided into multiple independent *blocks*, each one being shared by a cluster of cores.

The proposed LLC possesses features which make it highly-optimized for bandwidth. First and foremost of these features is the lack of data replication to significantly improve the effective capacity of the cache. Instead, the memory address space is interleaved across LLC blocks and each core can access either its local block as well as remote ones (i.e. other clusters' blocks). This scheme results in better hit rates (and, hence, better memory bandwidth) compared to a LLC with data replication. The hit rate improvement is even more pronounced for larger CMPs because the impact of data replication in such cases increases significantly.

One key aspect of the presented LLC is the address space interleaving granularity. Our experiments show that the LLC bandwidth is maximized when the address space is spread with 128-byte interleaving granularity across blocks. In this way, multiple LLC accesses can proceed in parallel, at the expense of locality. The extra latency to access remote LLC blocks is hidden with the adoption of the double buffering technique, which overlaps DMA transfers with computation.

We perform a similar study about address space interleaving at the mem-

ory side. In this case, we also find that fine-grained interleavings across memory controllers help improving bandwidth. Memory interleaving is not a novel technique. Our contribution is to determine what is the best granularity to use across memory controllers in conjunction with the proposed LLC organization. We conclude that both, the proposed LLC and the fine-grained interleaved memory synergistically improve bandwidth and performance. For a CMP with 128 cores and 64-MB LLC, our organization shows 21% performance improvement over a traditional LLC. Additionally, the fine-grained interleaving across memory controllers provides an extra 42% improvement on performance. By adopting both optimizations, we obtain 72% total performance improvement.

8.2 Bandwidth-Optimized Register File

The second contribution of this thesis consists in a register file organization for throughput-aware computation, referred to as the Vector String Register File (VSRF). The VSRF is composed of 2048 256-bit registers, organized in eight banks, which are accessed through an indirection mechanism based on register mappings. In this thesis, the VSRF is studied in the context of an in-line vector-based accelerator (VBA) design [29], which is plugged into an A2 core.

The register file capacity is leveraged to keep data as much as possible and reduce the accesses to the cache hierarchy. For example, an application loads large blocks of data on the onset into the VSRF, operates on the entire block of data, keeps intermediate results in the VSRF, and stores final results at the end. This computation model reduces the pressure on the cache hierarchy, which results in higher throughput than some state-of-the-art digital signal processors (DSPs). For instance, a 2048-point fixed-point FFT executes at a rate of 1884 millions of samples per second (Msps) at 2.3GHz. This is 1.5 to 3.5 times higher than state-of-the-art DSP solutions, for the same clock frequency. Even more, the design proposed constitutes a fully-programmable, scalable design for a much broader range of applications than DSPs.

To further improve throughput, we propose to embed computation logic

into the register file. We refer to this approach as the *processor-in-regfile* (PIR) strategy. By leveraging the register file bank-based organization, we attach small special-function *local computation elements* (LCEs) to each bank. Each LCE is a SIMD computation element, and all of them can proceed concurrently. In other words, the PIR strategy constitutes a highly-parallel super-wide-SIMD device, appropriate for throughput-aware computation. For instance, a Turbo Decoder implemented at LCE level can decode a 6144-element codeword at a rate of 230 Mbps. In the context of base station applications, the throughput demanded by the 3GPP LTE wireless standard (75 Mbps) is met with just one VBA with VSRF and embedded LCEs. For LTE-Advanced, it can be satisfied with two to three VBAs proceeding in parallel.

8.3 Power Management Techniques for CMPs

The third contribution of this thesis is a simple heuristic capable of optimizing power-performance efficiency in CMPs. In the context of multi-threaded applications, software threads sometimes present high degrees of data sharing. In those scenarios, placing software threads closer (i.e. across fewer physical cores) may help performance. But placing threads in fewer cores also generates per-core power gating (PCPG) opportunities to reduce chip power consumption.

The proposed thread consolidation heuristic (TCH) is implemented at operating system level and evaluated in an IBM BladeCenter system with a POWER7 processor. TCH tracks an application's performance and power consumption by gathering performance counters and power readings from the chip. Based on this information, TCH gauges if software threads should be placed closer (*consolidation*) or moved away one from the other (*unconsolidation*). After taking a decision, TCH computes the performance and power benefits of the new thread placement with respect to the previous one. If the last decision (e.g., a consolidation) gives rise to a more power-performance efficient threads placement, TCH may decide to emphasize it (e.g., to consolidate more). Otherwise, TCH may undo the new placement and go back

to the previous (more efficient) one.

In chips with PCPG capabilities, this simple heuristic generates important opportunities to actuate the PCPG knob. In such scenarios, TCH can improve power-performance efficiency by a factor of up to 2.1 with respect to the default Linux scheduler.

Bibliography

[1] 3GPP. LTE release 8 major parameters. `http://www.3gpp.org/LTE`.

[2] 3GPP TR 36.913 V10.0.0 (2011-03). 3rd generation partnership project; technical specification group radio access network; requirements for further advancements for evolved universal terrestrial radio access (E-UTRA) (LTE-Advanced) (release 10); 2011.

[3] 3GPP TS 36.211 V9.1.0 (2010-03). 3rd generation partnership project; technical specification group radio access network; evolved universal terrestrial radio access (E-UTRA); physical channels and modulation (release 9); 2010.

[4] Advanced Micro Devices, Inc. AMD accelerated processing units. `http://fusion.amd.com`.

[5] D. Anand, K. Gorman, M. Jacunski, and A. Paparelli. Embedded DRAM in 45-nm technology and beyond. *IEEE Design Test of Computers*, 28(1):14 –21, January-February 2011.

[6] M. Azimi, N. Cherukuri, D. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. Vaidya. Integration challenges and tradeoffs for tera-scale architectures. *Intel Technology Journal*, 11(3):173–184, August 2007.

[7] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, 20(2):284–287, March 1974.

[8] L. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40:33–37, December 2007.

[9] P. Bellens, J. M. Perez, R. M. Badia, and J. Labarta. CellSs: a programming model for the Cell BE architecture. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, 2006.

[10] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *Proceedings of the IEEE International Conference on Communications*, volume 2 of *ICC '93*, pages 1064–1070, May 1993.

[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, 2009.

[12] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[13] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, 2008.

[14] P. Bose, A. Buyuktosunoglu, J. Darringer, M. Gupta, M. Healy, H. Jacobson, I. Nair, J. Rivers, J. Shin, A. Vega, and A. Weger. Power management of multi-core chips: Challenges and pitfalls. In *Proceedings of the Design, Automation Test in Europe Conference*, DATE '12, pages 977 –982, 2012.

[15] D. Bovet and M. Cesati. *Understanding The Linux Kernel*. O'Reilly & Associates Inc, 2005.

[16] A. Branover, D. Foley, and M. Steinman. AMD Fusion APU: Llano. *IEEE Micro*, 32(2):28 –37, March-April 2012.

[17] J. Brown, S. Woodward, B. Bass, and C. Johnson. IBM Power Edge of Network processor: A wire-speed system on a chip. *IEEE Micro*, 31(2):76–85, March-April 2011.

[18] D. Burgess, E. Gieske, J. Holt, T. Hoy, and G. Whisenhunt. e6500: Freescale's low-power, high-performance multithreaded embedded processor. *IEEE Micro*, 32(5):26–36, September-October 2012.

[19] G. J. Burnett and E. G. Coffman, Jr. A study of interleaved memory systems. In *Proceedings of the May 5-7, 1970, spring joint computer conference*, AFIPS '70 (Spring), pages 467–474, 1970.

[20] D. R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[21] 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 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 431–442, 2011.

[22] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[23] T. Chen, Z. Sura, K. O'Brien, and J. K. O'Brien. Optimizing the use of static buffers for DMA on a Cell chip. In *Proceedings of the 19th International Conference on Languages and Compilers for Parallel Computing*, LCPC'06, pages 314–329, 2007.

[24] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & cap: adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '11, pages 175–185, 2011.

[25] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[26] W. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. C. Harting, V. Parikh, J. Park, and D. Sheffield. Efficient embedded computing. *IEEE Computer*, 41:27–32, July 2008.

[27] R. Dennard, F. Gaensslen, H. Yu, V. Rideout, E. Bassous, and A. Leblanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Solid-State Circuits Newsletter*, 12(1):38–50, winter 2007.

[28] J. Derby, T. Heil, M. Franceschini, A. Krishna, R. Montoye, D. Sreedhar, A. Vega, H. Yeo, and C. Johnson. Vector-based acceleration in the IBM PowerEN™ processor to enable software-defined radio. In *Proceedings of the 2011 Software Defined Radio Technical Forum (SDR 2011)*, Washington, DC, USA, November 2011.

[29] J. Derby, R. Montoye, and J. Moreira. VICTORIA: VMX indirect compute technology oriented towards in-line acceleration. In *Proceedings of the 3rd Conference on Computing Frontiers*, CF '06, pages 303–312. ACM, 2006.

[30] K. Diefendorff. POWER4 focuses on memory bandwidth. *Microprocessor Report*, pages 11–17, October 1999.

[31] G. S. Ditlow, R. K. Montoye, S. N. Storino, S. M. Dance, S. Ehrenreich, B. M. Fleischer, T. W. Fox, K. M. Holmes, J. Mihara, Y. Nakamura, S. Onishi, R. Shearer, D. Wendel, and L. Chang. A 4R2W register file for a 2.3GHz wire-speed POWER™ processor with double-pumped write operation. In *Proceedings of the 2011 IEEE International Solid-State Circuits Conference*, ISSCC '11.

[32] B. Elkin and V. Indukuru. Commonly used metrics for performance analysis - POWER7. `https://www.power.org/events/Power7/POWER7_Commonly_Used_Metrics_for_Performance_Analysis.pdf`.

[33] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Pro-*

ceedings of the 38th Annual International Symposium on Computer Architecture, ISCA '11, pages 365–376, 2011.

[34] G. Fettweis and E. Zimmermann. ICT energy consumption - trends and challenges. In *Proceedings of the 11th International Symposium on Wireless Personal Multimedia Communications*, WPMC '08, pages 2006–2009, September 2008.

[35] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, A. Drake, L. Pesantez, T. Gloekler, J. Tierno, P. Bose, and A. Buyuktosunoglu. Introducing the adaptive energy management features of the POWER7 chip. *IEEE Micro*, 31(2):60–75, March-April 2011.

[36] M. Floyd, M. Ware, K. Rajamani, T. Gloekler, B. Brock, P. Bose, A. Buyuktosunoglu, J. C. Rubio, B. Schubert, B. Spruth, J. A. Tierno, and L. Pesantez. Adaptive energy-management features of the IBM POWER7 chip. *IBM Journal of Research and Development*, 55(3):8:1–8:18, May-June 2011.

[37] H. Franke, J. Xenidis, C. Basso, B. Bass, S. Woodward, J. Brown, and C. Johnson. Introduction to the wire-speed processor and architecture. *IBM Journal of Research and Development*, 54:27–37, January 2010.

[38] Freescale Semiconductor, Inc. Beyond DSPs. Giving customers another choice. `http://www.freescale.com/files/dsp/doc/brochure/BYNDDSPBRO.pdf`.

[39] Freescale Semiconductor, Inc. QorIQ Qonverge PSC9132 for picocell base station solutions. `http://www.freescale.com/files/32bit/doc/fact_sheet/QORIQPSC9132FS.pdf`.

[40] M. Garland and D. Kirk. Understanding throughput-oriented architectures. *Communications of the ACM*, 53:58–66, November 2010.

[41] M. Gomaa, M. D. Powell, 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*, ASPLOS XI, pages 260–270, 2004.

[42] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser. Many-core vs. many-thread machines: Stay away from the valley. *IEEE Computer Architecture Letters*, 8(1):25–28, 2009.

[43] L. Gwennap. TI tackles small base stations. *Microprocessor Report*, pages 10–15, June 2011.

[44] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 184–195, 2009.

[45] J.-M. Hsu and C.-L. Wang. A parallel decoding scheme for turbo codes. In *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, volume 4 of *ISCAS '98*, pages 445–448, May 1998.

[46] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, ISLPED '04, pages 32–37, 2004.

[47] H. Inoue, T. Moriyama, H. Komatsu, and T. Nakatani. AA-sort: A new parallel sorting algorithm for multi-core SIMD processors. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT '07, pages 189–198. IEEE Computer Society, September 2007.

[48] Intel Corp. First the tick, now the tock: Next generation Intel microarchitecture (Nehalem). `http://www.intel.com/content/dam/doc/white-paper/intel-microarchitecture-white-paper.pdf`.

[49] Intel Corp. Intel® Xeon® processor E5 family. `http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-5000-sequence.html`.

[50] Intel Corp. Threading building blocks. `http://www.threadingbuildingblocks.org/`, 2008.

[51] Intel Corp. Intel Xeon processor 7500 series. `http://www.intel.com/Assets/PDF/datasheet/323341.pdf`, 2010.

[52] 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*, MICRO '06, pages 347–358, 2006.

[53] C. Isci, A. Buyuktosunoglu, and M. Martonosi. Long-term workload phases: Duration predictions and applications to DVFS. *IEEE Micro*, 25(5):39–51, September 2005.

[54] ITRS. International Technology Roadmap for Semiconductors. 2006 update. `http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm`.

[55] S. S. Iyer, J. E. Barth, Jr., P. C. Parries, J. P. Norum, J. P. Rice, L. R. Logan, and D. Hoyniak. Embedded DRAM: Technology platform for the Blue Gene/L chip. *IBM Journal of Research and Development*, 49(2.3):333 –350, March 2005.

[56] S. S. Iyer, G. Freeman, C. Brodsky, A. I. Chou, D. Corliss, S. H. Jain, N. Lustig, V. McGahay, S. Narasimha, J. Norum, K. A. Nummy, P. Parries, S. Sankaran, C. D. Sheraw, P. R. Varanasi, G. Wang, M. E. Weybright, X. Yu, E. Crabbe, and P. Agnello. 45-nm silicon-on-insulator CMOS technology integrating embedded DRAM for high-performance server and ASIC applications. *IBM Journal of Research and Development*, 55(3):5:1 –5:14, May-June 2011.

[57] C. Johnson, D. Allen, J. Brown, S. Vanderwiel, R. Hoover, H. Achilles, C.-Y. Cher, G. May, H. Franke, J. Xenedis, and C. Basso. A wire-speed Power^{TM} processor: 2.3GHz 45nm SOI with 16 cores and 64 threads. In *Proceedings of the 2010 IEEE International Solid-State Circuits Conference*, ISSCC '10, pages 104–105, February.

[58] J. Kahle, M. Day, P. Hofstee, C. Johns, T. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, 49(4/5):589–604, 2005.

[59] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. POWER7: IBM's next-generation server processor. *IEEE Micro*, 30(2):7–15, March-April 2010.

[60] J. Kelm, D. Johnson, M. Johnson, N. Crago, W. Tuohy, A. Mahesri, S. Lumetta, M. Frank, and S. Patel. Rigel: an architecture and scalable programming interface for a 1000-core accelerator. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 140–151, 2009.

[61] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. Dally. A programmable 512 GOPS stream processor for signal, image, and video processing. *IEEE Journal of Solid-State Circuits*, 43(1):202–213, January 2008.

[62] J. Koomey. Growth in data center electricity use 2005 to 2010. `http://www.analyticspress.com/datacenters.html`, 2011.

[63] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proceedings of the 4th International Conference on Autonomic Computing*, ICAC '07, pages 4–, 2007.

[64] J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis. Comparative evaluation of memory models for chip multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 5:12:1–12:30, December 2008.

[65] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power management of datacenter workloads using per-core power gating. *IEEE Computer Architecture Letters*, 8(2):48–51, February 2009.

[66] F. Liu, X. Jiang, and Y. Solihin. Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance. In *Proceedings of the 16th International Symposium on High-Performance Computer Architecture*, HPCA '10, pages 1–12, January 2010.

[67] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram. A case for guarded power gating for multi-core processors. In *Proceedings of the 17th International Symposium on High-Performance Computer Architecture*, HPCA '11, pages 291–300, 2011.

[68] D. Meisner, B. Gold, and T. Wenisch. PowerNap: eliminating server idle power. In *Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 205–216, 2009.

[69] A. Mericas, B. Elkin, and V. R. Indukuru. Comprehensive PMU event reference - POWER7. `http://www.power.org/documentation/comprehensive-pmu-event-reference-power7/`.

[70] Mindspeed Technologies, Inc. Baseband processors. `http://www.mindspeed.com/products/baseband-processors`.

[71] I. Molnar. Modular scheduler core and completely fair scheduler [CFS]. `http://lwn.net/Articles/230501/`.

[72] E. Musoll. Energy and thermal tradeoffs in hardware-based load balancing for clustered multi-core architectures implementing power gating. In *Proceedings of the 2008 IEEE Symposium on Application Specific Processors*, SASP '08, pages 89 –94, 2008.

[73] J. Nickolls and W. Dally. The GPU computing era. *IEEE Micro*, 30:56–69, March 2010.

[74] Nvidia Corp. GeForce GTX 580. `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580`.

[75] Nvidia Corp. NVIDIA CUDA C programming guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`.

[76] Nvidia Corp. Nvidia's next generation CUDA compute architecture: Fermi. `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`.

[77] Nvidia Corp. Tesla® Kepler™ GPU accelerators. `http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf`.

[78] P. R. Panda, A. Shrivastava, B. Silpa, and K. Gummidipudi. *Power-Efficient System Design*. Springer, 1st edition, 2010.

[79] D. Patterson. The top 10 innovations in the new NVIDIA Fermi architecture, and the top 3 next challenges. Technical report, NVidia, 2009.

[80] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A case for intelligent RAM. *IEEE Micro*, 17:34–44, March 1997.

[81] Picochip Ltd. PC323 HSPA+ SoC. `http://www.picochip.com/page/99/`.

[82] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: fine-grained power management for multi-core systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 302–313, 2009.

[83] A. Rico, F. Cabarcas, A. Quesada, M. Pavlovic, A. Vega, C. Villavieja, Y. Etsion, and A. Ramirez. Scalable simulation of decoupled accelerator architectures. Technical Report UPC-DAC-RR-2010-10, University of Catalonia, 2010.

[84] A. Rico, F. Cabarcas, C. Villavieja, M. Pavlovic, A. Vega, Y. Etsion, A. Ramirez, and M. Valero. On the simulation of large-scale architectures using multiple application abstraction levels. *ACM Transactions on Architecture and Code Optimization*, 8(4):36:1–36:20, January 2012.

[85] A. Rico, J. Derby, R. Montoye, T. Heil, C.-Y. Cher, and P. Bose. Performance and power evaluation of an in-line accelerator. In *Proceedings of the 7th ACM International Conference on Computing Frontiers*, CF '10, pages 81–82, 2010.

[86] B. Rogers, A. Krishna, G. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 371–382, 2009.

[87] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, and S. Kottapalli. A 45nm 8-core enterprise Xeon® processor. In *Proceedings of the 2009 IEEE International Solid-State Circuits Conference*, ISSCC '09, pages 56 –57, feb. 2009.

[88] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora. Power reduction techniques for an 8-core Xeon® processor. In *Proceedings of the 2009 European Solid-State Circuits Conference*, ESSCIRC '09, pages 340–343, September 2009.

[89] S. Schneider, J. Yeom, and D. Nikolopoulos. Programming multiprocessors with explicitly managed memory hierarchies. *IEEE Computer*, 42:28–34, December 2009.

[90] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of*

the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS X, pages 45–57, 2002.

[91] E. Siever, S. Figgins, A. Weber, R. Love, and A. Robbins. *Linux in a Nutshell*. O'Reilly Media, Inc., 2005.

[92] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams. IBM POWER7 multicore server processor. *IBM Journal of Research and Development*, 55(3):1:1 –1:29, May-June 2011.

[93] H. Sloate. Matrix representations for sorting and the fast Fourier transform. *IEEE Transactions on Circuits and Systems*, 21(1):109–116, January 1974.

[94] A. J. Smith. Multiprocessor memory organization and memory interference. *Communications of the ACM*, 20(10):754–761, 1977.

[95] Strategy Analytics. Base station power amplifier transistor market expected to exceed $1 billion in 2014. `http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5006`.

[96] D. Tam, R. Azimi, and M. Stumm. Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 47–58, 2007.

[97] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, January 2002.

[98] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 363–374, 2008.

[99] F. W. Terman. A study of interleaved memory systems by trace driven simulation. In *ANSS '76: Proceedings of the 4th symposium on Simulation of computer systems*, pages 3–9, Piscataway, NJ, USA, 1976. IEEE Press.

[100] Texas Instruments Inc. TMS320TCI6616. Breakthrough performance for wireless base stations. `http://www.ti.com/lit/ml/sprt579/sprt579.pdf`.

[101] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 51–62, 2008.

[102] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. Cacti 5.0. Technical Report HPL-2007-167, HP Labs.

[103] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *Proceedings of the 25 years of the International Symposia on Computer Architecture (selected papers)*, ISCA '98, pages 533–544, 1998.

[104] D. Wendel, R. Kalla, R. Cargoni, J. Clables, J. Friedrich, R. Frech, J. Kahle, B. Sinharoy, W. Starke, S. Taylor, S. Weitzel, S. Chu, S. Islam, and V. Zyuban. The implementation of POWER7$^{\text{TM}}$: A highly parallel and scalable multi-core high-end server processor. In *Proceedings of the 2010 IEEE International Solid-State Circuits Conference*, ISSCC '10, pages 102 –103, 2010.

[105] L. Zhao, R. Iyer, M. Upton, and D. Newell. Towards hybrid last level caches for chip-multiprocessors. *ACM SIGARCH Computer Architecture News*, 36(2):56–63, 2008.

[106] V. Zyuban, J. Friedrich, C. J. Gonzalez, R. Rao, M. D. Brown, M. M. Ziegler, H. Jacobson, S. Islam, S. Chu, P. Kartschoke, G. Fiorenza,

M. Boersma, and J. A. Culp. Power optimization methodology for the IBM POWER7 microprocessor. *IBM Journal of Research and Development*, 55(3):267–275, May 2011.