Shared resource aware scheduling on power-constrained tiled many-core processors☆

Sudhanshu Shekhar Jha [a,*], Wim Heirman [b], Ayose Falcón [c], Jordi Tubella [a], Antonio González [a], Lieven Eeckhout [d]

[a] DAC, Universitat Politécnica de Catalunya, Spain
[b] Intel Corporation, Belgium
[c] HP Inc., Spain
[d] ELIS, Ghent University, Belgium

HIGHLIGHTS

- A low-overhead and high scalable hierarchical power manager on a tiled many-core architecture with shared LLC and VR.
- Shared DVFS and cache adaptation can degrade performance of co-scheduled threads on a tile.
- DVFS and cache-aware thread migration (DCTM) to ensure optimum per-tile co-scheduling of compatible threads at runtime.
- DCTM assisted hierarchical power manager improves performance by up to 20% compared to conventional centralized power manager with per-core VR.

ARTICLE INFO

Keywords:
Many-core tiled architecture
Thread migration
Power budget
Adaptive microarchitecture

ABSTRACT

Power management through dynamic core, cache and frequency adaptation is becoming a necessity in today's power-constrained many-core environments. Unfortunately, as core count grows, the complexity of both the adaptation hardware and the power management algorithms increases exponentially. This calls for hierarchical solutions, such as on-chip voltage regulators per-tile rather than per-core, along with multi-level power management. As power-driven adaptation of shared resources affects multiple threads at once, the efficiency in a tile-organized many-core processor architecture hinges on the ability to co-schedule compatible threads to tiles in tandem with hardware adaptations per tile and per core.

In this paper, we propose a two-tier hierarchical power management methodology to exploit per-tile voltage regulators and clustered last-level caches. In addition, we include a novel thread migration layer that (i) analyzes threads running on the tiled many-core processor for shared resource sensitivity in tandem with core, cache and frequency adaptation, and (ii) co-schedules threads per tile with compatible behavior. On a 256-core setup with 4 cores per tile, we show that adding sensitivity-based thread migration to a two-tier power manager improves system performance by 10% on average (and up to 20%) while using 4× less on-chip voltage regulators. It also achieves a performance advantage of 4.2% on average (and up to 12%) over existing solutions that do not take DVFS sensitivity into account.

1. Introduction

Industry-wide adoption of chip multiprocessors (CMPs) is driven by the need to maintain the performance trend in a power-efficient way on par with Moore's law [40]. With continued emphasis on technology scaling for increased circuit densities, controlling chip power consumption has become a first-order design constraint. Due to the end of Dennard scaling [12] (slowed supply voltage scaling), we may become so power-constrained that we are no longer able to power on all transistors at the same time—*dark silicon* [16]. Runtime factors such as thermal emergencies [7] and power capping [19] further constrain the available chip power. Owing to all the above factors, power budgeting on many-core systems has received considerable attention recently [22,36,37,39,49,51].

☆ This work is a collaborative effort.

* Corresponding author.
E-mail addresses: sjha@ac.upc.edu (S.S. Jha), wim.heirman@intel.com (W. Heirman), ayose.falcon@hp.com (A. Falcón), jordit@ac.upc.edu (J. Tubella), antonio@ac.upc.edu (A. González), lieven.eeckhout@ugent.be (L. Eeckhout).

Dynamic voltage and frequency scaling (DVFS) for multiple clock domain micro-architectures has been studied extensively in prior work [11,24,25,49,52]. Current commercial implementations of fully integrated voltage regulators (FIVR) [8,32] support multiple on-chip frequency/voltage domains with fast adaptation, although per-core voltage regulators incur significant area overhead—previous works [8,31,48] suggest that the area of on-die per core voltage regulators is approximately 12.5% of core area. Other techniques such as core micro-architecture adaptation [3,13,20,43,30], cache adaptation [1,38,53,46] and network-on-chip adaptation [46] have been shown to be quite effective at managing power in isolation at high to moderate power budgets. Under more stringent power conditions, core gating [36,33] along with the above techniques can be used at the potential risk of starving threads.

Most existing power management schemes use a centralized approach to regulate power dissipation based on power monitoring and performance characteristics. Unfortunately, the complexity and overhead of centralized power management increases exponentially with core count [14]. Moreover, the area overhead of on-chip voltage regulators is significant which limits the number of voltage/frequency domains one can have on the chip. Hence, it becomes a necessity to employ a hierarchical approach as we scale fine-grain power management to large many-core processors at increasingly stringent power budgets. We therefore propose a *two-tier hierarchical power manager* for tile-based many-core architectures; each tile consists of a small number of cores and a shared L2 cache within a single voltage–frequency domain. The two-tier power manager first distributes power across tiles, and then across cores within a tile. The architecture also provides support for core, cache and frequency adaptations to avoid core gating at moderate to stringent power budgets.

Tiled many-core processors pose an interesting challenge when it comes to hardware adaptation and scheduling. Changing frequency and re-configuring the shared L2 cache affects all threads running in the tile. It therefore becomes important to *migrate* threads, such that threads with *compatible* behavior are co-scheduled onto the same tile. Since the execution behavior varies over time, periodic re-evaluation and dynamic thread migration is also required. We therefore classify threads based on their sensitivity to both cache and frequency dynamically at runtime. We propose DVFS and Cache-aware Thread Migration (*DCTM*): a scheduler running on top of the two-tier hierarchical power manager to ensure an optimal co-schedule for all threads running on the power-constrained tiled many-core processor while accounting for the effects of hardware adaptation.

In this work, we make the following contributions:

- We propose an *integrated two-tier hierarchical power management* for tiled many-core architectures, in which we first manage power across tiles and then within a tile.
- For a collection of multi-program and multi-threaded workloads, we report that our two-tier hierarchical power manager outperforms a centralized power manager by 3% on average, and up to 20% for a 256-core setup.
- We make the observation that thread scheduling is essential in a tiled many-core architecture to account for thread sensitivity towards shared resources. We classify threads based on their sensitivity to both cache and frequency adaptation, and we propose *DVFS and Cache-Aware Thread Migration (DCTM)* to optimize per-tile co-scheduling of compatible threads.
- We provide a comprehensive evaluation of *DCTM* on a tiled many-core processor. We use multi-program workloads consisting of both single-threaded and multi-threaded applications, and we report that *DCTM* improves system performance by 10% on average, and up to 20%. *DCTM* outperforms existing solutions by 4.2% on average (and up to 12%).

2. Motivation

2.1. Limitations of a centralized approach

In the context of power management in many-core processors, prior works [38,11,36] have relied on a central entity (micro-controller) to manage power using one or more micro-architectural techniques to trade off performance at high to moderate power budgets. At stringent power budgets, neither of power management schemes like DVFS nor core adaptation nor cache resizing *in isolation* can provide a viable solution. As a result, prior work [33,36] had to resort to core gating at stringent power envelops. Previously proposed state-of-the-art frameworks [38,30,43] provide an integrated framework for multi/many-core architectures by combining and coordinating core adaptation, cache resizing and/or per-core DVFS to maximize system performance across a wide range of power budgets. These frameworks provide some form of global power management that operates on the runtime statistics of each core to decide on an optimal per-core working configuration. During each time slice, a per-core *Performance Monitoring Unit* (PMU) tracks activity statistics using hardware counters, and predicts/projects the performance and power of all possible configurations. Each core's PMU sends a list of optimal configurations to the *Global Power Manager* (GPM), which globally optimizes the many-core configuration within the given power budget. The GPM instructs each core to reconfigure itself based on the global optimization.

In commercial designs, both the per-core PMU and global GPM are already present in some form [45]. The PMU typically collects power consumption and junction temperatures, and performs control functions such as P-state (DVFS) and C-state (various levels of power gating) transitions. The GPM is implemented as an integrated micro-controller and runs firmware algorithms that interface with the PMUs and on-chip voltage regulators. The PMU keeps track of a core's activity and controls the micro-architectural configuration in response to requests made by the GPM; the GPM combines information from all cores and performs the global power/performance optimization, see *Centralized Approach* in Fig. 1. But as core count continues to grow, the centralized approach becomes inviable: Deng et al. [11] report *quadratic* computational complexity, while Li and Martinez [36] suggest the computational complexity to be *logarithmic* to core count. In future many-core processors [6], a centralized GPM – even with logarithmic complexity – would be a severe bottleneck.

Because a centralized power manager does not scale favorably towards large many-core processors and fine-grain hardware adaptations, we propose *two-tier hierarchical* power management (see Section 3)—*first contribution in this work*.

2.2. Cache-aware thread migration (Cruise)

When threads are co-scheduled on a multi-core processor with a shared last-level cache (LLC), conflicting thread behavior can lead to suboptimal performance. For instance, when a thread whose working set fits in the shared cache is co-scheduled with a streaming application, the quick succession of cache misses from the streaming application may push the working set of the first application out of the shared cache, thereby significantly degrading its performance. Jaleel et al. [27] propose Cruise: a hardware/software co-designed scheduling methodology that uses knowledge of the underlying LLC replacement policy and application cache utility information to determine how best to co-schedule applications in multi-core systems with a shared LLC.

Cruise monitors the number of LLC accesses per kilo instructions (APKI) and miss rate (MR) for each application. Application classification based on these metrics along with co-scheduling rules then optimize overall system performance. The applications are classified in the following categories:

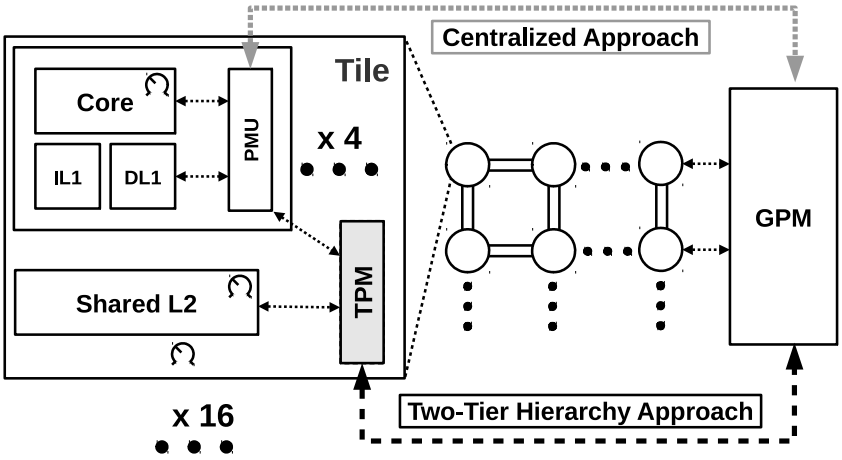


Fig. 1. Generic tiled many-core architecture with *Centralized* (top) versus *Hierarchical* (bottom) power management.

- Core Cache Fitting (CCF): CCF applications fit in the smaller levels of the cache hierarchy and hence the LLC size has little impact on performance.
- LLC Trashing (LLCT): LLCT applications are mostly streaming applications with large working sets—larger than the available LLC size. The LLCT applications degrade performance of any application that benefits from the shared LLC.
- LLC Friendly (LLCFR): LLCFR applications are sensitive to the shared LLC size. They benefit from additional LLC capacity, but performance degrades when co-executed with LLCT applications.

The co-scheduling rules in Cruise are as follows[1]:

1. Group LLCT applications onto the same tile/LLC.
2. Spread CCF applications across all tiles/LLCs.
3. Co-schedule LLCFR with CCF applications.

The performance of LLCFR/LLCF applications degrades significantly when they do not receive the bulk of the shared LLC, hence Cruise schedules LLCFR applications with CCF applications whenever possible.

Cruise assumes that all cores run at the same clock frequency. In other words, it does not take DVFS sensitivity into account. This is a limitation as LLCT and (especially) LLCFR applications, being mixed compute- and memory-bound, may be quite sensitive to frequency. We overcome this limitation by proposing DCTM (see Section 4)—*second contribution in this work.*

3. Two-tier hierarchical power management

The *Centralized* approach as described in Section 2.1 is inappropriate for large-scale many-core processors, for two reasons. First, it assumes per-core DVFS adaptation which is infeasible for many-core processors as it requires on-chip voltage regulators for all cores, which would incur fairly high chip area overhead [8,31,48]. Second, the runtime complexity and overhead of a *Centralized* approach increases considerably with core count.

To address these two limitations, we group cores per tile and add an intermediate layer for power management, the *Tile Power Manager* (*TPM*); see *Two-Tier Hierarchy Approach* in Fig. 1.

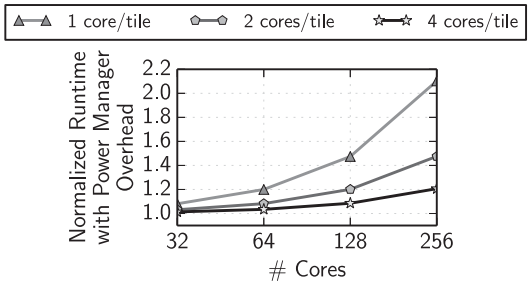


Fig. 2. Normalized runtime overhead (as $1.y\times$ increase over ideal) for *Centralized* (1 core/tile) and *Hierarchical* Power Management with varying tile size (2–4 core/tile) at 1 ms time slice.

Chip power is managed via a hierarchical power manager with a GPM steering the per-tile TPMs.[2] This organization reduces the runtime overhead of the power manager dramatically. To quantify the power manager's runtime overhead, we set up the following simulation experiment. We consider an average multi-program workload on a many-core processor with varying core count (we run workload *WL0*, see Section 5 for more experimental details). The power manager is invoked every 1 ms. Fig. 2 quantifies the worst case theoretical run-time overhead of both *Centralized* and *Hierarchical* power managers normalized to its idealized power management with *zero* run-time overhead. The curve/line *1 core/tile* refers to *Centralized* Power Management, whereas data pertaining to *2–4 core/tile* points refers to the normalized runtime overhead of the *Hierarchical* Power Management at different granularities. We observe that the overhead increases substantially with core count. However, when considering a tiled architecture and a two-tier hierarchical power manager, we are able to significantly reduce the runtime overhead of the power manager. In other words, by keeping the GPM relatively simple and passing more functionality to the TPMs, we avoid GPM to be a bottleneck at high core count. Moreover, as all TPMs can work in parallel, the complexity of the two-tier approach equals $O(G) + O(T_c \log T_c)$, with T_c denoting the number of physical cores per tile, and G the complexity of the GPM (constant in our case). One could adopt an even deeper hierarchy, which would be beneficial in a design with more arbitration levels (intermediate nodes acting as arbitrators for a group of tiles).

[1] In addition to the above mentioned categories, the authors also identify LLC fitting (LLCF) applications by monitoring the miss rate of the application with half the capacity of LLC. In general, these applications exhibit cache characteristics that are similar to LLCFR. In our implementation, we classify LLCF as LLCFR to limit additional hardware overhead especially pertaining to the smallest shared LLC size (see Table 1).

[2] Note that a micro-controller (MCU) based implementation is assumed for the TPM and GPM in both the *Hierarchical* and *Centralized* power managers—this is in-line with previously proposed implementations [45,38,30,43].

4. DVFS and Cache-aware Thread Migration (DCTM)

A tiled many-core processor architecture with hierarchical power management, as we just established in the previous section, poses a new challenge as threads running on the same tile share the L2 cache (LLC) and a common clock frequency. In other words, and in contrast to Cruise, threads running on the same tile not only share the LLC but also share a common clock frequency. Therefore, it is important to take both cache size sensitivity and frequency sensitivity into account when mapping threads to tiles, i.e., the thread migration layer needs to be aware of the sensitivity to both DVFS and LLC size.

4.1. DVFS and LLC sensitivity analysis

To understand an application's sensitivity to clock frequency and LLC size, we set up the following off-line analysis. We run simulations with 55 SPEC CPU2006 application traces for 750 million instructions to observe the performance sensitivity with respect to both LLC and frequency. Fig. 3 plots application performance sensitivity to frequency changes, expressed as the ratio between its performance reduction and the reduction in frequency that was applied. Applications are clustered by their LLC-aware classification type (following Cruise), and plotted in ascending order of sensitivity within each cluster based on Eq. (1):

$$sensitivity_{freq} = \frac{(MIPS_{freq_A}/MIPS_{freq_B})}{(freq_A/freq_B)}. \quad (1)$$

Intuitively, memory-bound applications (LLCT) should have low sensitivity to a change in frequency, while workloads that are completely core-cache fitting (CCF) would see a linear degradation as they are compute-bound. We observe that LLCT applications can still be affected by frequency variations (see the extreme end of LLCT region). The performance of these applications could be significantly affected at stringent power budgets.

We categorize applications into the following DVFS-aware classes, according to their performance sensitivity to DVFS based on Eq. (1):

- High sensitivity (HS, >66%): These applications are highly sensitive to DVFS. The performance of these applications is severely affected when migrated to a tile running at low frequency, whereas performance improves significantly if they can be migrated to a higher-frequency tile. These applications are generally compute-bound.
- Moderate sensitivity (MS, 35%–66%): These applications are moderately affected by DVFS. Applications with a mix of compute-bound and memory-bound operations are grouped in this category.
- Low sensitivity (LS, <35%): These applications degrade slightly when running at a low DVFS setting. It is therefore beneficial to reduce frequency as much as possible to save power. These applications are typically memory-bound.

When co-scheduling applications, the application categorization based on LLC usage (see Cruise, Section 2.2) needs to work in tandem with the DVFS sensitivity categorization as just described. Hence, combining the LLC and DVFS classifications, we have 3 × 3 categories of applications. Not all combinations occur in practice though, as there is some correlation between LLC and DVFS behavior; for instance, CCF applications are almost always compute-bound and hence have high DVFS sensitivity (HS). Fig. 3 identifies five categories: LLCT with LS and MS, LLCFR with MS and HS, and CCF with HS.

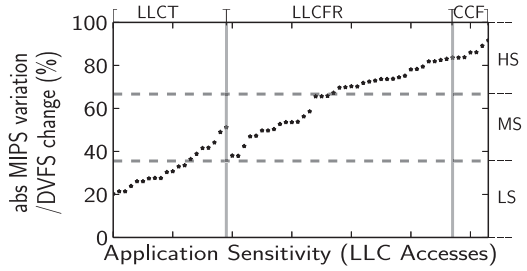


Fig. 3. Application classification based on LLC and DVFS sensitivity.

4.2. DCTM scheduling rules

DVFS and Cache-aware Thread Migration (DCTM) leverages these classifications to steer scheduling of threads to tiles. The power manager will then assign the appropriate adaptation per tile (for frequency and LLC size) and per core (for core configuration). Intuitively speaking, DCTM maps threads with the same classification onto the same tile. Tiles with only LS threads will naturally be configured to run at low frequency (saving power without sacrificing performance much), while tiles with only HS threads preferably use a larger fraction of the total power budget to run at a higher frequency and boost overall system performance. In contrast, mixing LS, MS and HS threads on a single tile leads to a suboptimal situation: either the tile is set to run at low frequency, penalizing performance for the HS threads; or it runs at high frequency which accommodates the HS threads, but wastes power as it does not improve performance of the LS threads. Combining this intuition with the cache-aware scheduling, we create the following scheduling rules for DCTM:

1. Co-schedule LLCT-LS applications on the same tile.
2. Co-schedule LLCT-MS applications on the same tile.
3. Co-schedule CCF-HS applications on tiles with LLCT-MS applications to account for performance impact due to shared LLC contention.
4. Co-schedule the remaining LLCFR-MS and LLCFR-HS applications on the remaining tiles. If possible, co-schedule LLCFR-HS applications on to tiles with CCF-HS applications/threads such that LLCFR-HS application can also utilize high V/F setting and also avoid performance degradation due to shared LLC contention.

The intuition behind co-scheduling all the LLCT-LS applications together onto a tile is that with relatively little allocated power, the co-running applications would incur minimal performance loss. Since the behavioral characteristics of all LLCT-LS applications are similar, the resource requirement would also be similar. The same intuition can be applied to LLCT-MS applications as well; being more sensitive to DVFS, these applications would have better performance than LLCT-LS applications and hence the GPM would allocate a larger fraction of the total power budget to these tiles compared to the LLCT-LS tiles. The applications in the LLCFR-MS and LLCFR-HS categories are co-scheduled or combined with CCF-HS to avoid the performance impact due to the shared LLC. Since the applications in these three categories have moderate to high performance along with much higher sensitivity to DVFS change than LLCT-LS and LLCT-MS applications, the GPM will allocate a relatively larger fraction of the power budget to these tiles, thereby limiting the performance degradation.

4.3. Putting it all together

The DCTM on top of two-tier hierarchical power manager runs at two time scales. The coarse-grain timescale, at 20 ms in our setup, groups threads to tiles using the DCTM scheduling rules as

just described in the previous section. One solution to classifying workloads in terms of LLC and DVFS sensitivity may be to employ sampling, i.e., by running a workload's performance at different frequency settings and different LLC sizes for short durations of time. The limitation is that it incurs significant overhead as we would need to monitor performance for various combinations of LLC size and frequency setting. Instead, we leverage the simple, yet effective analytical performance models proposed in [30] to estimate the performance impact of clock frequency (predicting performance at the target frequency based on a run at the current frequency) and LLC size (ATDs [44] to project the APKI and miss rate for different LLC sizes) on overall performance. Note that no additional computations are required as the projected values generated by the performance prediction models are reused by DCTM.

The fine-grain timescale, at 1 ms in our setup, distributes power across tiles: the GPM distributes power across all tiles, and within each tile, the TPM regulates the hardware adaptations as per the allocated power. Our processor architecture allows three adaptations: core adaptation, LLC resizing, and per-tile DVFS, as we will describe in more detail in Section 5.2. The first fine-grained time slice (1 ms) assumes no power capping, and runs each thread at the maximum configuration (largest core configuration, largest LLC size, highest frequency). We compute the performance of each tile as a ratio of total system performance, i.e., per-tile performance (measured in Million Instructions Per Second or MIPS) divided by chip-wide MIPS. The GPM distributes the total available power budget across all tiles for the next time slice per the MIPS ratios of the tiles in the previous slice, i.e., a high-performance tile is given a larger fraction of the available power budget. The intuition is that compute-intensive tiles need a larger fraction of the total power, boosting overall system performance. Once total power is distributed across the tiles, the TPMs then decide on the optimal configuration for the core, LLC and DVFS setting in each tile. TPM steers adaptation using the performance/power models proposed in [30], with the goal of optimizing performance within the available power budget. Note that, the adaptation and monitoring can be achieved using other frameworks as well with modifications.

4.4. Quantifying DVFS sensitivity: DCTM vs. Cruise

To illustrate the importance of being DVFS aware, we now compare the performance of *DCTM* against *Cruise* for one particular workload consisting of four LLCT SPEC CPU2006 benchmarks running on a tiled architecture with two cores per tile. (For *Cruise*, we replace the *DCTM* scheduling rules by *Cruise*'s at the coarse-grain timescale, while considering the same two-tier power manager at the fine-grain timescale.) Fig. 4 illustrates how *DCTM* obtains higher overall performance compared to *Cruise*. The applications are arranged in a random fashion at the start of the execution. The top graphs show per-thread performance (billion instructions per second—BIPS) for both *DCTM* and *Cruise*, while the bottom graphs show the power and frequency settings of both tiles. All graphs have time on the horizontal axis, and run over the course of 100 ms.

To *Cruise*, all four threads belong to the same category, hence no thread migrations are needed. Taking DVFS sensitivity into account as we do in *DCTM*, however, we find that threads th_0 and th_2 have low sensitivity (LLCT-LS) while th_1 and th_3 have medium sensitivity (LLCT-MS). *DCTM* will therefore swap threads th_1 and th_2 to co-schedule threads with LLCT-MS behavior together (Rule #2 in Section 4.2). After migration, Tile-0 will run both LLCT-MS threads while Tile-1 runs both LLCT-LS threads. Hence, the power budget for Tile-0 can be increased which, due to running threads with high DVFS sensitivity, translates into a significant performance boost. At the same time, the power and frequency of Tile-1 can be reduced at limited performance cost, given that it runs both of the LS threads.

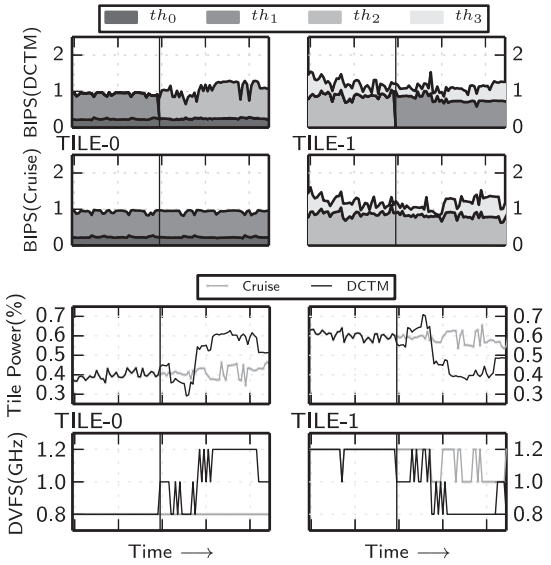


Fig. 4. *DCTM* and *Cruise* through time for 4 cores with LLCT applications.

The end result is an improvement in total system performance by 2.5% while staying within the same power budget.

5. Experimental setup

5.1. Simulation framework

Performance simulator: We use the Sniper user-level multi-core simulator [10], version 6.0. The Sniper simulator performs timing simulations for both multi-program workloads and multi-threaded, shared-memory applications. We use Instruction Window (IW) centric core models for detailed OOO execution accuracy. We use the most detailed cycle-level hardware-validated core models available in Sniper [9,10]. We add support for dynamically changing core and cache parameters. The core adaptation and DVFS transitions combined take 2 μs during which no computations can be performed—a conservative approach. When reducing the number of cache ways, dirty lines are written back through the simulated memory subsystem, consuming NoC and DRAM bandwidth (observed to account for no more than 5% of total DRAM bandwidth). We assume that thread migration takes 1000 cycles to transfer register state and restart execution at a remote core. In addition, the potential cold misses that transfer the thread's working set to the local caches are also included in the execution/runtime of the simulation. To reduce the variations due to thread migration, we execute 3 copies of each workload and report the average.

Power consumption. McPAT version 1.0 is used to estimate static and dynamic power consumption [34,35] for a 22 nm technology. Power savings incurred by reconfiguration are modeled by running McPAT with the modified target parameters (Table 1). Running McPAT along with the performance simulation allows us to emulate the behavior of hardware energy counters at simulated time slices of 1 ms. Note that, changing the V/F setting while keeping the other micro-architecture knobs unchanged, we observe that the array layout/size of SRAM and CAM structures does not change in McPAT.

5.2. Adaptive micro-architecture

To keep all the cores active even at stringent power budgets, we incorporate core micro-architectural adaptation, LLC adaptation and DVFS adaptation simultaneously, thereby providing various operational points in our adaptive tiled many-core processor.

Table 1
Micro-architectural adaptations.

Parameter	Values			
Core adaptations				
ROB size	16	32	64	128
Reservation station entries	4	8	16	32
Load queue entries	6	12	24	48
Store queue entries	4	8	16	32
DVFS adaptations per-tile				
Frequency (GHz)	0.8	1.0	1.2	–
Voltage (V)	0.7	0.75	0.8	–
Shared LLC adaptations per-tile				
Cache ways	4	8	12	16
Capacity (KB)	512	1024	1536	2048

Table 2
Tile-based many-core architecture.

Component	Parameters
Core configuration	
Core type	4-way issue OOO, 128-entry ROB
Load/store queue	48 load entries, 32 store entries
L1-I cache	32 KB, 4-way, 3 cycle access time
L1-D cache	32 KB, 4-way, 3 cycle access time
Tile configuration	
Tile size	4 cores
Core count	64, 128, 256
Tile count	16, 32, 64
L2 cache (per-tile)	2 MB, 16-way, 10 cycle access time
L2 prefetcher	Stride-based, 8 independent streams
Coherence protocol	Directory-based MESI, distributed tags
Network on chip	Mesh 16×1, 16×2, 16×4 32 GB/s/link
Main memory	8, 16, 32 controllers 80 ns latency, 128 GB/s total
Chip wide configuration	
Frequency-Vdd	1.2 GHz @ 0.8 V
Technology	22 nm
TDP	100, 190, 350 W

As described before, we use the notion of a Globally Asynchronous Locally Synchronous (GALS) design [26], in which each tile maintains its own voltage–frequency domain. The adaptive core/tile configuration is expressed as a tuple $[core, f_t, llc_t]$, denoting that the core is configured as *core*, running at frequency f_t and llc_t cache ways enabled for the given tile t (see also Table 1).

Core. Core adaptation pertains to reconfiguring the core micro-architecture. The core width can be adapted, along with the size of various structures (see 'Core adaptation' in Table 1). We maintain a quadratic relation between execution width and size of micro-architectural buffers [18]. Unused components are power-gated to reduce both static and dynamic power consumption, providing for an interesting opportunity for power savings for memory-bound or otherwise low-ILP applications. In our tiled architecture, we assume each core's micro-architecture can be adapted individually.

DVFS. DVFS adaptation is a widely used technique for enforcing power budgeting. In the proposed architecture, we assume the availability of on-die voltage regulators [8,32] per-tile to enable DVFS from 0.8 GHz at 0.7 V to 1.2 GHz at 0.8 V (see also Table 1), which is in line with the Intel Xeon Phi [29]. In the tiled architecture, the TPM needs to enforce a DVFS setting per-tile (affecting both cores and shared LLC). Choosing an appropriate DVFS setting per-tile is non-trivial as a single setting for all threads scheduled on the given tile might not be optimal for performance. Applications with higher sensitivity to DVFS changes are more likely to be affected by imposing a single DVFS setting per tile. Hence, we choose the DVFS setting so as to minimize the severity of the performance impact on the applications with high sensitivity to DVFS. If this setting over-provisions the per-tile power allocated by the GPM, we subsequently down-scale the core micro-architecture until the allocated tile budget is reached.

Shared LLC. For cache adaptation, we use a flushing selective-way LLC implementation [1], i.e., a shared LLC per-tile in our setup. By controlling which ways are active, we can power-gate portions of the cache to reduce its capacity and static power. We use selective ways (see also Table 1) because of its simple design—selective sets on the other hand require changes to the number of tag bits used [53]. By using the flushing cache policy when shrinking to a smaller number of ways (writing back dirty cache lines), we can turn off the corresponding cache ways sooner, reducing static power consumption of the cache. To estimate the effect of cache capacity changes, we use auxiliary tag directories (ATDs) [44] to estimate the miss rates (32 randomly selected sample sets) for different shared cache configurations. To project the performance impact of threads *sharing* the LLC, we create ATDs per core and annotate cache tags with a core identifier. This is only required for those sets that are part of the ATD's sample set.

5.3. Workloads

Multi-program workloads. We run a number of multi-program workloads composed of SPEC CPU2006 benchmarks; 29 programs in total, which along with all reference inputs leads to 55 benchmarks. We select representative simulation points of 750 million instructions each using PinPoints [42]. Four multi-program workloads with 64 benchmarks each are constructed by combining these 55 benchmarks as indicated in Table 3(a). The benchmarks in each workload are arranged in a random fashion unless mentioned otherwise. We replicate each workload by $2\times$ and $4\times$ for the 128-core and 256-core setups, respectively. Each benchmark is pinned to a core unless mentioned otherwise. We run the simulation for 200 ms to keep total simulation time within feasible limits. When a benchmark completes before this time, it is restarted on the same core. We quantify weighted speedup [47] or system throughput (STP) [17] which quantifies the aggregate throughput achieved by all cores in the system.

Multi-program multi-threaded workloads. We create workloads by combining multiple multi-threaded applications from the SPEC OMPM2001 [2] and NPB benchmark suites [4], see Table 3(b). For meaningful analysis, we use the *reference* input set for SPEC OMPM2001, and the *class A* input set for NPB. We construct two workloads, each running 64 threads in total: *NPB1* consists of four different NAS applications running concurrently with 16 threads each, while *OMPM* combines eight SPEC OMPM applications running 8 threads each. When running on the 128-core setup we replicate these workloads by $2\times$, and by $4\times$ for the 256-core setup. Execution of all multi-threaded applications in a workload begins after the last application has reached the region of interest (ROI). Again, we run each workload for 200 ms to keep total simulation time manageable.

6. Evaluation

We now evaluate *DCTM* on our power-constrained tiled many-core architecture. Unless mentioned otherwise, results are obtained using fine-grained hardware adaptation at 1 ms intervals, while thread migration is performed at 20 ms intervals. Each experiment fixes the available power budget to a fraction of the chip's nominal power consumption (see TDP in Table 2). We quantify performance in terms of system throughput (STP), which includes power management overhead.

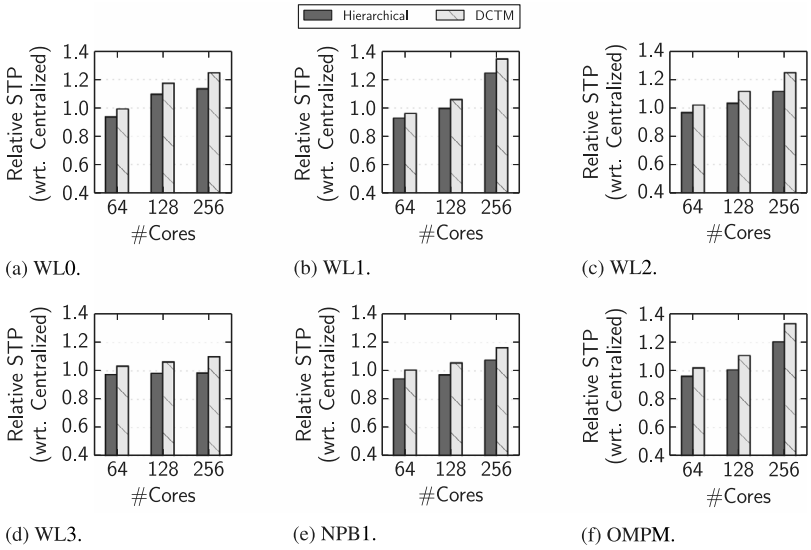


Fig. 5. STP (normalized to *Centralized*) for *Hierarchical* and *DCTM* at 60% power budget.

Table 3
Workloads.

(a) Multi-program workloads (SPEC CPU2006)

Workload	Description	Benchmarks
WL0	SPEC average	All 55 + 9 uniform random
WL1	Compute	8 compute bound, ×8
WL2	Mixed	8 compute + 8 memory, ×4
WL3	Memory	8 memory bound, ×8

(b) Multi-program multi-threaded workloads

Workload	Benchmarks	Input set	Threads
NAS parallel benchmark suite			
NPB1	BT, CG, FT, MG	Class A	16 each
SPEC OMPM 2001 suite			
OMPM	fma3d, swim, mgrid, applu, equake, apsi, gafort, wupwise	Reference	8 each

The evaluation is done in a number of steps. We first evaluate the scalability of two-tier hierarchical power management (*Hierarchical*) compared to the Chrysso-based *Centralized* power manager [30]. We next compare *DCTM* against Cruise [27], demonstrating the importance of being frequency-aware. We then evaluate the importance of dynamic thread migration, followed by a number of sensitivity analyses with respect to the thread migration interval and power distribution.

6.1. Hierarchical vs. centralized power management

We first evaluate the scalability of two-tier hierarchical power management versus a centralized approach. We consider the following power management policies: (i) *Centralized* which assumes centralized power management along with per-core DVFS[3]; (ii) *Hierarchical* which is our two-tier hierarchical power manager, with random mapping of threads to tiles, and per-tile DVFS; and (iii) *DCTM* which is our two-tier hierarchical power manager that migrates threads across tiles in a DVFS and LLC aware manner.

Fig. 5 quantifies relative STP (normalized to the *Centralized* approach) for the various workloads as a function of core count at a 60% power budget. The *Centralized* approach is quite effective at 64 cores. The overhead of the centralized power manager is limited, and the ability to exploit per-core DVFS yields a performance benefit over the two-tier *Hierarchical* approach with per-tile DVFS, by 7% on average. At larger core counts however, the overhead of the centralized power manager is not offset by the benefit from per-core DVFS, yielding a performance benefit for two-tier hierarchical power management, up to 24% for 256 cores (see *WL*1). The interesting insight here is that at large core counts, per-tile DVFS is in fact beneficial over per-core DVFS, which may seem counter-intuitive at first sight because there is less opportunity for fine-grain adaptation. The reason however is that per-tile DVFS facilitates a two-tier hierarchical power manager which incurs less overhead compared to a centralized power manager for a per-core DVFS architecture.

The results in Fig. 5 also show that being able to migrate threads such that compatible threads co-execute per tile, as done using *DCTM*, yields a substantial performance benefit over random thread assignment with *Hierarchical*, see for example *WL*1: 32.4% for *DCTM* versus 24% for *Hierarchical*. We observe the performance benefit to be consistent across all workloads.

Overall, we find two-tier hierarchical power management, and *DCTM* to be beneficial across all workloads. The performance benefit seems to be proportional to the number of compute-intensive benchmarks in the workload, see for example *WL*1 (compute-intensive) versus *WL*2 (mixed) versus *WL*3 (memory-intensive). The reason is that the power manager groups threads based on their sensitivity to LLC size and clock frequency, and allocates a larger fraction of the available power budget to tiles that benefit the most, which are typically the ones running compute-intensive benchmarks.

6.2. Two-tier approach: performance vs. power budget

As we mentioned in Section 4.4, application's sensitivity to DVFS could provide better performance than just considering LLC sensitivity. To illustrate this, Fig. 6 shows the STP improvement (as percentage) of a 256-core setup at different power budgets for *Cruise* and *DCTM*, relative to the *Hierarchical* performance.[4] Both *Cruise* and *DCTM* employ a two-tier hierarchical power manager.

[3] The *Centralized* power manager has logarithmic complexity, and effectively represents Chrysso proposed in [30]. Note that the *Centralized* power manager uses a single GPM to distribute power across the chip.

[4] Results for 64 and 128 cores are not included due to space constraints.

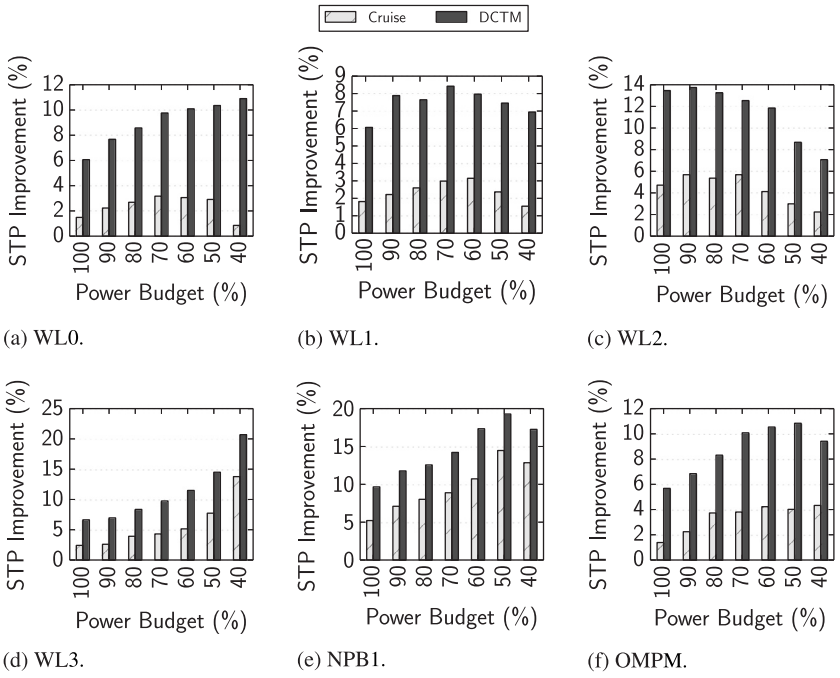


Fig. 6. STP improvement (percentage) for *DCTM* and *Cruise* over *Hierarchical* for the 256-core setup.

Fig. 6 shows the STP improvement (as a percentage) for the 256-core setup at different power budgets for *Cruise* and *DCTM*, relative to *Hierarchical*. The bottomline is that *DCTM* outperforms *Hierarchical* by 10% on average (across all workloads) and by up to 20%. *DCTM* outperforms DVFS-agnostic *Cruise* by 4.2% on average and by up to 12%.

There are a couple interesting trends to be observed for a number of individual workloads. For *WL*0 (average SPEC CPU), *DCTM* shows an increasing trend at increasingly smaller power budgets. The reason is that *WL*0 includes a wide range of applications with varying characteristics, which can be efficiently exploited using both DVFS and LLC sensitivities. For *WL*1 (compute-intensive SPEC CPU), *DCTM* yields a consistent improvement over *Cruise*, but is limited by the available power budget. For *WL*3 (memory-intensive SPEC CPU), we observe that both *DCTM* and *Cruise* are able to prevent excessive LLC trashing, which leads the STP improvement over *Centralized* to increase at smaller power budgets. However, by being DVFS-aware, *DCTM* still outperforms *Cruise* by 7% on average.

6.3. Static assignment vs. dynamic migration

An alternative to performing on-line thread migration could be to statically select a thread schedule a priori based on known average application characteristics. However, in addition to the potential problem of jobs periodically entering and leaving the system, a single application exhibits phase behavior that may cause its classification to change over time. Using an average class leads to suboptimal scheduling, showing that on-line migration is a necessary component of our approach.

To illustrate that static classification is not sufficient, we consider the example of the milc benchmark. Fig. 7 plots MIPS over time for milc at 80% power budget in a 64-core setup. The average behavior of milc can be classified as LLCT with moderate sensitivity to DVFS. Static assignment co-schedules this benchmark with other LLCT-MS or CCF applications. Although the average classification is LLCT (streaming behavior), milc shows phases during its execution where it is classified as LLCFR due to a reduced working set which does fit in the LLC. During this phase,

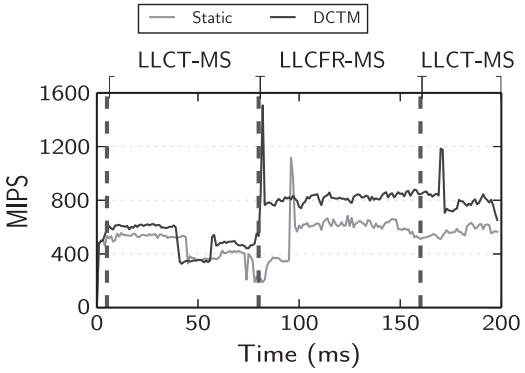
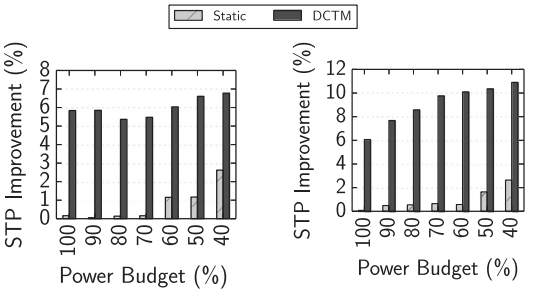


Fig. 7. *Static* and *DCTM* over time for *milc* on a 64-core setup at 80% power budget.

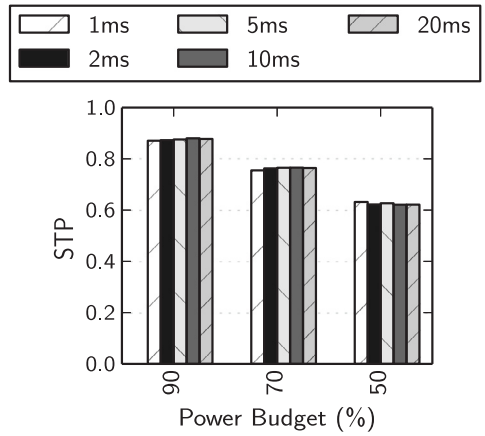
if the application remains co-scheduled with an LLCT application which will cause milc's working set to be evicted, performance will suffer compared to a situation where the LLCT thread is migrated away in favor of other LLCFR or CCF applications. Unlike the static assignment, DCTM is able to observe changes in cache access behavior at runtime and re-schedule accordingly, leading to higher performance for milc during its LLCFR phase.

We now evaluate the performance benefit of *DCTM* against *Static* classification in a more systematic way. *Static* classification follows the same classification and scheduling rules as *DCTM*; the only difference is that *Static* classification does so based on the application's average execution behavior. Fig. 8 shows the performance improvement of static assignment and *DCTM* over *Hierarchical*. At moderate to low power budgets, static assignment provides some improvement over random assignment as the restricted power budget requires significant reductions in both DVFS and LLC size, which can be tolerated better when compatible applications are co-scheduled. At higher power budgets, however, the architecture operates much closer to its full configuration, and static assignment fails to provide a significant advantage. In contrast, *DCTM* is able to exploit phase behavior in the applications, and can obtain the optimum co-schedule at each point in time. This gives *DCTM* a significant margin over both *Hierarchical* and

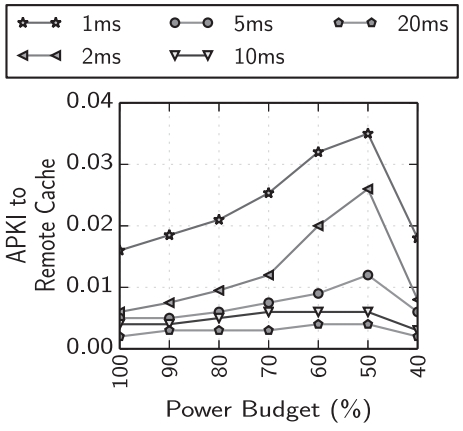


(a) 64 cores. (b) 256 cores.

Fig. 8. *Static* versus *DCTM* relative to *Hierarchical* for *WL0*.



(a) STP vs. power budget.



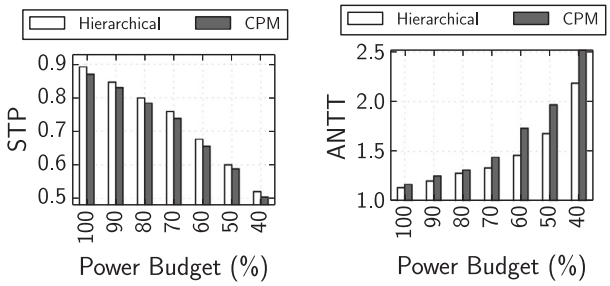
(b) Remote cache accesses.

Fig. 9. Sensitivity to DCTM's thread migration interval for *WL0* on 64 cores and L2 invalidates: *Hierarchical* vs. *DCTM* for single-thread *WL0* workload.

Static, showing that runtime migration can greatly improve power efficiency of large many-core systems.

6.4. Sensitivity to migration interval

Previous experiments considered a coarse-grain thread migration interval of 20 ms, while hardware adaptation was performed every millisecond. This is consistent with an implementation where adaptations are performed in a hardware power manager, while thread migrations is done by the operating system (with input from the power manager and/or performance counters to do the classification).



(a) STP. (b) ANTT.

Fig. 10. Fine-grain (*Hierarchical*) versus coarse-grain (*CPM*) power redistribution for the two-tier hierarchical power manager for *WL0* on 64-core setup.

In Fig. 9, we vary the *DCTM* thread migration interval between 1 and 20 ms, while leaving the power-aware hardware adaptation interval fixed at 1 ms. As discussed in Section 5.1, thread migration takes 1000 cycles to transfer register state and restart execution at a remote core, in addition to potential cold misses that transfer the thread's working set to the local caches (using the standard coherency protocol, which our simulations model in detail). Fig. 9(a) plots STP (relative to full configuration), when running the SPEC average multi-program workload *WL0* on a 64-core system at various power settings. No significant difference in performance is observed, showing that a 20 ms migration interval is sufficient. It is therefore possible to implement this layer in the operating system: hardware thread migration is not required and it is even possible to spend a significant amount of time and effort to compute the best schedule.

In contrast, doing thread migration too frequently can in fact be harmful, as Fig. 9(b) illustrates. For short migration intervals (below 5 ms), the amount of cache-to-cache transfers increases significantly, showing that working sets frequently have to be transferred across the chip, trailing the migrating threads. These cache-to-cache transfers cause both a reduction in thread performance and consume additional power, which has to be amortized by the improved schedule.

6.5. Fine-grained power redistribution

Our *Two-Tier Hierarchy Approach* reallocates power between the tiles in each hardware adaptation interval (1 ms timescale), in addition to making adaptations local to each tile. An alternative would be to run the global component (GPM) less frequently, while running TPM at a small timescale. Such approach is in fact proposed by Coordinated Power Management (CPM) [39], where a fast per-tile DVFS controller performs local optimization constrained to each tile's power budget, and a global manager redistributes power across the tiles every ten adaptation intervals. We implement a similar design with 1 ms and 10 ms time scales for the local (TPM) and global (GPM) power management, respectively. Fig. 10 shows STP (relative to full configuration) as a function of the available power. Our *Hierarchical* design (GPM and TPM adaptation at 1 ms timescale) outperforms CPM by 1%–8% in STP, in addition to providing better fairness between the running threads (measured in average normalized turnaround time, ANTT [17]). This is because in CPM, each tile needs to manage power over- and undershoots using a fixed power budget over a 10 ms interval. In contrast, when doing global adaptation at 1 ms time scales, power can be redistributed across tiles much faster, allowing compute-bound threads to borrow power from memory-bound threads running on different tiles within just 1 ms, so the system can respond much more quickly to changes in workload behavior.

7. Related work

Micro-architecture adaptation. A variety of prior work has explored techniques to improve power-efficiency by adapting micro-architecture structures on a per core basis. Some proposals adapt the instruction window [3] and the issue logic [20] to provide greater power/energy efficiency while showing a small reduction in application performance. ForwardFlow core [23] is proposed as a way to trade off core performance for power. Albonesi [1] and Yang et al. [53] evaluate shutting down portions of the cache, either a number of ways or a combination of ways and sets for improved energy efficiency or to trade off performance for power and energy. Eckert et al. [15] combine drowsy caches with front-end pipeline gating and demonstrate better performance-power scaling than dynamic frequency scaling, and even DVFS in some cases. Although their work shows that one can reconfigure the system to perform better than DVFS, they do not perform runtime optimizations of large many-cores in power-constrained environments. Finally, Dubach et al. [13] use machine-learning models (trained using profiling) to perform online adaptation of a single core at a time.

Centralized dynamic power management. Several prior works explore centralized dynamic power management. For example, Isci et al. [25] investigate a global power controller to determine different per-core DVFS values to maximize chip-wide MIPS. Teodorescu and Torrellas [49] propose variation-aware power-management DVFS algorithms for application scheduling on a CMP to save power or improve throughput at a given power budget. CoScale [11] deals with co-optimizing DVFS settings for both the CPU and DRAM. Other proposals use machine learning and neural networks to perform global DVFS with per-core adaptation [28] or global resource allocation [5]. Meng et al. [38] propose DVFS adaptation along with cache adaptation for a 4-core system. Chrysso [30] dynamically adjusts the capabilities of an out-of-order core, private cache and per-core DVFS at fine-grained time slice (10 ms) using simple analytical models and a centralized power manager to improve performance under given power budgets. Finally, Flicker [43] dynamically adjusts the capabilities of an out-of-order core at coarse-grained time slice (100 ms) using sampling-based global genetic algorithm to improve performance compared to core gating at moderate power budgets.

Tiled architecture and hierarchical power manager. In RCS [22], the authors propose mechanisms to uniformly change core resources with the number of cores (up to 12) to exploit application variability at a fixed power budget. The proposed scheme uses SVM-based machine-learning mechanisms to obtain the number of active cores (with corresponding micro-architectural variation) for each interval. PEPON [46] uses 10 DVFS adaptations for core and NoC along with selective-way resizing of a single shared LLC to provide feasible working configuration till moderate power budgets. Other proposals [21,39] use the concept of two-level power management schemes, viz. master–slave and global–local, respectively. Mishra et al. [39] use one of the 10 DVFS values per island (2/4 cores per island) under the given power budget. The mechanism uses a 2-level power manager—GPM-LPIC (digital controller per LPIC) called every 25/50 ms and 2.5/5 ms, respectively. Prior work has explored power management techniques on network-on-chip [50] to provide significant reduction in power dissipation of NoCs. A hierarchical control-theory based power manager [41] employs multiple PID controllers (one for each cluster and one for each application) in a synergistic fashion and manages to achieve optimal power-performance efficiency while respecting the TDP budget. This approach has poor scalability with increasing number of clusters and price-theory based demand–supply approach. Additionally, the coarse-grain power management could ensue thermal-throttling due to instantaneous power over-shoots.

To the best of our knowledge, none of the above works have evaluated three-way micro-architectural adaptation along with a thread migration layer for optimal shared resource utilization using a hierarchical power manager on a power-constrained tiled many-core architecture.

8. Conclusion

An integrated and scalable many-core power management is clearly needed as we move towards increasingly tighter power budgets. In this work, we leverage a two-tier hierarchical power manager due to its low overhead and high scalability on a tiled many-core architecture with shared LLC and per-tile DVFS at fine-grain time slices. We use (i) analytical performance and power models for the shared architecture and its adaptation, and (ii) we distribute power across tiles using GPM and then within a tile (in parallel across all tiles). We observe that thread scheduling is essential in such an architecture to account for thread sensitivity towards shared resources. We leverage DVFS and cache-aware thread migration (*DCTM*) to ensure optimum per-tile co-scheduling of compatible threads at runtime over the two-tier hierarchical power manager. Based on our evaluations, we show that *DCTM* outperforms Cruise [27] by 4.2% on average (and up to 12%) for both multi-program and multi-threaded workloads. Compared to a centralized power manager, *DCTM* improves performance by 10% on average (and up to 20%) while using 4× less on-chip voltage regulators.

Acknowledgments

This work was supported in part by the European Research Council under the European Community's 7th Framework Programme (FP7/2007–2013)/ERC Grant agreement no. 259295 and the Spanish Ministry of Economy and Competitiveness under grants TIN2010-18368 and TIN2013-44375-R. The experiments were run on computing infrastructure at the ExaScience Lab, Leuven, Belgium.

References

[1] David H. Albonesi, Selective cache ways: On-demand cache resource allocation, in: Proceedings of the 32nd Annual International Symposium on Microarchitecture, MICRO, November 1999, pp. 248–259.
[2] Vishal Aslot, Rudolf Eigenmann, Performance characteristics of the SPEC OMP2001 benchmarks, ACM SIGARCH Comput. Archit. News (2001) 31–40.
[3] R.Iris Bahar, Srilatha Manne, Power and energy reduction via pipeline balancing, in: Proceedings of the 28th Annual International Symposium on Computer Architecture, ISCA, June 2001, pp. 218–229.
[4] David H. Bailey, Eric Barszcz, John T. Barton, David S. Browning, Russell L. Carter, Leonardo Dagum, Rod A. Fatoohi, Paul O. Frederickson, Thomas A. Lasinski, Rob S. Schreiber, H.D. Simon, V. Venkatakrishnan, S.K. Weeratunga, The NAS parallel benchmarks, Int. J. High Perform. Comput. Appl. (1991) 63–73.
[5] Ramazan Bitirgen, Engin Ipek, Jose F. Martinez, Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach, in: Proceedings of the 41st Annual International Symposium on Microarchitecture, MICRO, November 2008, pp. 318–329.
[6] Shekhar Borkar, Thousand core chips: A technology perspective, in: Proceedings of the 44th Annual Design Automation Conference, DAC, 2007, pp. 746–749.
[7] David Brooks, Margaret Martonosi, Dynamic thermal management for high-performance microprocessors, in: Proceedings of the International Symposium on High Performance Computer Architecture, HPCA, January 2001, pp. 171–182.
[8] Edward A. Burton, Gerhard Schrom, Fabrice Paillet, Jonathan Douglas, William J. Lambert, Kaladhar Radhakrishnan, Michael J. Hill, FIVR – Fully integrated voltage regulators on 4th generation Intel® Core™ SoCs, in: Proceedings of the 29th Applied Power Electronics Conference and Exposition, APEC, 2014, pp. 432–439.
[9] Trevor E. Carlson, Wim Heirman, Lieven Eeckhout, Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC, November 2011, pp. 1–12.

[10] Trevor E. Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, Lieven Eeckhout, An evaluation of high-level mechanistic core models, ACM Trans. Archit. Code Optim. (TACO) 11 (3) (2014) 28:1–28:25.

[11] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, Ricardo Bianchini, CoScale: Coordinating CPU and memory system DVFS in server systems, in: Proceedings of the 45th Annual International Symposium on Microarchitecture, MICRO, December 2012, pp. 143–154.

[12] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, A.R. LeBlanc, Design of ion-implanted MOSFET's with very small physical dimensions, IEEE J. Solid-State Circuits (1974) 256–268.

[13] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, Michael F.P. O'Boyle, A predictive model for dynamic microarchitectural adaptivity control, in: Proceedings of the 43rd Annual International Symposium on Microarchitecture, MICRO, December 2010, pp. 485–496.

[14] Thomas Ebi, M. Faruque, Jörg Henkel, TAPE: Thermal-aware agent-based power economy multi/many-core architectures, in: Proceedings of the International Conference on Computer-Aided Design, ICCAD, 2009, pp. 302–309.

[15] Yasuko Eckert, Srilatha Manne, Michael J. Schulte, David A. Wood, Something old and something new: P-states can borrow microarchitecture techniques too, in: Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED, July 2012, pp. 385–390.

[16] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, Doug Burger, Dark silicon and the end of multicore scaling, in: Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA, 2011, pp. 365–376.

[17] Stijn Eyerman, Lieven. Eeckhout, System-level performance metrics for multiprogram workloads, IEEE MICRO (2008) 42–53.

[18] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, James E. Smith, A mechanistic performance model for superscalar out-of-order processors, ACM Trans. Comput. Syst. (TOCS) (2009) 1–37.

[19] Xiaobo Fan, Wolf-Dietrich Weber, Luiz Andre Barroso, Power provisioning for a warehouse-sized computer, in: Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA, June 2007, pp. 13–23.

[20] Daniele Folegnani, Antonio González, Energy-effective issue logic, in: Proceedings of the 28th Annual International Symposium on Computer Architecture, ISCA, June 2001, pp. 230–239.

[21] Yang Ge, Qinru Qiu, Qing Wu, A multi-agent framework for thermal aware task migration in many-core systems, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20 (10) (2012) 1758–1771.

[22] Hamid Reza Ghasemi, Nam Sung Kim, RCS: Runtime resource and core scaling for power-constrained multi-core processors, in: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, PACT, August 2014, pp. 251–262.

[23] Dan Gibson, David A. Wood, Forwardflow: A scalable core for power-constrained CMPs, in: Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA, June 2010, pp. 14–25.

[24] Sebastian Herbert, Diana Marculescu, Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, in: Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED, 2007, pp. 38–43.

[25] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, Margaret Martonosi, An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget, in: Proceedings of the 39th Annual International Symposium on Microarchitecture, MICRO, December 2006, pp. 347–358.

[26] Anoop Iyer, Diana Marculescu, Power and performance evaluation of globally asynchronous locally synchronous processors, in: Proceedings of the 29th Annual International Symposium on Computer Architecture, ISCA, 2002, pp. 158–168.

[27] Aamer Jaleel, Hashem H. Najaf-abadi, Samantika Subramaniam, Simon C. Steely, Joel Emer, Cruise: Cache replacement and utility-aware scheduling, in: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, 2012, pp. 249–260.

[28] Ramkumar Jayaseelan, Tulika Mitra, A hybrid local–global approach for multi-core thermal management, in: Proceedings of the 2009 International Conference on Computer-Aided Design, ICCD, November 2009, pp. 314–320.

[29] James Jeffers, James Reinders, Intel Xeon Phi Coprocessor High Performance Programming. Newnes, 2013.

[30] Sudhanshu S. Jha, Wim Heirman, Ayose Falcón, Trevor E. Carlson, Kenzo Van Craeynest, Jordi Tubella, Antonio González, Lieven Eeckhout, Chrysso: An integrated power manager for constrained many-core processors, in: Proceedings of the ACM International Conference on Computing Frontiers, CF, May 2015, pp. 19:1–19:8.

[31] Harish K. Krishnamurthy, Vaibhav A. Vaidya, Pavan Kumar, George E. Matthew, Sheldon Weng, Bharani Thiruvengadam, Wayne Proefrock, Krishnan Ravichandran, Vivek De, A 500 MHz, 68% efficient, fully on-die digitally controlled buck voltage regulator on 22nm Tri-Gate CMOS, in: Proceedings of the IEEE Symposium on VLSI Circuits Digest of Technical Papers, 2014, pp. 1–2.

[32] Nasser Kurd, Muntaquim Chowdhury, Edward Burton, Thomas P. Thomas, Christopher Mozak, Brent Boswell, Manoj Lal, Anant Deval, Jonathan Douglas, Mahmoud Elassal, Ankireddy Nalamalpu, Timothy M. Wilson, Matthew Merten, Srinivas Chennupaty, Wilfred Gomes, Kumar Rajesh, 5.9 Haswell: A family of IA 22nm processors, in: Proceedings of the International Solid-State Circuits Conference Digest of Technical Papers, ISSCC, pp. 112–113, 2014.

[33] Jacob Leverich, Matteo Monchiero, Vanish Talwar, Partha Ranganathan, Christos Kozyrakis, Power management of datacenter workloads using per-core power gating, Comput. Archit. Lett. (2009) 48–51.

[34] Sheng Li, J. Ahn, Jay B. Brockman, Norman P. Jouppi, McPAT 1.0: An integrated power, area, and timing modeling framework for multicore architectures, in: HP Labs, 2009.

[35] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, Norman P. Jouppi, McPAT: An integrated power, area and timing modeling framework for multicore and manycore architectures, in: Proceedings of the 42nd Annual International Symposium on Microarchitecture, MICRO, December 2009, pp. 469–480.

[36] Jian Li, Jose F. Martinez, Dynamic power-performance adaptation of parallel computation on chip multiprocessors, in: Proceedings of the 12th International Symposium on High-Performance Computer Architecture, HPCA, 2006, pp. 77–87.

[37] Kai Ma, Xue Li, Ming Chen, Xiaorui Wang, Scalable power control for many-core architectures running multi-threaded applications, in: Proceedings of the 38th International Symposium on Computer Architecture, ISCA, 2011, pp. 449–460.

[38] Ke Meng, Russ Joseph, Robert P. Dick, Li Shang, Multi-optimization power management for chip multiprocessors, in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT, 2008, pp. 177–186.

[39] Asit K. Mishra, Shekhar Srikantaiah, Mahmut Kandemir, Chita R. Das, CPM in CMPs: Coordinated power management in chip-multiprocessors, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2010, pp. 1–12.

[40] Gordon E. Moore, Cramming more components onto integrated circuits, Electronics (1965) 116–144.

[41] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, Sanjay Vishin, Hierarchical power management for asymmetric multi-core in dark silicon era, in: Proceedings of the 50th Annual Design Automation Conference, DAC, 2013, pp. 174:1–174:9.

[42] Harish Patil, Robert Cohn, Mark Charney, Rajiv Kapoor, Andrew Sun, Anand Karunanidhi, Pinpointing representative portions of large Intel Itanium; programs with dynamic instrumentation, in: Proceedings of the 37th Annual International Symposium on Microarchitecture, MICRO, December 2004, pp. 81–92.

[43] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, Christine A. Shoemaker, Flicker: A dynamically adaptive architecture for power limited multicore systems, in; Proceedings of the 40th International Symposium on Computer Architecture ISCA, June 2013, pp. 13–23.

[44] Moinuddin K. Qureshi, Yale N. Patt, Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, in: Proceedings of the 39th Annual International Symposium on Microarchitecture, MICRO, 2006, pp. 423–432.

[45] Efraim Rotem, Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, Eliezer Weissmann, Power-management architecture of the Intel microarchitecture code-named Sandy Bridge, IEEE MICRO (2012) 20–27.

[46] Akbar Sharifi, Asit K. Mishra, Shekhar Srikantaiah, Mahmut Kandemir, Chita R. Das, PEPON: Performance-aware hierarchical power budgeting for NoC based multicores, in: Proceedings of the 21st International conference on Parallel Architectures and Compilation Techniques, PACT, 2012, pp. 65–74.

[47] Allan Snavely, Dean M. Tullsen, Symbiotic jobscheduling for simultaneous multithreading processor, in; Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, November 2000, pp. 234–244.

[48] N. Sturcken, M. Petracca, S. Warren, P. Mantovani, L.P. Carloni, A.V. Peterchev, Kenneth L. Shepard, A switched-inductor integrated voltage regulator with nonlinear feedback and network-on-chip load in 45 nm SOI, IEEE J. Solid-State Circuits (2012) 1935–1945.

[49] Radu Teodorescu, Josep Torrellas, Variation-aware application scheduling and power management for chip multiprocessors, in: Proceedings of the 35th International Symposium on Computer Architecture, ISCA, June 2008, pp. 363–374.

[50] Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Priya Iyer, Arvind Singh, Tiju Jacob, et al. An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS, in: Proceedings of the International Solid-State Circuits Conference Digest of Technical Papers, ISSCC, 2007, pp. 98–589.

[51] Yefu Wang, Kai Ma, Xiaorui Wang, Temperature-constrained power control for chip multiprocessors with online model estimation, in: Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA, 2009, pp. 314–324.

[52] Qiang Wu, Philo Juang, Margaret Martonosi, Douglas W. Clark, Formal online methods for voltage/frequency control in multiple clock domain microprocessors, in: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS, 2004, pp. 248–259.

[53] Se-Hyun Yang, Michael D. Powell, Babak Falsafi, T.N. Vijaykumar, Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay, in: Proceedings of the 8th International Symposium on High-Performance Computer Architecture, HPCA, February 2002, pp. 151–161.

Sudhanshu Shekhar Jha received B.E. in computer science engineering from Birla Institute of Technology, Mesra in 2006. Received Masters in embedded systems design from Advanced Learning and Research Institute (ALaRI), affiliated with ETH Zurich, Politecnico di Milano and Universita della Svizzera Italiana in Switzerland in 2011. Joined ARCO in 2011 and current research is focused towards performance optimization on power-constrained many-core architecture.

Wim Heirman obtained his M.Sc. and Ph.D. in computer architecture from Ghent University in 2003 and 2008, respectively. He currently works as a research scientist for Intel Corporation. His interests include many-core processor architecture and performance modeling.

Ayose Falcón received his B.S. (1998) and M.S. (2000) degrees in computer science from the University of Las Palmas de Gran Canaria, Spain. In 2005, he received a Ph.D. in computer science from the Universitat Politècnica de Catalunya (UPC). His Ph.D. research included fetch unit optimization – especially branch prediction and instruction cache prefetching – for superscalar and simultaneous multithreading processors. During his Ph.D. years, Ayose was a Summer Intern and then a research consultant at Intel Microprocessor Research Labs in Portland (OR, USA), and worked as a teaching assistant at UPC for one year. From 2004 to 2009, Ayose was a senior research scientist at the Exascale Computing Lab, HP Labs. His research interests included simulation and virtualization technologies, disciplines in which he published several papers and one book chapter, and disclosed 7 patents. He was one of the creators of COTSon, a full-system simulator co-developed by HP and AMD, which today is available as an open-source tool (http://cotson.sourceforge.net/). From 2010 to 2014, he was a Senior Research Scientist at the Intel Barcelona Research Center, where he has been the technical project lead and technical project manager of different projects. His research has focused on new microarchitecture paradigms and code generation techniques for future Intel microprocessors. With Intel, Ayose published several papers in internal conferences and disclosed 7 patents. Since 2014, Ayose is a senior R&D software engineer at HP Inc, working at the Large Format Printing division.

Jordi Tubella received his degree in computer science in 1986 and his Ph.D. in computer science in 1996, both from the Universitat Politècnica de Catalunya at Barcelona (Spain). He is a member of the Computer Architecture Department at the Universitat Politècnica de Catalunya since 1988, being a associate professor since 1998. His research interests focus on processor microarchitecture and parallel processing, with special interest on heterogeneous computing and speech recognition.

Antonio González (Ph.D. 1989) is a full professor at the Computer Architecture Department of the Universitat Politècnica de Catalunya, Barcelona (Spain), and was the founding director of the Intel Barcelona Research Center from 2002 to 2014.

His research has focused on computer architecture, compilers and parallel processing, with a special emphasis on processor microarchitecture and code generation. He has published over 325 papers, has given over 100 invited talks, holds over 40 patents and has advised 30 Ph.D. theses in these areas. He also has a long track record of innovations in commercial products, especially Intel products, during his stage as director of the Intel Barcelona Research Center.

Antonio has served as an associate editor of five IEEE and ACM journals, program chair for ISCA, MICRO, HPCA, ICS and ISPASS, general chair for MICRO and HPCA, and PC member for more than 100 symposia. Antonio's awards include the award to the best student in computer engineering in Spain graduating in 1986, the 2001 Rosina Ribalta award as the advisor of the best Ph.D. project in Information Technology and Communications, the 2008 Duran Farrell award for research in technology, the 2009 Aritmel National Award of Informatics to the Computer Engineer of the Year, the 2013 "King Jaime I Award" in New Technologies, and 2014 ICREA Academia Award. Antonio is an IEEE Fellow.

Lieven Eeckhout is a professor at Ghent University, Belgium, in the Department of Electronics and Information Systems (ELIS). He received his Ph.D. in Computer Science and Engineering from Ghent University in 2002. His research interests are in computer architecture, with a specific emphasis on performance evaluation and modeling. He served as the program (co-)chair for HPCA 2015, CGO 2013 and ISPASS 2009, and general chair for ISPASS 2010. He is the current editor-in-chief of IEEE Micro, and associate editor for IEEE Computer Architecture Letters, IEEE Transactions on Computers, and ACM Transactions on Architecture and Code Optimization.