Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors

Jian Li and José F. Martínez

Computer Systems Laboratory
Cornell University
Ithaca, NY 14853 USA

http://m3.csl.cornell.edu/

ABSTRACT

Previous proposals for power-aware thread-level parallelism on chip multiprocessors (CMPs) mostly focus on multiprogrammed workloads. Nonetheless, parallel computation of a single application is critical in light of the expanding performance demands of important future workloads. This work addresses the problem of dynamically optimizing power consumption of a parallel application that executes on a many-core CMP under a given performance constraint. The optimization space is two-dimensional, allowing changes in the number of active processors and applying dynamic voltage/frequency scaling. We demonstrate that the particular optimum operating point depends nontrivially on the power-performance characteristics of the CMP, the application's behavior, and the particular performance target. We present simple, low-overhead heuristics for dynamic optimization that significantly cut down on the search effort along both dimensions of the optimization space. In our evaluation of several parallel applications with different performance targets, these heuristics quickly lock on a configuration that yields optimal power savings in virtually all cases.

1 INTRODUCTION

Chip multiprocessors (CMPs) have emerged as a promising way to deliver sustained performance growth while relying less on raw circuit speed, and thus power [1]. That parallelism may bring power-performance advantages is not new: earlier VLSI works have discussed the trade-offs that sequential vs. parallel circuits present in silicon area and power consumption [6, 37]. Yet even as researchers have investigated extensively the power-performance issues of uniprocessor and, to a lesser extent, multiprogrammed CMP architectures [10, 14, 26, 32, 41, 42], to date there is still little understanding of the specific power-performance challenges involving parallel applications executing on CMPs.

In a parallel run, for example, the overall performance ultimately depends on all the processors; however, at any point in time, the critical path may depend on only a few of them. In that case, slowing down processors not in the critical path to save power may not affect the overall performance at all. Conversely, slowing down processors in the critical path will negatively impact performance, and the local savings may be easily negated by the extra waste on other processors due to longer execution time. Furthermore, the available parallelism and parallel efficiency may depend nontrivially on the problem size and execution environment. Moreover, it is viable to change the number of concurrent processors/threads at run-time [5, 16] to optimize execution across program regions, or to accommodate changes in the execution environment; however, the power-performance trade-offs that appear as a result of such run-time adaptive parallelism cannot be easily explained without considering the parallel application behavior.

As the number of cores per CMP increases and the opportunities for performance growth of single-threaded codes dwindle, we anticipate that many important future applications—as many as 80% by some industry projections [3, 25]—will be parallelized to utilize the potential of these CMP cores. With future CMPs likely support many threads on the same die, and its cores in turn supporting a number of voltage and frequency levels, the amount of possible power-performance configurations of a CMP is bound to be large, making it hard to find an optimal power-performance operating point for a parallel application, in particular at run-time.

In this work, we target run-time power-performance adaptation of future CMPs that run shared-memory parallel applications. This run-time adaptation takes place in the two-dimensional space constituted by (1) the possible number of active processors (2) the different voltage-frequency levels available. Specifically, in this paper, we explore one typical scenario—maximizing power savings while delivering a specified level of performance. This scenario essentially reduces energy cost per application, as energy is the integral of power over execution time. It aims to prolong battery life for embedded systems, or to reduce power supply capacity for high-end systems, as long as performance is satisfactory. We show that the optimum operating point depends nontrivially on the particular performance target, the application, and the hardware's power-performance characteristics. However, we demonstrate that the arrangement of the possible operating points generally allows for an efficient pruning of the search space, which enables low-overhead dynamic optimization. In that context, we present simple heuristic mechanisms that are able to quickly converge to a con-

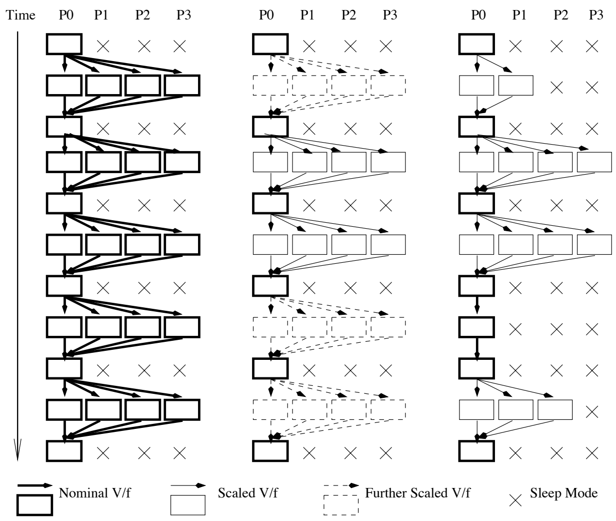

Figure 1: Execution of an imaginary parallel code on a CMP with four processors under three different scenarios: full throttle parallel execution (left), power-aware parallel execution regulated with DVFS exclusively (center), and power-aware parallel execution regulated with DVFS and adaptive parallelism (right). Regions are not drawn to scale with respect to each other.

figuration in the search space that achieves near-optimal power consumption and complies with the performance requirement in virtually all the cases that we study.

Our mechanism is implemented at the granularity of parallel regions, with a combination of modest software and hardware support. It does not change the application semantics, and can be made largely transparent to the application by encapsulating the code in a typical parallelization macro or directive.

The rest of the paper is organized as follows: Section 2 describes the scope of our study; Section 3 characterizes the search space and describes our proposed mechanisms; Section 4 lays out our experimental setup; Section 5 analyzes the results of our evaluation; and Section 6 discusses related work.

2 SCOPE

This section lays out the scope of our work. We use an imaginary execution of a parallel code on a CMP with four processors as an example. The code contains a series of interleaved serial and parallel regions. Figure 1 shows the execution of a fragment of such code under three different scenarios. In the figure, regions are not drawn to scale with respect to each other. In fact, we expect parallel regions to be dominant in many future applications, and thus the focus of our paper.

The left diagram represents a conventional parallel run at full throttle. The code runs on all four cores during parallel sections, all at nominal voltage/frequency levels—no dynamic voltage/frequency scaling (DVFS) is applied in any case.

The central diagram represents a power-aware execution where power can be regulated using whole-chip DVFS. In the example, parallel sections are slowed down by applying DVFS to all processors. This presents us with a power-performance trade-off, which we can exploit, for example, to reduce power consumption and still meet a predetermined performance target.

Unfortunately, this one-dimensional trade-off limits the power-performance optimization options. Specifically, no matter how relaxed the performance constraint, it is not possible to reduce power consumption in a parallel region below the static power consumption of all four cores at room temperature, since all of them remain active. Notice that if the chip's power budget is tight, executing in all cores may result in a very limited number of feasible DVFS levels. In the worst case, if the application's parallel efficiency degrades significantly at that degree of parallelism, the chip's limited power budget may make it simply impossible to meet the performance target [29].

The right diagram tries to address this limitation, by allowing parallel regions to execute on a variable number of processors. In this scenario, a parallel region executes on possibly a subset of the available processors, with DVFS properly adjusted to meet the performance target, and unused processors are brought down to sleep mode. (Naturally, we assume that the application does support execution of parallel regions with different number of processors. We address this assumption later in Section 3.3.)

On the one hand, this scenario is highly desirable, as it allows much greater flexibility in trading off power and performance. On the other hand, such a two-dimensional space can be quite large, especially as CMPs incorporate more cores, making the task of finding the optimal operating point a challenging one, particularly if the number of instances of a parallel region at run-time is such that a brute-force search could not be amortized easily. This motivates the need to explore whether a reasonably quick and easy procedure exists that can find an acceptable operating point at run time, such that power consumption is reduced significantly but the performance target is still met. Our work shows that this is feasible.

In our study, we limit ourselves to a homogeneous CMP that has chip-wide DVFS capability, and focus on power optimization at the granularity of a parallel region that must meet a certain performance target. We generally define our performance target as a steady-state rate (e.g., frames per second in a multimedia application). Finally, we generally assume a CMP in which each processor executes at most one application thread, and leave overthreading issues to future work.

3 DYNAMIC POWER-PERFORMANCE ADAPTATION

In this section, we first explain the general characteristics of the two-dimensional power-performance optimization space, using one of the applications from our experimental setup as an example (Section 4.3). Then, in the context of dynamic power optimization given a certain performance constraint, we describe simple run-time heuristics that cut down on the optimization space search significantly, and justify intuitively that such heuristics should

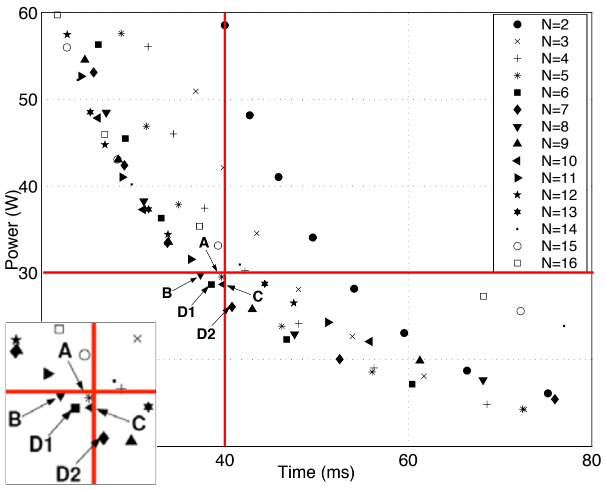

Figure 2: Power consumption and execution time of CMP configurations with varying number of processors N and voltage/frequency levels for an instance of BSOM's parallel region (a parallel data mining application). Target execution time and power are 40ms and 30W, respectively.

generally converge to an operating point that is reasonably close to the global optimum, which is later confirmed in our evaluation (Section 5.2). Finally, we comment on some hardware/software implementation issues of our mechanism.

3.1 Power-Performance Characterization

In this section, we build a case study around BSOM, the parallel data mining application (Section 4.3), to illustrate the power-performance trade-offs of parallel computation on a CMP. Fig. 2 shows the power consumption and execution time of an iteration of the main (parallel) loop, or *epoch*. Each point in the plot represents one CMP configuration, using a certain number of processors N (shown in the legend), and a particular DVFS level (not indicated directly in the plot). We vary N between two and sixteen processors, and for each N we explore sixteen different DVFS levels, distributed regularly along the allowable range (Section 4.1). In the figure, we limit our discussion to a window of power ranging from 10 to 60W, and execution time of an epoch ranging from 20 to 80ms. (Naturally, we are using a reduced input set to make simulation time affordable; execution time of each epoch is longer with realistic input sizes.)

In general, as expected, points for a fixed N move right and down as we apply DVFS, as it takes more time to execute one epoch, but on the other hand the power consumption is lower. The plot shows two well differentiated operating areas: On the one hand, operating points toward the upper-left corner lie in a *performance-oriented* (or *power-sensitive*) area. In this area, power consumption is generally high, but small reductions in voltage/frequency levels translate into significant savings in power consumption, at a relatively small performance cost. On the other hand, operating points toward the lower-right corner lie in a *power-oriented* or (*performance-sensitive*) area. In this area, while performance is generally low, small voltage/frequency increases result in large performance gains, at a relatively small power cost.

If we try to make sense of the curves depending on the performance target that we impose, we can again find two well differentiated areas. As we move left (strict performance target), configurations with a small number of processors need to operate at high DVFS levels to meet the performance target, thus consuming significant power. If, as it is the case in BSOM, parallel efficiency is generally high, a higher number of processors allows the CMP to meet the performance target while achieving significant power savings, by aggressively throttling voltage and frequency. On the other hand, toward the right end of the plot (relaxed performance target), most configurations can meet or exceed the performance target with very low voltage and frequency levels. In that case, the static power consumption of configurations with a high number of processors puts them at a disadvantage with respect to smaller configurations.

If we assume the target execution time for one epoch is 40ms (vertical line), and we consider 30W as the maximum allowable power dissipation (horizontal line), our search for valid configurations is confined, in principle, to the quadrant left of and below such bounds, respectively. Only a few operating points reside in this quadrant, which we label A, B, C, and D1 in the plot. Any of these points satisfy both performance and power constraints; the actual choice depends on a number of possible factors.

Configuration C in the plot, for example, yields acceptable performance and maximum power savings. Configuration D1 yields better performance at the same power cost, and thus would seem a better choice in terms of overall power-performance behavior. Configurations A and B consume a little more power, but they are still within the allowed limit.

One difference between these operating points lies in the number of processors required by each case, which varies widely—between five (configuration A) and ten (configuration C) in our experiment. If the application is also expected to minimize the number of allocated processors (for example, in a multiprogrammed environment), configuration C may prove too costly at $N = 10$. On the other hand, configuration A, which uses half as many processors, may constitute an attractive choice in this case, even as it consumes slightly more power. The opposite may be true if, for example, power density is a concern.

Finally, if we define our performance target as a rate (e.g., frames per second in MPEG decoder/encoder), and allow epochs to borrow leftover time from previous "fast" instances to execute slower and save more power while still meeting the target rate, further power savings opportunities may be possible. For example, we can alternate runs with $N = 6$ for one epoch, and $N = 7$ for the following epoch (configurations D1 and D2 in the plot, respectively). These configurations result, as the plot shows, in per-epoch execution times that are slightly under and over target, respectively. However, alternating between these two states is likely to meet or exceed the target rate (with D2 borrowing time from the slack created by D1). And the combination D1-D2 consumes, on average, less power than any other valid configuration by itself. Therefore, it can be argued that configuration D1 controls execution time as configuration D2 lowers power consumption, resulting in a favorable net balance.

3.2 Dynamic Power Optimization

When optimizing for power, the operating point that meets the performance requirement and minimizes power consumption lies, generally speaking, at the intersection of the performance target and the lower envelope of the power-performance curves for the different number of processors and DVFS levels. Unfortunately, such lower envelope is not generally known at run-time, and must be constructed or approximated.

If the number of instances of the parallel region is large enough that an exhaustive search is feasible, an easy way to construct such lower envelope at run-time is to try all possible number of processors and (legal) DVFS levels: For each number of processors, starting with the maximum DVFS level allowable by the chip's power budget (which we assume known), we gradually decrease DVFS levels (one step per instance of the parallel region) until the desired performance is no longer met. The DVFS level immediately before is the local minimum for that number of processors. Among all the local minima, a global minimum can be picked. The cost of an exhaustive search is $L \cdot N$ steps, with L being the number of DVFS levels and N the number of processors on the CMP.

Often times, however, an exhaustive search may not be possible, or its overhead prove prohibitive, because the parallel region is not invoked enough times as to amortize the search phase. In that case, search heuristics that can converge toward the global optimum much faster are highly desirable.

After the search has concluded, the algorithm enters steady state. In this mode, the goal is to minimize power consumption while making sure that the target rate (e.g., frames per second) is met. In this mode, as we finish execution of one instance, we compute the distance to the next target (since we may have under- or overshot the current target). Then we select, among the operating points recorded during our search phase, the one whose execution time constitutes the tightest upper bound. This gives us the processor count and DVFS level to use next. If we undershot the current target, this constitutes an opportunity to save extra power. If we overshot, we may now pick a more aggressive operating point, which will allow us to catch up.

Reducing Search Space: Processor Dimension

We propose to use a combination of binary search and hill-climbing optimization to prune the search space along the processor dimension. In hill-climbing optimization, search continues until a local optimum is observed—i.e., the immediate proximity of that point in any allowed search direction yields a less optimal operating point. Hill-climbing is limited in the sense that it may get "stuck" at a local optimum that is significantly worse than the global optimum in the search space. Three main factors influence the quality of hill-climbing algorithms: (1) the general shape of the search space; (2) whether the search algorithm generally conforms to such shape; and (3) possible heuristics to overcome local optima situations, such as the use of "jitter" or "momentum." In this paper, we choose to focus on the first two factors, and do not explore the third one.

From the insights developed in Section 3.1, we can build a search heuristic around the observed general trends in the search space, namely: (1) On tight performance constraints, configurations with more processors are generally favored, provided the application exhibits enough parallel efficiency, since they can meet the requirement at lower DVFS levels. (2) On loose performance constraints, configurations with fewer processors are generally preferable, since they can meet the requirement with low DVFS levels and they do not constitute as large an aggregation of static power; (3) on middle-ground performance constraints, multiple configurations that yield similar power-performance levels may be possible.

Cases (1) and (2) are likely to yield good operating points, since configurations are generally arranged in a monotonic fashion along the time constraint, which a hill-climbing algorithm can exploit well. In Case (3), the chances of getting "stuck" at a local optimum may be higher, however our evaluation empirically shows that the local optima found by our algorithm are generally close to the global optima in that scenario.

We conduct the hill-climbing optimization using a binary search along the dimension of the number of processors. Generally speaking, the hill-climbing algorithm starts at some mid-point number of processors p, and gradually decreases DVFS levels until the performance target is missed, as it is the case of the exhaustive search explained above. The DVFS level immediately above is the optimum for that p. Then, another p' half-way between the current configuration and either of the active endpoints is chosen, and the process is repeated. If the optimum for that p' is better, the other side is disregarded, and the binary search continues on that side. Otherwise, the algorithm switches to the other side, disregarding further attempts on the first side. When neither side is better, or we run out of options, the search ends.

The choice of which side to explore first may be important in cases where local optima may exist on both sides of the search, and thus the final number of processors may depend on the order in which they are searched. On the other hand, in the "monotonic" areas of the search space described above, the search is likely to converge toward the optimum. In any case, the cost of this heuristic is $L \cdot \lg N$ steps—a significant improvement.

Reducing Search Space: DVFS Dimension

An orthogonal way to prune the search space is to exploit, along the DVFS dimension, the strong correlation between performance and frequency in many applications. The formula for execution time proposed by Hennessy and Patterson [17] is $t = IC \cdot CPI \cdot f^{-1}$, where IC is the dynamic instruction count, CPI is the average number of cycles per instruction, and f is the operating frequency. If we make the simplifying assumption that CPI is independent of the clock frequency, then the the ratio between the actual and target execution times of a parallel region should be approximately equal to the ratio of the target and actual clock frequency.

This is a powerful result that would allow us, theoretically, to execute the parallel region once at clock frequency f, measure execution time t, and in one shot derive the target frequency f_{target} that would yield our performance target t_{target} (which is a known value of course).

In practice, however, neither is frequency in DVFS scaling a continuous function, nor is CPI independent of the clock frequency.[1] While the former limitation would still allow us to pick the optimum frequency in one shot (the closest legal frequency above the exact solution), the latter introduces some inaccuracy that may result in over- or undershooting the performance target. Nevertheless, observe that we can now apply a new iteration of the procedure, this time using execution time and frequency of the latest run. The key insight is that, with each additional iteration, the ratio should be much smaller than before, and thus convergence should be fast (unless the application exhibits erratic behavior, in which case none of the proposed heuristics is likely to work anyway).

As before, we start with the highest DVFS level allowable by the chip's power budget for the number of processors currently under consideration by the hill-climbing algorithm. If the execution time is unfavorable (i.e., the performance target is stricter than the measured performance), we cannot meet the required performance at the current number of processors, and thus we move on to another number of processors. If, on the other hand, the execution time is favorable, we can apply the formula to compute the new target frequency. The new *legal* target frequency maybe at the same DVFS level, in which case we stop searching, or at a lower DVFS level, in which case we iterate once more.

If we eventually miss the performance target, applying the formula again results in a target frequency that is necessarily faster, and since we always pick the closest legal frequency *above* the exact solution, this is guaranteed to move to a higher DVFS level. In the general case where this DVFS level has not been tried before, we iterate once more.

Notice that we may have already tried this DVFS level. Because of the general convergence property of the algorithm, the execution time at this DVFS level was most likely favorable the last time around, and thus we may stop searching. However, it might occur that this DVFS level was also recorded as unfavorable. In this (empirically rare) case, we simply select the closest recorded DVFS level above that yielded a favorable execution time and stop searching.

When combined with the hill-climbing heuristic, we estimate the expected cost to be $\alpha \lg N$ steps, where α is a function that grows much slower than L. In our evaluation (Section 5), where the number of DVFS levels is 16, the above procedure converges in about three iterations in most cases.

Thus, intuitively, the combination of the search reduction heuristics along each axis of the two-dimensional search space should converge quickly to an operating point that is reasonably close to the global optimum.

3.3 Implementation Issues

We envision implementing our proposed mechanism as a combination of modest hardware and software support. On the hardware side, we mainly require support to measure power and performance directly, chip-wide DVFS, and the ability to put cores to sleep. On the software side, we need support to execute parallel regions with different processor counts. We address each one in turn.

Hardware Support

Our mechanism obviously requires the ability to apply DVFS, although we limit our study to simple chip-wide DVFS, leaving potentially more versatile mechanisms such as core-level DVFS for future work.[2] Moreover, because the relative power-performance characteristics across different operating points may not be easily correlated to indirect metrics (e.g., IPC or cache miss rates), we would like to *directly* measure both power and performance to characterize such operating points, and thus we need to provide such support as well. While measuring performance can be achieved using well-known mechanisms based on programmable hardware counters, the hardware support to directly measure and regulate power is not as obvious.

In a recent publication [35], Intel describes its upcoming Foxton technology for Itanium Montecito. It utilizes on-chip sensors and an embedded microcontroller attached to the processor core to directly measure power and temperature, and apply DVFS to maximize the processor's performance while abiding by power/temperature constraints. We believe this microcontroller-based approach offers great potential and flexibility for our purposes, and thus propose that our mechanism be supported with similar hardware.

The Foxton-like microcontroller should be properly interfaced with the software, so that the choices of DVFS

[1] In memory-bound applications, for example, CPI may improve with lower frequencies, as off-chip memory accesses may become effectively faster in terms of processor clock cycles.

[2] In a multiprogrammed scenario, where the application receives a partition of the entire chip, our mechanism may require *partition*-wide DVFS. In that case, each partition may operate under different, interdependent power/temperature budgets. For the sake of simplicity, in this work we intentionally ignore this scenario.

CMP Size	16-way
Processor Core	Alpha 21264 [8]
Process Technology	65nm
Nominal Frequency	3.2GHz
Nominal V_{dd}	1.1v [21]
V_{th}	0.18v [21]
Ambient Temperature	45°C
Die Size	244.5mm^2 (15.6mm × 15.6mm)
L1 I-, D-Cache	64kB, 64B line, 2-way, 2-cycle RT
Unified L2 Cache	Shared on chip, 4MB, 128B line, 8-way, 12-cycle RT
Memory	75ns RT

Table 1: The CMP configuration modeled in the experiments. In the table, RT stands for round-trip.

and number of cores can be communicated and/or agreed upon properly. In that respect, an intermediate layer is needed to identify a parallel region (monitoring instruction addresses) and, based on past history and progress, decide on the appropriate course of action in terms of number of processors and DVFS level to apply to the execution of the next instance of such a region. The particular decision of which parts to map onto hardware or software libraries is more of an engineering issue that falls out of the scope of this paper. One definite requirement, however, is the ability to execute a parallel region on different number of processors. We address this next.

Software Support

In our proposed mechanism, both system and application should be able to support different processor counts on different instances of a parallel region.

On the system side, the operating system could assign a (probably oversized) partition to the application, which would basically determine the processor count range at the application's disposal. During the search phase, our mechanism would pick different number of processors and put the rest of the partition in a low-power sleep mode. Once the mechanism converges to a local optimum and enters steady state, any excess of processors could be given back to the operating system, or put in a low-power sleep mode for the duration of the program. (Notice that processors in a low-power sleep mode may still be required to respond to snoop requests if their cached updates have not been written back [30].)

On the application side, the parallel region should be written to support different processor counts. Fortunately, this is often supported (in fact, it is frequently the default mode) by widely used APIs for shared-memory programming, most notably OpenMP [36]. (Notice that some applications do restrict the possible number of processors to certain values, e.g. powers of two. While any search heuristic could be easily adapted to this scenario, we do not explicitly address it.) Furthermore, we envision the software-side support to be encapsulated in the existing parallel directives of such APIs, making it virtually transparent to the programmer or compiler.

4 SIMULATION ENVIRONMENT

4.1 Architecture

Our study uses a detailed model of a 16-processor CMP. CMP cores are modeled after the Alpha 21264 (EV6) processor [8]. Each processor core has private L1 instruction and data caches. All cores share a 4MB on-chip L2 cache through a common bus, and implement a MESI cache coherence protocol [9]. Table 1 lists relevant cache and memory parameters.

We choose a 65nm process technology. The original EV6 ran at 600MHz on a 350nm process technology; by proceeding similarly to [26], we determine the clock frequency of our 65nm EV6 cores to be 3.2GHz. We set nominal supply and threshold voltages at 1.1v and 0.18v, respectively [21], and in-box ambient air temperature at 45°C [33, 43]. Using CACTI [44], we obtain an estimated chip area of 244.5mm^2 (15.6mm × 15.6mm), using a scaling method similar to [27].

For the sake of simplicity, we assume global voltage/frequency scaling for the entire chip. (While it is conceivable to allow each core to run at a different frequency, the applicability and performance impact in the context of a parallel execution is nontrivial and beyond the scope of this paper.) Frequency can scale from 3.2GHz down to 200MHz, and we resort to [20] to establish the relationship between frequency and supply voltage. Notice that, because voltage/frequency scaling is applied at the chip level, on-chip latencies (e.g., on-chip cache hit time) do not vary in terms of cycles. However, a round trip to (off-chip) memory takes the same amount of time regardless of the voltage/frequency scaling applied on chip, and thus the round-trip memory latency in processor cycles goes down as we downscale frequency.

4.2 Power Model

We use Wattch to model the switching activity and dynamic power consumption of the on-chip functional blocks. As for static power consumption, we approximate it as a fraction of the dynamic power consumption [7, 43]. In our model, this fraction is exponentially dependent on the temperature [7]. The average operating temperature (over the chip area) in our model ranges from in-box ambient air temperature (45°C) to a maximum operating temperature of 100°C, in agreement with multiple contemporary processor chip designs. We use the HotSpot thermal model [43] for chip temperature estimation.

Wattch is reasonably accurate in relative terms; however, the absolute power values can be off by a nontrivial amount [26]. Because we use power values to communicate across two different tools (Wattch and HotSpot), we ought to ensure we do so in a meaningful way. We achieve this by renormalizing power values as follows.

We use HotSpot to determine the maximum operational power consumption (dynamic plus static), which is the one that yields the maximum operating temperature of 100°C. Then, using the dynamic/static ratio that corresponds to that temperature [7], we derive the dynamic component.

We now need to establish the connection with Wattch. To do so, we use a compute-intensive microbenchmark to recreate a quasi-maximum power consumption scenario at nominal voltage and frequency levels in our simulation model, and obtain Wattch's dynamic power value. This number is often different from the one obtained through HotSpot using the method explained above. To overcome this gap, we calculate the ratio between Wattch and HotSpot's dynamic power values, and use it throughout the experiments to renormalize wattage obtained with Wattch in our simulations as needed. This makes it possible for both tools to work together. While the absolute power may again not be exact, the results should be meaningful in relative terms. Using both tools, plus the power ratio/temperature curve, we are able to connect dynamic and static power consumption with temperature for any voltage and frequency scaling point.

Finally, we notice that the temperature and power density of the shared L2 cache is significantly lower than the rest of the chip across all the applications studied. Reasons include: much less switching activity; aggressive clock gating in the model [4]; and large L2 cache dissipation area. This observation is in agreement with published work by others [7, 10]. To obtain meaningful temperature figures, we exclude L2 from the temperature calculation. However, we do include the power consumption of L2 in the reported power consumption.

4.3 Applications

We select six parallel applications from different problem domains: MPGdec and MPGenc, two popular video decoding/encoding applications, from ALPBench [31]; FMM, Volrend, and Water-Ns, which represent N-body, rendering, and molecular dynamics applications, respectively, from the SPLASH-2 suite [45]; and BSOM, a parallelized data mining application [28]. The execution time in all these applications is spent mostly on one single parallel region, which suits our purpose. In our experiments, we do not change the problem sizes as we change the number of cores. Table 2 lists the applications and their execution parameters. The number of instances in the parallel regions of FMM (steps of an N-body problem), Volrend (rendering from different viewpoints), and Water-Ns (steps of a molecular dynamics problem) as included in the SPLASH-2 benchmark suite are only a handful, and thus we increase that number to obtain a sufficient number of samples for our evaluation (50 for Volrend and Water-Ns, only 10 for FMM given the extended simulation time of each region). On the other hand, in MPGdec/MPGenc we simulate 60 frames (out of the original 150) to get resonable simulation times. In all cases, we skip initialization and then simulate to completion.

We assume all processors are available to the application, and that unused processors are put to sleep. We also do not model the overhead of switching among DVFS levels, as we reasonably assume that real-world parallel instances of interest would each run for much longer than the typical tens of microseconds for DVFS switching.

Application	*Description*	*Problem Size*
BSOM	Batched Self-Organizing Maps of neural network	16k records, 104 dim., 16-node network, 50 epochs
FMM	Fast Multipole Method	16k particles, 10 steps
MPGdec	MPEG-2 decoder	flowg.mpg (Stanford) 352×240, 60 frames
MPGenc	MPEG-2 encoder	flowg.mpg (Stanford) 352×240, 60 frames
Volrend	Volume rendering using a ray casting technique	head 50 viewpoints
Water-Ns	Forces and potentials of water molecules	512 molecules 50 steps

Table 2: Applications used in the experiments.

Because these applications are generally not written to change the number of processors dynamically, we approximate this behavior by simulating in two phases: In the first phase, we execute each application once for every combination of processor number and DVFS level, and collect the measured execution time and power consumption for each instance of the parallel region. In the second phase, for each application, we simulate the different optimization mechanisms with Matlab, using the processor count and DVFS level selected by the optimization mechanisms in each step to pick the execution time and power consumption of the appropriate instance from the first phase, which in turn serve to determine the next step in the optimization mechanism. In the case of FMM, whose simulation time was particularly long, we traverse its ten instances five times to come up with a total number similar to the other applications.

5 EVALUATION

In Section 3.2, we discuss the extent to which our combined heuristics may cut down on the two-dimensional space search of processor count and DVFS. We predict this to be important for parallel applications for which the number of instances or steps is moderate, particularly as we scale up the number of processors on a CMP. For our proposed mechanisms to be useful, however, we need to address two additional questions: (1) For a parallel region and a particular performance target, is the choice of operating point(s) that minimize power consumption a nontrivial one that makes a space search useful? (2) In spite of the reduced knowledge about the search space, do the proposed mechanisms achieve optimum power savings and still meet the performance requirement? Our evaluation shows that the answer to these two questions is Yes. In what follows, we address each question in turn.

In all our experiments, we set the chip's total power budget to that of one processor running at peak performance.

5.1 Optimization Space

In our first experiment, we assess the power optimization opportunities for each processor count within range (1 to 16). Specifically, for a particular processor count, we measure power and performance for each possible operating point within budget. Then, in steady state, we pick

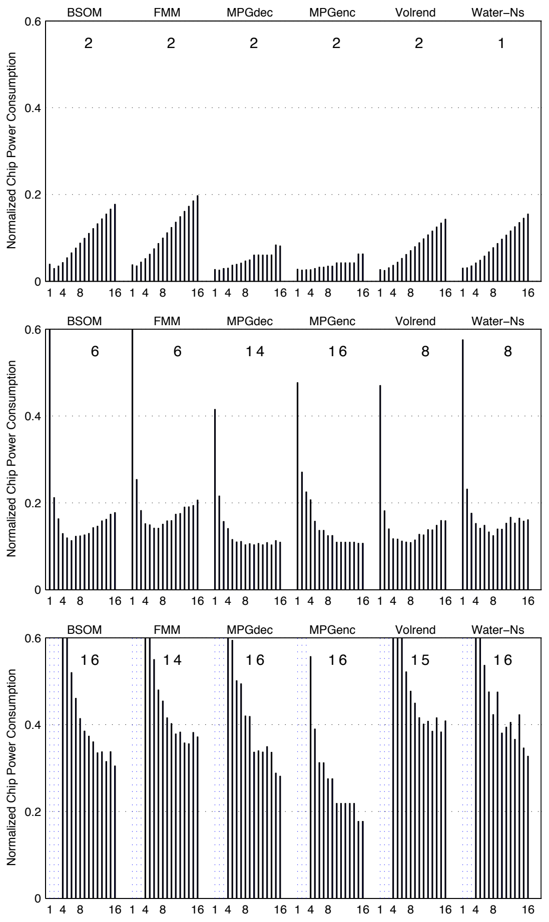

Figure 3: Optimal chip power consumption, normalized to the power budget, of configurations with different processor counts [1,16] for all the applications under study, for loose (top), intermediate (middle), and tight (bottom) performance targets. Dotted bars indicate configurations that cannot meet the specified performance target within the power budget. The numbers on top of bar groups correspond to the number of processors that yields the lowest power consumption.

at each instance the DVFS level whose recorded execution time is the tightest upper bound to our next target. (Recall that, in steady state, as a result of variability in the execution time across instances, we accummulate any deviation from the target into the next instance's). Our goal is to see whether there exists a "universal pick" of processor count regardless of the performance target. To do that, for each application, we pick three deadlines: (1) a "loose" performance target, roughly equivalent to one fourth of the fastest possible execution on one processor within the power budget; (2) an intermediate deadline, roughly equivalent to the fastest possible execution on one processor within the power budget; and (3) a "tight" performance target, roughly equivalent to the fastest possible execution on four processors, still within the power budget. We normalize all power measurements to the chip's power budget. Fig. 3 shows the results.

The plots show that significant (and often nonlinear) differences exist in the optimized power consumption for each processor count, even as the performance of all plotted configurations are within 2-3% of the target (not shown). Generally speaking, given a particular performance target, an increase in the number of processors allows for DVFS downscaling. This may initially result in overall power savings, however as we keep increasing the number of processors and we run out of DVFS levels, static power starts to dominate, eventually reversing the power savings trend.

With a loose performance target (top), configurations with low processor count can downscale DVFS aggressively, consuming little power, but leaving little room for further DVFS reduction to higher processor counts, which soon experience increased (static) power consumption. On the other hand, a tight performance target (bottom) requires configurations with low processor count to use high DVFS levels (and thus power) in order to meet the performance constraint, which allows a prolongued trend of power savings as we increase the processor count (provided the application scales well [29].)

Notice that, with greater static power consumption in future process technologies, the differences in power consumption for the scenarios with loose and intemediate performance targets (top and middle plots, respectively) are bound to increase, as static power will be more dominant in configurations with high processor count.

Moreover, across the plots, it becomes evident that the optimum processor count shifts depending on the performance target. Each plot shows, on top of the bars for each application, the number of processors of the configuration that minimizes power consumption.

Thus, to find the configuration that minimizes power consumption, it is generally necessary to connect the particular performance target to the power and performance characteristics of the application and the hardware, which is precisely what our proposed mechanisms try to do at run time.

5.2 Effectiveness of Optimization Mechanisms

We now investigate the effectiveness of the proposed optimization mechanisms. We combine the DVFS search heuristic and the hill-climbing binary processor count search heuristic; this we call HC (for Hill Climbing). HC starts with eight processors, and initially veers toward a lower number of processors (four). After that, and for the duration of the search phase, HC reuses the last profitable direction in determining which new processor count to try first. Once in steady state, HC uses the operating points visited during the search phase to try and match the target rate.

We also try a variant of HC which we call HC-Fixed, in which, once in steady state, we can only use the operating points visited during the search phase whose processor count is that of the local optimum found by the hill-climbing heuristic. This represents a scenario in which, for whatever reason, we do not want to use a variable number of processors in steady state.

We use the same power budget and performance targets as before. For comparison purposes, for each application

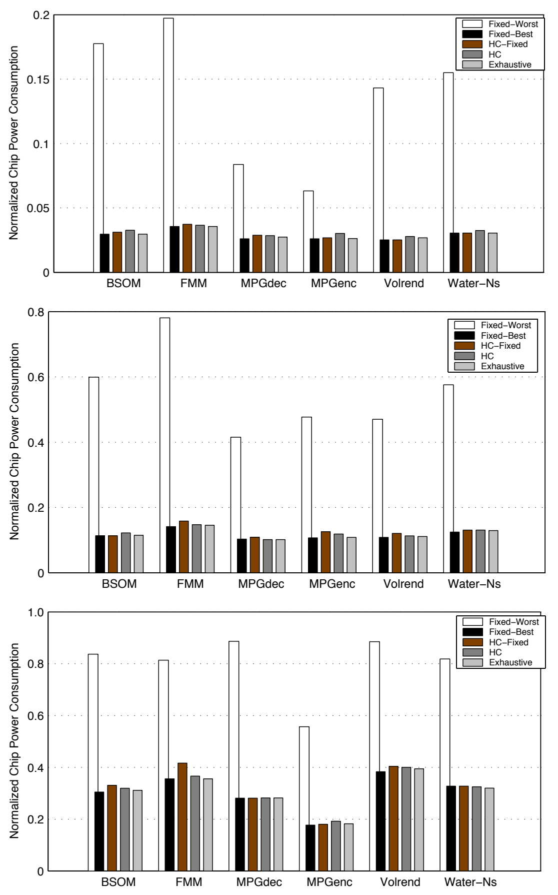

Figure 4: Chip power consumption of two proposed mechanisms (HC and HC-Fixed) against the best (Fixed-Best) and worst (Fixed-Worst) configurations of the earlier experiment, and a configuration with full knowledge of the search space (Exhaustive), for loose (top), intermediate (middle), and tight (bottom) performance targets. All bars are normalized to the power budget. Notice the different scales on the Y axes.

and performance target, we plot the power consumption of the best (Fixed-Best) and worst (Fixed-Worst) configurations from the earlier experiment. (Recall that Fixed-Best and Fixed-Worst can use *any* legal DVFS level for their processor count. This is unlike HC and, in particular, HC-Fixed.)

Finally, we also compare against a configuration that has full knowledge of the execution time and power consumption of every operating point for each instance of the parallel region under study. We call this optimistic configuration Exhaustive.

Fig. 4 shows the power consumption for each configuration, all normalized to the power budget. As in the case of the earlier experiments, all configurations successfully execute within 2-3% of the performance target in steady state (not shown). The results are very encouraging: In virtually all cases, our proposed mechanisms (HC and HC-Fixed) are capable of achieving the same level of power savings of not only the best configuration from the earlier experiment (Fixed-Best), but even Exhaustive, which has full knowledge of the power-performance behavior of all possible operating points at each moment. This result is quite remarkable, considering that HC can only leverage the approximately 12 operating points visited during the search phase in most cases (about four steps along the processor dimension, with around three steps along the DVFS dimension each time). Even more remarkable is HC-Fixed which, with only about three visited operating points along the range of the final processor count, closely tracks the performance of the more powerful configurations, with the possible exception of FMM on a fast performance target (bottom plot), in which HC-Fixed trails the other optimization mechanisms somewhat (but HC does not).

Overall, the results of our evaluation show that, indeed:

- For a parallel region and a particular performance target, the choice of operating point(s) that minimize power consumption in our experimental setup is nontrivial, which makes optimization space search useful.

- In spite of the reduced knowledge about the optimization space, the proposed mechanisms HC and HC-Fixed, based on the presented heuristics for space search reduction, achieve virtually optimum power savings for the applications under study.

6 RELATED WORK

There is a rich collection of literature on power- and thermal-aware simultaneous multithreading (SMT) and CMP designs (or similar architecture configurations), most of which focuses on multiprogrammed workloads [10, 14, 26, 32, 41, 42]. In contrast, our work focuses on power-performance issues of CMPs in the context of parallel applications.

Huh et al. [19] conduct an in-depth exploration of the design space of CMPs. However, they do not address power. More recently, Ekman and Stenström [11] conduct a design-space study of CMPs in which they address some power issues. Assuming a certain silicon budget, they compare chips with different numbers of cores, and correspondingly different core sizes. They argue that parallel applications with limited scalability but some instruction-level parallelism may run better on CMPs with few, wide-issue cores. They also argue that CMPs with few, wide-issue cores and with many, narrow-issue cores consume roughly the same power, as cache activity offsets savings at the cores. Our work assumes a given chip design, and explores the issues of minimizing power consumption by judiciously applying the optimum number of processors and voltage/frequency levels to a parallel region, given certain performance constraints.

Grochowski et al. [15] discuss trade-offs between microprocessor processing speed vs. throughput in a power-constrained environment. They postulate that a microprocessor that can achieve both high scalar performance and high throughput performance ought to be able to dynamically vary the amount of energy expended to process each instruction, according to the amount of parallelism available in the software. To achieve this, they survey four techniques: dynamic voltage/frequency scaling (DVFS), asymmetric cores, variable-sized cores, and speculation control, and conclude that a combination of DVFS and asymmetric cores is best.

More recently, Annavaram et al. [2] use an asymmetric CMP to maximize the performance of a multithreaded application, by assuming that nontrivial serial regions exist in the application. They use the notion of energy per instruction (EPI) throttle to orchestrate the application's execution on its sequential and parallel portions under a fixed power budget. For the sequential portions, they assign a faster but more power-hungry processor. For the parallel portions, depending on the number of threads that are inherent in the application's parallelization and the number of available processors, they assign multiple slower but power-thrifty processors. In our work, we study dynamic optimization on a parallel region running on a symmetric CMP with a large configuration space.

Kaxiras et al. [24] compare the power consumption of an SMT and a CMP digital signal processing chip for mobile phone applications. They do not explicitly study parallel applications in the "traditional" sense. For example, they approximate a parallel encoder with four independent MPEG encoder threads, each thread processing one quarter of the original image size. A speech encoder and a speech decoder are connected in a pipelined fashion to a channel encoder and decoder, respectively. The issues that we address in our work cannot be easily conveyed in this context.

Kadayif et al. [22] propose to shut down idle processors in order to save energy when running nested loops on a CMP. The authors also study a pre-activation strategy based on compiler analysis to reduce the wake-up overhead of powered-off processors. Although they address program granularity and power, they do not exploit DVFS in their solution, which is fundamental in our work.

In a different work, Kadayif et al. [23] propose to use DVFS to slow down lightly loaded threads, to compensate for load imbalance in a program and save power and energy. They use the compiler to estimate the load imbalance of array-based loops on single-issue processor cores. The authors also mention the opportunity for further energy savings by using less than the number of available processor cores using profile information. However, the connection of DVFS to parallelization granularity of the code is not fleshed out.

In the context of cache-coherent shared-memory multiprocessors, Moshovos, et al. [34] reduce energy consumption by filtering snoop requests in a bus-based parallel system. Saldanha and Lipasti [40] observe significant potential of energy savings by using serial snooping for load misses. Li, et al. [30] propose saving energy wasted in barrier spin-waiting, by predicting a processor's stall time and, if warranted, forcing it into an appropriate ACPI-like low-power sleep state. This work is complementary to ours in that it does not consider the number of processors, and does not attack power consumption during useful activity by the parallel application.

In an environment of loosely-coupled web servers running independent workloads, several studies evaluate different policies to control the number of active servers (and thus their performance level) to preserve power while maintaining acceptable quality of service [12, 13, 38, 39]. Elnozahy et al [12] evaluate policies that employ various combinations of independent and coordinated dynamic voltage/frequency scaling, and node vary-on/vary-off, to reduce the aggregated power consumption of a web server cluster during periods of reduced workload. They evaluate the policies with simulations, and show that the combination of coordinated voltage/frequency scaling and node vary-on/vary-off obtains the largest power savings. They only consider dynamic power in their simulations.

In the context of micro-architectures, Heo and Asanović [18] study the effectiveness of pipelining as a power-saving tool in a uniprocessor. They examine the relationship between the logic depth per stage and the supply voltage in deep submicron technology under different conditions. This is complementary to our work, since we study power-performance issues of using multiple cores on a CMP.

7 CONCLUSIONS

In this work, we have addressed the problem of dynamic power optimization of parallel execution on many-core, DVFS-capable CMPs under given performance restrictions. We have shown that the number of available processors and DVFS levels may constitute a considerable search space, and the particular optimum depends nontrivially on the power-performance CMP characteristics, the application's behavior, and the specific performance target. To attack this problem, we have proposed simple heuristics that can be used to cut down on the search effort along both dimensions of the optimization space. Our evaluation shows that these heuristics produce near-optimum results in virtually all cases considered.

ACKNOWLEDGMENTS

We thank Engin İpek and the anonymous reviewers for valuable feedback. This work is supported in part by NSF awards CNS-0509404 and CCF-0429922, and gifts from Intel.

REFERENCES

[1] T. Agerwala. Computer architecture: Challenges and opportunities for the next decade. In *International Symposium on Computer Architecture*, München, Germany, June 2004. (Keynote presentation).

[2] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law through EPI throttling. In *International Symposium on Computer Architecture*, pages 298–309, Madison, Wisconsin, June 2005.

[3] S. Y. Borkar. Platform 2015: Intel processor and platform evolution for the next decade. Technical report, Intel White Paper, Mar. 2005.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, pages 83–94, Vancouver, Canada, June 2000.

[5] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky. Adaptive parallelism and Piranha. *IEEE Computer*, 28(1):40–49, Jan. 1995.

[6] A. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, Apr. 1992.

[7] P. Chaparro, J. González, and A. González. Thermal-effective clustered microarchitectures. In *Workshop on Temperature-Aware Computer Systems*, München, Germany, June 2004.

[8] Compaq Computer Corporation, Shrewsbury, Massachusetts. *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.

[9] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[10] J. Donald and M. Martonosi. Temperature-aware design issues for SMT and CMP architectures. In *Workshop on Complexity-Effective Design*, München, Germany, June 2004.

[11] M. Ekman and P. Stenström. Performance and power impact of issue-width in chip-multiprocessor cores. In *International Conference on Parallel Processing*, pages 359–368, Kaohsiung, Taiwan, Oct. 2003.

[12] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Workshop on Power Aware Computing Systems*, pages 179–196, Cambridge, MA, Feb. 2002.

[13] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, Mar. 2003.

[14] S. Ghiasi and D. Grunwald. Design choices for thermal control in dual-core processors. In *Workshop on Complexity-Effective Design*, München, Germany, June 2004.

[15] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. In *International Conference on Computer Design*, pages 236–243, San Jose, CA, Oct. 2004.

[16] M. Hall and M. Martonosi. Adaptive parallelism in compiler-parallelized code. In *SUIF Compiler Workshop*, Stanford University, CA, Aug. 1997.

[17] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier Science Pte Ltd., third edition, 2003.

[18] S. Heo and K. Asanović. Power-optimal pipelining in deep submicron technology. In *International Symposium on Low Power Electronics and Design*, Newport Beach, CA, Aug. 2004.

[19] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, Barcelona, Spain, Sept. 2001.

[20] Intel Corporation. *Intel Pentium M Processor on 90nm Process with 2-MB L2 Cache Datasheet*, June 2004.

[21] The ITRS Technology Working Groups. *International Technology Roadmap for Semiconductors (ITRS)*, http://public.itrs.net.

[22] I. Kadayif, M. Kandemir, and U. Sezer. An integer linear programming based approach for parallelizing applications in on-chip multiprocessors. In *IEEE/ACM Design Automation Conference*, pages 703–708, New Orleans, LA, June 2002.

[23] I. Kadayif, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Exploiting processor workload heterogeneity for reducing energy consumption in chip multiprocessors. In *Design, Automation and Test in Europe*, pages 1158–1163, Paris, France, Feb. 2004.

[24] S. Kaxiras, G. Narlikar, A. D. Berenbaum, and Z. Hu. Comparing power consumption of an SMT and a CMP DSP for mobile phone workloads. In *International Conference on Compilers, Architecture, and Systhesis for Embedded Systems*, pages 211–220, Atlanta, Georgia, Nov. 2001.

[25] D. J. Kuck. Platform 2015 software: Enabling innovation in parallelism for the next decade. Technical report, Intel White Paper, Mar. 2005.

[26] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *International Symposium on Microarchitecture*, pages 81–92, San Diego, CA, Dec. 2003.

[27] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *International Symposium on Computer Architecture*, pages 64–75, München, Germany, June 2004.

[28] R. D. Lawrence, G. S. Almasi, and H. E. Rushmeier. A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems. *Data Mining and Knowledge Discovery*, 3(2):171–195, Sept. 1999.

[29] J. Li and J. F. Martínez. Power-Performance implications of thread-level parallelism on chip multiprocessors. In *International Symposium on Performance Analysis of Systems and Software*, Austin, TX, Mar. 2005.

[30] J. Li, J. F. Martínez, and M. C. Huang. The Thrifty Barrier: Energy-aware synchronization in shared-memory multiprocessors. In *International Symposium on High-Performance Computer Architecture*, pages 14–23, Madrid, Spain, Feb. 2004.

[31] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The ALPBench benchmark suite for complex multimedia applications. In *IEEE International Symposium on Workload Characterization*, Austin, TX, Oct. 2006.

[32] Y. Li, D. Brooks, Z. Hu, and K. Skadron. Performance, energy, and temperature considerations for SMT and CMP architectures. In *International Symposium on High-Performance Computer Architecture*, San Francisco, CA, Feb. 2005.

[33] R. Majan. Thermal management of CPUs: A perspective on trends, needs and opportunities. In *International Workshop on Thermal Investigations of ICs and Systems*, Madrid, Spain, Oct. 2002. Keynote presentation.

[34] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary. JETTY: Filtering snoops for reduced energy consumption in SMP servers. In *International Symposium on High-Performance Computer Architecture*, pages 85–96, Nuevo Leone, Mexico, Jan. 2001.

[35] S. Naffziger, B. Stackhouse, and T. Grutkowski. The implementation of a 2-core multi-threaded itanium-family processor. In *IEEE International Solid-State Circuits Conference*, San Francisco, CA, Feb. 2005.

[36] OpenMP Architecture Review Board. *OpenMP Specifications*, http://www.openmp.org.

[37] K. K. Parhi. *VLSI Digital Signal Processing Systems*. John Wiley and Sons, Inc., New York, NY, 1999.

[38] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *International Workshop on Compilers and Operating Systems for Low Power*, Barcelona, Spain, Sept. 2001.

[39] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *International Symposium on Performance Analysis of Systems and Software*, pages 111–122, Austin, TX, Mar. 2003.

[40] C. Saldanha and M. Lipasti. Power efficient cache coherence. In *Workshop on Memory Performance Issues*, Göteborg, Sweden, June 2001.

[41] R. Sasanka, S. V. Adve, Y. Chen, and E. Debes. Comparing the energy efficiency of CMP and SMT architectures for multimedia workloads. In *International Conference on Supercomputing*, pages 196–206, Malo, France, June–July 2004.

[42] J. S. Seng, D. M. Tullsen, and G. Z. N. Cai. Power-sensitive multithreaded architecture. In *International Conference on Computer Design*, pages 199–208, Austin, Texas, Sept. 2000.

[43] K. Skadron, M. Stan, W. Huang, and S. Velusamy. Temperature-aware microarchitecture: Extended discussion and results. Technical Report CS-2003-08, University of Virginia, Apr. 2003.

[44] S. Wilton and N. Jouppi. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[45] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy, June 1995.