Understanding Fundamental Design Choices in Single-ISA Heterogeneous Multicore Architectures

KENZO VAN CRAEYNEST and LIEVEN EECKHOUT, Ghent University

Single-ISA heterogeneous multicore processors have gained substantial interest over the past few years because of their power efficiency, as they offer the potential for high overall chip throughput within a given power budget. Prior work in heterogeneous architectures has mainly focused on how heterogeneity can improve overall system throughput. To what extent heterogeneity affects per-program performance has remained largely unanswered. In this article, we aim at understanding how heterogeneity affects both chip throughput and per-program performance; how heterogeneous architectures compare to homogeneous architectures under both performance metrics; and how fundamental design choices, such as core type, cache size, and off-chip bandwidth, affect performance.

We use analytical modeling to explore a large space of single-ISA heterogeneous architectures. The analytical model has linear-time complexity in the number of core types and programs of interest, and offers a unique opportunity for exploring the large space of both homogeneous and heterogeneous multicore processors in limited time. Our analysis provides several interesting insights: While it is true that heterogeneity can improve system throughput, it fundamentally trades per-program performance for chip throughput; although some heterogeneous configurations yield better throughput and per-program performance than homogeneous designs, some homogeneous configurations are optimal for particular throughput versus per-program performance trade-offs. Two core types provide most of the benefits from heterogeneity and a larger number of core types does not contribute much; job-to-core mapping is both important and challenging for heterogeneous multicore processors to achieve optimum performance. Limited off-chip bandwidth does alter some of the fundamental design choices in heterogeneous multicore architectures, such as the need for large on-chip caches for achieving high throughput, and per-program performance degrading more relative to throughput under constrained off-chip bandwidth.

Categories and Subject Descriptors: C.1.3 [**Computer Systems Organization**]: Processor Architectures—*Heterogeneous (hybrid) systems*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling techniques*; C.0 [**Computer Systems Organization**]: General—*Modeling of computer architecture*

General Terms: Design, Performance, Experimentation

Additional Key Words and Phrases: Multicore processor, single-ISA heterogeneous architecture, analytical modeling, design space exploration

ACM Reference Format:
Van Craeynest, K. and Eeckhout, L. 2013. Understanding fundamental design choices in single-ISA heterogeneous multicore architectures. ACM Trans. Architec. Code Optim. 9, 4, Article 32 (January 2013), 23 pages.
DOI = 10.1145/2400682.2400691 http://doi.acm.org/10.1145/2400682.2400691

K. Van Craeynest is supported through a doctoral fellowship by the Agency for Innovation by Science and Technology (WT). Additional support is provided by the UGhent ICT department, the FWO project G. 0255.08, the UGhent- BOF project 01Z04109 and the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC Grant No. 259295.
Authors' addresses: K. Van Craeynest and L. Eeckhout (corresponding author), ELIS Department, Ghent University, Belgium; email: lieven.eeckhout@elis.ugent.be.

DOI 10.1145/2400682.2400691 http://doi.acm.org/10.1145/2400682.2400691

1. INTRODUCTION

Heterogeneous multicore processor architectures have received substantial interest in both academia and industry over the past years. One of the primary drivers for heterogeneous architectures is higher performance for a given power budget, or higher power efficiency for a given performance target, by dynamically scheduling jobs on the highest-performance or most power-efficient core on the chip. By doing so, total chip throughput is maximized while not exceeding the total power budget, or vice versa, power and energy consumption is reduced while maintaining specific performance targets. Prior work has reported substantial power and energy savings through heterogeneity [Kumar et al. 2004]. Industry is actively pursuing the road of heterogeneity: example heterogeneous architectures are the IBM Cell processor with its 8 special-purpose engines and one general-purpose RISC core [Kahle et al. 2005], as well as recently introduced CPU chips with an integrated GPU such as Intel's Sandy Bridge [Intel 2008], AMD's Fusion [AMD 2008], and NVidia's Tegra [NVidia 2010]. Other commercial offerings integrate different general-purpose CPU core types; see for example NVidia's Kal-El [NVidia 2011] which integrates four performance-tuned cores along with one energy-tuned core, and ARM's big.LITTLE chip [Greenhalgh 2011], which integrates a high-performance big core with a low-energy small core on a single chip. The latter two examples are so-called single-ISA heterogeneous multicores, which means that the different core types implement the same Instruction-Set Architecture (ISA).

In this article, we aim at exploring the heterogeneous single-ISA multicore architecture design space, and address some of the fundamental questions related to heterogeneity, such as: What are the performance benefits from heterogeneity over homogeneous architectures, i.e., how does heterogeneity affect chip throughput versus job turnaround time? If heterogeneity yields any performance benefits, what level of heterogeneity should be supported, i.e., how many different core types should be integrated? Do two different core types provide most of the benefit or do we need more core types? And what should these core types look like? Should we go for extreme core types, i.e., aggressive 4-wide out-of-order cores versus scalar in-order processor cores? Or should we deploy middle-of-the road cores along with extreme core types? How does heterogeneity affect off-chip bandwidth requirements? Or, vice versa, how do bandwidth limitations affect some of the fundamental trade-offs in heterogeneous architectures? In spite of the substantial amount of prior work done in this area, a comprehensive study exploring these heterogeneous processor trade-offs and design choices has not been published before, to the best of our knowledge.

We use analytical modeling for exploring the heterogeneous multicore design space. The analytical model employed in this work estimates heterogeneous multicore performance from single-core runs, i.e., the model has linear-time complexity in the number of core types while enabling performance predictions for arbitrary compositions of heterogeneous architectures and workloads. Moreover, it allows for quantifying heterogeneous architecture performance for a large number (hundreds) of possible job mixes in a reasonable amount of time. Performing the same study using architectural simulation would have been completely infeasible because of its time complexity: simulating and exploring a large heterogeneous architecture design space for a very large number of job mixes is impossible in a reasonable amount of time.

Our methodology uses analytical modeling to estimate performance for an arbitrary heterogeneous processor architecture and arbitrary job mixes. In contrast to prior work which focused on total chip throughput (also called weighted speedup), we quantify performance along two dimensions: we measure both overall system throughput and per-program performance (average job turnaround time). Although throughput and per-program performance are not independent, we find it very insightful to analyze multicore processor in terms of these performance axes. In particular, we determine the frontier of Pareto-optimal processor architectures that provide the optimum trade-off between system throughput versus average job turnaround time. Pareto-optimality implies that there exist no design points that outperform the Pareto-optimal frontier on all objectives (both system throughput *and* job turnaround time) at the same time. In other words, one cannot say whether one Pareto-optimal configuration outperforms another Pareto-optimal configuration—instead, Pareto-optimal configurations represent different trade-offs. We apply our methodology to SPEC CPU2006 workload mixes and we consider heterogeneous multicore processor configurations with up to five core types ranging from simple single-issue in-order cores to aggressive four-wide out-of-order cores.

This analysis leads to several interesting and insightful observations.

—While it is true that, as reported by prior work, replacing an aggressive out-of-order core in a homogeneous architecture with several simple in-order cores improves system throughput (while assuming a fixed chip area), it also decreases average per-program performance. Conversely, trading a number of simple in-order cores for an aggressive out-of-order core improves per-program performance, but it also decreases total system throughput. So, fundamentally, heterogeneity trades job turnaround time for total system throughput.

—Homogeneous architectures cover a broad range of the performance spectrum in terms of throughput versus job turnaround time. For a fixed chip area budget, a limited number of aggressive out-of-order cores yield short job turnaround time with limited system throughput; a large number of simple in-order cores on the other hand yield high system throughput at the cost of longer job turnaround times. Mediocre cores yield intermediate design trade-offs. Heterogeneity on the other hand allows for designing multicore processors with more fine-grained trade-offs in system throughput versus job turnaround time. Interestingly though, although there exist heterogeneous design points that outperform homogeneous designs both in terms of throughput and per-program performance, some homogeneous design points appear on the heterogeneous architecture Pareto frontier. In other words, some homogeneous configurations are optimal for particular throughput versus job turnaround time trade-offs.

—We find that two core types offer most of the performance benefits from heterogeneity, i.e., going to a larger number of core types does not contribute much. However, performance is greatly affected by which core types are chosen and different compositions lead to different performance trade-offs. Further, some compositions of core types do not yield Pareto-optimal configurations. For other compositions, the number of cores of each core type determines whether the heterogeneous multicore processor is globally optimal.

—Limited off-chip bandwidth has a significant impact on the fundamental design choices in heterogeneous architectures. When limiting off-chip bandwidth, increasing system throughput comes at the cost of a proportionally larger degradation in per-program performance. Further, although a homogeneous design with many small cores yields the highest throughput assuming infinite bandwidth, only heterogeneous designs can achieve the highest throughput under limited off-chip bandwidth. Finally, architectures designed for high throughput should employ large LLCs in order to reduce off-chip bandwidth pressure.

—We also find that the effectiveness of heterogeneous architectures heavily depends on how jobs are mapped on the different core types. Simple heuristics based on CPI or miss rates to discern compute- versus memory-intensive jobs do not achieve optimum performance. Instead, more accurate estimates that compare relative performance across core types are needed for effective job-to-core mapping.

The key contributions of the article are to comprehensively explore the heterogeneous multicore design space and provide insight in some of the fundamental trade-offs and design choices. We use analytical modeling to do so which enables exploring many more machine configurations and workloads than what is possible to consider with cycle-accurate simulation. Further, one of the key trade-offs that we study relates to chip throughput versus per-program performance. Prior work on the other hand focused on throughput only, for the most part; and prior work that did consider both chip throughput and per-program performance assumed different workload conditions. Grochowski et al. [2004] and Annavaram et al. [2005] considered multithreaded workloads and advocated spending more energy per instruction during serial phases (e.g., run serial phases on big cores ot at higher frequency/voltage in a heterogeneous multicore); Kumar et al. [2004] focus on throughput when assuming a fixed number of independent programs in a multi-program workload, and focus on per-program performance when considering variable active thread counts in multi-program workloads. In this work, we consider abundant numbers of active thread counts, i.e., at least as many independent programs as there are cores, and we find that heterogeneous multicores provide a trade-off between chip throughput and per-program performance, even under such workload conditions.

The remainder of this article is organized as follows. We first describe the analytical model that we use in our exploration (Section 2). Subsequently, we present our methodology for exploring the heterogeneous processor architecture design space (Section 3). After detailing our experimental setup (Section 4), we then present our results and findings (Section 5). Finally, we describe related work (Section 6) and conclude (Section 7).

2. MULTICORE PERFORMANCE MODELING

The analytical model used in this article is called the Multi-Program Performance Model (MPPM), which we developed as part of our prior work [Van Craeynest and Eeckhout 2011] and which is a method for quickly estimating multi-program multicore performance from single-core simulation runs. MPPM collects a profile during single-core simulation that captures a program's memory behavior as well as its phase behavior. It then employs an iterative method to model the performance entanglement between co-executing programs on a multicore processor with shared caches: the iterative method captures how per-program performance affects the amount of resource sharing, and, vice versa, how resource sharing in its turn affects per-program performance.

We refer to Figure 1 for a general overview of MPPM, and we briefly describe MPPM in the next few subsections.

2.1. Single-Core Simulation Profiling

The first step in the MPPM framework is to perform single-core simulation profiling, which collects three characteristics.

—*Single-core CPI* is the number of Cycles Per Instruction (CPI) when running the single-core workload in isolation, i.e., there are no coexecuting programs and the single-core workload has access to the entire memory hierarchy.

—*Memory CPI* is the fraction of the single-core CPI waiting for memory.

—*Stack Distance Counters (SDCs)* capture the program's temporal memory access behavior in set-associative (or fully associative) caches [Mattson et al. 1970]. We collect SDCs for each program on the LLC without cache sharing, i.e., by running the program in isolation. An SDC for an A-way set-associative cache involves $A+1$ counters, $C_1, C_2, \ldots, C_A, C_{>A}$, and is computed as follows. On each access, one of the counters is incremented. If the access is to the ith position in the LRU stack for that set, the ith counter C_i is incremented. If the cache access involves a miss, then the $C_{>A}$ is incremented.

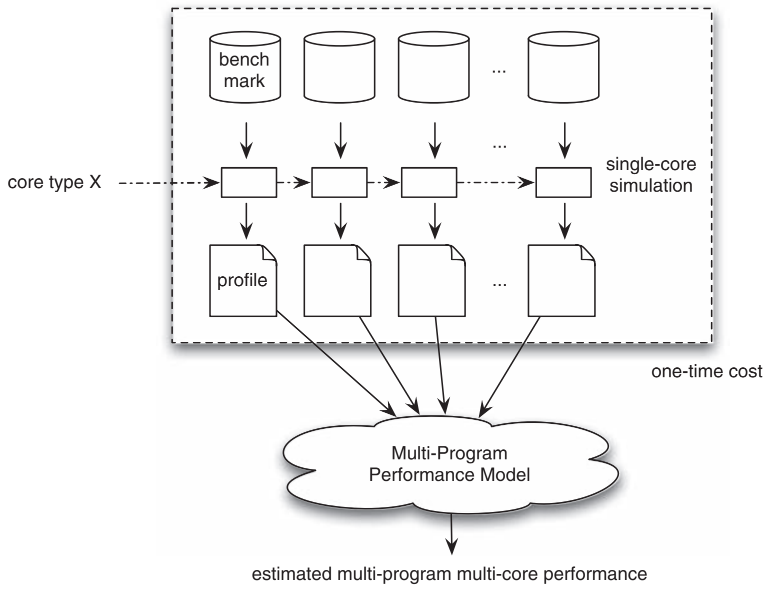

bench mark
...
core type X
single-core simulation
profile
one-time cost
Multi-Program Performance Model
estimated multi-program multi-core performance

Fig. 1. General overview of MPPM.

Each of these performance characteristics are measured on a per-interval basis. The reason is to be able to model the impact of time-varying phase behavior on resource contention in multicore processors. In our setup, we measure these characteristics for every interval of 20 million (dynamically executed) instructions.

Single-core simulation profiling runs need to be done only once for all the benchmarks and all the core types of interest. This is a one-time cost. Once we have collected the single-core simulation profiles, we can then quickly estimate performance for both homogeneous and heterogeneous multicore architectures for arbitrary workload mixes and multicore configurations.

2.2. Iterative Multicore Performance Estimation

The single-core performance characteristics mentioned in the previous section serve as input to an iterative multicore performance model, called the Multi-Program Performance Model (MPPM). The concept of the MPPM is to initially start from the single-core performance measurements and then iteratively converge on how resource contention in shared resources affects per-core performance in a multicore processor. The reason for the iterative process is the tight performance entanglement between per-core performance and resource contention, i.e., per-core performance affects the amount of resource contention, and vice versa, resource contention affects per-core performance. In order to model this tight performance entanglement, the model initially estimates the amount of resource contention assuming each program makes progress as per the single-core simulations; however, the amount of resource sharing affects per-core progress, which in its turn affects resource sharing. Hence, in the next iteration, per-core progress is adjusted to incorporate how resource contention affects per-core progress. This, in its turn, may again affect the amount of resource contention seen, which leads to the second iteration, etc. This iterative process continues until convergence.

We initialize the algorithm by assuming that all programs experience the same relative slowdown and execute at single-core speed initially. Further, we assume that all programs start at the beginning of the execution trace. Once these initial conditions are set, the iterative process starts.

At each iteration, we first determine the slowest program in the workload mix, or the program in the multi-program workload mix with the highest (estimated) multicore CPI. We determine the number of cycles it takes for the slowest program to execute L million instructions (L equals 200 million instructions in our setup), and we estimate the amount of resource sharing in the LLC over this period of time. This is done using the SDCs. The SDCs serve as input to a cache contention model that estimates the additional number of conflict misses due to cache sharing in the LLC. There exist several cache contention models [Chandra et al. 2005; Eklöv et al. 2011; Lee et al. 2008]. We use the Frequency of Access (FOA) model proposed by Chandra et al. [2005] because it is a fairly simple model and we found it accurate enough for our needs. The average memory access latency is determined by dividing the memory CPI with the number of misses. By multiplying the number of additional conflict LLC misses with the average penalty per LLC miss, we obtain an estimate for the number of lost cycles due to cache sharing for each of the programs in the multi-program mix. The lost cycles are then accounted for in the estimated per-program multicore CPI, i.e., we account for the fact that a program gets slowed down because of resource contention. In the next iteration, MPPM then takes the updated multicore CPI for each of the programs and computes resource sharing accordingly. This way, we model how resource sharing affects per-program performance and vice versa.

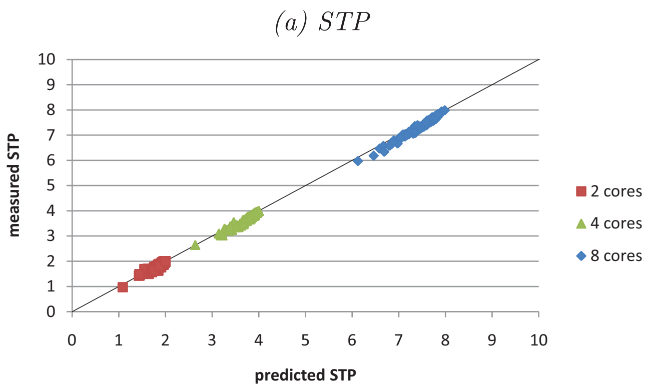
This iterative process is repeated until a stop criterion is met. Each iteration involves 200M instructions for the slowest running program, and the iterative process continues until the slowest running program in the workload mix has executed 5B instructions in total. Given that our instruction traces are 1B instructions in size, this means that the slowest program needs to iterate over its entire trace five times. Faster running programs may iterate over their trace more than five times. We found that the performance numbers converged given this stop criterion.

The output produced by MPPM is a CPI estimate for each of the programs in the multi-program mix. From these CPI estimates we can then compute System ThroughPut (STP) and average normalized turnaround time (ANTT), as we will explain later and which are metrics for system-level throughput and per-program performance, respectively.

2.3. MPPM Evaluation

Validation of the model against detailed simulation for a range of multicore configurations with 2, 4, and 8 cores and 150 multi-program workload mixes demonstrates MPPM's accuracy. (We refer to later in the article for a detailed description of the experimental setup.) Figure 2 reports scatter plots for STP (system throughput) and ANTT (average normalized turnaround time); the predicted metrics are shown versus the measured metrics. Each dot represents one multi-program workload mix. Perfect prediction would imply all dots to lie on the bisector. We observe a strong correlation between the measured and predicted performance metrics, i.e., all the dots lie around and are close to the bisector. The average error across these workload mixes equals 1.4%, 1.6%, and 1.7% for STP and 2, 4, and 8 cores, respectively; and 1.5%, 1.9%, and 2.1% for ANTT and 2, 4, and 8 cores, respectively.

Note again that MPPM itself does not involve detailed cycle-accurate multicore simulation. The process as explained above only involves "analytical" simulation in which we employ analytical models for estimating multicore performance. This yields a very fast multicore performance estimation technique: MPPM makes a multicore performance estimate in less than a second, provided that the single-core simulation runs were done beforehand.



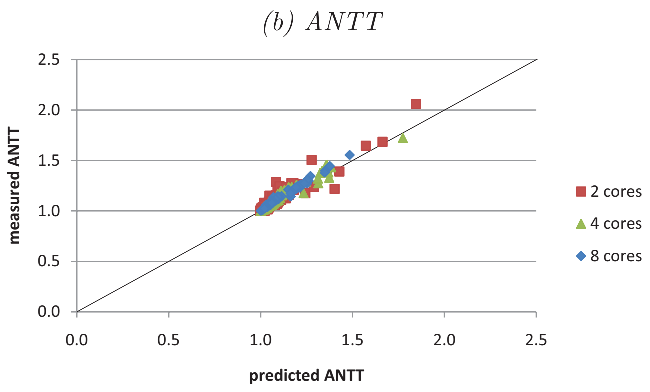


Fig. 2. Accuracy of MPPM for predicting (a) STP and (b) ANTT; measured values on vertical axis versus predicted values on the horizontal axis.

3. EXPLORING THE HETEROGENEOUS MULTICORE DESIGN SPACE

MPPM thus provides a way for estimating multicore performance based on single-core simulation results. More specifically, we leverage the MPPM framework to explore the heterogeneous multicore design space in this article. This is done as follows; see also Figure 3.

We first perform single-core simulation runs for all of the benchmarks and all of the core types of interest. In this article, we consider five core types and the SPEC CPU2006 benchmarks as our workloads. These single-core simulations need to be done only once, and enable us to explore the heterogeneous design space for arbitrary combinations of number of cores and core types, and for arbitrary job mixes. This matrix of single-core simulation results serves as input for our design space exploration.

For estimating multicore performance, we consider the single-core simulation results for the core types of interest; the core types could be diverse in case of a heterogeneous design or the same in case of a homogeneous design (step no. 1 in Figure 3). We then randomly pick N benchmarks, with N the number of cores (step no. 2), and we assign benchmarks to cores (step no. 3). As we will observe later in the article, benchmark-to-core mapping has an important impact on overall performance. In this article, unless mentioned otherwise, we map the job that benefits the most to the most aggressive core, and so forth until all jobs are mapped to cores. (We provide more details on the job-to-core mapping approach later in the article, and we find this heuristic to be close to

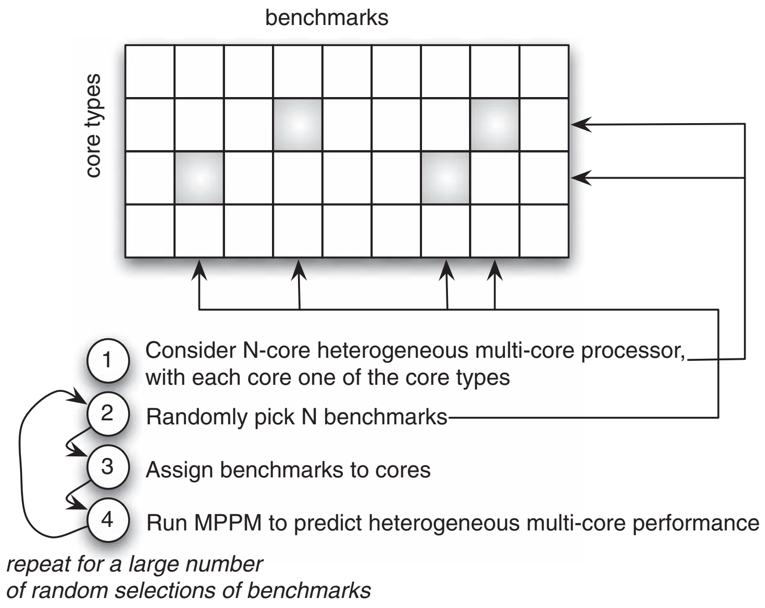

benchmarks
core types
1 Consider N-core heterogeneous multi-core processor, with each core one of the core types
2 Randomly pick N benchmarks
3 Assign benchmarks to cores
4 Run MPPM to predict heterogeneous multi-core performance
repeat for a large number of random selections of benchmarks

Fig. 3. Using MPPM for exploring the heterogeneous multicore design space.

optimal; see Section 5.6.) We then use MPPM to estimate multicore performance (step no. 4). This whole process (steps no. 2 through no. 4) is iterated for a large number of randomly chosen benchmark mixes (500 in total per experiment). The key feature of MPPM is that it is based on an analytical model that can be evaluated very quickly. One experiment takes approximately one day to complete using MPPM; this includes the one-time cost of single-core simulations for all workloads and core types, as well as computing the MPPM model. The traditional methodology using detailed architectural simulation would take more than 80 days for performing the same experiment. Put differently, MPPM enables considering a large set of job mixes in limited time, which increases confidence in the results compared to detailed simulation, which would limit the number of possible job mixes to just a few examples because of simulation time constraints.

3.1. Heterogeneous Multicore Design Space

We consider five core types in our design space exploration: 4-wide and 2-wide out-of-order cores, and 4-wide, 2-wide, and scalar in-order cores. For the out-of-order cores, we assume a 128-entry and 32-entry reorder buffer for the 4-wide and 2-wide core, respectively. We assume a core to have private L1 instruction and data caches, as well as a private L2 cache. The L1 caches are 32KB in size; the L2 caches are 256KB in size and are 8-way set-associative. The L3 cache is shared among the cores and is the last-level cache (LLC) in our setup; we vary the LLC size between 1MB, 2MB, and 4MB in our experiments, and we assume the LLC to be 16-way set-associative. All caches implement an LRU replacement policy.

The area cost models for each core type are derived from chip die photos from Intel Quad-Core Nehalem and Intel Atom processors; see also Table I. Nehalem implements 4-wide out-of-order cores, whereas Intel Atom implements 2-wide in-order cores. We empirically observed a 1 to 4 ratio in chip area between these core architectures (in the same chip technology). We also observed that one slice of 512KB LLC corresponds roughly to half an Intel Atom core or one-eighth of an Intel Nehalem core. Hence, we

Table I. Chip Area Cost Model

	#BCEs
scalar in-order core	1
2-wide in-order core	2
4-wide in-order core	3
2-wide out-of-order core	4
4-wide out-of-order core	8
512KB LLC slice	1

assign one Base Core Equivalent (BCE) [Hill and Marty 2008] to a 512KB LLC slice; 2 BCEs to a 2-wide in-order core (alike Intel Atom), and 8 BCEs to a 4-wide out-of-order core (alike Intel Nehalem). We extrapolated towards scalar in-order, 4-wide in-order, and 2-wide out-of-order cores as shown in Table I.

We consider 40 BCEs in total in our experiments. This corresponds to the chip area of the Intel Quad-Core Nehalem processor, which includes four 4-wide out-of-order cores along with a 4MB LLC. In the experiments to follow we vary the configuration of the multicore processor architecture and we consider both homogeneous and heterogeneous designs, while bounding total chip area to 40 BCEs.

Unless mentioned otherwise, we assume unlimited off-chip bandwidth; however, in the results section, we do study how limited off-chip bandwidth affects heterogeneous multicore performance.

3.2. Multicore Performance

An important distinction between this work and prior work in heterogeneous multicore architectures is that we focus on two metrics for quantifying multicore performance from two complementary perspectives; prior work primarily focused on a single metric, namely weighted speedup which quantifies performance from a system perspective only and does not take into account per-program performance. By using two metrics we quantify multicore performance when running multi-program workloads from both a system's and a user's perspective. We consider system throughput (STP) as a metric to quantify system performance, along with average normalized turnaround time (ANTT) to quantify user-perceived per-program performance. The original definition by Eyerman and Eeckhout [2008] introduced STP and ANTT assuming homogeneous multicore architectures. However, these definitions are inappropriate for heterogeneous designs. The next subsection describes the original definitions of STP and ANTT, followed by a discussion on how we extended these metrics for heterogeneous designs.

3.2.1. STP and ANTT for Homogeneous Multicores. STP measures multicore performance from a system's perspective and quantifies the accumulated progress by all the programs in the multi-program workload mix. STP equals weighted speedup proposed by Snavely and Tullsen [2000] and is a higher-is-better metric:

$$STP = \sum_{p=1}^{n} \frac{CPI_{SC,p}}{CPI_{MC,p}},$$

with $CPI_{SC,p}$ the CPI for program p when executed on a single core in isolation, and $CPI_{MC,p}$ the CPI for program p when executed on a multicore processor while being coexecuted with other jobs on the other cores. In a multicore setup with n independent jobs and cores, STP is bounded by n, i.e., STP equal to n can only be achieved if all programs achieve single-core performance on a multicore processor. Obviously, this is unlikely to happen because of resource sharing in shared caches (e.g., LLC), off-chip bandwidth, main memory, etc. So, in practice, STP is smaller than n with n the number of cores.

ANTT focuses on user-perceived performance and quantifies the average slowdown during multicore execution relative to single-core, isolated execution. ANTT is the reciprocal of the hmean metric proposed by Luo et al. [2001].

$$ANTT = \frac{1}{n}\sum_{p}\frac{CPI_{MC,p}}{CPI_{SC,p}}.$$

Ideally, a program does not experience any slowdown when coexecuted with other jobs on the multicore processor, and hence, ANTT would equal one. In practice though, coexecuting jobs affect each other's performance, and hence, ANTT is typically larger than one. Consequently, ANTT is a lower-is-better metric.

3.2.2. STP and ANTT for Heterogeneous Multicores. STP and ANTT as defined above for homogeneous multicores have no meaning when used for heterogeneous multicores. The reason is that single-core CPI ($CPI_{SC,p}$) is not well defined in a heterogeneous multicore because there are different core types, and hence, the question is which one to measure single-core CPI for. Picking different core types to measure single-core CPI for the different jobs would preclude comparing heterogeneous designs against each other. Hence, we need to agree on a single core type on which to measure single-core CPI ($CPI_{SC,p}$). In this work we arbitrarily consider the 4-wide out-of-order core as our baseline core to measure $CPI_{SC,p}$; the baseline core is assumed to have the entire cache hierarchy (including the shared LLC) to its disposal. Both STP and ANTT are then computed relative to the single-core CPI on this baseline core. In other words, STP now quantifies system throughput achieved over a single 4-wide out-of-order core, e.g., an STP of 8 means that this design achieves an 8× higher total system throughput compared to a single baseline 4-wide out-of-order core. Likewise, ANTT quantifies the average normalized turnaround time relative to a single 4-wide out-of-order core, e.g., an ANTT of 4 means that this design yields an average per-program slowdown of a factor 4× relative to a single 4-wide out-of-order core.

4. EXPERIMENTAL SETUP

We use a multicore processor simulator based on CMP$im [Jaleel et al. 2008], which is an x86 simulator built on top of Pin [Luk et al. 2005]. CMP$im is a user-level simulator and allows for simulating both single-core and multicore processor architectures. Our version of the CMP$im simulator is the one available from the Cache Replacement Championship[1]. We use CMP$im for obtaining the single-core simulation results that serve as input to the performance model. Further, CMP$im was also used in the validation experiments previously reported in Section 2.3.

We consider all the SPEC CPU2006 benchmarks with their reference inputs. All the benchmarks were compiled with the GNU C compiler version 4.3.4 and optimization level `-O2`. We use SimPoint [Sherwood et al. 2002] to pick representative simulation points of one billion instructions each.

Figure 4 shows performance in terms of average normalized IPC across all benchmarks for the five core types considered in this study as a function of the square root of the area (counted in the number of BCEs; see also Table I). This data complies with Pollack's Law [Borkar 2007] which states that core performance is proportional to the square root of the chip area.

5. RESULTS

We now explore the heterogeneous multicore design space using the methodology just described. This is done in a number of steps. We start by exploring the homogeneous

[1]http://www.jilp.org/jwac-1/.

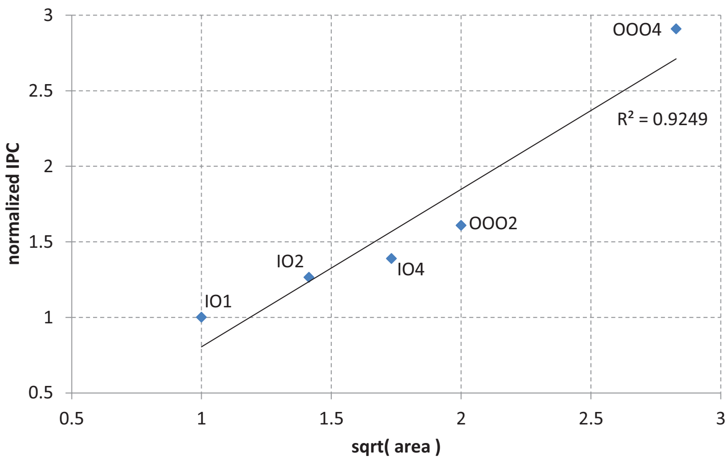


Fig. 4. Average normalized IPC for the five core configurations considered in this study as a function of the square root of the area counted in BCEs.

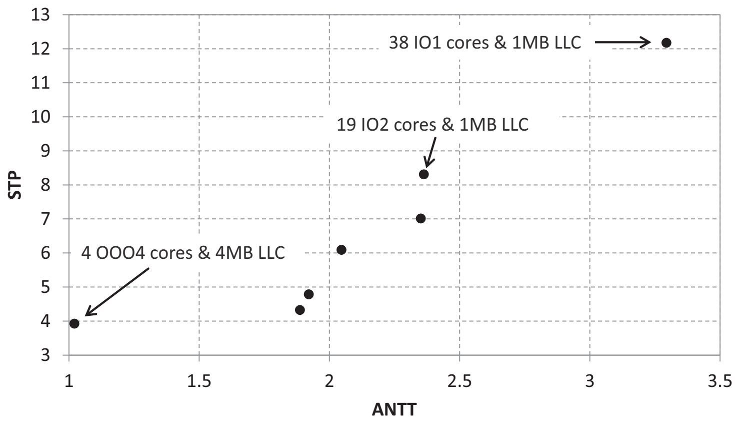


Fig. 5. Pareto-optimal homogeneous multicore configurations as a function of STP (vertical axis) and ANTT (horizontal axis).

versus heterogeneous multicore design spaces, and we explore how the number of core types affects heterogeneous multicore performance. Subsequently, we study the importance of job-to-core mapping, and how heterogeneous multicore performance is affected by off-chip bandwidth. Finally, we evaluate how sensitive heterogeneous multicore performance is with respect to LLC size, and which core types should be employed in a heterogeneous design.

5.1. Homogeneous Multicore processors

Before exploring the heterogeneous multicore processor design space, we start with exploring the homogeneous multicore design space. For this experiment, we consider all possible homogeneous multicore design points with all possible LLC cache sizes that fit in 40 BCEs; further, we assume unlimited off-chip bandwidth for now. Out of this set of possible homogeneous multicore designs, we determine the Pareto-optimal ones in terms of system throughput versus average job turnaround time. Figure 5 shows the Pareto-optimal homogeneous multicore design points as a function of STP (vertical

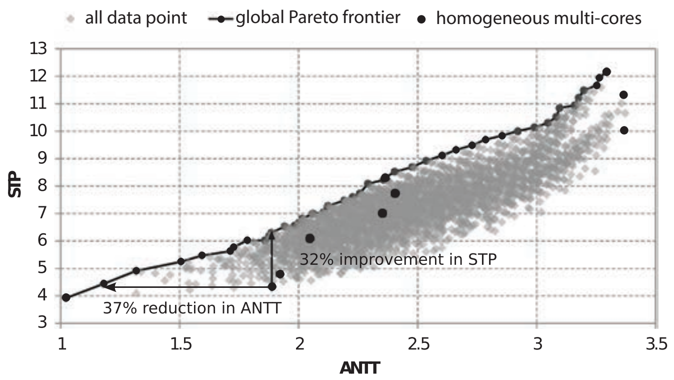


Fig. 6. Pareto frontier of multicore configurations, along with all processor configurations explored including the homogeneous design points.

axis) and ANTT (horizontal axis). The design points vary from a multicore design with four 4-wide out-of-order cores and a 4MB LLC—alike Intel Quad-Core Nehalem—with an STP of 3.92 and an ANTT of 1.02, to 38 scalar in-order cores and a 1MB LLC which achieves an STP of 12.17 and an ANTT of 3.29. In other words, the aggressive out-of-order core design yields excellent per-program performance, yet, total chip throughput is limited. Conversely, the simple in-order core design yields more than 3× higher system throughput, yet, per-program performance is also more than 3× lower.

Finding no. 0: Changing core types in a homogeneous multicore architecture yields different trade-offs in system throughput and per-program performance. In essence, simple in-order cores trade per-program performance for throughput, and conversely, aggressive out-of-order cores trade throughput for per-program performance. Mediocre cores lead to multicore design points in the spectrum between aggressive out-of-order cores and simple in-order cores. These trade-offs are well known, but it is important to restate them here in light of the exploration for heterogeneous architectures.

5.2. Pareto-Optimal Heterogeneous Multicores

Now that we have a good understanding of the homogeneous multicore design space, we move to heterogeneous architectures. We consider all possible heterogeneous multicore configurations with at most two different core types. Again, we assume all possible LLC cache sizes, a total chip area of 40 BCEs, and off-chip bandwidth being unlimited. (We consider the impact of limited off-chip bandwidth on heterogeneous multicore design considerations in the next section; the reason for considering unlimited off-chip bandwidth here is to solely focus on the impact of core types initially and study the fundamental impact of core types on heterogeneous multicore performance.) We determine the Pareto frontier as a function of STP and ANTT out of this set of heterogeneous multicore configurations. Figure 6 shows all the design points, along with the Pareto frontier and the homogeneous multicore designs (as points of reference).

Finding no. 1. Heterogeneity poses a trade-off between system throughput and per-program performance. Prior work motivated heterogeneity as a way for increasing system throughput within a given power budget [Kumar et al. 2004]. While our results confirm this finding, Figure 6 shows that heterogeneity also increases average job turnaround time, or in other words, average per-program performance degrades; compare the heterogeneous multicore configurations on the Pareto frontier against the homogeneous design point at the bottom left on the Pareto frontier (consisting of four 4-wide out-of-order cores). Similarly, heterogeneity improves per-program performance compared to a homogeneous design that achieves the highest throughput at the top right on the Pareto frontier (consisting of 38 single-issue in-order cores). Hence, fundamentally, heterogeneity trades per-program performance for system throughput, and vice versa.

Finding no. 2. Heterogeneity enables making more fine-grained performance trade-offs. Although changing the core types in a homogeneous multicore design enables trading off system throughput against per-program performance, as discussed in the previous section, heterogeneity enables more fine-grained performance trade-offs to be made. Some trade-off points in system throughput versus per-program performance can only be achieved through heterogeneity.

Finding no. 3. Some heterogeneous multicore configurations outperform specific homogeneous architectures in both system throughput and per-program performance. Heterogeneity yields a number of configurations with significantly better performance compared to homogeneous designs. For example, (see also Figure 6) through heterogeneity, one can achieve a 37% reduction in ANTT while achieving similar STP, or conversely, one can achieve a 32% STP increase at the same ANTT. The reason is that heterogeneity allows for mapping jobs to cores that are most appropriate for the job at hand.

Finding no. 4. Some homogeneous design points yield optimal performance trade-offs. Another interesting observation is that some homogeneous designs also appear on the Pareto frontier for the heterogeneous designs: the three homogeneous configurations labeled in Figure 5 with scalar in-order cores, dual-issue in-order cores, and aggressive 4-wide out-of-order cores, respectively, also appear on the Pareto frontier in Figure 6. In other words, heterogeneity does not provide a range of architectures that outperform homogeneous architectures over the entire STP vs. ANTT range. Instead, heterogeneity provides a broader range of STP vs. ANTT trade-offs, and there are many more design points on the Pareto frontier that can be obtained through heterogeneity than what can be achieved through homogeneous designs. However, particular trade-offs are best achieved through a homogeneous design. This, we believe, is an interesting and novel insight: heterogeneity, fundamentally, trades per-program performance for system throughput, and while it is true that heterogeneous architectures can outperform homogeneous architectures for some throughput versus per-program trade-off points, heterogeneous designs do not always outperform homogeneous designs, and some throughput versus per-program performance trade-offs are best achieved through homogeneous designs.

Finding no. 5. Two core types provide most of the benefits from heterogeneity. In the previous experiment, we considered at most two core types. An interesting question is whether adding additional core types improves heterogeneous multicore architecture performance above two core types. Figure 7 shows the Pareto frontier for at most two, three, four, and five core types. Interestingly, adding more than two core types does not improve performance much. The highest improvement observed in throughput and turnaround time is no larger than 6.6% and 7.9%, respectively, going from two to three core types; beyond three core types, the improvement is less than 0.3%. Hence, we conclude that two core types provide most of the benefits through heterogeneity, and three or more core types does not contribute much.

5.3. Limiting Off-Chip Bandwidth

So far, we assumed that off-chip bandwidth is unlimited. We now study the impact of limited off-chip bandwidth on heterogeneous multicore processor design. We consider a simple bandwidth model to this end. We compute the average off-chip bandwidth requirements for each program in the job mix by multiplying the number of LLC misses per instruction with the achieved per-program IPC, clock frequency, and the machine's LLC cache line size (64 bytes). The sum of the per-program off-chip

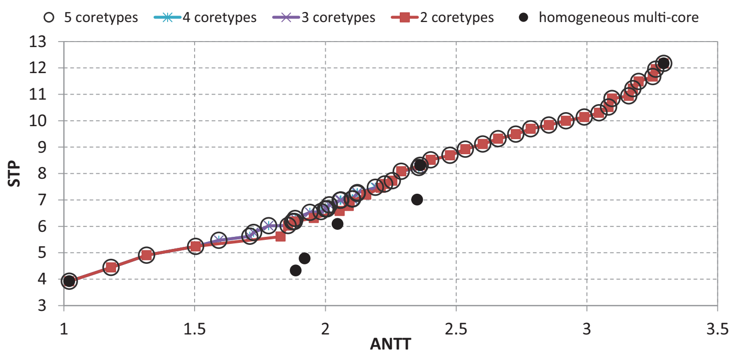


Fig. 7. Pareto frontier for heterogeneous multicore architectures with a varying number of core types.

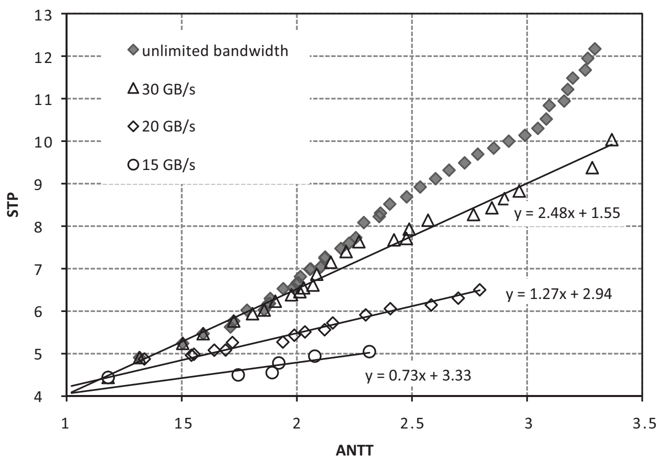


Fig. 8. Evaluating how off-chip bandwidth limitations affect heterogeneous multicore performance.

bandwidth requirements then yields the total off-chip bandwidth requirements. If the aggregate off-chip bandwidth demands exceed the maximum off-chip bandwidth, we discard the design point and we consider it to be invalid.

Figure 8 shows the Pareto frontier for unlimited off-chip bandwidth as well as for limited bandwidth at 30GB/s, 20GB/s, and 15GB/s. As expected, limited off-chip bandwidth puts a limit on the maximum achievable system throughput, e.g., compare the unlimited bandwidth curve versus the 30GB/s curve: the maximum achievable STP goes down from 12.17 to 10.03; ANTT varies across a similar range. When limiting off-chip bandwidth even further to 20GB/s and 15GB/s, we observe a decrease in achievable STP and ANTT. Further, limiting off-chip bandwidth puts a limit on how per-program performance can be traded for throughput, i.e., the range of possible design points is reduced.

Finding no. 6. When limiting off-chip bandwidth, increasing system throughput comes at the cost of a proportionally larger degradation in per-program performance. Interestingly, we observe an almost linear relationship between STP and ANTT for the heterogeneous design points on the Pareto frontier under limited bandwidth constraints; the linear fits are shown in Figure 8. Note that the slope decreases with decreasing off-chip bandwidth. This implies that if a processor designer aims at increasing system throughput, limitations in off-chip bandwidth will force the

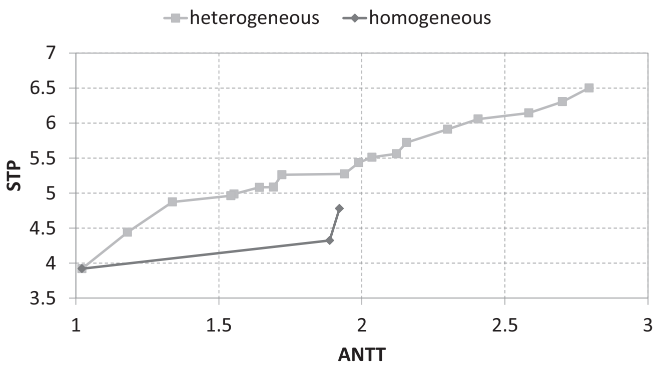


Fig. 9. Pareto frontier for heterogeneous and homogeneous multicore designs under 20GB/s off-chip bandwidth constraints.

designer to tolerate increasingly larger job turnaround times. In other words, if the goal is to improve system throughput by a given percentage, per-program performance will degrade by an increasingly larger percentage at lower off-chip bandwidths.

Finding no. 7. Highest throughput can only be achieved through heterogeneity under off-chip bandwidth constraints. It is interesting to study how homogeneous multicores fare under limited off-chip bandwidth constraints. Figure 9 shows the Pareto frontier for heterogeneous and homogeneous designs at 20GB/s of off-chip bandwidth. The key observation here is that the highest throughput cannot be achieved through homogeneity under this particular off-chip bandwidth constraint; only heterogeneity can achieve these high levels of throughput. The reason is that a large number of small cores imposes a large LLC to satisfy bandwidth constraints, which leads to suboptimal performance compared to a heterogeneous design with a few slightly more aggressive cores and a smaller LLC.

5.4. Impact of LLC Size

As observed in the previous section, off-chip bandwidth has significant impact on (heterogeneous) multicore performance. Caches are effective at reducing off-chip bandwidth pressure: cache hits do not need to go off chip, thereby saving off-chip traffic. Figure 10(a) shows the heterogeneous multicore Pareto frontier for different cache sizes while assuming infinite off-chip bandwidth. Unsurprisingly perhaps, the smallest LLC (1MB) configuration yields the highest throughput. In other words, unlimited off-chip bandwidth leads to integrating more cores and not larger caches for optimum performance.

Finding no. 8. Large LLCs yield highest throughput under off-chip bandwidth constraints. Figure 10(b) shows the heterogeneous multicore Pareto frontier with off-chip bandwidth limited to 20GB/s. We observe a very different result under limited off-chip bandwidth. Counterintuitively and surprisingly, the highest system throughput is achieved for the largest LLC (see right-hand side in Figure 10(b)). The reason is that a large LLC reduces the off-chip bandwidth pressure imposed by employing many small cores to achieve high throughput. In other words, the high-throughput designs on the right-hand side of Figure 10(b) are bandwidth-constrained, hence, they benefit from a larger LLC to reduce bandwidth pressure. Balanced system throughput and per-program performance (middle part in Figure 10(b)) is achieved by employing more (or at least a couple) aggressive big cores which impose less off-chip bandwidth traffic, and hence, a smaller LLC is optimal. However, for the designs on the left-hand side of Figure 10(a) and (b) which are optimized for per-program performance, a large LLC

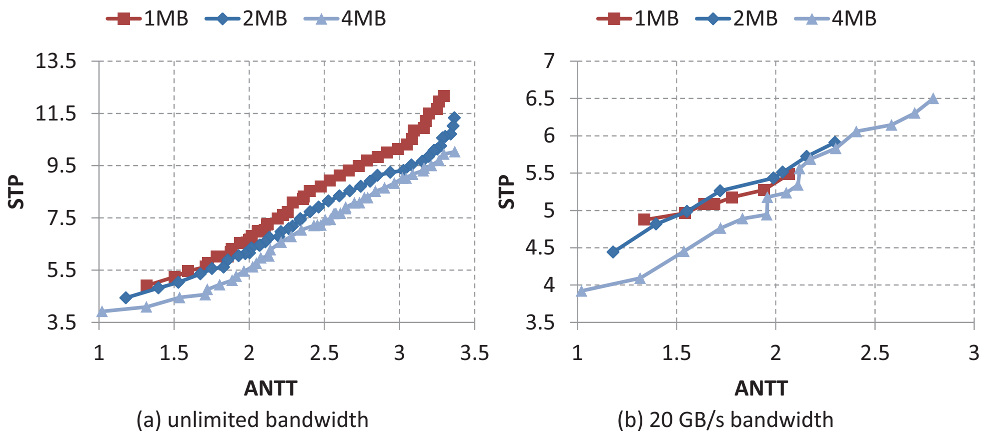


Fig. 10. Evaluating how LLC size affects Pareto-optimal heterogeneous multicore performance, assuming (a) unlimited and (b) 20GB/s off-chip bandwidth.

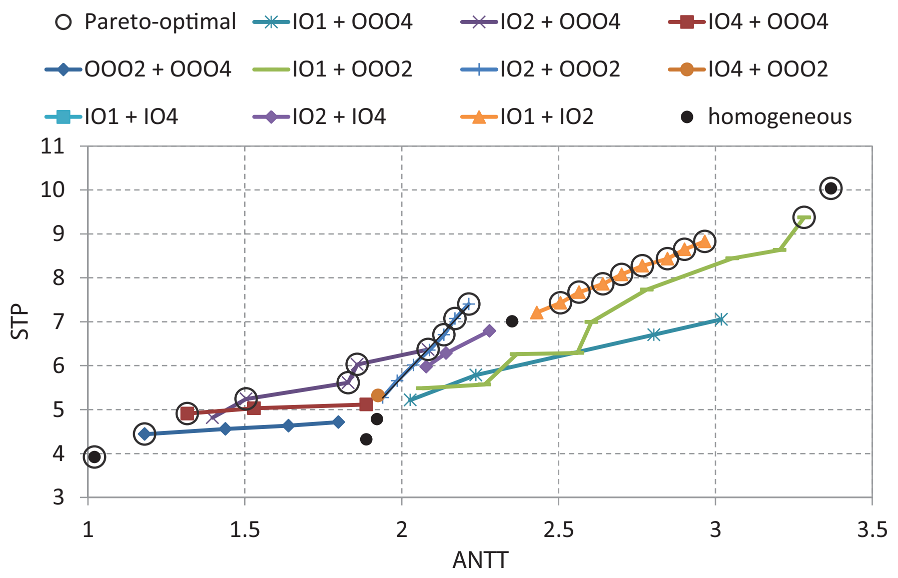


Fig. 11. Pareto frontiers for heterogeneous multicores with two core types, assuming 30GB/s off-chip bandwidth.

is optimal again. The reason is that a large LLC (along with aggressive out-of-order cores) yields the highest per-program performance. In other words, these designs are not bandwidth-constrained but optimized for per-program performance, and hence also benefit from a large LLC.

5.5. Which Core Types to Employ in a Heterogeneous Design?

As mentioned throughout the article, there is a clear performance benefit to be achieved from heterogeneity for particular performance trade-offs. An open question though is what the core types should be for optimum performance. Figure 11 shows the Pareto frontier for all possible combinations of two core types; we assume off-chip bandwidth is limited to 30GB/s. The Pareto-optimal points across all Pareto frontiers per two core types obviously corresponds to the global Pareto frontier shown in Figure 8.

Finding no. 9. Particular compositions for heterogeneity yield particular performance trade-offs, and some compositions do not yield Pareto-optimal performance. The interesting observation from Figure 11 is that the composition of a Pareto-optimal heterogeneous multicore varies along the Pareto frontier. At high throughput, a Pareto-optimal heterogeneous multicore is to be composed of single-issue and dual-issue in-order cores. This is in line with prior work which advocated simple cores for server-type throughput applications in the datacenter [Kongetira et al. 2005; Kgil et al. 2006; Lim et al. 2008; Reddi et al. 2010; Mudge and Hölzle 2010]: highest throughput is achieved using many small cores. For high per-program performance, Pareto-optimal heterogeneous multicores should employ at least one out-of-order core type. Interestingly, a heterogeneous multicore composition with two particular core types that is Pareto-optimal locally is not necessarily Pareto-optimal globally, i.e., the exact number of cores of a given type is important and determines whether the heterogeneous multicore is globally optimal. If not, there exist compositions with other core types that yield better throughput and per-program performance.

Note also that some heterogeneous multicore compositions do not yield Pareto-optimal performance; see for example heterogeneous designs with four-issue in-order and two-issue out-of-order cores, as well as heterogeneous designs with two-issue and four-issue in-order cores. This might be unsurprising and can be understood intuitively given the relatively small performance and chip area differences between these core types; see also Section 4. However, heterogeneous architectures with single-issue in-order and four-issue out-of-order cores also fall in this category. This is a surprising result because these two core types are the most extreme core types in the mix. The reason is that single-issue in-order cores call for a larger LLC to meet the off-chip bandwhich constraints. Two-issue in-order cores on the other hand put less aggregate pressure on off-chip bandwidth (because one two-issue core generates less off-chip traffic than two single-issue cores for the same chip area). As a result, single-issue in-order cores demand for a larger LLC which is suboptimal compared to having fewer mediocre (dual-issue in-order) cores and a slightly smaller LLC.

5.6. Job-to-Core Mapping

An important challenge with heterogeneous multicore architectures is how to schedule or map the jobs across the different core types in order to maximize performance. We consider four job-to-core mapping strategies.

—Optimal mapping maps jobs to cores so that overall performance is optimized. One could either optimize throughput or optimize turnaround time; here we maximize throughput (STP). The optimal mapping is obtained by exhaustively trying out all possible job-to-core mappings and picking the best one. This is an oracle and cannot be achieved in practice.

—Cache-miss-rate-based mapping maps the job with the highest LLC miss rate to the lowest-end core, the job with the second highest LLC miss rate to the second lowest-end core, etc. In other words, we map compute-intensive jobs to the high-end cores and the memory-intensive jobs to the low-end cores. The intuition is to map jobs to the core type where they would presumably benefit the most. Several previously proposed scheduling algorithms for heterogeneous architectures are based on this heuristic; see for example Koufaty et al. [2010]; Kumar et al. [2004]. We explored IPC-based mapping strategies as well, following several other prior proposals [Becchi and Crowley 2008], but obtained similar results as for cache-miss-rate-based mapping, hence, the IPC-based mapping results are omitted from the article.

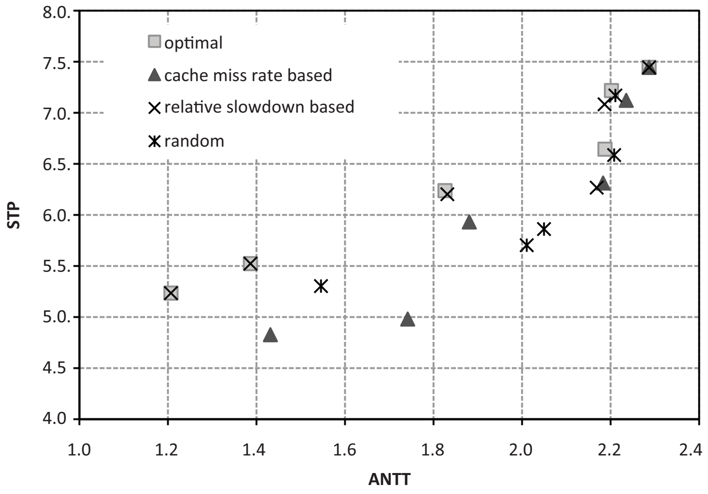


Fig. 12. Evaluating how job-to-core mapping affects heterogeneous multicore performance.

—A relative slowdown mapping assumes that it knows the relative performance for each job on each of the core types. This mapper iteratively picks the job in the job mix that would experience the largest relative slowdown from not being scheduled on the highest-end core, and maps that job to the highest-end core; the job is removed from the job mix and the core is no longer schedulable, after which the mapper picks the next job.
—Random mapping, as it says, performs a random mapping of jobs to cores.

Figure 12 compares these job-to-core mapping strategies for six heterogeneous multicore processors with two core types, namely 4-wide out-of-order and 2-wide in-order cores. Again, we consider 500 randomly chosen multi-program workload mixes.

Finding no. 10. Job-to-core mapping is both important and challenging for achieving optimum performance on heterogeneous multicore architectures. Clearly, random mapping as well as a simple heuristic such as cache-miss-rate-based mapping are far from optimal. Random mapping is oblivious to the fact that the underlying hardware is heterogeneous and, as a result, it is not surprising that random mapping does not yield optimum performance. The cache-miss-rate-based mapping strategy apparently does not have enough information about how to optimally map jobs to cores. The reason is that cache miss rate based mapping in unaware of memory-level parallelism and how misses translate into overall performance. Relative-slowdown-based mapping holds enough information for making a (close to) optimal mapping. Note though that the information needed by relative slowdown mapping is substantial as it requires the knowledge of how well jobs perform on the different core types; this may require extensive profiling which may be not be achievable in practice. Hence, we can consider this approach as an idealized mapping approach. (Note we used the relative slowdown mapping throughout the article.) We conclude from this experiment that job-to-core mapping on heterogeneous multicore systems is a nontrivial problem, that simple heuristics as presented in the literature are suboptimal, and that solving the mapping problem can yield substantial performance benefits.

5.7. Workloads

Any experimental study is bound to the workloads used in the study. In other words, some of the conclusions may be somewhat biased by the set of workloads considered.

However, we expect the more general conclusions to still hold true for other types of workloads.

Finding no. 11. Although the SPEC CPU benchmark suite comprises a fairly broad set of workloads, as for any experimental study, some of the conclusions reached in this article may be bound by the set of chosen workloads, however, we expect the overall insights to hold true across workload domains. In particular, this article identified specific heterogeneous multicore configurations to be optimal. We expect this to be workload dependent, i.e., another set of workloads is likely to yield a different set of Pareto-optimal architecture configurations. Nevertheless, the more general findings in this article, we believe, are likely to hold true for other workloads as well.

6. RELATED WORK

The design space of heterogeneous multicore architectures is huge. The weakest form of heterogeneity involves different cores only varying in clock frequency (microarchitecture and ISA is the same across the cores); this form of heterogeneity may stem from process variation which may cause cores on the same chip differing substantially in the amount of power that they consume and in the maximum frequency that they can support [Teodorescu and Torrellas 2008]. Per-core DVFS is employed in the AMD Opteron Quad-Core processor [Dorsey et al. 2007] and the Intel Montecito [McGowen et al. 2006] and enables heterogeneity by varying clock frequency per core. A stronger form of heterogeneity are the so-called single-ISA heterogeneous multicores: all the cores implement the same ISA but differ in their microarchitecture [Kumar et al. 2003]. Commercial examples are the NVidia Kal-El [NVidia 2011] and the ARM big.LITTLE chip [Greenhalgh 2011], as mentioned in the Introduction. Overlapping-ISA heterogeneous multicores feature different cores with overlapping ISAs, i.e., all cores implement the same ISA except for a small set of instructions that is unique to each core type [Li et al. 2010]. The strongest form of heterogeneity involves different cores with different ISAs and microarchitectures. Examples are CPU/GPU integration, such as Intel's Sandy Bridge [Intel 2008], AMD's Fusion [AMD 2008], and NVidia's Tegra [NVidia 2010], or accelerator-based architectures such as the IBM Cell [Kahle et al. 2005]. The remainder of this related work section focuses on single-ISA heterogeneous architectures, which is the subject of this article.

Kumar et al. [2003] were the first to propose single-ISA heterogeneous multicores. They propose thread migration during runtime and powering down unused cores to exploit the time-varying behavior of applications to maximize performance and power efficiency. In their follow-on work, Kumar et al. [2004] proposed scheduling different programs to different core types in single-ISA heterogeneous multicore architectures. They showed that scheduling programs to the most power-efficient core in a heterogeneous multicore processor can lead to substantial improvements in system throughput (weighted speedup) for static workloads and substantial reductions in job response times for dynamic workloads in which jobs come and go as they complete. In contrast to our work, Kumar et al. did not make the observation that heterogeneous architectures fundamentally trade per-program performance for throughput. Kumar et al. [2006] explore principles for designing single-ISA heterogeneous multicore architectures. They consider in-order as well as out-of-order cores, and they vary core configurations (pipeline width, number of functional units, number of rename registers, reorder buffer size, etc.) as well as cache size and associativity, while considering both area and power cost. In contrast to our work, they limit the design space to four cores only and assume no interactions among cores (no shared LLC). Further, they focus on system throughput only, and do not consider per-program performance.

Grochowski et al. [2004] study the trade-off in per-program performance versus chip throughput in a power-constrained environment. They make the fundamental observation that in order to achieve both high per-program performance and high throughput, a processor needs the ability to dynamically vary the amount of energy expended per instruction according to the amount of parallelism available in software, a technique called Energy Per Instruction (EPI) throttling. They survey four architectural techniques to do so: voltage/frequency scaling, heterogeneity, variable-size cores, and speculation control, and conclude that heterogeneous multicores along with voltage/frequency scaling are most promising to dynamically achieve high per-program performance when few threads are active and high throughput when many threads are active. Annavaram et al. [2005] experimentally evaluate the idea of EPI throttling using prototype hardware by applying clock throttling to cores in a homogeneous shared-memory processor, effectively creating a heterogeneous multicore system. They report substantial performance improvements compared to a homogeneous multicore within a given power budget while running multithreaded applications. These papers did not evaluate design trade-offs in a heterogeneous multicore processor and how these trade-offs affect per-program performance versus throughput. Furthermore, these papers argue workload phases with limited thread-level parallelism should be sped up by consuming more energy per instruction, thereby achieving high per-program performance; in this work, we find that, even under abundant numbers of independent programs and threads, heterogeneous multicores provide a trade-off between per-thread performance and chip throughput.

A substantial body work has been done on scheduling for heterogeneous multicore processors. Several proposals propose static or offline scheduling based on program characteristics [Chen and John 2009; Shelepov et al. 2009]. An obvious limitation is that static or offline scheduling does not allow for taking advantage of time-varying workload execution behavior. Other proposals employ sampling-based scheduling [Kumar et al. 2003, 2004; Becchi and Crowley 2008; Winter et al. 2010], i.e., a program is executed on different core types for a short amount of time and the system then dynamically maps the program on the most performance-/power-efficient core dynamically. Yet other proposals use heuristics such as schedule memory-intensive programs on small cores and compute-intensive programs on more aggressive cores; see for example Ghiasi et al. [2005]; Shelepov et al. [2009]; Koufaty et al. [2010]; Li et al. [2010]. Patsilaras et al. [2012] study how to best integrate an MLP technique (such as runahead execution [Mutlu et al. 2003]) into a heterogeneous multicore processor. Van Craeynest et al. [2012] propose PIE, an analytical model for steering scheduling in heterogeneous multicores, which aims at approaching relative slowdown mapping; in this article, we found relative slowdown mapping to be close to optimal mapping.

7. CONCLUSION

The single-ISA heterogeneous multicore design space is huge and there are many fundamental design choices to be made. Hence, getting insight in the design space is far from trivial. The core types may vary from simple in-order to complex out-of-order cores, and there are many possible compositions of core types and number of cores. In addition, the design is constrained by chip area limitations as well as limits in off-chip bandwidth, which leads to interesting design trade-offs while considering core types, number of cores, and LLC size. Understanding these design trade-offs is further complicated by the methodology—which is likely the reason why no such study has not been published before, to the best of our knowledge: heterogeneous multicore design exploration is complicated by the huge design space, complex interactions through shared resources such as the LLC, the very large number of possible workload mixes, and the sensitivity of the exploration to job-to-core mapping. Clearly, detailed simulation is too slow to be a useful tool for such exploratory analyses.

In this article, we used analytical modeling to explore the heterogeneous design space: analytical modeling is fast and allows for exploring many design trade-offs in limited time. The input to the analytical model is obtained in linear time in the number of core types, and workloads of interest; all possible combinations of number of cores, core types, and workload mixes can be quickly evaluated from this initial profile while taking into account interactions in the shared LLC. Further, analytical modeling facilitates focusing on the major performance trends and insights. In contrast to prior work, we also focus on both system throughput and per-program performance (prior work work focused on system throughput only) and we explore Pareto-optimal configurations.

This analysis provides a number of interesting insights. (1) While it is true that heterogeneity can improve system throughput, it fundamentally trades per-program performance for chip throughput. (2) Some homogeneous multicore configurations yield optimal performance trade-offs, however, heterogeneity enables making more fine-grained design choices, and yields better throughput and per-program performance than homogeneous designs for particular performance targets. (3) Two core types provide most of the benefits from heterogeneity and a larger number of core types does not contribute much, however, the choice of core types is critical for optimum performance and for achieving particular performance targets. (4) Limited off-chip bandwidth changes some of the fundamental design choices in heterogeneous architectures, such as the need for large on-chip caches for achieving high throughput, and per-program performance degrading more relative to throughput under constrained off-chip bandwidth. Further, while a homogeneous design with many small cores achieves highest throughput assuming infinite bandwidth, only heterogeneous designs can achieve the highest possible throughput under bandwidth constraints. (5) Job-to-core mapping is both important and challenging for heterogeneous multicore processors to achieve optimum performance.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their constructive and insightful feedback.

REFERENCES

AMD. 2008. The future is fusion: The industry-changing impact of accelerated computing. http://sites.amd.com/us/Documents/AMD_fusion_Whitepaper.pdf.

Annavaram, M., Grochowski, E., and Shen, J. 2005. Mitigating amdahl's law through EPI throttling. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 298–309.

Becchi, M. and Crowley, P. 2008. Dynamic thread assignment on heterogeneous multiprocessor architectures. *J. Instruct.-Level Parall. 10*, 1–26.

Borkar, S. 2007. Thousand core chips—A technology perspective. In *Proceedings of the Design Automation Conference (DAC)*. 746–749.

Chandra, D., Guo, F., Kim, S., and Solihin, Y. 2005. Predicting inter-thread cache contention on a chip-multiprocessor architecture. In *Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA)*. 340–351.

Chen, J. and John, L. K. 2009. Efficient program scheduling for heterogeneous multi-core processors. In *Proceedings of the 46th Design Automation Conference (DAC)*. 927–930.

Dorsey, J., Searles, S., Ciraula, M., Johnson, S., Bujanos, N., Wu, D., Braganza, M., Meyers, S., Fang, E., and Kumar, R. 2007. An integrated quad-core Opteron processor. In *Proceedings of the International Solid State Circuits Conference (ISSCC)*. 102–103.

Eklöv, D., Black-Schaffer, D., and Hagersten, E. 2011. Fast modeling of cache contention in multicore systems. In *Proceedings of the 6th International Conference on High Performance and Embedded Architecture and Compilation (HiPEAC)*. 147–158.

Eyerman, S. and Eeckhout, L. 2008. System-Level performance metrics for multi-program workloads. *IEEE Micro 28,* 3, 42–53.

GHIASI, S., KELLER, T., AND RAWSON, F. 2005. Scheduling for heterogeneous processors in server systems. In *Proceedings of the 2nd Conference on Computing Frontiers (CF)*. 199–210.

GREENHALGH, P. 2011. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7: Improving energy efficiency in high-performance mobile platforms. http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf.

GROCHOWSKI, E., RONEN, R., SHEN, J., AND WANG, H. 2004. Best of both latency and throughput. In *Proceedings of the International Conference on Computer Design (ICCD)*. 236–243.

HILL, M. D. AND MARTY, M. R. 2008. Amdahl's law in the multicore era. *IEEE Comput. 41,* 7, 33–38.

INTEL. 2008. 2nd generation Intel Core vPro processor family. http://www.intel.com/content/dam/doc/white-paper/core-vpro-2nd-generation-core-vpro-processor-family-paper.pdf.

JALEEL, A., COHN, R. S., LUK, C.-K., AND JACOB, B. 2008. CMP$im: A pin-based on-the-fly multi-core cache simulator. In *Proceedings of the 4th Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), held in conjunction with the International Symposium on Computer Architecture (ISCA)*.

KAHLE, J. A., DAY, M. N., HOFSTEE, H. P., JOHNS, C. R., MAEURER, T. R., AND SHIPPY, D. 2005. Introduction to the cell multiprocessor. *IBM J. Res. Devel. 49*, 589–604.

KGIL, T., D'SOUZA, S., SAIDI, A., N, B., DRESLINSKI, R., REINHARDT, S., FLAUTNER, K., AND MUDGE, T. 2006. PicoServer: Using 3D stacking technology to enable a compact energy efficient chip multiprocessor. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 117–128.

KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. 2005. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro 25,* 2, 21–29.

KOUFATY, D., REDDY, D., AND HAHN, S. 2010. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. 125–138.

KUMAR, R., FARKAS, K. I., JOUPPI, N. P., RANGANATHAN, P., AND TULLSEN, D. M. 2003. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the ACM/IEEE Annual International Symposium on Microarchitecture (MICRO)*. 81–92.

KUMAR, R., TULLSEN, D. M., AND JOUPPI, N. P. 2006. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 23–32.

KUMAR, R., TULLSEN, D. M., RANGANATHAN, P., JOUPPI, N. P., AND FARKAS, K. I. 2004. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 64–75.

LEE, B., COLLINS, J., WANG, H., AND BROOKS, D. 2008. CPR: Composable performance regression for scalable multiprocessor models. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 270–281.

LI, T., BRETT, P., KNAUERHASE, R., KOUFATY, D., REDDY, D., AND HAHN, S. 2010. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*. 1–12.

LIM, K., RANGANATHAN, P., CHANG, J., PATEL, C., MUDGE, T., AND REINHARDT, S. 2008. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 315–326.

LUK, C.-K., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., WALLACE, S., REDDI, V. J., AND HAZELWOOD, K. 2005. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI)*. 190–200.

LUO, K., GUMMARAJU, J., AND FRANKLIN, M. 2001. Balancing throughput and fairness in SMT processors. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 164–171.

MATTSON, R. L., GECSEI, J., SLUTZ, D. R., AND TRAIGER, I. L. 1970. Evaluation techniques for storage hierarchies. *IBM Syst. J. 9,* 2, 78–117.

MCGOWEN, R., POIRIER, C. A., BOSTAK, C., IGNOWSKI, J., MILLICAN, M., PARKS, W. H., AND NAFFZIGER, S. 2006. Power and temperature control on a 90-nm itanium family processor. *IEEE J. Solid-State Circ. 41,* 1, 229–237.

MUDGE, T. AND HÖLZLE, U. 2010. Challenges and opportunities for extremely energy-efficient processors. *IEEE Micro 30,* 4, 20–24.

MUTLU, O., STARK, J., WILKERSON, C., AND PATT, Y. N. 2003. Runahead execution: An alternative to very large instruction windows for out-of-order processors. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*. 129–140.

NVIDIA. 2010. The benefits of multiple CPU cores in mobile devices. http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf.

NVIDIA. 2011. Variable SMP – A multi-core CPU architecture for low power and high performance. http://www.nvidia.com/content/PDF/tegra_white_papers/Variable-SMP-A-Multi-Core-CPU-Architecture-for-Low-Power-and-High-Performance-v1.1.pdf.

PATSILARAS, G., CHOUDHARY, N. K., AND TUCK, J. 2012. Effiï‹~ciently exploiting memory level parallelism on asymmetric coupled cores in the dark silicon era. *ACM Trans. Archit. Code Optim. 8*.

REDDI, V. J., LEE, B. C., CHILIMBI, T., AND VAID, K. 2010. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 26–36.

SHELEPOV, D., ALCAIDE, J. C. S., JEFFERY, S., FEDOROVA, A., PEREZ, N., HUANG, Z. F., BLAGODUROV, S., AND KUMAR, V. 2009. HASS: A scheduler for heterogeneous multicore systems. *Oper. Syst. Rev. 43*, 66–75.

SHERWOOD, T., PERELMAN, E., HAMERLY, G., AND CALDER, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 45–57.

SNAVELY, A. AND TULLSEN, D. M. 2000. Symbiotic jobscheduling for simultaneous multithreading processor. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 234–244.

TEODORESCU, R. AND TORRELLAS, J. 2008. Variation-Aware application scheduling and power management for chip multiprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 363–374.

VAN CRAEYNEST, K. AND EECKHOUT, L. 2011. The multi-program performance model: Debunking current practice in multi-core simulation. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 26–37.

VAN CRAEYNEST, K., JALEEL, A., EECKHOUT, L., NARVAEZ, P., AND EMER, J. S. 2012. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 213–224.

WINTER, J. A., ALBONESI, D. H., AND SHOEMAKER, C. A. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 29–40.

Received December 2011; revised September 2012; accepted October 2012