Heterogeneous Microarchitectures Trump Voltage Scaling for Low-Power Cores

Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Ronald Dreslinski Jr.,
Thomas F. Wenisch, and Scott Mahlke
Advanced Computer Architecture Laboratory
Ann Arbor, MI, USA
{lukefahr, shrupad, reetudas, rdreslin, twenisch, mahlke}@umich.edu

ABSTRACT

Heterogeneous architectures offer many potential avenues for improving energy efficiency in today's low-power cores. Two common approaches are dynamic voltage/frequency scaling (DVFS) and heterogeneous microarchitectures (HMs). Traditionally both approaches have incurred large switching overheads, which limit their applicability to coarse-grain program phases. However, recent research has demonstrated low-overhead mechanisms that enable switching at granularities as low as 1K instructions. The question remains, in this fine-grained switching regime, which form of heterogeneity offers better energy efficiency for a given level of performance?

The effectiveness of these techniques depend critically on both efficient architectural implementation and accurate scheduling to maximize energy efficiency for a given level of performance. Therefore, we develop $PaTH$, an offline analysis tool, to compute (near-)optimal schedules, allowing us to determine Pareto-optimal energy savings for a given architecture. We leverage $PaTH$ to study the potential energy efficiency of fine-grained DVFS and HMs, as well as a hybrid approach. We show that HMs achieve higher energy savings than DVFS for a given level of performance. While at a coarse granularity the combination of DVFS and HMs still proves beneficial, for fine-grained scheduling their combination makes little sense as HMs alone provide the bulk of the energy efficiency.

Categories and Subject Descriptors

C.1.4 [**Architectures**]: Heterogeneous systems

Keywords

Heterogeneous Multicores; DVFS; Fine-Grained Architectures; Energy Efficiency

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.

http://dx.doi.org/10.1145/2628071.2628078 .

1. INTRODUCTION

Transistor scaling has historically allowed architects to build complex processor designs with ever-decreasing energy consumption. However, with the end of Dennard scaling [7], architects must seek new alternatives to improve energy efficiency [14]. A variety of proposed techniques trade off a small decrease in performance for a large reduction in energy consumption through utilizing a range of energy efficient execution modes. The most common technique, Dynamic Voltage/Frequency Scaling (DVFS) creates energy efficient modes by lowering the voltage and frequency. The second technique, Heterogeneous Microarchitectures (HMs), reduces energy by migrating execution to a more efficient, but lower performing, core microarchitecture. Traditionaly both DVFS and HMs incurred large overheads to transition between modes, limiting their applicability to coarse-grained program phases [21, 22]. However, recent efforts [19, 26, 27, 30] have drastically reduced these transition penalties, enabling switching between modes at much finer granularities.

A scheduler divides the application into numerous intervals and migrates execution to the most appropriate mode for each interval. Sub-optimal scheduling decisions reduce the realized energy savings, masking the efficiency potential of the underlying architecture. A naive approach to bound the efficiency potential of each heterogeneous architecture is to exhaustively explore *all* possible schedules. Unfortunately the complexity of this approach increases exponentially with the number of intervals and the number of modes. Emerging architectures capable of fine-grained switching further confound such analysis by drastically increasing the number of intervals in which a scheduling decision must be made. To manage this massive state space, we have developed Pareto-optimal Tradeoffs for Heterogeneity, or $PaTH$, an offline analysis tool that determines (near-)optimal scheduling decisions and calculates the Pareto-optimal tradeoff space between performance and energy savings for a specific architecture and workload.

We then use $PaTH$ to explore the design space of heterogeneous low-power core architectures to determine which architectures offer the best energy savings for a given performance level. $PaTH$ enables us to study the effects of intelligent scheduling as well as heterogeneity in low-power cores. Prior works, such as [2, 13], which relied on static analysis, required both HMs and DVFS to provide sufficient energy-efficient performance levels. Counter-intuitively, we find that fine-grained scheduling allows HMs to achieve higher energy efficincy for similar performance levels alone, effec-

tively rendering fine-grained DVFS unnecessary. Overall, this paper offers the following contributions:

- We evaluate the effect of scheduling on different forms of heterogeneity on low-power cores. We study the potential tradeoffs of traditional DVFS and HMs, as well as emerging fine-grained DVFS and HMs for single-threaded applications. Overall, our results show that HMs are able to provide higher energy savings than DVFS for a given level of performance. Additionally, we find that fine-grained scheduling with HMs alone negates the need for fine-grained DVFS.
- We further analyze individual benchmarks and show significant variation in their use of DVFS and HMs to achieve Pareto-optimal performance/energy tradeoffs. We analyze common behavioral patterns across applications and their affinity to various forms of heterogeneity.
- This analysis is possible through *PaTH*, an offline analysis tool that can evaluate Pareto-optimal tradeoffs between performance and energy for a given architecture. By relying on approximations that bound the Pareto-optimal tradeoffs within an accuracy of $\pm 2.5\%$, *PaTH* efficiently searches the entire state space of possible schedules with a tractable analysis turnaround time.

2. BACKGROUND

The most prevalent heterogeneity techniques for trading performance for energy efficiency are DVFS and HMs. Today both suffer from large transition overheads. However, researchers have proposed approaches to minimize the transition latencies for both techniques.

2.1 Dynamic Voltage and Frequency Scaling

In CMOS technologies, significant power savings are possible by lowering the voltage. However, this necessitates lowering the frequency to meet timing constraints, leading to a performance loss. In prior technology generations, the frequency and voltage scaled roughly equally. Therefore, a reduction in voltage would yield a roughly equivalent reduction in frequency (and performance) but allow a much higher overall power reduction, justifying the loss in performance. However, typically low-power cores rely on low-leakage technologies, which sacrifice voltage scaling for decreased static power consumption. This, in combination with shrinking technology nodes, means voltage reductions are requiring larger corresponding reductions in frequency (and performance), marginalizing much of the benefits of voltage scaling.

2.1.1 Coarse-Grained Approaches

Traditional DVFS designs rely on off-chip voltage regulators, built using very large filtering components and connected over long wires, requiring tens of microseconds to change the voltage [19]. To amortize the overhead of voltage transitions, the granularity of switching was limited to Operating System (OS) scheduling intervals.

For these architectures, a variety of techniques have been proposed to decide when to switch DVFS levels, either at compile time [39] or dynamically using a JIT [37], or the OS [40, 35, 15]. Additionally, there have been studies that attempt to find the theoretical best performance/energy tradeoffs for a DVFS system using a given number of voltage/frequency levels [38, 16].

2.1.2 Fine-Grained Approaches

On-chip regulators bring the voltage regulator on the same chip as the core, reducing the wire length and the need for large capacitors, allowing more rapid transitions between voltage levels. These regulators can transition across the full voltage range in less than 20ns, but currently suffer from $\sim 20\%$ power conversion efficiency losses [18, 23]. Eyerman and Eeckhout study the benefits of on-chip regulators to save energy for an individual off-chip cache miss [10].

A competing technique, *dual-rail supplies*, uses two separate off-chip regulators and allows an individual core to connect to either of the two supply rails [27, 8]. Again, this technique provides sub-20ns transitions, but at the cost of an additional $\sim 10\%$ area overhead to route the extra power rail [8]. Furthermore, this technique limits the possible voltage choices to the number of rails. Li et al. propose a controller that uses dual-rail supplies to scale down the voltage and frequency on individual L2 cache misses [25].

2.2 Heterogeneous Microarchitectures

An alternative heterogeneity technique is to use single-ISA heterogeneous multicore systems, which reduce energy consumption by switching to a more efficient core when power-hungry structures in a high performance core are underutilized [21]. Researchers have conducted extensive studies of the types of cores required for multiple workloads [22], operating system codes [28], and ILP and MLP intensive codes [31].

2.2.1 Coarse-Grained Approaches

ARM Ltd. designed big.LITTLE [12], a heterogeneous multicore system, consisting of both a high performance out-of-order core (Cortex-A15) and low power in-order core (Cortex-A7). While the A7 sacrifices $\sim\frac{1}{2}$ the performance of the A15 running Dhrystone, it consumes $\sim 6x$ less power. This combination of performance and power allow the A7 to operate $\sim 3x$ more energy efficiently than the A15. As both cores have separate L2 caches, the transition between cores and rebuilding of the cache state can require as long as 20K cycles, forcing coarse-grained switching.

The challenge of when to switch between HMs is similar to changing voltages with DVFS. Van Craeynest et al. [36] propose a model that measures CPI, MLP, and ILP to predict the performance on the inactive processor. Other techniques discover an application's core bias by monitoring stall sources [20] or building architectural signatures [33] to allow the OS to migrate the application to the most appropriate core.

2.2.2 Fine-Grained Approaches.

The first approach to achieve fine-grained HMs is to enable adaptive cores by resizing hardware structures to be more efficient [1], allowing the core to lower its power consumption with minimal impact on performance. Proposed architectures attempt to optimize individual structures, such as the IQ, ROB, LSQ, and cache sizes [3, 1, 4, 9, 24].

An alternative approach is to share structures between multiple cores, most commonly caches and pipelines. Cache sharing solutions rely on multiplexing the L1 caches between

multiple cores to eliminate much of the state transfer overheads when switching [29, 32]. Other researchers have proposed multiplexing pipeline stages between cores to achieve heterogeneity [30]. The hybrid approach combines two separate pipelines that share caches and a unified frontend. However, this hybrid approach has higher design complexity, and it increases the power consumption of the more energy efficient core by a reported $\sim 9\%$ [26].

2.3 Tradeoff Analysis

As both heterogeneity techniques tradeoff performance for energy savings, at a coarse granularity researchers have attempted to decide which technique is preferable. Grochowski et al. conclude that both DVFS and HMs are necessary for the best energy/performance tradeoffs [13]. Azizi et al. analyze the performance/energy tradeoffs for both DVFS and HMs and conclude that HMs can deliver large performance jumps, while DVFS delivers very good incremental performance changes [2], However, their study assumes a static mapping of applications to cores, neglecting the potential benefits of scheduling.

3. PARETO-OPTIMAL TRADEOFFS FOR HETEROGENEITY

One of the main challenges to performing a study of heterogeneity tradeoffs is to find an algorithm which optimally selects the best combination of HMs and/or DVFS in each region of execution, while minimizing the energy consumption subject to a performance constraint. For HMs, we consider high performance out-of-order core (**Big**) and energy efficient in-order core (**Little**). For DVFS, we consider six different voltage/frequency levels. A single core at a specific frequency is henceforth referred to as a **mode**. The Pareto-optimal mode for each region can vary depending on the performance constraint. Therefore, to maximize energy savings, we need to determine the Pareto-optimal mode for each region of the application *for all possible performance constraints*.

We approach this challenge by executing the application under all possible modes and dividing the application into independent blocks, called **quanta**, while collecting energy and performance characteristics for each. We form schedules by selecting a unique mode for each quantum. We then attempt to examine all possible schedules to determine the subset of schedules that forms an optimum performance/energy tradeoff. As this will prove intractable, we rely on an approximation technique that allows us to compute the performance/energy tradeoffs within a bounded level of error. We further include an optimization technique that allows us to prune large trees of provably non-optimal schedules without having to compute their energy and delay.

3.1 Quanta Independence

We claim that quanta can be evaluated independently of each other provided the switching overheads are sufficiently negligible. We analyze independence in the two scenarios, where consecutive quanta use the same mode (non-switching) or different modes (switching). In Section 5.3, we further evaluate the effects of switching overheads.

Independence: Given two sequential quanta, i and $i+1$, x_i and x_{i+1} represent the core/voltage modes for quanta i and $i+1$, respectively. The choice of x_{i+1} is *independent* of x_i if $Energy(x_{i+1}|x_i)$ and $Delay(x_{i+1}|x_i)$ are constant for all x_i, i.e. given any mode, x_i, for quantum i, the resulting energy and delay for mode x_{i+1} in quantum $i+1$ does not change.

For fine-grained architectures, a quantum is assumed to consist of 1K committed instructions. We argue this length represents the lower bound at which the overheads of switching modes are negligible, allowing quanta to be evaluated independently in fine-grained architectures. For coarse-grained architectures, we assume a quantum length of 10M instructions.

In the non-switching case ($x_i = x_{i+1}$), there is no mode change so there is no departure from the behavior of the original program run, i.e. no overheads for switching modes. Any inter-quanta interactions, e.g., overlapped cache miss or updated branch prediction, are accounted for in the existing execution trace.

For the switching case ($x_i \neq x_{i+1}$), the claim of independence relies on being able to switch modes with minimal overheads. HMs are typically designed to share caches and pipeline stages in an attempt to minimize switching overheads [26]. DVFS approaches rely on either moving the power supply on-chip or building multiple off-chip power supplies [19, 25].

As switching can potentially occur as frequently as every 1K instructions, the effects of pipeline interactions need to also be considered. For example, if an out-of-order core executes two consecutive quanta, a load miss in the second quantum may be partially overlapped by another miss in the previous quantum. However, if a switch occurs, this overlap may not occur, and the load miss in the second quantum will incur the full miss latency. These interactions were studied in greater detail in our prior work which concluded that for fine-grained architectures, the effects of these interactions were negligible provided the quantum length does not shrink below 1K instructions [26].

The claim of independence is easy to illustrate for traditional coarse-grained architectures, which were designed to rely on large quanta sizes to ensure negligible switching overheads. These architectures have transition latencies in the tens of thousands of cycles resulting in negligible impact when contrasted with the millions of cycles required for the quanta's computation.

3.2 Exhaustive Schedules

While executing the instructions in a quantum, a core incurs some delay and consumes some energy. These values differ between core/voltage modes due to their different implementations. Example performance and energy characteristics for three quanta of a fictional big.LITTLE system are shown in Table 1. For the sake of simplicity, no voltage scaling is shown. A schedule is formed by making a unique mode choice for each quantum.

Schedule: $X = \{x_1, ..., x_n\}$, an ordered mapping to a single mode, x_i for each possible quantum, i.

The easiest approach to finding the lowest energy consumption for a given level of performance is to exhaustively compute all possible schedules, calculate their resulting energy and delay characteristics, and find the lowest energy schedule for each delay. Table 2 is an exhaustive list of all possible schedules of the example in Table 1. For this example, we assume negligible switching overhead. The delay for schedule {B,L,B} is calculated by summing the delay

	Quantum 1		Quantum 2		Quantum 3	
Core	Delay	Energy	Delay	Energy	Delay	Energy
Big	10 ms	50 mJ	20 ms	60 mJ	30 ms	60 mJ
Little	20 ms	20 mJ	40 ms	50 mJ	35 ms	40 mJ

Table 1: Example Characteristics. Delay and energy characteristics for a fictional big.LITTLE system over three quanta of 10M instructions each.

Number	Schedule	Delay (ms)	Energy (mJ)
1	{B,B,B}	60	170
2	{B,B,L}	65	150
3	{B,L,B}	80	160
4	{B,L,L}	85	140
5	{L,B,B}	70	140
6	{L,B,L}	75	120
7	{L,L,B}	90	130
8	{L,L,L}	95	110

Table 2: Exhaustive Schedule Characteristics. Delay and energy characteristics for *all possible schedules* of the fictional Big (B) and Little (L) cores over the quanta of Table 1.

for choosing the Big core for quantum 1 ($10ms$), the Little core for quantum 2 ($40ms$), and the Big core for quantum 3 ($30ms$), yielding a total of $80ms$. The energy is computed similarly.

3.3 Pareto Frontier

Using exhaustive schedules we can compute the best schedule, and corresponding energy consumption, for a given performance level. Given an exhaustive list of schedules and their corresponding energy and delay characteristics, we can combine all Pareto optimal schedules to form the Pareto frontier of the energy/delay tradeoff.

Pareto Optimal Schedule: A schedule $X = \{x_1, ..., x_n\}$ is a *Pareto Optimal Schedule* if and only if there exists no schedule, $Y = \{y_1, ..., y_n\}$, over the same quanta, such that $X \neq Y$, $Energy(Y) < Energy(X)$, *and* $Delay(Y) < Delay(X)$.

Figure 1 illustrates the characteristics of the schedules of Table 2. The point (90 ms,130 mJ), formed by the schedule {L,L,B}, is made non-Pareto optimal by the point (75ms, 120mJ), formed by {L,B,L}. Together the Pareto optimal points form the Pareto frontier, which represents the best possible tradeoffs between energy and delay.

3.4 Pareto Optimal Regions

Pareto frontiers can effectively express the best energy-performance tradeoffs but they require exhaustively exploring all possible schedules. The computational complexity required to compute all schedules is n^q, where n is the number of possible core modes and q is the number of quanta. As further described in Section 4, we consider 12 possible core modes and 1M quanta. The number of possible schedules quickly becomes too large to exhaustively search the exact Pareto frontier.

One of *PaTH*'s main contributions is an algorithm to produce the Pareto Optimal tradeoffs within a bounded level of error while still exploring the entire design space in a tractable amount of time. Our approach is divided into four

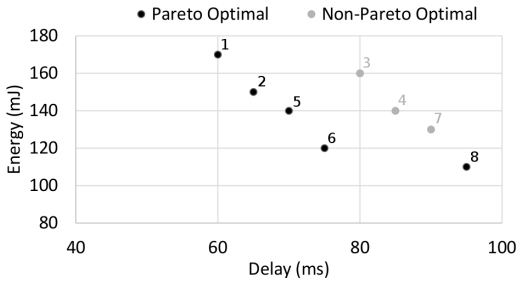

Figure 1: Pareto Frontier of Schedule Characteristics. The resulting energy and delay characteristics for all possible schedules of Table 2. Some schedules *both consume more energy and require more delay* than others, (i.e. are up and to the right), making them *non-Pareto optimal schedules*. The remaining Pareto optimal schedules form the Pareto frontier.

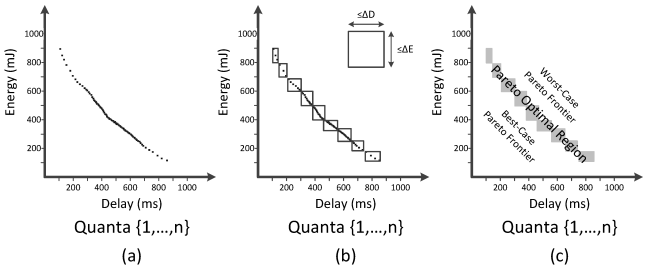

Figure 2: Approximating the Pareto Frontier. When the segment's Pareto frontier is complete (a), it needs to be approximated to allow merging with another segment's frontier (a). This is accomplished by grouping all schedules that differ by $\leq \Delta E$ and $\leq \Delta D$ (b). These schedules are then approximated by both a best-case schedule, with the least energy and delay possible for all encompassed schedules and worst-case schedule, with the highest energy and delay possible (c). Together these approximate schedules form the best-case and worst-case frontiers. The exact Pareto frontier must lie between them in the Pareto optimal region.

steps: (1) group the execution into analyzable *segments*, where each segment consists of many quanta, (2) exhaustively compute the Pareto frontier within a segment, (3) approximate all segments by introducing a bounded amount of error, and (4) convolve segments together to form the overall Pareto frontier.

By using a divide-and-conquer approach, we can compute the Pareto frontier in parallel by computing Pareto frontiers for sequential segments of the application simultaneously and convolving them to form the overall frontier. Convolution, however, requires summing the energy and delay of each schedule on the first segment's Pareto frontier (for quanta $\{1, ..., n\}$) with each schedule on the second segment's Pareto frontier (for quanta $\{n+1, ..., m\}$). The quadratic time complexity of this operation again yields an intractable problem. Therefore, before combining each partial frontier, we first use *approximation* to reduce the number of schedules that must be convolved in order to make the computation manageable.

To approximate each segment's Pareto frontier while maintaining a bounded error, we replace many schedules that

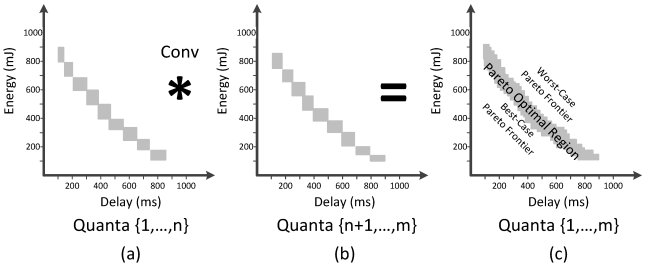

Figure 3: Merging Two Pareto Frontiers Segments. To combine quanta {1,..,n} with {n+1,..,m}, the approximated frontier from Figure 2 (a) must be convolved with the succeeding segment (b). This results in a combination of all possible best and worst-case schedules of the first segment with all possible best and worst-case schedules of the second segment, resulting in a Pareto optimal region for quanta {1,..,m} (c). The schedules that result from the convolution will bound the true Pareto frontier in the Pareto optimal region. For our analysis, we bound the total error introduced by approximation and merging to ±2.5% of the true Pareto frontier.

have similar energy *and* delay with two approximated schedules, as illustrated in Figure 2. We first select the highest energy schedule, and select nearby schedules until we meet the bound on either the energy *or* the delay error. We then replace all encompassed schedules with two approximated schedules, a best-case and worst-case. The best-case schedule consists of the least energy and least delay of any encompassed schedule. Note that least energy typically comes from a different schedule than the least delay. This schedule is Pareto optimal to all encompassed schedules. The worst case schedule is similar, but with the worst energy and delay, implying all encompassed schedules are Pareto optimal to the worst-case schedule. Together, these two schedules bound the actual Pareto frontier between them in the **Pareto optimal region**.

As illustrated in Figure 3, when combining two segments, we convolve all the best and worst-case schedules for the first segment with all best and worst-case schedules of the second segment. Hence, the combined Pareto frontier comprises two curves, representing the upper and lower bounds of the Pareto optimal region within which the true frontier must lie. By limiting the error introduced during the approximation phase such that the difference between the worst case frontier and the best case frontier is sufficiently small, we achieve a bounded estimate of the true Pareto frontier to an accuracy of ±2.5%. Our approach is a departure from traditional approaches which either optimize locally within a segment but do not combine them, [19, 39], or rely on approximations to combine segments but do not provide an overall error guarantee, [38].

3.5 Early Schedule Pruning

Rather than exhaustively computing all possible schedules within a segment to form the Pareto frontier, we can utilize early schedule pruning to reduce the number of schedules that must be generated. This optimization takes advantage of independence to detect non-optimal sub-schedules before all subsequent schedules are enumerated, removing unnecessary computation without introducing any error.

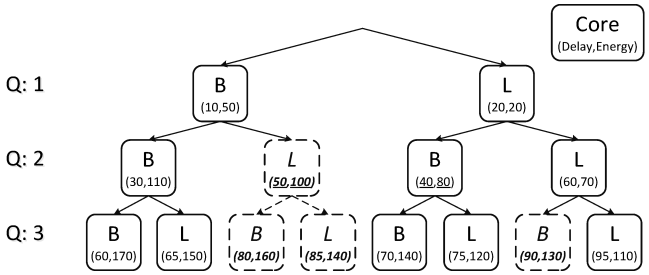

Figure 4: Early Scheduling Pruning Optimization. Once a sub-schedule becomes non-Pareto optimal (illustrated with a dashed border), all subsequent schedules will remain non-Pareto optimal and need not be considered. For example, the schedule {B,L} is made non-Pareto optimal by schedule {L,B}. Therefore all schedules based on {B,L} will never be Pareto optimal and need not be computed.

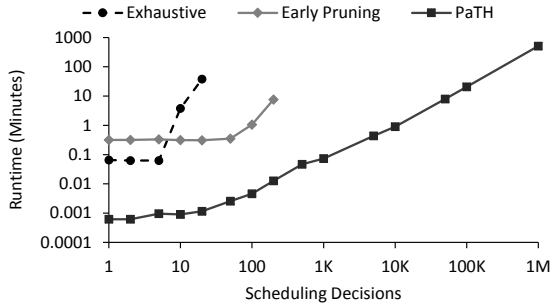

Figure 5: Runtime Overheads. Exhaustive computes all possible schedules, Early Pruning prunes non-optimal sub-schedules early. *PaTH* uses bounded approximations, and can analyze much larger scheduling windows. Scheduling decisions represent 1K and 10M instructions for fine-grained and coarse-grained approaches, respectively.

Non-Optimal Sub-schedule: Any schedule, $Z = \{x_1, ..., x_n, z_{n+1}\}$, that includes a *non-optimal sub-schedule*, $X = \{x_1, ..., x_n\}$, is also *non-optimal* as there must exist a sub-schedule, $Y = \{y_1, ..., y_n\}$, such that there exists a schedule, $Z^* = \{y_1, ..., y_n, z_{n+1}\}$, that yields $Energy(Z^*) < Energy(Z)$ and $Delay(Z^*) < Delay(Z)$.

Let us again consider the quanta from Table 1. Figure 4 illustrates the formation of schedules as a tree. The first level contains two sub-schedules, {B} and {L}, for a Big or Little core. The second level contains four sub-schedules, and the third has eight complete schedules. However, note that if we compute the energy and delay for all sub-schedules on the second level, we observe that the schedule {B,L} is made non-Pareto optimal by the schedule {L,B}. Any schedule that begins with {B,L} will always be non-Pareto optimal by a equivalent schedule that begins with {L,B}. Therefore schedules derived from the {B,L} sub-schedule need not be computed as they will never be on the Pareto frontier.

3.6 Runtime Overheads

Figure 5 illustrates the runtime overheads associated with forming the Pareto optimal tradeoffs for *gcc* with various approaches. The runtime measurements for *PaTH* were collected on an Intel 2.2GHz 8-core machine with hyper-

Architectural Feature	Parameters
Big Core	3-Issue Out-Of-Order 2048 Entry BTB Tournament Predictor 15 stage pipeline 48 ROB entries 82 entry register file
Little Core	2-Issue In-Order 512 Entry BTB 2-Bit Predictor 8 stage pipeline 32 entry register file

Table 3: Experimental coarse-grained core parameters

Architectural Feature	Parameters
Frontend	3-Issue 2048 Entry BTB Tournament Predictor
Big Backend	3-Issue Out-Of-Order 15 stage pipeline 48 ROB entries 82 entry register file
Little Backend	2-Issue In-Order 8 stage pipeline 32 entry register file

Table 4: Experimental fine-grained core parameters

threading. The Exhaustive approach computes all solutions to form the exact frontier while Early Pruning computes an identical frontier, but can eliminate non-optimal sub-schedules to reduce computation overhead. *PaTH* relies on bounded approximations and parallelization to scale to a much higher number of decisions.

The large overhead at the beginning of both Exhaustive and Early Pruning is due to memory pre-allocation. While Exhaustive only computes the Pareto frontier once, Early Pruning must compute the Pareto frontier for each quantum, causing higher overheads initially before the cumulative benefits of early pruning become effective. However, neither approach can tractably analyze more than 1K decisions. *PaTH*'s bounded approximations and parallelism allow it to analyze 1M decisions, or 1B instructions for fine-grained approaches, in a little over 8 hours.

3.7 Multi-Threaded Extension

While we have relied on *PaTH* to evaluate single-threaded applications, the tool can also optimize multi-threaded applications. When each core is optimized individually knowledge of thread interactions is lost, potentially wasting energy savings. *PaTH* can also optimize both *within* a core as well as *across* cores. If given c cores and q quantums, rather than selecting one mode for the quantum it must select c modes, one for each core. However, this increases the computational complexity from n^q to $(cn)^q$. This increase in computational complexity requires additional optimization steps in *PaTH*, which are left for future work.

4. METHODOLOGY

As the goal of this work is to determine the relative benefits of competing forms of heterogeneity, we model the power and performance for fine-grained DVFS and HMs, as well as their combination. Additionally, we model more traditional coarse-grained equavalent systems for comparison.

Memory	Size	Access Cycles	Latency
L1 Instruction Cache	32 KB	1 cycle	-
L1 Data Cache	32 KB	2 cycles	-
L2 Cache	1 MB	-	7.5ns
Main Memory	1024MB	-	40ns

Table 5: Experimental memory system

	28nm LP-FDSOI
Voltage (V)	Frequency (MHz)
1.1	2000
1.0	1800
0.9	1500
0.8	1200
0.7	900
0.6	600

Table 6: Realistic DVFS Levels. Achievable frequencies for a given voltage domain for *28nm Low Power - Fully Depleted Silicon on Insulator*. Data taken from [11].

4.1 Architecture Modeling

For fine-grained HMs, we mimic the architecture proposed by our prior work which relies on a single core with two separate backends, the Big (out-of-order) and Little (in-order) [26]. These multiplex access to shared L1 caches, fetch engine, and branch predictor. We require that only one backend is active at a time. Additionally, the Little backend now must access structures that were designed for a higher performance Big backend. The power implications of this architecture are further discussed in Section 4.2.

We choose to model both ideal fine-grained DVFS as well as *dual-rail* DVFS. Ideal DVFS (or just DVFS), is capable of utilizing the full voltage/frequency range of the core at a fine granularity, but *neglects the on-chip regulator conversion losses* [19]. The Dual-Rail (DR) model allows the core to rapidly transition between separate off-chip voltage supplies. As additional voltage rails incur area and/or metal layer overheads, we assume a *limit of two voltage/frequency modes* [8].

At a coarse granularity, we use separate caches and frontends to create a system similar to ARM's big.LITTLE. This system is capable of utilizing both microarchitectures and the full frequency range. However, it is limited to only being able to switch as often as every 10M instructions.

To model performance, we use the gem5 simulator [5]. We use full system simulations for 1B instructions (after fast forwarding for 2B instructions) using the ARM ISA running on Ubuntu 11.04. We evaluate the potential core modes using the SpecInt 2006 benchmarks compiled using gcc with the -O2 optimization flag. The microarchitectural parameters for the coarse-grained and fine-grained core models are given in Tables 3 and 4 respectively. The memory system latencies are shown in Table 5. We assume the frequency of both the L1 instruction and data caches scale with the core, resulting in constant access cycles, while the L2 cache and Main Memory are maintained at a constant frequency, resulting in constant access latency.

Model	Core	Perf. (MIPS)	Avg. Power (mW)	Efficiency (MInsts/J)
Coarse	Big	1986	2056	965
	Little	1191	369	3227
Fine	Big	1986	2081	954
	Little	1230	393	3129

Table 7: Modeling Baselines. The performance, power, and energy efficiency characteristics of our modeled architectures.

4.2 Power Modeling

Rather than relying on a simple scaling equation, our approach relies on realistic voltage/frequency modes for a modern technology. As such, we use published voltage and frequency levels for an ARM A9 processor using a cutting-edge process technology, 28nm Fully Depleted Silicon On Insulator (FDSOI) [11]. Table 6 gives the specific voltage and frequency levels used in this study. *The voltage levels that occurred most frequently in the schedules generated by PaTH, 1.1V and 1.0V, form the supplies for the DR systems.*

We model core power consumption using McPAT [34]. To determine the optimum performance/energy tradeoffs more accurately, we compute the energy consumption for each quantum individually. Similarly to [6], we extend McPAT to save CACTI models and re-use them for each quantum, and augment it to allow non-standard voltages. Finally, we update McPAT's internal scaling models to more accurately reflect current manufacturing technologies by using ITRS projections from 2012 to add the 28nm FD-SOI technology node [17].

For *coarse-grained* approaches, we assume a perfect transition and that inactive cores *are power-gated* when not in use. For fine-grained ideal DVFS, we assume a perfect on-chip regulator [19]. Based on [25], when raising voltages we assume execution continues at the original frequency until the voltage stabilizes, whereupon execution switches to the higher frequency. When lowering voltages, execution switches to the lower frequency before gradually lowering the voltage to match. Similarly to [26], we assume that, due to the rapid switching between microarchitectures, *fine-grained* HMs *are not power-gated*, and can only save energy through clock-gating. This implies both backends will be dissipating static energy even though only one is active.

Additionally, when using multiple backends, the shared structures must be provisioned to support the highest performance backend, effectively over-provisioning for the more energy efficient backend, reducing it's overall energy efficiency. The over-provisioning results in a Little backend whose power consumption is higher than a custom designed energy-efficient core. The fine-grained HMs must rely on operating in energy efficient mode more frequently to overcome the reduced energy efficient mode savings.

4.3 Architecture Baselines

The performance, average power, and energy efficiency for the baseline architectures for the SPECINT2006 benchmarks are given in Table 7. Note that, due to an over-provisioned frontend, the fine-grained Little achieves a slightly higher performance than it's coarse-grained counterpart. Secondly, note that both fine-grained architectures have both a higher average power and lower efficiency than their coarse-grained counterparts due to overheads of sharing architectural components and increased leakage energy of the inactive backend. As a brief comparison, Table 8 illustrates the relative differences between microarchitectures for both our coarse-grained system and ARM's when running Dhrystone [12].

System	Evaluation	ΔPerformance	ΔEnergy
Industry	ARM A7/A15	1.9x	3.5x
Modeled	Gem5+Mcpat	2.09x	3.01x

Table 8: Model Comparison. A comparison of the energy and performance differences between our modeled coarse-grained HMs and an industrial equivalent [12].

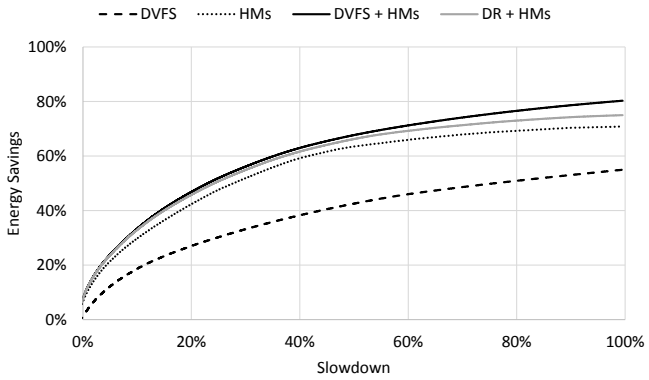

Figure 6: Pareto Frontier for Fine-Grained Switching. Note that HMs achieve more energy savings than DVFS. Additionally DVFS + HMs and DR + HMs achieve only modest improvements to HMs.

5. TRADEOFFS IN HETEROGENEITY

Using *PaTH*, we can now study the performance/energy tradeoffs between different forms of heterogeneity. Section 6 provides more in-depth analysis on how program characteristics affect the tradeoff.

We study four different fine-grained **heterogeneous configurations**: only ideal voltage heterogeneity (DVFS), only microarchitectural heterogeneity (HMs), a combination of the two (DVFS + HMs), and a combination of HMs and a dual-rail voltage system (DR + HMs). The DR design utilizes only two voltage/frequency modes. The details of these configurations are further discussed in Section 4.

5.1 Fine-Grained Comparison

Figure 6 illustrates the Pareto frontier of energy savings and slowdown for each fine-grained heterogeneous configuration. The slowdown and energy savings are normalized to the performance and energy of the highest performance microarchitecture (Big) at the highest frequency(2GHz), where a 100% slowdown indicates a doubling of the runtime. To interpret the tradeoff, observe the energy savings for each heterogeneous configuration at a specific slowdown. Higher lines indicate better configurations, as they achieve higher energy savings for a given slowdown.

Figure 6 illustrates that HMs are able to achieve higher energy savings for a given level of slowdown than DVFS regardless of the slowdown level. Analysis of the microar-

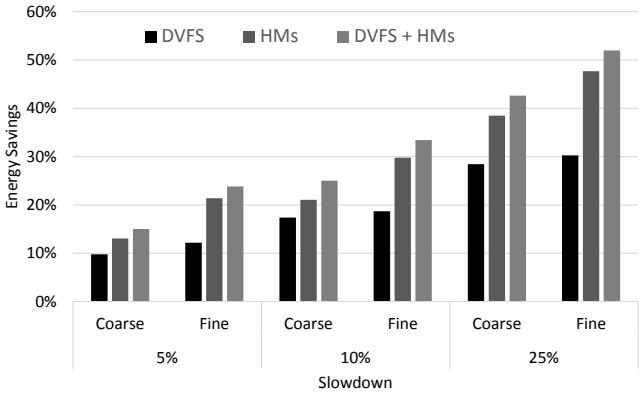

Figure 7: Impact of Switching Granularity. At smaller slowdowns, the benefits of fine-grain switching are more apparent. Also, HMs achieve better tradeoffs than both coarse-grained DVFS + HMs and fine-grained DVFS.

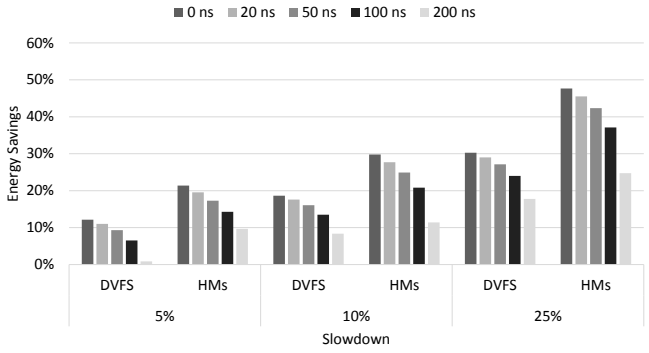

Figure 8: Impact of Switching Overheads Note that even experiencing a 100ns delay, HMs are still able to achieve better tradeoffs than idealistic DVFS.

chitectures helps confirm this observation. DVFS can only operate a complex architecture slower, which only yields its maximum effectiveness when stalled on memory-bound phases. However, HMs can switch to a simpler architecture (with a shorter pipeline) while maintaining the same frequency. Therefore HMs are able to improve efficiency under more phases with poor performance, such as low ILP or high branch misprediction levels, as well as memory-bound phases. Figure 6 indicates that the combination of DVFS + HMs offers an additional 3-4% energy savings over only HMs until about 60% slowdowns. *Thus HMs, rather than DVFS, provide more opportunities to switch to a more energy efficient mode without suffering significant performance loss.* Please see Section 6 for more in-depth analysis based on program behavior.

5.2 Switching Granularity

Figure 7 compares the tradeoffs for each heterogeneous configuration with different switching granularities, both fine-grained (1K instruction quanta) and coarse-grained (10M instruction quanta). Here we observe that at coarse switching granularities, DVFS saves 10%, 17% and 28% of the energy while HMs delivers 13%, 21%, and 38%. Finally, with DVFS + HMs, energy savings of 15%, 25% and 43% are possible.

Focusing on the benefits of fine-grained switching, we observe that DVFS provides an additional 2% energy savings for a given level of performance over coarse-grained DVFS. However HMs are able to save 7-10% more energy than coarse-grained HMs. The minimal improvement of fine-grained DVFS is caused by the nature of the phases it targets. L2 misses, where DVFS garners it's primary benefits, tend to occur in bursts, allowing them to be captured by both coarse and fine-grained approaches. However, fine-grained HMs are able to capture short phases of multiple branch mispredicts or low-ILP that cannot be captured by coarse-grained HMs or DVFS. These additional capabilities allow *HMs to benefit more from fine-grained switching than DVFS.*

Focusing on a 5% slowdown, we observe that coarse-grained DVFS + HMs allow 15% energy savings, a 3x energy savings return for the performance loss. Fine-grained HMs alone are able to achieve a 21% energy savings. Finally, using a fine-grained DVFS + HMs system, the core is able to save 25% of the energy consumption with a 5% slowdown, a 5X energy savings return for the performance loss. *Fine-grained architectures are able to save 10% more energy than the best coarse-grained approach.*

5.3 Switching Overheads

Thus far in our analysis we have neglected the overheads of switching. However, switching as frequently as every 1K instructions could have an impact on the tradeoff. To model the overheads of switching, we augment *PaTH* to track the number of switches for each schedule and then factor in various levels of fixed overhead per switch. As discussed in Section 4.2, we assume that fine-grained DVFS systems switch between different levels without any execution stalls, but suffer the power consumption for the highest voltage during the transition. For transitions between fine-grained HMs, we assume the execution halts for a fixed number of cycles, and during that delay both microarchitectures are consuming their average power.

Figure 8 illustrates the results of factoring in delays of various lengths. The achieved energy savings are shown with 0 (ideal), 20, 50, 100, and 200 ns delays. As the exact transition latency is design dependent, we have included a wide range of latencies. Kim et al. developed an on-chip voltage regulator capable of scaling over the full voltage range in 20ns [18]. Lukefahr et al. report a fine-grained HM that incurs less than 1% overhead for switching at granularities of 1K instructions, indicating a less than 20ns switching delay.

As shown in Figure 8, at a 5% slowdown, HMs suffer a 2%, 4%, 7%, and 12% drop in energy savings for 20, 50, 100, and 200 ns latencies. Similarly DVFS suffers a 1%, 3%, 6%, and 11% drop. The observation is that even with a 100ns latency cost per switch, HMs are still able to achieve higher energy savings than ideal DVFS.

5.4 Leakage Overheads

For fine-grained HMs we assume the inactive backend is clock-gated but not power-gated. Therefore, another major variable in the performance/energy tradeoff for these architectures is the leakage energy incurred by the inactive backend. When in Little mode, the leakage energy of Big adds extra energy overheads to Little. Figure 9 illustrates the impact on energy savings when we increase the assumed leak-

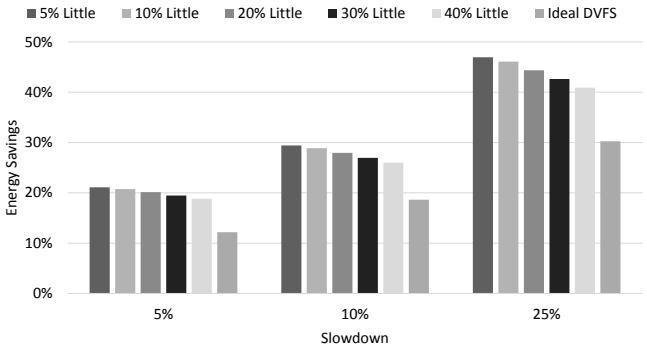

Figure 9: Impact of Leakage Energy Overheads for Fine-Grained HMs. Each bar illustrates the energy savings as Big's leakage energy adds overheads to the total power in Little mode. Here, even with a 40% power overhead from Big's leakage, fine-grained HMs still achieve better performance/energy tradeoffs than ideal DVFS.

age power overheads of Big relative to Little's total power. Note the Little backend adds insignificant leakage while the Big backend is active. Different Pareto-optimal schedules result as leakage is increased, causing *PaTH* to shift some quanta to a higher performance mode. At a 5% slowdown level, a majority of the application is still run on the Big backend to meet performance requirements, resulting in a minor impact of leakage energy as the Little backend is not as heavily utilized, However, as the allowed slowdown increases, the Little backend is increasingly utilized and the overall energy savings reflect the additional leakage incurred when using the Little backend. Even with an additional 40% leakage power overhead, fine-grained HM's are still able to achieve better energy savings for a given level of performance than ideal DVFS. By analyzing the results from [11], for a low leakage 28nm FDSOI process, we expect the overhead of the Big's leakage to add an additional 10-20% power overhead when in Little mode.

5.5 Discussion

Prior works, most notably [2], relied on static application-to-core allocation to derive their performance/energy pareto frontiers. They arrive at two core designs, then used voltage scaling to provide incremental performance improvements at the cost of additional energy. However, our results indicate that HMs provide a superior tradeoff space than DVFS. In fact, our results suggest that building fine-grained DVFS is unnecessary when fine-grained HMs are available. Additionally, we show that these results are not affected by sensitivity to either switching overheads or leakage energy.

We believe there are two main differences that help to explain the discrepancy between prior works. First, we have more restrictive DVFS scaling due to the limitations imposed by the use of low-leakage transistors for low-power cores. Second, prior studies neglect to include the benefits of intelligent scheduling. By using scheduling to intelligently switch between different HMs, we can achieve many of the same performance levels available through DVFS with a higher energy savings.

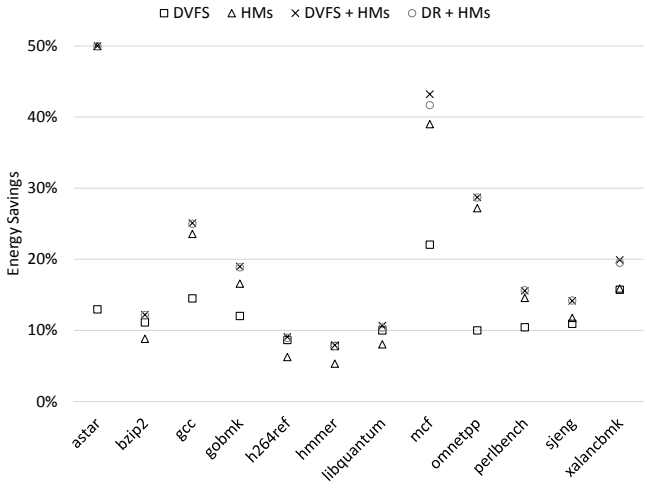

Figure 10: Per Benchmark Energy Savings for a 5% Slowdown

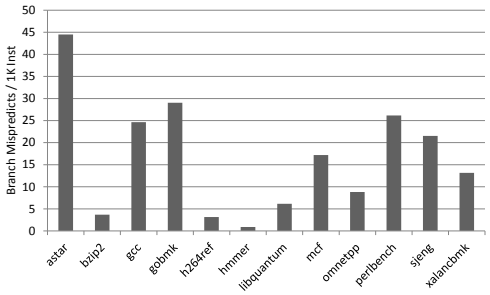

Figure 11: Branch mispredictions per thousand instructions

6. PROGRAMMATIC BEHAVIOR

Heterogeneity provides energy savings by mapping low performance application phases to more energy efficient modes (lower voltage or simpler microarchitecture). The goal of this section is to study the programmatic behavior of individual benchmarks and examine their predisposition to the different forms of heterogeneity through illustrative case studies. Figure 10 illustrates the wide range of potential energy savings for the different techniques across all the benchmarks studied. This range indicates that some benchmarks are better able to utilize DVFS, others HMs, and still others see little benefit from either fine-grained approach.

6.1 Expectations

To begin our study, we show a few metrics that we believe are most useful in explaining why some benchmarks prefer one form of heterogeneity over another. These metrics include branch mispredicts (Figure 11), L2 cache accesses (Figure 12), and instructions per cycle (Figure 13). These measurements were taken using the highest frequency level for the Big microarchitecture.

The first metric, branch mispredicts, is chosen to illustrate a scenario which puts Little at an advantage. The Little core, with its shorter pipeline length, can recover from branch mispredictions faster. Additionally, during periods of high mispredictions, lowering the frequency of a Big core neither increases branch prediction accuracy nor reduces

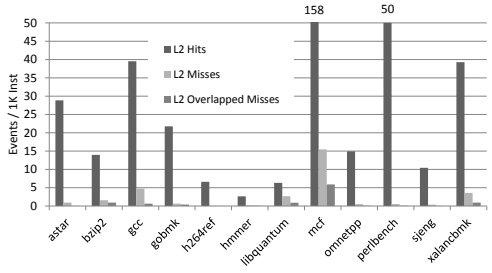

Figure 12: Categorized L2 accesses per thousand instructions

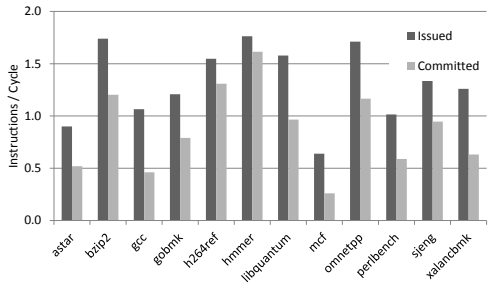

Figure 13: Issued and committed instructions per cycle in a thousand instruction window

misprediction penalties. Therefore, we consider benchmarks with highly unpredictable control flow to be more amenable to HMs.

The next metric is L2 cache accesses, which include three separate quantities, L2 hits, L2 misses, and L2 overlapable misses. As a high performance Big core cannot avoid stalling on L2 misses, these offer an ideal opportunity to switch to a Little core. However, if the L2 misses are independent, they can be issued in parallel by a Big core operating at a lower voltage. This is not possible with the Little core, which cannot exploit this parallelism. Therefore, if the application is generating a large number of independent L2 misses, DVFS is a better option. Alternatively, if it has large number of dependent L2 misses HMs are a better option.

Finally, the third metric is instructions per cycle. We observe that for several computation-bound benchmarks, neither form of heterogeneity is significantly utilized. The performance of these benchmarks is limited primarily by the frequency and issue width. Therefore they prefer the highest frequency on the core with the highest issue width. As DVFS reduces frequency, and HMs reduce issue width, both approaches deliver less than average energy returns.

Given Figures 11-13, we expect that $bzip2$, $hmmer$, $h264ref$ should be computation-bound. More control-bound applications, such as $astar$, $gobmk$, and $perlbench$, which have higher branch misprediction rates and lower numbers of L2 misses, should prefer HMs. mcf, $libquantum$, $xalancbmk$, which have higher L2 miss rates and numerous parallel loads, should prefer DVFS. The remaining benchmarks have intermediate levels of both branch mispredictions and L2 misses, and we expect them to be able to utilize some combination of both types of heterogeneity.

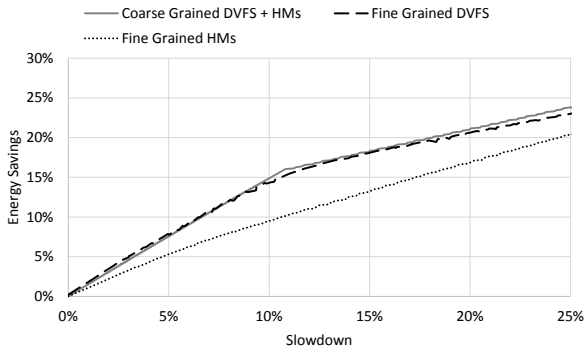

Figure 14: Pareto Frontier for $hmmer$. Note DVFS achieves better tradeoffs than HMs. Also the coarse-grained and fine-grained tradeoffs are nearly identical.

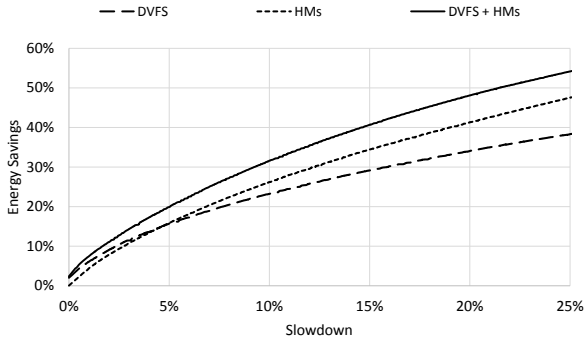

Figure 15: Pareto Frontier for $xalancbmk$. Note the much higher energy savings achieved using DVFS than HMs for small levels of slowdown.

6.2 Benchmark Classifications

The first, and perhaps least interesting, class of benchmarks are the computation-bound. In Figure 14, we have chosen $hmmer$ as the representative of this class. Due to the stable long-term phases of this benchmark, fine-grained approaches gain almost nothing over a coarse-grained approach. Also, these benchmarks prefer DVFS simply because the performance difference between modes is smaller. $hmmer$ effectively utilizes a 10% slower DVFS level to incur a 10% slowdown. However, its energy saving returns are only slightly better than the performance loss required to achieve it.

The first memory-bound application we study is $xalancbmk$. This benchmark has a moderate amount of L2 misses, but a large fraction of them can be overlapped. Figure 15 illustrates that the Pareto frontier for fine-grained DVFS is above (i.e. saves more energy than) fine-grained HMs, until around a 5% performance loss. For this benchmark DVFS, which can exploit parallel L2 misses, gives a better return than HMs. The return is because Little, while more energy efficient, must stall on every L2 miss. Beyond 5%, there is not a sufficient number of parallel loads to continue to make DVFS more efficient.

The second memory-bound benchmark we highlight is mcf. Figure 12 shows high levels of memory parallelism that indicate this benchmark should prefer to use DVFS. However, Figure 16 illustrates that it actually prefers HMs. This counter-intuitive result can be explained by noting that mcf

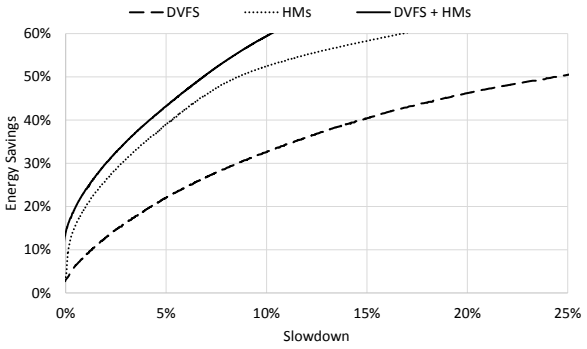

Figure 16: Pareto Frontier for *mcf*. Unlike expected, HMs are preferred to DVFS.

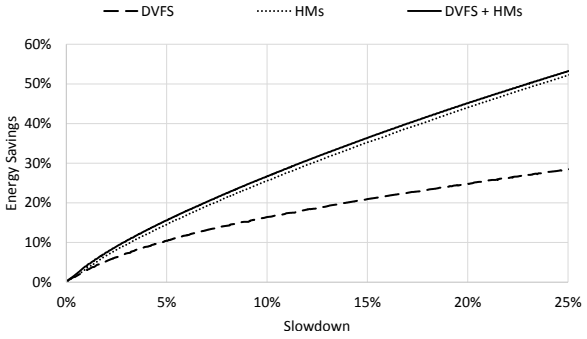

Figure 17: Pareto Frontier for *perlbench*. Note the utilization of HMs due to L2 pointer-chasing.

exhibits phased behaviors. While the initial phase contains a large number of parallel loads, after the phase change the number of parallel loads reduces considerably. Therefore, while the average parallel load count is still relatively high, a majority of this parallelism comes in the first phase of the execution. Therefore, *mcf* prefers to run a combination of DVFS and HMs to optimize individual execution phases.

The final application class consists of control-bound applications. For this example we choose to scrutinize *perlbench*, which has one of the highest branch misprediction rates. This benchmark also has a large number of L2 hits which tends to form dependence chains. Figure 17 illustrates that a combination of frequent branch mispredictions and serial memory accesses causes this benchmark to prefer HMs. In fact, *perlbench* maps so well to HMs that the addition of DVFS adds almost no benefit.

6.3 Coarse vs. Fine Grain

Now we return our attention to how switching granularity affects individual benchmarks. Benchmarks like *hmmer*, shown in Figure 14, which have stable long-term phases, allow coarse-grained approaches to fare as well as fine-grained. However, benchmarks with unstable phase behavior benefit from using a fine-grained system.

Figure 18 illustrates the benefits of fine-grained HMs for *omnetpp*. While coarse-grained HMs achieve a nearly linear tradeoff, fine-grained HMs are able to achieve 3x more savings than coarse-grained HMs at a 10% slowdown target. Figure 19 illustrates that if measured at a fine granularity, *omnetpp* shows two finely interleaved main performance

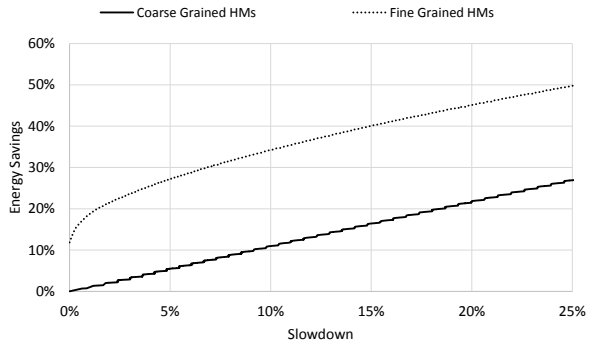

Figure 18: Pareto Frontier for *omnetpp*. Fine grained HMs achieve better tradeoffs than coarse-grained HMs.

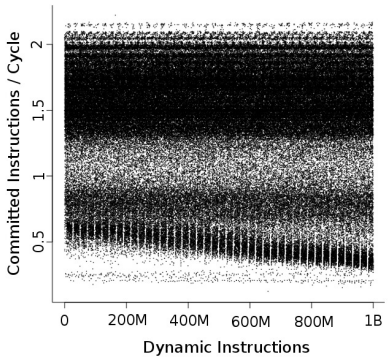

Figure 19: Committed Instruction / Cycle for *omnetpp*. Each point represents 1K dynamic instructions. Note the high performance band, which benefits from out-of-order, and the low performance band , which does not.

bands, one higher and one lower. Fine-grained switching is required to exploit them. The Big mode is required for the higher band, but the lower band can be run on the Little mode with nearly the same performance, reducing the energy consumed without impacting performance. As the low performance bands are not caused by parallel L2 misses, DVFS cannot target these phases as effectively as HMs.

6.4 Combined Heterogeneity

In some benchmarks, DVFS performs well for some phases of the application and HMs for others. Often initially DVFS achieves a better tradeoff, but there are a limited number of phases that can exploit DVFS, so HMs begin to dominate for higher slowdowns. This leads to a scenario where both are used simultaneously. A variety of benchmarks utilize both DVFS and HMs synergistically to achieve increased energy savings, e.g, *xalancbmk* and *mcf* of Figures 15 and 16.

7. CONCLUSION

This paper explored the implications of scheduling using different forms of heterogeneity to maximize energy efficiency for a given level of performance. We developed *PaTH*, an offline analysis tool that is capable of determining the Pareto-optimal performance/energy tradeoffs for a heterogeneous architecture. *PaTH* is capable of efficiently searching the enormous state space of possible schedules with tractable analysis turnaround time, by relying on

bounded approximations to find the Pareto-optimal performance/energy tradeoffs within an accuracy of ±2.5%. We then use *PaTH* to show the performance and energy tradeoffs for DVFS and HMs, as well as their combination.

Overall, our results show that HMs are able to provide higher energy efficiency than DVFS for a given level of performance. While at a coarse granularity the combination of DVFS and HMs still proves beneficial, for fine-grained scheduling their combination makes little sense as HMs alone provide the bulk of the energy efficiency. Additionally, we analyzed the effects of switching granularity and overheads on the choice of heterogeneity. Finally, we demonstrated that different benchmarks prefer different types and amounts of heterogeneity.

8. ACKNOWLEDGEMENTS

This work is supported in part by ARM Ltd and by the National Science Foundation under grant SHF-1217917. The authors would like to thank the fellow members of the CCCP research group, our shepherd (Osman Unsal), and the anonymous reviewers for their time, suggestions, and valuable feedback.

9. REFERENCES

[1] D. Albonesi, R. Balasubramonian, S. Dropsbo, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster, "Dynamically tuning processor resources with adaptive processing," *IEEE Computer*, vol. 36, no. 12, pp. 49 –58, Dec. 2003.

[2] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 26–36.

[3] R. Bahar and S. Manne, "Power and energy reduction via pipeline balancing," *Proc. of the 28th Annual International Symposium on Computer Architecture*, vol. 29, no. 2, pp. 218–229, 2001.

[4] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures," in *Proc. of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 245–257.

[5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[6] T. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, 2011, pp. 1–12.

[7] R. Dennard, F. Gaensslen, H.-N. Yu, V. LEO RIDEOVT, E. Bassous, and A. R. Leblanc, "Design of ion-implanted mosfet's with very small physical dimensions," *Solid-State Circuits Society Newsletter, IEEE*, vol. 12, no. 1, pp. 38–50, 2007.

[8] R. Dreslinski, "Near threshold computing: From single core to many-core energy efficient architectures," Ph.D. dissertation, University of Michigan, 2011.

[9] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle, "A predictive model for dynamic microarchitectural adaptivity control," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43, 2010, pp. 485–496.

[10] S. Eyerman and L. Eeckhout, "Fine-grained dvfs using on-chip regulators," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 1, pp. 1:1–1:24, Feb. 2011.

[11] P. Flatresse, G. Cesana, and X. Cauchy, "Planar fully depleted silicon technology to design competitive soc at 28nm and beyond," Feb. 2012, http://www.soiconsortium.org/link-812.php.

[12] P. Greenhalgh, "Big.little processing with arm cortex-a15 & cortex-a7," Sep. 2011.

[13] E. Grochowski, R. Ronen, J. Shen, and P. Wang, "Best of both latency and throughput," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, 2004, pp. 236–243.

[14] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein, "Scaling, power, and the future of cmos," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, 2005, pp. 7 pp.–15.

[15] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. of the 39th Annual International Symposium on Microarchitecture*, Dec. 2006, pp. 347–358.

[16] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 international symposium on Low power electronics and design*, ser. ISLPED '98, 1998, pp. 197–202.

[17] ITRS, "International technology roadmap for semiconductors 2012," 2012, http://www.itrs.net/.

[18] W. Kim, D. Brooks, and G.-Y. Wei, "A fully-integrated 3-level dc-dc converter for nanosecond-scale dvfs," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 1, pp. 206 –219, Jan. 2012.

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

[20] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *Proc. of the 5th European Conference on Computer Systems*, 2010, pp. 125–138.

[21] 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 *Proc. of the 36th Annual International Symposium on Microarchitecture*, Dec. 2003, pp. 81–92.

[22] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proc. of the 15th International Conference on Parallel Architectures and Compilation Techniques*, 2006, pp. 23–32.

[23] H.-P. Le, J. Crossley, S. R. Sanders, and E. Alon, "A sub-ns response fully integrated battery-connected switched-capacitor voltage regulator delivering 0.19 w/mm 2 at 73% efficiency," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*. IEEE, 2013, pp. 372–373.

[24] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim, "Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling," in *Proc. of the 20th International Conference on Parallel Architectures and Compilation Techniques*, 2011, pp. 111–120.

[25] H. Li, C.-Y. Cher, T. N. Vijaykumar, and K. Roy, "Vsv: L2-miss-driven variable supply-voltage scaling for low power," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36, 2003, pp. 19–28.

[26] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite cores: Pushing heterogeneity into a core," in *Proc. of the 45th Annual International Symposium on Microarchitecture*, 2012, pp. 317–328.

[27] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu, "Booster: Reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips," in *Proc. of the 18th International Symposium on High-Performance Computer Architecture*, vol. 0, 2012, pp. 1–12.

[28] J. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using asymmetric single-isa cmps to save energy on operating systems," *IEEE Micro*, vol. 28, no. 3, pp. 26–41, May 2008.

[29] H. Najaf-abadi, N. Choudhary, and E. Rotenberg, "Core-selectability in chip multiprocessors," in *Parallel Architectures and Compilation Techniques, 2009. PACT '09. 18th International Conference on*, 2009, pp. 113–122.

[30] H. Najaf-abadi and E. Rotenberg, "Architectural contesting," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, 2009, pp. 189–200.

[31] G. Patsilaras, N. K. Choudhary, and J. Tuck, "Efficiently exploiting memory level parallelism on asymmetric coupled cores in the dark silicon era," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 28:1–28:21, Jan. 2012.

[32] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread motion: fine-grained power management for multi-core systems," in *Proc. of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 302–313.

[33] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "Hass: A scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, Apr. 2009.

[34] L. Sheng, H. A. Jung, R. Strong, J.B.Brockman, D. Tullsen, and N. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009, pp. 469–480.

[35] J. Suh and M. Dubois, "Dynamic mips rate stabilization in out-of-order processors," in *Proc. of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 46–56.

[36] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *Proceedings of the 39th International Symposium on Computer Architecture*, ser. ISCA '12, 2012, pp. 213–224.

[37] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks, "A dynamic compilation framework for controlling microprocessor energy and performance," in *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 38, 2005, pp. 271–282.

[38] F. Xie, M. Martonosi, and S. Malik, "Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation," in *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, 2005, pp. 287–292.

[39] ——, "Compile-time dynamic voltage scaling settings: opportunities and limits," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, ser. PLDI '03, 2003, pp. 49–62.

[40] ——, "Efficient behavior-driven runtime dynamic voltage scaling policies," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '05, 2005, pp. 105–110.