A Theoretical Foundation for Scheduling and Designing Heterogeneous Processors for Interactive Applications

Shaolei Ren[1] and Yuxiong He[2] and Kathryn S. McKinley[2]

[1] Florida International University, Miami, FL
[2] Microsoft Research, Redmond, WA

Abstract

To improve performance and meet power constraints, vendors are introducing heterogeneous multicores that combine high performance and low power cores. However, choosing which cores and scheduling applications on them remain open problems. This paper presents a scheduling algorithm that provably minimizes energy on heterogeneous multicores and meets latency constraints for interactive applications, such as search, recommendations, advertisements, and games. Because interactive applications must respond quickly to satisfy users, they impose multiple constraints, including average, tail, *and* maximum latency. We introduce SEM (Slow-to-fast, Energy optimization for Multiple constraints), which minimizes energy by choosing core speeds and how long to execute jobs on each core. We prove SEM minimizes energy without *a priori* knowledge of job service demand, satisfies multiple latency constraints simultaneously, and only migrates jobs from slower to faster cores. We address practical concerns of migration overhead and congestion. We prove optimizing energy for *average* latency requires homogeneous cores, whereas optimizing energy for *tail* and *deadline* constraints requires heterogeneous cores. For interactive applications, we create a formal foundation for scheduling and selecting cores in heterogeneous systems.

1 Introduction

Power constraints are forcing computer architects to turn to heterogeneous multicore hardware to improve performance. For instance, smartphones are shipping with Qualcomm's Snapdragon and ARM's Cortex-A15 [15], which include high performance and low power cores with the same instruction set, called big/little and Asymmetric Multicore Processors (AMP). Design principles for selecting cores in heterogeneous system and scheduling algorithms that optimize their energy consumption, however, remain open problems. This paper presents a scheduling algorithm that provably minimizes energy on heterogeneous processors serving interactive applications. We prove and establish scheduling insights and design principles with practical implications for heterogeneous core selection.

Interactive applications are latency-sensitive. Examples include serving web pages, games, search, advertising, recommendations, and mobile applications. Since interactive applications must be responsive to attract and please users, they must meet *latency* requirements. Furthermore, they must be energy efficient. In the data center, power is an increasingly higher fraction of total costs [19,24,38]. A 1% energy saving may translate to millions of dollars. On mobile, energy efficiency translates directly into longer battery life and happier users.

Prior schedulers that optimize for energy efficiency and heterogeneity have major limitations. (1) They must predict demand for each request, scheduling high demand jobs to high performance fast cores and other jobs to low power slow cores [1,9,11,37,40,42]. Unfortunately, the service demand of individual requests in interactive applications is usually unknown and difficult to predict [23]. (2) For unknown service demand, prior work only optimizes for a *single simple* latency constraint [23,36,39], such as average latency or maximum latency, and is inadequate for two reasons. First, many applications strive for consistency by reducing tail latency (e.g., 95th- and 99th-percentile) or variance [13,17], which average and maximum latency do not model. Second, some applications *require* a combination of low average, tail, and worst-case latencies [12,16]. For example, search, finance applications, ads, and commerce have customer requirements and expectations for average and tail latency [12,13,17,24,38].

This paper shows how to optimize energy efficiency of interactive workloads subject to multiple latency constraints by exploiting heterogeneous multicores, addressing the aforementioned challenges as follows.

Unknown service demand. Instead of predicting individual job demand, we exploit the service demand *distribution* measured online or offline, which changes slowly over time [23,27]. We schedule incoming jobs to appropriate cores without knowing their individual service demands.

Multiple latency constraints. The scheduling literature typically optimizes for average or maximum latency only. To generalize and combine latency constraints, we use L_p norms [2–5,18,26,32,41]. The L_p norms encapsulate maximum latency ($p \to \infty$) and average latency ($p = 1$) as special cases. Optimizing for larger values of p places more emphasis on the latency of longer jobs. Appropriate values of p effectively mitigate unfairness and extreme outliers for long jobs [2,5]. Optimizing the L_1 and L_2 norms together reduce latency variance, which makes latency more *predictable* and improves user experience [33].

This paper presents an optimal algorithm that minimizes energy on heterogeneous processors given a demand distribution and latency constraints. We quantitatively characterize the optimal schedule and the ratio of fast to slow core speeds in a heterogeneous system. We present an *optimal* scheduling algorithm, called SEM (Slow-to-fast, Energy optimization for Multiple constraints). Given a service demand distribution, SEM schedules interactive jobs on heterogeneous multicore processors to minimize energy consumption while simultaneously satisfying multiple L_p norm latency constraints.

We show *an optimal schedule migrates jobs from slower to faster cores.* Ideally, we want to schedule high demand (long) jobs on fast cores to meet latency requirements and short jobs on slow cores to save energy *without* a prior knowledge of service demand. SEM exploits this observation by scheduling short jobs on energy efficient slow cores where they complete with high probability and then migrating long jobs to fast cores to meet the latency constraints.

We show *more heterogeneity is desirable for higher p,* where p is the L_p norm moment and the heterogeneity degree is the ratio of the fastest to slowest core speed. Given a single average latency constraint ($p = 1$), the energy optimal schedule requires a homogeneous processor. For all other latency constraints ($p > 1$) and multiple constraints, the optimal schedule requires heterogeneous processors.

We show *bounds on the ratio of the fastest and slowest core speeds for an optimal heterogeneous processor*. The result indicates that the more heterogeneous workload is and/or the less power additional core performance consumes, the more heterogeneous the hardware needs to be. Our result provides a formal and quantitative guide for selecting core speeds while designing heterogeneous processors. For practical choices of p and measured service load distributions, the ratio ranges from two to eight. Systems with this degree of heterogeneity are thus quite practical to assemble from current server, client, and mobile cores.

Due to space constraints, we state the theorems and intuitions here and refer readers elsewhere for the proofs [28]. We leave to future work experimental evaluation of energy. Our own prior work exploits the slow-to-fast insight to optimize *performance* (not energy) of interactive applications [27]. We achieved substantial performance improvements in simulation and on real systems by configuring Simultaneous Multi-Threading (SMT) hardware as a dynamic heterogeneous multicore [27]. No prior work presents an optimal algorithm or theory for energy efficiency under multiple latency constraints, nor provides guidelines for selecting core speeds. This work is the first formal analysis to deliver these properties for scheduling interactive workloads on heterogeneous multicore processors for energy minimization subject to multiple latency constraints.

2 Job, Processor, and Scheduling Models

This section and Table 1 describe our job, processor, and scheduling model.

Job model We focus on CPU intensive interactive services such as search, ads, finance option pricing, games, and serving dynamic web page content [6, 19, 24, 42]. Each interactive service request is a *job*. Each job has *work* w (service demand), which represents the number of CPU cycles the job takes to complete. Since it is often impossible to accurately predict a job's service demand [23], we model w as a discrete random variable whose value is unknown until the job completes. We divide the service demand into N bins and the *size* of the i-th bin is denoted by w_i, which we obtain by measuring the distribution of work for the application. The choice of "bin" sizes is determined by the measurement accuracy, and our model is not restricted to any particular choices. The job service demand w follows a distribution that only takes values out of the set $\mathcal{W} = \{\tilde{w}_1, \tilde{w}_2, \cdots, \tilde{w}_N\}$, where we define $\tilde{w}_i = \sum_{j=1}^{i} w_j$, for $i = 1, 2, \cdots, N$. This assumption is not restrictive. In practice, a job's service demand cannot be continuous and is typically grouped into a finite number of bins [36].

	Definition		Definition
w	CPU service demand	x_i	Speed of core i
w_i	Size of the i-th demand bin	$z(x)$	Power consumption
f_i	Probability of demand $\tilde{w}_i = \sum_{j=1}^{i} w_j$	$e(x)$	Energy function
F_i	Cumulative distribution	L_p	L_p norm with moment p
F_i^c	Complementary cumulative distribution	$\tilde{D}(p)$	L_p norm latency constraint

Table 1: Symbols and definitions

Let $\{f_1, f_2, \cdots, f_N\}$ and $\{F_0, F_1, \cdots, F_N\}$ be the probability distribution and cumulative distribution of the job's service demand, respectively: $f_i = \Pr(w = \tilde{w}_i)$ and $F_i = \sum_{j=1}^{i} f_j$, for $i = 1, 2, \cdots, N$. While the service demand of any single job is unknown *a priori*, we assume the aggregate service demand distribution of jobs is measured with online or offline profiling as in previous work [23].

Processor model We adopt a standard processor model. With speed $x > 0$, a core will consume a power of $z(x)$. Correspondingly, the energy consumption per unit work is $e(x) = z(x)/x$. The processing time for a unit work increases linearly with respect to the inverse of core speed. Given a particular application, the effective speed x and power $z(x)$ can be obtained by system measurements. Consequently, the effective speed x may differ from the clock rate of CPU and both clock speed an power may vary depending on the application [14, 20].

We assume the energy function $e(x)$ is continuously differentiable, increasing, and strictly convex in $x \geq 0$. This assumption is validated extensively by both analytical models and measurement studies [14, 23, 36]. In practice, if a slower core consumes more power and thus energy than a fast one, it wont be built. Because of CMOS circuit characteristics, energy is well approximated as $e(x) = b \cdot x^{\alpha-1} + c$ for core speed x, where the power exponent $\alpha \geq 2$ and static energy $c \geq 0$ [8, 23]. We concentrate on heterogeneous multicores which consists of multiple diverse cores, but our approach applies to cores with multiple speeds realized with DVFS.

We refer to the core executing the i-th bin of a job's demand as core i, for $i = 1, 2, \cdots, N$. We denote the core speed and power consumption of core i by x_i and $z_i = z(x_i)$, respectively. The energy consumption per unit work of core i is given by $e_i = e(x_i) = z(x_i)/x_i$. Two cores i and j may be equivalent in some cases, i.e., $x_i = x_j$, for $i, j = 1, 2, \cdots, N$. For example, one core will execute multiple bins when demand for this core differs between two or more jobs.

3 Scheduling objective —Energy

Our scheduling objective is minimize average energy on a heterogeneous processor when scheduling interactive jobs that are subject to multiple latency constraints. The scheduler determines the core speeds x_i for each bin $i = 1, 2, \cdots, N$. We express the average energy consumption of a job as

$$\bar{e}(\mathbf{x}) = \sum_{i=1}^{N} \left[\sum_{j=1}^{i} z_j \cdot \frac{w_j}{x_j} \right] \cdot f_i = \sum_{i=1}^{N} [1 - F_{i-1}] \cdot e(x_i) \cdot w_i, \tag{1}$$

where $e_i = e(x_i) = z(x_i)/x_i$ is the energy per unit work consumed by core i and $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ is a vector expression. The term "$\sum_{j=1}^{i} z_j \cdot \frac{w_j}{x_j}$" represents the energy consumption of a job with a service demand of $\sum_{j=1}^{i} w_j$ (which occurs with a probability of f_i), and hence we have the average energy consumption as $\sum_{i=1}^{N} \left[\sum_{j=1}^{i} z_j \cdot \frac{w_j}{x_j}\right] \cdot f_i$. Equivalently, we can rewrite the average energy consumption as $\sum_{i=1}^{N} [1 - F_{i-1}] \cdot e(x_i) \cdot w_i$, where $(1 - F_{i-1})$ is the probability that the i-th bin of the service demand is processed (i.e., the probability that a job has at least a service demand of $\sum_{j=1}^{i} w_j$).

4 Scheduling constraints —Latency

Many prior studies mainly focused on *single* and *simple* latency constraints, such as maximum latency (deadline) or average latency [23, 36]. Motivated by recent work that addresses latency requirements in contexts such as load balancing [2, 26], we introduce the L_p norm to generalize latency constraints. For concision, we sometimes abbreviate the L_p norm with L_p. Specifically, given the core speeds $\mathbf{x} = (x_1, x_2, \cdots, x_N)$, we mathematically express the L_p norm for latency as follows

$$D(p) = \left[\sum_{i=1}^{N} (t_i)^p \cdot f_i\right]^{\frac{1}{p}} = \left\{\sum_{i=1}^{N} \left[\sum_{j=1}^{i} \frac{w_j}{x_j}\right]^p \cdot f_i\right\}^{\frac{1}{p}}, \tag{2}$$

where $p \geq 1$ and $t_i = \sum_{j=1}^{i} \frac{w_j}{x_j}$ is the latency of a job with a service demand of $\tilde{w}_i = \sum_{j=1}^{i} w_j$. The L_p norm for latency generalizes over maximum and average latency. Given $p = \infty$, L_∞ is maximum latency and given $p = 1$, L_1 is average latency. Intuitively, larger values of p emphasize optimizing the latency of longer jobs, effectively mitigating unfairness and extreme outliers for long jobs [2, 5].

Latency variance determines the *predictability* of a scheduling algorithm [33] and depends on the L_2 and L_1 through the simple expression $L_2 - L_1$. For average latency and latency variance, we can apply various techniques, such as Chebyshev inequality, to bound tail distributions and estimate high-percentile latency. Thus, simultaneously considering multiple L_p latency constraints, such as the L_1 and L_2 norms, well characterizes requirements on interactive applications [2–4, 26].

This paper focuses on interactive applications where the actual demand of individual jobs is unknown and hence all jobs have the same latency constraints, e.g., all web pages have similar latency constraints, since users will abandon the browser if responses are too slow. Differentiated services for different jobs are beyond the scope of this paper and could be interesting future work.

5 Problem Formulation and Algorithm

This section formalizes the energy minimization problem and presents the SEM scheduling algorithm, which minimizes energy subject to latency constraints.

The inputs to SEM are the probability distribution of service demand f_i, the size of each service demand bin w_i, and energy consumption per unit work $e(x)$ in terms of the processing speed x. SEM outputs the optimal job schedule, which prescribes a sequence of core speeds $x_1, x_2, \cdots, x_N$, where x_i is the core speed to process the i-th service demand bin. An incoming job with unknown service demand will execute on the prescribed sequence of core speeds until completion. For example, given an application that has jobs with service demands of $1, 2, 5$, or 10 (units of work) and some probability distribution, then there are 4 service demand bins with the following sizes: $w_1 = 1, w_2 = 2 - 1 = 1, w_3 = 5 - 2 = 3, w_4 = 10 - 5 = 5$. Given a set of L_p latency constraints, SEM determines the optimal core speed x_i for executing each service demand bin w_i. For example, $x_1 = 1$ GHz, $x_2 = x_3 = 1.5$ Ghz, and $x_4 = 3$ GHz. This scheduling plan is determined offline and then used in deployment. In deployment, when a job arrives, it's service demand is unknown. SEM first processes the job on a 1 GHz core. If the job does not completed after 1 unit of work, SEM migrates the job to a 1.5 GHz core. If the job does not completed after processing another $w_2 + w_3 = 4$ units of work, SEM migrate it to a 3GHz core, and continue processing the job until it completes. Formally, this problem is stated as follows.

$$\mathbf{P1}: \ \min_{\mathbf{x}} \sum_{i=1}^{N} \{[1 - F_{i-1}] \cdot e(x_i) \cdot w_i\} \tag{3}$$

$$s.t., \ \left\{ \sum_{i=1}^{N} \left[\sum_{j=1}^{i} \frac{w_j}{x_j} \right]^{p_k} f_i \right\}^{\frac{1}{p_k}} \leq \tilde{D}(p_k), \tag{4}$$

$$\text{for } k = 1, 2, \cdots, K,$$

$$\mathbf{x} \succeq \mathbf{0}, \tag{5}$$

where $\succeq$ is an element-wise operator, constraining all the core speeds to be non-negative. This formulation assumes that the core speeds $x_1, x_2, \cdots, x_N$ can be continuously chosen from any non-negative values. In other words, here core speeds are unconstrained. (Section 8 shows how to handle the limited numbers of core speeds available in practice.) The objective function in (3) minimizes the average energy of all jobs. The latency constraints in (4) are imposed with K different norms where $1 \leq p_1 < p_2 < \cdots < p_K \leq \infty$. Note that imposing a tail latency constraint of L_∞ excludes outlier jobs, e.g., for 95-percentile latency, the 5% longest jobs are excluded by the L_∞ norm.

This **P1** formulation is a convex optimization problem. The latency constraints in Inequality (4) are convex because L_p norms are convex when $p \geq 1$. The speed constraints in Inequality (5) are linear. A linear combination of the energy consumption per unit work $e(x)$ is strictly convex in terms of the processing speed x due to CMOS characteristics [23]. The objective function in (3) is also convex. Since **P1** is convex, there exist efficient algorithms that find the globally optimal solution, which we denote as $\mathbf{x}^*$.

We derive the solution to **P1** using a primal-dual iterative approach. A companion technical report presents the algorithm and its proof [28]. We set a threshold ϵ as a stopping criterion such that the iteration stops once the difference of the L_2 norm between two consecutively iterated values is below the threshold. The iterative approach has a iteration-complexity bounded by $\mathcal{O}(1\backslash\epsilon^2)$ [22].

Note that we analytically derive the solution to **P1** instead of using a convex solver. The analytical form exposes important properties of the optimal solution and has implications for hardware core choices that we discuss in Section 6 and Section 7. These properties cannot be derived using a convex solver.

Further note that we only compute an optimal schedule *once* offline for any given job service demand distribution and heterogeneous system. Our online scheduler simply applies the precomputed optimal schedule, executing a job on each core speed for the precomputed specified optimal time, until the job completes. Therefore, the computational overhead in deployment is negligible.

6 An Optimal Schedule Migrates from Slow to Fast Cores

Under the optimal schedule, core speeds monotonically increase as hardware processes more of the job's work. In other words, *an optimal scheduler need only migrate a job from slower to faster cores.* Theorem 1 formalizes this property. While prior studies [23, 27, 36, 39] show to use the "slow to fast" property under the maximum latency constraint in different contexts such as DVFS, in contrast, Theorem 1 is the first formal result that applies it to the more general case of *any* latency norm constraint and with *multiple* latency norm constraints.

Theorem 1. *The optimal core speeds that solve **P1** satisfy* $0 < x_1^* \leq x_2^* \leq \cdots \leq x_N^*$*. If only the* L_1 *latency constraint is imposed, then* $x_1^* = x_2^* = \cdots = x_N^*$*.*

Proof. The technical report contains the proof [28]. ■

Theorem 1 tells us, without a priori knowledge of each job's service demand, an optimal schedule first processes a job on a slow core. If the job does not complete within some time interval (because it is long), SEM migrates it to faster cores. Thus, a short job completes on slower cores to save energy while a long job uses faster cores to meet the latency constraints. Consequently, the average energy consumption is minimized while satisfying latency constraints.

The intuition behind Theorem 1 is that long jobs have a greater impact on latency constraints. In particular, the latency norm constraint specified by Equation (2) is mostly dominated by long jobs (the larger p_k, the more dominated by long jobs, which can be seen by taking the partial derivative of (2) with respect to the latency experienced by jobs with various demands). In the extreme case, when $p_k \to \infty$, only the maximum latency incurred by the longest jobs is important. Thus, we want to process the long jobs fast enough to meet the latency constraints. On the other hand, processing short jobs using slower cores saves energy without penalizing the latency constraints.

If only the average latency constraint ($p = 1$) is considered, Theorem 1 reduces to a special case where $x_1^* = x_2^* = \cdots = x_N^*$, i.e., the optimal schedule uses a homogeneous processor. Intuitively, this reduction holds because delaying short and long jobs have the same impact on the L_1 norm. More formally, the technical report [28] derives that $e(x_i)x_i^2$ is the same for all $i = 1, 2, \cdots, N$ and hence, homogeneous speeds are optimal when only satisfying the L_1 norm latency constraint. For all other latency constraints ($p > 1$) and multiple constraints, the optimal energy-efficient schedule requires heterogeneous processors.

7 Implications for Cores in a Heterogeneous System

This section analyzes how latency constraints, workload characteristics, and core power/performance characteristics effect core choices in a heterogeneous system.

7.1 Effect of latency constraints on heterogeneity

Given Theorem 1, a key question is what core speeds to include in a heterogeneous system. In practice, the fastest cores are limited by physics and the software will be tuned such that the fastest core speed can satisfy the most demanding jobs. We therefore exploit this theorem to select the remaining lower power cores by investigating the ratio of the fastest x_N^* to the slowest x_1^* speed. We define this ratio as the *degree of heterogeneity*, giving a formal quantitative guideline for selecting core speeds in a heterogeneous processor. Our analysis shows that *more heterogeneity is desired for larger* p in the L_p norm constraint.

We derive this result using a widely-used class of energy functions [23] expressed in the form $e(x) = b \cdot x^{\alpha-1}$, where $b > 0$ and $\alpha \geq 2$ (corresponding to a power function of $z(x) = b \cdot x^{\alpha}$ [23]). The lack of a closed-form expression of the optimal core speeds $\mathbf{x}^*$ makes it prohibitive to derive the exact value of the degree of heterogeneity. We instead exploit monotonicity to derive upper and lower bounds, using Theorem 2 to show that degree of heterogeneity is monotonically increasing in $p \geq 1$.

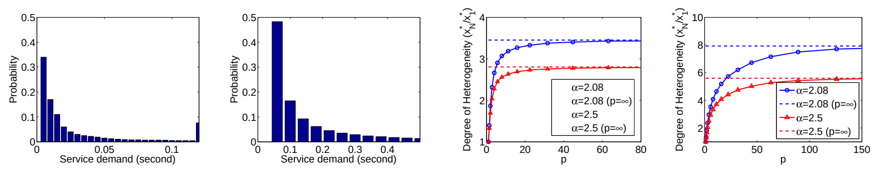
Theorem 2. *Given $e(x) = b \cdot x^{\alpha-1}$ and one L_p latency constraint, then the degree of heterogeneity $\frac{x_N^*}{x_1^*}$ increases with increasing p for $p \geq 1$.*

Proof. The technical report contains the proof [28]. ■

Theorem 2 proves that as $p \geq 1$ increases, the optimal degree of heterogeneity also increases; the latency constraint thus imposes the optimal choice of core speeds. More precisely, given two different values of p, we can select two different latency constraints, under which the corresponding minimum core speeds are the same using the optimal job schedule. Under a latency constraint with a larger p value, long jobs require faster cores, because larger values of p place a more stringent requirement on the latency of longer jobs. Thus, if p increases, so does x_N^*/x_1^*. Furthermore, we prove a lemma in the technical report [28] that the degree of heterogeneity is a constant for a given p regardless of latency constraints, which establishes *hardware requires more heterogeneity for larger* p.

7.2 How much heterogeneity is desirable?

This section explores how much heterogeneity is desirable. We use Theorem 2 to derive both upper and lower bounds on degree of heterogeneity in Theorem 3. This result delivers quantitative guidance for selecting the cores in heterogeneous multicore processors for interactive applications.



(a) Bing queries (b) Derivatives (c) Bing queries (d) Derivatives

Fig. 1: (a) (b) Service demand distributions of Bing and Financial derivative workloads. Most jobs are short, but long jobs are not negligible. (c) (d) Degree of heterogeneity as a function of p given one L_p constraint and power model: $z(x) = 21 \cdot x^{\alpha}$.

Theorem 3. *Given $e(x) = b \cdot x^{\alpha-1}$ and K L_p latency constraints specified by $1 \leq p_1 \leq p_2 \leq \cdots \leq p_K \leq \infty$, then the degree of heterogeneity $\frac{x_N^*}{x_1^*}$ satisfies:*

$$1 \leq \frac{x_N^*}{x_1^*} \leq \left(\frac{1}{f_N}\right)^{\frac{1}{\alpha}} \tag{6}$$

where f_N is the probability that a job has the maximum service demand of $\tilde{w}_N$.

We call a latency constraint dominant *if and only if satisfying it ensures that all the other latency constraints, if any, are also satisfied under the optimal schedule. Thus, the dominant latency constraint is the most stringent requirement. When average latency is dominant, the first inequality above becomes an equality: $x_N^*/x_1^* = 1$. When maximum latency is dominant, the second inequality becomes equality: $\frac{x_N^*}{x_1^*} = \left(\frac{1}{f_N}\right)^{\frac{1}{\alpha}}$.*

Proof. The technical report contains the proof [28]. ■

Theorem 3 has two interesting implications.

1. Workload heterogeneity prefers hardware heterogeneity. The upper bound on the degree of heterogeneity increases as f_N decreases (i.e., with fewer long jobs). When the workload is homogeneous, all jobs have the same service demand and $f_N = 1$. In this case, Theorem 3 indicates that $x_N^*/x_1^* = 1$ and homogeneous hardware is optimal. For a heterogeneous workload where f_N is small, the value of x_N^*/x_1^* may become very large. When slow cores complete short jobs, they save energy, whereas with the optimal schedule, the fastest processors process long jobs to satisfy the maximum latency constraint without incurring too much average energy, since f_N is small.

2. Core power and performance influences on hardware heterogeneity. When the speed of a core increases, so does its power consumption. We observe from (6) that the upper bound on the degree of heterogeneity decreases with α. A larger α indicates power consumption grows faster than core speed and hence using fast cores will significantly increases average energy consumption and degree of heterogeneity will be smaller.

Example We consider two example interactive workloads, Bing web search and Monte Carlo financial pricing (see elsewhere for details [27]). They illustrate how latency constraints, workload, and core performance and power characteristics affect the desired heterogeneity. Figure 1(a) and Figure 1(b) show the distributions of service demand for the two applications, measured in terms of the job processing time on an Intel i7-2600 Sandy Bridge core. The demand spike in Figure 1(a) occurs because the search engine caps job processing time at 120 ms and returns the top results found so far. Search engines often cap query processing time and return partial results to tradeoff quality and response time [17].

Figure 1(c) and 1(d) show how the degree of heterogeneity (Y-axis x_n^*/x_1^*) changes as a function of p in L_p with Bing and financial applications, respectively, when we can choose any core speed. We normalize speed to an i7-2600 Sandy Bridge core and use the power model: $z(x) = 21 \cdot x^{\alpha}$, because $z(1) = 21W$ is the power consumption of the i7-2600 Sandy Bridge core. Blue and red lines represent the cases of $\alpha = 2.08$ (a lower energy cost for performance) and $\alpha = 2.5$ (a higher energy cost for performance) respectively.

Figure 1(c) and 1(d) confirm Theorems 2 and 3. (1) When p increases, the degree of heterogeneity increases and has an upper bound, as predicted. In particular, a homogeneous processor is optimal in terms of energy consumption when $p = 1$ (average latency), whereas the maximum degree of heterogeneity is desirable when $p = \infty$ (a deadline). (2) The degree of heterogeneity decreases with larger α because faster cores consume proportionally more energy. (3) Comparing Figure 1(c) and 1(d) shows financial derivative pricing requires a higher degree of heterogeneity than Bing web search given the same p because the longest jobs are rarer in derivatives (f_N is smaller). The rarer the long jobs, the faster the fastest core we can choose without compromising average energy because the prolific short jobs execute on the slowest low power cores.

8 Discrete Core Speeds, Migration, and Congestion

This section extends SEM to address the following practical considerations: (1) a limited selection of core speeds, (2) job migration overhead, and (3) congestion due to multiple jobs competing for the same core(s).

Discrete core speeds Given a set of core speeds, $0 < s_1 \le s_2 \le \cdots \le s_M$, we formulate our problem as follows:

$$\mathbf{P2}: \min_{\mathbf{x}} \sum_{i=1}^{N} \{[1 - F_{i-1}] \cdot e(x_i) \cdot w_i\} \tag{7}$$

$$s.t., \text{ Constraint (4)} \tag{8}$$

$$x_i \in \{s_1, s_2, \cdots, s_M\}, i = 1, 2, \cdots, N. \tag{9}$$

P2 is a combinatorial optimization problem, which is notoriously difficult to solve [39]. We use an efficient branch-and-bound algorithm to produce solutions arbitrarily close-to-optimal. A greedy solution finds a schedule that will consume more energy than the optimal schedule (i.e., the upper bound), whereas the job schedule obtained by replacing "$x_i \in \{s_1, s_2, \cdots, s_M\}$" with $x_i \in [s_1, s_M]$ and then using convex optimization will produce an average energy consumption that is less than the optimal schedule (i.e., the lower bound). By iteratively finding and refining the upper and lower bounds until the gap becomes sufficiently small, we identify a schedule arbitrarily close to the optimal schedule [7]. The technical report contains the details of the solution and the derivation [28].

P2 is an NP-hard problem, even if only the maximum latency constraint is considered [39]. Without specifying the maximum number of iterations, the proposed algorithm may iterate up to M^N times, enumerating all the possible solutions in the worst case. Nevertheless, the beauty of branch-and-bound algorithm is that it typically converges much faster, which we also observe. In fact, with an appropriately-set stopping criterion, the number of iterations required for convergence is upper bounded, and in practice, the actual number of iterations is typically even much smaller than the upper bound. The complete analysis of convergence rate is beyond our scope, and interested readers are referred to the literature [7].

Moreover, as we discussed in Section 5, we only compute an optimal schedule *once* offline for any given job service demand distribution and heterogeneous processor. Our online scheduler simply applies the precomputed optimal schedule. Therefore, the computational overhead in deployment is negligible.

Migration overhead Migrating a job from one core to another incurs overhead from copying job state and warming up caches. Our experiments show that job migration overheads are fairly small on both web search [17] and interactive finance applications. One migration is less than 50 microseconds, less than 0.1% of the maximum latency requirement in the order of 100 milliseconds. Moreover, a job can only migrate up to $Q-1$ times, where Q is the number of different core speeds. Because Q is very small ($2 \sim 4$) in practice and many short jobs completed on slow cores, SEM often does not incur much migration overhead.

To extend our solution when migration costs are high, e.g., migrating a job between two servers, we describe a heuristic approach to incorporate migration overhead in the analytical model. This approach is conservative and assumes worst-case migration overhead. More specifically, let τ^o represent the migration overhead, quantified by the time during which a core cannot process any work. In the worst case, a job with a demand of $\tilde{w}_i = \sum_{j=1}^{i} w_j$ may migrate up to $(i-1)$ times, for $i = 1, 2, \cdots, N$. Thus, the new worst-case latency constraint becomes

$$\left\{ \sum_{i=1}^{N} \left[\sum_{j=1}^{i} \frac{w_j}{x_j} + (i-1) \cdot \tau^o \right]^{p_k} f_i \right\}^{\frac{1}{p_k}} \leq \tilde{D}(p_k). \tag{10}$$

By neglecting the constant energy consumption incurred by the migration process in the worst case, we reformulate the energy minimization problem **P2** by replacing the latency constraint (8) with (10) to account for the migration overhead. The solution can be found in a similar way following our preceding analysis.

Congestion We briefly discuss how to apply SEM as a building block when congestion or queuing delay results in multiple jobs demanding the same core at the same time. A key observation is that the presence of congestion may cause a violation in the latency constraints if we directly apply SEM. To satisfy the desired latency that includes both processing delay and queueing delay, we can impose a more stringent constraints for the processing delay which, if *appropriately* chosen and after adding the queueing delay, will satisfy the total latency constraints. To choose the appropriate L_p norm constraint to handle this delay, we propose integral control to dynamically adjust the processing delay constraint based on the difference between the observed latency and the target latency (latency constraint). The control function is expressed as

$$\tilde{D}_i(p_k) = \tilde{D}_{i-1}(p_k) + V \cdot d_i(p_k), \text{ for } k = 1, 2, \cdots, K,$$

where $i = 1, 2, \cdots$ represents time steps, $\tilde{D}_i(p_k)$ is the output of the integral controller at time i representing the augmented L_p norm constraint on the processing delay. $V > 0$ defines the ratio of the control adjustment to the control error and $d_i(p_k)$ is the difference between the target and observed latency. Thus, if the observed latency is greater than the constraint, $d_i(p_k) < 0$, a more stringent processing delay constraint, $\tilde{D}_i(p_k)$, will be imposed for the next time step, and vice versa.

Finally, note that using the above method to address congestion will not alter the value of p. Thus, our slow to fast scheduling insight and the quantitative upper and lower bounds on the ratio of fast to slow core speeds still hold.

9 Related Work

Heterogeneous multicore processors As computer architects face the end of Dennard scaling, they are turning to heterogeneous multicore processors, which combine high performance but high power cores with lower power and lower performance cores to meet a variety of performance objectives, i.e., throughput, energy, power, etc. To effectively utilize these systems, a scheduler must match jobs to an appropriate core. Four types of schedulers have been proposed to allocate jobs or parts of jobs to different cores. (1) With known or predicted resource demand, incoming jobs are scheduled to the most appropriate core [9, 11, 40]. (2) With known performance requirements, latency-sensitive applications such as games or videos are processed by fast cores, whereas latency-tolerant applications such as background services are processed by slow cores [15,25,29]. (3) With known job characteristics, complementary job allocation is applied to maximize the server utilization while avoiding resource bottlenecks (e.g., memory-intensive jobs and CPU-intensive jobs are allocated to the same server [35]). (4) If a single job has different phases [21,30,31], such as parallel phases and sequential phases, schedulers map the sequential phase on a high-performance core and the parallel phase on a number of energy-efficient cores.

L_p norms and multiple latencies Because the L_p norms are a general class of constraints, researchers have applied them in various contexts, such as minimizing the total latency via online load balancing [5, 32] and multi-user scheduling of wireless networks [41]. Our study considers multiple L_p norm latency constraints simultaneously for individual interactive services. Prior work mainly considers multiple latency constraints to provide differentiated performance guarantees to

different traffic classes [10,34], whereas we exploit the diversity of demand within the requests, without requiring knowledge about the demand of any individual request, to meet constraints for a variety of interactive applications.

Latency sensitive and real-time scheduling Related work also considers exploiting heterogeneous processors and DVFS to improve energy-efficiency for latency-sensitive and real-time jobs [1,23,36,37,39]. Some of them [1,37,40,42] assume that the service demand of each job is either known or accurately predicted, which is not available for many applications. Other studies on DVFS and real-time systems assume unknown service demand [23,36,39], but they consider a hard deadline as the only latency constraint. Our prior work [27] studies scheduling interactive workloads on a heterogeneous processor for quality/throughput maximization (not energy minimization) subject to a single deadline constraint. While it also leverages the "slow to fast" insight, it always uses fast cores first whenever they are available for performance optimization. In contrast, SEM starts jobs on slow cores and migrates them to fast cores along the execution to minimize energy. Moreover, this prior work [27] does not address multiple latency constraints and it does not deliver quantitative insights for selecting cores in heterogeneous processors. To the best of our knowledge, we offer the first formal analysis to characterize the optimal schedule and hardware design for scheduling latency-sensitive jobs on heterogeneous processors with multiple latency constraints without requiring a priori knowledge of the service demand of each individual job.

10 Conclusion

This paper presents an efficient scheduling algorithm for interactive jobs on heterogeneous processors subject to multiple latency constraints expressed in the form of L_p norms and optimizes energy. We introduce the SEM scheduling which advances the existing research in two key ways. (1) The SEM algorithm does not rely on the service demand of each individual job, which is difficult and even impossible to obtain in many interactive applications such as web search. (2) The SEM algorithm explicitly incorporates multiple L_p norm latency constraints which, compared to prior work, more accurately characterize the explicit and implicit multiple service level agreements on the latency of interactive applications. We prove that an optimal schedule only migrates jobs from slower to faster cores. Moreover, we quantify how to select cores in heterogeneous hardware for interactive applications. The more the system needs to limit outliers, the more heterogeneous the hardware needs to be. The more heterogeneous the workload service demand is, the less power additional performance costs and the more heterogeneous the hardware needs to be.

References

1. Albers, S., Muller, F., and Schmelzer, S. Speed scaling on parallel processors. In *SPAA* (2007).
2. Anand, S., Garg, N., and Kumar, A. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA* (2012).

3. Azar, Y., and Epstein, A. Convex programming for scheduling unrelated parallel machines. In *STOC* (2005).
4. Azar, Y., Epstein, L., Richter, Y., and Woeginger, G. J. All-norm approximation algorithms. In *SWAT* (2002).
5. Bansal, N., and Pruhs, K. Server scheduling in the l_p norm: A rising tide lifts all boat. In *STOC* (2003).
6. Bornholt, J., Mytkowicz, T., and McKinley, K. S. The model is not enough: Understanding energy consumption in mobile devices. In *Hot Chips* (2012).
7. Boyd, S., Ghosh, A., and Magnani, A. Branch and bound methods, `http://www.stanford.edu/class/ee392o/bb.pdf`, 2003.
8. Brooks, D., Bose, P., Schuster, S., Jacobson, H., Kudva, P., Buyuktosunoglu, A., Wellman, J., Zyuban, V., Gupta, M., and Cook, P. Power-aware microarchitecture: Design and modeling challenges for next generation microprocessors. In *Micro* (2000).
9. Cao, T., Blackburn, S. M., Goa, T., and McKinley, K. S. The yin and yang of power and performance for asymmetric hardware and managed software. In *ISCA* (2012).
10. Chao, H. J., and Uzun, N. An atm queue manager handling multiple delay and loss priorities. *IEEE/ACM Trans. Networking 3* (December 1995), 652–659.
11. Chen, J., and John, L. K. Efficient program scheduling for heterogeneous multicore processors. In *DAC* (2009).
12. Dean, J., and Barroso, L. A. The tail at scale. *CACM 56*, 2 (2013), 74–80.
13. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. In *SOSP* (2007).
14. Esmaeilzadeh, H., Cao, T., Xi, Y., Blackburn, S. M., and McKinley, K. S. Looking back on the language and hardware revolutions: Measured power, performance, and scaling. In *ASPLOS* (2011).
15. Greenhalgh, P. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7. *ARM Whitepaper* (September 2011).
16. Harchol-Balter, M. The effect of heavy-tailed job size distributions on computer system design. In *Applications of Heavy Tailed Distributions in Economics* (1999).
17. He, Y., Elnikety, S., Larus, J., and Yan, C. Zeta: Scheduling interactive services with partial execution. In *SOCC* (2012).
18. Im, S., and Moseley, B. An online scalable algorithm for minimizing l_k-norms of weighted flow time on unrelated machines. In *SODA* (2011).
19. Janapa Reddi, V., Lee, B. C., Chilimbi, T., and Vaid, K. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *ISCA* (2010).
20. Kotla, R., Devgan, A., Ghiasi, S., Keller, T., and Rawson, F. Characterizing the impact of different memory-intensity levels. In *WWC* (2004).
21. Kumar, R., Farkas, K. I., Jouppi, N. P., Ranganathan, P., and Tullsen, D. M. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO* (2003).
22. Lan, G., Lu, Z., and Monteiro, R. D. Primal-dual first-order methods with $\mathcal{O}(1\backslash\epsilon)$ iteration-complexity for cone programming. *Mathematical Programming 126*, 1 (2011), 1–29.
23. Lorch, J. R., and Smit, A. J. Improving dynamic voltage scaling algorithms with PACE. In *SIGMETRICS* (2001).
24. Meisner, D., Sadler, C. M., Barroso, L. A., Weber, W.-D., and Wenisch, T. F. Power management of online data-intensive services. In *ISCA* (2011).

25. Nathuji, R., Isci, C., and Gorbatov, E. Exploiting platform heterogeneity for power efficient data centers. In *ICAC* (2007).
26. Pruhs, K. Competitive online scheduling for server systems. In *SIGMETRICS* (2007).
27. Ren, S., He, Y., Elnikety, S., and McKinley, K. S. Exploiting processor heterogeneity in interactive systems. In *ICAC* (2013).
28. Ren, S., He, Y., and McKinley, K. S. A theoretical foundation for scheduling and designing heterogeneous processors for interactive applications. Tech. Rep. TR-2014-101, Microsoft Research, 2014.
29. Srinivasan, S., Iyer, R., Zhao, L., and Illikkal, R. HeteroScouts: Hardware assist for OS scheduling in heterogeneous CMPs. In *SIGMETRICS* (2011).
30. Suleman, M. A., Mutlu, O., Qureshi, M. K., and Patt, Y. N. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS* (2009).
31. Suleman, M. A., Patt, Y. N., Sprangle, E., Rohillah, A., Ghuloum, A., and Carmean, D. Asymmetric chip multiprocessors: Balancing hardware efficiency and programmer efficiency. *TR-HPS-2007-001* (2007).
32. Suri, S., Tóth, C. D., and Zhou, Y. Selfish load balancing and atomic congestion games. In *SPAA* (2004).
33. Wierman, A., and Harchol-Balter, M. Classifying scheduling policies with respect to higher moments of conditional response time. In *SIGMETRICS* (2005).
34. Xie, Y., and Yang, T. Cell discarding policies supporting multiple delay and loss requirements in atm networks. In *Globecom* (1997).
35. Xiong, W., and Kansal, A. Energy efficient data intensive distributed computing. In *IEEE Data Eng. Bull.* (2011).
36. Xu, R., Xi, C., Melhem, R., and Moss, D. Practical PACE for embedded systems. In *EMSOFT* (2004).
37. Yao, F. F., Demers, A. J., and Shenker, S. J. A scheduling model for reduced CPU energy. In *FOCS* (1995).
38. Yi, J., Maghoul, F., and Pedersen, J. Deciphering mobile search patterns: A study of Yahoo! mobile search queries. In *WWW* (2008).
39. Yuan, W., and Nahrstedt, K. Energy-efficient CPU scheduling for multimedia applications. *ACM Trans. Computer Systems 24*, 3 (2006), 292–331.
40. Yun, H., Wu, P.-L., Arya, A., Kim, C., Abdelzaher, T. F., and Sha, L. System-wide energy optimization for multiple DVS components and real-time tasks. *Real-Time Systems 47*, 5 (2011), 489–515.
41. Zeng, W., Ng, C., and Medard, M. Joint coding and scheduling optimization in wireless systems with varying delay sensitivities. In *SECON* (2012).
42. Zhu, Y., and Reddi, V. J. High-performance and energy-efficient mobile web browsing on big/little systems. In *HPCA* (2013).