Multicore Model from Abstract Single Core Inputs

Emily Blem[†] Hadi Esmaeilzadeh[‡] Renée St. Amant[§] Karthikeyan Sankaralingam[†] Doug Burger[◇]
[†]University of Wisconsin-Madison [‡]University of Washington [§]The University of Texas at Austin [◇]Microsoft Research
blem@cs.wisc.edu hadianeh@cs.washington.edu stamant@cs.utexas.edu karu@cs.wisc.edu dburger@microsoft.com

Abstract—This paper describes a first order multicore model to project a tighter upper bound on performance than previous Amdahl's Law based approaches. The speedup over a known baseline is a function of the core performance, microarchitectural features, application parameters, chip organization, and multicore topology. The model is flexible enough to consider both CPU and GPU like organizations as well as modern topologies from symmetric to aggressive heterogeneous (asymmetric, dynamic, and fused) designs. This extended model incorporates first order effects—exposing more bottlenecks than previous applications of Amdahl's Law—while remaining simple and flexible enough to be adapted for many applications.

Index Terms—Performance modeling, multicores, parallelism

1 Introduction

Amdahl's Law [1] is extensively used to generate upper-bound multicore speedup projections. Hill and Marty's extensions study a range of multicore topologies [7]. They model the trade-off between core resources (r) and core performance as $perf(r) = \sqrt{r}$ (Pollack's Rule [10]). Performance of various multicore topologies is then a function of r, $perf(r)$, and the fraction of code that is parallelizable. We tighten their upper-bound on multicore performance using real core measurements in place of the $perf(r)$ function and consider the impact of additional architectural and application characteristics.

To improve multicore performance projection accuracy while maintaining an abstract, easily implemented model, we develop a first order model using architectural and application input data. We model a heterogeneous chip with a mix of CPU- and GPU-like cores with varying performance. The chip's topology may be symmetric, asymmetric, dynamic, or even dynamically compose cores together (fused). This flexibility and wide design space coverage permits projections for a diverse and comprehensive set of multicores.

The range of low to high performing cores is represented using simple performance/resource constraint trade-offs (the *trade-off function*). These model inputs can be derived from a diverse set of input measures for performance (e.g., SPECmark, CoreMark) and resource constraints (e.g., area, power). They are then used to predict performance in a diverse set of applications with only a few additional application-specific parameters. The trade-off function may be concrete points from measured data, a function based on curve fitting to known designs, or an abstract function of expected trade-offs. Examples of the input trade-offs using curve fitting are Pareto frontiers like those presented in Azizi et al. [2] and Esmaeilzadeh et al. [4], and more abstract trade-off functions like Pollack's Rule [10].

Elements of this model were proposed in our previous work [4], wherein the model was intricately tied to empirically measured Pareto-frontier curves for Power/Performance and Area/Performance using SPEC scores. While related, this work makes the following contributions of its own. First, and most importantly, in this work, we separate out this interdependency to SPEC and Pareto frontiers and present a stand-alone model that can be used for performance projection given specific program and architecture inputs, completely obviating the need for SPEC scores — as part of this, we also increase its flexibility in modeling different microarchitectures. Specifically, we present a complete model consisting of five components: core performance, memory bandwidth performance, chip constraints, multicore performance, and overall speedup. Second, we refine the model's cache handling by allowing either a hit-rate number or an analytical model input of its own. Third, we present a detailed validation to show the accuracy of the model and its flexibility in handling microarchitecture effects. We have expanded the validation to include a microarchitectural study for the CPU validation and a more detailed study of the number of cores and the impact of memory bandwidth for the GPU validation, as well as a comparison against Amdahls Law projections. An implementation of this model with a web-based front-end has been publicly released at http://research.cs.wisc.edu/vertical/DarkSilicon.

Below and in Figure 1, we summarize the model components' intuition, simple inputs, and useful outputs.

Core Performance: This component finds the expected performance of a single core. The impact of core microarchitecture, CPU-like or GPU-like organization, and memory access latency are all incorporated here. The inputs include organization, CPI, frequency, cache hierarchy, and cache miss rates. CPI and frequency are functions of the trade-off function's performance.

Memory Bandwidth Performance: This component finds the maximum performance given memory bandwidth constraints by finding the number of memory transactions per instruction. The inputs are the trade-off function design point, application memory use, and maximum memory bandwidth.

Chip and Topology Constraints: This component finds the number of each type of core to include on the chip, given the multicore topology, organization, trade-off function, and resource constraints. Four multicore topologies are described in detail in Table 1. Each core is described as a combination of the trade-off function design point and thread style.

Multicore Performance: This component finds the expected serial and parallel performance for an application on a multicore chip with a particular topology. Inputs include outputs from the previous three models: core performance, number of cores of each type, and the memory bandwidth performance. We assume the first order bottlenecks for chip performance are core performance and memory bandwidth, and compute the total serial and parallel performance using these constraints.

Multicore Speedup: This component accumulates the results with the fractions of parallel and serial code into a single speedup projection using an Amdahl's Law based approach.

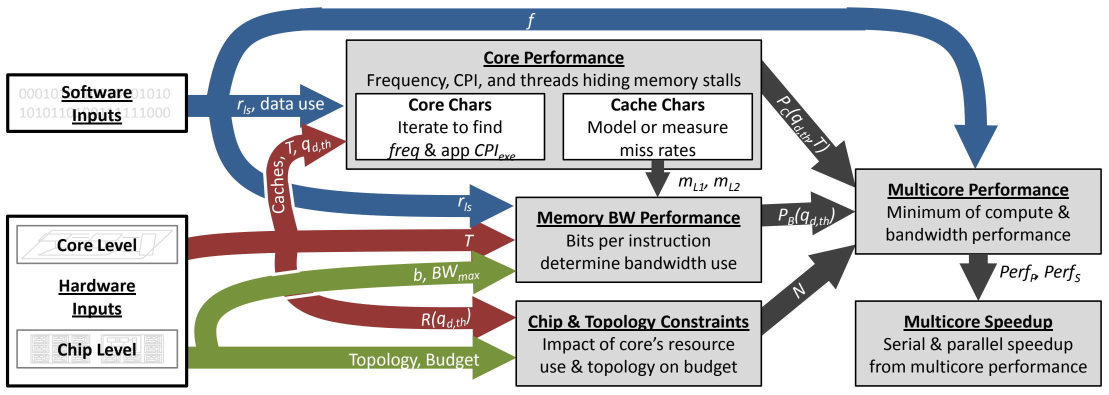
f
Software Inputs
r_{ls}, data use
Core Performance
Frequency, CPI, and threads hiding memory stalls
Core Chars
Iterate to find freq & app CPI_{exe}
Cache Chars
Model or measure miss rates
$P_C(q_{d,th}, T)$
Caches, T, $q_{d,th}$
m_{L1}, m_{L2}
r_{ls}
Memory BW Performance
Bits per instruction determine bandwidth use
$P_B(q_{d,th})$
Multicore Performance
Minimum of compute & bandwidth performance
Core Level
Hardware Inputs
Chip Level
T
b, BW_{max}
$R(q_{d,th})$
Topology, Budget
Chip & Topology Constraints
Impact of core's resource use & topology on budget
N
$Perf_P$, $Perf_S$
Multicore Speedup
Serial & parallel speedup from multicore performance

Fig. 1. Model component overview. Shaded boxes are key model components and watermarked boxes are inputs.

TABLE 1
Multicore Topology Descriptions
(a) CPU: Single-Threaded (ST) Cores

Topology	Serial Mode	Parallel Mode
Symmetric	1 Small ST Core	N Small ST Cores
Asymmetric	1 Large ST Core	1 Large ST & N Small ST Cores
Dynamic	1 Large ST Core	N Small ST Cores
Fused	1 Large ST Core	N Small ST Cores

(b) GPU: Multi-Threaded (MT) and ST Cores

Topology	Serial Mode	Parallel Mode
Symmetric	1 MT Core Thread	N Small MT Cores
Asymmetric	1 Large ST Core	1 Large ST & N Small MT Cores
Dynamic	1 Large ST Core	N Small MT Cores
Fused	1 Large ST Core	N Small MT Cores

TABLE 2
Model Input Parameters

Input	Description
$q_{d,th}$	Core performance (e.g., SPECmark, d: S/L, th: ST/MT)
$R(q_{d,th})$	Core resource cost (area, power) from trade-off func
$freq$	Core frequency (MHz) from computation model
CPI_{exe}	Cycles per inst (zero-latency cache) from comp model
f	Fraction of code that can be parallelized
T	Number of threads per core (CPU or GPU style)
r_{ls}	Fraction of instructions that are loads or stores
m_{L1}	L1 cache miss rate (from cache model or measured)
m_{L2}	L2 cache miss rate (from cache model or measured)
t_{L1}	L1 cache access time (cycles)
t_{L2}	L2 cache access time (cycles)
t_{mem}	Memory access time (cycles)
BW_{max}	Maximum memory bandwidth (GB/s)
b	Bytes per memory access (B)

2 Model Description

This section describes in detail the model's five components. Building on Guz et al.'s model for an idealized symmetric multicore with a perfectly parallelized workload (Equations 1-4 below) [6] and Hill et al.'s heterogeneous multicore extensions to Amdahl's Law using abstract inputs [1], [7], we use real core and multicore parameters and add additional hardware details to find a tighter upper-bound on multicore performance. The simple inputs listed in Table 2 are easily measured, derived from known results, or even based on intuition.

Each component is described by first outlining how it is modeled and then discussing any derived model inputs. This approach highlights our novel incorporation of real application behavior and realistic microarchitectural features.

2.1 Core Performance

Single core performance ($P_C(q_{d,th}, T)$) is calculated in terms of instructions per second in Equation 1 by scaling the core utilization (η) by the ratio of the processor frequency to CPI_{exe}:

$$P_C(q_{d,th}, T) = \eta \times freq/CPI_{exe} \qquad (1)$$

The CPI_{exe} parameter does not include stalls due to cache accesses, which are considered separately in the core utilization (η). The core utilization is the fraction of time that threads running on the core can keep it busy. The maximum number of threads per core is a key component of the core style: CPU-like cores are single-threaded (ST) and GPU-like cores are many threaded (e.g., 1024 threads per 8 cores). Topologies may have a mix of large ($d = L$) and small ($d = S$) cores. Core utilization is modeled as a function of the average time (cycles) spent waiting for each load or store (t), fraction of instructions that are loads or stores (r_{ls}), and the CPI_{exe}:

$$\eta = \min\left(1, \frac{T}{1 + t \times r_{ls}/CPI_{exe}}\right) \qquad (2)$$

We assume two levels of cache and an off-chip memory that contains all application data. The average time spent waiting for loads and stores is a function of the time to access the caches (t_{L1} and t_{L2}), time to visit memory (t_{mem}), and the predicted cache miss rate (m_{L1} and m_{L2}):

$$t = (1 - m_{L1})t_{L1} + m_{L1}(1 - m_{L2})t_{L2} + m_{L1}m_{L2}t_{mem} \qquad (3)$$

Although a cache miss model could be inserted here (e.g., [9]), we assume the miss rates are known here. Changes in miss rates due to increasing number of threads per cache should be reflected in the inputs. We assume the number of cycles per cache access is constant as the frequency changes, while memory latency (in cycles) increases linearly with frequency increases. Note that Fermi style GPUs could be modeled using this cache latency model and cache miss rate inputs.

Derived Inputs To incorporate known single-threaded performance/resource trade-offs into the model, we convert single-threaded performance into an estimated core frequency and per-application CPI_{exe}. This novel approach uses generic single-threaded performance results (e.g. SPECmark scores) to derive parameters for specific multi-threaded applications. This works because, assuming comparable memory systems, the key performance bottlenecks are the processor frequency ($freq$) and microarchitecture (summarized by CPI_{exe}). The approach finds the processor frequency and microarchitecture impact based only on the known single-threaded performance and the frequency of the highest and lowest performing processor, so it can be applied to abstract design points where only the performance/resource trade-offs are known.

***freq*:** To find each core's frequency, we assume frequency scales linearly with performance, from the frequency of the lowest performing point to the that of the highest performing point.

CPI_{exe}: Each application's CPI_{exe} is dependent on its instruction mix and use of hardware optimizations (e.g., functional units and out-of-order processing). Since the measured CPI_{exe} for each benchmark is not available, the core model is used to generate per benchmark CPI_{exe} estimates for each design point. With all other model inputs kept constant, the approach iteratively searches for the CPI_{exe} at each processor design point. The initial assumption is that the highest performing core has a CPI_{exe} of ℓ. The smallest core should have a CPI_{exe} such that the ratio of its performance to the highest performing core's performance is the same as the ratio of their measured scores. Since the performance increase between any two points should be the same using either the measured performance or model, the same ratio relationship is used to estimate a per benchmark CPI_{exe} for each processor design point. This flexible approach uses the measured performance to generate reasonable performance model inputs at each design point.

2.2 Memory Bandwidth Performance

The maximum performance possible given off-chip bandwidth constraints ($P_B(q_{d,th})$, in instructions per second) is the ratio of the maximum bandwidth to the average number of bytes of data required per operation:

$$P_B(q_{d,th}) = \frac{BW_{max}}{b \times r_{ls} \times m_{L1} \times m_{L2}} \tag{4}$$

This is from Guz et al.'s model; previous bandwidth saturation work includes a learning model by Suleman et al. [?].

2.3 Multicore Topology and Chip Constraints

The number of cores, N, that fit in the chip's resource budget is dependent on the types of cores used, the uncore components included (e.g., a second level of cache), and the chip topology. The type of cores used ($q_{d,th}$) affects the resources required based on the known performance/resource constraint trade-off ($R(q_{d,th})$). Examples of how to compute N for area constrained chips are in Hill and Marty [7] and examples for area and power constrained chips are in Esmaeilzadeh et al. [4].

Asymmetric, dynamic, and fused multicore topologies are similar at the performance level, but significantly different at the resource constraint level. Asymmetric cores have one large core and multiple smaller cores, all of which always consume power. For power-constrained chips, we study the dynamic core; the large core is disabled during parallel code segments and the smaller cores are disabled during serial code segments. For power- and area-constrained chips, we study the fused core, where the smaller cores are fused together to form one larger core during serial execution. Although the fused topology primarily affects the resource budget, it may have performance impacts: fused cores may not perform as well as similarly sized serial cores in asymmetric or dynamic topologies. From Atom and Tesla die photo inspections, we estimate that 8 small MT cores, their shared L1 cache, and their thread register file can fit in the same area as an Atom processor and assume a similar power correspondence.

2.4 Multicore Performance

This model component finds serial performance ($Perf_S$) and parallel performance ($Perf_P$) of an either CPU or GPU multicore. Two performance bottlenecks limit $Perf_P$ and $Perf_S$: computation capability and bandwidth limits.

The maximum computation performance is the sum of the $P_C(q_{d,th}, T)$ values for all cores being used. By definition, for completely serial code, this is just the performance of a single core, $P_C(q_{d,th}, T)$. For parallel code, $P_C(q_{d,th}, T)$ is generally multiplied by the number of active cores.

During serial and parallel phases, core resources are allocated based on the topology as described above and in Table 1; serial performance ($Perf_S$) and parallel performance ($Perf_P$) are thus computed based on the particular topology and organization's execution paradigm. For CPU organizations, topology specific $Perf_S$ and $Perf_P$ calculations are below:

Symmetric	$Perf_S = \min(P_C(q_{S,ST}, 1), P_B(q_{S,ST}))$
	$Perf_P = \min(N \times P_C(q_{S,ST}, 1), P_B(q_{S,ST}))$
Asymmetric	$Perf_S = \min(P_C(q_{L,ST}, 1), P_B(q_{L,ST}))$
	$Perf_P = \min(P_C(q_{L,ST}, 1) + N \times P_C(q_{S,ST}, 1), P_B(q_{S,ST}))$
Dynamic & Fused	$Perf_S = \min(P_C(q_{L,ST}, 1), P_B(q_{L,ST}))$
	$Perf_P = \min(N \times P_C(q_{S,ST}, 1), P_B(q_{S,ST}))$

For GPU organizations, the symmetric organization is similar to a simple GPU. Heterogeneous organizations have one CPU-like core for serial work and GPU-like cores for parallel work:

Symmetric	$Perf_S = \min(P_C(q_{S,MT}, 1), P_B(q_{S,MT}))$
	$Perf_P = \min(N \times P_C(q_{S,MT}, T_{MT}), P_B(q_{S,MT}))$
Asymmetric	$Perf_S = \min(P_C(q_{L,ST}, 1), P_B(q_{L,ST}))$
	$Perf_P = \min(P_C(q_{L,ST}, 1) + N \times P_C(q_{S,MT}, T_{MT}), P_B(q_{S,MT}))$
Dynamic & Fused	$Perf_S = \min(P_C(q_{L,ST}, 1), P_B(q_{L,ST}))$
	$Perf_P = \min(N \times P_C(q_{S,MT}, T_{MT}), P_B(q_{S,MT}))$

2.5 Multicore Speedup

Per Amdahl's law [1], system speedup is $\frac{1}{(1-f)+\frac{f}{S}}$ where f represents the portion that can be parallelized, and S represents the speedup achievable on the parallelized portion. Following the example from Hill and Marty, we expand this approach to use performance results from the previous section.

To find the overall speedup, the model finds $Perf_S$ and $Perf_P$ and a baseline core and topology's (e.g., a quadcore Nehalem's) serial and parallel performance as computed using the multicore performance equations in Section 2.4 ($Base_S$ and $Base_P$, respectively). The serial portion of code is thus sped up by $S_{Serial} = Perf_S / Base_S$ and the parallel portion of the code is sped up by $S_{Parallel} = Perf_P / Base_P$. The overall speedup is then:

$$Speedup = 1 / \left(\frac{1-f}{S_{Serial}} + \frac{f}{S_{Parallel}} \right) \tag{5}$$

To find the fraction of parallel code, f, for an application, we find the application speedup when the number of cores varies. We do an Amdahl's law-based curve fit of the speedups across the different numbers of cores, and optimistically assume that the parallelized portions of the code could be further parallelized across an infinite number of cores without penalty.

3 Model Assumptions

The model's accuracy is limited by our optimistic assumptions, thus making our speedup projections over-predictions.

Memory Latency: A key assumption in the model is that memory accesses from a processor are in order and blocking. We observe that for high performance cores where CPI_{exe} approaches 0, performance is limited by memory latency due to the in order and blocking assumption:

$$\lim_{CPI_{exe} \to 0} P_C(q_{d,th}, T) = \lim_{CPI_{exe} \to 0} \frac{freq}{CPI_{exe}} \min\left(1, \frac{T}{1 + t\frac{r_{ls}}{CPI_{exe}}}\right) = \frac{T freq}{t \times r_{ls}}$$

This limitation is expected to have negligible impact when CPI_{exe} is greater than 1, but its impact increases as CPI_{exe} decreases, implying more superscalar functionality in the core.

Microarchitecture: We assume memory accesses from different cores do not cause stalls. Further, we assume that the interconnect has zero latency, shared memory protocols have no

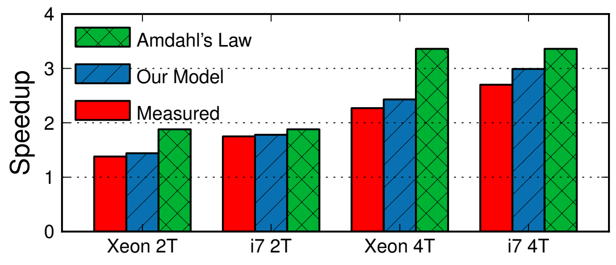

Fig. 2. CPU Validation: geo mean speedup over 1 i7 thread

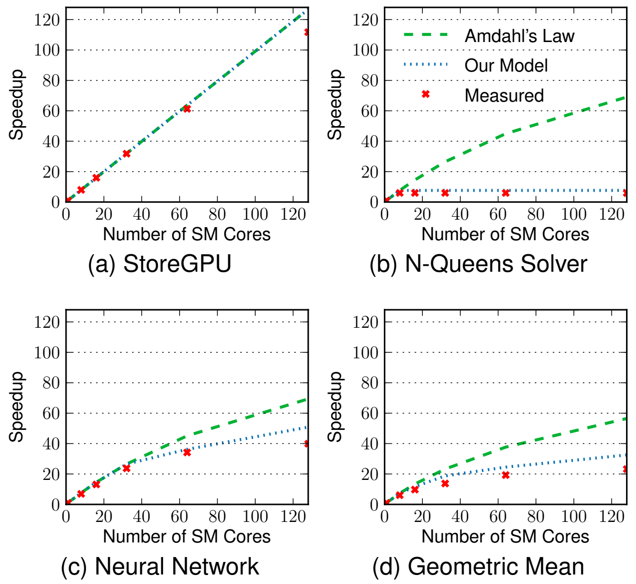

Fig. 3. GPU Validation: speedup over one SM

performance impact, threads never migrate, and thread swap time is negligible. The assumptions cause the model to over-predict performance, making projected speedups optimistic.

Application Behavior: The model optimistically assumes the workload is homogeneous, parallel code sections are infinitely parallelizable, and no thread synchronization, operating system serialization, or swapping overheads occur.

4 Validation

To validate the model, we compare speedup projections from the model to measured and simulated speedups for CPU and GPU organizations. To validate the model's tighter bound, we also find the speedup projections using Amdahl's Law.

For CPU-style cores, we compare the predicted speedup to measured speedups on 11 of the 13 PARSEC benchmarks using either 2 or 4 threads on two four-core chips, a Xeon E5520 and a Core i7 860, as shown in Figure 4. We assume $t_{L1} = 3$ cycles, $t_{L2} = 20$ cycles, and $t_{mem} = 200$ cycles at 3.2Ghz. The model captures the impact of the differing microarchitectures and memory systems, is still optimistic, and provides a tighter bound that the Amdahl's Law projections. We use the average CPI_{exe} approach as described above, and thus report the average speedup over the set of PARSEC benchmarks.

We validate GPU speedup projections comparing speedups generated using the model to those simulated using GPGPUSim (v. 3, PTX mode) [3]. Figure 3 shows results for 7 of GPGPUSim's 12 CUDA benchmarks. The three specific benchmarks shown demonstrate three cases: the highly parallel StoreGPU, N-Queens Solver's limited number of threads, and Neural Network's memory bandwidth saturation. The geometric mean (3d) shows that the model maintains a tighter upper bound on speedup projections than Amdahl's Law projections. We use the number of threads and blocks generated by the CUDA code to deal with occupancy issues beyond the warp level, and estimate f from the speedup between 1 and 32 SMs (assuming perfect memory). f therefore includes the effect of blocking and other serializing behavior. For the GPU model, we use $t_{L1} = 1$ and let t_{mem} be the average memory latency over the entire kernel as reported by GPGPUSim for a 32 SM GPU (varies from 492-608 cycles). These refined inputs improved model accuracy over that in [4]. Our approach is philosophically different than Hong and Kim's approach of using model inputs based on the CUDA program implementation [8]. More detailed qualitative and quantitative comparison is an interesting avenue for future work. For both CPU and GPU chips, if architectural or software changes affected the rate of loads and stores, cache miss rates, cache access latencies, CPI_{exe}, core frequency, or the memory bandwidth, there would be additional improvements over Amdahl's Law projections.

5 Model Utility and Concluding Thoughts

Our first order multicore model projects a tighter upper bound on performance than previous Amdahl's Law based approaches. This extended model incorporates abstracted single threaded core designs, resource constraints, target application parameters, desired architectural features, and additional first order effects—exposing more bottlenecks than previous versions of the model—while remaining simple and flexible enough to be adapted for many applications.

The simple performance/resource constraint trade-offs that are inputs to the model can be derived and used in a diverse set of applications. Examples of the input trade-off functions may be Pareto frontiers like those presented in Azizi et al. [2] and Esmaeilzadeh et al. [4], known designs that an architect is deciding between, or even more abstract trade-off functions like Pollack's Rule [10] used in Hill and Marty [7]. The proposed speedup calculation technique can be applied to even more detailed parallelism models, such as Eyerman and Eeckhout's critical sections model [5]. This complete model can complement simulation based studies and facilitate rapid design space exploration.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. AFIPS, 1996.

[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. ISCA, 2010.

[3] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. ISPASS, 2009.

[4] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon & the end of multicore scaling. ISCA, 2011.

[5] S. Eyerman and L. Eeckhout. Modeling critical sections in Amdahl's law and its implications for multicore design. 2010.

[6] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser. Many-core vs. many-thread machines: Stay away from the valley. *IEEE Computer Architecture Letters*, 2009.

[7] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, July 2008.

[8] S. Hong and H. Kim. An analytic model for a GPU architecture with memory-level and thread-level parallelism awareness. ISCA, 2009.

[9] B. Jacob, P. Chen, S. Silverman, and T. Mudge. An analytic model for designing memory hierarchies. *IEEE ToC*, Oct 1996.

[10] F. Pollack. New microarchitecture challenges in the coming generations of CMOS process technologies. MICRO Keynote, 1999.