Beyond Amdahl's Law: An Objective Function That Links Multiprocessor Performance Gains To Delay and Energy

Andrew S. Cassidy, *Member, IEEE,* and Andreas G. Andreou, *Fellow, IEEE*

Abstract—Beginning with Amdahl's law, we derive a general objective function that links parallel processing performance gains at the system level, to energy and delay in the sub-system microarchitecture structures. The objective function employs parameterized models of computation and communication to represent the characteristics of processors, memories, and communications networks. The interaction of the latter microarchitectural elements defines global system performance in terms of energy-delay cost. Following the derivation, we demonstrate its utility by applying it to the problem of Chip Multi-Processor (CMP) architecture exploration. Given a set of application and architectural parameters, we solve for the optimal CMP architecture for six different architectural optimization examples. We find the parameters that minimize the total system cost, defined by the objective function under the area constraint of a single die. The analytical formulation presented in this paper is general and offers the foundation for the quantitative and rapid evaluation of computer architectures under different constraints including that of single die area.

1 Introduction

Amdahl's law, is a simple and intuitive argument about performance gains in large scale multiprocessor computer systems, that has its origin in a 1967 conference paper by Gene Amdahl [1]. Amdahl's Law is often stated in a more general form, that includes the speedup of computation due to any enhancement (architectural or algorithmic). The key idea is that any speedup is ultimately limited by the fraction of the algorithm that is able to use the enhancement. In the years following Amdahl's conference paper, the original verbal description has been cast into a mathematical equation (see for example [2]). Amdahl's fundamental insight regarding parallel processing, has been successfully applied in many contexts, including the design of High Performance Computing (HPC) systems and off the shelf multiprocessor systems known as "Beowulf" clusters [3]. For a good discussion of Amdahl's Law with insights on its applicability and limitations, see papers by Gustafson [4] and Krishnaprasad [5]. The elegance of Amdahl's simple but powerful argument has more recently motivated scientists and engineers to formulate other simple empirical "rules of thumb" to aid the design of Petascale data-intensive computer architectures [6] also known as "Graywulf" clusters [7]. It is not surprising that with the advent of Chip Multi-Processors (CMPs), with dozens or hundreds and potentially thousand of processor cores [8], computer scientists have applied Amdahl's Law in CMP architecture exploration [9], [10], [11].

CMPs first appeared in research labs in the mid-1990s [12], [13] and by the mid-2000s, major commercial microprocessor manufacturers including IBM, Sun, AMD, and Intel had all released CMP products. This paradigm shift to chip-multiprocessing has brought about new opportunities for high performance and high efficiency computing, but realizing optimal architectures is a challenge for today's computer architects. In particular, as CMPs transition from a few processor cores on a chip to dozens or hundreds, the importance of system-level design increases greatly. Questions that need to be answered include: how should local memories be sized and organized? How many processors are most efficient, or even effective? How can the system keep hundreds of processors fed with data, in order to prevent stalling? What architecture is required to keep data flowing with minimal latency in a network of hundreds of processors? While detailed simulations are an important step towards implementation, high-level analytical methods for system optimization, such as those presented here, play an important role in rapidly narrowing the system design space prior to detailed modeling. They illuminate high-level design trade-offs and present solutions for optimal performance and efficiency.

The next generation of CMPs will include asymmetric or heterogeneous processors of multiple varieties, a complex memory hierarchy, as well as an assortment of application specific algorithm accelerators. In this case, the parts of the system cannot be optimized individually, rather a global optimization approach is required. Hence,

A.S. Cassidy and A.G. Andreou are with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218 USA e-mail: {acassidy, andreou}@jhu.edu
Manuscript received June 30, 2010; revised May 20, 2011

compact analytical models such as Amdahl's Law, have the potential to rapidly narrow the system design space prior to more detailed simulations.

Motivated by the need for a design methodology to address architectural space exploration in CMPs, we derive a simple objective function that links parallel processing in a computer system of N processors, to the costs of energy and delay, the traditional metrics of VLSI [14]. The area of a single die is often a hard physical and economical constraint. Parallel processing systems should be optimized with respect to two performance objectives: speed (inverse of delay) and energy. The root of our model is a cost function formulation of Amdahl's law, that employs parameterized models of computation, communication, energy and delay. After deriving the generalized objective function, we demonstrate its utility by applying it to the problem of CMP architecture exploration where the constraint is the area of a single die. Given a set of application and architectural parameters, we solve for the optimal CMP architecture for six different architectural optimization examples, by finding the architectural parameters that minimize the total system cost. Using our analytical model, we demonstrate methods for solving the constrained optimization problem to find the optimal CMP architecture. A preliminary version of this work was presented in [15] and an example of an application demonstrating the applicability of the proposed methodology in [16].

The theoretical framework presented in this paper can be applied to a wide range of architectures and optimizations, including asymmetric CMPs and shared access structures such as buses, memories, networks, and other communication fabrics. Even though, by way of example, we have focused on CMPs where the area of a single die is the constraint, other constraints such as energy and monetary cost can be employed to explore different dimensions of computer architecture tradeoffs. These principles apply to large scale parallel computing architectures in addition to micro-parallel processors such as CMPs. Because our objective function is built on delay and energy, fundamental costs governed by the laws of physics, our approach can also be used to model and analyze non-traditional computing architectures such as the human cortex.

In Section 2 we begin with the theoretical foundations and derive a general objective function to link multiprocessor gains to delay and energy costs. Models for the different components in the architecture (processor and memory hierarchy) are presented in Section 3. We present concrete results for six architectural optimization examples in Section 4, followed by application of our model to commercially available CMPs in Section 5, discussion in Section 6, and conclusions in Section 7.

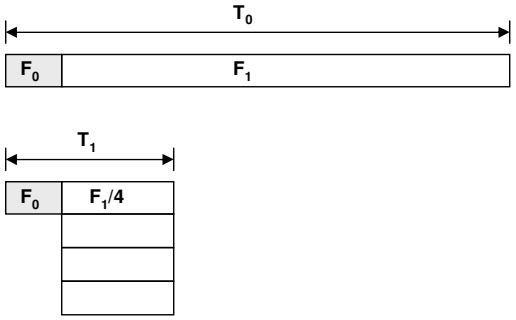

Fig. 1. Timing diagram: symmetric multiprocessing with two degrees of parallelism. (top) Algorithm executed on single processor. (bottom) Algorithm executed in parallel, with the number of processors $N = 4$.

2 THEORETICAL FOUNDATIONS

2.1 Amdahl's Law

The overall speedup SP, as a result of an algorithmic or architectural enhancement in a computing system is given by:

$$SP = \frac{t_{old}}{t_{new}} = \frac{1}{(1 - F_{enh}) + \frac{F_{enh}}{S_{enh}}} \quad (1)$$

where t_{old} and t_{new} are the old and new execution times of the same algorithm on old and new (or enhanced) architectures respectively [2]. F_{enh} is the fraction of the algorithm that is enhanced while $(1 - F_{enh})$ is the non-enhanced fraction of the algorithm. To maximize speedup, two quantities must be maximized: the speedup of the enhancement S_{enh}, and the fraction of the computation that can be enhanced F_{enh}.

When applied to multiprocessor systems, F_{enh} is the parallel fraction of the algorithm, while $(1 - F_{enh})$ is the serial fraction of the algorithm. The enhancement speedup S_{enh}, is the number of processors that the parallel portion of the algorithm is distributed over. The impact of parallel processing on the execution time of an algorithm can be visualized with the help of the timing diagrams in Fig. 1. The bar on top, of length T_0, signifies the time to complete a given algorithm on a single processor. The algorithm is split into two fractions, F_0 designating the serial fraction of the algorithm, and F_1 the parallel fraction of the algorithm. When the parallel fraction of the algorithm is executed on four processors in parallel, the overall execution time is reduced from T_0 to T_1. The speedup of T_1 with respect to T_0 is:

$$SP = \frac{T_0}{T_1} = \frac{1}{F_0 + \frac{F_1}{4}} \quad (2)$$

Equation (1) specifies a quantitative measure of the speedup or performance improvement between two computational architectures, and captures the essence of Amdahl's Law [1] as we know it today.

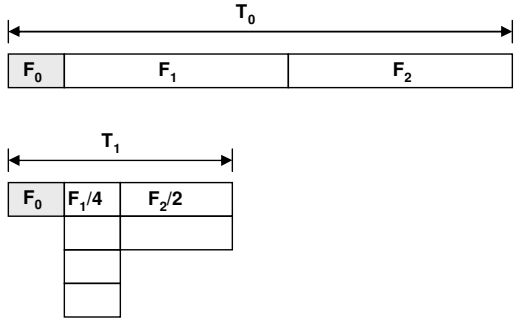

Fig. 2. Timing diagram: symmetric multi-processing with multiple degrees of parallelism. For $F_0, N_0 = 1$ (serial), for $F_1, N_1 = 4$ (parallel), and for $F_2, N_2 = 2$ (parallel).

2.2 Generalizing Degrees of Speedup

In Section 2.1, we have seen the traditional formulation of Amdahl's Law as found in the standard computer architecture textbooks. This formulation (1), partitions the algorithm into two categories, the enhanced fraction and the non-enhanced fraction. The splitting of the algorithms into two categories is somewhat arbitrary. Starting from Amdahl's original intuitive argument, we postulate that an algorithm can be split up into K fractional components F_j, where the sum of the fractions must add up to 1. Each fractional component has an enhancement speedup S_j, associated with that fraction of the algorithm. Furthermore, we note that while we label S_j as "speedup" for values of S_j greater than 1, it is equivalent to a "slowdown" for values of S_j less than 1. This first step is a generalization of Amdahl's Law to incorporate multiple degrees of speedup:

$$SP = \frac{1}{\frac{F_0}{S_0} + \frac{F_1}{S_1} + ... + \frac{F_{K-1}}{S_{K-1}}} = \frac{1}{\sum_{j=0}^{K-1} \frac{F_j}{S_j}} \quad (3)$$

subject to the constraint that $\sum_{j=0}^{K-1} F_j = 1$. We can link the fractions of the algorithms to the instructions executed. Each term in the denominator corresponds to the fraction of instructions F_j executed with a speedup of S_j. Rigorously, the fraction F_j is defined as:

$$F_j = \frac{Q_j}{\sum_j Q_j} \quad (4)$$

where Q_j is the number of instructions executed with the jth speedup and $\sum_j Q_j$ is the total number of instructions.

The timing diagram in Fig. 2 illustrates the effect of multiple degrees of parallelism on the execution time of an algorithm. While some portion of the algorithm is serial ($N_0 = 1$), another portion can be parallelized across four processors ($N_1 = 4$), and a third fraction of the algorithm can only be parallelized across two processors ($N_2 = 2$). In this example, the speedup of T_1 with respect to T_0 is:

$$SP = \frac{T_0}{T_1} = \frac{1}{\frac{F_0}{1} + \frac{F_1}{4} + \frac{F_2}{2}} \quad (5)$$

This generalization to multiple degrees of speedup is important for a few reasons. The fact that applications are common divided into only two fractions (serial and parallel) reduces the power of the model to describe real applications which have multiple phases. For example, the standard Automatic Speech Recognition algorithm has four phases: DSP for the acoustic front end, Gaussian Mixture Modeling (GMM), Hidden Markov Modeling (HMM), and Language Modeling [17]. The DSP phase is generally serial. The GMM phase is almost entirely parallelizable. The HMM and language modeling phases both have serial and parallel aspects. This is a concrete example of multiple levels of parallelism. Moreover, the F_j's could also be "phases" of the program, where the speedup changes based on changing application characteristics. Suppose we have a simple program with two phases, both serial, but phase $j = 0$ is compute dominated, while phase $j = 1$ is memory dominated. In this simple example, the parallel speedups S_0 and S_1 are both 1, but the instruction distributions are vastly different and will execute with different delay and energy. Modeling these effects will become more clear with the full objective function, but the point is that generalization enables modeling the complex distributions of application statistics. Finally, this generalization is important for modeling heterogeneous CMPs. They are not covered here, however they are the subject of forthcoming work, and the principles outlined here lay the foundations for them.

We proceed with a cost minimization approach, where an architecture's cost is the inverse of its speedup. Thus maximizing speedup and minimizing the cost are equivalent.

2.3 Delay Cost

Consider now a model at the processor microarchitectural level. From the definition of expected value [18], for M possible classes of instructions, the expected time (or delay D) for a processor to execute an instruction is:

$$\mathbf{E}[D] = \sum_{i=0}^{M-1} d_i \; p(d_i) \quad (6)$$

where each class of instructions requires d_i time to execute. Execution time depends on the level of the memory hierarchy accessed, functional unit latencies, superscalar instruction level parallelism (ILP), as well as communication latencies. The execution times for each instruction are distributed according to the probability distribution $p(d_i)$. Given the execution trace of specific program on a specific processor for a specific dataset, the probabilities are:

$$p(d_i) = \frac{Q_i}{\sum_{i=0}^{M-1} Q_i} = G_i \quad (7)$$

where Q_i is the number of instructions with the ith delay and $\sum_i Q_i$ is the total number of instructions executed. This is equivalent to the fraction of instructions

G_i executed with the ith delay. These fractions follow the law of probabilities such that: $\sum_{i=0}^{M-1} p(d_i) = 1$ and $\sum_{i=0}^{M-1} G_i = 1$.

Given that our goal is to minimize the expected cost in terms of the delay of the architecture, then we want to minimize the following cost function (changing notation slightly such that $D_i = d_i$):

$$J_D = \sum_{i=0}^{M-1} G_i D_i \quad (8)$$

2.4 Delay Cost with Parallelism

For an ideal parallel processor, program execution is divided equally across the number of parallel processors N. For example, if a set of computations must be performed on a data set, then the same computations could be performed in parallel on N processors, each with $1/N$ of the data set. The time to execute this algorithm is reduced by $1/N$, thus the expected delay is:

$$\mathbf{E}[D] = \sum_{i=0}^{M-1} \frac{d_i}{N}\, p(d_i) = \frac{1}{N} \sum_{i=0}^{M-1} d_i\, p(d_i) \quad (9)$$

and

$$J_D = \frac{1}{N} \sum_{i=0}^{M-1} G_i D_i \quad (10)$$

In a realistic (non-ideal) parallel processor, program execution cannot be perfectly divided across the number of parallel processors N. Rather only some portion of the algorithm can be parallelized (the parallel fraction, F_p), while the remaining portion is executed sequentially (the serial fraction, F_s). The parallel fraction executes with an expected cost of (10), while the serial fraction executes with an expected cost of (8). Thus the total cost is a weighted combination of the parallel and serial fractions:

$$J_D = \frac{F_p}{N} \sum_{i=0}^{M-1} Gp_i Dp_i + \frac{F_s}{1} \sum_{i=0}^{M-1} Gs_i Ds_i \quad (11)$$

where $F_p + F_s = 1$. Gp_i and Gs_i are the instruction distribution fractions with delays Dp_i and Ds_i for the parallel and serial portions of the algorithm respectively. If the algorithm contains an arbitrary number of levels of parallelism, (11) can be generalized:

$$J_D = \sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{i=0}^{M-1} G_{ij} D_{ij} \quad (12)$$

where $\sum_{j=0}^{K-1} F_j = 1$, $N_j = 1$ for the serial fraction of the algorithm, and K is the number of levels of parallelism. A derivation of (12) directly from the expected value of a joint distribution of random variables is given in the Online Supporting Material.

The timing diagram in Fig. 3 depicts the various components of the generalized objective function in (12). At the top level of description we have the algorithm with multiple degrees of parallelism. The fractions of the algorithm with corresponding levels of parallelism are $F_0, N_0 = 1$ (serial), $F_1, N_1 = 4$ (parallel), and $F_2, N_2 = 2$ (parallel) as in Fig. 2. At the second level, these algorithm fractions are further subdivided into delay cost components D_i corresponding to the delay for each instruction to access the register file (R), L1 cache ($L1$), L2 cache ($L2$), or main memory (M). Summing the number of instructions for each delay component and normalizing yields the fraction for each delay category G_{ij}.

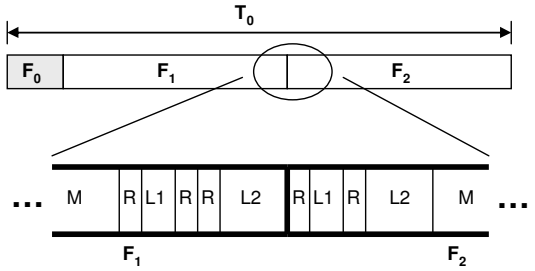

Fig. 3. Timing diagram: algorithmic parallelism fractions (F_j) and corresponding cost components $D_{ij} \in \{R, L1, L2, M\}$.

2.5 Energy Cost

Up to this point, we have considered delay as the primary computing cost to be minimized. Future systems will be optimized in terms of a different metric: the weighted energy-delay product. It has become clear that the primary barrier to realizing Exascale systems is power consumption. Scaling current technology will reach a power barrier before Exascale systems can be realized [19]. Energy consumption is also an important constraint in embedded processors. Hence we now proceed to add energy as a component to the cost function. While the energy-delay product is a commonly used metric [20], we observe that it too is subject to Amdahl's Law. That is to say, any architectural enhancement to reduce energy consumption will only be effective for the fraction of the algorithm that uses that architectural enhancement. We now derive the energy cost function.

Closely following the approach in Sections 2.3 and 2.4, the expected energy consumption for a processor core to execute an instruction is:

$$\mathbf{E}[E] = \sum_{i=0}^{M-1} e_i\, p(e_i) \quad (13)$$

where each class of instructions requires e_i Joules of energy to execute. Similar to the delay case, the execution energy depends on the level of the memory hierarchy accessed, the complexity of the processor core, as well as any communication operations. Based on this correlation, delay and energy costs are naturally grouped into the same instruction classes. Thus the probability distributions for the delay costs and energy costs can be merged for simplicity: $p(d_i) = p(e_i)$, but could also be independent for greater modeling detail. Exchanging the

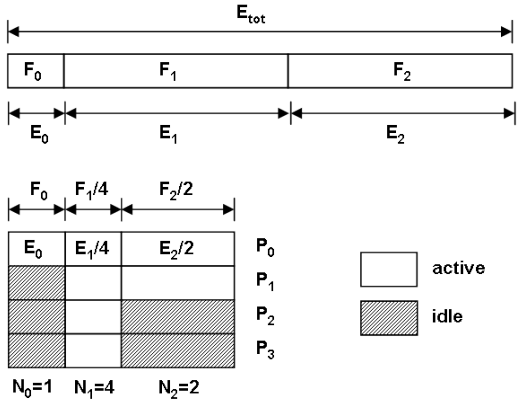

Fig. 4. Energy diagram: symmetric multi-processing with multiple degrees of parallelism. For $F_0, N_0 = 1$ (serial), for $F_1, N_1 = 4$ (parallel), and for $F_2, N_2 = 2$ (parallel).

probabilities for instruction fractions G_i and changing notation slightly such that $E_i = e_i$, we arrive at the expected energy cost for a core:

$$J_E = \sum_{i=0}^{M-1} G_i E_i \quad (14)$$

In order to account for parallel processing energy, consider Fig. 4. There are four processors in the system, and three algorithm fractions (F_0, F_1, F_2). As with the delay case, the fractions of the program F_j are divided by the degree of parallelism N_j, but there are N total processors executing in parallel (some active and some idle). During each phase or fraction of the algorithm there are N_j active processors and $N - N_j$ idle processors. Thus we must multiply by the number of active or idle processors. We sum the contributions from all of the active and idle processors, using $N_{jh} \in \{N_{jA}, N_{jI}\}$ to account for the number of processors that are active or idle. Including parallelism, the energy cost function is:

$$J_E = \sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{h \in \{A,I\}} N_{jh} \sum_{i=0}^{M-1} G_{ijh} E_{ijh} \quad (15)$$

The number of active processors N_{jA} always equals N_j, which cancels with the N_j in the outer summation. Similarly, the number of idle processors N_{jI} always equals $N - N_j$ (but doesn't cancel).

2.6 A Generalized Objective Function

Combining the delay cost function (12) and the energy cost function (15) according to the energy-delay product, we obtain a generalized objective function J_{ED} that links the gains from multiprocessor architecture to delay and energy costs:

$$J_{ED} = \left[\sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{i=0}^{M-1} G_{ij} D_{ij} \right] \times \left[\sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{h \in \{A,I\}} N_{jh} \sum_{i=0}^{M-1} G_{ijh} E_{ijh} \right]^{\gamma} \quad (16)$$

In the outer summations, F_j is the fraction of the algorithm that has parallelism of N_j. Since F_j is a fraction, $\sum_{j=0}^{K-1} F_j$ must equal 1. In the inner summations, each of the algorithm fractions F_j, are subdivided into constituent cost components. G_{ij} is the fraction of F_j that has the ijth cost component D_{ij} or E_{ijh}. The ijth delay is D_{ij} and the ijth energy cost is E_{ijh} for the jth fraction of the algorithm and the active or idle processors, $h \in \{A, I\}$. Since G_{ij} is also a fraction, $\sum_{i=0}^{M-1} G_{ij}$ must equal 1. In the outer summation of the delay term, N_j in the denominator reflects the speedup in delay obtained by parallelizing the algorithm over N processors. In the middle summation of the energy term, N_{jh} is the number of active or idle processors during the jth phase of the algorithm.

Adding an exponential weighting parameter γ to the energy side of the equation allows energy and delay to be unequally weighted. In the realm of energy efficient design, two metrics are typically used for design evaluation: the energy-delay product ED and energy-delay squared ED^2. The energy-delay product equally weights the contribution of delay and energy, while energy-delay squared doubly weights the contribution of delay in order to emphasize performance over energy savings. In our model, using $\gamma = 1$ results in the standard energy-delay product, while with $\gamma = 0.5$, the contribution of delay is twice as large as the contribution of energy, analogous to the energy-delay squared metric.

An alternative method of combining energy and delay is the energy-delay dot product [15]:

$$J_{E \cdot D} = \sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{i=0}^{M-1} G_{ij} D_{ij} (N_j E_{ij})^{\gamma} \quad (17)$$

The energy-delay dot product (17) divides the energy-delay cost into constituent components, such that we optimize the energy-delay product of the constituent components of the architecture. On the other hand, with the strict energy-delay product (16), we optimize the energy-delay product of the overall system architecture. Our goal is to globally optimize the overall system architecture, thus we use the strict energy-delay product (16) in this paper.

2.7 Architecture Design

The hardware/software codesign and optimization process consists of finding the values of the parameters N_j, F_j, G_{ij}, D_{ij}, and E_{ijh} that minimize the objective function (16). The software application/algorithm affects

the parameters F_j, G_{ij}, while the hardware architecture affects N_j, G_{ij}, D_{ij} and E_{ijh}[1]. Formally, the optimization problem is stated as finding the optimal parameter values that minimize the cost function:

$$\{N_j^{opt}, F_j^{opt}, G_{ij}^{opt}, D_{ij}^{opt}, E_{ijh}^{opt}\} = \underset{N_j,F_j,G_{ij},D_{ij},E_{ijh}}{\operatorname{argmin}} J_{ED} \quad (18)$$

In actuality, we do not control all of these parameters directly. Rather, we have design choices about the architecture and architectural elements. We want to minimize the cost function over the set of all possible architectures:

$$\text{ARCH}^{opt} = \underset{\forall \text{ARCH}}{\operatorname{argmin}} J_{ED} \quad (19)$$

If we consider only the hardware architecture (by assuming the algorithm, data set, and thus instruction mix are given), then G_{ij}, D_{ij} and E_{ijh} can be expressed as functions of the architecture. For example, in CMP design where the constraint is the area of a single die, these parameters can be defined as functions of area, as we will see in the next section. The architecture is the area allocated to the processor cores and memory (and communications elements), and the optimization is stated as:

$$\{A_p^{opt}, A_{cache}^{opt}, N_j^{opt}\} = \underset{A_p,A_{cache},N_j}{\operatorname{argmin}} J_{ED} \quad (20)$$

The generalized cost function formulation in (16), together with the low level models of energy and delay that will be summarized in Section 3 constitute the framework that is employed to analytically explore CMP design in Section 4 for the optimal architecture.

3 Low-Level Models: Performance Specifications and Constraints

In the CMP examples that we consider in Section 4, the constraint is the area of a single silicon die. The design variables are: the number of cores in the CMP N, the area of a processor core A_P, and the area of the L2 cache A_{L2}. These variables capture the microarchitecture (i.e. processor core complexity), the memory hierarchy, and the interconnection and relate to the parameters G_{ij}, D_{ij} and E_{ijh} through low level physical and empirical models. These models represent computational and architectural structures, the application/algorithm characteristics, and the physical characteristics. In this section we summarize the underlying computational models and parameters used in our analyses. The parameters for instruction fractions F_j and G_{ij}, delay costs D_{ij}, and energy costs E_{ijh} are summarized in Table 1. (See the Online Supporting Material for estimates of the values of these parameters.)

1. At first it appears that the instruction fractions G_{ij} are a function of only the algorithm and the dataset. However after further thought it becomes apparent that the delay and energy for these fractions are heavily dependant on the memory hierarchy – a key component of the hardware architecture.

TABLE 1
Parameter Definitions

Parameter	Symbol	Value
Parallelism Frac.	F_0	10%
	F_1	90%
Memory Hierarchy Hit Rates	HR_{L1}	95%
	HR_{L2}	95%
	HR_{mem}	100%
Instruction Frac.	G_0	HR_{L1}
	G_1	$(1-HR_{L1})HR_{L2}$
	G_2	$(1-HR_{L1})(1-HR_{L2})HR_{mem}$
Instruction Frac. (Variable A_{L2})	G_1	$(1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})$
	G_2	$(1-G_0)\kappa A_{L2}^{-\frac{1}{2}}$
Memory Access Delay Costs	D_0	$D_{L1} = 1$ cycle
	D_1	$D_{L2} = 10$ cycles
	D_2	$D_{mem} = 200$ cycles
Memory Access Energy Costs	E_{L1}	3.6 pJ
	E_{L2}	18.5 pJ
	E_{mem}	168.5 pJ
Computational Energy Costs	E_{active}	$E_{FPU} + E_{RF} + E_{L1-I} = 19.7$ pJ
	E_{idle}	$E_{L1-I} = 3.6$ pJ
Total Energy Costs	E_0	$E_{active} = 19.7$ pJ
	E_1	$E_{idle} + E_{L2} = 22.1$ pJ
	E_2	$E_{idle} + E_{mem} = 172.1$ pJ

3.1 Area Constraint

For the fixed area constraint, a first order model of the total area utilization on a single die A_{tot} is:

$$A_{tot} = N(A_P + A_{L2}) + A_{fix} \quad (21)$$

where N is the number of cores in the CMP, A_P is area of a processor core, A_{L2} is the area of the L2 cache, and A_{fix} accounts for the fixed area functions (I/O, memory controller, test and debug circuitry, etc.)

3.2 Relationship between Computation and Area

The next subsections detail the functional relationships between the physical quantities of delay and energy and the low level architectural structures in terms of area.

3.2.1 Cache Memory Area Models

Cache size (area) is a variable whose value must be determined as a result of the optimization process. The instruction fractions G_{ij} depend on the hit rates HR at each level of the cache hierarchy. In turn, hit rates are highly dependant on the size of the cache. An approximate rule of thumb for caches is that miss rate MR is inversely proportional to the square root of the size (or area, A_{cache}) of the cache [21]:

$$MR = \frac{\kappa}{\sqrt{A_{cache}}} = \kappa A_{cache}^{-\frac{1}{2}} \quad (22)$$

where the relationship between hit rate and miss rate is: $HR = (1 - MR)$. Fig. 5 shows the change in miss rates as cache size changes, for various values of κ. For example, if an application has a miss rate of 3.125% with a 64KB cache, κ is 8.0, as designated by the black dot in Fig. 5. Recently, Hartstein et al. investigated the theory

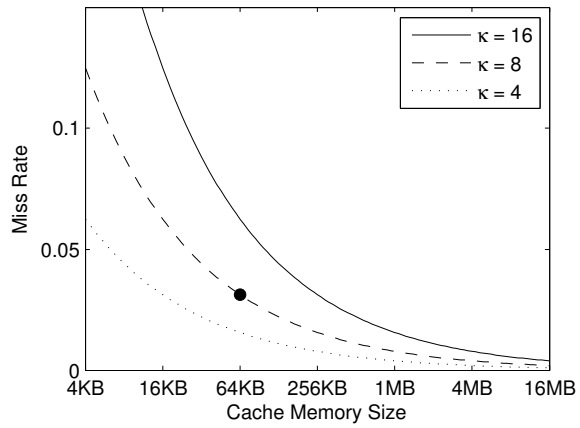

Fig. 5. Cache miss rates as a function of cache size for different values of κ.

and foundation for this power law [22]. Substantiating the relation, they proposed a generalization:

$$MR = \kappa A_{cache}^{-\xi} \quad (23)$$

where ξ takes on values between 0.3 and 0.7. This generalized form could be directly substituted into our approach without difficulty.

3.2.2 Processor Area Models

Pollack's Rule [8], relates the performance of a processor to the square root of its area, or inversely:

$$CPI = \beta A_P^{-\frac{1}{2}} \quad (24)$$

where CPI is cycles-per-instruction, a measure of time necessary to execute one instruction. Pollack's empirical observation captures the effect of microarchitectural techniques, such as those associated with super-scalar architectures (the number of arithmetic or logic functional units, instruction issue width, in vs. out of order execution, etc.) The quadratic growth of superscalar processor area with respect to performance was first observed by Olukotun et al. [12], [23] and used as a rationale for designing chip multiprocessors rather than building superscalar processors of increasingly larger complexity. The parameter β defined as:

$$\beta = \sqrt{A_{P0}} \quad (25)$$

where A_{P0} is the area of a baseline processor with CPI of 1 in the targeted process technology.

Any other differentiable function could be employed to relate processor performance as captured by CPI to the processor area such as a logarithmic or sigmoidal function. For example, a refinement to Pollack's Rule could be used to modestly generalize the relationship between performance and processor area, by adding an additional parameter, ζ_p:

$$CPI = \beta A_P^{-\zeta_p} \quad (26)$$

3.3 Relationship between Computation and Energy

The system dynamic power is a measure of the energy expended during computation and communication. The traditional method for estimating microprocessor power consumption is using instruction set simulators (ISS) with integrated energy models such as: Wattch [24], powerTimer [25], and simplePower [26]. The CACTI simulator [27], [28] is an ISS for modeling cache memories (including energy). These energy estimators incorporate detailed energy models for specific functional units with explicitly defined logic structures. Power is then estimated from the transition frequencies observed while running a specific application on the ISS. While detailed and accurate, this approach is rather time consuming for high-level design exploration.

Without developing detailed architectural models, we can still make intelligent assumptions about energy consumption. For example, a basic model from Su and Despain [29] breaks cache energy into three constituent components: decoding energy, array energy, and I/O energy. Array energy dominates the other components and is itself proportional to the word line size times the bit line size. Thus, cache memory energy consumption is linearly proportional to the area (and thus size) of the cache. Kamble and Ghose [30], [31] created a more detailed model, analyzing the capacitances in the bit lines, word lines, input and output lines. The result, however, is the same. The bit line energy, which accounts for between 80 and 98% of the energy dissipation [31], is proportional to $N_{rows} \times N_{bit}$. Thus cache memory energy dissipation scales linearly with cache area. If cache sub-banking is used, the bitline capacitance is proportional to $N_{rows} \times K_{bit}$, where K_{bit} is constant assuming that the number of sub-banks increases as the cache size increases. Using this assumption, cache memory energy dissipation scales with the square root of cache area:

$$E_{cache} = \rho_{cache} A_{cache}^{\frac{1}{2}} \quad (27)$$

where $\rho_{cache} = E_{M_0}/A_{M_0}$ is the energy to access a baseline cache memory.

Industry data provide further insight into high-level energy trends for processor cores. Pollack states "power is proportional to die-area $\times$ frequency" [32], which implies a linear scaling of energy versus microprocessor area. This is intuitively based on the proportional relationship between die area and capacitance. Importantly however, Pollack also notes that static memory has an order of magnitude lower active power (and lower leakage power) per area than logic. This ratio directly affects the design trade-off between cache size and processor area. In our analyses, we also use the relation that the energy of the processor is linearly proportional to the size of the processor:

$$E_P = \rho_P A_P \quad (28)$$

where the constant $\rho_P = E_{P_0}/A_{P_0}$ is derived from the energy of a baseline (minimum sized) processor.

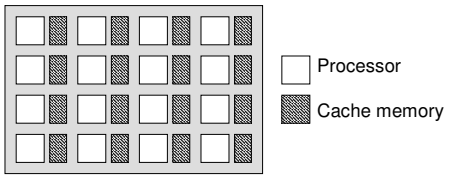

Fig. 6. Baseline symmetric CMP block diagram: each core has a dedicated L2 cache (N=16)

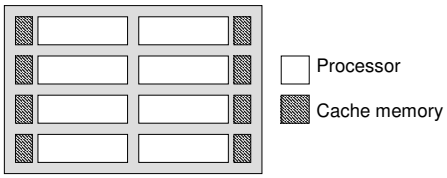

Fig. 7. Symmetric CMP block diagram: trading off the number of processor cores ($N = 8$) and the area of each processor core (A_P).

4 CMP Design Examples

In this section, we use the generalized objective function derived in Section 2 to explore the architecture design space for symmetric CMPs. Six examples build from simple to progressively more complex, demonstrating the increasing architectural complexity that can be modeled using the generalized objective function. The first example examines the tradeoffs of processor complexity versus the number of processors. The second example optimizes the size/area of the L2 cache versus the number of processors. The third example combines the analyses and optimizes both the processor performance, the L2 cache size, and the number of processors. The fourth through sixth examples repeat the first three, while including energy during optimization.

For quick conversion between processor area and cache memory area, as well as the convenient logarithmic representation, we represent all area values (processor, cache memory) in our examples in terms of bytes. That is, the area equivalent to a memory of that number of bytes. For example, a processor of size 2^{16} is the same area as a 64kB cache memory. We assume a linear scaling of memory size and area with a conversion of $7.5mm^2$ per 1MB in a 45nm process technology. See the Online Supporting Material for estimates of the delay, energy, and area values used in the following optimization examples. In addition, the Online Supporting Material contains the full mathematical details for each of the following optimization examples.

4.1 Processor area vs. number of processors (A_P vs. N)

In our first example, we apply the cost function to optimizing the size (or performance) of each processor core and number of processors in a symmetric CMP, while holding the total area of the chip constant. Figs. 6 and 7 depict the area tradeoff between a larger quantity of smaller processor cores and fewer more powerful (but larger) processor cores.

In this example, the algorithm is divided into two fractions, the serial fraction and the parallel fraction, thus $K = 2$. Instead of enumerating the computational costs over i, we use CPI as an aggregate measure of computational cost, therefore:

$$CPI_j = \sum_{i=0}^{M-1} G_{ij} D_{ij} \tag{29}$$

Assuming the instruction mix is the same for the parallel and serial fractions of the algorithm and a symmetric CMP, then $CPI_0 = CPI_1$. In this example we do not consider energy ($\gamma = 0$), thus the cost function is J_D, where the suffix denotes delay and has units of cycles. From the generalized cost function (16), we derive the cost function specific to our optimization:

$$J_D = \sum_{j=0,1} F_j N_j^{-1} \sum_{i=0}^{M-1} G_{ij} D_{ij} \tag{30}$$

$$= \frac{F_0}{1} CPI_0 + \frac{F_1}{N} CPI_1 = \left(F_0 + \frac{F_1}{N} \right) CPI \tag{31}$$

where N is the number of cores in the CMP. To perform the optimization, we constrain the total area of the chip to be fixed (21). Rearranging, we obtain an expression for N:

$$N = \frac{A_{tot} - A_{fix}}{A_P + A_{L2}} \tag{32}$$

We substitute this expression for N into (31) and for CPI we use Pollack's Rule (24), resulting in:

$$J_D = \left[F_0 + \frac{F_1 (A_P + A_{L2})}{(A_{tot} - A_{fix})} \right] \beta A_P^{-\frac{1}{2}} \tag{33}$$

Choosing $\beta = A_{P0}^{\frac{1}{2}}$, the inverse cost function $(1/J_D)$ is equivalent to the speedup over a single processor baseline $P0$. The speedup is plotted in Fig. 8 for different values of F_1, which correspond to algorithms with different parallelism characteristics.

Returning to the formula for the objective function, the optimal core area A_P and number of cores N, are determined by finding the peak in the speedup $(1/J_D)$ curve. By differentiating the objective function (33) with respect to A_P we get:

$$\frac{dJ_D}{dA_P} = -\frac{1}{2} \left(F_0 + \frac{F_1 A_{L2}}{A_{tot} - A_{fix}} \right) \beta A_P^{-\frac{3}{2}} + \frac{F_1}{2(A_{tot} - A_{fix})} \beta A_P^{-\frac{1}{2}} = 0 \tag{34}$$

After a bit of algebra:

$$A_P = \frac{F_0 (A_{tot} - A_{fix})}{F_1} + A_{L2} \tag{35}$$

Solving 35, the optimal processor area A_P for each algorithm (F_1 value) is designated by the black dot in Fig. 8 at the peak of the speedup $(1/J_D)$ curve.

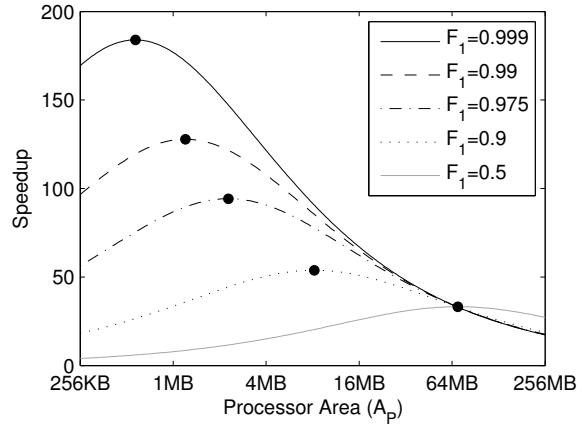

Fig. 8. Speedup ($1/J_D$) for different values of F_1, assuming non-zero A_{fix} and A_{L2}.

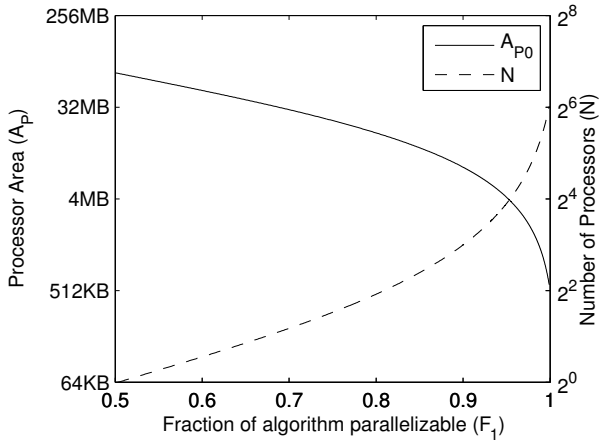

Fig. 9. Optimal processor area (A_P) and number of parallel cores (N).

After calculating the optimal processor area, A_P, the corresponding optimal number of cores N is found using (32). Using this method, we can solve for the optimal architectural parameters over a range of values of F_1, the parallel fraction of the algorithm. This corresponds to applications or algorithms with different levels of parallelism. Sweeping F_1, the optimal values of A_P and N are plotted in Fig. 9. Intuitively, as the parallel fraction of the algorithm F_1 increases towards 1.0, the optimal architecture becomes more cores of smaller size.

4.2 L2 cache area vs. number of processors (A_{L2} vs. N)

An alternative optimization problem is to trade off the number of cores and the L2 cache size of each core, given a fixed total chip area for a symmetric CMP. This tradeoff is shown in Fig. 10 (with respect to Fig. 6). The number of cores in the CMP is N, and the area constraint is defined in (21). We also neglect energy in this example. Our goal is to minimize the cost of the architecture in terms of delay, by trading off the parallel

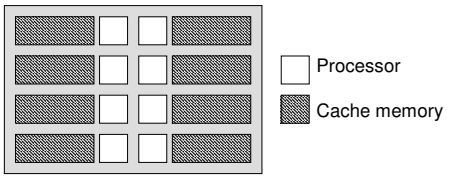

Fig. 10. Symmetric CMP block diagram: trading off the number of processor cores ($N = 8$) and the L2 cache area (A_{L2}).

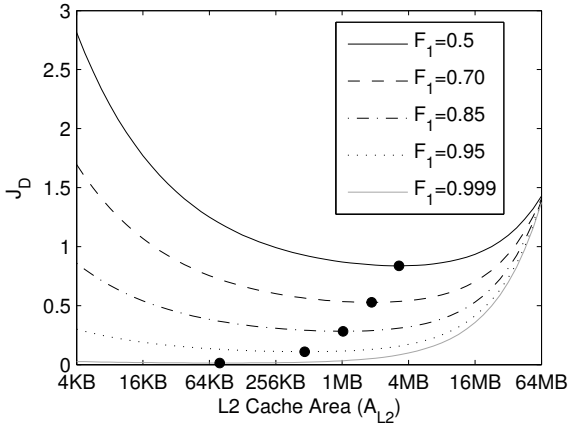

Fig. 11. Objective function J_D over the range of L2 cache sizes (A_{L2}). Black dot designates the minimum cost and therefore optimum L2 cache size. J_D is a time value with units of cycles (per instruction).

performance gain with the gains due to increasing L2 cache size, subject to the constraint that the total area A_{tot} is constant. The optimization is formulated with the following parameters: $K = 2$, $M = 3$ and $\gamma = 0$. From (16) and the variable L2 cache size and hit rate (22), the cost function for this optimization is:

$$J_D = \left(F_0 + \frac{F_1}{N}\right)\Big[G_0 D_0 + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})D_1 + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} D_2\Big] \quad (36)$$

The first term, $G_0 D_0$, accounts for the instructions (and data) that hit in the register file or the L1 cache. The second term, $(1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})D_1$, is the fraction of instructions that hit in the L2 cache. And the final term, $(1-G_0)\kappa A_{L2}^{-\frac{1}{2}} D_2$, is the fraction of instructions that miss in the L2 cache and must go to main memory. The fraction of the algorithm that is parallelizable over multiple cores is F_1, while the serial fraction is $F_0 = (1-F_1)$, and correspondingly, $N_1 = N$ while $N_0 = 1$. Fig. 11 depicts the cost curve J_D for several values of F_1, over the range of the possible of values A_{L2}. The closed form solution for the optimum is found using the method of Lagrange multipliers. We minimize the Lagrangian:

$$L(N, A_{L2}, \lambda) = J_D + \lambda[N(A_P + A_{L2}) + A_{fix} - A_{tot}] \quad (37)$$

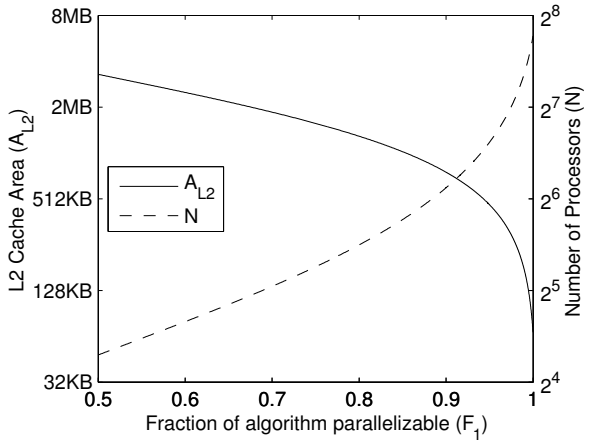

Fig. 12. Optimal L2 cache size and number of cores for various values of F_1, the parallelizable fraction of the algorithm.

where J_D is given in (36). Differentiating (37) with respect to N, λ, A_{L2}:

$$\begin{aligned}\frac{\partial L}{\partial N} &= -N^{-2}F_1\left[G_0D_0+(1-G_0)D_1+\right.\\ &\quad \left.(1-G_0)(D_2-D_1)\kappa A_{L2}^{-\frac{1}{2}}\right]+\\ &\quad \lambda(A_P+A_{L2})=0 \qquad (38)\\ \frac{\partial L}{\partial \lambda} &= N(A_P+A_{L2})+A_{fix}-A_{tot}=0 \qquad (39)\\ \frac{\partial L}{\partial A_{L2}} &= \left(F_0+\frac{F_1}{N}\right)\left[(1-G_0)(D_2-D_1)\frac{-\kappa}{2}A_{L2}^{-\frac{3}{2}}\right]+\\ &\quad \lambda(N)=0 \qquad (40)\end{aligned}$$

Now we have three equations (38, 39, 40) and three unknowns N, λ, A_{L2}. Substituting and simplifying to solve the system of equations gives us a closed form expression for A_{L2} (full mathematical details are given in the Online Supporting Material):

$$\begin{aligned}0 &= [F_1A_P+(A_{tot}-A_{fix})(1-F_1)]\\ &\quad [(1-G_0)(D_2-D_1)]\frac{-\kappa}{2}+\\ &\quad [F_1(1-G_0)(D_2-D_1)]\frac{\kappa}{2}A_{L2}+\\ &\quad F_1\left[G_0D_0+(1-G_0)D_1\right]A_{L2}^{\frac{3}{2}} \qquad (41)\end{aligned}$$

Equation (41) is a polynomial function of A_{L2} which can be solved for the optimal A_{L2} with numerical methods[2]. Then using the optimal A_{L2} value, we can find the corresponding optimal number of cores N for the architecture. The optimum L2 cache size for each cost curve is designated by the black dots in Fig. 11. Fig. 12 shows the optimization results for a range of values of F_1, the parallel fraction of the algorithm. The solutions vary widely based on the characteristics of the algorithm, with the optimal L2 cache size varying over 64x (approximately 64KB to 4MB) as the parallel fraction of the algorithm increases. This shows that if parallelism is available, increasing the number of cores improves performance over increasing the cache memory size.

2. We utilized the 'fzero' command in MatLab. Given a function to solve, and two endpoints over which the function is of opposite signs, this command finds the zero within the range.

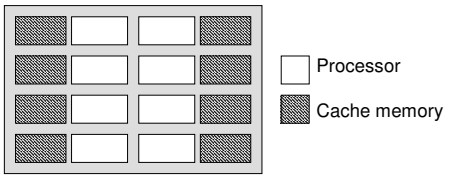

Fig. 13. Symmetric CMP block diagram: trading off the number of processor cores ($N = 8$) with the area of each processor core (A_P) and the L2 cache area (A_{L2}).

4.3 Processor area vs. L2 cache area vs. number of processors (A_P vs. A_{L2} vs. N)

In the previous two examples we have separately demonstrated processor optimization and cache memory optimization, dealt with as independent optimization problems. In this example, we present an example of a joint optimization of number of cores, processor area, and L2 cache memory area. Fig. 13 depicts the tradeoff between processor core size and cache memory area for a variable number of cores (with respect to Fig. 6).

We begin again by assuming that the algorithm has only two fractions, a serial fraction and a parallel fraction (K=2) and $CPI_0 = CPI_1$. We modify the assumption that $CPI_j = \sum_{i=0}^{M-1} G_{ij}D_{ij}$, and instead assume that $CPI_j = G_{0j}D_{0j}$, that is, the performance of the processor using only L1 cache. We add an L2 cache of variable size as part of the optimization, as well as main memory. Thus, $M = 3$. From the generalized cost function (16), we derive the cost function specific to our optimization:

$$\begin{aligned}J_D = \left(F_0+\frac{F_1}{N}\right)\Big[&G_0\beta A_P^{-\frac{1}{2}}+(1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})D_1+\\ &(1-G_0)(\kappa A_{L2}^{-\frac{1}{2}})D_2\Big] \qquad (42)\end{aligned}$$

Using the fixed total chip area constraint (21), the Lagrangian is:

$$L(N, A_{L2}, \lambda) = J_D + \lambda[N(A_P + A_{L2}) + A_{fix} - A_{tot}] \qquad (43)$$

As in the previous example, we differentiate the Lagrangian: $\frac{\partial L}{\partial A_P}, \frac{\partial L}{\partial A_{L2}}, \frac{\partial L}{\partial N}, \frac{\partial L}{\partial \lambda}$, algebraically simplify the system of four equations and four unknowns, and solve for the optimal architecture parameters A_P, A_{L2}, N using numerical methods. Full mathematical details are given in the Online Supporting Material. Fig. 14 depicts the cost surface J_D (42) for various values of A_P and A_{L2}. N takes on the value that satisfies the fixed area constraint. The optimum architecture is designated by the black dot. The thick solid line represents the boundary condition for valid architectures. If we solve for the optimum architecture for multiple values of F_1, we obtain a curve representing the optimum architecture for different amounts of application parallelism, shown

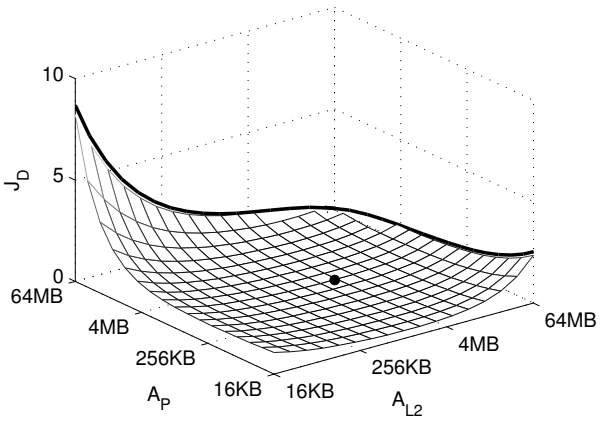

Fig. 14. Objective function J_D for processor area (A_P) and L2 cache area (A_{L2}). Units of area in memory byte equivalent units. J_D is a time value with units of cycles (per instruction).

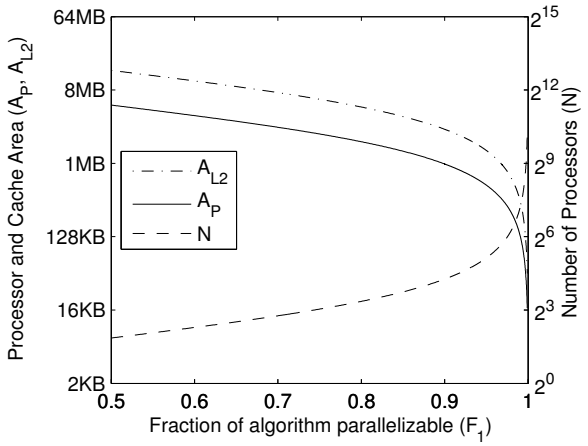

Fig. 15. Optimal area in memory byte equivalent units for processor (A_P), L2 cache (A_{L2}), and number of processor cores (N).

in Fig. 15. As the parallelism increases, the number of parallel cores N increases, and as a result A_P and A_{L2} must decrease. The relative scaling of A_P and A_{L2} is given by β and κ, and they both decrease at the same rate (as N increases) due to their performance both scaling by the power of $-\frac{1}{2}$, as specified by the underlying models (22) and (24). This suggests a constant balance between processor size and cache memory size as parallelism increases.

4.4 Processor area vs. number of processors with energy cost (A_P vs. N)

Up to this point, the optimization examples have neglected energy, focusing only on minimizing delay. We now reanalyze the three earlier examples, this time with energy included in the analysis. First, we reanalyze the tradeoff between the number of processor cores N and the processor area A_P for a symmetric CMP, as presented in Section 4.1. The cost function is derived as follows. For delay, we use (31) and for energy, we expand (15) for $K = 2, M = 1, G_{0jh} = 1.0$:

$$J_E = \sum_{j=0}^{K-1} \frac{F_j}{N_j} \sum_{h\in\{A,I\}} N_{jh} \sum_{i=0}^{M-1} G_{ijh} E_{ijh} \quad (47)$$

$$= \frac{F_0}{N_0}(N_0 E_A + (N - N_0)E_I) + \frac{F_1}{N_1}(N_1 E_A + (N - N_1)E_I) \quad (48)$$

In the serial fraction of the algorithm F_0, there is one active core $N_{0A} = N_0 = 1$, and $N_{0I} = N - N_0 = N - 1$ idle cores. In the parallel fraction of the algorithm F_1, all N cores are active: $N_{1A} = N_1 = N$ and no idle cores $N_{1I} = N - N_1 = 0$. For processor energy, we use the linear relation between processor area and energy (28). Processor energy is divided into active energy $E_A = \rho_{pA} A_P$, and idle energy $E_I = \rho_{pI} A_P$.

$$J_E = F_0 \frac{1}{1}(E_{L1} + \rho_{pA} A_P) + F_0 \frac{(N-1)}{1} \rho_{pI} A_P + F_1 \frac{N}{N}(E_{L1} + \rho_{pA} A_P) \quad (49)$$

Combining delay (31) and energy (49), as specified by the generalized cost function (16), the resulting cost function is given in (44) in Fig. 16. The Lagrangian is:

$$L(N, A_P, \lambda) = J_{ED} + \lambda[N(A_P + A_{L2}) + A_{fix} - A_{tot}] \quad (50)$$

Once again we solve this optimization problem using the method of Lagrange multipliers. After differentiating the Lagrangian, we have three equations with three unknowns (A_P, N, λ). However, in this case we cannot simplify the resulting equations with variable substitutions and the closed form solution for the optimal architecture is intractable. As a result, to find the minimum of the cost curve, we employ Newton's method. First he variables and the partial derivatives of the Lagrangian are aggregated into matrices:

$$\mathbb{X} = [A_P, N, \lambda]^T \quad , \quad \mathbb{F} = \left[\frac{\partial L}{\partial A_P}, \frac{\partial L}{\partial N}, \frac{\partial L}{\partial \lambda}\right] \quad (51)$$

as well as the Jacobian of $\mathbb{F}$:

$$\mathbb{J} = \begin{bmatrix} \frac{\partial \mathbf{F}1}{\partial A_P} & \frac{\partial \mathbf{F}1}{\partial N} & \frac{\partial \mathbf{F}1}{\partial \lambda} \\ \frac{\partial \mathbf{F}2}{\partial A_P} & \frac{\partial \mathbf{F}2}{\partial N} & \frac{\partial \mathbf{F}2}{\partial \lambda} \\ \frac{\partial \mathbf{F}3}{\partial A_P} & \frac{\partial \mathbf{F}3}{\partial N} & \frac{\partial \mathbf{F}3}{\partial \lambda} \end{bmatrix} \quad (52)$$

Then the update rule for the variables $\mathbb{X}$ for each iteration k of Newton's method is:

$$\mathbb{X}(k+1) = \mathbb{X}(k) - (\mathbb{F}/\mathbb{J})^T \quad (53)$$

After a finite number of iterations, the optimal architectural values are contained in $\mathbb{X}$.

We repeatedly solve for the optimum using various values of F_1, the parallel fraction of the algorithm. The

$$J_{ED} = \left[\left(F_0 + \frac{F_1}{N}\right)\beta A_P^{-\frac{1}{2}}\right] \times [F_0(1)(E_{L1} + \rho_{pA}A_P) + F_0(N-1)\rho_{pI}A_P + F_1(E_{L1} + \rho_{pA}A_P)]^{\gamma} \quad (44)$$

$$J_{ED} = \left[\left(F_0 + \frac{F_1}{N}\right)\left(G_0 D_0 + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})D_1 + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} D_2\right)\right] \times \left[F_0(1)\left(G_0 E_0 + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})(E_I + \rho_{L2}A_{L2}^{\frac{1}{2}}) + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} E_2\right) + F_0(N-1)E_I + F_1\left(G_0 E_0 + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})(E_I + \rho_{L2}A_{L2}^{\frac{1}{2}}) + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} E_2\right)\right]^{\gamma} \quad (45)$$

$$J_{ED} = \left[\left(F_0 + \frac{F_1}{N}\right)\left(G_0\beta A_P^{-\frac{1}{2}} + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})D_1 + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} D_2\right)\right] \times \left[F_0(1)\left(G_0\rho_{pA}A_P + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})(E_I + \rho_{L2}A_{L2}^{\frac{1}{2}}) + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} E_2\right) + F_0(N-1)\rho_{pI}A_P + F_1\left(G_0\rho_{pA}A_P + (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}})(E_I + \rho_{L2}A_{L2}^{\frac{1}{2}}) + (1-G_0)\kappa A_{L2}^{-\frac{1}{2}} E_2\right)\right]^{\gamma} \quad (46)$$

Fig. 16. Cost functions J_{ED} for the examples in Sections 4.4–4.6.

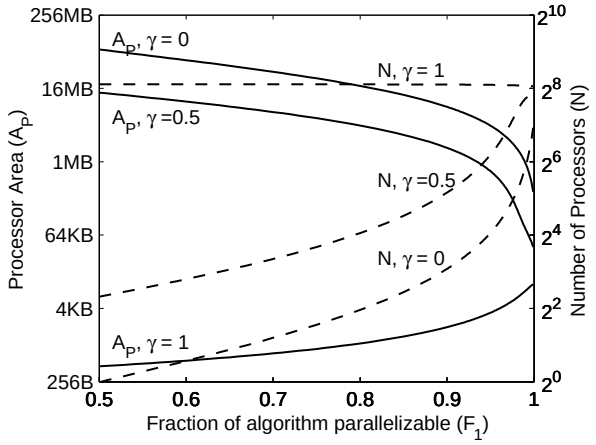

Fig. 17. Optimal processor size A_P and number of processor cores N as a function of the parallel fraction of the algorithm F_1, for increasing γ. Units of area in memory byte equivalent units.

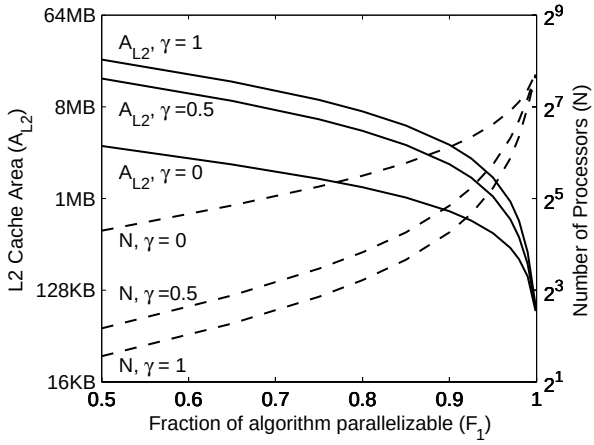

Fig. 18. Optimal L2 cache size A_{L2} and number of processor cores N as a function of the parallel fraction of the algorithm F_1, for increasing γ. Units of area in memory byte equivalent units.

solution for increasing values of γ are plotted in Fig. 17. As the importance of energy increases in the weighting of the cost function, the optimal architecture shifts to minimize the area (and complexity) of the processor core A_P and increase the number of parallel computational units N. This implies that as the importance of energy increases, performance gained from parallelism is more energy efficient than performance gained from advanced microarchitectural techniques (that increase A_P). As a result, CMPs will have more processor cores that are smaller and simpler, consuming less power per processor core and deriving increased performance from parallelism. This makes sense as larger processors get linearly more expensive in terms of power, but only faster by the power of $\frac{1}{2}$. Similarly, increasing the number of cores N has linear improvement in speed for the parallel fraction of the algorithm, but has small effect on the energy (moderately increasing the wasted idle power).

4.5 L2 cache area vs. number of processors with energy cost (A_{L2} vs. N)

In this example, we reanalyze the tradeoff between the number of processor cores N and the L2 cache size A_{L2} for a symmetric CMP, as presented without energy in Section 4.2. We use the relationship that cache energy is proportional to the square root of cache area (27) and processor energy during the cache access is constant (E_{idle}). The cost function specific to this optimization is given in (45). The solution for the optimal architecture is once again found using the method of Lagrange multipliers and Newton's method. Full mathematical details are found in the Online Supporting Material.

Solving for the optimal architecture using various values of F_1, the parallel fraction of the algorithm, and plot the solutions are plotted in Fig. 18 for $\gamma = 0, 0.5,$ and 1. As energy becomes more important in the optimization, cache memories become larger and larger and

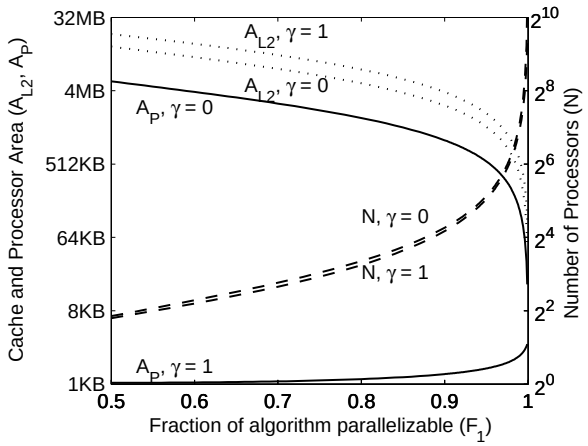

Fig. 19. Optimal areas for processor (A_P), L2 cache (A_{L2}), and number of processor cores (N) as a function of the parallelism in the algorithm (F_1), for $\gamma = 0.0$ and $\gamma = 1.0$. Units of area in memory byte equivalent units.

the architecture has fewer cores. There are three reasons for this: cache memories have lower power consumption than processor logic per area so that decreasing the number of cores decreases the area devoted to power hungry cores, cache memories scale favorably in terms of energy–as the square root of area, and finally, cache memories improve the hit rate, reducing the number of energy-expensive off-chip accesses that occur.

4.6 Processor area vs. L2 cache area vs. number of processors with energy cost (A_P vs. A_{L2} vs. N)

Finally, we reanalyze the case presented in Section 4.3, including energy in finding the optimal tradeoff between the number of processor cores N, the area of the processor core A_P, and the area of the L2 cache A_{L2}. Using (28) for processor energy and (27) for L2 cache energy results in the cost function specific to this optimization (45), given above. Using the method of Lagrange multipliers, we differentiate the Lagrangian and solve four equations with four unknowns (N, A_P, A_{L2}, λ). This is followed by Newton's method to solve for the optimal number of processors N, processor area A_P, and L2 cache size A_{L2}, as detailed in the Online Supplemental Material.

For a range of values of F_1, the optimal values of A_P, A_{L2}, and N are plotted in Fig. 19 for values of $\gamma = 0$ and 1. The curves reinforce the trends seen in the previous two sections. Processor size is minimized and cache size increases as energy is taken into account in the optimization. This is because processor energy scales linearly with processor area, while cache memory energy scales with the square root of area. Unlike the previous examples however, the number of cores changes little, since memory and processor core size can be traded off directly and not just by changing the number of cores to satisfy the fixed area constraint.

TABLE 2
Empirical CMP Data

#	Year	nm	Name	Cores	Ref.
1	2004	90	Intel Pentium 4	1	[33]
2	2005	90	Sun UltraSPARC	1	[34]
3	2005	130	Intel Itanium	1	[35]
4	2004	90	AMD Athlon 64	2	[36]
5	2007	65	AMD Phenom	4	[36]
6	2009	45	AMD Phenom II	4	[36]
7	2006	65	Intel Core 2 (Conroe)	2	[37]
8	2006	65	Intel Core 2 (Kentsfield)	4	[37]
9	2008	45	Intel Core i7 (Nehelem)	4	[37]
10	2001	180	IBM Power 4	2	[38]
11	2004	130	IBM Power 5	2	[39]
12	2007	65	IBM Power 6	2	[40]
13	2007	65	AMD Opteron	4	[36], [41]
14	2009	65	Intel Itanium 2	4	[42]
15	2005	90	Sun Niagara	8	[43]
16	2007	65	Sun 2nd gen. SPARC	8	[44], [45]
17	2009	65	Sun 3rd gen. SPARC	16	[46]
18	2007	90	Azul Systems Vega 2*	48	[47]
19	2005	90	IBM Xenon (Xbox360)*	3	[48]
20	2006	90	Cell processor (PS3)	9	[49], [50]
21	2010	45	Intel Larrabee‡	32	[51]
22	2000	280	Intel IXP1200	7	[37]
23	2002	180	Intel IXP240x*	9	[37]
24	2002	130	Intel IXP280x*	17	[37]
25	2008	65	Intel Teraflops chip	80	[52]
26	2007	90	Tilera TILE64*	64	[53]
27	2006	130	ClearSpeed CSX600*	96	[54]
28	2008	90	ClearSpeed CSX700*	192	[54]
29	2008	90	Nvidia GeForce 9†	128	[55]
30	2009	65	Nvidia GeForce 200†	240	[56]
31	2007	55	AMD Radeon 2900 XT	320	[36]
32	2008	55	AMD Radeon 4870 XT†	800	[36]
33	2008	130	Storm-I Stream Processor	82	[57]

Notes:
* The total chip area was not reported, so it was estimated from the reported number of transistors in the chip, using an average number of transistors per mm^2 for the particular process technology.
† Memory size (in MB) is estimated, extrapolating from earlier GPUs.
‡ Neither chip area nor number of transistors per chip was reported, so total area was estimated from wafer photographs.

5 Results

As one test of our approach, we consider the tradeoff between processor area, on-chip memory area, and the number of processor cores, assuming a symmetric CMP. Here we compare the theoretical predictions from our objective function with a sample set of actual commercial or research CMPs. The result, shown in Fig. 20, is a plot of the normalized memory per core versus the number of cores, for each CMP architecture. The solid line indicates the optimal memory per core as predicted by our theoretical model, without including energy costs. It was generated from the solution given in Section 4.3.

The sample set contains CMPs from many classes: 1–18 are desktop and server CMPs, 19–21 are game/graphics chips, 22–24 are network processors, 25–28 are processor arrays, and 29–33 are graphics processing units (GPUs). Table 2 lists the full cross listing of CMP architecture and corresponding numerical index. The area of each chip was scaled to a 90nm process equivalent. The normalized memory per core is measured as a percentage of total

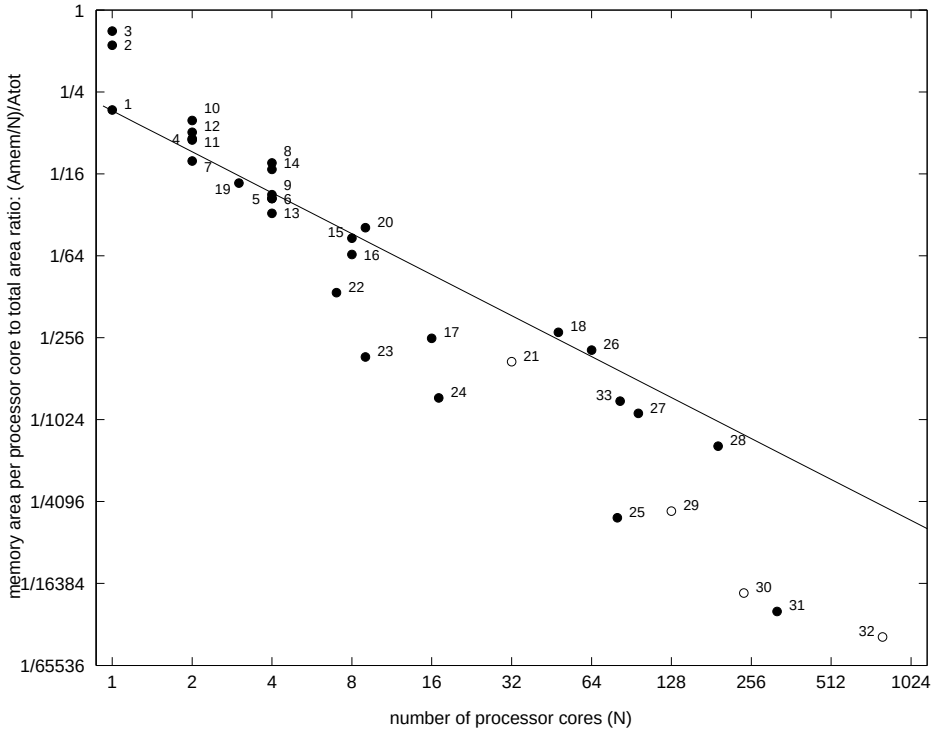

Fig. 20. Log-log plot of normalized memory per core vs. number of processor cores. Open circles signify estimated data points. The solid line indicates the optimal memory per core as predicted by our theoretical model, without including energy costs. Table 2 lists the corresponding numerical index for each CMP.

chip area: $A_{mem}/(A_{tot}N)$ where A_{mem} is the total cache memory area, A_{tot} is the total chip area, and N is the number of cores.

In Fig. 20, many designs fit the predicted optimal architecture curve (within a factor of 2x above and below the curve). This demonstrates the ability of our approach to capture the high-level tradeoffs between computation and memory in real systems. Architectures that fall below the fit line, indicate they are under-capitalized in terms of memory per processor core for general purpose computing applications. Observing that 22–24 are network processors and 29–33 are GPUs, it is reasonable to claim that network processors and GPUs are not general purpose processors and should not be expected to conform to the predictions of a general purpose model. However, there is a significant movement afoot to use GPUs for scientific computing, the goal of Nvidia's CUDA and AMD's CAL frameworks. Thus, although GPU architectures may be optimal for graphics applications, our results contend that GPU architectures are less than optimal for general purpose and scientific computations. These architectures would benefit from a factor of 4 to 8 increase in memory per core if used for non-graphics computation. The Storm-I stream processor architecture, 33 and designs from Clearspeed, 27–28 appear to be a more promising alternatives for this class of applications. Another CMP that is particularly sub-optimal in terms of memory per processor core is Intel's Teraflops research chip, 25. In contrast, the architectures from Tilera, 26 and Azul Systems, 18 have a far more optimal allocation of memory per processor core.

6 Discussion

Today microprocessor design is largely based on simulators [58], [59], [60], [61], [62]. While this approach has served the computer architecture design community well, the recent advances in CMPs technology poses serious challenges to the community. In response, researchers have begun to investigate analytical methods for optimizing CMP architectures. Notably, work by Hill and Marty [9] use a measure of processor performance to augment Amdahl's Law, and apply it to symmetric, asymmetric, and dynamic multicore processors in order to quantitatively compare CMP architectures. Our model is broader, incorporating more architectural design elements such as memory hierarchy and communication contention, in addition to processor performance. Our model also allows the inclusion of more detailed models for more accurate system modeling. In addition, after a few algebraic transformations and appropriate parameters, we can show that Hill and Marty's expression for the speedup of a symmetric CMP (the first equation in [9]), is a special case of our own model. First, in order to

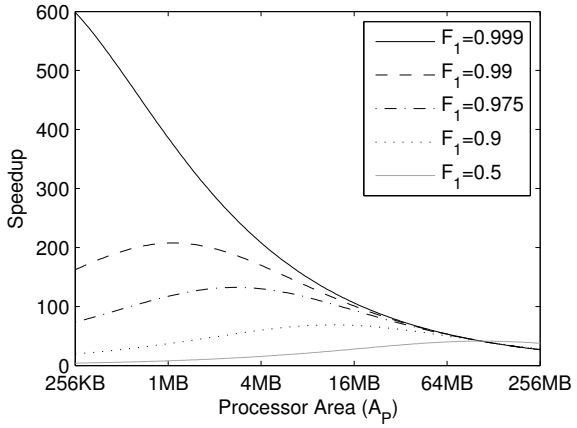

Fig. 21. Speedup for different values of F_1, assuming $A_{fix} = A_{L2} = 0$. Compare with Fig. 2(b) in [9]. Units of area in memory byte equivalent units.

match Hill and Marty's approach, we must use $A_{fix} = 0$ and $A_{L2} = 0$. The cost function (33) simplifies to:

$$J_D = F_0 \beta A_P^{-\frac{1}{2}} + \frac{F_1 A_P}{A_{tot}} \beta A_P^{-\frac{1}{2}} \quad (54)$$

Next, we formulate an expression for speedup. Since we are not accounting for energy, the units of J_D are cycles. Creating a ratio between the cost of the architecture J_D and the cost of a baseline architecture, J_D^{P0}, we obtain a measure of speedup. Let the baseline architecture be a single processor ($F_0 = 1$) of minimum size (A_{P0}), thus $J_D^{P0} = \beta A_{P0}^{-\frac{1}{2}}$.

$$speedup = \frac{J_D^{P0}}{J_D} = \frac{\beta A_{P0}^{-\frac{1}{2}}}{F_0 \beta A_P^{-\frac{1}{2}} + \frac{F_1 A_P}{A_{tot}} \beta A_P^{-\frac{1}{2}}} \quad (55)$$

In order to normalize the numerator to 1, we choose $\beta = A_{P0}^{\frac{1}{2}}$:

$$speedup = \frac{1}{F_0 A_{P0}^{\frac{1}{2}} A_P^{-\frac{1}{2}} + \frac{F_1 A_P}{A_{tot}} A_{P0}^{\frac{1}{2}} A_P^{-\frac{1}{2}}} \quad (56)$$

Following Hill and Marty's notation, $perf(r) = \sqrt{\frac{A_P}{A_{P0}}}$, $F_1 = f$, $F_0 = 1 - f$, $A_P = r$ and $A_{tot} = n$, resulting in:

$$speedup = \frac{1}{\frac{F_0}{perf(r)} + \frac{F_1 A_P}{perf(r) A_{tot}}} \quad (57)$$

$$= \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) n}} \quad (58)$$

This is Hill and Marty's expression for the speedup of a symmetric CMP [9]. Equation (56) is plotted in Fig. 21, matching Fig. 2(b) in [9]. Comparing Fig. 8 and 21, it is apparent that using a value of zero for A_{L2} and A_{fix} leads to overly optimistic estimations of speedup. (Note the change in y-axis scale between the figures.) Further, the optimal processor area (the area value at which each curve is at a maximum) is under estimated.

In another recent work, Woo and Lee [11] extend the model of Hill and Marty to include energy. Our model is similar, in that energy is broken out for serial and parallel processors, active and idle processors, and can be applied to both symmetric and asymmetric architectures. In contrast, we include a microarchitectural model of energy, such as the memory hierarchy and communication energy, and they do not. In the Online Supporting Material, we demonstrate that with a number of simplifying assumptions, our model is equivalent to Woo and Lee's model.

Oh et al. [63] developed a detailed model of cache memory and performed tradeoffs between the cache size and number of cores on the CMP. We perform a similar area constrained design tradeoff analysis in Section 4.2. Their cache model is more detailed, however, they lack a global system expression, inclusion of energy, and a closed form solution to the optimum. Moreover, there is nothing to preclude using more detailed cache models in our global objective function.

Two other approaches, [64] and [65], perform similar constrained design space exploration of CMP architectures, including thermal analysis. They emphasize the importance of joint optimization across interrelated variables and inclusion of constraints during optimization. However, both approaches are based on simulators, in contrast to our analytical approach to optimization.

6.1 Methodology

We have shown considerable math in this paper in order to demonstrate our methodology and in particular, closed form solutions to our objective function. However, if the closed form solution is intractable or one wishes to avoid the algebra, our objective function can be solved using Newton's method to find the optimal solution. This automation step is proceeded by using symbolic solvers to perform the partial derivatives (using Maple [66] or Sage [67] for example). In this case, building the CMP model is reduced to expressing the system in terms of the objective function – writing down a single equation. In addition to being able to rapidly create CMP models, design exploration and optimization executes in fractions of second. Fast model creation and fast execution time represent two significant advantages over simulation based approaches for exploring a large design space. (See Fig. 2 in [59] for representative execution times of simulation based approaches.)

In this paper we have made the assumption of general purpose computing applications, particularly for cache miss rate parameters κ, G_i and the processor performance versus area constant β. However, our approach is in no way limited only to general purpose computing. With appropriate parameters and underlying models (for example, the relationship between processor performance and area), our model easily extends to other application classes. In forthcoming work, we have extended our generalized objective function to the

optimization of asymmetric CMPs as well as to shared access models of bus and network communication and memory bandwidth.

Our goal for this paper is to establish a system-level analytical model for parallel computational architectures that incorporates computation, memory, and parallelism, combined with delay and energy costs in an area constrained setting. Our six examples have built from simple to progressively more complex in an effort to make our modeling approach understandable. As a result, we have focused on first-order, system-level effects. Thus there are many detailed effects that we have not yet included in our model, as presented here. Examples include cache sharing and interference, L3 caches, and cache coherency. More detailed models can be built using our approach by incorporating more complex lower-level models. For example, in Section 4.2, we modeled independent L2 caches, without sharing or interference between cores. More detailed low-models that include these effects could be derived from analytical models such as [68], [69], [70]. Similarly, we do not specifically model of cache coherency. It could be included by considering how it affects area, energy, and delay. It increases communication traffic on the CMP network, which increases congestion and potentially increases delay for any instructions that require data from the network. And, depending on the coherency protocol, it may increase the latency and energy to access data, depending on the distance to access the data (local, neighbor, distant, etc). Modeling this level of detail is a good subject for future work.

There are also effects that we have not considered here, but are straightforward extensions to that which we have presented. For example, we assumed that the L2 cache was the last level cache, while many CMPs have internal L3 caches. An L3 cache is easily modeled by extending the inner summation with an extra G_{ij} term for the fraction of instructions that hit in the L3 cache, (after L2 and before main memory). Of course, an L3 model including sharing or interference would require more work. Similarly, effects such as access to off-chip resources (I/O, disks, network) can be modeled by adding extra G_{ij} terms (and D_{ij}, E_{ijh}) and expanding the inner summation. Another effect to consider is that all of our examples have assumed the delays for the parallel and serial sections are the same ($D_i0 = D_i1, \forall i$), resulting a cost function of the form:

$$J_D = (F_0 + F_1/N)(G_0D_0 + G_1D_1 + G_2D_2).$$

However, the instruction mix is generally different for the parallel and serial fractions of the algorithm. For example, communication and access to the memory hierarchy is much larger in the parallel fraction than in the serial fraction. However, there is nothing in the model that precludes different delays for the parallel and serial algorithm fractions, using a cost function of the form:

$$\begin{aligned} J_D &= F_0(G_{00}D_{00} + G_{10}D_{10} + G_{20}D_{20}) + \\ &\quad F_1/N(G_{01}D_{01} + G_{11}D_{11} + G_{21}D_{21}) \end{aligned}$$

TABLE 3
Summary of symmetric CMP examples

Section	Optimization	Variables	Energy
4.1	Processor	N vs. A_P	-
4.2	Memory	N vs. A_{L2}	-
4.3	Full Optimization	N vs. A_P vs. A_{L2}	-
4.4	Processor	N vs. A_P	Yes
4.5	Memory	N vs. A_{L2}	Yes
4.6	Full Optimization	N vs. A_P vs. A_{L2}	Yes

TABLE 4
Summary of parameters for each example

Parameter	4.1	4.2	4.3	4.4	4.5	4.6
K	2	2	2	2	2	2
F_0	F_0	F_0	F_0	F_0	F_0	F_0
F_1	F_1	F_1	F_1	F_1	F_1	F_1
N_0	1	1	1	1	1	1
N_1	N	N	N	N	N	N
M	1	3	3	1	3	3
G_0	1	G_0	G_0	1	G_0	G_0
G_1	-	$\hat{G_1}$	$\hat{G_1}$	-	$\hat{G_1}$	$\hat{G_1}$
G_2	-	$\hat{G_2}$	$\hat{G_2}$	-	$\hat{G_2}$	$\hat{G_2}$
D_0	CPI	D_0	CPI	CPI	D_0	CPI
D_1	-	D_1	D_1	-	D_1	D_1
D_2	-	D_2	D_2	-	D_2	D_2
γ	0	0	0	γ	γ	γ
E_0	-	-	-	E_P	E_0	E_P
E_1	-	-	-	-	E_M	E_M
E_2	-	-	-	-	E_2	E_2

$$\begin{aligned} \hat{G_1} &= (1 - G_0)(1 - \kappa A_{L2}^{-\frac{1}{2}}) \\ \hat{G_2} &= (1 - G_0)(\kappa A_{L2}^{-\frac{1}{2}}) \\ E_P &= \rho_P A_P = E_{active}/A_{P_0}A_P \\ E_M &= E_{idle} + \rho_{L2}A_{L2}^{\frac{1}{2}} = E_{idle} + E_1/A_{L2_0}A_{L2}^{\frac{1}{2}} \end{aligned}$$

where F_0, G_{i0}, D_{i0} correspond to the serial fraction, and F_1, G_{i1}, D_{i1} correspond to the parallel fraction. With accurate estimates of the statistics for the G_{ij}'s and D_{ij}'s for both the serial and parallel fractions, the result is a more accurate model.

6.2 Summary of Examples

Our six optimization examples are summarized in Table 3 and the parameters for each of the examples are summarized in Table 4. The first three examples show the tradeoffs of the number of cores N versus the size of each core and its cache memory as a function of the parallelism in the application. As the parallel fraction of the application F_1 increases, the optimal number of cores N grows rapidly. The last three examples replicate the earlier examples while including energy.

7 Conclusion

With the historical advent of Very Large Scale Integrated (VLSI) systems and Systems on a Chip (SoC), simple analytical models that characterize the architectural

components in VLSI systems much like Amdahl's law, have proven to be crucial to the success VLSI design methodologies. This trend was exemplified in the models for memory area, circuit delay and system parallelism found in the seminal textbook on VLSI design [14]. Over the past twenty years analytical models were developed for many different levels of design abstraction, including the transistor, wire and element level, the logic gate and standard cell level, up to the functional unit level. Eventually, many analytical models became part of automatic algorithms for generating complete circuits and designs. For example, there are currently placement and routing CAD tools for physical layout, as well as tools for logic synthesis and standard cell mapping.

In the new era of many-core CMPs, full system optimization is the dominant design theme. Memory architecture and communication must be optimized along with processor microarchitecture. In modern processors, the cost of off-chip communication is high in terms of delay and energy. And although there are some methods for latency hiding (e.g. prefetching, non-blocking writes, etc.), we observe that there are no available techniques for "energy hiding."

In this paper we have presented a cost function formulation of parallel processing performance based on low level energy and delay costs. We derived our objective function beginning with Amdahl's law. Using a constrained optimization framework and cost function minimization, we have demonstrated an approach for high-level architectural optimization of CMPs. Our approach finds the CMP architecture that maximizes the parallel energy-delay performance, subject to the fixed total area constraint. This approach is useful for illuminating architectural trends in CMP design. However, the primary goal is to explore the high-level design space prior to refining the design space (with instruction set simulator models). Ultimately, our objective function is the cost function to be optimized in a high-level automated design tool for designing symmetric and asymmetric multi- and many-core CMPs. This CMP architectural optimization is required in order to realize next-generation Exascale systems [19].

Our generalized cost function is capable of capturing and optimizing a rich set of complex behaviors, inherent in the design tradeoffs of CMP system design. By no means are the examples presented here exhaustive. Our goal is to present a framework and approach for optimization. We expect that further research will produce refined expressions for the performance, energy, and area tradeoffs used for optimization.

Acknowledgments

This research was motivated by the discussions at the Second Kavli Futures Symposium titled held in Costa-Rica in January of 2009, under the theme of "Real Problems for Imagined Computers." More specifically, the generalization of Amdahl's Law as a cost function formulation that includes the energy-delay product was inspired by the "Bandwidth-Cost Integral," an idea that was conceived by the members of the working group "Obstacles to Exascale," Bill Dally, Tim Cornwell, Ravi Nair, Michael Roukes, Horst Simon, and A.G. Andreou. This work was also partially supported by the European FP7 project SCANDLE and an ONR MURI N000141010278.

References

[1] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proceedings AFIPS Spring Joint Computer Conference*, 1967.

[2] J. L Hennessy and D. A Patterson, "Computer architecture: a quantitative approach, (3rd edition)," *Morgan Kaufmann Publishers*, 2002.

[3] R. G. Brown, "Maximizing Beowulf Performance," *Proceedings of 4th Annual Linux Showcase and Conference*, 2000.

[4] J. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, 1988.

[5] S. Krishnaprasad, "Uses and abuses of Amdahl's Law," *Journal of Computing Sciences in Colleges*, 2001.

[6] G. Bell, J. Gray, and A. Szalay, "Petascale computational systems," *Computer*, 2006.

[7] A. Szalay, G. Bell, J. Vandenberg, A. Wonders, R. Burns, D. Fay, J. Heasley, T. Hey, M. Nieto-SantiSteban, A. Thakar, C. van Ingen, and R. Wilton, "GrayWulf: Scalable Clustered Architecture for Data Intensive Computing," in *42nd Hawaii International Conference on System Sciences, (HICSS 2009)*, 2009.

[8] S. Borkar, "Thousand core chips: a technology perspective," *Proceedings of the 44th annual Design Automation Conference (DAC '07)*, 2007.

[9] M. Hill and M. Marty, "Amdahl's Law in the multicore era," *Computer*, 2008.

[10] J. M. Paul and B. H. Meyer, "Amdahl's Law revisited for single chip systems," *International Journal of Parallel Programming*, 2007.

[11] D. H. Woo and H. Lee, "Extending Amdahl's Law for energy-efficient computing in the many-core era," *Computer*, 2008.

[12] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," *Proceedings of the 7th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, 1996.

[13] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: a scalable architecture based on single-chip multiprocessing," *Proceedings of the 27th annual International Symposium on Computer Architecture (ISCA '00)*, 2000.

[14] C. Mead and L. Conway, "Introduction to VLSI systems," *Addison-Wesley Publishers*, 1979.

[15] A. S. Cassidy and A. G. Andreou, "Analytical methods for the design and optimization of chip-multiprocessor architectures," *43rd Annual Conference on Information Sciences and Systems (CISS 2009)*, 2009.

[16] A. S. Cassidy, K. Yu, H. Zhou, and A. G. Andreou, "A high-level analytical model for application specific CMP design exploration," *Proceedings of the 2011 Conference on Design Automation & Test in Europe (DATE 2011)*, 2011.

[17] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. A. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, "The HTK book," *University of Cambridge*, 2009.

[18] J. A. Rice, "Mathematical Statistics and Data Analysis (3rd Edition)," *Duxbury Press*, 2006.

[19] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "ExaScale computing study: technology challenges in achieving exascale systems," *DARPA IPTO Report*, 2008.

[20] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid State Circuits*, 1996.

[21] S. Przybylski, M. Horowitz, and J. Hennessy, "Characteristics of performance-optimal multi-level cache hierarchies," *Proceedings of the 16th annual International Symposium on Computer Architecture (ISCA '89)*, 1989.

[22] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma, "On the nature of cache miss behavior: is it square root of 2?" *Journal of Instruction-Level Parallelism*, 2008.

[23] L. Codrescu, M. Deb-Pant, T. Taha, J. Eble, S. Wills, and J. Meindl, "Exploring microprocessor architectures for gigascale integration," *20th Anniversary Conference on Advanced Research in VLSI*, 1999.

[24] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *Proceedings of the 27th annual International Symposium on Computer Architecture (ISCA '00)*, 2000.

[25] D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, P. Emma, and M. Rosenfield, "New methology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *Ibm Journal Of Research And Development*, 2003.

[26] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," *ACM SIGARC Computer Architecture News*, 2000.

[27] P. Shivakumar and N. Jouppi, "CACTI 3.0: an integrated cache timing, power, and area model," *Compaq Computer Corporation (WRL Research Report 2001/2)*, 2001.

[28] S. Thoziyoor, N. Muralimanohar, and J. Ahn, "CACTI 5.1," *HP Laboratories Technical Report (HPL-2008-20)*, 2008.

[29] C.-L. Su and A. Despain, "Cache design trade-offs for power and performance optimization: a case study," *Proceedings of the 1995 international symposium on Low power design (ISLPED '95)*, 1995.

[30] M. Kamble and K. Ghose, "Analytical energy dissipation models for low power caches," *Proceedings 1997 International Symposium on Low Power Electronics and Design*, 1997.

[31] ——, "Energy-efficiency of VLSI caches: a comparative study," *Proceedings Tenth International Conference on VLSI Design*, 1997.

[32] F. Pollack, "New microarchitecture challenges in the coming generations of CMOS process technologies (keynote address)(abstract only)," *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture (MICRO 32)*, 1999.

[33] J. Schutz and C. Webb, "A scalable X86 CPU design for 90 nm process," in *2004 IEEE International Solid-State Circuits Conference, Digest of Technical Papers (ISSCC 2004)*, 2004.

[34] H. McIntyre, D. Wendell, K. Lin, P. Kaushik, S. Seshadri, and Wang, "A 4-MB on-chip L2 cache for a 90-nm 1.6-GHz 64-bit microprocessor," *IEEE Journal of Solid State Circuits*, 2005.

[35] C. McNairy and R. Bhatia, "Montecito: a dual-core, dual-thread Itanium processor," *IEEE MICRO*, 2005.

[36] AMD, "AMD website," *www.amd.com/*, 2010.

[37] Intel, "Intel website," *www.intel.com*, 2010.

[38] J. Tendler, J. Dodson, J. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *Ibm Journal Of Research And Development*, 2002.

[39] R. Kalla, B. Sinharoy, and J. Tendler, "IBM Power5 chip: a dual-core multithreaded processor," *IEEE MICRO*, 2004.

[40] H. Le, W. Starke, J. Fields, F. O'Connell, D. Nguyen, B. Ronchetti, W. Sauer, E. Schwarz, and M. Vaden, "IBM POWER6 microarchitecture," *Ibm Journal Of Research And Development*, 2007.

[41] P. Conway and B. Hughes, "The AMD Opteron northbridge architecture," *IEEE MICRO*, 2007.

[42] B. Stackhouse, S. Bhimji, C. Bostak, D. Bradley, B. Cherkauer, J. Desai, E. Francom, M. Gowan, P. Gronowski, D. Krueger, C. Morganti, and S. Troyer, "A 65nm 2-billion transistor quad-core Itanium processor," *IEEE Journal of Solid State Circuits*, 2009.

[43] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: a 32-way multithreaded SPARC processor," *IEEE MICRO*, 2005.

[44] M. Shah, J. Barren, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Sana, D. Sheahan, L. Spracklen, and A. Wynn, "UltraSPARC T2: a highly-treaded, power-efficient, SPARC SOC," *IEEE Asian Solid-State Circuits Conference, (ASSCC '07)*, 2007.

[45] U. Nawathe, M. Hassan, K. Yen, A. Kumar, A. Ramachandran, and D. Greenhill, "Implementation of an 8-core, 64-thread, power-efficient SPARC server on a chip," *IEEE Journal of Solid State Circuits*, 2008.

[46] G. Konstadinidis, M. Tremblay, S. Chaudhry, M. Rashid, P. Lai, Y. Otaguro, Y. Orginos, S. Parampalli, M. Steigerwald, S. Gundala, R. Pyapali, L. Rarick, I. Elkin, Y. Ge, and I. Parulkar, "Architecture and Physical Implementation of a Third Generation 65nm, 16 Core, 32 Thread Chip-Multithreading SPARC Processor," *IEEE Journal of Solid State Circuits*, 2009.

[47] Azul, "Azul website," *www.azulsystems.com*, 2010.

[48] J. Andrews and N. Baker, "Xbox 360 System Architecture," *IEEE MICRO*, 2006.

[49] D. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa, "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *IEEE Journal of Solid State Circuits*, 2006.

[50] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: built for speed," *IEEE MICRO*, 2006.

[51] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Transactions on Graphics*, 2008.

[52] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," in *2007 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC 2007)*, 2007.

[53] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," in *IEEE International Solid-State Circuits Conference, Digest of Technical Papers(ISSCC 2008)*, 2008.

[54] Clearspeed, "ClearSpeed website," *www.clearspeed.com*, 2010.

[55] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU Computing," in *Proceedings of the IEEE*, 2008.

[56] Nvidia, "NVIDIA website," *www.nvidia.com*, 2010.

[57] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. Dally, "A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing," *IEEE Journal of Solid State Circuits*, 2008.

[58] D. Burger and T. Austin, "The SimpleScalar tool set, version 2.0," *SIGARCH Computer Architecture News*, 1997.

[59] L. Zhao, R. Iyer, J. Moses, R. lllikkal, S. Makineni, and D. Newell, "Exploring large-scale CMP architectures using ManySim," *IEEE MICRO*, 2007.

[60] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, 2002.

[61] C. Hughes, V. Pai, P. Ranganathan, and S. Adve, "Rsim: simulating shared-memory multiprocessors with ILP processors," *Computer*, 2002.

[62] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOS machine simulator to study complex computer systems," *Transactions on Modeling and Computer Simulation (TOMACS)*, 1997.

[63] T. Oh, H. Lee, K. Lee, and S. Cho, "An Analytical Model to Study Optimal Area Breakdown between Cores and Caches in a Chip Multiprocessor," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI 09)*, 2009.

[64] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP design space exploration subject to physical constraints," *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, 2006.

[65] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: a power/performance/thermal view," *Proceedings of the 20th annual International Conference on Supercomputing (ICS '06)*, 2006.

[66] Maplesoft, "Maple," *www.maplesoft.com/products/Maple/index.aspx*, 2010.

[67] Sagemath, "Sage," *www.sagemath.org/*, 2010.

[68] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *Proceedings 11th International Symposium on High-Performance Computer Architecture (HPCA)*, 2005.

[69] X. Chen and T. Aamodt, "A first-order fine-grained multithreaded throughput model," *15th International Symposium on High Performance Computer Architecture (HPCA 2009)*, 2009.

[70] D. J. Sorin, V. S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood, "Analytic evaluation of shared-memory systems with ILP processors," in *ACM SIGARC Computer Architecture News*, 1998.

IEEE Transactions on Computers, vol. 61, no. 8, pp. 1110–1126, Aug. 2012.

[24] D. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa, "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *IEEE Journal of Solid State Circuits*, 2006.

[25] M. Shah, J. Barren, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Sana, D. Sheahan, L. Spracklen, and A. Wynn, "UltraSPARC T2: a highly-treaded, power-efficient, SPARC SOC," *IEEE Asian Solid-State Circuits Conference, (ASSCC '07)*, 2007.

[26] G. Konstadinidis, M. Tremblay, S. Chaudhry, M. Rashid, P. Lai, Y. Otaguro, Y. Orginos, S. Parampalli, M. Steigerwald, S. Gundala, R. Pyapali, L. Rarick, I. Elkin, Y. Ge, and I. Parulkar, "Architecture and Physical Implementation of a Third Generation 65nm, 16 Core, 32 Thread Chip-Multithreading SPARC Processor," *IEEE Journal of Solid State Circuits*, 2009.

[27] Maplesoft, "Maple," *www.maplesoft.com/products/Maple/index.aspx*, 2010.

[28] Sagemath, "Sage," *www.sagemath.org/*, 2010.

[29] D. H. Woo and H. Lee, "Extending Amdahl's Law for energy-efficient computing in the many-core era," *Computer*, 2008.

[30] M. Hill and M. Marty, "Amdahl's Law in the multicore era," *Computer*, 2008.