Evaluating the Combined Effect of Memory Capacity and Concurrency for Many-Core Chip Design

YU-HANG LIU, ACM Member, IEEE Member
XIAN-HE SUN, ACM Senior Member, IEEE Fellow Computer Science Department,
Illinois Institute of Technology, Institute of Computing Technology, Chinese Academy of Sciences

9

Modern memory systems are structured under hierarchy and concurrency. The combined impact of hierarchy and concurrency, however, is application dependent and difficult to describe. In this article, we introduce C^2-Bound, a data-driven analytical model that serves the purpose of optimizing many-core design. C^2-Bound considers both memory capacity and data access concurrency. It utilizes the combined power of the newly proposed latency model, concurrent average memory access time, and the well-known memory-bounded speedup model (Sun-Ni's law) to facilitate computing tasks. Compared to traditional chip designs that lack the notion of memory capacity and concurrency, the C^2-Bound model finds that memory bound factors significantly impact the optimal number of cores as well as their optimal silicon area allocations, especially for data-intensive applications with a non-parallelizable sequential portion. Therefore, our model is valuable to the design of next-generation many-core architectures that target big data processing, where working sets are usually larger than the conventional scientific computing. These findings are evidenced by our detailed simulations, which show, with C^2-Bound, the design space of chip design can be narrowed down significantly up to four orders of magnitude. C^2-Bound analytic results can be either used in reconfigurable hardware environments or, by software designers, applied to scheduling, partitioning, and allocating resources among diverse applications.

Categories and Subject Descriptors: B.3.2 [**Design Styles**]: Cache Memories; D.3.3 [**Computer Systems Organization**]: Performance of Systems

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Memory wall, data stall time, memory bound, data access concurrency, Sun-Ni's law, chip design, concurrent average memory access time (C-AMAT)

ACM Reference Format:
Yu-Hang Liu and Xian-He Sun. 2017. Evaluating the combined effect of memory capacity and concurrency for many-core chip design. ACM Trans. Model. Perform. Eval. Comput. Syst. 2, 2, Article 9 (March 2017), 25 pages.
DOI: http://dx.doi.org/10.1145/3038915

This work is supported in part by the National Science Foundation under grants CCF-1536079, CNS-1162540, and CCF-0937877; the National High Technology Research and Development Program ("863" Program) of China under grant 2015AA015303; and the National Natural Science Foundation of China under grants 61521092 and 61272132.
Authors'addresses: Y.-H. Liu is current with the Institute of Computing Technology, Chinese Academy of Sciences, No. 6 Kexueyuan South Road Zhongguancun, Haidian District Beijing, China, 100190; email: liuyuhang@ict.ac.cn; X.-H. Sun is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL, US, 60616; email: sun@iit.edu.

DOI: http://dx.doi.org/10.1145/3038915

1. INTRODUCTION

As data are the objects and results of computation, the role of memory capacity and concurrency is increasingly important for data-intensive applications in the "big data" era. The interaction between capacity and concurrency is subtle, and their combined effect has not been fully explored [Liu and Sun 2015a]. In this study, we provide a better understanding of this interaction and, more importantly, apply the understanding successfully to chip design space exploration.

The design of many-core processor is an important and challenging task. An on-chip multiprocessor (CMP) is an integrated circuit that consists of two or more independent processing units (called "cores") to read and execute program instructions. The design space exploration of on-chip multiprocessors is to investigate potential configurations of the integrated circuit with diverse goals regarding to performance, power consumption, and simultaneous processing of multiple tasks. In recent years, on-chip multiprocessors have become the mainstream of microprocessors that underpin the pivotal computing infrastructure [Borkar 2007]. In the meantime, the number of cores is continuously increasing on processors. This incease imposes additional challenge to the already extremely huge design space of CMP that is composed of intractable combinations of a large number of architecture parameters for the optimization [Ipek et al. 2008].

The essence of design is to find the optimal values of architectural parameters that form a design space. Microprocessor architectural design space exploration (DSE) is often investigated using a simulation or emulation of various target architectures. Therefore, architecture researchers heavily rely on software simulator and Field Programmable Gate Array (FPGA) emulator for performance evaluation of processor architectures. However, as shown in Table I, the simulation speed is extremely slow, a typical slowdown of the real execution time in the order of 10^5 to 10^6. As a result, simulating a relatively small program that takes 1min to execute requires approximately one month to a year to simulate [Eeckhout 2010]. The huge design space and the low simulation speed prevent computer architects from thoroughly exploring the intractable design space. The traditional brute-force algorithm of first-simulate-then-compare becomes challenging if not infeasible.

Effectively dealing with the notoriously difficult DSE problem has become the key of multi-core design in practice. Prediction and Ranker methods were two of the most advanced works for quickly finding the optimal architacture in DSE [Ipek et al. 2008; Chen et al. 2014]. The basis of the methods was machine-learning technologies, including artificial neural network (ANN) and boost prediction. While still in need of substantial time for training and predicting, the machine-learning technologies encapsulated the detailed interactions between various architecture components and these interaction information can assist architects in understanding the system design. In this article, we want to explicitly show the relationship among the basic architecture and application parameters for better understanding and optimizing, especially in the initial phase of the DSE process.

Analytical modeling is extremely fast compared to other methods and explicitly depicts the interactions among the architecture components. In practice, an analytical model should capture the most important features of the architecture with a limited number of measurable parameters. If a model has too many parameters or has immeasurable parameters, then it will not be useful. We will keep this point in mind in our discussion.

A variety of analytical models have been presented recently [Cassidy and Andreou 2009; Cassidy et al. 2011; Cassidy and Andreou 2012; Hill and Marty 2008; Woo and Lee 2008; Sun and Chen 2010] to address these challenges. For example, Cassidy and Andreou incorporate sequential data access delay in terms of average memory access time (AMAT) into Amdahl's law [Cassidy and Andreou 2009; Cassidy et al.

Table I. The Speed of Different Simulators

Simulator	ISA	Micro-architecture	Speed
Intel	X86	Core 2	1–10KHz
AMD	X86	Opteron	1–10KHz
IBM	Power	Power5	200KIPS
PTLSim	X86	AMD Athlon	270KIPS
Sim-outorder	Alpha	Alpha 21264	740KIPS
GEMS	Sparc	Generic	69KIPS
GEM5	X86, Alpha, MIPS, etc.	Generic	100–200KIPS

2011; Cassidy and Andreou 2012] while Hill and Marty apply Amdahl's concepts to multicore architectures based on a hardware cost model [Hill and Marty 2008]. Woo and Lee follow up with the consideration of energy efficiency [Woo and Lee 2008]. In contrast, Sun and Chen consider the impact of memory capacity for many-core design, but they do not explicitly incorporate data access concurrency in their analysis [Sun and Chen 2010]. Although these works, more or less, optimize the design space of CMP in different forms, few of them explore the memory concurrency. As a consequence, data access patterns in these studies are exploited from a sequential perspective using the AMAT metric, which cannot truly reflect the current situation. In addition, most of these studies are also short on the consideration of memory capacity impact on problem sizes as well, which is another important factor in the design space of CMP.

As data access delay is dominating the overhead in modern big-data processing, it has become the most preeminent performance bottleneck of computing systems. For example, the processor stall time due to data access typically contributes 50% to 70% of the total application execution time [Hardavellas et al. 2007; Somogyi et al. 2009]. Therefore, incorporating data access patterns into performance models becomes vitally important. In fact, memory concurrency as the main form of data access patterns has become a more prevalent factor to the design of efficient memory systems (e.g., multi-port, multi-banked, pipelined cache, non-blocking cache, runahead, and simultaneous multithread). Meanwhile, in addition to data concurrency, in scalable computing problem size increases with computing resources, and the increase is usually bounded by the available memory size [Sun and Ni 1990; Woo et al. 1995; Bienia et al. 2008]. Hence, the assumption that the problem size is fixed would cause misleading results. As a result, it is crucial to include memory concurrency and memory capacity considerations in the many-core DSE in order to keep up with the increasing importance of modern memory systems and the emerging of data intensive applications. To the best of our knowledge, we propose for the first time to simultaneously consider data access concurrency and memory capacity to explore the design space of CMP.

Memory wall and memory bound are two well-known performance constraints [Sun and Ni 1990; Wulf and McKee 1995]. C-AMAT is a new model that unifies the combined impact of data locality and concurrency on data access. Thus, applying C-AMAT to many-core design is a natural choice [Sun and Wang 2014; Sun 2014]. In the meantime, Sun-Ni's law is a generalization of Amdahl's law and Gustafson's law [Sun and Ni 1990; Amdahl 1967; Gustafson 1988]. While these scalable laws are well studied, they are traditionally discussed in the context of supercomputing. The Sun-Nis law (memory-bounded speedup) is originally designed for scalability study, where each node (chip) has a fixed memory, so increasing the number of node will increase the memory size proportionally and give a speed up bound under the physic memory bound. The details of the memory hierarchy and sequential (single node) performance variation, however, are not considered in Sun-Ni's law. Therefore, applying Sun-Ni's law for many-core processor design is challenging and has practical significance. In this article, we present

C^2-Bound, a data-driven analytical model that incorporates both memory concurrency and memory capacity factors as well as both execution time and scalability measurement for many-core design. The essence of this model is to take advantage of both the C-AMAT and Sun-Ni's law to optimize the design space of CMP. In particular, this study makes the following contributions:

—The C^2-Bound model is proposed to consider both memory locality and concurrency at the same time. To this end, we derive program-specific model parameters from traces and consider adaptively reshaping (through allocating, scheduling) the underlying architecture.
—The C^2-Bound model considers the problem sizes that are bounded by variable memory capacities. When the number of cores, N, is changed, the on-chip cache capacity will be adjusted correspondingly, and then the problem size will be scaled to a different value. We termed it as the problem size scale function $g(N)$ and found $g(N)$ is a vital factor in balancing the number of cores and the size of the caches. When $g(N) < O(N)$, few cores but large caches are needed; when $g(N) \geq O(N)$, more cores and smaller caches are preferred. These results can significantly narrow the large design space and they can only be got from analytical model rather than from an architect's intuition.
—The C^2-Bound model presents an interface between analysis and simulation. With the Analysis plus Simulation (APS) algorithm, the newly proposed DSE model has been integrated into the GEM5 simulator to supervise simulation [Binkert et al. 2011]. The APS approach drastically reduces the number of required simulations. Representative results for diverse applications confirm the feasibility and correctness of the newly proposed analytical model for many-core processor design. The C^2-Bound model has been implemented as an automatic tool to find an application-specific optimal architecture.
—Based on the C^2-Bound model, a few important but subtle concepts have become clear mathematically. The memory bound effect of "big data" has been revealed analytically. Meanwhile, the difference between "high performance computing" (HPC) and "high throughput computing" (HTC) has been explored and formalized clearly. As these concepts are fundamental and widely used in the computing community, the formula and quantification in this work will be of significant importance.

The remainder of this article is organized as follows. The next section provides some preliminary knowledge of C-AMAT and Sun-Ni's law. Section 3 then proposes the data-driven C^2-Bound analytical model for many-core design. Section 4 presents application specific design exploration case studies. Section 5 further discusses memory concurrency and memory capacity-bounded problem size. Section 6 reviews related work in many-core design exploration. Finally, Section 7 concludes this study and discusses potential future work.

2. MEMORY BOUNDS IN TERMS OF LATENCY AND CAPACITY

In this article, unless otherwise stated, a memory system is the whole memory hierarchy rather than only the main memory. Latency and capacity are two bounds of memory on the achievable computing performance. C-AMAT, a new performance metric, accounts for concurrency at both the component and system levels for modern memory design [Sun and Wang 2014; Sun 2014; Liu and Sun 2015b]. C-AMAT represents measures and analyzes data access delay from a single program perspective. In contrast, Sun-Ni's law highlights the impact of memory bounded problem sizes on parallel speedup [Sun and Ni 1990]. Both Amdahl's law [Amdahl 1967] and Gustafson's law [Gustafson 1988] are the special cases of Sun-Ni's law.

2.1. C-AMAT

The conventional memory metric AMAT formulation is shown in Equation (1) [Hennessy and Patterson 2012], where H is the hit time of data accesses, MR is the miss rate, and AMP is the average miss penalty. AMP is the sum of all miss access latencies divided by the total number of misses. AMAT does not consider the concurrency of data accesses in terms of either hits or misses, based on the assumption that data accesses are sequential, one after another; further, AMAT does not take into account that with concurrent accesses, hits and misses may coexist within the same cycle. The sequential assumption governing AMAT worked well in the past but applies less accurately for modern processor architectures and memory systems where concurrency is paramount. For example, in an out-of-order processor, when a miss occurs, other instructions can be executed while the memory system is servicing the miss. Moreover, concurrency features such as multi-port, multi-bank, and multi-rank allow multiple outstanding reads and writes to co-exist at a given time in the memory system, depending on the underlying hardware support. Therefore, some of the data access latencies can be hidden,

$$AMAT = H + MR \times AMP. \tag{1}$$

To cover the concurrent read and write properties of modern memory systems, the C-AMAT model is proposed in Equation (2) [Sun and Wang 2014]. The first parameter H is the same as that in AMAT. The second parameter C_H represents hit concurrency; the third parameter C_M represents the pure miss concurrency. C_H can be contributed by caches with multi-port, multi-bank or pipelined structures. C_M can be contributed by non-blocking cache structures. In addition, out-of-order execution, multi-issue pipeline, multi-threading, and CMP can all increase C_H and C_M. The pure miss rate, pMR, differs from the conventional MR. pMR is the ratio of the number of pure (rather than conventional) misses over the total number of accesses. A pure miss here means that a miss contains at least one miss cycle that does not have any hit access activity [Sun and Wang 2014]. pAMP is the average number of pure miss cycles per miss access,

$$C\text{-}AMAT = \frac{H}{C_H} + pMR \times \frac{pAMP}{C_M}. \tag{2}$$

As shown in Equation (3), the ratio of AMAT and C-AMAT is the data access concurrency, which will be abbreviated to C,

$$C = \frac{AMAT}{C\text{-}AMAT}. \tag{3}$$

Generally, C is greater than or equal to 1. When $C = 1$, we can say there exists no concurrency. At this time, $C_H = 1$, $C_M = 1$, pMR = MR, and pAMP = AMP. Therefore, AMAT can be seen a special case of C-AMAT. In our later discussion, we will use Equation (3) to denote the data access concurrency.

Figure 1 demonstrates C-AMAT concept. There are five different memory accesses, and each access contains three cycles for cache hit operations. If it is a miss, then additional miss penalty cycles will be required. The number of miss penalty cycles is uncertain, depending on where the missed data can be obtained from and its contention impact during the data access. Accesses 1, 2, and 5 are hit accesses; Accesses 3 and 4 are miss accesses. Access 3 has a 3-cycle miss penalty; Access 4 has only a 1-cycle miss penalty. When considering the access concurrency, only Access 3 contains 2 pure miss cycles. Although Access 4 has 1 miss cycle, this cycle is not a pure miss cycle because it overlaps with the hit cycles of Access 5. Therefore, according to our new definition of (concurrent) pure miss rate, the pure miss rate of the five accesses is 0.2, instead of 0.4, that of the conventional non-concurrent version. When miss cycles are overlapping with hit accesses, the processor will not stall; the processor can continue

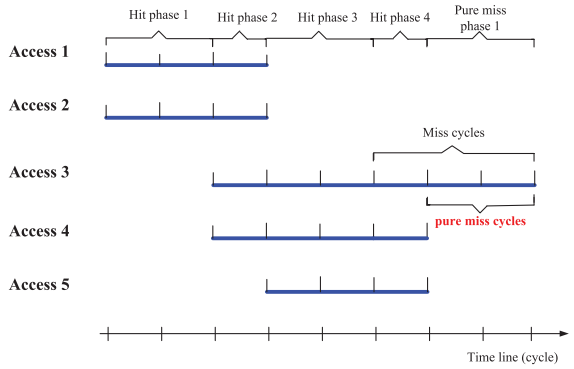

Fig. 1. A demo of C-AMAT and pure miss.

processing the data provided by the hit accesses. According to Equation (2), C-AMAT is 8 cycles of 5 accesses or 1.6 cycles per access, whereas by Equation (1) AMAT is $3 + 0.4 \times 2$ or 3.8 cycles per access. The difference between C-AMAT and AMAT is the contribution of concurrent data access. In this example, concurrency has increased memory performance to 2.375-fold.

In Figure 1, there are 4 hit phases, namely Hit phases 1, 2, 3, and 4, which contain 2, 4, 3, and 1 concurrent hit cache assesses with lasting cycle 2, 1, 2, 1, respectively. Therefore, $C_H = 2 \times 2/6 + 4 \times 1/6 + 3 \times 2/6 + 1 \times 1/6 = 5/2$. And there is only 1 pure miss phase with 1 pure miss concurrency which lasts for 2 cycles. Therefore $C_M = 1 \times 2/2 = 1$; pAMP = 2/1 =2; pMR = 1/5. Thus, formula (2) is equal to

$$C\text{-}AMAT = \frac{H}{C_H} + pMR \times \frac{pAMP}{C_M} = \frac{3}{5/2} + \frac{1}{5} \times \frac{2}{1} = 1.6.$$

The value of the parameters is in performance analysis and optimization. The invaluable contribution of C-AMAT is that it provides an unified formulation to capture the joint performance impact of locality and concurrency.

2.2. Sun-Ni's Law

Realizing that the problem size may be constrained by memory capacity, Sun-Ni's law was proposed in 1990 [Sun and Ni 1990]. Assume each computing node is a processor-memory pair. Increasing the number of processors, N, then, will increase the memory capacity as well. Assume $h(x)$ is the relationship between problem size and memory capacity size; that is,

$$W = h(M) \text{ and } W' = h(N \times M), \tag{4}$$

where M is the memory capacity of one node, W is the original problem size, and W' is the scaled problem size.

Let $g(N)$ be the function that reflects the parallel problem increase factor as the memory capacity increases N times (in the next subsection, we will present detailed examples to illustrate $g(N)$). By definition,

$$g(N) = W'/W. \tag{5}$$

Then we have

$$g(N) = h(N \times M)/h(M) = h(N \times h^{-1}(W))/h(M). \tag{6}$$

Thus memory capacity-bounded speedup is

$$Speedup_{Sun\text{-}Ni} = \frac{f_{seq} \times W + (1 - f_{seq}) \times h(N \times h^{-1}(W))}{f_{seq} \times W + \frac{(1-f_{seq}) \times h(N \times h^{-1}(W))}{N}}. \quad (7)$$

f_{seq} is the sequential portion of the problem size. Note that for any power function $h(x) = ax^b$ and for any rational numbers a and b [Sun and Ni 1990], we have

$$h(N \times x) = a(N \times x)^b = N^b \times ax^b = N^b \times h(x) = g(N) \times h(x). \quad (8)$$

Therefore,

$$Speedup_{Sun\text{-}Ni} = \frac{f_{seq} \times W + (1 - f_{seq}) \times g(N) \times W}{f_{seq} \times W + \frac{(1-f_{seq}) \times g(N) \times W}{N}}. \quad (9)$$

That is, Equation (10) holds,

$$Speedup_{Sun\text{-}Ni} = \frac{f_{seq} + (1 - f_{seq}) \times g(N)}{f_{seq} + \frac{(1-f_{seq}) \times g(N)}{N}}, \quad (10)$$

where $g(1) = 1$. Taking $g(N) = N^{3/2}$ as an example, the speedup is

$$Speedup_{Sun\text{-}Ni} = \frac{f_{seq} + (1 - f_{seq}) \times N^{3/2}}{f_{seq} + (1 - f_{seq}) \times N^{1/2}} = O(N). \quad (11)$$

When $g(N) = 1$, Equation (10) is Amdahl's law. When $g(N) = N$, Equation (10) is Gustafson's law. Because Amdahl's law [Amdahl 1967] as well as Gustafson's law [Gustafson 1988] both can be seen as the special cases of Sun-Ni's law [Sun and Ni 1990], we will use Sun-Ni's law in Equation (10) as the basis for our further discussion.

To make this more understandable, we take some applications as examples to illustrate the role of $g(N)$. The dense matrix multiplication is a well-known routine frequently found in applications. For dense matrices with dimension n, the computation requirement of matrix multiplication is $2n^3$ and the memory requirement is $3n^2$. Thus,

$$W = 2n^3 \text{ and } M = 3n^2. \quad (12)$$

Writing W as a function of M, we have Equation (13),

$$W = h(M) = \left(\frac{2M}{3}\right)^{3/2}. \quad (13)$$

Therefore,

$$W' = h(N \times M) = \left(\frac{2NM}{3}\right)^{3/2} = N^{3/2} \times h(M). \quad (14)$$

That is,

$$g(N) = h(N \times M)/h(M) = N^{3/2}. \quad (15)$$

In a similar manner, given the computation complexity and memory complexity, we can get the $g(N)$ value for any application. $g(N)$ represents the data reuse rate when memory is scaled N times. Table II shows the values of some applications. As will be shown later, $g(N)$ significantly impacts the optimal CMP configuration for different phases of an application and diverse applications. The merit of this work is that it presents an Analytical plus Simulation method to automatically obtain the quantitative solution before detailed simulation.

Table II. The $g(N)$ Factors of Some Applications

Application	Computation	Memory	g(N)
Dense matrix multiplication	N^3	N^2	$N^{3/2}$
Band sparse matrix multiplication	N	N	N
Stencil	N	N	N
Fast Fourier Transform	N	$Nlog_2N$	$2N$

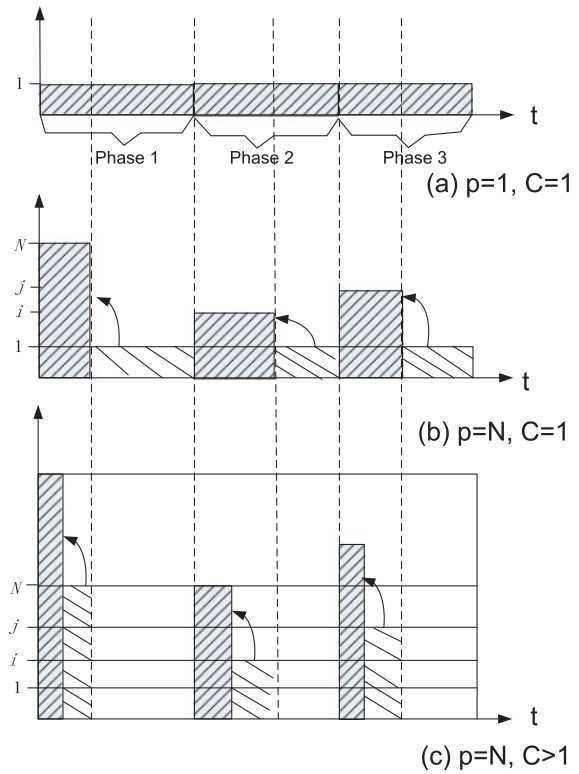

Fig. 2. The impact of process level concurrency and memory level concurrency on program running time.

3. THE C^2-BOUND CMP DSE MODEL

We formalize the many-core design space exploration as an optimization problem. Note that object function and constraints are needed for an optimization problem. We will follow a framework that is similar to that of Cassidy and Andreou to model the optimization problem [Cassidy and Andreou 2009; Cassidy et al. 2011; Cassidy and Andreou 2012]. In this section, we first present the execution time object function and constraints for optimization. Then, we propose the methodology for automatic collection of the needed parameter values and then resolve the optimization problem. Last, we will discuss how the analytical results can facilitate simulation.

3.1. Impact of C-AMAT on Data Stall Time

Given a fixed problem size, the impact of memory-level concurrency and process-level concurrency can be illustrated in Figure 2, where the x-axis is time, and the y-axis is the amount of work being done in parallel. Subgraph (a) shows the case when there is only one process ($p = 1$) and no memory concurrency ($C = 1$). Subgraph (b) shows the case when multiple processes are available ($p = N$) but still without memory concurrency

($C = 1$). Subgraph (c) shows the highest concurrent case when multiple processes are available ($p = N$) with memory concurrency ($C > 1$). The shadowed area is the total amount of operations done. The sum of the length of all the shadowed rectangles is the time it would take to run.

Quantifying the combined effect of the memory level concurrency and process level concurrency, as demonstrated in Figure 2, is difficult due to the entangled interaction between data access patterns and underlying computing system. What makes the problem become even more challenging is that the problem size is usually a function of the available memory capacity.

In this study, the bound impact of memory concurrency and memory capacity on achievable many-core performance are examined. The model proposed is called the C^2-Bound, where C^2 denotes the consideration of both data access concurrency and "memory" capacity. Note that the memory capacity here is on-chip memory capacity (more discussion will be presented in Section 5).

Cassidy and Andreou have developed an objective function that links multiprocessor performance gains to data access delay and energy consumption [Cassidy and Andreou 2012]. We will follow the same discussion flow as theirs, that is, first propose object function and physical constraints and then do the optimization.

3.2. Execution Time Object Function

Let problem size (in terms of the dynamic Instruction Count) be IC. Equation (16) is the classic formulation of Central Processing Unit (CPU) time of sequential processing in terms of data stall time [Hennessy and Patterson 2012],

$$CPU\text{-}time = IC \times (CPI_{exe} + data\text{-}stall\text{-}time) \times Cycle\text{-}time. \tag{16}$$

Equation (17) is the conventional data stall time formula based on AMAT [Hennessy and Patterson 2012],

$$Data\text{-}stall\text{-}time = f_{mem} \times AMAT. \tag{17}$$

The AMAT-based Equation (17) only considers memory locality but not concurrency. Equation (17) no longer holds true when data access concurrency exists.

Recently, we have extended Equation (17) to consider both locality and concurrency [Liu and Sun 2015b]. The extended C-AMAT-based execution time is Equation (18). The $overlapRatio_{c\text{-}m}$ is the ratio of computation and memory access overlapping time over total memory active time. A rigorous proof has been made [Liu and Sun 2015b] with regard to the correctness and generality of Equation (18) for a single processor. However, two questions remain for our C^2-Bound: (1) extension to multiprocessors and (2) the consideration of memory capacity,

$$T = IC \times (CPI_{exe} + f_{mem} \times C\text{-}AMAT \times (1 - overlapRatio_{c\text{-}m})) \times Cycle\text{-}time. \tag{18}$$

We take Equation (18) as the start point to study the scalability issues. According to Sun-Ni's law, the execution time object function can be formed as Equation (19),

$$J_D = T_1 + \frac{g(N) \times T_N}{N}. \tag{19}$$

T_1 is the execution time of the serial part of the workload IC_1. T_N is the sequential execution time of parallel part of the workload IC_2. The portion of IC_1 to IC is f_{seq} and the portion of IC_2 to IC is $1 - f_{seq}$. That is,

$$IC_1 = IC \times f_{seq} \text{ and } IC_2 = IC \times (1 - f_{seq}).$$

As the parallel degree i can be from 1 to N, Equation (19) can be generalized as follows. For the brevity of discussion, we use the simple version as Equation (19), but

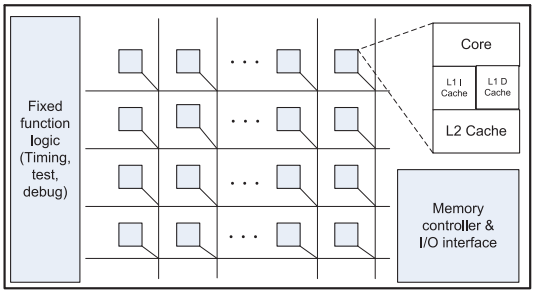

Fig. 3. Chip multiple processors.

in real CMP DSE we have implemented the generalized version,

$$J_D = \sum_{i=1}^{N} (g(i) \times T_i / i).$$

According to Sun-Ni's Law, the problem size IC can be scalable with memory capacity, and recall that the memory capacity is increasing linearly with N, so the following relation holds true, where IC_0 is the problem size when $N = 1$,

$$IC = g(N) \times IC_0. \tag{20}$$

Therefore, combining Equation (18), (19), and (20), we can derive Equation (21) as the object function for application execution time

$$J_D = IC_0 \times (CPI_{exe} + f_{mem} \times C\text{-}AMAT \times (1 - overlapRatio_{c\text{-}m}))\left(f_{seq} + \frac{g(N) \times (1 - f_{seq})}{N}\right). \tag{21}$$

Equation (21) will be used as the object function for optimization. In Equation (21), the features of data access patterns have been denoted by C-AMAT, especially data access concurrency within a single core. Now we move onto developing the constraints for the optimization problem.

3.3. Physical Constraints

Figure 3 shows a schematic illustration of CMP architecture. There are three basic components: the NoC-connected cores, the fixed function logic (timing, test, and debug), memory controllers, and input/output (I/O) interfaces. The cores each access their own subset of a coherent or non-coherent L2 cache to provide high-bandwidth L2 cache access.

Pollack's rule states that "the microprocessor performance increase due to microarchitecture advance is roughly proportional to the square root of the increase in complexity" [Borkar 2007],

$$CPI_{exe} \propto {A_0}^{-1/2}.$$

We use the rule to model the computing performance of a processor core as shown in Equation (22),

$$CPI_{exe} = k_0 {A_0}^{-1/2} + \phi_0. \tag{22}$$

Assume the chip has A area in total. For the brevity of discussion, we also assume the processor core is symmetric. As shown in Equation (23), all the cores have equal private areas. The case for asymmetric and dynamic multicore processors can be derived following Equation (24),

$$A = N(A_0 + A_1 + A_2) + A_c, \tag{23}$$

$$A = A_c + \sum_{i=1}^{N} (A_{0i} + A_{1i} + A_{2i}), \tag{24}$$

where N is the number of cores in CMP, A_0 is the area of a processors core (excluding its private cache), and A_1 is the area of the private cache of a processor. A_2 is the area of the L2 cache allocated for a given processor. A_c is the area allocated for the shared functions including shared caches, interconnections, memory controllers, test and debug, and so on.

The relation between MR and cache capacity has been explored by Hartstein et al. [2008]. The relation also holds true for pMR and cache capacity, since in most cases the gap between MR and pMR is very small [Liu and Sun 2015b]. For different data access patterns, pure miss rate has different sensitivities to the allocated cache size. This sensitivity difference can be descripted by Equation (25). The higher value of ε implies that pMR is more sensitive to cache area A_i. When ε is close to zero, the sensitivity will be very small. Equation (25) links C-AMAT value and cache size. Thus, the object function Equation (2) and constraint function Equation (23) are linked,

$$pMR_i \propto {A_i}^{-\varepsilon}. \tag{25}$$

The upper bound of concurrency, C, is limited by a few microarchitecture parameters, such as instruction window size, reorder buffer (ROB) size and miss status holding registers (MSHR) capacity in each cache layer [Chou et al. 2004]. When the upper bound of C increases, the A_0, A_1, A_2, and A_c all will be increased. For simplicity, we assume that the increase of A_0, A_1, A_2, and A_c are the same, and thus we do not need to model the area cost of increasing C.

Now that the object function and constraints considering both C-AMAT and Sun-Ni's Law have been discussed, we are ready to solve the optimization problem.

3.4. Optimization Problem and Solving

Given that the total silicon area of the chip is fixed, the allocation of the silicon for core logic, L1 cache, and L2 cache will influence the application performance. The law of diminishing marginal utility should be considered into the allocation. Based on the object function (Equation (21)) and physical constraints (Equation (23)), we formalize the CMP DSE as an optimization problem shown in Equation (26),

$$Min\, Equation(21)\ s.t.\ Equation(23). \tag{26}$$

We solve the optimization problem using the method of Lagrange multipliers, minimizing

$$L(A_1, A_2, \lambda, N) = J_D + \lambda[N(A_0 + A_1 + A_2) + A_c - A]. \tag{27}$$

Differentiating Equation (27) with respect to A_0, A_1, A_2, λ, and N, we can build a set of nonlinear equations. When using tools to analyze Equation (27), we find that

$$\frac{\partial L}{\partial N} > 0 \text{ if and only if } g(N) \geq O(N). \tag{28}$$

Therefore, the optimization falls into two cases. (1) When $g(N) \geq O(N)$, the workload size, and thus execution time, increases with N, so a single core has the shortest T and there is no need to explore the optimal N. In a given N, however, we can find the optimal cache size parameters A_0, A_1, and A_2. (2) When $g(N) < O(N)$, the workload size increase speed $g(N)$ is lower than N, and we can find the optimal core number N to minimize execution time T, and its corresponding A_0, A_1, and A_2.

Equation (28) gives us an opportunity to formalize the difference of HPC and HTC mathematically. Performance is always deemed the reciprocal of execution time, so

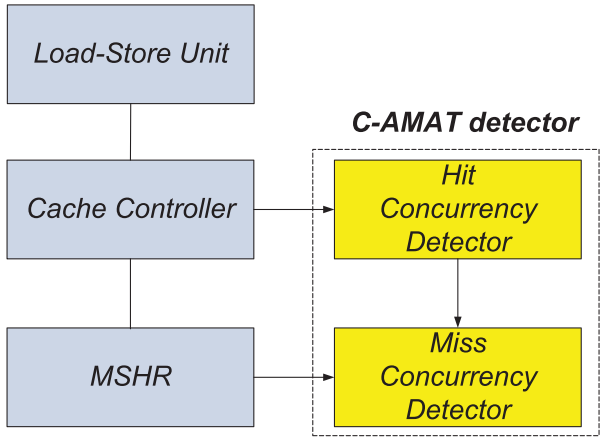

Fig. 4. C-AMAT detector.

from the HPC perspective, the goal of optimization is to minimize execution time. However, when the workload size is scalable, the optimum execution time increases with the workload proportionally. From the HTC perspective, the goal of optimization is to maximize W/T. Here W is the size of "big data."

When workload size is fixed, maximizing W/T is equivalent to minimizing T. In general, maximizing W/T is a common goal of the two cases. Maximizing W/T will boost data processing speed, which is vital for big data analysis.

3.5. APS Methodology

The analytic model Equation (27) provides a good theoretical result for CMP design, but in engineering design we often needs a more accurate design layout based on simulation. In the following, we discuss how the C^2-Bound model can guide CMP simulation.

We propose the automatic APS method for C^2-Bound-based CMP DSE. The APS method follows a "characterization + optimization + simulation" flow. The characterization collects the input parameters for the optimization. The input information can be obtained directly from an application development manual, analyzed by a compiler, or profiled by hardware detection structures.

Figure 4 illustrates the C-AMAT analyzer, which is a hardware detection system. The Hit Concurrency Detector (HCD) counts the total hit cycles and records each hit phase in order to calculate the average hit concurrency. The HCD also notifies the Miss Concurrency Detector (MCD) whether a current cycle has a hit access. Therefore, with the hit information from HCD and the miss information from MSHR, MCD is able to obtain the total number of pure miss cycles.

We have successfully collected all the needed input parameters for an application running on physical machines using Performance Application Programming Interface (PAPI) [Browne et al. 2000] and HPCToolkit [Adhianto et al. 2010]. Moreover, we also achieved the same goal with the help of GEM5 and DRAMSim2 simulators [Binkert et al. 2011; Rosenfeld et al. 2011].

Taking the parameters as inputs, Figure 5 shows the optimization flow. The left column lists the four steps: the input, formalization, solving, and the output, while the right column presents the implementation details. Note that the solution of the nonlinear equations can be found using Newton's method. We have implemented an efficient solver for the nonlinear equation set.

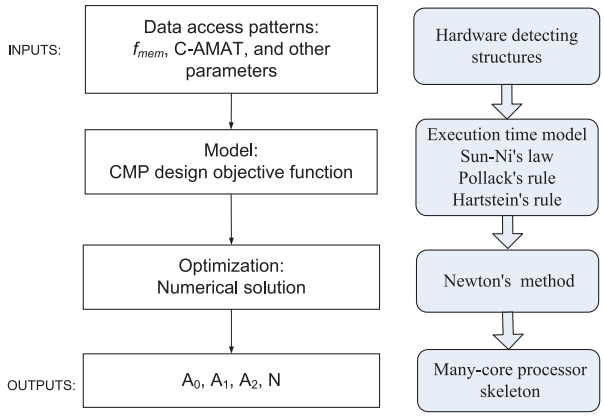

Fig. 5. Analytical methodology overview.

APS Algorithm for CMP DSE

```
// Initially measure the parameters through application characterization
  For each application, using tools to measure
      f_mem, C-AMAT, and other parameters
// Building optimization equation set
Differentiating execution time with respect to A0, A1, A2, λ, N:
  // case I when the workload size is linear or super-linear scalable
    If (g(N) ≥ O(N)) Do
      Optimizing to maximize W/T
      For a given N, output A0, A1, A2
  // case II when the workload size is fixed or inferior-linear
    If (g(N) < O(N)) Do
      Optimizing to minimize T
      Output A0, A1, A2, N
// Simulation to refine the solution
  Simulating the adjacent regions in the design space nearby the solution presented
    by the analytical model
  Output the final solution of all the microarchitecture parameters
End
```

Fig. 6. APS algorithm.

The solver can be integrated with a simulator to guide detailed evaluation. Only the adjacent regions in the design space near the solution presented by the C^2-Bound model are worth the time-consuming simulation. We formally present the APS algorithm in Figure 6.

APS is the collaboration of analytical modeling and detailed simulation. The optimal core count and the space allocation between processing and caches are determined by the optimization model. Once these fundamental parameters are fixed, the skeleton of CMP becomes clear. Based on the skeleton, microarchitecture parameters such as issue width and ROB size can be efficiently evaluated via simulation, since the design space has been narrowed significantly.

In next section, the detailed results of the verification and case studies are presented.

Table III. Default Configurations of Processor Structure

Parameter	Configurations
Number of cores	Up to 16
Clock frequency	2.4GHz
Non-blocking enabled	Out of order
Issue-width	decode/retire up to four instructions
Pipeline depth	15 stages
ROB size	128-entry
Number of MSHRs	16
Number of targets per MSHR	4
Branche predictor	Fetch up to 4 branches

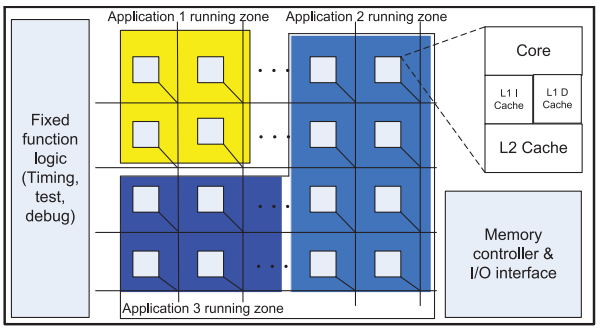

Fig. 7. Core allocation for multiple tasks in a CMP.

4. VALIDATION AND CASE STUDY

The state-of-the-art cycle-accurate simulator GEM5 [Binkert et al. 2011] and DRAMSim2 [Rosenfeld et al. 2011] are integrated to provide an appropriate memory performance simulation. As shown in Table III, we model a detailed four-way out-of-order processor with a 128-entry reorder buffer, a two-level cache hierarchy. The memory hierarchy provides enough hardware to support high concurrency, which is similar to an Intel Core i7 system [Levinthal 2009]. We also implemented on-line detecting structure for the C-AMAT analyzer shown in Figure 4.

We used benchmark suites, SPLASH-2 and PARSEC, which have several input datasets at different scales [Woo et al. 1995; Bienia et al. 2008]. Aided by SimPoint [Hamerly et al. 2004], 10 billion dynamic instructions for each benchmark were simulated to collect statistics.

Recent product announcements show a trend toward aggressive integration of many cores on a single chip to maximize throughput. However, efficiently utilizing rich resources is not easy. The behavior of an application changes phase by phase during its execution. There is no fixed hardware configuration that can work best for all the possible behaviors. Each design has its own pros and cons, depending on the interaction between the data access patterns and the underlying memory system.

Fortunately, most programs have periodic behaviors and their data access patterns are predictable [Hamerly et al. 2004]. With a set of lightweight counters, we are able to deploy proper optimization techniques to timely adapt to the underlying data access pattern changes of an application.

C^2-Bound analytic results can be either used in reconfigurable hardware environments or by software designers and applied to scheduling, partitioning, and allocating resources among diverse applications. Figure 7 includes three applications. As the sequential portion f_{seq} is very large and memory concurrency C is very low, the first

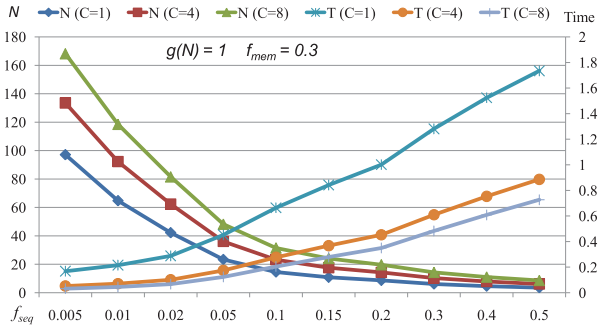

Fig. 8. Fixed problem size scaling ($g(N) = 1$, $f_{mem} = 0.3$).

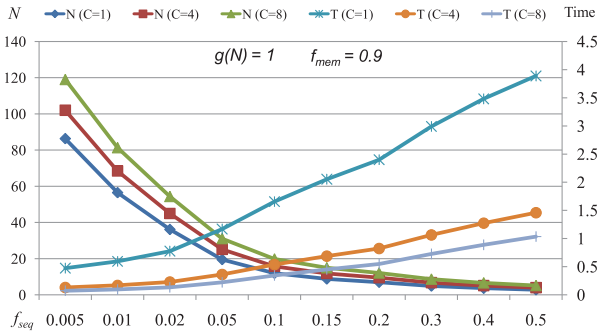

Fig. 9. Fixed problem size scaling ($g(N) = 1$, $f_{mem} = 0.9$).

application need the least number of cores and thus the benefit for allocating more cores to the first application is marginal. On the other hand, the second application has a low f_{seq} and a high C. Therefore, it is sensible to assign more cores to the second application. The third application falls somewhere between these two extremes. In this manner, the application demand can be well matched into the underlying hardware.

In the following section, we will discuss the optimal core numbers in two cases. One case is when problem size is fixed, that is, $g(N) = 1$; the second case is when workload is super linearly scalable, that is $g(N) = N^{3/2}$, which is representive for a large number of scientific applications. In each case, we will discuss three levels of memory concurrency: One has no memory concurrency, that is, $C = 1$, and one has a moderate memory concurrency, $C = 4$; the last is high memory concurrency, $C = 8$.

The purpose in this section is not to present all the results of the model but only to verify its correctness and effectiveness. We have implemented the model online to perceive the changing of the application data access patterns.

4.1. Case Study I: Problem Size Is fixed $g(N) = 1$

In the first case, we keep the workload size fixed, that is, $g(N) = 1$. Therefore, the analysis can be focused on the impact of data access patterns in terms of C-AMAT.

Assume the sequential portion of a parallel algorithm is f_{seq}. The optimal system scale N can be found for different f_{seq} values. As shown in Figures 8 and 9, memory concurrency with 1, 4, and 8, show different optimal core number requirements, especially when the value of f_{seq} is small.

Memory concurrency is a vital factor influencing the scalability of a system. When f_{seq} increases, the optimal system scale decreases. In the extreme case, if 99.5% of

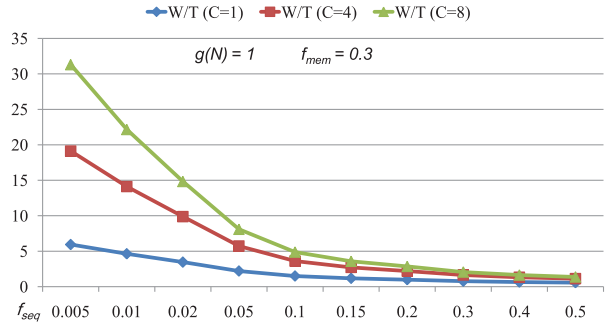

Fig. 10. The W/T of fixed problem size scaling ($g(N) = 1$, $f_{mem} = 0.3$).

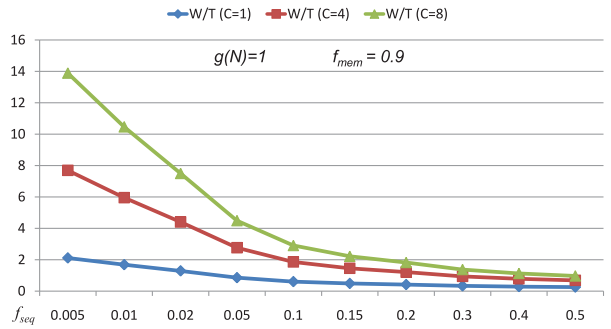

Fig. 11. The W/T of fixed problem size scaling ($g(N) = 1$, $f_{mem} = 0.9$).

the program is parallelizable, hundreds of cores are needed. However, if only a small portion of the program can be parallelized, only dozens of cores are needed, and adding more cores has no benefit and may counterproductive. The results obtained here show that our model successfully reflects the Amdahl's law.

Memory concurrency shows significant impact on the execution time, T. Higher C implies more reduction of T. This impact is especially significant when the sequential portion f is small. For example, as shown in Figure 8, when f_{seq} is 0.5%, the speedup ratio of $T(C = 1)$ over $T(C = 8)$ is 5.25; on the other hand, when f_{seq} is 50%, the speedup ratio of $T(C = 1)$ over $T(C = 8)$ is 2.38. The reason is simple. It is just Amdahl's law applied to data access. When f_{seq} is smaller, more cores will be needed, and there will be more opportunities to achieve high memory concurrency, since more cores can issue more data accesses concurrently.

Comparing Figures 8 and 9, we find that when data access frequency f_{mem} is higher, fewer cores are needed. This implies that when the workload size is fixed, if the data access frequency becomes higher, then more die area should be invested on cache and the number of cores needed should be reduced due to resource contention.

Based on the same data of Figures 8 and 9, now let us examine the throughput W/T, the ratio of problem size W over execution time T. As shown in Figure 10, by increasing f_{seq}, the ratio is decreased. Note that we have already selected the optimal core numbers and optimally allocated the silicon resource for architecture components. The decreasing trend of W/T is the result of the fixed problem size. This fact illustrates the importance of $g(N)$, the scale function of the memory-bounded problem size.

Comparing Figures 10 and 11, we find that when data access frequency f_{mem} increases, the throughput W/T decreases. That shows intensive data access can decrease computation efficiency.

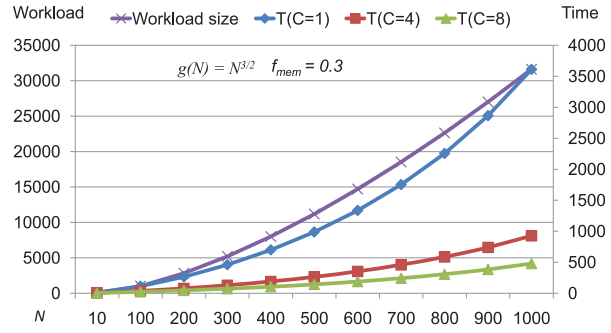

Fig. 12. The problem size W and execution time T of memory bounded scaling ($g(N) = N^{3/2}$, $f_{mem} = 0.3$).

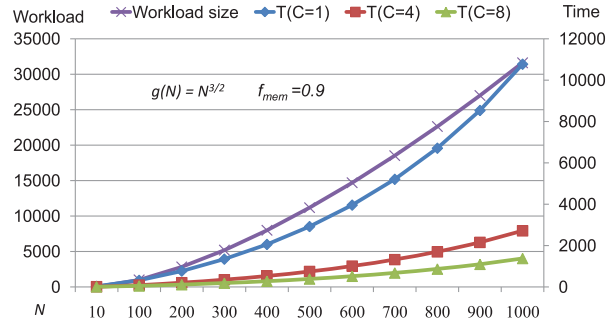

Fig. 13. The problem size W and execution time T of memory bounded scaling ($g(N) = N^{3/2}$, $f_{mem} = 0.9$).

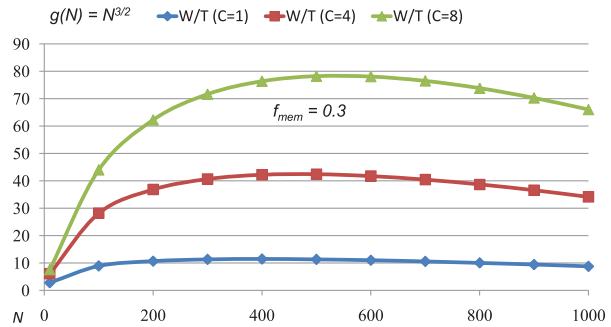

Fig. 14. The W/T of memory bounded scaling ($g(N) = N^{3/2}$, $f_{mem} = 0.3$).

4.2. Case Study II: Problem Size Increases with Scale Function $g(N) = N^{3/2}$

In the second case, we scale up the problem size with the underlying function $g(N) = N^{3/2}$.

Figures 12 and 13 show the problem size W and the execution time T of memory bounded scaling when $g(N) = N^{3/2}$ and data access frequency f_{mem} is 0.3 or 0.9. Comparing Figures 12 and 13, we find that the execution time (T) increases with data access frequency f_{mem}. Figures 14 and 15 show the throughput (W/T) values correspondingly. Comparing Figures 14 and 15, we find that the throughput (W/T) decreases with data access frequency f_{mem}.

When there is no memory concurrency ($C = 1$), the scalability curve of the execution time T is close to the problem size curve. On the other hand, higher memory

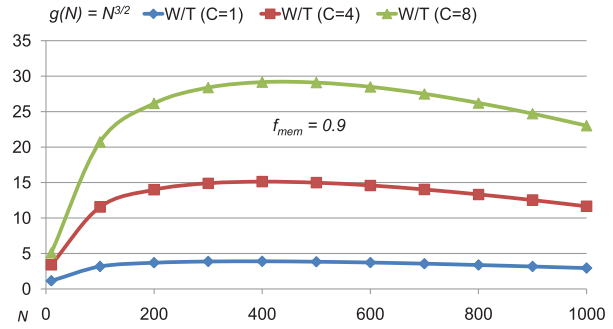

Fig. 15. The W/T of memory bounded scaling ($g(N) = N^{3/2}$, $f_{mem} = 0.9$).

concurrency leads to a better scalability in terms of execution time. For example, as shown in Figures 12 and 13, when N is 1,000, the speedup ratio of $T(C = 8)$ over $T(C = 1)$ is very significant. This tells us that, even with a fixed number of processing cores, improving data access performance via memory concurrency can obtain significant speedup. This fact is very important for the design of future supercomputers.

Based on the same data of Figures 12 and 13, Figures 14 and 15 show the curve of W/T. It can be seen that when $g(N) \geq O(N)$, higher memory concurrency makes many-core computing more efficient in terms of increasing throughput W/T. When there is no memory concurrency ($C = 1$), about 100 cores are enough to achieve the best throughput. When N is more than 100, the ratio of W over T remains approximately the same. However, when memory concurrency increases, W/T first increases and then fluctuates.

This fact can be explained as follows. At the beginning, the workload size increases with the number of cores, N. Since the N value is not significantly large, the cache hierarchy is not a bottleneck. Therefore, the increasing speed of T is lower than that of workload size W. However, when N becomes very large in CMPs, due to a fixed total area, increasing N requires the reduction of the cache size per core, which will increase cache miss after a threshold and thus decrease W/T. As W/T first increases and then fluctuates, we can find an optimal point to achieve the best throughput to foster the utilization of many-core processors. The die area tradeoff between computing and memory is an application of the Sun-Ni's law. The difference is that now we have extended it to CMP with the additional consideration of memory hierarchy.

The results presented in Figure 12 to 15 demonstrate the importance of memory concurrency and its relation with the number of cores. In general, more cores correspond to smaller cache area; the results for area allocation for cache are not presented here due to the page limitation but can be obtained at the same time with the optimization of the number of cores.

Therefore, the optimal core count N and the space allocation between processing and caches A_1 and A_2 all have been determined by our optimization model as shown in Equation (27). Since these are the most fundamental parameters of CMP, the skeleton of CMP becomes clear. Based on the skeleton, microarchitecture parameters such as issue width and ROB size can be efficiently evaluated via detailed simulation since the design space has been narrowed significantly.

Using the cycle-accurate simulator GEM5 [Binkert et al. 2011], we have done a DSE to find the optimal chip configurations for the fluidanimate benchmark from PARSEC[Bienia et al. 2008]. The fluidanimate is a computer animation application with large working sets. Six parameters (A_0, A_1, A_2, N, issue width, and ROB size) are considered and each parameter has 10 optional values, so the whole design space size

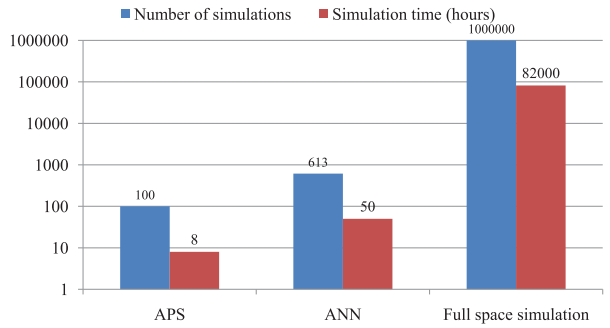

Fig. 16. The number of simulations and the total length of simulation time.

is one million (10^6). As shown in Figure 16, with the help of the C^2-Bound analysis, we do not need to run simulations to explore the A_0, A_1, A_2, and N parameters. For the rest of the parameters, issue width and ROB size, only 100 (10^2) simulations are needed. Therefore, the design space has been narrowed significantly by up to four orders of magnitude, from one million to one hundred. To evaluate the accuracy of the APS method, we run simulations to traverse the full design space, which use 128 Intel Xeon processors running for 4 weeks. Then we get performance data for each of the 10^6 different configurations, with which the APS performance data are compared, and the error is 5.96%. When concurrency is not considered, that is, when AMAT is used rather than C-AMAT, the error is increased to 112%; when capacity effect is not considered, the error is increased to 31%; when neither concurrency nor capacity effect is considered, the error is 178%. Therefore, the C^2-Bound accurately reflects the time model of modern processors.

The error 5.96% of C^2-Bound may come from the following aspects. The first is Pollack's rule in Equation (22), which shows the transformation ratio of computing over die area. The second is Hartstein's rule in Equation (25), which indicates the translation ratio of miss rate over die area size. Both formulas are empirical equations rather than laws. The third is measurement and transfer delay of performance counters. When the counter value can be measured and transferred effectively and in a timely manner, the error can be narrowed significantly. Finally, part of the error may be due to the assumption that area allocation does not change when the upper bound of concurrency increases.

Note that APS only used about 1h with 8 processors rather than 4 weeks with 128 processors. The full space simulation used 82kh while APS only uses 8h. The speedup is 10.25kilofold. Compared to the time saving, the 5.96% error is acceptable.

We also use the well-known machine-learning method ANN [Ipek et al. 2008] to predict the performance data in the huge design space. To achieve the same prediction accuracy (5.96% error), ANN needs 613 times of simulation while APS uses 100 times. Therefore, APS used only 16.3% of the simulation time to achieve the same prediction accuracy as ANN, and APS achieves 6.13-fold speedup.

5. DISCUSSIONS

For a many-core processor, a fundamental question is which layer of a memory hierarchy is the primary performance "bound" vital for many-core performance. To answer this question, we should consider three factors, latency, bandwidth, and capacity, simultaneously. Figure 17 shows a general many-core memory hierarchy, where Sun-Ni's law can be applied to each layer of the hierarchy for memory bounded analysis.

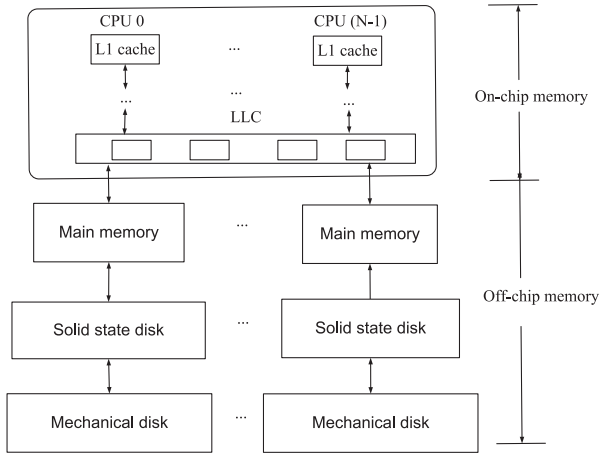

Fig. 17. The CMP memory hierarchy.

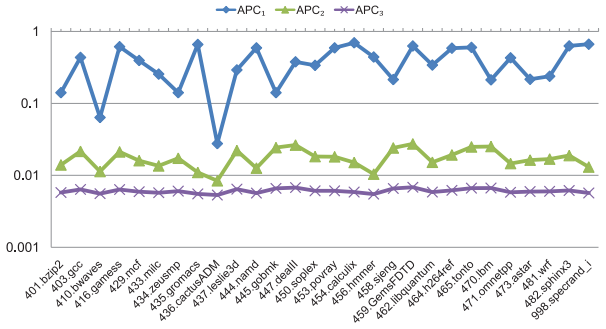

Fig. 18. The APC values at each layer of a memory hierarchy.

Initially, the interactions among the three factors are not straightforward. Fortunately, the Access Per memory-active Cycle (APC) metric can be used to represent the combined impact of latency and bandwidth [Wang and Sun 2014]. More interestingly, the previously used metric C-AMAT equals 1/APC [Sun and Wang 2014].

Data APC is a new metric to measure memory systems performance, which can be applied on each memory layer and considers both memory locality and concurrency [Wang and Sun 2014]. In Figure 18, the APC_1 is the APC value of the L1 cache, APC_2 is that of last level cache (LLC), and APC_3 is that of main memory. As a big gap between the performance of on- and off-chip cache has been shown in Figure 18, it is reasonable to conclude that in our C^2-Bound model the memory bound is the "on-chip memory bound." The "on-chip memory" herein is LLC for inclusive caches or the sum of all the on-chip caches for exclusive caches.

It is important to determine the optimal memory bounded problem size. The problem size differs with working set size. An application with a large problem size may not have a large working set. Assuming that the on-chip memory size allocated to a given application is X, the working set size [Denning 1968] is Y, and the problem size is Z, we can obtain the maximum value of the "LLC-bounded" problem size via solving the optimization problem shown in Equation (29),

$$Max \quad Z \quad s.t. \quad Y \leq X. \tag{29}$$

Note that the X value is the memory size allocated to an application. When multiple applications are running together, the X value of an application may be smaller than the total physical memory size.

The rationale behind Equation (29) is that, to fit the working set into on-chip memory, we should keep the working set size no more than the on-chip cache size; otherwise the chip performance will be decreased significantly [Denning 1968, 2005]. Assuming that the optimal on-chip memory bounded problem size is a, where $a = arg\,Max\,Z\ s.t.\,Y \leq X$, and that the real problem size is $b = Z$, the following two cases exist:

(1) If b is not bigger than a, then the application performance will be processor bound. The working set of the application is captured on chip, thus the application requires few off-chip accesses, which will cause the application performance to be insensitive to on-chip memory capacity and concurrency.
(2) If b is bigger than a, then the application performance will be limited by the rate that the data can be moved between the processor and the DRAM. Now, cache capacity and data access concurrency will impact the application performance more significantly. A "big data" application is deemed a large working set application and is likely to fall into this case. Therefore, Equation (29) shows the meaning of "big data" by quantifying the memory bound effect. To our knowledge, Equation (29) is the first analytical model for determining the optimal memory bounded problem size.

Applications may move between these two cases phase by phase, since their data access behaviors may be dynamic. Therefore, reconfigurable hardware or management software (for scheduling, partitioning, and allocating) is called for to achieve the dynamic matching between application and underlying hardware. To facilitate the implementation, an associated methodology in Figure 4 has been given to obtain the needed parameters online to facilitate the C^2-Bound model being used for these purposes.

Our model considers two new parameters: "data access concurrency" and "memory capacity-bounded problem size." The introduction of "data access concurrency" is based on the fact that concurrent data access exists at each layer of a memory hierarchy, and its impact should be considered in many-core design. The parameter memory capacity-bounded problem size is a vital factor in the scalability study, where increasing the number of cores is a common design choice of many-core design. The inclusion of these two new parameters makes the newly proposed C^2-Bound model significantly more appropriate for DSE than the existing locality-only and/or problem size fixed models for modern application-specific many-core design.

6. RELATED WORK

The many-core design exploration is a process to find an architecture with features that can well match application characteristics and then to utilize chip space efficiently to achieve excellent performance. Simulation and analytical modeling are the two basic approaches to accomplish this purpose. However, the simulation is costly and slow, typically a 1min execution of a real machine requires approximately 1 month to a year to simulate [Eeckhout 2010] an order of 10^5-to-10^6 increase in execution time. The huge design space and the high simulation cost prevent computer architects from exploring the intractable design space thoroughly. The conventional brute-force simulate-compare design process becomes painfully slow, if not infeasible, to find an optimal multicore architecture.

While simulation is an important step towards implementation, analytical methods can rapidly narrow the system design space prior to the detailed simulation. They illuminate high-level design tradeoffs and present solutions for optimal performance and efficiency.

As an analytical approach, the C^2-Bound analytical model introduced in this study is effective and new. Based on the C^2-Bound model, the newly proposed APS tool adopts an integrated analysis and simulation approach to utilize the merits of both methods. It has reduced the total simulation time significantly in an order of four folds.

Researchers have been investigating analytical methods for optimizing CMP architectures, in which some methods used in the machine-leaning domain have been used for analytical modeling such as genetic algorithms (GA) [Thompson and Pimentel 2013] and response surface modeling (RSM) [Palermo et al. 2009]. The GA and RSM are both closed-form expressions. As a result, the impacts of problem size and memory concurrency cannot be explicitly discussed.

Some open-form expressions were proposed without considering the variations of problem size and memory concurrency. Notably, the work by Hill and Marty uses a measure of processor performance to augment Amdahl's law and applies it to evaluate symmetric, asymmetric, and dynamic multi-core processors [Hill and Marty 2008]. In their work, the problem size is assumed to be fixed and the impact of memory concurrency is ignored.

Sun and Ni proposed the memory-bounded parallel speedup model, which is also known as Sun-Ni's law, in 1990 [Sun and Ni 1990]. The law shows that the scalability of computing is bounded by the problem size that is limited by memory capacity. The law that is valid for supercomputing also presents insights for CMP design. However, it needs to be revisited taking into account of the data access delay besides the problem size as well as to consider the CMP features, especially the physical resource constraints [Li et al. 2006].

With the consideration of memory-bounded problem size, Sun and Chen discussed the Sun-Ni's law based on the same CMP cost model presented by Hill and Marty and obtained very different and more optimistic results [Sun and Chen 2010]. Their results are important in locality based system design. Data access concurrency, however, was not explicitly incorporated.

Cassidy and Andreou incorporated sequential data access delay in terms of AMAT into Amdahl's law [Cassidy and Andreou 2009; Cassidy et al. 2011; Cassidy and Andreou 2012]. This work take data access patterns account into an analytical model. However, Cassidy and Andreou did not consider concurrent data accesses, and they also assumed fixed problem size. Their work can be taken as special cases of the newly proposed C^2-Bound model when there exists no memory concurrency or the problem size is fixed.

Compared with the Roofline model [Williams et al. 2010], C^2-Bound model in this work explicitly considers the impacts of memory concurrency and locality. Roofline model assumes that the working sets do not fit fully into on-chip caches and uses "operational intensity" to reflect locality. Note that the operational intensity may vary with the problem size, but in the Roofline model the issue of how to determine the optimal memory-bound problem size is not addressed [Hennessy and Patterson 2012].

In summary, our work difers fundamentally with the above investigations. We do not keep the assumptions that problem size is fixed and the memory access is sequential. For the first time, Amdahl's law is reevaluated with the simultaneously consideration of memory concurrency and memory capacity for the silicon area constrained many-core processor design.

7. CONCLUSIONS

While the number of transistors in a given die increases based on Moore's law, the utilization of these transistors continues to be a challenging task in Very Large Scale Integration (VLSI) design. This is especially true in recent years when data access becomes

the premier performance bottleneck of computing systems. To respond to the increasing importance and complexity of modern many-core architecture and memory systems, this study, for the first time, incorporates memory concurrency and memory-bounded problem size into many-core processor design space exploration. While maintaining simplicity and practical feasibility, with the consideration of both memory-concurrency and memory-bound, the newly proposed C^2-Bound model is significantly more accurate and more powerful than existing DSE models. It facilitates the studies of many-core data processing, workload scalability, and therefore reshapes the on-chip area allocation for processing cores, caches, and memory controllers. The C^2-Bound model has been implemented and can be executed automatically under the newly proposed APS algorithm for fast and accurate CMP DSE, with a combination of analysis and simulation. Analytic and implementation results show that C^2-Bound is feasible and effective. The analytical results have narrowed the design space significantly by up to four orders of magnitude. APS uses only 16.3% of the simulation time to achieve the same prediction result as the widely used standard machine-learning method, ANN [Ipek et al. 2008], for the "fluidanimate" benchmark.

The extension of CMP DSE to consider the concurrency driven data access latency and memory capacity bounded problem size is a complicated process. In this study, we have used our cumulated long time experience in memory bounded formulation and in C-AMAT development. While the C^2-Bound model is essential for next-generation data-centric processor design and for Exascale system design, this study is only the first step in considering data and scalability in CMP DSE. Moreover, energy consumption and temperature can be considered for multi-objective exploration in future refined versions. The extension of C^2-Bound to asymmetric CMP DSE can be done following the same framework with area model given by Equation (24).

The analytic results presented in this study can also be used in hardware reconfiguration environments or used by software designers for scheduling, partitioning, and allocating resource to achieve the dynamic matching between application behaviors and underlying hardware. In fact, the unified approach of combining memory-concurrency and memory-bound can be extended to general parallel computing as well.

In this article, we only focus on data access concurrency and memory bounded problem size for high-performance computing. In the future, more objects can be included in such an analysis. For example, the object function shown in Equation (21) can be reshaped to achieve a balance among performance, power, energy, and temperature [Cho and Melhem 2008; Huang et al. 2010].

ACKNOWLEDGMENTS

The authors thank Professor Yang Wang from Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, for his valuable suggestions. We also thank the members of the SCS lab of IIT for providing experimental mechanisms.

REFERENCES

Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R. Tallent. 2010. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurr. Comput.: Pract. Exp.* 22, 6 (2010), 685–701.

Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18–20, 1967, Spring Joint Computer Conference*. ACM, New York, NY, 483–485.

Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. ACM, New York, NY, 72–81.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, and others. 2011. The gem5 simulator. *ACM SIGARCH Comput. Arch. News* 39, 2 (2011), 1–7.

Shekhar Borkar. 2007. Thousand core chips: A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*. ACM, New York, NY, 746–749.

Shirley Browne, Jack Dongarra, Nathan Garner, George Ho, and Philip Mucci. 2000. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perf. Comput. Appl.* 14, 3 (2000), 189–204.

Andrew Cassidy and Andreas G. Andreou. 2009. Analytical methods for the design and optimization of chip-multiprocessor architectures. In *43rd Annual Conference on Information Sciences and Systems (CISS'09)*. IEEE, 482–487.

Andrew Cassidy, Kai Yu, Haolang Zhou, and Andreas G. Andreou. 2011. A high-level analytical model for application specific CMP design exploration. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'11)*. IEEE, 1–6.

Andrew S. Cassidy and Andreas G Andreou. 2012. Beyond amdahl's law: An objective function that links multiprocessor performance gains to delay and energy. *IEEE Trans. Comput.* 61, 8 (2012), 1110–1126.

Tianshi Chen, Qi Guo, Ke Tang, Olivier Temam, Zhiwei Xu, Zhi-Hua Zhou, and Yunji Chen. 2014. Archranker: A ranking approach to design space exploration. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. IEEE, 85–96.

Sangyeun Cho and Rami G. Melhem. 2008. Corollaries to Amdahl's law for energy. *Comput. Arch. Lett.* 7, 1 (2008), 25–28.

Yuan Chou, Brian Fahs, and Santosh Abraham. 2004. Microarchitecture optimizations for exploiting memory-level parallelism. *ACM SIGARCH Comput. Arch. News* 32, 2 (2004), 76.

Peter J. Denning. 1968. The working set model for program behavior. *Commun. ACM* 11, 5 (1968), 323–333.

Peter J. Denning. 2005. The locality principle. *Commun. ACM* 48, 7 (2005), 19–24.

Lieven Eeckhout. 2010. Computer architecture performance evaluation methods. *Synth. Lect. Comput. Arch.* 5, 1 (2010), 1–145.

John L. Gustafson. 1988. Reevaluating amdahl's law. *Commun. ACM* 31, 5 (1988), 532–533.

Greg Hamerly, Erez Perelman, and Brad Calder. 2004. How to use simpoint to pick simulation points. *ACM SIGMETRICS Perf. Eval. Rev.* 31, 4 (2004), 25–30.

Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju Mancheril, Anastassia Ailamaki, and Babak Falsafi. 2007. Database servers on chip multiprocessors: Limitations and opportunities. In *Proceedings of the Biennial Conference on Innovative Data Systems Research*. ACM, New York, NY.

A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma. 2008. On the nature of cache miss behavior: Is it $\sqrt{2}$? *J. Instr.-Level Parall.* 10, 8 (2008), 1–22.

John L. Hennessy and David A. Patterson. 2012. *Computer Architecture: A Quantitative Approach*. Elsevier, New York.

Mark D. Hill and Michael R. Marty. 2008. Amdahl's law in the multicore era. *Computer* 41, 7 (2008), 33–38.

Wei Huang, Kevin Skadron, Sudhanva Gurumurthi, Robert J. Ribando, and Mircea R. Stan. 2010. Exploring the thermal impact on manycore processor performance. In *Proceedings of the 26th Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM'10)*. IEEE, 191–197.

Engin Ipek, Sally A. Mckee, Karan Singh, Rich Caruana, Bronis R. De Supinski, and Martin Schulz. 2008. Efficient architectural design space exploration via predictive modeling. *ACM Trans. and Code Opt.* 4, 4 (2008), 178–184.

David Levinthal. 2009. Performance analysis guide for Intel core i7 processor and Intel Xeon 5500 processors. *Intel Performance Analysis Guide* 30 (2009), 18.

Yingmin Li, Benjamin Lee, David Brooks, Zhigang Hu, and Kevin Skadron. 2006. CMP design space exploration subject to physical constraints. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture, 2006*. IEEE, 17–28.

Yu-Hang Liu and Xian-He Sun. 2015a. C^2-Bound: A capacity and concurrency driven analytical model for manycore design. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage (SC'15)*. IEEE, 1–11.

Yu-Hang Liu and Xian-He Sun. 2015b. Reevaluating data stall time with the consideration of data access concurrency. *J. Comput. Sci. Technol.* 30, 2 (2015), 227–245.

Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. 2009. ReSPIR: A response surface-based Pareto iterative refinement for application-specific design space exploration. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 28, 12 (2009), 1816–1829.

Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *Comput. Arch. Lett.* 10, 1 (2011), 16–19.

Stephen Somogyi, Thomas F. Wenisch, Anastasia Ailamaki, and Babak Falsafi. 2009. Spatio-temporal memory streaming. *ACM SIGARCH Comput. Arch. News* 37, 3 (2009), 69–80.

Xian-He Sun. 2014. Concurrent-AMAT: A mathematical model for big data access. *HPC Mag.* 5, 1 (2014), 1–4.

Xian-He Sun and Yong Chen. 2010. Reevaluating Amdahl's law in the multicore era. *J. Parallel Distrib. Comput.* 70, 2 (2010), 183–188.

Xian-He Sun and Lionel M. Ni. 1990. Another view on parallel speedup. In *Proceedings of Supercomputing'90.* IEEE, 324–333.

Xian-He Sun and Dawei Wang. 2014. Concurrent average memory access time. *IEEE Comput.* 47, 5 (2014), 74–80.

Mark Thompson and Andy D. Pimentel. 2013. Exploiting domain knowledge in system-level mpsoc design space exploration. *J. Syst. Arch.* 59, 7 (2013), 351–360.

Dawei Wang and X. Sun. 2014. APC: A novel memory metric and measurement methodology for modern memory system. *IEEE Trans. Comput.* 63, 7 (2014), 1626–1639.

Samuel Williams, Andrew Waterman, and David Patterson. 2010. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2010), 65–76.

Dong Hyuk Woo and Hsien-Hsin S. Lee. 2008. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer* 41, 12 (2008), 24–31.

Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. *ACM SIGARCH Comput. Arch. News* 23 (1995), 24–36.

Wm A. Wulf and Sally A. McKee. 1995. Hitting the memory wall: Implications of the Obvious. *ACM SIGARCH Comput. Arch. News* 23, 1 (1995), 20–24.

Received March 2016; revised November 2016; accepted November 2016