APC: A Novel Memory Metric and Measurement Methodology for Modern Memory Systems

Dawei Wang, *Member*, *IEEE* and Xian-He Sun, *Fellow*, *IEEE*

Abstract—Due to the infamous "memory wall" problem and a drastic increase in the number of data intensive applications, memory rather than processors has become the leading performance bottleneck in modern computing systems. Evaluating and understanding memory system performance is increasingly becoming the core of high-end computing. Conventional memory metrics, such as miss ratio, AMAT, etc., are designed to measure a given memory performance parameter, and do not reflect the overall performance or complexity of a modern memory system. On the other hand, widely used system-performance metrics, such as IPC, are designed to measure CPU performance, and do not directly reflect memory performance. In this paper, we propose a novel memory metric called Access Per Cycle (APC), which is the number of data accesses per cycle, to measure the overall memory performance with respect to the complexity of modern memory systems. A unique contribution of APC is its separation of memory evaluation from CPU evaluation; therefore, it provides a quantitative measurement of the "data-intensiveness" of an application. Simulation results show that the memory performance measured by APC captures the concurrency complexity of modern memory systems, while other metrics cannot. APC is simple, effective, and is significantly more appropriate than existing memory metrics in evaluating modern memory systems.

Index Terms—Memory performance measurement, memory metric, measurement methodology

1 Introduction

The rapid advances of semiconductor technology have driven large increases in processor performance over the past thirty years. However, memory performance has not experienced such dramatic of gains as processors; this leaves memory performance lagging far behind CPU performance. This growing performance gap between processor and memory is referred to as the "memory wall" [1], [2]. The "memory wall" problem is experienced not only in main memory but also in on-die caches. For example, in the Intel Nehalem architecture CPU, each L1 data cache has a four-cycle hit latency, and each L2 cache has a 10-cycle hit latency [3]. Additionally, the IBM Power6 has a four-cycle L1 cache hit latency and an L2 cache hit latency of 24 cycles [4]. The large performance gap between processor and memory hierarchy makes memory-access the dominant performance factor in high-end computing. Recent research tries to improve the performance of memory systems. However, understanding the performance of modern hierarchical memory systems remains elusive for many researchers and practitioners.

While memory ("memory" is referred to as synonym for the entire memory hierarchy for the remainder of this paper) is the bottleneck for performance, how to measure and evaluate memory systems has become an important issue facing the high performance computing community. The conventionally used performance metrics, such as IPC (Instruction Per Cycle) and Flops (Floating point operations per second), are designed from a computing-centric point-of-view. As such, they are comprehensive but affected by instruction sets, CPU micro-architecture, memory hierarchy, and compiler technologies, and cannot be applied directly to measure the performance of a memory system. On the other hand, existing memory performance metrics, such as miss rate, bandwidth, and average memory access time (AMAT), are designed to measure a particular component of a memory system or the performance of a single access of the memory system. They are useful in optimization and evaluation of a given component, but cannot accurately characterize the performance of the memory system as whole. In general, component improvement does not necessarily lead to an improvement in overall performance. For instance, when miss rate decreases, IPC may not increase, and sometimes IPC will decrease. (See Section 4.2 for details.) When non-blocking caches are used, the AMAT metric shows a negative effect on IPC. (See Section 4.2.3 for details.) Since there is no known correlation study between existing memory metrics and the final system performance, a frequent and common question of practitioners is whether a component improvement actually leads to a system improvement. Therefore, an appropriate metric to measure memory systems is critically needed to analyze system design and performance enhancements.

There are several reasons that traditional memory performance metrics cannot characterize the overall performance of a memory system. First, modern CPUs exploit several ILP (Instruction Level Parallelism) technologies to overlap ALU instruction executions and memory accesses. Out-of-order execution overlaps CPU execution time and memory access delay, allowing an application to hide the miss penalty of an L1 data cache miss that hits the L2 cache. Multithreading technology, such as SMT [5] or fine-grained multithreading [6], can tolerant even longer misses through main memory by

• D. Wang is with the Department of Computer Sciences, Illinois Institute Technology, Chicago, IL 60616. E-mail: david.albert.wang@gmail.com.
• X. Sun is with the Department of Computer Sciences, Illinois Institute Technology, Chicago, IL 60616. E-mail: sun@iit.edu.

Manuscript received 07 Dec. 2011; revised 21 Dec. 2012; accepted 04 Feb. 2013. Date of publication 24 Feb. 2013; date of current version 27 June 2014. Recommended for acceptance by E. Miller.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.38

executing another thread. Speculation mechanisms are used to overcome control dependencies, which helps to avoid CPU stalls. Speculation can also activate memory access instructions that are not committed to the CPU registers due to miss predictions. Incorrect speculations can aggravate the burden of data access, and the correctness of speculations is hard to predict. Even worse, a wrong prediction does not necessarily mean that the prefetch is useless. If a wrongly predicted data load accesses the same cache block as the next data load, then the incorrect speculation can be seen as an effective data prefetch, and it provides a net benefit to memory performance. All of these problems make component-based measurement unsuitable for the measurement of the overall performance of a memory system.

Additionally, modern memory systems use a large number of advanced caching technologies to decrease cache latency. Some widely used cache optimization methods, such as non-blocking cache [7], pipelined cache [8], multibanked cache [9], and data prefetching [10], allow cache accesses generated by CPU or prefetcher to overlap with each other. These technologies make the relation between memory access and processor performance even more complicated, since the processor could continue executing instructions or accessing memory even under multiple cache misses. Even more complicated, the modern commercial state-of-the-art multi-core processors, such as IBM POWER7 [11], Intel Nehalem [3], or even embedded ARM processors Cortex-A15 [12], adopt one or two levels of private caches and one shared last level cache (SLLC) on chip. Threads executed on these cores contend or share their SLLC. This blurs the effects of whole memory system performance into final processor performance. These advanced memory technologies and private/shared memory hierarchal structures make the behavior of memory accesses much the same as instructions dispatch because they both contain methods for executing tens or even hundreds of operations at once. Evaluating memory systems from a single memory access or on a single component does not match the complexity of modern memory systems. As with the measurement of instruction executions, memory system evaluations should consider all the underlying parallelism and optimizations to measure the overall performance of a memory system.

Based on the above observations, a new metric for overall memory system performance, which is separate from CPU performance and in the meantime correlates with CPU performance, is needed. In the other words, it should correlate with IPC but measure data access only. The notion of correlating with IPC is important due to the fact that memory performance is a determining factor of the overall performance of modern computing systems. The requirement of separating computing from data access is to provide a better understanding of memory systems, a better understanding of the capacity of a memory system to handle the so-call data-intensive applications, and a better understanding of the match between computing power and memory system performance. To reach this goal, the Access Per Cycle (APC) metric is proposed following the design philosophy of Instructions Per Cycle (IPC).

In this study, the definition of APC is introduced, methods of determining the number of accesses and access cycles are explored, methods of determining measurements at different memory hierarchies of single-core or multi-core systems are discussed, and a series of simulations are conducted to confirm that APC is significantly more appropriate than the existing memory performance metrics. The statistical variable correlation coefficient is used to demonstrate that for single-core L1 APC has a 0.97 correlation coefficient value with the overall computing performance in terms of IPC, and the value for multi-core is 0.96, whereas conventional metrics have only a 0.67 correlation in the best scenarios.

The rest of this paper is organized as follows. Section 2 defines APC, and describes its measurement. Section 3 introduces the experiment methodology and setup. Section 4 compares APC with other conventional memory metrics in both single-core and multi-core cases. Section 5 discusses the applications of APC and presents a quantitative definition of Data Intensiveness based on the concept of APC. Section 6 presents related works, and Section 7 concludes this study and discusses future works.

TABLE 1
ILP and Memory Optimization Comparison

ILP Tech.	Memory Tech.	Key feature in common
Pipelined stage in CPU data path	Pipelined Cache	Operation overlapping
Multiple Function Unit	Multiport/Multibanked Cache	Simultaneous operation dispatching
Out-of-order execution	Non-blocking Cache	Do not stall ready operations
Branch prediction/ Speculation/ Runahead[13]	Data Prefetching	Pattern recognizing and only successful with certain possibility

2 APC Definition and Measurement Methodology

After more than thirty years of development, ILP and memory optimization technologies have many key features in common with modern computer systems. Based on this fact, we mimic the definition of Instruction Per Cycle (IPC) to the definition of Access Per Cycle (APC) metric. A formal definition is provided and an investigation is conducted into the correct measurement of memory access cycles in advanced non-blocking memory structures with data prefetching.

2.1 APC Definition

IPC (Instruction Per Cycle) has been widely used as a major performance evaluation metric in the computer architecture community. It reflects the overall performance in terms of the number of executed instructions per cycle. As computer systems evolve, the technologies adopted in Instruction Level Parallelism and memory optimization tend to be similar. Table 1 lists some common features existed in these technologies.

Based on the similarity between processors and memory systems, and inspired by the success of IPC, APC (Access Per Cycle) is proposed to evaluate memory system performance. APC measures the number of memory accesses per cycle. It can be applied at each level of a memory hierarchy. In other words, APC is the overall memory accesses requested at a certain memory level (i.e., L1, L2, L3, Main Memory) divided by the total number of memory active cycles at that level. Please notice that memory active (access) cycle is not the same as CPU cycle. So, more accurately, APC can be called APMAC (access per memory active cycle). Let M denote the total data access (load/store) at a certain memory level, and T denote the total cycles consumed by these accesses. According to the definition of APC,

$$APC = \frac{M}{T}. \tag{1}$$

The definition is simple enough. However, because modern memory systems adopt many advanced optimizations, such as pipelined cache, non-blocking cache, multi-banked cache, and data prefetching, several outstanding memory accesses may co-exist in the memory system at the same time. Therefore, counting cycles, T, is not as simple as it looks. In the APC definition, it is defined as the total cycle T to be measured based on the *overlapping mode*, which means when there are several memory accesses co-existing during the same cycle, T only increases by one. For memory accesses, the *non-overlapping mode* is adopted. That is all the memory accesses issued by CPU or prefetcher are counted, including all successful or non-successful speculated memory accesses, all successful and non-successful prefetching accesses, and all other concurrent accesses. The reason of including all kinds of memory accesses is that no matter whether the value is used by CPU, the memory access activities are carried out by the memory system, they are the workload of the memory system. In the non-overlapping mode, if there are two L1 cache load requests at the same time, M will increase by two.

Since APC carefully used memory active cycle in its definition, each memory level has its own memory active cycle and its APC value. In this paper, the focus is on APC for L1 cache with some discussion of main memory APC. The APC of L1 reflects the overall performance of the memory system, while the main memory's APC is important since it has the longest access latency in the memory hierarchy without considering I/O and file systems. The study of these two based on single/multi-core environments should be sufficient in illustrating the APC concept and demonstrating its effectiveness. To avoid confusion, the term $\mathrm{APC_D}$ is used for APC of L1 data cache, which is the number of L1 data cache accesses divided by the number of overall data cache access cycles. Additionally, the term $\mathrm{APC_I}$ is used for L1 instruction cache, which is the number of L1 instruction cache accesses divided by the number of overall instruction cache access cycles. Finally, main memory APC is denoted as $\mathrm{APC_M}$, which is the number of off-chip accesses divided by the number of overall main memory access cycles.

2.2 APC Measurement Methodology

To cooperate with modern CPU out-of-order speculation, multi-issue, multi-threading, and multi-core technologies, modern CPUs, such as Intel Core 2 [14], Itanium [15], and IBM POWER3 [16], employ non-blocking cache heavily at each level of a memory hierarchy in order to enhance memory access parallelism. Non-blocking cache can continue supplying data under a certain number of cache misses by adopting Miss Status Holding Register (MSHR) [7]. MSHR is a structured table. It records cache miss information, such as access type (load/store), access address, and return register. When the MSHR table is empty, there are no outstanding cache misses. When the MSHR is attached to LLC (Last Level Cache) and empty, designates there are no outstanding main memory accesses. When the MSHR table is full, the cache cannot queue more cache accesses and the CPU's memory accesses or next-level memory accesses are blocked due to lack of MSHR entries.

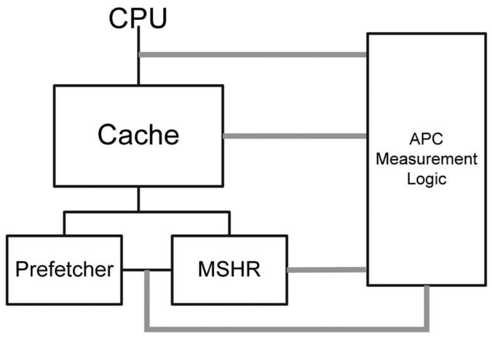


Fig. 1. APC Measurement Structure (Black lines are for cache structure, gray lines are for AML detecting logic).

In order to further overcome the "memory wall" high-latency problem, modern commercial CPUs adopt several kinds of data prefetcheres, such as Tagged Prefetcher in HP PA 7200 RISC [17], Sequential Prefetcher in IBM POWER3 [16], POWER4 [18], Stride Prefetcher [19] in Intel Core 2 [14], Nehalem [20], and Double Stride Prefetch in AMD memory controller [21]. To evaluate modern memory system performance, non-blocking cache combined with data prefetcher must be given consideration.

There are two cooperating modes between a Cache and Prefetcher. The first one is referred to as unified mode. In the unified mode, the cache and the related prefetcher have one unified data buffer and one port connected with the lower level cache. The prefetcher only observes miss pattern, and generates prefetching requests through the unified port. The prefetched cache block has no difference with normal cache blocks. They both follow the same placement and replacement algorithms. The prefetcher itself does not have separate buffer. The complexity of cache read/write and cache coherence problem are not aggravated by this prefetcher structure. The implementation of unified mode is relatively simple, and widely used in modern commercial general purpose CPUs, for example IBM POWER4 [18], and Intel Core 2 [14] and Nehalem [20]. The other cooperating mode is referred to as separate mode. In the separate mode, the cache and the prefetcher each has its own buffer and data port connected to the lower level cache individually. The prefetcher observes miss patterns, and directly sends memory prefetching requests to the lower level cache through its own connection. The returned cache block from the lower level cache will be stored in the prefetcher buffer. The cache will also examine the prefetcher buffer when a cache miss occurs. If the missed data is available in the prefetcher buffer, the data will be migrated into the cache buffer. The operation procedures of normal cache read/write and cache coherence are dramatically changed in the separate mode. The cost of the separate mode prevents its wide adoption in modern general purpose CPUs. For this reason, this study focuses on the non-blocking cache structure with a unified prefetcher, as shown in Fig. 1. There is only one data buffer inside the cache. The cache will send its miss accesses to both the Prefetcher and MSHR. The

TABLE 2
Pseudo Code for Memory Access Cycle Counting Logic

```
If(MSHR table is not empty) //Having pending cache miss/misses
    Mem_Cycle ++;
Else if(Cache is accessing)//Cache lookup exists
    Mem_Cycle ++;
Else if(CPU/Cache or Prefetcher/MSHR bus is active)
    //Having a request generated by CPU or prefetcher, or returning data
    Mem_Cycle ++;
Else
    Mem_Cycle does not change
```

Prefetcher uses the miss pattern to generate prefetching requests, which are also sent to the MSHR. The MSHR will register these miss accesses to support non-blocking access, and send miss accesses to the lower level cache. The MSHR is the only way requests are sent to the lower level cache. Also when data is retrieved from the lower level cache, a registration cancellation procedure is activated inside the MSHR.

Calculating an accurate number of overall memory access cycles in a non-blocking cache with unified prefetcher structure is not simple. There are two reasons. First, unlike IPC, not every clock cycle has memory access; therefore, any measurement scheme requires some form of access detection. Secondly, many different memory accesses can be overlapping with each other. When counting memory access cycles, it is not based on single memory access, but based on memory access detection structure of modern cache at every cycle. Only one clock cycle can be counted into the total memory access cycles even if there are several different memory accesses occurring at the same time. In practice, there could be many different ways to measure the clock cycles in the overlapping mode. In this study, a memory access cycle counting methodology and measurement logic (shown in Fig. 1) is proposed which is suitable for a modern memory hierarchy, utilizing non-blocking cache with a prefetcher.

To avoid overlapping memory accesses to be counted multiple times in an access cycle, the APC Measurement Logic (AML) simultaneously detects memory access activities from CPU, Prefetcher, Cache and MSHR. If at least one memory access activity exists in the CPU/Cache interface bus, Prefetcher/MSHR bus, or inside the Cache, or if outstanding cache miss/misses are registered in the MSHR, this clock cycle is counted as one memory access cycle. With this structure, it is possible to calculate total memory access cycles, referred to as T in Equation (1). Additionally, the AML will count the number of CPU load/store accesses by detecting CPU/Cache bus requests, and count the number of Prefetcher load accesses by detecting Prefetcher/MSHR bus requests. The total number of these two kinds of accesses added together is referred to as M in Equation (1). If there are several memory requests at the same time from the two kinds of buses, all are counted. With the number of total accesses M and the total memory access cycle T, APC can be evaluated for this level of cache. The pseudo code of memory access counting logic of on-chip caches including L1, L2, or even L3 caches, is shown in Table 2. For L2 and L3 caches, while the logic is the same, the implementation may need to detect multiple buses between upper level caches and itself.

In general, memory access cycles consist of three different timing stages, namely Bus Transferring Cycles, Cache Cycles, and MSHR Cycles. The three parts are mutually exclusive. The MSHR cycles is the time spent in MSHR, it reflects the next level of the memory hierarchies ability to supply data. Only when there is no MSHR cycle counted, should the Cache Cycle be examined. Cache Cycles are the time spent in the cache determining if a hit or miss has occurred. It reflects the local caches ability to provide data and is heavily dependent on local cache organization, such as cache capacity, associativity, etc. If both MSHR and cache are not active, Bus Transferring Cycles should be examined. Bus Transferring Cycles involve two kinds of buses, one is the CPU/Cache bus, and the other is the Prefetcher/MSHR bus. A CPU/Cache Bus Transferring Cycle is the time consumed by the CPU and Cache in transferring requests or data. A Prefetcher/MSHR Bus Transferring Cycle is the time consumed by the Prefetcher and MSHR in transferring data load requests.

The APC Measurement Logic (AML) in Fig. 1 only needs two long-bit sized registers (e.g., 64-bit register is sufficient for most computer systems) and some detecting logic. One register counts the total number of memory access cycles; the other counts memory accesses. While a memory accesses count is already provided by existing CPU performance counters, some detecting logic for counting the number of memory access cycles must be added. The detecting logic include CPU/Cache interface detecting logic, Prefetcher/MSHR interface detecting logic, cache detecting logic, and MSHR detecting logic. For MSHR, the detecting logic only needs to detect whether the MSHR table is empty, thus only one status bit is required. Modern caches adopt pipelined structures for data access where a typical pipeline structure consists of Decode, Compare Tags, Rd/Wr Data, and Drive Out stages [22]. For the cache detecting logic, only one bit for each stage is needed. Thus if the length of the pipeline stage is four, then the width of the cache detecting status register is four bits. For CPU/Cache interface detecting logic, the command and/or data bus need to be detected. The total bit-width of the command and data buses is usually less than 512 bits [3]. Therefore, the length of the CPU/Cache interface detecting logic should be less than or equal to 512 bits, assuming using one bit to detect one bit line of the bus. For Prefetcher/MSHR interface, it is only necessary to detect the command bus changes, which should require less area than the CPU/Cache interface detecting logic. Including all the detecting logic and the two registers, the total cost of AML is less than 1K bits, which is negligible when compared with modern transistor sizes in a CPU.

The biggest difference for AML between single core CPU and multi-core CPU is detecting the shared Last Level Cache (LLC). For multi-core version, the LLC is shared by many different cores. Therefore, multiple buses which are connected to the upper level private caches must be monitored simultaneously. The total number of memory accesses to the shared LLC is the summary of all bus requests. The data requests and transferring cycles existing in multiple buses at the same time must be measured in overlapping mode, which means only one cycle is added into total access cycles. If the shared LLC is L2 cache, private L1 Data Caches (DCache) and Instruction Caches (ICache) will have a common bus connected to the L2 cache. Therefore, for n-core two-level cache CPU, there are n buses must be monitored simultaneously for measuring

$\mathrm{APC_{L2}}$ (L2 APC) value. In this study, L2 as the shared LLC is examined. Therefore,

$$\mathrm{APC_{L2}} = \frac{\sum_0^n M_i + M_{prefetcher}}{\mathrm{T}}.$$

The AML for main memory requires the least hardware. When measuring main memory $\mathrm{APC_M}$, only main memory access count and LLC MSHR Cycles need to be detected. The former can be found in CPU performance counters; and the latter, only requires 1 bit to detect whether the MSHR table is empty or not. As a result, there is almost no extra hardware cost to measure the $\mathrm{APC_M}$. Therefore,

$$\mathrm{APC_M} = \frac{\text{Main Memory access count}}{\text{LLC MSHR cycles}}.$$

Our AML design is only one possible way to measure APC. It may not be the best design, but it is good enough to demonstrate that with a minor modification of current cache structures the APCs of all cache levels can be measured easily.

Table 3
Default Simulation Configuration

Parameter	Value
Processor Function units ROB, LSQ size	1core, 2 GHz, 8-issue width, 6 IntALU 1 cycle, 1 IntMul 3 cycles, 2 FPAdd 2 cycles, 1 FPCmp 2 cycles, 1 FPCvt 2 cycles, 1 FPMul 4 cycles, 1 FPDiv 12 cycles ROB 192, LQ 32, SQ 32
L1 caches	32KB Inst/32KB Data, 2-way, 64B line, hit latency: 2 cycle Inst/2 cycle Data, ICache 10 MSHR Entry, DCache 10 MSHR Entry No Prefetcher
L2 cache	2MB, 8-way, 64B line, 12-cycle hit latency, 20 MSHR Entry No Prefetcher
DRAM latency Width	200-cycle access latency 64 bits

3 Experiments Methodology and Setup

It is widely accepted that memory systems have become a key performance factor of overall application performance. Thus, the overall memory performance should influence and correlate to the overall application performance. An appropriate memory metric should reflect this correlation relation. The mathematical concept of correlation coefficient is used to describe the variation similarity between two variables. Therefore, it is expected that the memory metric with a higher correlation value is a more appropriate metric in measuring modern memory systems. This correlation relation should maintain even for most advanced memory optimizations. The correlation relation simply says if your memory measurement shows that you have optimized your memory performance, and then your application performance should improve as well, or at least not decrease.

In this section, the mathematical concept of correlation coefficient is introduced and its reflection of the relation between the two factors is described. With correlation coefficient as the proximity measurement, a series of experiments of utilizing single core and multi-core systems are executed, with changes to the hardware configurations, such as L1, L2 cache size and/or set associativity, main memory access latency, and the number of MSHR entries. In next section, the M5 (also known as GEM5) simulator is used to compare different memory performance metrics based on their correlation to IPC.

3.1 Introduction to Correlation Coefficient

The motivation of memory evaluation is due to the fact that the final system computing performance is heavily influenced by memory system performance. Therefore, an appropriate memory metric should reflect the system performance. In other words, the system performance should correlate with memory performance. Mathematically, correlation is measured by the statistic variable correlation coefficient (CC). The correlation coefficient describes the proximity between two variables' changing trends from a statistics viewpoint. The mathematic definition of correlation coefficient is shown in Equation (2). In the equation, array X and Y are the sampling points for two variables.

$$r_{xy} = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{\left[\left(\sum X^2 - \frac{(\sum X)^2}{n}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{n}\right)\right]}} \quad (2)$$

The correlation coefficient is a number between -1 and 1. The higher the absolute value of correlation coefficient is, the closer the relation between the two variables is. If it is 1 or -1, the two variables trends are perfectly match to each other; if it is 0, then there is no relation between the two variables. Generally speaking, if the absolute value is greater than 0.8, then it is believed the two variables have a strong relation; if greater than 0.9, it is a dominant relation; otherwise if less than 0.5, it is a weak relation. Also if the correlation coefficient value is larger than 0, it is a positive relation. That means if one increases, the other also increases; otherwise if it is less than 0, it is a negative relation between two variables. That means if one increases, the other decreases [23].

3.2 Experiment Setup

A detailed out-of-order speculative superscalar CPU model in the M5 simulator [24] was adopted, which models an Alpha 21264-like CPU. Unless stated otherwise, the experiments assume the following default processor and cache configuration showing in Table 3. Furthermore, the multi-core simulation is nearly identical to the default simulation except the L2 cache is share by the four cores.

Each experimental configuration is based on the default and only changes one or two parameter of the simulation. The detailed experimental configurations are shown in Table 4. Each configuration is simulated in single core and multi-core simulation environments.

There are three groups of configurations. The first group, including configuration C1 $\sim$ C17, are the basic cache/memory configurations, which only change the cache size,

Table 4
A Series of Simulation Configurations

ID	Description	Changed Parameter/s
C1	L1:32KB,2way; L2: 2MB,8way; Mem100ns	Default Config
C2	L1:32KB,4way; L2: 2MB,8way; Mem100ns	L1 Cache Assoc.
C3	L1:32KB,8way; L2: 2MB,8way; Mem100ns	L1 Cache Assoc.
C4	L1:64KB,2way; L2: 2MB,8way; Mem100ns	L1 Cache Size
C5	L1:64KB,4way; L2: 2MB,8way; Mem100ns	L1 Cache Size & Assoc.
C6	L1:64KB,8way; L2: 2MB,8way; Mem100ns	L1 Cache Size & Assoc.
C7	L1:I$32KB,2way, D$64KB,2way; L2: 2MB,8way; Mem100ns	Only DCache Size
C8	L1:I$64KB,2way, D$32KB, 2way; L2: 2MB,8way; Mem100ns	Only ICache Size
C9	L1:I$64KB,4way, D$32KB, 2way; L2: 2MB,8way; Mem100ns	Only ICache Size & Assoc.
C10	L1:I$64KB,8way, D$32KB, 2way; L2: 2MB,8way; Mem100ns	Only ICache Size & Assoc.
C11	L1:32KB,2way; L2: 4MB,8way; Mem100ns	L2 Cache Size
C12	L1:32KB,2way; L2: 8MB,8way; Mem100ns	L2 Cache Size
C13	L1:32KB,2way; L2: 2MB,16way; Mem100ns	L2 Cache Assoc.
C14	L1:32KB,2way; L2: 4MB,16way; Mem100ns	L2 Cache Size & Assoc.
C15	L1:32KB,2way; L2: 8MB,16way; Mem100ns	L2 Cache Size & Assoc.
C16	L1:32KB,2way; L2: 2MB,8way; Mem30ns	Main memory latency
C17	L1:32KB,2way; L2: 2MB,8way; Mem60ns	Main memory latency
C18	L1:32KB,2way, Stride Prefetch, degree 1; L2: 2MB,8way; Mem100ns	Prefetching Activation
C19	L1:32KB,2way, Stride Prefetch, degree 2; L2: 2MB,8way; Mem100ns	Prefetching Degree
C20	L1:32KB,2way, Stride Prefetch, degree 4; L2: 2MB,8way; Mem100ns	Prefetching Degree
C21	L1:32KB,2way, MSHR 1; L2: 2MB,8way; Mem100ns	MSHR Entry
C22	L1:32KB,2way, MSHR 2; L2: 2MB,8way; Mem100ns	MSHR Entry
C23	L1:32KB,2way, MSHR 16; L2: 2MB,8way; Mem100ns	MSHR Entry

associativity, or memory latency. These basic configurations have 10 MSHR entries each. $C18 \sim C23$ are advanced cache configurations, which consider the effect of non-blocking cache and prefetcher structures. $C18 \sim C20$, which compose the second group, add Stride Prefetcher to L1 data cache with different prefetching degrees of 1, 2 and 4, respectively. The third group, which consists of configuration $C21 \sim C23$, changes memory access parallelism by changing the number of MSHR entries. C21 changes the cache model into a blocking cache structure by changing the number of MSHR entries to 1. C22 and C23 increase the number of MSHR entries to 2 and 16, respectively. By changing the memory system configuration, it is possible to observe memory performance variation trends and IPC variation trends, and examine which memory metric has a performance trend that best matches IPC in both in single-core and multi-core environments.

The simulations for single core were conducted with 26 benchmarks from the SPEC CPU2006 suite [25]. Five benchmarks in the set were omitted because of compatibility issues with the simulator. For multi-core simulations, an additional three benchmarks were omitted because the simulator cannot support memory space larger than 16 GB. The benchmarks were compiled using GCC 4.3.2 with $-O2$ optimization. The test input sizes provided by the benchmark suite were adopted for all benchmarks. For each benchmark, up to one billion instructions were simulated to collect statistics, or all executed instructions if the benchmark finished before one billion instructions.

4 Evaluation and Evaluation Results

We use correlation to system performance, in terms of IPC, to verify the correctness of APC. The argument is that system performance should correlate with memory performance, and if a memory performance metric does not correlate with system performance then it must miss some important characteristics of memory system. For instance, given that the sale of Ford's cars correlates with the quality of its cars. If someone claims that based on his/her measurement improving the quality of Ford's cars, while all the other factors, such as cost, are unchanged, will hurt Ford's sale; then his/her measurement of quality is probably wrong. Our measurements will focus on concurrency measurement. Additionally, it will be shown that existing memory performance metrics, which are designed to measure the characteristic of memory hierarchy, lack of the ability to catch the concurrency complexity of modern memory systems.

Based on the above simulation configurations, the M5 simulator is used to collect different memory metrics. Each memory metric is then correlated against the IPC from two approaches. First, based on one configuration, each application's memory metric is correlated with its IPC. Second, we focus on one application, while changing memory configurations, the variation similarity between each memory metric and IPC is observed. The first approach tests the correctness of each memory performance metric. The second method tests the sensitivity of each memory performance metric. The combination of these two provides a solid foundation to determine the appropriateness of a metric. The results show that APC always correlates with IPC, and has the highest correlation value as well, while others are not. It is a clear winner.

4.1 Proximity of APC and IPC in Different Applications

Different memory metrics are compared. The memory metrics compared include, Access Per Cycle (APC), Hit Rate (HR, the counterpart of Miss rate), Hits Per 1K Instruction (HP1K, the counterpart of Misses per 1K instructions), Average Miss Penalty (AMP), and Average Memory Access Time (AMAT) [26]. For APC, HR, and HP1K, there should be a positive relation with IPC; for AMP and AMAT, there should be a negative relation with IPC. All metrics are compared at the L1 level.

To show the proximity of different memory metrics to IPC, SPEC CPU2006 is run for all configurations ($C1 \sim C23$) in single core mode with different L1 cache, L2 cache, main memory, prefetcher, and MSHR configuration parameters.

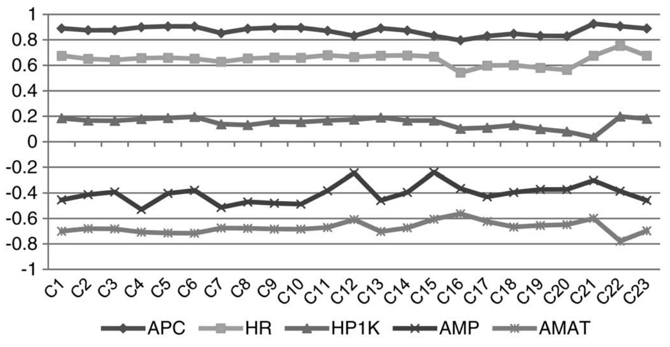


Fig. 2. Correlation coefficient of different memory metrics under different configurations.

The correlation coefficient for each memory metric against IPC is calculated and shown in Fig. 2. From Fig. 2 it can be observed that APC has the strongest correlation with IPC, whose average CC value is 0.871. This strong correlation between APC and IPC reflects the fact that the final system performance largely depends on the performance of the memory hierarchy.

Among other metrics, AMAT is the best with an average CC value of -0.670. This is most likely because it considers both hit latency and miss penalty. Compared with AMAT, APC improves the correlation value by 30.0%. Also, it is interesting to note that the simple metric HR has approximately the same correlation value as AMAT. This is likely due to the performance gap between CPU and memory growing larger and larger, say $200 \sim 400$ CPU cycles; therefore, AMAT is dominated by miss penalty and miss penalty is largely determined from the miss rate. Fig. 2 also shows that using HP1K as a form of hit rate to predict overall performance is not a smart choice as it has the smallest correlation value.

Please notice that a metric with a low average CC value means the metric may have missed certain characteristics of the memory system. In Section 4.2, it is shown that existing metrics have largely omitted the concurrency complexity of modern memory systems.

4.1.1 *Why Memory Dominates Overall Performance*

After examining all kinds of instructions issued by the processor when running SPEC CPU2006 on the M5 simulator, the proportion of each type of instruction is collected in Fig. 3. For CPU2006 applications, integer ALU has the largest proportion, which accounts for 55.3% of instructions on average. But integer ALU execution latency is fixed and small, only one clock cycle for each ALU operation. Load and store instructions occupy 34.0% of all instructions, and are second largest in percentage. With a two-cycle hit latency to L1 cache (the same latency as the execution of Float Add, Float Compare, and Float Convert instructions, the average proportion of these three kinds of instruction is only 5.93%), 12-cycle of hit latency for L2 cache (the same latency with execution of Float Divide instruction, the average proportion is only 0.2%), and 200-cycle of main memory access latency, memory accesses are the most time-consuming portion of the whole operations. Therefore, at the L1 level, memory system is a determine factor of system performance, IPC. In other word, when changing the memory configurations, the memory metric variation and IPC variation should be closely correlated to each other. This simple analysis further confirms the correctness of APC in providing the highest average CC value.

4.1.2 *Instruction Cache Influence*

In the Von Neumann architecture, memory access includes two parts, data access and instruction access. No matter whether an instruction is a memory access or not, the instruction must be read into CPU. Also, no matter what advanced technologies that are adopted for instruction execution, if there are not enough instructions fetched into CPU, then these technologies are useless. Thus, instruction access as well as data access should be considered in the measurement of memory systems.

To optimize the accuracy of APC, the L1 instruction cache APC, $\mathrm{APC_I}$ is introduced. $\mathrm{APC_I}$ is defined as the number of L1 instruction cache accesses divide by the number of overall instruction cache access cycles. Let $\mathrm{APC_D}$ be the L1 data cache APC, then we have $\mathrm{APC_{All}}$, which equals $\mathrm{APC_D} \times \mathrm{APC_I}$, is the APC for the overall memory performance including both data and instruction access. The formula $\mathrm{APC_D} \times \mathrm{APC_I}$ is derived by the conditional dependence. Data access requests are triggered by instructions. The correlation coefficient of $\mathrm{APC_{All}}$ under the default configuration (C1 in Table 4) improves by 2.18%. For all configurations ($\mathrm{C1} \sim \mathrm{C23}$) the average CC value of $\mathrm{APC_{All}}$ is 0.897, and is an improvement of 2.91% on average compared to $\mathrm{APC_D}$ alone. Such a small improvement is observed due to most applications having nearly perfect instruction accesses. Only a very small number of applications, such as gcc, GemsFDTD, and h264ref, have high instruction miss rates. For these applications, $\mathrm{APC_{All}}$ can add more accurate adjustments. The normalized $\mathrm{APC_I}$, $\mathrm{APC_D}$, $\mathrm{APC_{All}}$ and IPC of the default configuration (C1) are shown in Fig. 4. The normalization is based on the magnitude of each metric.

From Fig. 4, each configuration has almost the same variation trends for $\mathrm{APC_{All}}$ and IPC. Fig. 4 confirms that overall application performance is largely determined by the memory performance as is expressed by the APC metric.

4.2 Proximity of APC and IPC by Different Configurations

In this section, we focus on each application, to observe the impact of memory configuration on performance and correlation. Three groups of 23 simulation configurations are run in both single core and multi-core modes. The first group ($\mathrm{C1} \sim \mathrm{C17}$ in Table 4) changes basic cache/memory configurations, the second group ($\mathrm{C18} \sim \mathrm{C20}$) adds a stride prefetcher to L1 caches with different prefetching degrees, and the third group ($\mathrm{C21} \sim \mathrm{C23}$, in Table 4) examines non-blocking cache ability by altering the number of MSHR entries.

Firstly, each group is examined in single core mode step by step. Due to the similarities between multi-core and single core applications, all the three groups in multi-core mode are considered together at once. The simulation results of the first group in single core mode are presented below. In section 4.2.2, we analyze simulations with prefetcher changing. In section 4.2.3, simulations with all groups for single core mode are conducted. Finally, simulations of all groups for multi-core mode are conducted in section 4.2.4. Again, the correlation coefficient is used to evaluate the merits. APC, HR,

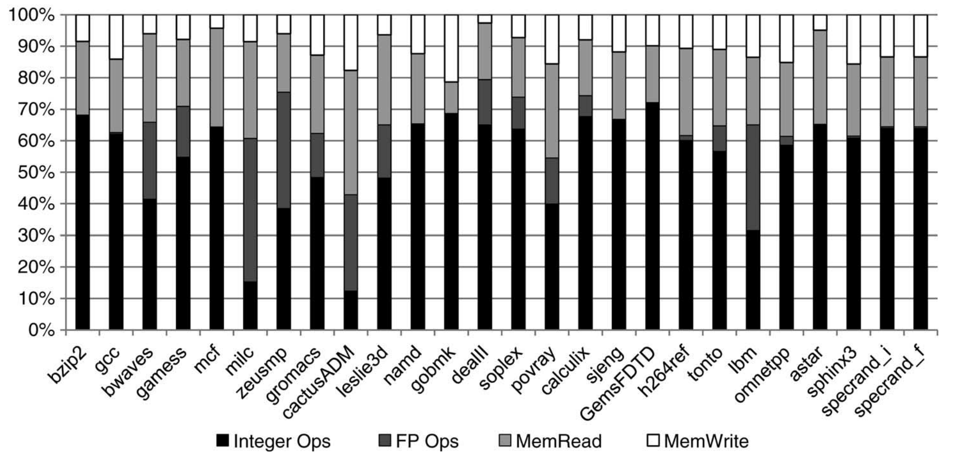


Fig. 3. The proportion of different kinds of instruction.

and HP1K should have a positive correlation coefficient value. AMP and AMAT should have a negative correlation coefficient value.

To clearly show the difference, APC_D and APC_{All}, are compared against the other four memory metrics from Fig. 5 through Fig. 8. Similar to APC which has two measures APC_D, APC_{All}, every conventional memory metric also has two measures which represent data cache performance only and comprehensive cache performance (data cache and instruction cache combined performance). For example, when comparing AMAT, $AMAT_D$ is used to describe data cache AMAT, and $AMAT_{All}$ (equal to $AMAT_D \times AMAT_I$) is used to describe comprehensive cache AMAT.

From Figs. 5 to 8, the first and second bars are APC's correlation values, the remaining bars of each benchmark belong to the corresponding memory metrics' correlation value. In Figs. 5 to 8 it is shown that, when the basic cache/memory configuration ($C1 \sim C17$ in Table 4) of memory hierarchy is changed, while AMAT correlates with IPC well, APC_{All} has the highest correlation coefficient value with IPC at an average value of 0.9632. Of the other memory metrics, $AMAT_{All}$ takes second place with an average value of -0.9393. This shows AMAT is quite a good metric for reflecting conventional memory hierarchy performance variation. However, other metrics show some non-correlation instances. For example, Hit Rate should have positive coefficient values, but for several applications its coefficient values are negative or approximate to zero. One of the reasons for the divergence is that HR does not explicitly include a lower level cache performance factor, such as miss latency information (where as APC and AMAT do consider lower level cache performance). So, when L2 cache size increases, or main memory latency decreases, IPC will increase, even though the Hit Rate has not changed. For instance, the benchmark *sjeng* has the HR correlation value of 0.058. This poor CC value is due to changing L2 and main memory parameters (size, associativity, and latency, experiment configuration ID, $C11 \sim C17$) which have little influence on the hit ratio of data cache and instruction cache and remain almost consistent at 0.8084 and 0.9671, respectively. However, the IPC changes from 0.677 to 0.872. In the benchmark *zeusmp*, when HR increases from 0.879 to 0.883, the IPC unexpectedly drops from 0.664 to 0.645 when simulation configuration changes from C2 to C3. The main reason for this mismatch is believed to be speculation mechanisms. When the cache associativity increases, not only does miss count decreases, the execution of mis-speculated instructions also decreases; thus, the memory access count will decrease as well. As stated earlier, not all mis-speculated memory accesses are harmful. APC, on the contrary, is highly immune to speculation, because when counting access cycles, the overlapping mode is used, and mis-speculated memory accesses are overlapped with correctly speculated memory accesses. In contrast, HR and HR1K which miss information on lower level cache structure, AMP only considers lower level cache miss accesses. When L1 data/instruction cache size increases or associativity increases, the number of miss access decreases, but the miss

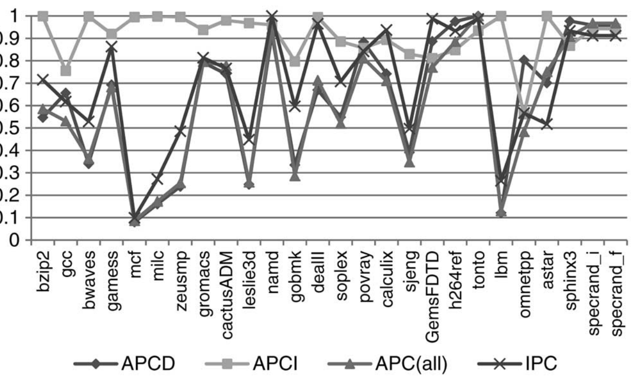


Fig. 4. Normalized IPC, APC_I, APC_D, and APC_{All}.

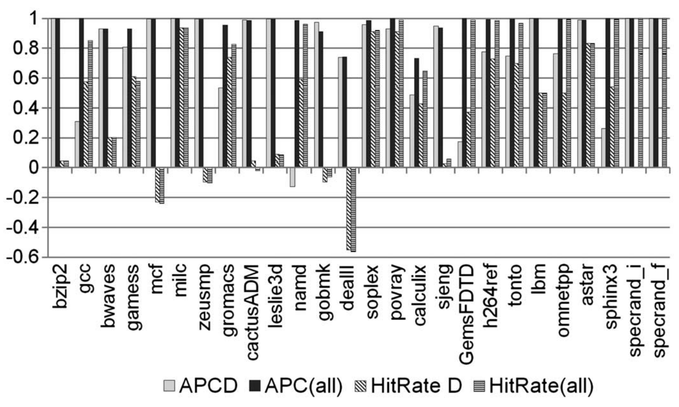


Fig. 5. Comparisons between APC and Hit Rate Correlation Coefficient.

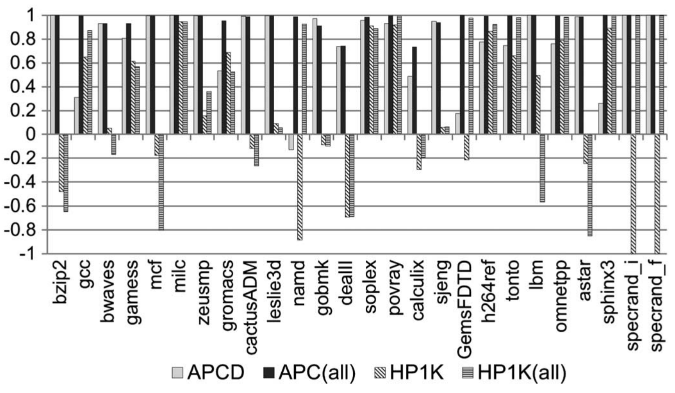


Fig. 6. Comparison between APC and Hit Rate per 1K instruction Correlation Coefficient.

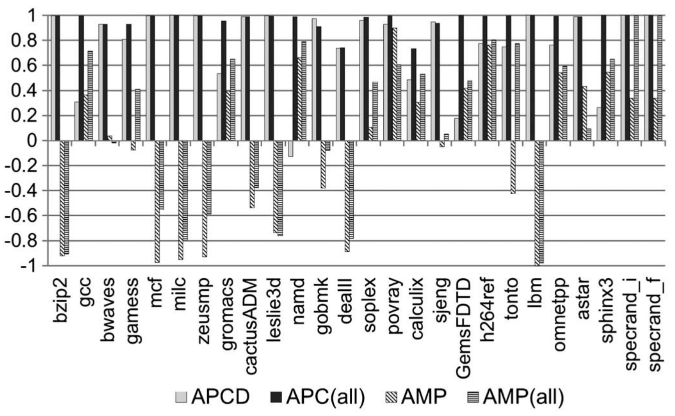


Fig. 7. Comparisons between APC and AMP Correlation Coefficient.

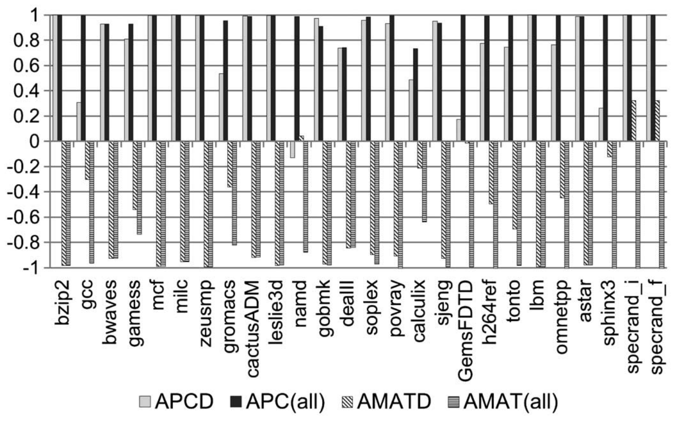


Fig. 8. Comparisons between APC and AMAT Correlation Coefficient.

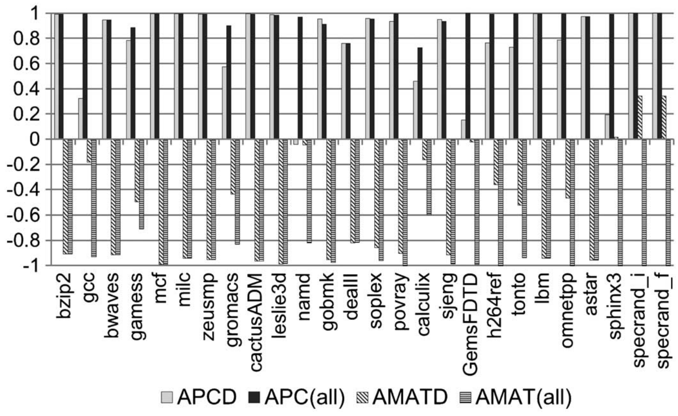


Fig. 9. Comparison between APC and AMAT Correlation Coefficient with prefetching degree changes.

latency for each miss may not change. In summary, component-based memory evaluation metrics, such as HR and AMP, could mislead designers even in a simple component-based system.

4.2.1 Analysis of Low Correlation with IPC

The lowest APC correlation value is 0.733. It is a good value in comparison to other metrics for that particular benchmark. But, let's take the opportunity to give a closer investigation in order to better understand the limitations of APC. There are only two applications having APC correlation values less than 0.8. One is *dealII* with value of 0.742; the other is *calculix* with value of 0.733. The reasons for the low correlations are: firstly, these two applications have very limited memory operations among all kinds of instructions, e.g., *dealII* has the smallest proportion (20.5%) of memory access among all SPEC applications; *calculix* has the 3rd smallest proportion (25.7%) of memory access among all SPEC applications; secondly, both have a high proportion of floating point calculations, which make these two applications' performance much more bounded by computation, not memory access. Another interesting observation is that the benchmark application *zeusmp* has the 2nd smallest proportion (24.6%) of memory access and the 2nd largest proportion (36.9%) of floating point calculations, which seems more like a computing intensive application, and should have an even smaller APC value than *calculix*. However, the APC value of *zeusmp* is 0.995, a considerably high and dominating correlation with IPC. The reason for this contradiction is that even though *zeusmp* has such little proportion of memory accesses, these memory accesses are highly related to main memory, the L2 cache miss rate of *zeusmp* is as high as $60 \sim 80\%$ under different simulation configurations. This phenomenon can be better explained by the value $\mathrm{APC_M}$ of *zeusmp* which has a high correlation value with IPC of (0.932). This means *zeusmp* performance is directly determined by main memory, not L1 and L2 caches. The relationships between the APCs of L1, L2 cache and main memory ($\mathrm{APC_M}$) are discussed in section 5. Also, please note that other memory metrics have about the same CC value with IPC for the benchmark *dealII* and *calculix*.

4.2.2 Adding Prefetcher to L1 Caches

To examine the prefetching effect on memory metrics, simulation configurations $\mathrm{C1} \sim \mathrm{C20}$ are combined together to generate a correlation coefficient with IPC. Because APC and AMAT have a strong correlation with IPC according to the results of the basic configuration experiments and other memory metrics have misleading indications, this section only compares APC and AMAT. The simulation results of correlation coefficient are shown in Fig. 9.

By adding a prefetcher to the L1 data cache, APC and AMAT still have similar average correlation coefficients of 0.957 and -0.925, respectively. These two CC values are both slightly lower than their own basic configurations CC values. A possible reason is the prefetcher generates some useless

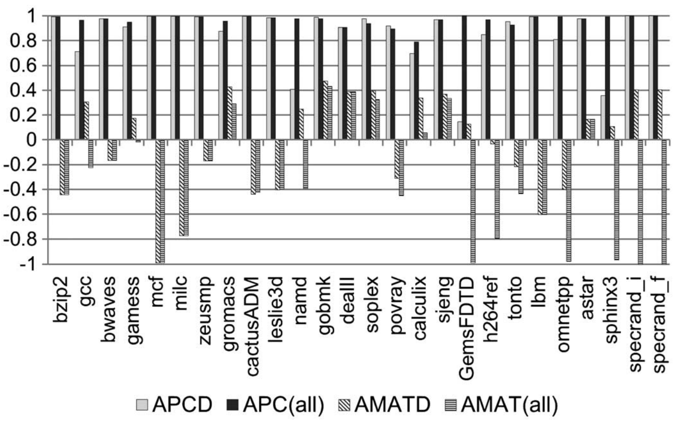


Fig. 10. Comparison between APC and AMAT Correlation Coefficient with MSHR changes.

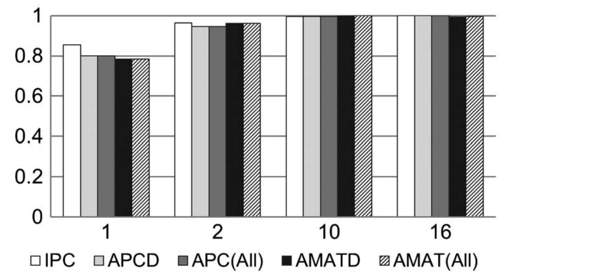


Fig. 11. IPC, APC and AMAT variation when increase MSHR entry.

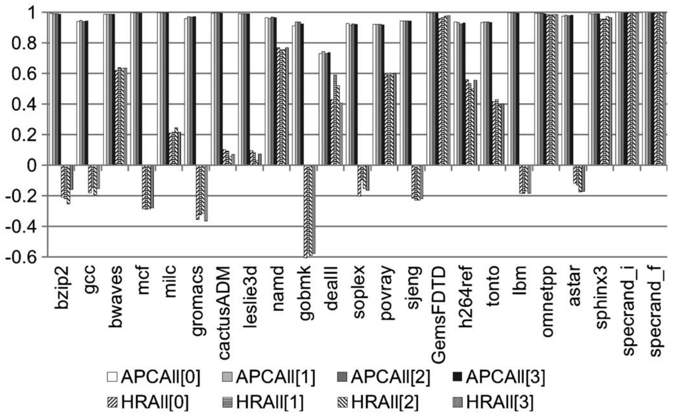


Fig. 12. Comparisons between APC and Hit Rate Correlation Coefficient in multi-core mode.

memory load requests. The useless accesses burden the memory hierarchy, but may increase memory performance and do not harm the overall application performance, since they only affect the cache capacity and memory bandwidth. Therefore, the CC values with a prefetcher are slightly lower than the CC values of the basic configurations. Also please note even though prefetchers can improve concurrency of lower memory accesses, the number of MSHR entries does not change, thus the maximum parallelism of lower cache hardware does not change. In this case, AMAT can still be an accurate memory metric to reflect memory performance.

4.2.3 Changing Access Parallelism in Cache Structure

APC and AMAT have very high correlation with IPC when changing basic cache/memory configurations and adding prefetching to caches. When the number of MSHR entries is altered to examine whether the two memory metrics still have a similar relation with IPC, the concurrency of data access is changed. Fig. 10 shows the correlation coefficient of APC and AMAT for all the 23 configurations listed in Table 4.

Fig. 10 shows that APC still has the same average correlation value of 0.9656. However, AMAT becomes problematic and can not catch the concurrency changes in the memory system. When the number of MSHR entries is increased (memory concurrency increases), the change should, and did, give a positive impact to IPC performance. APC correctly reflects this change and correlates with IPC well. However, AMAT does not understand the concurrency changes at all. About half of the benchmarks go to one direction and the rest go to the other direction. AMAT is a metric designed to catch the hierarchical characteristics of memory system and lacks the ability to catch the concurrency complexity of modern memory systems.

In technical terms when there are not enough MSHR entries, the CPU will be blocked by the memory system. APC can record the CPU blocked cycles, whereas AMAT cannot. Additionally, contention increases with the number of MSHR entries; therefore, the individual data access time, AMAT, increases as well.

Fig. 11 shows the variation normalized IPC, APC and AMAT of bzip2, when the number of MSHR entry is increased. The normalization is based on the magnitude of each metric. From Fig. 11 it can be observed that when the number of MSHR entries increases, the memory access parallelism is improved, thus the overall memory performance is enhanced, and the final application performance IPC is also increased. The variation of IPC and APC correlate well with each other. However, AMAT gives a false indication about memory system performance.

4.2.4 Simulations in Multi-core mode

To further validate whether APC is the most accurate memory metric in describing the overall memory performance in multi-core mode, three groups of 23 configurations are simulated with a 4-core structure. Each core has a private split L1 caches, and only one shared L2 cache. In multi-core mode, each core's memory metrics are correlated with its own IPC value. To clearly show the difference, $\mathrm{APC}_{\mathrm{All}}$ is compared against the other four comprehensive memory metrics from Figs. 12 to 15.

In Figs. 12 to 15, the first four bars are APC's correlation values, the remaining bars of each benchmark belong to the correspondent memory metrics' correlation values. Figs. 12 to 15 show that $\mathrm{APC}_{\mathrm{All}}$ still has the highest correlation coefficient value with IPC at an average value for all applications of 0.9613, and is almost the same value with single core condition. All of the other memory metrics have misleading instances. The inherent shortcomings of existing memory metrics are already discussed in the single core studies. We will not repeat this analysis here.

Through the above simulation and analyses, we have demonstrated that under different applications, different configurations, different advanced cache optimizations, and single-core or multi-core conditions, APC always works effectively. APC directly correlates to overall system performance

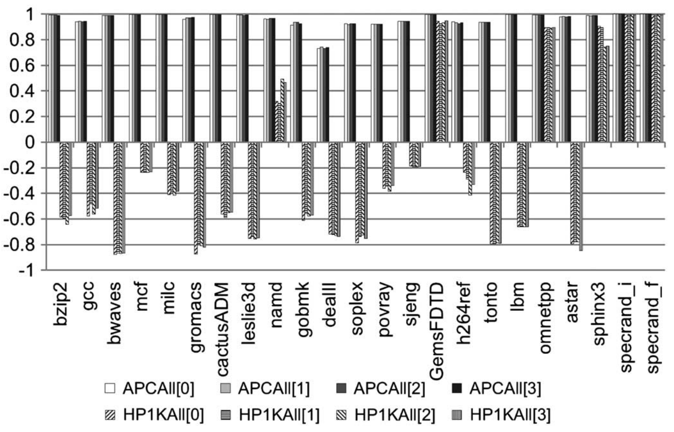


Fig. 13. Comparison between APC and Hit Rate per 1K instruction Correlation Coefficient in multi-core mode.

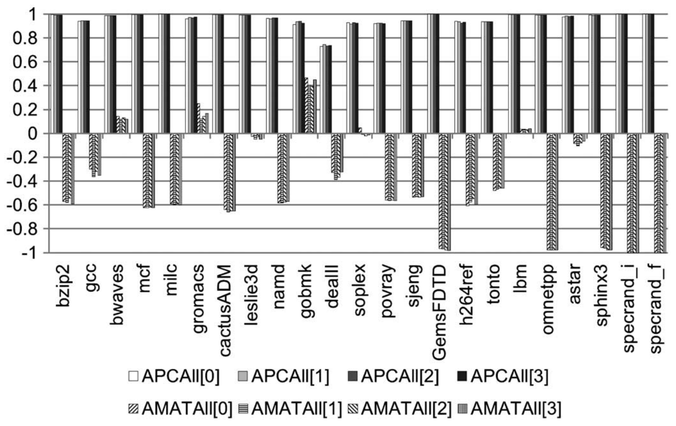


Fig. 15. Comparisons between APC and AMAT Correlation Coefficient in multi-core mode.

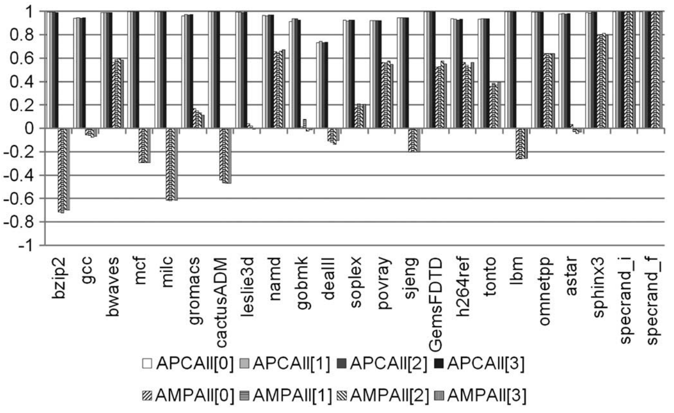


Fig. 14. Comparisons between APC and AMP Correlation Coefficient in multi-core mode.

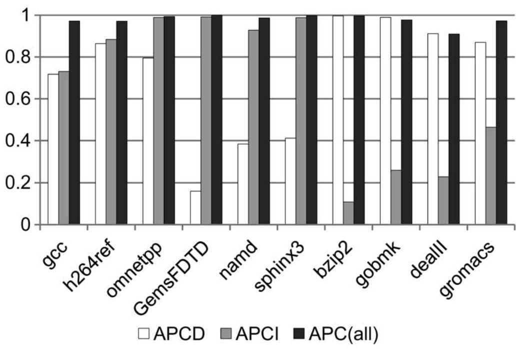


Fig. 16. Comparison between $\mathrm{APC_D}$, $\mathrm{APC_I}$, and $\mathrm{APC_{All}}$ Correlation Coefficient values

in all the testing. In contrast, other existing memory metrics cannot accurately reflect the system performance, and are sometimes misleading about performance. The unique merit of APC is its ability to catch the concurrency characteristics of modern memory systems. It is the only memory metric which can correctly and accurately reflect the overall performance of modern hierarchical memory systems. APC is the most appropriate metric to characterize modern memory system performance.

5 Discussion

Throughout the last section analyses, APC proved to be the best memory performance metric, and is able to most accurately represent the modern multi-level overall performance of memory systems. In this section, some key values of APC which help to understanding application behaviors are discussed. Also, an important quantitative definition of data intensive applications based on the APC concept are given.

Please recall APC can be applied to different levels of a memory hierarchy for performance evaluation. We have used L1 APC in verifying the correctness of APC in Section 4. In this section, we will extend our discussion to APC of other levels of a memory hierarchy in order to fully explore the full potential of APC.

5.1 A Quantitative Analyses of Instruction Cache Effect

Seven out of 26 benchmark applications have $10 \sim 84\%$ difference between their $\mathrm{APC_D}$ and $\mathrm{APC_{All}}$ correlation values. $\mathrm{APC_D}$ only reflects the data cache contribution to the overall application performance. $\mathrm{APC_{All}}$ considers both data and instruction cache effects together. These variations reflect the importance of instruction cache in the application execution. The bigger the difference, the more prominent the effect of the instruction cache on overall performance. This could give us a hit in optimizing memory systems. For example, Fig. 16 shows the $\mathrm{APC_D}$, $\mathrm{APC_I}$, and $\mathrm{APC_{All}}$ value of six benchmarks (from *gcc* to *sphinx3*) which have large variations; whereas the other four benchmarks *bzip2*, *gobmk*, *dealII, and gromacs* have a similar value for $\mathrm{APC_D}$ and $\mathrm{APC_{All}}$. For the latter four applications, the overall performance is directly dominated by data cache performance, where instruction cache with good code stream locality has very little affect (their correlation value of $\mathrm{APC_I}$ are relatively small). For the applications with big differences, data and instruction caches may both be important, e.g., *gcc*, *h264ref* or the instruction cache alone may have a large impact on overall application performance, e.g., *namd*, *GemsFDTD*, *sphinx3*. With this information, a more appropriate choice for memory configuration can be made.

5.2 Bottleneck Inside Memory System

From Figs. 5 through 10, with the dominating correlation between APC and IPC, it is clear that the overall performance

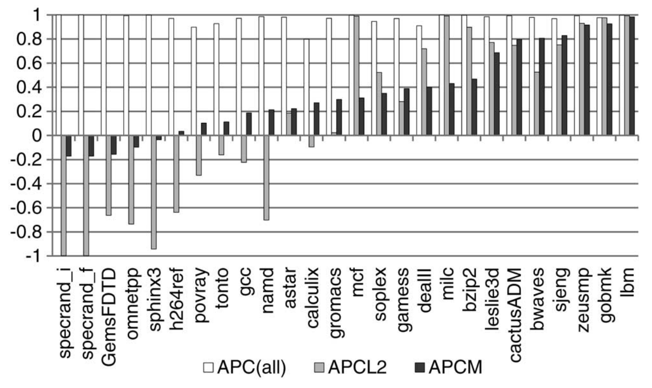


Fig. 17. Comparison between $\mathrm{APC_{All}}$, $\mathrm{APC_{L2}}$, and $\mathrm{APC_M}$ Correlation Coefficient values.

is now determined by memory system performance. Therefore, an important question is at which level of the memory system is the actual performance bottleneck. According to the APC's definition, each level of the memory hierarchy has its own APC values: L1 data and instruction caches have $\mathrm{APC_D}$ and $\mathrm{APC_I}$ respectively; L2 cache has $\mathrm{APC_{L2}}$; and main memory has $\mathrm{APC_M}$. However, the APC of each level not only represents the performance of its memory level, but also includes all the lower levels of the memory hierarchy. For example, the value of $\mathrm{APC_D}$ represents the memory performance of L1 data cache, L2 cache and main memory; and $\mathrm{APC_{L2}}$ represents the memory performance of L2 cache and main memory. Only $\mathrm{APC_M}$, which is the lowest level in the memory hierarchy, represents main memory itself when disk storage is not considered. Therefore, by correlating IPC with APC at each level, one can find the lowest level that has a dominating correlation with IPC and can quantitatively detect the performance bottlenecks inside the memory system. Fig. 17 shows the correlation value of each level of APC for 26 benchmarks. For example, benchmark *specrand_i*, which simply generates a sequence of pseudorandom numbers starting with a known seed, is computation intensive instead of data intensive. The miss rate of L1 cache for specrand_i applications is as low as 0.0018% in our simulation results. Therefore, only L1 cache performance has great influence on IPC for the *specrand_i* application. L2 and other lower level caches in the memory hierarchy do not affect the *specrand_i* application performance, which is correctly reflected by the first bar set of Fig. 17. For benchmark *mcf*, both $\mathrm{APC_{All}}$ and $\mathrm{APC_{L2}}$ have a dominant relation to IPC, but its $\mathrm{APC_M}$ does not. That means the performance of the *mcf* application is determined by its L2 cache performance, and has a good locality. More evidences to support this observation are from Fig. 18. Fig. 18 is the L2 cache miss rates of *mcf* and *lbm.* When increasing L2 cache size from 2 MB to 8 MB with a fixed L1 cache configuration (C13 $\sim$ C15 in Table 4). According to Fig. 18, the L2 cache miss rate of *mcf* suddenly drops from 34.0% to 4.8%. That means, when L2 cache size is equal to or larger than 8 MB, the benchmark *mcf* mainly accesses data in L2 cache. The main memory is not a bottleneck; and the overall performance of *mcf* is mainly determined by L2 cache performance. This observation is again correctly reflected in APC measurement as seen from Fig. 17. For benchmark *lbm*, according to Fig. 17, since all levels of APC have dominating correlation with IPC, the performance of *lbm* is determined by

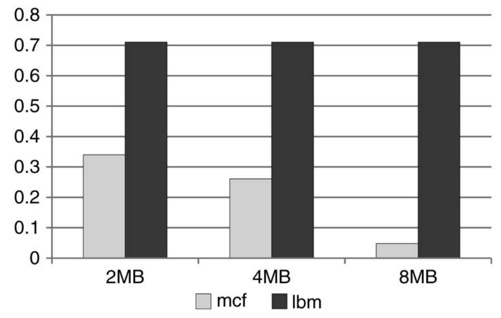


Fig. 18. L2 cache miss rate variations for mcf and lbm.

the performance of the lowest level, namely the main memory level. This result is supported by Fig. 18 which shows that the L2 miss rates of *lbm* are approximately unchanged when increasing L2 cache size. The miss rates are all above 70%.

5.3 A Quantitative Definition of Data Intensive Application

The term "*Data-intensive Applications*" and “Data-intensive Computing” are widely-used terms in describing application or computing where data movement, instead of computing, is the dominate factor. However, there is neither a commonly accepted definition of “data-intensive computing” nor a quantitative measurement of “data intensiveness”. As APC characterizes the overall memory performance, the IPC and APC correlation values provides a quantitative definition of data intensiveness. The idea is simple: if $\mathrm{APC_M}$ dominates IPC performance, then the application is data intensive and the degree of the domination provides a measurement of data intensiveness. The correlation value of $\mathrm{APC_M}$ is used to quantify the degree of data intensive. There are three reasons to use $\mathrm{APC_M}$ instead of $\mathrm{APC_D}$ to measure data intensive. First, due to the "memory-wall" problem, main memory latency becomes a vital performance bottleneck in the memory hierarchy and dramatically degrades CPU speed. Next, data reuse is not counted as part of data-intensiveness unless it has to be read from main memory again. Finally, according to the definition of APC, if $\mathrm{APC_M}$ has a dominate relation with IPC, then $\mathrm{APC_{L2}}$ and $\mathrm{APC_D}$ will have a dominate relation with IPC too. In other words, the cache hit influence is ignored, since we are interested in data movement, not usage, in considering data intensiveness. Also please note that the hardware cost for measuring $\mathrm{APC_M}$ is almost zero.

Fig. 17 is sorted according to $\mathrm{APC_M}$ correlation values in ascending order (the farther to the left, the smaller the value of $\mathrm{APC_M}$). The correlation value of $\mathrm{APC_M}$ is divided into three intervals, that is $(-1, 0.3)$, [0.3, 0.9), and [0.9, 1). Thirteen applications counted from the left side (from *specrand_i* to *gromacs*) fall into the first interval. According to the three APC values used in Fig. 17, it can be concluded that the application performance of these 13 applications are dominated by the L1 cache, not L2 or main memory because the correlation values of $\mathrm{APC_{L2}}$ and $\mathrm{APC_M}$ of these applications are negative or very small. As the correlation value of $\mathrm{APC_M}$ increases, the effect of main memory to the overall application becomes increasingly important. Therefore, some applications' performance in the second interval (from *mcf* to *sjeng*) is dominated by the L2 caches, e.g., *mcf*, *milc*. However, other applications, such as *bzip2*, require both L2 and main memory equally. For the third interval, the applications' performances are determined by main memory performance. This observation motivates us to define an application as data intensive if its correlation coefficient of $\mathrm{APC_M}$ and IPC is equal to or larger than 0.9. Another reason for picking 0.9 as the threshold is according to the mathematical definition of a correlation coefficient, when the correlation value of two variables is equal to or larger than 0.9, then the two variables have a dominant relation. Therefore, here we define that an application as data intensive *if and only if*

$$Data\ Intensive\ Application \equiv coe(APC_M, IPC) \geq 0.9,$$

and the value of the correlation provides a quantitative measurement of the data-intensiveness.

6 Related Work

Miss Rate (MR), Miss Per Kilo Instructions (MPKI), Average Miss Penalty (AMP), and Average Memory Access Time (AMAT) are some commonly used performance metrics in evaluating memory systems [26]. MR is defined as {the number of miss memory accesses} over {the number of total memory accesses}. Similarly, MPKI is defined as {$\mathrm{the\ number\ of\ miss}\ memory\,accesses \times 1000$} over {the number of total committed Instructions}. AMP equals {the summary of single miss latency} over {the number of miss memory accesses}. Finally, $\mathrm{AMAT} = \mathrm{Hit\ time} + \mathrm{MR} \times \mathrm{AMP}$. MR and MPKI only reflect the proportion of the data in or out of the cache; they don't reflect the penalty of the miss access. AMP only catches the penalty of the cache miss access; it doesn't show the hit in performance. Also for AMP, the single miss latency is counted based on single miss access, assuming there is not any other memory access present. AMAT is a comprehensive memory metric, but it is still based on the single data access point of view. It is the average period between start time and finish time of a single memory access and does not consider the memory access overlapping and parallelism. There have been several studies on Memory Level Parallelism (MLP) [27] in recent years. MLP is a common memory metric and is the average number of long-latency main memory outstanding accesses when there is at least one such outstanding access [28]. So

$$MLP = \sum_{t=0}^{n} \frac{MLP(t)}{Total\ Memory\ Access\ Cycles}. \tag{3}$$

$MLP(t)$ is *MLP* value at clock cycle t. The meaning of total memory access cycles in Equation (3) is the same as the APC definition. Sorin's $\mathrm{f_M}$ parameter [29] has a similar definition to the MLP in [27]. Sorin's $\mathrm{f_M}$ considers all main memory accesses, while Chou [28] only considers useful long-latency main memory access. Here, useful means that the data fetched by main memory access is finally used by CPU. In the APC approach, $\mathrm{APC_M}$, which is defined as the total number of off-chip access divided by the number of total main memory access cycles, reflects main memory access performance.

Assuming each off-chip memory access has a constant latency, say m cycles, then each memory access will count m times when calculating different $MLP(t)$, so we have $APC_M = MLP/m$. That means $\mathrm{APC_M}$ is directly proportional to MLP. Any indication from MLP on CPU micro-architecture could also be obtained from $\mathrm{APC_M}$. A known limitation of MLP is it only focuses on off-chip memory access based on the epoch memory access mode for some commercial or database workloads [27]. These kinds of applications usually have much more L1 and L2 cache misses, and their overall performances are heavily determined by main memory access. However some traditional CPU intensive applications, which only considering main memory access is far from enough. In addition, advanced process technology, such as EDRAM used in IBM POWER7, dramatically increase on-chip cache sizes up to 32 MB [30]. Using 3D integration technologies, one could implement a multi-layer CPU with one layer full of on-chip cache [31]. In contrast, APC not only can be used to analyze commercial applications, but also to analyze traditional scientific applications, so it has a much wider applicability. APC is a rethinking of memory performance from a data-centric view. It is much simple in measurement and can be applied to in-depth memory performance analysis. Nevertheless, MLP is a new metric drawing much attention for data-intensive applications. Being a superset of MLP demonstrates the preeminence of APC from another angle. In fact, it makes APC and MLP mutually supporting to each other.

7 Conclusion and Future Work

In this paper we propose a new memory metric APC, give its measurement methodology, and demonstrate that APC has a unique ability to catch the complexity, especially the concurrency complexity, of modern memory systems. APC is defined as memory Access Per Cycle. Here the cycle is the parallel data access cycle and is measured in the overlapping mode. For this reason, APC also can be accurately called APMAC (access per memory active cycle). Where overlapping is defined where concurrent memory accesses only increase the cycle count by one. An APC measurement logic (AML) is designed to measure APC. The application of APC and the APC of different levels of a memory hierarchy are discussed. Mathematical correlation is used to demonstrate the correctness and effectiveness of APC. Intensive simulations are conducted with a modern computer system simulator, M5, to verify the potential of APC and compare it with existing memory performance metrics. As a side result, we have shown that the conventional memory metric AMAT is a good metric to catch the variation of simple configuration changes of hierarchical memory systems, but inadequate to capture the concurrency complexity of modern memory systems. Other existing memory metrics, such as miss ratio, can be misleading even for a simple component configuration change in modern memory systems. Simulation and statistical results show that APC is a significantly more appropriate memory metric than other existing memory metrics for overall performance of a memory system. APC can be applied to different levels of a memory hierarchy as well. With the ability to measure performance at each level of a memory hierarchy, it can be clearly understood which component in a memory system should be enhanced first, reducing the cache miss ratio or increasing the bus bandwidth, for instance. It can better understand the matching of processor and memory and the matching of application and memory. In addition, the correlation of APC of memory and IPC provides a quantitative measurement of the data-intensiveness of an application. In this way, it provides a measurement of data-centric computing that could have profound impact in future data-centric algorithm design and system development. APC is shown to be a superset of MLP [28], a newly proposed metric aimed for main memory access and data-intensive applications. The recent interest on MLP further confirms the practical value of APC. In the future, APC will be extended to IO and network systems. Also based on the current available CPU performance counter, the validation of $\mathrm{APC_M}$ on a real machine will be evaluated. Based on the insights of these studies, new memory microarchitecture ideas for further improvement in APC will be investigated.

Acknowledgment

The authors are grateful to anonymous reviewers for their valuable comments and suggestions. This research was supported in part by National Science Foundation under NSF grant CCF-0621435 and CCF-0937877.

References

[1] X.-H. Sun and L. Ni, "Another view on parallel speedup," in *Proc. IEEE Supercomput.*, Nov. 1990, pp. 324–333.

[2] W. Wulf and S. McKee, "Hitting the wall: Implications of the obvious," in *Proc ACM SIGArch Comput. Archit. News*, Mar. 1995, pp. 20–24.

[3] M. E. Thomadakis, "The architecture of the Nehalem processor and Nehalem-EP SMP platforms," A research report of Texas A&M University, 2010 [Online]. Available: http://sc.tamu.edu/systems/eos/nehalem.pdf

[4] R. Fiedler, "Blue waters architecture," *Great lakes consortium for Petascale Computation*, Oct. 2010.

[5] D. M. Tullsen, S. J. Eggers, J. S. Emer *et al.*, "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor," in *Proc. 23rd Annu. Int. Symp. Comput. Archit.*, May 1996, pp. 191–202.

[6] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded SPARC processor," *IEEE Micro*, Mar./Apr. 2005, pp. 21–29.

[7] D. Kroft, " Lockup-free instruction fetch/prefetch cache organization," in *Proc. ISCA'81*, 1981, pp. 81–87.

[8] A. Agarwal, K. Roy, and T. N. Vijaykumar, "Exploring high bandwidth pipelined cache architecture for scaled technology," in *Proc. DATE*, 2003, pp. 10778–10783.

[9] J. A. Rivers, G. S. Tyson, E. S. Davidson *et al.*, "On high-bandwidth data cache design for multi-issue processors," in *Proc. 30th Int. Symp. Microarchit. (Micro-30)*, Dec. 1–3, 1997, pp. 46–56.

[10] S. Byna, Y. Chen, and X.-H. Sun, "Taxonomy of data prefetching for multicore processors," *J. Comput. Sci. Technol.*, vol. 24, no. 3, pp. 405–417, May 2009.

[11] B. Sinharoy, R. Kalla, W. J. Starke *et al.*, "IBM POWER7 multicore server processor," *IBM J. Res. Develop.*, vol. 55, no. 3, pp. 1:1–1:29, May/Jun. 2011.

[12] ARM Processor Introduction. (2011, Dec.) *Cortex-A15 MPCore Processor* [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a15.php.

[13] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: An alternative to very large instruction windows for out-of-order processors," in *Proc. 9th Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2003, pp. 129–140.

[14] Intel. *Inside Intel Core Microarchitecture and Smart Memory Access*. Intel, White Paper, 2006.

[15] Intel. (2011, Apr.) *Introduction to Microarchitectural Optimization for Itanium® 2 Processors*. Reference Manual [Online]. Available: http://developer.intel.com.

[16] S. Andersson, R. Bell, J. Hague *et al.* (2011, Apr.) *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide* [Online]. Available: http://www.redbooks.ibm.com.

[17] K. K. Chan, C. C. Hay, J. R. Keller, G. P. Kurpanek, F. X. Schumacher, and J. Zheng, "Design of the HP PA 7200CPU," *Hewlett-Packard J. Tech. Inf. Laboratories Hewlett-Packard Company*, vol. 47, no. 1, pp. 25–33, 1996.

[18] S. Fields, J. M. Tendler, and S. Dodson, "Power4 system microarchitecture," *IBM*, Tech. White Paper, 2001.

[19] T. F. Chen and J. L. Baer, "Effective hardware-based data prefetching for high performance processors," *IEEE Trans. Comput.*, vol. 44, no. 5, pp. 609–623, 1995.

[20] First the Tick, "Now the tock: Next generation Intel® microarchitecture (Nehalem)," *Intel*, White Paper, 2008.

[21] J. Owen and M. Steinman, "Northbridge architecture of AMD's Griffin microprocessor family," *IEEE Micro*, vol. 28, no. 2, pp. 10–18, 2008.

[22] A. Aggarwal, "Reducing latencies of pipelined cache accesses through set prediction," in *Proc. 19th Annu. Int. Conf. Supercomput. (ICS'05)*, Jun. 2005, pp. 2–9.

[23] J. Higgins. (2011, Apr.) "The radical statistician: A practical guide to unleashing the power of applied statistics in the real world," *Biddle Consulting Group* [Online]. Available: http://www.biddle.com/documents/bcg_comp_chapter2.pdf.

[24] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul./Aug. 2006.

[25] C. D. Spradling, "SPEC CPU2006 benchmark tools," in *Proc. ACM SIGARCH Comput. Archit. News*, 2007, pp. 130–134.

[26] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed. San Mateo, CA, USA: Morgan Kaufmann, Sep. 2006.

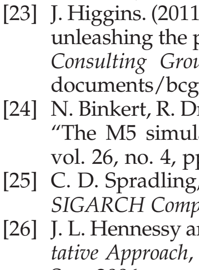
[27] A. Glew, "MLP yes! ILP no!" in *Proc. ASPLOS Wild and Crazy Idea a workshop Session'98*, Oct. 1998.

[28] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture optimizations for exploiting memory-level parallelism," in *Proc. 31st Int. Symp. Comput. Archit. (ISCA'04)*, Jun. 2004, pp. 76–89.

[29] D. Sorin Vijay, S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood, "Analytic evaluation of shared-memory systems with ILP processors," in *Proc. 25th Int. Symp. Comput. Archit.*, 1998, pp. 166–180.

[30] W. Starke, "POWER7: IBM's next generation balanced POWER server chip," in *Proc. Hotchip*, 2009, p. 21.

[31] Y.-F. Tsai, F. Wang, and Y. Xie, "Design space exploration for 3-D cache," in *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2008, pp. 444–455.



Dawei Wang received the PhD degree in computer science at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2009. He is now a postdoctoral researcher at the Scalable Computing Software laboratory in Illinois Institute of Technology, Chicago. His research interests include computer architecture, large scale interconnection networks, and architectural simulation & emulation. He is currently working on multi-core memory scheduling and prefetching area to improve multi-core memory access bandwidth utilization and minimize access latency.

Xian-He Sun is a distinguished professor of computer science and the chairman of the Department of Computer Science, the Illinois Institute of Technology (IIT), Chicago. He is the director of the Scalable Computing Software laboratory at IIT, and is a guest faculty in the Mathematics and Computer Science Division at the Argonne National Laboratory, Lemont. Before joining IIT, he worked at DoE Ames National Laboratory, at ICASE, NASA Langley Research Center, and at Louisiana State University, Baton Rouge. He is an IEEE fellow. His research interests include parallel and distributed processing, memory and I/O systems, software systems, and performance evaluation and optimization.