Preprint: to appear in ACM SIGMETRICS Performance Evaluation Review, Volume 40 Issue 2, 2012

APC: A Performance Metric of Memory Systems

Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA 60616
sun@iit.edu

Dawei Wang
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA 60616
dwang31@iit.edu

ABSTRACT
Due to the infamous "memory wall" problem and a drastic increase in the number of data intensive applications, memory rather than processor has become the leading performance bottleneck of 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, average miss latency, average memory access time, etc., are designed to measure a given memory performance parameter, and do not reflect the overall performance of a memory system. On the other hand, widely used system measurement metrics, such as IPC and Flops are designed to measure CPU performance, and do not directly reflect memory performance. In this paper, we proposed a novel memory metric, Access Per Cycle (APC), to measure overall memory performance with consideration of 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. The concept of APC is introduced; a constructive investigation counting the number of data accesses and access cycles at differing levels of the memory hierarchy is conducted; finally some important usages of APC are presented. Simulation results show that APC is significantly more appropriate than existing memory metrics in evaluating modern memory systems.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: *design studies, measurement techniques, performance attributes.*

General Terms
Measurement, Performance

Keywords
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 as dramatic a gain as that of processors; leaving memory performance lagging far behind CPU performance. This enlarging performance gap between processor and memory is referred to as the "memory wall" [1] [2]. The "memory wall" problem exists not only in main memory but also in the on-die caches. For example, in the Intel Nehalem architecture CPU, each L1 data cache has a 4-cycle hit latency; and each L2 cache has a 10-cycle hit latency [3]. Additionally, the IBM Power6 has a 4-cycle L1 cache hit latency, and an L2 cache hit latency of 24 cycles [4]. These large performance gaps between the processor and memory hierarchy make the memory system the dominant performance factor in high-end computing. Intensive research has recently been conducted to improve the performance of memory systems. However, understanding the performance of modern hierarchical memory systems remains elusive for researchers and practitioners.

Because hierarchical memory systems are a bottleneck of performance, measuring and evaluating memory systems has become an important issue facing the high performance computing community. 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, and measure the overall computing performance. They are comprehensive and affected by a multitude of factors such as instruction sets, CPU micro-architecture, memory hierarchy, compiler technologies, etc. and as such are not appropriate measurements of 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), only measure a particular component of a memory system. They are useful in optimization and evaluation of a given component, but cannot accurately characterize the performance of the memory system as a whole. In general, a component improvement does not necessarily lead to an improvement in terms of overall computing performance or overall memory performance. For instance, in some cases when miss rate decreases, IPC will decrease instead of increasing (Please see section 4.2). When the non-blocking cache optimization method is used, the AMAT metric shows a misleading effect on IPC (Please see section 4.3).

There are several reasons that traditional memory performance metrics cannot directly 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 executions overlap 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] could tolerate even longer misses through main memory by executing another thread. Speculation mechanisms are used to overcome control

dependencies, which help to avoid CPU stalls. Speculation could 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, but the effect of miss speculations is hard to predict. A incorrect prediction is not always 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 benefits the memory performance. All of these scenarios make component-based measurements unsuitable for the measurement of the overall performance of a modern memory system.

Additionally, modern memory systems use a large number of advanced caching technologies to decrease average access latency. Some widely used cache optimization methods, such as non-blocking cache [7], pipelined cache [8], and multibanked cache [9], allow cache accesses to overlap with each other. These technologies make the relationship between memory access and processor performance even more complicated, since the processor could continue accessing memory under multiple cache misses. Thus, the influence of the improvement of one particular component in a memory system becomes increasingly tangled and elusive. Evaluating memory systems from a single memory access or on a single component does not reflect the complexity of modern memory systems. Advanced memory technologies make the behavior of memory access similar to instruction dispatch, because they both execute several operations concurrently. 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.

A new metric for overall memory system performance, which is separate from CPU performance but 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).

The introduction of APC is four-fold. First, the definition of APC is introduced; next methods of determining the number of accesses and access cycles are explored; then methods of determining measurements at different memory hierarchy levels are discussed; finally a series of simulations are conducted to confirm that APC is significantly more appropriate than existing memory performance metrics. The statistical variable correlation coefficient is used to demonstrate that APC has a 0.97 correlation coefficient value with the overall computing performance in terms of IPC, whereas conventional metrics only have a 0.67 correlation in the best scenarios.

The rest of this paper is organized as follows. Section 2 defines APC, and describes how it is measured. Section 3 introduces the experiment methodology and setup. Section 4 compares APC with other conventional memory metrics by way of the correlation coefficient. Section 5 discusses the application of APC and provides a quantitative definition of data intensive from a memory system point of view. Section 6 presents related works, and Section 7 concludes this study.

2. INTRODUCTION TO APC

In this section, the formal definition of APC is provided and investigation is carried out on the measurement of memory access cycles of APC in advanced non-blocking memory structures.

2.1 APC Definition

IPC reflects overall performance in terms of the number of executed instructions per cycle. After more than thirty years development, the ILP and memory optimization technologies have many key features in common. Table 1 lists a comparison of some common technologies adopted by them.

Table 1. ILP and memory optimization comparison

ILP Tech.	Memory Tech.	Key feature in common
Pipelined stage in CPU data path	Pipelined Cache	Micro-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[10]	Data Prefetching	Pattern recognizing and only successful with certain possibility

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. Generally speaking, APC is measured as the number of memory accesses per cycle. Also, the APC metric can be used to evaluate the memory performance at each level of a memory hierarchy. More specifically, APC is the number of memory accesses requested at a certain memory level (ie: L1, L2, L3, Main Memory) divided by the number of memory access cycles at that level. 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,

$$APC = \frac{M}{T} \qquad (1)$$

The definition is simple enough. However, because modern memory systems adopt many advanced optimizations, such as pipelined cache, non-blocking cache, and multibanked cache, etc. several outstanding memory accesses may co-exist in the memory system at the same time. Counting cycle T is not as simple as it may seem. 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 are counted, including all successful or non-successful speculations and including all concurrent accesses. For example, if two L1 cache load requests exist at the same time, M will increase by two.

According to the APC definition, each memory level has its own APC value. This paper focuses on APC for L1 cache and includes discussions of the 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 should be sufficient in illustrating the APC concept and demonstrating its effectiveness. To avoid confusion,

the term APC_D is used for L1 data cache, APC_I is used for L1 instruction cache, which is the number of L1 data or instruction cache accesses divided by the number of overall cache access cycles of their own. Main memory APC is denoted as 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 [11], Itanium [12], and IBM POWER3 [13], employ non-blocking cache 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), address, return register, etc. When the MSHR table is full, the cache disallows additional cache accesses, the CPU or up-level memory access requests are blocked due to the lack of MSHR entry. When the MSHR is attached to LLC (Last Level Cache) and full, designates there are no more outstanding main memory accesses.

Calculating an accurate number of overall memory access cycles in a non-blocking cache 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 overlapped. Ideally, only one cycle should be counted in the total 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 under the overlapping mode. In this study, we propose an APC measurement logic, as illustrated in Fig. 1, that supports the overlapping mode to test the potential of APC.

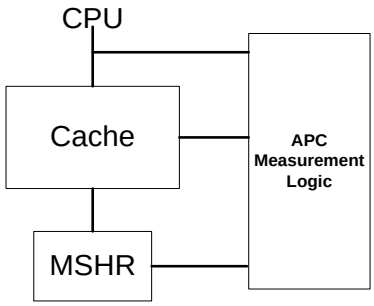

Figure 1. APC Measurement Structure

To avoid overlapping memory accesses from being counted multiple times in an access cycle, the APC Measurement Logic (AML) simultaneously detects memory access activities from CPU, cache and MSHR. If at least one memory access activity exists in the CPU/Cache interface 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. Additionally, the AML will count the number of CPU load/store accesses by detecting CPU/Cache bus requests. If there are several memory requests at the same time from the bus, all are counted. Through this counting, the number of total accesses *M* and the total memory access cycle *T* can be obtained, and the APC can be calculated for this level of cache. The pseudo code of memory access counting logic for 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, multiple buses between upper level caches and itself may be detected in implementation simultaneously.

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 bus is active)//Having a request or returning data Mem_Cycle ++;
Else Mem_Cycle does not change

In general, memory access cycles consist of three different timing stages, namely CPU/Cache Transferring Cycle, Cache Cycle, and MSHR Cycle. The three parts are mutually exclusive. The MSHR Cycle is the time spent in MSHR, and reflects the next level of the memory hierarchy ability to supply data. Only when the MSHR is not working does the Cache Cycle start to count. The Cache Cycle is the time spent in the cache determining if a hit or miss has occurred. It reflects the local cache's ability to provide data. It heavily depends on local cache organizations, such as cache capacity, associativity, etc. If both MSHR and cache are not working, then CPU/Cache Transferring Cycle starts to count. The CPU/Cache Transferring Cycle is the time consumed by CPU and Cache in transferring requests or data.

The AML consists of two registers (e.g. 64-bit register is sufficient for most computer systems) and some detecting logic. One register counts the total number of memory cycles; the other counts memory accesses. Please notice that the memory accesses count is already provided by existing CPU performance counters. However, some detecting logic for counting the number of memory cycles must be added. The detecting logic includes CPU/Cache 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, and only one status bit is required. Modern caches adopt pipelined structures for data access. A typical pipeline structure consists of Decode, Compare Tags, Rd/Wr Data, and Drive Out stages [14]. 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. Adding all the detecting logic and the two registers, the total cost of AML is less than 1K bits, which is negligible compared with modern transistor sizes of a CPU.

The AML for L1 cache can be extended to main memory with little variation. When measuring main memory APC_M, main memory access count and LLC MSHR Cycle are needed to be detected. The former can be found in CPU performance counters; and the latter, only need 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 APC_M. Therefore,

$$APC_M = \frac{\text{Main Memory access count}}{\text{LLC MSHR cycles}} \qquad (2)$$

With the modification to current cache structures, when running a real benchmark on the computer, the APCs of all cache levels and the IPC can be obtained.

3. EXPERIMENTS METHODOLOGY AND SETUP

In this section, the mathematical concept of correlation coefficient is introduced. With correlation coefficient as the proximity measurement, a series of experiments are executed with different cache and main memory parameters.

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. The mathematical statistic variable *correlation coefficient* (CC) is used in this study to determine which memory metric most closely trends with the IPC variation. Correlation coefficient describes the proximity between two variables' changing trends from a statistics viewpoint. It measures how well two variables match with each other.

The correlation coefficient is a number between -1 and 1. The higher the CC absolute value is, the closer the relation between the two variables would be. 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. 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 the two variables. That means if one increases, the other decreases [15].

3.2 Experiment Setup

A detailed out-of-order CPU model in the M5 simulator [16] 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.

Table 3. Simulation Configuration Parameters

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
L2 cache	2MB, 8-way, 64B line, 12-cycle hit latency, 20 MSHR Entry
DRAM latency/Width	200-cycle access latency/64 bits

There are some other experiment configurations based on the default configuration. Each of them only changes one or two parameter/s of the simulation. The detailed experiment configurations are shown in Table 4.

Table 4. A Serials 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, MSHR 1; L2: 2MB,8way; Mem100ns	MSHR Entry
C19	L1:32KB,2way, MSHR 2; L2: 2MB,8way; Mem100ns	MSHR Entry
C20	L1:32KB,2way, MSHR 16; L2: 2MB,8way; Mem100ns	MSHR Entry

The configuration C1~C17 are the basic cache/memory configurations, which only change the cache size, associativity, or memory latency. These basic configurations have 10 MSHR entries each. The C18 changes the cache model into a blocking cache structure. C19 and C20 increase memory level parallelism by increasing the MSHR entry to 2 and 16 respectively. By changing memory system configurations, 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 that of IPC's.

The simulations were conducted with 26 benchmarks from SPEC CPU2006 suite [17]. Five benchmarks in the set were omitted because of compatibility issues with the simulator. 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, one billion instructions were simulated to collect statistics, or all executed instructions if the benchmark finished before one billion instructions.

4. EVALUATION RESULTS

Based on the above configurations, the M5 simulator was used to collect the measurements of different memory metrics. Each memory metric is then correlated against the IPC from two approaches. First, based on one configuration, we correlate each application's memory metric 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 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 has the highest correlation value with IPC in both cases. Therefore, it is a clear winner.

4.1 Proximity for different applications

IPC with different memory metrics were compared. The memory metrics compared include, Access per Cycle (APC), Hit Rate (HR, the counterpart of Miss rate), Hits per 1K instruction (HPKI, the counterpart of Misses per 1K instructions), average miss

penalty (AMP), and Average Memory Access Time (AMAT) [18]. For APC, HR, and HPKI, there should be a positive relation with IPC; for AMP and AMAT, there should be a negative relation with IPC. To show the proximity of different memory metrics with IPC, Spec CPU2006 was run for all configurations (C1~C20) with different L1, L2 cache, main memory, and MSHR configuring parameters. The correlation coefficient for each memory metric against IPC was calculated and shown in Fig. 2. From Fig. 2 it can be observed that APC has the strongest relation with IPC, whose average CC value is 0.876. This strong relation between APC and IPC also reflects the fact that the final system performance largely depends on the performance of the memory hierarchy.

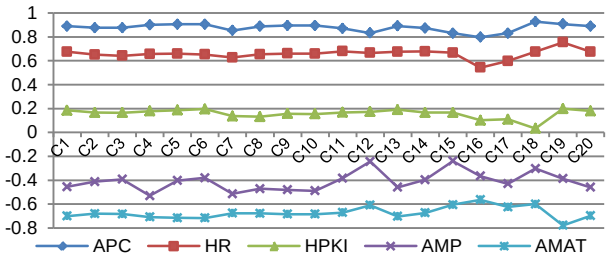

Figure 2. Correlation coefficient of different memory metrics under different configurations

Among other metrics, AMAT is the best one with average CC value of -0.672, since it considers both hit latency and miss penalty. Compared with AMAT, APC improves correlation value by 30.4%. Also, it is interesting to notice that the simple metric HR has approximately the same correlation value as AMAT. This is probably due to the performance gap between CPU and memory growing larger and larger, say 200~400 CPU cycles; therefore, AMAT is dominated by miss penalty, and miss penalty is largely determined by miss rate. Fig. 2 also shows that using HPKI as a form of hit rate to predict overall performance is not a smart choice. It has the smallest correlation value.

4.1.1 Instruction Cache Affection

According to the Von Neumann's architecture, memory access includes two parts, data access and instruction access. Whether the instruction is memory access or not, it should be read into CPU. Also, no matter what advanced technologies are adopted in instruction execution, if there are not enough instructions fetched into the CPU, these technologies are useless. Thus, to be accurate, one should consider both data access and instruction access in the measurement of memory systems.

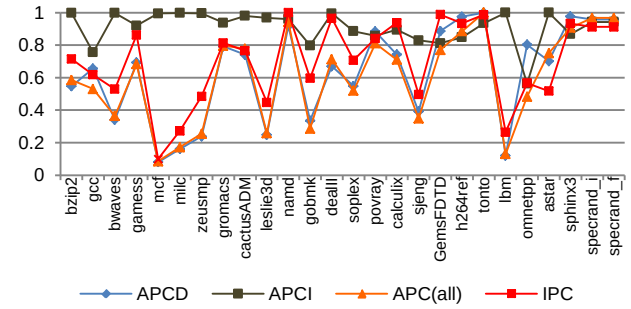

Figure 3. Normalized IPC, APC_I, APC_D, and APC_{All}

To optimize the accuracy of APC, the L1 instruction cache APC, APC_I is adopted. Using APC_D to denote the L1 data cache APC, APC_{All}, which equals $APC_D \times APC_I$, represents a new metric to evaluate overall memory performance. The reason APC_D and APC_I are correlated with multiplication is inspired by the conditional possibility. Only when the instruction is efficiently fetched into the CPU, can a data access request be generated. The correlation coefficient of APC_{All} under the default configuration (C1) improves by 2.18%. For all configurations (C1~C20), the accuracy of APC_{All} is 0.899, an improvement of 2.58% on average compared to APC_D. The reason is that for most applications the instruction accesses are almost perfect. Only very small numbers of applications, such as gcc, GemsFDTD, and h264ref, have large instruction miss rates. For these applications, APC_{All} can add more accurate adjustments. The normalized APC_I, APC_D, APC_{All} and IPC of default configuration (C1) are shown in Fig. 3. The normalization is based on the magnitude of each metric. From Fig. 3, it can be seen that the APC_{All} and IPC have almost the same variation trends. Fig. 3 confirms that overall application performance is largely determined by the memory performance as can be expressed by the APC metric.

4.2 Proximity for different configurations

In this section, we focus on each application, in order to observe the impact of memory configuration on performance and correlation. Two groups of 20 simulation configurations are run. The first group (C1~C17 in Table 4) changes basic cache/memory configuration, each configuration inside this group changes L1/L2 cache size or associativity, or memory latency respectively. The other group (C18~C20) changes non-blocking cache ability with altering the number of MSHR entry. The simulation results of the first group are presented in this section. In the next section, simulations with both groups are conducted. To clearly show the difference, APC_D and APC_{All} are compared against the other four memory metrics from Fig. 4 to Fig. 7. Similar to APC which has two measures APC_D and APC_{All}, every conventional memory metric also has two measures which represent data cache performance only and comprehensive cache performance (data and instruction cache combined performance). For example, when considering 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.

In Fig. 4 to Fig. 7, it can be seen that APC_{All} has the highest correlation coefficient value with IPC, with a average value for all applications of 0.9632, and this means that APC_{All} and IPC have a dominant relation. For other memory metrics, $AMAT_{All}$ has the closest relation with IPC, with an average value of -0.9393, a little lower than APC_{All}. This shows if advanced data access technologies, such as non-blocking, are not considered, AMAT is a quite good metric in reflecting memory performance variation. However, when considering non-blocking structure, AMAT is misleading. (Please refer to next section for details). For other metrics, there are some misleading indications as well. For example, Hit Rate should have positive coefficient values, but for several applications its coefficient values are negative or approximate to zero. One reason for the divergence is that HR does not explicitly include lower level cache performance factors, such as miss latency information (whereas APC and AMAT do consider lower level cache performance), so when L2 cache size increases or main memory latency decreases, the IPC will increase, but Hit Rate does not change. For instance, the benchmark *sjeng* has the HR correlation value of 0.058, which is mainly because when changing L2 and main memory's parameters, the hit ratio of data cache and instruction cache remain almost the same value, 0.8084 and 0.9671 respectively. However, the IPC changes from 0.677 to 0.872. It is also found for 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

of the mismatch is believed to be the speculation mechanism. When the cache associativity increases, it does not only decrease misses count, but also decreases the memory access count, because of omitting to execute mis-speculated instructions. As stated earlier, not all mis-speculated memory accesses are harmful. On the contrary, APC is much immune to speculation, because when counting access cycles, the overlapping mode is adopted, and the mis-speculated memory accesses are overlapped with correctly speculated memory accesses. In contrast to HR and HPKI missing information of lower level cache, AMP only considers lower level cache miss access. When L1 data/instruction cache size increases or associativity increases, the number of miss accesses decreases, but the miss latency for each miss may not change.

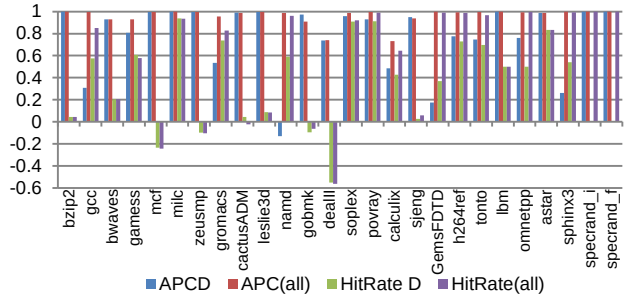

Figure 4. The Correlation Coefficients of APC and HR

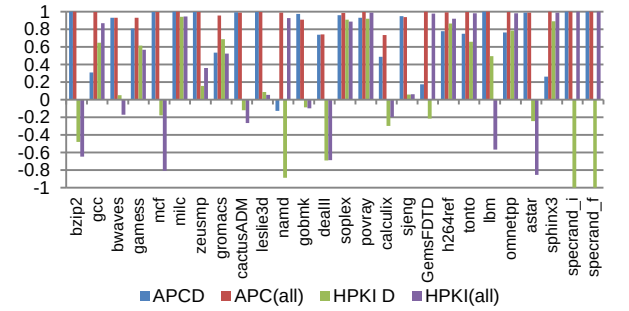

Figure 5. The Correlation Coefficients of APC and HPKI

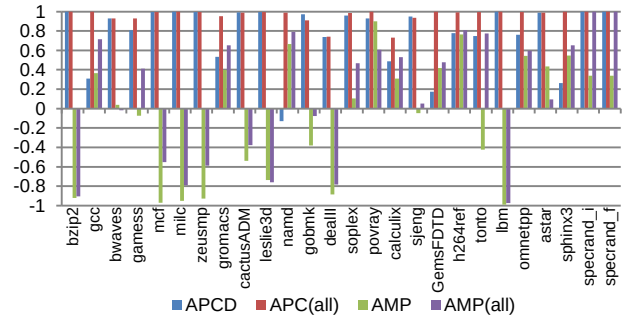

Figure 6. The Correlation Coefficients of APC and AMP

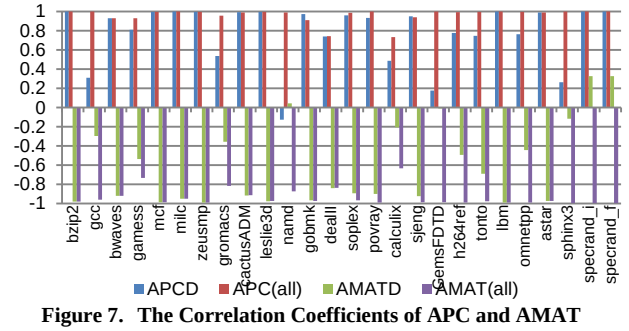

Figure 7. The Correlation Coefficients of APC and AMAT

4.3 Changing access parallelism in Cache Structure

APC and AMAT have very similar correlation values with IPC when changing basic cache/memory configurations. Here the number of MSHR entries is altered to examine whether the two memory metrics still have a similar relation with IPC. Fig. 8 shows the correlation coefficient of APC and AMAT for all the twenty configurations listed in Table 4.

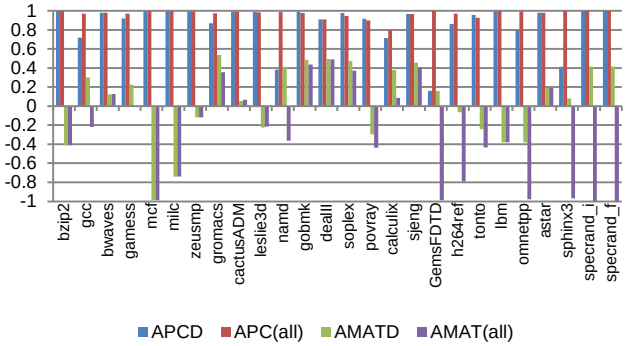

Figure 8. The Correlation Coefficients of APC and AMAT with MSHR changes

Fig. 8 shows that APC still has the same correlation, with an average value of 0.9696. However, AMAT could not correctly reflect IPC for most applications. The reason is that when there are not enough MSHR entries, the CPU will be blocked by the memory system. APC can record the CPU blocked cycles by counting MSHR Cycle, whereas AMAT cannot. On the contrary, contention increases with the number of MSHR entries; therefore, the individual data access time, that is AMAT, will increase as well. The variation of IPC and APC matches well with each other. However, AMAT gives a false indication about memory system performance.

Through the above simulation and analyses, under different applications and under different configurations, it can be seen that the APC metric is the most appropriate performance metric for memory systems. APC can directly determine the overall system performance in all the testing. In contrast, other existing memory metrics cannot accurately reflect the system performance, and sometimes even mislead the performance.

5. DISCUSSION

In this section, some key usages of APC, which help to understand application behaviors, are discussed. Also an important quantitative definition of data intensive application based on the APC concept is given.

5.1 Bottleneck inside memory system

From Fig.4 through Fig.8, with the dominating correlation between APC and IPC, it is clear that the performance is now determined by memory system performance. An important question then is at which level of the memory system is the actual bottleneck of performance. According to the APC's definition, each level of memory hierarchy has its own APC values: L1 data and instruction caches have APC_D and APC_I respectively; L2 cache has APC_{L2}; and main memory has APC_M. However, each level's APC not only represent the performance of its memory level, but also includes all the lower levels of the memory hierarchy. For example, the value of APC_D represents the memory performance of L1 data cache, L2 cache and main memory; and APC_{L2} represents the memory performance of L2 cache and main memory. Only 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 bottleneck inside the memory system. Fig. 9 shows the correlation value of all level APCs for 26 benchmarks. For example, for benchmark *mcf*, both APC_{All} and APC_{L2} have dominant relation with IPC, but its APC_M does not. That means the performance of *mcf* application is determined by its L2 cache performance, and has a good locality. For benchmark *lbm*, since all levels of APC have dominating correlation with IPC, the performance of *lbm* is determined by the performance of the lowest level, namely the main memory level.

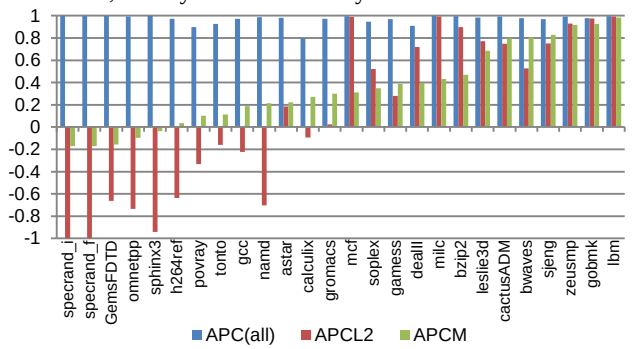

Figure 9. Correlation Coefficients of APC_{All}, APC_{L2}, and APC_M

5.2 A quantitative definition of data intensive

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 no commonly accepted definition of "data-intensive computing" or quantitative measurement of "data intensive". As APC characterizes the overall memory performance, the IPC and APC correlation value provides a quantitative definition of data intensive. The idea is simple: if APC_M dominates IPC performance, then the application is data intensive. The degree of the domination provides a measurement of data intensiveness. We use the correlation value of APC_M to quantify the degree of data intensive. There are three reasons to use APC_M instead of APC_D to measure data intensive. First, due to the "memory-wall" problem, memory latency becomes the most important performance bottleneck in the memory hierarchy and dramatically drags CPU speed. Next, we do not count data re-use as part of data-intensiveness unless it has to be read from main memory again. Finally, according to the definition APCs, if the APC_M has a dominate relation with IPC, then APC_{L2} and APC_D will have a dominate relation with IPC too. In other words, we want to drop the cache hit influence, since we are interested in data movement, not usage, concerning data intensiveness. Also please note the hardware cost for measuring APC_M is almost zero.

Fig. 9 is sorted according to APC_M correlation values in ascending order (the farther to the left, the smaller the value of APC_M). The correlation value of 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 Figure 9, 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 APC_{L2} and APC_M of these applications are negative or very small, As the correlation value of APC_M increases, the effect of main memory to the overall application becomes increasingly important. Therefore, in the second interval (from *mcf* to *sjeng*), some applications' performance are dominated by the L2 caches, e.g. *mcf, milc*. However, for some other applications, such as *bzip2*, L2 and main memory are both important. For the third interval, the applications' performances are dominatingly determined by main memory performance. This observation motivates us to define an application data intensive if its correlation coefficient of APC_M and IPC is equal to or larger than 0.9. Another reason for picking 0.9 as the threshold is that, according to mathematical definition of 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 is data intensive *if and only if*

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

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 [18]. MR is defined as {the number of miss memory accesses} over {the number of total memory accesses}. Similarly, MPKI is defined as {the number of miss memory accesses × 1000} over {the number of total committed Instructions}. AMP equals {the summary of single miss latency} over {the number of miss memory accesses}. Finally, AMAT = Hit time + MR×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 as if there were no 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. It does not consider the memory access overlapping and parallelism. There have been several studies on Memory Level Parallelism (MLP) [19] in recent years. MLP is a main memory metric. It is the average number of long-latency main memory outstanding accesses when there is at least one such outstanding access [20]. So

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

MLP(t) is *MLP* value at clock cycle *t*. The meaning of total memory access cycles in Equation (3) is the same with APC definition. Sorin's f_M parameter [21] has a similar definition to the MLP in [20]. Sorin's f_M considers all main memory accesses, while Chou [20] 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, 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)*s, so $APC_M = MLP/m$. That means APC_M is directly proportional to MLP. Any analyzing indication from MLP on CPU micro-architecture could also be gotten from 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 [20]. These kinds of applications usually have much more L1 and L2 cache misses, and their overall performances are heavily determined by main memory access. But for some traditional CPU intensive applications, only considering main memory access is far from enough. In addition, advanced process technology, such as EDRAM used in IBM POWER7, also dramatically increase on-chip cache size up to 32MB [22]. Using 3D integration technology, one could implement multi-layer CPU with one layer full of on-chip cache [23]. 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 application. 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 proposed a new memory metrics APC, gave its measurement methodology, and demonstrated its unique ability in measuring the overall and layered performance of modern hierarchical memory systems. Intensive simulations were conducted with a modern computer system simulator, M5, to verify the potential of APC and compared it with existing memory performance metrics. Simulation and statistical results show that APC is a significantly more appropriate memory metric than other existing memory metrics when reflecting overall performance of a memory system. APC can be applied on different levels of a memory hierarchy. Based on the correlation coefficient with different level APCs and IPC, the bottleneck of a memory hierarchy can be identified. The ability to accurately measure overall performance of a memory system has its practical importance. With the overall performance, we can clearly understand which component in a memory system should be enhanced first, reducing the cache miss ratio or increasing the bus bandwidth, for instance; we can better understand the matching of processor and memory and the matching of application and memory. In addition, the correlation of APC and IPC provides a quantitative measurement of data-intensiveness of an application. In this way, it provides a measurement of data-centric computing, and could have profound impact in future data-centric algorithm design and system development. In the future, we plan to extend APC to multi-core environments and to file systems, and plan to use APC to evaluate more advanced memory technologies, such as hardware prefetching. Based on the insights of these studies, new data-centric system optimizations will be investigated.

8. REFERENCES

[1] X.-H. Sun, and L. Ni, Another View on Parallel Speedup, *Proc. of IEEE Supercomputing'90*, NY, Nov. 1990.

[2] W. Wulf and S. McKee. Hitting the wall: Implications of the obvious. ACM SIGArch Computer Architecture News, Mar. 1995.

[3] Michael E. Thomadakis, The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms, A research report of Texas A&M University, Mar. 2011. http://sc.tamu.edu/systems/eos / nehalem.pdf

[4] Robert Fiedler, Blue Waters Architecture, Great lakes consortium for Petascale Computation. Oct. 2010.

[5] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, et.al Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *in Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996, 191~202.

[6] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun, Niagara: A 32-Way Multithreaded SPARC Processor, *IEEE Micro*, 21-29, Mar. 2005.

[7] D. Kroft. Lockup-free Instruction Fetch/Prefetch Cache Organization. *In 8th International Symposium on Computer Architecture* (ISCA81), 81–87, 1981.

[8] Amit Agarwal, Kaushik Roy, T. N. Vijaykumar: Exploring High Bandwidth Pipelined Cache Architecture for Scaled Technology. 2003: 10778-10783

[9] Jude A. Rivers, Gary S. Tyson, Edward S. Davidson, et.al, On High-Bandwidth Data Cache Design for Multi-Issue Processors, *IEEE Micro-30*, Dec. 1997.

[10] Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt, Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors, *in Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*, 129-140, Anaheim, CA, Feb. 2003.

[11] Intel White Paper, Inside Intel Core Microarchitecture and Smart Memory Access, http://www.iuma.ulpgc.es/~nunez/procesadoresILP/Intel64-Core-smartmemoryaccess-sma.pdf. 2011.

[12] Intel Reference Manual, "Introduction to Microarchitectural Optimization for Itanium®2 Processors", http://developer.intel.com, Apr. 2011.

[13] Stefan Andersson, Ron Bell, et.al, RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide, http://www.redbooks.ibm.com, Apr. 2011.

[14] Aneesh Aggarwal, Reducing Latencies of Pipelined Cache Accesses Through Set Prediction, *in Proceedings of the 19th annual international conference on Supercomputing (ICS '05)*, 2-9, Boston, MA, Jun. 2005.

[15] Jim Higgins, The Radical Statistician: A Practical Guide to Unleashing the Power of Applied Statistics in the Real World, Biddle Consulting Group, Apr. 2011. http://www.biddle.com/documents/bcg_comp_chapter2.pdf,

[16] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro, 26(4):52*, Jul. 2006.

[17] C. D. Spradling, SPEC CPU2006 benchmark tools. *ACM SIGARCH Computer Architecture News*, 2007.

[18] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 4th edition, Sep. 2006.

[19] A. Glew, “MLP yes! ILP no!,” in ASPLOS Wild and Crazy Idea Session ‘98, Oct. 1998.

[20] Y. Chou, B. Fahs and S. Abraham, Microarchitecture Optimizations for Exploiting Memory-Level Parallelism, *in 31st International Symposium on Computer Architecture* (ISCA04), Jun. 2004.

[21] D. Sorin Vijay, S. Pai, Sarita V. Adve, Mary K. Vernon, and David A. Wood, Analytic Evaluation of Shared-Memory Systems with ILP Processors, *in 25th International Symposium on Computer Architecture*, 1998

[22] William Starke, POWER7: IBM's Next Generation Balanced POWER Server Chip, in Hotchip 21, Aug. 2009.

[23] Yuh-Fang Tsai, Feng Wang, Yuan Xie, Design Space Exploration for 3-D Cache, *in IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 444 - 455, 2008.