Liu YH, Sun XH. Reevaluating data stall time with the consideration of data access concurrency. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(2): 227–245 Mar. 2015. DOI 10.1007/s11390-015-1517-2

Reevaluating Data Stall Time with the Consideration of Data Access Concurrency

Yu-Hang Liu (刘宇航), *Member, CCF, ACM, IEEE*, and Xian-He Sun (孙贤和), *Fellow, IEEE*

Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616-3793, U.S.A.

E-mail: {yuhang.liu, sun}@iit.edu

Received November 21, 2014; revised January 8, 2015.

Abstract Data access delay has become the prominent performance bottleneck of high-end computing systems. The key to reducing data access delay in system design is to diminish data stall time. Memory locality and concurrency are the two essential factors influencing the performance of modern memory systems. However, existing studies in reducing data stall time rarely focus on utilizing data access concurrency because the impact of memory concurrency on overall memory system performance is not well understood. In this study, a pair of novel data stall time models, the L-C model for the combined effort of locality and concurrency and the P-M model for the effect of pure miss on data stall time, are presented. The models provide a new understanding of data access delay and provide new directions for performance optimization. Based on these new models, a summary table of advanced cache optimizations is presented. It has 38 entries contributed by data concurrency while only has 21 entries contributed by data locality, which shows the value of data concurrency. The L-C and P-M models and their associated results and opportunities introduced in this study are important and necessary for future data-centric architecture and algorithm design of modern computing systems.

Keywords memory wall, data stall time, memory concurrency, concurrent average memory access time (C-AMAT)

1 Introduction

The term "memory wall" was introduced by Wulf and McKee in 1994[1]. During the last two decades, processors have advanced from in-order to out-of-order and from uni-core to multi-core; memory systems have advanced with many concurrency-oriented features such as multi-port, multi-banked, pipelined and non-blocking cache. Moreover, data intensive applications have become common in diverse fields such as bioinformatics, computer aided design, and complex social media interactions[2-3]. Compared to 20 years ago, the landscape of computing has changed.

Data stall time is the time CPU waiting for data. Many factors can cause data stall, including data access delay, branch, control, and data dependencies. Data stall time contributes 50% to 70 % of the total application execution time and is the most prominent performance bottleneck of computing systems[4-5]. Memory wall compounded with the drastic increasing of data intensive applications makes data access delay the leading performance bottleneck of high-end computing. For this reason, intensive research has been conducted by various researchers in recent years to reduce data stall time. The major results have been summarized by Hennessy and Patterson in [6]. However, existing studies are mostly focused on one aspect of data stall time reduction and data locality. Although researchers have realized the importance of memory concurrency[7-9], there is no good understanding on utilizing data concurrency for the overall system performance.

From a hardware perspective, concurrency is already available in modern memory systems, such as in each memory clock cycle, multiple accesses may be issued. As a result, many accesses may co-exist in the same cycle. Memory access concurrency also exists at each level of a memory hierarchy, due to hardware designs such as multi-port and/or multi-bank caches.

Regular Paper

Special Section on Applications and Industry

The work was supported in part by the National Science Foundation of USA under Grant Nos. CNS-1162540, CCF-0937877, and CNS-0751200.

This study reevaluates data stall time with the consideration of both locality and concurrency at the same time and focuses on "pure miss". It reveals new directions and mechanisms for reducing data stall time based on a unified and balanced consideration of both data locality and concurrency, which have not received enough attention despite their increasing importance.

Modern memory systems have become increasingly more complex and cannot be analyzed without a model. Different optimizations and components are introduced to cope with the memory problem[6]. Most of these optimizations and components, however, are focused on one of the properties of memory systems, such as miss rate. Their overall influence on the underlying computing system is unclear in general. Average memory access time (AMAT) is a great tool to analyze the contribution of each memory performance parameter toward the overall memory access time in terms of locality. But, as indicated in [6], AMAT needs to be enhanced to identify data access concurrency to cope with the complexity of modern memory systems. In order to evaluate the impact of concurrency on data stall time, we use the newly proposed concurrent average memory access time (C-AMAT) model[10-11] in this study. C-AMAT is an extension of the conventional AMAT and includes memory concurrency into the average memory access time. C-AMAT confirms that memory concurrency is a first class citizen of memory system performance and is as significant as memory locality.

An analytical data stall time model that simultaneously considers concurrency and locality, named L-C model, is presented. Therefore, concurrency-oriented techniques such as prefetching, non-blocking, out-of-order execution, and multi-bank design can be analyzed and evaluated. Moreover, the L-C model has been related directly to the application execution time, and a series of analytical results are provided to facilitate understanding and reducing data stall time in modern computer architectures. Therefore, we provide a model for the reduction of applications' execution time.

This study makes the following main contributions.

1) We present a data stall time model in terms of C-AMAT. The model considers both locality and concurrency characteristics of data access patterns, and thus it is called L-C model and can be applied to general state-of-the-art processors that are no longer limited to in-order processors and can involve memory concurrency features.

2) We present a data stall time model in terms of "pure miss". The model is focused on the real source of data stall, pure miss, and thus it is called P-M model and reshapes traditional mind that "all the misses are harmful for performance".

3) We derive three theoretical results with respect to the properties of pure miss rate, pure miss penalty, and pure miss concurrency in the data stall time model, exposing the relationship between data stall time and data access patterns. The results are useful in the reduction of data stall time (therefore execution time) in modern computer architectures.

4) We reveal pitfalls of current memory systems and new opportunities for a future memory system, based on the model and theoretical results.

We introduce a framework (see Table 3) with regard to data stall time optimizations. The framework presents 38 new concurrency-contributed entries for data stall time reduction, showing the model's effectiveness for memory optimization and for future architecture and software design on modern processors.

The rest of this paper is organized as follows. Section 2 provides a brief review of locality and concurrency of data access. Section 3 proposes the data stall time model. Then, Section 4 presents a series of theorems for data stall time. Section 5 describes our experimental details and validations for our results. Section 6 discusses opportunities for future memory systems and pitfalls of current memory systems in reducing data stall time. Section 7 summarizes related work, and finally, Section 8 provides concluding remarks and future work.

2 Backgrounds on Memory Locality and Concurrency

There are three commonly used performance metrics for evaluating memory systems[6], namely miss rate (MR), average miss penalty (AMP), and average memory access time (AMAT). MR reflects the portion of the accessed data in or out of the cache, but not the penalty of the miss access. AMP only reports the penalty of the cache misses; it does not show the performance degradation. AMAT is a comprehensive memory metric, but it is still based on a single data access viewpoint, and does not consider the memory hit and miss access concurrency. The conventional AMAT formulation is shown in (1). AMAT does not consider the concurrency of memory accesses, in either the hit or the miss section of the formula.

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

As shown in (2), C-AMAT extends AMAT to consider memory concurrency. C_H can be contributed by caches with multi-port or multi-bank. C_M can be contributed by non-blocking cache structure. In addition, out-of-order execution, multiple issue pipeline, SMT, and CMP, can increase both C_H and C_M.

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

According to the definition in [10], a pure miss is the one which does not overlap with any hit access activity. The pure miss rate pMR is the number of pure misses divided by the total number of accesses, which is different from conventional miss rate MR. $pAMP$ is the average number of pure miss cycles per miss access. The notation abbreviations used in this study are given in Table 1.

Table 1. Symbol Abbreviations

Notation	Meaning
$AMAT$	Average memory access time
$C\text{-}AMAT$	Concurrent AMAT
H	Hit time
MR	(Conventional) miss rate
pMR	Pure miss rate
AMP	(Conventional) average miss penalty
$pAMP$	Pure AMP
C_H	Hit concurrency
C_M	Pure miss concurrency
$CPU\text{-}time$	Application execution time
IC	Instruction count
CPI_{exe}	Processor computing cycles with zero data access delay
f_{mem}	Portion of the instructions which access memory
$IssueRatio$	Probability of a new data access being issued in a clock cycle
$C\text{-}AMAT_{\text{stall}}$	Data stall time per access
$cycle\text{-}time$	Length of each clock cycle
T_{memAcc}	Memory access time
T_{memStall}	Data stall time
C_{memAcc}	Memory access count
$overlapCycles_{\text{c-m}}$	Overlapping cycles between compute cycles and memory active cycles
$overlapRatio_{\text{c-m}}$	Ratio of the compute and memory access overlapping time over the total memory active time

For clarity, Fig.1 provides a demonstration example to illustrate the "pure miss" and hit cycle concept. There are five different memory accesses in Fig.1. Each access contains three cycles for cache hit operations. If it is a miss, additional miss penalty cycles will be required, and the number of miss penalty cycles is uncertain. Accesses 1, 2, and 5 are hit accesses; accesses 3 and 4 are miss accesses. Access 3 has a 3-cycle miss penalty; access 4 has only a 1-cycle miss penalty. When considering the access concurrency, only access 3 contains two pure miss cycles. Though access 4 has one miss cycle, this cycle is not a pure miss cycle, because it overlaps with the hit cycles of access 5. Therefore, the (pure) miss rate of the five accesses is 0.2, according to our new definition of concurrent (pure) miss rate, instead of 0.4 for the conventional non-concurrent version.

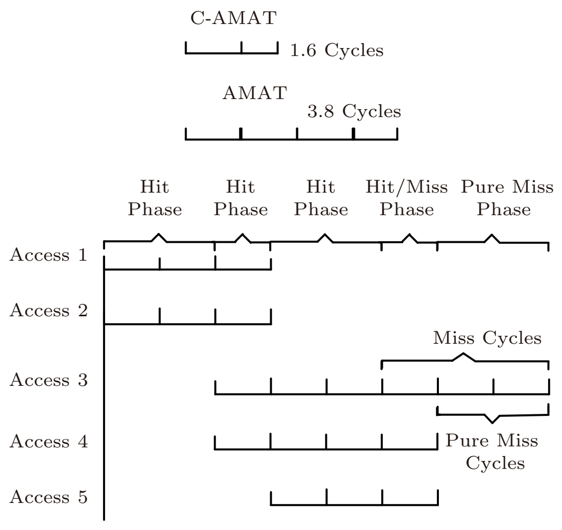

Fig.1. C-AMAT and pure miss example.

The reason for bypass misses whose cycles are overlapping with hit accesses is that this kind of miss accesses will not cause processor stall; the processor can continue processing with the hit accesses. According to (2), C-AMAT is eight cycles out of five accesses or 1.6 cycles per access; whereas by (1), AMAT is 3 + 0.4 × 2 or 3.8 cycles per access. The difference between C-AMAT and AMAT is the contribution of concurrent data access. In this example, concurrency has doubled memory performance.

Similar to AMAT, C-AMAT can be reduced by optimizing its five parameters. In particular, increasing the average hit and miss concurrency, and decreasing the hit access time, miss rate, and miss penalty will improve the system performance. Please notice that while the three parameters, hit access time, pure miss rate, and pure miss penalty, are inherited from AMAT, in C-AMAT, the terms of miss rate and miss penalty are redefined as pure miss rate and pure miss penalty.

Since the hit access time, H, is unchanged from AMAT and is well studied, this study focuses on the optimization of the other four parameters of C-AMAT.

More information of C-AMAT can be found in [10] and [11]. This paper focuses on using C-AMAT as a tool to reduce data stall time. It is an application of C-AMAT but not an introduction of C-AMAT, which has already been well documented in [10] and [11].

3 Proposed Data Stall Time Models

C-AMAT has two unique features:

1) C-AMAT unifies the combined impact of data access locality and concurrency on data access delay;

2) C-AMAT describes the impact of the "pure miss", which is deemed as the source of data stall, on overall data access delay.

For the former, the impacts of data locality and concurrency are entangled and often conflict with each other. A unified description of the joint impact of locality and concurrency is indispensable for data stall time reduction, and is missing in current studies. C-AMAT is a timely tool for data intensive applications. To effectively apply C-AMAT in engineering practice, we need to have a better understanding of the relation between C-AMAT and CPU time. That is, when we optimize the locality or the concurrency in terms of the five parameters of C-AMAT, what is the effect on the CPU time?

For the latter, the feature 2), we notice that only the "pure misses" cause data stall, thus focusing on reducing (non-pure) miss numbers is not always necessary and often wasteful. For this reason, we have developed a model to analyze and reduce pure miss only.

In this section, based on the two features of C-AMAT, we formally derive models and formulations to unify the impact of data locality and concurrency on pure miss reduction.

3.1 Combined Impact of Locality and Concurrency

The execution time of each processor in a system consists of two components[6]: processor active time and data stall time. Here the processor active time is the time during which the processor is busy executing the instructions of the user program. The data stall time is the time when the processor is stalled waiting for memory reference. This time consists of the access delay, the contention delay, and, in multi-thread cases, the latency overhead due to cache coherency and consistency.

(3) gives the CPU time of an in-order processor in terms of these two components of time[6].

$$\textit{CPU-time} = IC \times (CPI_{\text{exe}} + f_{\text{mem}} \times AMAT) \times \textit{cycle-time}. \tag{3}$$

In an in-order processor, when a data miss occurs, the processor waits for the data to be fetched before continuing. This can result in a data stall lasting several cycles, depending on where in the memory hierarchy the data resides[6]. However, in an out-of-order processor, when a miss occurs, other instructions can be executed while the memory system is servicing the miss. This allows multiple outstanding reads and writes, depending on the underlying hardware supports. Therefore, some of the memory system latency is hidden. An out-of-order processor is regarded as being stalled in a clock cycle if it does not retire the maximum possible number of instructions in that cycle[6].

There are three kinds of concurrency. The first kind is between the non-memory instructions within the computing components of the processor such as ALU and FPU, which is referred to as instruction level parallelism (ILP). The second kind is between concurrent memory references, which is the data access parallelism, and has already been considered by C-AMAT. The last kind is between computing and memory accesses, which can be characterized by $overlapRatio_{\text{c-m}}$ shown in (4). $overlapRatio_{\text{c-m}}$ is the ratio of the computing and memory access overlapping time over the total memory access time. We focus on the latter two types of concurrency, formulating their impacts on final performance.

$$overlapRatio_{\text{c-m}} = \frac{overlapCycles_{\text{c-m}}}{T_{\text{memAcc}}}. \tag{4}$$

If the processor is stalled, the memory must be active, and vice versa. In other words, the total memory active time does not completely equal the processor data stall time. (3) does not hold for out-of-order processors, and cannot reflect the concurrency features in the modern complex memory systems.

The data stall time model can be derived in terms of C-AMAT, and then the final CPU time can be expressed as follows, which is called L-C model.

Theorem 1. *Locality-Concurrency (L-C) model*:

$$\textit{CPU-time} = IC \times (CPI_{\text{exe}} + f_{\text{mem}} \times \textit{C-AMAT} \times (1 - overlapRatio_{\text{c-m}})) \times \textit{cycle-time}.$$

Proof.

Based on definition[10],

$$C\text{-}AMAT = \frac{T_{\text{memAcc}}}{C_{\text{memAcc}}}.$$

Therefore,

$$\begin{aligned}&C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{T_{\text{memAcc}}}{C_{\text{memAcc}}} \times (1 - overlapRatio_{\text{c-m}}).\end{aligned}$$

Because,

$$overlapRatio_{\text{c-m}} = \frac{overlapCycles_{\text{c-m}}}{T_{\text{memAcc}}}.$$

Thus,

$$\begin{aligned}&C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{T_{\text{memAcc}}}{C_{\text{memAcc}}} \times \left(1 - \frac{overlapCycles_{\text{c-m}}}{T_{\text{memAcc}}}\right).\end{aligned}$$

That is,

$$\begin{aligned}&C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{T_{\text{memAcc}}}{C_{\text{memAcc}}} \times \frac{T_{\text{memAcc}} - overlapCycles_{\text{c-m}}}{T_{\text{memAcc}}}.\end{aligned} \quad (5)$$

Writing memory access time in terms of computing time and data stall time, we get (6).

$$overlapCycles_{\text{c-m}} + T_{\text{memStall}} = T_{\text{memAcc}}. \quad (6)$$

Combining (5) and (6), we can derive

$$\begin{aligned}&C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{T_{\text{memAcc}}}{C_{\text{memAcc}}} \times \frac{T_{\text{memStall}}}{T_{\text{memAcc}}}.\end{aligned}$$

And recall that instruction number multiplied by memory access frequency is equal to the number of memory accesses,

$$IC \times f_{\text{mem}} = C_{\text{memAcc}}.$$

We can get

$$\begin{aligned}&f_{\text{mem}} \times C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{C_{\text{memAcc}}}{IC} \times \frac{T_{\text{memAcc}}}{C_{\text{memAcc}}} \times \frac{T_{\text{memStall}}}{T_{\text{memAcc}}}.\end{aligned}$$

Thus, (7) holds.

$$\begin{aligned}&f_{\text{mem}} \times C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{T_{\text{memStall}}}{IC}.\end{aligned} \quad (7)$$

That is, the number of data stall cycles per instruction equals $f_{\text{mem}} \times C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})$ if memory concurrence is considered.

By definition, CPI_{exe} can be expressed as average compute time per instruction,

$$CPI_{\text{exe}} = \frac{T_{\text{compute}}}{IC}. \quad (8)$$

Therefore, combining (7) and (8), we can get (9).

$$\begin{aligned}&CPI_{\text{exe}} + f_{\text{mem}} \times C\text{-}AMAT \times\\ &(1 - overlapRatio_{\text{c-m}})\\ =\ &\frac{T_{\text{compute}} + T_{\text{memStall}}}{IC}.\end{aligned} \quad (9)$$

Since the total application execution time is the computing time plus the data stall time, and *CPI* is the amortized application execution time per instruction,

$$CPI = \frac{T_{\text{compute}} + T_{\text{memStall}}}{IC}.$$

Therefore,

$$\begin{aligned}&CPI_{\text{exe}} + f_{\text{mem}} \times C\text{-}AMAT \times\\ &(1 - overlapRatio_{\text{c-m}}) = CPI.\end{aligned}$$

And because $CPU\text{-}time = IC \times CPI \times cycle\text{-}time$ is a classical formula[6], the L-C model holds. □

The proof of the L-C model leads to two interesting corollaries.

Corollary 1. *When memory concurrency is considered,*

$$\begin{aligned}&\text{data stall cycles per instruction}\\ =\ &f_{\text{mem}} \times C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}}).\end{aligned}$$

Proof. This is (7). □

Corollary 2. (3) *is a special case of the L-C model where memory concurrency is not considered.*

Proof. A direct result of Corollary 1. □

The correctness of the L-C model is based on the following assumptions. First, C-AMAT here is measured for L1 caches. Second, (3) is designed for a single program. As an extension of (3), in the L-C model, C-AMAT is also considered and measured only for a single program. In multi-program environment, C-AMAT can be measured for each program individually.

Recall that C-AMAT contains AMAT as a special case where memory concurrency does not exist. When memory concurrency does not exist, $C_H = 1$ and $C_M = 1$, $pAMP = AMP$, $pMR = MR$; therefore, $C\text{-}AMAT = AMAT$. Especially, $overlapRatio_{\text{c-m}} = 0$. At this time, the L-C model becomes (3). Therefore, the L-C model is valid regardless of the processor type and memory concurrency features.

Fig.2 shows an execution scenario of instruction computing and memory access. This scenario involves both instruction level parallelism and data access parallelism. The data access activities are the same as those in Fig.1. There are six instructions that are completed in eight cycles, and only the sixth one does not need data access, so $CPI = 8/6$, $f_{\text{mem}} = 5/6$. Because the compute phase of the six instructions takes six cycles, $CPI_{\text{exe}} = 6/6$. The $C\text{-}AMAT$ value is 8/5. The memory access time T_{memAcc} is eight cycles, in which six cycles are overlapped by compute, and thus $overlapCycles_{\text{c-m}}$ is 6 cycles, and $overlapRatio_{\text{c-m}}$ is 6/8. Therefore, $CPI_{\text{exe}} + f_{\text{mem}} \times C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}}) = 8/6$, which is also the value of CPI. Due to $CPU\text{-}time = IC \times CPI \times cycle\text{-}time$, the L-C model holds.

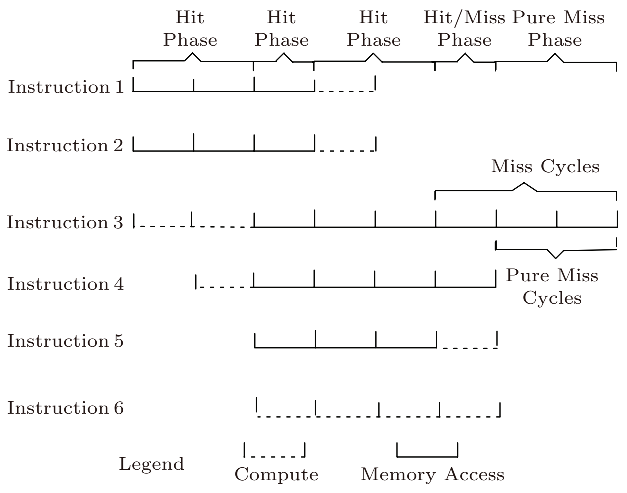

Fig.2. Execution scenario of instruction compute and memory access.

The L-C model shows that application execution time is linearly related with data stall time and the parameters of the data stall time are analytically available and practically measurable.

3.2 Effect of Pure Miss on Data Stall Time

Please notice that miss cycles which are overlapped with hit accesses do not cause processor stall, since the processor can continue generating memory accesses while waiting for the missing data to return from lower memory hierarchies. This effect can be represented by $overlapRatio_{\text{c-m}}$ which is defined in (4). In terms of C-AMAT, (10) holds.

$$overlapRatio_{\text{c-m}} = \frac{H}{C_H} / C\text{-}AMAT. \quad (10)$$

We present the proof of (10). Combining the L-C model and (10), we can derive the P-M model as follows.

Theorem 2. *Pure-Miss (P-M) Model*:

$$CPU\text{-}time = IC \times \left(CPI_{\text{exe}} + f_{\text{mem}} \times \frac{pMR \times pAMP}{C_M}\right) \times cycle\text{-}time,$$

where the data stall time is

$$data\ stall\ time = IC \times f_{\text{mem}} \times \frac{pMR \times pAMP}{C_M} \times cycle\text{-}time.$$

Generally, the P-M model directly shows the effect of pure miss on data stall time.

Proof. We first prove (10).

According to the definition of $overlapRatio_{\text{c-m}}$ in (4),

$$overlapRatio_{\text{c-m}} = \frac{overlapCycles_{\text{c-m}}}{T_{\text{memAcc}}}.$$

Because the processor can compute during hit phases,

$$overlapCycles_{\text{c-m}} = T_H,$$

and the memory access time is the sum of hit time and miss time,

$$T_{\text{memAcc}} = T_H + T_M.$$

Then we derive

$$overlapRatio_{\text{c-m}} = \frac{T_H}{T_H + T_M},$$

where H is the number of hit cycles when accessing the current cache layer. Every cache access needs to spend H cycles to determine whether this is a hit or a miss access. Note H is a constant value in our cache model. Pure miss rate (pMR) in this study is an extended version of the traditional miss rate definition with the consideration of concurrency. Only when a miss access has no overlapping with any hit accesses, this miss access is a pure miss access. Thus,

$$pMR = \frac{C_{\text{MemPMiss}}}{C_{\text{MemAcc}}},$$

where C_{MemPMiss} is the total number of pure misses. $pAMP$ is the average miss penalty which only considers pure miss accesses:

$$pAMP = \frac{T_{\text{MemPMiss}}}{C_{\text{MemPMiss}}},$$

where T_{MemPMiss} is the sum of total pure miss cycles. The pure miss cycles are the cache miss access cycles without any hit access. Thus

$$\begin{aligned}
&\frac{H}{C_H} + pMR \times \frac{pAMP}{C_M} \\
&= \frac{H}{\sum\limits_{i=0}^{N} C_i \times \frac{t_i}{T_H}} + \frac{C_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \times \frac{T_{\text{MemPMiss}}}{C_{\text{MemPMiss}}} \times \\
&\quad \frac{1}{\sum\limits_{j=0}^{M} C_j \times \frac{t_j}{T_M}} \\
&= \frac{H \times T_H}{\sum\limits_{i=0}^{N} C_i \times t_i} + \frac{C_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \times \frac{T_{\text{MemPMiss}}}{C_{\text{MemPMiss}}} \times \\
&\quad \frac{T_M}{\sum\limits_{j=0}^{M} C_j \times t_j}. \qquad (11)
\end{aligned}$$

Because,

$$\sum_{i=0}^{N} C_i \times t_i = C_{\text{MemAcc}} \times H,$$
$$\sum_{j=0}^{M} C_j \times t_j = T_{\text{MemPMiss}},$$

thus,

$$\begin{aligned}
(11) &= \frac{H \times T_H}{C_{\text{MemAcc}} \times H} + \frac{T_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \times \frac{T_M}{T_{\text{MemPMiss}}} \\
&= \frac{T_H}{C_{\text{MemAcc}}} + \frac{T_M}{C_{\text{MemAcc}}}.
\end{aligned}$$

Therefore,

$$\frac{\frac{H}{C_H}}{\frac{H}{C_H} + pMR \times \frac{pAMP}{C_M}} = \frac{\frac{H \times T_H}{C_{\text{MemAcc}} \times H}}{\frac{T_H + T_M}{C_{\text{MemAcc}}}} = \frac{T_H}{T_H + T_M},$$

and thus

$$overlapRatio_{\text{c-m}} = \frac{\frac{H}{C_H}}{\frac{H}{C_H} + pMR \times \frac{pAMP}{C_M}}.$$

We use the scenario in Fig.2 to check if the P-M model holds. pMR is 1/5, $pAMP$ is 2, C_M is 1, and thus $pMR \times pAMP/C_M$ is 2/5 which is equal to $C\text{-}AMAT \times (1 - overlapRatio_{\text{c-m}})$. Recall the L-C model, we can see that P-M model holds. □

In the following sections, the optimization of the parameters in the data stall time model is discussed.

4 Theoretical Properties of the Data Stall Time Models

The L-C model transfers the task of reducing data stall time to increase $overlapRatio_{\text{c-m}}$ and decrease f_{mem} and C-AMAT. For $overlapRatio_{\text{c-m}}$, the optimization involves the overlapping between computing and memory access. For f_{mem}, the optimization is known and is given in [6].

The P-M model further points that pMR, $pAMP$, and C_M are the real sources of data stall. In this section, we focus on the improvement of them, and three theorems are introduced accordingly. As shown by (2), C-AMAT differentiates conventional miss and pure miss. The pure miss is the real culprit for data stall time. We find that hit activity has a remarkable effect on conventional miss. $IssueRatio$ is defined as the probability of a new data access issued by the CPU in a clock cycle. For example, in Fig.2, five of the 10 cycles have new data access being issued, and thus $IssueRatio$ is 50%. We find that $IssueRatio$ can ease the impact of pMR and $pAMP$ on the final performance.

4.1 Factors Influencing *pMR*

Theorem 3 (pMR Theorem). *Assuming memory accesses are issued independently and hit time is three cycles, we have the following equation.*

$$\begin{aligned}
pMR = (1 - (&H \times IssueRatio - H \times IssueRatio^2 + \\
&IssueRatio^3)^{(AMAT-H)}) \times MR.
\end{aligned}$$

Proof. T_i denotes the time in the i-th cycle. For any memory access whose start time and end time are T_s and T_e respectively, there are $(e - s + 1)$ elapsed cycles, which are T_s, T_{s+1}, …, T_{e-1}, and T_e respectively.

$[T_s, T_e]$ interval will be referred to as the service time of the reference. Assuming the hit time lasts for h cycles from T_s to T_{s+h-1}, the miss phase will last $e - (s + h) + 1$ cycles.

$\mathrm{P}(e \geqslant s + h) = MR$, and thus $\mathrm{P}(e < s + h) = 1 - MR$,

P(each cycle in the $[T_s, T_e]$ interval is a hit cycle $|e < s + h) = 1$.

Therefore,

pMR
= P(there is at least one pure miss cycle in the $[T_s, T_e]$ interval)
= 1− P(there is no pure miss cycle in the $[T_s, T_e]$ interval)
= 1− P(each cycle in the $[T_s, T_e]$ interval is a hit cycle)

$= 1-($ P$(e \geqslant s+h)\times$ P(each cycle in the $[T_s, T_e]$ interval is a hit cycle $|e \geqslant s+h) +$ P$(e < s+h) \times$ P (each cycle in the $[T_s, T_e]$ interval is a hit cycle $|e < s+h)$).

$= 1-(MR\times$ P(each cycle in the $[T_s, T_e]$ interval is a hit cycle $|e \geqslant s+h) + (1-MR)\times 1)$. (12)

Because P(each cycle in the $[T_s, T_{s+h-1}]$ interval is a hit cycle $|e \geqslant s+h) = 1$,

P (each cycle in the $[T_s, T_e]$ interval is a hit cycle $|e \geqslant s+h)$

$=$ P(each cycle in the $[T_{s+h}, T_e]$ interval is a hit cycle $|e \geqslant s+h)$. (13)

Because when $e \geqslant s+h$, each cycle within $[T_{s+h}, T_e]$ for the current access is in miss phase,

(13)

$=$ P(for each cycle within $[T_{s+h}, T_e]$, at least one access that is different from current one is in the hit phase $|e \geqslant s+h)$

$=$ (P(at one cycle, at least one access that is different from current one is hit))$^{e-(s+h)+1}$. (14)

Assuming event A denotes "at cycle t, at least one access that is different from current one is in the hit phase", event A_0 denotes "at cycle t that is the current cycle, at least one access that is different from current one is issued", event A_1 denotes "at cycle $t-1$, at least one access that is different from current one is issued", event A_2 denotes "at cycle $t-2$, at least one access that is different from current one is issued", ..., event A_{h-1} denotes "at cycle $t-(h-1)$, at least one access that is different from current one is issued",

then

$$A = \bigcup_{i=0}^{h-1} A_i.$$

Therefore,

$$\text{P}(A) = \text{P}(\bigcup_{i=0}^{h-1} A_i).$$

Here we assume h equals 3 as the typical value of modern L1 cache.

$$\begin{aligned}\text{P}(A) &= \text{P}(\bigcup_{i=0}^{2} A_i) = \text{P}(A_0)+\text{P}(A_1)+\text{P}(A_2) - \\ &\quad \text{P}(A_0A_1) - \text{P}(A_0A_2) - \text{P}(A_1A_2) + \\ &\quad \text{P}(A_0A_1A_2).\end{aligned}$$

Suppose the memory accesses are issued in equal probability,

$$\text{P}(A_0) = \text{P}(A_1) = \text{P}(A_2) = IssueRatio.$$

Then

$$\begin{aligned}\text{P}(A) &= 3\times IssueRatio - 3\times IssueRatio^2 + \\ &\quad IssueRatio^3.\end{aligned}$$

Therefore,

$$\begin{aligned}(14) &= (3\times IssueRatio - 3\times IssueRatio^2 + \\ &\quad IssueRatio^3)^{AMAT-H}. \qquad (15)\end{aligned}$$

Combining (12) and (15), the accurate gap between pMR and MR is presented as follows.

$$\begin{aligned}&MR - pMR \\ &= MR - (1-(MR\times(H\times IssueRatio - \\ &\quad H\times IssueRatio^2 + IssueRatio^3)^{AMAT-H} + \\ &\quad (1-MR)\times 1)) \\ &= MR\times(H\times IssueRatio - H\times IssueRatio^2 + \\ &\quad IssueRatio^3)^{AMAT-H},\end{aligned}$$

and thus

$$\begin{aligned}\frac{pMR}{MR} &= 1-(H\times IssueRatio - H\times IssueRatio^2 + \\ &\quad IssueRatio^3)^{(AMAT-H)}. \qquad \square\end{aligned}$$

The pMR theorem serves the purpose for demonstrating the influential factors of pMR. From the pMR theorem, it can be observed that pMR can be reduced by decreasing the values of MR, AMAT, or by increasing the memory access issue ratio. The former indicates that the conventional methods for reducing MR and AMAT can be applied to reduce pMR directly. The latter illustrates the weight of memory concurrency in data-intensive computing, which further confirms the timely importance of C-AMAT and C-AMAT based optimizations.

The hit time of L1 cache is generally close to processor clock time, and the typical value is 1~3 cycles. The assumption that H is 3 cycles is not strict, and the results of when the hit time is 1 or 2 cycles are similar to those in Fig.3.

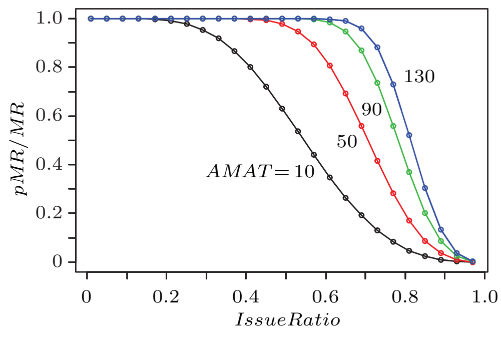

Fig.3. Effect of *IssueRatio* on *pMR*.

Based on the pMR theorem, Fig.3 is plotted to show the relationship of these factors. pMR is directly influenced by the memory access issue ratio. When the access issue ratio is low, the pure miss rate is close to the conventional miss rate. When the issue rate is high, pMR is much smaller than MR.

Fig.3 also shows that, compared with the issue ratio, the average access time AMAT plays a secondary role in pure miss rate reduction. Reducing AMAT will help pMR, but, when memory access issue ratio is high, the value of AMAT does little to influence pMR. Each line in Fig.3 corresponds to different AMAT values from 10 to 130 cycles, at an interval of 40 cycles.

4.2 Factors Influencing *pAMP*

$pAMP$ shows different characteristics compared to pMR, and exposes more opportunities for performance optimization, even for non-data intensive applications.

Theorem 4 ($pAMP$ Theorem). *Assuming the memory accesses are issued independently, $pAMP$ can be represented as:*

$$pAMP = (1-(3\times IssueRatio - 3\times IssueRatio^2 + IssueRatio^3))\times MR\times AMP.$$

Proof.

$$\begin{aligned}&pAMP\\ &= \text{P(at cycle } t\text{, no access that is different from current one is in the hit phase } |e > s+h-1)\ (e-s-h+1)\\ &= (1-\mathrm{P}(A))\ (e-s-h+1),\end{aligned}$$

where $e-s-h+1 = MR\times AMP$ (AMP is calculated for misses, but $e-s-h+1$ is counted for all accesses).

Recall

$$\mathrm{P}(A) = 3\times IssueRatio - 3\times IssueRatio^2 + IssueRatio^3;$$

therefore,

$$\frac{pAMP}{AMP} = (1-(3\times IssueRatio - 3\times IssueRatio^2 + IssueRatio^3))\times MR. \qquad \square$$

The $pAMP$ theorem justifies that $pAMP$ can be reduced by decreasing MR or AMP, or by increasing the memory access issue ratio. The former implies that the conventional methods for reducing MR and AMP can be applied to reduce $pAMP$, and the latter shows that memory concurrency is especially important for data-intensive computing.

Based on the $pAMP$ theorem, Fig.4 is drawn to show the relationship of these factors. $pAMP$ is directly influenced by the memory access issue ratio. When the ratio is low, the pure miss penalty is close to the conventional miss penalty. When the issue ratio is high, $pAMP$ is much smaller than AMP. Fig.4 also shows that the conventional miss rate MR plays a secondary role in pure miss penalty reduction. Reducing MR will help $pAMP$, but when the memory access issue ratio is high, the value of MR does little to influence $pAMP$. Each line in the figure corresponds to different MR values from 0.1 to 1.0.

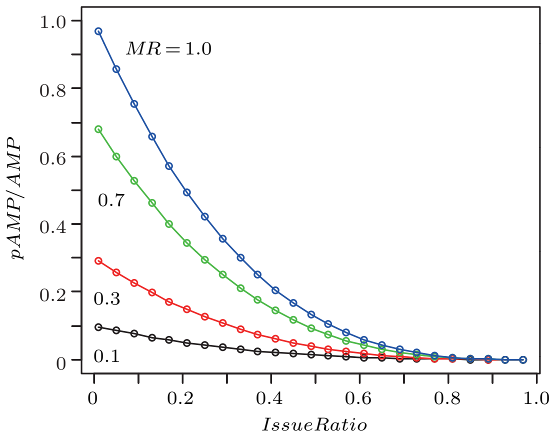

Fig.4. Effect of $IssueRatio$ on $pAMP$.

4.3 Factors Influencing C_H and C_M

Compared with pMR and $pAMP$, C_H and C_M are concurrency parameters which characterize memory concurrency directly.

Theorem 5 (C_H and C_M Theorem). *Assuming the instruction window size is IW, the portions of the data dependency, control dependency, and memory references are $f_{\text{data\_dep}}$, $f_{\text{control\_dep}}$, and f_{mem} respectively, within the window, and the hit rate is $(1-MR)$, then*

$$C_H = \min|\{IW\times f_{\text{mem}}\times (1-f_{\text{data\_dep}}-f_{\text{control\_dep}})\times(1-MR), \textit{number of cache port}\times \textit{number of cache pipeline stage}\}. \tag{16}$$

Similarly, for the average miss concurrency C_M,

$$C_M = \min\{IW\times f_{\text{mem}}\times(1-f_{\text{data\_dep}}-f_{\text{control\_dep}})\times MR, \#MSHR\}, \tag{17}$$

where #MSHR is the dynamic number of MSHR entries.

Proof.

As illustrated in Fig.5, miss concurrency C_M can be derived through the following five steps.

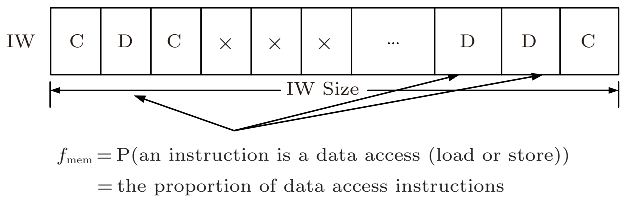
IW
C
D
C
×
×
×
...
D
D
C
IW Size
$f_{\text{mem}} = \text{P(an instruction is a data access (load or store))}$
$= \text{the proportion of data access instructions}$

Fig.5. Miss concurrency is affected by both hardware and software. C: compute instruction. D: data access instruction.

First, IW, the size of the instruction window (IW) sets a hard limit on the ability of an ILP processor to look ahead to find latency hiding instructions.

Second, within the instruction window, not all of the instructions issue memory requests, and the actual number of memory requests is $IW \times f_{\text{mem}}$.

Third, overlapping the execution of non-memory instructions with stalled accesses has two requirements. 1) There must be instructions available that do not depend upon the data being accessed (data dependency). 2) The processor should know the next instruction to be executed, regardless of the value of the data being read (control dependency). Control dependencies are caused by branches within uni-processor applications, and both branches and synchronization stalls within multiprocessor applications. Here the number of the memory references that are not related with data dependency and control dependency is $IW \times f_{\text{mem}} \times (1 - f_{\text{data\_dep}} - f_{\text{control\_dep}})$.

Fourth, among the memory accesses, the portion of the hit ones is $(1 - MR)$. Thus the demand size is determined as $IW \times f_{\text{mem}} \times (1 - f_{\text{data\_dep}} - f_{\text{control\_dep}}) \times (1 - MR)$.

Finally, the actual miss concurrency may be less than the demand size, due to the physical MSHR size presenting a hardware limitation for the maximum miss concurrency, which will be discussed more in Subsection 5.2.4. This has concluded the proof of (17).

The proof of the C_H theorem can be conducted in a similar manner. □

(16) and (17) are similar, and thus here we only present the discussion on one of them, (17), because miss penalty is much larger than hit time.

From (17), it can be seen that the average miss concurrency increases with the decrease in data and control dependency, or the increase of MR, and the size of IW and MSHR. The former three parameters, IW, f_{men}, $f_{\text{data\_dep}}$, are related to application features, while the later two parameters, MR, $\#MSHR$ are related to architecture features. The former three are also the conventional concerns of AMAT. Only, with the consideration of concurrency, the data and control dependency becomes more complex here than that in AMAT.

We consider all the possible values of the parameters in the equations, and then plot the cumulative distribution of pure miss concurrency as shown in Fig.6. In an ideal case, MSHR supports all the outstanding misses within the instruction window, and in this case, average miss concurrency is optimal. If we use a large IW (ROB is assumed with the same size) and let MSHR support more misses, then the maximum value of C_M increases.

By combining the theorems together, we conclude that pMR and $pAMP$ are directly influenced by data access intensity. This feature is vital for data-intensive computing. Also, locality remains important under memory concurrency because conventional locality-oriented techniques are useful for the reduction of pMR and $pAMP$.

The C_M theorem shows that access concurrency is the result of the interaction between application and architecture features. Rather than discussing the activities within a processor following a cache miss via a series of simulation experiments[12], the C_M theorem presents the general theoretical result to reveal the essence behind pure miss concurrency, which allows a clear direction to maximize concurrency.

5 Experimental Verifications

Experimental simulations are conducted to verify the theoretical results. The experiment settings and results are presented in this section.

5.1 Experimental Setup

As shown in Fig.7, the state-of-the-art cycle-accurate simulators GEM5[13] and DRAMSim2[14] are integrated together. The C-AMAT analyzer with detectors for both hits and misses is implemented.

SPEC CPU2006 benchmark suite[15] is used in our simulations. The benchmarks are compiled using GCC 4.3.0 and the -O3 optimization level, and are executed using reference input sets.

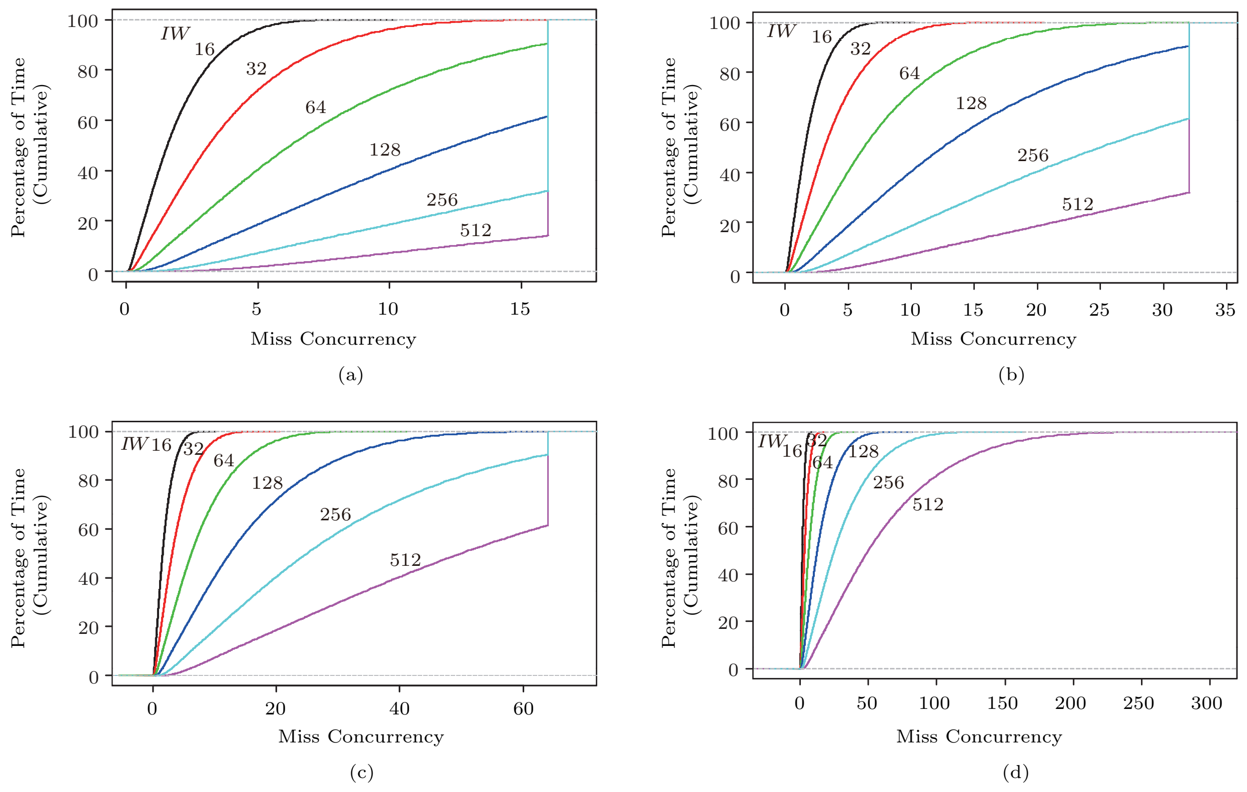

Fig.6. Cumulative distribution of miss concurrency when MSHR simultaneously supports different number of misses. (a) #MSHR = 16. (b) #MSHR = 32. (c) #MSHR = 64. (d) #MSHR = 512.

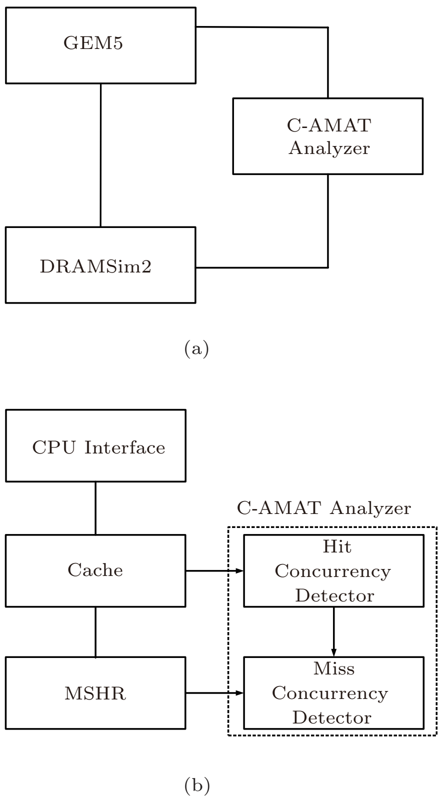

Fig.7. (a) Simulation infrastructure and (b) C-AMAT detecting structure.

A detailed out-of-order CPU model in the GEM5 simulator is adopted. The experiments assume the default configurations as shown in Table 2. For each benchmark, 10 million instructions are simulated to collect statistics.

5.2 Results and Analysis

We use the simulator and real benchmarks to partly check if the analytical model is right. Due to space limitation, only a few results are presented.

5.2.1 *Data Stall Time Model Result*

For the L-C model, we verify two things: the L-C model is correct and (3) is incorrect when data concurrency exists.

As shown in Fig.8, the weighted AMAT value ($f_{\text{mem}} \times AMAT$) is often larger than the total CPI. Certainly, that is wrong. AMAT-based (3) does not work well when memory concurrency is dominant. On the other hand, Fig.9 shows that the total CPI of an application can be decomposed into processor active time and C-AMAT based data stall time, where the data stall time is computed in terms of C-AMAT. The data stall time in Fig.9 is obtained through the expression in Corollary 1 and is equal to the directly measured values.

Table 2. Default Machine Configurations

Parameter	Value
Core	4 GHz out of order processor
	4 issue width
	128-entry instruction window and ROB
	48-entry load and store buffer
	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
Caches	L1 caches: 32 KB Inst/32 KB data, 2-way, 64 B cache line, 3-cycle hit latency
	L1 MSHR: ICache 8 MSHR/DCache 8 MSHR, 32 targets/MSHR
	L2 cache: 512 KB, 16-way, 64 B cache line, 24-cycle hit latency
	L2 MSHR: 16 MSHR, 32 targets/MSHR
Main memory	200 MHz bus cycle, 8 GB DDR2-PC3200
	Close page policy
	Latency: 12.5-12.5-12.5 ns (tRP-tRCD-CL)
	32 DRAM banks

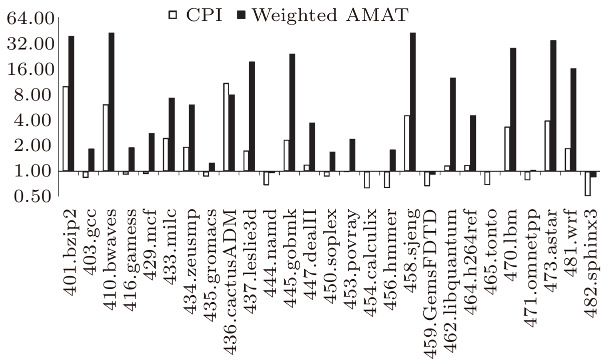

Fig.8. Weighted AMAT is larger than total CPI.

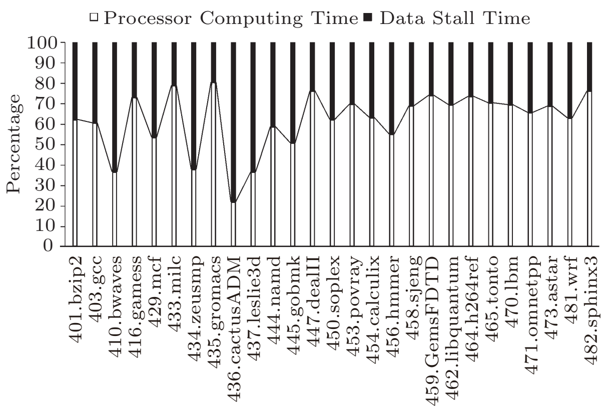

Fig.9. C-AMAT matches the actual measurement.

5.2.2 *Pure Miss Rate Result*

As shown in Fig.10, for benchmarks of SPEC CPU2006, only a small percent of their access cycles issue new memory access, with the largest one less than 30%.

Therefore, according to Fig.3, the ratio between pMR and MR should be approximate to 1, which is consistent with the simulation results shown in Fig.11. The low ratio of issue memory accesses makes optimizing C-AMAT via pMR reduction a vain attempt for SPEC CPU2006 under current system implementation. This does not mean pMR has no potential. As we show in the next subsection, this in fact shows that current system design has pitfalls, which requires further investigation. Based on Fig.3, if the ratio of issue memory-access cycles is high, pMR could be significantly different with MR. Fig.10 and Fig.11 show that current computing systems have not fully utilized memory concurrency. New ways to improve memory performance via concurrency are summarized in Table 3.

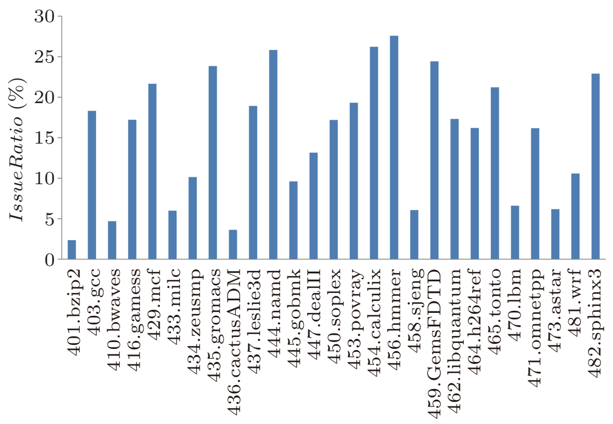

Fig.10. *IssueRatios* of 27 benchmarks from SPEC CPU2006.

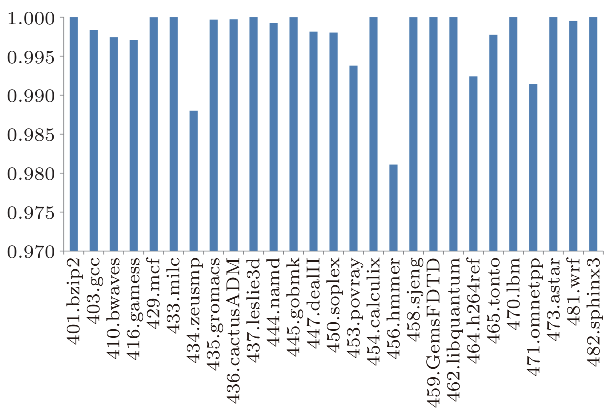

Fig.11. Gap between pure miss rate and conventional miss rate.

For applications whose memory access probabilities in each cycle are large enough, while AMAT is small, the pMR reduction will be remarkably significant.

5.2.3 Pure Miss Penalty Result

Fig.12 illustrates that the ratio between $pAMP$ and AMP is affected by MR and $IssueRatio$. The ratio between $pAMP$ and AMP is in negative correlation to $IssueRatio$ and in positive correlation to MR. These trends are consistent with the curves shown in Fig.4.

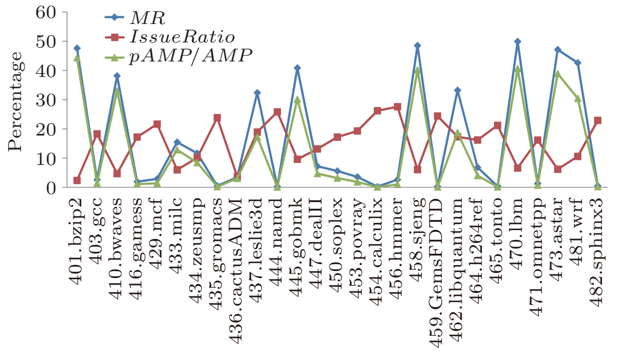

Fig.12. Ratio between $pAMP$ and AMP is affected by MR and $IssueRatio$.

5.2.4 Pure Miss Concurrency Result

A detailed validation may need many experimental configurations to verify the effects of IW size, data dependency, control dependency, physical MSHR size, etc. Readers may take the data in [10] for reference, which fits well for the C_M theorem. Here, our focus is on the discussion of the following three questions:

Q1: What is the main source of miss concurrency of an application?

Q2: How does MSHR impact the miss concurrency?

Q3: What is the relationship between data locality and miss concurrency in terms of MSHR?

Based on the results of a number of comparative experiments, it is discovered that the average miss concurrency is not proportional to the physical MSHR size (as shown in Fig.13), and the maximum miss concurrency could be much larger than the physical MSHR size. There are large gaps between them. These indicate that the miss concurrency has not been optimized for most applications. The reasoning behind this phenomenon is two-fold. 1) Concurrency is limited by the inherited data access concurrency of the application in an instruction window. 2) Even when a miss occurs, locality does exist in the form of reusing opportunity.

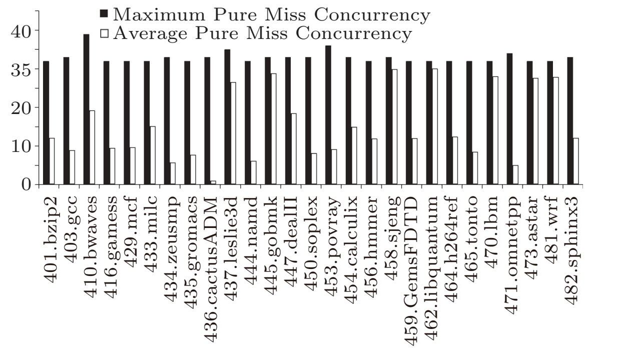

Fig.13. Maximum and average pure miss concurrency of L1 data cache.

According to the C_M theorem, the gaps between maximum and average miss concurrency are due to the small number of memory accesses within the instruction window, or the data and control dependencies within the application. This claim is confirmed by our simulation results.

Fig.13 shows that "436.cactusADM"[15] has the worst average pure miss concurrency. "436.cactusADM" solves the Einstein evolution equations. The main data structure of 436.cactusADM is a 3-D array with iteration over points. Each iteration only accesses the nearest neighbors in one dimension, which incurs many data dependencies and limits the data access concurrency. As a result, $f_{\text{data\_dep}}$ is high and f_{mem} is low within the IW.

Fig.13 also shows "458.sjeng" and "462.libquantum" are examples with high miss concurrency. They have good reasons to be so. As a chess-play program, "458.sjeng"[15] attempts to find the best move via a combination of tree searches, advanced move ordering, positional evaluation, and heuristic forward pruning. Multiple independent searches can occur simultaneously. That is, $f_{\text{data\_dep}}$ is low and f_{mem} is high within the IW, which incurs many concurrent misses. As a program simulating a quantum computer, "462.libquantum" implements the Shor's factorization algorithm[15], which relies heavily on the ability of a quantum computer to be in many states simultaneously. Physicists call this behavior a "superposition" of states. To compute the period of a function, the algorithm evaluates the function at all points simultaneously, and thus f_{mem} is high. Since these points are independent with each other, $f_{\text{data\_dep}}$ is low. These application features continually incur a number of outstanding data accesses. $f_{\text{control\_dep}}$ can be analyzed in a similar manner through understanding the applications.

For the first question, using the above analysis, we can conclude that miss concurrency depends heavily on the number of memory accesses in an instruction window and their issue dependency, which is the term $IW \times f_{\text{mem}} \times (1 - f_{\text{data\_dep}} - f_{\text{control\_dep}}) \times MR$ denoted in the C_M theorem.

The second question is related to the use of hardware structure MSHRs. Multiple misses sharing one MSHR at the same time are referred to as "reusability of MSHR". The reason of MSHR reuse is that the misses that fall into the same cache line will be held by a single MSHR. As a result, due to the spatial locality of misses, the occurrence of misses sometimes shows a clustering feature. As shown in Fig.14, there are different degrees of reusability in each physical MSHR, from 2 to 16. Therefore, the static and the dynamic number of MSHRs must be defined. The static number is the number of physical MSHRs. Each physical MSHR can be shared by many missing accesses and thus can be seen as multiple logical MSHRs. The dynamic number is the number of logical MSHRs, which is referred to as #MSHR. Combining Fig.13 and Fig.14, in addition to the instruction window size IW, dynamic MSHR number #MSHR plays an important role in affecting the cumulative distribution of miss concurrency.

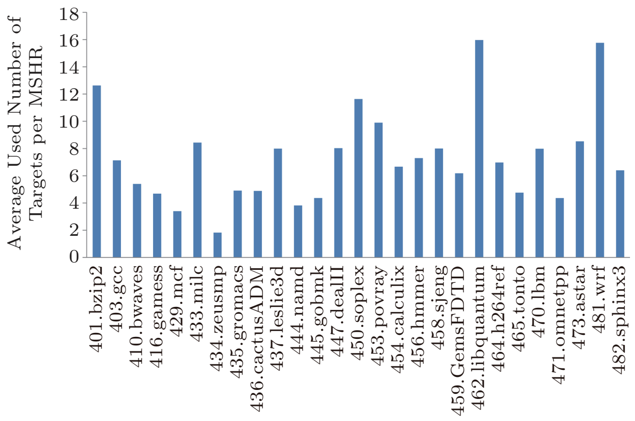

Fig.14. Average reusability of each physical MSHR.

Due to area and power consumption constraints, the static MSHR number cannot be very large. However, the dynamic number can be much larger than the static number, and the MSHR reusability mostly compensates for the limitation of the static MSHR number. Nevertheless, as shown in Fig.14, for an application that the MSHR reusability is low such as 434.zeusmp, the miss concurrency is strictly limited. Note that the number of logical MSHRs is the maximum number of concurrent misses the MSHRs can support. Depending on the locality of the missing accesses, in the best case, miss concurrency can be several dozen times of the physical MSHR number; in the worst case, it would be no more than the physical MSHR number.

Therefore, for the third question, it can be concluded that the locality of missing accesses plays an important role in utilizing MSHRs.

In summary, the C_M theorem reveals that miss concurrency is affected by architecture and application features, which are consistent with the experimental results shown in Fig.13 and the data presented in [10].

6 Pitfalls and Opportunities

Locality and concurrency are two basic and vital factors of memory optimizations. The impact of data locality is well understood. Now, with the five theorems of data access concurrency, we are ready for a better understanding of pitfalls of current memory systems and for finding new opportunities for possible improvements.

Based on the L-C model, both CPU time and data stall time can be reduced via C-AMAT reduction. The memory performance optimization can be achieved by optimizing one or more of the five parameters of C-AMAT. Table 3 shows the techniques which can be leveraged for data stall time reduction considering the impact of these five parameters. The beauty of the theorems is that they quantitatively express the relations between the parameters and the final performance. In other words, the entries in Table 3 can be further evaluated and quantified based on the theorems in our theoretical study.

Take Loop Interchange for an example, because data reusability is improved, MR will be improved. Then according to the theorems, pMR and $pAMP$ will also be improved. In this case, both AMAT and C-AMAT will be reduced.

However, if only the hit concurrency and the miss concurrency are improved, AMAT is not aware of the optimization, but the data stall time model as given by the L-C model shows a remarkable effect due to the C-AMAT reduction.

Fig.15 shows three demonstrative examples. These examples again confirm that concurrency has brought in a new dimension of data stall time reduction, which has not received its deserved attention. In addition to software improvement, hardware methods can also be

Table 3. Techniques Showing Impact on Data Stall Time

Class	Item	$IssueRatio$	MR	pMR	AMP	$pAMP$	C_H	C_M	$AMAT$	$C\text{-}AMAT$
Hardware techniques	Pipelined cache access	+		⊕	−	⊕	⊕	−	−	⊕
	Non-blocking caches	+		⊕	−	⊕		⊕	−	⊕
	Multi-banked caches	+		⊕	−	⊕	⊕	⊕	−	⊕
	Large IW & ROB, run-ahead	+		⊕	−	⊕	⊕	⊕	−	⊕
	SMT	+	−		−	⊕	⊕	⊕	−	⊕
Compiler techniques	Loop interchange		+	⊕					+	⊕
	Matrices blocking		+	⊕					+	⊕
	Data and control dependency related optimization						⊕	⊕		⊕
Application techniques	Copy data into local scalar variables and operate on local copies		+	⊕	+	⊕			+	⊕
	Vectorize the code		+	⊕	+	⊕			+	⊕
	Split structs into hot and cold parts, where the hot part has a pointer to the cold part		+	⊕	+	⊕			+	⊕

Note: + or ⊕ means that the technique improves the factor, − means it hurts the factor, and blank means it has no necessary impact. These notions are used in the same manner as those of Hennessy and Patterson[6].

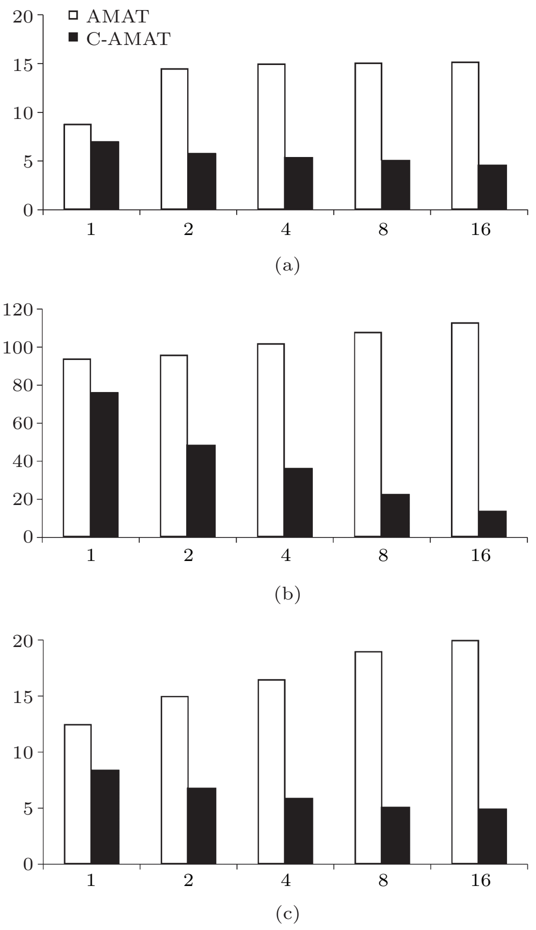

Fig.15. Impacts of concurrency on C-AMAT and AMAT. (a) L1 data cache AMAT and C-AMAT when changing MSHR size. (b) L2 Cache AMAT and C-AMAT when changing core number. (c) L1 data cache AMAT and C-AMAT when changing pipeline issue width.

deployed to achieve application awareness. Since future architectures may present some configurability for users[16], runtime hardware adaption is possible.

Table 3 presents an extended table of Fig.2.11 in page 96 in [6] to summarize the impact of advanced cache optimizations on AMAT and C-AMAT performance parameters. Following [6], in Table 3, the rows represent different optimizations, while the columns are various optimization directions. The optimizations are mostly chosen from the 16 memory-system optimization mechanisms introduced by Hennessy and Patterson[6]. The plus, +, and the minus, −, results are from there as well and are also confirmed in our simulations. Five new columns are added for the four new parameters and C-AMAT. The entries in the newly added columns can be analyzed using the theorems, and actual values can be determined via simulation.

According to the pMR theorem, pMR is the product of $IssueRatio$ expression and MR, and then, based on the impact of $IssueRatio$ and MR as given in [6], the impact results in the pMR column can be derived. Similarly, according to the $pAMP$ theorem, $pAMP$ is the product of $IssueRatio$ expression, MR, and AMP, and then the results in the $pAMP$ column can be derived. According to the C_M theorem, the results for C_M can be derived. Finally, according to the P-M model, the results for data stall time can be derived.

Please notice that the row four of Table 3, hardware techniques: large IW and ROB, run-ahead, is not in the original table of Hennessy and Patterson[6]. The reason is that large IW and ROB, run-ahead[17] only influence C-AMAT and do not influence AMAT. Here we only use the large IW and ROB, run-ahead to demonstrate memory concurrency technologies have not received their deserved attention. The list is far from complete. Large MSHR size, multi-banks, and multi-memory-channels are other "concurrency only" technologies, for examples. In addition, new concurrency only technologies can be developed, if their contribution can be well defined and measured. Table 3 shows the power of C-AMAT in unifying the impact of data locality and data concurrency technologies, in bringing up concurrency only technologies to the spotlight, and in calling new technologies to utilize data concurrency.

There are 38 entries in Table 3 from C-AMAT (marked with "cycle"), while only 21 entries are from AMAT parameters. That shows the importance and potential of data access concurrency, as well as C-AMAT, in cache optimization. Please notice that the table is created based on known cache optimizations which are motivated by data locality optimization. With the publication of this paper, we hope to see the development of concurrency motivated cache optimizations. While 38 is an amazing number, the potential of data access concurrency is clearly not fully explored by Table 3.

Since data stall time is the primary data access overhead, each parameter in the data stall time formulation should be optimized. That is, we should reduce pure miss rate and pure miss penalty and increase hit and pure miss concurrency. However, we find that pitfalls do exist in currency memory systems, which limit the optimization of these parameters. These pitfalls are mostly rooted in the inadequate consideration of data concurrency in system evaluation.

$IssueRatio$ in the theorems is a vital factor that indicates the intensity of hit activities. Whenever a hit access exists, there is no data stall. The larger $IssueRatio$ is, the more intensive the hit access is, and thus the smaller data stall time will be.

As a typical case for benchmarks from SPEC, Fig.16 shows that the memory system has not received any new requests in 70.6% clock cycles. That means, while the SPEC applications are data active, the distribution of their data access intensity is non-unified. This is a pitfall. Following the pure miss concept, we want the hit distributed as evenly as possible.

Assuming memory access issue number per clock cycle is X, recall the C_M theorem, $X = IW \times f_{\text{mem}} \times (1 - f_{\text{data\_dep}} - f_{\text{control\_dep}})$.

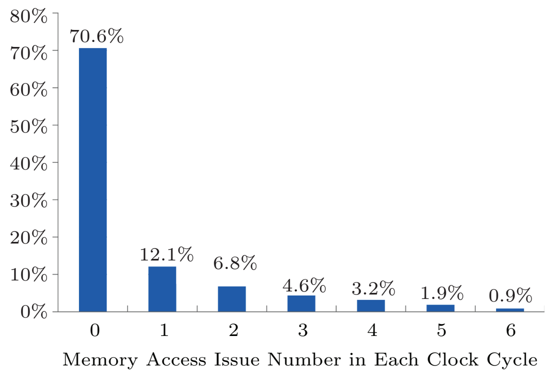

Fig.16. Distribution law of data access issue number in each cycle.

Then, as the probability of a new data access is issued in a clock cycle, $IssueRatio$ in the theorems can be expressed as $IssueRatio = \text{P}\{X \geqslant 1\}$.

For Fig.16, $IssueRatio = \text{P}\{X \geqslant 1\} = 29.4\%$. Because AMAT is larger than 50 cycles and $IssueRatio$, 29.4% is less than the threshold value, 45%, according to the pMR theorem, the pure miss rate cannot be reduced significantly (see Fig.3).

According to the distribution law shown in Fig.16, the expectation of X is

$$E(X) = \sum_{i=0}^{6} i \times \text{P}\{X = i\} = 0.672.$$

This means that there exist average 0.672 new accesses issued per clock cycle. If the memory accesses can be issued evenly, then $IssueRatio$ becomes 67.2%. Then recalling the pMR theorem, the ratio between pure miss rate and conventional miss rate can be reduced significantly (see Fig.3). Moreover, recalling the $pAMP$ theorem, the ratio between pure miss penalty and conventional miss penalty also can be reduced (see Fig.4). If the conventional miss rate and the conventional miss penalty are constant or do not increase significantly, then the final data stall time C-AMAT$_{\text{stall}}$ will be shortened.

With the help of L-C and P-M models, the theorems, and Table 3, new research on reducing data stall time can be carried out. Possible new directions include:

1) Developing techniques to increase data access issue ratio, $IssueRatio$. Run-ahead and very large instruction windows techniques have been proposed for this purpose, yet they have not been adopted in current commercial processors[17].

2) Utilizing both memory locality and concurrency. They are both included in C-AMAT and directly re-

lated to data stall time (see the L-C and P-M models). Their relationship has been shown in the theorems.

3) Paying attention to the control and data dependencies $f_{\text{data\_dep}}$ and $f_{\text{control\_dep}}$. While dependency is not a new research topic, it is still unsolved and breakthrough innovations are needed[18].

Miss concurrency working set, where misses occur during the same time window (see the C_M theorem), is a new concept demanding special attention. It captures both locality and concurrency for misses. Concurrent data accesses with good locality may request the same cache line in a burst manner, which would lead to a pure miss and then cause a data stall. This observation calls for non-localized concurrent data access, which presents a new opportunity for performance optimization against the conventional wisdom of locality.

In summary, the data stall time formulation presented in this work has reshaped the conventional computer hardware and software design principle of "locality is always good" and calls for a joint consideration of locality and concurrency.

7 Related Work

Historically, the memory wall problem was studied from the perspective of AMAT, which includes McKee's original memory wall work in 1994 and later work in 2004[1-2]. Decades later, in 2014, concurrency has become as important as locality, but AMAT has inherent limitations that prevent it from accurately characterizing modern computing systems.

Hennessy and Patterson recognized the inadequacy of AMAT and introduced the term of non-overlapped latency[6]. This is a redefinition of AMAT and is conceptually correct. However, the method to determine overlapped miss is not explored and a measurable formulation is needed.

As data stall becomes more prevalent in many hardware and software design issues, an explicit expression of data stall time becomes an urgent need which has yet to be satisfied. To illustrate this point, we present the following examples.

First, an explicit expression of data stall time is needed to review the memory wall problem. Over the last two decades, processors have been equipped with many concurrency-oriented features. However, AMAT cannot reflect concurrency to illustrate the benefits and losses of these techniques.

Second, (18) is proposed in [7] relating MLP to overall performance, where *Cycles* are total execution cycles, $Cycles_{\text{perf}}$ is the number of execution cycles if the furthest on-chip cache is perfect, $Overlap_{\text{CM}}$ is the fractional overlap of compute cycles with off-chip cycles, *NumMisses* is the number of off-chip accesses, *MissPenalty* is the latency of each off-chip access, and *MLP* is the average memory level parallelism. As far as we know, this is the work most similar to ours. However, there are fundamental differences. The MLP is a special case of our prior work Access Per Cycle (APC)[19-20], which focuses on measurement rather than analysis. With the L-C or P-M model rather than (18), we know more detailed information since we distinguish between the pure and non-pure miss and the miss and hit concurrency.

$$\begin{aligned} Cycles = {} & Cycles_{\text{perf}}(1 - Overlap_{\text{CM}}) + \\ & NumMisses \times MissPenalty/MLP. \end{aligned} \tag{18}$$

Third, scheduling heterogeneous multi-cores through performance impact estimation (PIE)[21] uses an approximate data stall time formula which is not accurate. Better result of the scheduling can be received if our formula is used to make the PIE with less errors.

Fourth, Wang *et al.* used an approximate formula of data stall time as (19) that plays a key role in their bandwidth partitioning[22]. The *APC* in (19) is completely different from the *APC* in our work[7-8]. In our definition of *APC*, the cycles are memory active cycles. They are not CPU cycles as those used in IPC. However, in the *APC* of (19) used in [22], the cycle is CPU cycle. Despite being simpler, (19) cannot present information about locality and concurrency, let alone the trade-off between them. Therefore, if the L-C or P-M model rather than (19) is used, the result of the bandwidth partitioning would be enhanced.

$$IPC = APC/API. \tag{19}$$

Fifth, with locality information, cache can be managed efficiently[23]. While we cheer for the contribution of [23], we find that it only uses locality information and it can be drastically enhanced if guided with the models introduced in this work.

Finally, Iakymchuk and Bientinesi aimed at modeling the performance of linear algorithms without executing either the algorithms or any parts of them[24]. They proposed an analytical model based on the detailed knowledge of the algorithm, though a general runtime measurable data stall time formulation is not given.

In summary, the data stall time formulation and its associated mechanisms presented in this work are promising tools to facilitate existing and future techniques to reduce data stall time and to mitigate the memory wall problem, and Table 3 presents the possible ways to reduce data stall time, through utilizing data access concurrency as well as data locality.

Due to its complexity, we have focused on single core environments in this study. Please recall, C-AMAT can be measured at each layer of a memory hierarchy[10]. In multiple-core systems, C-AMAT can be measured for the private cache for each core respectively, and for the shared L2 cache, a common C-AMAT can be measured for all the cores, and so on. The contention due to shared resource is reflected in the C-AMAT. Therefore, we can use the L-C model or P-M model for each core respectively. While the model can be extended to multi-core environments, we leave an in-depth study of multi-core environments as a future work, and do not discuss it further in this study.

8 Conclusions and Future Work

While the memory wall problem remains a test for system design, the computing landscape has changed from compute-centric to data-centric. This landscape change requires a fundamental rethinking of memory system design and optimization. Various advanced memory technologies and optimizations have been developed to cope with the memory wall problem. Each of them has their strengths, weaknesses, assumptions, and limitations. How to combine these technologies for an optimal memory system design and how to improve existing technology are open research issues in the computing community. In this study, with two models and three theorems, we tried to provide a theoretical foundation for the optimization of modern memory systems.

AMAT (Average Memory Access Time)[1,6] is the conventional model of memory system design, which considers the data locality for modern hierarchical memory systems. C-AMAT (Concurrent AMAT)[10-11] is a newly proposed model which extends AMAT to consider both data locality and data concurrency. In this study, we first derived a C-AMAT based CPU data stall time model, named L-C (Locality-Concurrency) model. Inherited from C-AMAT, the L-C model reflects the combined impact of data locality and concurrency on the final CPU time of the computing system. Next, we introduced the P-M (Pure-Miss) model to formulate CPU time in terms of pure cache misses. It shows the impact of pure miss on CPU performance. Equally important, in proving the P-M model, we delivered an explicit representation of the overlapping of computation and data access ((10)), which opens a door for utilizing this overlapping. Compared with AMAT, C-AMAT has two new performance parameters, concurrent hit and concurrent (pure) miss, and two redefined performance parameters, pure miss rate and pure miss penalty. Three theorems then were developed to characterize and optimize the four parameters, respectively, where concurrent hit and concurrent miss are addressed under one theorem. Finally, simulation tests were conducted to verify and illustrate our theoretical findings, and a table was presented to summarize the impact of advanced cache optimizations on AMAT and C-AMAT performance parameters. Thirty-eight entries in Table 3 are contributed by C-AMAT parameters, while only 21 entries are contributed by AMAT parameters. This shows the importance and potential of data access concurrency, as well as C-AMAT, in cache optimization.

Please notice that the table is created based on known cache optimizations which are motivated by data locality optimization. This study calls for concurrency-based cache optimizations, which may or may not improve data locality. In addition to increasing concurrent hit, the P-M model and its associated computing/data access equation presented in this study have shown two new directions for concurrency-based cache optimizations.

All the results presented in this study are based on C-AMAT. C-AMAT is a rethinking of memory performance from a data-centric view. By the definition of C-AMAT, its cycle is measured in terms of memory active cycle, rather than the conventional CPU cycle. Besides, its cycle is measured in overlapping mode, not the conventional sequential mode[10]. It is a simple formula but a very different one. C-AMAT requests some deep thinking to fully appreciate its value and beauty.

In the future, we intend to continue conducting study in the direction of data access optimization, perform more case studies on domain applications, and explore new optimization mechanisms based on the theoretical findings presented in this study.

Acknowledgment We would like to thank the Scalable Computing Software (SCS) group in the Illinois Institute of Technology and anonymous reviewers for their valuable and professional comments on earlier drafts of this work.

References

[1] Wulf W A, McKee S A. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News*, 1995, 23(1): 20-24.

[2] McKee S A. Reflections on the memory wall. In *Proc. the 1st Conference on Computing Frontiers*, April 2004, p.162.

[3] Borkar S, Chien A A. The future of microprocessors. *Communications of the ACM*, 2011, 54(5): 67-77.

[4] Nikos H, Ippokratis P, Ryan J *et al.* Database servers on chip multiprocessors: Limitations and opportunities. In *Proc. the 3rd Biennial Conference on Innovative Data Systems Research*, Jan. 2007.

[5] Somogyi S, Wenisch T, Ailamaki A *et al.* Spatio-temporal memory streaming. *ACM SIGARCH Computer Architecture News*, 2009, 37(3): 69-80.

[6] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach (5th edition). Morgan Kaufmann, 2011

[7] Chou Y, Fahs B, Abraham S. Microarchitecture optimizations for memory-level parallelism. In *Proc. the 31st International Symposium on Computer Architecture*, June 2004, pp.19-23.

[8] Qureshi M K, Lynch D N, Mutlu O *et al.* A case for MLP-aware cache replacement. *ACM SIGARCH Computer Architecture News*, 2006, 34(2): 167-178.

[9] Moreto M, Cazorla F J, Ramirez A *et al.* MLP-aware dynamic cache partitioning. In *Proc. the 3rd Int. Conf. High Performance Embedded Architectures and Compilers,* Jan. 2008, pp.337-352.

[10] Sun X H, Wang D. Concurrent average memory access time. *IEEE Computer*, 2014, 47(5): 74-80.

[11] Sun X H. Concurrent-AMAT: A mathematical model for Big Data access. *HPC Magazine*. http://www.hpcmagazine.eu/state-of-the-art/c-amat-a-model-for-big-data-access/, May 2014.

[12] Karkhanis T, Smith J E. A day in the life of a data cache miss. In *Proc. the 2nd Workshop on Memory Performance Issues*, May 2002.

[13] Binkert N, Beckmann B, Black G *et al.* The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011, 39(2): 1-7.

[14] Rosenfeld P, Cooper-Balis E, Jacob B. DRAMSim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 2011, 10(1): 16-19.

[15] Spradling C D. SPEC CPU2006 benchmark tools. *ACM SIGARCH Computer Architecture News*, 2007, 35(1): 130-134.

[16] Wu Y, Chen Y, Chen T *et al.* An elastic architecture adaptable to various application scenarios. *Journal of Computer Science and Technology*, 2014, 29(2): 227-238.

[17] Mutlu O, Stark J, Wilkerson C *et al.* Runahead execution: An alternative to very large instruction windows for out-of-order processors. In *Proc. the 9th International Symposium on High-Performance Computer Architecture*, Feb. 2003, pp.129-140.

[18] Ketterlin A, Clauss P. Profiling data-dependence to assist parallelization: Framework, scope, and optimization. In *Proc. the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2012, pp.437-448.

[19] Sun X H, Wang D. APC: A performance metric of memory systems. *ACM SIGMETRICS Performance Evaluation Review*, 2012, 40(2): 125-130.

[20] Wang D, Sun X H. APC: A novel memory metric and measurement methodology for modern memory system. *IEEE Transactions on Computers*, 2014, 63(7): 1626-1639.

[21] Van Craeynest K, Jaleel A, Eeckhout L *et al.* Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *Proc. the 39th Annual International Symposium on Computer Architecture* (*ISCA*), June 2012, pp.213-224.

[22] Wang R, Chen L, Pinkston T M. An analytical performance model for partitioning off-chip memory bandwidth. In *Proc. the 27th IEEE International Symposium on Parallel and Distributed Processing*, May 2013, pp.165-176.

[23] Kurian G, Khan O, Devadas S. The locality-aware adaptive cache coherence protocol. In *Proc. the 40th Annual International Symposium on Computer Architecture*, June 2013, pp.523-534.

[24] Iakymchuk R, Bientinesi P. Modeling performance through memory-stalls. *ACM SIGMETRICS Performance Evaluation Review*, 2012, 40(2): 86-91.

Yu-Hang Liu received his Ph.D. degree in computer science at Beihang University, Beijing, in 2013. He is now a postdoctoral researcher at the Scalable Computing Software Laboratory in Illinois Institute of Technology, Chicago. He is a member of CCF, ACM, and IEEE. His research interests include computer architecture, and memory performance modeling & optimization. He is currently working on multi-core memory scheduling, partitioning 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 at the Illinois Institute of Technology (IIT). He is the director of the Scalable Computing Software Laboratory at IIT, an IEEE fellow, the past chairman of the Computer Science Department of IIT, and is a guest faculty in the Mathematics and Computer Science Division at the Argonne National Laboratory. Before joining IIT, he worked at DoE Ames National Laboratory, at ICASE, NASA Langley Research Center, and at Louisiana State University, Baton Rouge. Dr. Sun's research interests include parallel and distributed processing, memory and I/O systems, software systems for big data applications, and performance evaluation and optimization.