An Analytical Performance Model for Partitioning Off-Chip Memory Bandwidth

Ruisheng Wang* Lizhong Chen† Timothy Mark Pinkston‡
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, California, USA
{ruishenw,lizhongc†,tpink‡}@usc.edu*

***Abstract*—With the emergence of multi-programmed workloads for Chip Multiprocessors (CMP), Quality of Service (QoS) of each co-scheduled application on the CMP is increasingly gaining importance. As more and more applications are consolidated into a single chip to compete for the limited off-chip memory bandwidth, off-chip memory bandwidth partitioning makes an increasing impact on system performance. Although various existing heuristic-based memory scheduling schemes have achieved significant system performance improvement by better partitioning the bandwidth, it is still not clear what are the best ways to partition off-chip bandwidth for improving different system performance objectives.**

The goal of this paper is to understand how off-chip memory bandwidth partitioning affects various system performance objectives. To achieve this goal, we propose an analytical model that is simple yet powerful enough to reveal the relationship between various memory bandwidth partitioning schemes and different system performance objectives. From our model, optimal memory bandwidth partitioning schemes for different system-level objectives are derived. Experimental results from a cycle-accurate full-system simulator show that, for heterogeneous workloads, performance improvements over *No_partitioning/Equal_partitioning* in terms of harmonic weighted speedup, minimum fairness, weighted speedup and sum of IPCs are 20.3%/2.1%, 49.8%/38.7%, 32.8%/7.6% and 64.2%/24%, on average, with our corresponding optimal partitioning schemes (i.e., *Square_root*, *Proportional, Priority_APC*, *Priority_API*), respectively.

***Keywords*-off-chip memory bandwidth; partitioning; analytical model**

I. Introduction

Modern computer systems are becoming increasingly limited by memory performance. While processor performance is expected to double every 18 months, the bandwidth of a memory chip increases by only 10% per year [1]. As memory performance has become increasingly important to overall system performance, the need to carefully schedule memory operations has increased. In general, memory scheduling can affect bandwidth usage in two ways: to increase memory bandwidth utilization and to balance memory bandwidth partitioning among co-scheduled applications.

Early research work related to memory scheduling [2]–[4] mainly focuses on improving bandwidth utilization. For example, prioritizing memory column accesses over row accesses will reduce average DRAM row access delay [2]. Batching and delaying write requests will mitigate DRAM write-to-read turnover delay [5]. As average memory access latency reduces, applications will run faster, which in turn create more off-chip memory requests to occupy more bandwidth.

Recently, with the emergence of multi-programmed workloads for CMP, Quality of Service (QoS) of co-scheduled applications on the CMP has gained increasing attention. Instead of merely increasing bandwidth utilization, various memory scheduling schemes [6]–[13] are proposed to improve the overall CMP system QoS performance by better balancing the memory bandwidth among co-scheduled applications. For example, Fair Queue Memory System [6] partitions the memory bandwidth equally for each application to avoid the starvation problem. Stall-time Fair scheduling [8] improves system fairness further by equalizing the memory-induced slowdowns of co-scheduled applications. Most recent proposed memory scheduling schemes [9], [11], [13] aim to strike a balance between fairness and throughput. While heuristic-based memory bandwidth redistributions among co-scheduled applications in these memory scheduling schemes have improved the system performance significantly, questions like how partitioning of off-chip bandwidth exactly affects system performance and what are the best partitioning schemes for various system performance objectives are still unclear.

Liu, et al., [14] propose an analytical queueing model to derive an optimal memory bandwidth partitioning for weighted speedup. However, the applicability of their model is limited because the assumption of poisson arrival of memory requests is not always valid.

The goal of this paper is to understand and optimize how off-chip memory bandwidth partitioning affects various system performance objectives. To achieve this goal, we propose a unified analytical model that is simple yet powerful enough to reveal the relationship between off-chip memory bandwidth partitioning and various system-level performance objectives. Compared to the prior state-of-art proposal [14] which is limited to the queueing assumption and the single system objective, our model is versatile enough to derive optimal memory bandwidth partitioning schemes for any IPC-based system performance metrics in a more generalized CMP system (i.e., no queueing assumption).

Our contributions are summarized as follows.

1) A unified analytical model is proposed to reveal the relationship between off-chip memory bandwidth par-

titioning and various system performance objectives.
2) From our model, different optimal off-chip bandwidth partitioning schemes are derived to maximize a broad range of system objectives, including throughput-oriented metrics (i.e., sum of IPCs and weighted speedup), fairness, and harmonic weighted speedup.
3) Results from application of the model are obtained by simulation using a cycle-accurate full-system simulator (i.e., GEM5 + DRAMSim2) and show that compared to *No_partitioning/Equal_partitioning* scheme, the optimal memory bandwidth partitioning schemes derived from our model, i.e., *Square_root*, *Proportional, Priority_APC* and *Priority_API*, improve corresponding system performance objectives, i.e., harmonic weighted speedup, minimum fairness, weighted speedup and sum of IPCs under heterogeneous workloads by 20.3%/2.1%, 49.8%/38.7%, 32.8%/7.6% and 64.2%/24%, respectively. Extended experiments on scalability analysis and QoS guaranteed partitioning are also conducted.

The remainder of this paper is organized as follows. Section II provides the background and our motivation. We present our analytical model for optimizing off-chip memory bandwidth partitioning in Section III. We discuss the implementation details about our proposal in IV. Our experimental setup and evaluation are presented in Section V and Section VI, respectively. Related work is discussed in Section VII. We conclude the paper in Section VIII.

II. Background and Motivation

A. Memory Scheduling

As off-chip memory bandwidth becomes the system bottleneck in modern computer systems [1], memory scheduling is increasingly influencing overall system performance. In general, memory scheduling can improve system performance in two different ways: one is to increase memory bandwidth utilization; the other is to better partition memory bandwidth among co-scheduled applications.

1) Increasing Bandwidth Utilization: Early research work related to memory request scheduling policy [2]–[4] mainly focuses on improving bandwidth utilization. For an application, the more memory bandwidth it can occupy (i.e., higher memory Accesses Per Cycle (APC)), the faster it runs (i.e., higher Instructions Per Cycle IPC) since IPC is proportional to APC (i.e., $IPC = APC/API$ while memory Access Per Instruction (API) is constant to memory scheduling).

Memory scheduling can improve bandwidth utilization by leveraging DRAM characteristics. For example, by prioritizing column accesses over row accesses, FR-FCFS [2] maximizes row buffer hit rate. By batching and delaying write operations, Virtual Write Queue [5] mitigates write-to-read turnover delay. Memory scheduling can also reduce bank conflicts [3] which, in turn, improves bank-level parallelism [9]. As average memory access delay reduces with these memory scheduling policies, applications will run faster, which in turn create more off-chip memory requests per unit of time to utilize more bandwidth.

2) Better Bandwidth Partitioning: In the era of chip multiprocessors (CMP), more and more applications are co-scheduled on a single chip to share the off-chip memory bandwidth. This exacerbates the contention problem on shared off-chip memory bandwidth. Thus, sharing memory bandwidth among co-scheduled applications becomes increasingly important to overall system performance. Traditional memory scheduling schemes (e.g., FR-FCFS [2]) improve memory bandwidth utilization through biased scheduling, which will suffer serious starvation problems [6], [7]. Instead of merely increasing bandwidth utilization, various memory bandwidth partitioning schemes [6], [8], [9], [14] are proposed to improve the overall CMP system QoS performance by better balancing the speedups among co-scheduled applications. For example, Fair Queue Memory System [6] partitions the bandwidth equally to each application to avoid starvation. Stall-time Fair scheduling [8] improves system fairness further by equalizing the memory-induced slowdowns of co-scheduled applications.

3) Combination of the Two: Partitioning off-chip memory bandwidth and increasing bandwidth utilization can be integrated together seamlessly. Memory bandwidth partitioning aims to adjust the ratio of the number of accumulated memory requests of each application served over a relatively long period of time (e.g., 10 million CPU cycles), while various mechanisms (e.g., FR-FCFS) for improving memory bandwidth utilization focuses on optimizing the orders of neighboring memory requests. Bandwidth partitioning can be enforced in the context that some particular orders (e.g., column access are prioritized over row access) are also largely maintained. Existing bandwidth partitioning schemes are capable of both optimizing memory requests and enforcing bandwidth shares of co-scheduled applications. For example, FQ-VFTF [6] and DSFQ [8] use priority inversion blocking time (e.g., t_{RAS}) and Starvation Prevention Threshold (SPT), respectively, to maximize row buffer hits over a short interval and enforce bandwidth partitioning in the long run. In this paper, we focus on understanding how memory bandwidth partitioning affects various system performance objectives. Therefore, in the model presented in the next section, we assume that memory bandwidth utilization with different partitioning schemes is constant.

B. Motivation

Although memory bandwidth partitioning cannot increase memory system throughput, it still can improve various system performance objectives significantly by better balancing the speedups of co-scheduled applications for two reasons. On the one hand, different applications have different sensitivities to bandwidth resources, e.g., low-API applications can gain higher IPC improvement than the ones with high APIs by increasing the same amount of off-chip memory bandwidth. On the other hand, different applications can have different impacts on the overall system performance objective. For example, the system performance metric may

be defined in such way that applications with higher priority have more weights than those with lower priority. Thus, allocating more bandwidth to high-priority applications will have more performance gain.

Various memory bandwidth partitioning schemes have been proposed to improve system QoS performance. For example, Nesbit, et al., [6] uses an *Equal* partitioning scheme to avoid starvation, which improves fairness. Liu, et al., [14] derives an optimal memory bandwidth partitioning for weighted speedup based on a queuing theory model. However, these works do not have enough generality to answer the question of what is the best partitioning for all kinds of performance objectives.

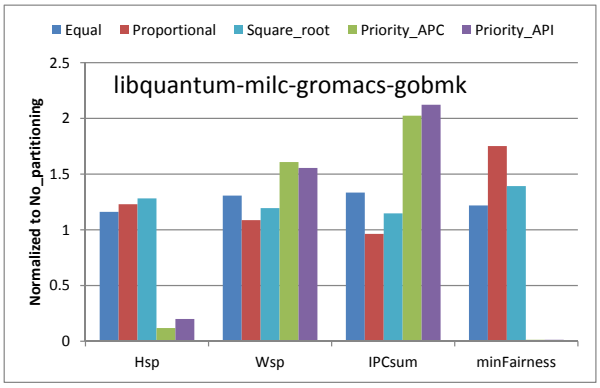

Figure 1. Normalized performance to *No_partitioning* of harmonic weighted speedup, minimum fairness, sum of IPCs and weighted speedup with five bandwidth partitioning schemes of *Equal*, *Proportional*, *Square_root*, *Priority_API* and *Priority_APC*.

Figure 1 shows the potential impact of different memory bandwidth partitioning schemes on different system performance objectives. Four SPEC2006 applications (i.e., libquantum, milc, gromacs and gobmk) with five partitioning schemes including *Equal*, *Proportional*, *Square_root*, *Priority_API* and *Priority_APC* run on a four-core CMP. Detailed descriptions about each partitioning scheme can be found in Section V-D. We compared four different system performance metrics including harmonic weighted speedup, minimum fairness (defined in Eq. (14)), sum of IPCs and weighted speedup among all the partitioning schemes. We use DDR2-400 (i.e., total off-chip bandwidth is 3.2 GB/s) as the off-chip memory system. The detailed configuration information is provided in Section V-B. All performance results are normalized to *No_partitioning*.

From Figure 1, we can see that different partitioning schemes favor different system performance objectives. *Square_root* yields highest harmonic weighted speedup. *Proportional* partitioning has best minimum fairness. *Priority_APC* is best for weighted speedup while *Priority_API* achieves highest sum of IPCs. *Equal* partitioning, which is most commonly used in previous work, can improve the system performance in most cases compared to *No_partitioning*. However, it is not optimal for any system objective that is evaluated. From the figure, we can see that there is no single partitioning scheme optimized for all the system performance objectives. Different partitioning schemes are required to optimize different objectives. It is thus important to understand how off-chip memory bandwidth partitioning affects different system performance objectives.

III. Our Proposed Model

In this section, we propose an analytical performance model for partitioning off-chip memory bandwidth. We first present the general structure of the model. Then we show how to derive a series of optimal partitioning schemes for a broad range of system performance objectives, which includes throughput oriented metrics (i.e., weighted speedup (W_{sp}) and sum of IPCs (IPC_{sum}), fairness and harmonic weighted speedup (H_{sp}). At last, we show our model can be applied to the context where the QoS of some applications are guaranteed while the performance for the rest of best effort services are maximized. The terminology used throughout this paper is listed in Table I.

A. General Structure of the Model

The off-chip memory bandwidth that an application occupies in the system can be measured in terms of memory Accesses Per Cycle (APC). Note that this unit can be easily converted to more commonly used units for off-chip memory bandwidth, such as Bytes Per Second (B/s) if CPU frequency and last level cache line size are given. For example, assume cache line size is 64 bytes and CPU clock frequency is 5Ghz, then 0.01 APC equals 3.2GB/s (i.e., $GB/s = APC \times Cache\_Line\_Size \times CPU\_Frequency$).

The performance of an application can be measured in terms of Instructions Per Cycle (IPC). The memory Access Per Instruction (API) of one application depends the program itself and the input data set. The API value is not affected by bandwidth partitioning. Hence, the impact of memory bandwidth usage of an application on performance (i.e., IPC) can be expressed in a simple equation:

$$IPC = APC/API \tag{1}$$

The API of the application can be measured online, and APC is controlled by the memory bandwidth partitioning. Equation (1) shows the relationship between bandwidth usage and performance. This equation also shows the sensitivity of an application to its off-chip bandwidth occupancy. Generally, the performance of an application with higher API is less sensitive to the bandwidth resource.

Moreover, total memory accesses per cycle ($\sum_{i=1}^{N} APC_{shared,i}$) equals the total off-chip bandwidth utilized (B), so we have

$$\sum_{i=1}^{N} APC_{shared,i} = B \tag{2}$$

In our model, we consider B is a constant value to different memory bandwidth partitioning schemes. By doing this, we can factor out the performance changes caused by bandwidth utilization improvement and focus on understanding

Table I
NOTATIONS

Abbreviation	Meaning
$IPC_{alone,i}$	Instructions Per Cycle that application i can achieve when it runs alone with the dedicated off-chip memory bandwidth
$IPC_{shared,i}$	Instructions Per Cycle that application i can achieve when it shares the off-chip memory bandwidth with other applications in a CMP
$APC_{alone,i}$	Memory Accesses Per Cycle for application i when it runs alone (i.e., $API_i \times IPC_{alone,i}$)
$APC_{shared,i}$	Memory Accesses Per Cycle for application i when it shares the off-chip memory bandwidth with other applications in a CMP (i.e., $API_i \times IPC_{shared,i}$)
API_i	Memory Accesses Per Instruction for application i
β_i	Fraction of total bandwidth assigned to application i (Note that $\sum_{i=1}^{N} \beta_i = 1$)
B	Total utilized off-chip memory bandwidth (i.e., total memory accesses served per cycle)

the system performance changes caused by pure memory bandwidth partitioning. If a performance objective can be expressed in term of IPCs (e.g., sum of IPCs), then it can be translated into APC equations based on Eq. (1). The total bandwidth constraint (i.e., Eq. (2)) and the APC based performance objective function (e.g., sum of IPCs) can be formulated into a constrained optimization problem. By solving this optimization problem, the optimal bandwidth partitioning scheme for a particular performance objective can be obtained.

In the following subsections, we show how to find different optimal partitioning schemes for different system objectives by formulating the constrained optimization problem.

B. Harmonic Weighted Speedup

Harmonic Weighted Speedup (H_{sp}) [15] is a metric that strikes a balance between throughput and fairness. It is defined as the following:

$$H_{sp} = \frac{N}{\sum_{i=1}^{N} \frac{IPC_{alone,i}}{IPC_{shared,i}}} = \frac{N}{\sum_{i=1}^{N} \frac{APC_{alone,i}}{APC_{shared,i}}} \tag{3}$$

We can find maximum H_{sp} expressed in Eq. (3), with the constraint expressed in Eq. (2), by using Lagrange Multipliers:

$$\max H_{sp} = \frac{N \times B}{\left(\sum_{i=1}^{N} \sqrt{APC_{alone,i}}\right)^2} \tag{4}$$

when

$$APC_{shared,i} = B \cdot \frac{\sqrt{APC_{alone,i}}}{\sum_{j=1}^{N} \sqrt{APC_{alone,j}}} \tag{5}$$

The fraction of bandwidth share of application i (β_i) is proportional to its memory access frequency in shared mode (i.e., $APC_{share,i}$). Therefore, we can obtain the optimal memory bandwidth partitioning (i.e., the ratio of bandwidth shared), which is

$$\frac{\beta_i}{\beta_j} = \frac{APC_{shared,i}}{APC_{shared,j}} = \frac{\sqrt{APC_{alone,i}}}{\sqrt{APC_{alone,j}}}$$

This equation shows that the optimal bandwidth share of application i is proportional to the *square root* of its inherent memory access frequency (i.e., $APC_{alone,i}$). Hence, we refer to this partitioning scheme as *Square_root*. From this optimal partitioning scheme, we can see that the optimal bandwidth partitioning for harmonic weighted speedup tends to slightly (but not overly) constrain applications with high miss frequencies, preventing them from dominating the bandwidth usage and starving applications with lower miss frequencies, as has been observed in previous research [6], [8].

We can also have the weighted speedup expression of $Square\_root$ partitioning scheme, which is

$$W_{sp}^{sqrt} = \frac{B}{N} \times \left(\sum_{i=1}^{N} \frac{1}{\sqrt{APC_{alone,i}}}\right)^2 \tag{6}$$

C. Fairness

A CMP system is fair if the speedups of equal-priority applications running together on the CMP system are the same. So, ideal fairness is achieved when:

$$\begin{aligned} \frac{IPC_{shared,i}}{IPC_{alone,i}} &= \frac{IPC_{shared,j}}{IPC_{alone,j}} \\ \Rightarrow \frac{APC_{shared,i}}{APC_{alone,i}} &= \frac{APC_{shared,j}}{APC_{alone,j}} \end{aligned} \tag{7}$$

To achieve this ideal fairness, we can have optimal off-chip bandwidth partitioning:

$$\frac{\beta_i}{\beta_j} = \frac{APC_{shared,i}}{APC_{shared,j}} = \frac{APC_{alone,i}}{APC_{alone,j}}$$

This shows that the optimal bandwidth share of application i is *proportional* to its inherent memory access frequency ($APC_{alone,i}$). Hence, we refer to this partitioning scheme as *Proportional* partitioning.

Although the *Proportional* partitioning scheme is ideal for fairness, it is suboptimal for harmonic weighted speedup and weighted speedup compared to the *Square_root* partitioning scheme. The harmonic weighted speedup and weighted speedup of *Proportional* partitioning are the same, which are

$$H_{sp}^{prop} = W_{sp}^{prop} = \frac{B}{\sum_{i=1}^{N} APC_{alone,i}} \tag{8}$$

From Eq. (8), we can see that the harmonic weighted speedup and weighted speedup of $Proportional$ scheme are worse than those of *Square_root* partitioning scheme (compared

to Eq. (4) and Eq. (6), respectively) according to Cauchy's inequality. Compared to the $Square\_root$ scheme, the *Proportional* scheme tends to allocate more bandwidth resources to bandwidth insensitive applications (i.e., applications with high API), which degrades the overall throughput. This also reflects that different partitioning schemes favor different optimizing objectives.

D. Weighted Speedup

Weighted Speedup [16] aims to measure the overall reduction in execution time, by normalizing each application's performance to its inherent IPC value (i.e., IPC_{alone}).

The optimal partitioning scheme can be found by maximizing W_{sp}, expressed in Eq. (9), subject to the total bandwidth constraint (i.e., Eq. (2)):

$$W_{sp} = \sum_{i=1}^{N} \frac{IPC_{shared,i}}{IPC_{alone,i}}/N = \sum_{i=1}^{N} \frac{APC_{shared,i}}{APC_{alone,i}}/N \quad (9)$$

This optimization problem can be formulated as a fractional Knapsack Problem [17]. Take $APC_{shared,i}$ as the quantity of the item i. Note that $APC_{shared,i}$ can be fractional. The value of each item is $1/\left(N \times APC_{alone,i}\right)$. The maximum quantity that we can carry in the bag is B (i.e., total off-chip bandwidth). Our goal is to maximize the sum of values of the items (i.e., maximize W_{sp}) so that the total quantity must be less than the knapsack's capacity. The fractional knapsack problem can be solved by a greedy algorithm. The best scheme is to get as many items with higher *value* (i.e., $1/\left(N \times APC_{alone,i}\right)$) as possible, in other words, always give the application with lower APC_{alone} higher priority. Note that the maximum bandwidth one application can occupy (i.e., $APC_{shared,i}$) is bounded by $APC_{alone,i}$. This priority-based memory scheduling can be considered as a special form of partitioning, which first allocates enough off-chip memory bandwidth to the application with its maximum occupancy capacity, and then allocates the remaining bandwidth to the application with secondary priority, and so on. Obviously, this scheme causes starvation for applications with higher APC_{alone}, degrading fairness significantly. We refer to this partitioning scheme as ***Priority_APC*** since it prioritizes applications based on their APC_{alone}.

E. Sum of IPCs

When latency is less critical, *Sum of IPCs* can be used to measure the overall system throughput.

To get optimal performance, we can maximize Eq. (10), subject to the constraint of Eq. (2):

$$IPC_{sum} = \sum_{i=1}^{N} IPC_{shared,i} = \sum_{i=1}^{N} \left(\frac{APC_{shared,i}}{API_i} \right) \quad (10)$$

Similar to weighted speedup, this problem can also be formulated as a fractional Knapsack Problem and easily solved by a greedy algorithm. To achieve maximum IPC_{sum}, applications with lower APIs should have higher priority. For this priority-based scheduling, some applications will gain more benefits than others, which implies it has poor fairness properties. We refer to this partitioning scheme as *Priority_API* since it prioritizes the applications based on their API.

F. Discussion

Our model is simple yet powerful enough to reveal the relationship between various memory bandwidth partitioning schemes and different system performance objectives. For example, given a particular memory bandwidth partitioning, we can easily have the bandwidth share of each application (i.e, APC_i). Then we can translate APC_i to IPC_i, based on Eq. (1), and calculate the final IPC-based system performance objective (e.g., weighted speedup). From our analytical model, we can easily analyze how a particular partitioning scheme performs for a particular evaluation metric or derive what is the best bandwidth partitioning for this particular metric. For example, we can have thorough comparison of the performance in terms of harmonic weighted speedup and weighted speedup between $Proportional$ and $Square\_root$ partitioning schemes, based on Eq. (4), (6) and (8), derived from our model.

For our model, four optimal bandwidth schemes (i.e., $Square\_Root$, $Proportional$, $Priority\_APC$ and $Priority\_API$) are derived to maximize four different system objectives, i.e., harmonic weighted speedup, fairness, weighted speedup and sum of IPCs, respectively. In general, for other bandwidth partitioning schemes, the closer it is to our optimal partitioning scheme, the better performance it will achieve.

Our optimal partitioning schemes match the characteristics of various system performance metrics. For example, the priority-based partitioning schemes (i.e., $Priority\_API$ and $Priority\_APC$) correspond to throughput oriented metrics (i.e., sum of IPCs and weighted speedup), which implies possible starvation problems. $Square\_root$ scheme is a balance between $Priority\_APC$ scheme and $Proportional$. Among these three partitioning schemes, $Priority\_APC$ allocates the most bandwidth to low-APC applications while $Proportional$ allocates the least. $Square\_root$'s allocation to low-APC applications is in the middle. This balance reflects that harmonic weighted speedup is a metric that balances throughput and fairness.

Our optimal partitioning schemes also increase the understanding of various system performance metrics. For example, weighted speedup is proposed to overcome the unfairness caused by sum of IPCs for multi-programmed workloads [16]. However, from our optimal partitioning scheme for weighted speedup (i.e., $Priority\_APC$), we can see that a system optimized for weighted speedup can still suffer starvation, which implies that weighted speedup is still not good enough for multi-programmed environments. The comparison among our $Square\_root$, $Priority\_APC$ and $Proportional$ schemes not only qualitatively but also quantitatively show how harmonic weighted speedup strikes a balance between throughput and fairness [15].

Our model is more versatile than a prior state-of-art proposal [14] in two aspects. First, our proposed model can be applied to a more practical CMP system without a queueing assumption. Second, although we only show four optimal bandwidth partitioning for four system objectives, our proposed model can be used for deriving optimal bandwidth partitioning for any IPC based system performance metrics.

G. QoS Guarantee

In a more general case, applications in a CMP system can be divided into two groups: QoS Guarantee and Best Effort. For QoS Guaranteed workload, the performance of applications needs to be guaranteed by allocating sufficient resources. Meanwhile, applications in the Best Effort group should use the rest of the bandwidth resources efficiently to maximize their overall performance. Our model can be used to efficiently allocate off-chip memory bandwidth resource in this case. We guarantee the performance of applications in the QoS-Guaranteed group by allocating enough bandwidth (i.e., $B_{QoS} = IPC_{target} \times API$), and maximize system performance (e.g., weighted speedup) for the applications in the Best Effort group by using the rest of the bandwidth (B_{BE}). By combining the bandwidth constraint (i.e., Eq. (11)) and the system objectives (e.g., Eq. (3), (7), (9) and (10)), we can find the optimal performance for best-effort applications with the bandwidth of B_{BE}.

$$\sum_{i=1}^{N_{BE}} APC_{shared,i}^{BE} = B_{BE} = B - B_{QoS} \tag{11}$$

where N_{BE} is the number of applications in the Best Effort group.

IV. Implementation Details

A. CMP Architecture

We assume a typical CMP architecture throughout this paper. Both L1 and L2 caches are private[1]. Off-chip memory bandwidth is shared by multiple cores on the chip. Throughout this paper, we assume that each core runs only one application.

B. Bandwidth Enforcement Mechanism

The enforcement mechanism for our off-chip memory bandwidth partitioning is slightly modified from DRAM Start-Time Fair (DSTF [7]). Basically, we calculate a start-time tag when the packet arrives, the memory scheduler chooses the packet with smallest start-time tag to be served first. The start-time tag is calculated as follows.

Let S_i^a be the start-time tag of the i^{th} memory request of application and β^a be the fraction of off-chip bandwidth that application a is assigned. Our start-time tags are

$$S_i^a = S_{i-1}^a + 1/\beta^\alpha$$

Note that this start-time tag calculation is different from [7]. Our start-tag only depends on the start-time tag of the last served packet (S_{i-1}^a) and service rate β^α. It does not depend on packet arrival time. Our modification allows one application to use more bandwidth share if it did not receive enough allocated bandwidth fraction during the previous period. This modification will allow low memory intensive applications to more easily achieve their bandwidth share.

C. APC_{alone} Profiling

Our partitioning schemes need $APC_{alone,i}$ to calculate the bandwidth share of each application. We can get $APC_{alone,i}$ as follows:

$$APC_{alone,i} = IPC_{alone,i} \times API_i = \frac{N_{accesses,i}}{T_{cyc,alone,i}} \tag{12}$$

where $N_{accesses,i}$ is the number of memory accesses (both reads and writes) served for application i and $T_{cyc,alone,i}$ is the number of cycles for application i running in a standalone context with these $N_{accesses,i}$ memory requests served. $N_{accesses,i}$ can be counted online easily.

We can find $T_{cyc,alone,i}$ through

$$T_{cyc,alone,i} = T_{cyc,shared,i} - T_{cyc,interference,i} \tag{13}$$

where $T_{cyc,shared,i}$ is the number of cycles for application i running in a shared context with the $N_{accesses,i}$ memory requests served and can be counted online. $T_{cyc,interference,i}$ is the memory interference delay and counted for each application. Memory interference occurs when an application's memory request is blocked by the requests from another application. The memory interference, including DRAM bus and bank conflict, can be detected as described in [8], [19]. At each cycle, if interference for application i is detected, we increment $T_{cyc,interference,i}$ by one. So, in total, three counters (i.e., $N_{accesses,i}$, $T_{cyc,shared,i}$ and $T_{cyc,interference,i}$) are added for each application.

Note that our profiled $APC_{alone,i}$ is an approximation. To exactly profile the standalone performance of co-scheduled applications in a shared CMP context is still challenging [20]. However, the inaccuracy existing in $APC_{alone,i}$ estimation will not affect the efficiency of our partitioning scheme since $APC_{alone,i}$ is just a reference value and we use the same values (i.e., our estimated $APC_{alone,i}$) in both our partitioning calculation and final performance result calculation. In a more general QoS-enabled system, the operating system may specify performance objectives along with some reference values (e.g., $IPC_{ref,i}$ or $APC_{ref,i}$). In such a scenario, our partitioning framework can take these reference values (e.g., $IPC_{ref,i}$ or $APC_{ref,i}$) as input to avoid estimation of $APC_{alone,i}$.

$APC_{alone,i}$ is profiled periodically (e.g., every 10 million cycles). When an application's behavior changes, its

[1]Note that our model can also be extended to a partitioned shared L2 CMP system. In a shared L2 CMP, an application's API will be affected by its L2 cache capacity share. Hence, we can extend our model by replacing API_i with $API_{shared,i}$ and $API_{alone,i}$ properly. Both $API_{shared,i}$ and $API_{alone,i}$ are constant to memory bandwidth partitioning and can be obtained online with a non-invasive resource profiler [18].

$APC_{alone,i}$ will be updated (based on Eq. (12) and (13)) correspondingly. Our partitioning schemes will change an application's bandwidth share correspondingly when $APC_{alone,i}$ (either itself or its co-scheduled applications) changes.

V. Experimental Setup

A. Performance Metrics

We evaluate the efficacy and versatility of our model by using four general optimization targets, including (1) sum of IPCs, (2) weighted speedup, (3) fairness, and (4) harmonic weighted speedup. We also evaluate the situation where strict QoS needs to be guaranteed. The expression of harmonic weighted speedup, weighted speedup, and sum of IPCs can be found in Eq. (3) (9) and (10), respectively. To measure the fairness, we use minimum Fairness criteria [21], which is defined as in Eq. (14):

$$MinF = N \times \min_{i}^{N} \left\{ \frac{IPC_{shared,i}}{IPC_{alone,i}} \right\} \tag{14}$$

The minimum fairness of the system depends on minimum speedup of co-scheduled applications. If all the applications have at least $1/N$ speedup (i.e., $minF \geq 1$), we say the system achieves minimum fairness [21]. Note that minimum speedup is also equivalent to maximum slowdown metrics [22].

B. Architecture

We evaluate our model using simulation models of a four-core CMP with a DDR2-400 memory subsystem. We use a cycle-accurate full-system simulator GEM5 [23] to model the out-of-order cores and the on-chip L1/L2 caches. The DRAM subsystem is modeled by DRAMSim2 [24]. Table II summarizes the baseline configuration for our four-core CMP system.

Table II
Baseline system configuration

Core	5 GHz out of order processor
	Decode/Issue/Execute/Retire up to 8 instructions
	192-entry reorder buffer
Front End	16-bit BTB tag, 4K-entry BTB
	Tournament branch predictor
Caches	L1 I-cache/D-cache: 32KB, 2-way, 1 ns, 64B line
	Private unified L2: 256KB, 8-way, 5 ns, 64B line
DRAM	200 MHz bus cycle, 8 GB DDR2-PC3200
	Close page policy
	8B-wide data bus
	Latency: 12.5-12.5-12.5ns (tRP-tRCD-CL)
	Address Mapping: channel/row/col/bank/rank
	32 DRAM banks

We fast-forward each application by 500 million instructions (in atomic mode) to warm-up the cache and then run 10 million cycles to profile APC_{alone} of each application, and finally run 10 million cycles to collect final performance results.

Table III
Benchmark Classification

Name	Type	$APKC_{alone}$	$APKI$	Intensity
lbm	FP	9.38517	53.1331	high
libquantum	INT	6.91693	34.1188	middle
milc	FP	6.87143	42.2216	middle
soplex	FP	6.05614	37.8789	middle
hmmer	INT	5.29083	4.6008	middle
omnetpp	INT	5.18984	30.5707	middle
sphinx3	FP	4.88898	13.5657	middle
leslie3d	FP	4.3855	7.5847	middle
bzip2	INT	3.93331	5.6413	low
gromacs	FP	3.36604	5.1976	low
h264ref	INT	3.04387	2.2705	low
zeusmp	FP	2.42424	4.521	low
gobmk	INT	1.91485	4.0668	low
namd	FP	0.61975	0.428	low
sjeng	INT	0.559802	0.7906	low
povray	FP	0.553825	0.6977	low

C. Benchmarks

1) Benchmark Classification: We use the SPEC CPU 2006 benchmarks with reference input for our evaluation. Each benchmark was compiled using GCC or GFORTRAN with the -O3 option. We use Simpoint [25] to select a representative portion of applications for our experiments.

We classify benchmarks into three categories based on their inherent memory access frequency (i.e., APC_{alone}). We refer to a benchmark as highly memory intensive if its memory Accesses Per Kilo Cycle ($APKC_{alone}$) is greater than 8. If the $APKC_{alone}$ value is greater than 4 but less than 8, we say the benchmark has middle memory intensity. If the $APKC_{alone}$ value is less than 4, we refer to it as low intensive. This classification is based on measurements made when each benchmark runs alone. Table III shows the characteristics of the benchmarks that appear in the evaluated workloads.

2) Workload Construction: In our four-core experiments, each workload includes four applications. Each core runs one application. The mixed workloads are constructed into two categories: heterogeneous and homogeneous. For the heterogeneous workload, applications are picked from different memory-intensity groups, while for the homogeneous workload, applications are from the same memory-intensity group. We define the heterogeneity as the Relative Standard Deviation (RSD) of APC_{alone}s of co-scheduled applications. We say a workload is heterogeneous if its heterogeneity is greater than 30. Otherwise, if its heterogeneity is smaller than 30, we say it is homogeneous.

The mixed workloads used in our evaluation are shown in Table IV.

D. Partitioning Schemes

We evaluate seven partitioning schemes in our experiments as follows:

No_partitioning: This scheme does not manage off-chip memory bandwidth resources with partitioning. The memory controller serves all the memory requests based on a First Come First Served (FCFS) Policy.

Table IV
WORKLOAD CONSTRUCTION

workload	benchmark	heterogeneity(RSD)
homo-1	libquantum-milc-soplex-hmmer	12.27
homo-2	libquantum-milc-soplex-omnetpp	13.02
homo-3	hmmer-gromacs-sphinx3-leslie3d	18.55
homo-4	hmmer-gromacs-bzip2-leslie3d	19.16
homo-5	h264ref-zeusmp-bzip2-gromacs	19.74
homo-6	h264ref-zeusmp-gobmk-gromacs	24.06
homo-7	h264ref-zeusmp-gobmk-bzip2	29.71
hetero-1	milc-soplex-zeusmp-bzip2	41.93
hetero-2	soplex-hmmer-gromacs-gobmk	45.10
hetero-3	libquantum-soplex-zeusmp-h264ref	47.92
hetero-4	lbm-soplex-h264ref-bzip2	50.31
hetero-5	libquantum-milc-gromacs-gobmk	52.99
hetero-6	lbm-libquantum-gromacs-zeusmp	58.31
hetero-7	lbm-milc-gobmk-zeusmp	69.84

Equal: This scheme assigns an equal fraction of off-chip memory bandwidth to each individual application. This scheme is proposed in [6]. The fraction of the total off-chip memory bandwidth that application i is assigned to (β_i) is $\beta_i = \frac{1}{N}$.

2/3_Power: In this partitioning scheme, the fraction of off-chip memory bandwidth (β_i) that application i is assigned to is proportional to the two-thirds power of its inherent memory access frequency (i.e., $APC_{alone,i}$.), which is $\beta_i = \frac{(APC_{alone,i})^{2/3}}{\sum_{j=1}^{N}(APC_{alone,j})^{2/3}}$. This partitioning scheme is proposed in [14] (i.e., Eq. (19) in [14]) as the best partitioning scheme for weighted speedup based on their queuing model.

Proportional: In this partitioning scheme, the fraction of off-chip memory bandwidth (β_i) that application i is assigned to is proportional to its inherent memory access frequency ($APC_{alone,i}$). This is the best partitioning scheme for fairness. The fraction of the amount of total off-chip bandwidth that application i should be assigned to (β_i) is $\beta_i = \frac{APC_{alone,i}}{\sum_{j=1}^{N} APC_{alone,j}}$.

Square_root: In this partitioning scheme, the fraction of off-chip memory bandwidth (β_i) that is assigned to each application is proportional to the square root of their inherent memory access frequencies (i.e., APC_{alone}). This is the optimal partitioning scheme for harmonic weighted speedup. The fraction of total off-chip bandwidth that application i is assigned (β_i) is $\beta_i = \frac{\sqrt{APC_{alone,i}}}{\sum_{j=1}^{N}\sqrt{APC_{alone,j}}}$.

Priority_API: This scheme prioritizes the memory requests from applications with lower API over ones with higher API. This scheme is best for sum of IPCs.

Priority_APC: This scheme priorities the memory requests from applications with lower APC_{alone} over ones with higher APC_{alone}. This scheme is best for weighted speedup.

VI. EVALUATION

A. Performance

Figure 2 shows the performance comparison of six off-chip memory bandwidth partitioning schemes (i.e., *Equal*, *Proportional*, *Square_root*, *2/3_power*, *Priority_APC* and *Priority_API*) in terms of four system performance objectives (i.e., harmonic weighted speedup, fairness, weighted speedup and sum of IPCs). All the performance results are normalized to *No_partitioning*. From Figure 2, we can see that different partitioning schemes favor different system objectives.

Generally, for heterogeneous workloads, due to the large variety in applications' sensitivities to the off-chip memory bandwidth resource, the performance differences among different partitioning schemes are large, e.g., *Priority_API* has 50% more performance than *Proportional* in terms of sum of IPCs. The performance of homogeneous workloads are less diverse in terms of all performance metrics with different partitioning schemes because all partitioning schemes are similar. For example, if two applications have exact same inherent memory access frequency (APC_{alone}), there will be no difference among *Equal*, *Proportional*, *Square_root* partitioning schemes. In the rest of this section, without explicit mention, all the data is from the measurement of heterogeneous workloads.

For *No_partitioning*, high API applications tend to occupy more off-chip bandwidth resources to starve out low API applications. Since high API applications have low sensitivity to the bandwidth, *No_partitioning* has poor overall throughput.

Equal partitioning has moderate performance improvements in harmonic weighted speedup (17.7%), weighted speedup (23.4%) and sum of IPCs (32.4%) over *No_partitioning*. These performance improvements are because it increases the overall throughput by allocating more bandwidth to low API applications. It has relatively poor fairness since the speedups of high API applications are degraded, which causes unbalanced speedups between high API and low API applications. Note that *Equal* partitioning is not the optimal partitioning scheme for any of the objectives that are evaluated.

Square_root partitioning yields best performance (20.3%) in terms of harmonic weighted speedup, as is expected. It also has moderate performance improvements in terms of both fairness (26.7%) and throughput, e.g., sum of IPC (16.2%) and weighted speedup (16.2%), which implies harmonic weighted speedup itself is a metric that balances both fairness and throughput.

Proportional partitioning is best for the fairness metric as expected. It has the worst performance in terms of throughput-oriented metrics (i.e., IPC_{sum} and W_{sp}) since it does not favor low API applications to achieve high IPC. Note that *Proportional* partitioning is different from *No_partitioning*, which implies the bandwidth that an application occupies naturally is not exactly proportional to its inherent memory access frequency (APC_{alone}).

2/3_power partitioning scheme partitions bandwidth in between *Square_root* and *Proportional,* which implies its performance is also between those two. For example, in terms of fairness, it is better than *Square_root* and worse than *Proportional*. In terms of harmonic weighted speedup, it is higher than *Proportional* but lower than *Square_root*.

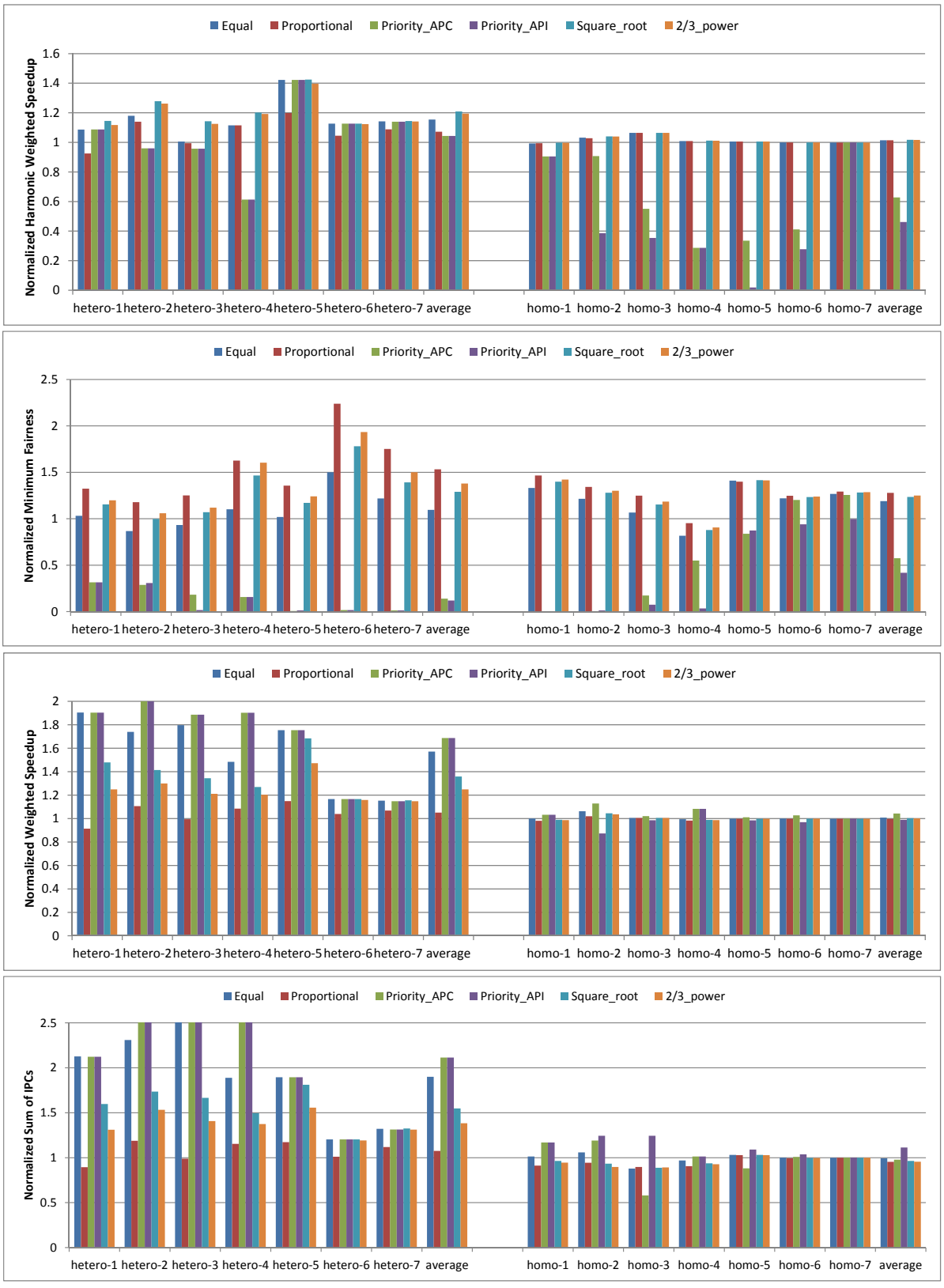

Figure 2. Normalized performance to *No_partitioning* of (a) harmonic weighted speedup (b) minimum Fairness (c) weighted speedup and (d) sum of IPCs with six partitioning schemes of *Equal*, *Proportional*, *Square_root*, *2/3_power*, *Priority_APC and Priority_API*.

Although the *2/3_power* scheme is expected to produce highest weighted speedup in [14] based on their model, our experimental results show that this is not the case. *2/3_power* only achieves 84.4% of *Priority_APC* in terms of weighted speedup. The main reason comes from different assumptions. In [14], the memory access frequency (denoted as MA in [14]) of an application is assumed to stay unchanged no matter how fast/slow an application runs, which leads to limited performance gain (W_{sp}) by always favoring applications with low APC. However, in our model, the invariant is API. Memory access frequency (APC_{share}) will change based on the application execution speed (IPC_{share}). When an application runs faster, its memory access frequency will become higher correspondingly. This matches reality more closely. While [14] presumes the optimum bandwidth partitioning for weighted speedup should slightly but *not overly constrain* applications with high miss frequencies, our model indicates that strictly prioritizing low-APC applications over high-APC (i.e., *should overly constrain* applications with high miss frequencies) is the best scheme for weighted speedup. Although weighted speedup is proposed to overcome the unfairness caused by sum of IPCs for multi-programmed workloads [16], our results show that W_{sp} is a throughput-oriented metric intrinsically and is still not good enough for multi-programmed environments. Systems that are optimized for weighted speedup can still suffer from starvation. Compared to the model in [14], our conclusion is valid due to the use of more realistic assumptions.

As is expected, for priority-based partitioning, *Priority_API* and *Priority_APC* achieve highest performance for sum of IPCs and weighted speedup, respectively. However, they yield very poor performance for fairness and harmonic weighted speedup since starvation happens. Due to the strict priority policy, memory requests from applications with high APC_{alone} or API may not get served at all. *Priority_API* and *Priority_APC* achieve the same result for heterogeneous workload because applications with higher API are applications with higher APC_{alone}. However, for homogeneous workloads, high API applications may not be high APC_{alone} applications, for example, hmmer has higher APC_{alone} but lower API than leslie3d.

In summary, our optimal partitioning schemes achieve better performance for corresponding performance objectives. Among the rest of partitioning schemes, the closer they are to the optimal scheme, the better results that can be achieved.

B. QoS Guarantee

In this experiment, we evaluate two mixed workloads. $Mix-1$ is constructed from four applications: lbm, libquantum, omnetpp and hmmer. $Mix-2$ has h264ref, zeusmp, leslie3d and hmmer. They run on an four-core CMP. For both workloads, the objective is to guarantee the IPC of hmmer to be 0.6[2] and maximize the performance of the rest applications.

[2]We choose this value empirically to ensure that this IPC target is reachable through allocating sufficient memory bandwidth.

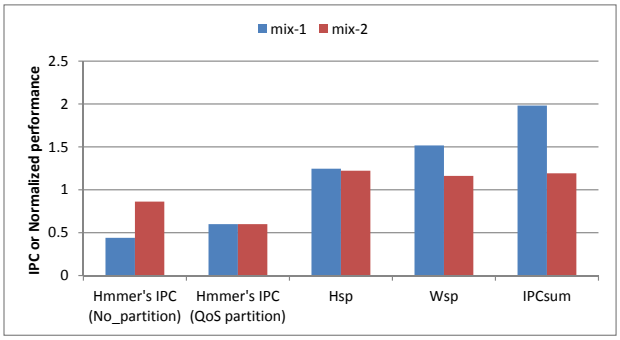

Figure 3. The IPCs of QoS Guaranteed application and normalized performance (i.e., harmonic weighed speedup, weighted speedup and sum of IPCs) of Best Effort applications in two mixed workloads.

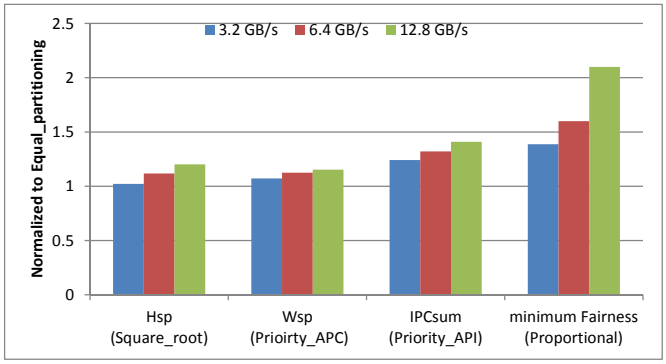

Figure 4. Normalized Performance to *Equal* partitioning of heterogeneous workloads in terms of harmonic weighted speedup, weighted speedup, sum of IPCs and minimum fairness with total bandwidth scaling from 3.2 GB/s to 12.8 GB/s.

Figure 3 shows the IPCs of the QoS Guaranteed application (i.e. hmmer) and the normalized performance of Best Effort applications (i.e. rest of three applications) in these two mixed workloads. The first two groups shows the performance (IPC) of hmmer with *No_partitioning* and QoS Guaranteed partitioning respectively. The remaining three groups show the performance results (i.e., normalized to *No_partitioning*) of the best effort applications in terms of harmonic weighted speedup, weighted speedup and sum of IPCs, respectively. From the figure, we can see that with *No_partitioning* scheme, the performance of hmmer is not guaranteed, i.e., either is below or above 0.6. Our QoS Guaranteed partitioning maintains hmmer's IPC at 0.6 in both workloads. Additionally, we can see that the performance of the best effort applications have been largely improved compared to *No_partitioning*.

C. Scalability

Figure 4 shows how performance changes when the off-chip bandwidth scales from 3.2GB/s to 12.8GB/s. Note that we change bandwidth by only changing the memory bus frequency, while the latency related parameters are not changed (i.e., tRP-tRCD-CL is 12.5-12.5-12.5ns for all bandwidths). With the scaling of bandwidth, the number

of cores is also scaled proportionally (4, 8, 16 cores for 3.2, 6.4 and 12.8GB/s, respectively). The heterogeneous workloads in Table IV are scaled with 1, 2, 4 copies of each application for 3.2, 6.4 and 12.8GB/s, respectively. For each performance objective, the performance results are from its corresponding optimal partitioning scheme and normalized to Equal partitioning.

From the figure, we can see that with the increase in bandwidth, the improvements of all metrics (i.e., H_{sp} , W_{sp}, IPC_{sum} and minFairness) increase. The reason is that with increase in bandwidth, the workloads become more heterogeneous, i.e., the APC_{alone} of bandwidth bounded applications (e.g., lbm) increases much faster than that of latency bounded applications (e.g., leslie3d). For example, as the bandwidth scales from 3.2GB/s to 6.4GB/s, the APC_{alone} of lbm goes from 9.38154 to 17.2329 (i.e., increases by 83.7%) while the APC_{alone} of leslie3d only goes from 4.38538 to 5.46125 (i.e., increases 24.5%). In a more heterogeneous environments, differences between equal partitioning and various optimal partitioning become larger, which implies the normalized performance (to equal partitioning) of the optimal partitioning schemes become higher. Hence, our optimal partitioning schemes scale well as the number of cores and off-chip memory bandwidth increases.

VII. Related Work

Generally, research work related to scheduling policies for memory requests can be categorized into two groups: (1) to increase the bandwidth utilization by reordering various types of memory requests and (2) to balance the performance of co-scheduled applications in a shared CMP context by partitioning off-chip memory bandwidth.

Improving Memory Bandwidth Utilization: Proposals from the first category (like FR-FCFS [2] and Virtual Write Queue [5]) focus on improving memory bandwidth utilization by considering the characteristics of modern DRAM systems. The system throughput will increase if off-chip memory bandwidth utilization is improved. While FR-FCFS [2] scheduling reduces row buffer miss delay, Virtual Write Queue [5] mitigates write-to-read turnover delay. Both works improve the bandwidth utilization by reducing average memory access delay. Minimalist Open-page policy [13] can increase bandwidth utilization by balancing locality and parallelism. Previous work in this category focus only on improving overall system throughput without considering the Quality of Service (QoS) of each individual application (e.g., fairness) in a shared CMP context.

Balancing Memory Bandwidth Partitioning: With the emergence of multi-programmed workloads for CMP, Quality of Service of each independent workload is increasingly important. Various off-chip memory bandwidth partitioning schemes have been proposed to improve fairness among co-scheduled applications [6], [8], [9], [14]. Nesbit, et al., [6] propose to divide bandwidth equally among all the applications to avoid starving low memory intensive workloads. Mutlu, et al., [8] propose Stall-time Fair Memory Scheduler (STFM) to equalize the memory slowdowns experienced by co-scheduled applications. Most recent works related to memory scheduling focus on improving both throughput and fairness [9], [11]. Parallelism-Aware Batch-Scheduling (PARBS) [9] tries to improve overall QoS objectives without adversely effecting individual workload's efficiency. Thread Cluster Memory (TCM) [11] scheduling improves both system performance and fairness by clustering different types of threads together. Self-Optimizing Memory Controllers [26] and MORSE [27] use a machine learning approach to select best scheduling sequence. Although those heuristic-based memory scheduling schemes gain system performance by distributing bandwidth among co-scheduled applications in a better way, they do not explicitly specify how much bandwidth should be allocated to each application. Therefore, it is still unclear how bandwidth partitioning affect system performance and what are the best partitioning schemes for different performance metrics.

Liu, et al., [14] propose an analytical model to derive an optimal memory bandwidth partitioning for weighted speedup. This work is most related to ours. The model proposed in [14] assumes poisson arrival of memory requests, which limits its applicability to more realistic scenarios. The goal of our paper is to understand how best to partition off-chip memory bandwidth in terms of various system performance objectives. Our proposed model is general enough to be applied in more general scenarios (i.e., no queueing assumption) and to be used to optimize multiple system performance metrics.

VIII. Conclusion

The goal of this paper is to understand and optimize how off-chip bandwidth partitioning affects different system performance objectives. We present an analytical model to derive optimal memory bandwidth partitioning schemes for various system QoS objectives. From the model, we derive four optimal off-chip memory bandwidth partitioning schemes, including *Square_root*, *Proportional*, *Priority_APC*, and *Priority_API* for four different system-level performance objectives, which are harmonic weighted speedup, fairness, weighted speedup, and sum of IPCs. Our model is applied and verified by experimental results from a cycle-accurate full-system simulator (GEM5 with DRAMSim2). Experimental results show that, for heterogeneous workloads, performance improvements over *No_partitioning/Equal_partitioning* in terms of harmonic weighted speedup, minimum fairness, weighted speedup and sum of IPCs are 20.3%/2.1%, 49.8%/38.7%, 32.8%/7.6% and 64.2%/24%, on average, with our corresponding optimal partitioning schemes (i.e., *Square_root*, *Proportional, Priority_APC*, and *Priority_API*), respectively.

Acknowledgment

We sincerely thank the anonymous reviewers for their helpful comments and suggestions. This research was supported, in part, by the National Science Foundation (NSF), grant CCF-0946388.

REFERENCES

[1] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for cmp scaling," in *Proceedings of the 36th International Symposium on Computer architecture*, ISCA '09. New York, NY, USA: ACM, 2009, pp. 371–382.

[2] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proceedings of the 27th International Symposium on Computer Architecture*, ISCA '00. New York, NY, USA: ACM, 2000, pp. 128–138.

[3] S. Rixner, "Memory controller optimizations for web servers," in *Proceedings of the 37th International Symposium on Microarchitecture*, MICRO 37. Washington, DC, USA: IEEE Computer Society, 2004, pp. 355–366.

[4] I. Hur and C. Lin, "Adaptive history-based memory schedulers," in *Proceedings of the 37th International Symposium on Microarchitecture*, MICRO 37. Washington, DC, USA: IEEE Computer Society, 2004, pp. 343–354.

[5] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter, and L. K. John, "The virtual write queue: Coordinating dram and last-level cache policies," in *Proceedings of the 37th International Symposium on Computer Architecture*, ISCA '10. New York, NY, USA: ACM, 2010, pp. 72–82.

[6] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair queuing memory systems," in *Proceedings of the 39th International Symposium on Microarchitecture*, MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 208–222.

[7] N. Rafique, W.-T. Lim, and M. Thottethodi, "Effective management of dram bandwidth in multicore processors," in *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 245–258.

[8] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *Proceedings of the 40th International Symposium on Microarchitecture*, MICRO 40. Washington, DC, USA: IEEE Computer Society, 2007, pp. 146–160.

[9] ——, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in *Proceedings of the 35th International Symposium on Computer Architecture*, ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 63–74.

[10] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, HPCA '10, 2010, pp. 1 –12.

[11] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proceedings of the 43rd International Symposium on Microarchitecture*, MICRO 43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 65–76.

[12] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *Proceedings of the 44th International Symposium on Microarchitecture*, MICRO-44 '11. New York, NY, USA: ACM, 2011, pp. 374–385.

[13] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A dram page-mode scheduling policy for the many-core era," in *Proceedings of the 44th International Symposium on Microarchitecture*, MICRO-44 '11. New York, NY, USA: ACM, 2011, pp. 24–35.

[14] F. Liu, X. Jiang, and Y. Solihin, "Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance," in *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, HPCA '10, jan. 2010, pp. 1 –12.

[15] K. Luo, J. Gummaraju, and M. Franklin, "Balancing thoughput and fairness in smt processors," in *Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*, ISPASS '01, 2001, pp. 164 –171.

[16] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreaded processor," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX. New York, NY, USA: ACM, 2000, pp. 234–244.

[17] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, 1st ed. Springer, 12 2010.

[18] D. Kaseridis, J. Stuecheli, J. Chen, and L. John, "A bandwidth-aware memory-subsystem resource management using non-invasive resource profilers for large cmp systems," in *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, HPCA '10, 2010, pp. 1 –11.

[19] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 335–346.

[20] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "Mise: Providing performance predictability and improving fairness in shared main memory systems," in *Proceedings of the 19th International Symposium on High Performance Computer Architecture*, HPCA '13, 2013.

[21] H. Vandierendonck and A. Seznec, "Fairness metrics for multi-threaded processors," *Computer Architecture Letters*, vol. 10, no. 1, pp. 4 –7, 2011.

[22] R. Gabor, A. Mendelson, and S. Weiss, "Service level agreement for multithreaded processors," *ACM Transactions on Architecture and Code Optimization*, vol. 6, no. 2, pp. 6:1–6:33, Jul. 2009.

[23] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[24] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, 2011.

[25] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program analysis," *Journal of Instruction Level Parallelism*, 2005.

[26] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Proceedings of the 35th International Symposium on Computer Architecture*, ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 39–50.

[27] J. Mukundan and J. Martinez, "Morse: Multi-objective reconfigurable self-optimizing memory scheduler," in *Proceedings of the 18th International Symposium on High Performance Computer Architecture*, HPCA '12, feb. 2012, pp. 1 –12.