An Analytical Performance Model for Co-Management of Last-Level Cache and Bandwidth Sharing

Taecheol Oh, Kiyeon Lee, and Sangyeun Cho
Computer Science Department, University of Pittsburgh
Pittsburgh, PA 15260, USA

***Abstract*—Processor cores in a chip multiprocessor (CMP) typically share a large last-level cache and the off-chip memory bandwidth. Previous studies demonstrate that explicit cache capacity and off-chip bandwidth partitioning can yield better overall system performance than without partitioning. However, little work has been done to study the interaction between cache capacity partitioning and off-chip bandwidth allocation. This paper develops a hybrid analytical model that takes into account the two partitioning problems together in order to capture their inter-dependence. With an elaborate case study, we show that an optimal resource management strategy would require a coordinated allocation of the cache and the off-chip bandwidth resources.**

***Keywords*—Chip multiprocessor (CMP), performance modeling, simulation, resource sharing.**

I. Introduction

Chip multiprocessor (CMP) architectures have become the mainstream computing platform. To improve system performance and reduce the performance volatility of individual threads, explicit last level cache and off-chip bandwidth partitioning schemes have been proposed [5], [7], [8], [13]–[17], [21]. Explicit resource allocation is also important in a virtualized environment where tasks with different quality-of-service (QoS) goals are co-scheduled [18]. While the effect of cache capacity partitioning and off-chip bandwidth partitioning has been studied extensively, few studies have looked at how cache partitioning and bandwidth partitioning interact with each another. To address this void, this paper takes a first step to develop a simple yet powerful analytical model that considers the two allocation problems in an integrated manner. Our model is "hybrid" in the sense that it requires one-time profiling of target benchmarks to extract key parameters.

A. Previous related work

The problem of cache partitioning is, given N cores each running a task that share a cache capacity of C, to determine $\{c_1, c_2, \ldots, c_N\}$, $\sum c_i = C$ and allocate cache capacity c_i to the task on core i such that a desired objective function is maximized. Typical objective functions are aggregate throughput, weighted speedup, and fairness.

Suh et al. [21] dynamically allocate cache capacity to two co-scheduled tasks to minimize the number of cache misses. They introduce the notion of *marginal gain* to guide capacity allocation decisions. Iyer [5] proposes a framework called CQoS (cache QoS) to facilitate reasoning and design practices to offer QoS at the shared cache level. Likewise, in [7], Kim et al. propose a set of algorithms that promote fairness as the objective function for cache partitioning. Qureshi and Patt [16] proposed a low-overhead hardware mechanism to accurately monitor the utility of cache capacity given to a task (a notion similar to marginal gain). Unlike the above previous work that resorts to hardware support for partitioning, Lin et al. [8] uses the OS page allocation mechanism to "color" pages and control cache partitioning in software.

Researchers have studied the DRAM bandwidth partitioning problem [13]–[15], [17]. Nesbit et al. [15] focus on achieving fairness in the presence of DRAM bandwidth sharing. Rafique et al. [17] reduce the average DRAM access latency by adaptively adjusting bandwidth shares among co-scheduled tasks. Mutlu and Moscibroda [13], [14] achieve higher overall bandwidth utilization and system performance without degrading fairness with intelligent grouping/scheduling of memory accesses. However, these previous works assume private caches and/or do not consider coordination of cache and DRAM bandwidth partitioning.

Finally, Bitirgen et al. [2] proposed a global resource manager that uses artificial neural networks to allocate shared cache capacity and off-chip bandwidth together. Their simulation based study shows that the two partitioning problems must be tackled together for best performance. Our results, obtained with explicit analytical modeling, corroborate their observation. More recently, Liu et al. [9] presented a detailed analytical model to study how off-chip memory bandwidth partitioning affects CMP system performance. Similar to our work, they develop an additive CPI model that incorporates the effect of finite cache capacity and DRAM bandwidth. The main differences of their model and ours are: (1) Liu et al. use an in-order core model while we model an out-of-order core; (2) They use Little's law to estimate queueing delays (determined by the arrival rate of the memory access process and the delay of each access) while we employ a probabilistic model with an access arrival histogram collected from actual applications; and (3) They focus on dynamic partitioning of cache and DRAM bandwidth while we first focus on static partitioning to more easily discover correlations between particular partitioning decisions. Despite the differences, both works reveal synergistic interactions between

cache partitioning and DRAM bandwidth allocation.

B. Our contributions

We make the following contributions in this paper:

• We propose a profiling and analysis method to predict the effect of limited bandwidth on a program's performance. The bandwidth available to a program is controlled by allocating a number of "memory access slots." We account for the effect of the available bandwidth using a queueing model and the one-time profiling of the target program. The capability to predict the impact of limited bandwidth allows a designer to reason about the effect of co-scheduling multiple programs on a CMP chip.

• We validate our model against a detailed architecture simulator capable of modeling out-of-order cores. Our validation results demonstrate that the proposed model predicts the performance trend that agrees with results produced by the simulator.

• We develop an analytical model to explore the coordinated management of the shared L2 cache and the off-chip bandwidth resources. With an elaborate case study, we show that our model can be successfully used to consider the effect of allocating individual resources simultaneously and can correctly guide the decision process for effective co-management of the two resources.

C. Paper organization

The remainder of this paper is organized as follows. We will first describe the baseline performance model in Section II, followed by Section III which details how we model the effect of limited off-chip bandwidth. Section IV validates the model. A detailed case study is presented in Section V and finally, Section VI will summarize the conclusions of this paper.

II. Base Model

While our model is not fundamentally limited by the number of cores, we will use a two-core CMP as an example processor for illustration. We will first discuss basic constraints and develop performance models.

A. Cache size and bandwidth constraints

When cache and bandwidth resources are shared among processor cores, the performance of a core is limited by the cache capacity and the bandwidth it is allocated. Consider two threads, A and B, sharing the cache space and the bandwidth. The sum of cache capacity assigned to each thread, (S_A, S_B), must be less than the total cache capacity of the system, (S_s):

$$S_A + S_B \quad \leq \quad S_s \tag{1}$$

Likewise, the total bandwidth requirements of the threads must not exceed the system bandwidth:

$$\begin{aligned} BW_r &= \sum BW_{thread} \\ BW_r &\leq BW_s \end{aligned} \tag{2}$$

The main problem we consider in this paper is, then, to find optimal S_A, S_B and BW_A, BW_B that maximize the system throughput, fairness, or both.

B. Performance model

We measure the performance of a core in terms of IPC (instructions per cycle), or inversely, CPI. The aggregate system performance is then simply sum of IPCs of the cores. The cache capacity and off-chip bandwidth constraints will add to the overall execution time of a program and its CPI. To incorporate the extra queueing delays caused by the resource allocation we have:

$$CPI \quad = \quad CPI_{ideal} + CPI_{fcp} + CPI_{queue} \tag{3}$$

where CPI_{ideal} is an unconstrained CPI (infinite cache capacity and bandwidth), CPI_{fcp} is the extra delay per instruction caused by a finite cache capacity, and CPI_{queue} is the extra queueing delay per instruction caused by limited off-chip bandwidth.

C. Cache misses and memory-level parallelism

In our model, we consider out-of-order superscalar processor cores. In-order processor blocks on a long latency cache miss, and hence, it does not saturate the off-chip bandwidth unless we have hundreds of cores. Modeling the performance of an out-of-order superscalar processor is not straightforward. The processor continues to issue independent cache accesses on long latency misses to hide the latency. We adopt the first-order superscalar processor model [6] to estimate the impact of a long latency data cache miss on the out-of-order superscalar processor performance.

The first-order model estimates the penalty of an "isolated cache miss" and an "overlapped cache miss" differently. The penalty of an isolated cache miss is approximated by the cache miss latency. To handle overlapped cache misses, the first-order model focuses on the distance between the memory instructions that generated the misses. Since the processor can issue cache accesses only if the corresponding memory instructions are in the ROB (reorder buffer [3]) at the same time, the model assumes a cache miss is overlapped with other cache misses that are within the range of the ROB size.

In general, if m is the number of long latency data cache misses and $f(i)$ is the probability that misses will occur in groups of i, the miss penalty becomes: $(miss \quad latency \times \sum_{i=1}^{m} f(i)/i)$ where distribution $f(i)$ can be collected while we profile the target benchmark.

Finally, CPI_{fcp} is:

$$\begin{aligned} CPI_{fcp} &= MPI \times miss\ penalty \\ &= MPI \times lat_M \times \sum_{i=1}^{m} \frac{f(i)}{i} \end{aligned} \tag{4}$$

where MPI is average misses per instruction and lat_M is the memory access latency.

III. Modeling Effects of Limited Bandwidth

Insufficient off-chip bandwidth results in processor stalls and affects the performance of not only individual applications but also the overall system. This section derives a bandwidth model that can be plugged into Equation (3). The terms used in our analytical model are listed in Table I.

A. Bandwidth formulation

Memory bandwidth is the rate at which data can be read from or stored into a memory by a processor.

$$\text{Bandwidth } (bytes/second) = \frac{Bytes\ transferred\ between\ cache\ and\ memory}{Execution\ time} \quad (5)$$

To calculate the bytes transferred between cache and memory in Equation (5), we multiply the number of cache misses and the cache block size.

$$Cache\ misses \times Cache\ block\ size\ (BS) \quad (6)$$

We use several program parameters to obtain the execution time in Equation (5): total instruction count (IC), ideal CPI (CPI_{ideal}), misses per instruction (MPI), memory access latency (lat_M), and clock cycle time ($CCT = 1/F$).

$$\text{Execution time } (T) = IC \times (CPI_{ideal} + MPI \times lat_M) \times CCT \quad (7)$$

With Equations (6) and (7), the bandwidth requirement (BW_r) for a thread can be written as:

$$\begin{aligned} BW_r &= \frac{IC \times MPI \times BS \times F}{IC \times (CPI_{ideal} + MPI \times lat_M)} \\ &= \frac{MPI \times BS \times F}{CPI_{ideal} + MPI \times lat_M} \end{aligned} \quad (8)$$

The off-chip interface is modeled as a single-server queueing system which serves requests on a first come first serve (FCFS) basis. Hence, requests that find the bus busy must wait in a queue until the bus becomes free. Thus, the effect of off-chip bandwidth contention is the extra queueing delay (lat_{queue}) seen by off-chip memory requests. With the extra memory access latency due to contention, the total memory bandwidth consumption (BW_r) is smaller than the system bandwidth (BW_s) as expressed below:

$$BW_s \geq \sum_{i=1}^{N} \frac{MPI_i \times BS \times F}{CPI_{ideal\_i} + MPI_i \times (lat_M + lat_{queue\_i})} \quad (9)$$

where N is the number of threads running on a CMP.

B. Calculating average queueing delay

To estimate the average extra queueing delay of a thread under bandwidth limitation, we profile the elapsed cycles between consecutive L2 misses and create a histogram we call "$miss\text{-}inter\text{-}cycle$." The histogram is obtained from running a program without bandwidth limitation to observe how dense the program would generate off-chip accesses throughout the execution.

After obtaining the histogram, to compute a reasonable average queueing delay with various off-chip bandwidth capacity, we employ a *queueing delay calculator* (`qdc`) based on a $G/D/m$ queueing system: G—general input distribution ($miss\text{-}inter\text{-}cycle$), D—processing time of each station (memory latency), m—number of stations (slot count). `qdc` is essentially an event-driven Monte-Carlo simulator.

TABLE I
INPUT PARAMETERS.

System parameters	
BS	Cache block size (Bytes)
F	CPU clock frequency (Hz)
BW_s	System bandwidth (Bytes/sec)
$slot$	Number of off-chip memory access interfaces
Thread parameters	
MPI	Misses per instruction
MLP_{effect}	The effect of memory-level parallelism on memory access latency
CPI_{ideal}	Ideal CPI with infinite L2 cache size
lat_M	Memory access latency
lat_{queue}	Extra queueing delay due to contention
$slot_i$	Fraction of off-chip b/w assigned to thread i

C. Computing additive CPI components

Let us revisit Equation (4) and rewrite it:

$$\begin{aligned} CPI_{fcp} &= MPI \times MLP_{effect} \times lat_M \\ &= MPI_0 \left(\frac{C}{C_0}\right)^{-\alpha} \cdot MLP_{effect} \cdot lat_M \end{aligned} \quad (10)$$

We use the power law [4] to project MPI for various cache sizes based on MPI_0, the value for a reference cache size. MLP_{effect} is simply $\sum_{i=1}^{m} f(i)/i$.

Recall that we assume the extra queueing delay due to contention on average to be lat_{queue}. Then, for an application thread, we have:

$$\begin{aligned} CPI_{queue} &= MPI \times lat_{queue} \\ &= MPI_0 \cdot \left(\frac{C}{C_0}\right)^{-\alpha} \times lat_{queue} \end{aligned} \quad (11)$$

Interestingly, `qdc` has revealed that the queueing delay due to contention decreases as we add more bandwidth (slots) according to the power law. That is, we have:

$$lat_{queue} = lat_{queue0} \cdot \left(\frac{Slot}{Slot_0}\right)^{-\beta} \quad (12)$$

Fig. 1 illustrates this finding. It plots the queueing delays calculated by Equation (12) (solid line) and the estimated values by `qdc` with different off-chip bandwidth capacity or slot count.

Finally, Equation (3) is rewritten:

$$CPI = CPI_{ideal} + MPI_0 \cdot \left(\frac{C}{C_0}\right)^{-\alpha} \times \left(MLP_{effect} \times lat_M + lat_{queue0} \cdot \left(\frac{Slot}{Slot_0}\right)^{-\beta}\right) \quad (13)$$

This equation reveals that the effect of the extra queueing delay on CPI depends on MPI. Since the value of MPI varies with different applications, each application suffers from the queueing delay to a different degree. Hence, bandwidth allocation (partitioning) can improve the overall system performance if it favors applications that are more sensitive to the queueing delay (i.e., having a large MPI value) over applications that are less sensitive.

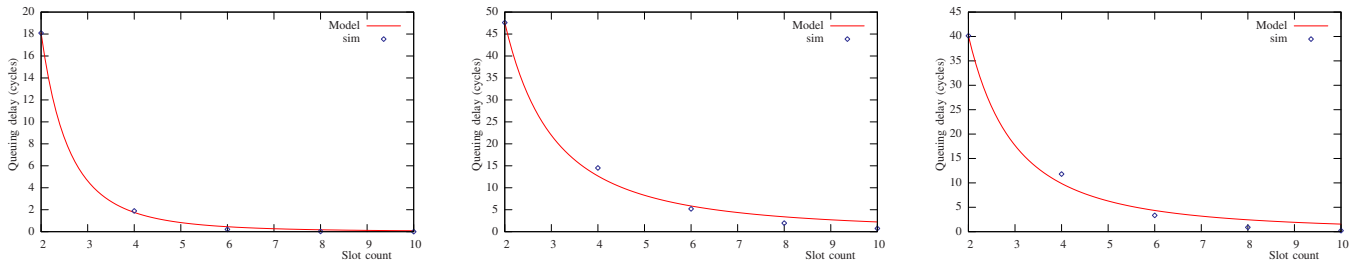

Fig. 1. Examples showing that extra queueing delay with off-chip bandwidth constraint follows a power law: the queueing delay varies as a power of slot count—*astar*, *bwaves* and *cactusADM*.

D. Bandwidth allocation

Off-chip bandwidth allocation determines how to distribute memory request bursts from processor cores over the off-chip interfaces to best utilize the available bandwidth. The off-chip bandwidth can be allocated or partitioned among cores in both space and time; for example, indicate that a core needs exclusive access to off-chip bandwidth 60% of the time, or that it requires 75% of the off-chip bandwidth. In our work, we focus our attention solely on the spatial dimension of partitioning.

In order to intuitively study the off-chip bandwidth allocation effect, we model the available bandwidth with allocatable *memory access interfaces* or *slots*, N_{slot}. For instance, with 64 byte cache block size, 1 GHz frequency, and 200 cycle off-chip memory access latency, the peak off-chip bandwidth of the system can is N_{slot} × (block size)/(memory access latency) or $N_{slot}\times$ 320 MB/sec.

E. Limitations

To make our model tractable, we have introduced simplifying assumptions. First, we limit our study to CMPs with identical, single threaded cores. Second, we assume workload characteristics do not change over their execution. Third, we assume each core has a private portion in shared L2 cache and threads that run on different cores do not share data. Lastly, we do not evaluate the power implications of various potential CMP configurations.

IV. Validation

A. Experimental setup

To validate our model, we compare our model and Zesto [10], a detailed CMP architecture simulator. Table II lists the CMP configuration parameters used in our experiments. We study six SPEC CPU2006 benchmarks: *astar*, *bwaves*, *cactusADM*, *gobmk*, *h264ref*, and *hmmer*. To quantify the prediction error of our models, we take the absolute difference between the CPI predicted by our model and the CPI measured by the simulator, divided by the CPI measured by the simulator.

B. Results

To obtain the key parameters of our model, we first collect CPI_{ideal} (CPI with infinite L2 cache), MPI_0 (misses per instruction) for a baseline cache size, and MLP_{effect} (the effect of memory-level parallelism on memory access latency) from one-time profiling of each benchmark. To collect the power law parameters α and β, we run each benchmark with different L2 cache sizes and off-chip access slot counts.

TABLE II
CMP configuration parameters.

CMP	2 cores on chip, shared on-chip L2 cache
Core	4-issue OOO, 96-entry ROB
L1 cache	private Icache and Dcache for each core Icache: 32KB, 64B cache block size, 4-way Dcache: 32KB, 64B cache block size, 4-way
Shared L2 cache	4MB, 64B cache block size 32-way, 12 cycle hit latency
Memory	∼3.2 GB/s off-chip bandwidth 200 cycle round trip latency

We first verify the accuracy of the out-of-order processor CPI model described in Section II-C. Fig. 2 shows CPIs collected by simulation and our model. The figure shows that the overall trends of the predicted CPIs obtained with our model closely follow the trends shown by the simulation. The prediction errors of our model are low. The arithmetic and geometric mean of the errors with different cache sizes are 4.8% and 3.9%, respectively. With different off-chip bandwidth, the arithmetic and geometric mean of the errors are 6.0% and 2.4%.

From the results of the benchmarks with cache size and slot count variation, interestingly we observe that the applications may get affected by both cache capacity and off-chip bandwidth limitation, or by either of the two constraints. For example, the cache capacity has an impact on *astar* to some extent, up to 50% of CPI increase (with 128 KB). However, the off-chip bandwidth constraint barely affects the application's performance. Applications similar to *astar* will benefit from having more cache capacity rather than increasing the off-chip bandwidth.

The cache capacity affects the CPI of *bwaves* by a very small margin, up to 1 or 2% of CPI increase (with 128 KB), but *bwaves* suffers from small off-chip bandwidth, up to 40% of CPI increase with 2 slot counts. The CPI of this type of application scarcely increases when the cache capacity decreases. Applications similar to *bwaves* will benefit from allocating more off-chip bandwidth rather than increasing the cache size. *cactusADM* has an impact from the cache capacity,

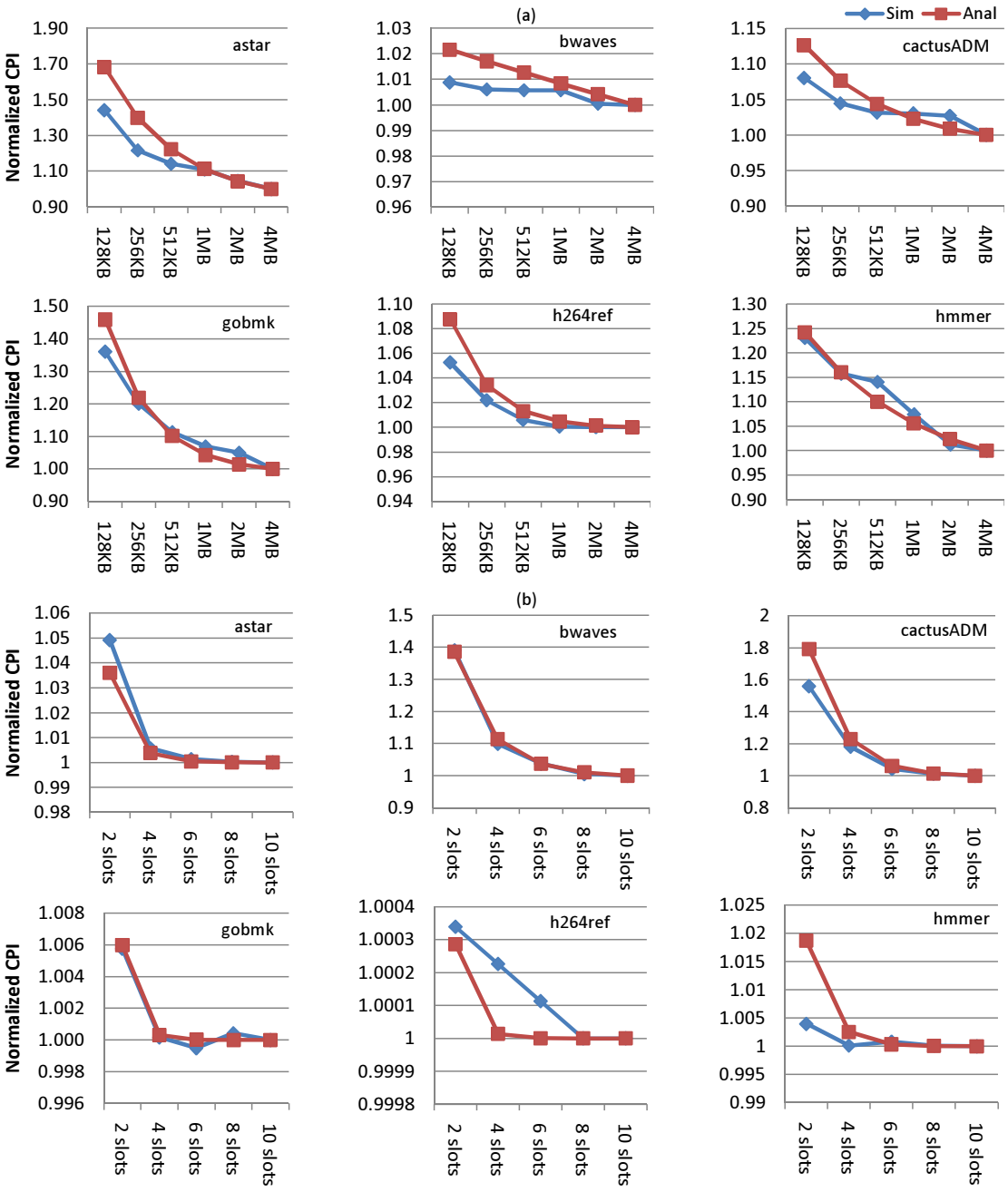

Fig. 2. CPIs collected with our model and Zesto (a) using different L2 cache sizes and (b) allocating different off-chip bandwidth. The CPIs are normalized to the CPI with 4MB cache size in (a) and to the CPI with slot count 10 in (b).

up to 12% of CPI increase (with 128 KB), and the off-chip bandwidth constraint, up to 60% of CPI increase with the slot count 2. The performance of such applications can easily be affected when either the off-chip bandwidth or cache capacity is decreased.

V. Case Study

A. Evaluated architecture and workloads

Table III lists the base parameters that we selected hypothetically to reveal the capability of our model. We use two threads in this study—thread A and thread B.

B. System optimization objectives

We have three system-level optimization objectives focusing on performance and fairness. "Throughput" measures the combined progress rate of all co-scheduled threads, whereas "fairness" measures how uniformly the threads are slowed down due to resource sharing.

Throughput. The aggregate performance of a system can be defined as the sum of the throughput of all cores in the system.

TABLE III
INPUT PARAMETERS.

System Parameters		
BS, cache block size (Byte)		64 B
F, CPU clock frequency (Hz)		1 Ghz
$Slot$, peak off-chip bandwidth		1.6 GB/s
lat_M, L2 miss penalty (cycles)		10 cycles
	Thread A	Thread B
MLP_{effect}, the effect of memory-level parallelism on memory access latency	0.55	0.25
CPI_{ideal}, ideal CPI with infinite L2	1.05	0.3
MPI_0, baseline MPI	0.027	0.077
lat_{queue0}, baseline queueing delay	78.09	50.34
α, power law factor for cache size	0.88	0.78
β, power law factor for queueing delay	3.36	4.29

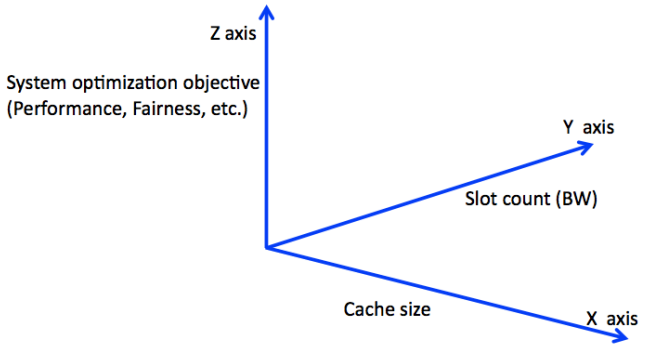

Fig. 3. Shared resources (cache and off-chip bandwidth) variation on 3-dimensional graph.

We use:

$$IPC_{system} = \sum_{i=1}^{N_{core}} IPC_i \quad (14)$$

where N_{core} represents the number of cores in the system.

Fairness. Improving the throughput of a system may cause unwanted degradation in some application's performance. Weighted speedup (WS) was proposed by Snavely and Tullsen [19] to measure the fairness of co-scheduled threads. More precisely, if thread i achieves an IPC of $IPC_{alone,i}$ when running alone in a CMP system and achieves an IPC of IPC_i when running simultaneously with applications on other cores, fairness is expressed as

$$\sum_{i=1}^{N_{core}} WS_i = \sum_{i=1}^{N_{core}} \frac{IPC_i}{IPC_{alone,i}} = \sum_{i=1}^{N_{core}} \frac{CPI_{alone,i}}{CPI_i} \quad (15)$$

where WS_i is the weighted speedup for thread i.

Harmonic mean of normalized IPC. Using the fairness and the throughput metric by itself may have drawbacks. The fairness does not give any importance to the overall throughput and the throughput does not provide any fairness among threads. The harmonic mean fairness (harmonic mean of normalized IPCs) balances both fairness and performance [11].

$$HM_{IPC} = \frac{N_{core}}{\sum_{i=1}^{N_{core}} \frac{IPC_{alone,i}}{IPC_i}} \quad (16)$$

C. Approximating optimal cache and off-chip bandwidth allocation

To observe the impact of varying allocated L2 cache capacity and off-chip bandwidth, we draw color-mapped 3D figures. Fig. 3 presents conceptually the dimensions we vary in the CMP resource management problem. The X axis represents the L2 cache capacity and the Y axis represents the off-chip bandwidth allocated for thread A. The rest of the L2 cache capacity and the off-chip bandwidth are allocated for thread B. Our analytical model draws a "terrain"-like surface (variations along the Z axis) in the area defined by the X and Y axis. With the movement of thread A along the X and Y axis, we can observe how the two resources affect individual thread and the system performance by focusing on the Z axis. We vary the L2 cache size from 128 KB to 4 MB, the slot count from 1 to 4, assuming the off-chip bandwidth is 1.6 GB/sec (5 slots) in the studied system.

Fig. 4 shows the summation of two threads' off-chip bandwidth requirements (normalized to the peak off-chip bandwidth) over different cache capacity and off-chip bandwidth allocations. The color in the figure varies from black to yellow via purple and red according to the Z axis value. The lowest area in Fig. 4(a) is shown in purple and the highest area is shown in red and yellow. Fig. 4(b) shows the 3D plot projected onto a 2D plane.

With the above arrangement, it becomes intuitive for us to inspect at a high level how the total off-chip bandwidth requirements change with different cache capacity allocation points. Assuming that the actual sustained DRAM bandwidth doesn't exceed about two thirds of the peak DRAM bandwidth, we carved out the regions that actually show higher bandwidth than that, as shown in Fig. 4(b). That is, the regions within the dotted lines $\overline{AB}$ and $\overline{CD}$ in the plot represent the cache allocation points where the system performance will be likely constrained by the limited off-chip bandwidth.

D. Throughput

Fig. 5 shows the summation of two threads' IPC. The color is changed from black to red via blue and green according to the Z axis value. The highest area in Fig. 5(a) is shown in red and the lowest area in black. In Fig. 5(a), the solid line (A) makes a steep curve, suggesting that the overall IPC can be affected by the off-chip bandwidth variation when thread A is given a smaller amount of cache capacity than thread B.

On the other hand, line (B) shows that the off-chip bandwidth variation has a little effect on the overall IPC when thread A uses much larger cache capacity than thread B. From the figure, we can roughly determine that thread A should be assigned with smaller cache capacity and off-chip bandwidth than thread B.

We project the 3D plot in Fig. 5(a) onto a 2D plane, as shown in Fig. 5(b), to find the best resource allocation. Regardless of how we allocate cache and off-chip bandwidth in the black and blue area, it is hard to achieve high throughput.

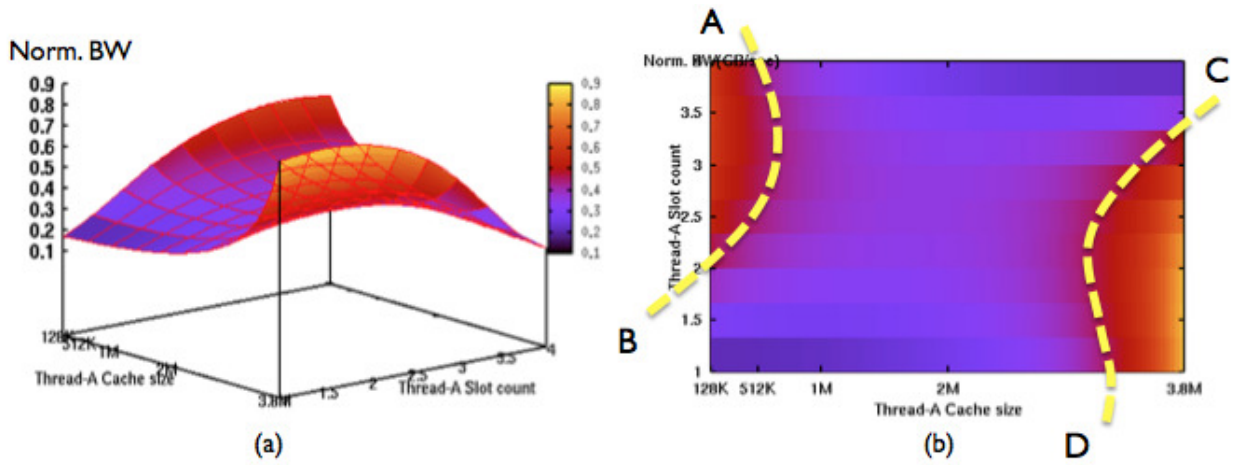

Fig. 4. Off-chip bandwidth requirement of two threads: (a) Summation of two threads' off-chip bandwidth requirement, (b) projection of the 3D graph in (a) onto a 2D plane. The line $\overline{AB}$ and $\overline{CD}$ in (b) indicate the boundary for the optimal resource sharing with regard to the off-chip bandwidth constraint.

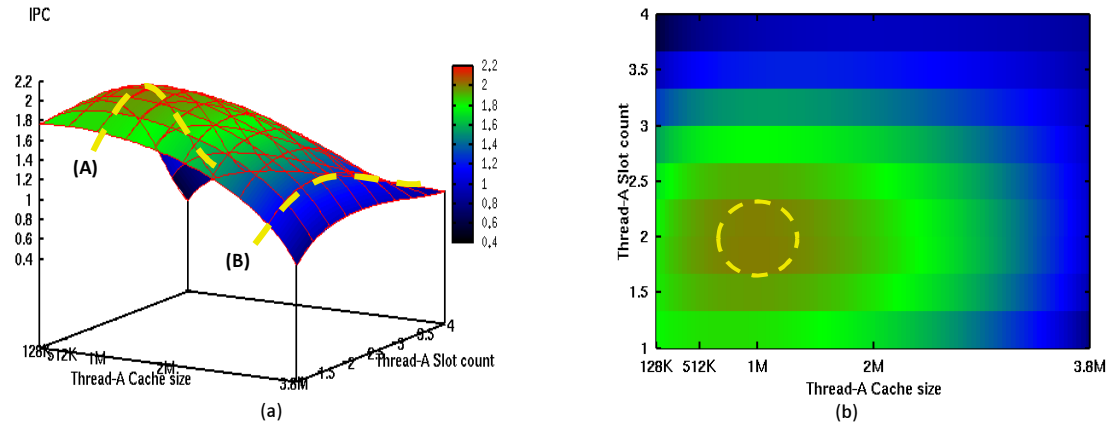

Fig. 5. Overall IPC of two threads: (a) Summation of two threads' IPC, (b) projection of the 3D graph in (a) onto a 2D plane. The circled area in (b) reveals the best resource allocation for the system throughput.

Nevertheless, in the green and red area, we can accomplish a fairly high throughput of the system. The red area reveals the best resource allocation of cache and off-chip bandwidth to obtain the highest IPC: thread A—1 MB cache and 2 slots, Thread B—3 MB cache and 3 slots.

E. Fairness

To achieve fairness among the applications, we use the weighted speedup metric. The weighted speedup estimates the reduction in execution time of each thread, by normalizing each thread's performance to its inherent IPC value obtained when the thread runs alone.

Fig. 6 shows the summation of weighted speedup of each thread. Fig. 6(a) shows how different resource allocations among threads impact the fairness of the system. The highest area in Fig. 6(a) is colored in red and the lowest area is colored in black. Line (A) makes a relatively steep curve, suggesting that the off-chip bandwidth variation can affect the weighted speedup when thread A and thread B have same cache size. Line (B) has a relatively gentle curve with the variation on the off-chip bandwidth. From the figure, we can determine that thread A and B should have similar cache capacity.

We again map the 3D graph in Fig. 6(a) onto the 2D plane, as shown in Fig. 6(b). Regardless of how we allocate resources in the black and blue area, it is difficult to achieve fairness between the two threads. The red area reveals the best resource allocation of cache capacity and off-chip bandwidth for the fairness: Thread A—2 MB cache and 2.3 slots, Thread B—2 MB cache and 2.7 slots.

F. Harmonic mean of normalized IPC

Until now, we used throughput and fairness as an independent metric. Let us consider the harmonic mean of normalized IPC, which is a metric that combines both throughput and fairness, as described in Section V-B.

Fig. 7 shows the summation of the harmonic mean of the normalized IPC of each thread. In Fig. 7(a), line (A) makes a steep curve, demonstrating that the off-chip bandwidth variation can affect both overall IPC and overall fairness when thread A is given a smaller cache capacity than thread B.

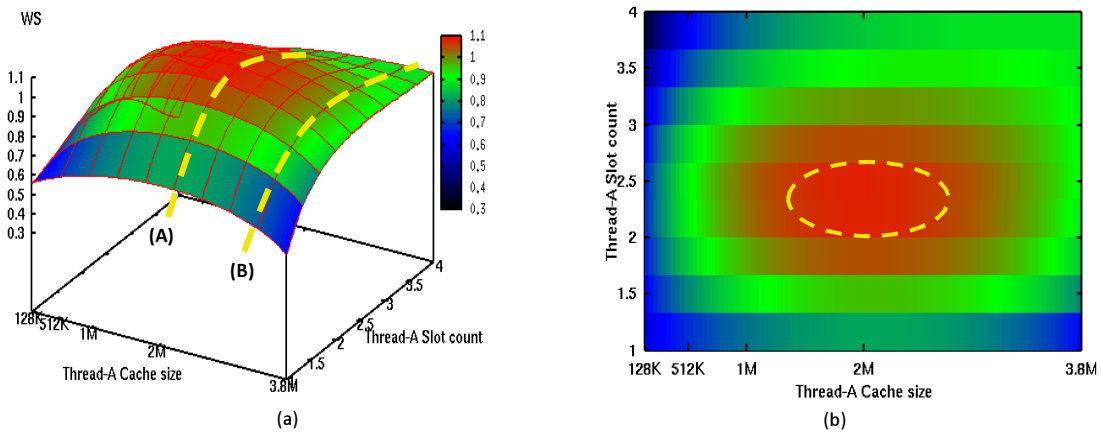

Fig. 6. Weighted speedup of two threads: (a) Summation of two threads' weighted speedup, (b) projection of the 3D graph in (a) onto a 2D plane. The circled area in (b) reveals the best resource allocation for the fairness.

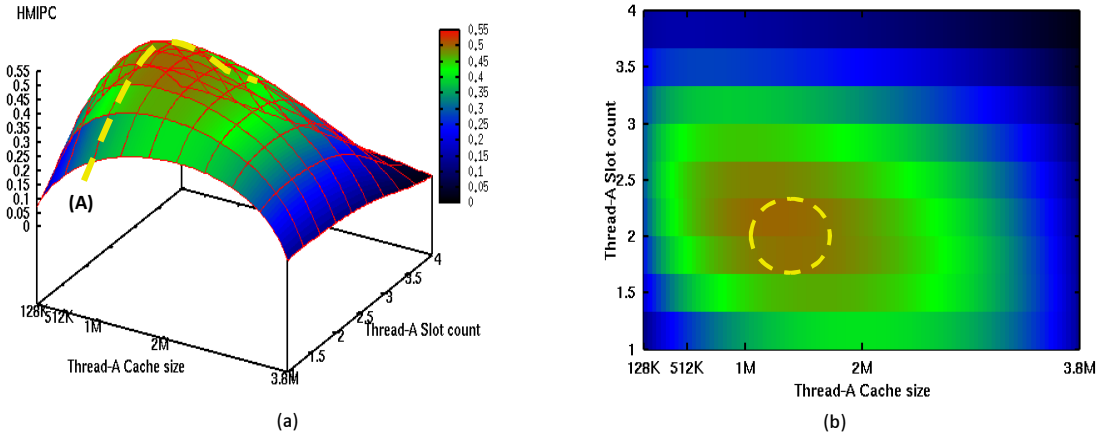

Fig. 7. Harmonic mean of normalized IPC of two threads: (a) Summation of two threads' harmonic mean of normalized IPC, (b) projection of the 3D graph in (a) onto a 2D plane. The circled area in (b) reveals the best resource allocation for the throughput and fairness.

In Fig. 7(b), the red area shows the best resource allocation of cache capacity and off-chip bandwidth when we simultaneously consider both the throughput and the fairness: thread A—1.3 MB cache and 2.2 slots, thread B—2.7 MB cache and 2.8 slots.

VI. Summary and Conclusions

We discussed how we can achieve a system optimization goal by considering the cache capacity allocation and the off-chip bandwidth allocation simultaneously. We highlighted the importance of achieving a balance between the two. We also proposed a simple and powerful model that considers both cache capacity and off-chip bandwidth at the same time. By combining the two resource allocation problems, resources can be distributed more efficiently to achieve higher throughput as well as fairness.

It has previously been recognized that off-chip bandwidth allocation may have little impact on the system performance because cache allocation has a larger impact on performance in a CMP architecture. Interestingly, however, such a hypothesis is not necessarily correct. Our case study shows that addressing the cache capacity and off-chip bandwidth allocation problems together can improve the overall system performance, implying that there should be a synergistic interaction between them.

Fig. 8 indicates the best resource (cache and off-chip bandwidth) allocations for each optimization objective in the system: throughput (A), fairness (B), and both throughput and fairness (C). Higher throughput ensures higher utilization of processor resources, and fairness ensures that all threads are given equal opportunity and that no threads are forced to starve. Typically, different threads have different instruction execution rates, and hence show different throughput. Therefore, when allocating resources among threads, merely considering the throughput will give priority to threads with high throughput, causing the rest to potentially suffer. On the other hand, considering only fairness will result in inefficient use of resources. Hence, focusing on a single objective provides the best combined resource allocation for the objective itself, but it may not provide the best resource allocation for other system objectives.

In conclusion, we find that our analytical model is useful for studying the two important CMP resource management

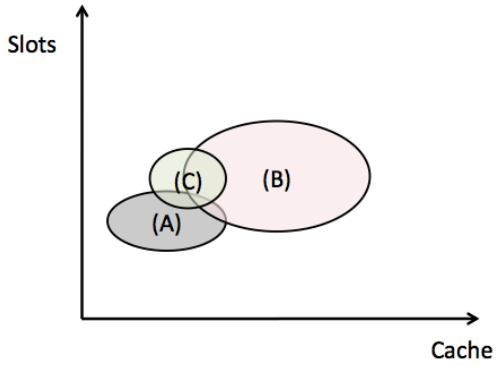

Fig. 8. Balanced optimal allocation points of shared resources (cache and off-chip bandwidth) for 3 different optimization goals: (A) throughput, (B) fairness, and (C) both throughput and fairness.

problems together—cache capacity allocation and off-chip bandwidth allocation. Unlike previous efforts, our model is capable of modeling out-of-order processor cores. Our validation study reveals that the proposed model has reasonable accuracy in predicting the performance impact as we change the cache capacity and the off-chip bandwidth allocation for a thread. Our model requires a one-time profiling of target threads using a fast single-thread architecture simulator.

ACKNOWLEDGEMENT

This work was supported in part by the US NSF grants: CCF-1064976, CCF-1059283, and CCF-0702236; and by the Central Research Development Fund (CRDF) of the University of Pittsburgh. This work was done as part of the first author's thesis research and portions of this paper appear in his thesis.

REFERENCES

[1] AMD Multi-core. http://multicore.amd.com
[2] R. Bitirgen, E. Ipek, and J. F. Martinez. "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach" *Proc. Int'l Symp. Microarchitecture (MICRO)*, Dec. 2008.
[3] J. L. Hennessy and D. A. Patterson. "Computer Architecture: A Quantitative Approach" *Morgan Kaufmann*, 2003.
[4] J. Huh, D. Burger, and S. W. Keckler. "Exploring the Design Space of Future CMPs" *Proc. Int'l Conf. Parallel architectures and computation Techniques (PACT)*, 2001.
[5] R. Iyer. "CQoS: a Framework for Enabling QoS in Shared Caches of CMP Platforms," *Proc. Int'l Conf. Supercomputing (ICS)*, June 2004.
[6] T. S. Karkhanis and J. E. Smith. "A First Order Superscalar Processor Model," *SIGARCH Comput. Archit. News*, 2004.
[7] S. Kim, D. Chandra, and Y. Solihin. "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2004.
[8] J. Lin et al. "Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems," *Proc. Int'l Symp. High Performance Computer Architecture (HPCA)*, Feb. 2008.
[9] F. Liu, X. Jiang, and Y. Solihin. "Understanding How Off-Chip Memory Bandwidth Partitioning in Chip Multiprocessors Affects System Performance," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA)*, Jan. 2010.
[10] G. H. Loh, S. Subramaniam, and Y. Xie. "Zesto: A Cycle-Level Simulator for Highly Detailed Microarchitecture Exploration," *Proc. Int'l Conf. Performance Analysis of Software and Systems (ISPASS)*, Apr. 2009.
[11] K. Luo, J. Gummaraju, and M. Franklin. "Balancing throughput and fairness in smt processors," *Proc. Int'l Conf. Performance Analysis of Software and Systems (ISPASS)*, 2001.
[12] Micron Technology. "TN-47-21: FBDIMM-Channel Utilization (Bandwidth and Power)" http://www.micron.com, 2006.
[13] O. Mutlu and T. Moscibroda. "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," *Proc. Int'l Symp. Microarchitecture (MICRO)*, Dec. 2007.
[14] O. Mutlu and T. Moscibroda. "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," *Proc. Int'l Symp. Computer Architecture (ISCA)*, June 2008.
[15] K. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. "Fair Queueing Memory Systems," *Proc. Int'l Symp. Microarchitecture (MICRO)*, Dec. 2006.
[16] M. K. Qureshi and Y. N. Patt. "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," *Proc. Int'l Symp. Microarchitecture (MICRO)*, Dec. 2006.
[17] N. Rafique, W. Lim, and M. Thottethodi. "Effective Management of DRAM Bandwidth in Multicore Processors," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2007.
[18] J. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*, Morgan Kaufmann, 2005.
[19] A. Snavely and D. M. Tullsen. "Symbiotic Job Scheduling for a Simultaneous Multithreading Processor," *Proc. Int'l Conf. Architecture Support for Programming Language and Operating Systems (ASPLOS)*, Jan. 2000.
[20] SPEC CPU 2006. http://www.spec.org/cpu2006/.
[21] G. E. Suh, S. Devadas, and L. Rudolph. "A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA)*, Feb. 2002.