Performance Estimation of Multiple-Cache IP-based Systems: Case Study of an Interdependency Problem and Application of an Extended Shared Memory Model

Sungjoo Yoo Kyoungseok Rha Youngchul Cho Jinyong Jung Kiyoung Choi

Design Automation Laboratory
School of Electrical Engineering
Seoul National University
Seoul 151-742, Korea
{ysj,contron,rams,jyjung,kchoi}@poppy.snu.ac.kr

Abstract

In estimating the performance of multiple-cache IP-based systems, we face a problem of interdependency between cache configuration and system behavior. In this paper, we investigate the effects of the interdependency on system performance in a case study. We present a method that gives fast and accurate estimation of system performance by simulating IP cores at the behavioral level with annotated delays and by simulating the multiple-cache communication architecture with an extended shared memory model. Experiments show the effectiveness of the proposed method.

1 Introduction

As communication architectures of IP-based systems, shared-bus architectures are widely used in many IP-based systems. Recently, research on performance estimation [1][2] and optimization [3] of such communication architectures are gaining more and more attention. In shared-bus architectures, since performance bottleneck comes mostly from excessive bus conflicts, placing data caches near IP cores can give significant improvement of system performance by reducing bus conflicts. In such multiple-cache IP-based systems (where each IP accesses shared data via its own cache), due to the huge design space of cache configurations[1], finding an optimal cache configuration in terms of system performance metrics such as runtime (best, average, or worst case), power consumption, etc, requires fast and accurate performance estimation.

In applying traditional trace-driven cache simulation methods [4][5][6][7] to the performance estimation of multiple-cache IP-based systems, we face a significant problem. In such methods, address traces are initially obtained and assumed invariant over various cache configurations. However, in multiple-cache IP-based systems, system behavior, i.e. address traces can vary significantly as the cache configuration changes. Thus, cache simulation using the address trace, which does not vary for the whole design space of cache configurations, which we call a **static trace**, can give significantly inaccurate performance estimation, and possibly incorrect design decisions.

In this paper, in a case study, we investigate the effects of the interdependency between cache configuration and system behavior in terms of system performance estimation. In the case study with a CDMA modem system [8][9] and a CD2DAT system, we measure the accuracy of cache simulation with static traces for various cache configurations and analyze the effects of interdependency. We also present a performance estimation method that calculates accurate system performance by simulating IP cores at the behavioral level with annotated delays and by simulating the multiple-cache shared-bus architecture with an extended shared memory model.

This paper is organized as follows. In section 2, we review related work. In section 3, we introduce a multiple-cache shared-bus communication architecture. In section 4, we give the case study. In section 5, we present a performance estimation method and related experimental results. We give conclusion in section 6.

[1] Even a single cache can have several parameters such as cache size, associativity, block (line) size, sub-block size, replacement policy, etc.

CODES 2000 San Diego CAUSA

2 Related Work

Cache issues have been investigated in many studies of IP-based design. In [10], processing elements are assumed to have caches and task-level analysis of cache effects is performed assuming well-contained tasks that cause only compulsory misses. In [11], an instruction cache simulation method is presented which gives accurate estimation of intra-task cache conflicts and approximates inter-task cache conflicts. In [12], an iterative cache simulation method, comparable to one-pass cache simulation in terms of runtime, has been presented. Energy consumption issues related to cache (and bus) dedicated to the processor have been investigated in [13]. In [14], to reduce the latency of fetching internal data to the boundary of IP core, a prefetch technique is presented. For fast and accurate performance estimation of shared-bus architectures (without caches), methods based on scheduling abstract traces are presented in [1] and [2]. As simulation-based performance estimation methods, in [15], three methods (macromodeling, cached simulation, and statistical sampling/sequence compaction) are presented to give efficient estimation of power consumption of SoC design.

In [16], a study of the effects of shared memory bus (e.g. bus conflict) and a method of modeling a shared bus have been presented. Since the method proposed in this paper extends the basic concept [16] of modeling the shared bus at the behavioral level to the multiple-cache shared-bus architecture with shared memory, we named our model an **extended shared memory model**. In this paper, our contribution is to investigate an interdependency problem in multiple-cache IP-based systems and to present a high-level model of multiple-cache communication architecture thereby speeding up the estimation of system performance.

3 Preliminaries

Figure 1 shows an example of a shared-bus communication architecture for IP-based systems considered in this paper. In the architecture, each IP core may (or may not) have a data cache to access data shared by other IP cores. For the coherency of multiple caches, we assume an invalidation-based cache coherency protocol

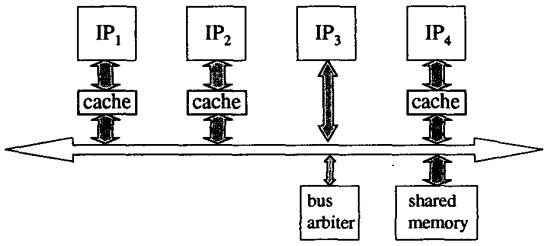

Figure 1: An example of shared-bus communication architecture.

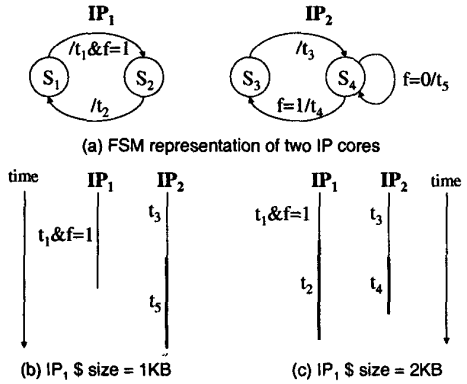

Figure 2: Interdependency between cache configuration and system behavior: an illustrative example.

[17]. In the protocol, each block (line) in a cache has five states : **invalid, exclusive clean/modified, shared clean/modified. Exclusive** represents that only the block has the valid copy among caches. **Shared** represents that the block shares the same copy with another cache(s). **Clean** represents that the contents of the block is the same with that of the shared memory and **modified** represents the copy in the shared memory is not valid and only the copy of the cache block is valid.

For cache coherency, each cache sends four types of special message (CO_RD, WR, INV, and CO_RD_INV) to the other caches in one of three cases : read miss, write miss, and write hit with shared clean/modified. When read miss, if the victim block[2] is in state **valid**, its contents is written back to shared memory with a WR type message. To read a new block from another cache or the shared memory, a CO_RD type message is used. When it is read from another cache (the owner of the block), the contents of shared memory is also updated (**memory reflection**). When write miss, a WR type message is used to write back the victim block in the case that its state is **valid**. To read a new block which will be overwritten immediately, a CO_RD_INV type message is used. When write hit to a block with state **shared**, an INV type message is used for **write invalidation**.

4 Case Study

In this section, we first give an example of the interdependency between cache configuration and system behavior. Then, we give experimental results showing the effects of interdependency in two practical systems.

4.1 An Illustrative Example

Figure 2 (a) illustrates an example of system representation with two IP cores, IP_1 and IP_2 which are modeled with two concurrent FSM's. Circles and edges represent states and transitions, respectively. Each transition is labeled by "guard/action". Consider the two IP cores communicate with each other (via caches and shared memory) on the communication architecture shown in Figure 1. Figure 2 (b) and (c) are assumed to represent snapshots of execution traces of two IP cores when IP_1 has a data cache with sizes of 1KB and 2KB. In the figures, each vertical line segment corresponds to an action related with a transition.

Let us assume the following scenario of the system execution. IP_1 is in state S_1 and IP_2 in state S_3 in Figure 2 (a). Both IP_1 and IP_2 make transitions t_1&f=1 and t_3, respectively. In state transition t_1&f=1, IP_1 sets a shared variable **f** to 1 (assuming **f** is initially 0). In the case that the cache size of IP_1 is 1KB, due to frequent cache misses, the state transition of IP_1, t_1&f=1 runs slow and finishes later than t_3 of IP_2 as shown in Figure 2 (b). In this case, when IP_2 finishes transition t_3, the variable **f** is still 0. Thus, IP_2 makes transition t_5 as shown in Figure 2 (b). However, in the case that the size of IP_1 cache is 2KB, as shown in Figure 2 (c), transition t_1&f=1 runs faster (by reduced cache misses assuming that the larger cache yields more hits) and finishes earlier than transition t_3. Thus, when transition t_3 finishes, the shared variable **f** is 1. In this case, IP_2 makes transition t_4. Figure 2 (b) and (c) show that there can be two different execution traces of IP_2 for two different cache configurations of IP_1.

As illustrated in the example of Figure 2, in multiple-cache systems, the execution trace, i.e. the address trace of the system can change depending on cache configurations. In general, the interdependency problem between cache configuration and system behavior belongs to the interdependency problem between timing and behavior. In this paper, we limit the problem to the interdependency between cache configuration and system behavior, i.e. address traces in multiple-cache IP-based systems.

4.2 Effects of the Interdependency

4.2.1 Case Examples

For the case study, we used a CDMA modem system [8][9] and a CD2DAT system from Ptolemy demo examples. The CDMA modem system is the transmitter of IS-95 CDMA mobile phone [18]. It consists of three IP cores: IP_1, IP_2 and IP_3. IP_1 performs CRC (cyclic redundancy code) generation, convolutional encoding, and symbol repetition. IP_2 is a block interleaver. IP_3 performs 64-ary orthogonal modulation, data randomization, long code generation, PN (pseudo-noise) code generation, and QPSK modulation. The CD2DAT system consists of four IP cores each of which performs a specific FIR operation. For the experiment, IP cores were written in SystemC [19] at the cycle-accurate behavioral level. The code sizes are 938 lines (CDMA) and 1,047 lines (CD2DAT).

Figure 3 shows the architectural view of CD2DAT system. Type information (CO_RD, WR, INV, or CO_RD_INV) is also carried on the address bus. The cycle-accurate behavioral model of cache, a bus arbiter, and shared memory were written in SystemC and their total code size is 1,549 lines.

4.2.2 Experiments

In our experiments, each cache has the following design parameters: number of blocks **b**, block size **s**, and associativity **a**. The ranges of three parameters are (1, 2, 4, 8, 16, 32, 64, 128, or 256) for parameter **b**, (1, 2, 4, 8, 16, or 32) for parameter **s**, and (1, 2, 4, or 8) for parameter **a**. We generated 100 static traces for each system[3] and for each static trace, we generated 100 random cache configurations. Thus, for each of 100*100 = 10,000 cache configurations, we ran cycle-accurate behavioral simulation of the sys-

[2] A **victim block** is the cache block to be replaced by cache conflict.

[3] To generate a static trace, we choose a random cache configuration and run cycle-accurate simulation of the system. During the cycle-accurate simulation, assuming perfect caches and no bus conflict, we extract address accesses as the static trace.

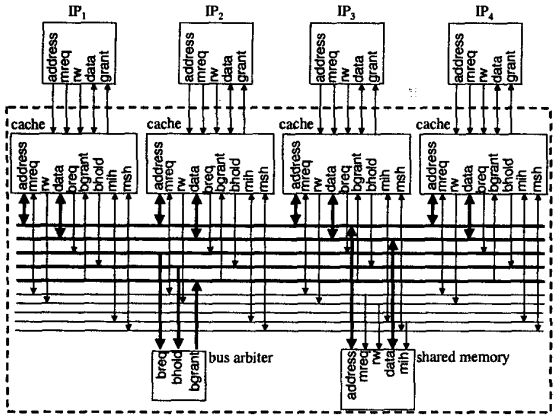

Figure 3: A detailed representation of multiple-cache communication architecture for the CD2DAT system.

Table 1: Comparison of estimation accuracy.

cache configurations. (b,s,a)'s	performance (clock cycles)		
	CA	static	error
CDMA modem			
(2,16,4) (4,16,1) (128,16,4)	147,848	105,577	-28.59%
(32,16,4) (64,16,2) (16,16,4)	104,043	141,013	35.53%
(2,8,4) (1,8,2) (32,8,4)	150,602	113,698	-24.50%
(64,8,4) (128,8,2) (1,8,4)	104,279	132,706	27.26%
CD2DAT			
(2,16,1) (8,16,2) (4,16,1) (8,16,2)	931,667	282,439	-69.68%
(32,1,4) (128,1,1) (128,1,4) (8,1,2)	221,102	476,494	115.51%
(2,2,4) (8,2,2) (2,2,4) (32,2,1)	942,366	451,405	-52.10%
(64,8,4) (1,8,1) (128,8,2) (4,8,1)	594,876	330,133	-44.50%

tem and trace-driven cache simulation.[4] The average numbers of memory accesses in static traces are 89,558 (CDMA) and 811,238 (CD2DAT).

Table 1 shows some cache configurations of the two systems where static traces yield large error in the estimation of system performance. In the first column, cache configurations of three (four) caches of CDMA modem system (CD2DAT system) are shown. In the column headings, **CA** represents the cycle-accurate simulation and **static** represents trace-driven cache simulation with static traces. As shown in the table, static traces give large error (up to 115%), especially when some of cache parameters have minimum values, i.e. at the boundary of the parameter space,

To visualize the variation of estimation error, we obtained the estimation error varying two arbitrarily selected parameters of cache configurations while the other parameters are fixed. Figure 4 shows the performance estimation error. In Figure 4 (a) and (b), we vary the numbers of blocks of two caches (in the figure, $b(IP_1)$ and $b(IP_2)$ for CDMA modem and $b(IP_1)$ and $b(IP_4)$ for CD2DAT) in each system. In Figure 4 (c) and (d), we vary the number of blocks of one cache and the degree of associativity of another cache (in the figure, $b(IP_1)$ and $a(IP_2)$ for CDMA modem and $b(IP_1)$ and $a(IP_4)$ for CD2DAT) in each system. The other fixed cache parameters are shown in the corresponding figures.

In our experiments, we observed that in the cases of multiple-cache systems, there may be no universal trace to give accurate performance estimation over all the possible cache configurations. Thus, trace-driven cache simulation with static traces can give significant error in performance estimation. Since such an error is due to the interdependency between cache configuration and system behavior, for accurate estimation of system performance, simulation of the system behavior at the behavioral level seems to be inevitably performed with the efficient simulation of multiple caches and communication architecture.

[4] We use our own trace-driven cache simulator for multiple-cache systems.

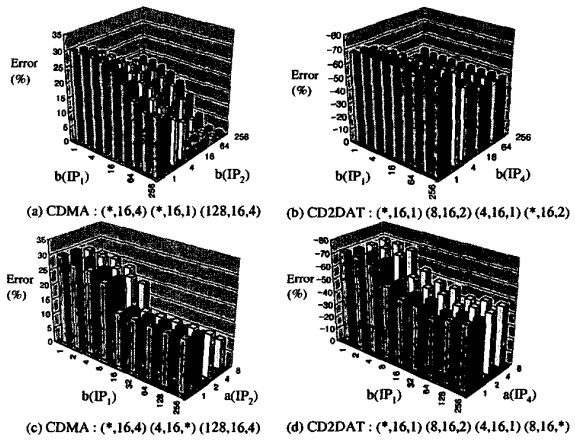

Figure 4: Performance estimation error of trace-driven cache simulation.

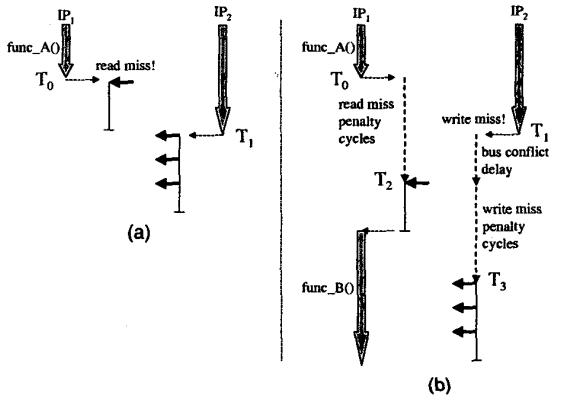

Figure 5: Illustration of scheduling IP core simulation and memory accesses.

5 Proposed Performance Estimation Method

In this section, we present a method that estimates system performance accurately by simulating IP cores at the behavioral level with annotated delays and by simulating the multiple-cache communication architecture with an extended shared memory model. We assume that the abstraction-levels of behavioral models of IP cores are selected by designers considering the trade-off between simulation speed and estimation accuracy in terms of behavioral IP simulation. In this paper, we focus on fast and accurate simulation of multiple-cache communication architecture. In our work, we model memory accesses from IP cores to caches (and related memory accesses among caches and shared memory) as an **extended shared memory model. The extended shared memory model is an integration of cache simulation (with timing information) and communication architecture simulation.** It controls the simulation order of IP core simulation and memory accesses to itself. In the next subsection, we give an example of the control. In section 5.2, we explain the extended shared memory model in detail.

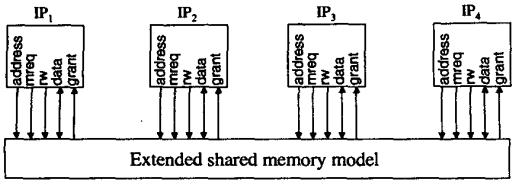

Figure 6: An architecture of IP cores and the extended shared memory model.

5.1 Scheduling IP Core Simulation and Memory Accesses

The extended shared memory model controls the simulation order of IP core simulation and memory accesses to itself by scheduling itself, memory accesses, and simulation events going to the simulation models of IP cores. Figure 5 illustrates a case of the scheduling. Assume that two IP cores IP_1 and IP_2 are running and communicating with each other via caches and shared memory. First, simulation of two IP cores run independently until each IP core performs a memory access. Figure 5 (a) shows that IP_1 performs a read access at time T_0 and IP_2 starts to perform three consecutive memory write accesses at time T_1 ($T_0 < T_1$). Assume that IP_1's read access yields a read miss and the state of victim cache block is **invalid**. Since there is no cache occupying the bus at time T_0, the cache of IP_1 can start to read a cache block from the other cache or shared memory at that time. In this case, the memory read access is delayed (by the extended shared memory model) until time T_2 by the read miss penalty cycles. At time T_2, the read access is performed, then simulation of IP_1 resumes.

Assume that the memory access of IP_2 yields a write miss and the state of victim block is **valid**. Since the bus is occupied by the cache of IP_1 at time T_1, the memory access of IP_2 is delayed (by the extended shared memory model) until time T_2 when the bus is released. At time T_2, it is delayed again until time T_3 by the write miss penalty cycles, i.e. the sum of write-back delay and read delay for CO_RD_INV type. At time T_3, three consecutive memory write accesses start to be performed. The scheduling continues in this way. The extended shared memory model implements the scheduling.

Each IP core is assumed to initiate a memory access by making events on its address bus, **address** and memory request signals, **mreq** and **rw**. When granted by **grant** signal, it reads/writes a data item from/to the extended shared memory model via its data bus, **data**. Figure 6 shows the architecture consisting of four IP cores of CD2DAT system and the extended shared memory model. Compared with the one in Figure 3, the dashed box in Figure 3 is replaced by the extended shared memory model in Figure 6. Simulation of the whole system consisting of simulation models of IP cores and the extended shared memory model can be performed in event-driven simulation. In our implementation, we use the SystemC simulation environment as the event-driven simulator.

5.2 An Extended Shared Memory Model

Figure 7 shows the pseudo code of extended shared memory model. We assume that each IP core has its own cache and caches are connected by a single shared bus.[5] When an IP core initiates a memory access by events on its address bus and control signals (mreq and rw), a structure called MA (abbreviation of memory access) is constructed (line 1). The MA is represented with a tuple <**IP, address, rw, victim_block_state, delay, ts**> where **IP, address,** and **rw** represent the IP core that initiates the memory access, the accessed address, and the direction of memory access (read or write).

[5] Considering some IP cores that have not caches is straightforward in our model since all the memory accesses by them have only to be considered to be misses to virtual caches with no data storage. Considering extended shared memory models for partitioned or hierarchical bus architectures with caches can be our future work.

```
1  if IP[i].addr.event(), then insert(QueueMA, new MA(IP[i], IP[i].addr, IP[i].rw));
2  if IP[i].data.event(), then MEM[IP[i].addr] = IP[i].data;
3  while(1) {
4     CurrMA = GetHighestPriority(QueueMA); // dequeue CurrMA
5     if CurrMA.ts > CurrTime, then { schedule(itself, CurrMA.ts); return; }
6     switch ( check_hit(CurrMA) ) {
7     case read_hit || write_hit && CurrMA.block_state == Exclusive :
8        schedule(CurrMA.IP.grant, '1', CurrTime + CurrMA.delay);
9        if read_hit, then
10          schedule(CurrMA.IP.data, MEM[CurrMA.addr], CurrTime + CurrMA.delay);
11    case other_write_hit :
12       if CurrMA.ts >= NextBusFreeTime, then
13          schedule(CurrMA.IP.grant, '1', CurrTime + INVDelay + CurrMA.delay);
14          NextBusFreeTime = CurrTime + INVDelay;
15       else, CurrMA.ts = NextBusFreeTime; insert(QueueMA, CurrMA);
16    case read_miss :
17       if CurrMA.ts >= NextBusFreeTime, then
18          if CurrMA.victim_block_state == Modified, then
19             schedule(CurrMA.IP.grant, '1', CurrMA.IP.data, MEM[CurrMA.addr],
20                      CurrTime + WBDelay + CO_RDDelay + CurrMA.delay);
21             NextBusFreeTime = CurrTime + WBDelay + CO_RDDelay;
22          else
23             schedule(CurrMA.IP.grant, '1', CurrMA.IP.data, MEM[CurrMA.addr],
24                      CurrTime + CO_RDDelay + CurrMA.delay);
25             NextBusFreeTime = CurrTime + CO_RDDelay;
26       else, CurrMA.ts = NextBusFreeTime; insert(QueueMA, CurrMA);
27    case write_miss :
28       if CurrMA.ts >= NextBusFreeTime, then
29          if CurrMA.victim_block_state == Modified, then
30             schedule(CurrMA.IP.grant, '1', CurrTime + WBDelay +
31                                            CO_RD_INVDelay + CurrMA.delay);
32             NextBusFreeTime = CurrTime + WBDelay + CO_RD_INVDelay;
33          else
34             schedule(CurrMA.IP.grant, '1', CurrTime + CO_RD_INVDelay +
35                                           CurrMA.delay);
36             NextBusFreeTime = CurrTime + CO_RD_INVDelay;
37       else, CurrMA.ts = NextBusFreeTime; insert(QueueMA, CurrMA);
38    }
39    UpdateMemoryBlockInformation(CurrMA);
40 }
```

Figure 7: Pseudo code of the extended shared memory model.

Victim_block_state is the state of victim cache block to be replaced by the memory access. **Delay** is the delay of memory access if there are no read/write miss and no write invalidation associated with it. **Ts** is a timestamp and is set to CurrTime (the current simulated time of the system) when the MA is constructed. After constructed, the new MA is inserted into a queue QueueMA. MA's in the queue are sorted in the increasing order of timestamp.

In line 4, GetHighestPriority() returns the MA having the highest bus access priority among the earliest MA's. A bus access priority is assigned (by the designer) to each cache attached to an IP core and, correspondingly, to the memory accesses by the cache. In line 5, if the timestamp of the earliest MA is later than CurrTime, the extended memory model schedules itself at the timestamp. In line 6, the current memory access is tested if it gives cache hit or miss. For the test, the information of address block (**flags** that indicate which cache has the copy of the block and **states** of the block in the caches) are managed. From line 7 to the end of the pseudo code, for each case of the test results, events (on **grant** and **data**) to be sent to the IP core (CurrMA.IP) are scheduled or currMA is scheduled again at a future time in the case that the memory access cannot be executed due to bus conflict. To check to see if the bus is available, a variable NextBusFreeTime is compared with the timestamp of CurrMA.

In the case of read_hit or write_hit to the block with state **exclusive** (line 7), since there is no bus access required by the cache, CurrMA.IP is granted after CurrMA.delay. In case of memory read access, the contents of the memory location (MEM[CurrMA.address]) is scheduled to be placed on the data bus of the IP core after CurrMA.delay. To schedule the event in the event queue of the event-

Table 2: Runtime comparison.

cache configurations. (b,s,a)'s	simulation runtime (second)		
	CA	proposed	speedup
CDMA modem			
(2,16,4) (4,16,1) (128,16,4)	15.0	0.46	32.6
(32,16,4) (64,16,2) (16,16,4)	10.5	0.29	36.2
(2,8,4) (1,8,2) (32,8,4)	14.6	0.47	31.1
(64,8,4) (128,8,2) (1,8,4)	9.9	0.27	36.7
CD2DAT			
(2,16,1) (8,16,2) (4,16,1) (8,16,2)	32.0	3.48	9.2
(32,1,4) (128,1,1) (128,1,4) (8,1,2)	5.8	0.64	9.0
(2,2,4) (8,2,2) (2,2,4) (32,2,1)	30.7	3.64	8.4
(64,8,4) (1,8,1) (128,8,2) (4,8,1)	17.4	2.51	6.9

driven simulator, function **schedule()** is used (line 10). Note that, in the extended shared memory model, memory contents have only to exist only in a single memory array (in the pseudo cose, MEM[]), without managing the memory contents of individual caches.

In line 11, **other_write_hit** case represents write hit to a cache block with state **modified**. If the bus is available (line 12), then grant signal is scheduled after INVDelay (write invalidation delay) and CurrMA.delay (line 13). In line 14, NextBusFreeTime is incremented by INVDelay.[6] Note that the cache holds the shared bus only for the time interval of INVDelay (not "INVDelay + CurrMA.delay") and the actual operation that the IP core writes (reads) a data item to (from) the cache (i.e. the extended shared memory model) is done at the boundary between the IP core and the cache without occupying the shared bus. When the IP core is granted, it writes a data item to the data bus. The event on the data bus is detected and the contents at the memory location is updated in line 2. In line 15, if the shared bus is not available, then CurrMA is scheduled again at NextBusFreeTime.

In line 18, if the state of victim block is **modified**, the accessed data is scheduled to be placed on the data bus of IP core after WBDelay (write-back delay) + CO_RD_Delay (delay of new line read) + CurrMA.delay (line 19 and 20). If the state is not **modified** (i.e. **clean** or **invalid**), then WBDelay is not included in the delay calculation (line 23 and 24). In the case of **write_miss** (from line 27 to line 37), the difference between read_miss case is CO_RD_INVDelay (delay of new line read with CO_RD_INV type message) is used instead of CO_RD_Delay. After a memory access is performed, the information of accessed memory block (e.g. flags and states) is updated in line 39.

5.3 Experiments

Since the specification of an invalidation-based cache coherency protocol [17] has been modeled in the extended shared memory model, it gives accurate cycle counts. Thus, we compare the runtimes of two types of simulation : (1) simulation of behavioral models of IP cores (with annotated delay) and the cycle-accurate communication modules and (2) simulation of behavioral models of IP cores (with annotated delay) and the extended shared memory model. Table 2 compares runtimes for several cache configurations used in Table 1 and shows the speedup of using the extended shared memory model compared to the case that the cycle-accurate communication modules are simulated. In the two examples, we performed more experiments whose details are not shown in this paper due to the page limit and obtained simulation speedup, $25 \sim 45$ times (CDMA) and $7 \sim 9$ times (CD2DAT).

The speedup comes from simulating the multiple-cache shared-bus communication architecture at a higher abstraction level with the extended shared memory model. For instance, the extended shared memory model eliminates data movement and transfer of four types of messages between caches, which are frequent in the operation of multiple-cache systems. It also eliminates snoopying (for cache coherency check) that requires time-consuming operations such as comparison of address tags. The speedup depends on the relative simulation workload required by IP cores and the extended shared memory model.

[6] Delay values (in our case, INVDelay, WBDelay, CO_RDDelay, and CO_RD_INVDelay) associated with the cache coherency protocol can be easily characterized by the protocol and implementation specification.

6 Conclusion

We have presented a case study to show the effects of interdependency between cache configuration and system behavior. We also presented an extended shared memory model for fast and accurate performance estimation of multiple-cache IP-based system. The method gives accurate performance estimation and yields significant performance improvement in terms of simulation runtime compared with the cases that the cycle-accurate models of communication modules are simulated.

References

[1] K. Lahiri, A. Raghunathan, and S. Dey, "Fast Performance Analysis of Bus-Based System-On-Chip Communication Architectures", *Proc. Int. Conf. on Computer Aided Design*, Nov. 1999.

[2] K. Lahiri, A. Raghunathan, and S. Dey, "Performance Analysis of Systems with Multi-Channel Communication Architectures", *Proc. Int'l Conf. on VLSI Design*, Jan. 2000.

[3] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey, "Communication Architecture Tuners: A Methodology for the Design of High-Performance Communication Architectures for System-on-Chips", *Proc. Design Automation Conf.*, June 2000.

[4] M. D. Hill and A. J. Smith, "Evaluating Associativity in CPU Caches", *IEEE Transactions on Computers*, pp. 1612–1630, Dec. 1989.

[5] W. Wang and J. Baer, "Efficient Trace-driven Simulation Methods for Cache Performance Analysis", *Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pp. 27–36, May 1990.

[6] A. J. Smith, "Two Methods for the Efficient Analysis of Memory Address Trace Data", *IEEE Transactions on Software Engineering*, pp. 94–101, Jan. 1977.

[7] A. Agarwal and M. Huffman, "Blocking: Exploiting Spatial Locality for Trace Compaction", *Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pp. 48–57, May 1990.

[8] Qualcomm, Inc., "CDMA System Engineering Training Handbook", 1993.

[9] S. Yoo, J. Lee, J. Jung, K. Rha, Y. Cho, and K. Choi, "Fast Prototyping of an IS-95 CDMA Cellular Phone : a Case Study", *Proc. the 6th Conference of Asia Pacific Chip Design Languages*, pp. 61–66, Oct. 1999.

[10] Y. Li and J. Henkel, "A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems", *Proc. Design Automation Conf.*, pp. 188–193, June 1998.

[11] M. Lajolo, L. Lavagno, and A. Sangiovanni-Vincentelli, "Fast Instruction Cache Simulation Strategies in a Hardware/Software Co-Design Environment", *Proc. Asia South Pacific Design Automation Conference*, Jan. 1999.

[12] Z. Wu and W. Wolf, "Iterative Cache Simulation of Embedded CPUs with Trace Stripping", *Proc. Int. Workshop on Hardware-Software Codesign*, pp. 95–99, May 1999.

[13] T. D. Givargis, F. Vahid, and J. Henkel, "Interface and Cache Power Exploration for Core-Based Embedded System Design", *Proc. Int. Conf. on Computer Aided Design*, Nov. 1999.

[14] R. L. Lysecky, F. Vahid, T. D. Givargis, and R. Patel, "Pre-fetching for Improved Core Interfacing", *Proc. Int. Symposium on System Synthesis*, Nov. 1999.

[15] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Efficient Power Co-Estimation Techniques for System-on-chip Design", *Proc. Design Automation and Test in Europe*, Mar. 2000.

[16] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, and A. Sangiovanni-Vincentelli, "Modeling Shared Memory Access Effects during Performance Analysis of HW/SW Systems", *Proc. Int. Workshop on Hardware-Software Codesign*, Mar. 1998.

[17] B. Catanzaro, *Multiprocessor System Architecture, A Technical Survey of Multiprocessor/Multithreaded Systems using SPARC, Multi-level Bus Architectures and solaris (SunOS)*, Sun Microsystems, 1994.

[18] TIA/EIA-95A, "Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular Systems", 1995.

[19] Synopsys, Inc., "SystemC Reference Manual, Release 0.9", available at http://www.systemc.org/.