Integrating Cache Coherence Protocols for Heterogeneous Multiprocessor Systems, Part 2

EXPERIMENTS WITH ACTUAL HETEROGENEOUS MULTIPROCESSOR PLATFORMS ON A SHARED-BUS MEASURE THE EFFECTIVENESS OF TWO CACHE COHERENCE TECHNIQUES. THIS INTEGRATION APPROACH, SNOOP-HIT BUFFER, AND THE ACCOMPANYING REGION-BASED CACHE COHERENCE APPROACH YIELD SIGNIFICANT SPEEDUPS COMPARED TO A PURE SOFTWARE SOLUTION.

Taeweon Suh
Hsien-Hsin S. Lee
Douglas M. Blough
Georgia Institute of Technology

Today, the heterogeneous demands of embedded-systems applications naturally lead to heterogeneous multiprocessor system-on-chip (SoC) platform designs based on intellectual property (IP) cores. Although the integration of heterogeneous processors on SoCs is an attractive way to flexibly provide diverse and targeted functionality, such designs become complex in terms of maintaining cache coherence in shared-memory architectures.

In part l of this article, we presented integration techniques for cache coherence among heterogeneous multiprocessors on a shared-bus. These techniques include read-to-write conversion and/or shared-signal assertion and deassertion. We also had proposed a snoop-hit buffer (to boost performance) and region-based cache coherence (to recover the lost protocol states).[1] In this part of the article, we present two examples of integrating heterogeneous processors and show the limitation of integrating processors without native support for cache coherence. Finally, we discuss the Verilog simulation results of applying our techniques to actual heterogeneous multiprocessor platforms.

Case studies

Here, we present two implementations using commercially available embedded processors: the PowerPC 755, a write-back-enhanced Intel 486, and an ARM920T. The PowerPC 755 processor uses the MEI protocol, and Intel 486 supports a modified MESI protocol. The ARM920T does not offer native support for cache coherence. The examples of this case study focus purely on maintaining cache coherence. Thus, we selected this combination of processors solely to illustrate how to apply our techniques to the integration of actual heterogeneous processors.

A multiprocessor platform employs a shared bus for data transactions between main mem-

Published by the IEEE Computer Society

ory and processors. Companies proposed several bus architectures for SoCs; these included IBM's CoreConnect (http://www.chips.ibm.com/products/coreconnect), Palmchip's CoreFrame,[2] and ARM's Advanced Microcontroller Bus Architecture (AMBA). A common characteristic among these architectures is that they use two separate pipelined buses: one each for high- and low-speed devices. In this article, we study the Advanced System Bus (ASB), an AMBA bus, in the integration of the ARM920T with the PowerPC 755 as the shared-bus protocol. AMBA is one of the most popular bus protocols used in embedded system designs. We chose ASB because the ARM920T's Advanced High-Performance Bus (AHB) was unavailable in Seamless processor models.

As Figure 1a shows, the schematic diagram illustrates the integration of a Power PC755 and an Intel 486, representing a case of the PF3.[1] Wrappers are necessary for the protocol conversions between the processors' interfaces and the bus, in addition to the read-to-write conversion. On the PowerPC 755 side, the read-to-write conversion is unnecessary because the S state is not present in the state machine. On the Intel 486 side, however, we should remove the S state by asserting the INV (invalidation request) input signal, a cache coherency protocol pin. The Intel486 cache controller samples the signal on snoop cycles. If INV is asserted, the cache controller invalidates an addressed cache line in the E or S state. If a cache line is in the M state, the line is drained to memory. Normally, INV is deasserted on read snoop cycles and asserted on write snoop cycles. However, to remove the S state, it should be asserted on both read and write snoop cycles.

In the Intel 486's cache, cache lines are defined as either write-back or write-through at allocation time in enhanced bus mode, depending on the WB/WT (write-back/write-through) pin status. Only write-through lines can have the S state, and only write-back lines can have the E state. Therefore, the protocol for write-through lines is the SI protocol, while the protocol for write-back lines is the MEI protocol.

When a snoop hit occurs on the M state line of the Intel 486 cache, the HITM (hit/miss to a modified line) output signal is asserted, and, correspondingly, the wrapper around the PowerPC 755 informs the core of a snoop hit by asserting the ARTRY (address retry) input signal. Then, the PowerPC 755 immediately yields the bus mastership to Intel486 so that the cache controller in the Intel 486 drains the modified line to memory. When a snoop hit occurs on the M state line of the PowerPC 755's data cache, the PowerPC 755 asserts the ARTRY output signal. The arbiter then immediately asserts BOFF (backoff) so Intel 486 yields the bus mastership to the PowerPC755. Then, the cache controller in the PowerPC 755 drains the modified line to memory.

Figure 1b shows another example of a heterogeneous platform using PowerPC 755 and ARM920T representing a case of PF2.[1] The same methodology used in the ARM920T is applicable to PF1.[1] The wrapper in the figure converts the PowerPC bus protocol to the ASB protocol, and vice versa. The wrapper also

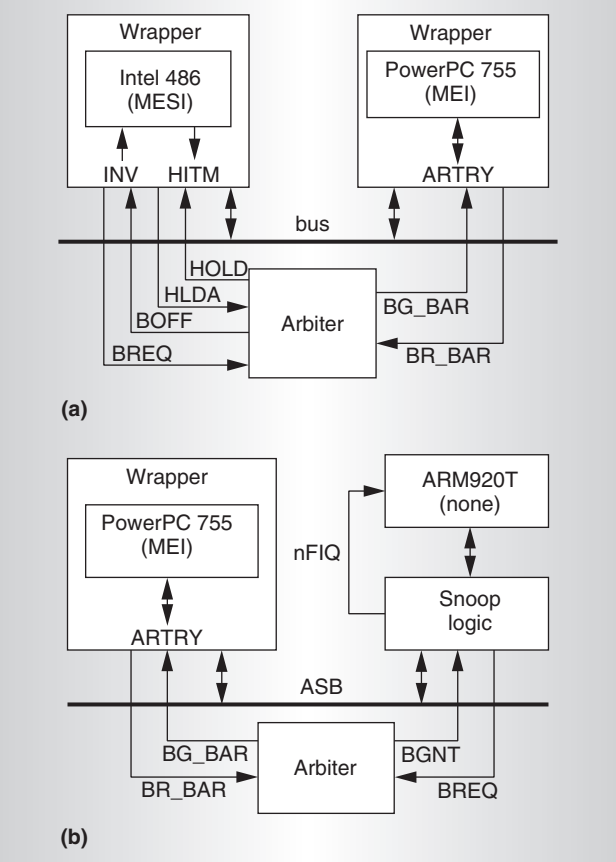

Figure 1. Coherence in the PowerPC 755 and Intel 486 (a); and PowerPC 755 and ARM920T (b).

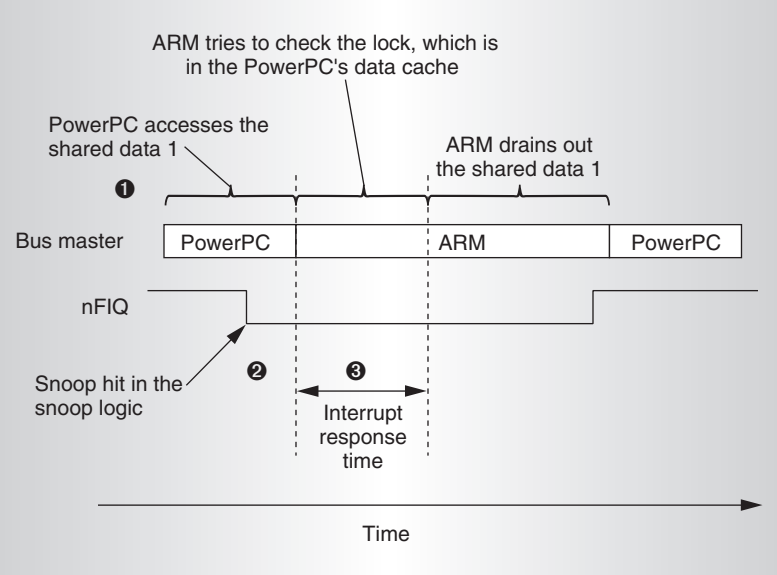

Figure 2. Hardware deadlock problem.

allows the PowerPC 755 to monitor the bus transactions generated by the ARM920T. The snoop logic provides snooping capability for the ARM920T, which does not have native cache coherence support. It keeps track of all the address tags of the ARM920T's data cache inside a tag content-addressable memory (or tag CAM) by monitoring bus transactions initiated by the ARM920T. Our preliminary synthesized result using a 0.18-µm TSMC library shows that the snoop logic occupies 11.18 percent of the fully customized ARM920T area, supporting 16-Mbytes of shared memory. When the tag of a requested address generated by the PowerPC 755 matches an entry of the tag CAM, it triggers a snoop hit to the ARM920T by asserting nFIQ (fast interrupt). An interrupt service routine is responsible for draining the snoop-hit cache line if the line is modified, or invalidating it if the line is clean.

Even though this architecture can make caches coherent, there is a limitation when at least one of the processors in a heterogeneous processor platform does not have native cache coherence hardware such as in the case of ARM920T. PF1 and PF2 pertain to this category, and Figure 2 illustrates the problem. Suppose that the architecture allows lock variables and shared data 1 to be cached, and the shared data 1 is currently in the data cache of the ARM920T. After acquiring the lock, the PowerPC tries to access the shared data 1 as shown in step ❶. Therefore, a snoop hit occurs in the snoop logic, and nFIQ is asserted as illustrated in step ❷. Then, the ARM processor is supposed to drain or invalidate the addressed cache line in the interrupt service routine. However, because most commercial processors implement pipelining and precise interrupt, the ARM might or might not respond to the interrupt immediately. During the interrupt response time, as shown in step ❸, ARM might check the lock to see whether it has been released. Lock variables are currently in the PowerPC's data cache, because the PowerPC accessed the lock lately. Therefore, PowerPC should drain the cache line storing the lock variables to memory. However, if the PowerPC gains the bus mastership, it is supposed to retry the transaction, which it did in step ❶, instead of draining the lock variables. We call this situation *hardware deadlock*.

There are two solutions to preventing the hardware deadlock problem. The first option is not to cache the lock variables. The other alternative is to use the SoCLC discussed in the first part of this article. For the first solution, a software lock, such as the Bakery algorithm, can be used as a lock mechanism even though it is inefficient from a performance standpoint. For the second solution, it needs a simple lock module sitting on a bus, as explained in the lock mechanism section in part 1 of this article. Since the lock variables are not cached in either case, the hardware deadlock does not occur.

Even though we focus our discussion on a lock variable, the same problem can occur in critical sections where applications implement multiple locks in a system. For this reason, a system can have only one lock in PF1 or PF2, requiring that the program perform all shared-variable operations within critical sections.

Performance evaluation

Hardware simulation demands an enormous amount of time to run real applications. We experimented with the Verilog simulation of a MPEG decoding application on a two-processor platform with three small frames. Using the simulation environment listed in Table 1, the simulation took more than three days to finish on a SUN UltraSparc workstation, making a complete evaluation of our approach too time-consuming. Therefore, we designed a suite of microbenchmark programs

Table 1. Simulation environments.

Environments	Description
Simulators	Seamless CVE, ModelSim
Operating frequencies	
PowerPC 755	100 MHz*
ARM920T	50 MHz*
ASB	50 MHz*
Instruction and data caches	Enabled
Memory access time	
Single word	7 cycles**
Burst (8 words)	7 cycles for the first word, 1 cycle for each subsequent word**

* These low operating frequencies are because of the limitation of simulation models. We expect similar results for simulations with higher operating frequencies.

** This memory access latency varies as described in this article.

to evaluate our methodology's impact.

Performance of integration techniques

Our microbenchmark suite consists of a worst-case scenario (WCS), a typical-case scenario (TCS), and a best-case scenario (BCS). In these programs, one task runs on each processor. Each task intensively tries to access a critical section (shared memory) protected by the SoCLC lock mechanism. Once a task acquires the lock, it accesses shared data quantified by cache lines and modifies them before exiting the critical section. We implemented the microbenchmarks in a way that each task acquires the lock alternatively, which means the simulation assumes the worst-case situation for lock acquisition and releasing.

We used a *pure software solution* as our baseline system, in which the programmer is responsible for draining or invalidating all the shared data in the critical section before exiting. Because the lock mechanism protects the critical sections, users should know which shared data is in use in the critical sections. To flush all of the used shared data in the critical sections in the simulations, we recursively used "asm volatile("mcr p15, 0, %0,c7, c14, 1"::"r"(addr))" for the ARM920T and "asm volatile ("dcbf 0, %0"::"r"(addr))" for the PowerPC 755. The Atalanta RTOS also can enable the cache flush instructions for draining all touched shared data at the end of each critical section, in case there is no hardware coherence support.

We call the proposed solution with the read-to-write conversion and shared signal assertion/deassertion the *simple hardware approach*. We refer to the use of a snoop-hit buffer in addition to a simple hardware approach as the *snoop-hit buffer approach*. Table 1 summarizes the simulation environments and the hardware configurations. We use two- (PowerPC 755 and ARM920T) and four-processor (three PowerPC 755s and one ARM920T) platforms to quantify the performance.

The Intel 486 and PowerPC 755 platform should outperform the PowerPC 755 and ARM920T platform due to the absence of an interrupt service routine. (We are unable to report the results of the PowerPC 755 and Intel 486 system because the Seamless processor model for the Intel 486 doesn't fully support the coherence functionality.)

We simulated and measured the performance of our proposed approaches and the baseline, using hardware-software cosimulations. The Seamless and ModelSim from Mentor Graphics were used as simulators. We varied the memory latency from 7-1-1-1-1-1-1-1, 13-2-2-2-2-2-2-2 ... to 97-9-9-9-9-9-9-9. The string 7-1-1-1-1-1-1-1 means 7-cycle access time for the first word and 1-cycle access time for each seven trailing words (a cache line is eight words). The miss penalty indicated by the *x*-axis in all the figures represents the latency required for fetching the entire cache line, that is, all eight words.

WCS performance. Figure 3 shows the WCS results. In the WCS, each task keeps accessing the same blocks of memory. Figure 3a shows simulation results on the two-

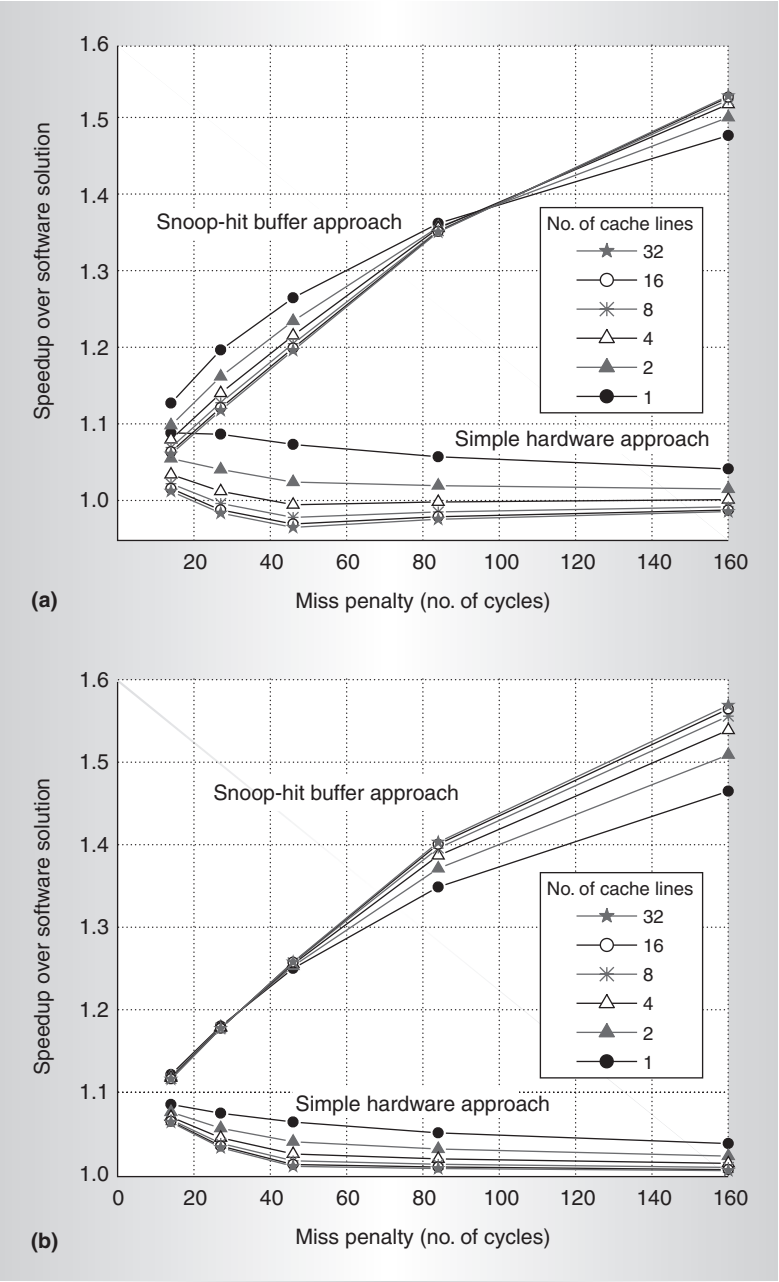

Figure 3. Worst-case scenario (WCS) results on platforms with two (a) and four (b) processors.

processor platform, where the simple hardware approach performs better than the pure software solution with few exceptions. These exceptions come from cache line replacements and/or interrupt processing overheads that vary as the miss penalty changes. The simulation with the snoop-hit buffer approach shows at least a 6.3 percent performance improvement against the software solution for all WCS simulations. As the miss penalty increases, the performance of the snoop-hit buffer approach increases dramatically. The simulation result shows up to a 53.4 percent performance improvement when the miss penalty equals 160 cycles and the number of accessed cache line is 32.

Figure 3b shows simulation results on the 4-processor platform, where the simple hardware approach always has better performance (at least 0.97 percent improvement) with no exceptions, since only one processor needs the interrupt service routine. The simulation with the snoop-hit buffer approach shows an 11.8 to 57.1 percent performance improvement compared to the pure software solution.

BCS performance. In the BCS, each processor accesses different critical sections, which means that snoop-hits do not occur. In the pure software solution, each processor has to drain all the accessed shared data before exiting the critical section. However, in the proposed solution, this is unnecessary.

Figure 4a shows the simulation results on the two-processor platform, which achieved a 49.2 percent performance improvement with a 14-cycle miss penalty and one accessed cache line. The speedup increases as the miss penalty and/or the number of accessed cache lines increases. Simulation with 32 cache lines shows a 407 percent performance improvement with a 160-cycle miss penalty, compared to the pure-software solution. Because snoop hits do not occur in BCS, the simple hardware and snoop-hit buffer approaches show the same results. Figure 4b shows the simulation results on the four-processor platform, which achieved a performance improvement of from 51 to 426 percent.

TCS performance. In the TCS, each task randomly picks up shared blocks of memory from among 10 blocks before entering into the critical section. Figure 5a (on p. 76) shows the simulation results on the two-processor platform. The simple hardware approach shows a 21.7 to 54.2 percent performance improvement, and the snoop-hit buffer approach shows a 24.5 to 214 percent performance improvement compared to the pure software solution. Figure 5b shows the simulation results on the four-processor platform. The

simple hardware approach shows a 27 to 68.6 percent performance improvement, and the snoop-hit buffer approach shows a 46.4 to 226 percent performance improvement over that of the pure-software solution.

RBCC performance

We evaluated RBCC performance with two benchmarks, Microbench and the RTOS model, using the hardware platform described in part 1 of this article, Figure 3. Similar to the work described in the previous section, Microbench consists of WCS, BCS, and TCS benchmarks. In these benchmarks, one task runs on each processor, and each task accesses the same memory blocks after acquiring the lock of SoCLC. We use a system *without RBCC* as the baseline.

WCS performance. In the WCS, the three ARM processors—which use the MESI protocols—keep writing to the same block of memory, while the PowerPC 755 executes an idle task. Because processors keep writing to the same memory blocks after acquiring the lock, the S state in the MESI protocol does not affect performance. Thus, in Figure 6a (on p. 77), RBCC shows the same performance as the without-RBCC system (speedup = 1). However, the snoop-hit buffer approach dramatically enhances performance because every snoop hit takes advantage of this buffer. The simulation shows a 2.1 to 56.9 percent performance improvement as the number of accessed cache line and/or the miss penalty increases.

BCS performance. In the BCS, the three ARM processors keep reading the same block of memory while the PowerPC 755 executes an idle task. Without RBCC, since the integrated coherence protocol should be MEI, whenever a processor reads blocks of memory, other processors should invalidate the cache lines, if they have cached them. However, with RBCC, they need not invalidate these cache lines, because the MESI protocol has the S state.

The simulation results in Figure 6b show a 13 percent performance improvement with a 14-cycle miss penalty and one accessed cache line. The speedup increases as the miss penalty and/or the number of accessed cache lines increases. The simulation with 32 cache lines shows a 3.06× speedup against the without-RBCC system with a 160-cycle miss penalty. Because a snoop hit (which occurs when a processor tries to read or write modified cache lines of other processors) never occurs, the snoop-hit buffer does not affect performance.

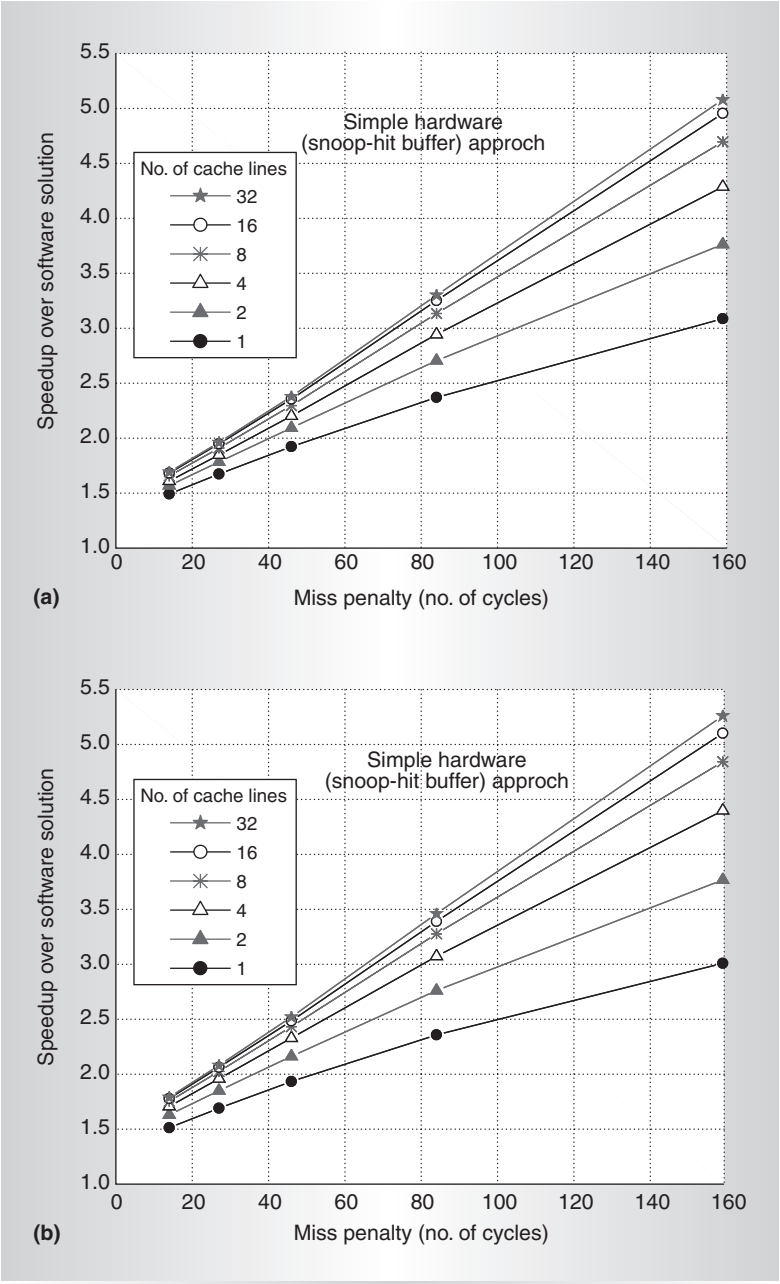

Figure 4. Best-case scenario (BCS) results on platforms with two (a) and four (b) processors.

TCS performance. In the TCS, all four processors access the MEI memory area, and the three ARM processors access the MESI mem-

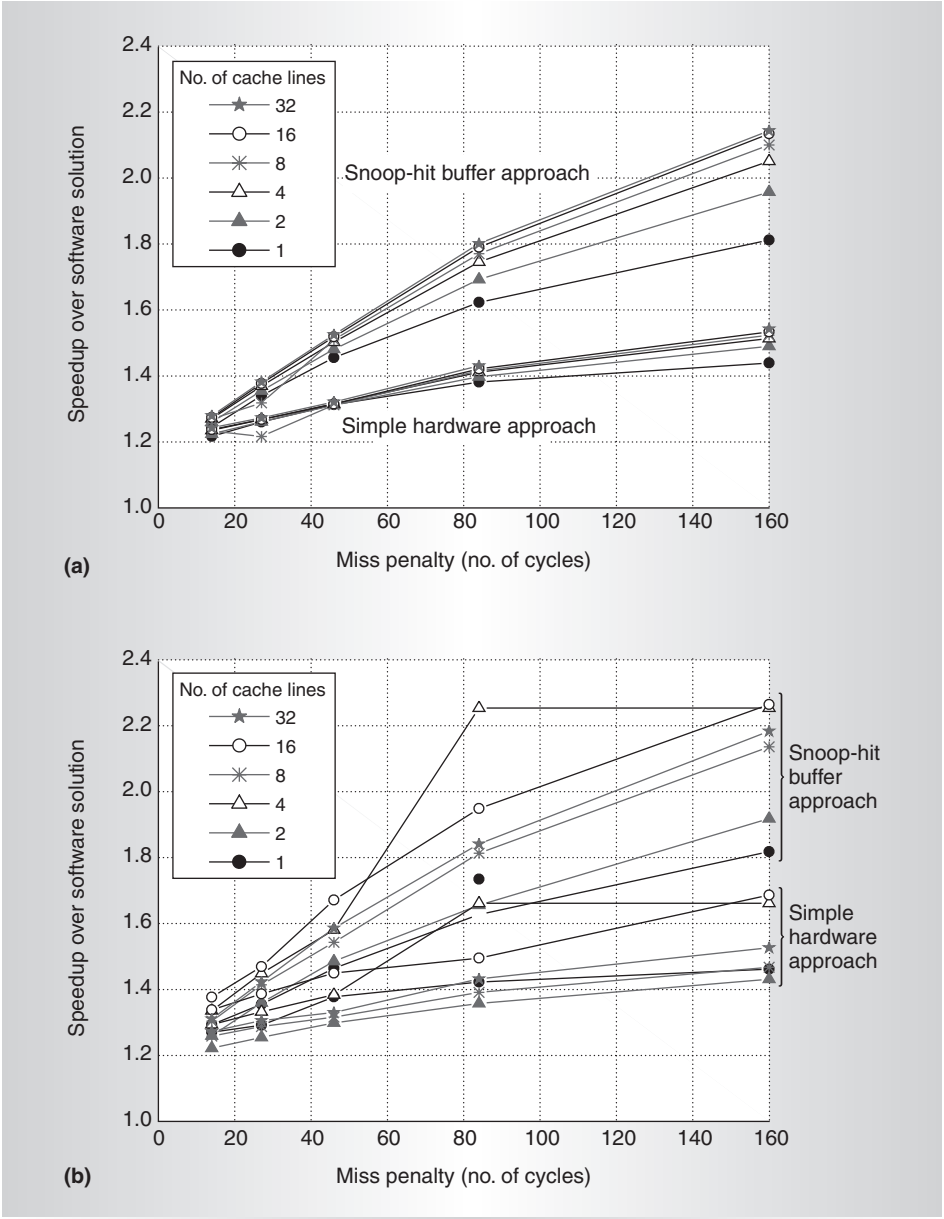

Figure 5. Typical-case scenario (TCS) results on platforms with two (a) and four (b) processors.

ory area. The ARM processors access MEI or MESI memory area with a 50 percent probability each. The simulation assumes each ARM processor performs read operations 80 percent of the time, thus providing an opportunity to use the S state around 50 percent of the time (0.80^3).

Figure 6c shows the simulation results. RBCC shows a 0.5 to 11.4 percent performance improvement compared to the without-RBCC system. The snoop-hit buffer approach without RBCC enhances performance by 2.1 to 19.6 percent. RBCC with the snoop-hit buffer approach increases performance from 2.7 to 36.4 percent.

RTOS kernel performance. We also modeled another benchmark, part of the Atalanta RTOS kernel. As discussed in part 1 of this article, Atalanta was tailored for heterogeneous multiprocessor platforms. For interprocessor communication and synchronization, Atalanta provides shared-address space. Thus, processors sharing system objects, such as semaphores and mailboxes, can directly access the other processors' task control blocks (TCB).

Consider an example in which processor 1 and processor 2 share semaphore A, and processor 1 now has semaphore A, and processor 2 is waiting for it. When processor 1 is done with semaphore A and releases it, processor 1 updates and inserts the waiting tasks' TCB to the ready state by changing the state field in the TCB and then updates and inserts the TCB into the ready list of processor 2. Then, processor 1 generates an interrupt to processor 2, so processor 2 can reschedule tasks, and runs the highest-priority task. In Atalanta, tasks' TCBs on each processor are connected through a doubly linked list, according to the tasks' priorities. This simulation models and measures this task insertion and deletion mechanism in the Atalanta RTOS.

Using the same RBCC simulation platform of Figure 3 in part 1 of this article, the integration technique, we obtain the results in Figure 7, which shows the performance enhancement of RBCC as the miss penalty increases. The notation "2T-16T" stands for

two tasks running on each processor in the MEI protocol area with 16 tasks running on each processor in the MESI protocol area. A processor randomly selects two tasks to delete and insert into TCBs. After the deletion and insertion of the tasks' TCBs from the doubly linked lists, a processor that modified the TCB of other processor generates an interrupt to the processor owning the inserted task. Then, the interrupted processor repeats the procedure, that is, the insertion and deletion of two randomly selected tasks and the generation of an interrupt.

In the 2T-2T case of Figure 7, RBCC shows marginal performance improvement (0.4 to 0.9 percent) over the without-RBCC system. This comes from the short length of the doubly linked list: Only two tasks run on each processor in the MESI protocol area, so the list's length is two. The insertion and deletion in this short linked list demands the modification of fields in both lists, leading almost no usage of the S state. Thus, RBCC provides only marginal performance improvement. This marginal improvement comes from sharing the array to reference the first ready list of tasks on each processor.

However, the 2T-16T case shows an 11 to 29 percent improvement, because 16 tasks are connected through the doubly linked list in TCBs, and depending on the position of insertion and deletion, we modify the fields in only two or three TCBs. This leads to increased usage of the S state.

The snoop-hit buffer approach without RBCC shows a 4.4 to 41.1 percent improvement for the 2T-2T case, and a 2.9 to 26.1 percent improvement for the 2T-16T case. Finally, RBCC with the snoop-hit buffer enhances performance by 4.5 to 43.0 percent for the 2T-2T case, and by 15.3 to 77.0 percent for the 2T-16T case.

In parts 1 and 2 of this article, we propose a systematic integration technique to guarantee cache coherency for heterogeneous multiprocessor SoCs. As these performance results have shown, our methodology provides a viable and effective solution for integrating heterogeneous cache coherence protocols in a system.

Whereas our discussion has thus far focused on maintaining cache coherence in a shared-

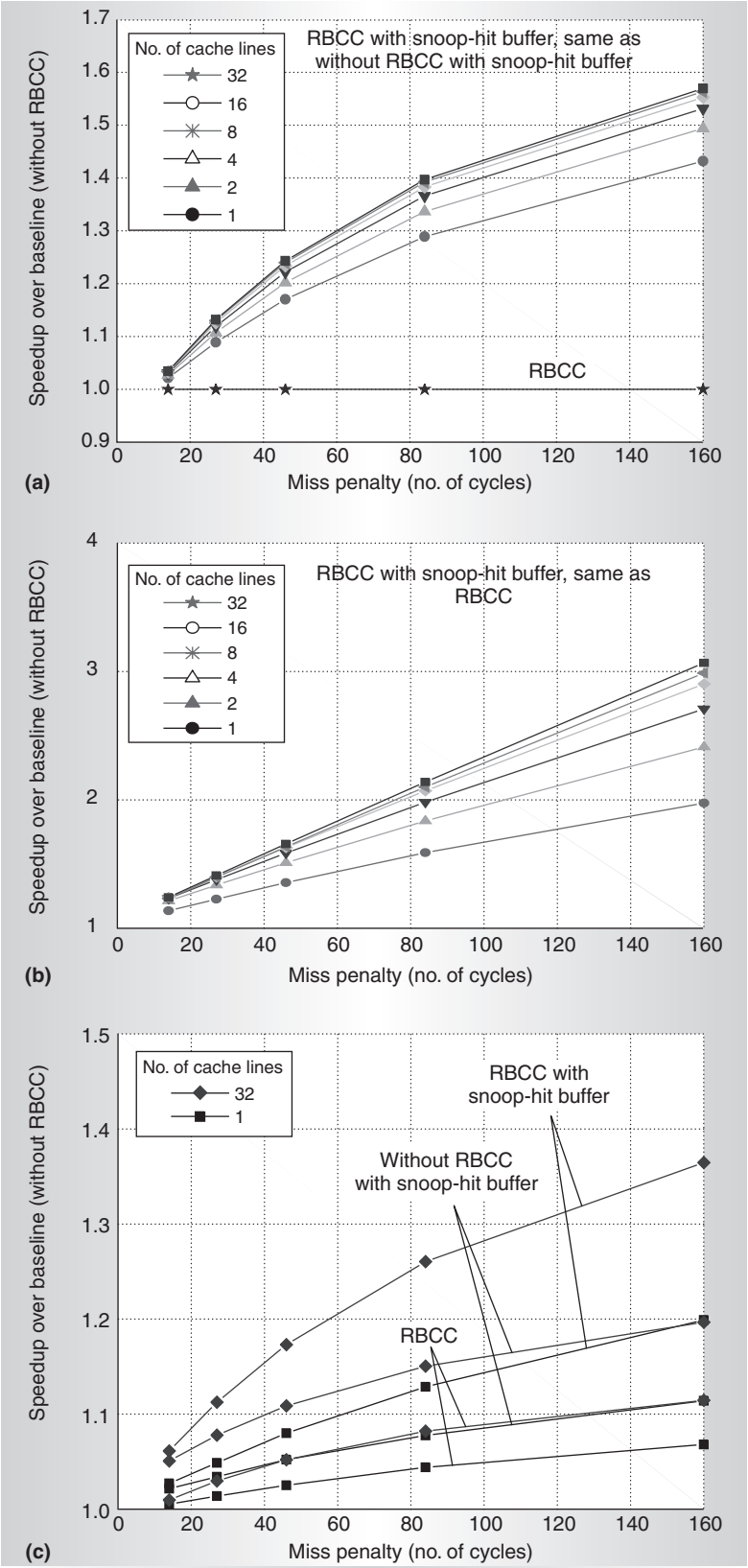

Figure 6. RBCC results for WCS (a), BCS (b), and TCS (c).

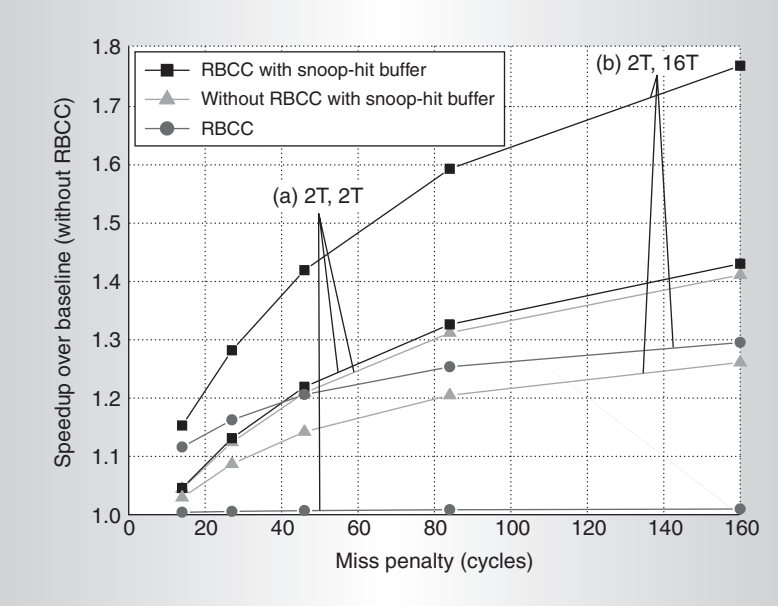

Figure 7. RTOS kernel simulation results.

bus architecture, today's SoCs are often based on a multiple-bus architecture. Even with multiple buses inside, SoCs wouldn't adopt dedicated memory interface for each processor because of high cost. For example, Texas Instrument's OMAP has a C55x DSP and ARM925T, and each processor has its own internal bus. However, they share an external DRAM interface. We are investigating an inexpensive solution to exploiting cache coherence on a non-shared bus, shared-memory architecture in SoCs.

References

1. T. Suh, H-H.S. Lee, and D.M. Blough, "Integrating Cache Coherence Protocols for Heterogeneous Multiprocessor Systems, Part 1," *IEEE Micro*, vol. 24, no. 4, Jul-Aug. 2004, pp. 33-41.
2. B. Gordan, "An Efficient Bus Architecture for System-on-a-Chip Design," *Proc. IEEE Custom Integrated Circuits Conf.* (CICC 99), IEEE Press, 1999, pp. 623-626.

Taeweon Suh is a PhD student in the School of Electrical and Computer Engineering, Georgia Institute of Technology. His research interests include embedded systems, computer architecture, DSPs, and networks. Suh has a BS in electrical engineering from Korea University and an MS in electronics engineering from Seoul National University. He is a student member of ACM.

Hsien-Hsin S. Lee is an assistant professor in the School of Electrical and Computer Engineering, Georgia Institute of Technology. His research interests include microarchitecture, low-power systems, design automation, and security. Lee has a BSEE from National Tsinghua University, Taiwan, and an MSE and a PhD in computer science and engineering from the University of Michigan. He is a member of ACM and IEEE.

Douglas M. Blough is a professor of electrical and computer engineering at the Georgia Institute of Technology. His research interests include parallel and distributed systems; dependability and security; and wireless ad hoc networks. Blough has a BS in electrical engineering and an MS and a PhD in computer science from Johns Hopkins University, Baltimore. He is a senior member of IEEE.

Send questions and comments to Taeweon Suh, Hsien-Hsin S. Lee, and Douglas M. Blough, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0250; {suhtw, leehs, doug.blough}@ece.gatech.edu.