Performance Evaluation and Optimization of Dual-Port SDRAM Architecture for Mobile Embedded Systems

Hoeseok Yang, Sungchan Kim, Hae-woo Park, Jinwoo Kim, and Soonhoi Ha
School of EECS, Seoul National University, Korea
{hyang, sungchan.kim, starlet, jwkim, sha}@iris.snu.ac.kr

ABSTRACT
Recently dual-port SDRAM (DPSDRAM) architecture tailored for dual-processor based mobile embedded systems has been announced where a single memory chip plays the role of the local memories and the shared memory for both processors. In order to keep memory consistency from simultaneous accesses of both ports, every access to the shared memory should be protected by a synchronization mechanism, which can result in substantial access latency. We propose two optimization techniques by exploiting the communication patterns of target application: lock-priority scheme and static-copy scheme. Further, by dividing the shared bank into multiple blocks, we enable simultaneous accesses to different blocks and achieve considerable performance gain. Experiments on a virtual prototyping system show a promising result that we achieve about 20-50% performance gain compared to the base DPSDRAM architecture.

Categories and Subject Descriptors
C.3 [**Computer Systems**]: Special-purpose and Application-based Systems – *Real-time and embedded systems.*

General Terms
Design.

Keywords
Memory architecture, dual-port SDRAM, mobile embedded system

1. INTRODUCTION
Mobile embedded systems support diverse multimedia functions like audio, video, and even 3D games, which never cease to demand more powerful computation capability. The typical architecture is a dual processor system that consists of a baseband processor and a powerful application processor. The baseband processor handles essential call-processing and modem functions while the application processor performs computation-intensive applications. Though an application processor is typically made of a powerful multi-core system-on-chip (SoC), we regard it as a single core processor in this paper.

Figure 1 shows three different dual-processor architectures. One is to reuse existing shared media, such as peripheral device bus and general purpose I/O ports, for communication. Figure 1(a) depicts an existent architecture where two processors communicate with each other through LCD bus. Although it provides a cheap solution by reusing a peripheral bus, it suffers from low bandwidth.

CASES'07, Sep. 30–Oct. 5, 2007, Salzburg, Austria.

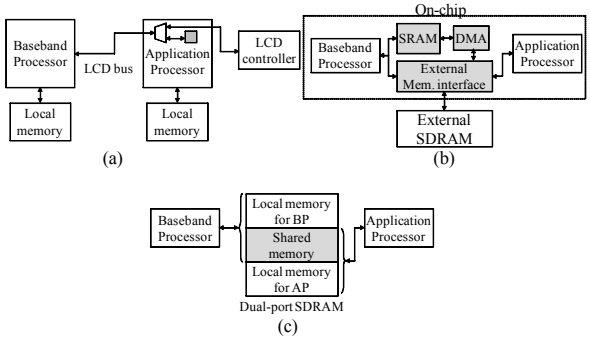

Figure 1. Various communication architectures for dual-processor: (a) communication using peripheral bus, (b) the MXC solution [1], and (c) the dual-port SDRAM (OneDRAM™ [6]).

Dual-port memory architecture is a promising solution to gain performance improvement. Many researches have focused on efficient inter-processor communications in multi-processor environment considering memory architecture optimizations [2][3] and multi-port memory [4][5] respectively. Recently, a novel architecture, called MXC (Mobile Extreme Convergence), has been introduced [1]. It consists of two processor cores, ARM1136™ and StarCore™ SC140 DSP, in a single chip. The communication between two processors is made through on-chip SRAM as depicted in Figure 1(b). Once the baseband processor writes a data to the on-chip SRAM, DMA conveys it to the external SDRAM that is accessible for the application processor. The processors share an external single-port SDRAM for their own local memory accesses, which means that the single-port interface still remains potential performance bottleneck. Since all components are implemented in a single chip, it solves the problem of increased package count.

Very recently, a new dual-port SDRAM device (DPSDRAM), OneDRAM™, has been announced by Samsung Electronics [6], which consists of one shared bank and two dedicated banks for both processors as illustrated in Figure 1(c). Two dedicated banks are served as local memory areas for processors. There is a special purpose bank (the grey box in the middle) for shared memory space for inter-processor communications. Compared with the MXC solution, it has two major advantages: (1) it provides larger shared space; (2) Contrary to a single chip solution like the MXC, there are no constraints on the kinds of processors so as to be easily applicable to various mobile embedded systems. Throughout the rest of the paper, we will refer the structure of OneDRAM™ to the base DPSDRAM architecture.

To share the same memory area, synchronization overhead should be paid. The DPSDRAM provides a hardware semaphore to give exclusive access to the shared region. Before accessing the shared region, the hardware semaphore should be obtained. Moreover, software semaphore is also managed for shared data structure. Since the shared space is usually set to non-cacheable, it turns out that the synchronization overhead may nullify the performance gain of the DPSDRAM architecture.

To reduce such synchronization overheads, we propose two optimization techniques in addition to the base DPSDRAM architecture: *lock-priority* scheme and *static-copy* scheme. The

optimizations are based on the observation that the characteristics of communication requirements of two processors are different. A typical scenario is that the baseband processor receives a 'big' data structure, for example a video frame, from the air and transfers it to the application processor that will process the data structure. Thus, the baseband processor triggers communication infrequently but with relatively big data once it happens. On the other hand, the application processor deals with relatively small data but frequently. This asymmetric duo gives the chance of optimization.

Whereas OneDRAM™ provides only single lock for the entire shared bank, we go further to a more general structure where the shared bank is divided into multiple sub-regions with separate hardware semaphores. Such architecture allows simultaneous accesses to different sub-regions in the bank for both processors and subsequently enables efficient and flexible operations as well as finer control on the shared bank. OneDRAM™ is a specific instantiation of a general DPSDRAM architecture with a single sub-region in the shared bank.

2. DUAL-PORT SDRAM ARCHITECTURE

2.1 Structural Overview

The structure of the base DPSDRAM architecture is shown in Figure 2. The baseband processor and the application processor are connected to a DPSDRAM that has two dedicated memory banks for the processors and a shared memory bank. In the figure, we explicitly draw the hardware semaphore that is included in the memory device. To avoid access conflicts, both processors should acquire the hardware semaphore prior to shared memory bank access. In the figure, solid lines denote data paths while dashed lines stand for control paths to convey hardware semaphore commands. Since the hardware semaphore resides inside the memory device and address-mapped, all commands of a processor are passed by the data bus.

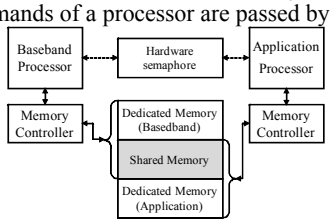

Figure 2. Structure of the base DPSDRAM architecture.

2.2 Inter-Processor Communication

While the hardware semaphore supports physical mutual exclusion, it is yet insufficient. We need to provide software semaphore for logical mutual exclusion. Since the software semaphores should be also shared between processors, they are located in the shared bank. Therefore, we have to pay the overhead of grasping a hardware semaphore and a software semaphore before accessing a shared data structure in the shared bank. It is crucial to minimize such synchronization overhead to get the maximum performance benefit of the DPSDRAM architecture.

Figure 3 shows the pseudo code for the shared memory access APIs. As shown in the code, several memory accesses for mutual exclusion are needed. Even if a processor can acquire all locks immediately, 9 additional memory accesses are required: 4 for software semaphore lock, 3 for software semaphore unlock, 1 for hardware semaphore lock, and 1 for hardware semaphore unlock as annotated in the pseudo code. The procedure to lock/unlock software semaphore consists of locking hardware semaphore, accessing software semaphore in the shared memory, and unlocking hardware semaphore. Only single access is required to lock/unlock hardware semaphore as it guarantees the atomicity. *Software_semaphore_write(lock, ACQUIRED)* needs to access shared memory twice for reading and marking software semaphore as in use while s*oftware_semaphore_write(lock, RELEASED)* needs just one to make it available.

Note that the overhead of managing the software semaphore can be amortized if a large chuck of data is accessed. Since both the writer and the reader processors access the shared bank, the overhead becomes doubled for each communication. Consequently, at least 9 memory accesses are added for the shared memory access.

```
software_semaphore_lock(lock){
   hardware_semaphore_lock(lock_for_sw_semaphore_area);  // 1 access
   software_semaphore_write(lock, ACQUIRED);             // 2 accesses
   hardware_semaphore_unlock(lock_for_sw_semaphore_area); // 1 access
}                                                        // total 4 additional accesses
                                                         // for SW semaphore lock
software_semaphore_unlock(lock){
   hardware_semaphore_lock(lock_for_sw_semaphore_area);  // 1 access
   software_semaphore_write(lock, RELEASED);             // 1 access
   hardware_semaphore_unlock(lock_for_sw_semaphore_area); // 1 access
}                                                        // total 3 additional accesses
                                                         // for SW semaphore unlock

access_shared_data(address_size){
   software_semaphore_lock(lock_for_data);               // 4 accesses for SW semaphore lock
   for each shared_block for (address, address+size-1) do {
      hardware_semaphore_lock(lock_for_shared_block);    // 1 access for HW semaphore lock
      access_shared_block(shared_block);
      hardware_semaphore_unlock(lock_for_shared_block);  // 1 access for HW semaphore unlock
   }
   software_semaphore_unlock(lock_for_data);             // 3 accesses for SW semaphore unlock
}                                                        // total 9 additional accesses needed
                                                         // for a shared memory access
```

Figure 3. Pseudo code for shared memory access. Annotated access counts assume that all locks can be obtained with no delay.

3. ARCHITECTURE OPTIMIZATION TECHNIQUES

In this section, we describe three techniques to reduce synchronization overhead by exploiting the communication patterns in a typical mobile embedded system.

3.1 Lock-Priority

As explained earlier, in mobile embedded systems, the baseband processor and the application processor are not symmetric in their communication requirements. The baseband processor triggers communication infrequently but with relatively big data once it happens, while the application processor deals with relatively small data but frequently. For the baseband processor, it is important to reduce the waiting time for obtaining the hardware lock because there are real-time constraints to handle communication with the air. On the other hand, for the application processor, the access overhead for the hardware lock is the most important factor to the communication performance. Since the application processor accesses small portions of data frequently, the synchronization overhead for hardware lock is accumulated.

Based on this observation, we propose an optimization technique called the lock-priority technique, which gives the right of access to the application processor by default. If the baseband processor requests the access, the system switches the access privilege to the baseband processor immediately. As soon as the baseband processor releases the hardware lock, the access privilege is bestowed to the application processor automatically. Therefore, the application processor needs not send a request for the hardware semaphore when it accesses the shared memory bank. It removes the synchronization overhead of getting the hardware lock which requires 6 memory accesses as illustrated in Figure 3. On the other hand, the baseband processor experiences no latency of accessing the shared memory. As a result, the proposed lock-priority technique satisfies both processors with different wish lists.

Figure 4 shows the modified architecture with a lock manager in the application processor side, which implements the lock-priority technique explained above. The lock manager has two control input ports from both application and baseband processors, which indicate

the access request for the shard memory bank. The operation of the lock manager is represented in the form of a finite state machine.

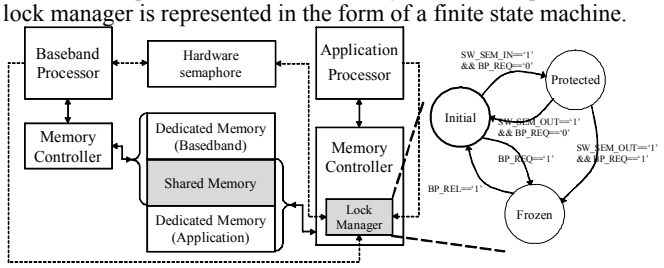

Figure 4. Architecture Optimization I: lock-priority.

The 'Initial' state indicates that the hardware lock is given to the application processor by default. The application processor is allowed to access the shared memory as soon as it requires. If the baseband processor requests the hardware lock, the lock manager should release the hardware lock of the application processor immediately, changing its state to the 'Frozen' state in which the grant of the shared memory access to the application processor is postponed until the baseband processor finishes its transfer and releases the lock.

If the application processor accesses a software semaphore in the 'Initial' state, the lock manager should hold the hardware lock until the end of the transaction, moving to the 'Protected' state. After the baseband processor releases the hardware lock, the lock manager gets the lock back automatically to the application processor and returns to the 'Initial' state again.

3.2 Static-Copy

Our second optimization is based on the observation that the application processor tends to access the same data several times. A good example is a 3D rendering application. After a 3D data set is written into the shared memory bank by the baseband processor, the application processor accesses the data set multiple times to perform 3D rendering.

Then, it is beneficial to copy the data set into the local memory of the application processor in order not to pay the synchronization overhead of shared memory access. Since the local memory is cacheable while the shared memory is not, performance gain becomes larger as the access frequency to the same data set increases. Thus we propose another technique, called the static-copy technique, where the baseband processor transfers the data directly to the local memory of the application processor using a customized DMA controller that shares data path with the application processor as shown in Figure 5(a).

In the static-copy technique, the shared memory is only used for transient buffer and the system behaves as a distributed memory system where inter-processor communication is achieved by the "put-get" mechanism [7]. The application processor notifies the receiving location to the baseband processor before communication begins. Then, the baseband processor puts the data to the specified location without intervention of the application processor. This mechanism is particularly useful when the application processor includes a hardware logic that needs to access the shared data. Since it is not trivial for the hardware logic to manage the hardware semaphores and the software semaphores, the hardware logic can access the data via the local memory with the proposed static-copy technique.

The pseudo code of the static-copy is shown in Figure 6. The baseband processor acquires the hardware semaphore of the shared bank, copies data into the shared bank, and release the hardware semaphore. Then, it lets the customized DMA copy the data to local memory of the application processor.

Figure 5(a) displays the static-copy enabled architecture with a customized DMA controller at the application processor side. Once the baseband processor finishes its transfer to the shared memory, it triggers the DMA controller to move the data from the shared bank to the local memory of the application processor.

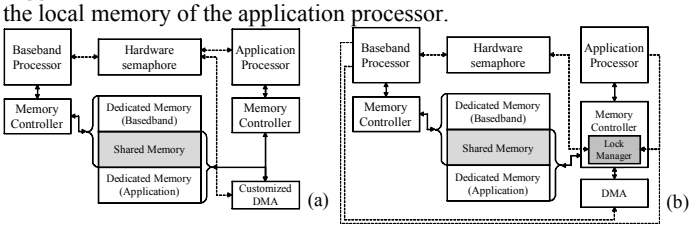

Figure 5. Architecture Optimization II: static-copy (a) without the lock manager and (b) with the lock manager.

```
// API static_copy (src→ via→ dst)
//   dest: destination address
//   src: source address
//   via: shared memory address
//   size: data size
static_copy(dest, src, via, size){
   hardware_semaphore_lock(shared_block_via);
   memory_copy(via, src, size)
   hardware_semaphore_unlock(shared_block_via);
   trigger_DMA(dest, via, size);
}
```

Figure 6. Pseudo code of the static-copy technique.

The customized DMA controller should be able to manage the hardware semaphore differently from a normal DMA controller. Before the baseband processor triggers the DMA, it releases the hardware lock. Once the DMA is triggered, it acquires the hardware lock and holds the lock until a transfer is finished. During the DMA transfer, the baseband processor may not access the associated shared memory block. Figure 5(b) shows the improved architecture integrated with the lock manager for the lock-priority technique. The lock manager takes the role of getting the hardware lock instead of the DMA. Then a simpler DMA controller can be used in the architecture.

Since the static-copy technique incurs redundant copy overhead (from shared memory to local memory), data transfer time becomes longer than before. However, such overhead of data transfer can be hidden by overlapping computation time of the application processor with DMA operation, which will be verified by experiments.

3.3 Multiple Blocks in the Shared Bank

To maximize the parallelism between consecutive data transfers, we can divide the shared bank into multiple blocks as shown in Figure 7. Permitting simultaneous accesses to multiple blocks in the shared bank, each block needs its own hardware semaphore logic separately.

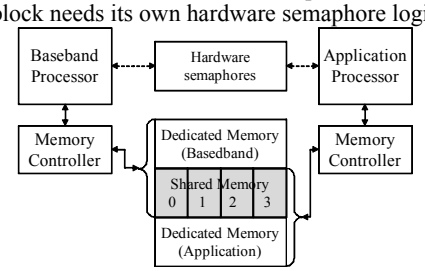

Figure 7. Architecture Optimization III: dividing the shared bank into multiple blocks.

Simultaneous accesses make it possible to overlap consecutive transfers with each other to boost up the data transfer performance. This is the basic principle of the well-known *double-buffering* technique in producer-consumer data transfer: Multiple blocks of the shared bank behave as simultaneously accessible buffers in the double-buffering technique. As a result, read operations of the application processor are overlapped with write operations of the baseband processor between the successive transfers. In the static-copy scheme, using multiple blocks by turns as transient buffer, a series of static-copy transfers can be overlapped as well.

4. EXPERIMENTS

Since there is no DPSDRAM architecture system available commercially yet, we built a virtual prototyping system that is composed of two ARMv5 ISA compatible instruction set simulators (ISS), a DPSDRAM model, and other peripheral device models. Our processor simulators are based on Sim-Panalyzer [8], which is a derivation of SimpleScalar [9]. Each ISS is implemented using UNIX pthread and the local memory is modeled as a thread variable. The shared objects such as hardware semaphore and shared memory are modeled with shared variables that are protected by mutex. To synchronize the accesses to shared objects the simulator always suspends the thread whenever it tries to access the shared objects.

The lock-priority scheme is modeled in the central synchronization controller. The customized DMA is also modeled by calculating the delay of DMA operation and by annotating the completion of DMA transfer to the ISS threads.

For DPSDRAM architectures, we assume that each processor has 16 kilobytes direct-mapped instruction cache and 16 kilobytes 4-way set-associative data cache. The clock frequencies of the baseband processor and the application processor were set to 250MHz and 500MHz respectively to model a realistic architecture where the application processor is faster than the baseband processor. All memory interfaces of two processors and the DPSDRAM run at 100MHz. We also consider the traditional peripheral bus architecture in Figure 1(a) for performance comparison. To simulate the traditional peripheral bus architecture, two off-chip single-port SDRAMs were modeled additionally and the clock frequency of the peripheral bus (LCD bus) was set to 20MHz.

4.1 Evaluation of DPSDRAM Architectures with Various Optimization Techniques

The first experiment is to verify the effectiveness of the base DPSDRAM architecture and the optimized architectures proposed in Section 3 over the LCD bus architecture of Figure 1(a). Figure 8 shows the performance comparison of various DPSDRAM architectures. 'A' denotes the base DPSDRAM architecture, while 'B', 'C', and 'D' stand for the extended architecture applying static-copy, lock-priority, and both to the base DPSDRAM architecture, respectively. The number following the alphabet means the number of accesses by the application processor to the same data set.

For the experiment, we extract the actual communication patterns from the trace data with an existent LCD-bus based architecture for a 3G mobile phone application: the baseband processor typically writes about 1-10 kilobytes data into the shared memory at once and the application processor takes the data by 32 bytes per read respectively. We synthesized a synthetic but practical workload based on this typical data transfer scenario, and set the total amount of transferred data to 256 kilobytes.

Since the performance of the LCD bus architecture is inferior to the base DPSDRAM architecture by about 4 times, we omit it for brevity. Regarding non-repetitive accesses, compared with the base architecture, both the static-copy and lock-priority scheme show the similar performance gain of about 20%. Such performance gains owe to reduction of read operations' overhead at the application processor. In case of the static-copy scheme, the sending time increases due to DMA operations from the shared bank to the local bank of the application processor. In the lock-priority scheme 56% of the reading time of the application processor is consumed by the holding time of the application processor waiting for the hardware lock available. The integration of the static-copy and the lock-priority techniques shows the best performance with 33% reduction compared to the base DPSDRAM architecture, which is mainly due to the fact that the delay by 'application processor holding' does not appear any longer, i.e., the application processor need not access the shared bank because of the static-copy technique.

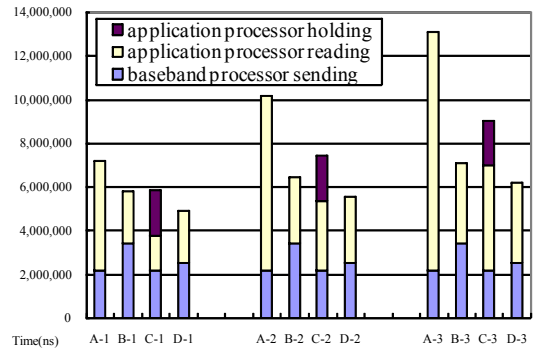

Figure 8. Performance comparison of DPSDRAM architectures varying the number of accesses.

Repetitive accesses to the same data set by the application processor make the static-copy technique more attractive. To verify the effectiveness of the static-copy scheme, we vary the access counts of the application processor to the same data set. The base DPSDRAM and the lock-priority technique show poor scalability as shown in Figure 8. The base DPSDRAM suffers from significant synchronization overheads as the number of accesses increases. The performance gap between the static-copy scheme and the lock-priority scheme gets wider as the access count increases. In case of the static-copy, although DMA overhead is added for copying data from shared bank to local bank, we can utilize the caching effect that outweighs this copying overhead more as the number of data accesses increases. It makes the static-copy scheme better scalable than the lock-priority scheme. Moreover, the synchronization overhead in the static-copy can be removed by combining with the lock-priority technique, which leads to further performance gain up to 53% over the base DPSDRAM architecture.

4.2 Evaluation of Using Multiple Blocks in the Shared Bank

The second experiment validates the proposed optimization that divides the shared bank into multiple blocks. Additional performance gain can be achieved by allowing simultaneous accesses between difference blocks in the shared bank by using double-buffering. While the baseband processor puts a data set to one shared block, the application processor may fetch another data set from the other block in parallel.

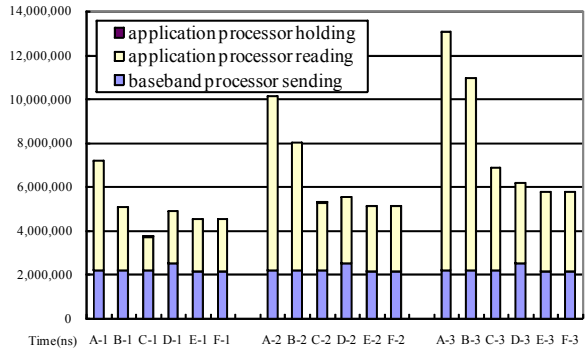

Figure 9. Performance comparison of the DPSDRAM architectures exploiting multiple blocks in the shared bank.

In this sub-section, we refer the base DPSDRAM architecture to the architecture with four separate blocks in the shared bank without optimization technique (lock-priority or static-copy), which is labeled 'A'. We denote the base DPSDRAM with a double-buffering technique by 'B'. The architecture 'C' and 'D' are the optimized

architectures by the lock-priority scheme and the static-copy scheme respectively. The architecture 'E' combines the double-buffering with the static-copy technique while all the techniques (the lock-priority, static copy, and double-buffering) are incorporated in the architecture 'F'. The performance comparison of those architectures is presented in Figure 9.

The advantage of having multiple blocks in the shared bank appears clear in the architectures with double-buffering. Just applying double-buffering technique alone to the base DPSDRAM architecture, 'B', shows 17-30% performance gain against the base DPSDRAM architecture. Although some access conflicts between processors are eliminated by permitting simultaneous accesses to different blocks, the performance gain becomes smaller as the number of accesses increases. By combining the lock-priority scheme and double-buffering, 'C', 25-40% of performance gain is obtained when compared to the architecture with the double-buffering only and about 30% performance gain compared to the lock-priority scheme only. It is noteworthy that despite the poor scalability of the lock-priority scheme, the architecture with the lock-priority and double-buffering techniques shows the improved performance close to that of the architecture with both static-copy and lock-priority without double-buffering.

It is also advantageous for the static-copy technique to allow double-buffing by dividing the shared bank. If we use multiple blocks as transient buffers in the static-copy scheme, we can get similar performance gain by overlapping the consecutive data transfers. In the architecture 'E', the double-buffering reduces the baseband processor sending time by about 15% and the overall performance gain over the architecture with the static-copy only, 'D', is about 8%. The architecture 'F' that has the double-buffered static-copy and lock-priority schemes, however, shows no further performance improvement since the DMA transfer to the local memory of the application processor and the data transfer to the shared bank are overlapped, hiding the gain of the additional lock-priority scheme.

4.3 Evaluation with 3D Rendering Application

In order to assess the proposed optimization techniques by an industrial strength application, we consider a 3D rendering application, which is a part of in-house mobile phone software development suit. In the example, the baseband processor provides the application processor with tens or hundreds kilobytes of data that are to be rendered and texture-mapped at once. On the contrary, the application processor performs 3D rendering task that requires frequent reading of tens of bytes. We observed that most of data transfers except for a few control data are unidirectional and the application processor accesses the same data repetitively, which makes the static-copy scheme efficient. Furthermore, there are some time constraints, for example the number of frames to be rendered in one second, which stresses the importance of improving the speed of data transfer. We extracted memory access traces of which total amount of transferred data is about 2 megabytes from both processors.

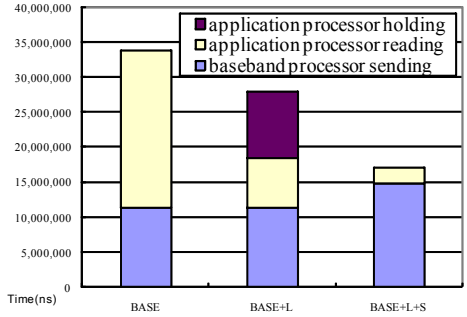

Figure 10. Performance comparison of the DPSDRAM architectures in 3D rendering examples.

Figure 10 shows the data transfer time according to three communication architectures. 'BASE' denotes the base DPSDRAM architecture while '+S' and '+L' mean the static-copy and lock-priority techniques respectively. The LCD bus architecture works about 6 times slower than the base DPSDRAM architecture even though it does not appear in the graph. Further performance improvements can be obtained with the lock-priority and static-copy. In case of the architecture with the lock-priority only, there is about 20% performance improvement compared with the base DPSDRAM architecture. Considering both the lock-priority and the static-copy achieves additional improvement to about 50%.

5. CONCLUSION

In this paper, we have evaluated a dual-port SDRAM architecture for mobile embedded systems. To minimize the non-negligible synchronization overhead, we proposed three optimization techniques: static-copy, lock-priority, and multiple shared blocks. Both lock-priority and static-copy techniques reduce the synchronization overhead by exploiting typical communication patterns of mobile embedded systems.

The effectiveness of the DPSDRAM architecture has been evaluated by extensive experiments. The base DPSDRAM architecture was about 4 times faster than the traditional peripheral bus (LCD bus) architecture. Furthermore, by combining the three optimization techniques, the performance was improved by 20-50% over the base DPSDRAM architecture.

6. ACKNOWLEDGMENTS

This work was supported by BK21 project, SystemIC 2010 project funded by Korean MOCIE, and Creative Research Initiative sponsored by KOSEF research program(R17-2007-086-01001-0). This work was also partly sponsored by ETRI SoC Industry Promotion Center, Human Resource Development Project for IT-SoC Architect. The ICT and ISRC at Seoul National University and IDEC provide research facilities for this study.

7. REFERENCES

[1] MXC300-30: 3G Single Core Modem Platform, Freescale Semiconductor [Online]. Available: http://www.freescale.com.

[2] T. V. Meeuwen, A. Vandecappelle, A. V. Zelst, F. Catthoor, and D. Verkest, "System-level interconnect architecture exploration for custom memory organizations," in Proceedings of International Symposium on System Synthesis, pp. 13-18, Sep. 2001.

[3] F. Gharsalli, D Lyonnard, S Meftali, F Rousseau, A. A. Jerraya, "Unifying memory and processor wrapper architecture in multiprocessor SoC design," in Proceedings of International Symposium on System Synthesis, pp. 26-31, Oct. 2002.

[4] K. Patel, E. Macii, and M. Poncino, "Synthesis of partitioned shared memory architectures for energy-efficient multi-processor SoC," in Proceedings of the conference on Design Automation and Test in Europe, pp. 10700-10701, Feb. 2004.

[5] Integrated Device Technology, "Dual port memory simplifies wireless base station design," Application Note, AN-409, Jan. 2004.

[6] Samsung Electronics Inc, "Fusion Memory Solution OneDRAM™," [Online]. Available: http://www.samsung.com/PressCenter/PressRelease/PressRelease.asp?seq=20061213_0000306480

[7] K. Hayashi, T. Doi, T. Horie, Y. Koyanagi, O. Shiraki, N. Imamura, T. Shimizu, H. Ishihata, and T. Shindo, "AP1000+: Architectural support of PUT/GET interface for parallelizing compiler," in Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 196-207, Oct. 1994.

[8] The SimpleScalar-Arm Power Modeling Project [Online], Available: http://www.eecs.umich.edu/~panalyzer.

[9] SimpleScalar LLC [Online], Available: http://www.simplescalar.com