On-Chip COMA Cache-Coherence Protocol for Microgrids of Microthreaded Cores

Li Zhang[1] and Chris Jesshope[2]

[1] Informatics Institute, University of Amsterdam
[2] Kruislaan 403, Amsterdam 1098SJ, the Netherlands
{zhangli,jesshope}@science.uva.nl

Abstract. This paper describes an on-chip COMA cache coherency protocol to support the microthread model of concurrent program composition. The model gives a sound basis for building multi-core computers as it captures concurrency, abstracts communication and identifies resources, such as processor groups explicitly and where mapping and scheduling is performed dynamically. The result is a model where binary compatibility is guaranteed over arbitrary numbers of cores and where backward binary compatibility is also assured. We present the design of a memory system with relaxed synchronisation and consistency constraints that matches the characteristics of this model. We exploit an on-chip COMA organisation, which provides a flexible and transparent partitioning between processors and memory. This paper describes the coherency protocol and consistency model and describes work undertaken on the validation of the model and the development of a co-simulator to the Microgrid CMP emulator.

1 Introduction

It is now widely accepted that future computer systems must manage massive concurrency. Even on chip, that concurrency will be significant and asynchronous, having many of the same characteristics as grid systems. The constraints driving this are based on exponential functions meeting hard limits. For example, the hard limit on power dissipation will limit clock frequency and the limit of chip area is already a problem with respect to the area reachable in a single clock cycle. These constraints are the final nail in the coffin of the sequential model of computation. In the past, superscalar processors were able to exploit the implicit concurrency in sequential programs but they have very poor scaling as a consequence of their centralised model of synchronisation and scheduling. There is therefore a dire need for the development of more distributed models that still retain the advantages of the sequential model, namely composability and determinism.

Basic research into scalable, on-chip instruction execution has led in two distinct directions. The first has been the resurrection of dataflow instruction execution. In this model, instructions are mapped to ALUs and data is moved between instructions by using other instructions rather than named memory locations as

L. Bougé et al. (Eds.): Euro-Par 2007 Workshops, LNCS 4854, pp. 38–48, 2008.

targets of an operation [1,2]. The problem with this approach is that it does not support a memory model that can be used with conventional languages. Both approaches referenced have addressed this limitation but at some expense to the concurrency they are able to expose.

An alternative approach is to embrace explicit concurrency in the execution model, while at the same time maintaining the properties defined above that have made the sequential model so ubiquitous. The microthread model achieves this by capturing abstract concurrency in a conventional RISC-like ISA [3]. A *create* instruction dynamically defines concurrent execution as a *family* of threads based on a single thread definition. Because each member of the family has a unique index assigned to it, heterogeneous as well as homogeneous concurrency is supported. Families are parameterised and can be infinite in range. Concurrency can be created hierarchically as the thread definition for one family may contain creates for subordinate families. Create is therefore used to replace the sequential constructs of looping and function calls with concurrent equivalents. Moreover, the create instruction binds a *unit of work*, which is a family and its subordinate families to a *place*, which is a collection of processors.

Registers in this model implement a blocking read and provide fine-grain synchronisation between threads. This is similar to dataflow and in distinct contrast to other thread models such as Simultaneous Multi-Threading (SMT) [4] that synchronise on memory. The distributed register file also provides the mechanism for scheduling instructions from threads as continuations are stored in registers waiting for data, which are rescheduled on a write. The distribution of register files between multiple cores enables scalable data-driven execution of microthreaded code across many processors. It also provides significant tolerance to latency, as memory operations are decoupled by this register-based synchronisation. Current processor designs allow hundreds of threads to execute locally, typically allowing tolerance of up to a thousand cycles in memory accesses.

Threads in a family are mapped dynamically but deterministically to a set of cores and this provides the binary code compatibility. Moreover, legacy binary programs that use function calls and loops can be executed as singleton families in this model, providing backward compatibility.

While registers are used for fine-grain synchronisation between threads (and a thread and the memory system). A bulk synchronous model is provided on memory. Typically families of threads will update indexed data-structures in memory and a *sync* action (using a return code to a synchronising register) will provide the synchronisation required in order to create another family to consume the data structure. The memory model has a relaxed memory consistency compared to other thread-based approaches and requires the design of a new memory architecture and coherency protocol to fully exploit it. Memory written by a family is only defined following termination of all threads in that family. Race conditions are not excluded but the only reason they would exist (apart from bugs) is to allow explicit non-determinism, as may be found in some chaotic algorithms. We adopt location consistency on these races [5].

2 Background

With multi-core chips, it is even more difficult to break the 'memory wall' [6]. According to recent analysis (for instance Intel Pentium M platrom [7]), typical access to level 1 cache takes 1 to 3 cycles, access to level 2 cache takes around 10 to 20 cycles, while the RAM access may take more than a hundred cycles. Current trends in solving this problem are to increase the cache size and, as a result, the chip area of modern microprocessors is already dominated by cache. However a multi-core design must also consider scalable throughput from memory. An example is the IBM Power 4/5 architectures, which use three identical cache controllers for L2 cache, where cachelines are hashed across the controllers. Distribution brings further problems to the memory design and a choice must be made on how to implement sharing and coherence. In the Power 4/5 IBM provides coherence with four snoop processors implemented on the L2 controllers, whereas in their Cell processor they partition the memory locally and force the user to explicitly code the mapping of data to maintain coherence. Bus-based snooping on the other hand is not scalable.

Trends in multi-cores designs can be seen in the Intel's 80-core Tera-Scale Research Chips [8], where each core has a local 256 KB memory associated via Through Silicon Vias (TSV) and where all processing cores are connected to the network on-chip. Such distributed structures provide scalability, but the local memory implementation lacks flexibility and would destroy the abstraction over mapping that gives binary code compatibility in the microthread model, where families of microthreads can be assigned freely to different processing cores on chip.

2.1 Requirements

The requirement for a memory system in a Microgrid of microthreaded processors must provide the abstraction of a shared memory but achieve this across potentially thousands of processing cores, while providing scalable throughput both on and off chip. The ameliorating factor in this difficult design is that the processors tolerate a large amount of latency, which has led us to resurrect and specialise a paradigm used in earlier parallel computers, such as the Kendal Square KSR1 [9]. We introduce a Cache Only Memory Architecture (COMA) [10] for the on-chip cache system. In COMA, all the memory modules can be considered as large caches, called Attraction Memory (AM). Data is stored by cacheline but the line has no home location as a CC-NUMA [11], where the physical location of a memory address is always known. COMA adds complexity to locating a data in the memory but at the same time, increases the chances of data being in the local cache. In a Microgrid of microthreaded processors we propose a cache memory based on the COMA approach and allow data to migrate dynamically within the on-chip memory. A significant difference between the on-chip COMA and traditional COMA system is that the traditional COMA system will hold all data in the system without a backing store. However, on-chip COMA is unlikely to provide enough space to store so much data. The on-chip COMA therefore has a backing store for data off the chip, where one or more DRAM interfaces or links or some other Microgrid chips will provide an interface for storing incoming data. The main contribution of this paper is the protocol required to implement the memory consistency model in such an on-chip COMA memory system, its verification and subsequent use in a memory co-simulator.

2.2 The Microgrid Multi-core Chip

A Microgrid is a tiled multi-core chip designed using microthreaded multiprocessors. Because the microthread model uses a unique method of program composition using the create instruction, all structures in the microgrid must support this model. This includes on-chip networks and the memory organisation. At different levels in a program's concurrency hierarchy, families of threads are distributed to configured rings of processors, allocated from a pool of processors. This requires dynamic processor allocation similar to the way in which malloc is used for memory management. This is illustrated in Figure 1.a which shows a dynamic snapshot of an executing microthread program. The node marked SEP maintains a map of resources used and allocates and configures rings of processors over the low bandwidth grid network. The circuit-switched, ring networks that form clusters from the microthreaded processors provide protocols for family creation and termination and also allow adjacent processors in a ring to share data between their local register files to provide the distributed shared register file in the cluster. The ring network is shown in more detail in Figure 1.b.

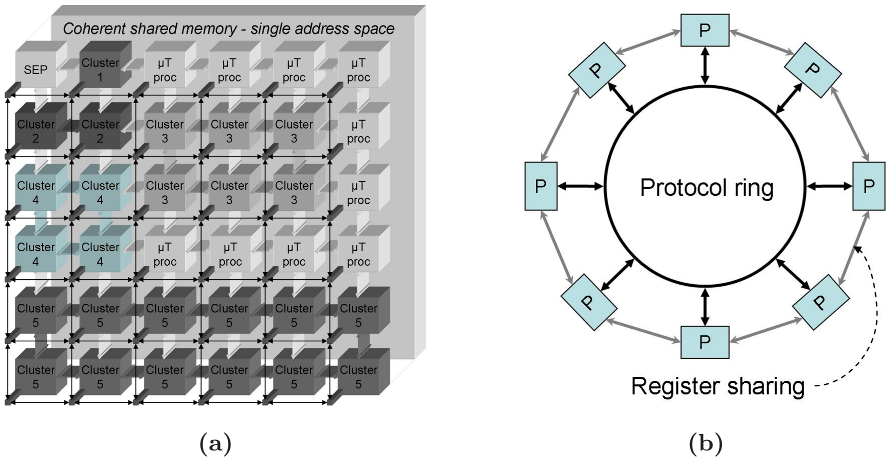

Fig. 1. (a) Microgrid of microthreaded processors (μT proc) configured into dynamic clusters with ring interconnections. A request for a configured cluster is performed by the SEP over the resource management and delegation network, which is the low-bandwidth, packet-routed grid linking all processors. (b) Details of the ring network configured to support the execution of families of threads.

Each microthreaded processor in a cluster has a small (1KByte) level 1 D-cache. Missing in this cache does not stall the processor as memory operations are decoupled by suspending the thread on the target register location. Memory requests asynchronously update the register file when the memory access completes. In order to manage this, all memory requests must be tagged by target location in the register file as well as family identifier, for memory synchronisation.

The single address space defined in Figure 1.a is distributed over the chip in order to provide scalable performance. It is partitioned into Level 2 cache blocks, where a small number of processors will share requests on their L1 cache misses. In a traditional NUMA organisation, data has a home location, which means when the processor suffers a miss on the L2 cache, the processor will try to access the home location of the address. The access to a remote memory might take more than a thousand cycles. COMA, on the contrary, does not have a home location for a given piece of data. In COMA, the dynamic migration of data increases the possibility of finding the data in the local L2 cache or possibly another L2 cache module on chip. The downside of this is that to locate data is relatively expensive and to do this, directories are utilised. COMAs are also developed in different structures. Data Diffusion Memory (DDM) [12] uses a tree structure, where each level of the tree is associated with a Directory that holds information about the data availability below this level. (N.b. a directory does not store any data, just state information). The Kendall Square multiprocessor [9] on the other hand used a hierarchical snooping ring network. Generally, the protocols used are very similar to snooping protocols.

Details of the structure and protocol of the Microgrid on-chip COMA are given in the next section.

3 Memory Hierarchy Design

The memory system includes both the on-chip cache system and an off-chip communication interface or interfaces. Presently it is assumed that the off-chip interface is connected to a multi-bank memory storage, which is able to provide a high bandwidth for feeding multiple processors on a chip. The address space is physically interleaved across different memory banks.

The on-chip cache system is designed to have 2 levels of caches. Each processor is closely associated with its own small and fast L1 cache. L2 caches are relatively big, and each could supply data for multiple processor-L1 cache pairs. The L2 caches and their associated L1 caches are connected with snooping buses. Since the local bus configuration is not scalable, the number of L1 caches connected on a bus is restricted. Assuming each L2 cache can support between 4 to 8 processor-L1 cache pairs, then the number of L2 caches for current technology (say 128 processors) is 16 to 32. In future, perhaps thousands or even tens of thousands of processors may be integrated onto a single chip. Thus, the network utilised to connect L2 caches will be a hierarchy of ring networks, which map onto the hierarchical nature of the concurrency trees generated in the microthreaded model. We believe this will provide scalability in both bandwidth and cost as well as locality in communication.

To reduce the coherence pressure on a ring network, we group adjacent L2 caches together, and the caches in the same group are connected with a uni-directional ring network. Furthermore, all groups are connected by a higher level ring to allow for memory requests between groups. The lower and higher level ring networks are called level-1 (L1) and level-2 (L2) ring networks respectively. The joints between L1 and L2 rings are Directories, the structures designed to direct the flow of certain requests and reduce the network traffic. Each directory holds the information about all the data available in the group it is associated with. Like caches, directories hold information in a set-associative manner. Furthermore, the items in the directory are tailored to the cacheline size. Each item in the directory holds information about the cacheline tag and some state information without any data values. The state information can tell the availability and exclusiveness of a certain cacheline in the group it associates with. For instance, when a directory indicates a certain data is exclusive, it means that the certain cacheline can only be found valid in the current group, although the cacheline inside the group can be shared across different caches. On the L2 ring a Root Directory (RD) holds all the state information about the available data on-chip. It helps decide whether to send out a request off the chip. The memory controller connected with RD will carry out the off-chip communication.

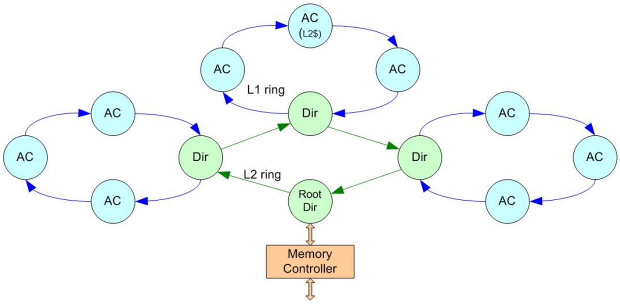

Fig. 2. Attraction Caches are organized around a 2-level hierarchical ring network as on-chip COMA

A normal snooping L2 cache only serves requests from processors and passively changes its state information. However, in this on-chip cache, each L2 cache not only serves the request received locally, but also serves requests received from the network. The property is very similar to COMA, which allows different Attraction Memories to serve data automatically so that data can flow to the place where it was mostly or recently used. Thus we call our on-chip cache system an on-chip COMA cache hierarchy. Furthermore, since the L2 cache behaves similarly to an attraction memory, to differentiate the cache from a normal cache we also call these L2 caches Attraction Caches (AC). The on-chip

COMA structure is depicted in Figure 2. In the figure, there are only 3 L1 rings and 3 ACs are shown, but the number of rings and ACs can be set arbitrarily. Since the memory access patterns in our model can be very different from normal processors due to its distributed large register file design and swift context swtich capability, the detailed design parameters have to be decided by co-simulation with our existing multi-processor simulator.

4 On-Chip COMA Cache-Coherence Protocol

A cache-coherence protocol maintains the consistency in a cache system. The design of the protocol for our model also has to address the issues such as minimising off-chip communication and providing a solution to the high bandwidth requirement. Our L1 cache is very small, simply providing a buffering and prefetching functionality. It uses a simple protocol with only two cacheline states, Valid and Invalid. Here we focus on the design of Attraction Cache protocol.

In a distributed shared memory architecture, almost all the cache coherence protocols are based on MOESI variations. MOESI represents five cacheline states, Modified (M), Owned (O), Exclusive (E), Shared (S), and Invalid (I) states. The cachelines in M and O states have the ownership of the data; they are also called dirty. The line at M/E states holds data exclusively. A shared cacheline only has the validness of data. Invalid data means the line is not present in the current cache. S and I states are the basic states that represent the validness of a cacheline. M and O states tend to keep the dirty values on the chip, which helps reduce off-chip communication. The Exclusive state is useful when repeated writes and reads to the same location happens in the cache. This situation is unlikely to happen because in our architecture data dependencies are generally captured at the register level rather than the memory level. Thus without Exclusive state a MOSI protocol is chosen. Since the memory requests are served asynchronously, the outstanding requests have to be remembered in the current cache. Consequently, two basic states are provided, ReadPending (RP) and WritePending (WP) states. The RP line waits for the valid data to be loaded and the WP line waits for the exclusiveness of the line to be acquired before performing a Write operation locally.

The AC is able to handle 10 different requests, which are listed on Figure 3. In the following text only the acronyms are mentioned. Requests LR and LW can only be issued by the processor. Both RS and RE try to load the data remotely, while RE will acquire exclusiveness at the same time. SR and ER are the corresponding replies for RS and RE. The request BR represents the eviction of a cacheline. The BR can invalidate a shared line directly; it also has to preserve the dirty data by writing them back to the main memory with WB request. The IV request is normally generated by an LW request, which needs to acquire the exclusiveness of a data copy. When the IV returns to the initiator, it is regarded as a DE which represents the acquisition of the data exclusiveness.

As described above, the directories hold information about the current state of the data in the group. Three different states, Invalid (IN), Shared (SH), and

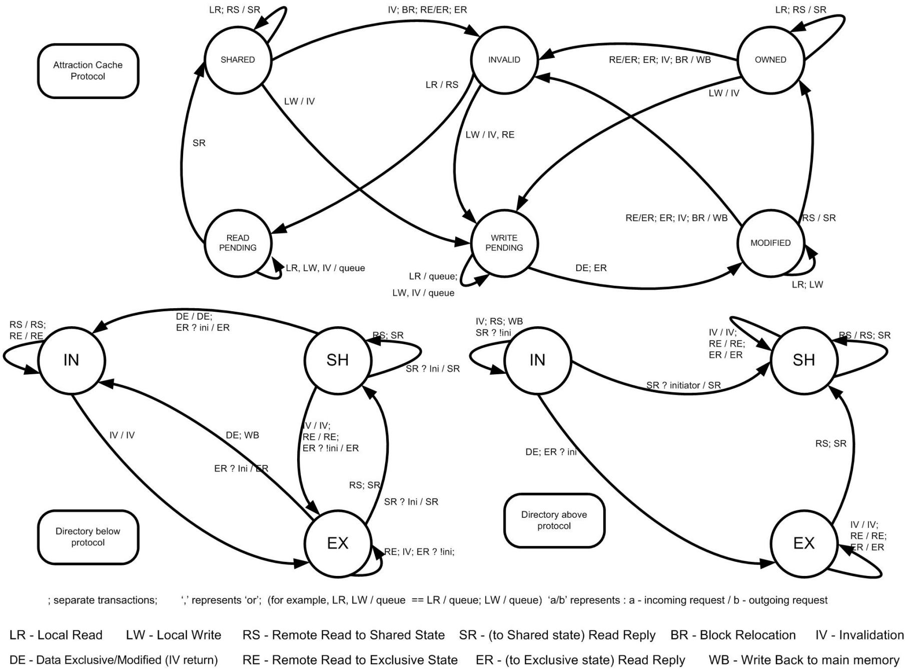

Fig. 3. MOSI protocol State Transition Diagram of Attraction Cache and Directory

Exclusive (EX), can be assigned to each item in a directory. As a joint of L1 and L2 rings, a directory also determines whether a certain request should be passed to the next node in the same or a different level. For instance, an RE request is received from L1 ring by a directory which has the corresponding item at EX state. Being aware the sub-system holds the exclusiveness, the directory will propagate the RE in the L1 level without incuring any traffic on L2 ring. The detailed state transitions of AC and directory are depicted in Figure 3.

5 Consistency Model

In a multi-processor system, a consistency model places specific requirement on the sequence that shared memory accesses from one process may be observed by other processes. A number of consistency models have been proposed, including Sequential Consistency (SC) [13], Release Consistency (RC) [14], and Location Consistency (LC) [5]. Different consistency models balance the programming complexity and system performance. LC is claimed to be the weakest consistency model. Unlike SC, LC does not make the assumption that all writes to the same memory location are serialized in some order observable to all processors. The program will behave the same as on other systems as deterministic code is being executed. Furthermore, unlike RC the synchronization is not happening for different blocks of code, but in terms of each individual memory location. This partial order is only maintained for each individual memory locations. The issuing order between memory accesses on different memory locations can be adjusted by the compiler to achieve better performance.

In our on-chip COMA architecture, the maintenance of sequential consistency is very expensive. For instance, if two processors are writing to the same memory location concurrently on a ring. The processors separate the ring into two arcs. The ACs on different arcs will observe the two requests in different orders, which is forbidden in the sequential consistency model. Fortunately the microthread memory model allows us to adopt the more relaxed LC as the consistency model, which does not require the strict order under non-deterministic situations.

To exploit the potential of Location Consistency, a suspended request queue structure is proposed for each cache and directory. In the Attraction Cache, a cacheline will be locked in a temporary state when a new request to the same location cannot be served directly. Thus, the incoming request has to be saved temporarily in a buffer. To avoid blocking the request to other memory locations which can be served directly, a queue structure is proposed to be associated with each suspended cacheline. As the reply to the locked line returns, the associated suspended line can be reactivated and served directly. Since we are using the LC consistency model, the order of serving requests from different locations does not need to be handled by the memory system. As a result, LC is actually extended to the cacheline level. The temporary ReadPending and WritePending states are actually combinations of the states in the data table and Tstates in queue table depicted in Figure 4. Each queue head in the queue table is associated with a linked list in the request buffer. An Empty Queue Head (EQH) maintains the list of empty slots in the request buffer. Similar queue structure is also implemented in the directories.

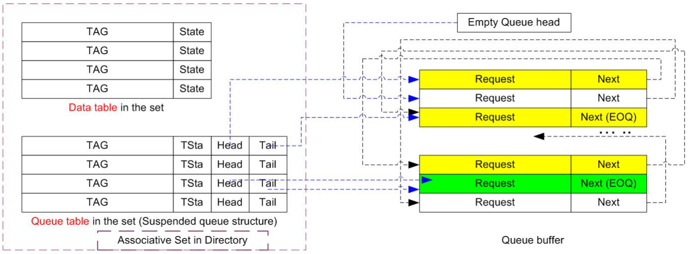

Fig. 4. Suspended Queue Structure in Attraction Cache

6 Conclusion and Future Work

The paper gives an overview of the microthreaded CMP architecture. With its capability for explicit context switching and scheduling based on thread continuations held in a large distributed register files, the model can tolerate long memory access latency and give high throughput. Targeting VLSI technology in the 10 to 15 year timeframe, we have introduced an on-chip distributed shared memory architecture and defined its operation. The choice of an on-chip COMA system is to more efficiently utilise the overall memory bandwidth. Two levels of ring networks are utilised to distribute memory storage across the network for a large number of processors on chip and directories are used to direct the memory transaction flow within the on-chip cache hierarchy. We have analised the microthreaded memory model and have used Location Consistency as the consistency model. Finally, for coherence a variant of the MOSI protocol has been specified and implemented to maintain coherence for the caches. A novel feature of this work is the proposed Suspended Request Queue implementation for both Attraction Cache and Directory to further reduced the traffic on the network.

Currently we are intensively verifying the cache coherence protocol by specifying its complete behavior in Murphy description language [15]. The language allows the user to specify initial system state and rules in addition to the procedures. From the initial state Murphi will automatically fire different rules according to the conditions specified for them. The process will continue until all system states are explored. By checking the correctness of each system state, the protocol can be verified. The technique is called State Enumeration [16]. At the current stage, the protocol has been proved free of deadlock with Murphi and we are verifying the implementation of location consistency.

In addition, the simulation of the memory system using SystemC has been completed and tested using synthetic traces. In the near future, this memory simulator will be integrated into the Microgrid CMP emulator to evaluate the overall chip architecture in a cycle-accurate manner. The design parameters and protocols will then be be tuned for the on-chip COMA memory system. Furthermore, we are developing a memory protection scheme that will provide families of microthreads exclusive access to memory protection domains.

References

1. Mercaldi, M., Swanson, S., Petersen, A., Putnam, A., Schwerin, A., Oskin, M., Eggers, S.J.: Instruction scheduling for a tiled dataflow architecture. In: ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, pp. 141–150. ACM Press, New York (2006)
2. Burger, D., Keckler, S.W., McKinley, K.S., Dahlin, M., John, L.K., Lin, C., Moore, C.R., Burrill, J., McDonald, R.G., Yoder, W.: The TRIPS Team: Scaling to the end of silicon with edge architectures. Computer 37(7), 44–55 (2004)
3. Jesshope, C.R.: A model for the design and programming of multicores. In: Advances in Parallel Computing, IOS Press, Amsterdam (Draft paper submitted, 2007)
4. Tullsen, D.M., Lo, J.L., Eggers, S.J., Levy, H.M.: Supporting fine-grained synchronization on a simultaneous multithreading processor. In: HPCA 1999: Proceedings of the 5th International Symposium on High Performance Computer Architecture, Washington, DC, USA, p. 54 (1999)

5. Gao, G.R., Sarkar, V.: Location consistency-a new memory model and cache consistency protocol. IEEE Transactions on Computers 49(8), 798–813 (2000)
6. Wulf, W.A., McKee, S.A.: Hitting the memory wall: implications of the obvious. SIGARCH Comput. Archit. News 23(1), 20–24 (1995)
7. Besedin, D.: Testing platforms with rightmark memory analyzer benchmark. part 5: Intel pentium m platform (dothan) (May 2004), http://www.digit-life.com/articles2/rmma/rmma-dothan.html
8. Intel Corporation: Intel develops tera-scale research chips (September 2006), http://www.intel.com/go/terascale/
9. Corporation, K.S.R.: Ksr1 technical summary. Technical report (1992)
10. Dahlgren, F., Torrellas, J.: Cache-only memory architectures. Computer 32(6), 72–79 (1999)
11. LeBlanc, T.J., Marsh, B.D., Scott, M.L.: Memory management for large-scale numa multiprocessors. Technical report, Rochester, NY, USA (1989)
12. Hagersten, E., Landin, A., Haridi, S.: DDM - a cache-only memory architecture. IEEE Computer 25(9), 44–54 (1992)
13. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Computers 28(9), 690–691 (1979)
14. Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P.B., Gupta, A., Hennessy, J.L.: Memory consistency and event ordering in scalable shared-memory multiprocessors. In: 25 Years ISCA: Retrospectives and Reprints, pp. 376–387 (1998)
15. Dill, D.L.: The murphi verification system. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 390–393. Springer, Heidelberg (1996)
16. Pong, F., Dubois, M.: Verification techniques for cache coherence protocols. ACM Computing Surveys 29(1), 82–126 (1997)