Architectural Implications of Cache Coherence Protocols with Network Applications on Chip MultiProcessors

Kyueun Yi, Jean-Luc Gaudiot

Department of Electrical Engineering and Computer Science
University of California, Irvine, CA 92697-2625, USA
{kyueuny, gaudiot}@uci.edu

Abstract. Network processors are specialized integrated circuits used to process packets in such network equipment as core routers, edge routers, and access routers. As predicted by Gilder's law, Internet traffic has doubled each year since 1997 and this trend is showing no signs of abating. Since all emerging network applications which require deep packet classification and security-related processing should be run at line rates and since network speed and network applications complexity continue increasing, future network processors should simultaneously meet two requirements: high performance and high programmability. Single processor performance will not be sufficient to support the requirements which will be imposed on future network processors. In this paper, we consider the CMP model as the baseline architecture of future network processors. We investigate the architectural implications of cache coherence protocols with network workloads on CMPs. Our results show that the token protocol which uses the tokens to control read/write permission of shared data blocks shows better performance than the directory protocol by a factor of 13.4%.

1 Introduction

Network processors are specialized integrated circuits used to process packets in such network equipment as core routers, edge routers, and access routers. Indeed, as predicted by Gilder's law, Internet traffic has continued doubling every year since 1997 [1]. Further, emerging network applications such as QoS, URL matching, virus detection, intrusion detection, and load balancing require deep packet classification processing [2] and security-related processing. The emerging deep packet classification processing and security-related processing are more computation-intensive than all other network applications [3]. Indeed, all network applications which require deep packet classification processing and security-related processing should be run at line rates. Since network

This work is partly supported by the National Science Foundation under Grant No. CCF-0541403. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

speeds and network applications complexity continue increasing, future network processors should simultaneously meet the two conflicting requirements of high performance and high programmability.

Since single processors will not be powerful enough to simultaneously reach high performance and high programmability [4], multiprocessors or multithreaded architectures have recently been proposed as the baseline architectures of network processors. Since modern processors have focused on exploiting Instruction-Level Parallelism (ILP), they have been quite successful at it and there is little room to exploit Instruction-Level Parallelism any longer. To overcome the limits of Instruction-Level Parallelism in wide-issue superscalar machines, two alternative microarchitectures have been proposed: Simultaneous MultiThreading (SMT) [5] and Chip MultiProcessor (CMP) [6] in order to exploit multiple threads (Thread-Level Parallelism - TLP).

Since the individual processor cores of CMPs are simple, CMPs can have low design complexity and achieve very high clock rates. Further, CMPs can efficiently reduce overall power consumption while maintaining the overall performance of them by increasing the number of processor cores and reducing clock rate of single-thread processor [7]. Since CMP provides performance scalability as well as programmability, CMP is an attractive candidate for future network processors. Indeed, single chip multiprocessors perform 50-100% better than wide-issue superscalar [6] with thread-level parallelism and multiprogramming workload. In this paper, we use the CMP architecture model as the baseline for future network processors.

When multiple processor cores are integrated on a single chip with shared cache, two different processor cores can have different values for the same location of the shared cache. This problem is called as the cache coherence problem. The architectural implications of cache coherence protocols such as the snooping protocol and the directory protocol have been investigated with commercial workloads, multiprogramming and OS workloads, and scientific/technical workloads [8, 9]. Martin *et al.* have presented the Token Protocol which uses the tokens to control read/write permission of shared cache blocks. They have compared the performance of the token protocol with other cache coherence protocols such as the snooping protocol and the directory protocol with commercial workloads [10]

In this paper, we investigate the architectural implications of cache coherence protocols on CMP processors with network workloads. The performance of each protocol (the directory protocol and the token protocol) are measured against each other, as the number of processor cores is made to vary. The token protocol shows better performance than the directory protocol by 13.4%. As the number of processor cores is increased, the number of instructions used to complete the application is also increased due to multithreading mechanism and cache coherence protocol. The results will help to design single chip multiprocessors for network workloads.

The rest of this paper is organized as follows. Section 2 describes past research on architectural implications of network workloads on single thread as well as cache coherence protocols on CMP. Cache coherence protocols which are used in this paper are explained in section 3. Our simulation environment and methodology are presented in section 4. We present the architectural implications of network workloads on CMP in section 5. Finally, we summarize our observations in section 6.

2 Related Work

As an investigation of architectural implications with network workloads on CMP, Crowley *et al.* have compared the performance of different architectures such as a SuperScalar (SS) processor, a fine-grained multithreaded processor (FGMT), a single chip multiprocessor (CMP), and a simultaneous multithreaded (SMT) processor [11]. With equivalent processor resources and dynamically exploiting both instruction-level parallelism and thread-level parallelism, SMT shows better performance than CMP and better than FGMT and SS by a factor of two.

Nahum *et al.* have presented an experimental performance of packet-level parallelism on shared-memory multiprocessor [12]. They have found that limited packet-level parallelism exists within a single connection under TCP. However, an available packet-level parallelism is increased by using multiple connections.

The architectural implications of cache coherence protocols are investigated when the following parameters are changed: numbers of processors, cache size, and block size in the cache [8]. The snooping protocol and the directory protocol are evaluated with online transaction processing workload (OLTP) and scientific/technical workloads.

Martin *et al.* have measured and compared the performance of cache coherence protocols such as the snooping protocol, the directory protocol, and the token protocol with commercial workloads [10]. They have found that the token protocol is 25-65% faster than the snooping protocol and 6-18% faster than the directory protocol.

3 Cache Coherence Protocols

It is well known that if multiple processors share a cache, two different processors can have different values for the same location of the shared cache. This problem is referred to as the cache coherence problem. A cache is said to be coherent if any read of memory location returns the most recently written value of that data element. Cache coherence for multiple processors is maintained with cache coherence protocols which make all processors have a consistent view of the shared cache and manage the read/write of data in the shared cache.

A major role of cache coherence protocols is that of tracking the state of any shared data block. The MOESI coherence state model [13] is used to represent the state of shared cache data blocks in this paper. Each cache block has 5 states in MOESI coherence state model as follows:

- Modified state: No other processor has a copy of the data block. The copy of the data in main memory is incorrect.
- Owned state: Only one processor can be in the owned state. All other processors can have a copy of the most recent in the shared state. The copy of the data in main memory can be incorrect.
- Exclusive state: No other processor has a copy of the data. The copy of the data in main memory is also the most recent.
- Shared state: Other processors can also have copies of the data in the shared state. The copy of data in main memory is the most recent.

– Invalid state: Either main memory or another processor cache can have valid copies of the data.

The previously mentioned MOESI cache coherence state model can use different cache coherence protocols which track the sharing status of a copy of block of shared cache. Three kinds of cache coherence protocols are briefly explained below.

Snooping protocols: The cache of each processor has a copy of block of shared cache and a copy of the sharing status. Snooping protocols do not have a centralized location to maintain the states of cache blocks. All cache controllers in each processor continuously snoop on the bus to discover whether they have a copy of any block currently requested on the bus. The use of broadcast limits the scalability of bus-based snooping protocols.

Directory protocols: The sharing status of a block is maintained in the directory of the home node. The directory keeps information as to which caches have copies of the block, whether it is dirty, etc. Each access to a cache block of shared cache requires to first access the directory to find out the state of the cache block. Since directory protocols do not use the bus unlike snooping protocols, directory protocols do not need for all processors to watch the interconnection network.

The Token Protocol using broadcast: The token is the base unit which is used to control the read/write permissions to shared cache blocks in the token protocol. The token protocol [10] exchanges and counts tokens to control read/write permissions to the shared cache blocks. Each logical block of a shared cache has a fixed number of tokens. When each processor has at least one of the block's token, it can read the cache block. When each processor has all of the block's tokens, it can write the cache block.

4 Simulation Environment and Methodology

We present the simulator and benchmark programs which are used to investigate the architecture implications of cache coherence protocols with network workloads on CMP. There are 4 kinds of the options for CMP implementation [14], a conventional microprocessor, a simple chip multiprocessor, a shared-cache chip multiprocessor, and a multithreaded, shared-cache chip multiprocessor. In this paper, we used a shared-cache chip multiprocessor which has a private L1 data cache, a private L1 instruction cache, and a shared L2 unified cache.

4.1 Simulator

CMP is simulated with the Simics full-system functional execution-driven simulator [15] and Ruby of GEMS [16] as a cache simulator. The processor modeled is the UltraSPARC III. The simulated system runs an unmodified Solaris operating system version 9 and 1, 2, 4, and 8 processor-cores CMPs are simulated. Two cache coherence protocols, MOESI-directory and MOESI-token, are used to evaluate network workloads. The L2 cache is organized as a non-uniform cache architecture. The configuration of L1 I-cache and L1 D-cache is 16KB, 4-way and 3 cycles. The configuration of L2 shared cache is 16MB, 4-way and 6 cycles.

4.2 NetBench

NetBench is commonly used for the evaluation of network processors and composed of nine benchmark programs [17]. NetBench is a set of benchmarks used for single thread, generally not for multithreaded parallel applications with a shared memory. Since CMP architectures allow the exploitation of thread-level parallelism, in order to run NetBench on multiprocessors with shared memory, we need to modify NetBench to support multiple threads like SPLASH-2 [18] in which child processes share the same virtual address space as their parent process. To decompose a single thread into multiple threads, we exploited packet-level parallelism as shown in Figure 1. Figure 2 shows the implementation of Figure 1.

Among nine benchmark programs of NetBench, TL (Table Lookup), ROUTE, DRR (Deficit Round Robin), and NAT (Network Address Translation) works with the routing table. When these four benchmark programs are modified to have multiple threads, they require synchronization mechanisms to share the routing table among multiple threads. Since the synchronization mechanisms cause additional spinlocks, we do not use these four programs to investigate the architectural implications of cache coherence protocol with network workloads on CMP. The DH benchmark, Diffie-Hellman public-key encryption-decryption mechanism, does not require a packet trace. Thus, we used CRC (Cyclic Redundancy Check), MD5 (Message Digest), and URL (Uniform Resource Locator) to investigate the architectural implications of cache coherence protocols with network workloads on CMP. These three benchmarks are compiled with *gcc -O3* in SunOS 5.8. To warm up the cache, we ran these programs with 160 packets. Then we processed 5,000 packets. The simulation results are gathered between two MAGIC_BREAKPOINT instructions [19] as shown in Figure 2.

The original NetBench uses the traces from Columbia University available in the public domain [20]. However, destination and source IP addresses of this trace are anonymized for privacy protection. Hence, for our purposes, we used other real packet traces [21].

5 Evaluation Results

We present the evaluation results of two cache coherency protocols, the directory protocol and the token protocol, with network workloads and investigate the architectural implications of these two cache coherence protocols on CMP.

Equation (1) shows the CPU time needed to execute a program. If two systems have the same clock cycle and run the same instructions per program, then the first term and the third term of Equation (1) are fixed and the CPU time depends on only the Clock cycles Per Instruction (CPI). Thus, to compare the performance of single processors which run single threaded and user-level programs, CPI (or IPC which is the inverse of CPI) is commonly used. However, CPI is inaccurate for multithreaded workloads running on multiprocessors. The inaccuracy is caused by the incorrect assumption that instructions per program is constant during execution of the programs. Multithreaded workloads running on multiprocessors can have different instruction paths which are caused by spinlocks and other synchronization mechanisms. The different instruction

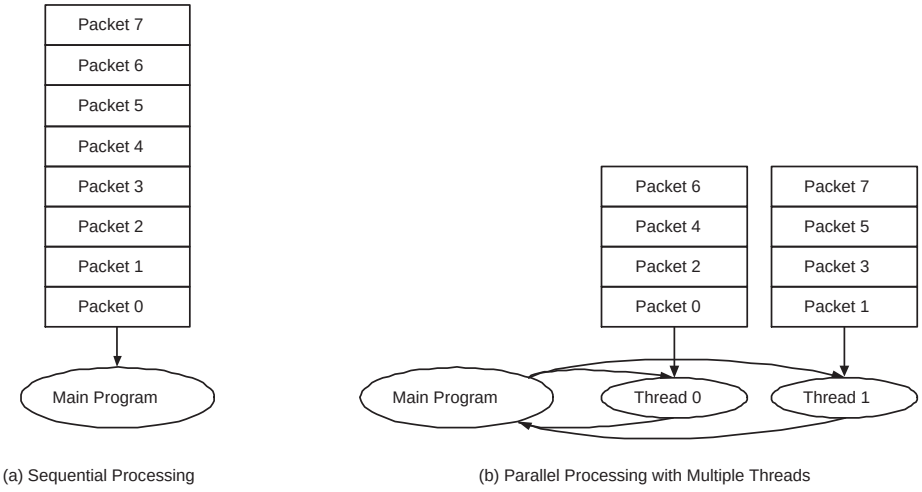


Fig. 1. Conversion single thread to multiple threads with packet-level parallelism

paths change the number of instructions to perform the same amount of work [22]. For this reason, we used as a measure the total number of cycles required to complete the programs for the performance comparison.

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \quad (1)$$

5.1 Performance comparison of cache coherence protocols

As mentioned in related work, the token protocol showed better performance than the snooping protocol and the directory protocol in commercial workloads. Figure 3 shows the performance comparison between the directory protocol and the token protocol with 3 benchmark programs from NetBench. As shown in Figure 3, the token protocol shows better performance than the directory protocol by 13.4%.

As the number of processor cores is increased, the number of cycles required to complete the programs is also increased. This effect is due to the fact that the number of instructions needed to deal with multiple threads in an operating system is increased as the number of threads is increased. In the following subsection, we investigate the instruction overhead due to the multithreading mechanism.

5.2 Instruction overhead due to the multithreading mechanism

Since multithreading is used to exploit packet-level parallelism, the instruction overhead due to the multithreading mechanism is measured. To measure the instruction overhead due to multithreading mechanism, the numbers of instructions which are used to complete benchmark programs are compared two cases: normal program without

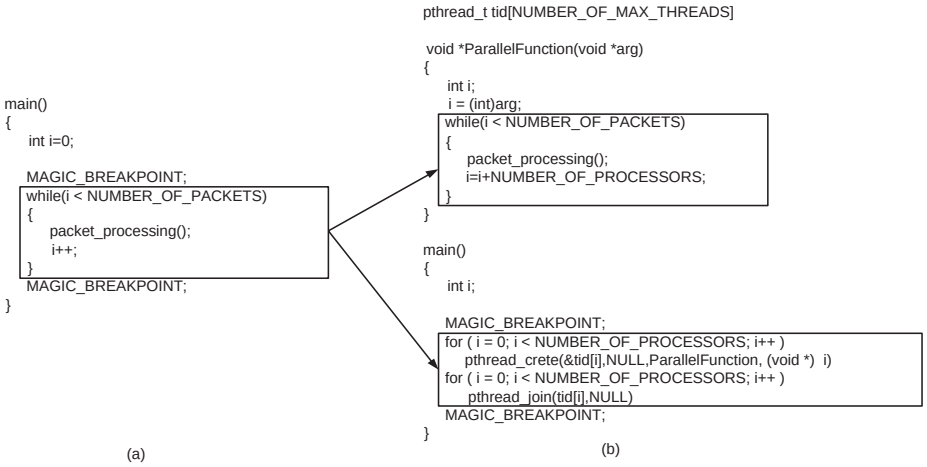


Fig. 2. Example codes to convert single thread to multiple threads with packet-level parallelism

multithreading mechanism and multithreaded program with 1-thread. As shown in Figure 4, when multithreading mechanism is used, the numbers of instructions of the directory protocol and the token protocol are increased by 9.7% and 10.3%, respectively.

5.3 L2 cache misses

The performance of a shared cache is influenced by the cache misses which occur in the single processor and the coherence misses which arise from inter-processor communication in multiprocessors.

Figure 5 shows the number of cache misses in the L2 shared cache. As the number of processor cores is increased, the number of L2 cache misses is also increased. The reason for the increase in the number of L2 cache misses is the memory contention since the L2 cache is shared among all processors. Most L2 shared cache misses are coherence misses in the multiple processor cores as shown in (a) and (b) of Figure 5.

5.4 Architectural implications

The token protocol shows better performance than the directory protocol by a factor of 13.4%. When a single-thread program is decomposed into a multithreaded program with the ability to exploit packet-level parallelism, the instruction overhead due to multithreading mechanism occurs. The instruction overhead due to multithreading is almost 10%. Most L2 shared cache misses are coherence misses in multiple processor cores. Thus, we need to reduce instruction overhead due to multithreading and L2 shared cache misses for performance enhancement of future network processors which are based on CMP.

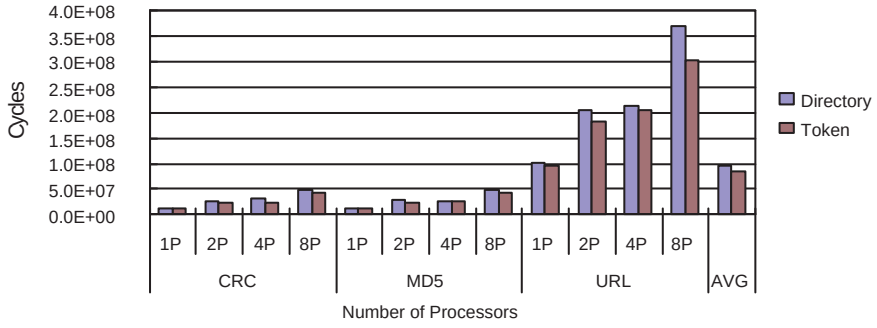


Fig. 3. Performance comparison of cache coherence protocols

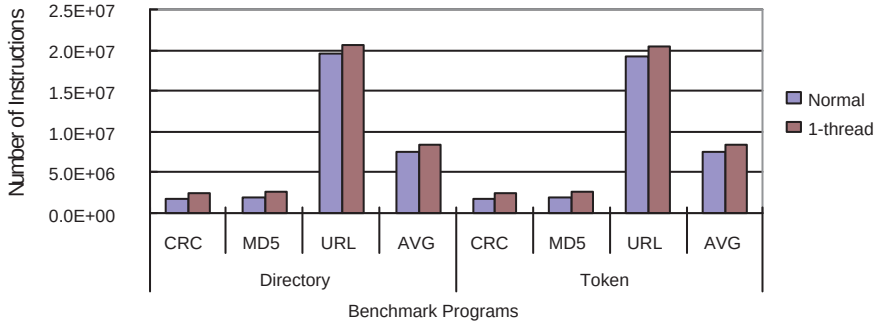
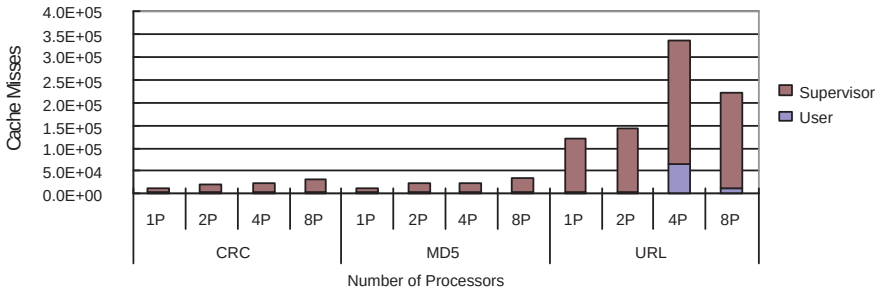


Fig. 4. Instruction overhead due to multithreading mechanism

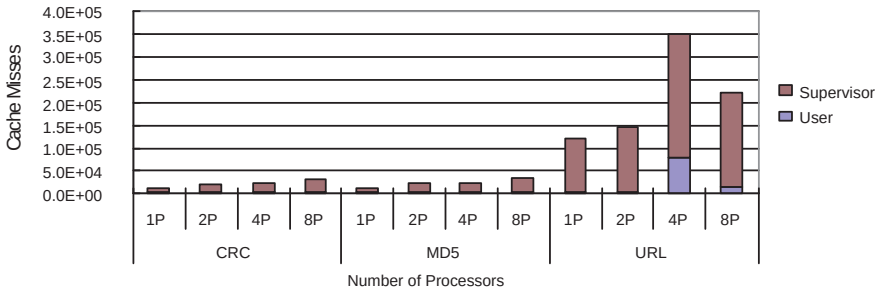
6 Conclusions

The architectural implications of cache coherence protocols, the directory protocol and the token protocol, are investigated on the CMP processor model subjected to network application workloads. The performance has been measured for each protocol with varying numbers of processors. The token protocol shows better performance than the directory protocol by a factor of 13.4%. When single-thread programs are decomposed into multithreaded programs with exploiting packet-level parallelism, the instruction overhead due to multithreading mechanism occurs. The instruction overheads of the directory protocol and the token protocol are 9.7% and 10.3%, respectively. Most L2 shared cache misses are coherence misses in multiple processor cores.

As future work, the architectural implications of cache coherence protocols need to be investigated with varying cache size and block size. We will also investigate the ar-



(a) Number of L2 cache misses in the directory protocol



(b) Number of L2 cache misses in the token protocol

Fig. 5. Number of L2 cache misses

chitectural implications of simple CMP multiprocessor with network workloads which exploit multiprogramming instead of multithreading.

References

1. Odlyzko, A.M.: Internet traffic growth: Sources and implications. In: Proceesings of SPIE Optical Transmission Systems and Equipment for WDM Networking II. Volume 5247. (Aug 2003) 1–15
2. Gebali, F., Rafiq, A.N.M.E.: Processor Array Architectures for Deep Packet Classification. IEEE Transactions on Parallel and Distributed Systems **17**(3) (March 2006) 241–251
3. Kant, K., Iyer, R., Mohapatra, P.: Architectural Impacet of Secure Socket Layer on Internet Servers. In: The Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers and Processors, Austin, Texas, USA (September 2000) 7–14
4. Lee, B.K., John, L.K.: NpBench: A Benchmark Suite for Control plane and Data plane Applications for Network Processors. In: Proceesings of 21st International Conference on Computer Design(ICCD'03). (2003) 226–233

5. Tullsen, D.M., Eggers, S.J., Levy, H.M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism. In: Proceedings of 22nd Annual International Symposium on Computer Architecture. (1995) 392–403
6. Olukotun, K., Nayfeh, B.A., Hammond, L., Wilson, K., Chang, K.: The Case for a Single-Chip Multiprocessor. SIGOPS Operating System Reviev **30**(5) (1996) 2–11
7. Crowley, P., Franklin, M., Buhler, J., Chamberlain, R.: Impact of CMP Design on High-Performance Embedded Computing. In: Proceedings of the High Performance Embedded Computing Workshop (HPEC 2006), MIT Lincoln Laboratory, Lexington, MA (September 2006)
8. J. L. Hennessy and D. A. Patterson: Computer Architecture A Quantitative Approach, 3rd edition. Morgan Kaufmann (2003)
9. Barroso, L.A., Gharachorloo, K., Bugnion, E.: Memory System Characterization of Commercial Workloads. In: ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture, Washington, DC, USA, IEEE Computer Society (1998) 3–14
10. Martin, M.M.K., Hill, M.D., Wood, D.A.: Token Coherence: Decoupling Performance and Correctness. In: ISCA '03: Proceedings of The 30th Annual International Symposium on Computer Architecture, New York, NY, USA, ACM Press (2003) 182–193
11. Crowley, P., Fiuczynski, M.E., Baer, J.L., Bershad, B.N.: Characterization Processor Architectures for Programmable Network Interfaces. In: Proceedings of the 2000 International Conference on Supercomputing. (2000)
12. Nahum, E.M., Yates, D.J., Kurose, J.F., Towsley, D.F.: Performance issues in parallelized network protocols. In: Operating Systems Design and Implementation. (1994) 125–137
13. Sweazey, P., Smith, A.J.: A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus. SIGARCH Computer Architecture News **14**(2) (1986) 414–423
14. Olukotun, K., Hammond, L.: The future of microprocessors. ACM Queue **3**(7) (2005) 26–29
15. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A Full System Simulation Platform. IEEE Computer **35**(2) (February 2002) 50–58
16. Martin, M.M.K., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D., Wood, D.A.: Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. SIGARCH Computer Architecture News **33**(4) (2005) 92–99
17. Memik, G., Mangione-Smith, W.H., Hu, W.: NetBench: A Benchmarking Suite for Network Processors. In: ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, Piscataway, NJ, USA, IEEE Press (2001) 39–42
18. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations. In: Proceedings of the 22th International Symposium on Computer Architecture. (1995) 24–36
19. Virtutech: Simics User Guide for Unix. 2.2.19 edn. (August 2005)
20. Passive Measurement and Analysis project, National Laboratory for Applied Network Research, http://moat.nlanr.net/Traces
21. D. E. Comer: Computer Networks and Internets with Internet Applications, 4th edition. Prentice Hall (2004)
22. Alameldeen, A.R., Wood, D.A.: IPC Considered Harmful for Multiprocessor Workloads. IEEE Micro **26**(4) (Jun/Aug 2006) 8–17