Reducing the Associativity and Size of Step Caches in CRCW Operation

Martti Forsell[1]

[1]VTT Technical Research Centre of Finland
Platform Architectures Team
Box 1100, FI-90571 Oulu, Finland
Martti.Forsell@VTT.Fi

Abstract

Step caches are caches in which data entered to an cache array is kept valid only until the end of ongoing step of execution. Together with an advanced pipelined multithreaded architecture they can be used to implement concurrent read concurrent write (CRCW) memory access in shared memory multiprocessor systems on chip (MP-SOC) without cache coherency problems. Unfortunately obvious step cache architectures assume full associativity, which can become expensive since the size and thus associativity of caches equal the number of threads per processor being at least the square root of the number of processors. In this paper, we describe a technique to radically reduce the associativity and even size of step caches in CRCW operation. We give a short performance evaluation of limited associativity step cache systems with different settings using simple parallel programs on a parametrical MP-SOC framework. According to the evaluation, the performance of limited associativity step cache systems comes very close to that of fully associative step cache systems, while decreasing the size of caches decreases the performance gradually.

1. Introduction

Due to ongoing gradual changes in silicon technology, designs relying on long wires and extremely high frequency components will face serious feasibility and yield problems and are expected to be gradually replaced with designs trading frequency to silicon area and parallelism [Brooks00, Mudge01, Flynn05]. This will not be easy since so far majority of computing engines has been designed with maximal frequency and sequential execution paradigm in mind. Among the most promising technologies utilizing parallelism are *multiprocessor systems on chip* (MP-SOC) and *networks on chip* (NOC) [Jantsch03]. Unfortunately the architecture of most MP-SOC designs has been mainly copied from the sequential architecture limiting their use to certain very narrow application areas [Cesario02,Forsell02b,Ye04]. The latest architectural advances, however, indicate that wide application areas can be covered with a single MP-SOC architecture [Forsell02a,Forsell05b]. One missing features of these architectures has been ability to provide *concurrent read concurrent write* (CRCW) memory access providing logarithmically faster execution time for a class of algorithms [Jaja92]. The existing solutions have sequentialized references on memory modules causing severe congestion since an excessive amount of references gets routed to the same target location [Forsell02a]. There exists an alternative solution avoiding congestion by combining messages that are targeted to the same location, but this requires a separate sorting phase prior to actual routing phase, which decreases the performance and increases the complexity [Ranade91, Keller01].

Recently, a novel class of cache memories attached to processors, called *step caches*, has been proposed to eliminate extra steps in CRCW operation [Forsell05a]. The main idea of step caches is that cache lines are valid only until the end of the ongoing step of multithreaded execution. Together with an advanced pipelined multithreaded architecture they can be used to implement CRCW memory access in shared memory multithreaded MP-SOCs without cache coherency problems. Unfortunately obvious step cache architectures assume full associativity, which can become expensive in terms of silicon area since the size and associativity of caches equal the number of threads per processor being at least the square root of the number of processors. In this paper, we describe a technique to radically reduce the associativity and even size of step caches in CRCW operation. We give a short performance evaluation of limited associativity step cache systems with different associativity and size settings using simple parallel programs on a parametrical MP-SOC framework. According to the evaluation, the performance of limited associativity step caches systems comes close to that of fully associative

1-4244-0054-6/06/$20.00

step cache systems while decreasing the size of caches decreases the performance gradually.

The rest of the paper is organized so that in section 2 we give basic definitions of caches, describe structure and operation of step caches and outline implementation of CRCW operations in MP-SOCs. In section 3 we introduce a technique to reduce the associativity and size of step caches in CRCW operation. An evaluation of limited associativity step caches is given in section 4. Finally, in section 5 we give our conclusions.

2. Step Caches and CRCW

In order to define basic terms and to understand better the idea of step caches with respect to ordinary caches, we give a short overview of cache memories before proceeding to structure and operation of step caches.

2.1 Cache Memories

Cache memories are small and fast associative memories that are commonly used to balance the speed difference between processors and main memory by keeping the potentially most frequently used data in the cache from which it can be quickly retrieved [Kilburn62]. In a case of a memory access, data is first searched from or stored to the cache. If it is found, a cache hit occurs and data can be delivered fast back to the processor. If data is not found, a cache miss occurs and an access to the slower main memory is performed. After the access is completed data is delivered to both the processor and stored into the cache so that further references to the same data can be performed faster. Typically, it is beneficial to move a number of consecutive data words (a cache line) to the cache since it is likely that data itself or data located close to it will be accessed again soon. Since the size of the cache is usually much smaller than that of the main memory, it may happen that the place required for data to be stored in the cache is already occupied by previously accessed data. In such a case previous data can be overwritten after it is possibly stored into the main memory, since cache and main memory may not be coherent as a result of a cache-only write operation. In modern computers there are usually more than one level of caches between processors and main memories forming a hierarchy of memories increasing in size but decreasing in access time.

Caches are divided to three classes according to placement convention—fully associative, *N*-way set associative, and direct mapped: In fully associative caches data to be stored in the cache can be placed anywhere in the cache, while in direct mapped caches there is only one place that can hold particular data. The *N*-way set associative cache organization in an intermediate solution between fully associative and direct mapped caches allowing data to be stored into one of the *N* alternative places. The way of defining which of the alternative locations will be used if they all are occuopied is called replacement policy. Typical replacement policies include least frequently used, first in first out, and random. The associativity is implemented by storing partially the address next to corresponding data storage for each cache line and comparing if the address of searched data matches the addresses of stored data.

Typical measures related to caches are

Size	The capacity of the cache
Line length	Length of a cache line in words
Associativity	Associativity of data placement as described above
Latency	The latency of cache access in processor clock cycles
Hit rate	The ratio between cache hits and total cache accesses

2.2 Step Caches

A step cache is a *C* line, single *W*-bit word per line cache with two special fields (pending, step), a slightly modified control logic, and step-aware replacement policy. Figure 1 shows the organization of a fully associative step cache consisting of *C* lines, *C* comparators, *C* to 1 multiplexer, and simplified decay logic matched for a T_p-threaded processor attached to a *M*-word main memory.

Each cache line has the following fields:

In use	A bit indicating that the line is in use.
Pending	A bit indicating that the data is currently being retrieved from the memory
Tag	The address tag of the line
Step	Two least significant bits of the step of the data write.
Data	Storage for a data word.

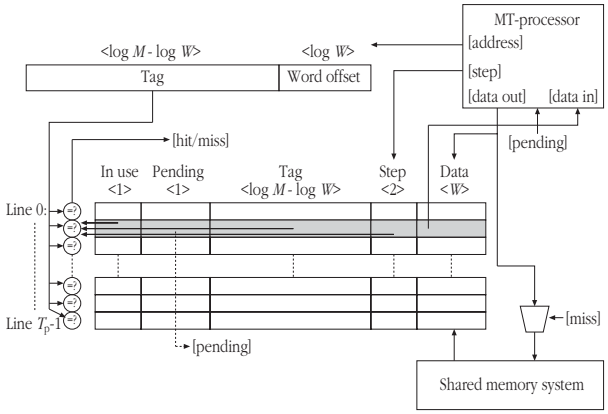

Figure 1. The organization of a fully associative step cache.

Step caches operate similarly as ordinary caches with few exceptions: Each time a processor refers the memory a step cache search is performed. A hit is detected on a cache line if the line is in use, the address tag matches the tag of the line, and the least significant bits of step of the reference matches the step of the line. In the case of a hit, a write is just ignored and a read is just completed by accessing the data from the cache. In the case of a miss, the reference is stored into the cache using the replacement policy and marked as pending (for reads). At the same time with storing to the cache line, the reference is sent to the lower-level memory system. When a reply of a read arrives from the memory, the data is put to the data field of the line storing the reference information and the pending field is cleared. The cache decay logic takes care of invalidating the lines before their step field matches again to the least significant bits of current step.

Step caches avoid cache coherency problems since the references within a single step of multithreaded execution are independent by definition. Similarly, there is no need to write any data from a step cache to the memory since its content is always coherent.

2.2 Implementing Concurrent Access

Consider an advanced shared memory MP-SOC system with P T_p-threaded W-bit processors attached to the M-word shared memory. Step caches with capacity and associativity of T_p lines per processor can be used to efficiently implement concurrent memory access on such systems by attaching a step cache between processors and the memory system. This is because they filter out step-wisely all but the first reference for each referred location and avoid conflict misses due to sufficient capacity and step-aware replacement policy. The traffic generated by concurrent accesses drops radically because at most P references can occur per a memory location per single step.

3. Reducing the Associativity and Size

The associativity requirement of the step cache solution described in section 2 equals T_p, which is at least the square root of the number of processors in advanced MP-SOC machines to hide the latency of memory accesses [Forsell02a]. This can explode the silicon area requirements of the step cached system if the number of processors is larger than, say, 4 or the number of threads per processor is larger than 16.

The associativity can be reduced to a S-way set associative or even direct mapped organization by allowing an initiated sequence of references to a certain memory location to be interrupted by another reference to a different location if the capacity of the referred set of cache lines is exceeded. In order to distribute this kind of conflicts (almost) evenly over the cache lines, access addresses are hashed with a randomly selected hashing function. This kind of technique is already used in memory modules of some MP-SOCs and parallel computers to balance the speed difference between processors and memory banks with a high probability [Forsell02a, Keller01].

Consider a similar MP-SOC setup than in section 2. Let $h(x)$ be a linear hashing function randomly selected from a family of hashing functions implemented by a special hasher block attached between the processor and step cache. Assume that the fully associative step caches are replaced with caches featuring capacity of T_p $h(x)$-hashed lines per processor and associativity $S<T_p$ (see Figure 2). The obtained solution implements concurrent memory access with a low overhead with respect to the fully associative solution with a high probability since the number of conflicts and thus additional traffic remains low with a high probability. As a result of decreasing associativity of step caches, the number of comparators and degree of the multiplexer drops from T_p to S. At the same time also the size dependence of step caches on the number of threads per processor, T_p, is cut, but this happens of course with the cost of increased memory traffic. In the next section we will evaluate the practical effect of reducing associativity and size of step caches to CRCW access performance of MP-SOCs.

4. Evaluation

We evaluated the proposed limited associativity and size step cache solutions and their fully associative counterparts assuming a step aware FIFO replacement policy by measuring key properties of parallel execution of five two-

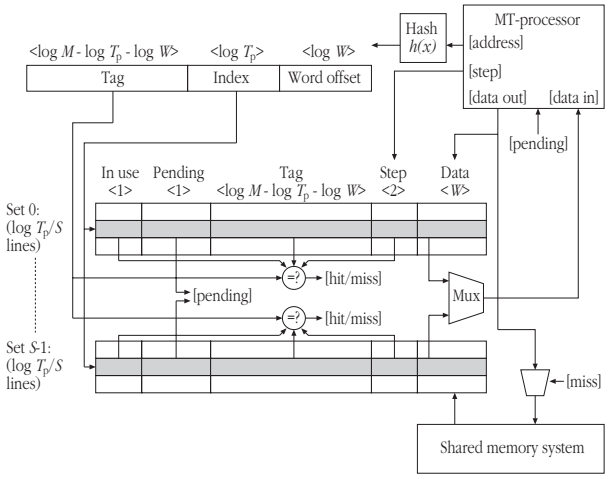

Figure 2. The organization of a limited associativity (S-way set associative) step cache.

component programs (see Table 1) mixing the CRCW and exclusive read exclusive write (EREW) memory access components in 0%-100%, 25%-75%, 50%-50%, 75%-25%, 100%-0% ratio on three step cached systems (see Table 2) based on the Eclipse MP-SOC framework [Forsell02a] with three associativity settings (direct mapped, 4-way set associative, 16-way set associative and fully associative) and three size settings (T_p, $T_p/2$ and $T_p/4$). For reference purposes we made similar measurements on similarly configured non-step cached systems and ideal shared memory systems.

The benchmark program components were written in Eclipse assembler and optimized by hand. The CRCW component performs a series of concurrent reads and writes for each thread and the EREW component performs memory patterns extracted from a random SPEC CPU 2000 memory access traces on Alpha processor and shifted by a random constant for each thread. The two-component trace-based benchmarking approach was used to be able to determine the effect of concurrent memory access in various use cases and to overcome the practical limitations of the application development system. This kind of an evaluation method gives quite reliable on results on performance and usability of the proposed step cache solution. Since the actual SPEC programs are not likely to refer memory as often as the traces executed at the rate of one reference per step, the obtained results may be a bit too pessimistic.

The measurements were made with a modified IPSMSim simulator [Forsell02c]. For each evaluated benchmark-configuration pair we measured the execution time, step cache hit rate, and maximum latency of memory references. Due to extensive simulation times the 64 processor configuration was evaluated only for the 50%-50% component mix ratio benchmark and a shortened number of steps, and thus these results are not fully comparable to those of 4 and 16 processor configurations and are presented in as a separate graph.

CRCW	A parallel program component that reads and writes concurrently a given sequence of locations in the shared memory
EREW	A parallel program component issuing a memory pattern extracted from random SPEC CPU 2000 data reference trace on Alpha architecture [Milenkovic03] for each thread.

Table 1. The program components that are mixed in 0%-100%, 25%-75%, 50%-50%, 75%-25% and 100%-0% ratios to form five benchmark programs.

	E4	E16	E64
Processors	4	16	64
Threads per processor	512	512	512
Functional units	5	5	5
Bank access time	1 c	1 c	1 c
Bank cycle time	1 c	1 c	1 c
Length of FIFOs	16	16	16

Table 2. The configurations of machines used in evaluation (c=processor clock cycles).

The results of measurements are shown in Figures 3-6. We can make the following observations from the results:

• The overhead of CRCW operation in the step cached systems decreases as the degree of concurrent access increases indicating that the step cache technique works efficiently (see Figure 3 and 4). The overhead remains very low for the fully associative and 4-way set associative configurations and stays at acceptable level for the rest of the step cached configurations except for the 4-way associative quarter sized configurations while the overhead comes close to the number of processors for the non-step cached systems due to sequentialization of references at the module level. Increasing the number of processors increases slowly the overhead since the architectural capacity for hiding the memory latency, i.e. the number of threads per processor, is not increased accordingly but is kept as a constant (see Figures 3 and 4).

• The step cache hit rate comes close to the CRCW-EREW component mix ratio in the step cached configurations except in the 4-way set associative quarter size configuration where it begins to decrease as the degree of concurrent access decreased (see Figure 5). As the number of processors is increased the hit rate decreases very slowly due to constant threading configuration.

• Due to efficiency of the step caching solutions the maximum latency of memory access decreases radically as the degree of concurrent access increases for the fully associative and 4-way set associative configurations (see Figure 6). Decreasing latency can be detected also with the direct mapped and limited size organizations, but now decreasing happens more gradually and randomly. Finally, the latency of memory operations increases radically as the degree of concurrent access increases in the non-step cached organizations. This is expected behavior since the amount of sequentialized references per location increases as the degree of concurrent access increases.

5. Conclusions

We have described a technique to radically reduce the associativity and even size of step caches in advanced shared memory MP-SOCs in CRCW operation by allowing

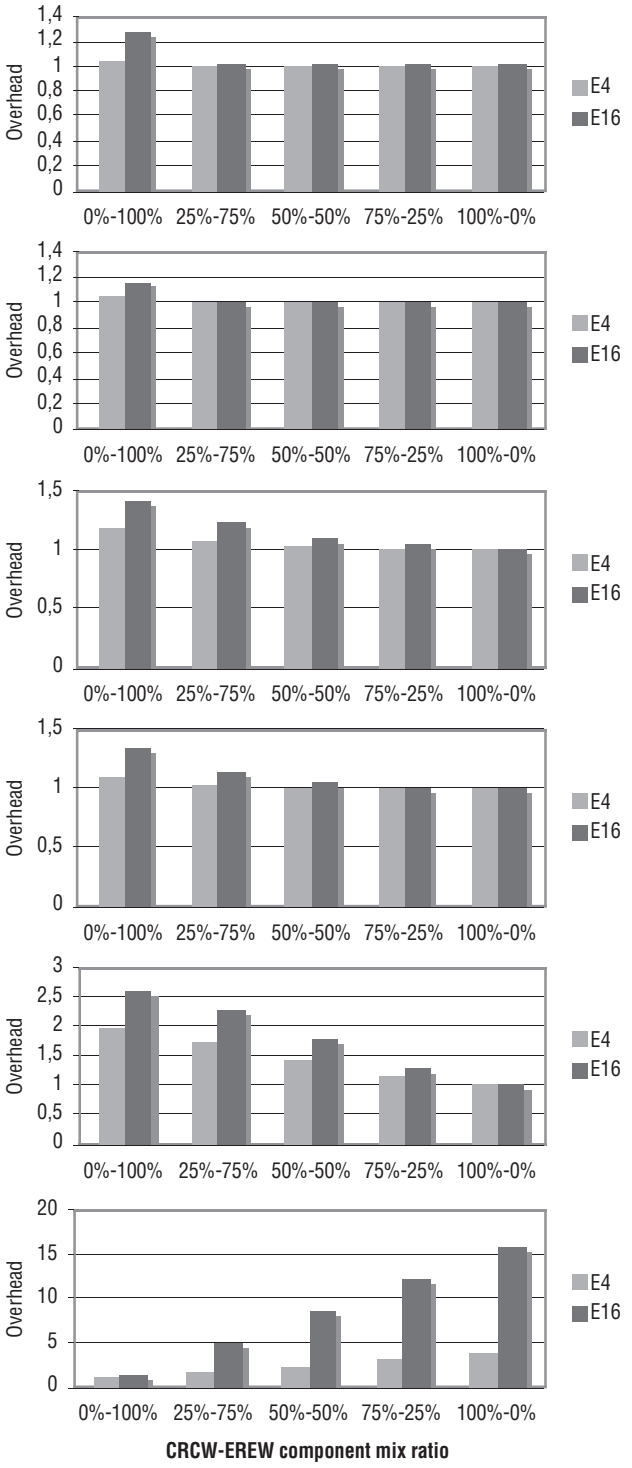

Figure 3. The execution time overhead of concurrent memory access with respect to similarly configured ideal machine. From top to bottom: fully associative, 4-way set associative, direct mapped,4-way set associative half size, 4-way set associative quarter size, and non-step cached.

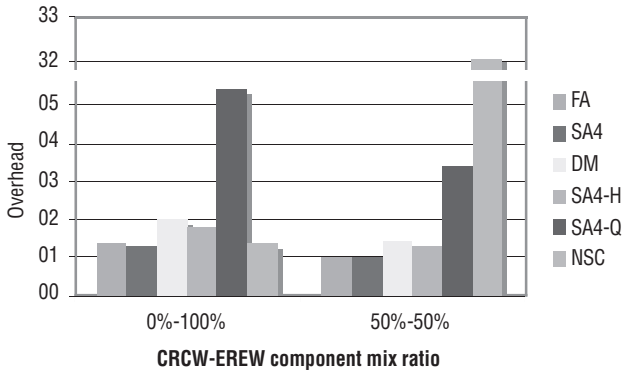

Figure 4. The execution time overhead of concurrent memory access with respect to similarly configured ideal machine for 0%-100% and 50%-50% CRCW-EREW component mix ratios and E64. (FA=fully associative, SA4 = 4-way set associative, DM = direct mapped, SA4-H = 4-way set associative half size, SA4-Q = 4-way set associative quarter size, NSC = non-step cached.

conflict misses and distributing them evenly over the cache lines with a high probability. We committed a performance evaluation of limited associativity and size step cached systems with different settings using simple parallel programs on our parametrical MP-SOC framework. According to the evaluation, the performance of limited associativity step cached systems comes very close to that of fully associative step cache systems and decreasing the size of caches decreases the performance of the measured systems gradually. As the side effect of decreasing associativity and size the overhead of exclusive memory access increases, but it is possible to partially eliminate this problem by adding separate non-step cached memory referencing instructions for exclusive access.

Our future work includes more thorough testing of step caching and expanding it for active memory multioperations [Forsell05b] that are useful in further dropping the execution time of parallel algorithms. This is not trivial since the CRCW implementation technique described in this paper does not work with multioperations taking two steps to execute and potentially having different conflict pattern for each step. We also plan to extend the proposed step caching technique to systems with long latency memory modules to allow applying this technique to high performance computing systems requiring extensive amounts of memory. Finally we are planning a journal publication on the architectural support for the computational model that can be implemented using the proposed step caching technique.

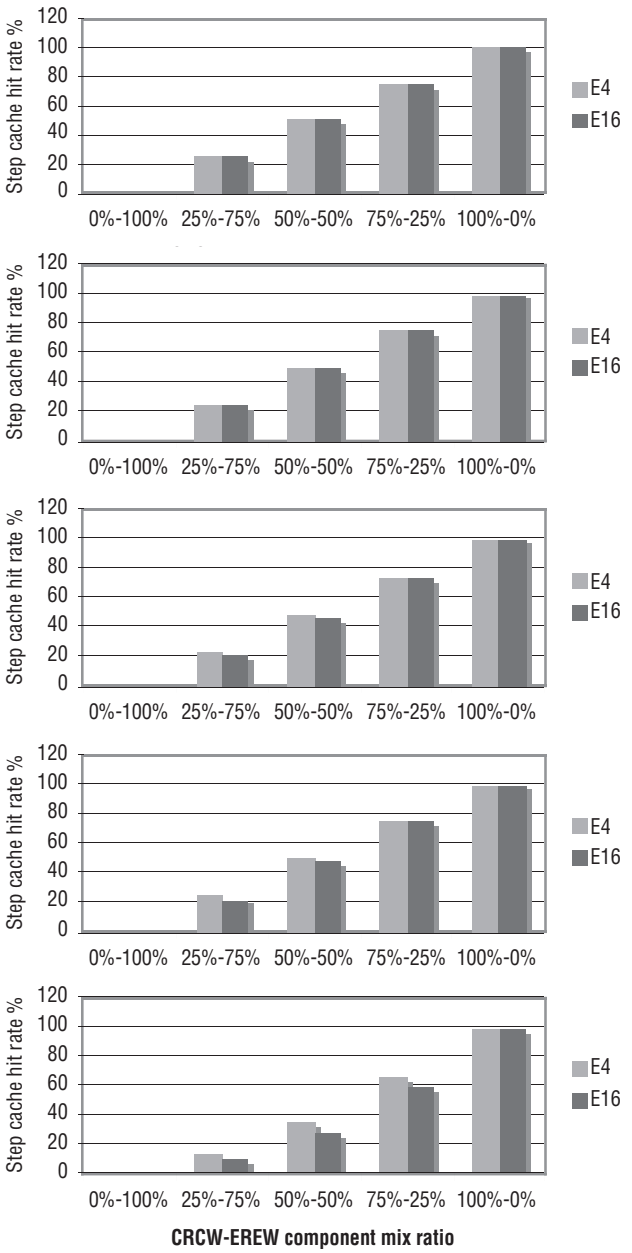

Figure 5. The hit rate of step caches. From top to bottom: fully associative, 4-way set associative, direct mapped, 4-way set associative, direct mapped,4-way set associative half size, and 4-way set associative quarter size.

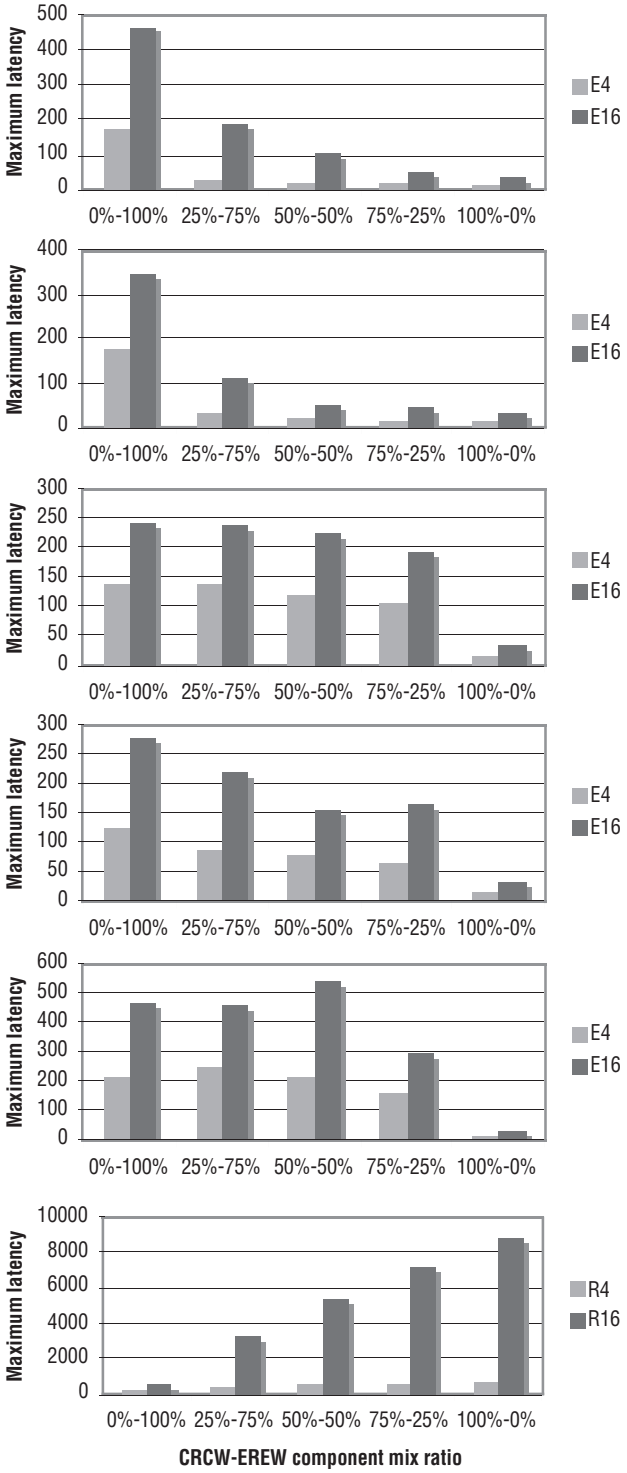

Figure 6. The maximum latency of step caches in clock cycles. From top to bottom: fully associative, 4-way set associative, direct mapped, 4-way set associative, direct mapped, 4-way set associative half size, 4-way set associative quarter size, and non-step cached.

Acknowledgements

This work was supported by the grant 107177 of the Academy of Finland.

References

[Brooks00] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta and P. Cook, Power-Aware Microarchitecture: Design and Modeling Challenges for Next Generation Microprocessors, IEEE Micro 20, 6 (November-December 2000), 26-32.

[Cesario02] W. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, A. Jerraya, Multiprocessor SoC platforms: a component-based design approach, IEEE Design and Test of Computers 19, 6 (2002), 52-63.

[Forsell02a] M. Forsell, A Scalable High-Performance Computing Solution for Network on Chips, IEEE Micro 22, 5 (September-October 2002), 46-55.

[Forsell02b] M. Forsell, Architectural differences of efficient sequential and parallel computers, Journal of Systems Architecture 47, 13 (July 2002), 1017-1041.

[Forsell02c] M. Forsell, Advanced Simulation Environment for Shared Memory Network-on-Chips, In the Proceedings of the 20th IEEE NORCHIP Conference, November 11-12, 2002, Copenhagen, Denmark, 31-36.

[Forsell05a] M. Forsell, Step Caches—a Novel Approach to Concurrent Memory Access on Shared Memory MP-SOCs, In the Proceedings of the 23th IEEE NORCHIP Conference, November 21-22, 2005, Oulu, Finland, 74-77.

[Forsell05b] M. Forsell, Realizing constant time parallel algorithms with active memory modules, International Journal of Electronic Business 3, 3-4 (2005), 255-263.

[Flynn05] M. Flynn, Microprocessor Design Issues: Thoughts on the Road Ahead, IEEE Micro 25, 3 (May-June 2005), 16-31.

[Jaja92] J. Jaja, Introduction to Parallel Algorithms, Addison-Wesley, Reading, 1992.

[Jantsch03] A. Jantsch and H. Tenhunen (editors), Networks on Chip, Kluver Academic Publishers, Boston, 2003.

[Keller01] J. Keller, C. Keßler, and J. Träff, Practical PRAM Programming, Wiley, New York, 2001.

[Kilburn62] T. Kilburn, D. Edwards, M. Lanigan, and F. Sumner, One-level storage system, IRE Transactions on Electronic Computers EC-11, April 1962, 223-235.

[Milenkovic03] A. Milenkovic and M. Milenkovic, Exploiting Streams in Instruction and Data Address Trace Compression, In the Proceedings of the IEEE 6th Annual Workshop on Workload Characterization, October 27, 2003, Austin, USA, 99-107.

[Mudge01] T. Mudge, Power: A First-Class Architectural Design Constraint, Computer 34, 4 (April 2001), 52-58.

[Ranade91] A. Ranade, How to Emulate Shared Memory, Journal of Computer and System Sciences 42, (1991), 307-326.

[Ye04] T. Ye, L. Benini and G. De Micheli, Packetization and routing analysis of on-chip multiprocessor networks, Journal of Systems Architecture 50, 2-3 (2004), 81-104.