Memory Subsystem Characterization in a 16-Core Snoop-Based Chip-Multiprocessor Architecture

Francisco J. Villa, Manuel E. Acacio, and José M. García

Departamento de Ingeniería y Tecnología de Computadores,
Universidad de Murcia, 30071 Murcia, Spain
{fj.villa, meacacio, jmgarcia}@ditec.um.es

Abstract. In this paper we present an exhaustive evaluation of the memory subsystem in a chip-multiprocessor (CMP) architecture composed of 16 cores. The characterization is performed making use of a new simulator that we have called DCMPSIM and extends the Rice Simulator for ILP Multiprocessors (RSIM) with the functionality required to model a contemporary CMP in great detail.

To better understand the behavior of the memory subsystem, we propose a taxonomy of the L1 cache misses found in CMPs which subsequently we use to determine where the hot spots of the memory hierarchy are and, thus, where computer architects have to place special emphasis to improve the performance of future dense single-chip multiprocessors, which will integrate 16 or more processor cores.

Keywords: Dense chip-multiprocessors, memory subsystem, snoop-based cache-coherence, high-performance interconnection networks.

1 Introduction

As integration scales grows, the number of transistors available on a die is increasingly becoming larger, and billion transistor chips will soon be possible. The role of computer designers is to translate all this raw potential into increased computational power. For this, efficient architectures must be designed. One of the approaches recently proposed in the literature to make efficient use of this huge number of transistors are single chip-multiprocessors [1,2]. A CMP integrates several processor cores onto a chip, as well as other resources such as the cache hierarchy and the interconnection network. As a result of this, the traditional advantages of parallel architectures are kept, but some of their drawbacks, such as wire delays or network latencies, are minimized.

The viability and importance of chip multiprocessors is further supported by a number of recently announced commercial, small-scale CMP designs [3,4]. However, nowadays the best organization of the components in this kind of architecture is not clear, and there is still much work to be done in order to improve the performance and maximize the utilization of the resources in a CMP. Besides, state-of-the-art CMPs and recent commercial releases integrate 2, 4 or at most 8 processor cores onto the chip, but they do not deal with future

L.T. Yang et al. (Eds.): HPCC 2005, LNCS 3726, pp. 223–232, 2005.

dense-CMPs (or D-CMPs), in which 16 processor cores or more are expected to be integrated. D-CMPs impose new restrictions that do not appear in current chip-multiprocessors and, thus, it is necessary to evaluate the problems of this kind of architecture.

This paper presents an exhaustive evaluation of the memory subsystem in a chip-multiprocessor (CMP) architecture composed of 16 cores. To the best of our knowledge this is the first characterization for a dense-CMP architecture with a detailed execution model for each core. As we are specially interested in the behavior of the memory hierarchy, we propose a taxonomy of the private L1 cache misses in terms of how these misses are satisfied. This taxonomy provides statistics which help us to identify the bottlenecks of a future chip multiprocessor. The evaluation is accomplished making use of a new simulator which is an extension of the well-known RSIM (Rice Simulator for ILP Multiprocessors) [5] and that we have called DCMPSIM. It models accurately a CMP with a configurable number of cores, private L1 caches connected together via a split-transaction bus and a shared, multibanked L2 cache.

The key contributions of this work are:

- We perform a detailed memory subsystem evaluation of a 16-core snoop-based CMP architecture when executing parallel workloads. This contrasts with the majority of previously published works, as CMPs has been used usually to execute multiprogrammed workloads.
- We present a novel taxonomy of the L1 cache misses found in a CMP.
- We show that the main bottleneck of a 16-core snoop-based D-CMP is the shared bus. Besides, we see that the vast majority of transactions snooped by the cache controllers are not concerned with the lines in that cache, which induces too much unnecessary work.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 describes briefly some implementation issues of the new simulator. We present a taxonomy of L1 cache misses found in CMPs in Section 4. In Section 5 we show a detailed performance evaluation of a CMP composed by 16 cores. Finally, Section 6 contains our main conclusions and outlines some future work.

2 Related Work

Some works have evaluated the usefulness of the coherence actions obtaining similar results than ours in the context of a 4-way SMP in [6] and in the context of a 16-core CMP [7]. There are some other recent works in the literature dealing with the performance of the memory hierarchy in CMP architectures. Beckmann and Wood present several proposals to reduce the impact of wire delays on large, shared L2 caches in future CMPs [8]. They implement a directory-based coherence protocol and a 2D-mesh interconnection network, evaluating a CMP composed of 8 processor cores.

Liu *et al.* [9] study several L2 cache organizations in order to increase the utilization (and in last term, the performance of the memory hierarchy) of this structure. They propose a mechanism that dynamically assigns L2 splits to each processor. In this way, they obtain better utilization of the L2 cache, taking into account the demands of each processor at every moment. However, they do not consider other possible bottlenecks, such as the presence of the shared bus. As in the previous work, they simulate a CMP composed of 8 cores. In [10] the authors propose a central coherence unit and a new cache coherence protocol in order to reduce shared-bus transaction time. They evaluate several configurations with a variable number of processors, ranging from 2 to 8. However, the integration scale and memory subsystem latencies assumed in the paper differ highly from those found nowadays, so their results are not comparable with ours.

Finally, some other authors have studied the memory hierarchy when combining chip-multiprocessing with speculative multithreading, although most of them focus on the support for thread-level memory speculation [11,12,13]. Among them, Yanagawa *et al.* [14] perform a complexity analysis of a cache controller designed by extending a MSI controller to support thread-level memory speculation. They use a directory-based mechanism to maintain coherence, and find that the main component of memory latency is the delay incurred when accessing the directory.

3 DCMPSIM: A Detailed CMP Simulator Based on RSIM

DCMPSIM models the architecture shown in Figure 1. In this architecture, we have an arbitrary number of processor cores, each one with its unified L1 instruction/data cache. All the cores share a unified, multibanked L2 cache through a split-transaction bus. Finally, main memory is interleaved, and each L2 bank is connected to a main memory module.

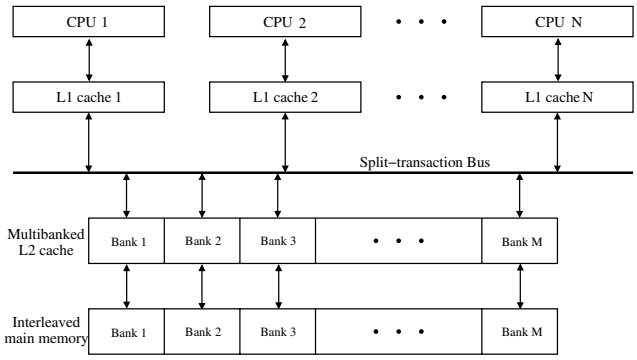

Fig. 1. The CMP architecture implemented

As we stated in Section 1, our tool has been derived from Rice Simulator for ILP Multiprocessors (RSIM) [5]. RSIM implements a directory protocol [15] in order to keep cache-coherence between nodes; directory protocols are commonly employed in machines composed of a large number of processors (in these cases, it is imperative that the coherence protocol scales with the number of processors, which is not possible with snoop-based protocols) or when there is an unordered interconnection network, such as a mesh or a torus, so it is not possible to rely in the network to ensure a partial order of the memory references.

However, our CMP architecture interconnects the private L1 caches via a shared, ordered, split-transaction bus. In this case, cache-coherence can be maintained by using a snoop-based protocol. As most CMPs do, we have implemented the MOESI snooping protocol [16]. MOESI protocol introduces an *owned* state which is used when one or more caches have a valid copy of the line, main memory does not hold a valid copy, and thus, one of the caches with a copy of the line is responsible for providing the line when it is requested by another processor. This mechanism is an optimization of the MESI protocol, which tries to take advantage of the shorter access time of a small cache when compared with a bigger memory structure.

When implementing a snoop based protocol, it is necessary to adopt some design decisions concerning how conflicting requests are managed. In our design, the L1 cache controllers snoop replies as well as requests (this is not the case of L2 cache controller, which only snoops requests), so read requests to the same location are optimized. When a cache controller sees that there is an outstanding read to the same line, it does not put the new request into the bus, but obtains the value through the reply to the original request. In the case of an outstanding write, subsequent reads or writes are not allowed to proceed until the previous write is completed. These design decisions are commonly taken in most SMP designs [17].

4 A Taxonomy of the L1 Cache Misses Found in CMPs

We are interested in measuring the latency of L1 misses in order to evaluate the performance of the memory subsystem. L1 miss latency can be divided into three categories, corresponding to the cycles spent at the L1 cache controller, the shared bus and those spent obtaining data. More specifically, these three components are the following:

- $T_{controller}$ is the time spent in the L1 cache controller. It includes the time spent until the request is inserted in the bus as well as the time spent processing the corresponding reply.
- T_{bus} includes the time taken to obtain the bus and to transmit the packets.
- T_{mem} is the time needed to get data from the structure which provides it (it can be another L1 cache, one of the L2 cache banks or main memory).

Furthermore, to identify more clearly where the hot spots of the memory hierarchy are, we present a taxonomy of the L1 cache misses found in a CMP

architecture (more generally, a taxonomy of the misses of the lower level of private caches) in terms of which memory structure provides the requested data. In this way, we can determine whether a structure is more critical than another and, even more, how future memory hierarchy optimizations will affect each memory component. The classification assumes a MOESI coherence protocol and identifies four categories:

1. Misses satisfied by another L1 cache (or $-to-$ misses): the line is in a single cache, or the line is in several caches and one of them has the line in *Owned* state.
2. Misses satisfied by the L2 cache (Hit L2 misses): no L1 cache can provide the line, and the L2 has a valid copy of it.
3. Misses satisfied by main memory (Mem misses): neither L1 caches nor the L2 cache have a copy of the line, so data must be obtained accessing main memory.
4. Invalidation or upgrade misses (Inv misses): the faulting cache has a valid copy of the line in *Shared* state but it tries to write the memory line, for which exclusive access is needed. It is necessary to place a *BusUpgrade* request in the bus to invalidate the rest of the copies of the line (if any) and gain exclusive access to it. For this kind of misses, the T_{mem} component of the latency is zero because no data is needed.

5 Experimental Results

In this section, we present a detailed performance evaluation of a CMP composed of 16 out-of-order processor cores similar to the MIPS R10000 processor with an optimized implementation of the sequential consistency memory model that includes load speculation and allows stores to graduate before completion. The architecture is similar to Piranha [1] but twice as many processors are simulated. In Table 1 we can see the configuration of the architecture we have evaluated. L1 cache sizes have been set commensurate to the total number of cores.

Table 1. Base architecture configuration

Parameter	Value
Number of cores	16
L1 size	8KB
L1 associativity	4-way
L1 latency	1 cycle tags + 1 cycle data
L2 size	2MB
Number of L2 banks	8
L2 associativity	8
L2 latency	2 cycles tags + 8 cycles data
Line size	32 bytes
Memory latency	120 cycles
Bus Arbitration	3 cycles
Bus cycle	3 cycles
Bus width	32 bytes

Table 2. Applications and input sizes used in this work

Application	Input size
BARNES-HUT	4096 bodies, 4 time steps
EM3D	38400 nodes, degree 2, 15% remote, 25 time steps
FFT	256K complex doubles
OCEAN	130x130 ocean
RADIX	1M keys, 1024 radix
UNSTRUCTURED	Mesh.2K, 5 time steps
WATER-NSQ	512 molecules, 4 time steps

Table 3. Classification of L1 accesses (on average)

Application	L1 hit	MSHR Coalesced	Bus Coalesced	$-to-$	L2 Hit	Mem.	Inv.
BARNES-HUT	75.6%	11.1%	0.25%	0.09% (0.67%)	13.17% (99.1%)	0.002% (0.02%)	0.03% (0.2%)
EM3D	74.6%	9.5%	0.001%	0.46% (1.77%)	10.05% (38.74%)	15.42% (59.47%)	0.004% (0.02%)
FFT	82.58%	11.16%	0%	0.0006% (0.01%)	2.95% (47.13%)	3.31% (52.85%)	0.0003% (0.005%)
OCEAN	75.87%	12.87%	0.06%	0.71% (6.38%)	8.5% (75.94%)	1.75% (15.59%)	0.24% (2.1%)
RADIX	89.02%	3.11%	0%	0.06% (0.71%)	5.74% (72.95%)	2.06% (26.2%)	0.01% (0.13%)
UNSTRUCT	81.42%	6.37%	0.05%	6.88% (56.62%)	4.34% (35.7%)	0.01% (0.12%)	0.91% (7.56%)
WATER-NSQ	86.56%	9.95%	0.002%	1.91% (54.66%)	1.57% (44.84%)	0.0001% (0.001%)	0.02% (0.5%)

Table 2 describes the benchmarks we have used in our experiments. This set of parallel scientific applications covers a variety of computation and sharing patterns. BARNES-HUT, FFT, OCEAN, RADIX and WATER-NSQ belong to the SPLASH-2 benchmark suite [18]. EM3D is a shared memory implementation of the Split-C benchmark [19]. UNSTRUCTURED is a computational fluid dynamics application [20]. The input sizes for the applications have been chosen taking into account the cache sizes and number of cores in the baseline architecture.

We can see, in Table 3, a classification of how the accesses to the L1 cache are solved. Most of these accesses are captured at the L1 cache, whose hit rates range from 74.6% to 89.02%. The number of occasions in which an access matches an outstanding request issued by the same processor (MSHR Coalesced) is greater than 10% for some applications (this is a consequence of using an optimized implementation of sequential consistency); however, the number of accesses matching an outstanding request issued by another processor (Bus Coalesced) is zero or near to zero for all the applications. The last four columns correspond to the taxonomy presented in Section 4. We show the percentage over the total number of accesses to the L1 caches and, in brackets, the percentage over the total number of misses.

For two applications (UNSTRUCTURED and WATER-NSQ) a significant number of misses are satisfied by another L1 cache ($-to-$ misses). For the remaining applications, most misses are satisfied by the L2 cache, except in

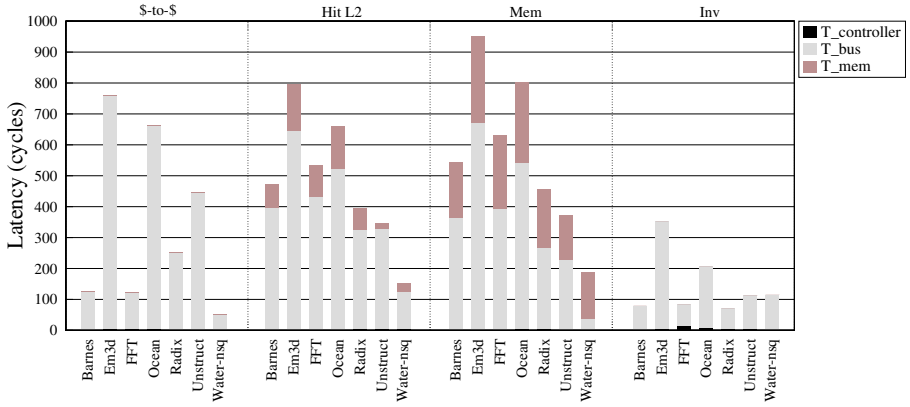

Fig. 2. Average Latency for $-to-$, Hit L2, Mem and Inv misses

the case of EM3D, for which 59.47% of the misses reach main memory. In all the applications (except in the case of UNSTRUCTURED), the number of Inv misses represents a small fraction of the total.

Once we have seen this classification of the L1 accesses, we are going to analyze the latencies suffered by each type of L1 miss. In Figure 2 we can see the average latency for $-to-$, Hit L2, Mem and Inv misses divided into the components described in Section 4 for all the applications. The main component of the overall L1 miss latency for all the miss types is the time spent at the bus. Logically, the miss type with lower latency is the Inv miss, as there is not a *memory* component and we do not have to wait for a reply. We also see that the time spent at the controller is negligible when compared with the bus latency.

We see that latencies are very variable, even for the same application if we compare different miss types. To better understand this erratic behavior, we must analyze the statistics in Table 4 (as well as those presented in Table 3). Table 4 shows the total number of requests and replies snooped by cache controllers (columns 1 and 2). Columns 3 and 4 contains the average number and percentage

Table 4. Classification of the snooped requests and replies

Application	Requests snooped	Replies snooped	Useful requests	Useful replies	L2 useful requests
BARNES-HUT	3,758,643	3,749,512	198,456 (5.27%)	251,489 (6.7%)	97,541 (2.59%)
EM3D	9,475,489	9,424,412	10,052 (0.11%)	587,125 (6.23%)	1,145,478 (12.15%)
FFT	4,912,331	4,912,085	45 (0.0009%)	307,022 (6,25%)	613,951 (12,5%)
OCEAN	4,119,636	4,033,051	19,259 (0.47%)	258,823 (6.42%)	471,302 (11.44%)
RADIX	3,292,750	3,288,383	1,495 (0.05%)	205,806 (6.26%)	408,114 (12.39%)
UNSTRUCT	20,726,502	19,168,246	831,566 (4.01%)	1,301,762 (6.79%)	928,474 (4.48%)
WATER-NSQ	2,837,894	2,823,757	97,868 (3.45%)	177,463 (6.28%)	159,081 (5.61%)

of useful requests and replies snooped by each L1 controller. By useful request we mean a request that implies some action over the local copy of the line at the L1 cache, and by useful reply we mean a reply that contains data and the cache is waiting for. Finally, column 5 shows the number of useful requests snooped by each L2 cache bank (the L2 cache does not snoop replies). The main result is the small number of useful requests (very insignificant in most applications). This implies that in the vast majority of occasions, it is not necessary that the L1 cache controllers snoop the requests that appear in the bus, as these requests are referred to a line that it is not in the cache. We can also see, comparing columns 1 and 2, how the number of snooped requests does not match exactly the number of snooped replies. This is caused by the Inv misses, which do not need a reply.

We have seen that one of the main hot spots in this architecture is the bus, so in order to evaluate the potential of future techniques aimed at alleviating this bottleneck, we simulate the same architecture but with an ideal network, in which the arbitration delay is zero cycles and the bus cycle duration is equal to the processor cycle. This network is equivalent to have one bus working at the same frequency than the processor for each private L1 cache.

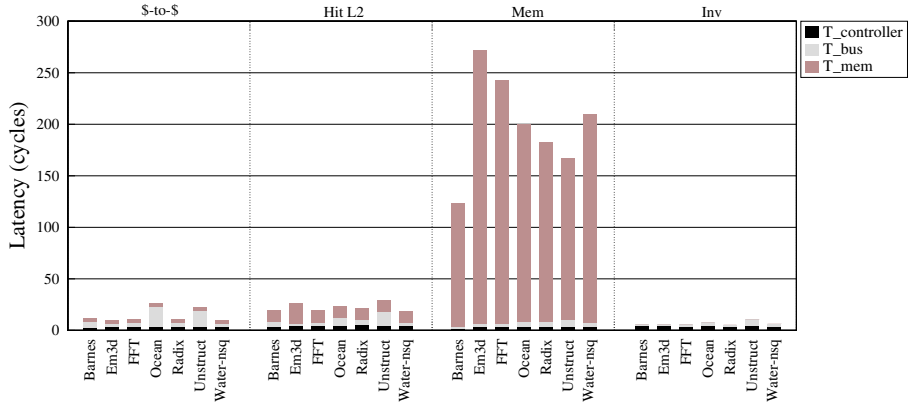

Fig. 3. Average Latency for $-to-$, Hit L2, Mem and Inv misses with an ideal network

When we use an ideal interconnection network, bus latencies are drastically reduced (see Figure 3). The overall latency for $-to-$, Hit L2 and Inv misses is reduced by a factor of 10 on average. Reductions for Mem misses are less impressive, as the *memory* component is hard limited by main memory latency. We can also see that the T_{mem} component of the latency is approximately 30% greater for Mem misses. As we have removed the bus bottleneck, requests arrive at a higher rate to memory controllers and, thus, these controllers are more loaded than in the baseline architecture. These results demonstrate the potential savings of future proposals using a high-performance point-to-point interconnection network providing more bandwidth than a bus, although different cache coherence protocols are to be implemented.

6 Conclusions

In this paper we have presented a L1 cache miss taxonomy of a snoop-based D-CMP (dense chip-multiprocessor) based on how the misses are satisfied. Our results point out that the main bottleneck of this kind of architecture is the shared bus, concluding that this type of interconnection does not provide enough bandwidth for a CMP composed of 16 cores. We have shown that a snoop-based coherence protocol induces too much unnecessary work at the cache controllers, as the majority of snooped transactions are not concerned with the lines in that cache. By simulating a perfect interconnection network, we have seen that it is possible to reduce the latencies of L1 misses by a factor up to 20 in some cases. This gives us a theoretical limit for future proposals in which the bus is replaced by a higher-performance interconnection network.

As future work, we are interested in modeling some different point-to-point interconnection networks and evaluate their performance. This may require some modifications in the coherence mechanism, so it will be necessary to design a cache-coherence protocol tailored to the particularities of a CMP architecture.

Acknowledgments

This work has been supported by the Spanish Ministry of Ciencia y Tecnología and the European Union (Feder Funds) under grant TIC2003-08154-C06-03.

References

1. Barroso, L.A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., Verghese, B.: "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing". In: Proc. of 27th Int'l Symp. on Computer Architecture. (2000) 282–293
2. Hammond, L., Hubbert, B.A., Siu, M., Prabhu, M.K., Chen, M., Olukotun, K.: "The Stanford Hydra CMP". IEEE Micro **20** (2000) 71–84
3. Kalla, R., Sinharoy, B., Tendler, J.M.: IBM Power5 Chip: A Dual-Core Multithreaded Processor. IEEE Micro **24** (2004) 40–47
4. Krewell, K.: UltraSPARC IV Mirrors Predecessor. Micro. Report, pp. 1-3 (2003)
5. Hughes, C.J., Pai, V.S.P., Ranganathan, P., Adve, S.V.: RSIM: Simulating Shared-Memory Multiprocessors with ILP Proccesors. IEEE Computer **35** (2002) 68–76
6. Moshovos, A., Memik, G., Falsafi, B., Choudhary, A.: "JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers". In: Proc. of 7th Int'l Symp. on High-Performance Computer Architecture. (2001) 85–96
7. Ekman, M., Dahlgren, F., Stenström, P.: "Evaluation of Snoop-Energy Reduction Techniques for Chip-Multiprocessors". In: Proc. of 1st Workshop on Duplicating, Deconstructing and Debunking. (2002) 2–11
8. Beckmann, B., Wood, D.: "Managing Wire Delay in Large Chip-Multiprocessor Caches". In: Proc. of 37th Int'l Symp. on Microarchitecture. (2004) 319–330
9. Liu, C., Sivasubramaniam, A., Kandemir, M.: Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs. In: Proc. of 10th Int'l Symp. on High Performance Computer Architecture. (2004) 176–185

10. Takahasi, M., Takano, H., Kaneko, E., Suzuki, S.: "A Shared-bus Control Mechanism and a Cache Coherence Protocol for a High-performance On-chip Multiprocessor". In: Proc. of 2nd Int'l Conference on High-Performance Computer Architecture. (1996) 314–322
11. Hammond, L., Willey, M., Olukotun, K.: "Data Speculation Support for a Chip Multiprocessor". In: Proc. of the 8th Int'l Symp. on Architectural Support for Parallel Languages and Operating Systems. (1998) 58–69
12. Krishnan, V., Torrellas, J.: "A Chip-Multiprocessor Architecture with Speculative Multithreading". IEEE Transactions On Computers **48** (1999) 866–880
13. Steffan, J.G., Colohan, C.B., Zhai, A., Mowry, T.C.: "A Scalable Approach to Thread-Level Speculation". In: Proc. of 27th Int'l Symp. on Computer Architecture. (2000) 1–12
14. Yanagawa, Y., Hung, L.D., Iwama, C., Barli, N.D., Sakai, S., Tanaka, H.: "Complexity Analysis of A Cache Controller for Speculative Multithreading Chip Multiprocessors". In: Proc. of 10th Int'l Conference on High Performance Computing. (2003) 393–404
15. Culler, D.E., Singh, J.P., Gupta, A.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, Inc. (1999)
16. Sweazey, P., Smith, A.J.: A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus. In: Proc. of 13th Int'l Symp. on Computer Architecture. (1986) 414–423
17. Charlesworth, A.: "The Sun Fireplane Interconnect". In: Proc. of SC2001 High Performance Networking and Computing Conference. (2001) 1–14
18. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: "The SPLASH-2 programs: Characterization and Methodological Considerations". In: Proc. of 22nd Int'l Symp. on Computer Architecture. (1995) 24–36
19. Culler, D.E., Dusseau, A., Goldstein, S.C., Krishnamurthy, A., Lumetta, S., Luna, S., von Eicken, T., Yelick, K.: "Parallel Programming in Split-C". In: Proc. of Int'l SC1993 High Performance Networking and Computing Conference. (1993) 262–273
20. Mukherjee, S.S., Sharma, S.D., Hill, M.D., Larus, J.R., Rogers, A., Saltz, L.: "Efficient Support for Irregular Applications on Distributed-Memory Machines". In: Proc. of 5th Int'l Symp. on Principles and Practice of Parallel Programing. (1995) 68–79