Exploiting Parallelism in a Network of Workstations Using COMA-BC

Benjamín Sahelices Fernández, Diego R. Llanos Ferraris y Agustín de Dios Hernández
{benja, diego, agustin}@infor.uva.es

Departamento de Informática. Universidad de Valladolid.
Valladolid. España.

Abstract

In this paper we put forward a design for a multicomputer system based on a network of workstations which we call COMA-BC. It has a common address space in which a shared variables programming model can be used. The management of the shared address space is performed in a similar way to that in existing multiprocessor COMA systems. To be exact, the shared address space is divided into blocks, and their copies reside in the attraction memories of the workstations.

The key piece in this system is the coherence cache protocol that we have developed. The goal of the protocol is to minimize the number and size of the messages travelling through the network so that the parallel applications can be executed without creating inconsistencies in the different copies of the blocks residing in the different nodes of the system.

The proposed system has not been built, but a simulation environment has been specifically developed. This environment allows the simulation of the execution of parallel standard applications in COMA-BC. This simulation environment is driven by execution. Using the results and a simple analytic model, results have been obtained concerning the performance of the execution of standard parallel applications in terms of acceleration and efficiency. These results show the viability of a COMA-BC system as a way of exploiting parallelism at a low cost using workstations.

1 Introduction

COMA-BC [20] is a multicomputer system based on a set of workstations connected within a standard shared medium network. This network has only one physical line which is used to send and receive all the information exchanged in the system. COMA-BC is a shared memory system with physically distributed memory modules. The different processors use the same address space and a fraction of this space is common for all of them [16]. Each workstation in COMA-BC is standard UNIX-type.

The idea of a multicomputer system based on a network of workstations is not original [17, 2], but COMA-BC has new features consisting of the management of the shared memory using the same techniques used in COMA multicomputers [11].

The design target of COMA-BC is to minimize the communication time in parallel workloads. The use of a standard local area network supposes that the communication time depends on two factors: the number of interchanged messages and their size. In the COMA-BC design the main concepts underlying COMA multiprocessors have been used to enable communication with a small number of small messages.

The shared memory space in COMA-BC is divided into blocks. Each workstation (system node) can have access to a copy of each block. These copies are sent through the LAN to the nodes that need the information. The blocks have no home nodes, that is, there is no node where the information of each block resides permanently. Each node has a copy of the block, but it is just a copy. Thus, the local memory of each node can be considered a cache of the shared address space; this local memory is called attraction memory (AM), using the usual terminology of the multiprocessor COMA system . The memory blocks stored in the attraction memory are cache copies of the blocks of the shared address space. That is why we call them "cache blocks". In the same way as in a COMA multiprocessor, the only physical location of the shared address space is the set of attraction memories, that is, the set of cache memories of the system.

When one processor tries to use a specific location of the shared space there are two possibilities: either a local copy is stored in the local attraction memory or there is no local copy in the attraction memory. If a copy is not available then a cache miss happens. That miss is always related to a read or write memory operation.

One cache block can have multiple copies in different nodes; that is why one of the key elements in the development of COMA-BC is its coherence protocol. It is responsible for maintaining the coherence of the information stored in the different cache copies of each block of the shared space. This coherence protocol is a cache coherence protocol. The COMA-BC protocol developed is invalidation-based and uses a hybrid snoopy and directory scheme. The reason for using this technique is to best exploit the characteristics of the shared medium local area network in two ways: 1) each node can *see* every block movement through the network, 2) each node can send a message that is received simultaneously by all the nodes. The coherence directories are used to eliminate race conditions. A detailed description of the directories system is given later in this paper.

A COMA-BC system can be built in two different ways: 1) with standard workstations, in which the management of access to the shared address space is carried out by software running in the local workstation, 2) with modified workstations using a specifically designed memory controller that manages every access to the shared space and every action related to the coherence protocol.

In order to check the correctness of COMA-BC, simulation and verification studies have to be done on three levels: 1) Verification of the coherence protocol for systems with 2 and 3 nodes. 2) Simulation of the system with coloured Petri Nets and with other models based on finite state machines, up to 100 nodes. 3) Simulation based on a synthetic workload using Ptolemy.

A multiprocessor simulation system has been specifically developed for measuring the capacity of a COMA-BC system. This simulation system is program driven; a significant number of standard parallel applications have been executed on it. The applications belong to the SPLASH-2 parallel benchmark. The real execution of such applications has permitted a series of interesting indexes to be obtained: such as, the number of accesses to the shared address space, the number of access misses, the number of network messages needed and the size of the named messages. At the same time a simple analytic model has been developed to provide temporal indexes (speedups) using the results from the simulation system. The analytic model uses the results of the multiprocessor simulation as inputs together with the data relative to the processing speeds of the nodes and the description of the physical characteristics of the local area network used to build COMA-BC.

The results obtained show that COMA-BC can be used as a platform for exploiting parallelism with a standard LAN interconnecting workstation.

This article is organized as follows: firstly a detailed description of COMA-BC is given. Secondly, due to its importance, the functioning of the COMA-BC coherence protocol is specified. Thirdly, the simulation and verification procedure of the correctness of COMA-BC is explained. Fourthly, the multiprocessor simulation environment is described, giving special emphasis to the performance indexes obtained from the COMA-BC simulation. After that the analytic model is studied which allows us to obtain performance results in terms of acceleration for the various standard parallel applications executed in the simulator. Finally, the experimental results obtained are explained and discussed and the main conclusions of this work are summarized.

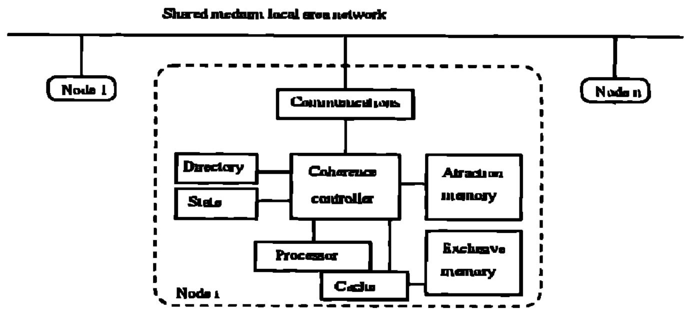

Figure 1: COMA-BC Architecture

2 Description of a COMA-BC System

The architecture of a COMA-BC system appears in figure 1. COMA-BC systems are made up of a set of nodes connected by a common medium interconnection network. The nodes are workstations. The network is a standard LAN. The coherence messages are sent and received using the network. Each node has a processor that uses the information contained in its local memory. The local memory of each node is divided in two parts: exclusive and attraction memory. Exclusive memory contains the text of the local processor programs and the data that is used exclusively by the local processor. Attraction memory plays the role of a cache memory in a shared address space; the data needed for several nodes is stored in this address space. The exclusive memory works in the same way as the main memory of an ordinary computer; that is the reason why we focus on the description of the attraction memory.

As mentioned above, attraction memories are the physical support, and the only physical support, of the shared address space of the different nodes. In addition, each attraction memory works as a cache of the shared space. Each attraction memory works as a cache of the said space, but there is no other memory component in the system, apart from the attraction memories, that act as a physical support. Accordingly, that shared address space must be managed using the same rules as those used in COMA multiprocessors. In this kind of system, information does not reside permanently in the same host node, but the information moves through the system, residing in the nodes that most frequently access that information. All the nodes have the same characteristics for accessing and managing the information.

The shared address space is divided into cache blocks. Each attraction memory is divided into a set of frames, each of which can store one cache block. The relationship between blocks and frames is established by a direct mapping. In COMA-BC each attraction memory can store one copy of the whole shared address space. This is due to an initial design decision that tends to minimize the number of necessary operations in the network thus making a replacement mechanism unnecessary. The disadvantage of this decision is that the attraction memories do not make up pieces of memory that can be joined together to form a larger common space.

3 COMA-BC Protocol

The COMA-BC cache coherence protocol has been especially designed to be able to manage the cache coherence in the COMA-BC system while taking into account an important design restriction: the interconnection support is a standard shared medium local area network with one physical line interconnecting computers. The best example of this kind of network is an Ethernet network.

The COMA-BC protocol is built on the basis of assigning a state to each copy of a cache block (from now on simply a "copy").This state is used to indicate that the information contained in the copy is valid, and can be read directly by the processor, or that the information contained in the copy is not valid because a copy of it has been written on another node by the corresponding processor. If the information is not valid and the processor tries to read it, then the coherence controller detects a cache miss and it should request a valid copy. State information is also used by the coherence controller to indicate whether a processor can write on a copy without coherence violation. COMA-BC protocol uses five states for each copy. These states are: 1) Invalid, 2) Clean, 3) Dirty, 4) Invalid awaiting RRI (briefly IARRI), and 5) Invalid awaiting RRB (briefly IARRB).

The COMA-BC protocol is invalidation based. This means that when a node writes a copy, its coherence controller sends a message through the network in such a way that every other node in the system receives the message and invalidates its own copy. This behavior is specially well adapted to shared medium networks that are able to send broadcast information with just one message. In addition, every coherence controller receives all messages sent through the network by every node; this means that it receives all of the coherence information for every copy of a cache block and uses it to maintain state and directory information updated. This is usual in snoopy bus coherence protocols.

Besides the state information of each copy, the COMA-BC protocol uses the concept of the owner node of a block. The only processor that can write over a copy is the owner node processor. Each coherence controller manages a directory containing the owner nodes of every block of the shared space. The ownership concept is dynamic, changing in the execution of the programs to those nodes that need to write on each block. When a non owner node needs to write over its copy, it must first ask the owner in order to get the ownership of the block; to do this it uses directory information to locate the owner. When a node gains ownership, it can write over its copy. Accordingly, the directory managed by each coherence controller has as many entries as blocks in the shared space and each entry stores just one number from 1 to n, where n is the number of nodes in the system. To this end, each node in a COMA-BC system has one assigned number that does not change and is used to identify each node in the coherence directories. A node will consider itself to be the owner of a block when its own node number appears in the corresponding directory information. The protocol must guarantee that just one node is the owner of each block.

4 Validation of the COMA-BC Protocol

The study to validate the COMA-BC protocol has to be carried out on three levels: 1) Different simulations have been done using Petri Nets and other models based on finite state machines. 2) The verification of the protocol for systems with 2 and 3 nodes has been done. 3) The simulation of the global operation of COMA-BC has been performed. We shall now comment on each of these three levels.

1) Simulations of the protocol using Petri Nets. Firstly, the protocol has been formalized using Colored Petri Nets [13]. Using this formalization the initial design errors were easily detected and corrected. The development of this formalization was done using the DesignCPN [10] software.

2) Verification of the protocol. The verification of the protocol has been carried out using the method of expansion of states applied to a COMA-BC model based on a finite states machine. The said study has been carried out using the software tool Murφ [7] version 3.0. Due to the problem of the explosion of states inherent to this type of verification, the simulation could only be performed for two and three nodes. For these, as the verification is an exhaustive technique, the appearance of protocol errors has been completely ruled out. For systems larger than three nodes, the same software tool has been used with the same finite states machine

model in order to obtain simulation results with up to 100 nodes. There were no detected errors in the normal working of the protocol.

3) Global simulation of the system. Finally, a global simulation of the system has been carried out using the simulation tool Ptolemy. In this third phase of the validation, the simulation was driven by synthetic workloads. Basically, using the simulation environment, the different elements of the COMA-BC system have been reproduced, including the processor, the coherence controller, the network interface and the intercommunication network. Basing ourselves on the experiments carried out, we have been able to rule out the existence of working errors in the system, considering it globally, where these errors are due to design faults in the COMA-BC protocol.

5 The COMA-BC Multiprocessor Simulation Environment

The prototype of a COMA-BC system has not been built. Thus, to get results concerning the performance of the proposed system it has been necessary to build a multiprocessor simulation environment which, together with an analytical model, allows the simulation of standard parallel applications and, finally, to obtain performance indexes of the parallel execution in terms of efficiency and speedup. In this section, we shall explain the multiprocessor simulation environment and in the following section we shall comment on the characteristics of the analytic model used.

The developed simulation environment is based on carrying out a working simulation of COMA-BC driven by the execution of standard parallel applications [6, 15, 3]. The said simulation is performed using standard workstations connected via a 10 Mbps Ethernet network. Each workstation reproduces one COMA-BC node executing its part of the parallel application and generating all the memory references to the shared address space that a real COMA-BC system would generate. Each workstation executes two processes which implement the simulation and which we shall call the functional emulator and the architecture emulator, representing the application and the coherence controller respectively.

The architecture emulator is a process that reproduces the functions of a COMA-BC coherence controller. It receives requests for access to the shared address space, manages access misses and executes all the protocol actions needed to maintain the system coherence.

The functional emulator is a process that simulates the execution of the part of the parallel application that is executed in a particular COMA-BC node. When this functional emulator is executed, in reality, what is executed in the workstation processor is the part corresponding to the parallel application, as the functional emulator builds itself by instrumenting every access to the shared address space [8]. To be exact, this instrumentation consists in detecting all accesses to the memory address that are inside the shared space and to replace them with a function call that sends a message to the architecture emulator. Once the functional emulator has sent the request, it waits until it is resolved. This instrumentation is implemented by expanding the source code of the parallel application. To do this the tool Pegaxo [12] is used. This expanded source code can be assembled and linked to the destination workstation and the resulting executable code is what makes up, in fact, the functional emulator.

The described simulation environment allows the execution of parallel applications developed in C with the programming style based on the use of the Argonne [18] PARMACS macros. Pegaxo expands the PARMACS macros and replaces each memory access by an invocation to a certain procedure. Pegaxo has been developed for HPPA and SunSPARC architectures. It is important for this expansion process that the processors used as the base for the simulation are RISC architectures because it reduces the complexity of analyzing the source code as the set of memory access instructions is very simple.

To manage the various parallel processes involved in the simulations (that is: pairs of functional-architecture emulators in the different nodes) and to implement the communications between them, PVM has been used. PVM allows the sending and receiving of protocol events between the different architecture emulators (there is one architecture emulator on each workstation) and the sending and receiving operations between the functional and the architecture emulators on each workstation.

The main drawback of PVM in this simulation is that the messages it uses to access the network (that is, those that reproduce the protocol events between nodes) cannot be broadcast messages, they must have one particular node destination. Then, it is impossible to send events with just one access to the shared medium of the network. To avoid this problem, each architecture emulator simulates the sending of a PVM message with the protocol event to each one of the remaining architecture emulators and waits for the confirmation from each one of them.

As explained before, the COMA-BC multiprocessor simulation allows us to obtain indexes of how parallel applications really behave in a COMA-BC system. These indexes are obtained from a series of statistical variables updated during the execution of the simulation; some of them are updated by the functional emulator and others by the architecture emulator. The information obtained from these statistical variables is finally reduced to a set of indexes as follows: 1) i: total number of executed instructions. 2) m: total number of read and write executed instruction. 3) s: total number of read and write instructions sent to the shared address space. 4) e: total number of events or messages interchanged using the network to manage access misses. 5) l_m: average size of messages interchanged in the network, expressed in bytes per message. These indexes are obtained for each one of the workstations in the parallel simulation environment, and thus refer to how each parallel branch of the application has been executed.

6 Analytical Model

A simple analytic model has been developed to obtain performance results of the parallel applications executing in a COMA-BC system. This model is based on the consideration that all the workstations in COMA-BC are physically identical. The said analytical model has both input and output data. The input data of the model can be divided into two subsets: 1) The results obtained in the multiprocessor simulation through the colection of statistical variables explained in section 5. 2) The parameters describing the physical characteristics of a real COMA-BC system. Inside the second group there are two more subsets: i) Parameters describing the physical characteristics of the processor, including the duration of the processor cycle time (τ) and the number of cycles needed to execute each instruction. ii) Parameters describing the behavior of the interconnection network of the workstations; which have been chosen representing the behaviour of the interconnecting system by means of the model LogP [4].

Using the analytic model, execution time of a parallel application in a COMA-BC system can be obtained. This index represents the execution time of the same application in a real COMA-BC system with the same number of processors as used in the simulation. Using this index, significant performance indexes, such as speedup and efficiency for each parallel workload for the different processors, can be obtained.

The total execution time of the parallel application is calculated from the beginning of the execution of the code, supposing that this is simultaneous for all the processors, up to the moment the parallel code ends. As the different workstations can execute parts of the workload of different sizes, the execution time T(n), in a set of n workstations, is defined as the biggest of all the execution times calculated for all the workstations. Note that in the simulation phase, the output data described in section 5 refers to each workstation involved. From this point, these results refer to the workstation which has the biggest workload.

The analytic model obtains the total execution time in a set of n workstations as the sum of two terms:

$$T(n) = t_c + t_r$$

where t_c represents the time spent to execute that part of the workload that does not refer to the communication through the network. However, t_r represents the time taken in communications through the network that are necessary to execute the workload.

To calculate t_c, the time spent in the execution of each instruction is considered as being divided into three parts: 1) a fixed time for each instruction which we call t_i; 2) an extra fixed time t_m, if the instruction is a read or write to memory instruction; 3) another extra fixed time t_s, if the read or write to memory falls inside the shared address space. Then:

$$t_c = it_i + mt_m + st_s = (ic_i + mc_m + sc_s)\tau$$

These three terms c_i, c_m and c_s represent the times t_i, t_m and t_s expressed in number of clock cycles of the processor. They constitute the three input parameters of the model and they describe the number of cycles necessary to execute each instruction. They must be adjusted (the same as τ) as a function of the physical architecture of the workstations used.

To calculate t_r that same time is considered to include all the operations necessary to execute, using the network, all the read and write instructions carried through the shared address space and that have generated cache misses. In other words, the time t_r refers to all the memory references that need the network to be completed. t_r can be expressed simply as

$$t_r = ew$$

, where w represents the average time needed to send an event or message through the network. In order to calculate w we use a representation of the network based on the LogP model. To be exact, the frequency of the messages is supposed to be low enough as to ignore the time term between two consecutive messages ("gap") [14]. Then w can be expressed as:

$$w = o_s(l_m) + o_r(l_m) + L(l_m) = a + bl_m$$

where $o_s(l)$, $o_r(l)$ and $L(l)$, are respectively, the fixed cost of sending a message, the fixed cost of receiving a message and the latency of the network for messages of l byte. The terms a and b are the input parameters of the model that describe the behavior of the interconnection network. The term l_m represents the average size of the messages interchanged, as mentioned above.

7 Experimental Results

The execution of six applications of Splash-2 [21] has been simulated. Specifically LU, Ocean, FFT, Radix, Barnes-Hut and Radiosity. The workload for each application is the standard which is described in [21].

The execution of the named applications has been done in a simulated COMA-BC system with 2, 4, 8 and 16 workstations. To execute the parallel simulation, the same number of workstations as simulated nodes has been used, all of them connected with a 10 Mbps Ethernet network. A functional emulator and an architecture emulator is executed on each workstation. The results obtained are detailed in the table 1. These results correspond to the workstation that had to execute the biggest workload in the simulation; that is, the node that limits the total execution time of the application. All the simulation experiments were carried out using a block size of 256 bytes. The reason is that this size allows us to obtain a compromise between spatial locality and the access miss rate [20].

With these simulation results and using the analytic model proposed in the previous section, an estimation of the execution time, speedup and efficiency can be obtained for a specific COMA-BC system. The COMA-BC system proposed is based on workstations with a cycle time of τ 7 ns, that is, a clock frequency of 167 MHz. In accordance with the data obtained from standard architectures, the values for the rest of the parameters are: $c_i = 1$ cycle, $c_m = 2$ cycles and $c_s = 10$ cycles. There are five possibilities for the interconnection networks: a) Ethernet network at 10 Mbps with the stack of TCP/IP protocols, b) Ethernet network at 10 Mbps without TCP/IP and with active messages, c) Fast Ethernet at 100 Mbps, d) FDDI network with active message layer [19] and e) Myrinet network [1] (ATM with fast messages). Each of these methods of interconnection can be assigned input parameters, which we call a and b, for the analytic model. These two parameters are obtained from the bibliography [14, 19, 9] and are detailed in the table 2. Finally, feeding the results from the simulation and the parameters that describe the specific COMA-BC system into the analytic model, speedup and efficiency data is obtained (table 3).

From the results obtained, it can be concluded that a COMA-BC system with a fixed low cost of sending and receiving messages to the network and low latency in transmissions to the network is viable for obtaining speedup in the execution of parallel applications over a network of workstations. Unfortunately, there are no other performance results referring to similar systems, that is, systems based on a network of workstations with a programming model of shared variables. There are results, however, of speedup for multiprocessor systems with similar memory management, such as SGI-Challenge and Origin2000 [5]. Comparing with these systems, the speedups obtained in the best of the cases considered for COMA-BC (a Myrinet network) are about half that of the speedups of the aforesaid systems. This is not much in absolute terms but it can be considered a good result, if we take into account that they have been obtained using a system costing much less than half that of the former and using standard resources (workstation, interconnection network).

It is clear that the construction of a COMA-BC system based on an interconnection network with high fixed costs of sending-receiving messages and low bandwidth is not viable; such systems could be an Ethernet at 10 Mbps or Fast-Ethernet at 100 Mbps with TCP/IP. If the fixed cost of sending a message is reduced, the results improve considerably, as can be observed when, in the Ethernet 10 Mbps network, the TCP/IP stack is substituted for active messages.

8 Conclusions

We have shown that COMA-BC is an example of how a distributed shared memory system over a network of workstations can be implemented. We have also established the usefulness of applying the concepts of COMA multiprocessors to the design of COMA-BC. The idea of having only "cache copies", that is, copies of the cache blocks that migrate to those nodes that need them at each instant has been seen to be useful for carrying out this design. Moreover, having separated the concept of cache block of the shared space with respect to the pages of virtual memory, the size of the blocks used is small, thus less time is needed to travel through the interconnecting network.

The simulation study shows the viability of the proposed system for exploiting parallelism in a network of workstations. In the first version, it is not possible to use a standard network in COMA-BC because of the loss of performance. To be exact, we have shown that it is not possible to use a standard Ethernet network with the TCP/IP protocol stack. It is necessary to use lower latency networks and protocol stacks with lower costs of sending and receiving messages, such as the protocol stack of active messages.

Finally, it is necessary to emphasize that the key concept in COMA-BC is the cache coherence protocol, because it manages the coherence of the block copies in the different workstations without an excessive increase of traffic in the network. This is a difficult task because the proposed system is not hierarchical and because a shared medium interconnection network is used.

Aplication	N.proc.	i	m	s	e	l_m
LU	2	111.120.576	9.532.517	9.455.220	4.740	93,19
	4	61.426.112	5.319.629	5.276.108	5.877	96,57
	8	35.691.344	3.126.284	3.097.428	7.850	99,87
	16	22.216.460	2.087.982	1.991.933	12.244	104,19
Ocean	2	717.016.960	85.066.107	76.334.645	86.482	91,60
	4	366.595.168	43.372.635	38.971.205	180.019	93,35
	8	190.834.640	22.602.074	20.283.136	168.036	99,71
FFT	2	117.054.488	7.720.275	6.635.076	32.352	87,50
	4	58.921.556	3.899.197	3.354.554	34.887	93,61
	8	29.907.108	1.995.776	1.720.205	34.776	98,39
	16	15.638.624	1.100.263	942.759	37.334	103,26
Radix	2	334.413.600	35.459.178	11.317.628	82.748	96,73
	4	168.359.968	17.862.344	5.775.935	58.873	111,23
	8	85.498.096	9.087.985	3.023.299	52.396	112,29
	16	48.170.856	5.209.282	2.080.522	55.677	128,68
Barnes-Hut	2	901.257.408	139.745.300	82.960.168	100.758	89,70
	4	450.539.424	69.787.588	41.469.285	116.681	99.20
	8	245.949.632	37.058.161	22.926.158	127.303	112.48
Radiosity	2	508.791.008	51.124.661	18.256.955	306.004	91.83
	4	297.416.416	20.905.983	7.623.726	230.898	94.37
	8	150.688.000	11.969.352	4.515.872	188.231	98.94

Table 1: Experimental results obtained in the simulation of the execution of several Splash-2 applications

Network	a (μs)	b (μs)
Ethernet-10 Mbps TCP-IP	200	0.8
Ethernet-10 Mbps active messages	15	0.8
Ethernet-100 Mbps TCP-IP	200	0.08
FDDI active messages	15	0.08
Myrinet	10	0.026

Table 2: Input parameters for the analytic model to the characterization of the interconnection network

Aplication	Number proc.	Eth.10Mbps TCP		Eth.10Mbps act.msgs.		Ethernet 100 Mbps		FDDI		Myrinet	
		S(n)	E(n)	S(n)	E(n)	S(n)	E(n)	S(n)	E(n)	S(n)	E(n)
LU	2	0.99	0.49	1.42	0.71	1.13	0.56	1.72	0.86	1.78	0.89
	4	1.13	0.28	1.98	0.49	1.38	0.34	2.87	0.71	3.06	0.76
	8	1.04	0.13	2.19	0.27	1.35	0.16	4.16	0.52	4.76	0.59
	16	0.74	0.04	1.76	0.11	1.00	0.06	4.67	0.29	6.04	0.37
Ocean	2	0.64	0.32	1.15	0.57	0.77	0.38	1.69	0.84	1.82	0.91
	4	0.39	0.09	0.94	0.23	0.52	0.13	2.26	0.56	2.81	0.7
	8	0.44	0.05	1.12	0.14	0.6	0.07	3.25	0.4	4.42	0.55
FFT	2	0.26	0.13	0.62	0.31	0.33	0.16	1.28	0.64	1.52	0.76
	4	0.25	0.06	0.66	0.16	0.34	0.08	1.8	0.45	2.4	0.6
	8	0.26	0.03	0.69	0.08	0.35	0.04	2.31	0.28	3.43	0.42
	16	0.24	0.01	0.65	0.04	0.34	0.02	2.5	0.15	4.1	0.25
Radix	2	0.26	0.13	0.59	0.29	0.34	0.17	1.28	0.64	1.53	0.76
	4	0.36	0.09	0.83	0.20	0.5	0.12	2.17	0.54	2.76	0.69
	8	0.43	0.05	1.03	0.12	0.6	0.07	3.19	0.39	4.43	0.55
	16	0.39	0.02	0.92	0.05	0.58	0.03	3.53	0.22	5.5	0.34
Barnes-Hut	2	0.65	0.32	1.17	0.58	0.78	0.39	1.68	0.84	1.8	0.9
	4	0.68	0.17	1.45	0.36	0.88	0.22	2.81	0.7	3.21	0.8
	8	0.64	0.08	1.47	0.18	0.89	0.11	3.9	0.48	4.91	0.61
Radiosity	2	0.17	0.08	0.46	0.23	0.23	0.11	1.28	0.64	1.73	0.86
	4	0.23	0.05	0.63	0.15	0.31	0.07	1.95	0.48	2.8	0.7
	8	0.26	0.03	0.64	0.08	0.39	0.04	1.67	0.20	2.17	0.27

Table 3: Performance results expressed as speedups and efficiency for several Splash-2 applications in COMA-BC

References

[1] N. Boden et al. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, (2):29–36, 1995.

[2] E. Anderson, D.E. Culler, and D.A. Patterson. A case for NOW (networks of workstations). *IEEE Micro*, (2), 1995.

[3] S. Cho, J. Kong, and L. Gyungho. Coherence and replacement protocol of DICE - a bus-based COMA multiprocessor. Technical report, Dept. of Computer Science and Engineering. University of Minnesota, 1996.

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of 4th ACM Symposium on Principles and Practice of Parallel Programming*, 1993.

[5] D.E. Culler, J.P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan-Kaufmann Publishers, 1999.

[6] F. Dahlgren. A program driven simulation model of an MIMD multiprocessor. In *Proceedings of 24th Annual Simulation Symposium*, pages 40–49. IEEE Computer Society, 1991.

[7] D.L. Dill, A.J. Drexler, A.J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proceedings of IEEE Int'l Conference on Computer Design*, pages 522–525, 1992.

[8] S.J. Eggers. Techniques for efficient in-line tracing on a shared memory multiprocessor. In *Proceedings of the ACM SIGMetrics Int. Conf. on Measurement and Modelling of Computer Systems*, pages 37–47, 1990.

[9] L. Giannini and A. Chien. A software architecture for global address space communication on clusters: Put/get on fast messages. In *Proceedings of the 7th High Performance Distributed Computing (HPDC7) Conference*, 1998.

[10] CPN Group. Design/cpn online manuals (http://www.daimi.aau.dk/designcpn/). Technical report, University of Aarhus. Denmark., 1999.

[11] E. Hagersten, A. Landin, and S. Haridi. DDM: A cache-only memory architecture. *IEEE Computer*, pages 44–54, 1992.

[12] J.A. Illescas, B. Sahelices, and A. de Dios. Pegaxo: un entorno de simulacin de sistema multiprocesador de memoria compartida. In *Proceedings of VIII Jornadas de Paralelismo. Departamento de Informtica. Universidad de Extremadura.*, pages 131–140, 1997.

[13] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis, Methods and Practical Use. Volume 1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, 1992.

[14] K.K. Keeton, T.E. Anderson, and D.A. Patterson. LogP quantified: The case for low overhead local area networks. In *Proceedings of Hot Interconnects III: A Symposium on High Performance Interconnects*. Stanford University, 1995.

[15] A. Landin and F. Dahlgren. Bus-based COMA - reducing traffic in shared-bus multiprocessors. In *Proceedings of 2th IEEE Conference on High Performance Computer Architecture*, 1996.

[16] D. Lenoski and W.D. Weber. *Scalable Shared Memory Multiprocessing*. Morgan-Kaufmann Publishers, 1995.

[17] K. Li. *Shared Virtual Memory on Loosely-Coupled Processors*. PhD thesis, Yale University, New Haven, Connecticut, 1986.

[18] E. Lusk, R. Overbeek, J. Boyle, R. Butler, T. Disz, B. Glickfeld, J. Patterson, and R. Stevens. *Portable Programs for Parallel Processors*. Holt, Rinehart and Winston, Inc., 1987.

[19] R. Martin. HPAM: An active message layer for a network of HP workstations. In *Proceedings of Hot Interconnects II*, 1994.

[20] B. Sahelices. *COMA-BC: Una arquitectura de memoria sólo cache en bus común no jerárquica*. PhD thesis, Departamento de Informática. Universidad de Valladolid, 1998.

[21] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of 22th International Symposium on Computer Architecture (ISCA) Conference*, pages 24–36, 1995.