A Performance Evaluation of Cluster-based Architectures

Abstract

This paper investigates the performance of shared-memory cluster-based architectures where each cluster is a shared-bus multiprocessor augmented with a protocol processor maintaining cache coherence across clusters. For a given number of processors, sixteen in this study, we evaluate the performance of various cluster configurations. We also consider the impact of adding a remote shared cache in each cluster. We use Mean Value Analysis to estimate the cache miss latencies of various types and the overall execution time. The service demands of shared resources are characterized in detail by examining the sub-requests issued in resolving cache misses. In addition to the architectural system parameters and the service demands on resources, the analytical model needs parameters pertinent to applications. The latter, in particular cache miss profiles, are obtained by trace-driven simulation of three benchmarks.

Our results show that without remote caches the performance of cluster-based architectures is mixed. In some configurations, the negative effects of the longer latency of inter-cluster misses and of the contention on the protocol processor are too large to counter-balance the lower contention on the data buses. For two out of the three applications best results are obtained when the system has clusters of size 2 or 4. The cluster-based architectures with remote caches consistently outperform the single bus system for all 3 applications. We also exercise the model with parameters reflecting the current trend in technology making the processor relatively faster than the bus and memory. Under these new conditions, our results show a clear performance advantage for the cluster-based architectures, with or without remote caches, over single bus systems.

Keywords: shared-memory multiprocessors, clusters, protocol processors, mean value analysis.

1 Introduction

Single bus shared-memory multiprocessors, or multis [4], have enjoyed a tremendous success. However, the number of processors that can be incorporated in the system is limited since the single bus soon becomes an overcommitted resource. The lack of expansion is exacerbated by the fact that the rate of increase in processor speed grows faster than the corresponding increase in bus bandwidth and decrease in memory latency. For example, systems that were balanced with a dozen processors of the x86 vintage might become saturated if more than four processors of the Pentium Pro class are attached to the single bus. One way to expand the number of processors while keeping the shared-memory paradigm is to consider each multi as a *cluster* and to link clusters together using some interconnection network such as one (or two) mesh [12] or an SCI ring [14].

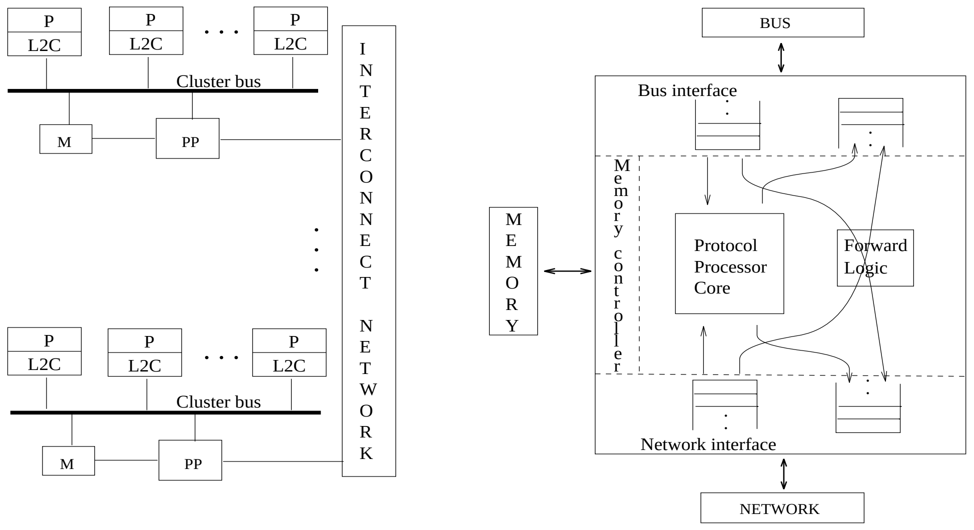
In cluster systems, memory requests are satisfied either locally, i.e., within a cluster (intra-cluster), or externally, i.e., by another cluster (inter-cluster). Intra-cluster cache coherence is enforced using a snoopy protocol while some form of directory-based coherence is used for inter-cluster transactions. Intra-cluster misses utilize the internal resources of the cluster, namely the common bus, the local memory, the snooping caches of the other processors in the cluster, and, on occasion, the protocol processor (see below). Inter-cluster misses utilize the network and additional logic for managing the directory information and for transmitting control information and data between clusters. The current trend[8, 14, 17] is to use programmable processors, hereafter called *protocol processors*, rather than hardwired controllers for the management of inter-cluster transactions. This choice is motivated by the ease in expansibility, the opportunity of tailoring coherence protocols to the needs of an application, and the possibility of introducing user-directed communication primitives. The drawback is that inter-cluster misses, that already suffer a delay an order of magnitude longer than that of intra-cluster misses because of the network latency, will be more costly than in the hardwired approach. A balanced system should then be such that the number of intra-cluster misses does not saturate the cluster buses and at the same time the number of inter-cluster misses should not be too large since it takes much longer to resolve them.

In this paper, we evaluate cluster-based architectures where each cluster is a shared-memory single bus multiprocessor to which is associated a protocol processor. We use Mean Value Analysis (MVA) to estimate the contention at major shared resources, the cache miss latencies, and the overall execution time. The input parameters of the analytical model include architectural parameters such as the cache and line sizes as well as the number of cycles needed for primitive, contention-free, operations, and application dependent parameters such as the cache miss profiles. These application dependent parameters are obtained via trace-driven simulation. We compare the performance of various cluster-based configurations with different application parameters, including systems that contain a remote cache for reducing inter-cluster references.

The rest of the paper is organized as follows. In section 2, we present the basic architectural model and its variations. In section 3, we introduce the analytical model showing how to get an estimate of the cache miss latencies and overall execution time. In section 4, we describe our choice of architectural parameters and how we obtained the application parameters needed for the evaluation. In Section 5, we show the quantitative results obtained by exercising the model and we highlight where performance bottlenecks might arise and what configurations are best for various parameters. Section 6 presents related work. Finally, we summarize our results in Section 7.

2 Architectural models

2.1 Base architecture and alternatives



(a): The base architecture (b): The protocol processor

Figure 1: The architectural models

The base architecture, shown in Figure 1a, consists of *clusters* connected to each other by an interconnection network such as a mesh. Each cluster is a single bus shared-memory multiprocessor augmented with a protocol processor. A private cache (or cache hierarchy) is associated with each processor in a cluster. The caches are kept coherent via a snoopy protocol. Memory modules can be accessed through the bus and through the protocol processor. Thus, *local* memory accesses, such as cache misses for private data, can be directly placed on the bus instead of being routed through the protocol processor. In addition to data, the memory also contains the coherence directory for the (shared) memory associated with the cluster. From the directory's viewpoint, nodes are clusters not individual processors. Data that is not local is called *remote*.

The protocol processor in the base architecture, Figure 1b, contains a protocol processing core for running protocol handlers (we assume a full directory coherence protocol), a network interface for sending and receiving inter-cluster messages, a bus interface for communicating with the intra-cluster, or local, processors, and a memory interface for accessing memory. In order to decrease the load on the protocol processing core and to reduce inter-cluster latencies, we include a forwarding module between the bus and network interfaces for those messages where it is easy to decide that they do not need access to local memory and/or directory. Examples of such messages would be a read request whose home node is another cluster and the corresponding "write-back" from the home cluster.

Because there is a large discrepancy between the latencies of intra-cluster and inter-cluster misses, the performance of cluster-based architectures is sensitive to the proportion of the two kinds of misses. Intuitively, this proportion is closely related to the number of processors in the cluster. If we increase this number, the ratio of intra-cluster misses to inter-cluster misses will increase. At one extreme, as all processors reside in a single cluster, all cache misses become intra-cluster. At the same time, contention on the bus will degrade performance. On the other hand, if the number of processors per cluster is too small, then we might have too many inter-cluster misses with long latency. At the other extreme, each cluster has only one processor and all cache misses not serviced by the local memory are inter-cluster misses. One purpose of this study is to evaluate cluster-based architectures when we vary the number of processors per cluster; we will monitor how the cache miss distribution reacts to this parameter and what impact it has on overall performance.

One variant to the base architecture that we will also study is clusters with large remote caches like those found in two recent systems [14, 17]. The remote caches are shared by all processors in the cluster and contain only remote data. Including a remote cache in a cluster will reduce the number of inter-cluster misses; we will assess the benefits of introducing this new resource in the system.

2.2 Classification of misses

Cache coherence in the hierarchical architecture is maintained by using a combination of snoopy and directory-based cache coherence mechanisms. Detailed protocol operations are similar to those found in [12]. For our modeling purposes, we need to classify the cache misses according to their service demand on the shared resources. In Table 1 we summarize the actions needed for each *type* of read misses. A similar table for write misses is given in Appendix A.

3 Analytical model

3.1 Estimating the execution time

A program's execution time can be estimated as:

$$T_{execute} = T_{instrs} + \sum_{r} M_r \times L_r \tag{1}$$

where T_{instrs} is the time to execute the instructions on each processor (assuming good load balance), M_r is the number of type r cache misses and L_r is the latency for type r misses. If there is no contention, L_r is the summation of service times of the resources used by a type r miss. If there *is* contention, then L_r is the summation of service times *plus the queuing times* on those resources, i.e.:

$$L_r = \sum_{k}(W_{k,r} + S_{k,r}) \tag{2}$$

In Equation (2), $W_{k,r}$ is the total waiting time of a request of type r on a resource of type k, and similarly $S_{k,r}$ is the total service demand of request of type r on resource of type k. As shown in Table 1 each miss request of a given type generates a specific sequence of sub-requests. These sub-requests, the resources they use, and their semantics are shown in Table 2. Table 3 is a synthesis of Table 1 and Table 2 showing in detail the sub-requests and associated resources for each miss

Miss Types	Resources	Situations and Operations
*Intra_R*1	Bus Cache	*Requested data exists in one of the local caches.* The snoopy mechanism transfers data from cache to cache. The directory state does not change, and the protocol processor is not involved.
*Intra_R*2	Bus PP Memory	*Home node is local*, requested data is not in local caches. Home node issues a memory read and updates the directory.
*Inter_R*3	Bus PP Network Cache Memory	*Home node is local*, data is owned by a remote cluster. Home node sends a write-back to the owner. When the data comes back, it is supplied to the requesting processor and written back to memory. Home node updates the directory.
*Inter_R*4	Bus PP Network Memory	*Home node is remote*, data is clean. Request is forwarded to home node. Home node issues a memory read, sends the data to the requesting node, and updates the directory.
*Inter_R*5	Bus PP Network Cache Memory	*Home node is remote*, data is owned by the home cluster. Request is forwarded to home node. Home node issues a write-back locally. When the data comes back, it is forwarded to the requesting node and written to memory. Home node updates the directory.
*Inter_R*6	Bus PP Network Cache Memory	*Home node is remote*, data is owned by a third cluster. Request is forwarded to home node. Home node requests the owner to write back. When data arrives, it is forwarded to the requesting cluster and written back to memory. Home node updates the directory.

Table 1: Classification of read misses. This table lists six types of read misses, resources used by each type, and a high-level description of the protocol followed on a miss.

type (similar tables for write misses can be found in Appendix B). For example, a read miss of type *Inter_R*3 first generates the following 4 sub-requests in its home cluster:

1. Request for data on cluster's address bus (Areq on resource Abus)
2. Insertion in the input queue of the bus interface (BreqI on BI(I))
3. Protocol processing operation (PPops on PPcore)
4. Send a write-back request to a remote cluster (NreqO on NI(O))

Then in the remote cluster, the write-back request will pass through the network input queue (NreqI), the forwarding module (Freq), the bus interface output queue (BreqO), and be placed on the address bus (Areq). From there on, there will be another sequence of sub-requests to get the data from one of the caches or local memory of the remote cluster, pass it to the bus interface, forwarding module, network interface, and network. Finally, there will be a last sequence of sub-requests in the home node.

Sub-requests	Resources:	Abbrev.	Meaning
Areq	Address bus:	Abus	Request data/write-back invalidation/ownership
Xdat/Xack/Xown	Data bus:	Dbus	Transfer data/acknowledgement/ ownership
Rl2	L2 cache:	L2	Read data from L2 cache
Rmem/Wmem	Memory:	Mem	Read/Write memory
Rrc/Wrc	Remote cache:	RC	Read/Write remote cache
BreqI/BdatI/BackI	Bus interface (IN):	BI(I)	Receive req/dat/ack from BI
BreqO/BdatO/BownO	Bus interface (OUT):	BI(O)	Send req/dat/own to BI
NreqI/NdatI/NackI/NownI	Network interface (IN):	NI(I)	Receive req/dat/ack/own from NI
NreqO/NdatO/NackO/NownO	Network interface (OUT):	NI(O)	Send req/dat/ack/own to NI
Freq/Fdat/Fack/Fown	Forwarding module:	Fwd	Forward messages
PPops	PP core:	PPcore	Protocol processing operations e.g., PPrecv/PPsend, PPsched, DIRstatus, DIRadd, etc
PPrecv/PPsend	PP core:	PPcore	Send/Receive message by PP core
PPsched	PP core:	PPcore	Dispatch/Invoke protocol handler
DIRstatus	PP core	PPcore	Look up status of a cache block
DIRadd	PP core:	PPcore	Add a node
DIRdel	PP core:	PPcore	Delete a node
DIRrtrv	PP core:	PPcore	Retrieve a node

Table 2: Definitions of sub-requests and resources required. This table shows the sub-requests, their names, the resources they use with their abbreviated names, and the actions following each sub-request. Only a subset of the protocol processing operations are displayed.

3.2 Architectural assumptions

As evidenced by the example in the previous paragraph and by Tables 3, 11, and 12, our architectural model is quite detailed. Nonetheless, we make a few assumptions and simplifications to keep the analysis computationally tractable and effective.

To start with, we consider only symmetric systems, i.e., each cluster contains the same number of processors, same amount of memory, and remote cache if any. Then looking at each component of a cluster, we assume the following:

- The instructions executed on the compute processor are perfectly pipelined. Therefore the CPI is one under an ideal memory system.
- L2 cache misses block the compute processors that must wait for the memory requests to complete before executing their next instructions.
- The L2 caches have dual ported tags. Hence, snooping on the bus does not interfere with a processor accessing its L2 cache if the processor and the snoop controller access different cache lines. We will ignore contention at the L2 level since the concurrent access to the same L2 cache line by the processor and by the snoop controller is extremely rare.

Type of	Sub-requests									
Misses	Abus	Dbus	L2	Mem	BI(I)	BI(O)	NI(I)	NI(O)	Fwd	PPcore
Intra_R1	Areq	Xdat	Rl2							
Intra_R2	Areq	Xdat		Rmem	BreqI					PPops
Inter_R3	Areq	Xdat		Wmem	BreqI	BdatO	NdatI	NreqO		PPops
	Areq	Xdat	Rl2		BdatI	BreqO	NreqI	NdatO	Freq Fdat	
Inter_R4	Areq	Xdat			BreqI	BdatO	NdatI	NreqO	Freq Fdat	
				Rmem			NreqI	NdatO		PPops
Inter_R5	Areq	Xdat			BreqI	BdatO	NdatI	NreqO	Freq Fdat	
	Areq	Xdat	Rl2	Wmem	BdatI	BreqO	NreqI	NdatO		PPops
Inter_R6	Areq	Xdat			BreqI	BdatO	NdatI	NreqO	Freq Fdat	
				Wmem			NreqI NdatI	NreqO NDatO		PPops
	Areq	Xdat	Rl2		BdatI	BreqO	NreqI	NdatO	Freq Fdat	

Table 3: Service demands of read misses on shared resources. Rows of this table show the sub-requests and the resources needed (each resource has its own column) in a given cluster for a particular type of read miss (cf. Table 2 for the meanings of abbreviations). For misses involving multiple clusters, the first row shows the service demands for the local cluster; the second and third rows (if present) are the service demands for the second and the third clusters involved.

- The cluster buses are split transaction [7]. The address bus and the data bus are two separated resources and can be used simultaneously for different transactions.

Finally, we do not consider the contention on the network for three reasons: (1) The bandwidth provided by current network technology appears to be sufficient for the size of the systems investigated in this paper and contention is not an important factor; (2) Many models have been developed to analyze the performance of various networks [1, 13, 5]; and (3) The MVA (Mean Value Analysis) technique we use cannot cope easily with the network contention directly. If need be the results of the contention models just alluded to could be incorporated in the analysis (cf. Section 6).

3.3 Modeling contention

Returning to Equation (2), $S_{k,r}$ and $W_{k,r}$ are respectively the summations of the service times and waiting times of the sub-requests issued to k by a type r miss. While the $S_{k,r}$'s are architectural parameters, contention is present in the determination of the $W_{k,r}$'s. We use a closed queuing network to model the cluster architectures, with compute processors as customers, and buses, memories, and protocol processors as service centers. We will use the MVA technique and the following notations to solve $W_{k,r}$.

- N is the total number of processors in the system.
- N_c is the number of processors per cluster, i.e., the cluster size.
- P_r is the probability that a miss request is of type r.
- I is the average number of instruction cycles between cache misses.
- R is the mean total time between cache misses.
- U_k is the utilization at type k center.
- $Q_{k,loc}$ ($Q_{k,rmt}$) is the arrival queue length at k observed by a local (remote) request.
- $s_{k,r,i,loc}$ ($s_{k,r,i,rmt}$) is the service demand of the ith sub-request to k from a local (remote) r miss. Recall that a cache miss may issue multiple sub-requests to a service center.
- s_k is the average sub-service demand on type k center.
- $m_{k,r,loc}$ ($m_{k,r,rmt}$) is the number of sub-requests to k issued by a local (remote) r miss. If r does not require local (remote) k resource, $m_{k,r,loc} = 0$ ($m_{k,r,rmt} = 0$).
- $m_{k,r}$ is the number of local and remote sub-requests issued to type k resource by a type r miss. $m_{k,r} = m_{k,r,loc} + m_{k,r,rmt}$. If r does not use type k resource at all, $m_{k,r} = 0$.
- $w_{k,loc}$ ($w_{k,rmt}$) is the waiting time at k for a sub-request issued by a local (remote) miss.

Based on the assumption that a cache miss blocks the compute processor, each processor alternates between executing instructions and waiting for miss requests served by the memory system. Thus the mean total time between cache misses can be expressed by the instruction cycles between two misses under an ideal memory system plus the time spent in the memory system:

$$R = I + \sum_r (P_r \times \sum_k (W_{k,r} + S_{k,r})) \tag{3}$$

where

$$W_{k,r} = \begin{cases} 0, & \textit{if } k \textit{ is L2 cache or network} \\ m_{k,r,loc} \times w_{k,loc} + m_{k,r,rmt} \times w_{k,rmt}, & \textit{otherwise} \end{cases} \tag{4}$$

$$S_{k,r} = \sum_{i=1}^{m_{k,r,loc}} s_{k,r,i,loc} + \sum_{j=1}^{m_{k,r,rmt}} s_{k,r,j,rmt} \tag{5}$$

Equations (4) and (5) express the total waiting time and total service time in terms of the waiting time and the service time of sub-requests respectively. We now consider the queuing effect at each type of service center on the waiting time of each sub-request. Suppose a type r miss issued from cluster C requires resource k of the local cluster, and the same type of resource k of a remote cluster. Since the system is symmetric, for each service performed at a remote cluster, resource k of C will need to perform the equivalent service on behalf of a request of type r issued from a remote cluster. Therefore, for any request issued from the local cluster (C), the arrival queue length at k ($Q_{k,loc}$) is contributed by (1) all the sub-requests issued by the other $N_c - 1$ local processors and (2) by the remote requests of the $N - N_c$ remote processors, with each request having a probability $\frac{N_c}{N-N_c}$ of requesting cluster C. Similarly, for any request issued from a remote cluster, the arrival queue length at k ($Q_{k,rmt}$) is contributed by the sub-requests issued by the local N_c processors and $\frac{N_c}{N-N_c}$

of the sub-requests issued by the other $N - N_c - 1$ remote processors. Hence, the arrival queue lengths at center k observed by a local or a remote request are:

$$\begin{aligned} Q_{k,loc} &= \frac{\sum_{\mathrm{r}} \mathrm{P_r} \times [(\mathrm{N_c}-1) \times \sum_{\mathrm{i=1}}^{\mathrm{m_{k,r,loc}}} (\mathrm{w_{k,loc}} + \mathrm{s_{k,r,i,loc}}) + (\mathrm{N}-\mathrm{N_c}) \times \frac{\mathrm{N_c}}{\mathrm{N}-\mathrm{N_c}} \times \sum_{\mathrm{j=1}}^{\mathrm{m_{k,r,rmt}}} (\mathrm{w_{k,rmt}} + \mathrm{s_{k,r,j,rmt}})]}{\mathrm{R}} \\ Q_{k,rmt} &= \frac{\sum_{\mathrm{r}} \mathrm{P_r} \times [\mathrm{N_c} \times \sum_{\mathrm{i=1}}^{\mathrm{m_{k,r,loc}}} (\mathrm{w_{k,loc}} + \mathrm{s_{k,r,i,loc}}) + (\mathrm{N_c} - \frac{\mathrm{N_c}}{\mathrm{N}-\mathrm{N_c}}) \times \sum_{\mathrm{j=1}}^{\mathrm{m_{k,r,rmt}}} (\mathrm{w_{k,rmt}} + \mathrm{s_{k,r,j,rmt}})]}{\mathrm{R}} \end{aligned} \tag{6}$$

Finally, we can estimate the waiting time for each sub-request by the arrival queue lengths and utlization.

$$\begin{aligned} w_{k,loc} &= (Q_{k,loc} - U_k) \times s_k + U_k \times s_k/2 \\ w_{k,rmt} &= (Q_{k,rmt} - U_k) \times s_k + U_k \times s_k/2 \end{aligned} \tag{7}$$

where

$$s_k = (\sum_{r\ use\ k} P_r \times \frac{S_{k,r}}{m_{k,r}})/(\sum_{r\ use\ k} P_r) \tag{8}$$

$$U_k = N_c \times \frac{\sum_r P_r \times S_{k,r}}{R} \tag{9}$$

Since Equations (3)-(9) contain cyclic interdependencies, $W_{k,r}$ will be solved iteratively with $w_{k,loc}$ and $w_{k,rmt}$ initialized to 0.

4 Model parameters

The input parameters of the model consist of those solely determined by the architectures, such as the service demands of the sub-requests listed in Table 3, and of those that are dependent upon the applications as well as the hardware and software systems, such as the numbers and types of cache misses.

4.1 Architectural parameters

We fix the total number of processors (16 processors) and the total amount of memory in the system. We will vary the number of processors in each cluster, or cluster size, from a single processor per cluster (as in FLASH [8]), to 2, 4, 8, and finally 16, i.e., a single cluster corresponding to a conventional shared-bus multiprocessor. In the alternative designs that employ remote caches, we will keep the remote cache size per processor a constant while altering the cluster size, so that comparisons between various configurations are as fair as possible. Other architectural parameters such as those related to the L2 caches and the cluster bus are given in Table 4. The latencies of the sub-requests are given in Table 13 in the Appendix C. The network latency is 24 cycles.

L2 cache	$size = 64kb$ $line\ size = 64byte$ $assoc = 1$
Remote cache per processor	$size = 256kb$ $line\ size = 64byte$ $assoc = 4$
Datapath width of PP	64bits
Data bus width	64bits
CPU clock/bus clock	2

Table 4: Architectural parameters

Program	Problem size	Total instr(M)	Shared read(M)	Shared write(M)
FFT	64k points	33.32	6.00	5.73
RADIX	0.5M integers, 1024 radix	29.55	6.70	3.46
RAYTRACE	teapot	375.93	54.60	0.22

Table 5: Applications: problem sizes, number of instructions executed (in millions), number of shared read references (in millions), number of shared write references (in millions).

4.2 Applications parameters

In order to exercise the model properly we need reasonably balanced workloads. To that effect, we selected FFT, RADIX, and RAYTRACE from the SPLASH-2 benchmark suite [23]. The algorithms for FFT and RADIX are given in [3] and [6] respectively. RAYTRACE is an image processing program which renders a three-dimensional scene using ray tracing [18]. Table 5 summarizes some pertinent statistics about the applications: problem size, number of instructions, number of read and write references to shared data.

For determining the application dependent parameters, we have performed trace-driven simulations in which we assume that cache misses are insensitive to the accurate timing information, hence we use constant latencies for cache misses. The assumption is reasonable for many applications for which false sharing is a rare case such as FFT[1] The application dependent parameters of principal interest are the numbers of cache misses of each type and the probability that a replaced cache block is dirty and, in that case, whether its home node is local or remote. To gather these statistics, we use Mint [20] as the simulation tool. Mint is a software package that emulates multi-processing execution environments and generates memory reference events which drive a memory system simulator.

Figures 2, 3, and 4 show the cache miss profiles of the three applications. The four and half pairs of cache miss profiles correspond to architectures of cluster size 16, 8, 4, 2, and 1 respectively;

[1]The version we use has been optimized to eliminate false sharing and to facilitate bulk data transfer.

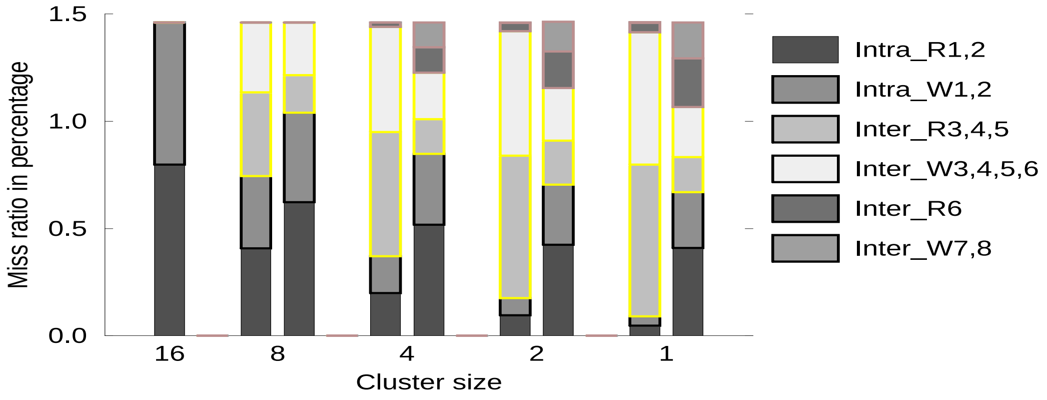


Figure 2: The cache miss profile of FFT. The left (right) bar of each pair corresponds to an architecture not containing (containing) remote caches. $Intra\_R1,2$ and $Intra\_W1,2$ are intra-cluster read and write misses. $Inter\_R3,4,5$ and $Inter\_W3,4,5,6$ are inter-cluster misses which involve two clusters. $Inter\_R6$ and $Inter\_W7,8$ are misses involving three clusters. Tables 1 and 10 give detailed explanation of each type of misses.

the left (right) bar of each pair corresponds to an architecture not containing (containing) remote caches. Since the chosen applications have not been programmed to take advantage of the cluster architectures, the numbers of intra-cluster misses of shared references drop, as expected, when the cluster size decreases. In the cases of FFT and RADIX, when the cluster size halves, the intra-cluster misses reduce by half approximately. A plausible explanation for this is that data in processor caches and/or cluster memories is uniformly distributed and that the average number of processors sharing a piece of datum is 2. Therefore when the cluster size reduces by half, nearly one-half of the intra-cluster misses have to cross the cluster boundary. For RAYTRACE, the average number of processors sharing a datum is higher. As a result, when a cluster splits into two, a smaller portion of the intra-cluster misses turn into inter-cluster misses.

Adding remote caches changes significantly the cache miss profiles. The presence of remote caches increases the retention of remote data. When a line, whose home is in a remote cluster, is replaced in an L2 cache, it may still exist in the remote cache either because of some inclusion property or because on a replacement the first option is to write to the remote cache. Thus, the remote caches can transform some of the inter-cluster misses caused by conflict mapping or capacity limitation in the L2 caches into intra-cluster misses. The extent of the reduction in the inter-cluster misses is determined by the proportion of conflict and capacity misses in these inter-cluster misses.

From [23], we know that when processor caches are of limited size, both RADIX and RAYTRACE encounter significant amounts of capacity misses. For these two applications, the remote caches are very effective in transforming the inter-cluster misses into intra-cluster misses. On the other hand, FFT has a smaller proportion of capacity misses, therefore the remote caches are not as helpful. As we increase the size of L2 caches (not shown), the benefit of the remote cache diminishes. Finally,

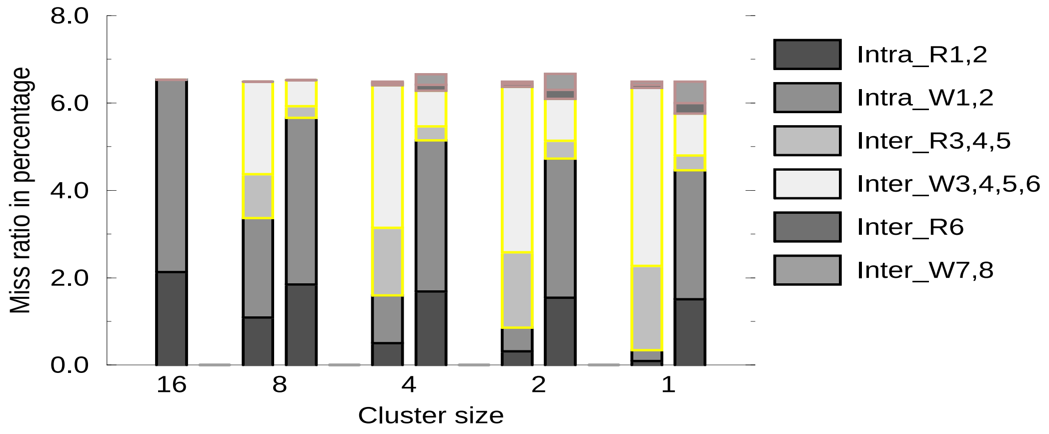


Figure 3: The cache miss profile of RADIX. The left (right) bar of each pair corresponds to an architecture not containing (containing) remote caches. $Intra\_R1,2$ and $Intra\_W1,2$ are intra-cluster read and write misses. $Inter\_R3,4,5$ and $Inter\_W3,4,5,6$ are inter-cluster misses which involve two clusters. $Inter\_R6$ and $Inter\_W7,8$ are misses involving three clusters. Tables 1 and 10 give detailed explanation of each type of misses.

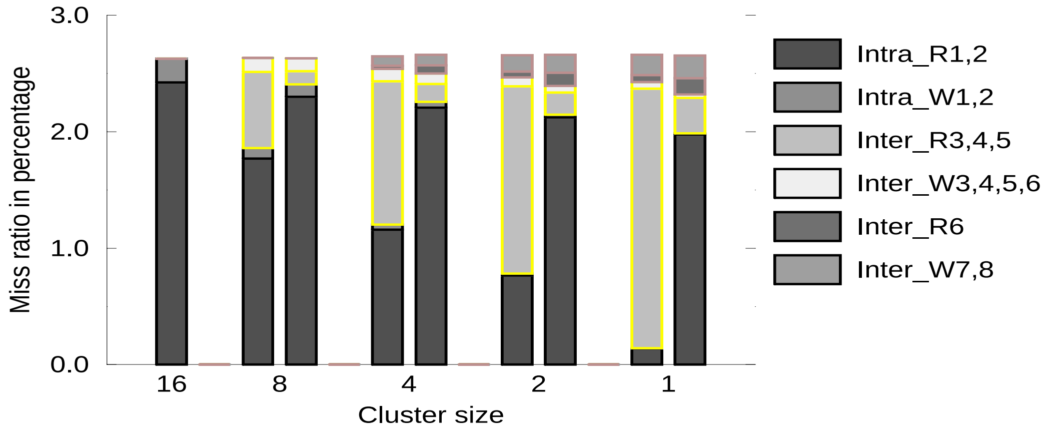


Figure 4: The cache miss profile of RAYTRACE. The left (right) bar of each pair corresponds to an architecture not containing (containing) remote caches. $Intra\_R1,2$ and $Intra\_W1,2$ are intra-cluster read and write misses. $Inter\_R3,4,5$ and $Inter\_W3,4,5,6$ are inter-cluster misses which involve two clusters. $Inter\_R6$ and $Inter\_W7,8$ are misses involving three clusters. Tables 1 and 10 give detailed explanation of each type of misses.

we note that remote caches also bring a negative impact. Because they tend to retain data in remote clusters longer, they increase the 3-hop inter-cluster misses (misses involving three clusters).

5 Exercising the model: performance evaluation

The cache miss profiles by themselves are not sufficient to assess the performance of the architectures. For example, in a single bus system, all cache misses are intra-cluster misses. However if the bus is saturated, the miss latency of the intra-cluster misses will significantly increase. When we use a balanced multi-cluster architecture, the inter-cluster misses may still incur a latency longer than that of the misses of the saturated single bus system but the overall performance may be better since the intra-cluster misses of the multi-cluster architecture can be resolved faster.

5.1 Average service demand and contention

We have shown in Tables 3 and 11 the service demands of each type of cache misses. Based on these tables, the parameters given in Table 13, and the cache miss profile, we can compute an application's average service demand (per processor) for each resource (cf. Equation (8) of Section 3.3). Table 6 displays average service demands for the RADIX application. The magnitude of the resource service demands determines the relative intensity of contention. The resource with the highest demand is subject to the highest contention and will be the first to saturate among all. As can be seen in Table 6, of the 10 shared resources, the data bus and the protocol processing core are the two resources with the heaviest demand. This is true not only of RADIX but also of the other two applications FFT and RAYTRACE. We will therefore focus our attention on these two resources.

As the cluster size becomes smaller, the number of intra-cluster misses decreases. The reduction in the number of intra-cluster misses does not imply a reduction in service demand on the data bus. Independently of whether the miss is intra or inter-cluster, the data has to transit on the local cluster's bus. In fact, the demand on the bus might increase in inter-cluster misses if the data is owned by a cache of the remote cluster. In that case, the data bus of the remote cluster is used rather than the direct link from protocol processor to memory. According to the symmetry principle that we used earlier, the overall demand on the data bus service center is therefore increased. In our experiments, most of the remote requests were serviced by memory and the demand on the data bus remained practically unchanged with cluster size. Of course, as the number of requesters in the cluster decreased, contention for the bus decreased.

For the protocol processing core, a larger inter-cluster miss ratio certainly leads to a higher service demand. Like in the case of the data bus, the increase is rather limited. For example, looking at RADIX, the percentage of the inter-cluster misses rises from 50% to 75% (cf. Figure 3), but the service demand increases by less than 10% when the cluster size is halved from 8 to 4. There are two reasons that can explain this fact. First, some of the intra-cluster misses, specifically $Instr\_R2$ or $Intra\_W2$, also require the service from the protocol processing core. Second, the design of the forwarding module offloads the service demand from the protocol processor for the inter-cluster misses.

Cluster size	Average service demands									
	NO remote cache									
	Abus	Dbus	RC	Mem	BI(I)	BI(O)	NI(I)	NI(O)	Fwd	PPcore
16	6.69	31.46	0.00	10.12	0.00	0.00	0.00	0.00	0.00	0.00
8	7.32	33.07	0.00	11.48	10.17	4.90	11.80	18.84	10.16	71.43
4	7.48	33.34	0.00	11.95	10.47	6.42	18.50	29.59	15.99	77.00
2	7.59	33.60	0.00	12.41	10.83	7.43	21.46	34.23	18.53	80.88
1	7.56	33.56	0.00	12.54	10.85	8.06	23.85	38.02	20.54	81.61
	WITH remote cache									
8	7.38	33.50	3.00	8.47	6.78	4.35	6.35	10.50	5.47	52.21
4	7.68	34.35	4.62	7.60	5.61	5.56	10.66	17.66	9.26	49.73
2	7.93	35.21	5.28	7.59	5.49	6.54	12.81	21.07	11.11	50.85
1	7.91	35.53	5.43	7.62	5.58	7.04	14.43	23.57	12.43	51.05

Table 6: RADIX's average service demands.

Note than when remote caches are employed, the demand on the data bus is barely altered while the demand on the protocol processing core drops significantly. In this architectural variation, the contention on the protocol processor will be governed by the cluster size as well as by the presence of the remote caches.

5.2 Overall performance

In order to assess the overall performance, i.e., execution time, of the various configurations we exercise the model presented in Section 3.3. We first compute the waiting times of each cache miss type on a given resource (cf. Equation (4) that has to be solved iteratively) and combine them with the service demands computed as in the previous section to obtain the cache miss latencies. These latencies are shown in Table 7 for the RADIX application. We obtain the execution time by applying Equation (1). The results for the three applications are shown in Table 8, which shows the execution times of the 3 applications normalized to the execution in a single cluster.

From Table 7 we see that both the intra and the inter-cluster miss latencies decrease monotonically as more resources are added to the system. The large differences (from 60% to 270%) between the intra-cluster miss latency of the single bus system and those of other architectures indicate that the contention in the former system is substantial. When the single cluster system splits into two, the contention on the data bus drops dramatically but the intra-cluster miss latency remains high. This is because the contention on the protocol processor is severe. As mentioned before, some of the intra-cluster misses (*Intra_R/W2* and local replacement of a dirty line) of the multi-cluster architectures call for the service of the protocol processor. In other words, the intra-cluster miss latency is affected by the contention on the protocol processor.

The normalized execution time is determined principally by the number of instructions executed between cache misses (I in Equation (3)) and the average service demand on each resource, rather

	Miss latencies			
	NO remote cache		WITH remote cache	
Cluster size	Intra-cluster	Inter-cluster	Intra-cluster	Inter-cluster
16	259	-	-	-
8	158	416	119	384
4	80	278	60	278
2	71	235	50	244
1	71	222	42	234

Table 7: RADIX's average cache miss latencies

than by the shared miss ratio. RADIX and RAYTRACE have almost the same miss issuing rate. Therefore, even though RADIX has a much higher miss ratio (6.8%) than RAYTRACE does (2.7%), their normalized execution times do not differ considerably as long as there are at least 4 clusters. The reason that RAYTRACE has slightly better normalized execution times is that its average service demand on the protocol processor is significantly less. As a result, when there are as many as 8 processors in each cluster, the overall performance of RAYTRACE surpasses the single cluster system, while for RADIX, the turning point occurs at the cluster size of 4. FFT, on the other hand, has a miss issuing rate lower than RADIX and RAYTRACE. Its contention on the single bus system is not as high as those of the other two applications. Thus, for this specific application, when no remote caches are employed, the performances of cluster architectures do not match up that of the single bus system.

When remote caches are included the cluster architectures consistently perform better than the single cluster system on all three applications from 15% to 33%. The benefits incurred by the presence of the remote caches derive from the fact that they increase the proportion of the intra-cluster misses *and* they also reduce the intra-cluster miss latency while increasing the inter-cluster miss latency to a smaller extent (cf. Table 7). The latency of intra-cluster misses decreases because the data supplied by remote cache do not involve the protocol processor. The slightly larger inter-cluster miss latency is caused by higher 3-hop inter-cluster miss ratio, *not* by the contention on the protocol processor, which, in fact, is reduced by the presence of the remote caches.

In summary, our model shows that cluster architectures, with no optimization in the software to take advantage of the architectural organization, would provide little benefit over a flat architecture. However, by increasing the configurations with remote caches, the performance improvements are quite noticeable. This last observation certainly justifies the design decisions in recent cluster architectures [14, 17].

The architectural parameters of Table 4 are justifiable with the current technology for processors, caches, and buses. As newer processor designs adopt more aggressive superscalar techniques the miss issuing rate will increase. In other words, the number of instruction cycles between misses decreases. Second, CPU clocks are getting faster, making the bus and memory relatively slower. This translates into larger (in terms of CPU cycles) service demands on these resources. Obviously, both trends tend to increase the contention on these shared resources. A feature that would counter

	Normalized execution time					
	RADIX		FFT		RAYTRACE	
Cluster size	No RC	With RC	No RC	With RC	No RC	With RC
16	1.00	-	1.00	-	1.00	-
8	1.17	0.88	1.22	0.90	0.79	0.69
4	0.94	0.71	1.07	0.85	0.82	0.67
2	0.89	0.70	1.06	0.87	0.85	0.68
1	0.90	0.71	1.06	0.88	0.91	0.70

Table 8: Execution times of the 3 applications for various configurations. The execution times are normalized to the execution time on a single cluster. RC stands for remote cache.

balance these effects would be the presence of wider cluster data buses and wider data paths in the protocol processor. Data transfers in these resources that are potential bottlenecks would be sped up.

As a simple experiment to see the effects on cluster architectures of these technological trends, we doubled four parameters:

1. The ideal miss issuing rate (or $1/I$), i.e., we assume that the CPI of each processor was halved.
2. The ratio of CPU clock rate vs. bus clock rate, i.e., we assumed that the CPU clock will improve at twice the rate of the bus clock.
3. The width of the data bus.
4. The width ofthe data path of the protocol processor, with the increase of this parameter reducing the latency of data transactions in the protocol processor.

With these changes in parameters, the service demand on the data bus remains approximately the same since the doubling of the CPU to bus clock ratio is counter-balanced by the doubling of the width in the data bus. However, contention is increasing since the issue rate is also doubled.

The results of exercising the model with these new parameters are shown in Table 9. Now cluster architectures, with and without remote caches, are clearly more advantageous. When there are 4 processors in each cluster, the cluster architectures perform 29%-46% better than the single bus system without the remote caches, and 43%-57% better with the remote caches. This is because the contention on the data bus of the single bus system will be intensified at a rate faster than the contention on the protocol processor of the cluster architectures.

6 Related work

The behaviors of the service centers contained in our model generally satisfy the two assumptions required by the MVA technique, i.e., routing homogeneity and service time homogeneity [11]. Vernon, Lazowska and Zahorjan have shown that the MVA technique applied to shared-bus multiprocessor

	Normalized execution time					
	RADIX		FFT		RAYTRACE	
Cluster size	No RC	With RC	No RC	With RC	No RC	With RC
16	1.00	-	1.00	-	1.00	-
8	0.86	0.66	0.84	0.65	0.58	0.55
4	0.71	0.57	0.66	0.50	0.54	0.43
2	0.69	0.57	0.62	0.49	0.55	0.42
1	0.69	0.57	0.62	0.49	0.60	0.43

Table 9: Normalized execution time for the "faster" architecture.

systems is of remarkable accuracy [22]. Vernon et al. applied the same technique to studying the performance of purely bus-based hierarchical multiprocessors [21]. Our model and theirs share some common elements because of the similarity in the architectures. Torrellas, Hennessy and Weil also developed analytical models to investigate the impact of several architectural and application parameters in shared-memory processors based on DASH [19]. The main differences between their model and ours are that they used an open queuing network and that the inter-cluster interface was hardwired [12]. They found that contention for the bus dominated that of the cluster interface[2]. We show that this conclusion is no longer true in all cases when the design shifts from hardwired controller to protocol processor.

Holt et al. studied, via simulation, the effects of the occupancy of the protocol processor and of the latency of the network for a Flash-like architecture where the number of processors per cluster is one [9]. They showed that the time consumed by the protocol processors did have an impact on the performance, especially with a fast network. However it is not clear whether contention causes considerable delay in protocol processors or not. Cluster-based architectures where inter-cluster communication is provided by a (hierarchy of) bus and a shared cache(s) have been studied also via simulation by Nayfeh et al. [15, 16] and by Anderson [2]. Conclusions on the viability of clusters are mixed depending on the amount of memory requests and their locality.

As several processors share a protocol processor, the latter is likely to become the performance bottleneck. Some researchers have looked at the idea of using one of the processors in the cluster as a protocol processor. If a cache or page miss blocks the computation, the compute processors may as well be used to resolve the fault. Karlsson and Stenstrom simulated such a scheme for a multiprocessor which used an ATM network to link clusters of bus-based multiprocessors [10]. The intra-cluster coherence is maintained by cache snoopy hardware while inter-cluster coherence is maintained at the page level by distributed virtual shared memory software. With a network latency of $100us$, the chance of finding a compute processor idle is quite high (52%-92% for 4 processors/cluster) even when a simple round-robin policy is used for scheduling the task of protocol processing. The bottleneck is in the ATM interface and the software must be tailored to the application for significant speed-ups to occur. The authors conclude that in the context of such an architecture, and accompanying software, the presence of a separate protocol processor is not necessary.

[2]The number of processors per cluster was fixed at four.

In our study we assumed that the interconnection network is of sufficient bandwidth and used a contention-free network latency in the model. However a network model can be easily incorporated in the evaluation if the contention is not negligible. There exists a rich set of analytical models for various network topologies [1, 13, 5]. These models usually take the request rate as an input parameter and compute the network latency taking into effect the contention. This latency could be used in our model instead of the contention-free network latency to achieve a more accurate response time for inter-cluster misses. The response time then would in turn affect the request rate issued to the network. The two models would have to iterate until they both converge.

7 Conclusion

In this paper, we have applied Mean Value Analysis to assess the performance of cluster-based architectures. We have accurately characterized the service demand of shared resources by examining in detail the sub-requests involved in the resolution of cache misses. In addition to the overall system parameters and the service demands on shared resources, the analytical model needs parameters pertinent to applications, i.e., the cache miss profiles. These parameters were obtained via trace-driven simulation for three applications.

We have compared the performance of cluster-based architectures of various cluster sizes with and without remote caches. Our simulation and analytical results shed some light on three different aspects of the impact of cluster-architectures[3]: the applications' cache miss profile, the service demand and the contention for shared resources, and the overall performance normalized to the single bus system. We summarize these results as follows:

- Many existing parallel programs, and our example programs, are written for machines with a flat architectural model. For these applications, and for the cluster-based architectures without remote caches, a significant portion of intra-cluster misses (33% to 50%) becomes inter-cluster misses of longer latency when the cluster size is reduced by half.
- When large remote caches are present, they retain, as expected, much of the remote data in the cluster. They prevent part of the intra-cluster misses (capacity misses and conflict misses) from becoming inter-cluster misses when the cluster size decreases.
- In the cluster-based architectures which employ protocol processors and maintain cache coherence in software (i.e., the protocol processor is distinct from the bus and network interfaces), the protocol processor and the data buses are the resources for which there is the greatest service demand.
- Service demand on the data bus remains approximately the same independently of the cluster size (although contention decreases with cluster size). The service demand on the protocol processor increases as the cluster size decreases, but it does not increase as fast as the rate of inter-cluster misses.
- Remote caches reduce the service demand on the protocol processor significantly but have little impact on the service demand on the data bus.

[3]We are in the process of validating the models with simulations.

- The contention for shared resources is determined mainly by three factors: the miss issuing rate, the cluster size, and the presence of a remote cache. The contention on the data bus and on the protocol processor are the prime factors influencing the performance of the cluster-based architectures.
- Without the remote caches, the performance of the cluster-based architectures is mixed. For two out of the three applications, a cluster size of 2 or 4 gives the best results (improvements in the range of 6% to 18%.
- When remote caches are present, all configurations consistently outperform the flat architecture (improvements in the range 15-33%).
- Finally, if we modify the service demand parameters to reflect the technological trend of faster processors with respect to the speed of buses and latency of memories, we see that the cluster-based architectures will have an increasing performance edge over the single bus system.

References

[1] A. Agarwal. Limits on interconnection network performance. *IEEE Trans. on Parellel and Distributed Systems*, 2(4):398–412, April 1991.

[2] C. S. Anderson. *Improving Performance of Bus-Based Multiprocessors.* PhD thesis, Dept. of Computer Science, Univ. of Washington, 1995.

[3] D. H. Bailey. FFT in external or hierarchical memory. *J. of Supercomputing*, 4(1):23–35, March 1990.

[4] G. Bell. Multis: A new class of multiprocessor computers. *Science*, pages 462–467, April 1985.

[5] L. N. Bhuyan, Q. Yang, and D. P. Agrawal. Performance of multiprocessor interconnection networks. *Computer*, 22(2):25–37, February 1989.

[6] G. E. Blelloch et al. A comparison of sorting algorithms for the connection machine CM-2. In *Proc. of the Symposium on Parallel Algorithms and Architecture*, pages 3–16, 1991.

[7] M. Galles and E. Williams. Performance optimizations, implementation, and verification of the SGI challenge multiprocessor. In *Proc. of the 27th Hawaii International Conference on System Sciences. Vol. I: Architecture*, pages 134–43, 1994.

[8] J. Heinlein, K. Gharachorloo, S. Dresser, and A. Gupta. Integration of message passing and shared memory in the Stanford FLASH multiprocessor. In *Proc. of 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 38–50, 1994.

[9] C. Holt. The effects of latency, occupancy, and bandwidth in distributed shared memory multiprocessors. Technical report, Dept. of Computer Science, Stanford Univ., 1995. CSL-TR-95-660.

[10] M. Karlsson and P. Stenstrom. Performance evaluation of a cluster-based multiprocessor built from ATM switches and bus-based multiprocessor servers. In *Proc. of 2nd Conf. on High-Performance Computer Architecture*, pages 4–13, 1996.

[11] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance.* Prentice-Hall, Inc., 1984.

[12] D. Lenoski et al. The Standford DASH multiprocessor. *IEEE Transactions on Computer*, 25(3):63–79, March 1992.

[13] T. Lin and L. Kleinrock. Performance analysis of finite-buffered multistage interconnection networks with a general traffic pattern. In *Proc. of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 68–78, 1991.

[14] T. Lovett and R. Clapp. STiNG: A CC-NUMA computer system for the commercial marketplace. In *Proc. of 23rd International Symposium on Computer Architecture*, pages 308–317, 1996.

[15] B. A. Nayfeh and K. Olukotun. Exploring the design space for a shared-cache multiprocessor. In *Proc. of 21st International Symposium on Computer Architecture*, pages 166–175, 1994.

[16] B. A. Nayfeh, K. Olukotun, and J.P. Singh. The impact of shared-cache clustering in small-scale shared-memory multiprocessors. In *Proc. of 2nd Conference on High-Performance Computer Architecture*, pages 74–84, 1996.

[17] A. Nowatzyk et al. The S3.mp scalable shared memory multiprocessor. In *Proc. of the 1995 International Conference on Parallel Processing, Vol II*, pages 1–10, 1995.

[18] J. P. Singh, A. Gupta, and M. Levoy. Parallel visualization algorithms: performance and architecture implications. *IEEE Computer*, 27(7):45–55, July 1994.

[19] J. Torrellas, J. Hennessy, and T. Weil. Analysis of critical architectural and program parameters in a hierarchical shared-memory multiprocessor. In *Proc. of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 163–72, 1990.

[20] J. E. Veenstra and R. J. Fowler. MINT: a front end for efficient simulation of shared-memory multiprocessors. In *Proc. of the 2nd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 201–7, 1994.

[21] M. Vernon, R. Jog, and G. S. Sohi. Performance analysis of hierarchical cache-consistent multiprocessors. In *Proc. of the International Seminar on Performance of Distributed and Parallel Systems*, pages 111–126, 1988.

[22] M. K. Vernon, E. D. Lazowska, and J. Zahorjan. An accurate and efficient performance analysis technique for multiprocessor snooping cache-consistency protocols. In *Proc. of 15th International Symposium on Computer Architecture*, pages 308–315, 1988.

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

A Classification of shared write misses

Miss Types	Resources	Situations and Operations
$Intra\_W1$	Bus Cache	*Requested data is owned by one of the local caches.* Requesting processor issues an ownership request on bus. Data is transferred from cache to cache with ownership while local cache copies are invalidated.
$Intra\_W2$	Bus PP Cache or Memory	*Home node is local*, requested data is clean in local cluster only. Requesting processor issues an ownership request on bus. Data is transferred from cache/memory/ , local copies are invalidated. Home node grants ownership and updates the directory.
$Inter\_W3$	Bus PP Network Cache	*Home node is local*, data is owned by a remote cluster. Request for ownership is submitted to home node. Home node requests the owner to write back/invalidate data. When data arrives, home node supplies it to the requesting processor with ownership, and updates the directory.
$Inter\_W4$	Bus PP Network Cache or Memory	*Home node is local*, data is clean in remote clusters too. Requesting processor issues ownership request on bus. Data is transferred from cache/memory/ , local copies are invalidated. Home node sends invalidation to remote clusters caching the data. After receiving acknowledgments home node grants ownership and updates the directory.
$Inter\_W5$	Bus PP Network Cache	*Home node is remote*, data is owned by the home node. Request for ownership is forwarded to home node. Home node issues a write-back/invalidation locally, sends the data with ownership to the requesting cluster, and updates the directory.
$Inter\_W6$	Bus PP Network Cache or Memory	*Home node is remote*, data is clean in local cluster only. Request for ownership is forwarded to home node. Data is transferred from cache/memory/ , local copies are invalidated. Home node grants ownership and updates the directory.
$Inter\_W7$	Bus PP Network Cache	*Home node is remote*, data is owned by a third cluster. Request for ownership is forwarded to home node. Home node issues a write-back/invalidation to the owner. When data comes back, home node sends the data with ownership to the requesting cluster and updates the directory
$Inter\_W8$	Bus PP Network Cache or Memory	*Home node is remote*, data is clean in remote clusters. Request for ownership is forwarded to home node. Data is transferred from cache/memory/ , local copies are invalidated. Home node sends invalidations to clusters caching the data. After receiving acknowledgments, home node grants ownership and updates the directory.

Table 10: Classification of write misses. This table lists eight types of write misses, resources used by each type, and a high-level description of the protocol followed on a miss.

B Service demand of shared write misses and cache replacements

Type of Misses	Sub-requests								
	Abus	Dbus	L2	Mem	BI(I)	BI(O)	NI(I)	NI(O)	Fwd	PPcore
Intra_W1	Areq	Xdat Xown	Rl2							
Intra_W2	Areq	[Xdat] Xown	[Rl2]	[Rmem]	BreqI	BownO				PPops
Inter_W3	Areq	Xdat Xown			BreqI	BdatO BownO	NdatI	NreqO		PPops
	Areq	Xdat	Rl2		BdatI	BreqO	NreqI	NdatO	Freq Fdat	
Inter_W4	Areq	[Xdat] Xown	[Rl2]	[Rmem]	BreqI	BownO	NackIs	NreqOs		PPops
	Areq	Xack			BackI	BreqO	NreqI	NackO	Freq Fack	
Inter_W5	Areq	Xdat Xown			BreqI	BdatO BownO	NdatI NownI	NreqO	Freq Fdat Fown	
	Areq	Xdat	Rl2		BdatI	BreqO	NreqI	NdatO NownO		PPops
Inter_W6	Areq	[Xdat] Xown	[Rl2]		BreqI	[BdatO] BownO	[NdatI] NownI	NreqO	Freq [Fdat] Fown	
	[Areq]	[XAck]		[Rmem]	[BackI]	[BreqO]	NreqI	[NdatO] NackO		PPops
Inter_W7	Areq	Xdat Xown			BreqI	BdatO BownO	NdatI NownI	NreqO	Freq Fdat Fown	
							NreqI NdatI	NreqO NdatO NownO		PPops
	Areq	Xdat	Rl2		BdatI	BreqO	NreqI	NdatO	Freq Fdat	
Inter_W8	Areq	[Xdat] Xown	[Rl2]		BreqI	[BdatO] BownO	[NdatI] NownI	NreqO	Freq [Fdat] Fown	
	[Areq]	[Xack]		[Rmem]	[BackI]	[BreqO]	NreqI NackIs	NreqOs [NdatO] NownO		PPops
	Areq	Xack			BackI	BreqO	NreqI	NackO	Freq Fack	

Table 11: Service demands of write misses on shared resources. Rows of this table show the sub-requests and the resources needed (each resource has its own column) in a given cluster for a particular type of write miss. For misses involving multiple clusters, the first row shows the service demands for the local cluster; the second and third rows (if present) are the service demands for the second and the third clusters involved.

Type of Replacement	Sub-requests									
	Abus	Dbus	L2	Mem	BI(I)	BI(O)	NI(I)	NI(O)	Fwd	PPcore
Dirty HomeLocal	Areq	Xdat		Wmem	BreqI BdatI					PPops
Dirty HomeRemote	Areq	Xdat			BreqI BdatI			NreqO NdatO	Freq Fdat	
				Wmem			NreqI NdatI			PPops

Table 12: Service demands for replacing dirty data in cache. Rows of this table show the sub-requests and the resources needed (each resource has its own column) in a given cluster for the two possible replacement requests. For the replacement type involving two clusters, the first row shows the service demands for the local cluster; the second row shows the service demands for the second cluster involved.

C Sub-request latencies

Sub-requests	Cycles	Resource abbrev.	Meaning
Areq	2	Abus	Request data/write-back invalidation/ownership
Xdat/Xack/Xown	2	Dbus	Transfer data/acknowledgement/ ownership
Rl2	4	L2	Read data from L2 cache
Rmem/Wmem	8	Mem	Read/Write memory
Rrc/Wrc	8	RC	Read/Write remote cache
BreqI/BdatI/BackI	2	BI(I)	Receive req/dat/ack from BI
BreqO/BdatO/BownO	2	BI(O)	Send req/dat/own to BI
NreqI/NdatI/NackI/NownI	4	NI(I)	Receive req/dat/ack/own from NI
NreqO/NdatO/NackO/NownO	8	NI(O)	Send req/dat/ack/own to NI
Freq/Fdat/Fack/Fown	3	Fwd	Forward messages
PPops		PPcore	Protocol processing operations e.g., PPrecv/PPsend, PPsched, DIRstatus, DIRadd, etc
PPsend/PPrecv	3/8	PPcore	Send/Receive message by PP core
PPsched	4	PPcore	Dispatch/Invoke protocol handler
DIRstatus	5	PPcore	Look up status of a cache block
DIRadd	6	PPcore	Add a node
DIRdel	6	PPcore	Delete a node
DIRrtrv	3	PPcore	Retrieve a node

Table 13: Latencies of sub-requests. The table is produced from Table 2 by replacing the "resources" column with the number of cycles taken by each sub-request. The bus transactions are counted in bus clock cycle. The latencies of data transactions are for the first 8 bytes.