How Much Does Network Contention Affect Distributed Shared Memory Performance?*

Donglai Dai and Dhabaleswar K. Panda
Dept. of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Tel: (614)-292-5199, Fax: (614)-292-2911
E-mail: {dai,panda}@cis.ohio-state.edu

Contact Author: Dhabaleswar K. Panda

Abstract: Many research results in recent years have focused on the design of distributed shared memory (DSM) systems. However, most of these results are centered around either careful design of node controllers or cache coherence protocols. While evaluating these designs, simplified models of networks (constant network latency or average latency based on the network size) are typically used. Such models completely ignore network contention and thus, do not capture the latency of remote memory references accurately. While this approach may seem reasonable for designing better node architectures or coherency protocols, it serves poorly for understanding and solving the network latency bottleneck problem. In order to help network designers to design better networks for DSM systems, in this paper, we focus on two major goals: 1) to estimate the impact of network link contention and network interface contention on the overall performance of DSM applications and 2) to study the impact of critical architectural parameters on network contention. We achieve these goals by evaluating a set of applications from the widely used SPLASH2 suite on a closely integrated processor and network simulator for a CC-NUMA system. The simulator models the processor, cache, and memory access references at instruction level and models the network at a flit transfer level. A series of three network models are proposed to isolate network link contention and network interface contention. For an 8×8 wormhole system, the study indicates that network contention can degrade performance up to 59.8%. Out of this, up to 7.2% is caused by network interface contention alone. The study indicates that network contention becomes dominant for DSM systems using small caches, wide cache line sizes, low degrees of associativity, high processing node speeds, high memory speeds, low network speeds, or small network widths.

Keywords: Parallel architecture, distributed shared memory systems, interconnection networks, wormhole routing, directory-based protocols, cache coherence, and meshes.

*This research is supported in part by NSF Grant MIP-9309627 and NSF Career Award MIP-9502294.

Contents

1 Introduction

Tremendous amount of effort has been spent in recent years in designing and developing distributed shared memory (DSM) multiprocessors. While many hardware and software challenges encountered in small to moderate scale DSM systems have been studied and solved, research remains to be done for machines which can be efficiently scaled beyond a few dozen processors. Long memory latency has been a lead impediment to achieving the full performance potential of such systems.

The fundamental reason for the long memory latency is the physical distribution of the computation and data in DSMs. Besides the speed gap between a processor and memory, two dominant factors in such long latency are: 1) overhead of the cache coherence protocol and 2) delays in the underlying communication network. Previous research in DSMs has mostly concentrated on techniques useful for designing better node architectures, protocol controllers, efficient coherence protocols at different levels in the system or applications, etc. Almost all of these evaluations are based on the assumption of a point-to-point communication network with some fixed (constant or a function of the network diameter) latency. Representative examples include various memory consistency models [11], data pre-fetching [25], data forwarding/updating [21], remote get/put operations [27], integrated or decoupled protocol controllers [8, 10, 12, 31], estimating accuracy vs. performance in simulating DSM systems [2], software DSM systems [23, 1, 19], and explicit communication primitives [29]. Research towards reducing network latency has been largely left to the (interconnection) network community. However, most recently, several papers [18, 9, 31] have reported that network latency is becoming a key architectural bottleneck in designing large scale DSM systems after integrating some of the above techniques.

Under a closer examination, it can be observed that network latency contains two components: minimal communication latency and contention delays. Minimal communication latency is largely defined by the state of technology in circuit speed and switch design. Various network contention delays are caused by limited resources like queue sizes, injection channels, consumption channels, link contention, router contention, etc, in the system. Current and next generation networks promise to exploit performance aggressively by using different kinds of adaptive routing [7, 20, 15, 28, 32] and collective communication primitives [24], in addition to higher speed. The essence behind these communication innovations is to reduce traffic volume and network contention. In order to exploit the potential performance provided by such new networks in DSM systems, some major design issues like directory organization, coherence protocol, structure of protocol engines, etc., need to be re-investigated. Other issues like memory consistency models, data pre-fetching/forwarding/updating and so on, need to be integrated into the resulting designs. Our previous research [3, 4] has applied one such cost-effective collective communication mechanism, i.e., multidestination message passing, to reduce the overhead of invalidations in wormhole routed DSM systems.

Despite the evidence of the above success, some important questions remain unclear about the potential performance gain by employing these advanced network communication mechanisms in DSM systems, given the state-of-the-art network technology. In this paper, we take on this challenge and study the problem of network contention in DSM systems. Specifically, we aim to answer the following five questions:

- For machines with more than a dozen processors, how much does network contention affects the performance of DSM systems when running realistic applications?
- There are many design choices for cache structure on each node. How does each design parameter, i.e., cache size, line size, set associativity, etc., affect the network contention?
- Memory controllers have become increasingly complicated in modern node designs. Without loss of generality, we consider the two most important design parameters: memory response delay and memory bus width. What is the effect of each of them on the network contention?
- As a common trend, processor speeds increase much sooner than the speeds of any other components in a computer system. What impact does the increase of processor speed have on the network contention?
- In contrast to increasing the processor speed, a network with smaller minimum communication latency or wider links can be built at a higher cost. To what extent will such a design reduce network contention?

In order to understand the network contention in DSM systems and to answer the above questions, we first classify the types of potential network contention: limited queue/buffer sizes in a network interface (sending and receiving buffer contention), injection channel contention, consumption channel contention, virtual channel and physical link contention, etc. After reviewing some existing network models used in popular DSM projects, we describe a methodology consisting of three network models (no-contention network, interface contention only network, and detailed network) to isolate and evaluate the effect of different types of network contention occurring at the network interface and within the network respectively. The principle of this methodology can be used to investigate one or more types of network contention in a DSM system.

We perform an in-depth quantitative analysis on network contention for a CC-NUMA architecture using a 2D wormhole [26] mesh interconnection network. We consider a system architecture similar to the FLASH system [12]. Our simulator models the processor, cache, and memory access references at an instruction level and models the network at a flit transfer level. The evaluations are based on four representative applications ported from the SPLASH2 suite [13]. For an 8×8 system, the results show that network contention can degrade overall performance of DSM applications by up to 60%. Out of this, less than 7.2% is caused by network interface contention, implying that the contention inside the network alone is a dominant component of the increase in network latency. This suggests that detailed modeling of network components is essential to estimate accurate performance of DSM applications.

We also evaluate the effect of network contention when a set of major architectural parameters for a DSM system are varied. These parameters include cache size, cache line size, cache set associativity, processing node speed, memory speed, memory bus width, network speed, and network width. It is shown that smaller caches, larger cache lines, lower set associativity, higher processing node speeds, higher memory speeds, lower network speeds and narrower networks can significantly increase the effect of network contention on application performance. These results indicate that variations of these architectural parameters need to be evaluated together with detailed modeling of network components to derive realistic guidelines for designing new generation DSM systems. The study also suggests that there exists great potential to improve the performance of current and future generation DSM systems by developing techniques for reducing network contention.

The rest of the paper is organized as follows. Section 2 characterizes components of network contention in DSM systems. Section 3 provides a methodology for evaluating network contention. Details of simulation environment are discussed in Section 4. Simulation results and discussions are presented in Section 5. Concluding remarks are made in Section 6.

2 Modeling Network Contention in DSMs

Distributed shared memory systems generally fall into three categories: hardware, software, and hybrid. The fundamental difference between these categories is that data coherence is maintained by hardware, software, or a combination of the two. All of them solve the coherence problem based on similar directory-based protocols. In this section, we present our study using a particular hardware cache coherent DSM with a wormhole interconnection network (commonly referred to as CC-NUMA — cache coherent non-uniform memory access multiprocessor) as an example. However, the principle can be applied to other DSMs as well.

2.1 CC-NUMA and its Network Contention

A typical cache coherent DSM machine consists of a set of identical processing nodes connected by an interconnection network. Each node contains a processor with its cache, a portion of globally shared memory, a node controller, and a network interface. Memory is organized as blocks (lines) and each node is the *home node* to the memory blocks it contains. A directory entry is associated with each block to keep track of the current sharing information of the block in the system. The node controller serves as the central control point within a node and enforces the coherence protocol by handling and synchronizing all data movement into or out of the node based on the directory contents. The network interface sends/receives messages into/from the interconnection network via a router. In order to avoid deadlocks, usually two separate (physical or virtual) networks are used, one for request messages and the other for response messages. For more information about request and response messages, and the associated deadlock scenarios, the reader is requested to refer to [11, 12].

Let us consider the transmission of a message from one node to another, as illustrated in Fig. 1. Various resources are required for this to succeed. First, space must be available for the message to be constructed in the sending buffer at the sender's network interface. Such space may not be available because of a number of reasons such as: 1) disparity between the processing rates of the node controller and the network interface, 2) link contention between messages in some part of the network, etc. *Sending buffer contention* is said to occur when a message is blocked due to the lack of such space in the sending buffer.

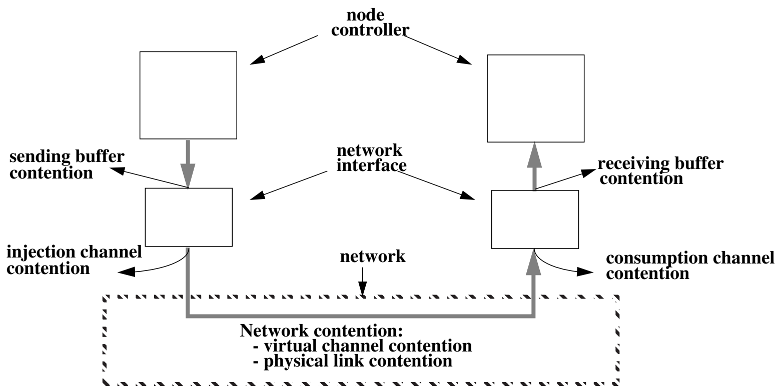

Figure 1: Various types of network contention in a DSM system.

Once the message has been generated, an *injection channel* must be obtained to copy the message flit by flit into the associated router. Due to the limited number of injection channels a network interface has (1 or 2 in current generation DSM machines), a message may have to wait for the injection channels because they are being used for transmitting other messages.

Such a scenario is referred to as *injection channel contention*.

The entire message moves flit by flit in a pipelined fashion from the source to its destination after it has been injected into the network. Flits hold links between neighboring routers and the associated buffers in the routers. Handshaking flow control signals are used along each link to advance the flits. As the header flit of the message passes through each router along its path, the router must make a routing decision and reserve the corresponding outgoing link for the remaining flits. If the outgoing link is not available, the message will be blocked in the network. This is known as *link contention* (or *virtual channel contention* if multiple virtual channels share a single physical link).

In case multiple virtual channels share a physical link, if more than one virtual channel has flits ready for transfer, the physical link is multiplexed among the channels (*demand-multiplexing*). This effectively reduces the moving rate of each message involved. We refer to this scenario as *physical link contention*.

After the header flit of the message arrives at the router associated with the destination, a consumption channel in the network interface of the destination node must be obtained to copy the message flit by flit from the router into the network interface. Again, due to the limited number of consumption channels a network interface has (1 or 2 in current generation DSM machines), a message may be blocked because the consumption channels are being used by other messages. This is known as *consumption channel contention*. Consumption channel contention can cause considerably increase in the latency of a message under hot-spot traffic.

Finally, a receiving buffer in the destination's network interface must be obtained to assemble the entire message before it can be delivered to the node controller. If there is no receiving buffer available, the message will again be blocked while holding the consumption channel and other links/virtual channels in the network. This is known as *receiving buffer contention*.

It is clear that the communication delay of a message can be significantly affected by any of the above types of contention. However, due to the temporal nature of network contention, it is usually very difficult to measure its effect on a real DSM machine. Even if it were possible to collect data on the network contention in a real DSM machine, it is impossible to compare the results from two different machines. This is because of (i) the differences in theirs design and implementation, and (ii) the fact that different machines have different networks that affect the types and mixes of contention even when everything else is the same.

Some analytical contention models have been used before. These models often require parameters like average number of messages generated per node, average length of a message, average distance a message travels, etc., to predict the corresponding network latency. From a DSM system perspective, a major flaw of such an analytical model is that it does not consider the crucial dependency between computation and communication in an application. This leaves simulation as the only plausible approach for estimating the effect of various types of network contention.

2.2 Existing Network Models Used in Popular DSM Projects

Due to the complexity of modeling a network, most of the research work in DSMs has ignored network contention partially or entirely. The original WWT [14] is a parallel DSM simulator developed by the Wisconsin Wind Tunnel research group. In this simulator, a constant network latency of 100 processor cycles is assigned for every message independent of the length of the message, the distance traveled, and other traffic in the network. A DSM simulator used in the Stanford FLASH [16] research group models the network with slightly higher accuracy—network interfaces are modeled in this simulator. The network latency of every message is calculated based on the length of the message and half of the diameter of the network. The Typhoon [30, 31] simulator from the same Wisconsin Wind Tunnel group models the network interface and assigns 100 processor cycles to every message as the delay of the message inside the network. None of these DSM simulators has modeled the channel contention or physical link contention. Our results in Section 5 show that these types of contention inside the network dominate the overall network contention.

When network latency is not a performance bottleneck, such an approach is valid and it has helped the community discover many valuable techniques and mechanisms for DSM node design. However, when the network becomes a bottleneck, this approach fails to offer the designers insight into the reasons for this bottleneck: network contention is one of the most important of these reasons. Isolating the impact of the various types of network contention helps an architect identify the design choices for improving network performance, such as, routing adaptivity, network support for collective communication, etc.

In next section, we propose some modifications to these network models which can provide us useful information about the various types of network contention.

3 Methodology for Evaluating Network Contention

In this section, we describe our methodology for isolating and evaluating the impact of various types of network contention that occur in a DSM system. The basic idea is to construct a series of network models. Each of these simulates the network at an increasing level of detail and is driven by the same DSM node and memory simulator while collecting the statistical data using exactly the same input. Comparing the predicted performance metrics of any two such DSM simulators, the difference tells us the effect of types of contention which are modeled by the more sophisticated simulator.

For this study, our main goal is to evaluate the overall performance impact of various types of network contention. Secondarily, we want to examine the performance impact of only those kinds of contention that occur inside the network interfaces. Once the overall performance impact of various types of network contention and the impact of network interface contention have been determined, the impact of the kinds of contention occurring inside the network alone can be inferred. To achieve these goals, we construct a series of three network models as described below.

3.1 No-Contention Network Model (NCM)

This model is an enhancement of the network model used in the original WWT simulator [14]. It assumes: 1) infinite number of sending buffers, 2) infinite number of injection channels, 3) infinite number of consumption channels, 4) infinite number of receiving buffers, and 5) no traffic interference inside the network. These assumptions guarantee that no network contention can ever occur during any communication. Unlike the WWT, in our model each message is delayed by exactly the total time for construction, network propagation, and consumption. The construction time and consumption time vary depending on the length of the message. The network propagation time is a function of both the length and the distance traveled by the message.

3.2 Interface Only Network Model (NIM)

The NIM model is an enhancement of the network model used in the FLASH project [16]. The NIM model simulates detailed management of the limited number of sending buffers, receiving buffers, injection channels, and consumption channels. The NIM model still assumes no traffic interference inside the network. Such a model captures the types of contention occurring within a network interface. For every message, if there is no contention within a network interface, the message is delayed in this model by exactly the same amount of time as in the NCM model. It is clear that any difference in the predicted performance between this model and the NCM model is caused by contention occurring inside the network interfaces during application execution.

3.3 Detailed Network Model (DNM)

The DNM model simulates every possible type of contention in the network. It simulates the detailed management of transmission links and intermediate switches (at a flit level), in additional to modeling the network interface. The flow control mechanism used by the network must be modeled as closely as possible. For example, the handshaking signals in an asynchronous network must be modeled. For every message that is not involved in contention inside the network, the message is delayed by exactly the same amount of time as in the NIM model. Such a model captures all types of contention within the network interfaces and every part of network. It is clear that any difference in the predicted performance between this model and the NCM model is caused by contention occurring in the entire network (including the network interfaces). Any difference in the predicted performance between this model and the NIM model is caused by contention occurring inside network alone (excluding the network interfaces).

The relations among the above three network models is shown in Fig. 2. It is to be noted that this methodology is general and applicable for evaluating the impact of network contention in any type of DSM system: hardware, software, and hybrid. In following section, we describe in much more detail, the specific DSM simulation implementation which we used for collecting all the results presented in this paper.

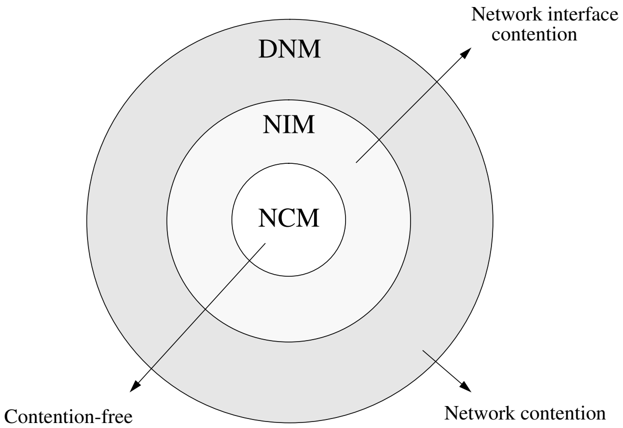

Figure 2: Relations among the network models (NCM, NIM, DNM).

4 Simulation Environment

In order to apply above methodology for evaluating the impact of network contention in a DSM system, we simulate a machine with an architecture and coherence protocol similar to the FLASH machine [12] but with some important differences described later. In this section, we first describe the important details of the simulated processing node and each of the network models that are used, and then briefly discuss the applications in our experiments. We describe a *baseline configuration* whose architectural parameters have the set of values defined in this section. In next section, when we study the effect of each important architectural parameter on network contention, we will specify the differences from this baseline configuration.

4.1 Simulated Architecture

The hardware cache coherent multiprocessor we simulate is illustrated in Fig. 3. The machine has 64 processing nodes. The processor in each processing node is assumed to be a 200 MHz single-issue microprocessor with a perfect instruction cache and a 128 KB 2-way set associative data cache with a line size of 16 bytes. The cache is assumed to operate in dual-port mode using write-back and write-allocate policies. The instruction latencies, issue rules, and memory interface are modeled based on the DLX design [17]. The memory bus is assumed to be 8 bytes wide. On a memory block access, the first word of the block is returned in 30 processor cycles (150 ns). The successive words in the block follow in a pipelining fashion. The machine is assumed to be using a full-mapped, invalidation-based, three-state directory coherence protocol [12, 10]. The node simulator models the internal structures, and the queuing and contention at the node controller, main memory, and cache. The sizes of the queues between the internal structures and the appropriate actions taken if a queue is full are shown in Table 1. Following the assumptions in [18], the node controller incurs a small fixed occupancy for purely generating a request message into the network. The same occupancy is also incurred for receiving a response message from the network. The node controller at the home node of a remote request/response incurs a higher fixed occupancy, because data (in most cases) and directory information must be retrieved and manipulated. In such a case, the memory access is assumed to proceed in parallel with the node controller. The node controller also incurs extra occupancy per invalidation sent. In a scenario where the processor of a node has a load/store miss to a clean block whose home node is the node itself, we assume the memory module can be pipelined with the cache. Thus the retrieve of the block, the filling of the cache line, and the manipulation of the directory information are overlapped. Table 2 shows the memory hierarchy parameters and node controller occupancy delays used in our simulated architecture.

Table 1: Default sizes of various queues in a simulated node controller.

Queue	Size	Action if full
Incoming network queue	8 messages	Messages blocked in the network
Outgoing network queue	8 messages	Node controller stalls
Memory request queue	1 request	Node controller stalls
Incoming processor queue	8 messages	Processor stalls
Outgoing processor queue	1 messages	Node controller stalls

The major difference between our simulated architecture and the FLASH architecture are as follows: sequential memory consistency, instead of the release memory consistency used in the

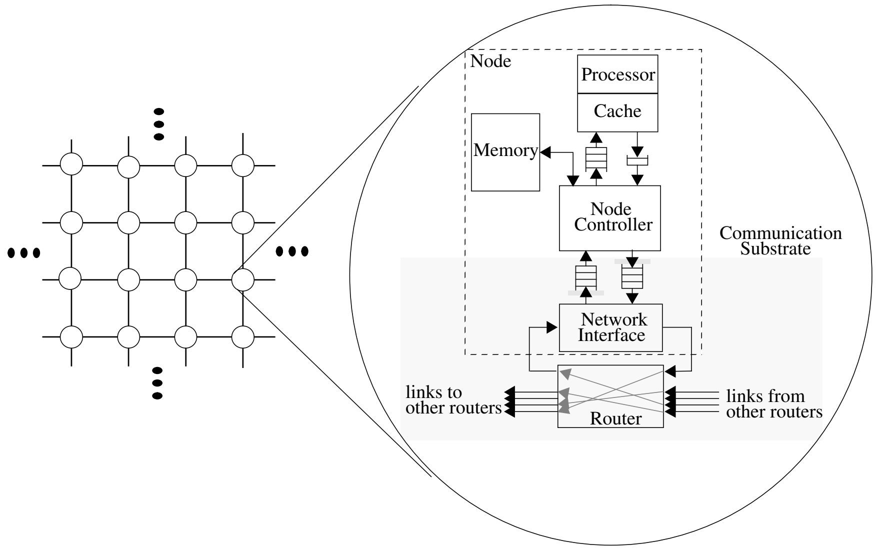

Figure 3: Simulated node architecture and interconnection network.

FLASH architecture, is enforced by the coherence protocol. Therefore, at most one outstanding request can be issued by a processor. A load/store miss stalls the processor until the first word of data is returned.

4.2 Implementing Network Models

We assume the processing nodes in the machine are connected with a pair of virtual networks sharing a physical 2D 8×8 wormhole network [26]. The physical network is assumed to operate at a frequency of 200 MHz. The method adopted for modeling the network by each of the network simulators is described in detail in following three subsections.

4.2.1 No-Contention Network Model (NCM)

The NCM network simulator is quite simple. It takes a message from the node controller at the source node and informs the node controller at the destination node that a message has arrived after a time delay equal to the calculated network latency of the message. The network latency of a message is calculated as:

$$latency = T_{outgoing} + (T_{rout} + T_{link}) * distance + (T_{sw} + T_{link}) * length + T_{incoming}$$

where *distance* is the number of hops from the source to the destination, and *length* is the length of the message in flits. $T_{outgoing}$ is the time delay for a message to be constructed by the network interface. T_{rout} is the time needed for a router to make a routing decision based on its knowledge about the network and the requirement of a message contained in the header flit. T_{link} is the time for any flit of a message to propagate over a single physical link. T_{sw} is the time for a non-header flit of a message to pass through a router. The parameters used in our simulation for these network delays are listed in Table 2. However, for a simple three-state directory-based coherence protocol, such an implementation will cause memory inconsistency due to the non-FIFO delivery between a source and destination pair. Following the method described in [31], we modeled additional end-to-end flow control to guarantee the FIFO property of message delivery between each pair of nodes. It is to be noted that the incoming/outgoing network queue size at a network interface listed in Table 1 is ignored in the NCM network simulator. (i.e., these queues are assumed to be of infinite length as described in section 3.1.)

4.2.2 Interface Only Network Model (NIM)

This simulator takes into account detailed modeling and timing delays associated with structures such as the sending buffers, receiving buffers, injection channels, and consumption channels within the network interface. For a node to send a message, a space in the sending buffer must be reserved for $T_{outgoing}$ time to construct the message. Once the message is constructed, if there is an injection channel available, it is reserved for $(T_{link} + T_{sw}) * length$ amount of time to pump the entire message into the network. At exactly $(T_{link} + T_{rout}) * distance$ time after the header flit of the message is injected into the network at the network interface of the source node, a consumption channel and a space in the receiving buffer at the network interface of the destination node is reserved for $(T_{link} + T_{sw}) * length$ time. Once the message has been consumed entirely into the receiving buffer, the node controller at the receiving node is informed about its arrival after $T_{incoming}$ amount of time. In case any of the above reservations can not succeed (because of other messages to the network), the message can not move forward and continues to hold the resources it has acquired. It is to be noted that such a network simulator guarantees the FIFO property of message delivery between each pair of nodes if there is only one injection and one consumption channel at each network interface.

4.2.3 Detailed Network Model (DNM)

The implementation of the DNM network simulator is the most sophisticated one. It models the internal structures such as data links, channel buffers, signal lines, etc., within and between the routers, in addition to the network interface. It accurately simulates the mechanisms for wormhole switching such as: distributed routing, book-keeping on channel status, and flit-level asynchronous ready/empty handshaking. For detailed information about wormhole routing, the reader is requested to refer to [5, 6, 26]. In this simulator, when a message is first injected into the network, the header flit of the message reserves the channels in the routers for the remaining flits of the message, while moving forward along its path from the source to the destination. When the header moves into a router, a delay of T_{rout} is incurred for the routing decision. If the (desired) outgoing channel (decided by the routing scheme) is available, the header flit reserves that channel and reaches the next router after a time elapse of T_{link}. Each non-header flit of the message can move forward only when its immediate predecessor has already moved as detected using the asynchronous handshaking signals. Every non-header flit incurs a latency of T_{sw} at every router and T_{link} at every link on the path to the message destination. The tail flit of the message releases the previous channel when it leaves each router. When two virtual channels over the same physical link are both transmitting messages, the physical link is demand-multiplexed, resulting in slowdown of both messages. It is to be noted that such a network simulator guarantees the FIFO property of message delivery between each pair of nodes because of the simulated dimension order routing scheme.

Table 2 summarizes the default values for all important parameters used in the simulation. Table 3 shows the contention-free latency of a typical memory miss to a neighboring home node. To illustrate how the parameters in Table 2 get translated into the times in Table 3, we show the time breakdown of contention-free remote read miss to a neighboring home node in Table 4. These memory latencies are comparable to the latencies presented in [22, 16], reflecting the accuracy of our simulations.

4.3 Applications

We use four applications — Barnes, LU, Radix, and Water in our simulation evaluation. All these are real applications and challenging computational kernels ported from the widely used SPLASH2 suite. Table 5 gives the basic traffic pattern and input sizes for the various applications. These applications are well-structured and heavily optimized for DSM machines. For more information about the applications, the reader is requested to refer to [13].

Barnes is an application representative of the class of hierarchical N-body methods, used in astrophysics, electrostatics, and plasma physics, among others. In this application, a globally shared space-motion-mass tree (its primary data structure) is continually reconstructed and repartitioned at every time-step. Particles are assigned to each processor and accessed through pointer arrays. The communication pattern in Barnes is hierarchical and irregular.

LU is the computation kernel of lower and upper matrix decomposition. It represents a wide range of dense linear algebra applications. The 2D $n \times n$ input matrix of double precision numbers is converted into to a 4D $N \times N \times B \times B$ matrix for purposes of reducing communication and load balancing. The elements assigned to the same processor are allocated contiguously to improve spatial locality. The communication pattern is one-to-many and largely regular.

Table 2: Default system parameters used in simulation.

Memory Hierarchy Parameters	
Processor frequency	200MHz
Cache access	1 cycle
Cache line size (L)	16 bytes
Cache set associativity	2
Cache size per node	128 Kbytes
Memory word width (W)	8 bytes/cycle
Memory response delay	30 cycles
Cache block fill time	30+L/W cycles
Memory block access time	30+L/W cycles
Node Controller Occupancy	
Directory check	7 cycles
Directory check and update	14 cycles
Each invalidation request	12 cycles
Message forward	3 cycles
Network Interface Parameters	
Outgoing message startup ($T_{outgoing}$)	15 cycles
Incoming message dispatch ($T_{incoming}$)	8 cycles
Control message size	6 bytes
Data message size	22 bytes
Injection channels per node	1
Consumption channels per node	1
Network Parameters	
Network frequency	200MHz
Channel width / Flit size	2 bytes
Link Propagation (T_{link})	1 network cycle
Router switch delay (T_{sw})	1 network cycle
Routing time (T_{rout})	4 network cycles
Physical network	1
Virtual networks	2
Virtual channels per virtual network	1

Table 3: Contention-free latencies of typical memory miss to neighboring nodes in 5ns cycles.

Memory Access	Home Location	Directory State	# Inv or CpyBack	Latency
Load	local	not cached	0	33
	remote	not cached	0	151
	remote (2-party)	home dirty	1	166
	remote (3-party)	remote dirty	1	335
Store	local	not cached	0	33
	local	remote shared	1	140
	remote	remote shared	0	161
	remote (2-party)	home shared	1	176
	remote (3-party)	remote shared	1	304
	remote (3-party)	remote dirty	1	335

Table 4: Latency break down of a clean read-miss to neighboring node in 5ns cycles.

Sub-operation	Latency
Cache-miss to request in the network	19
Request transmit time (6 bytes)	14
Request at the home node to output header transmit	45
Data return in network (22 bytes)	30
Response arrival to beginning of cache fill	11
Cache fill	32
Total	151

Table 5: Applications and default problem sizes.

Application	Description	Communication Pattern	Problem Size
Barnes	Hierarchical N-body galaxy simulation	irregular, hierarchical	2048 particles, $\theta = 1.0$, 4 time steps
LU	Blocked dense linear algebra	one-to-many regular	256×256 doubles, 8×8 blocks,
Radix	Integer radix sort	all-to-all, irregular	262144 keys, 512 radix, max 524288
Water	N-body molecular dynamics simulation	near neighbor irregular	512 molecules, 4 time steps

Radix is the integer radix sort kernel, a major task in data base applications. It iteratively creates local histograms by permuting keys and by merging them into a global histogram. The permutation phase requires, depending on the key values, all-to-all irregular communication (in the worst case).

Water is an N-body molecular dynamics application. It evaluates forces and potentials in a system of water molecules over a period of time. The main data structure is an array of the molecules in the system. It adopts a predictor-corrector method ($O(n^2)$ algorithm) to compute the forces and potentials in each time-step. Irregular communication occurs when the cell list is updated which in turn is caused by the movement of molecules into and out of cells.

5 Simulation Results and Discussion

We now present the results of our simulations for the applications described above. In Section 5.1 we evaluate the contention within the network interface and inside the network. Sections 5.2 and 5.3 examine the impact of alternate designs of cache and memory on network contention. Section 5.4 investigates the impact of processing node speed. Finally, we evaluate the impact of different designs of the network itself in Section 5.5. For convenience of reference, we identify the three DSM simulators described in the previous section by the network model each one uses. All results are reported using the total execution time, normalized to that time predicted by the NCM simulator for the same application. Each such time is further broken down into four categories: processor computation time (Computation), read stall time (Read), write stall time (Write), and synchronization stall time (Sync).

5.1 Network Contention in Baseline System

Figure 4 shows the execution time of each application using the the NCM, NIM, and DNM simulator, respectively, in the DSM system using the baseline configuration. As expected, network contention slows down every application. However, the exact effect depends on application, ranging from 5.3% (in Water) to 16.8% (in Radix). Radix incurs such high network contention because of the all-to-all irregular communication occurring during its histogram merge phase. It is clear that communication contention enlarges the stalling time of each category in every application. The results show that contention within the network interface alone only accounts for a small portion of total communication contention, less than 0.84% in our experiments. In other words, contention inside the network is the dominant cause of higher network latency. This observation remains true for all our experiments.

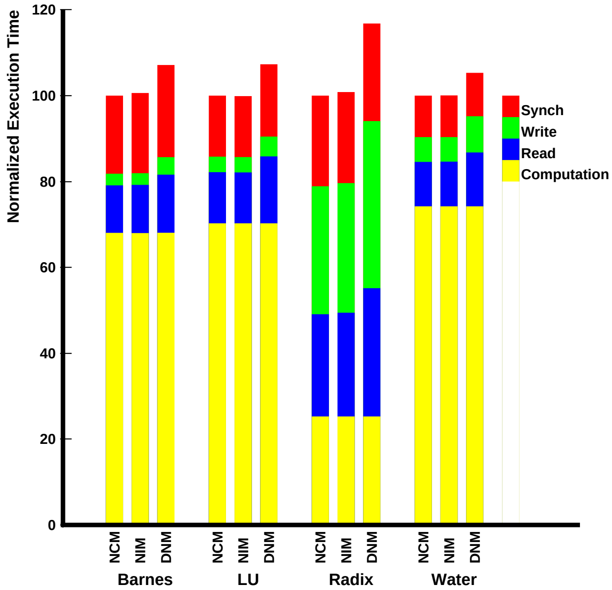

Figure 4: Breakdown of application overall execution time on the baseline system.

5.2 Impact of Cache Design on Network Contention

Let us now examine the impact of alternate design choices of processing node cache on the network contention. Sections 5.2.1, 5.2.2, and 5.2.3 respectively look at this issue by changing the cache size, cache line size, and set associativity.

5.2.1 Effect of Cache Size per Processing Node

Figure 5 depicts the execution time breakdowns for the experiments with 32KB, 64KB, 128KB, and 256KB cache respectively. Two observations are noteworthy. First, as the cache size increases, the network contention in each application reduces to a constant amount (approximately). This is because when cache is large enough, the largest working set for an application can fit into the cache completely. Our results show that even in such a case the slowdown caused by communication contention ranges from 4.8% (in Water) to 16.8% (in Radix). This also shows that our baseline configuration gives us a rather conservative estimate of the network contention[1]. Second, when the cache size decreases, the network contention becomes more significant. The primary reason for this is the extra traffic in the network caused by cache capacity misses. This is true even when the smallest working set of an application still fits in the cache.

5.2.2 Effect of Cache Line Size

Figure 6 shows the execution times for the experiments with cache line size of 16, 32, 64, and 128 bytes respectively in the same 128 KB cache. As the cache line grows larger, the network contention in all application shows the bath-tub behavior, i.e., first decreasing to a pollution point and then increasing significantly. The surprising discovery from these results is that the pollution point is relatively small, around 32 or 64 bytes per line, for three out of the four applications. The reason for this is as follows. When the cache line increases, the reply messages which contain data grow longer. The length increase in this type of message causes increase in the time required for transmitting them, thus longer time is spent on the links and routers, causing more network contention. Such an effect not only slows down a message of this type, but also increases the latency of messages of other types. Similar to a uniprocessor system, longer cache line exploits spatial locality existing in an application at the cost of reducing the benefits of temporal locality. The effect of data pre-fetching because of longer cache line can reduce the number of cache misses. However, the increase in cache line size does not reduce

[1]This is because the baseline configuration assumes a cache size large enough to hold the largest working set of any of the applications we used. This represents a system that generates very little cache-capacity miss traffic and therefore *underestimates* network contention.

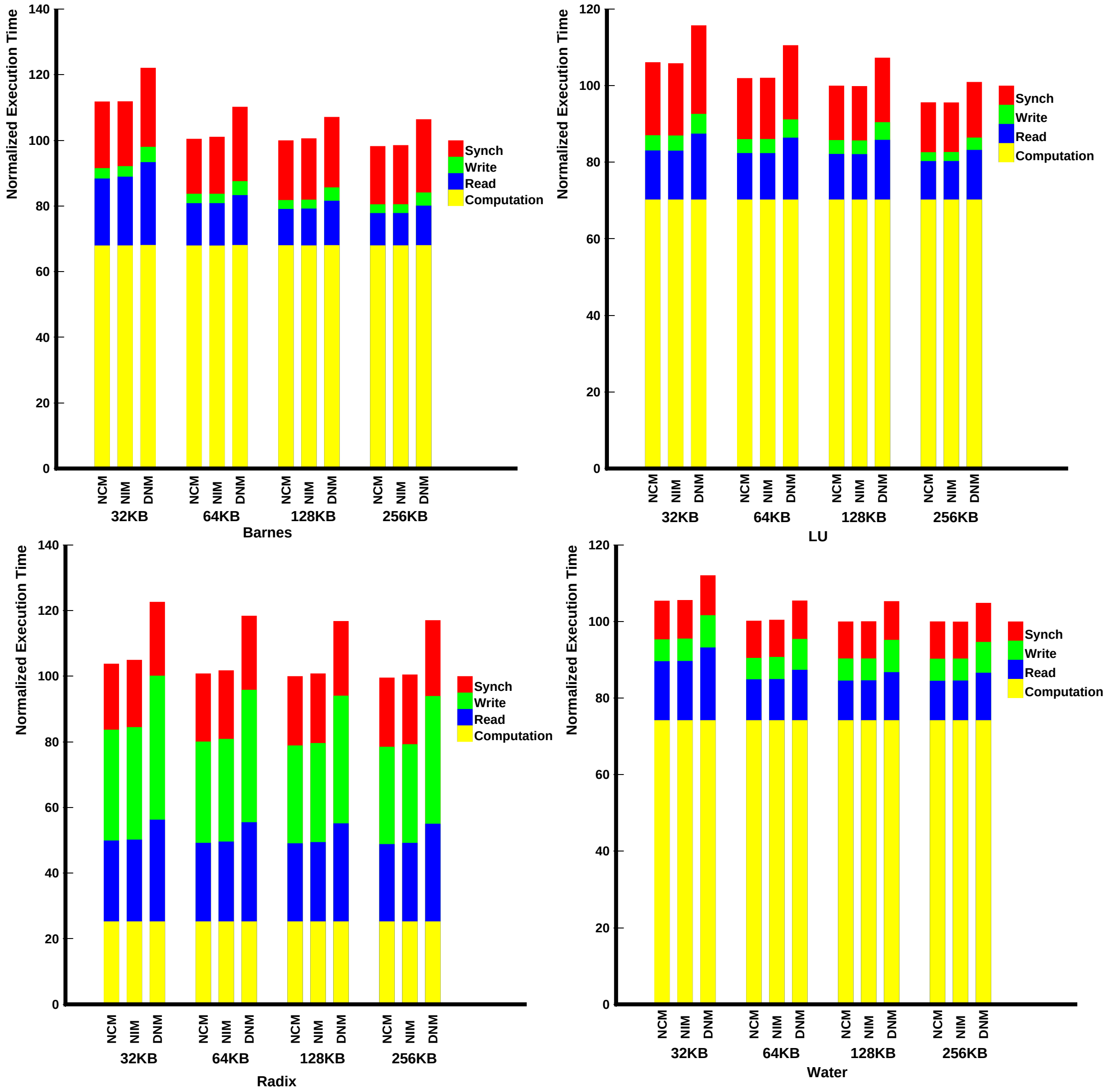

Figure 5: Effect of cache size per node on network contention.

the number of messages by proportion after when the lines size is long enough. In DSM, this leads to an increase in overall volume of network traffic because of unnecessary data pre-fetching results in more network contention and the network latency increases explosively. Such an effect can be observed by examining the difference between the results of NCM and DNM simulators for Radix: the proportion of the network latency in the overall execution time grows from 19.6% for a 32 byte cache line, to 34.7% for a 64 byte cache line, and to 59.8% for a 128 byte cache line. The reason that Radix shows this effect earlier than other applications is because the communication to computation ratio is higher in this application because of the histogram merging performed.

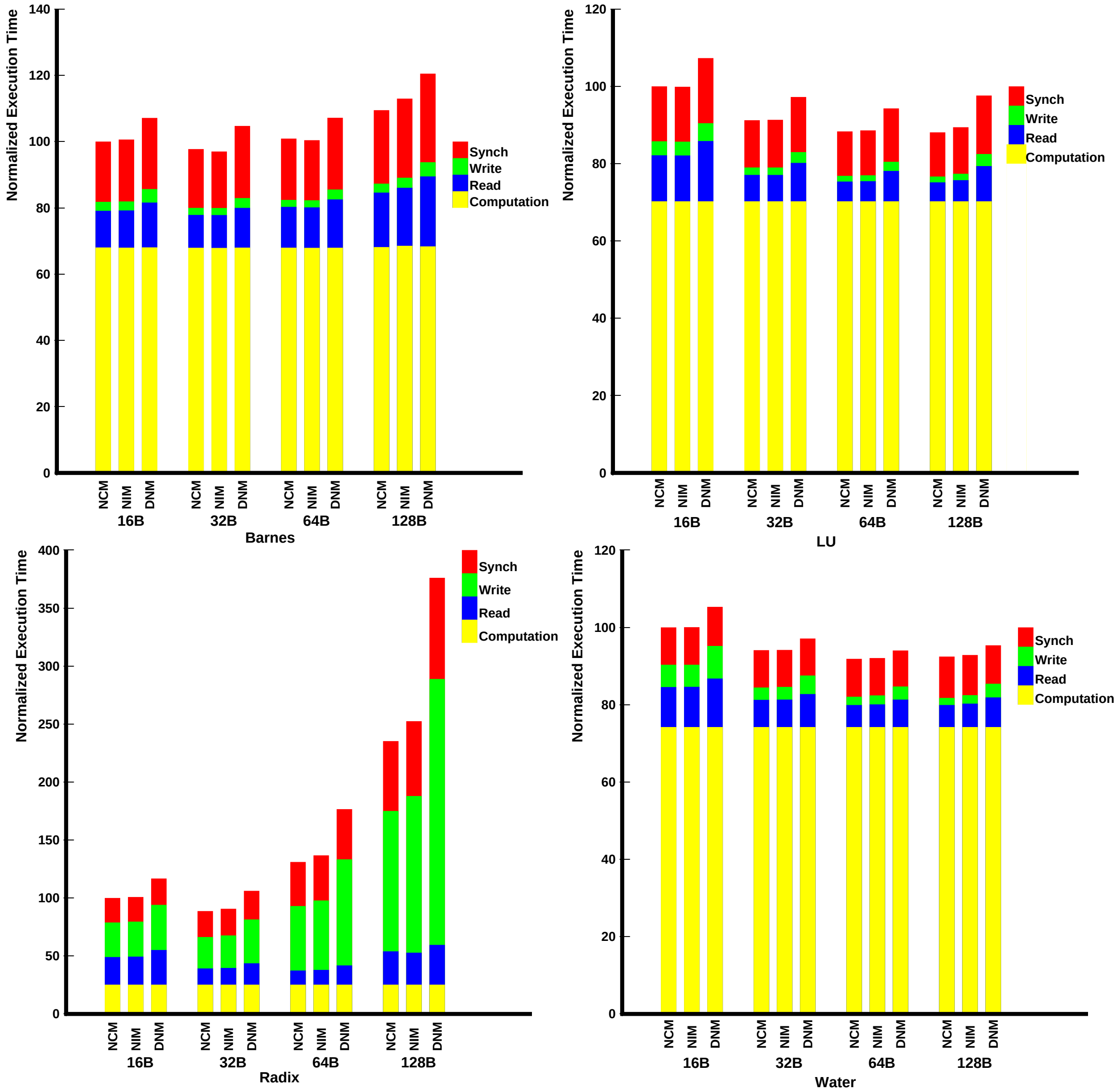

Figure 6: Effect of cache line size on network contention.

5.2.3 Effect of Cache Associativity

The execution times of the applications when cache is organized as direct-mapped, 2-way, and 4-way associative are shown in Fig. 7. When the set associativity is equal to or greater than 2, the proportion of network contention remains almost constant in each application. When the cache is organized as direct-mapped, the proportion of network contention increases in all applications because the cache mapping conflicts increase the network traffic. Our experiments show that a 9.1% (Barnes), 12.4% (LU), 21.1% (Radix), 6.6% (Water) performance degradation occurs for a direct-mapped cache of size 128KB.

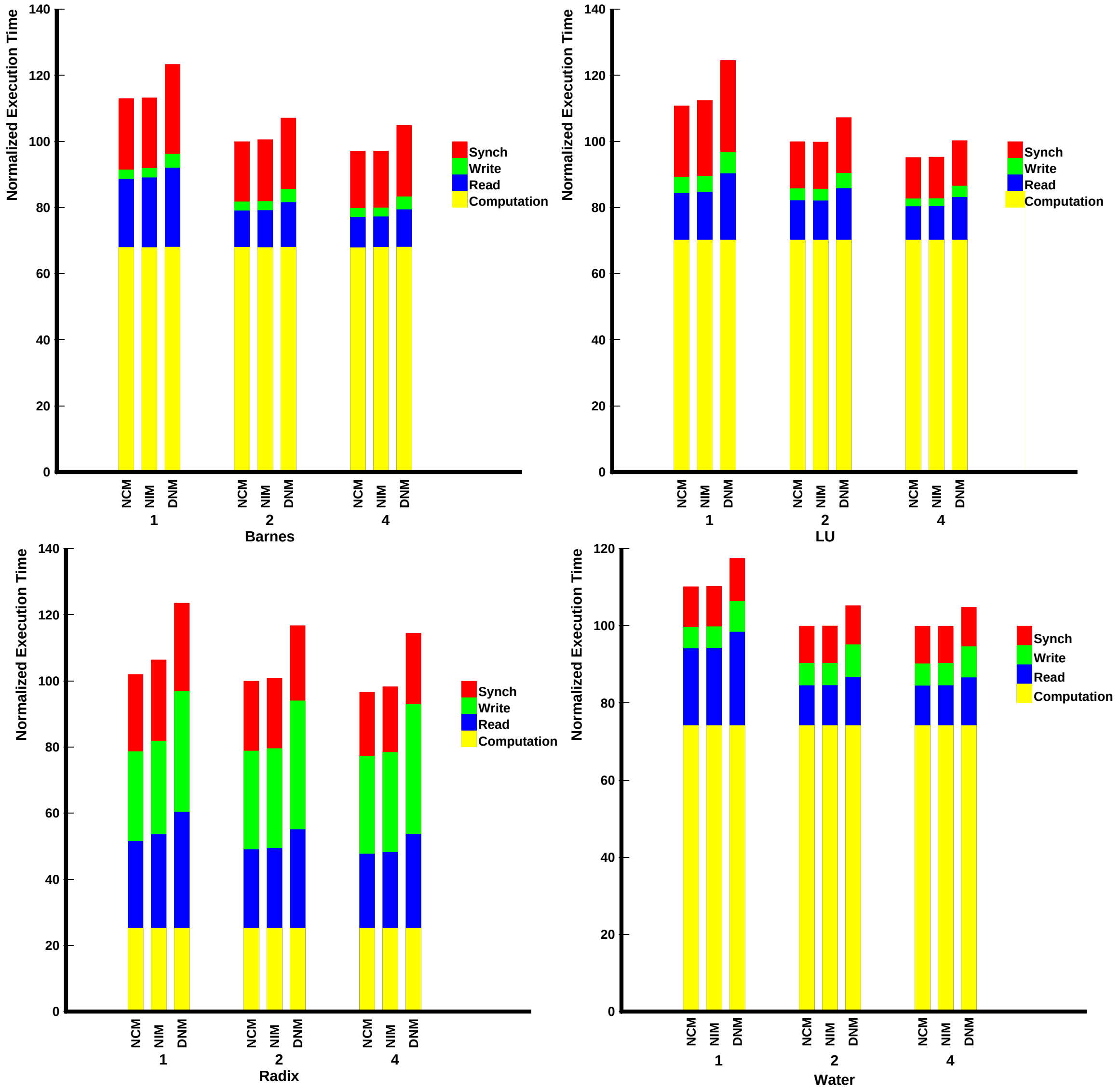

Figure 7: Effect of cache associativity on network contention.

In summary, network contention is sensitive to the design choices of cache in a processing node. This is especially true for cache line size selection. Overall, alternative cache designs other than our baseline configuration tend to increase the impact of network contention in a DSM system.

5.3 Impact of Memory Design on Network Contention

Let us now evaluate the impact of alternative design choices of the memory module in the processing node on the network contention. We focus on the two primary memory metrics: memory response time (in Section 5.3.1) and memory bus width (in Section 5.3.2).

5.3.1 Effect of Memory Response Time

Memory response time is the time interval from the issue of memory access command to the time when the first word in a memory block is available for use. The experimental results shown in Fig. 8 are for different response time of 20, 30, 40 processor cycles. As much as 7.4% performance degradation occurs in in Barnes, 7.9% in LU, 22.2% in Radix, and 3.1% in Water for a memory response time of 20 processor clock cycles.

As the memory responds faster, the network contention increases slightly in all applications because the request and reply messages are generated in a shorter time, resulting in greater traffic congestion.

5.3.2 Effect of Memory Bus Width

Memory bus width decides the amount of time needed for finishing an access to a memory block after the first word is available. The experimental results shown in Fig. 9 are for different memory bus widths of 4, 8, and 16 bytes. In these experiments, we use a block (line) size of 32 bytes. As the memory bus becomes wider, the network contention barely changes in each application because of the assumption of memory pipelining.

Generally speaking, when the memory module becomes faster in a node, the network contention increases. Between memory response time and memory bus width, the former has a stronger impact on the network contention.

5.4 Effect of Node Speed

In this section, we examine the impact of node speed on the network contention. Our system is assumed to use an integrated node controller. Therefore, when the node speed increases,

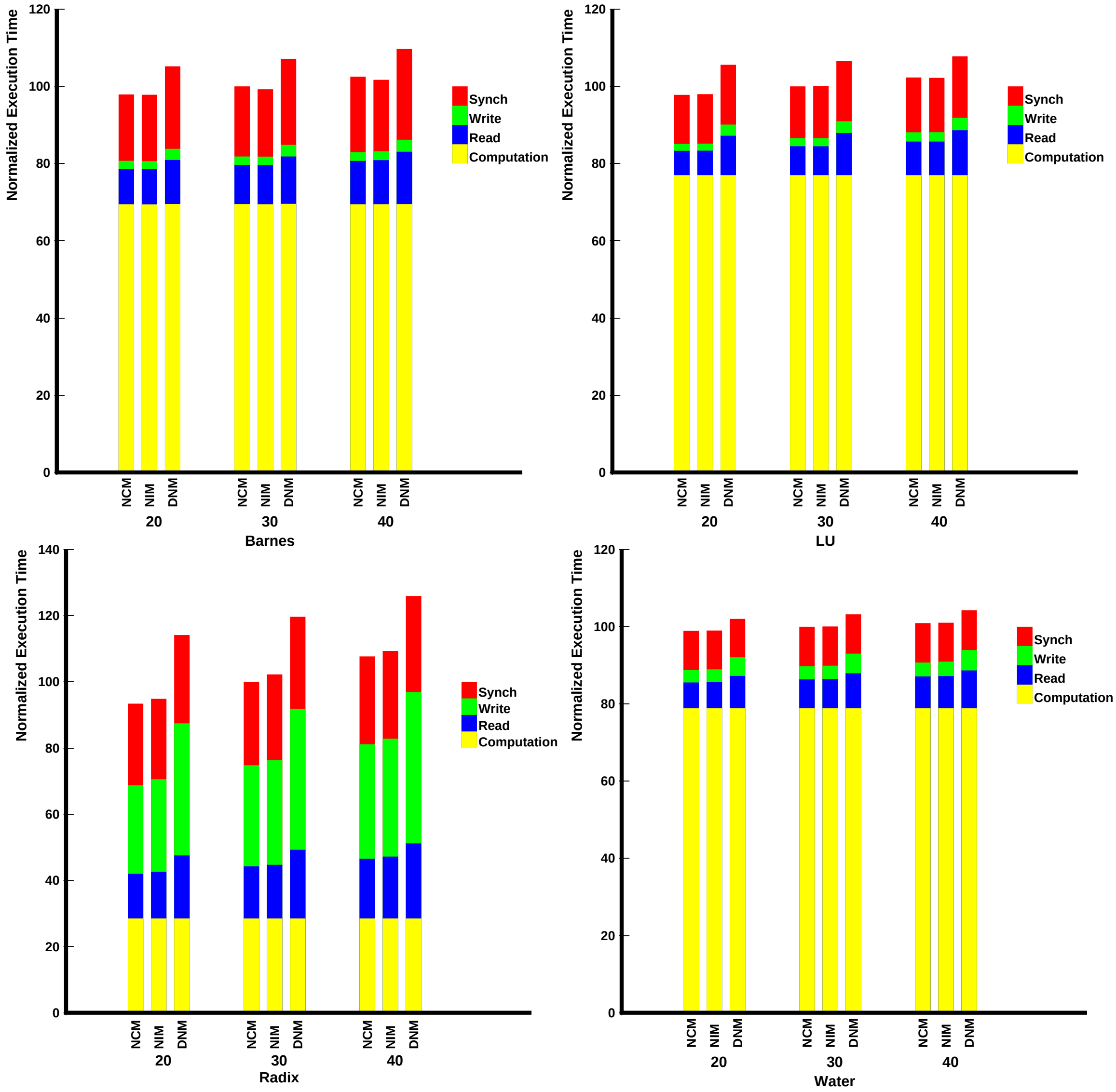

Figure 8: Effect of memory response time on network contention.

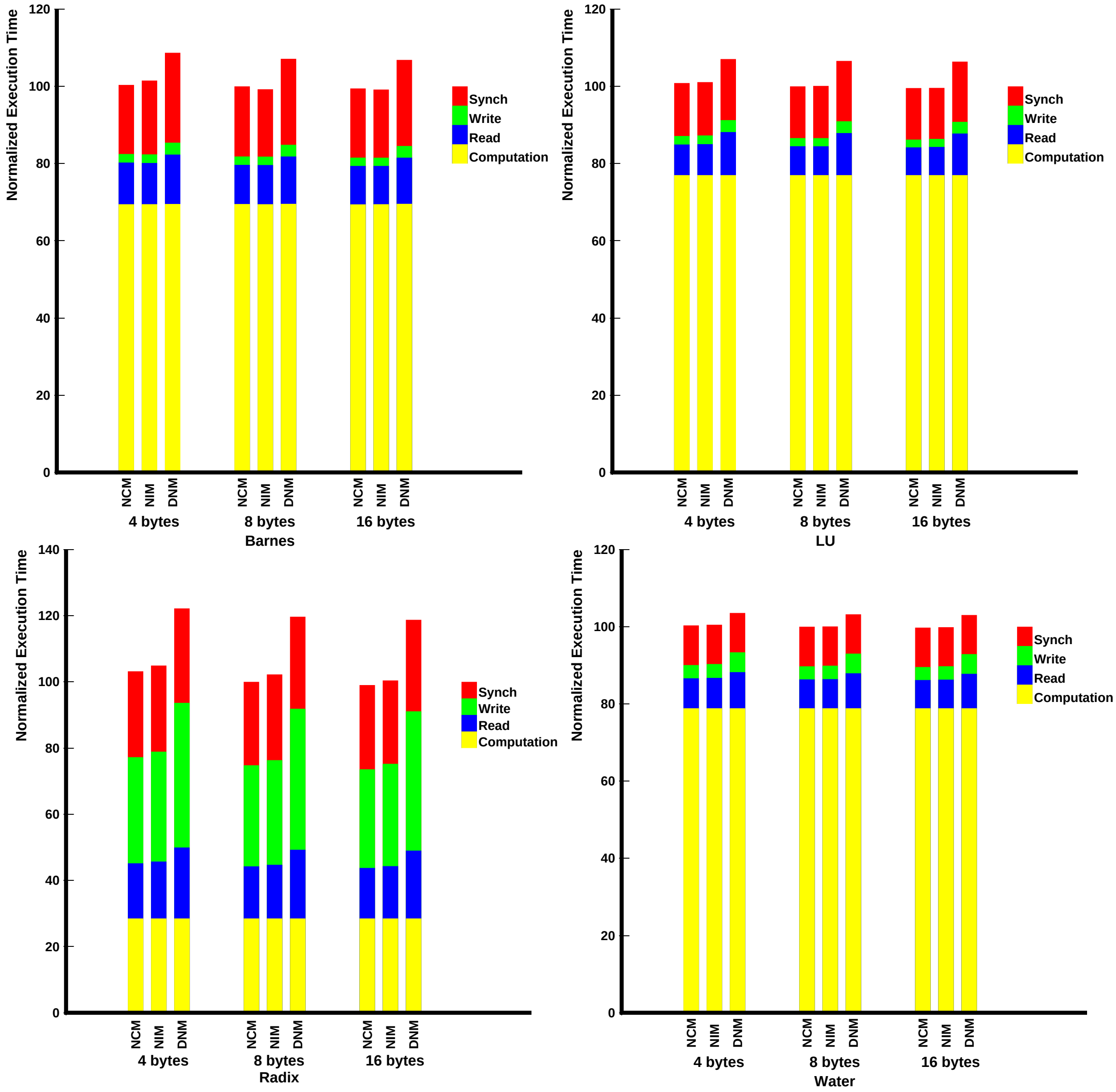

Figure 9: Effect of memory bus width on network contention.

all parts in a node (processor, cache, memory, node controller, and network interface) are faster. The results in Fig. 10 show that the network contention increases considerably in most application. Specifically, the performance degradation because of network contention worsens from 5.3% to 11.3% in Barnes, 5.3% to 7.9% in LU, 16.9% to 22.2% in Radix, 3.1% to 3.2% in Water, as the node speed increases from 100MHz to 400MHz. The reason for this is, again, that the network is stressed in trying to cope with more traffic generated by the faster processor in a given amount of time.

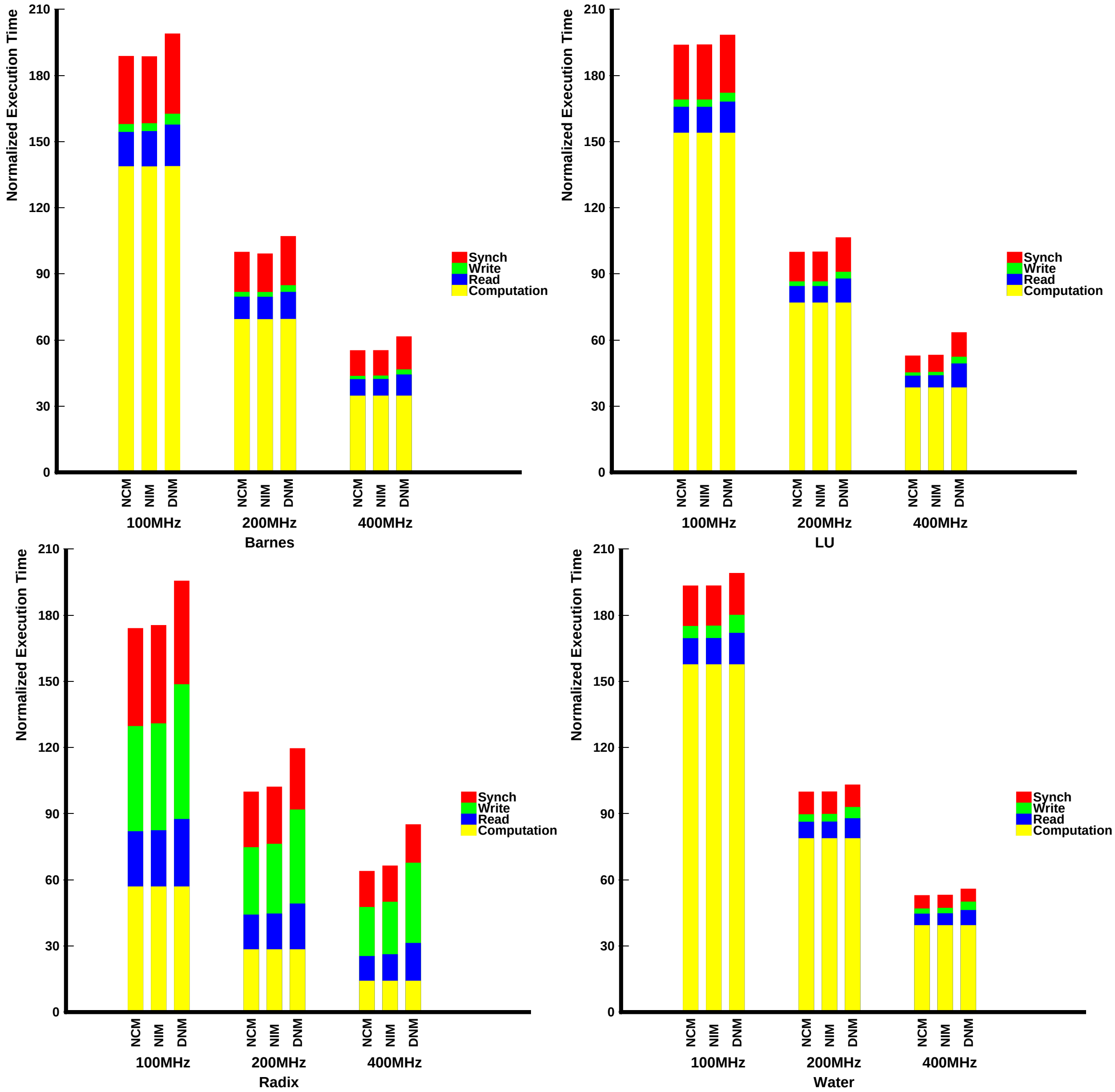

Figure 10: Effect of node speed on network contention.

5.5 Impact of Network Design on Network Contention

In this section, we examine the impact on the network contention with different network designs.

5.5.1 Effect of Network Speed

Not surprisingly, from the results in Fig. 11, the network contention is significantly worse in a slower network (e.g., LANs in DSMs based on networks of workstations) than that in a faster network. Our results show that performance degradation changes from 4.8% to 11.4% in Barnes, 2.1% to 19.9% in LU, 23.7% to 32.2% in Radix, 2.7% to 5.6% in Water as the network speed changes from 400MHz to 100MHz.

5.5.2 Effect of Network Flit Width

As expected, from the results in Fig. 12, the network contention is significantly worse in a network with narrow links than in a network with wider links. For example, the performance degradation increases from 7.2% to 9.5% in Barnes, 2.8% to 20.1% in LU, 12.9% to 29.0% in Radix, and 4.4% to 6.9% in Water, as the network width reduces from 32 to 8 bits.

In general, a more powerful network does reduce the network contention. However, considering the commonly used narrow links and slower networks, from a DSM perspective, network contention is an important factor for the large network latency in DSM systems.

6 Conclusion

In this paper, we have studied the impact of network contention on the performance of four representative applications (from the SPLASH2 benchmark suite) on a DSM system. By implementing three separate network models, we have isolated the impact of the contention in the network interface, the contention in the network alone, and the overall network contention. We ran the four representative applications on these three network models, and studied the impact of the various types of contention while different architectural parameters were varied. The architectural parameters that were varied are: cache size, cache line size, cache set associativity, processing node speed, memory speed, memory bus width, network speed, and network width.

Our results show that the impact of network contention on overall application performance is considerable. Furthermore, the major component of this network contention occurs inside the network alone (injection channel contention, consumption channel contention, virtual channel contention, and physical channel contention). If network contention is taken into account, application performance can differ by as much as 60% when compared to the corresponding

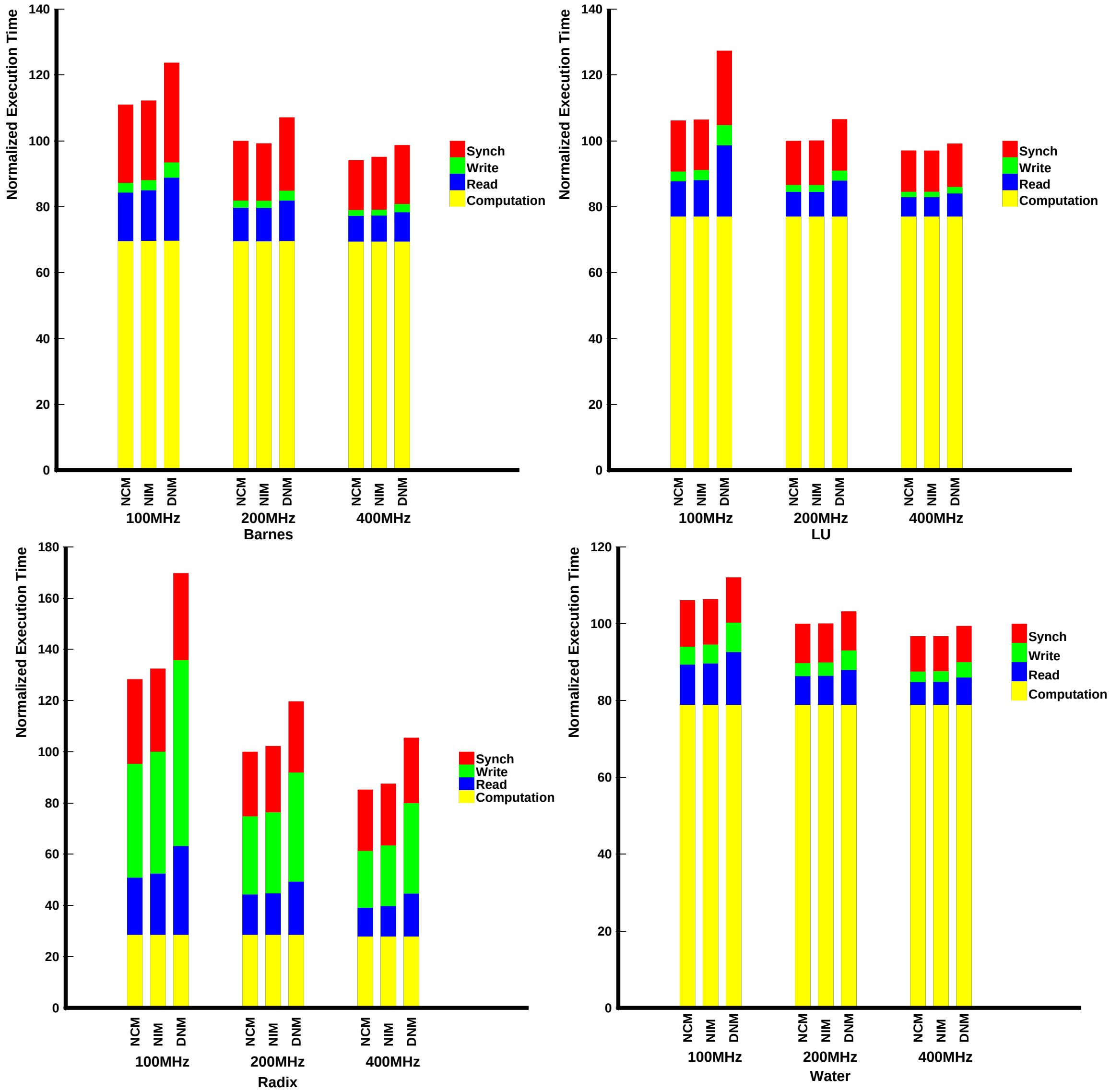

Figure 11: Effect of network speed on network contention.

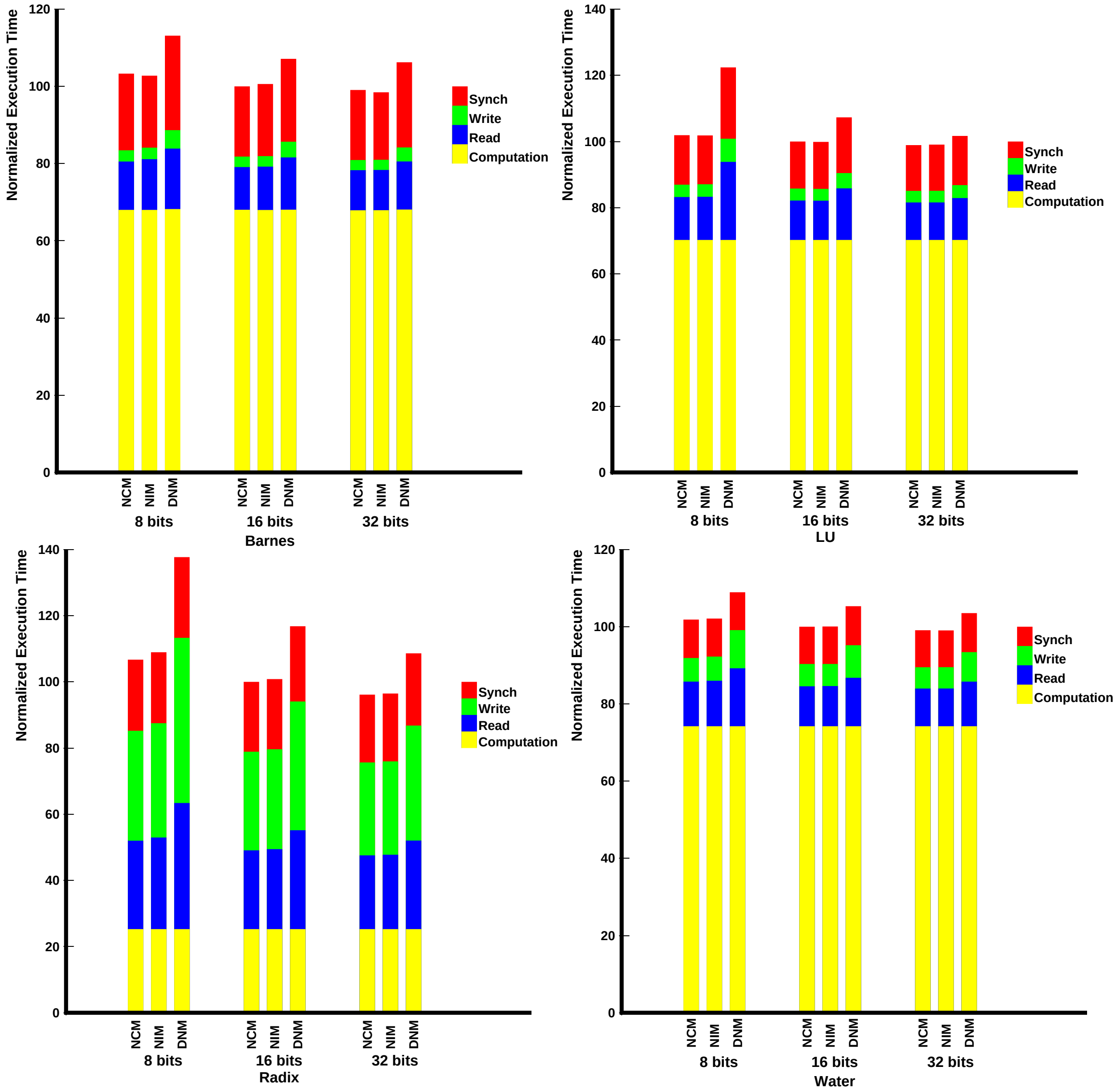

Figure 12: Effect of flit width on network contention.

performance under models that do not take any sort of network contention into account. When compared with models that take network interface contention into account (and assume contention free transmission within the network), application performance can differ by as much as 50%. These results clearly demonstrate the importance of modeling all types of network contention in general, and contention within the network alone in particular, while designing DSM systems.

Our study also shows that the various architectural parameters can have considerable impact on the effect of network contention on overall application performance. Smaller caches, larger cache lines, lower set associativity, higher processing node speeds, higher memory speeds, lower network speeds and narrower networks can significantly increase this effect of network contention on application performance. These results show that changes in any of these parameters of a DSM system can have a much greater impact on overall DSM performance if studied in conjunction with methods for reducing the amount of network contention (such as routing adaptivity, hardware support for collective communications, etc.). Alternatively, a study of methods to reduce network contention can result in possible changes in these architectural parameters so that the system can deliver similar (or better) performance at a lower cost.

Acknowledgment

We want to thank Rajeev Sivaram for having interesting discussion on this topic and for providing significant feedback during the preparation of this manuscript.

References

[1] C. Amza, A. L. Cox, and et al. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.

[2] D. C. Burger and D. A. Wood. Accuracy vs. Performance in Parallel Simulation of Interconnection Networks. In *Proceedings of the International Symposium on Parallel Processing*, April 1995.

[3] D. Dai and D. K. Panda. Reducing Cache Invalidation Overheads in Wormhole DSMs Using Multidestination Message Passing. In *International Conference on Parallel Processing*, pages I:138–145, Chicago, IL, Aug 1996.

[4] Donglai Dai and D. K. Panda. Efficient Schemes for Limited Directory-Based DSMs Using Multidestination Message Passing. Technical Report OSU-CISRC-11/96-TR61, Dept. of Computer and Information Science, The Ohio State University, Nov 1996.

[5] W. J. Dally. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, pages 194–205, Mar 1992.

[6] W. J. Dally and C. L. Seitz. The Torus Routing Chip. *Distributed Computing*, 1:187–196, 1986.

[7] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[8] A. Agarwal et al. The MIT Alewife Machine: Architecture and Performance. In *International Symposium on Computer Architecture*, pages 2–13, 1995.

[9] C. Holt et al. The Effects of Latency, Occupancy, and Bandwidth in Distributed Shared Memory Multiprocessors. Technical Report CSL-TR-95-660, Stanford University, 1995.

[10] D. Lenoski et al. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 148–159, May 1990.

[11] D. Lenoski et al. The Stanford DASH Multiprocessor. *IEEE Computer*, pages 63–79, March 1992.

[12] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proceedings of the International Symposium on Computer Architecture*, pages 302–313, 1994.

[13] S. C. Woo et al. The SPLASH-2 Programs: Chracterization and Methodological Considerations. In *International Symposium on Computer Architecture*, pages 24–36, 1995.

[14] S. K. Reinhardt et al. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 48–60, May 1993.

[15] C. J. Glass and L. Ni. Maximally Fully Adaptive Routing in 2D Meshes. In *Proceedings of the International Conference on Parallel Processing*, pages I:101–104, 1992.

[16] M. Heinrich, J. Kuskin, and Others. The performance impact of flexibility in the stanford flash multiprocessor. In *The sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, San Jose, CA, October 1994.

[17] J. L. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1990.

[18] C. Holt, J. P. Singh, and J. Hennessy. Application and Architectural Bottlenecks in Large Scale Distributed Shared Memory Machines. In *Proceedings of the International Symposium on Computer Architecture*, pages 134–145, May 1996.

[19] Yousef Khalidi and Umakishore Ramachandran. An Implementation of Distributed Shared Memory. *Software–Practice and Experience*, 21(5):443–464, May 1991.

[20] J. H. Kim and A. A. Chien. An Evaluation of Planar-Adaptive Routing (PAR). In *Proceedings of the Symposium on Parallel and Distributed Pro cessing*, pages 470–478, 1992.

[21] D. A. Koufaty, X. Chen, D. K. Poulsen, and J. Torrellas. Data Forwarding in Scalable Shared-Memory Multiprocessors. In *ACM International Conference on Supercomputing*, pages 255–264, 1995.

[22] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH Prototype: Logic Overhead and Performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41–61, Jan 1993.

[23] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, November 1989.

[24] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[25] T. Mowry and A. Gupta. Tolerating Latency Through Software-Controlled Prefetching in Shared-Memory Multiprocessor. *Journal of Parallel and Distributed Computing*, June 1991.

[26] L. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, pages 62–76, Feb. 1993.

[27] W. Oed. *Massively Parallel Processor System Cray T3D.* Cray Research GmbH, 1993.

[28] G. D. Pifarre, L. Gravano, S. A. Felperin, and J. Sanz. Fully Adaptive Minimal Deadlock-Free Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):247–263, 1994.

[29] Xiaohan Qin and Jean-Loup Baer. On the Use and Performance of Explicit Communication Primitives in Cache-coherent Multiprocessor Systems. In *Proceedings of the Third International Symposium on High Performance Computer Architecture*, February 1997.

[30] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the International Symposium on Computer Architecture*, pages 325–337, April 1994.

[31] S. K. Reinhardt, R. W. Pfile, and D. A. Wood. Decoupled Hardware Support for Distributed Shared Memory. In *Proceedings of the International Symposium on Computer Architecture*, May 1996.

[32] J. H. Upadhyay, V. Varavithya, and P. Mohapatra. Efficient and Balanced Adaptive Routing in Two-Dimensional Meshes. In *International Symposium on High Performance Computer Architecture*, pages 112–121, 1995.