How Can We Design Better Networks for DSM Systems?*

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: Most DSM research in current years have ignored the impact of interconnection network altogether. Similarly, most of the interconnection network research have focused on better network designs by using synthetic (uniform/non-uniform) traffic. Both these trends do not lead to any concrete guidelines about designing better networks for the emerging Distributed Shared Memory (DSM) paradigm. In this paper, we address these issues by taking a three-step approach. First, we propose a comprehensive parameterized model to estimate the performance of an application on a DSM system. This model takes into account all key aspects of a DSM system: application, processor, memory hierarchy, coherence protocol, network, etc. Next, using this model we evaluate the impact of different network design choices (link speed, link width, topology, ratio between router to physical link delay) on the overall performance of DSM applications and establish guidelines for designing better networks for DSM systems. Finally, we use simulations of SPLASH2 benchmark suites to validate our design guidelines. Some of the important design guidelines established in this paper are: 1) better performance is achieved by increasing link speed instead of link width, 2) changing topology of a network under constant bisection bandwidth constraint is not at all beneficial, 3) network contention experienced by short messages is very crucial to the overall performance, etc. These guidelines together with several others lays a good foundation for designing better networks for current and future generation DSM systems.

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

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

Contents

1 Introduction **1**

2 Parameterized Performance Model for a Typical DSM system **2**

2.1 Characterizing Threads and Synchronizations 2

2.2 Characterizing Memory Reference Latency 4

2.3 Characterizing Communication Latency 5

2.4 Putting All Components Together 5

3 Analyzing the Impact of Network Components **7**

3.1 Impact of Link Speed 8

3.2 Impact of Link Width 9

3.3 Impact of Topology 9

4 Simulation Results **10**

4.1 Simulation Environment 10

4.2 Performance Metrics 10

4.3 Impact of Network Speed 12

4.4 Impact of Link Width 13

4.5 Impact of Topology 13

4.5.1 Under Constant Link Width Constraint 14

4.5.2 Under Constant Bisection Bandwidth Constraint 14

5 Conclusion **15**

1 Introduction

Distributed Shared Memory (DSM) multiprocessors are gaining popularity among parallel systems because of the availability of globally shared address space and the ease of programmability. However, obtaining full potential performance from current generation DSM systems is somewhat limited due to the *high latency* associated with shared memory accesses. In a hardware DSM system this latency is affected by various design choices such as node architecture, memory hierarchy, coherence protocol, and interconnection network.

Most DSM research in recent years has focussed on the design of node architectures, coherence protocols, memory consistency techniques, etc. [8, 12]. Many of these studies have either ignored the interconnection network altogether or considered very idealistic model of the network - a model with fixed delay, infinite bandwidth, and no contention. Thus, these studies do not provide any concrete guidelines for designing efficient networks for DSM systems.

During the recent years, a lot of studies have also taken place in the interconnection network community to design better networks [6]. These studies have considered the impact of topology, routing adaptivity, link width, link speed, router delay, etc. on the overall network performance. However, the design guidelines established in these studies are based on synthetic traffic (uniform/non-uniform/hot-spot), arbitrary/bi-modal message lengths, and message generation intervals following some probability distributions. These studies are also geared towards showing better network performance in terms of 'latency vs. throughput'.

The above approach to establish guidelines for designing better networks is quite general. It does not consider many realistic characteristics associated with network messages in a typical DSM system. Such characteristics include *cause-effect* relationship between messages (remote read/write request message followed by a reply message), *bi-modal* traffic (short messages reflecting control messages in a cache-coherency protocol and long messages reflecting transfer of cache-lines), periodic generation of messages by processors based on the computational granularity associated with a thread, etc. Thus, for a DSM system with a given node architecture and cache coherency protocol, the existing network design studies do not provide guidelines on the impact of different network components (link speed, link width, topology, routing adaptivity, router delay, etc.) on the overall performance of the system.

In [4], we have reported the performance impact of network contention under various designs of cache and memory module, processor speed, and different network components. But how to design better networks for DSM systems to reduce or avoid the degradation of performance caused by contention remains a challenge. In this paper we take on such a challenge and approach it in three steps. First, we develop a parameterized performance model for a CC-NUMA architecture using a wormhole [15] k-ary n-cube interconnection network. The node architecture is assumed to be similar to the FLASH system [9]. This model integrates application characteristics (number of threads, number of synchronization points, and granularity of computation), components of the node architecture (access time for cache and memory, protocol overhead, and number of outstanding memory requests), and network components (link speed, link width, router delay, topology under constant link width, and topology under constant bisection bandwidth). Next, using this model we evaluate the impact of different network components on the overall performance of DSM applications. Finally, we use an application-driven simulation approach to validate the impact of network components and obtain additional insights such as the impact of network contention. The 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 [10], running on a 64 processor system.

Our study establishes several guidelines for designing better networks for DSM systems. Some of the important guidelines obtained from this study are: a) better performance is achieved by increasing link speed instead of link width, b) increasing dimension of a network under constant link width constraint is somewhat beneficial, c) increasing dimension of a network under constant bisection bandwidth constraint is not at all beneficial, d) routing delay plays an important factor in the design choices of other network components, e) network contention experienced by short messages is very crucial to the overall performance. These guidelines together with several others provide a good foundation for designing better networks for current and future generation DSM systems.

The rest of the paper is organized as follows. A parameterized performance model for DSM systems is developed in Section 2. The impact of network components on the overall performance of the DSM systems is established in Section 3. Simulation experiments and results are presented in Section 4. Finally, concluding remarks are made in Section 5.

2 Parameterized Performance Model for a Typical DSM system

In this section, we derive an analytical performance model for a typical DSM system. This model captures the following five most important factors: 1) application characteristics, 2) advanced processor features, 3) memory hierarchies and response times, 4) overhead of coherence protocols, and 5) network behaviors.

The DSM machine assumed in this model reflects current trends in designing hardware DSM systems. It consists of a set of identical processing nodes connected by a general network. Each node contains a processor with its cache, a portion of global memory, a node controller, and a network interface. Memory is organized as blocks 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 among the nodes based on the directory contents. The network interface sends/receives messages into/from the network for the node.

2.1 Characterizing Threads and Synchronizations

As shown in Fig. 1 (a), conceptually, any shared-memory application can be decomposed into a set of concurrent and coordinated threads. The set size can grow or shrink dynamically during the execution of the application as threads are created and terminated. Each of these threads carries out a certain amount of computation and communicates with one another by reading from or writing to common memory locations. In order to guarantee the correctness of the application, a subset of these reads/writes must be carried out in a specific order via inserting explicit synchronization primitives, such as barriers and locks. As a general practice to save the high overhead of thread creation and termination, the number of threads are typically kept constant and the same threads are used before and after synchronization points. For the purpose of performance modeling, such a thread exhibits a repeated pattern as follows: executing several instructions successively for computation and then followed by a memory access instruction. Let us call the short run of these successive instructions as a computation *grain* of the application, and denote its average execution duration as T_{grn}. This can be characterized as follows:

$$T_{grn} = g * CPI * T_{pclk} \tag{1}$$

where g is the number of instructions in a grain, known as *grain size*, CPI is the average number of cycles per instruction of the processor, and T_{pclk} is the processor's clock cycle time. In essence, the value of g represents the inherited computation to communication ratio of an application.

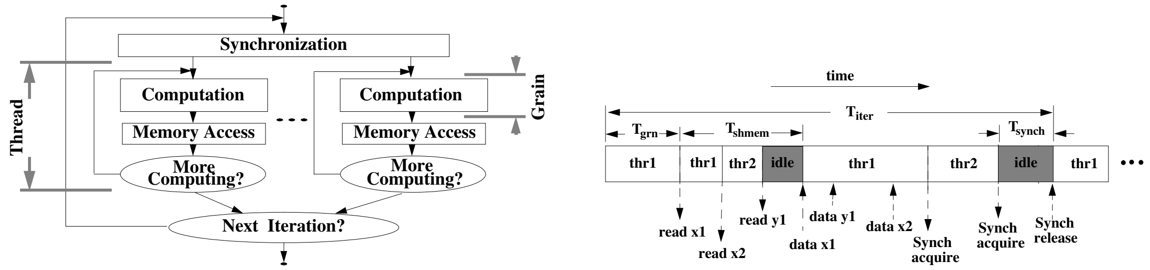

Figure 1: Time decomposition of a typical DSM application: (a) a programmer's point of view, (b) a processor's point of view.

The above programmer's point of view helps us to understand the essential behavior of an application on a given DSM system. Based on this, a processor's point of view, as shown in Fig. 1(b), can be established to provide us additional insights into the execution time of an application. The alternate view describes the behavior of an individual processing node running its portion of the application. Let us assume that the processing node in our modeled system supports multithreading and multiple outstanding requests. Such a model of the processing node is powerful enough to capture almost all the characteristics introduced by current latency reduction or latency tolerance techniques, such as lockup-free caches [17], hardware and software data prefetching [2, 14], relaxed memory consistency [8], speculative load and execution [16], multithreading [13], etc.

Let us consider the behavior of a typical application on a processing node of a DSM system for a short duration, as depicted in Fig. 1 (b). Assume two threads are mapped onto this processor. At the beginning of the observed duration, which is also assumed as the beginning of one iteration, thread $thr1$ executes a grain and issues a nonblocking read from location $x1$. The thread continues to execute its next grain, then issues a nonblocking read to $x2$ at the end of that grain and is suspended because of lack of parallelism. Instead of waiting for either x's data to be available, a context switch takes place. The processor starts running a grain from the second thread $thr2$. At the end, a blocking read to $y1$ is issued. At the moment, the processor becomes idle until one of the data items is available. Assume that as soon as the earliest item $x1$ is available, thread $thr1$ resumes to execute the next grain. The data items $y1$ and

$x2$ become available during the current grain. This allows thread $thr1$ to continue to execute another grain, which waits for $x2$. Finally, thread $thr1$ issues a synchronization acquire signal and is blocked. Immediately, thread $thr2$ starts running its next grain and is suspended at the grain's end after issuing its synchronization acquire signal. Again, the processor is left idle until a synchronization release signal comes back. Once the expected synchronization release signal arrives, the next iteration starts.

Clearly, any grain only contains instructions which do not involve any shared memory reference or synchronization operation. Let us define the following parameters: T_{grn} as the average grain time per thread, T_{shmem} as the average latency of each shared memory reference, T_{sync} as the average idle time from a processor's point of view due to each synchronization, N_{ref} as the average number of shared memory references per iteration per thread, N_{pend} as the maximum number of simultaneous outstanding memory requests allowed, N_{thr} as the average number of threads mapped to each processing node, and N_{iter} is the number of iterations in the program. Based on the earlier observation and assuming a perfect load balance, the overall execution time of an application can be expressed as below.[1]

$$T_{exec} = (((T_{grn} + T_{shmem}) * N_{ref}/N_{pend}) * N_{thr} + T_{sync}) * N_{iter} \qquad (2)$$

For a given DSM system, let us now estimate the value of T_{shmem}.

2.2 Characterizing Memory Reference Latency

The average latency of each shared memory reference T_{shmem} can be expressed as follows:

$$T_{shmem} = R_{read} * T_{read} + R_{write} * T_{write} \qquad (3)$$

where R_{read} or R_{write} stands for rates (percentages) of reads or writes out of total shared memory references. The average shared read latency T_{read} can be expressed as follows:

$$T_{read} = R_{rhit} * T_{hit} + R_{rmiss} * T_{r,mem} \qquad (4)$$

where R_{rhit} and R_{rmiss} are the hit and miss ratios for shared reads, T_{hit} is the latency for a cache hit reference, and $T_{r,mem}$ is the latency for a read miss reference. Similarly, the average shared write latency T_{write} can be expressed as follows:

$$T_{write} = R_{whit} * T_{hit} + R_{wmiss} * T_{w,mem} \qquad (5)$$

where R_{whit} and R_{wmiss} are the hit and miss ratios for shared writes.

The values of $T_{r,mem}$ and $T_{w,mem}$ depend on two factors: a) cache coherence protocol being used and b) memory access latency. The overhead of a cache coherency protocol can be further characterized as two components: delays incurred at the node controllers and the average number of messages generated for each read or write operation. In our representative DSM system, we assume directory-based protocols. Let us assume that the node controller incurs a fixed delay T_{prot} for handling each message and manipulating the directory. In a situation where data must be retrieved from the main memory, the access is assumed to proceed in parallel with

[1]For simplicity and clarity, we ignore the overheads for context switching and scheduling in our model.

the node controller operation. 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 access to the memory module can be pipelined with the cache. In other words, the retrieval of the block, the filling of the cache line, and the manipulation of the directory information are all overlapped. Such overlapping is common in modern DSM systems. Under these assumptions, $T_{r,mem}$ and $T_{w,mem}$ can be estimated as follows:

$$T_{r,mem} = T_{prot} + T_{mmod} + N_{r,s} * T_{smsg} + N_{r,l} * T_{lmsg} \tag{6}$$

$$T_{w,mem} = T_{prot} + T_{mmod} + N_{w,s} * T_{smsg} + N_{w,l} * T_{lmsg} \tag{7}$$

where T_{mmod} denotes the memory module access time, $N_{r,s}$ and $N_{r,l}$ denote the average number of short and long messages generated by the coherency protocol which fall on the critical path of the program's execution per read operation. Similarly, $N_{w,s}$ and $N_{w,l}$ are those numbers per write. It is to be noted that such a bimodal distribution of the message length has very important implications on the overall performance of a DSM system and on the design of better networks for a DSM system. We study these implications in Section 3 in detail.

2.3 Characterizing Communication Latency

Let us break down the components of latency for short (T_{smsg}) and long (T_{lmsg}) messages. Both contain certain amount of overheads at both sender (T_{snd}) and receiver sides (T_{rcv}). They also contain network delays, T_{snet} and T_{lnet}, respectively. Thus, we have:

$$T_{smsg} = T_{snd} + T_{snet} + T_{rcv} \tag{8}$$

$$T_{lmsg} = T_{snd} + T_{mmod} + T_{lnet} + T_{rcv} \tag{9}$$

It is to be noted that Eqn. 9 contains a term T_{mmod}. This is required for long messages (for example reply messages containing data) because the memory/cache block must be copied before the message is sent into the network. It is also to be noted that this overhead is different from the T_{mmod} components used in Eqns. 6 and 7. The components in Eqns. 6 and 7 reflect overheads for local copying.

2.4 Putting All Components Together

Let us put all components together by substituting and merging Equations 2- 9 This leads to the overall estimated execution time of an application as:

$$\begin{aligned}
T_{exec} = (((& T_{grn} \\
& + (R_{read} * R_{rhit} + R_{write} * R_{whit}) * T_{hit} \\
& + (R_{read} * R_{rmiss} + R_{write} * R_{wmiss}) * (T_{prot} + T_{mmod}) \\
& + (R_{read} * R_{rmiss} * N_{r,s} + R_{write} * R_{wmiss} * N_{w,s}) * (T_{snd} + T_{rcv}) \\
& + (R_{read} * R_{rmiss} * N_{r,l} + R_{write} * R_{wmiss} * N_{w,l}) * (T_{snd} + T_{mmod} + T_{rcv}) \\
& + (R_{read} * R_{rmiss} * N_{r,s} + R_{write} * R_{wmiss} * N_{w,s}) * T_{snet} \\
& + (R_{read} * R_{rmiss} * N_{r,l} + R_{write} * R_{wmiss} * N_{w,l}) * T_{lnet}) \\
& * N_{ref}/N_{pend}) * N_{thr} + T_{sync}) * N_{iter}
\end{aligned} \tag{10}$$

Let us rename some of the coefficients as below,

$$R_{hit} = R_{read} * R_{rhit} + R_{write} * R_{whit} \tag{11}$$

$$R_{miss} = R_{read} * R_{rmiss} + R_{write} * R_{wmiss} \tag{12}$$

where R_{hit} and R_{miss} reflect the overall cache hit/miss ratio of all shared memory references. Similarly, we can also combine the number of messages being generated as below,

$$N_{short} = R_{read} * R_{rmiss} * N_{r,s} + R_{write} * R_{wmiss} * N_{w,s} \tag{13}$$

$$N_{long} = R_{read} * R_{rmiss} * N_{r,l} + R_{write} * R_{wmiss} * N_{w,l} \tag{14}$$

where N_{short} and N_{long} mean the average number of short and long messages generated by the cache coherence protocol for each shared memory reference and which fall on the critical path of the program's execution. Using these compressed notations, the overall execution time can be re-expressed as follows:

$$\begin{aligned} T_{exec} = (((&T_{grn} + R_{hit} * T_{hit} + R_{miss} * (T_{prot} + T_{mmod}) \\ &+ (N_{short} + N_{long}) * (T_{snd} + T_{rcv}) + N_{long} * T_{mmod} \\ &+ N_{short} * T_{snet} + N_{long} * T_{lnet}) * N_{ref}/N_{pend}) * N_{thr} + T_{sync}) * N_{iter} \end{aligned} \tag{15}$$

The above equation captures the five most important factors of a DSM system, as outlined in the beginning of this section. The term T_{hit} captures the access time to the top level memory hierarchy. The term $(T_{prot} + T_{mmod})$ captures the overhead of coherence protocol including directory maintenance and the response delay of memory module. The term $(T_{snd} + T_{rcv})$ reflects the overheads associated with network interface. Factors N_{pend} and N_{thr} represent a set of advanced features provided in the processors to exploit more parallelism existed in an application. Parameters T_{grn}, N_{ref}, and N_{iter} largely depend on the application characteristics. Parameters T_{snet} and T_{lnet} represent the basic network latency without contention. Lastly, parameters R_{hit}, R_{miss}, N_{short}, and N_{long} represent a combined effect of the application characteristics, the overall memory hierarchy, and the coherence protocol.

Once a set of values are given to the above parameters/terms, this model can predict the performance of a DSM system. It can also identify the bottlenecks associated with different components of the system. In the following section, we consider this model to study the impact of network design choices.

It can be observed from the the recent literature [1, 10] that the current generation DSM systems and application characteristics exhibit the following range of values for the parameters used in our model: T_{grn} from 2 to 500 processor cycles, R_{hit} from 0.90 to 0.99, R_{miss} from 0.01 to 0.10, T_{hit} is from 0 to 2 processor cycles, T_{mmod} from 10 to 50 processor cycles, $T_{prot} + T_{mmod}$ from 20 to 55 processor cycles, N_{short} from 0.10 to 0.30, N_{long} from 0.02 to 0.10, $T_{snd} + T_{rcv}$ from 10 to 30 processor cycles, N_{ref} from 2 to 100, N_{pend} from 1 to 8, N_{thr} from 1 to 4, and T_{sync} from 5% to 20% of T_{exec}/N_{iter}.

In the following section, in order to study the impact of different components of a network, we use the following representative set of values: $R_{hit} = 0.97$, $T_{hit} = 0$, $R_{miss} = 0.03$, $T_{mmod} = 24$, $T_{prot} = 2$, $T_{snd} = 15$, $T_{rcv} = 5$, $N_{short} = 0.10$, $N_{long} = 0.02$, $T_{sync} = 2 * T_{grn}$, $N_{ref} = 20$,

$N_{pend} = 1$, $N_{thr} = 1$. All these values are expressed in terms of numbers of processor clock cycles. For convenience of discussion, let us denote $T_{iter} = T_{exec}/N_{iter}$.

We consider three different computational granularities to represent three different application types: *Type 1* ($T_{grn} = 80$), *Type 2* ($T_{grn} = 40$), and *Type 3* ($T_{grn} = 5$). These represent medium coarse-grain to fine-grain applications. After substituting the parameters in Eqn. 15 with all these values, the value of T_{iter} for each type of applications can be written as follows:

$$T_{iter,type1} = 1833.2 + 0.10 * T_{snet} + 0.02 * T_{lnet} \quad (16)$$

$$T_{iter,type2} = 953.2 + 0.10 * T_{snet} + 0.02 * T_{lnet} \quad (17)$$

$$T_{iter,type3} = 183.2 + 0.10 * T_{snet} + 0.02 * T_{lnet} \quad (18)$$

3 Analyzing the Impact of Network Components

In this section we use the model derived in the previous section and study the impact of different network components on the overall performance of DSM systems. We consider the following four major components of a network design: link speed, link width, routing delay, and topology. We consider k-ary n-cube wormhole interconnection, a popular choice among the current generation DSM systems.

It is well known that the average message latency of a general network 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 typically caused by limited resources in the network like queue sizes, injection channels, consumption channels, physical links, router/switch contention, etc. Since it is difficult to model network contention analytically [6], we consider no-contention message latencies in this section. In the following section, we derive message latencies with contention from simulation results and compare them with the no-contention latencies.

Consider the average message latencies for short and long messages, as indicated in Eqns. 8 and 9, respectively. Based on the wormhole-routing principles, the network latency components (T_{snet} and T_{lnet}) can be estimated as follows:

$$T_{snet} = D * (T_{rout} + T_{phy}) + (L_{short}/W - 1) * (T_{sw} + T_{phy}) \quad (19)$$

$$T_{lnet} = D * (T_{rout} + T_{phy}) + (L_{long}/W - 1) * (T_{sw} + T_{phy}) \quad (20)$$

where D is the average distance (number of hops) traveled by a message, L_{short} is the length of a short message, L_{long} is the length of a long message, W is the width of a link (flit) in the network, T_{rout} and T_{sw} are the routing and switching delays at each router/switch for a message, T_{phy} is the propagation delay of a flit along a single link. The short messages in a DSM system are due to control messages resulted from cache coherency. The length of such messages are determined by the number of message types used in a cache coherency protocol and the total number of cache blocks in the system. The length of a long message depends on the above factors as well as the cache line size. In this paper, we assume a directory-based cache coherency protocol with 32 different message types, 64 processing nodes, 512 MBytes of shared global memory, and cache line size of 64 bytes. These lead to the following message lengths: $L_{short} = 6$ bytes and $L_{long} = 70$ bytes.

3.1 Impact of Link Speed

Let us study the impact of link speed on the overall performance of DSM applications using the model derived so far. Consider an 8×8 mesh with 16-bit wide links as our network. This leads to $D = 8$ and $W = 2$. Let us consider three different values of link speed (100 MHz, 200 MHz, and 400 MHz). These speeds correspond to link bandwidths being 200, 400, and 800 MBytes, respectively. It is to be noted that in this paper, we have assumed default processor speed to be 200 MHz, as discussed in Section 2. For a 100 MHz link, we have $T_{phy} = T_{sw} = 1$ network cycle (can be indicated as *ncycle*) which is equal to 2 processor cycles (can be indicated as *pcycle*). Depending on the current trends in router design, let us consider two different representative values for routing delays (T_{rout}): 1 ncycle and 4 ncycles.

By using the above values in Eqn. 19, we can estimate the no-contention latency for short messages to be 40 and 88 pcycles, respectively. Similarly, based on Eqn. 20, the no-contention latency for long messages can be estimated as 168 and 216 pcycles, respectively. If we change the link speed to 200 and 400 MHz, the latencies for T_{snet} and T_{lnet} reduce linearly as shown in Table 1. However, higher routing delay has a substantial penalty on the message latencies.

Table 1: Predicted performance for three different application types under different network design choices.

Network Design Choices			T_{snet} (pcyc.)	T_{lnet} (pcyc.)	Type 1		Type 2		Type 3	
					Estim.	Relat.	Estim.	Relat.	Estim.	Relat.
Link Speed (MHz)	T_{rout} : 1 ncyc.	100	40	168	1980.4	100.0%	1100.4	100.0%	330.4	100.0%
		200	20	84	1906.8	96.3%	1026.8	93.3%	256.8	77.7%
		400	10	42	1870.0	94.4%	990.0	90.0%	220.0	66.6%
	T_{rout} : 4 ncyc.	100	88	216	1095.6	100.0%	1215.6	100.0%	445.6	100.0%
		200	44	108	1964.4	93.7%	1084.4	89.2%	314.4	70.6%
		400	22	54	1898.8	90.6%	1018.8	83.8%	248.8	55.8%
Link Width (bytes)	T_{rout} : 1 ncyc.	1	26	154	1946.8	100.0%	1066.8	100.0%	296.8	100.0%
		2	20	84	1906.8	97.9%	1026.8	96.3%	256.8	86.5%
		4	18	50	1889.2	97.0%	1009.2	94.6%	239.2	80.6%
	T_{rout} : 4 ncyc.	1	50	178	2004.4	100.0%	1124.4	100.0%	354.4	100.0%
		2	44	108	1964.4	98.0%	1084.4	96.4%	314.4	88.7%
		4	42	74	1946.8	97.1%	1066.8	94.9%	296.8	83.7%
Topol. (dim, width (bytes))	const. link width	2, 2	20	84	1906.8	100.0%	1026.8	100.0%	256.8	100.0%
		3, 2	16	80	1897.2	99.5%	1017.2	99.1%	247.2	96.3%
		6, 2	16	80	1897.2	99.5%	1017.2	99.1%	247.2	96.3%
	const. bisec. width	2, 4	18	50	1889.2	100.0%	1009.2	100.0%	239.2	100.0%
		3, 2	16	80	1897.2	100.4%	1017.2	100.8%	247.2	103.3%
		6, 1	22	150	1937.2	102.5%	1057.2	104.8%	287.2	120.1%

Now let us observe the overall impact of these message latencies on the execution times for three representative application types, as outlined at the end of Section 2. Eqns. 16, 17, and 18 outlined the average iteration times for these application types in terms of T_{snet} and T_{lnet}. These times are computed and shown in the right six columns of Table 1. It can be observed that as the granularity of an application decreases, increasing link speed has a substantial benefit on the overall execution time. These benefits can vary from 6% for a medium coarse-grain application to 33% for a fine-grain application. These benefits are more for networks with higher routing delays, ranging from 9% to 44%.

3.2 Impact of Link Width

Next, let us consider the impact of increasing network link width. Let us assume an 8×8 network with 200 MHz link speed. Consider three different values for link width: 1 byte, 2 bytes, and 4 bytes. The fourth and fifth columns of Table 1 show the respective latencies for short and long messages under two different routing delays. It can be observed that the delays do not reduce linearly as seen with link speed. This is because only a portion of the message latency is affected by the link width, as shown in Eqns. 19 and 20.

Let us consider the overall impact on the execution time, as shown in Table 1. It can be observed that increasing link width can benefit in reduction of execution time by 3% to 19% for lower routing delay and 2.9% to 16% for higher routing delay. Comparing these benefits with those of increasing link speed, it can be observed that increasing link speed has better benefit than increasing link width.

3.3 Impact of Topology

Finally, we consider the impact of changes in topology. For a fair comparison, we consider two different strategies for varying topologies: *under constant link width constraint* and *under constant bisection bandwidth constraint* [5]. Under the former constraint, link width is kept constant as the dimension of the network changes. Under the second constraint, link width is changed so as to maintain constant bisection bandwidth.

Let us consider three different topologies: 2d (8×8), 4d ($4\times4\times4$) and 6d ($2\times2\times2\times2\times2\times2$) hypercube. Under constant link width constraint, let us assume the link width to be kept constant at 2 bytes. The link speed is kept constant at 200 MHz (400 MBytes/sec bandwidth). Changes in topology lead to a change in the average distance traveled by the messages (D=8, 6, and 6 for the respective topologies). Third and fourth columns of Table 1 show the values of T_{snet} and T_{lnet} under changing topologies for lower routing delay ($T_{route} = 1$ ncycle). It can be observed that the latencies for 4d and 6d systems are identical because the average distance traveled for both (given) topologies happen to be identical. With respect to the overall reduction in execution time, increasing the dimension of the network provides benefit. However, it is very small (0.5% to 3.7% only).

Under constant bisection bandwidth constraint, the link width for hypercube topology is assumed as 1 byte. This indicates that the link width of 4d and 2d systems needs to be 2 bytes and 4 bytes, respectively. The latencies for short and long messages under this constraint show interesting trends. As the dimension of the network changes from 2d to 4d to 6d, the average latency for small messages first reduces and then increases. The reduction is due to lowering the average distance traveled. However, it is quickly overpowered by reduction in link width. The average latency for long messages continue to rise with increase in dimension because of reduction in link width. These changes show increasing effect on the overall execution time for the three representative applications. The increase is quite substantial for fine-grain applications, indicating that higher dimension networks under constant bisection bandwidth constraint are not suitable for designing DSM systems.

The guidelines derived in this section are based on only no-contention message latency. In the following section, we present detailed simulation results to validate these results as well as study the impact of message contention on these guidelines.

4 Simulation Results

In order to validate the analytical model and the guidelines derived in the last section, we simulated a DSM system with an architecture and coherence protocol similar to the FLASH machine [9] but with some differences described later. In this section, first, we describe the default architectural parameters used in the simulation. Then briefly, we discuss the applications in our experiments. Finally, we study the impact of different network components.

4.1 Simulation Environment

We simulated a 64-node DSM system with a default interconnection as an 8×8 mesh. The processor in each processing node was 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 64 bytes. The cache was assumed to operate in dual-port mode using write-back and write-allocate policies. The instruction latencies, issue rules, and memory interface were modeled based on the DLX design [11]. 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 pipelined fashion. The machine is assumed to be using a full-mapped, invalidation-based, three-state directory coherence protocol [9, 7, 3]. The synchronization protocol assumed in the system is the QOLB protocol similar to the one used in DASH [8]. The node simulator models the internal structures, and the queuing and contention at the node controller, main memory, and cache. Table 2 shows the memory hierarchy parameters and node controller occupancy delays used in our simulated architecture.

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

We used four applications — Barnes, LU, Radix, and Water in our simulation. These are ported from the widely used SPLASH2 suite [10]. Table 3 lists problem 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 [10].

4.2 Performance Metrics

We present all our simulation results in two sets: execution time and network latency. This helps us to correlate the behavior of underlying network with the overall DSM system performance and provide important insights into the network design issues. The overall execution time is broken down into four components: computation, memory read waiting, memory write waiting,

Table 2: Default system parameters used in simulation.

Memory Hierarchy Parameters		Network Parameters	
Processor frequency	200MHz	Network frequency	200MHz
Cache access	1 cycle	Channel width / Flit size	2 bytes
Cache line size (L)	64 bytes	Link Propagation (T_{link})	1 network cycle
Cache set associativity	2	Router switch delay (T_{sw})	1 network cycle
Cache size per node	128 Kbytes	Routing time (T_{rout})	1 network cycles
Memory word width (W)	8 bytes/cycle	Physical network	1
Memory response delay	30 cycles	Virtual networks	2
Cache block fill time	30+L/W cycles	Virtual channels per virtual network	1
Memory block access time	30+L/W cycles		
Node Controller Occupancy		**Network Interface Parameters**	
Directory check	7 cycles	Outgoing message startup ($T_{outgoing}$)	15 cycles
Directory check and update	14 cycles	Incoming message dispatch ($T_{incoming}$)	8 cycles
Each invalidation request	12 cycles	Control message size	6 bytes
Message forward	3 cycles	Data message size	70 bytes
		Injection channels per node	1
		Consumption channels per node	1

and synchronization waiting. In the Figs 2(a), 2(c), 3(a), 3(c), 4(a), and 5(a), all the time values presented are normalized to that of the left most system for each application.

Table 3: Applications and the used input sizes.

Application	Problem Size
Barnes	$8K$ particles, $\theta = 1.0$, 4 time steps
LU	512×512 doubles, 8×8 blocks,
Radix	$1M$ keys, $1K$ radix, max $1M$
Water	512 molecules, 4 time steps

The network latencies of two types (short and long) of messages are presented separately in absolute time (microseconds). Furthermore, we present the average latencies of short and long messages in three different ways: *estimated, ideal, and real.* The *estimated latency* was calculated according to Eqn. 19 or 20, where D is taken as half of the network diameter, i.e., the average distance traveled per message under uniform traffic. The ideal latency for short and long messages were computed dynamically during the simulation execution by accumulating the essential delays (such as $T_{rout}, T_{phy}, T_{sw}$) along the actual traversal path of the messages. The *ideal latency* presented in this section (shown with a legend 'w/o contention') is the average per message among all the messages of the same type. In other words, this should be the measured value of average latency per message if there exists no contention in the network. Lastly, the *real latency* presented in this section (shown with a legend '/w contention') is the average of network latencies across all messages of the same type. The network latency for a message was measured from the start of its injection at the sender until the completion of its consumption at the receiver. This latency includes contention at injection channel, consumption channel, and communication links. It is to be noted that the difference between the values of ideal latency and estimated latency of a message type tells us how close is the traffic of that message type to the 'uniform' traffic. Similarly, the difference between the values of ideal latency and real latency of a type tells us the effect of network contention.

4.3 Impact of Network Speed

Figures 2(a) and (c) show the impact of network speed on the overall execution time. The network speed was changed from 100 MHz to 400 MHz (200 MBytes/sec to 800 MBytes/sec bandwidth). Figures 2(b) and (d) show the respective message latencies. These results refer to two different routing delays ($T_{rout} = 1$ and 4 network cycles), as discussed in Section 3. As predicted in section 3.1, it can be observed that the overall execution time reduces as the speed of the network increases. The reduction in execution time varies across applications, ranging from 7.3% in Water to 54.6% in Radix. From the timing breakdowns, it can be observed that Water is most computation intensive where as Radix is communication intensive. In other words, the average grain size in Water is quite large compare to that in Radix. It can also be observed that the reduction in execution time is higher for networks with higher routing delay, as predicted in section 3.1.

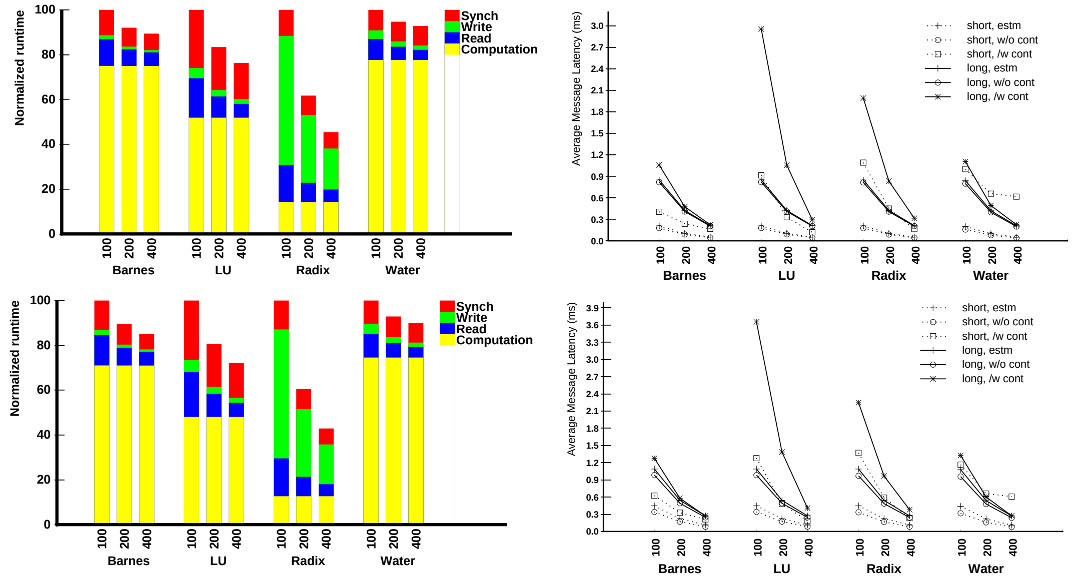

Figure 2: Impact of different network speed: (a) execution time and (b) average message latency for $T_{rout} = 1$ cycle; (c) execution time and (d) average message latency for $T_{rout} = 4$ cycles.

Figures 2(b) and (d) show the average message latencies for the same set of simulation runs. Several important observations can be drawn from these data. First of all, the curves of the estimated latency are slightly above to the curves of ideal latency for both short and long messages. The two set of curves follow exactly the same trend and their difference margin is very small. This reveals the fact that the traffic of both message types (short and long) in a typical DSM system is closer to 'uniform' traffic with a somewhat smaller average distance per message because of locality. Similar trends can also be observed from the remaining subsections.

Second, the real (with contention) latency might be much higher than the ideal latency under certain network designs. In such scenarios, contention inside the network becomes the dominant factor for network latency. Since both types of messages share the same network, the link contention between short and long messages causes the delay for short messages to be disproportionately greater. Increasing network speed causes contention to be reduced substantially for short messages, leading to overall reduction in execution time (up to 40%). Similar trends can also be observed from the remaining subsections.

4.4 Impact of Link Width

Figures 3(a) and (b) show the execution time and average latencies for the applications running under the default system configuration with varying link widths, (8, 16, or 32 bits each). These results are for smaller routing delay. Figures 3(a) and (b) show the trend for higher routing delay. As expected from the analysis in section 3.2, it can be observed that wider links help reducing the overall execution time. The reduction varies from 5.3% in Water to 51.7% in Radix. The reduction is smaller for higher routing delay. The performance improvements are also smaller compared to the respective improvements obtained by increasing link speed.

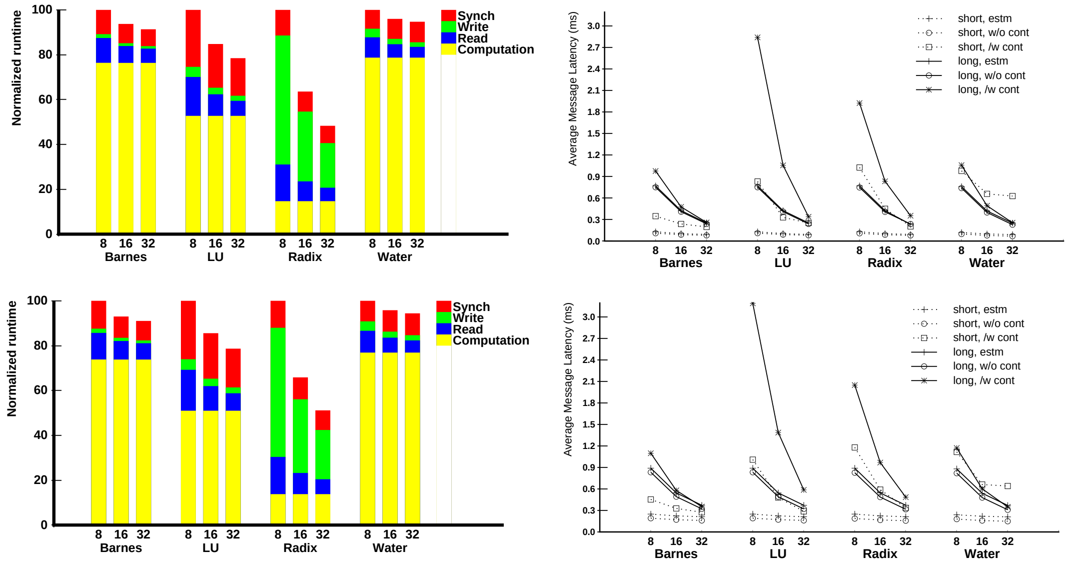

Figure 3: Impact of link width: (a) execution time and (b) average message latency for $T_{rout} = 1$ cycle; (c) execution time and (d) average message latency for $T_{rout} = 4$ cycles.

4.5 Impact of Topology

We examined the impact of different topologies. We compared the results for 2D (8×8), 3D (4^3), and 6D (2^6) meshes under two different constraints: constant link width and constant bisection bandwidth.

4.5.1 Under Constant Link Width Constraint

Figures 4 (a) and (b) show the execution time and average latencies of each application running under various topologies. The link width was kept at 16bits for all experiments. The results were derived under lower routing delay. As predicted in Section 3, it can be observed that increasing the dimension of the network helps in reducing the execution time. However, the relative improvement is very small.

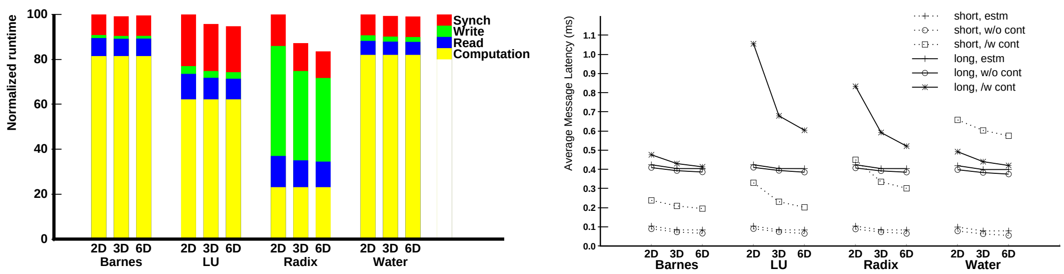

Figure 4: Impact of network topology under constant link width constraint: (a) execution time and (b) average message latency.

4.5.2 Under Constant Bisection Bandwidth Constraint

Figures 5 (a) and (b) show the execution time and average latencies of each application running under various topologies. The link width was varied to maintain constant bisection bandwidth. The results were derived under lower routing delay. As predicted in Section 3, it can be observed that increasing the dimension of the network has negative impact on the overall execution time. With increase in network dimension, both short and long messages undergo increasing contention, leading to increase in overall execution time. Thus, 2D systems are shown to be best under this constraint.

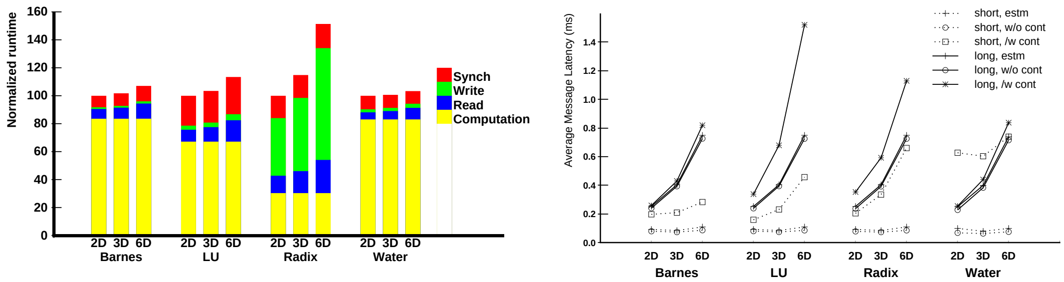

Figure 5: Impact of network topology under constant bisection bandwidth constraint: (a) execution time and (b) average message latency.

5 Conclusion

Advances in processor technology is making network latency an increasingly important architectural bottleneck in DSM systems. In addition to the continued research efforts in developing latency reduction and latency tolerance techniques, a better understanding of the basic communication characteristics of DSM systems/applications and a set of concrete guidelines for designing more efficient networks targeted specifically for DSM systems are very desirable. In this paper, we have developed a comprehensive parameterized model to estimate the performance of a CC-NUMA multiprocessor system using k-ary n-cube wormhole interconnection. This model integrates all the key aspects of a DSM system: application characteristics (number of threads, number of synchronization points, and granularity of computation), components of the node architecture (access time for cache and memory, protocol overhead, and number of outstanding memory requests allowed), and network components (link speed, link width, router delay, and topology). Based on this model we have studied the impact of different network components on the overall performance of DSM systems/applications. Finally, we have validated the impact of network components and obtained additional insights such as the impact of network contention by results from execution-driven simulation of SPLASH2 benchmarks.

Our study establishes several guidelines for designing better networks for DSM systems. Some of the important guidelines obtained from this study are: a) better performance is achieved by increasing link speed instead of link width, b) increasing dimension of a network under constant link width constraint is somewhat beneficial, c) increasing dimension of a network under constant bisection bandwidth constraint is not at all beneficial, d) routing delay plays an important factor in the design choices of other network components, e) network contention experienced by short messages is very crucial to the overall performance. These guidelines can be used by the architects as a good foundation to design better networks for current and future generation DSM systems.

References

[1] S. Chandra, J. R. Larus, and A. Rogers. Where is time spent in message-passing and shared-memory programs? In *The sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, pages 61–73, San Jose, CA, October 1994.

[2] T. Chen and J.-L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. In *Proceedings of the 21st ACM Annual International Symposium on Computer Architecture (ISCA-21)*, pages 223–232, April 1994.

[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] D. Dai and D. K. Panda. How Much Does Network Contention Affect Distributed Shared Memory Performance? . Technical Report OSU-CISRC-2/97-TR14, Dept. of Computer and Information Science, The Ohio State University, Febuary 1997.

[5] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Network. *IEEE Transactions on Computers*, pages 775–785, June 1990.

[6] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach.* The IEEE Press, 1996.

[7] 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.

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

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

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

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

[12] 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.

[13] J. Laudon, A. Gupta, and J. Horowitz. Interleaving: A multihtreading technique targeting multiprocessors and workstations. In *The sixth International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, pages 308–318, San Jose, CA, October 1994.

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

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

[16] A. Rogers and Kai Li. Software support for speculative loads. In *The Fifth International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 38–50, October 1992.

[17] G. Sohi and M. Franklin. High-performance data memory systems for superscalar processors. In *The Fourth International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pages 53–62, April 1991.