The Impact of Communication Locality on Large-Scale Multiprocessor Performance

Kirk L. Johnson
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

As multiprocessor sizes scale and computer architects turn to interconnection networks with non-uniform communication latencies, the lure of exploiting communication locality to increase performance becomes inevitable. Models that accurately quantify locality effects provide invaluable insight into the importance of exploiting locality as machine sizes and features change. This paper presents a framework for modeling the impact of communication locality on system performance. The framework provides a means for combining simple models of application, processor, and network behavior to obtain a combined model that accurately reflects feedback effects between processors and networks. We introduce a model that characterizes application behavior with three parameters that capture computation grain, sensitivity to communication latency, and amount of locality present at execution time. The combined model is validated with measurements taken from a detailed simulator for a complete multiprocessor system. Using the combined model, we show that exploiting communication locality provides gains which are at most linear in the factor by which average communication distance is reduced when the number of outstanding communication transactions per processor is bounded. The combined model is also used to obtain rough upper bounds on the performance improvement from exploiting locality to minimize communication distance.

1 Introduction

At the heart of any multiprocessor lies the interconnection network through which processing nodes communicate with one another. The simplest types of interconnection networks allow all distinct pairs of processors to communicate with the same latency, ignoring contention effects. The simplicity of the cost model provided by such *uniform communication latency* (UCL) interconnection networks affords simplicity in software systems (e.g. compilers, schedulers, operating systems) built upon architectures that use such networks. Unfortunately, mechanisms used to implement UCL networks are not *scalable*. Single-level shared-bus architectures are limited by bus bandwidth and are unable to support reasonable communication loads from more than a few dozen processors. Full crossbars might provide nearly uniform communication latencies, but excessive hardware requirements prevent them from being scalable. Indirect, multistage networks that circumvent the bandwidth problems of buses and quadratic hardware requirements of full crossbars provide bandwidth which scales with machine size. But this increased bandwidth comes at a price: *all* communication latency increases with the number of processors in the system.[1]

Appears in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

Because the construction of scalable UCL networks appears to be intractable, future large-scale multiprocessors will probably use some type of *non-uniform communication latency* (NUCL) interconnection network (e.g. meshes, trees). In such networks, all processors are able to communicate with one another, but latencies may vary greatly. In terms of average communication latency between all pairs of processors, such networks may do no better than the most efficient UCL networks. However, as machine sizes scale, NUCL networks have an advantage: some processors can remain "close" to one another, regardless of machine size; other processors gradually become "further" away. The smaller the distance between a pair of communicating processors, the lower the communication latency. Moreover, the network bandwidth required for communication typically decreases with communication distance.

This difference gives NUCL architectures an advantage over UCL architectures. As machine sizes scale, applications running on UCL architectures face increasing latencies for all communication. On the other hand, with NUCL architectures, some applications can be organized such that communication patterns tend to favor nearby processors. Such applications should experience performance gains commensurate with the degree that communication latencies can be reduced.

We prefer UCL and NUCL over the more conventional UMA (*uniform memory access*) and NUMA (*non-uniform memory access*) nomenclature, because the conventional terms imply shared-memory semantics. UCL and NUCL capture the same fundamental difference for a more general class of architectures.

1.1 What is Locality?

Applications often take advantage of *communication locality* to realize performance gains. Communication locality is a property of both applications and architectures. *Application locality* (or *algorithmic locality*) is that which is present in the organization of an application, independent of architectural details. *Architectural locality* represents the ability of an architecture to exploit

[1] Conventional wisdom dictates that this increase is logarithmic in the number of processors in the system; proponents of "three-space realizability" arguments claim a somewhat higher rate [9].

application locality.

Two components contribute to application locality. The first, *temporal locality*, represents the effect of decreasing the communication frequency between application threads. Applications that minimize inter-thread communication by maximizing data reuse tend to exhibit good temporal locality. The second component, *physical locality*, represents the effect of affinity in the communication patterns amongst an application's threads. Applications tend to have good physical locality to the extent that their inter-thread communication graphs have relatively low bisection width and high diameter. An application in which all distinct pairs of threads communicate equally has no physical locality.

With this understanding, the fundamental difference between UCL and NUCL architectures becomes clear. Both classes of architectures allow exploitation of temporal locality, but UCL architectures are only able to exploit physical locality to the extent that threads can be collocated when there are more threads than processors. In addition to being able to collocate threads, NUCL architectures can exploit physical locality by placing communicating threads on processors such that average communication distance is minimized.

1.2 Managing Communication Latency

As machine sizes scale, communication latencies will also increase. Architects have proposed many approaches for managing that latency; these approaches can be divided into three broad categories.

- Those that attempt to *avoid* long latency operations. These approaches exploit temporal locality in the application and focus on enhancing data reuse (e.g. caches, compilation for data locality).
- Those that attempt to *reduce* communication latency. These approaches exploit physical locality in the application and focus on minimizing communication distance.
- Those that attempt to *tolerate* long latency operations. These approaches typically focus on software paradigms and processor architectures which allow useful work to overlap long latency operations (e.g. multithreading, relaxed memory consistency models, data prefetching).

This paper focuses on the impact of the latter two approaches on multiprocessor performance. Numerous researchers have demonstrated the importance of the first approach (exploiting temporal locality); compilation techniques for increasing temporal locality continue to be an active area of research [10, 12].

1.3 Contributions of this Paper

This paper describes a framework for analyzing the impact of communication costs on end application performance for a broad class of multiprocessors. This framework is then used to quantify the impact of physical locality in large-scale multiprocessors using interconnection networks organized as k-ary n-dimensional meshes.

The modeling framework provides a mechanism for combining separate models of applications, communication mechanisms (e.g. coherence protocols), and interconnection networks. Each of the component models can be changed independently of the others; this provides a means of gauging the importance of particular application or architectural features in controlled experiments. Component models are combined such that applications receive feedback from interconnection networks and only proceed at rates appropriate to observed communication latencies. The application model introduced in this paper characterizes application behavior with three parameters: the application's computational grain size, its ability to tolerate communication latency, and the amount of physical locality present at execution time. Model predictions agree closely with measurements taken from detailed simulations of a complete multiprocessor system.

We demonstrate that under a reasonable set of assumptions about processor and application behavior, the impact of contention in k-ary n-dimensional mesh networks is bounded. In particular, we use the combined model to show that when the number of outstanding communication transactions per processor is bounded, average per-hop communication latency in large machines approaches a constant value; this effect is observable in machines with a few thousand processors. Because average per-hop latency approaches a constant value under this condition, average communication latencies are linear in communication distance.

This fact has profound impact on the performance improvements available by exploiting physical locality. Any gain due to exploiting physical locality is bounded by the degree to which communication latencies are reduced. Since communication latencies are linear in communication distance, exploiting physical locality to reduce communication distances provides gains which are at most linear in the factor by which communication distances are reduced.

We also use the combined model to obtain rough upper bounds on the performance improvement possible by exploiting physical locality in two-dimensional mesh networks. In the architecture used in the simulation experiments, network switches are clocked twice as fast as processors. Under these conditions, our analysis indicates that these upper bounds are around two for machines with 1,000 processors and roughly 50 for machines with a million processors. By using the model to investigate the impact of changing the relative balance between computation and communication speeds, we demonstrate that these bounds are significantly larger for architectures with less bandwidth-rich interconnection networks—decreasing the relative speed of the network by a factor of eight increases the upper bounds by approximately a factor of three.

1.4 Overview

The rest of this paper is organized as follows: Section 2 describes the modeling framework and its components. Section 3 describes the set of experiments used to verify the model. The section concludes by presenting results from these experiments and noting that they agree closely with the values predicted by the model. Section 4 uses the model to predict performance across a wide range of machine configurations. Section 5 provides a brief overview of related work. Section 6 closes with a summary of our major results. Finally, Appendix A summarizes the nomenclature used in the rest of the paper.

2 A Framework for Modeling

This section presents a framework for modeling the impact of communication costs on application performance. The framework consists of three components: an application model describes processor behavior in terms of abstract communication transactions, a transaction model describes the resources required to satisfy communication transactions, and a network model characterizes the behavior of the underlying interconnection network. The application and transaction models are combined to obtain a node model which describes the behavior of individual multiprocessor nodes as seen by the interconnection network. The final combined model is obtained by joining the node and network models. These models are joined such that applications effectively receive feedback from the network and only inject messages at rates appropriate to the message latencies they actually observe.

In order to compare application performance for different machine configurations and applications with different amounts of physical locality, a metric of performance is needed. This framework uses average transaction rate as a basis for such comparisons, as discussed in Section 2.6. Simulation results demonstrating the accuracy of the model are presented in Section 3.

2.1 Application Model

The application model describes the behavior of individual processors running their portion of an application by indicating how per-processor communication transaction issue rate depends on transaction latency. The model captures both application- and processor-specific details. It characterizes not only the basic computational grain size of an application but also how behavior changes when transaction latencies increase. Computational grain size is an informal notion; another common term for the same notion is computation-to-communication ratio. In this paper, this notion is captured by T_r, the average amount of useful work done by a thread between successive communication transactions. The dependence of processor behavior on observed transaction latency is captured by the *latency sensitivity*, s, as described below. We show that s is intimately related to the degree to which an architecture and application utilize multiple outstanding communication transactions.

For architectures with UCL interconnection networks, these two parameters (T_r and s) are sufficient to characterize application behavior. For architectures with NUCL interconnection networks, additional information about communication patterns that exist during application execution (e.g. which processing nodes communicate with one another) is also necessary. These patterns depend on the amount of physical locality present in an application, the topology of the interconnection network, and the degree to which the mapping used to assign application threads to processors exploits these features to minimize communication distances. Because good metrics for these factors are still an area of active research, this paper uses an operational definition of physical locality. All information about communication patterns is captured by a single number: average communication distance d, measured in network hops. This parameter captures the amount of physical locality present at execution time. For interconnection networks with topologies more complex than that of the k-ary n-dimensional meshes used in this paper, more detailed representations might be necessary.

For the applications and architecture described in this paper, communication transactions are cache coherency transactions. The development of the framework does not depend on this fact; given an appropriate transaction model, the framework could be used to model applications and architectures that utilize other styles of interprocessor communication.

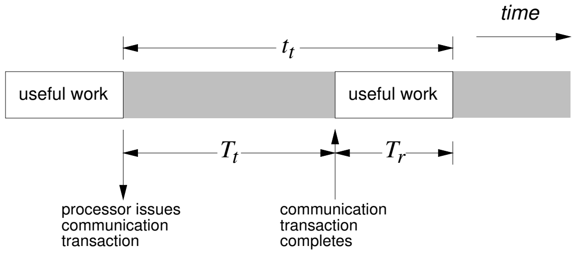

Figure 1: A simple processor in action. Shaded regions between periods of useful work represent time spent waiting for communication transactions to complete.

In order to derive a simple application model from first principles, consider an application for which the average run length between successive communication transactions by a thread is T_r cycles. Assume each transaction takes an average of T_t cycles to complete. When processors can only support a single outstanding communication transaction, as depicted in Figure 1, the average *inter-transaction issue time* t_t is given by

$$t_t = T_r + T_t \tag{1}$$

or

$$T_t = t_t - T_r \tag{2}$$

That is, inter-transaction issue time and average transaction latency T_t are linearly related with a slope of 1 and intercept determined by the computational grain of the application. In what follows, such relationships between t_t and T_t will be referred to as *application transaction curves.*

Extending this model to accommodate processor architectures capable of supporting multiple outstanding transactions is straightforward. Consider, for example, the case of block multithreaded processors with p hardware contexts and a context switch time of T_s cycles (see Figure 2). Each thread runs until it issues a communication transaction, at which time the processor switches contexts and restarts execution of the next thread. Such processors have two basic modes of operation. In the first mode, transaction latencies are small enough to ensure that transactions always complete before the invoking thread is ready to run again. The following inequality expresses this condition:

$$T_t < pT_s + (p-1)T_r \tag{3}$$

When operating in the first mode, processors are completely able to mask communication latency. A new transaction is invoked every $T_s + T_r$ cycles, thus the average inter-transaction issue time is simply

$$t_t = T_r + T_s \tag{4}$$

This equation represents the minimum inter-transaction issue time for a given application when using multithreaded processors.

When the inequality in Equation (3) does not hold, processors operate in a second mode where they are no longer able to completely mask communication latencies. When operating in the

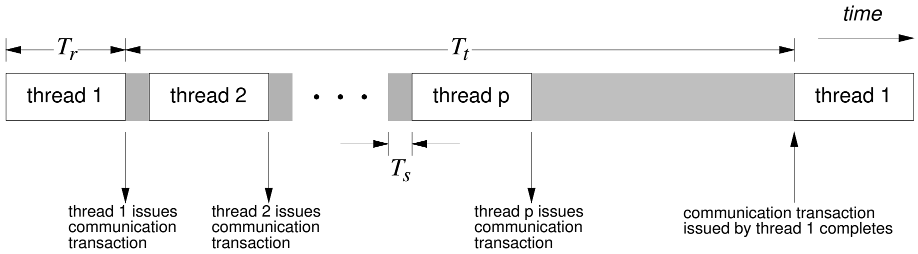

Figure 2: A block multithreaded processor in action. Hashed regions between thread runs represent context switching time; the shaded region at the right represents time spent waiting for communication transactions to complete.

second mode, processors invoke p transactions every $T_r + T_t$ cycles (see Figure 2), thus the average inter-transaction issue time is given by

$$t_t = \frac{T_r + T_t}{p} \tag{5}$$

or

$$T_t = pt_t - T_r \tag{6}$$

As with non-multithreaded processors, application behavior is characterized by a linear relationship between t_t and T_t. In fact, the only difference due to p-multithreaded processors is an extra factor of p in the slope of the $t_t - T_t$ relationship (compare Equations (2) and (6)) and the lower bound on average inter-transaction issue time given by Equation (4).

None of the experiments described in this paper yielded inter-transaction issue times approaching the lower bound given in Equation (4). Thus the rest of this paper drops explicit references to this bound.

Increases in the slope of an application transaction curve indicate decreased sensitivity of application performance to increases in transaction latency. For example, consider two applications, $\mathcal{A}$ and $\mathcal{B}$, such that the slope of $\mathcal{B}$'s application transaction curve is twice that of $\mathcal{A}$. If an increase in transaction latency of x cycles causes $\mathcal{A}$'s average inter-transaction issue time to increase by y cycles, then a similar increase in the transaction latency observed by $\mathcal{B}$ will only increase $\mathcal{B}$'s average inter-transaction issue time by $y/2$ cycles.

Extending the application model to accommodate other mechanisms for supporting multiple outstanding transactions (e.g. weak ordering, data prefetching) is equally straightforward. The impact of all such mechanisms is essentially the same: applications for which processors keep an average of k transactions outstanding have application transaction curves with slopes k times greater than those for the same application running on processors which can only support a single outstanding transaction. Since increases in slope indicate decreased sensitivity to communication latency, this corresponds nicely with the intuition that architectures embodying such mechanisms should be less sensitive to increasing communication latencies.

Note that the constant offset in the application model (Equation (6)) depends on an application's computational grain, as captured by T_r. When the same application model is used to characterize application behavior for a range of machine configurations, there is an implicit assumption that problem size is being scaled such that T_r remains constant. For many applications this may require using problem sizes proportional to the total number of hardware contexts available.

2.2 Transaction Model

The transaction model describes the resources required to satisfy a communication transaction. It encompasses two pieces of information: the average number of network messages which must be sent to satisfy a transaction and the average latency of a transaction.

Intuitively, transaction latency T_t consists of some fixed delay plus a component dependent on average message latency T_m. The fixed delay arises from factors such as message sending and receiving overhead, processing necessary to maintain coherence in cache-coherent systems, and the like. Thus average transaction latency is modeled as

$$T_t = cT_m + T_f \tag{7}$$

where c indicates the extent to which transaction latency depends on message latency, and T_f represents any fixed delay inherent in the transaction mechanism. In essence, c is the number of messages in the "critical path" of a transaction. For simple transaction mechanisms which require only single message exchanges between peers, $c = 2$.

The average number of messages required to satisfy a communication transaction is assumed to be some constant g. Observe that the average inter-message injection time t_m for an individual node and the average inter-transaction issue time t_t are related by a factor of g:

$$t_t = gt_m \tag{8}$$

2.3 Node Model

The node model describes the behavior of a multiprocessor node from the perspective of the interconnection network. The node model indicates how fast nodes will inject messages into the interconnection network as a function of average message latency. In essence, the node model provides the same information for messages that the application model provides for transactions.

The node model is derived from the application and transaction models by substituting Equations (7) and (8) into Equation (6) and simplifying.

$$T_m = \frac{pg}{c} t_m - \frac{T_r + T_f}{c} \tag{9}$$

Not surprisingly, Equation (9) indicates that average inter-message injection time t_m and average message latency T_m are linearly related. The slope of the relationship depends on the constants from the transaction model and the slope of the application transaction curve; the intercept on the computational grain and constants from the transaction model. Such relationships between t_m and T_m will be referred to as *application message curves*.

The intuition about the slope of application transaction curves applies to application message curves as well: the greater the slope of an application message curve, the less sensitive t_m is to increases in T_m. Accordingly, we will refer to the slope of an application message curve as the *latency sensitivity* s. Larger values of s correspond to decreased sensitivity of application performance to changes in average message latency. Note that s is proportional to p, the average number of outstanding transactions, where the constant of proportionality is determined by the parameters of the transaction model.

2.4 Network Model

The network model describes the behavior of the interconnection network by providing average message latency as a function of average message size, message injection rate, and average communication distance. In this paper, we use a model for packet-switched, buffered mesh networks, but the framework can easily accommodate models for other types of packet-switched networks such as that for indirect networks given in [8]. We believe that modifications to allow for circuit-switched networks would be straightforward.

The network model used in this paper is that for packet-switched k-ary n-dimensional torus networks with separate unidirectional channels in both mesh directions presented by Agarwal in [1]. The model assumes that messages are wormhole routed according to an e-cube routing scheme [6]. In this model, channel utilization ρ and average message latency T_m are given by

$$\rho = \frac{r_m B k_d}{2} \tag{10}$$

$$T_m = n k_d T_h + B \tag{11}$$

where

- r_m is the average message injection rate for each node. r_m is related to the average inter-message injection time according to

$$r_m = \frac{1}{t_m} \tag{12}$$

- B is the average message size, in flits
- k_d is the average distance per dimension each message must travel, which is given by the average communication distance d divided by the number of dimensions n:

$$k_d = \frac{d}{n} \tag{13}$$

- T_h is the average per-hop latency for the head of a message, which is given by

$$T_h = 1 + \left(\frac{\rho B}{1-\rho}\right)\left(\frac{k_d - 1}{k_d^2}\right)\left(\frac{n+1}{n}\right) \tag{14}$$

Average communication distance d (and thus k_d) depends on the amount of physical locality in an application and the extent to which good thread-to-processor mappings are used. In addition to reducing average communication distance, good mappings can also lead to lower values of T_h, the average per-hop latency. This happens because reductions in k_d provide corresponding reductions in channel utilization ρ (see Equations (10) and (14)). Thus the benefits of exploiting physical locality are twofold: both communication distance and network contention are reduced.

This paper extends Agarwal's basic network model slightly. First, as given, Equation (14) is only valid for $k_d \geq 1$. Good mappings of applications that exhibit a considerable degree of physical locality can easily violate this condition. Under these circumstances, messages will tend to encounter very little contention delay. Thus, for $k_d < 1$, T_h is taken to be 1.

A second extension reflects the effect of contention for the channels which connect processing nodes to the network. An accurate network model must account for this factor, for it can become significant in the presence of long messages or high message rates. For the experiments described in Section 3, contention for these channels added two to five network cycles to the average message latency. Extending the model to account for this contention introduces additional complexity to the solution of the combined model without providing further insight into the issues at hand. Accordingly, this paper does not treat such an extension. The modeled values shown in Sections 3 and 4, however, reflect the inclusion of this factor as discussed in [7].

2.5 Combined Model

We obtain the combined model by using the node and network models to provide feedback to one another so that individual nodes "back off" as message latencies increase, injecting messages into the network at rates appropriate to the message latencies they actually observe. Equating the right hand sides of Equations (9) and (11) provides us with a quadratic polynomial in r_m, the average message injection rate. Given values for the various application and network model parameters, the quadratic is easily solved; the resulting value of r_m represents the predicted per-node message injection for the application and network described by the given parameter values. Other values of interest (e.g. channel utilization, average inter-transaction issue time) are obtained by substituting the predicted value of r_m into the appropriate model equations.

2.6 Performance Metrics

In order to compare performance of different machine configurations, a performance metric is needed. Such a metric should relate directly to end performance—the rate at which useful work gets done.

Within the framework presented in this paper, the average amount of useful work each thread performs between issuing communication transactions is assumed to be some constant T_r. Since the average inter-transaction issue time is t_t, the rate at which useful work gets done can be computed directly as T_r/t_t. Note that this value is proportional to the average transaction issue rate r_t, where r_t is given by

$$r_t = \frac{1}{t_t} \tag{15}$$

When T_r is held constant, r_t serves as a good metric of per-processor performance—increases in r_t are indicative of proportional increases in actual per-processor performance. A good metric of aggregate performance for an N-processor machine is thus

Nr_t. For a particular application model, the performance obtained with two different machine configurations can be compared by computing the ratio of the aggregate performance obtained in each case. Similar computations can be used to compare the performance of different application models for a particular machine configuration.

3 Experimentation

This section describes the simulation experiments used to validate the combined model. It begins with a description of the multiprocessor architecture for which the simulations were run, followed by a description of the application code used. The section concludes with a presentation of simulation results validating the models developed in Section 2.

3.1 The Architecture

The architecture simulated in these experiments is that of the MIT Alewife machine [2]. The basic Alewife architecture consists of processor/memory nodes communicating over a packet-switched interconnection network organized as a low-dimensional mesh.

Each processor/memory node consists of a Sparcle processor [3], a floating-point coprocessor, several megabytes of DRAM, a 64-kilobyte unified instruction/data cache (direct mapped, 16-byte lines), and a controller that serves as both memory and network interface. In order to support block multithreading, Sparcle provides four hardware contexts. Switches between hardware contexts can be effected in 11 cycles. In addition to providing the processor direct access to message sending and receiving facilities, the memory/network interface implements the LimitLESS protocol [4] to provide coherent caches and shared memory. Components on a processor/memory node are clocked at 33 to 40 MHz.

The mesh-organized interconnection network provides two eight-bit unidirectional channels between each neighboring pair of nodes; one for each direction. The bandwidth of each channel is around 60 to 80 Mbytes/sec; network switches are clocked twice as fast as processors. The base delay through a network switch is a single network cycle. Each network switch is connected to its corresponding processing node through a pair of eight-bit channels; one channel is used to deliver messages from the node to the network, the other to deliver messages from the network to the processing node. A moderate amount of buffering is provided on each switch. Messages are wormhole routed according to an e-cube routing scheme [6].

All simulations were run assuming a 64-node machine organized as a two-dimensional, radix-eight torus. The simulator used provides instruction level simulation of Sparcle and cycle-by-cycle simulation of the cache, controller, and network components.

3.2 The Application

The model of the previous section described application behavior in terms of computation grain T_r, latency sensitivity s, and average communication distance d. In order to validate the model, we devised a synthetic application with several interesting properties. First, it has a particularly small computation grain size. As the goal of these experiments was to measure the impact of varying communication costs on end performance, this is useful—intuition dictates that these effects should be more pronounced in applications with small computation grain. Second, the application has particularly good physical locality: its 64 threads communicate with one another according to a radix-eight two-dimensional torus organization. This allowed us to drastically vary average communication distance by using different thread-to-processor mappings. Third, by using the block multithreading features of Sparcle to vary the number of transactions the application kept outstanding, we could vary the latency sensitivity of the application.

Each of the simulation experiments used the same application code. In that code, each thread maintains a single word of state in local memory and repeatedly iterates through a simple inner-loop. During the course of one pass through the inner-loop, a thread reads the value from each of its neighbors' state words, performs some trivial computation, and writes a new value to its own state word. Threads make no effort to synchronize with one another.

Note that inter-thread communication takes place through shared memory. Each time a thread reads a neighbor's state or updates its own state, inter-thread communication is effected, if necessary, through appropriate cache-coherency transactions. Because Alewife provides coherent caches and all threads run the same inner-loop, the great majority of attempts to read a neighbor's state word cause cache-coherency traffic—it is likely one thread will update its state word before another can read it twice. Similarly, when a thread updates its local state word, cache-coherency traffic will usually be needed to invalidate read copies of the word from neighboring threads' caches.

Two parameters were varied across the suite of simulations. First, by using different thread-to-processor mappings, the average number of hops required to communicate between neighboring threads in the application's communication graph could be changed drastically. Average communication distances for the nine mappings used in these experiments ranged from one to just over six network hops.[2]

The second parameter that was changed between experiments was the degree of multithreading, p. In each case, the simulation run consisted of as many independent instances of the basic application as there were hardware contexts, with exactly one thread from each application instance resident on every processor. No data was shared between application instances, thus threads on the same node do not communicate with one another. Experiments were run with one, two, and four hardware contexts.

A priori knowledge of the cache line size and details of the coherency protocol provides the expected average message size and average number of messages per transaction; these values are 96 bits ($B = 12$ flits, assuming 8-bit network channels) and $g = 3.2$ messages per transaction, respectively.

3.3 Verification of the Model

The rest of this section presents experimental results which demonstrate the accuracy of the combined model.

Figure 3 shows application message curves as measured for the simulation experiments described earlier in this section. As predicted by Equation (9), t_m and T_m are linearly related. Note that increasing the number of contexts causes the application message curve slope to increase in the expected manner. That is, the

[2] Randomly chosen thread-to-processor mappings on a machine of this size lead to expected average communication distances of just over four network hops (see Equation (17)).

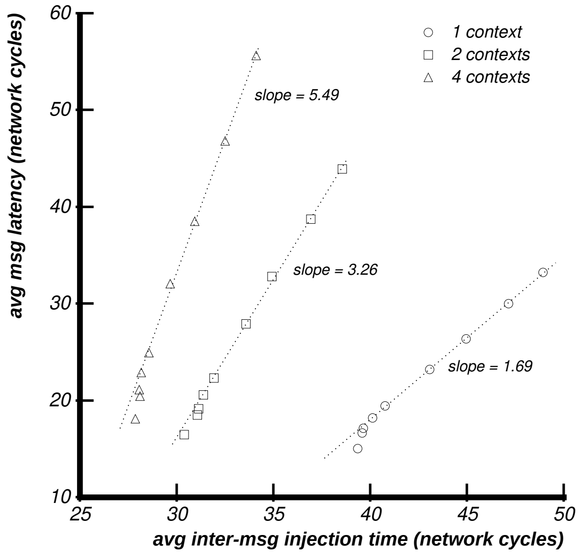

Figure 3: Average inter-message injection time t_m vs. average message latency T_m. Symbols denote simulation results; dotted lines represent linear regressions of measured values.

slope for two contexts is roughly twice the slope for one context, and so on.

Several phenomena cause the increases in slope to be slightly less than expected. Most of the difference can be attributed to an unfavorable interaction between the asynchronous nature of the application code used in the experiments and the cache coherence protocol.[3] Because of this interaction, the constant c in Equation (7) increases with the number of contexts; in the experiments with four contexts, c is measured to be 15 percent larger than in experiments with one context. Since c occurs in the denominator of the slope of the node model (Equation (9)), increases in c manifest themselves as decreases in application message curve slope.

Figures 4 and 5 plot average communication distance against average message rate and average message latency. Symbols indicate results from individual simulation runs; dotted curves show the values predicted by the model. Note that predicted values for message rate are consistently within a few percent of measured values and predicted values for message latency track measured values to within a few network cycles.

4 Results and Discussion

A number of interesting questions can be addressed given a model that accurately describes the effects of changing communication costs on application behavior. This section provides analysis and discussion of two such issues. First, we show that when the number of outstanding communication transactions per processor is bounded, communication latencies in large machines are linear in communication distance. Because of this fact, the exploitation of physical locality provides gains which are at most linear in the factor by which average communication distance is reduced.

Second, we use the combined model to obtain rough upper bounds on the performance improvement possible from exploit-

[3] No real applications have exhibited this behavior.

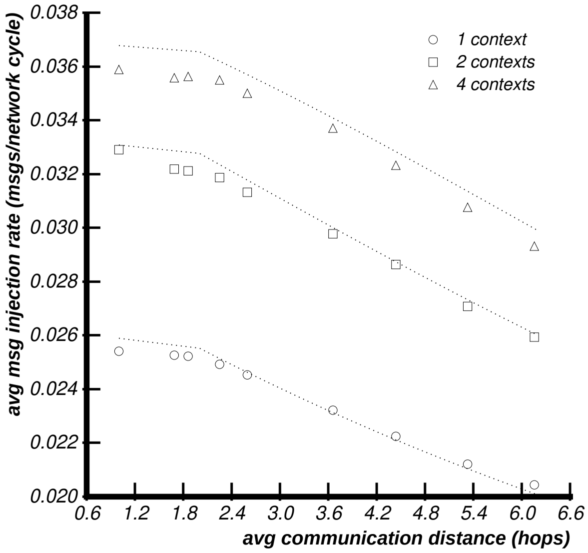

Figure 4: Average communication distance d vs. average message injection rate r_m. Symbols denote simulation results; dotted lines show values predicted by combined model.

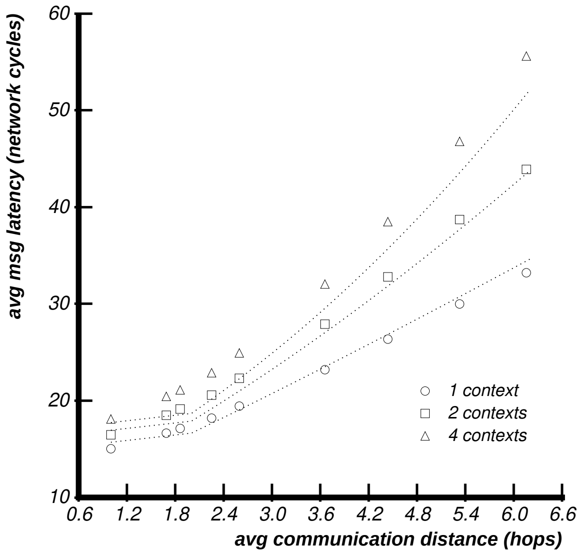

Figure 5: Average communication distance d vs. average message latency T_m. Symbols denote simulation results; dotted lines show values predicted by combined model.

ing physical locality in two-dimensional mesh networks. For the architecture described in Section 3, we show that these bounds are somewhat less than one might expect. We establish that this is due to the relatively high bandwidth provided by the network used in that architecture and show that the bounds increase when less bandwidth-rich interconnection networks are used—decreasing the relative speed of the network by a factor of eight leads to upper bounds that are roughly three times larger.

4.1 Bandwidth vs. Latency

As machine sizes scale, applications with little physical locality place increasing bandwidth demands on interconnection networks. Increases in application bandwidth requirements in turn cause contention effects to become more pronounced. Using the framework presented in this paper, we demonstrate that under a reasonable set of assumptions about application and processor behavior, the impact of contention effects is bounded, even for very large machines and communication-intensive applications that induce heavy network loads. Consequently, we show that the performance benefit from exploiting physical locality to reduce communication distances is at most linear in the factor that average communication distance is reduced.

By combining Equations (10), (11), and (14), one can show that as average communication distances increase, the average time it takes a message to travel a single network hop approaches a limiting value given by

$$T_h = \frac{Bs}{2n} \tag{16}$$

See [7] for details. Note that this limiting value depends only on average message size B, latency sensitivity s, and network dimension n. For a given application and architecture, B and n are constants, while s is linearly related to p, the average number of outstanding communication transactions.[4]

Intuitively, T_h approaches this limiting value because of the linkage between application and network behavior. If each node can only have some finite number of transactions outstanding (i.e. s is finite), increasing transaction latencies cause transaction issue rates to fall. This negative feedback keeps processors from loading interconnection networks to a point where communication latencies become unbounded.

Thus, as machine sizes scale and communication distances increase, T_h approaches the value given by Equation (16). In turn, this implies that average communication latency is linear in communication distance. Note that processor architectures able to support multiple outstanding transactions afford higher values of s, which lead to proportional increases in the limiting value of T_h.

These conclusions have a profound impact on the potential benefit of physical locality. Any gain due to exploiting physical locality is bounded by the degree by which communication latencies are reduced. Since communication latencies are linear in communication distance, reducing average communication distance by some factor x can only provide performance gains which are linear in x.

The smaller the computational grain of an application, the faster the limiting value of T_h will be approached. For applications with small computational grain, the limiting case can be approached for relatively small machines. For example, consider Figure 6. T_h is plotted against machine size for two applications; a two-dimensional interconnection network is assumed. The solid curve represents the application of Section 3 running on a multithreaded processor with two hardware contexts. The dashed curve shows the effect of artificially increasing the computational grain size by a factor of ten. In this case ($s = 3.26$, $B = 12$, $n = 2$), the limiting value of T_h is approximately 9.8 network cycles. For the application with smaller computational grain size, T_h reaches over eighty percent of its limiting value with a few thousand processors. When the grain size is increased, T_h still approaches the same limiting value, albeit quite a bit more slowly, as expected. Curves for one and four hardware contexts behave similarly.

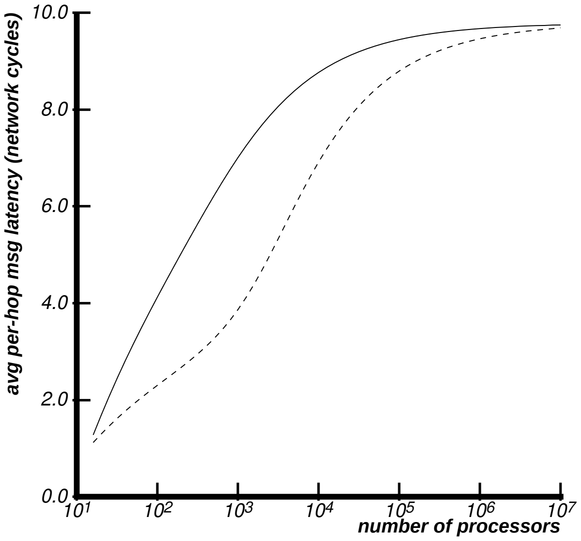

Figure 6: Average per-hop message latency T_h vs. number of processors N. The solid line represents the application of Section 3 running with two hardware contexts, assuming random communication patterns. The dashed line shows the impact of increasing the computation grain of that application by a factor of ten.

4.2 Physical Locality

Intuition dictates that application performance should benefit from thread-to-processor mappings that reduce overall communication distance. The more physical locality present in an application, the greater the gains possible through reducing communication distances. This section demonstrates that while such gains are certainly possible, they are somewhat less than one might initially expect for the architecture described in Section 3—no more than a factor of two or so for a 1,000 processor machine. An examination of the factors leading to this less-than-expected impact indicates that it is primarily due to the relatively high ratio of communication bandwidth to computation speed in that architecture. Recomputing the gains for architectures with progressively slower networks confirms this fact showing that larger gains are possible when processors are faster relative to the speed of the interconnection network.

Any application with finite latency sensitivity s will be communication-bound for large enough message latency T_m. Once T_m exceeds some threshold, increases in communication latencies will cause proportional decreases in application performance.

[4] For realistic applications and architectures, s is finite.

Equation (9) indicates that the smaller an application's computational grain size, the lower this threshold will be. Further, the lower this threshold, the greater the impact exploiting physical locality can have on end performance—reductions in communication latency are more beneficial when applications are already communication-bound. Thus, for an application with substantial physical locality and a very small computational grain size, comparing the performance obtained with best-case mappings and mappings which ignore physical locality provides a rough upper bound on the potential benefit of exploiting physical locality for *any* application. Since the application described in Section 3 has the desired properties, it is well suited for use in this comparison.

Random mappings represent the expected case when applications possess no physical locality or physical locality is ignored when mapping threads to processors. Under these circumstances, essentially random communication patterns result. For torus-connected k-ary n-dimensional interconnection networks, assuming random communication patterns and nodes that never send messages to themselves, the average distance traversed by a message, d, is given by[5]

$$d = \frac{nk^{n+1}}{4(k^n - 1)} \tag{17}$$

See [1] or [11] for details.

For this application and network, because the communication patterns of the application are the same as the network topology, an ideal mapping in which every communication requires but a single network hop is trivially obtained. Such a mapping represents the expected best case.

As discussed in Section 2.6, average transaction issue rate is a good metric of per-processor performance. Thus, in order to compare performance obtained with different mappings of a particular application on a given machine configuration, it is sufficient to compute the ratio of the transaction issue rates obtained in each case. When this ratio compares the performance of ideal and random mappings for a given machine size N, it is termed the *expected gain* due to exploitation of physical locality.

Figure 7 shows a log-log plot of expected gain against machine sizes from ten to a million processors for the application of Section 3 running with one, two, and four hardware contexts. The curves are strikingly similar—each starts at unity gain for ten processors, reaches a gain of two at around 1,000 processors, then quickly enters a region where the randomly-mapped application is communication-bound. Expected gains for one million processors range from 40 to 55. Recall that because the application being measured has very small computation grain, these are rough upper bounds on the gains available to any application.

Initially, an expected gain of only a factor of two for a 1,000 processor machine seems somewhat surprising. On a machine of that size, average communication distance for random mappings is nearly a factor of 16 larger than the single-hop distances afforded by an ideal mapping. Further, there's good reason to expect that for random mappings, T_h will be substantially larger, by a factor of four or more, than for an ideal mapping (see Figure 6). It is reasonable to expect, then, that the nk_dT_h term of Equation (11) will be a factor of 50 or 100 larger for a random mapping than for an ideal mapping.

[5] Equation (17) is only exact for even values of k. For odd values of k, the correct value is smaller by $\frac{nk^{n-1}}{4(k^n-1)}$. Since this term is $O(\frac{1}{k})$, it rapidly becomes insignificant as k increases.

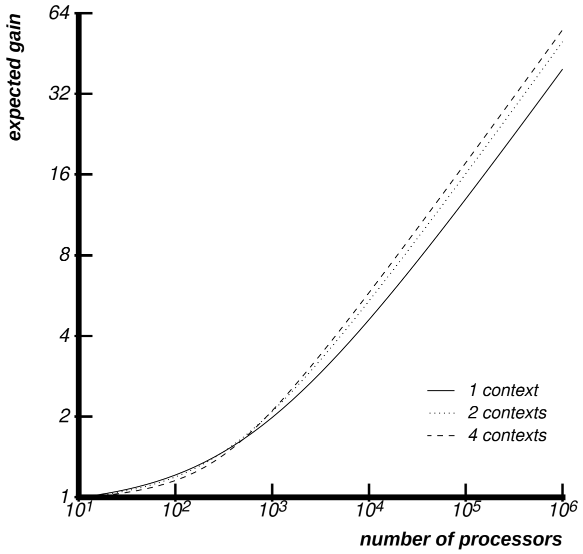

Figure 7: Expected gain due to exploitation of physical locality vs. number of processors (two-dimensional mesh). Each curve compares performance of ideal mapping vs. random mapping for a given degree of processor multithreading.

These results indicate that a large disparity exists between the magnitudes of the difference in communication costs and the difference in end performance when comparing ideal and random mappings. By using the application and network models from Section 2, the degree to which various factors contribute to end performance can be quantified. Such a breakdown allows identification of the phenomena that lead to the apparent disparity.

Substituting Equations (7) and (11) into Equation (6) and solving for the average inter-transaction issue time t_t gives:

$$t_t = \frac{c}{p} nk_dT_h + \frac{c}{p} B + \frac{T_f}{p} + \frac{T_r}{p} \tag{18}$$

Each of the four terms in Equation (18) represents the contribution to t_t from a distinct component:

$\frac{c}{p} nk_dT_h$ represents the contribution from *variable message overhead*—those components of message latency that increase with increased communication distance.

$\frac{c}{p}B$ represents the contribution from *fixed message overhead*—those components of message latency that are fixed with respect to changes in communication distance.

$\frac{T_f}{p}$ represents the contribution from *fixed transaction overhead*—those components of transaction latency that are constant with respect to message latency.

$\frac{T_r}{p}$ represents the contribution from *actual CPU cycles*.

Of these four components, only variable message overhead increases with communication distance. Thus the exploitation of physical locality can only affect the contribution of variable message overhead.

Figure 8 plots the contribution of each of these components for ideal and random mappings of the application of Section 3 on a 1,000 processor machine for one, two, and four hardware

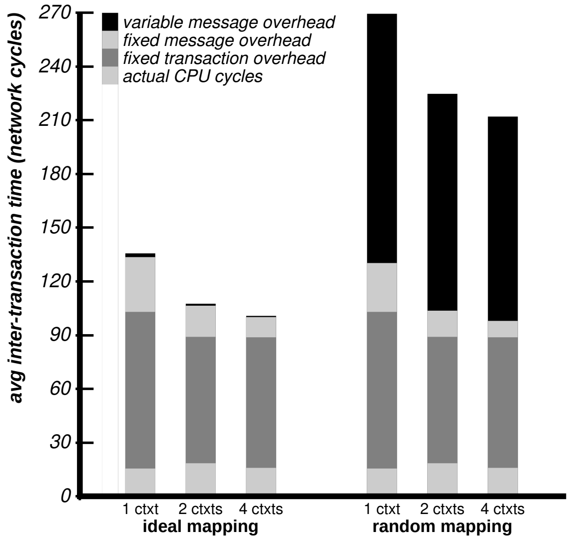

Figure 8: Breakdown of factors contributing to average inter-transaction issue time t_t for ideal and random mappings (1000 processors, $k = 10$, $n = 2$).

contexts. In moving from ideal to random mappings, the absolute contribution of fixed components declines slightly (no more than a few cycles)[6], while the contribution of variable message overhead increases drastically, as expected. Because these drastic increases only serve to bring the contribution due to variable message overhead on par with that from the fixed components, their net impact on inter-transaction issue time is limited to a factor of two or so.

In each of the six cases shown in Figure 8, fixed transaction overhead constitutes around two-thirds of the total fixed component. While this might seem inordinately large, in absolute terms, it corresponds to around 1 or 1.5 μs, assuming the timings discussed in Section 3. Further, recall that for this application an average transaction requires three or four network messages. This fixed overhead of 1 or 1.5 μs includes not only the sending and receiving overhead for those messages, but also any memory accesses and cache coherency processing necessary to complete a transaction.

When the relative balance of computation speed and communication bandwidth isn't as skewed in favor of communication as for the architecture described in Section 3, the fraction of inter-transaction time due to fixed transaction overhead and actual CPU cycles is reduced. Such a reduction increases the relative magnitude of increases in communication costs, thus allowing greater gains when physical locality is exploited.

Table 1 shows the impact of reducing network speed relative to processor speed by factors of two, four, and eight for the one context application for thousand- and million-processor machines. Similar results are obtained with two and four hardware contexts. As expected, the greater the relative cost of communication, the greater the benefit of exploiting physical locality.

Increasing the network dimension n affords both shorter average communication distances for random mappings on machines of a given size (see Equation (17)) and lower values of T_h for large machines (see Equation (16)). Because both of these effects

[6] This decline is due to decreased contention for processor-network channels, as discussed in Section 2.4.

Network Speed	Expected gain for 10^3 processors	Expected gain for 10^6 processors
2x faster	2.1	41.2
same	3.1	68.3
2x slower	4.5	101.6
4x slower	5.9	134.3

Table 1: Impact of reducing relative network speed on expected gains. Network speed is measured relative to processor speed; "2x faster" represents the architecture described in Section 3.

reduce communication latencies when random mappings are used without changing the situation when ideal mappings are used, the impact of exploiting physical locality on end performance is lower when higher dimensional networks ($n > 2$) are used.

5 Related Work

In [5], Chittor and Enbody present data obtained from running experiments similar to those described in Section 3 on the Ametek 2010, a distributed-memory, mesh-connected multiprocessor somewhat similar to that used in this paper. For the sizes of machines measured (up to 144 nodes), they note that the effect of contention for network resources, while observable, does not significantly impact end performance. From their measurements, they extrapolate that the impact of network contention will be far more substantial as machine sizes scale. Both conclusions are well borne out by the model presented in this paper.

On a different tack, Scherson and Corbett [11] introduce a framework for bounding the maximum expected speedup of different types of applications running on mesh-connected multiprocessors. Because they assume that communication latency is proportional to communication distance, the results presented in this paper lend credence to their model.

Finally, Agarwal [1] presents the network model described in Section 2. He uses the model to demonstrate that exploitation of physical locality enables networks to provide lower message latencies for a given message injection rate. However, because fixed message rates are assumed, independent of observed message latencies, the absolute conclusions drawn about the impact of exploiting physical locality are only appropriate for applications that are completely insensitive to changes in communication latency.

6 Summary

For multiprocessors with NUCL interconnection networks, physical locality will become an important element of the performance equation as machine sizes continue to increase. This paper develops a framework that allows accurate modeling of these effects. Using this framework, we obtain rough upper bounds on the performance impact of exploiting physical locality in architectures utilizing two-dimensional mesh interconnects.

We introduce a novel application model that characterizes application and processor behavior with three parameters relating to computation grain, latency sensitivity, and physical locality. We present a framework for combining this model with models for communication mechanisms and interconnection networks. Behavior predicted by the combined model agrees closely with that observed in detailed simulations of a complete multiprocessor system.

Previous studies of interconnection network behavior (e.g. [1]) fail to account for the feedback between networks and applications. We show that when the number of communication transactions each multiprocessor node can have outstanding is bounded, this feedback prevents processors from loading k-ary n-dimensional mesh interconnection networks to a point where communication latencies become unbounded. In fact, we show that when this condition is met, average per-hop communication latency approaches a constant value, the value of which is determined by average message size B, an application's latency sensitivity s (which is proportional to the average number of transactions a node keeps outstanding), and the network dimension n. Further, we demonstrate that this limiting value can be approached on machines with a few thousand processors. Finally, because average per-hop latencies are limited in this manner, we conclude that exploiting physical locality to reduce average communication distance can only provide performance improvements that are linear in the factor by which communication distance is reduced.

Using the modeling framework, we show that for two-dimensional mesh interconnects and the architecture used in the simulation experiments (in which processors are clocked half as fast as network switches), upper bounds on the performance improvement from exploiting physical locality are around two for machines with 1,000 processors and roughly 50 for machines with a million processors. Decreasing the relative network speed leads to larger upper bounds—slowing the network by a factor of eight increased these bounds by roughly a factor of three.

Finally, Figure 8 demonstrates the conventional maxim that fixed communication overheads must be small to support efficient execution of applications with small computation grain. The figure also raises an interesting corollary to this bit of common wisdom: The smaller the computation grain that can be efficiently supported, the larger the potential benefit of exploiting physical locality.

We believe that these basic results, suitably modified, will continue to hold for architectures using interconnection network topologies other than the k-ary n-dimensional meshes used in this paper.

7 Acknowledgments

The research reported in this paper is funded in part by NSF grant # MIP-9012773, in part by DARPA contract # N00014-87-K-0825, and in part by a NSF Presidential Young Investigator Award.

A Nomenclature

n mesh network dimension
k mesh network radix (side length)
N total number of processors

T_r average thread run length between successive communication transactions
s latency sensitivity
d average communication distance

p degree of hardware multithreading
T_s context switch time
c number of messages in "critical path" of a transaction
g average number of messages per transaction
T_f fixed component of transaction model

T_t average transaction latency
t_t average per-processor inter-transaction issue time
r_t average per-processor transaction issue rate

T_m average message latency
t_m average per-processor inter-message injection time
r_m average per-processor message injection rate

B average message size (in flits)
k_d average distance a message travels in each dimension
ρ network channel utilization
T_h average per-hop message latency

References

[1] Anant Agarwal. Limits on Interconnection Network Performance. *IEEE Transations on Parallel and Distributed Systems*, pages 398–412, October 1991.

[2] Anant Agarwal, David Chaiken, Kirk Johnson, David Kranz, John Kubiatowicz, Kiyoshi Kurihara, Beng-Hong Lim, Gino Maa, and Dan Nussbaum. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. Technical Report MIT/LCS/TM-454, MIT Laboratory for Computer Science, June 1991.

[3] Anant Agarwal, Beng-Hong Lim, David Kranz, and John Kubiatowicz. APRIL: A Processor Architecture for Multiprocessing. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 104–114, June 1990.

[4] David Chaiken, John Kubiatowicz, and Anant Agarwal. LimitLESS Directories: A Scalable Cache Coherence Scheme. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pages 224–234, April 1991.

[5] Suresh Chittor and Richard Enbody. Performance Evaluation of Mesh-Connected Wormhole-Routed Networks for Interprocessor Communication in Multicomputers. In *Proceedings of Supercomputing '90*, pages 647–656, November 1990.

[6] William J. Dally. Performance Analysis of *k*-ary *n*-cube Interconnection Networks. *IEEE Transations on Computers*, pages 775–785, June 1990.

[7] Kirk Johnson and Anant Agarwal. The Impact of Communication Locality on Large-Scale Multiprocessor Performance. Technical Report MIT/LCS/TM-463, MIT Laboratory for Computer Science, February 1992.

[8] Clyde P. Kruskal and Marc Snir. The Performance of Multistage Interconnection Networks for Multiprocessors. *IEEE Transactions on Computers*, pages 1091–1098, December 1983.

[9] Dan Nussbaum and Anant Agarwal. Scalability of Parallel Machines. *Communications of the ACM*, pages 56–61, March 1991.

[10] G. N. S. Prasanna. Structure Driven Multiprocessor Compilation of Numeric Problems. Technical Report MIT/LCS/TR-502, MIT Laboratory for Computer Science, April 1991.

[11] Isaac D. Scherson and Peter F. Corbett. Communications Overhead and the Expected Speedup of Multidimensional Mesh-Connected Parallel Processors. *Journal of Parallel and Distributed Computing*, pages 86–96, January 1991.

[12] Michael E. Wolf and Monica S. Lam. A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 30–44, June 1991.