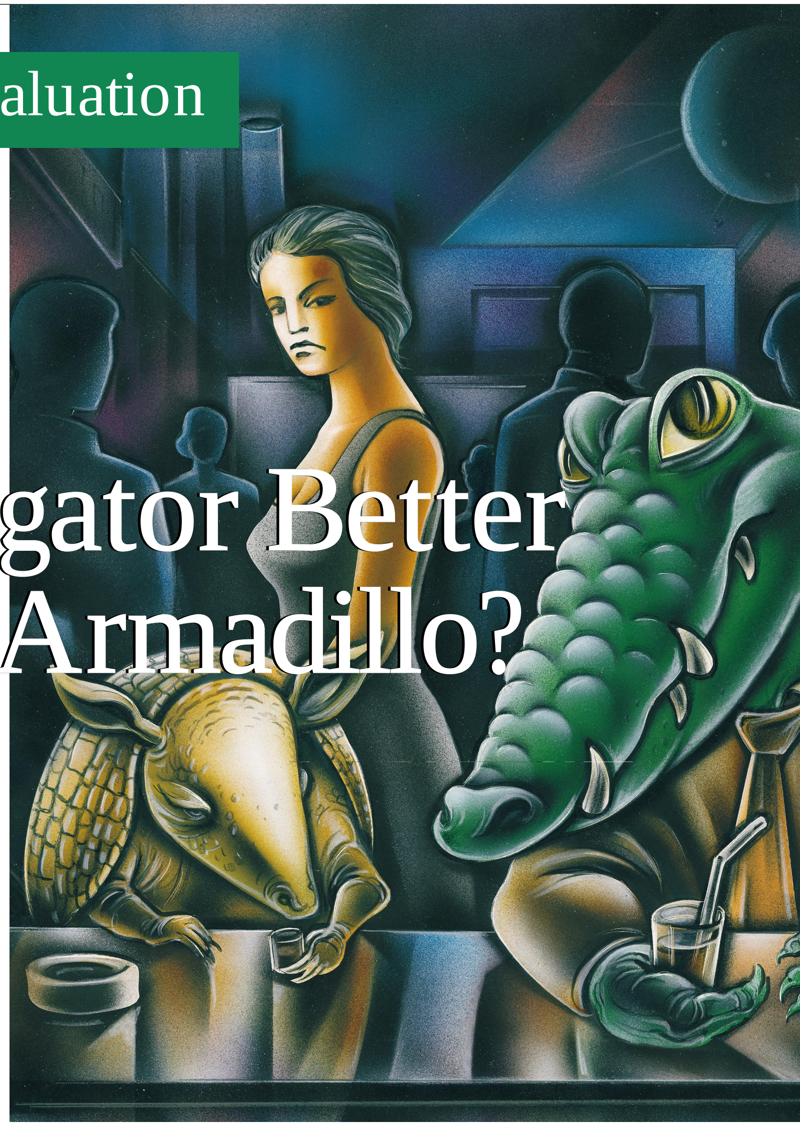
Problems with Comparing Interconnection Networks

Is an Alligator Better Than an Armadillo?



Kathy J. Liszka
The University of Akron
John K. Antonio
Texas Tech University
Howard Jay Siegel
Purdue University

Network effectiveness differs considerably across applications and operating environments, but even if these variables were fixed, cost and performance metrics must be chosen. Scientifically determining the best network is as difficult as saying with certainty that one animal is better than another.

Designing a parallel machine would be much easier if one interconnection network were "best" for all applications and all operating environments (including hardware, software, and financial factors). Unfortunately, no such network exists. Furthermore, even for a fixed application domain and a fixed operating environment, selecting the best network may be difficult because many cost and performance metrics could be used.

Suppose someone asked you to select the best animal. What features would you use to compare, say, an alligator and an armadillo? In some ways, the two are very similar: both have four legs, a rugged exterior, and sharp claws. However, in other ways the two are very different: one prefers a marshy environment, the other dry land; one has a long tail, the other a short tail; one is a reptile, the other a mammal. Which of the two, then, is a better animal and what makes it better? Is it which one could win a fight? Alligators are generally bigger, but how do you incorporate their need for more food because they are bigger? Moreover, what is a fair comparison of their ability to fight? Must you consider two armadillos against one alligator? If so, should their combined weight equal that of the alligator? Or should they be end-to-end the same length?

1063-6552/97/$10.00 © 1997 IEEE

Network parameters: characteristics that define and shape interconnection networks

As we describe in the main text, the task of fairly comparing two networks is complicated by the many parameters that characterize them. The list below is only representative, not exhaustive; the same is true of the supporting references. Both are meant to give some idea of the many factors that can vary across networks.

Network use

The way in which a network is used is critical to consider when measuring network performance. A network could be used such that only a fraction of the network's peak performance is achieved. Experienced programmers attempt to design parallel code so that their application programs use the network in ways that are well-suited for the target system.

Method of task decomposition and mapping

Both the method used to decompose a task and the method used to map subtasks onto the processors can affect network loading characteristics.[1] Consider the problem of how to best map eight subtasks, say S_0, ..., S_7, onto eight processors, P_0, ..., P_7. Assume the required communications occur at approximately the same time and that the communication pattern among subtasks is that shown in Figure A1. Also assume the processors are interconnected as the three-dimensional hypercube in Figure A2. If subtask S_i is mapped to processor P_i, some paths for the required communications will use multiple links of the hypercube, as Figure A3 shows. An alternative mapping, such as that in Figure A4, where each required path contains only one link, can use the available communication resources more effectively.

Processor and memory speed

The speed of a processor or a memory can affect the rate at which data can be moved into the network. If the speed is too low, the network's peak performance may not be realized.

Modes of parallelism supported

The amount of overhead processing (such as that associated with message protocols) for interprocessor communications depends on the mode of parallelism used. In SIMD mode, during a typical transfer each enabled processor sends a message to a distinct processor, implicitly synchronizing the `send` and `receive` commands and implicitly identifying the message. In contrast, MIMD mode requires overhead to identify which processor sent the message, to specify what the data item is, and to signal when a message has been sent and received.

Frequency of communications

This parameter quantifies the rate at which messages are generated for transmission through the network. For theoretical analyses and simulation studies, the frequency of communications is often assumed to be a random quantity with a known probability distribution. For real applications, the frequency of communications can vary during execution and depend on the input data in a very complicated way, making it difficult to model accurately.

Message-size range and distribution

In general, network performance depends on the distribution and range of message sizes to be transmitted, which is often difficult to predict for a given application because it may be data dependent. For simplicity, researchers often assume a uniform distribution of message sizes to conduct simulation studies. However, this assumption may not always be accurate.

Message destination assumptions

Will the number of received messages be uniformly distributed among the destination ports? Are the destinations for messages from a given source uniformly distributed? Will the message destinations favor a subset of locations, possibly causing a nonuniform use of network links?[2] To determine the amount of interference in the network, designers must consider these types of issues.

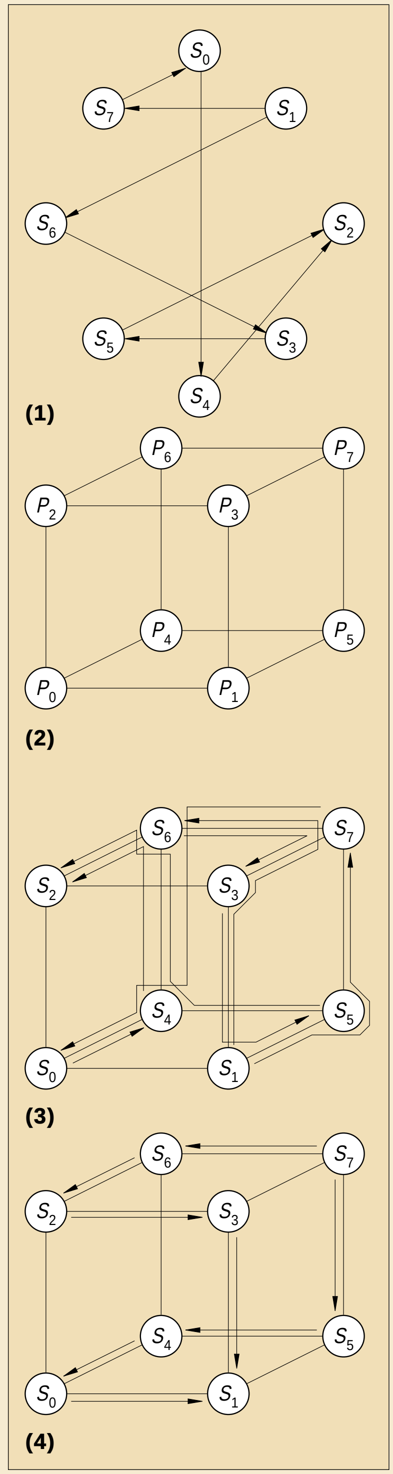


Figure A. (1) Sample communication pattern among eight subtasks in (2) a three-dimensional hypercube. (3) The result of assigning subtask S_i to processor P_i. (4) The result of a more efficient mapping of subtasks to processor P_i.

(*Continued on page 20*)

Hot-spot traffic

In this type of nonuniform traffic, messages from distinct sources are destined for the same network output port at approximately the same time.[3] Applications that use barrier-type synchronizations may create hot spot traffic. Performance degradation caused by hot spot traffic is due primarily to the contention for common network resources among the messages converging toward the common destination port. A secondary form of performance degradation occurs because hot-spot traffic can affect background traffic—network traffic not destined for the hot destination port.[4]

Routing control

Routing-control schemes aid in establishing network connections for interprocessor communication. Approaches vary in time and space requirements, capability, and flexibility. In using any of these schemes, designers must consider both deadlock avoidance and the time required to compute the routing control information.

Distributed

The routing decision is made locally at each switch on the basis of a routing tag in the header of the message. When a message arrives at a switch, the switch determines if it is the intended destination or, if not, onto which output link the message should be forwarded.

Centralized

A single control mechanism sets the connection patterns in all switches simultaneously. No local routing decisions are made at any switch. Centralized routing mechanisms are sometimes used for SIMD environments. Centralized control can simplify the hardware design for a SIMD machine, but it is less flexible than distributed control. Compute time tends to be higher because all the switches must be set.

Adaptive

This scheme is suitable for networks that have multiple paths between source-destination pairs, such as a hypercube or a multipath multistage network.[5] When local network congestion exists, a message may be dynamically rerouted through another, less congested, part of the network.[6] The performance improvement provided by adaptive routing typically comes at the cost of more complex routing protocols than nonadaptive routing.

Table-based

One approach is to store a lookup table at each switch.[7] A common implementation indexes each table by destination number and returns the output channel (number) required to establish the next link in the path. A table-based scheme provides great flexibility, but at the cost of table storage and lookup time at each switch.

Switching methodology

There are two basic switching mechanisms for transmitting messages in a network: circuit switching and packet switching. Combinations and variations of these two approaches have also been proposed and used. In general, approaches differ in how they treat the I/O buffering required at the switches

Circuit switching

In circuit switching, a dedicated path is established from the source to the destination and maintained for the entire message transmission. Because the path is dedicated, the transmission encounters no contention once the path is established. However, the establishment of the path may be delayed if other paths are already using links common to that path.

Packet switching

In packet switching, also called message switching or store-and-forward switching, messages are split into packets, which are sent through the network.

These kinds of comparison questions apply to interconnection networks. Suppose you are comparing the average message delay, for a given set of traffic conditions, for a hypercube network and a mesh network. Hypercube networks may involve more links than meshes. How do you incorporate that hypercube networks may require more complex hardware? Should the total path width of all the links the networks employ be the same? Or should the two networks require the same number of transistors per switch?

In this article, we explore the problems of determining which metrics or weighted set of metrics designers should use to compare networks and how they should apply these metrics to yield meaningful information. We also look at problems in conducting fair and scientific evaluations.

Choosing and applying metrics

In choosing and applying metrics, designers must answer three questions:

- What are the relevant metrics for my application domain and operating environment?
- How should I weight each metric?
- How can I make two network designs of equal "cost" so that I can compare them "fairly"?

Metrics and weighting issues

There are many ways to characterize and measure the performance of interconnection networks. We provide a representative subset of metrics and briefly list issues to consider in weighting them.

Latency

When data is transferred among the interconnected processors and memories, delays through the interconnection network can cause processor idle times, which can ultimately degrade overall system performance. Multitasking can help limit this source of processor idle time, but it may not improve the total execution time of any single task. The way you choose to characterize latency—for example, maximum or average—depends on the application.

Each packet has a header that contains routing information. One common packet-switching scheme is *output buffering*. In this scheme, when a packet arrives at an intermediate switch, its header is stored in an input latch. The output port of the switch is determined, and the packet proceeds to the appropriate output buffer, which holds the entire packet. Thus, a message does not block an entire path through the network, as in circuit switching.

Virtual cut-through

Like packet switching, virtual cut-through uses an input latch to receive and store a packet's header. It then checks the channel to the next switch for availability. If the channel is free, the packet bypasses, or "cuts through," the output buffer and proceeds to the next switch; otherwise, the entire packet is stored in the output buffer, as in normal packet switching.[8]

Partial cut-through

Partial cut-through is a variation of virtual cut-through. It operates on the same principle of attempting to bypass the output buffer if the channel is free and otherwise storing the packet in the buffer. However, with partial cut-through, if the channel becomes available during buffering, the packet can start transmitting to the next switch at that point in time.

Wormhole

Wormhole is another variation of virtual cut-through. In one approach to wormhole routing, a message is initially divided into packets, as is done in packet switching, and the packets are further divided into *flits* (flow control digits).[9] The first flit (the header, which contains the routing information) "cuts through" by proceeding directly to the next switch in the route if the channel is free. The trailing flits flow through the same path in the network, following the header flit in a pipeline fashion. If an output channel is not free when the header arrives at a switch, the header is blocked at the switch, as are the flits following the header. Once all channels are acquired along the route, the path is dedicated for the entire transmission of all the flits in the packet.[10]

Conflict resolution

For all switching mechanisms, a given switch output channel may be required by two or more messages at the same time. This is called a *conflict*. The *conflict-resolution scheme* detects and resolves conflicts by determining which requests should be deferred so that the remaining messages can proceed.

Combining

This packet-switching enhancement can lessen the performance degradation caused by situations such as hot-spot traffic (defined earlier). For example, the combining process can merge two read requests for the same physical memory location into one that continues on to the associated destination port, instead of sending both the original requests. Combined messages can also be combined. This feature increases the complexity of the network switch, but may significantly affect performance when shared variables are used for tasks that require large numbers of processors to access them at approximately the same times—for example, a job queue or a semaphore (synchronization primitive).[11] Another form of combining involves reduction operations.

HARDWARE IMPLEMENTATION

It is generally difficult to predict how varying a single hardware implementation parameter will affect network performance. Also, it is very complicated and costly to develop accurate simulations for (or actually manufacture) two networks that differ in only one hardware implementation parameter. The difficulty of comparing two such net-

(*Continued on page 22*)

Fault tolerance

The network's environment (an inaccessible satellite versus a university laboratory, for example) influences how important fault tolerance will be. To quantify metrics such as the number of faults tolerated and any degradation that results from a fault, you must establish a common fault model and a common fault-tolerance criterion.[1] The *fault model* characterizes the types of faults considered (such as permanent versus transient). The *fault-tolerance criterion* is the condition that must be met for the network to tolerate the fault. For example, in a multistage network, a fault may be considered tolerated if a blocked message can be sent to an intermediate destination and then to the final destination in a second pass.

Bisection width

This is the minimum number of wires that must be cut to separate the network into two halves. Larger bisection widths are better when data in one half of the system may be needed by the other half to perform a calculation.

Permuting ability

For a system of N processors, a *permutation* is a set of source-destination pairs that may be mathematically represented as a bijection from the set $\{0, 1, ..., N-1\}$ onto itself. In this context, a permutation is the transferring of a data item from each processor to a unique other processor, with all processors transmitting simultaneously. Different networks can require different amounts of time to process various permutations.[2] Permutations can occur in MIMD (multiple-instruction stream, multiple-data stream) machines when a communication is preceded by a barrier synchronization, and in SIMD (single-instruction stream, multiple-data stream) machines.

Control complexity

A Benes network can be set to perform any permutation in one pass;[3] the multistage cube may require multiple passes. However, a multistage cube network may be controlled in a distributed fashion by using the address of the destination processor;[2] a Benes network requires a works is compounded by the strong interdependence among the various hardware implementation parameters. For example, changing the maximum number of pins per chip may also affect the physical chip size, which may affect the number of chips that can be placed on a fixed-size printed-circuit board, which affects the number and physical length of connections among the PCBs and chips.

Implementation technology
The implementation technology relates to the physical materials and production methodology used to manufacture the chips and boards. Although implementation using a relatively expensive material or technology may enable certain chips to run faster, designers must consider the power requirements of the chips and their interaction with the other components. Several chip-attachment methods are also possible, which affect packaging costs, circuit use, circuit delay times, and overall system cost.

Design details
The design of a network includes the logic-level description for the switches, interfaces, and controllers. Factors to consider in logic circuit design are clock frequency, communication delay, partitioning for multichip configurations, and fault tolerance.

Number and width of communication links
For a given interconnection scheme (such as a hypercube), increasing the width of the channels increases the fraction of a message that can be transmitted in one network cycle. This increase comes at the "expense" of increasing the number of wires and the overall wiring complexity.[12] Suppose you have a fixed *network width*, defined here as the product of the number of links and the width per link. In some application domains, throughput may be better if you use a four-nearest-neighbor mesh with relatively wide communication links instead of a hypercube with relatively narrow communication links. In other application domains, the reverse may be true.

Packaging constraints
Packaging constraints include the physical size of the chips, required power consumption, circuit delay, reliability, heat dissipation, interconnection density, chip configuration, interboard distances, and production costs.[13] For example, if you use the system in an environment that has little extra space for cooling fans, required power consumption and heat dissipation can become critical constraints.

NUMBER OF PINS PER CHIP
Increasing the number of pins on a chip increases the space needed on the board to fan out the signals to and from the chip. However, using too few pins on a chip may necessitate multiple-chip configurations that can, in turn, create more complex (and less reliable) chip-interconnection schemes. One study compared a variety of multidimensional meshes (including the hypercube) under a particular set of pin-out constraints.[12] This involved examining the balance between the number of channels associated with a processor and the width of each channel.

Number of chips per board
Depending on the design choices, a particular network implementation may have multiple switches on a single chip, have one switch per chip, or require several chips for a single switch. When multiple chips are used, the designer must consider the physical sizes of the chips, the available area of each board, the connection complexity among the chips, mounting configurations for the boards, and so on.

Number of layers per board
To accommodate the required connec-

centralized control scheme (see sidebar, "Network parameters: characteristics that define and shape interconnection networks") and O($N \log N$) time to compute the network switch settings (for N processors).

Partitionability
Some networks may be partitioned into independent subnetworks, each with the same properties as the original network.[4] Partitionable systems include multiple-SIMD machines (such as the TMC CM-2[5]), MIMD machines (such as the IBM SP2[6]), and reconfigurable mixed-mode machines (such as PASM, a partitionable SIMD/MIMD machine[7]). Partitionability allows multiple users, aids fault tolerance, supports subtask parallelism, and provides the optimal size subset of processors for a given task.[8]

Graph theoretic definitions
You can use graph theory parameters, such as degree and diameter, to quantify network attributes. The graph nodes represent multistage network switches or single-stage (point-to-point) network processors; the arcs represent communication links. *Degree* is the number of communication links connected to a processor or switch. *Diameter* is the maximum value, across all node pairs, for the shortest distance (number of links) between any two nodes. Designers must carefully consider the significance and implications of a given performance metric such as diameter. The diameter of the twisted cube is approximately half that of the hypercube, yet in some cases the hypercube can have a smaller message delay.[9]

Unique path vs. multipath
Some networks, such as a multistage cube, have only one path between a given source and a given destination. Others, such as the hypercube and extra stage cube,[1] have multiple paths between source-destination pairs. Multipath networks may be more costly and difficult to control than unique-path networks, but they offer fault tolerance and enable routing around busy links. This demonstrates how improving

tions among multiple chips on a PCB, designers can use multiple layers to implement nonplanar connection patterns. To determine the number of layers to use, designers must consider the trade-offs among design complexity, wire lengths, and required area. Using more than two layers generally requires sophisticated wiring schemes or layout heuristics, which can increase design time and cost.

Manufacturability

When designing for a commercial product, being able to manufacture the network economically and in a reasonable time are important issues. Some possible considerations include component availability, use of custom versus off-the-shelf chips, time to assemble and test parts, wiring rules, and wire lengths.

Scalability

This parameter refers to the range of sizes (as measured, for example, by the number of the network's external I/O ports) that are appropriate for a given set of implementation parameters. In commercial markets, in which a wide range of network sizes may be required, implementation choices may improve scalability. In general, however, implementations designed for a wide range of network sizes may be achieving scalability at the expense of degraded performance for a given size in the range.

References

1. V. Chaudhary and J.K. Aggarwal, "A Generalized Scheme for Mapping Parallel Algorithms," *IEEE Trans. Parallel and Distributed Systems*, Mar. 1993, pp. 328–346.
2. T. Lang and L. Kurisaki, "Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks," *J. Parallel and Distributed Computing*, Sept. 1990, pp. 55–67.
3. M. Kumar and G.F. Pfister, "The Onset of Hot Spot Contention," in *Proc. Int'l Conf. Parallel Processing*, IEEE Computer Society Press, Los Alamitos, Calif., 1986, pp. 28–34.
4. M. Wang et al., "Using a Multipath Network for Reducing the Effects of Hot Spots," *IEEE Trans. Parallel and Distributed Systems*, Mar. 1995, pp. 252–268.
5. D. Rau, J.A.B. Fortes, and H.J. Siegel, "Destination Tag Routing Techniques Based on a State Model for the IADM Network," *IEEE Trans. Computers*, Mar. 1992, pp. 274–285.
6. P.T. Gaughan and S. Yalamanchili, "Adaptive Routing Protocols for Hypercube Interconnection Networks," *Computer*, Vol.26, No. 5, May 1993, pp. 12–23.
7. C.B. Stunkel et al., "The SP2 High-Performance Switch," *IBM Systems J.*, Vol. 34, No. 2, 1995, pp. 185–204.
8. P. Kermani and L. Kleinrock, "A Tradeoff Study of Switching Systems in Computer Communications Networks," *IEEE Trans. Computers*, Dec. 1980, pp. 1052–1060.
9. L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, Vol. 26, No. 2, Feb. 1993, pp. 62–76.
10. W.J. Dally and C.L. Sietz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, May 1987, pp. 547–553.
11. A. Gottlieb et al., "The NYU Ultracomputer—Designing an MIMD Shared-Memory Parallel Computer," *IEEE Trans. Computers*, Feb. 1983, pp. 175–189.
12. S. Abraham and K. Padmanabhan, "Performance of Multicomputer Networks under Pin-Out Constraints," *J. Parallel and Distributed Computing*, July 1991, pp. 237–248.
13. J.R. Nickolls, "Interconnection Architecture and Packaging in Massively Parallel Computers," *Proc. Packaging, Interconnects, and Optoelectronics for the Design of Parallel Computers Workshop*, 1992, pp. 4–8.

one network characteristic (fault tolerance) can degrade another (cost). If you use a weighted set of metrics, different weightings for fault tolerance and cost may result in a different decision about which network is best. A general methodology[10] has been developed to quantify the evaluation of network performance by constructing a weighted polynomial. However, fundamental to using this type of approach is deciding how to choose the weights.

Cost-effectiveness

This metric, or cost-performance ratio, is the network's performance divided by the cost to implement it. If cost-effectiveness is to be used as a scientific measure, it should be defined in terms of hardware required, rather than actual dollar cost, which can be easily changed by a new marketing policy rather than a technological advance. Although a commercial reality, it seems unscientific for one network design to become more cost-effective than another because one salesperson decided to give a temporary discount on a particular component to meet an end-of-year sales quota. However, measuring hardware cost without converting to dollars is awkward. What units *should* be used to add, say, the cost of optical links to the cost of memory for message buffers? A related problem is how exactly to determine if two network designs are indeed of equal cost. We address this later.

Other metrics

These include *throughput* (number of messages that arrive at their destinations during a given time interval), *multicasting time* (how long one processor takes to send the same message to multiple processors), *scalability* (range of machine sizes for which a network design is appropriate), and what the network will add to the host machine's volume, weight, and power consumption. Some network metrics are difficult to define quantitatively, but should be considered nonetheless. For example, ease of use is important from a practical viewpoint. Can the user easily understand and effectively use the network's important features? Can the user or compiler readily determine ways to use the network to obtain the best possible performance?

Making fair comparisons

A difficulty in comparing two network designs is determining if they are of equal cost so that the comparison is "fair." For example, for more than 16 processors, the number of links and switch complexity for a hypercube is greater than that for a four-nearest-neighbor mesh. To fairly compare the performance of these two networks, how should the mesh be augmented to make it of equal cost?

The problem of making fair comparisons is compounded by the numerous design and use parameters that may be varied. Network designers and implementers must understand how these parameters affect network performance. Major categories of parameters include network use, routing control, switching methodology, and hardware implementation. Designers must select values for one or more parameters within each category before they can effectively evaluate metrics. The sidebar, "Network parameters: characteristics that define and shape interconnection networks," gives specific examples in the four categories.

Varying only one network feature to isolate its impact can result in an unfair comparison due to the interdependency of the network features.

Evaluating networks

Having selected from numerous metrics and parameters, analysts must now find ways to apply them systematically. This consists of collecting additional insights through proven measurement tools and applying evaluation strategies.

Measurement tools

Measurement tools provide additional information about how the network can be expected to perform in its intended environment. Although these tools are valuable, analysts must apply them judiciously. The list of tools below is by no means exhaustive.

Mathematical models

These models provide important insight into basic network features and allow some degree of performance prediction. However, those modeling and analyzing networks for parallel processing systems often make assumptions for the sake of tractability that may not be totally realistic. When mathematically analyzing packet-switched networks, for example, analysts sometimes assume that the buffers for holding packets at each switch are of infinite length. The analytical results must be reasonable approximations of behavior when buffer sizes are fixed and realistic.

Statistically-based simulations

Such simulations characterize behavior on the basis of probabilistic models for network traffic. The complexity of network simulators varies depending on how closely they account for the implementation details of the networks they are simulating. Some simulators may account for the finite size of buffers for packet-switched networks, and thereby let analysts vary the "buffer size" parameter for different simulation studies. In contrast, other simulators may assume infinite buffer capacities, which simplifies the complexity of the simulator because it is no longer necessary to model the exception handling that occurs when data arrives at a full buffer.

Program tracing

This approach captures the application program's traffic pattern, such as the message source or destination and number of processor cycles elapsed between sending messages. Analysts can use trace information to determine (either through analysis or simulation) the effectiveness of executing the same program on a different machine. There is a potential problem, however. Using program tracing when communication patterns have been optimized for a network on a given machine may not allow a fair comparison when used to generate traffic patterns for another network.[11]

Benchmarking

This strategy involves executing a given task on several different machines and measuring and comparing the performance among systems. When using benchmarks to compare networks, analysts must remember that benchmark results are affected by many software layers and many implementation details of the whole parallel machine.[12] First, many algorithmic approaches may be used to implement a benchmark task. The mapping of the benchmark algorithm onto the processors of the system affects the communication patterns, which may affect

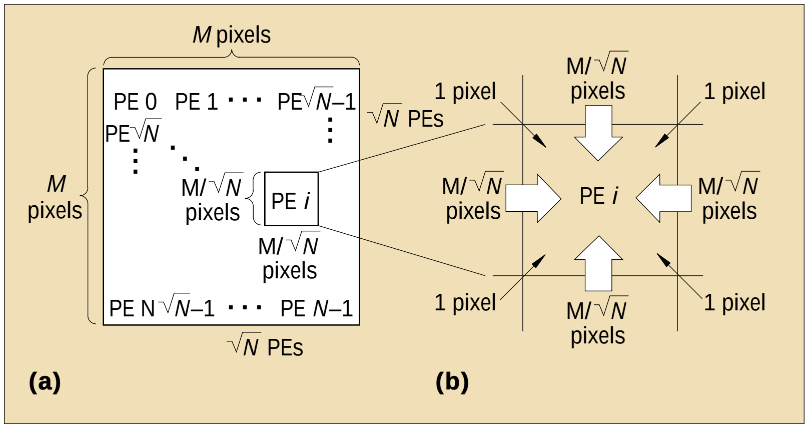


Figure 1. (a) Mapping an $(M / \sqrt{N}) \times (M / \sqrt{N})$ subimage onto each PE of a mesh and (b) pixels that must be received by PE i from neighboring PEs.

the delay characteristics. The language or compiler used to express or interpret the algorithm and interprocessor data transfers can significantly affect network performance. The measured performance is also a function of the operating system (for example, interprocessor communication protocols) that executes the algorithm.

The machines being benchmarked may vary considerably in hardware (such as technology used) and architecture (such as number of processors), which could also affect comparative network performance. Isolating the influence of the chosen network in the system becomes problematic, so interpreting benchmarking results must be done carefully.

Evaluation strategies

Given these measurement tools, how can designers fairly evaluate network features or application performance? Even after they have decided on which weighted set of metrics to use and have selected values for the various design and use parameters, they must still consider if it is possible to isolate the effect of changing a single feature. They must also select application implementations that will make the comparison fair.

Classical experimental method

The classical experimental method assumes that a given system is characterized by a set of parameters. To understand the effect a given parameter, say x, has on the system, all parameters except x are fixed and measurements and comparisons are made on the system. For example, suppose the goal is to determine the effect of network topology on performance, comparing a multidimensional mesh with a hypercube. Assume that the hypercube performs fastest using a distributed routing tag as a control scheme, and that the mesh performs fastest using a routing-table lookup as a control scheme at each switch node. Should you compare the networks with both using routing tags, both using table lookup, or each using what is best? If you use the same routing scheme for both, does this result in a fair comparison? What conclusions can you draw about the topological difference independent of the routing scheme? Furthermore, as we have shown, you must consider many other parameters in addition to the routing scheme if you want to fairly compare topologies.

If you compare the hypercube with routing tags and the mesh with table lookup, you invalidate the classical experimental method because two parameters are varied. Yet, without the classical experimental method, how can you isolate the effect of one network design or implementation parameter on performance?

This is an important dilemma that the interconnection network research community must begin to address.

Fixed mapping

Suppose two SIMD distributed-memory, parallel-processing systems differed only in the topology of their interconnection networks: one is a mesh, the other a ring. Both systems have N processing elements (processor-memory pairs), or *PEs*, labeled 0 to $N - 1$. Assume in the mesh topology that PE i is directly connected to PEs $i - 1$, $i + 1$, $i + \sqrt{N}$, and $i - \sqrt{N}$ In the ring topology, PE i has direct connections only to PEs $i - 1$ and $i + 1$. To simplify the presentation, we assume the number of data words sent and the number of links traversed determine the time for each data transfer between PEs (we could derive similar examples that consider network setup time, wormhole routing, and so on).

To apply benchmarking or the classical experimental method to compare the effects of the network topologies, suppose you decide to execute the same SIMD image-smoothing algorithm on both systems. The smoothing algorithm involves performing a smoothing operation for each pixel in an $M \times M$ image. Each smoothed pixel is the average value of the pixels in a 3×3 window centered around that pixel. Figure 1a depicts how pixel values from an $M \times M$ image are mapped onto N PEs interconnected as a $\sqrt{N} \times \sqrt{N}$ mesh, assuming square subimages. Each subimage is $(M/\sqrt{N}) \times (M/\sqrt{N})$ pixels. As the figure shows, the pixels are mapped onto the PEs so that adjacent subimages are mapped to adjacent PEs in the mesh.

During phase 1 of the algorithm, each PE executes the smoothing operation for those pixels within the interior (not along the edge) of its subimage. For this phase, each PE has all the pixel values required for all the smoothing operations in its local memory; thus no data transfers between PEs are required.

During phase 2, the $M / \sqrt{N}$ pixel values along each side of each square subimage are transferred to neighboring PEs, as Figure 1b shows. The four corner pixels from the diagonal neighbors are sent through intermediate PEs. The two pixel values needed by PE i from PE $i - \sqrt{N} + 1$ and PE $i + \sqrt{N} + 1$ will already have been transferred into PE $i + 1$, so only one transfer each is needed to move them from PE $i + 1$ to PE i. The situation for the two pixel values from the other diagonal neighbors is similar. Therefore, four transfers are needed for the diagonal pixels.

In phase 3 of the algorithm, the smoothing operation is applied to pixels on the boundary of each subimage.

Thus, during phases 1 and 3 combined, each PE performs M^2/N smoothing operations and all N PEs do this concurrently. For phase 2, $4(M / \sqrt{N}) + 4$ data transfers between PEs are required on the N PE mesh, where for each transfer up to all N PEs may send data simultaneously.[8]

For 16 PEs, Figure 2 illustrates the required data transfers to PE 6 for both the mesh and ring topologies. We assume that the image is distributed among the PEs for the ring network in the same way described earlier for the mesh (Figure 1).

Consider the number of inter-PE data transfers needed to smooth an $M \times M$ image using N PEs with a ring network. The $M / \sqrt{N}$ pixels along the right vertical boundary of each subimage must be transferred from each PE $i - 1$ to PE i. Likewise, with each left vertical boundary from PE $i + 1$ to PE i. To transfer the $M / \sqrt{N}$ pixels along the upper horizontal subimage boundary of each PE $i + \sqrt{N}$ to PE i requires $\sqrt{N}\,(M / \sqrt{N}) = M$ data transfers, because PE $i + \sqrt{N}$ and PE i are separated by $\sqrt{N}$ links. Likewise, to transfer the $M / \sqrt{N}$ pixels along the lower horizontal subimage boundary from each PE $i - \sqrt{N}$ to PE i. As with the mesh network, pixels from the diagonal neighbors can be sent through intermediate PEs. The total for the ring is $2M / \sqrt{N} + 2M + 4$ inter-PE data transfers. For $1 < \sqrt{N} \leq M$, this is greater than the required number of inter-PE data transfers for the mesh, given by $4(M / \sqrt{N}) + 4$. For example, if $M = 4{,}096$ and $N = 256$, then 8,708 inter-PE data transfers are required by the ring as compared to 1,028 for the mesh topology.

Is it fair to conclude that the ring topology is inferior to the mesh, even just for this application, as the above analysis seems to indicate? Consider the following variation in the mapping scheme for the previously described image-smoothing algorithm. Instead of dividing the $M \times M$ image into N square $(M / \sqrt{N}) \times (M / \sqrt{N})$ subimages, divide it into N rectangular $(M/N) \times M$ subimages.

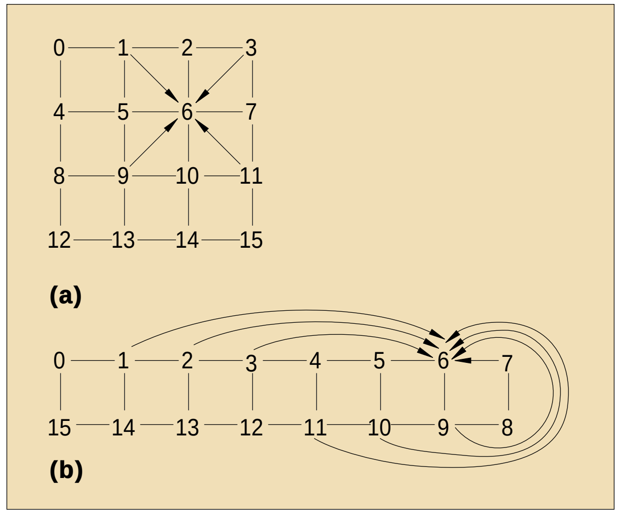


Figure 2. The source PEs for pixels needed by PE 6 for (a) the mesh and (b) the ring topologies for $N = 16$.

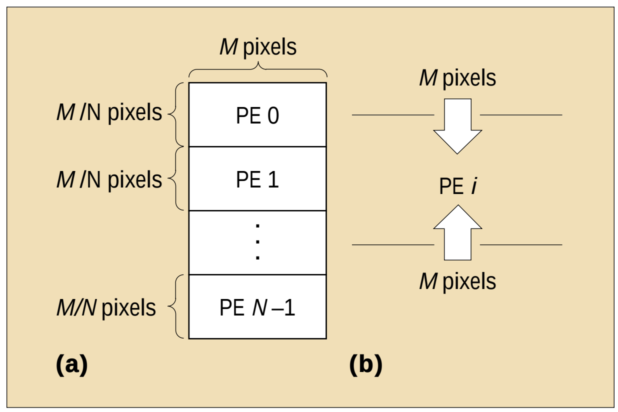


Figure 3. Mapping $(M / N) \times M$ subimages onto the PEs of the ring.

Figure 3 shows a mapping of pixel values from these N rectangular $(M/N) \times M$ subimages onto the ring topology. Phase 1 still smooths the interior pixels of each subimage. In phase 2, pixel values along the horizontal boundaries (top row and bottom row) of the rectangular subimages are transferred to neighboring PEs, requiring $2M$ inter-PE data transfers with the ring topology. Phase 3 still smoothes the subimage boundary pixels.

Consider if the N rectangular $(M/N) \times M$ subimages approach is used on a system with a mesh network without edge "wraparound" connections (for example, no direct connections from PE 3 to PE 0 in Figure 2a). Phases 1 and 3 are unchanged. In phase 2, the M pixels along each of the horizontal boundaries (top row and bottom row) of the rectangular subimages are transferred from PE $i + 1$ to

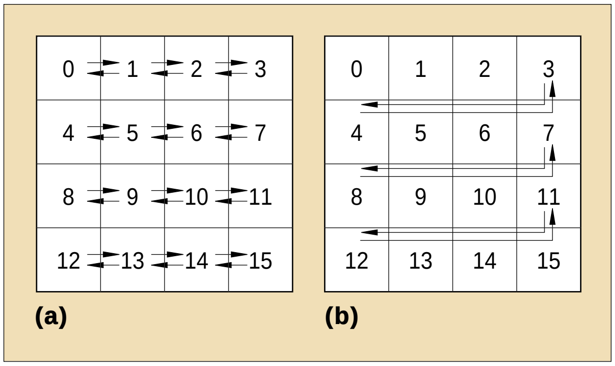


Figure 4. Mapping $(M/N) \times M$ subimages onto the PEs of the mesh with the directions of transfers shown: (a) $2M$ transfers; (b) $2(\sqrt{N}M)$ transfers.

PE i and from PE $i-1$ to PE i. Figure 4 shows the mapping of the $(M/N) \times M$ rectangular subimages onto a mesh. If PE i is directly connected to PEs $i-1$ and $i+1$, then $2M$ transfers are needed. However, PE 3 and PE 4, for example, need to exchange their boundary pixels, but have no direct connection. Thus, M pixels are transferred from PE $i\sqrt{N}-1$ (for example, 3) south to PE $(i+1)\sqrt{N}-1$ (for example, 7) and then west across the row of PEs to PE $i\sqrt{N}$ (for example, 4). This requires $M\sqrt{N}$ transfers. Another $M\sqrt{N}$ inter-PE data transfers are required to move the M horizontal boundary pixel values from PE $i\sqrt{N}$ to $i\sqrt{N}-1$. Therefore, the total communications required are $2M + 2M\sqrt{N}$ inter-PE data transfers for the mesh. This is much greater than the ring for this mapping. For example, if $M = 4{,}096$ and $N = 256$, the ring requires 8,192 data transfers, while the mesh with no wraparound connections requires 139,264 inter-PE data transfers. For this mapping strategy, the ring outperforms the mesh. Thus, if square subimages are used, the mesh is better; if rectangular (block of rows) subimages are used, the ring is better. The data allocation used determines which network is better.

The effect of the data allocation can have even more effect on system performance than just the number of data transfers. Consider image correlation, another window-based task, using a window of size 21×21. With rectangular subimages, PE i will need 10 rows of pixels from each of PEs $i+1$ and $i-1$. The ring will need $10 \cdot 2M$ transfers. With square subimages, PE i will need a 10-pixel-wide perimeter from each of its eight neighboring PEs. The mesh will need $10\,(4M/\sqrt{N}) + 10^2 \cdot 4$ transfers. For $M = 4{,}096$ and $N = 256$, the ring will require 81,920 transfers and the mesh will require 10,640 transfers, a difference of 71,280.

When considering the entire image, using a rectangular layout on the ring, there are 10 leftmost and 10 rightmost columns of pixels in all PEs that will not be processed (because they do not have the 21×21 window of pixels around them that is needed for the image-correlation process). Therefore, $10 \cdot 2(M/N)$, or 320, pixels will not require processing. (Using square subimages, there is no such savings, because a subset of the PEs will not contain edge pixels of the entire image.) If the time to process these 320 pixels is greater than the time to perform 71,280 inter-PE transfers, the ring network will result in a smaller total execution time than the mesh. For the example image-correlation task with a 21×21 window, calculations involving 440 neighbors are needed for each of the 320 pixels, so it is possible that the ring network will indeed be a better choice even when each network uses its best data layout.

Therefore, not only does the mapping of a task onto a parallel machine affect the number of transfer steps needed for different networks, but considerations such as the effect of avoiding processing edge pixels may affect overall performance and, hence, network choice.

We have shown that several fundamental questions remain open in evaluating and comparing interconnection networks:

- Which weighted collection of metrics should be used to evaluate network performance?
- How can the relative cost-effectiveness of different networks be determined using units other than dollars?
- Can analytically tractable theoretical models demonstrate sufficiently realistic behavior?
- How can benchmarks across different machines be devised to evaluate a particular network feature?
- With the strong interdependence among application, system, and network parameters, how can the classical experimental method be fairly applied to evaluate the effect of changing one parameter while holding all others constant?
- What parameters should be held constant (and with what values) when comparing and evaluating networks?
- In what sense can the implementation of two different networks be made "equal" for a meaningful comparison?

To establish a weighted set of metrics for a realistic environment, designers must formally incorporate quality of service. Trade-offs, such as fault tolerance versus network cost or number of processors supported versus physical volume, will affect how metrics are weighted. These quality of service factors also apply to the design of other subsystems of parallel machines.

While partial approaches to exploring network performance have been proposed,[9–11] no complete methodology exists. Research must be done to answer the above questions before we can quantify the relative importance of various network features. Only then may we be able to say which network is best for a given situation.

Acknowledgments

The authors thank Janet M. Siegel and Nancy Talbert for their comments, Muthucumaru Maheswaran for his assistance with the illustrations, and Marcelia Sawyers for her careful typing. A preliminary version of portions of this material was presented at The New Frontiers: A Workshop on Future Directions of Massively Parallel Processing. This research was supported in part by Rome Laboratory under contracts F30602-92-C-0150 and F30602-92-C-0108.

References

1. G.B. Adams III, D.P. Agrawal, and H.J. Siegel, "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks," *Computer*, June 1987, pp. 14–27.
2. D.H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers*, Dec. 1975, pp. 1145–1155.
3. V.E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, San Diego, 1965.
4. H.J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, 2nd ed., McGraw-Hill, New York, 1990.
5. L.W. Tucker and G.G. Robertson, "Architectures and Applications of the Connection Machine," *Computer*, Aug. 1988, pp. 26–38.
6. C.B. Stunkel et al., "The SP2 High-Performance Switch," *IBM Systems J.*, Vol. 34, No. 2, 1995, pp. 185–204.
7. H.J. Siegel et al., "The Design and Prototyping of the PASM Reconfigurable Parallel Processing System," in *Parallel Computing: Paradigms and Applications*, A. Y. Zomaya, ed., Int'l Thomson Computer Press, London, 1996, pp. 78–114.
8. H.J. Siegel, J.B. Armstrong, and D.W. Watson, "Mapping Computer-Vision-Related Tasks onto Reconfigurable Parallel-Processing Systems," *Computer*, Feb. 1992, pp. 54–63.
9. S. Abraham and K. Padmanabhan, "The Twisted Cube Topology for Multiprocessors: A Study in Network Asymmetry," *J. Parallel and Distributed Computing*, Sept. 1991, pp. 104–110.
10. M. Malek and W.W. Myre, "Figures of Merit for Interconnection Networks," *Proc. Workshop Interconnection Networks for Parallel and Distributed Processing*, IEEE Press, Piscataway, N.J., 1980, pp. 74–83.
11. A.A. Chien and M. Konstantinidou, "Workloads and Performance Metrics for Evaluating Parallel Interconnects," *IEEE CS Computer Architecture Technical Committee Newsletter*, Summer/Fall 1994, pp. 23–27.
12. S.H. Bokhari, "Multiphase Complete Exchange on Paragon, SP2, and CS-2," *IEEE Parallel & Distributed Technology*, Fall 1996, pp. 45–59.

Kathy J. Liszka is an assistant professor of mathematical sciences at the University of Akron. Her research interests are parallel algorithms and distributed computing. She received a PhD in computer science from Kent State University and is a member of the IEEE Computer Society, ACM, and Pi Mu Epsilon. Her address is the Dept. of Mathematical Sciences, The University of Akron, Ayer Hall 313, Akron, OH 44325-4002; liszka@uakron.edu.

John K. Antonio is an associate professor of computer science at Texas Tech University. His research interests include high-performance embedded systems, reconfigurable computing, and heterogeneous systems. He has coauthored more than 50 publications in these and related areas. Antonio received a BS, an MS, and a PhD in electrical engineering from Texas A&M University. He is a member of the IEEE Computer Society, and of the Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi honorary societies. He is the program chair for the 1998 Heterogeneous Computing Workshop, and has been on the Organizing Committee of the International Parallel Processing Symposium for the last five years. His address is the CS Dept., Texas Tech University, Box 43104, Lubbock, TX 79409-3104; antonio@ttu.edu.

Howard Jay Siegel is a professor of electrical and computer engineering and coordinator of the Parallel Processing Laboratory at Purdue University. His research interests include heterogeneous computing, parallel algorithms, interconnection networks, and the PASM reconfigurable parallel computer system. He received a BSEE and a BS in management from the Massachusetts Institute of Technology, and an MA, MSE, and a PhD in computer science from Princeton University. He is an IEEE Fellow and will be inducted as an ACM Fellow in 1998. Siegel has coauthored more than 240 technical papers, was a co-editor-in-chief of the *Journal of Parallel and Distributed Computing*, and served on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. His address is Parallel Processing Laboratory, ECE School, Purdue University, West Lafayette, IN 47907-1285; hj@purdue.edu.