Views of Mixed-Mode Computing and Network Evaluation

Howard Jay Siegel
John K. Antonio
{hj, jantonio}@ecn.purdue.edu
Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-1285, USA

Abstract

Trade-offs between the SIMD and MIMD models of architecture for parallelism are presented. Mixed-mode parallelism, where a machine can switch between the SIMD and MIMD modes of parallelism at instruction-level granularity with generally negligible overhead, is discussed. Advantages and disadvantages of mixed-mode parallelism, and an example of a mixed-mode parallel algorithm, are given. The relationship of mixed-mode processing to high-performance heterogeneous computing is overviewed. Difficulties involved with evaluating interconnection networks for parallel machines are then considered. There are a myriad of metrics that have been used in the literature. The problems involved with choosing the most appropriate metric or weighted set of metrics, and performing "fair" comparisons, are explored.

1: Introduction

Most large-scale parallel processing computers can be classified into two general architectural categories based upon the number of instruction streams active concurrently within the computational engine. Those parallel processing systems that execute a single thread of control are labeled single instruction stream, multiple data stream (SIMD) machines [14]. Those that have the capability of executing many separate threads of control are referred to as multiple instruction stream, multiple data stream (MIMD) machines [14]. Each of the SIMD and MIMD modes has advantages, some of which are overviewed in Section 2.

Mixed–mode parallel processing systems add a new dimension in that they are capable of executing instructions in both the SIMD and MIMD modes of parallelism, and can switch between the two modes at instruction-level granularity with very little overhead [13]. The goal

This research was supported by Rome Laboratory under contract number F30602-94-C-0022 and by NRaD under contract number N68786-91-D-1799. It used equipment supported by the National Science Foundation under grant CDA-9015696.

0-8186-6507/94 $4.00 © 1994 IEEE

of mixed-mode computing is to make available to the user many of the positive attributes of both SIMD and MIMD modes in a single machine. A list of machines incorporating mixed-mode parallelism, advantages and disadvantages of mixed-mode computing, and an example of the use of mixed-mode parallelism are given in Section 3.

Mixed-mode parallelism is one extreme form of heterogeneous computing, where a collection of modes of parallelism and/or different machine architectures are used together in a single machine or as a suite of interconnected machines. The goal of heterogeneous computing and the relationship of mixed-mode processing to general heterogeneous computing is discussed in Section 4.

One crucial component of any parallel machine is the interconnection network that provides inter-processor communication. A variety of networks have been implemented, and an even greater variety proposed. It is very difficult to compare different networks. Problems involved with choosing the most appropriate metric or weighted set of metrics for evaluating networks, and with performing "fair" comparisons by determining what networks are of "equal cost," are considered in Section 5.

2: Trade-offs Between the SIMD and MIMD Modes of Parallelism

The MIMD model includes a collection of N independent, tightly-coupled processors, each associated with a local memory, forming a processing element (PE). The processors each fetch and execute their own instruction stream, asynchronously with respect to one another. Synchronization must be explicitly programmed by the user. Also, the PEs have the capability to efficiently communicate via an interconnection network. The subclass of MIMD mode in which all PEs independently follow a single program is called single program, multiple data stream (SPMD) mode [11].

The SIMD model includes N PEs and a single control unit (CU). The PE's memory is only used to store data, not instructions. The CU is made up of a processor, memory, and an instruction broadcast queue. The queue broadcasts a single stream of instructions to the PEs in the computational engine. All enabled PEs execute the same instruction simultaneously, each on its own data. The queue will not issue the next instruction until all enabled processors have completed the current one, thereby implicitly synchronizing the PEs at the instruction level. Once the CU processor sends instructions to the queue, it may proceed with its own execution of operations while the queued instructions are broadcast to the PEs. Concurrent CU and PE execution is called CU/PE overlap.

There are many trade-offs between SIMD and MIMD modes. The advantages of SIMD mode include:

a) The single instruction stream and implicit synchronization of SIMD makes programs easier to create, understand, and debug. Also, as opposed to MIMD architectures, the user does not need to be concerned with the relative timings among the PEs.
b) In SIMD mode, the PEs are implicitly synchronized at the instruction level. Explicit synchronization primitives, such as semaphores, may be required in MIMD mode, and generally incur overhead.
c) The implicit synchronization of SIMD mode also allows more efficient inter-PE communication. If the PEs communicate through messages, during a given transfer all enabled PEs send a message to distinct PEs, thereby implicitly synchronizing the "send" and "receive" commands. The receiving PEs implicitly know when to read the message, who sent it, and why it was sent. MIMD architectures require the overhead of identification protocols and a scheme to signal when a message has been sent and received.

d) Control flow instructions and many scalar operations can be overlapped (i.e., executed concurrently) on the CU while the processors are executing instructions (this is implementation dependent); this is referred to as CU/PE overlap.
e) Only a single copy of the instructions needs to be stored in the system memory, thus possibly reducing memory cost and size, allowing for more data storage, and/or reducing communication between primary and secondary memory.
f) Cost is reduced by the need for only a single instruction decoder in the CU (versus one in each PE for MIMD mode).

The advantages of MIMD mode include:

a) MIMD is very flexible in that different operations may be performed on the different PEs simultaneously, i.e., there are multiple threads of control. Thus, MIMD is effective for a much wider range of algorithms, including tasks that can be parallelized based on functionality (i.e., MIMD can exploit data parallelism and functional parallelism, while SIMD is limited to the former [19]).
b) The multiple instruction streams of MIMD allow for more efficient execution of conditional statements (e.g., *if-then-else*) because each PE can independently follow either decision path. In SIMD mode, when conditionals depend on data local to PEs, all of the instructions for the *then* block must be broadcast, followed by all of the *else* block. The appropriate PEs are enabled for each block.
c) MIMD's asynchronous nature results in a higher effective execution rate for a sequence of instructions that take a variable amount of time to complete. That is, their execution time is data dependent (e.g., floating point operations on some processor architectures). In SIMD mode, a PE must wait until all the other PEs have completed an instruction before continuing to the next instruction; this is not required in MIMD mode.
d) MIMD machines do not have the added cost of a SIMD CU and the hardware for broadcasting instructions.

Much of the material in this section is summarized from [9]. The reader is referred to that paper and to [30] for more details and examples. Because both SIMD and MIMD modes have advantages, various mixed-mode machines have been proposed.

3: Mixed-Mode Parallelism

The mixed-mode model is similar to the SIMD model, except that the PEs have the capability of either fetching instructions from the instruction broadcast queue (SIMD mode) or from their own memory (MIMD mode). It is assumed that an efficient mechanism exists for switching between the two parallelism modes at instruction-level granularity with generally negligible overhead. A number of prototype mixed-mode machines have been built that incorporate, to varying degrees, the mixed-mode approach. These include PASM (Purdue University, USA) [30, 32], TRAC (University of Texas at Austin, USA) [24, 27], OPSILA (University of Nice, France) [5, 12], Triton (University of Karlsruhe, Germany) [18, 25], and EXECUBE (IBM Federal Systems Division, USA) [21].

Most of the advantages of each mode can be realized with a mixed-mode architecture that allows the most appropriate mode to be selected at each step in the execution of a program. Disadvantages of mixed-mode parallelism include higher hardware cost (because mixed-mode machines must have the hardware needed for both modes), more complicated use (because the mode switching ability adds another dimension of complexity for the programmer), and, when switching from MIMD to SIMD mode, some PEs may remain idle while they wait for the other PEs to reach the switch point (which they may not need to do if only MIMD mode was used).

As a very simple example of the use of a mixed-mode machine, consider the bitonic sorting [6] of sequences on the PASM prototype [13]. Assume there are M numbers and $N = 2^n$ PEs, where M is an integer multiple of N, that M/N numbers are stored in each PE, and that the M/N numbers within a PE are in sorted order. The goal is to have each PE contain a sorted list of M/N elements, where each of the elements in PE i is less than or equal to all of the the elements in PE k, for $i < k$. The regular bitonic sorting algorithm for $M = N$ is modified to accommodate the M/N sequence in each PE. As shown in Figure 1, an ordered merge is done between the local PE sequence X and the transferred sequence Y using local data conditional statements in merge(X,Y). The lesser half of the merged sequence is assigned the pointer X and the greater half is assigned the pointer Y. The pointers to the two lists may be swapped by swap(X,Y), based on a precomputed data-independent mask.

```
for k = 1 to log2N do
  { for i = 1 to k do
      { for q = 1 to M/N do
          { send X[q] to PE whose number differs in bit (k–i)
            Y[q] ← data received from network }
        merge(X,Y)
        swap(X,Y) } }
```

Figure 1: Bitonic sequence-sorting algorithm [13].

When choosing the mode of parallelism, the programmer must consider various characteristics of the algorithm. The ordered merge involves many comparisons, all of which can be more efficiently computed in MIMD mode. The innermost loop of the algorithm requires many network transfers, which are better performed in SIMD mode. In a mixed-mode implementation, the ordered merge and swap routines can be executed in MIMD mode, while the rest of the operations, including network transfers, are performed in SIMD mode. This approach has an advantage over pure SIMD or pure MIMD mode implementations because all comparisons are done in MIMD mode and all network transfers are done in SIMD mode. Additionally, there is potential in SIMD mode for overlapping operations done by the CU (i.e., loop index variable increment and compare) with operations done by the PEs (i.e., the loop body). It is shown in [13] that there is a noticeable improvement in execution time for the mixed-mode implementation. The mixed-mode results are shown to be the product of properties inherent to the modes of parallelism.

This simple example indicates some of the benefits of mixed-mode parallelism. Other examples of the use of mixed-mode parallelism include segmenting range data [17], using cyclic reduction [26], and performing singular value decomposition [33]. An example that demonstrates the potential disadvantage of mixed-mode when switching from MIMD mode to SIMD mode, mentioned earlier, is presented in [8]. The design and use of mixed-mode machines is an active research area both for the importance of mixed-mode as a model of parallelism, and for the relationship of mixed-mode parallelism to general heterogeneous computing, as discussed in the next section.

4: Heterogeneous Computing

A heterogeneous computing (HC) environment is a well-orchestrated and coordinated suite of high-performance machines that provides support for computationally intensive applications with diverse computing requirements [20]. An HC system includes a heterogeneous suite of machines, high-speed interconnections, interfaces, operating systems, communication protocols, and programming environments. HC is the effective use of these diverse hardware

and software components to meet the distinct and varied computational requirements of a given application. Implicit in this concept of HC is the idea that different subtasks with various machine architectural requirements are embedded in the applications executed by the HC system. The goal of HC is to decompose a task into computationally homogeneous subtasks, and then assign each subtask to the machine where it is best suited for execution [15, 16]. The set of subtasks may be executed as an ordered sequence and/or concurrently. In existing practical applications of HC, the user specifies the decomposition and, typically, the assignment of subtasks to machines. Doing this decomposition and assignment automatically is a long-term pursuit in the field of HC. The material in this section is summarized from [31]; the reader is referred to that book chapter for more information about HC.

Examples of existing HC systems can be divided into two categories: mixed-mode machines and mixed-machine systems [36]. Mixed-mode machines were discussed in the previous section. A mixed–machine system is a heterogeneous suite of independent machines of different types interconnected by a high-speed network. Unlike mixed-mode machines, switching execution among machines in a mixed-machine system requires measurable overhead because data may need to be transferred among machines. Thus, the mixed-machine systems considered for HC are assumed to have high-speed connections among machines that make decomposition at the subtask level feasible.

Mixed-mode machines are one extreme form of HC, where two different modes of parallelism are available in one machine. This is in contrast to mixed-machine HC systems, where a suite of machines can provide different modes of parallelism by having each mode in a different machine. Both types of heterogeneous systems can support tasks that include some subtasks that execute faster in SIMD mode and others that execute faster in MIMD mode. Decomposing a task for mixed-mode execution is easier than mixed-machine because the same PEs are used for both modes and, in general, no data has to be moved as a result of a mode change. This eliminates two major problems in the use of mixed-machine HC: moving data among machines and determining machine loads.

The study of mixed-mode machines and their use provides valuable information about the trade-offs between SIMD and MIMD parallelism, explores the advantages and disadvantages of mixed-mode computations, and provides a relatively simpler environment for developing algorithm mapping techniques that may possibly be adapted for the mixed-machine arena. For example, a block-based mode selection methodology developed for mixed-mode machines was then extended for use as a heuristic for the mixed-machine case [36]. Thus, mixed-mode machines are important for their advantages over single-mode machines and for their use in developing methodologies that may be adaptable for mixed-machine HC use.

5: Comparing Interconnection Networks

A myriad of networks have been proposed in the literature for providing the inter-PE communications in a parallel machine (e.g., see [10, 28, 35, 37]). Given this wide variety of approaches, there is much interest in a methodology for the direct comparison of networks with the goal of determining the "best" approach. The problem is one of determining which metric or weighted set of metrics should be used, and how the metrics should be applied to yield meaningful information. The material in this section is summarized from [29].

Some metrics commonly found in the literature include: message delay characteristics (minimum, maximum, average) (e.g., [1]), permuting ability (for SIMD mode) (e.g., [23]), partitionability (into independent subnetworks) (e.g., [28]), fault tolerance (and specifying the fault model and fault-tolerance criteria used) (e.g., [3]), bisection width (e.g., [4]), graph theoretic measures (e.g., diameter), throughput (minimum, maximum, average) (e.g., [34]), complexity of control (distributed (e.g., [22]) versus centralized (e.g., [7])), and cost-effectiveness. This

collection is only a representative subset of the metrics that have been proposed in the past; it demonstrates the range of metrics one can consider when evaluating networks. In general, it is difficult to select the "important" metrics, because they may vary with the intended application and operating assumptions. Basing design decisions on different sets of metrics will result in different design choices (e.g., diameter versus delay [2]).

Another difficulty in comparing two different network designs is determining if they are of equal "cost," so that the comparison will be "fair." The "cost" 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 by a technological advance. However, equating "costs" of different types of hardware without converting to dollars is awkward (e.g., how should one trade off the "costs" of additional optical links versus increased buffer size in a consistent and fair way?). The difficulty of determining "cost" also comes into play when attempting to use the "cost-effectiveness" metric; it seems unscientific for one design to become more cost-effective than another because a salesperson decided to give a discount.

The problem of performing fair comparisons is compounded by the numerous design and use parameters that may be varied in the comparison. These parameters include:

a) network loading parameters: method of task decomposition and mapping, frequency of communications, message sizes and distribution of sizes, message destination assumptions, nonuniform versus uniform traffic patterns, hot spots, assumed processor speed, assumed memory speed, modes of parallelism supported;
b) routing control parameters: distributed routing, centralized routing, table based routing, time required to compute;
c) switching methodology parameters: packet switching, circuit switching, virtual cut-through, wormhole routing, packet combining, conflict resolution schemes;
d) hardware implementation parameters: design details, implementation technology, number and width of communication links, packaging constraints, number of pins per chip, number of chips per board, number of layers on PC boards.

For comparison, which parameters should be held constant and which should be varied? What values or choices should be selected for the parameters that will be held constant? In the natural sciences, phenomena are often studied and new knowledge obtained by applying the experimental method. The experimental method assumes that a given system is parameterized by a collection of p variables. To understand the impact that a given parameter, say x, has on the system, all parameters except for x are fixed while x is varied and measurements and comparisons are made on the system.

Can the experimental method be used to evaluate and compare networks? If yes, then is it always possible to fix $p-1$ parameters in a meaningful way? If no, because it is desirable to use the "best" $p-1$ parameters for each experiment, then how can the differences among networks be meaningfully measured?

Consider an example of this problem. Assume, for the purposes of this discussion, that the goal is to compare a particular multi-dimensional mesh network topology with a hypercube network topology, that the hypercube performs fastest using a routing tag control scheme, and the mesh performs fastest using a table lookup control scheme at each switch node. Should the comparisons be made with both using routing tags, both using table lookup, or each using what is best? If the "use what is best" option is selected, how can the time and hardware costs for implementing the two different control schemes be balanced in a fair way so that the topological differences can be meaningfully compared? Furthermore, how can it be determined that any performance difference is due to the topology and not the control scheme?

In summary, the goal of this section has been to illustrate that there are many open fundamental questions in the area of evaluating and comparing interconnection networks. These

include: (a) Which metrics or weighted collection of metrics should be used to evaluate network performance? (b) What parameters should be held constant when comparing and evaluating networks? (c) In what sense can the implementations of two different networks be made "equal" for a meaningful comparison? It is important to try to find answers to these questions so that the comparative "goodness" of various network features can be quantified.

6: Summary

There are many trade-offs between the SIMD and MIMD modes of parallelism. Because each mode has advantages, it is important to study the design and use of mixed-mode machines, which can switch between modes at instruction-level granularity. In addition to being a useful model of parallel computation, mixed-mode machines provide a platform for developing software tools and programming techniques that are also applicable to mixed-machine heterogeneous systems.

When designing any type of parallel machine, an interconnection network design must be selected. There are significant problems involved with performing "fair" comparisons among interconnection networks, and solutions for these problems are important research topics.

Acknowledgments: The authors thank Janet M. Siegel and Daniel W. Watson for their comments, and thank our co-authors of the papers summarized here ([9, 13, 29, 31]).

References

[1] S. Abraham and K. Padmanabhan, "Performance of the direct binary n-cube network for multiprocessors," *IEEE Trans. Computers*, Vol. 38, No. 7, July 1989, pp. 1000-1011.

[2] S. Abraham and K. Padmanabhan, "The twisted cube topology for multiprocessors: a study in network asymmetry," *J. Parallel and Distributed Computing*, Vol. 13, No. 1, Sept. 1991, pp. 104-110.

[3] G. B. Adams III, D. P. Agrawal, and H. J. Siegel, "A survey and comparison of fault-tolerant multistage interconnection networks," *IEEE Computer*, Vol. 20, No. 6, June 1987, pp. 14-27.

[4] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. Parallel and Distributed Systems*, Vol. 2, No. 4, Oct. 1991, pp. 398-412.

[5] M. Auguin and F. Boeri, "The OPSILA computer," in *Parallel Languages and Architectures*, M. Consard, ed., Elsevier Science Publishers, Holland, 1986, pp. 143-153.

[6] K. E. Batcher, "Sorting networks and their applications," *AFIPS 1968 Spring Joint Computer Conf.*, 1968, pp. 307-314.

[7] J. Beetem, M. Denneau, and D. Weingarten, "The GF11 supercomputer," *12th Ann. Int'l Symp. Computer Architecture*, June 1985, pp. 108-115.

[8] T. B. Berg, S.-D. Kim, and H. J. Siegel, "Limitations imposed on mixed-mode performance of optimized phases due to temporal juxtaposition," *J. Parallel and Distributed Computing*, Vol. 13, No. 2, Oct. 1991, pp. 154-169.

[9] T. B. Berg and H. J. Siegel, "Instruction execution trade-offs for SIMD vs. MIMD vs. mixed-mode parallelism," *5th Int'l Parallel Processing Symp.*, May 1991, pp. 301 - 308.

[10] G. Broomell and J. R. Heath, "Classification categories and historical development of circuit switching topologies," *ACM Computing Surveys*, Vol. 15, No. 2, June 1983, pp. 95-133.

[11] F. Darema, D. A. George, V. A. Norton, and G. F. Pfister, "A single-program-multiple-data computational model for EPEX/ FORTRAN," *Parallel Computing*, Vol. 7, Apr. 1988, pp. 11-24.

[12] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on a SIMD/SPMD architecture: OPSILA," *9th Int'l Conf. on Pattern Recognition*, Nov. 1988, pp. 14-17.

[13] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting," *J. Parallel and Distributed Computing*, Vol. 11, No. 3, Mar. 1991, pp. 239-251.

[14] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, Vol. 54, No. 12, Dec. 1966, pp. 1901-1909.

[15] R. F. Freund, "SuperC or distributed heterogeneous HPC," *Computing Systems in Engineering*, Vol. 2, No. 4, 1991, pp. 349-355.

[16] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[17] N. Giolmas, D. W. Watson, D. M. Chelberg, and H. J. Siegel, "A parallel approach to hybrid range image segmentation," *6th Int'l Parallel Processing Symp.*, Mar. 1992, pp. 334-342.

[18] C. G. Herter, T. M. Warschko, W. F. Tichy, and M. Philippsen, "Triton/1: a massively-parallel mixed-mode computer designed to support high level languages," *Heterogeneous Processing Workshop*, Apr. 1993, pp. 65-70.

[19] L. H. Jamieson, "Characterizing parallel algorithms," in *The Characteristics of Parallel Algorithms*, L. H. Jamieson, D. B. Gannon, and R. J. Douglass, eds., MIT Press, Cambridge, MA, 1987, pp. 65-100.

[20] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[21] P. M. Kogge, "EXECUBE - a new architecture for scalable MPPs," *1994 Int'l Conf. on Parallel Processing*, Vol. I, Aug. 1994, pp. 77-84.

[22] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Computers*, Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.

[23] J. Lenfant, "Parallel permutations of data: A Benes network control algorithm for frequently used permutations," *IEEE Trans. Computers*, Vol. C-27, No. 7, July 1978, pp. 637-647.

[24] G. J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons*, John Wiley & Sons, New York, NY, 1987, pp. 149-174.

[25] M. Philippsen, T. Warschko, W. F. Tichy, and C. Herter, "Project Triton: towards improved programmability of parallel machines," *26th Hawaii Int'l Conf. on System Sciences*, Jan. 1993, pp.192-201.

[26] G. Saghi, H. J. Siegel, and J. L. Gray, "Predicting performance and selecting modes of parallelism: a case study using cyclic reduction on three parallel machines," *J. Parallel and Distributed Computing*, Vol. 19, No. 3, Nov. 1993, pp. 219-233.

[27] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lipovski, "An overview of the Texas Reconfigurable Array Computer," *AFIPS 1980 Nat'l Computer Conf.*, June 1980, pp. 631-641.

[28] H. J. Siegel, *Interconnection Netorks for Large-Scale Parallel Processing: Theory and Case Studies, 2nd Edition*, McGraw-Hill, New York, NY, 1990.

[29] H. J. Siegel, J. K. Antonio, and K. Liszka, "Metrics for metrics: why is it difficult to compare interconnection networks or how would you compare an alligator to an armadillo?," *New Frontiers: A Workshop on Future Directions of Massively Parallel Processing*, Oct. 1992, pp. 97-106.

[30] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping computer-vision-related tasks onto reconfigurable parallel processing systems," *IEEE Computer*, Vol. 25, No. 2, Feb. 1992, pp. 54-63.

[31] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Handbook of Parallel and Distributed Computing*, A. Y. Zomaya, ed., McGraw-Hill, to appear, 1995.

[32] H. J. Siegel, T. Schwederski, W. G. Nation, J. B. Armstrong, L. Wang, J. T. Kuehn, R. Gupta, M. D. Allemang, D. G. Meyer, and D. W. Watson, "The design and prototyping of the PASM reconfigurable parallel processing system," in *Parallel Computing: Paradigms and Applications*, A. Y. Zomaya, ed., Chapman and Hall, London, U.K., to appear 1994.

[33] R. R. Ulrey, A. A. Maciejewski, and H. J. Siegel, "Parallel algorithms for singular value decomposition," *8th Int'l Parallel Processing Symp.*, Apr. 1994, pp. 524-533.

[34] R. Upton and S. Tripathi, "On the performance evaluation of fine-grained SIMD computer architectures: an analysis of the Connection Machine," in *High Performance Computer Systems*, E. Gelenbe, ed., Elsevier Science Publishers, North-Holland, Amsterdam, Holland, 1988, pp. 129-141.

[35] A. Varma and C. S. Ragavendra, eds., *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*, IEEE Computer Society Press, Los Alamitos, CA, 1994.

[36] D. W. Watson, J. K. Antonio, H. J. Siegel, and M. J. Atallah, "Static program decomposition among machines in an SIMD/SPMD heterogeneous environment with non-constant mode switching costs," *Heterogeneous Computing Workshop*, Apr. 1994, pp. 58-65.

[37] C.-L. Wu and T. Y. Feng, eds., *Tutorial: Interconnection Networks for Parallel and Distributed Processing*, IEEE Computer Society Press, Silver Spring, MD, 1984.