Algorithm Scalability: A Poly-Algorithmic Approach *

Leah H. Jamieson† Ashfaq A. Khokhar†‡ Jamshed N. Patel†

†School of Electrical Engineering
Purdue University, West Lafayette, IN 47907
{lhj,jamshed}@ecn.purdue.edu

‡Department of Computer Sciences
Purdue University, West Lafayette, IN 47907
ashfaq@cs.purdue.edu

Abstract

We present evidence that scalability, which encompasses both performance and portability, can be achieved only by taking a poly-algorithmic approach. We summarize theoretical and experimental results for three very different types of parallel algorithms: two-dimensional FFTs, list ranking on a fine-grained machine, and fundamental communications operations on a coarse-grained machine. In each case, we observe that performance degrades if the same basic algorithm is used across the entire range of problem size / machine size combinations. We also observe that the algorithm that gives the best performance may depend on the communications topology and on attributes of the data set. Most importantly, we observe reversals in performance, where one algorithm outperforms another on one platform or on one type of input data, with the converse true on a different platform or on input data with different characteristics. We therefore conclude that informed algorithm selection is essential to achieving scalability.

1 Introduction

In its broadest informal sense, the notion of algorithm scalability should encompass both performance and portability. If an algorithm is scalable, changes in the problem size or changes in the processing platform should result in commensurate changes in the execution characteristics. A broad notion of scalability also suggests that qualitative "goodness" of performance should be preserved. For example, if an algorithm executed on a particular platform performs "well" in one region of the problem-size/system-size space, then it should also perform well in other regions of the space. Similarly, if the algorithm has good performance on one platform, then it should be able to achieve good performance on other platforms as well. Therefore scalability should also imply *preserving* a level of performance.

Traditional complexity analysis provides meaningful information about how algorithms executing on a single processor will behave as the problem size grows or the processor speed increases. Recent analyses have successfully included memory hierarchy effects in evaluations of performance. Moreover, decades of experience with sequential algorithms has given us reasonably sound intuition about good and bad growth patterns for sequential algorithms.

However, we have less experience with parallel algorithms, and the number of factors that affect the execution time of parallel algorithms is much larger than simply problem size and processor speed. Parameters that influence scalability may include (but are not limited to) problem size, processor speed, number of processors, communication topology, communication latency, processor and network bandwidth, data distribution, and memory organization. When the target platform may range from a single processor to a network of workstations to a massively parallel machine, the asymptotic and measured execution time may depend on attributes of the platform. Data dependent factors such as locality may have a significant effect on the performance. It has therefore become important to try to develop a notion of scalability for parallel algorithms that captures both the effect on execution time when a sequential algorithm is made parallel, and the effect on execution time when the attributes of the parallel platform change.

Explicit scalability metrics and models of parallel computation address some of the aspects of algorithm performance as a function of changing parameters. Most notions of scalability have dealt, in some form, with what happens to the execution time of a given algorithm when either the problem size or the machine size changes. For example, the isoefficiency function [5, 10] determines the rate at which the problem size has to grow as a function of the number of processors to maintain constant efficiency. A linear isoefficiency function implies that the algorithm is highly scalable. Similarly, scaled-speedup [6] uses constant speedup with a linear change in both machine and problem sizes as a measure of scalability. Most existing computational models, both fine-grained and coarse-grained, such as PRAM and BSP [19], predict algorithm performance as a function of problem size,

*This work was supported by the Advanced Research Projects Agency under contract DABT63-92-C-0022. The content of the information does not necessarily reflect the position or policy of the United States Government and no official endorsement should be inferred. We thank Susanne Hambrusch for many useful discussions.

but do not capture the effects of salient architectural features of parallel machines in analyzing the complexity of an algorithm. Scalability predictions based on such complexity measures therefore may not reflect the true performance of the algorithm. Recently, coarse-grained computational models such as LogP [4] and C^3 [7] have considered several architecture features in order to obtain better predictions of performance. Such models, combined with scalability metrics, therefore capture the effects of many of the parameters that influence scalability.

However, these models and measures focus on individual algorithms, and do not take into account the fact that there may be many different algorithms for solving a given computational problem. Although the same can be said for asymptotic complexity analysis of sequential algorithms, there is a fundamental difference that makes the consideration of alternative algorithms more compelling when dealing with parallel systems. In the case of sequential algorithms we concentrate on the asymptotic complexity because, in comparing algorithm A with algorithm B, it is sufficient to know which is expected to give lower execution time on large problems. The execution time on small problems will, by comparison, be small, so the relative performance of the two algorithms on small problems is less critical. In the case of parallel algorithms, the number of different parameters that affect execution time may lead to the situation where, *for a very large problem*, one algorithm may outperform another on one platform, but the converse may be true on a different platform. Similar reversals in performance as a function of algorithm may occur for data sets with different properties, in cases where there is no such dependence on data properties for sequential algorithms. Since the problem size is large, both the execution time and the difference in execution time may also be large. To achieve meaningful scalability that spans platforms and data properties therefore requires that the actual choice of algorithm be a factor in measuring performance.

In this paper we present evidence that scalability, in the sense of preserving performance, can be achieved only by taking a poly-algorithmic approach. In the following sections, we summarize theoretical and experimental results for three very different types of parallel algorithms: two-dimensional FFTs [9, 12], list ranking [14], and fundamental communications operations [8]. In each case, we observe that performance degrades if the same basic algorithm is used across the entire range of problem size / machine size combinations. We also observe that the choice of algorithm may depend on the communications topology and on attributes of the data set. We therefore conclude that scalability, in the sense of preserving performance, must be achieved by families of algorithms, rather than by individual algorithms. The algorithms in a family perform the same function but have different expected execution characteristics that depend on the platform type, number of processors, and characteristics of the communication operations, network, and data.

2 Parallel 2-D FFT Algorithms

The two-dimensional FFT is fundamental to a wide range of applications, including image processing, computational fluid dynamics, and power spectrum estimation. The algorithm has a regular structure that is independent of the input data. Two basic approaches to computing the 2-D FFT are *decimation* algorithms, in which the array is recursively divided to compute 2-D FFTs of smaller size, and *row-column* algorithms, in which the $n \times n$ 2-D FFT is computed in terms of $2n$ 1-D FFTs [15]. Parallel implementations of the latter may be *coarse-grained* (number of array rows or columns exceeds the number of processors) or *fine-grained* (number of processors exceeds the row or column size) [9, 12].

For a given problem size and machine size, the arithmetic complexity is the same for the decimation method and the row-column methods. Therefore the fastest algorithm is the one that has the lowest communication overhead. Both the number and type of communication operations differ for the various algorithms. Assuming synchronous (SIMD) execution, analysis of the interprocessor transfers for the different algorithms yields a division of the machine-size/problem-size space as shown in Fig. 1. In part (a) of the figure, it is assumed that all of the required permutations can be performed in one time unit, as would be the case in a system with a multistage interconnection network. In part (b), the time for each communication operation is assumed to be proportional to the distance that the data must travel, as would be the case in a mesh architecture. Not only do the analyses indicate that the choice of algorithm should depend on the the relationship between the number of processors and the problem size, but also that the dependence is very different for the two communications models [12].

3 Parallel List Ranking Algorithms

List ranking is a fundamental operation in many algorithms for graph theory and computer vision problems. Given a linked list of n cells, list ranking determines the distance of each cell from the head of the list. On a sequential machine, this problem can be solved in $O(n)$ time by simply traversing the list once. However, it is much more difficult to perform list ranking on parallel machines due to its irregular and data dependent communication patterns. Extensive theoretical and experimental work on list ranking on random lists has been performed [1, 2, 3, 17, 18].

We summarize experimental results for two representative approaches to list ranking on a fine-grained SIMD machine [14]. *Wyllie's Algorithm* [16, 18] is a simple algorithm that uses pointer jumping or dereferencing to find the rank of a cell. It has a running time of $O(\frac{n \log n}{p})$ on a p-processor machine. Anderson & Miller's *Randomized Algorithm* [1] reduces the expected running time for list ranking to $O(n/p)$, assuming $n/p \geq \log n$. The savings is accomplished by using coin-tossing to randomly splice cells out of the

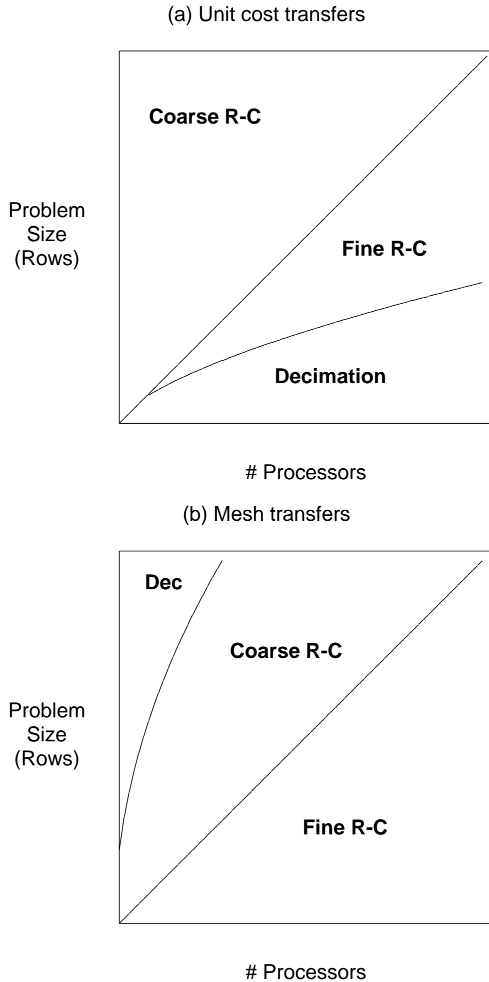


Figure 1: Division of the machine-size/problem-size space into regions of optimal performance for three different 2-D FFT algorithms: (a) assuming unit cost transfers, and (b) assuming mesh transfers.

list, then reconstructing the rank for the spliced-out cells, thereby eliminating excess work compared to the simple pointer-jumping approach.

3.1 List Ranking on Random Lists

Fig. 2 shows the performance of Wyllie's Algorithm and the Randomized Algorithm on the MasPar MP-1 using various machine and problem sizes. The algorithms were executed on random lists where cells were randomly pre-assigned to processors. Wyllie's Algorithm is more suited for smaller linked lists due to its small constant factors. The Randomized Algorithm outperforms Wyllie's Algorithm as the problem size increases. For a fixed-size list, the Randomized Algorithm shows better performance until the number of cells/processor becomes too small (related to the analytical condition that $n/p < \log n$), after which the overhead of the coin-tossing hurts its performance. The results agree with results reported on other machines [2, 17] and are consistent with the theoretical analysis. The experimental results can be used to determine the regions of problem and machine size for which each algorithm is fastest, as shown in Fig. 3.

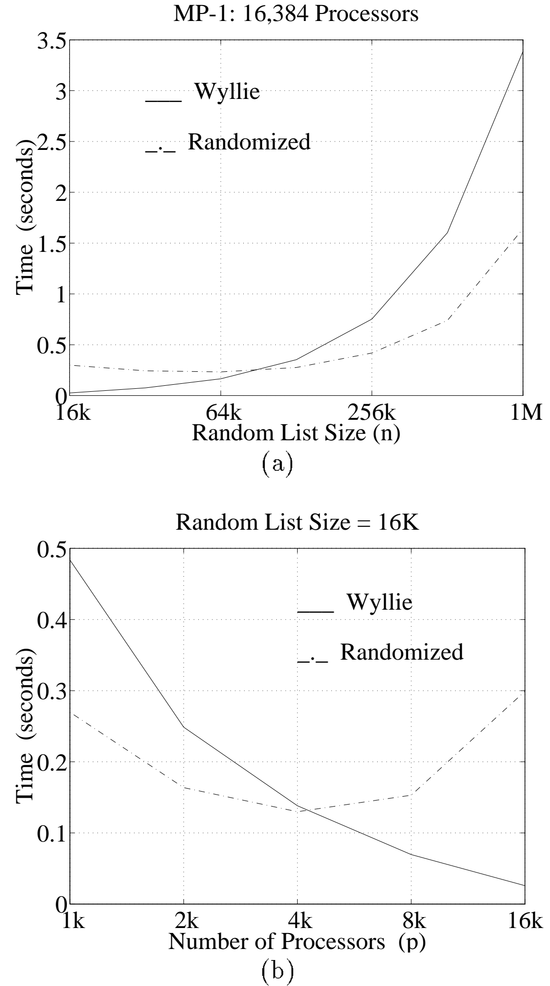


Figure 2: (a) Performance of Wyllie's and Randomized algorithms on the MP-1 for random lists of increasing size, and (b) performance of the algorithms on random lists of size 16K on MP-1 partitions of increasing size.

3.2 List Ranking on Image Edge Lists

In computer vision, list ranking is an intermediate step in various edge-based matching and recognition tasks. It is used to extract edges that are embedded in a 2-D image plane and then represent them in a compact data structure for efficient processing in subsequent steps [2]. Whereas a cell and its successor are unlikely to be in the same processor in the case of randomly generated lists, lists representing image edges are more likely to be local to a processor and its immediate neighbors. Although there is no advantage to

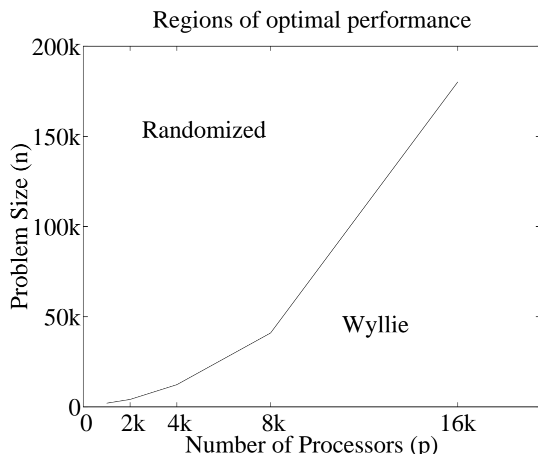
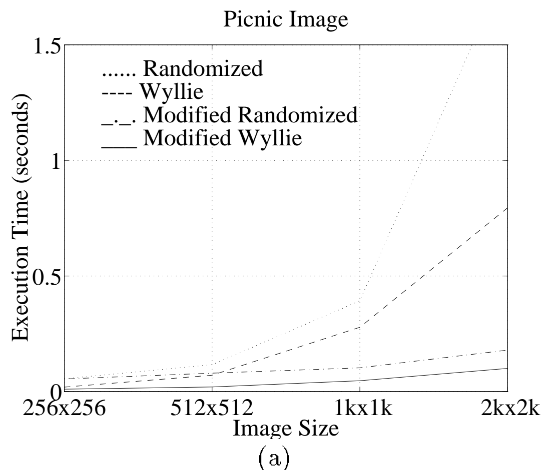


Figure 3: Experimentally determined regions in the problem-size/machine-size space where each list ranking algorithm is fastest, based on execution time on random lists.

designing sequential list ranking algorithms for image edge lists, significant performance gains can be realized for parallel implementations by modifying the list ranking algorithms described in the previous section to take advantage of the expected locality properties of image edge lists. This yields a *Modified Wyllie's Algorithm* and a *Modified Randomized Algorithm* [14]. However, the characteristics of edge lists derived from images can vary significantly depending on the image. In order to further demonstrate the need for a poly-algorithmic approach, we compare the original and modified algorithms on edge lists with different edge characteristics.

Fig. 4 shows the execution times of the original and modified algorithms on two different image edge maps. Fig. 4 (a) shows the execution times for varying sizes of an edge map derived from a picnic scene that has a large number of relatively short edges. Fig. 4 (b) shows the execution time for the edge map of a synthetically generated spiral image that has one extremely long edge. From these experiments, we see a complex interaction of algorithm, machine size, list size, and list characteristics. As expected, the modified algorithms significantly outperform the original algorithms, showing the value of choosing the algorithm to reflect the expected list characteristics (random or image). Moreover, which modified algorithm performs better depends on the details of the image list characteristics (short lists vs. long list) and, in the case of the spiral image, on the image size. (The scale of the plots, which was chosen to show the range of performance of all of the algorithms, somewhat hides the fact that the performance difference between the two modified algorithms is significant: the Modified Wyllie's algorithm is twice as fast for the the common case of images with short edges (Fig. 4 (a)); the Modified Randomized algorithm is as much as a factor of six faster for dense images with very long edge-lengths (Fig. 4 (b)).



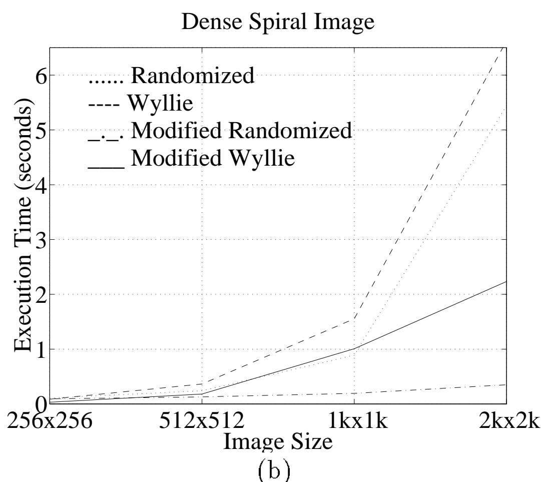


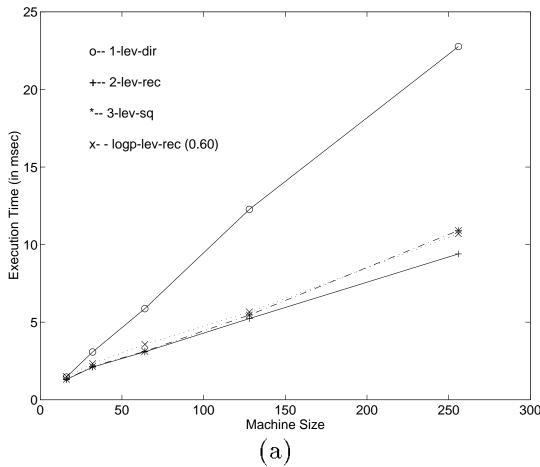
Figure 4: Performance of the list ranking algorithms for various size images on a 16,384 processor MP-1: (a) the picnic image; (b) the spiral image.

4 Communication Operations on Coarse-Grained Machines

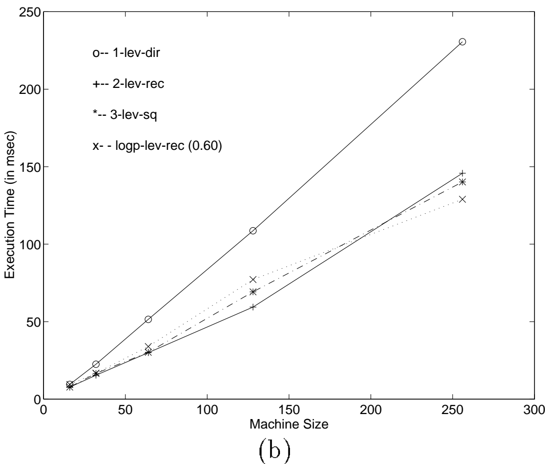
Communication operations are of significant importance for achieving high performance on coarse-grained machines. Scalable communication routines are the basis for making programs scalable. Communication patterns arising in many applications can be characterized in terms of scatter, gather, or personalized communication. We have considered the scalability of different personalized communication patterns including one-to-all, all-to-one, and all-to-all. In personalized communication, each destination processor receives a unique message. Such communication operations frequently arise in application algorithms. We have studied the impact of various machine and algorithm parameters on the performance of these communication operations. Once again our conclusion is that no single algorithm is scalable across the entire range of machine or message sizes. In the following,

we briefly present the performance results of various algorithms for all-to-one and all-to-all operations and discuss their scalability.

In all-to-one communication, every processor sends a unique message to a single destination processor. On the other hand, in all-to-all communication, every processor sends a unique message to every other processor. We have designed several algorithms for these communication operations. These algorithms differ in terms of total number of messages issued by the source processor, total number of messages received by a destination processor, maximum length of messages being routed, and number of intermediate processors involved in routing a message. These algorithms have been implemented on the Intel Delta. For details on these algorithms we refer to [8].



(a)



(b)

Figure 5: Scalability results for all-to-one algorithms on the Intel Delta when the destination processor receives a total of (a) 256 bytes and (b) 4 Kbytes, respectively, varying machine size.

Figure 5 shows the scalability behavior of different algorithms for all-to-one communication operation on the Intel Delta. For large message sizes, the algorithm which performs best on small machines does not perform well on larger machines. For example, for 4 Kbyte messages Algorithm 2-lev-rec performs best on a 128 processor machine, while on a 256 processor machine Algorithm logp-lev-rec(0.60) performs best.

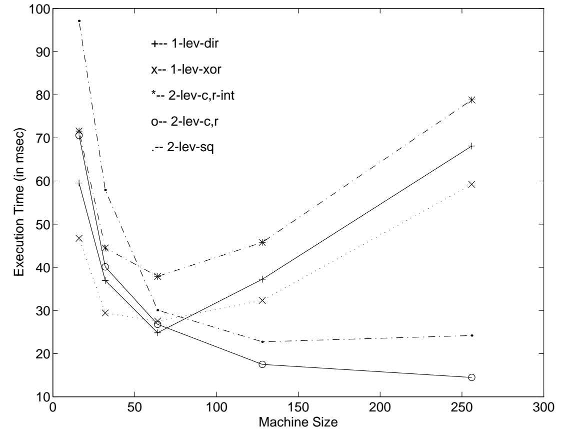


Figure 6: Scalability results for all-to-all algorithms on the Intel Delta when the total of all actual messages is 2 Mbytes, varying machine size.

Figure 6 shows the scalability results of all-to-all communication on various sizes of the Intel Delta when the total of all the messages sent among processors is 2 Mbytes. It is clearly shown that no single algorithm performs well for the entire range of machine sizes. Because the communication primitives may be used repeatedly in programs, it is critical that a fast algorithm be used for every machine size.

5 Conclusions

We have presented results from three significantly different problems showing that scalability, including portability, of parallel algorithms cannot be achieved by choosing a single parallel algorithm to perform an operation. The experimental results for communications primitives on the coarse-grained Intel Delta showed each algorithm's performance varying significantly as a function of the machine size. The analytical results for the structured, data-independent 2-D FFT showed the effect on execution time of the relationship between the problem size and the machine size. The analyses for the FFT algorithms also showed the dependence on the communication model. The experimental results for the data-dependent operation of list ranking, performed on the fine-grained MasPar MP-1, showed that algorithm performance depends on the relationship between the problem size and the machine size, and also on the properties of the input data. Moreover, as is seen most clearly in the all-to-all communications example and the spiral image list ranking example, the performance differences as a function of algorithm can be very large. Architecture parameters such as machine size and network characteristics, as

well as problem parameters including input size and data set characteristics, must therefore be considered when choosing an algorithm.

These experiments indicate that a poly-algorithmic approach is needed to achieve scalability. This has implications for both algorithm and software design. For algorithms that decompose a large problem into smaller problems, hybrid algorithms can use one algorithm until the problem is sufficiently small, then switch to a different algorithm. This is consistent with the approach taken for many serial algorithms (e.g., hybrid FFT-DFT algorithms for inputs whose size is not a power of two) and agrees with the approaches that have been used to develop processor-time optimal solutions for the list ranking problem [16]. For users of parallel systems, software support that assists in selecting a suitable algorithm will be essential to achieving the performance potential of such systems. We have successfully applied this poly-algorithmic approach in developing scalable libraries and library access software tools for the applications of computer vision and image processing [11, 13].

References

[1] R. J. Anderson and G. L. Miller, "A simple randomized parallel algorithm for list-ranking," *Information Processing Letters*, 33(5):269–73, 1990.

[2] L. T. Chen, L. S. Davis, and C. P. Kruskal, "Efficient parallel processing of image contours," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(1):69–81, 1993.

[3] R. Cole and U. Vishkin, "Faster optimal parallel prefix sums and list ranking," *Information and Computation*, 81(3):334-52, 1989.

[4] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* May 1993.

[5] A. Gupta and V. Kumar, "The Scalability of FFT on Parallel Computers," *IEEE Transactions on Parallel and Distributed Systems,* 4(8):922–932, 1993.

[6] J. L. Gustafson. "Reevaluating Amdahl's Law," *Communications of the ACM*, 32(5):532–533, 1988.

[7] S. E. Hambrusch and A. A. Khokhar, "An architecture-independent model for coarse-grained parallel machines," *6th IEEE Symposium on Parallel and Distributed Processing,* October 1994.

[8] S. E. Hambrusch, F. Hameed, and A. A. Khokhar, "Communication operations on coarse-grained mesh architectures," Technical Report CSD-TR-94-037, Department of Computer Sciences, Purdue University, May 1994. (To appear in *Parallel Computing*).

[9] L. H. Jamieson, P. T. Mueller, Jr., and H. J. Siegel, "FFT Algorithms for SIMD Parallel Processing Systems," *Journal of Parallel and Distributed Computing,* 3(1):48–71, 1986.

[10] V. Kumar and A. Gupta. "Analyzing scalability of parallel algorithms and architectures," *AHPCRC Preprint 92-020*, Army High Performance Computing Research Center, University of Minnesota, January 1992.

[11] L. H. Jamieson, E. J. Delp, J. N. Patel, C. C. Wang, and A. A. Khokhar. "A library-based program development environment for parallel image processing," *Scalable Parallel Libraries Conf.*, pp. 187–194, October 1993.

[12] J. N. Patel and L. H. Jamieson, "Evaluating scalability of the 2-D FFT on parallel computers," *Computer Architectures for Machine Perception '93*, pp. 109–116, December 1993.

[13] J. N. Patel, A. A. Khokhar, and L. H. Jamieson. "Implementation of parallel image processing algorithms in the Cloner environment," *IEEE Workshop on VLSI Signal Processing*, pp. 83–92, October 1994.

[14] J. N. Patel, A. A. Khokhar, and L. H. Jamieson, "Scalable parallel list ranking of image edges on fine-grained machines," *9th International Parallel Processing Symposium*, April 1995.

[15] L. R. Rabiner and G. Gold, *Theory and Application of Digital Signal Processing,* Prentice-Hall, Englewood Cliffs, N.J., 1975.

[16] M. Reid-Miller, G. L. Miller and F. Modugno, "List Ranking and Parallel Tree Contraction," in *Synthesis of Parallel Algorithms*, J. H. Reif, editor, Morgan Kaufmann Publishers, San Mateo, California, pp. 115-194, 1993.

[17] M. Reid-Miller, "List ranking and list scan on the Cray C-90," *Proceedings Symposium on Parallel Algorithms and Architectures*, pp. 104-113, 1994.

[18] J. C. Wyllie, "The complexity of parallel computations," Technical Report TR-79-387, Department of Computer Science, Cornell University, Ithaca, NY, August 1979.

[19] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the ACM,* August 1990, Vol. 33, No. 8.