Limits on Heterogeneous Supercomputing

John E. Savage

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-93-19
June 1993

Limits on Heterogeneous Supercomputing[†]

John E. Savage
Brown University
Department of Computer Science, Box 1910
Providence, Rhode Island 02912
Phone: (401) 863-7642, Fax: (401) 863-7657
Email: jes@cs.brown.edu
June 29, 1993

Abstract

Heterogeneous supercomputers, which couple traditional serial and parallel supercomputers via high-bandwidth channels, offer the potential for much higher performance than can be achieved with either type of machine alone. In this paper we define and analyze the capabilities of the Mesh SuperHet, a closely coupled heterogeneous supercomputer consisting of a serial front end and a d-dimensional toroidal mesh of coarse-grained parallel processors connected by a high-bandwidth channel. The channel consists of connections between individual front-end memory modules and processors on one face of the mesh. We exhibit problems for which this architecture is superior to a serial or parallel component alone and we develop tight performance bounds for sorting, the fast Fourier transform, and matrix multiplication on this machine. As heterogeneous supercomputers become more common, studies such as this will both reveal the fundamental limitations on such architectures and set the context for algorithm development.

Technical Report CS 93-19

[†]This work was supported in part by the Office of Naval Research under contract N00014-91-J-4052, ARPA Order 8225 and NSF Grant MIP-902570.

1 Introduction

Heterogeneous supercomputers consist of serial and parallel supercomputers coupled by high-bandwidth channels. Such machines can greatly reduce computation time for large problems. It is reported that McRae solved a chemical process resource-allocation problem "40 times faster than he could on a supercomputer alone" [1] using a CRAY Y-MP connected via a high-speed link to a CM-2 Connection Machine. Others report a factor of 5 to 10 reduction in elapsed time [12] for such architectures. An explanation for this remarkable performance is that many problems exhibit both serial and parallel aspects that can be exploited by the heterogeneous supercomputer.

In this paper we explore the potential of closely coupled heterogeneous supercomputers. We introduce the *Mesh SuperHet*, a realistic model consisting of a d-dimensional mesh of p coarse-grained processors connected to a serial computer. We exhibit problems that are more quickly solved with it than by either the serial front end or the parallel machine alone and we develop fast algorithms and lower bounds for representative problems on this machine, namely, sorting, the fast Fourier transform (FFT), and matrix multiplication. We show that a full speedup of order p is not attainable for sorting and the FFT, but is attainable for matrix multiplication, even with a very limited bandwidth between the mesh and the serial front end, if there is enough local memory per mesh processor.

As these examples illustrate, the performance of a closely coupled heterogeneous supercomputer depends on the architecture of the machine and the problems on which it is used. Key factors are the type and bandwidth of the connection between the serial and parallel machine, the speed of mesh processors, their interprocessor bandwidth, and the amount of memory per mesh processor, as well as the demands of particular computational problems. We make realistic assumptions about the architectural elements and *analyze problems that are so large they do not fit in the memory of the multidimensional mesh.*

The Mesh SuperHet consists of a high-performance serial machine closely coupled to a d-dimensional toroidal mesh with a substantial amount of memory per mesh processor. We study close coupling to understand how it affects performance. Close coupling is obtained by connecting individual processors on the face of the mesh to individual memory modules on the front end. We assume front-end memory modules are addressable via some type of fast routing network, as is typical for a serial supercomputer. A mesh is assumed for the parallel machine because the proximity of processors supports the high-speed local communication needed for

many large-scale computational problems. (Bilardi and Preparata [4] argue for the mesh as the parallel architecture of choice in the limit of very fast computations.) Thus, we have chosen to study a closely coupled heterogeneous supercomputer in which the front end connects memory modules via a low-diameter network and the back end connects memory modules via a high-diameter network.

For simplicity and without significant loss of generality, we assume: a) all processors have the same high speed, b) no latency on data movement on any links, c) uniform bandwidth on all links, and d) flat front-end memory modules of unlimited size. Assumption a) is made because we have not found the speed of the front end to be significant in our studies. Assumption b) holds if data is moved in large blocks, which is valid for the problems studied. The third assumption typically holds in practice to within a factor of 5-10, which does not affect our asymptotic analyses. The last assumption is not practical but can be simulated for many problems [6, 15]), including those studied here.

Large simulated memories are expensive and are an important distinguishing characteristic of serial supercomputers. Parallel machines typically do not support such memories on each processor because of their cost. Consequently, it is natural in heterogeneous machines to assume, as is done here, that large simulated memories are physically distinct from the memories of the parallel machine. On the other hand, if such memories are permitted on each processor of the mesh, our bounds on performance would be very different.

1.1 Related Computational Models

The Mesh SuperHet model is a generalization of two models. Atallah and Tsay [2] consider a heterogeneous computer which has a serial front end connected to a d-dimensional mesh in which the memory per mesh processor is small. They also assume that in one cycle as many words can be moved between the two machines as there are processors on one face of the mesh, namely $p^{1-1/d}$, but make no assumptions about the form of the communication between the two machines. They show that a speedup of $p^{1-1/d} \log p$ is possible for sorting-related problems in computational geometry. Dehne, Fabri and Rau-Chaplin [5] consider a machine consisting of a p-processor d-dimensional mesh in which the processors have enough memory to hold the data for an entire problem, equally distributed over the p memory units. When a sorting problem fits entirely on a two-dimensional mesh, they show that a

full speedup is possible if $m = 2^{\Omega(\sqrt{p})}$.[1]

Like Dehne et al., we assume that each processor on the mesh can have a large amount of memory. Like Atallah and Tsay, we assume the problem is so large that it cannot fit entirely on the processor array; a portion of it must reside on the front end. We severely restrict the Atallah-Tsay model by forcing data to move between processors on a mesh face and individual front-end memory units, one per mesh processor.

1.2 Detailed Description of Results

In Section 2 we give a detailed description of the Mesh SuperHet that spells out our assumptions on processor speeds, network bandwidth, and the interconnection between the serial and parallel machines. We argue that our results are dependent primarily on the number of mesh processors connected to front-end memories, and not their location, exact number or bandwidth.

In Section 3.1 we show that we can dispense with the serial front-end processor when it executes a number of steps that is no more than approximately the number of steps T divided by the diameter $p^{1/d}$ of the mesh face. We do this by simulating the serial processor on the mesh. We also exhibit in Section 3.2 problems that the 2-D Mesh SuperHet computes more quickly than does either a serial machine or the Mesh SuperHet with the front-end processor disabled.

In Section 4 we examine sorting on the Mesh SuperHet and derive a lower bound on the time to sort n items (Section 4.1) as well as a matching upper bound (up to a multiplicative constant) obtained by extending the Atallah-Tsay multidimensional-sorting algorithm to our model (Section 4.3). This extension gives a speedup of about $p^{1-1/d}\log(mp)$ when n is large. When m is small, we achieve the same results as Atallah and Tsay but with a much more restrictive model of data movement. When m is large, the effect of memory size is to increase the factor $\log(p)$ to $\log(mp)$ in the speedup of the original Atallah-Tsay algorithm. While this is not a strong dependence on local memory size, it can give important speedups at small cost. Since our extension of the Atallah-Tsay sorting algorithm is complex, we also derive a simpler and more realistic sorting algorithm (Section 4.2).

In Section 5 we examine the FFT algorithm and derive upper and lower bounds (Section 5.1) on its computation time that differ asymptotically only by constant

[1] For functions $f(n)$ and $g(n)$ on the positive integers, the notation $f = O(g)$ means that there exists an integer N and a real value $c > 0$ such that $f(n) \leq cg(n)$ for $n \geq N$. If $f = O(g)$, we write $g = \Omega(f)$. If both $f = O(g)$ and $g = \Omega(f)$, we write $f = \Theta(g)$.

factors. The FFT and sorting problems show the same speedups. As the mesh dimensionality d increases, the bandwidth of the interface between the serial and parallel machines grows, and both sorting and the FFT show speedups approaching the limit p.

In Section 6 we demonstrate that the matrix multiplication problem exhibits a very different dependence on the parameters of the Mesh SuperHet. A full speedup of order p is possible as long as $\sqrt{m}\beta \geq cp^{\lceil d/2 \rceil / d}$ for some constant c. This condition holds even when β, the number of mesh processors connected to front-end memories, is constant, as long as m, the memory per processor, grows as $p^{\lceil d/2 \rceil 2/d}$.

As these results demonstrate, the amount of speedup possible with the Mesh SuperHet depends both on the problems for which it is used and on the parameters of the machine, in particular, the bandwidth between serial and parallel processors and the amount of memory on each mesh processor. Kung [10] has explored conditions under which balance is obtained between computation and I/O time for matrix multiplication, Gaussian elimination, the fast Fourier transform, and sorting when executed on serial machines; he also comments on these issues for two-dimensional meshes.

Conclusions are drawn in Section 7.

2 The Mesh SuperHet Model

The Mesh SuperHet model described below captures essential features of an important class of heterogeneous supercomputer. It connects a front end containing a simulated large flat random-access memory to a d-dimensional array of processors. It achieves a high bandwidth between the mesh and the front end by connecting each processor on a face of the mesh to a front-end memory module. This assumption requires only small changes in the architecture of traditional serial supercomputers – the front-end addressing logic needs to arbitrate the traffic to and from the front-end memory units. The interface between these machines can be simulated by other architectures without seriously affecting the results derived for the Mesh SuperHet.

The connection between the front end and the mesh in the Mesh SuperHet imposes important restrictions on data movement between the two machines. When large volumes of data are moved from the mesh to the front end, care must be taken to assign data to memory modules in such a way that it can be retrieved later without creating serial bottlenecks. We show that many problems can be executed efficiently on this model despite the specialized connection between the two machines.

Definition 1 *The Mesh SuperHet has the following characteristics:*

- *A parallel computer consisting of a toroidal d-dimensional mesh of $p = s^d$ processors, s processors per dimension, each with m words in its local memory.*
- *A mesh-interprocessor bandwidth comparable to the mesh memory bandwidth.*
- *A serial front-end computer with s^{d-1} memory units and an I/O subsystem simulating a large flat serial memory.*
- *A front end that executes instructions at the same speed as mesh processors.*
- *Individual channels connecting some or all of the s^{d-1} processors on a face of the mesh to front-end memory units with a bandwidth comparable to the mesh interprocessor bandwidth.*
- *A common word size on the mesh and the front end.*

We make the following assumptions about the initial and final locations of the data on which the Mesh SuperHet operates:

- All data for a task is located initially in the front-end memory units.
- All results of a computation are located in the front-end memory at the completion of the task.

Figure 1 shows a 2-D version of the Mesh SuperHet in which the front-end memory is subdivided into modules connected to rows of the mesh. As is true of traditional serial supercomputers, a large investment in the addressing logic of the front-end memory is necessary to simulate a uniform high-speed memory. Not shown is the I/O subsystem; it too must be sufficiently large and fast to keep the front-end memory supplied with data in a timely fashion. The mesh network offers lower latency and higher speed local communication than can be obtained with other architectures, such as the hypercube.

Each processor in a d-dimensional toroidal mesh is indexed by a d-tuple $\underline{a} = (a_1, a_2, \ldots, a_d)$, $0 \leq a_i \leq s-1$, such that processors $\underline{a}$ and $\underline{b}$ are adjacent if and only if for some index i_0, $a_i = b_i$ for $i \neq i_0$ and $a_{i_0} = b_{i_0} \pm 1 \pmod{s}$. The mesh has p processors and $s = p^{1/d}$. Communication is between adjacent processors. We make the reasonable assumption that mesh-processor/memory-module bandwidth is comparable to the mesh. We also assume that data moves in sufficiently large blocks

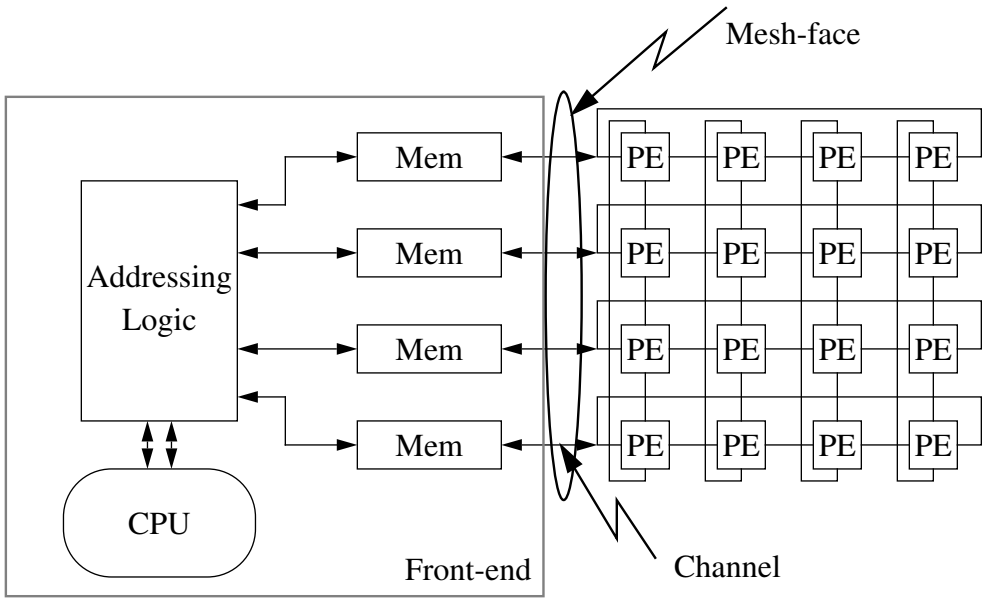

Figure 1: The two-dimensional Mesh SuperHet computer

or sufficiently infrequently that the effect of latency is minimized. In particular, we assume that the average times to move a word and execute a mesh processor instruction are comparable. The processing elements are assumed to have modest-sized random-access memories with m words per processor. The value of m can determine how efficiently the Mesh SuperHet uses the available computing power.

The assumption that interface processors connect to individual memory modules is realistic. Memories can be dual-ported at a small incremental cost, although important changes may be necessary in the front-end memory arbitration logic. We have chosen to limit the number of mesh processors connected to the front end to the size of face of the mesh. If the mesh were constructed as a dense cube to minimize wire lengths, it would only be practical to access memories on the surface of the cube. We have connected mesh processors to memory modules as a way to reduce the otherwise large burden that arises when data has to be communicated between $p^{1-1/d}$ interface processors and the front end when p is large.

2.1 Inter-Machine Communication

Let's now examine the following variants on the communication assumptions:

i) $p^{1-1/d}$ memory modules and processors are connected via channels but the processors are not all on one face of the mesh,

ii) the number of front-end memory modules and mesh-face processors connected via channels is substantially different from $p^{1-1/d}$,

iii) the bandwidth of the interface channels is substantially different from the mesh interprocessor bandwidth.

In the first case, data on the mesh can be routed to the appropriate interface processors. As shown later, this can be done in $O(mp^{1/d})$ steps which is comparable to the time to move data onto and off of the mesh. Thus, it is not crucial that the front end and mesh communicate via a mesh face.

The second assumption can have a substantial impact on the performance of the Mesh SuperHet. When all p mesh processors are connected to separate front-end memory units, every mesh processor has an unlimited amount of local memory. In this case it is possible for some problems, such as sorting for which a full speedup is not possible on the Mesh SuperHet, to show a full speedup of order p (see [2] and Section 4.3). In other cases, such as matrix multiplication, the number of channels connecting processors and memory units can be considerably less than $p^{1-1/d}$ and a full speedup is still possible (see Section 6).

In the third case, an increase in inter-machine channel bandwidth can compensate for having fewer interface processors. However, limitations on the speed of mesh processors put a ceiling on channel bandwidth. Decreasing the channel bandwidth has an effect similar to reducing the number of interface channels.

3 Testing the Limits of the Mesh SuperHet

In this section we consider two questions: "Is the front-end processor always needed?" and "Are there problems that are solved more quickly on a Mesh SuperHet than on either the serial front end or the parallel computer alone?" We examine these questions in the next two sections.

3.1 The Importance of the Serial Front End

The algorithms developed in Sections 4, 5 and 6 make limited use of the serial front end. This causes us to ask if the front end is always necessary. From a practical

point of view it is extremely convenient; many computational problems generate data in parallel but analyze it with code containing large numbers of branching statements that is hard to parallelize. Thus, it is highly desirable to have available a high-performance machine to run such analysis code on the data generated by a parallel machine.

While this discussion argues for the importance of the serial front end, it does not establish that it is always needed, especially when the mesh processors are equally fast. As the following theorem demonstrates, the work of the serial front end can done by the parallel processors as long as the front end computes for a relatively small fraction of the total time.

Theorem 1 *Let T_{serial} be the number of steps executed by the serial front end of a d-dimensional, p-processor Mesh SuperHet during a T-step computation. If $T_{serial} = O(T/p^{1/d})$, a complete computation can be done with a disabled front-end processor in $O(T)$ steps.*

Proof We describe a simulation of a Mesh SuperHet computation by such a model from which the front-end processor and addressing logic have been deleted. All mesh processors execute their original computations until the front end performs a serial step whereupon they pause while some of the mesh processors simulate a front-end step. They then resume their original computations. Counters can be added to programs to keep track of the progress of the serial and parallel computations. We show that $O(p^{1/d})$ steps on the mesh suffice to simulate one front-end step. This implies that the simulating program executes $O(p^{1/d} \times T_{serial} + T)$ steps, which is $O(T)$ under the hypothesis of the theorem.

Front-end operations on registers are simulated by a designated mesh-face processor. This processor also dispatches messages to other processors on the mesh face when a memory access is necessary. The time for each of these dispatches is $O(p^{1/d})$, the diameter of the mesh face, from which the result follows. □

P-complete problems [11] are believed hard to parallelize. Such problems run in serial polynomial time but are not thought to be executable on any parallel machine in time polynomial in the logarithm of the problem input size. Such problems are equally hard to solve on the Mesh SuperHet.

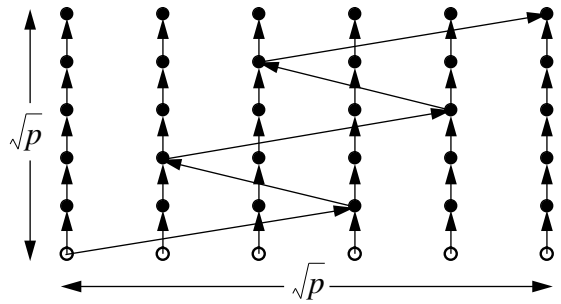

Figure 2: Computational graph that is computed on the Mesh SuperHet more efficiently than on the serial front end or parallel machine alone.

3.2 A Problem Better Solved on the Mesh SuperHet

we asked above whether there are problems that can be computed by a heterogeneous supercomputer more quickly than by either a serial or parallel supercomputer alone. In this section we show the existence of such problems for the 2-D p-processor Mesh SuperHet.

The graph shown in Figure 2 suggests a problem computed quickly on the 2-D Mesh SuperHet which cannot be computed quickly by either the serial or parallel component of the machine. Let $\sqrt{p}$ be even. Rows and columns are numbered $0, 1, 2, \ldots, \sqrt{p}-1$ from bottom to top and left to right. Let the vertex in row i and column j be denoted (i, j). Then edges are directed from from (i, j) to $(i, j+1)$ for $0 \leq i \leq \sqrt{p}-1$ and $1 \leq j \leq \sqrt{p}-1$, from $(j-1, j-1)$ to $(j, \sqrt{p}/2+j-1)$ for j odd, and from $(j-1, \sqrt{p}/2+j-2)$ to (j, j) for j even.

Let $\tau_{i,j}$ denote the value computed at vertex (i, j) of this graph and let the variables $\{\nu_{i,j}\}$ be over the set A. The open vertex $(0, j)$ computes $\tau_{0,j} = \nu_{0,j}$. The closed *unary* vertex (i, j) computes the function $\oplus$ of the value $\tau_{i-1,j}$ at the preceding vertex and the variable $\nu_{i,j}$, returning the value $\tau_{i,j} = \tau_{i-1,j} \oplus \nu_{i,j}$. The closed *binary* vertex (i, j) computes the function g of the value $\tau_{i-1,j}$ of the preceding vertex, the value of the remote vertex $\rho_{i,j}$, and the value of the variable $\nu_{i,j}$, returning the value $\tau_{i,j} = g(\tau_{i-1,j}, \rho_{i,j}, \nu_{i,j})$. Here $\rho_{i,j}$ is determined by the definition of the graph.

This function, which we denote $f^p_{MSH} : A^p \mapsto A^{\sqrt{p}}$, can be computed efficiently row by row on the 2-D Mesh SuperHet as follows: a) at each row the front-end processor fetches the datum needed from the appropriate memory module in one step and deposits it into a special location in the module in which it is needed; b) the mesh-face processor needing this value (see Figure 1) fetches it and combines it with

other data in a constant number of steps. Clearly this computation can be done in $O(\sqrt{p})$ steps. On the serial machine $\Theta(p)$ steps are needed. The computation can also be done in $O(p)$ steps on the parallel machine without the front-end processor by simulating one step of the serial front end with $O(\sqrt{p})$ steps of the mesh, as described in Theorem 1.

To derive a lower bound on computation time when the front end is not used, we consider a slightly more general problem. $O(p)$ steps appear to be needed when the front-end processor is not available because data must be transported great distances within the mesh to simulate each step of the front end. If an adversary knows what data need to be moved, it can store columns so that it can fetch each remote datum in one step. The adversary can be frustrated by generalizing the problem slightly. We add the $\sqrt{p}$ binary number variables $\{c_j \in \{0,1\}^{\lceil \log p \rceil} \mid 0 \leq j \leq \sqrt{p}-1\}$ to the problem. The meaning of these variables is that in the kth row vertex (k, c_k) is the only binary vertex and it uses $\tau_{k-1,c_{k-1}}$ as the remote value. Clearly the values $\{c_j\}$ can be chosen so that the computation is exactly that described by the graph of Figure 2 when columns are stored together.

It is not hard to generalize the algorithm given earlier to compute the value of this new function $f_{MSH}^{p,\sqrt{p}\log p}$ on the Mesh SuperHet in $O(\sqrt{p})$ steps; the front end reads the values of the variables $\{c_j\}$ and not only leaves the value of $\tau_{k-1,c_{k-1}}$ in a special register for the c_kth processor but also leaves a bit to inform it that it must compute the function g instead of $\oplus$. The function can also be computed in $O(p)$ steps on the serial machine or when the front-end processor is disabled.

Theorem 2 *The time to compute the function $f_{MSH}^{p,\sqrt{p}\log p}$ serially or on the 2-D Mesh SuperHet when the front-end processor is not used is $\Theta(p)$, whereas it is $\Theta(\sqrt{p})$ when the complete 2-D Mesh SuperHet is used.*

Proof All results except for the lower bound when the front-end processor is not used follow from the above discussion. The remaining result is a consequence of the fact that the binary vertices in each row are effectively embedded onto the rows of the 2-D mesh in such a way that they map to a path whose length in the mesh is $\Theta(p)$, thereby requiring at least this many steps to compute the function with the mesh and the front-end memory modules.

If another parallel machine is employed, the proof can be extended by embedding the binary vertices so they are far apart on the parallel machine. □

4 Sorting on the Mesh SuperHet

The sorting problem is to put a collection of n elements drawn from a linearly ordered set into ascending order. In this section we examine the performance of comparison-based sorting algorithms on the Mesh SuperHet.

Comparison-based algorithms compare and move pairs of elements but treat all data as indecomposable. We begin by deriving lower bounds on the time to sort on the Mesh SuperHet, and then present and analyze two sorting algorithms for this problem, a simple 2-D one that is asymptotically non-optimal but is nevertheless efficient, and another multidimensional one that is complex but is asymptotically optimal. Throughout we assume that each item to be sorted fits into one memory location.

4.1 A Lower Bound on the Time to Sort

Processors on the Mesh SuperHet can perform comparisons in parallel. The parallel decision tree model of computation, an extension of the well-known serial decision-tree model, captures essential features of such algorithms [8, pp. 185]. As shown in Figure 3 (a), a parallel decision tree performs multiple comparisons in each step and then executes one of several multiway branches. The path to a vertex is associated with the results of comparisons taken to reach it. Thus, the leaves of a full parallel decision tree to sort n words (see Figure 3(b)) identify the $n!$ permutations of these words. It should be noted that the paths through a parallel decision tree may not all have the same length and that several different paths may be associated with the same set of comparisons, although performed in different orders.

We now state and prove a lower bound on the number of steps needed by the Mesh SuperHet to sort n words using only comparisons.

Theorem 3 *The number of steps, $T_{sort}(n, m, p, d)$, to sort n elements on a p-processor, d-dimensional Mesh SuperHet with m words per processor satisfies the following inequality for $n \geq 4$:*

$$T_{sort}(n, m, p, d) \geq \frac{n \log n}{4(1 + 2\, p^{1-1/d} \log(2mp))}$$

Proof Divide time up into intervals of length $mp^{1/d}$. In any given interval let e words enter and l words leave the mesh. Clearly, $e + l \leq mp$. At the beginning of the interval there are at most mp words in the array. It follows that at most

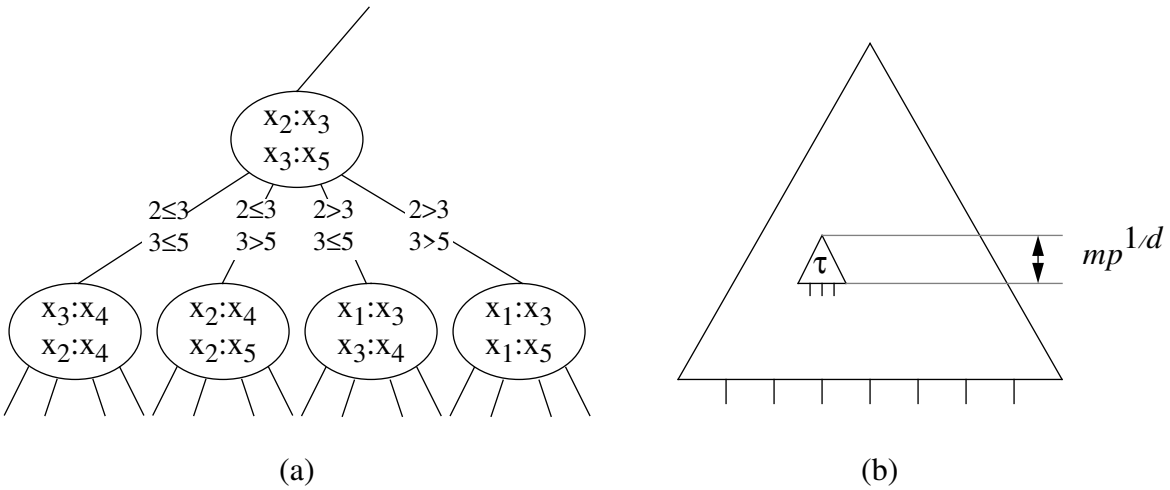

Figure 3: (a) Several levels of a parallel decision tree; (b) a large parallel decision tree with $mp^{1/d}$ levels identified.

$e + l + mp \leq 2mp$ words can interact on the mesh during any interval and that at most $(2mp)!$ permutations of these elements can be identified by the mesh during such an interval. In one step the front end can compare at most two words and at most $mp^{1/d}$ pairs of words can be compared in any one interval on the front end.

Shown in Figure 3(b) is a subtree τ of $mp^{1/d}$ levels of a parallel decision tree corresponding to one time interval. The root of the subtree is associated with a sequence of comparisons made in earlier intervals, and its leaves distinguish the additional decisions made in the interval. Since at most $(2mp)!$ permutations can be distinguished by the mesh and at most $2^{mp^{1/d}}$ by the front end during an interval, it follows that the number of distinguishable leaves of the subtree is at most $N_{subtree}$ where

$$N_{subtree} \leq (2mp)!2^{mp^{1/d}}$$

Consider a set of indistinguishable leaves of a subtree τ and the subtrees of the full tree attached to them. Clearly each of these attached subtrees can be replaced by any one of them without changing the outcomes produced by the parallel decision tree. We make such a change in the decision tree. Thus, it follows that the number of distinct permutations that can be distinguished by the

Mesh SuperHet in $T_{sort}(n,m,p,d)$ steps is at most $(N_{subtree})^{T_{sort}(n,m,p,d)/mp^{1/d}}$. Since the Mesh SuperHet must distinguish $n!$ permutations, we have the following lower bound on $T_{sort}(n,m,p,d)$:

$$T_{sort}(n,m,p,d) \geq \frac{mp^{1/d}\log n!}{(\log(2mp)!) + mp^{1/d}}$$

Since $(q/4)\log q \leq \log(q!)$ for $q \geq 4$ and $\log(q!) \leq q\log q$ for $q \geq 1$, we have

$$T_{sort}(n,m,p,d) \geq \frac{n\log n}{4(1+2\,p^{1-1/d}\log(2mp))}$$

Thus, the speedup increases with the dimension d and the bandwidth. □

This lower bound shows that a speedup of at most about $p^{1-1/d}\log(mp)$ is possible. A full speedup of p is not achievable because it takes time proportional to $mp^{1/d}$ to move mp words on and off the mesh, which is comparable to the time to merge mp words on the mesh. It is interesting to note that, as shown by Dehne *et al.* [5], if all n words are uniformly distributed initially over the p processors in a 2-D mesh they can be sorted with a full speedup if $m = 2^{\Omega(\sqrt{p})}$.

4.2 An Efficient 2D-Mesh SuperHet Sorting Algorithm

We now present an efficient *two-dimensional sorting algorithm* for the Mesh SuperHet. In a pre-processing stage it uses the mesh to sort $2n/mp$ sets of $mp/2$ words each. The front end then merges these sorted lists using a merging algorithm based on an extension of Batcher's bitonic sorter. In the next section we describe a much more complicated algorithm based on one given by Atallah and Tsay [2] for the d-dimensional case and modified to run under the Mesh SuperHet's severe restriction on data movement.

Dehne *et al.* [5] have shown that Batcher's oblivious bitonic sorting algorithm [3] can be converted to a merger of sorted sublists by replacing each compare-exchange operator with a *merge-split operator,* an operator that takes two sorted lists of k elements, merges them, and produces two lists, each of k elements, in which all elements in the first list are less than all elements in the second. Later we sketch a proof that the optimal mesh-sorting algorithm of Schnorr and Shamir [16] can be modified in the same way to create a mesh-merging algorithm.

```
The procedure Sort uses Batcher's bitonic sorter to organize merges
of lists on the mesh using a variant of the Schnorr-Shamir algorithm.

Sort(List)
    {Sort 2n/mp lists of mp/2 elements.}
    for (1 ≤ r ≤ 2n/mp)
      Move the rth list of mp/2 words to the mesh;
      In parallel sort p lists of m/2 elements serially;
      Merge these lists with the Schnorr-Shamir merging algorithm;
      Move the tth sorted list of mp/2 words to the front end;

    {Bitonically merge sorted lists}
    for (1 ≤ r ≤ [(2n/mp) log²(2n/mp)]/4)
      Choose the next pair of lists to be merged bitonically;
      Move the pair of lists to the mesh;
      Merge these lists with the Schnorr-Shamir merging algorithm;
      Move the smaller and larger lists to the front end;
```

Figure 4: A Mesh SuperHet sorting algorithm.

We use Schnorr-Shamir's mesh merger on the mesh and Batcher's bitonic merger on the front end. (The Schnorr-Shamir algorithm leaves words on the mesh in row-major order, which greatly simplifies data movement between the serial and parallel machines.) The full sorting algorithm is sketched in Figure 4.

The n words to be sorted are stored initially on the front end and are uniformly distributed over the $\sqrt{p}$ memory units. In $m\sqrt{p}$ steps mp words are moved to the mesh and stored in the p memories, m words per memory. Each processor serially sorts its list of m words. A Schnorr-Shamir merging algorithm is then used on the mesh to merge the sorted sublists into one list of mp elements. The Schnorr-Shamir algorithm leaves the elements on the array in snake-like row-major order. We assume that the smallest $mp/2$ words are at the top of the mesh and, when returned to the front-end memories, are put in the top $\sqrt{p}/2$ memory units. The largest $mp/2$ words are placed in the bottom $\sqrt{p}/2$ memory units. This placement of data is chosen to simplify subsequent merges. Later in this section we explain how data can be moved efficiently to achieve this objective.

After n/mp stages all n words have been moved to the mesh, sorted and moved back to the front end memory units. Although they are organized as n/mp sorted lists of mp words each, we treat them as $2n/mp$ sorted lists of $mp/2$ words. We then execute Batcher's bitonic merger on these smaller lists. Two lists of $mp/2$ words are moved to the mesh and merged using the Schnorr-Shamir mesh merger. This procedure works correctly as long as the two lists to be merged are in different halves of the front end memory, since in this case they can both be moved to the mesh in parallel in $m\sqrt{p}$ steps. When the two lists are in the same half of the memory, one is moved to its half of the mesh and then routed to the other half. This can be done in $2m\sqrt{p}$ steps. Thus, in at most $3m\sqrt{p}$ steps two sorted lists of $mp/2$ words can be moved onto the mesh. To insure that the front-end memories remain balanced, the two sorted lists resulting from the merge are moved back to the same halves of the front-end memory.

A Schnorr-Shamir mesh merger executes $3\sqrt{p}$ merge-split operations on p lists of m words, each of which is executed serially in $2m$ steps. Thus, a Schnorr-Shamir merge operation takes $6m\sqrt{p}$ steps. The first phase, in which n/mp lists of mp elements are sorted and then merged, takes $O((n/mp)m\log m)$ steps to sort each list and $(n/mp)8m\sqrt{p}$ steps to merge sorted sublists of m words each, including $2m\sqrt{p}$ steps to move data onto and off of the mesh. A total of $T_{first} = O((n\log m)/p + n/\sqrt{p})$ steps is used in the first phase.

In the second phase Batcher's bitonic merger is executed. Since each merge operation in bitonic merging is preceded and followed by a data-movement and each such movement may take as many as $3m\sqrt{p}$ steps, it follows that one merge and data movement stage of bitonic merging takes no more than $12m\sqrt{p}$ steps. Since bitonic merging of k lists takes at most $k(\log^2 k)/4$ merge and data movement steps and $k = 2n/mp$, it follows that the number of steps in the second phase, $T_{bitonic}$, is at most $6(n/\sqrt{p})\log^2(2n/mp)$.

Theorem 4 *A list of n words can be sorted on the 2-D p-processor Mesh SuperHet with m words per processor in $T_{simple.sort}(n,m,p,d)$ steps, where*

$$T_{simple.sort}(n,m,p,d) = O\left(\frac{n}{\sqrt{p}}\log^2\left(\frac{2n}{mp}\right) + \frac{n}{p}\log m\right)$$

When n is large, i.e. when $n \geq (mp/2)2^{\sqrt{(\log m)/\sqrt{p}}}$, the first term dominates and the achievable speedup is approximately $\sqrt{p}$.

We now show that the Schnorr-Shamir mesh sorting algorithm can be modified to merge p sorted lists of m elements each in time $O(mp)$.

Lemma 1 *When each compare-exchange operator in the Schnorr-Shamir $n \times n$ mesh-sorting algorithm is replaced by a merge-split operator, the algorithm correctly merges sorted sublists.*

Proof The Schnorr and Shamir algorithm sorts n^2 items on an $n \times n$ mesh of processors in $3n$ steps which is optimal when each processor holds a constant number of elements and n is of the form $n = 2^{4q}$, q an integer. The algorithm subdivides the mesh into blocks as well as vertical and horizontal slices, and sorts elements within blocks and slices. It suffices to use a linear time sorting algorithm, such as odd-even transposition sort ("bubble sort"), for these steps. Correctness of the merger follows by showing correctness of a merger obtained by replacing compare-exchange operations in odd-even transposition sort with merge-split operations. But this is a consequence of a straightforward application of the 0-1 principle [9], which can be extended to this case. □

4.3 An Asymptotically Optimal Sorting Algorithm

Atallah and Tsay [2] have described an algorithm for a number of sorting-based problems that gives a speedup of $p^{1-1/d} \log p$ when run on a heterogeneous supercomputer consisting of a serial machine and a p-processor, d-dimensional mesh with a small number of memory locations per processor. They assume the two machines are coupled by a channel of unrestricted type with bandwidth proportional to the size of one face of the mesh, $p^{1-1/d}$. In this section we generalize their algorithm to the Mesh SuperHet in which processors on one face of the mesh are connected individually to a like number of front-end memory units. Since the number of memory units involved can be large, efficiency demands that care be taken in organizing the data generated on the mesh to avoid storing large amounts of data in one memory unit that must be transferred serially to the mesh, thereby wasting the available parallelism.

Our extension of the Atallah-Tsay mesh-sorting algorithm to the Mesh SuperHet recursively sorts a list of n items and recursively forms $q = \lceil p^{1/(d+1)} \rceil$ sorted lists of n/q elements each. (Note that $q \leq p^{1/d}$, which is an integer.) To simplify the analysis we assume that $n = q^k$ for some integer k. Our algorithm then finds the $(mp/3)$th smallest item among the n elements and removes it and *most smaller items* from the lists. (The complex details of this removal process are central to our extension of the Atallah-Tsay algorithm.) This process is repeated with the next $(mp/3)$th smallest element until all lists are exhausted. Identification, removal and

```
Sort(List u of n items)
    Divide u into q = ⌈p^{1/(d+1)}⌉ sublists v_j, 1 ≤ j ≤ q;
    Recursively sort sublists into lists w_j, 1 ≤ j ≤ q;
    Merge sorted sublists {w_j, 1 ≤ j ≤ q};

Merge {w_j, 1 ≤ j ≤ q}
    Until done
      Find (mp/3)th smallest item;
      Delete mp/3 smallest items from lists;
      Shorten lists {w_j, 1 ≤ j ≤ q} (Update pointers);
      Move mp/3 smallest items to mesh;
      Sort items on mesh;
      Move sorted items to front end;
```

Figure 5: Sketch of an asymptotically optimal mesh-based sorting algorithm.

sorting of these items is done through coordinated action by the serial front end and the mesh that requires careful data management. At the level of detail shown in Figure 5, this algorithm is a sketch of the Atallah-Tsay algorithm. It differs only in that merging is done by extracting groups of $mp/3$ smallest elements instead of p elements. Our extension to this algorithm requires a careful implementation of the procedures `Merge` and `Extract`.

Let $T_{sort}(n, m, p, d)$ be the number of steps used by this sorting algorithm. We show that the combining phase, in which the complete sorted list is produced from the q sorted lists, can be done in $O(n/p^{1-1/d})$ steps. Thus,

$$T_{sort}(n, m, p, d) \leq qT_{sort}\left(\frac{n}{q}, m, p, d\right) + c\frac{n}{p^{1-1/d}}$$

where c is a positive constant. Also, we show that $T_{sort}((mp/3), m, p, d) = O(mp^{1/d} + m \log m)$. The solution to this recurrence is given below.

Theorem 5 *A set of n elements can be sorted on the p-processor, d-dimensional Mesh SuperHet with m words per mesh processor in $T_{sort}(n, m, p, d)$ steps where*

$$T_{sort}(n, m, p, d) = O\left(\frac{n \log n}{p^{1-1/d} \log(mp/3)} + \frac{n}{p} \log m\right)$$

The second term dominates when $n = 2^{O(\log(mp/3)\log(m)/p^{1/d})}$. *Thus, a full speedup is possible when* $n = m^k$, k *a constant, and* $m = 2^{\Omega(p^{1/d})}/p$. *Otherwise the speedup is at most* $O(p^{1-1/d}\log(mp/3))$.

Proof It is straightforward to show that the bound of the theorem satisfies the initial condition and the recurrence. We now show that the base case and the inductive hypothesis hold.

The mesh is used either to sort or merge lists of items. In every case we insure that each mesh processor has the same number $w \leq m$ of words in its local memory. To sort a list of wp items on the mesh, individual lists of w items are sorted sequentially in $O(w \log w)$ steps and these lists are merged in $O(wp^{1/d})$ steps, as described in the next paragraph. Thus, $T_{sort}(wp, m, p, d) = O(wp^{1/d} + w \log m)$ for $w \leq m$.

We use Batcher's bitonic sorting algorithm, extended to a merging network. On a d-dimensional mesh this network will merge p lists of w sorted items in $wp^{1/d}$ steps but leave the sorted items in positions that are not convenient for the recursive algorithm described below. Therefore, we group the w items in each memory into sorted sublists of $m/3$ elements and route them to the new positions described below. This routing can be done as a merging by assigning as key to each element a pair consisting of its destination mesh memory unit and its value. Thus, a routing can also be done as a succession of up to three mergings in a total of $O(wp^{1/d})$ steps.

A linear order is given to the mesh processors connected to front-end memory units. It is preferable that this order be as close as possible to that generated by the mesh sorting and merging algorithms. We will insure that all sorted lists of more than $mp/3$ items can be divided into sorted sections of exactly $mp/3$ items. By routing they will be distributed over the $p^{1/d}$ planes of the mesh and arranged on each plane as sublists of $m/3$ items per processor, ordered by the linear order of the interface processors. This will divide the list into $p^{1/d}$ sorted lists of $mp^{1-1/d}/3$ items each called *blocks*. A block can be stored in the $p^{1-1/d}$ front-end memories with $m/3$ items per memory unit.

A collection of n elements is divided into $q = \lceil p^{1/(d+1)} \rceil$ lists and sorted recursively. The $mp/3$th element in these sorted lists is then found using the efficient serial algorithm of Frederickson and Johnson
Lspace [7] discussed below. Our goal is to use this element to find and move to the mesh for sorting the $mp/3$ smallest elements in these lists, a step taken in

the Atallah-Tsay algorithm. We modify their procedure as described below to avoid unbalancing the data distribution over the memory modules.

The Frederickson-Johnson algorithm finds the kth smallest element in an $a \times b$ matrix X in which columns are sorted in non-decreasing order (exactly our case) in time $O(b + r \log(k/r))$ where $r = \min(k, b)$. We use the serial front end to execute this algorithm for the case $k = mp/3$, $a = n/q$, $b = q$ and $r = q$. It follows that the $mp/3$th element can be found in time $O(p^{1/d} \log mp)$ when d is a constant since $q \leq p^{1/d}$.

We now describe changes to the procedures `Merge` and `Extract` in Figure 5. It is helpful to consult Figure 6 as the algorithm is described. Our new procedures remove most of the smallest $mp/3$ elements from the q sorted sets by removing blocks (each containing $mp^{1-1/d}/3$ items). On the front end we serially examine blocks in each sorted list to identify those that contain elements less than or equal to the $mp/3$th smallest item. This is done by comparing the $mp/3$th element to the first and last elements in successive blocks. (Blocks are stored in the front-end memory in such a way that these two items are contained in the first and last front-end memory units.) There are at most $p^{1/d} + q \leq 2p^{1/d}$ such blocks, and they can be found in $O(p^{1/d})$ steps. The number of elements in all these blocks is at most mp and they will fit onto the mesh. These blocks are moved to the mesh and stored on mesh planes so that no memory is given more than m items. Those that have fewer than this many items are increased to m items by adding copies of an item ∞ which is larger than all other items. Each block is placed on one plane of the mesh.

To find the first $mp/3$ smallest items, the $mp/3$th smallest item is broadcast to all processors in time $O(p^{1/d})$. The processors on each plane hold at most three blocks of $mp^{1-1/d}/3$ items. Each processor scans its elements in $O(m)$ steps and determines where in its portion of the block the $mp/3$th element falls. In $O(dp^{1/d})$ steps (d a constant) it collects the identities of blocks that contain an item larger than the $mp/3$th smallest item. (If the $mp/3$th smallest element is the last element in a block, such a block is the next block.) This information is transmitted to the front end in $O(p^{1/d})$ steps.

The blocks containing elements larger than the $mp/3$th are written back to the front-end memories, replacing the items less than or equal to the $mp/3$th with ∞, as suggested in Figure 6. This action allows us to remove these smaller items when searching for the next $mp/3$ smallest items. This write-back operation can

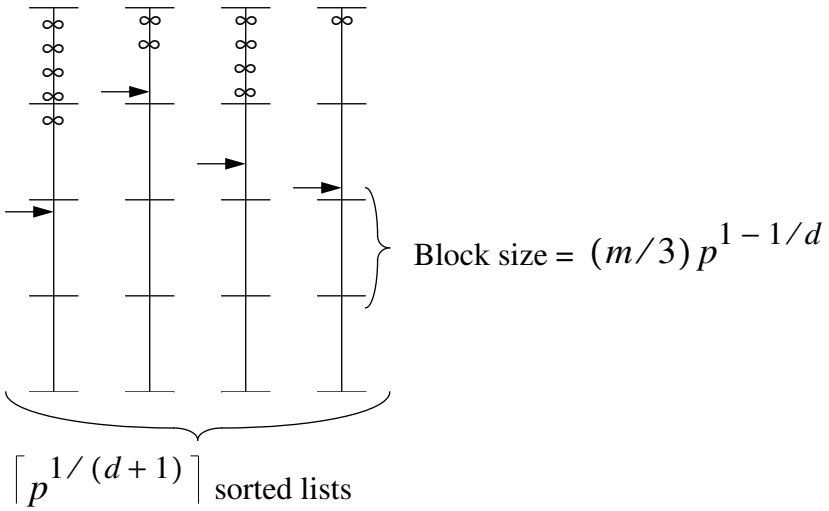

Figure 6: Sorted columns on which **Extract** operates. Columns are subdivided into blocks of $(m/3)p^{1-1/d}$ items which are moved to and from the mesh to extract the $(mp/3)$th next smallest items. Pointers show the next item larger than the next $(mp/3)$ elements in each column.

be done in $O(mp^{1/d})$ steps, the same number taken to move data to the mesh.

Now a list of the $mp/3$ smallest items is formed by assigning value ∞ to larger items, merging lists on individual processors, and merging these lists on the mesh. This can be done in $O(mp^{1/d})$ steps. When the merge is complete, the $mp/3$ smallest items are routed so they are stored in linear order on $p^{1/d}$ planes, $mp^{1-1/d}/3$ items per plane. This take $O(mp^{1/d})$ steps. The $mp/3$ smallest items are then moved to front-end memories in $O(mp^{1/d})$ additional steps.

It follows that the first $mp/3$ items can be identified, removed from the set of q lists, and stored in the front-end memory in $O(mp^{1/d})$ steps.

To find the next $mp/3$ smallest items, the serial machine first shortens its q lists by advancing a pointer within each list to the first item larger than the $mp/3$th, as shown in Figure 6. This is done using the locations transferred to the front end in the previous phase. The serial machine then computes the $mp/3$th smallest item among those remaining in $O(p^{1/d} \log mp)$ steps, again using the serial Frederickson-Johnson algorithm. All blocks containing items that are greater than the $mp/3$th smallest and less than or equal to the $2mp/3$th

smallest are found and moved to the mesh. Note that the elements less than or equal to the $mp/3$th element have previously been set to ∞. There are at most $p^{1/d} + 2q \leq 3p^{1/d}$ of these blocks containing at most mp items and they will fit on the mesh.

After the lists are moved to the mesh, the locations of the largest items less than or equal to the $2mp/3$th smallest item are identified and sent to the front end. The blocks containing an item larger than the $2mp/3$th smallest item are also found and sent to the front end. We write back blocks that contain an item less than or equal to the $2mp/3$th smallest item as well as a larger item, replacing with ∞ all items less than or equal to the $2mp/3$th smallest item. Finally, the lists on each processors are merged serially and these lists are merged in parallel using the mesh merging algorithm. The $2mp/3$ smaller items are routed to the appropriate planes in the appropriate order and then moved to the front end. These operations take $O(mp^{1/d})$ time.

This process is repeated a total of $3n/mp$ times, each such global step requiring $O(mp^{1/d})$ time. Thus, the total time to combine the q sorted lists is $O(n/p^{1-1/d})$. □

5 The Fast Fourier Transform Algorithm

The fast Fourier transform (FFT) algorithm on n inputs computes the discrete Fourier transform (DFT) in time $O(n \log n)$ on a serial computer when $n = 2^k$ for some integer k. In this section we develop Mesh SuperHet algorithms for it.

The DFT on n inputs $\underline{a} = (a_0, a_1, \ldots, a_{n-1})$ over a commutative ring R, $F(\underline{a}) = (f_0, f_1, \ldots, f_{n-1})$, is the value of the following polynomial $p(x)$:

$$p(x) = a_0 + a_1 \times x + a_2 \times x^2 + \ldots + a_{n-1} \times x^{n-1}$$

on the n roots of unity, ω^j, $0 \leq j \leq n-1$, i.e.

$$f_r = \sum_{k=0}^{n-1} a_k \times \omega^{kr} = p(\omega^r)$$

Here $\{a_i, \omega^i \mid 0 \leq i \leq n-1\}$ are elements of R and $+$ and $\times$ are the ring operations. The roots of unity satisfy $\omega^n = 1$ and $\sum_{k=0}^{n-1} \omega^{kl} = 0$ for $1 \leq l \leq n-1$.

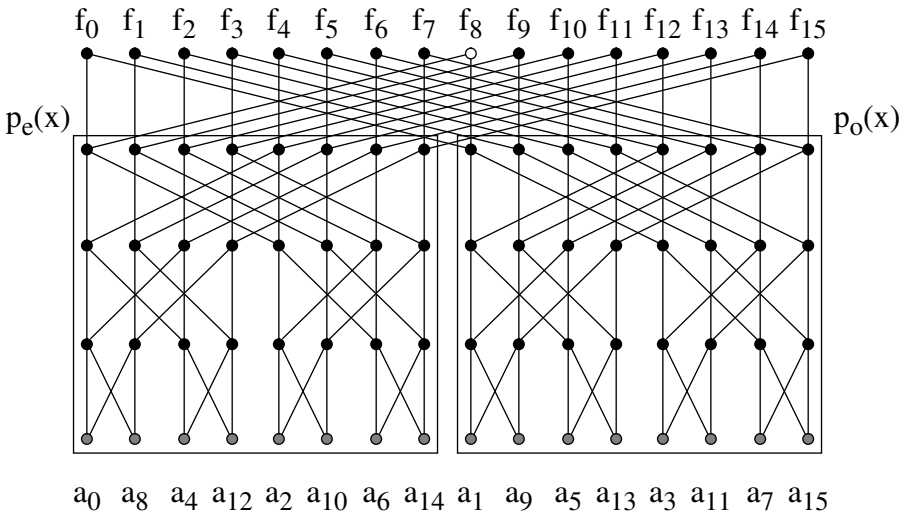

Figure 7: The graph $F^{(4)}$ of the FFT algorithm on 16 inputs.

The FFT algorithm follows from the decomposition of the polynomial $p(x)$ into even and odd polynomials given below. The FFT graph $F^{(k)}$ describes the operations and data movement in the FFT algorithm on $n = 2^k$; $F^{(4)}$ is shown in Figure 7.

$$\begin{aligned} p(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \\ &= (a_0 + a_2 x^2 + \ldots + a_{n-2} x^{n-2}) + \; x\,(a_1 + a_3 x^2 + \ldots + a_{n-1} x^{n-2}) \\ &= p_e(x^2) + x p_o(x^2) \end{aligned}$$

The FFT algorithm on n inputs, n a power of two, can be computed serially in $O(n \log n)$ steps. On a suitable n-processor parallel machine, the FFT algorithm can be computed in $O(\log n)$ steps.

We consider computing the FFT on the Mesh SuperHet. The algorithm we develop depends in an essential way on the decomposition property of the FFT stated below. Figure 8 illustrates the decomposition spelled out below.

Lemma 2 *[14] For $k \geq 2$ and each $1 \leq r \leq k-1$ the FFT graph $F^{(k)}$ on $n = 2^k$ inputs can be represented as the composition of 2^{k-r} disjoint topFFT subgraphs $F^{(r)}_{t,i}$ on 2^r inputs, $0 \leq i \leq 2^{k-r}-1$, with 2^r disjoint FFT bottomsubgraphs $F^{(k-r)}_{b,j}$ on 2^{k-r} inputs, $0 \leq j \leq 2^r - 1$, where the jth input vertex of $F^{(r)}_{t,i}$ is the ith output vertex of $F^{(k-r)}_{b,j}$. The qth output vertex of $F^{(k)}$ is the uth output of $F^{(r)}_{t,v}$ for the unique integers u and v satisfying $q = u2^{k-r} + v$, $0 \leq v \leq 2^{k-r} - 1$.*

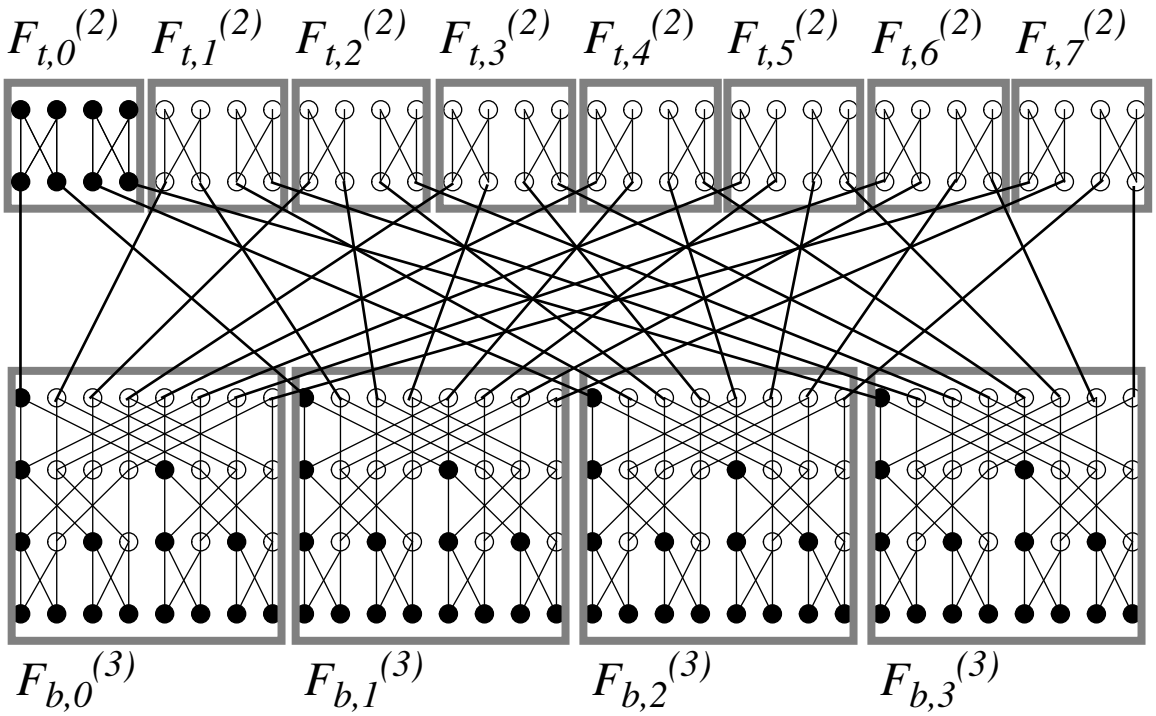

Figure 8: Decomposition of the FFT graph $F^{(5)}$ on 32 inputs.

We simplify the following analysis without materially changing the results by assuming that m and p are powers of 2.

Since the d-dimensional, p-processor Mesh SuperHet can store mp entries on the mesh, we first consider the computation on the mesh of $F^{(a)}$ for a such that $2^a = mp$. As necessary, we assume the existence of additional temporary storage for local computations and data movement.

To compute $F^{(a)}$ we decompose it into $2^{a-r} = m$ top subgraphs $\{F_{t,i}^{(r)}\}$ on $2^r = p$ inputs and $2^r = p$ bottom subgraphs $\{F_{b,j}^{(a-r)}\}$ on $2^{a-r} = m$ inputs. The bottom subgraphs are computed in parallel in $O(m \log m)$ steps on the p processors using the serial algorithm. The top FFT subgraphs are assigned to processors with the first m/p sugraphs assigned to the "first" of the p processors, the second m/p subgraphs assigned to the "second," etc. Then, the m outputs produced by the bottom subgraphs are divided into consecutive groups of m/p outputs and moved to the processors which compute the top subgraphs. This step, which amounts to a *matrix transpose*, can be done parallel in $O(mp^{1/d})$ steps using an exten-

sion to Batcher's sort, as discussed above. The top subgraphs are computed in $O((m/p)(p \log p)) = O(m \log p)$ steps. It follows that $F^{(a)}$ for $2^a = mp$ can be computed on the mesh in $O(m \log mp)$ steps.

Let's now apply the decomposition lemma to the computation of the FFT graph $F^{(k)}$ on $n = 2^k$ inputs. We decompose such graphs into $2^a = mp$ top FFT subgraphs on $2^{k-a} = n/mp$ inputs and $2^{k-a} = n/mp$ bottom FFT subgraphs on $2^a = mp$ inputs. The inputs to the ith top subgraph are the ith outputs of the bottom subgraphs. A storage strategy is needed for the outputs of the bottom sugraphs to insure that they are available in parallel to the top sugraphs. (This is another example of a *matrix transposition.*) If we were to store the mp outputs of each bottom subgraph in the same way, a conflict would result because the first top FFT subgraph would have to access all of its inputs from the first memory unit. We show that data can be organized to make it available in parallel to top FFT subgraphs.

Transposition of Data Processors on a mesh face are given a linear order. We use that order in this discussion. Data is transferred to front-end memory units in blocks of m words. We first examine the storage of the mp values of the first bottom subgraph. We treat these values as organized into p blocks of m words, $\{[1, 2, \ldots, m], [m+1, m+2, \ldots, 2m], \ldots, [(p-1)m+1, (p-1)m+2, \ldots, pm]\}$. We reorganize this data so that the qth block contains the entries $[q, p+q, 2p+q, \ldots, (m-1)p+q]$ for $1 \leq q \leq m$. The first $p^{1-1/d}$ such blocks are stored in parallel in the front-end memories, one block per memory. The second set of $p^{1-1/d}$ blocks are also stored one per memory. The second block in each memory unit is a block from the second set of $p^{1-1/d}$ blocks. This process is repeated until all mp results from the first bottom subgraph are stored. For the second bottom subgraph we permute the blocks cyclically right one place. For the lth block we permute the blocks cyclically right l places. The data is now available in parallel to top subgraphs. All of these operations can be done in $O(mp^{1/d})$ steps.

Now that the data-storage procedure is clear, we assume that n is divisible by mp and again use the FFT decomposition, this time on top subgraphs. It follows that if $T_{FFT}(n, m, p, d)$ is the number of steps to compute the n-input FFT on the p processor, d-dimensional Mesh SuperHet with m memory words per mesh processor, we have the following bounds on the number of computation steps:

$$\begin{aligned} T_{FFT}(mp, m, p, d) &= O(m \log mp + mp^{1/d}) \\ T_{FFT}(n, m, p, d) &\leq mp\, T_{FFT}(\frac{n}{mp}, m, p, d) + \frac{n}{mp}\left(T_{FFT}(mp, m, p, d) + O(mp^{1/d})\right) \end{aligned}$$

Theorem 6 *The FFT algorithm on n inputs can be computed on the p-processor, d-dimensional Mesh SuperHet with m words per mesh processor in $T_{FFT}(n, m, p, d)$ steps where*

$$T_{FFT}(n, m, p, d) = O\left(\frac{n \log n}{p^{1-1/d} \log mp}\left[1 + \frac{\log mp}{p^{1/d}}\right]\right)$$

Proof The bound shown above satisfies the base case of $n = mp$ and the recurrence given above. □

This algorithm provides a speedup of $O(p^{1-1/d} \log mp)$ when $m = 2^{O(p^{1/d})}/p$, the same speedup that applies to the sorting of large lists. This is not an artifact of the algorithm but is inherent in the problem and the architecture of the machine, as we show below. If p is moderately large, this condition on m will be satisfied.

5.1 A Lower Bound on the Time to Compute the FFT

A lower bound on the time to compute the FFT on the Mesh SuperHet can be derived from the lower bound on sorting for this machine and the following lemma.

Lemma 3 *[17] A permutation network on n inputs, $n = 2^k$, can be constructed from three FFT graphs by connecting the outputs of one to the inputs of another and interpreting each node of the graphs as compare-exchange operations.*

A permutation network can be used to sort its inputs. If we replace the operations at every node of an FFT graph with a compare-exchange operation, we make only a constant factor change in the computation time of a Mesh SuperHet algorithm for it. From this follows the lower bound given below.

Theorem 7 *The number of steps to compute the FFT on n elements on a p-processor, d-dimensional Mesh SuperHet with m words per processor, $T_{FFT}(n, m, p, d)$, satisfies the following inequality for $n \geq 4$:*

$$T_{FFT}(n, m, p, d) = \Omega\left(\frac{n \log n}{1 + 2\, p^{1-1/d} \log(2mp)}\right)$$

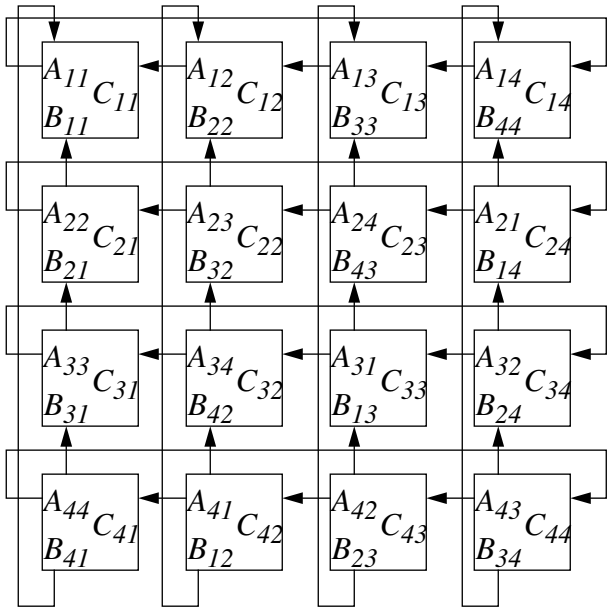

Figure 9: Placement of matrix elements in a 2-D mesh before starting the computation of the product $C = A \times B$. Elements in C remain stationary while those in A and B move horizontally and vertically, respectively.

6 Matrix Multiplication

We now examine multiplication of two $n \times n$ matrices on the d-dimensional Mesh SuperHet under a limitation on the number of channels between front-end memory units and mesh-face processors. We assume that there are at most $\beta \leq p^{1/d}$ such channels and that they are uniformly distributed over the mesh-face. The goal of this section is to understand the effect of β on performance of matrix multiplication on the Mesh SuperHet. We begin by studying a 2-D systolic-based matrix multiplication algorithm [13] which we extend to an arbitrary number of dimensions. We show that if the storage capacity of the mesh is large enough relative to β and p, a full speedup is possible on the Mesh SuperHet.

Shown in Figure 9 is an $s \times s$ 2-D mesh on which are laid out the entries of the $s \times s$ matrices $A = \{A_{i,j}\}$ and $B = \{B_{i,j}\}$. We assume that the (i, j) entries of each matrix are stored in the positions shown. (These positions are obtained from the standard positions by rotating the ith row of A left cyclically $i - 1$ places and rotating the jth column of B up cyclically $j - 1$ places.) The product of the

two matrices is then formed systolically, as suggested in Figure 9. In parallel each processor multiplies its two entries $A_{i,k}$ and $B_{k,j}$ and, except for the first step, adds the product to the previously accumulated value. The resulting sum $C_{i,j}$ stays in place and products are accumulated by rotating the rows of A left cyclically and the columns of B up cyclically until $C_{i,j}$ has been formed. This takes $O(s)$ steps, a speedup of order s^2 (the number of processors) over the equivalent serial algorithm.

This algorithm also works if $A_{i,k}$ and $B_{k,j}$ are matrices themselves, in which case the products $A_{i,k}B_{k,j}$ are also matrix multiplications. Consider the case in which $A_{i,k}$, $B_{k,j}$ and the result matrix $C_{i,j}$ are $\sqrt{m/3} \times \sqrt{m/3}$ matrices. The products $A_{i,k}B_{k,j}$ can be formed in $2m^{3/2}/3\sqrt{3}$ steps by a serial algorithm on each mesh processor and the result added to the previous value. The number of steps to move two such matrices from one processor/memory node to another is $2m/3$. Thus, it follows that the product of two $s\sqrt{m/3} \times s\sqrt{m/3}$ matrices can be formed in $s(2m^{3/2}/3\sqrt{3} + 2m/3))$ steps on the 2-D mesh. This represents a speedup on the order of s^2 over the equivalent serial algorithm.

We now extend the algorithm shown in Figure 9 to d dimensions, d even. Let a d-dimensional mesh be used to multiply two $s^{d/2}\sqrt{m/3} \times s^{d/2}\sqrt{m/3}$ matrices A and B. Treat each matrix as an $s \times s$ matrix whose entries are $s^{(d-2)/2}\sqrt{m/3} \times s^{(d-2)/2}\sqrt{m/3}$ matrices. Each node in the d-dimensional mesh is indexed by a d-tuple $\underline{a} = (a_1, a_2, \ldots, a_d)$, $0 \leq a_i \leq s-1$. The nodes with the same values for a_{d-1} and a_d form a $(d-2)$-dimensional mesh. When these submeshes are treated as nodes, they form a 2-D mesh. Place the two matrices A and B on the d-dimensional mesh so that their $s^{(d-2)/2}\sqrt{m/3} \times s^{(d-2)/2}\sqrt{m/3}$ submatrices are assigned to the appropriate $(d-2)$-dimensional submeshes. Multiply the matrices A and B with the algorithm shown in Figure 9 by multiplying their submatrices on the $(d-2)$-dimensional submeshes and shifting and adding the submatrices on the 2-D mesh defined by a_{d-1} and a_d. We assume as our inductive hypothesis that the product of two $s^k\sqrt{m/3} \times s^k\sqrt{m/3}$ matrices can be formed on the $2k$-dimensional mesh in $T_{mesh}(s, m, 2k)$ steps where

$$T_{mesh}(s, m, 2k) \leq s^k 4 \left(\frac{m}{3}\right)^{3/2} + ms\frac{s^k - 1}{s - 1}$$

This certainly holds for $k = 1$. Now consider the number of steps to multiply the matrices A and B when treated as two $s \times s$ matrices. The number of steps to add the $s^{(d-2)/2}\sqrt{m/3} \times s^{(d-2)/2}\sqrt{m/3}$ product of submatrices to the previous value of a product is $m/3$ and the number of steps to shift two such submatrices cyclically

one place is $2m/3$. It follows that

$$T_{mesh}(s, m, d) \leq s(T_{mesh}(s, m, d-2) + m)$$

which satisfies the inductive hypothesis.

Consider now the computation of the product $W = U \times V$ of two $n \times n$ matrices U and V on the $p = s^d$-processor 2-D Mesh SuperHet when n is divisible by p. Let $q = s^{d/2}\sqrt{m/3}$ and represent the $n \times n$ matrices U, V and W as $(n/q) \times (n/q)$ matrices containing as entries $q \times q$ matrices, as suggested below for $1 \leq r, s \leq (n/q)$:

$$U = (U_{r,s}) \;\; V = (V_{r,s}) \;\; W = (W_{r,s})$$

The product $W = U \times V$ is formed by taking the block inner products of rows of U with columns of V. These inner products are done with $(n/q)^3$ products and $(n/q)^2(n/q - 1)$ sums of $q \times q$ block matrices. The matrix multiplication algorithm uses the mesh to form such inner products.

Consider the inner product of the rth row of U with the sth column of V, namely,

$$W_{r,s} = \sum_{t=1}^{n/q} U_{r,t} V_{t,s}$$

When d is even, for each value of $1 \leq t \leq n/q$, move $U_{r,t}$ and $V_{t,s}$ to the mesh, multiply and discard them but retain their product P. The partial sum of $W_{r,s}$ is moved to the mesh and added to P and the result stored back on the front-end memory. When d is odd, represent U, V and W as $(n/q) \times (n/q)$ matrices with $q = s^{(d-1)/2}\sqrt{m/3}$. Move s pairs of submatrices $U_{r,t}$ and $V_{t,s}$ to the mesh and operate on them as described above.

Suppose now that β of the processors on one edge of the array are connected to a like number of front-end memory units. Let them be uniformly distributed over one face of the mesh. The number of steps necessary to move one $q \times q$ matrix onto or off of the mesh is $O(ms^d/\beta)$. From this follows the bound given below on the time for this matrix multiplication algorithm:

Theorem 8 *The number of steps to multiply two $n \times n$ matrices on the p-processor d-dimensional Mesh SuperHet with β channels between the mesh and front end, $T_{matrix}(n, m, p, d, \beta)$, satisfies the following upper bound:*

$$T_{matrix}(n, m, p, d, \beta) = O\left(\frac{n^3}{p}\left[1 + \frac{p^{\lceil d/2 \rceil / d}}{\sqrt{m}\beta}\right]\right)$$

Thus, a full speedup is possible when $m = \Omega(p^{e(d)}/\beta^2)$, where $e(d) = 2\lceil d/2 \rceil / d$ is 1 if d is even and $(1 + 1/d)$ otherwise. When m a constant and $d = 2$ or 3, a constant fraction of the $p^{1-1/d}$ channels on a mesh face must be used to provide a full speedup. For larger values of d at most $\beta = \Theta(p^{e(d)/2}) = O(p^{\frac{1}{2}+\epsilon})$ suffice, where $\epsilon = 1/2d$. When β is constant, m must satisfy $m = \Omega(p^{e(d)})$ to achieve a full speedup.

Proof When d is even we have the following bound for $q = s^{d/2}\sqrt{m/3}$:

$$\begin{aligned} T_{matrix}(n, m, p, d, \beta) &\leq \left(\frac{n}{q}\right)^3 \left(T_{mesh}(s, m, d) + \frac{ms^d}{\beta}\right) \\ &= O\left(\frac{2n^3}{p}\left[1 + \frac{s^{d/2}}{\beta\sqrt{m}}\right]\right) \end{aligned}$$

When d is even we have the following bound for $q = s^{(d-1)/2}\sqrt{m/3}$:

$$\begin{aligned} T_{matrix}(n, m, p, d, \beta) &\leq \frac{1}{s}\left(\frac{n}{q}\right)^3 \left(T_{mesh}(s, m, d-1) + \frac{ms^d}{\beta}\right) \\ &= O\left(\frac{2n^3}{p}\left[1 + \frac{s^{(d+1)/2}}{\beta\sqrt{m}}\right]\right) \end{aligned}$$

The bounds differ only in the exponent on s in the right-hand terms. □

As this result demonstrates, a full speedup can be achieved by matrix multiplication when only one channel ($\beta = 1$) is available if the amount of local memory on each mesh processor grows approximately linearly with the number p of processors. However, if the amount of local memory is constant, a large fraction of the channels is needed for two- and three-dimensional meshes to achieve the same result. For higher dimensions, the number of channels must grow approximately as $\sqrt{p}$.

7 Conclusion

We have introduced the Mesh SuperHet, a model reflecting essential features of realistic closely coupled heterogeneous supercomputers. The model combines a parallel p-processor, d-dimensional toroidal mesh with a serial front end whose memory is divided into modules each of which is accessible to a processor on one face of

the mesh. The model consists of two closely coupled computers, one containing a low-diameter network connecting its memory modules (the front end) and another containing a high-diameter network connecting its memories (the mesh). Coupling is between a face of the mesh and individual front-end memory modules.

We have exhibited problems for the 2-D Mesh SuperHet which execute in time $O(\sqrt{p})$ if the front end is used but in time $\Omega(p)$ if it is disabled, thereby showing for the first time that for some problems a heterogeneous supercomputer is essential for efficient execution. However, we have also shown that the serial front end can be disabled without more than a constant-factor loss in the running time T for those problems that use the front end for fewer than $O(T/p^{1/d})$ steps.

We have derived lower bounds on the time to sort and compute the FFT on the Mesh SuperHet and have shown that a full speedup is not possible with this architecture unless the degree of the mesh (and the bandwidth between it and the serial machine) is very large. For any finite degree mesh a full speedup is impossible. We have given a simple, fast Mesh SuperHet sorting algorithm based on Batcher's bitonic sorter as well as a complex, asymptotically optimal sorting algorithm which is an extension of one proposed by Atallah and Tsay for a less restricted heterogeneous supercomputer. We have also given an asymptotically optimal implementation of the FFT algorithm. We have examined matrix multiplication and shown that a full speedup for this problem is possible even with a small-bandwidth channel between the mesh and the serial machine if the memory per mesh processor is sufficient.

As heterogeneous supercomputers become more prevalent, they will be used for a large variety of computationally intensive tasks. Thus, it is important to understand the opportunities and limitations of this architecture. Not only must experiments be conducted to determine how best to exploit the architectural details of existing machines, studies such as this must be done to understand the fundamental limitations on such architectures and set the context for algorithm development.

8 Acknowledgements

Thanks are due to Jim Anderson, Jim Doll, Gerry Guralnik, and Don McClure for discussions of heterogeneous supercomputing and to David Carlson, Chuck Fiduccia, Philip Klein, Franco Preparata, Jim Smith, Jim Schwarzmeier and Markus Wloka for their comments on earlier versions of this paper.

Bibliography

[1] "Super Linkage," *Science* 251 (March 15, 1991), 1311.

[2] M. J. Atallah and J. -J. Tsay, "On the Parallel-Decomposability of Geometric Problems," *Algorithmica* 8 (1992), 209–231.

[3] K. E. Batcher, "Sorting Networks and Their Applications," *Procs. AFIPS Spring Joint Computer Conference* 32, 307–314.

[4] G. Bilardi and F. P. Preparata, "Horizons of Parallel Computation," *Springer LNCS 653* (Dec. 1992), 155–173.

[5] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers," *Procs. ACM Symposium on Computational Geometry*, 298–307.

[6] P. J. Denning, "The Working Set Model for Program Behavior," *Communications of the ACM* 11 (May 1968), 323–333.

[7] G. N. Frederickson and D. B. Johnson, "The Complexity of Selection and Ranking in X+Y and Matrices with Sorted Columns ," *Journal of Computer and System Sciences* 24 (1982), 197–208.

[8] J. JáJá, in *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1992.

[9] D. E. Knuth, *The Art of Computer Programming – Sorting and Searching* #3, Addison-Wesley, Reading, Mass., 1973.

[10] H. T. Kung, "Memory Requirements for Balanced Computer Architectures," *Journal of Complexity* 1 (1985), 147–157.

[11] R. E. Ladner, "The Circuit Value Problem is Log Space Complete for P," *ACM SIGACT News* 7 (1975), 18–20.

[12] H. Nicholas, G. Giras, V. Hartonas-Garmhausen, M. Kopko, C. Maher, and A. Ropelewski, "Distributing the Comparison of DNA and Protein Sequences Across Heterogeneous Supercomputers," *Procs. Supercomputing '91* (Nov. 18-22, 1991), 139–146.

[13] F. P. Preparata and J. E. Vuillemin, "Area-Time Optimal VLSI Networks for Multiplying Matrices," *Inf. Proc. Let.* 11 (1980), 77–80.

[14] J. E. Savage and S. Swamy, "Space-Time Tradeoffs on the FFT Algorithm," *IEEE Trans. on Info. Th.* IT-24 (Sept. 1978), 563–568.

[15] J. E. Savage, "Space-Time Tradeoffs in Memory Hierarchies," Department of Computer Science, Brown University, Report CS93-08, February 1993.

[16] C. P. Schnorr and A. Shamir, "An Optimal Sorting Algorithm for Mesh Connected Computers," *Procs. 18th Annl. Symposium on Theory of Computing* (May 28-30, 1986), 255–263.

[17] C. L. Wu and T. Y. Feng, "The Universality of the Shuffle-Exchange Network," *IEEE Trans. Computing* C-30 (May 1981), 324–332.