Performance analysis of the FFT algorithm on a shared-memory parallel architecture

by Ž. Cvetanović

This paper presents a model for the performance prediction of FFT algorithms executed on a shared-memory parallel computer consisting of N processors and the same number of memory modules. The model applies a deterministic analysis to estimate the communication delay through the interconnection network by assuming that all requests arrive at the network in bursts. Our results indicate that the communication delay is significantly affected by the method applied to allocate data to memory modules. For the case in which all data items referenced by a processor during an iteration are allocated to a single memory module, the best-case communication time complexity grows as $O[(\log N)^2/N]$. The worst-case communication time complexity for this case, obtained by a different allocation of data to memory modules, is increased to $O[(\log N)/\sqrt{N}]$ due to high network contention. For the case in which the data items referenced by different processors during an iteration are allocated to the same memory module, the communication time complexity is further increased to $O(\log N)$ since all N requests generated by processors are serialized at a single memory module. The methods developed in this paper can be applied for the performance prediction of other well-structured parallel iterative algorithms.

1. Introduction

Analyzing and predicting the performance of multiple-processor systems is a very complex task, since many factors jointly determine system behavior. In this paper, we present a model which provides methods for better understanding of the interaction among various factors which influence the performance. As a case study we have chosen the one-dimensional and two-dimensional Fast Fourier Transform (FFT) algorithms executed on a shared-memory parallel computer. The FFT algorithms play an important role in mathematical and numerical analysis. Some of the applications of FFT algorithms include time-series and wave analysis, spatial correlation, particle simulations, solvers of linear partial differential equations, Poisson's equation solvers, convolution, and digital filtering. Various parallel FFT algorithms have been studied by Pease [1] and by Hockney and Jesshope [2], and their adaptation for

execution on the shuffle-exchange parallel computer was investigated by Stone [3].

Our analysis is restricted to the parallel architecture consisting of N processors and the same number of global and local memories, where processors and global memories are connected through an interconnection network. Variations of such architecture include the NYU Ultracomputer by Gottlieb et al. [4], the more recently described IBM RP3 by Pfister et al. [5], the University of Illinois CEDAR computer by Gajski et al. [6], and the BBN Butterfly™ computer by Thomas et al. [7].

Although the scope of this paper is limited to the FFT algorithms and a particular class of parallel architectures, the methodology developed in this paper can be used to estimate the performance of other parallel programs and a variety of architectures, as demonstrated by Cvetanović [8]. The technique developed here provides methods for obtaining closed-form expressions for the total execution time on a parallel computer as a function of several parameters. With this function, we can study how different parameters affect the performance of parallel programs in order to determine their combination such that the total system design achieves its best cost-effectiveness.

Many models based on probabilistic methods have been proposed for performance analysis of parallel programs. Examples of such models include studies by Baskett and Smith [9], Bhandarkar [10], Heidelberger [11], Dubois and Briggs [12], and Mudge and Al-Sadoun [13]. The crucial difference between these models and our approach is in estimating the communication delay through the interconnection network. In probabilistic models, a certain probability distribution for an arrival process is assumed in order to estimate an average delay through the interconnection network. In our model, we perceive that the requests arrive at the network in bursts. This assumption is justified by the fact that the computation for FFT algorithms can be uniformly distributed among processors. Moreover, according to the parallel algorithms chosen, all processors are synchronized following each iteration of the algorithm (barrier synchronization). Since the communication pattern for these algorithms is defined for each iteration, we can apply a deterministic analysis to compute the execution time exactly instead of estimating an average time by using probabilistic methods.

The major questions we discuss in this paper include the following:

1. How is the performance of parallel FFT algorithms affected by altering the allocation of data to memory modules?
2. How does the performance of parallel FFT algorithms change as a function of problem size? How is performance affected by the match between the number of processors and the problem size?
3. How is the performance of the one-dimensional FFT different from the performance of the two-dimensional FFT of the same size?
4. Is it better to access global data instead of copying data to local memories?
5. How is performance affected by the level of granularity? What is the best grain size for a given problem size?
6. How is performance affected by architecture-dependent parameters such as processor speed and network speed?

Our results indicate that the communication delay and therefore the performance of a system are significantly affected by the method of allocating data to memory modules. For the case in which all data items referenced by a processor during an iteration are allocated to a single memory module, the best-case communication delay is obtained by applying the allocation which results in no contention within the interconnection network. The communication time complexity for the fixed-size problem and N processors grows as $O[(\log N)^2/N]$. The worst-case communication delay complexity for this case is obtained by applying a different strategy for allocating data to memory modules which results in the maximum possible number of conflicts within the network, and the delay increases to $O[(\log N)/\sqrt{N}]$. For the case in which all data items referenced by a processor during an iteration are allocated to different memory modules, the communication time complexity increases to $O(\log N)$. The reason for this is that all N requests from processors accessing data allocated to a single memory module need to be serialized at this module.

Our model also provides means for comparing two different data-access modes: where all data are accessed from global memories, and where data items are first copied to local memories and then accessed locally. If the number of processors is smaller than the problem size, our results indicate that the case with local/global copying performs better than the case with global data access, since copying a block of data can take advantage of pipelining data through stages of the network. This is particularly beneficial for the two-dimensional FFT, since in this case no performance improvement can be obtained with global data access when the problem size exceeds the number of processors.

The next section describes the model for computing the execution time and states the assumptions. In Section 3, we derive expressions for the best-case execution time, while in Section 4 we derive expressions for the worst-case execution time for the allocation in which all data referenced by a single processor during an iteration are allocated to a single memory module. In Section 5, we derive expressions for the execution time for the allocation in which all data items referenced by a single processor during an iteration are allocated to different memory modules. In Section 6, we compare our results to the results from the model proposed

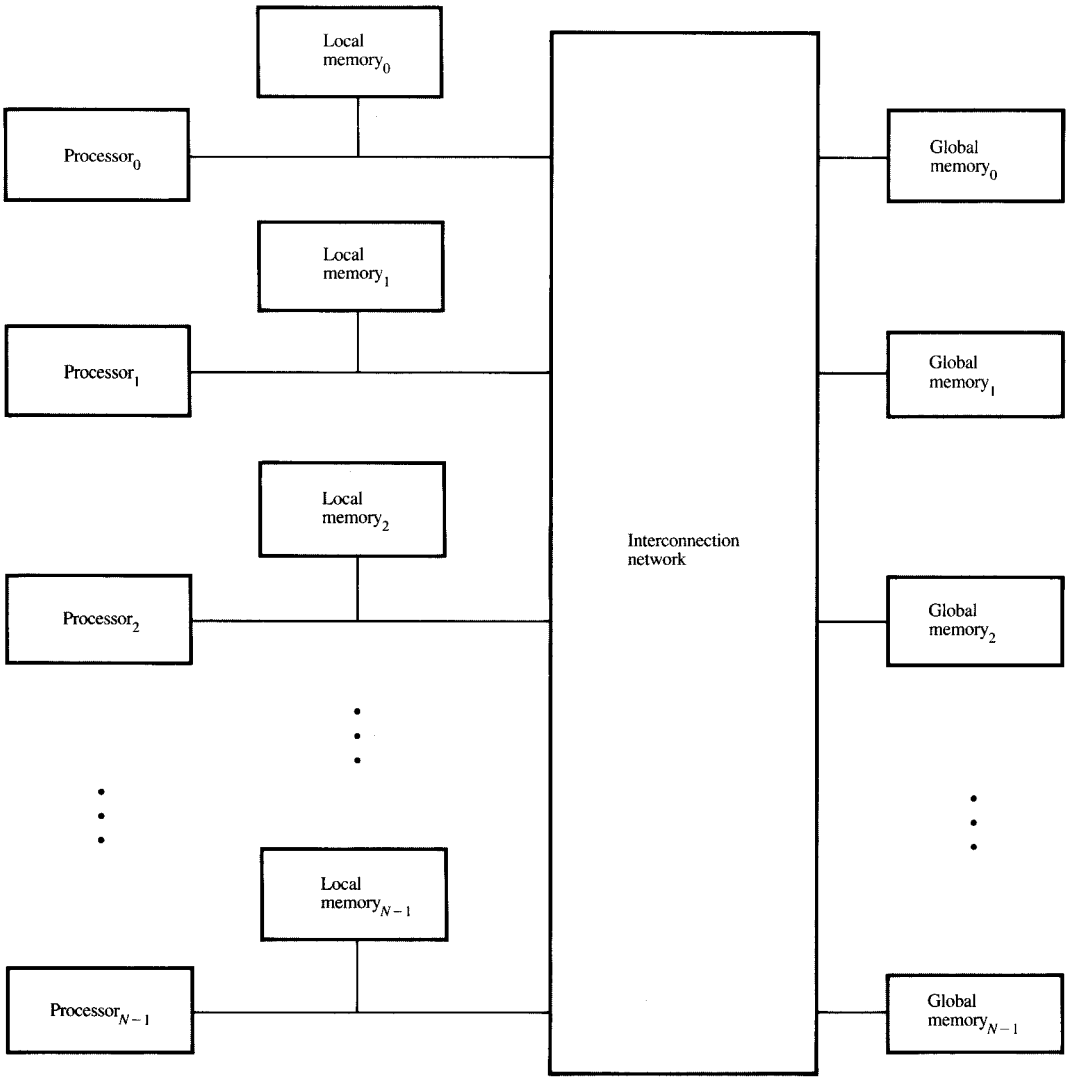

Figure 1

Shared-memory architecture.

by Norton and Silberger [14]. The conclusions and a general discussion appear in the last section.

2. Model and assumptions

We restrict our analysis to a parallel architecture consisting of N identical processors and N memory modules, with two levels of memory hierarchy. Each processor has associated with it a local memory, and we assume that the delay for accessing local data is included in the computation time. Processors are connected to global memories through an interconnection network, so that each processor can access any memory module. Each data access from any of N global memories requires traversing the interconnection network. **Figure 1** illustrates such architecture, which comprises N processors, N memory modules, and the interconnection network.

As a special case of the interconnection network, we assume the omega network proposed by Lawrie [15]. An $N \times N$ omega network with 2×2 switches consists of $\log N$ identical stages. Each stage consists of $N/2$ switches connected by the perfect-shuffle interconnection. An example of a 16×16 omega network is shown in Figure 1 of [16]. We further assume that each switch includes an infinite-length queue associated with every input port for maintaining multiple messages. Variations of such a network are proposed for many research multiple-processor prototypes being built at present, such as the IBM RP3 [5], the CEDAR [6], and the TRAC [17].

The FFT algorithms we analyze belong to a class of synchronous algorithms where all processors are synchronized following each iteration of the algorithm, and wait for the last processor to complete an iteration before proceeding to the next one. At the beginning of each iteration, processors must load data items for which the FFT is computed, which results in the first burst of requests arriving at the network. While the computation is performed, no communication through the interconnection structure is required. Upon completion of the computation, all processors store their results in global memories, which produces the second burst of data arriving at the network. Therefore, such synchronized iterative parallel algorithms are characterized by bursty arrivals at the interconnection network. Since the communication pattern for most of these algorithms is defined for every iteration, we can apply a deterministic analysis to compute the communication delay.

The model we describe encompasses the following assumptions:

1. In order to simplify expressions for the computation time of the FFT algorithms for M data items, we consider only factors with complexity of $O(M \log M)$ and neglect all factors with lower complexity. This assumption is justified by the fact that for large problems, the factors with lower complexity have less influence on the total execution time.
2. For the purpose of our analysis, we further assume that the additional overhead introduced by converting the algorithms into a form suitable for parallel execution can be neglected. Although this is not a realistic assumption, since parallel FFT algorithms require some extra computation for spawning parallel tasks and barrier synchronization, these factors represent a small portion of the computation time and for the purpose of our complexity analysis can be neglected. Note that our model can be enhanced to include this overhead in expressions for the total execution time.
3. We assume that no overlap is possible between computation and communication time slots. In [8] we have extended the analysis to the case where such overlap is feasible.
4. In order to further simplify the analysis, we also assume that both the problem size M and the number of processors N can be expressed as a power of two.

We now derive a global expression for the communication delay of FFT algorithms for the class of architectures described above. The effective communication time T_{ec} for a burst of arrivals at the interconnection network is computed as a sum of the network latency and the queuing delay. The latency is proportional to the number of network stages, and the queuing delay is directly proportional to the number of requests arriving in a burst and inversely proportional to the network bandwidth, and T_{ec} is obtained in accordance with the following expression:

$$T_{ec} = \left(D + \frac{NA}{BW} - 1\right)t_c. \tag{1}$$

In this expression, D is the network depth, or the number of network stages; NA is the number of requests arriving at the network in one burst; BW is the network bandwidth, or the number of requests accepted by the network without contention; and t_c is the communication delay per network stage, excluding contention. The factor Dt_c accounts for the network latency, while $(NA/BW - 1)t_c$ represents waiting time due to network contention.

If we assume that the same network contention is produced during each iteration, then the total communication time T_c for an algorithm is obtained by multiplying Equation (1) by the number of bursts per iteration and the number of iterations that require communication through the network, in accordance with the following expression:

$$T_c = 6(NCI)(NB)\left(D + \frac{NA}{BW} - 1\right)t_c. \tag{2}$$

In the above expression, NCI is the number of those iterations of the algorithm that require communication through the network, and NB is the number of bursts arriving at the network during an iteration. The whole expression is multiplied by a constant which is in this case equal to 6, since at each iteration two new data items are loaded from a global memory and the results are then stored in a global memory. We assume that for loading data the interconnection network is traversed twice (sending a request and receiving data), while it is traversed only once for storing data in a global memory.

For example, for a single shared communication resource (bus), the bandwidth is $BW = 1$, the depth is $D = 1$, and, hence, the communication time is proportional to the number of requests arriving at the bus in a burst. For a multistage network with depth equal to $\log N$, such as the omega network proposed by Lawrie [15], the maximum bandwidth is $BW = N$, provided that there are no conflicts within network switches and that network stages can be

Table 1 Communication time parameters.

	1D FFT				*2D FFT*	
	Global		*Copy*		*Global*	*Copy*
	Shuffle	*Nonshuffle*	*Shuffle*	*Nonshuffle*		
Communication iterations (NCI)	$\left\lceil \frac{\log M}{\log(M/N)} \right\rceil$	$\log N$	$\left\lceil \frac{\log M}{\log(M/N)} \right\rceil$	$\log N$	$2\frac{\sqrt{M}}{N}$	$2\frac{\sqrt{M}}{N}$
Bursts (NB)	$\frac{M}{N}$	$\frac{M}{N}$	1	1	$\sqrt{M}\log\sqrt{M}$	1
Arrivals in burst (NA)	N	N	M	M	N	$N\sqrt{M}$

Table 2 Best-case performance for the Chunk Allocation.

			Execution time	*Speedup*
1D FFT	*Global*	$N < M$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log N)^2 t_c$	$\frac{N}{1 + 6\frac{(\log N)^2}{\log M}\frac{t_c}{t_p}}$
		$N \geq M$	$(\log M)t_p + 6(\log M)(\log N)t_c$	$\frac{M}{1 + 6(\log N)\frac{t_c}{t_p}}$
	Local	$N < M$	$\frac{M}{N}(\log M)t_p + 6\left(\log N + \frac{M}{N} - 1\right)(\log N)t_c$	$\frac{N}{1 + 6\frac{(\log N)(N\log N + M - N)}{M\log M}\frac{t_c}{t_p}}$
		$N \geq M$	$(\log M)t_p + 6(\log M)(\log N)t_c$	$\frac{M}{1 + 6(\log N)\frac{t_c}{t_p}}$
2D FFT	*Global*	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log M)(\log N)t_c$	$\frac{N}{1 + 6(\log N)\frac{t_c}{t_p}}$
		$N \geq \sqrt{M}$	$\sqrt{M}(\log M)t_p + 6\sqrt{M}(\log M)(\log N)t_c$	$\frac{\sqrt{M}}{1 + 6(\log N)\frac{t_c}{t_p}}$
	Local	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 12\frac{\sqrt{M}}{N}(\log N + \sqrt{M} - 1)t_c$	$\frac{N}{1 + 12\frac{\log N + \sqrt{M} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$
		$N \geq \sqrt{M}$	$\sqrt{M}(\log M)t_p + 12(\log N + \sqrt{M} - 1)t_c$	$\frac{\sqrt{M}}{1 + 12\frac{\log N + \sqrt{M} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$

pipelined. Hence, the minimum communication delay for the omega network is

$$T_c = 6(NCI)(NB)\left(\log N + \frac{NA}{N} - 1\right)t_c. \quad (3)$$

The values for *NB*, *NCI*, and *NA* for different data-access modes and FFT algorithms are derived in the next section and are summarized in **Table 1.**

3. Best-case performance for the Chunk Allocation

We first analyze the case where the whole portion of the data array (*chunk*) referenced by a single processor during an iteration is allocated to a single memory module, so that all chunks can be accessed independently. We name this allocation a *Chunk Allocation.* In this section, we obtain expressions for the best-case communication time for the

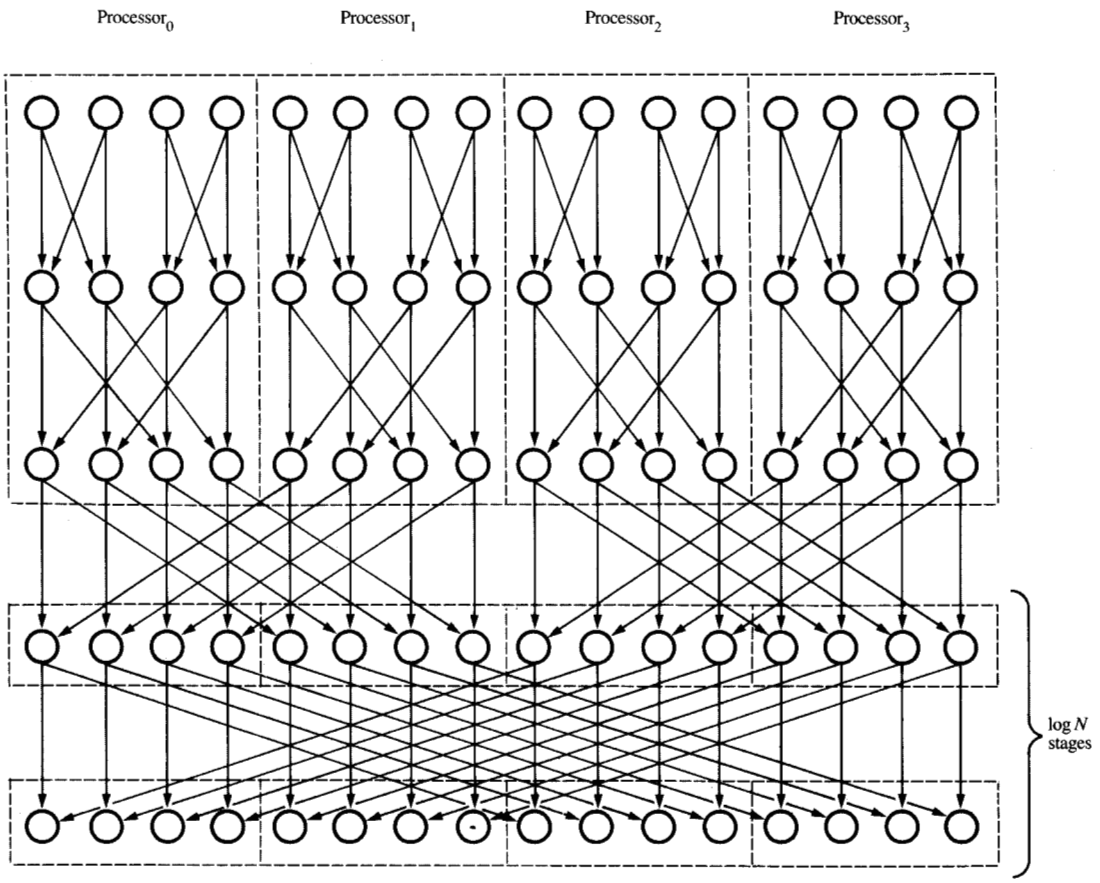

Figure 2

Nonshuffle algorithm for $M = 16$ data items, executed on $N = 4$ processors.

Chunk Allocation by assuming that the chunks are allocated to memory modules so that data can be referenced without generating contention within the network. During each iteration, a processor references data from another chunk, and since whole chunks are allocated to different memory modules, two or more processors cannot reference the same memory module at the same time. By Claim 4 of [16] and its proof, we show that such allocations are feasible and that all iterations of the FFT algorithm produce no contention within the interconnection network, provided that an initial allocation is contention-free.

Expressions for the best-case execution time and speedup for both one-dimensional and two-dimensional FFTs are derived in the sections that follow and are summarized in **Table 2.**

• *One-dimensional FFT*

The one-dimensional FFT algorithm for M data items is represented by a butterfly graph with $\log M$ stages. Each FFT algorithm requires a bit-reversal permutation at some point of the algorithm, so that the results are stored in the same order as the initial data items. In order to simplify the analysis, we neglect the cost associated with the bit-reversal permutation for all FFT algorithms analyzed in this paper. However, the model can be extended to include the bit-reversal permutation, since for a specific allocation of data to memory modules, the network contention and consequently the communication overhead can be computed exactly.

One method for parallelizing the one-dimensional FFT, which is analyzed in this section, is to divide the array of M data items into N chunks, with M/N consecutive items in

each of them. Each processor computes the FFT for the chunk containing M/N data items. An example of such partitioning is shown in **Figure 2,** where the FFT for $M = 16$ data items is executed on $N = 4$ processors. Note that the resulting graph is also a butterfly graph, since the property of the butterfly graph is that it is reducible. We call this algorithm the Nonshuffle Algorithm to distinguish it from the algorithm that applies a shuffle permutation during the iterations requiring communication (Shuffle Algorithm). The latter algorithm is used in Norton and Silberger's model [14] and is analyzed in Section 6.

We assume that the serial time for executing the one-dimensional FFT for M data items is

$$T_s = (M \log M) t_p, \tag{4}$$

where M is the number of data items, $\log M$ is the number of iterations of the algorithm, and t_p is the time to complete the processing of one data item during a single iteration.

The processing time on an N-processor system is obtained as the time on a single processor from Equation (4) divided by the number of processors:

$$T_p = \frac{M \log M}{N} t_p. \tag{5}$$

In order to compute the communication delay, we distinguish two cases according to two different data-access modes.

Case 1 Data items are referenced from global memories.

In this case, all data items are referenced from global memories. Each memory request traverses the interconnection network. We assume that for each data point, two data items are loaded from global memories, then the computation is performed, and then the result is written back to global memories. Processors are synchronized following each memory reference; all of them wait for the last one to complete the computation (barrier synchronization).

The communication time is obtained from Equation (3) by substituting $NB = M/N$ for the number of bursts during an iteration, $NCI = \log N$ for the number of iterations requiring communication, and $NA = N$ for the number of requests arriving at the network during t_c:

$$T_c = 6 \frac{M}{N} (\log N)^2 t_c. \tag{6}$$

The number of iterations requiring communication is equal to $\log N$ and is obtained by subtracting the number of iterations which do not require communication from the total number of iterations: $\log M - \log(M/N)$. Thus, since each processor references M/N data items, the total number of iterations for which a processor needs data from global memory is reduced by $\log(M/N)$, as illustrated in the example from Figure 2.

The total execution time for N processors, T_N, is obtained by summing Equations (5) and (6), since by our assumption the computation and communication time slots are not overlapped:

$$T_N = \frac{M \log M}{N} t_p + 6 \frac{M}{N} (\log N)^2 t_c. \tag{7}$$

The expression for the speedup S is obtained by dividing the serial time from Equation (4) by the execution time on an N-processor system from Equation (7):

$$S = \frac{T_s}{T_N} = \frac{N}{1 + 6 \frac{(\log N)^2}{\log M} \left(\frac{t_c}{t_p}\right)}. \tag{8}$$

The efficiency η is obtained by dividing the speedup from Equation (8) by the number of processors:

$$\eta = \frac{S}{N} = \frac{1}{1 + 6 \frac{(\log N)^2}{\log M} \left(\frac{t_c}{t_p}\right)}. \tag{9}$$

The cost/performance ratio is estimated as the product of the number of processors, which is proportional to the hardware complexity or cost, and the execution time from Equation (7), which is inversely proportional to the performance:

$$cost/performance = NT_N = M(\log M) t_p + 6M(\log N)^2 t_c. \tag{10}$$

If the number of processors N is greater than or equal to the problem size M, the total execution time is obtained from Equation (7) by substituting M for all N except for the N in $\log N$, the number of stages in the interconnection network:

$$T_N = (\log M)(t_p + 6 t_c \log N). \tag{11}$$

The speedup for this case is obtained by dividing Equation (4) by Equation (11):

$$S = \frac{M}{1 + 6 \left(\frac{t_c}{t_p}\right) \log N}. \tag{12}$$

Case 2 Data items are copied to local memories.

In this case, all M/N data items needed by a processor during an iteration are copied into its local memory before the computation is initiated. We assume that data items can be pipelined through the stages of the network. We further assume that no computation is performed until all data items are copied to local memories. After data copying is completed, the computation is initiated and processors access data from their local memories; thus the network is not used during that time. Upon completion of the computation, the resulting data items are copied back to global memories. At this point, processors are synchronized;

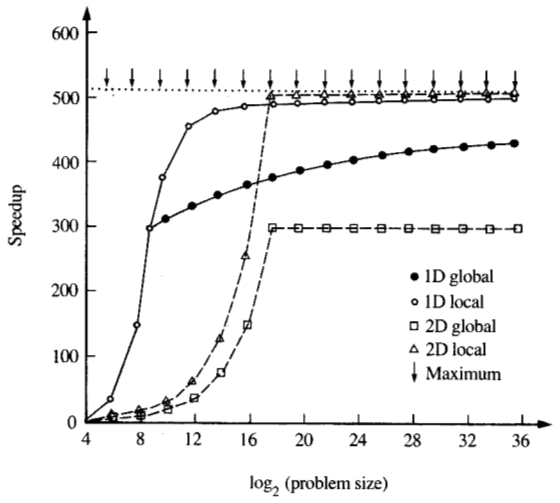

Figure 3

Speedup vs. problem size ($N = 512$, $t_p = 150$, $t_c = 2$).

all of them wait for the last one to complete the computation (barrier synchronization).

The communication time for this case is obtained from Equation (3) by substituting $NB = 1$ (since processors access global memories only once during the iteration), $NCI = \log N$ for the number of iterations requiring communication, and $NA = M$ for the number of requests arriving at the network during t_c:

$$T_c = 6\left(\log N + \frac{M}{N} - 1\right)(\log N)t_c. \tag{13}$$

The speedup is obtained by dividing Equation (4) by the sum of Equations (5) and (13):

$$S = \frac{N}{1 + 6\,\dfrac{(\log N)(N\log N + M - N)}{M\log M}\left(\dfrac{t_c}{t_p}\right)}. \tag{14}$$

If $N \geq M$, the execution time and the speedup are the same as in the case with global data, since each processor accesses only a single data item and pipelining of data through the network stages cannot be applied.

• *Two-dimensional FFT*

The two-dimensional FFT is computed for a $\sqrt{M} \times \sqrt{M}$ data array, so that the one-dimensional FFTs are computed first for the rows and then for the columns of the array. One way to execute the algorithm on N processors is to compute $\sqrt{M}/N$ one-dimensional FFTs for the rows at each processor. The same computation is then repeated for the columns of the array. The expressions for the best-case execution time and speedup for the two-dimensional FFT can be derived using Table 1 and following the same procedure as for the one-dimensional FFT. These results are summarized in Table 2.

• *Discussion of the results*

Figure 3 presents the speedup for the cases analyzed above as a function of the problem size for a fixed number of processors $N = 512$ and $t_p/t_c = 75$. The value for this ratio is chosen so that its complexity matches t_p/t_c estimated from the RP3 prototype.

On the basis of Figure 3, we observe the following:

1. The breakpoint for the one-dimensional FFT is at $M = N$, while the breakpoint for the two-dimensional FFT is at $M = N^2$ for reasons explained below.

   The speedup for the one-dimensional FFT increases with a complexity of $O(M)$ until the problem size becomes equal to the number of processors. Above this point, the computation time complexity increases $O(\log M)$ times faster than the communication time complexity, and the speedup approaches its maximum as M increases. The reason for this change of slope is that below the breakpoint, each processor performs a computation for a single data item at each iteration and is used once for each of $\log M$ iterations, while above this point there are not enough processors available and each processor performs a computation for M/N data items and is used M/N times per iteration.

   For the two-dimensional FFT, the speedup changes its complexity after the problem size exceeds N^2. The reason for this change is that below this breakpoint, each processor computes a single one-dimensional FFT for $\sqrt{M}$ data items, while above this point each processor computes $\sqrt{M}/N$ one-dimensional FFTs.
2. We now relate the performance for the case with global data access to the performance with local/global copying.

   For the one-dimensional FFT, the performance for both data-access modes is the same, provided that the problem size is smaller than the number of processors. This is so in both cases because each processor references a single data item and the time to load data from a global memory is the same in both cases. Above the breakpoint, the communication time for the case with global data access grows with a complexity of $O[M/N(\log N)^2]$, while the communication time for the case with local/global copying grows with a complexity of $O(M/N\log N)$. Therefore, the communication time for the case with local/global copying is $O(\log N)$ times lower than for the case with global data access. The reason for this improvement is that in the case with global data access, all data items are accessed one by one from a global memory, while in the case with local/global copying, portions of the data array consisting of M/N data items

are copied to local memories and data can be pipelined through stages of the interconnection network. With pipelining, each data item takes time of $O(1)$ instead of $O(\log N)$, which is proportional to the number of stages at the network.

For similar reasons, the communication time for the two-dimensional FFT and the case with copying data to local memories is $O(\log M \log N)$ times lower than the communication time for the case with global data access.

The speedup for the two-dimensional FFT with global data access saturates if the problem size exceeds N^2. This is because above this point the communication time has the same growth rate as the computation time. The reason for this is that for each data item, a global memory reference is required, independent of the problem size. This result suggests that for the two-dimensional FFT, copying data to local memories is very beneficial if the problem size exceeds N^2, since with global data access no further performance improvement can be expected as the problem size exceeds N^2.

3. We now compare the performance of one-dimensional and two-dimensional FFTs of the same size.

For the case with global data access and a fixed number of processors, the execution time complexity for the one-dimensional FFT grows as $O(\log M)$, while the execution time complexity for the two-dimensional FFT grows as $O(\sqrt{M}\log M)$ until the problem size becomes equal to the number of processors. This is because for the one-dimensional FFT each processor performs a computation for a single data item, while for the two-dimensional FFT each processor computes a complete one-dimensional FFT for $\sqrt{M}$ data items. If the problem size exceeds N^2, the time complexity for both FFTs increases as $O(M \log M)$, but the communication time complexity for the one-dimensional FFT increases $O[(\log M)/\log N]$ times slower than does that for the two-dimensional FFT. The reason is that for the one-dimensional FFT each processor performs the computation for M/N data items, and the number of iterations is reduced from $O(\log M)$ to $O(\log N)$.

The time complexity for the case with local/global copying exhibits behavior similar to that for the case with global data access.

Figure 4 shows the cost/performance ratio as a function of the number of processors for a fixed problem size $M = 2^{30}$ and $t_p/t_c = 75$. The best-case cost/performance should remain constant as the number of processors increases, since this indicates that the execution time decreases proportionally to the number of processors. If the cost/performance grows, then the decrease in the execution time is smaller than $O(N)$ due to some time wasted on communication or on contention for shared resources.

We can use the cost/performance ratio as a function of the number of processors to determine the optimal value of the number of processors for executing a problem of a given size. For fixed-size problems, the number of processors is also related to the granularity: the larger the number of processors used, the smaller the grain size that is executed on a single processor. By increasing the number of processors, the parallelism available in the application can be exploited better, but at the same time the communication requirements are increased.

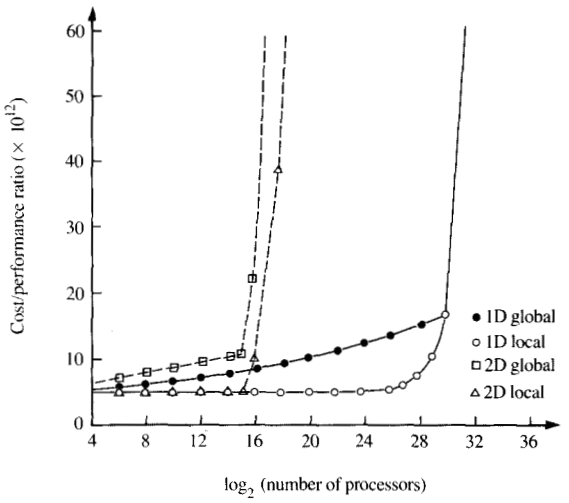

Figure 4

Cost/performance ratio vs. number of processors ($\log_2 M = 30$, $t_p = 150$, $t_c = 2$).

From Figure 4 we note that the cost/performance increases drastically after the number of processors N exceeds the problem size M for the one-dimensional FFT or $\sqrt{M}$ for the two-dimensional FFT. The reason for this change is that above this breakpoint, as the number of processors increases, the execution time grows as $O(\log N)$, so that the cost/performance increases as $O(N \log N)$. Below this breakpoint, the cost/performance for cases with global data access exhibits a small increase as the number of processors increases, due to some time wasted on communication. For the case with global data access, the cost/performance grows as $O(\log N)^2$ for the one-dimensional FFT, while its growth rate is decreased to $O(\log N)$ for the two-dimensional FFT. For the case with local/global copying, the cost/performance remains very close to a constant until N approaches M for the one-dimensional FFT or $\sqrt{M}$ for the two-dimensional FFT, since the communication delay is significantly reduced by taking advantage of pipelining data through the network stages.

Table 3 Worst-case performance for the Chunk Allocation.

			Execution time	*Speedup*
1D FFT	*Global*	$N < M$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log N)(\log N + \sqrt{N} - 1)t_c$	$\frac{N}{1 + 6\frac{(\log N)(\log N + \sqrt{N} - 1)}{\log M}\frac{t_c}{t_p}}$
		$M \le N < M^2$	$(\log M)t_p + 6(\log M)(\log N + \sqrt{N} - 1)t_c$	$\frac{M}{1 + 6(\log N + \sqrt{N} - 1)\frac{t_c}{t_p}}$
		$N \ge M^2$	$(\log M)t_p + 6(\log M)(\log N + M - 1)t_c$	$\frac{M}{1 + 6(\log N + M - 1)\frac{t_c}{t_p}}$
	Local	$N < M$	$\frac{M}{N}(\log M)t_p + 6\left(\log N + \frac{M}{\sqrt{N}} - 1\right)(\log N)t_c$	$\frac{N}{1 + 6\frac{(\log N)(N\log N + \sqrt{N}M - N)}{M\log M}\frac{t_c}{t_p}}$
		$M \le N < M^2$	$(\log M)t_p + 6(\log M)(\log N + \sqrt{N} - 1)t_c$	$\frac{M}{1 + 6(\log N + \sqrt{N} - 1)\frac{t_c}{t_p}}$
		$N \ge M^2$	$(\log M)t_p + 6(\log M)(\log N + M - 1)t_c$	$\frac{M}{1 + 6(\log N + M - 1)\frac{t_c}{t_p}}$
2D FFT	*Global*	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log M)(\log N + \sqrt{N} - 1)t_c$	$\frac{N}{1 + 6(\log N + \sqrt{N} - 1)\frac{t_c}{t_p}}$
		$\sqrt{M} \le N < M$	$\sqrt{M}(\log M)t_p + 6\sqrt{M}(\log M)(\log N + \sqrt{N} - 1)t_c$	$\frac{\sqrt{M}}{1 + 6(\log N + \sqrt{N} - 1)\frac{t_c}{t_p}}$
		$N \ge \sqrt{M}$	$\sqrt{M}(\log M)t_p + 6\sqrt{M}(\log M)(\log N + \sqrt{M} - 1)t_c$	$\frac{\sqrt{M}}{1 + 6(\log N + \sqrt{M} - 1)\frac{t_c}{t_p}}$
	Local	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 12\frac{\sqrt{M}}{N}(\log N + \sqrt{MN} - 1)t_c$	$\frac{N}{1 + 12\frac{\log N + \sqrt{MN} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$
		$\sqrt{M} \le N < M$	$\sqrt{M}(\log M)t_p + 12(\log N + \sqrt{N} - 1)t_c$	$\frac{\sqrt{M}}{1 + 12\frac{\log N + \sqrt{N} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$
		$N > M$	$\sqrt{M}(\log M)t_p + 12(\log N + \sqrt{M} - 1)t_c$	$\frac{\sqrt{M}}{1 + 12\frac{\log N + \sqrt{M} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$

4. Worst-case performance for the Chunk Allocation

The results derived in the previous section represent the best-case execution time, where the chunks of data are allocated to global memories such that the contention within the network can be neglected. In this section, we derive expressions for the worst-case execution time for the Chunk Allocation by assuming that chunks are allocated such that a maximum number of conflicts is produced within the network during each iteration of the algorithm. By Claim 5 of [16] and its proof, we show that such allocations are feasible and that all iterations of the algorithm produce a maximum number of conflicts provided that an initial allocation is conflict-free.

In order to compute the execution time, we need to determine the minimum bandwidth of the interconnection network. The minimum bandwidth is determined by the maximum number of conflicts within the network. Lang [18] has shown that the maximum number of conflicts for the shuffle-exchange network is equal to $2^{\lfloor \log N/2 \rfloor} = O(\sqrt{N})$.

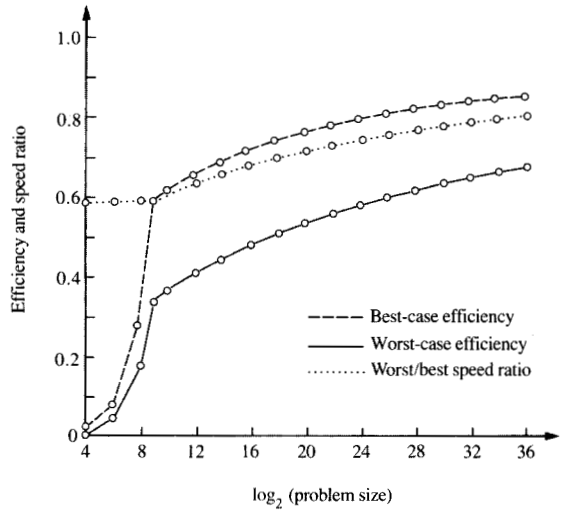

Figure 5

Best and worst efficiency and speed ratio vs. problem size ($N = 512$, $t_p = 150$, $t_c = 2$).

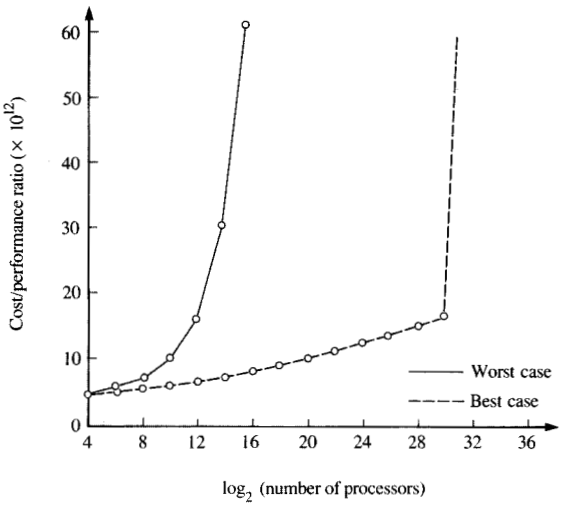

Figure 6

Best and worst cost/performance ratio vs. number of processors ($\log_2 M = 30$, $t_p = 150$, $t_c = 2$).

Since stages of the omega network are interconnected by the shuffle permutation, the same result can be applied to this network, in which the maximum number of conflicts is obtained in the middle network stage. Therefore, the minimum network bandwidth is determined as $BW = N/O(\sqrt{N}) = O(\sqrt{N})$. For the sake of simplicity in notation, we further assume that $N = 2^{2k}$, where k is an integer, and replace $O(\sqrt{N})$ with $\sqrt{N}$.

Using this result, the expression for the maximum execution time is obtained from Equation (2) by substituting $BW = \sqrt{N}$:

$$T_c = 6(NCI)(NB)\left(\log N + \frac{NA}{\sqrt{N}} - 1\right). \quad (15)$$

Expressions for the maximum execution time and minimum speedup for the Chunk Allocation and both one-dimensional and two-dimensional FFTs with two data-access modes are derived following the procedure from the previous section and are summarized in **Table 3**.

• *Discussion of the results*

We now relate the results for best-case and worst-case performance for the Chunk Allocation. We show the results for the one-dimensional FFT with global data access only, since all the other cases exhibit similar behavior.

Figure 5 presents the best and worst cases for efficiency as a function of problem size for a fixed number of processors $N = 512$ and $t_p/t_c = 75$. The dotted curve on the same diagram represents the speed ratio of the best and worst cases.

From this figure, we note that as M increases, the efficiency approaches unity for both cases. This is true since the communication delay is masked by large M with a factor of $O(\log M)$. In both cases, the computation time complexity grows as $O(M \log M)$ while the communication time complexity grows as $O(M)$; however, the worst-case communication time is increased by a factor of $O(\sqrt{N})$ compared to the best case.

The speed of the worst case is equal to 0.59 of the speed of the best case for all $M \leq N$. As M increases above this point, the speed ratio increases very slowly, since the performance for the worst case slowly approaches the best case for large M. Note that as M increases from 2^9 to 2^{36}, the speed of the worst case is increased by only about 20 percent compared to the speed of the best case.

Figure 6 presents the best and worst cases for the cost/performance ratio as a function of the number of processors for the one-dimensional FFT with global data access, a fixed problem size $M = 2^{30}$, and $t_p/t_c = 75$. Note that the bounds diverge as N increases, since the best-case cost/performance increases as $O(\log N)^2$, while the worst-case cost/performance increases as $O(\sqrt{N} \log N)$. The reason for this difference is that due to increased network contention, the network bandwidth for the worst-case Chunk Allocation is reduced from N to $\sqrt{N}$. Figure 6 shows that the worst-case

Table 4 Performance for the Nonchunk Allocation.

			Execution time	*Speedup*
1D FFT	*Global*	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log N)(\log N + N - 1)t_c$	$\frac{N}{1 + 6\frac{\log N}{\log M}(\log N + N - 1)\frac{t_c}{t_p}}$
		$\sqrt{M} \le N < M$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log N)\left(\log N + \frac{M}{N} - 1\right)t_c$	$\frac{N}{1 + 6\frac{\log N}{\log M}\left(\log N + \frac{M}{N} - 1\right)\frac{t_c}{t_p}}$
		$N \ge M$	$(\log M)t_p + 6(\log M)(\log N)t_c$	$\frac{M}{1 + 6(\log N)\frac{t_c}{t_p}}$
	Local	$N < M$	$\frac{M}{N}(\log M)t_p + 6\left(\log N + \frac{M}{N} - 1\right)(\log N)t_c$	$\frac{N}{1 + 6\frac{(\log N)(N\log N + M - N)}{M\log M}\frac{t_c}{t_p}}$
		$N \ge M$	$(\log M)t_p + 6(\log M)(\log N)t_c$	$\frac{M}{1 + 6(\log N)\frac{t_c}{t_p}}$
2D FFT	*Global*	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 6\frac{M}{N}(\log M)(\log N + N - 1)t_c$	$\frac{N}{1 + 6(\log N + N - 1)\frac{t_c}{t_p}}$
		$N \ge \sqrt{M}$	$\sqrt{M}(\log M)t_p + 6\sqrt{M}(\log M)(\log N + \sqrt{M} - 1)t_c$	$\frac{\sqrt{M}}{1 + 6(\log N + \sqrt{M} - 1)\frac{t_c}{t_p}}$
	Local	$N < \sqrt{M}$	$\frac{M}{N}(\log M)t_p + 12\frac{\sqrt{M}}{N}(\log N + \sqrt{M} - 1)t_c$	$\frac{N}{1 + 12\frac{\log N + \sqrt{M} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$
		$N \ge \sqrt{M}$	$\sqrt{M}(\log M)t_p + 12(\log N + \sqrt{M} - 1)t_c$	$\frac{\sqrt{M}}{1 + 12\frac{\log N + \sqrt{M} - 1}{\sqrt{M}\log M}\frac{t_c}{t_p}}$

cost/performance becomes significantly larger than the best-case cost/performance even for relatively small values of N.

5. Performance results for the Nonchunk Allocation

In all the cases analyzed previously, we have assumed that data items from one chunk are allocated to the same memory module (we have called this allocation the *Chunk Allocation*). In this section, we study how performance is affected by applying a different allocation where all data items from a single chunk are allocated to different memory modules (we call this allocation the *Nonchunk Allocation*). We then derive expressions for the communication delay for the Nonchunk Allocation.

While for the Chunk Allocation every processor accesses a different memory module during each iteration of the algorithm, for the Nonchunk Allocation two or more processors may request the same memory module during the same iteration. Expressions for the total execution time and the speedup for the Nonchunk Allocation are summarized in **Table 4.**

• *One-dimensional FFT*

For the one-dimensional FFT, we assume that according to the Nonchunk Allocation, a data item from each chunk up to the chunk size is allocated to a different memory module. Therefore, all processors reference the same memory module at the same time during each iteration.

Case 1 Data items are referenced from a global memory.

In order to compute the execution time for the case with global data access, we need to determine the bandwidth of the network, since all other parameters in Equation (2) are the same as in the previous sections and are summarized in Table 1.

First, we analyze the case where $N < \sqrt{M}$, or the number of processors is smaller than the number of data items in one chunk. In this case, all data items of a chunk are allocated to

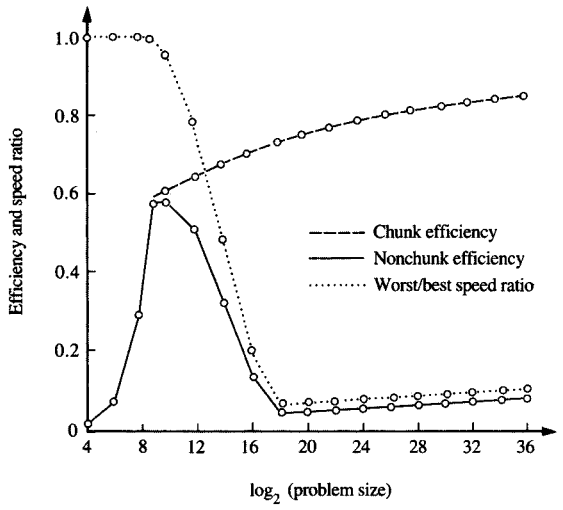

Figure 7

Efficiency for two allocations and speed ratio vs. problem size ($N = 512, t_p = 150, t_c = 2$).

different memory modules. Since all processors reference the same memory module, all N requests are in conflict at this memory and the bandwidth of the network is reduced to 1. The communication delay is obtained from Equation (2) by substituting $NB = M/N$, $NCI = \log N$, $NA = N$, and $BW = 1$:

$$T_c = 6\frac{M}{N}(\log N)(\log N + N - 1)t_c. \tag{16}$$

For the case where $\sqrt{M} \le N < M$, the number of processors is larger than M/N, the number of data items at each memory module. Therefore, the maximum of M/N requests are in conflict, and consequently, the bandwidth is equal to N^2/M. All other parameters are the same, and the communication delay is obtained from Equation (2):

$$T_c = 6\frac{M}{N}(\log N)\left(\log N + \frac{M}{N} - 1\right). \tag{17}$$

Case 2 Data items are copied to local memories.

The case with local/global copying for the Nonchunk Allocation is the same as for the Chunk Allocation, since data are pipelined through stages of the interconnection network. The bandwidth of the network is equal to N since, although N requests are sent to the same memory module, there are still $M - N$ requests for the other modules and the requests for different memory modules can be pipelined through the stages of the network. The expressions for the execution time and the speedup are therefore the same as for the Chunk Allocation.

• *Discussion of the results*

We now compare the best-case results for the Chunk and Nonchunk Allocations for the one-dimensional FFT with global data access.

Figure 7 presents efficiency as a function of problem size for a fixed number of processors $N = 512$ and both allocations. The efficiency exhibits very interesting behavior. If $M \le N$, the efficiency increases as $O(M)$ and is the same for both allocations. For $N < M \le N^2$, the efficiency for the Nonchunk Allocation decreases with a complexity of $O(M/\log M)$, whereas the efficiency for the Chunk Allocation increases with a complexity of $O(\log M)$. The reason for this discrepancy is that for the Nonchunk Allocation the communication time grows faster than the computation time as M increases, since M/N requests are serialized at a single memory module. Finally, for $M \ge N^2$ the efficiency for the Nonchunk Allocation increases, as does that of the Chunk Allocation, with a complexity of $O(\log M)$.

The speed ratio of the Nonchunk and Chunk Allocations is shown by the dotted line in Figure 7. Two allocations have the same speed of execution if $M \le N$. As the problem size exceeds N, the speed of the Nonchunk Allocation becomes much lower than that of the Chunk Allocation, until it approaches the point where it is just 0.062 of the speed of the Chunk Allocation. Above this point, the performance of the Nonchunk Allocation is slightly improved.

Figure 8 plots the speedup as a function of the number of processors N for a fixed problem size $M = 2^{30}$ and all three allocations analyzed above. Note that increasing the number of processors while keeping the problem size fixed is equivalent to increasing the level of granularity or decreasing the size of subproblems executed on processors.

From this figure we note that for the best-case Chunk Allocation, the speedup increases as $O[N/(\log N)^2]$ until the number of processors becomes equal to the problem size. After the number of processors exceeds the problem size, the speedup decreases as $O(\log N)$. The speedup for the worst-case Chunk Allocation, which is obtained by a different allocation of chunks to memory modules, increases with a complexity of $O[\sqrt{N}/(\log N)]$ if $N \le M$; after N exceeds M, it decreases with a complexity of $O(\sqrt{N})$. For example, if $N = M$, the speedup for the best-case Chunk Allocation is 776 times greater than the speedup for the worst-case Chunk Allocation.

The speedup for the best-case Nonchunk Allocation is composed of two curves. If $N \le \sqrt{M}$, the speedup is very low. It first increases and after reaching the maximum it decreases as N approaches $\sqrt{M}$. At this point the speedup from Figure 8 for the Chunk Allocation is 821 times greater than the speedup for the Nonchunk Allocation. If $N \le \sqrt{M}$, the number of processors is smaller than the chunk size and all requests from N processors are serialized at a single memory module. As N exceeds $\sqrt{M}$, the number of

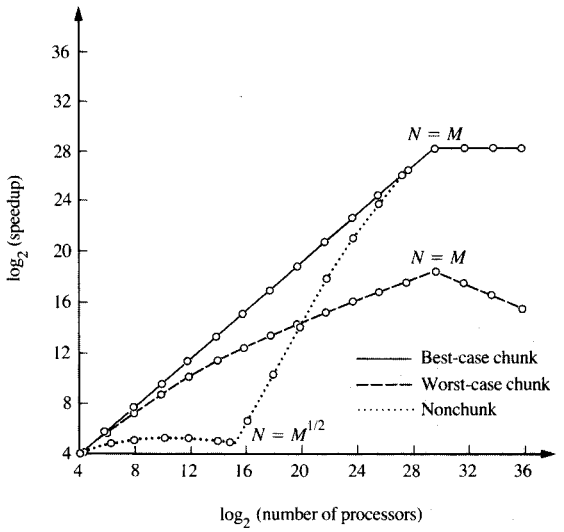

Figure 8

Speedup for three allocations vs. number of processors ($\log_2 M = 30, t_p = 150, t_c = 2$).

processors becomes greater than the chunk size, and the speedup for the best-case Nonchunk Allocation improves, approaching the speedup for the best-case Chunk Allocation. From these results we conclude that the method applied for allocating data items to memory modules has a significant effect on the performance of parallel FFT algorithms.

6. Shuffle Algorithm for the one-dimensional FFT

In this section, we present a different algorithm for executing the one-dimensional FFT on a shared-memory architecture. We use our model to estimate the performance of this algorithm in order to compare our results with the results from the model developed by Norton and Silberger [14]. Since this algorithm performs a shuffle permutation at each iteration requiring communication through the network, we call it the Shuffle Algorithm to distinguish it from the algorithm described in Sections 3 and 4 which we named the Nonshuffle Algorithm. The major difference between these two algorithms is that the Shuffle Algorithm reduces the number of iterations requiring communication from $\log N$ required by the Nonshuffle Algorithm to $\lceil (\log M)/\log(M/N) \rceil$. This can be very beneficial if the problem size is much larger than the number of processors.

Figure 9 presents the Shuffle Algorithm for the example of $M = 16$ data items and $N = 4$ processors. Note that the communication is required after each $\log(M/N)$ iterations of the computation. The data items needed by each processor for the next $\log(M/N)$ iterations are stored in global memories by the inverse shuffle of power of $\log(M/N)$. These data items are then loaded into local memories by each of the processors, so that the next $\log(M/N)$ iterations can proceed without requiring communication through the interconnection network.

As for the Nonchunk Allocation, the maximum number of conflicts in the memory module is obtained as the minimum of the number of processors and the size of the chunk, and is equal to $\min(N, M/N)$. Consequently, the bandwidth is equal to $\max(1, N^2/M)$. The expression for the communication overhead for the case with global data access is obtained from Equation (2) by substituting $NB = M/N$, $NCI = \lceil (\log M)/\log(M/N) \rceil$, and $NA = N$:

$$T_c = 6 \frac{M}{N} \left\lceil \frac{\log M}{\log(M/N)} \right\rceil \left(\log N + \frac{N}{BW} - 1 \right) t_c. \tag{18}$$

The best-case communication overhead is obtained by assuming that no conflicts are generated within the network and replacing $BW = N$ in Equation (18):

$$T_c = 6 \frac{M}{N} \left\lceil \frac{\log M}{\log(M/N)} \right\rceil (\log N) t_c.$$

The worst-case communication overhead is obtained by assuming a maximum number of conflicts at the memory module and substituting $BW = \max(1, N^2/M)$ in Equation (18):

$$T_c = 6 \frac{M}{N} \left\lceil \frac{\log M}{\log(M/N)} \right\rceil \left[\log N + \frac{N}{\max\left(1, \frac{N^2}{M}\right)} - 1 \right] t_c.$$

Expressions for the total execution time and other performance measures can be obtained following the procedure from the previous sections.

Figure 10 presents the speed ratios for two models and the one-dimensional FFT with global data access as a function of the problem size for $N = 512$ and $t_p/t_c = 100$. The speed of our worst-case estimates and the speed computed from Norton and Silberger's model are normalized to the speed of our best-case estimates. From this figure we note that the results from Norton and Silberger's model fall between our best and worst cases and that their complexity matches the complexity of our best-case estimates for large problems. While the results from Norton and Silberger's model approach our best-case estimates, they differ significantly from our worst-case estimates. We believe that the main reason for this discrepancy is that we have assumed that the requests for global memories arrive at the network in bursts and that for a specific allocation of data to memory modules they can decrease the network bandwidth by a factor of $O(N)$. In Norton and Silberger's model, the communications delay is estimated using an average memory-request rate at each iteration. Therefore, in this model the effect of changing

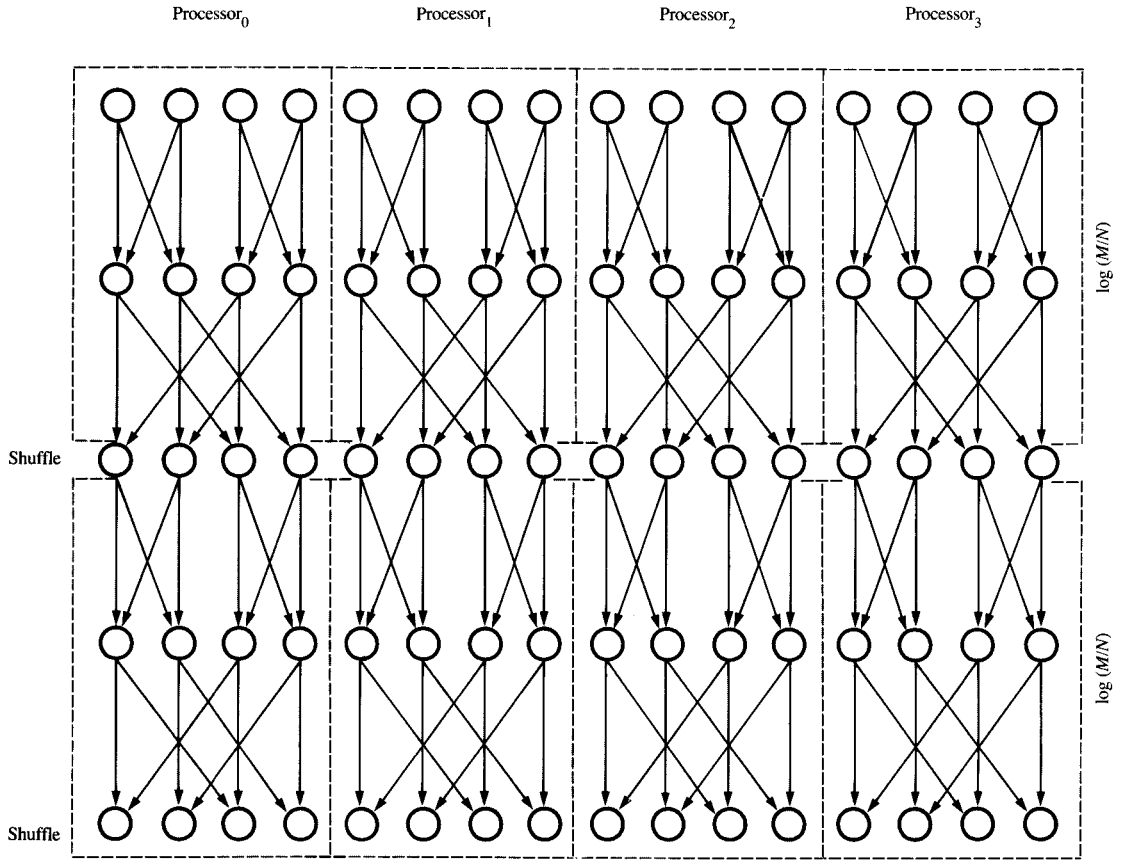

Figure 9

Shuffle algorithm for $M = 16$ data items, executed on $N = 4$ processors.

the allocation of data to memory modules does not have any effect on the performance.

This figure also shows that the best-case performance measures estimated by our model differ from the results obtained from Norton and Silberger's model for smaller problems. A possible reason for this discrepancy is that in estimating the computation time we neglect all factors less than $M \log M$, including both M and $\log M$ factors. For smaller M, these factors tend to influence the computation time more profoundly, and therefore in this region our performance estimates are different from Norton and Silberger's estimates.

7. Conclusions and general discussion

We have presented a deterministic model for predicting the performance of various FFT algorithms executed on a shared-memory parallel computer. By applying a deterministic analysis, we have obtained expressions for performance measures as a function of the problem size, the number of processors, the allocation of data to memory modules, and the speed of processing and communication. We used these expressions to analyze the interaction among various parameters. On the basis of these results, we can recognize performance bottlenecks, and determine how performance can be improved by changes in both the architecture and the application being executed.

The results obtained in this paper were related to the results from the model developed by Norton and Silberger. In contrast to their results, our estimate for the communication delay is very sensitive to the method used for initial allocation of data to memory modules. We have shown that when different allocations are applied, the

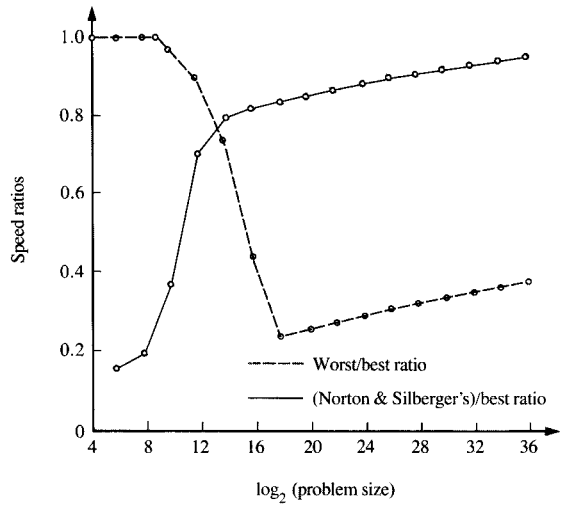

Figure 10

Speed ratios for two models vs. problem size ($N = 512$, $t_p = 200$, $t_c = 2$).

communication delays among the models differ by a factor of $O(N)$, where N is the number of processors.

With respect to different data-access modes, we found that the case with local/global copying performs better than the case with global data access, as long as the problem size is larger than the number of processors, since copying a large block of data can take advantage of pipelining data through the network stages.

The model presented here captures the bursty nature of arrivals at the interconnection network, which is a property of many synchronous algorithms. It provides a means for measuring the performance effect of different allocations of data to memory modules, which is not achievable using a probabilistic analysis. Using the technique demonstrated here, we can further investigate strategies for partitioning large problems in order to minimize the communication through the network. Also, the optimal number of processors for executing a fixed-size problem can be determined by maximizing expressions for the speedup as a function of the number of processors.

Although the scope of our analysis has been limited to FFT algorithms, the methods demonstrated here can be successfully applied to other iterative structured algorithms, as shown in [8]. Various iterative problems proposed for parallel execution can be classified on the basis of a method used for the processing and communication decomposition among N processors, as demonstrated by Vrsalović [19]. Our analysis, then, can be applied to those classes, so that the performance of various classes of problems rather than specific algorithms can be compared, thus enhancing the generality of methods adopted here.

However, since our primary goal was to investigate the general relationship among various parameters that influence the performance of multiple-processor systems, we have simplified the analysis by several assumptions. We now discuss some of the limitations of our approach and the ways of extending the model for application in a more general framework.

1. Owing to length constraints, our analysis in this paper has been limited to shared-memory parallel architecture. In [8(a)], we showed how the methods demonstrated here can be applied to other parallel architectures. Although we have estimated bounds for the communication delay for a specific interconnection network, our model can be generalized to include other interconnection structures. If it is possible to determine the worst and best bandwidth for an interconnection structure as a function of N, then this function can be substituted in our expressions for the performance measures, so that the impact of communication complexity on performance can be directly observed.
2. We have assumed that all processors are synchronized following each iteration of the algorithm and that the computation can be uniformly distributed over N processors, so that all processors send requests to the network simultaneously. If this assumption does not hold, then processors have different initiation times, resulting in an asynchronous execution. It would be interesting to investigate whether the model described here can be extended to include asynchronous parallel algorithms.
3. We have further assumed that iterative algorithms do not include data-dependent branches and that a regular pattern of data accesses is repeated at each iteration. For such algorithms, a static allocation of data to memories prior to execution can be applied. For algorithms with dynamic and irregular data access, the deterministic analysis can be applied only for portions between data-dependent points; otherwise the simulation-based predictions are more appropriate.
4. The analysis performed in this paper does not include the case where computation at processors can be overlapped with communication through the network. If overlap between processing and communication is allowed, then a similar analysis can be performed in order to obtain expressions for the communication delay, as shown in [8].
5. In this paper we have considered only the case where data-access times for both global and local memories are the same. The model can be easily extended to include different data-access times for global and local memories, so that the performance effect of changing memory speeds can be observed.

The results of this paper indicate that for well-structured problems such as the FFT, the congestion at the interconnection network can be reduced significantly by applying a static allocation of data to memories. One interesting problem that remains is to identify allocations that result in the best and worst performance for the FFT. We believe that communication requirements of various parallel algorithms have some common characteristics. If it is possible to classify algorithms according to their communication properties, then it would be worthwhile to investigate the best and worst allocations for each of these groups.

Acknowledgments

The author would like to thank Harold S. Stone for his numerous comments and suggestions during this research, and Alan Norton and Frederica Darema for their valuable interaction. This research was supported in part by the National Science Foundation under Grant No. MCS-7805298 and by the IBM Corporation under Contract No. 462914.

References and note

1. M. C. Pease, "An Adaptation of the Fast Fourier Transform for Parallel Processing," *J. ACM* **15,** 252–264 (April 1968).
2. R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger Ltd., Bristol, England, 1984.
3. H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers* **C-20,** No. 2, 153–161 (February 1971).
4. A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer—Designing a MIMD Shared Memory Parallel Machine," *IEEE Trans. Computers* **C-32,** No. 2, 175–189 (February 1983).
5. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proceedings of the 1985 International Conference on Parallel Processing*, Chicago, IL, IEEE Computer Society, August 1985, pp. 764–772.
6. D. Gajski, D. Kuck, D. Lawrie, and A. Sameh, "CEDAR," *Report No. UIUCDCS-R-1123*, Department of Computer Science, University of Illinois, Urbana, February 1983.
7. R. Thomas, W. Crowther, and R. Gurwitz, "Benchmark Results for a 256-Node Butterfly Parallel Processor," *Report No. 6355*, BBN Laboratories, Cambridge, MA, 1986. Butterfly is a trademark of BBN Advanced Computers, Inc., Cambridge, MA.
8. (a) Ž. Cvetanović, "Performance Analysis of Multiple-Processor Systems," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, May 1986. (b) Ž. Cvetanović, "The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems," *IEEE Trans. Computers* **C-36,** No. 4, 421–432 (April 1987).
9. F. Baskett and A. J. Smith, "Interference in Multiprocessor Computer Systems and Interleaved Memory," *Commun. ACM* **19,** No. 6, 327–334 (June 1976).
10. D. P. Bhandarkar, "Analysis of Memory Interference in Multiprocessors," *IEEE Trans. Computers* **C-24,** No. 9, 897–908 (September 1975).
11. P. Heidelberger, "Queuing Network Models for Parallel Processing with Asynchronous Tasks," *IEEE Trans. Computers* **C-31,** No. 11, 1099–1109 (November 1982).
12. M. Dubois and F. A. Briggs, "Performance of Synchronized Iterative Processes in Multiprocessor Systems," *IEEE Trans. Software Eng.* **SE-8,** No. 4, 419–431 (July 1982).
13. T. N. Mudge and Al-Sadoun, "A Semi-Markov Model for the Performance of Multiple-Bus," *Proceedings of the 1985 International Conference on Parallel Processing*, Chicago, IL, IEEE Computer Society, August 1985, pp. 521–531.
14. V. A. Norton and A. Silberger, "Parallelization and Performance Prediction of the Cooley-Tukey Algorithm for Shared-Memory Architectures," *Research Report RC-11885*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, May 1986.
15. D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers* **C-24,** No. 12, 175–189 (December 1975).
16. Ž. Cvetanović, "Best and Worst Mappings for the Omega Network," *IBM J. Res. Develop.* **31,** No. 4, 452–463 (July 1987, this issue).
17. M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lipovski, "An Overview of the Texas Reconfigurable Array Computer," *AFIPS NCC Conference Proceedings*, Anaheim, CA, May 1980, pp. 631–641.
18. T. Lang, "Interconnection Between Processors and Memory Modules Using the Shuffle-Exchange Network," *IEEE Trans. Computers* **C-25,** No. 5, 496–503 (May 1976).
19. D. Vrsalović, E. F. Gehringer, Z. Z. Segall, and D. P. Siewiorek, "The Influence of Parallel Decomposition Strategies on the Performance of Multiprocessor Systems," *Proceedings of the 12th Annual International Symposium on Computer Architecture*, Boston, MA, IEEE Computer Society and ACM, June 1985, pp. 396–405.

Received April 11, 1986; accepted for publication February 6, 1987

Žarka Cvetanović *Digital Equipment Corporation, 85 Swanson Road, Boxborough, Massachusetts 01719.* Dr. Cvetanović received a Ph.D. in computer and electrical engineering from the University of Massachusetts at Amherst in 1986. She is currently a principal engineer at the Digital Equipment Corporation. This paper was conceived and written during her summer 1985 appointment at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York. Her research interests are in the area of parallel algorithms and performance analysis of multiple-processor systems.