
ELSEVIER

Parallel Computing 27 (2001) 1921–1936

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Applications

Scalable parallel FFT for spectral simulations on a Beowulf cluster

P. Dmitruk [a,*], L.-P. Wang [b], W.H. Matthaeus [a], R. Zhang [c], D. Seckel [a]

[a] *Bartol Research Institute, University of Delaware, 217 Sharp lab, Newark, DE 19716, USA*
[b] *Department of Mechanical Engineering, University of Delaware, Newark, DE 19716, USA*
[c] *EXA Corporation, Lexington, MA, USA*

Received 10 November 2000; received in revised form 5 February 2001; accepted 5 April 2001

Abstract

The implementation and performance of the multidimensional Fast Fourier Transform (FFT) on a distributed memory Beowulf cluster is examined. We focus on the three-dimensional (3D) real transform, an essential computational component of Galerkin and pseudo-spectral codes. The approach studied is a 1D domain decomposition algorithm that relies on communication-intensive transpose operation involving P processors. Communication is based upon the standard portable message passing interface (MPI). We show that $1/P$ scaling for execution time at fixed problem size N^3 (i.e., linear speedup) can be obtained provided that (1) the transpose algorithm is optimized for simultaneous block communication by all processors; and (2) communication is arranged for non-overlapping pairwise communication between processors, thus eliminating blocking when standard fast ethernet interconnects are employed. This method provides the basis for implementation of scalable and efficient spectral method computations of hydrodynamic and magneto-hydrodynamic turbulence on Beowulf clusters assembled from standard commodity components. An example is presented using a 3D passive scalar code.

Keywords: Scalability; Parallel FFT; Spectral simulation; Beowulf clusters; Message passing

* Corresponding author.
E-mail addresses: pablo@bartol.udel.edu (P. Dmitruk), lwang@me.udel.edu (L.-P. Wang), yswhm@bartol.udel.edu (W.H. Matthaeus).

0167-8191/01/$ - see front matter
PII: S0167-8191(01)00120-X

1. Introduction

Spectral and pseudo-spectral method codes have been for many years the mainstay of turbulence research [3,7,14,19,23]. These codes make extensive use of the multidimensional Fast Fourier Transform (FFT) in calculation of nonlinear wavenumber space convolution sums, which are efficiently evaluated using transform methods [11,16]. The FFT also is central in codes based upon related methods involving cosine, Chebyshev and other transforms [6], as well as in image processing and other applications. The performance of these algorithms has depended upon an efficient implementation of the FFT on a progression of computing architectures. The crafting of multidimensional FFTs tuned to particular computing environments began in the early 1970s (using software that made use of special hardware features such as STACKLIB), and advanced through pipeline and vector approached on CYBER and Cray supercomputers [22]. FFTs have also been extensively studied on massively parallel processor (MPP) architectures beginning with SIMD approaches [15]. These approaches later evolved to the asynchronous MIMD parallel approaches that are commonplace today [2]. The latter range from shared memory multiprocessors having backplane-speed interprocessor communications to distributed memory networks of workstations (NOW). In recent years the Beowulf cluster concept has emerged as a promising approach for affordable and practical parallel computing [1]. An ad hoc definition of a Beowulf cluster (see http://www.beowulf.org on the web) is a collection of "commodity" processors, dedicated to parallel processing (unlike NOW) and communicating through the fastest and most efficient interconnects that are affordable. Real interconnect speed, including latency, is important in the distributed memory Beowulf approach, since communication is likely to be a bottleneck in many applications. Fast ethernet (100 Mb/s) is a standard at present, although Gigabit/s ethernet may soon replace it. In its original conception [20] Beowulf involves some efforts to optimize the software layers through which the individual processors access the interconnection network.

It has been recognized for some time [8,9] that parallel implementation would become essential in attaining high Reynolds number turbulence simulations. The basic strategies for parallel spectral methods have been discussed in the literature for some time [15,17]. These codes are typically based upon transform methods [11,16] that involve repetitively transforming between configuration space and wave vector space. On any computer architecture the efficiency of these codes depend upon efficient implementation of multidimensional FFTs, which comprise perhaps 60–80% of the total computational work. Thus there is a clear need to describe the key specific issues for efficient implementation of 3D FFTs on a Beowulf cluster, and this is the main motivation for the present paper.

For the 1D FFT [13] there exist both a binary exchange algorithm, in which the data required for a length N transform are distributed across all P processors, and a transpose algorithm in which the data are manipulated as a matrix of dimension $\sqrt{N} \times \sqrt{N}$. Strategies for the multidimensional FFT [5,10] are in some ways more straightforward, and can be organized according to the degree of locality of the underlying 1D FFT with the so-called transpose method being the most local [5,21].

In the latter approach all FFT operations are carried out locally on individual processors, and can be either 1D or for the 3D FFT depending upon whether the data decomposition is in "pencils" (2D decomposition) or in "slabs" (1D decomposition). For the 3D FFT most relevant to turbulence calculations the transpose algorithm usually admits favorable scaling properties provided that sufficient memory (N^3/P elements per array per processor) is available on each processor [9]. The transpose method also benefits from the fact that all FFT operations are local, and only data communication for the purpose of transposing the data matrix takes place across processors. For the transpose method it is simple to change or adapt the FFT engine that is used locally on each processor. Optimization of the communications strategy becomes an entirely separate issue.

Essentially all previous treatments of parallel multidimensional FFTs recognize the significance of communication strategy as it almost always is a controlling factor in distribution of work among processors [9]. This is due to the technological fact that $t_w > t_c$, where t_w is the interprocessor communication time per word and t_c is the computation time per operation on a processor. For spectral methods (e.g., [17]) as with most algorithms, efficiency (E) is (approximately) a function of the ratio t_w/t_c. Most of the existing literature explores FFT communications in the context of mesh [9], torus [5] or hypercube [13,17,21] interconnection topologies. Presumably this is a consequence of the architecture of massively parallel computers around a decade or so ago (e.g., CM-2, NCUBE/1). There has been also a great increase in the availability of shared memory computers (such as SGI Origin) or distributed memory machine with proprietary fast interconnects (IBM SP). Such fast interconnects tend to minimize t_w as well as latency and blockage (contention) effects. In the case of Beowulf clusters however the interconnects must be thought of somewhat differently. On the one hand, the growing popularity of Beowulf clustering is driven by the rapid increase in both processing and communication speed available in commodity products [4]. It is standard practice at this time to design a Beowulf cluster using switches that provide, in principle, a zero contention path between any pair of processors once the connection is set up. Until the past few years this kind of "crossbar" communication network was thought of as prohibitively expensive and in any case available only up to about $P \approx 25$ [10]. The availability of fast cheap switches, perhaps more than anything else, has made Beowulf clustering a serious option for supercomputing applications. On the other hand we must also bear in mind that Beowulf components may bring hidden limitations that require attention be paid to design of both the hardware and algorithms.

In the following sections we discuss optimization of multidimensional FFT on a Beowulf cluster for use in spectral simulations of fluids. The main focus is on optimization of the transpose method with respect to the general communication strategy. This is the topic of Section 2. Optimization of the specific implementation on a switched fast ethernet interconnection network is discussed in Section 3. Benchmarks are given in Section 4, where the speedup, total compute time and efficiency are discussed. Finally in Section 5 we summarize our results and present sample timings from 3D pseudo-spectral hydrodynamic turbulence code with passive scalar.

2. Parallel transpose method

The discrete multidimensional Fourier Transform can be written as

$$H(n_1,\ldots,n_L)=\sum_{k_L=0}^{N_L-1}\cdots\sum_{k_1=0}^{N_1-1}\exp(2\pi i k_L n_L/N_L)\cdots\exp(2\pi i k_1 n_1/N_1)h(k_1,\ldots,k_L). \tag{1}$$

An L-dimensional FFT can be computed by taking a sequence of nested 1D FFTs. Therefore, a 3D FFT calculation can be accomplished using either three 1D FFTs or one 2D FFT plus one 1D FFT.

On parallel computers with a distributed memory option, a simple parallel strategy is to slice the computational domain in one direction, e.g., z direction (see Fig. 1). The FFT operation in the sliced direction is relatively inefficient because the information is distributed on all nodes. Thus a special transpose is introduced. After doing a 2D FFT for each plane in each node, the y and z directions are switched and the data in z direction are converted into y direction so that each node has the complete information needed:

$$h'(k1,z,k2)=h(k1,k2,z). \tag{2}$$

Then after doing a 1D FFT in the new second direction and doing a transpose to interchange y and z again, the 3D FFT is complete. The last transpose can be avoided in codes that can make use of the FFT results in no particular order, for instance, when doing convolution products in pseudo-spectral codes. This is the case

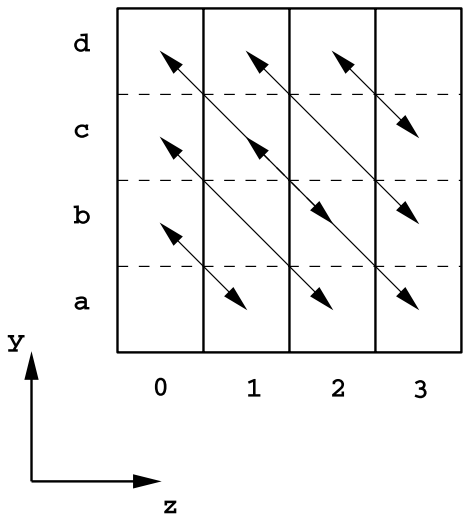

Fig. 1. The illustration of the block transpose algorithm. The computational domain in this example is sliced into four pieces in z direction. The data are exchanged between corresponding blocks in a parallel manner.

we adopted here, when measuring times in benchmarks. The FFT is called to be in "transposed" order.

Since the communication between nodes is the most time-consuming part of this parallel computation, a fully parallel communications algorithm is necessary for the first transpose. On a switched Beowulf cluster maximum efficiency demands that all the communications channels be kept full. To accomplish this we adopt an algorithm in which the transpose is accomplished in two stages. First, blocks of data are transposed between processors, storing the result in a temporary array. Then the transpose is completed by locally rearranging each transmitted block as it is copied from the temporary array to the main storage array. Fig. 1 demonstrates the algorithm based on four nodes. Since the diagonal blocks $a0$, $b1$, $c2$, $d3$ only need to convert data inside each node, so a total of $n-1$ times are needed to convert these blocks by message passing for each node (total $(n^2-2n)/2$). In the first step, the data move in blocks from $b0$ to $a1$, $c1$ to $b2$, $d2$ to $c3$, and $a3$ to $d0$. Secondly, the data move from $c0$ to $a2$, $d1$ to $b3$, $a2$ to $c0$, and $b3$ to $d1$. Finally, the data move from $d0$ to $a3$, $a1$ to $b0$, $b2$ to $c1$, and $c3$ to $d2$. After accomplishing the block transform across nodes, the transpose is completed by converting data inside each block.

This strategy for the parallel FFT does not take into account any specific optimization that may be done regarding the local FFT calculation on each node. This is in principle an advantage to this approach, as it adapts quite easily to whatever increased efficiency can be gained by improvement of the local FFT calculation. In the usual way we can develop an approximate quantitative model for execution time (or complexity) of the message passing parallel 3D FFT on a square grid of size N^3 using P processors. The total time per FFT per node is

$$T = T_{\mathrm{COMP}} + T_{\mathrm{COMM}}, \tag{3}$$

where the first term represents computation time per node, and the second represents communication time per channel including latency.

The computation time per node is the sum of the contribution from calculation of the local FFTs and the contribution from rearrangement of the local temporary array. Based upon standard operation counts for the FFT [18] and temporal array construction needed for the transposing routine, we may estimate

$$T_{\mathrm{COMP}} = \frac{5}{2}\frac{N^3\log_2(N^3)}{P}t_{\mathrm{c}} + \left(\frac{2}{P} + \frac{P-1}{P^2}\right)(N+2)N^2 t_{\mathrm{a}}, \tag{4}$$

where t_{c} is the single processor FFT computation time per word and t_{a} is the time for memory-to-memory copy of a word.

The size of a block that a processor has to send to (or receive from) another is $(N+2)N^2/P$ words (the $N+2$ factor, instead of N, comes from the fact that in a complex-to-real FFTs two extra locations are required in real arrays). Each processor needs to send and receive a block of this size from the other $P-1$ processors, so the total communication time per processor may be estimated as

$$T_{\mathrm{COMM}} = 2(P-1)(N+2)\left(\frac{N}{P}\right)^2 t_{\mathrm{w}} + 2(P-1)t_{\mathrm{s}}, \tag{5}$$

where t_w is time for transmission of a single word between nodes and t_s is the startup or latency time for a message. For large N and P the per processor time becomes

$$T \to t_c \frac{5}{2} \frac{N^3 \log_2(N^3)}{P} + t_a 3 \frac{N^3}{P} + t_w 2 \frac{N^3}{P} + t_s 2P. \tag{6}$$

A model for time estimation like this one can be very useful in performance analysis and projections for different size N, number of processors P and hardware or network parameters like t_c, t_w and t_s.

We will turn now to characterization of the performance of the block transpose 3D FFT on a small Beowulf cluster. An additional important feature pertaining to the sequence of message passing calls will be discussed in the context of the "experimental" evaluation.

3. Evaluation and testing

Efficient and scalable performance of the block transpose method can be readily demonstrated even on a small Beowulf cluster. Here we employ a 16-node dedicated cluster, each node consisting of a 400 MHz Celeron processor with 128 MB local random access memory (RAM). The nodes are linked by standard fast ethernet (100 Mb/s nominal speed) and a 16-port switch. The Linux operating system is employed, using the GNU Fortran compiler and the portable MPI message passing system ([12] and http://www-unix.mcs.anl.gov/mpi/mpich). For the local FFT in each processor we used the public available FFTW (http://www.fftw.org) by Frigo and Johnson.

In discussing the performance of the FFT algorithm it will be convenient to describe performance in terms of the formal estimates given above in Eqs. (4) and (5). For a particular configuration this requires that we know t_c, t_a, t_s, and t_w. These can be obtained empirically: $t_c \approx 6.5$ ns is determined running FFT in a single processor, $1/t_c \approx 153 \times 10^6$ corresponds to the FFTW algorithm floating point operations per second (a similar value to those reported for the benchmark version "benchFFT 2.0" on PCs machines in the FFTW page [1] http://www.fftw.org/benchfft/results/bench_-toc.html). This number is related not only to pure arithmetic operations but also to memory copy internal operations in the FFT algorithm, and so it is only a fraction of the peak Mflops of the CPU, in this case 400 Mflops for a Celeron 400 Mhz processor. $t_a \approx 70$ ns is determined running standard loops (for the memory-to-memory copy) on a single processor. These values correspond to the intermediate size $N = 128$; slightly different values can be obtained for different N, due to details on how the arrays for computations are handled both at a software level (code and compiler) and in the local memory management (cache, main memory). In this sense,

[1] If we run the benchmark version "benchFFT 2.0", we find rates over 200 Mflops for 1D and 2D Real $\to$ Complex. The benchmark optimizes setup and sequencing of FFT executions to minimize the effect of memory access. Our "real world code" does not enjoy the same memory access optimization; however, the reported 153 Mflops is the relevant result for studying scaling of our parallel algorithm.

the computational time model of Eq. (4) has to be taken as the first approximation, since t_c, t_a are not strictly independent of N or P.

The communications parameters can be found by performing simple numerical experiments on the cluster, as illustrated for our cluster in Fig. 2. Setting up a simple exchange of a data array between two nodes, we can vary the data length, performing each transfer many times for accuracy. For large message sizes, we find a near linear relationship of time to message length. The slope 85 μs/Kbyte of this line corresponds to a time per word (where 1 word = 4 Bytes = 32 bit in single precision computation) $t_w \approx 0.34$ μs/word. This also corresponds to a bandwidth of 94 Mb/s on our commodity fast ethernet interconnects. This estimate is obtained for two processors alone with the others silent, so it does not include blocking effects that may be associated, for example, with the architecture of the switch or of the ethernet interfaces. Below we will consider these effects, which have non-trivial consequences. Also from Fig. 2 we can estimate the y-intercept, which gives the latency or startup time $t_s \approx 100$ μs.

The performance of the 3D FFT under optimal conditions can be compared for varying problem sizes, and this is illustrated in Fig. 3. Also shown is a comparison with the performance estimates. Test cases are shown for three problem sizes, $N = 64$, 128 and 256. The processor number is varied from 1 to 16 for these tests. A fully switched network with optimal message passing strategy (discussed further below) is employed. The 3D FFT time is shown as a total amount and broken down into computational and communications timings. It can be seen from the figure that the scaling with P works as expected especially at the larger processor numbers. This provides confidence that the naive timing estimates are not far off, and therefore quite useful in planning and managing runs done on the cluster.

The efficiency is the ratio of total time on one processor to P times the P processor total time, $E = T_1/PT_P$, for a fixed size problem. There is always an ambiguity in the definition of the "time on one processor" for the computation of the efficiency of a

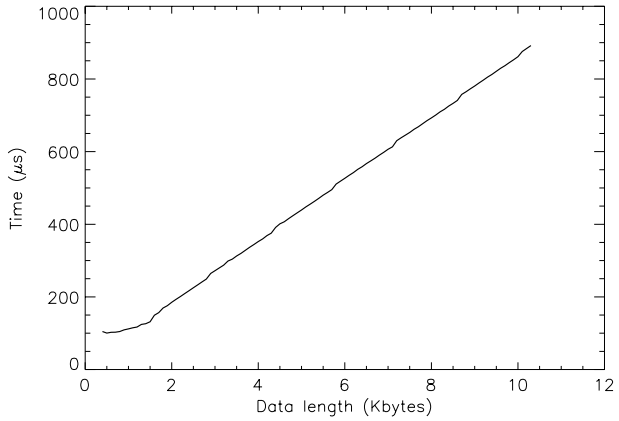

Fig. 2. Communication time between two nodes as a function of the length of the data transmitted. The slope of this line gives t_w, the communication time per word. The y-intercept gives the latency time.

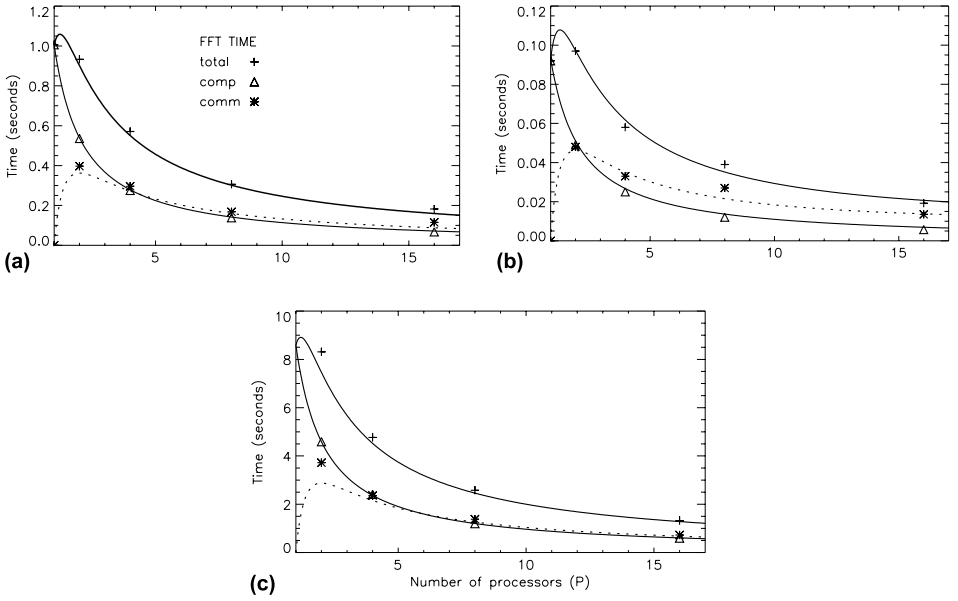

Fig. 3. FFT times (total, computation and communication) as a function of the number of processors P for sizes: (a) 128^3, (b) 64^3 and (c) 256^3. The points (plus signs, triangles and stars) correspond to measured times, the lines are the theoretical estimates from Eqs. (4) and (5).

parallel algorithm: one could use the parallel algorithm for the case $P = 1$ or instead use a different sequential, perhaps optimal, algorithm for a single processor. For instance, in this particular case, the 3D FFT is calculated in two steps, first doing 2D FFT for each plane in the box, and then doing 1D FFT along the remaining direction (see Section 2). This is to allow local computation of each step which is done in parallel by all the processors. For $P = 1$, this way of splitting the problem involves some local rearrangement of the arrays, and this is reflected in the extra term in Eq. (4) which is non-zero for $P = 1$. If instead, an algorithm in which we just call a 3D FFT is used, then, a more efficient result can be expected, although, internally in the FFT routine some rearrangement of the arrays has to be done anyway. The result would then be that the time on one processor T_1 would be lower, and so, the absolute efficiency would decrease. For this particular case, we measured the difference between the time of the parallel algorithm with $P = 1$ and the time of a sequential algorithm with a single 3D FFT call and obtain about a 15% lower value for this last case and so this gives a corresponding decrease in the absolute efficiency. For instance, with case $N = 128$ and $P = 16$ the efficiency decreases from 0.35 to 0.30. Aside from this ambiguity, and the relatively small differences in the results between definitions in this particular case, what is more relevant for the performance of a parallel algorithm is to maintain the efficiency at a constant level, which assures the so-called scalability of the problem. We adopted for the definition of efficiency to use the parallel algorithm with $P = 1$ for the time in one processor.

We show the efficiency as a function of P for the FFT problem of sizes $N = 64$, 128 and 256, in Fig. 4. It is apparent that E is maintained at reasonable levels for the parameters considered. There is also a suggestion that efficiency is better for larger problem size.

This is related to the estimate of speedup $S = T_1/T_P = PE$ that can be derived from the performance expressions Eqs. (4) and (5) as

$$S = \frac{P}{1 + \frac{(P-1)}{\left[\frac{5}{2}\log_2(N^3)t_c + 2t_a\right]P}\left(t_a + 2t_w + \frac{2P^2}{N^3}t_s\right)}. \tag{7}$$

Extrapolating this expression to larger number of processors, we illustrate, in Fig. 5 the projection of 3D FFT performance out to 128 processors. It can be clearly seen that the performance of the 64^3 problem degrades beyond about 30 processors, an effect attributable according to Eq. (7) to the latency time. The same effect can be seen for 128^3 problem size, but the good performance prevails up to $P = 100$ in this case. For the largest problem size the speedup remains nearly linear indicating high efficiency and overall good performance. This effect is also manifest in the calculated Mflops rate (million floating point operations per second, calculated as $5/2N^3\log_2(N^3)/T_P$) achieved by the test runs. These are shown in Fig. 6. The 3D FFT code achieves sustained rates of up to 800 Mflops for the 256^3 problem on 16 processors, but only 600 Mflops for the 64^3 problem on the same number of processors. However it is clear that for any fixed problem size the Mflops will reach some maximum as P is increased, again because of latency. Using 400 Mflops as the peak rate in a single processor (Celeron 400 Mhz) this Beowulf cluster of 16 processors is getting about 10% of the peak, for this particular problem.

Eq. (7) is also useful to estimate the effect on performance when improving single processor CPU speed (which decrease t_c) and/or increasing network bandwidth (for

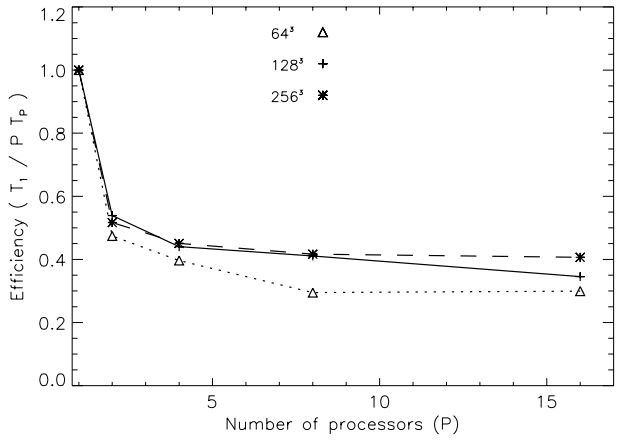

Fig. 4. FFT efficiency as a function of the number of processors P for sizes 64^3, 128^3 and 256^3. The lines here are only to connect measured values.

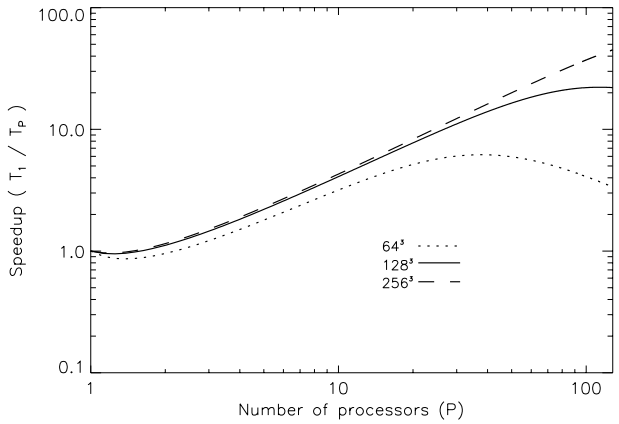

Fig. 5. FFT speedup projections based on the estimates of Eqs. (4) and (5) extrapolated up to 128 processors. Problem sizes of 64^3, 128^3 and 256^3 are shown.

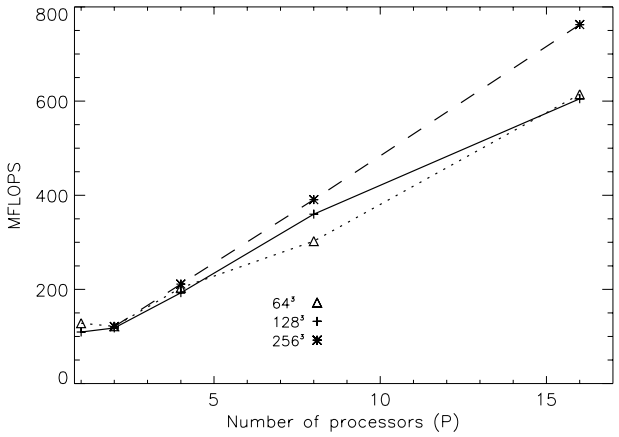

Fig. 6. Measured FFT million floating point operations per second (MFLOPS) as a function of the number of processors P for sizes 64^3, 128^3 and 256^3. The lines here are only to connect measured values.

instance, a gigabit network would decrease t_w by approximately a factor 10 with respect to the fast ethernet network). Decisions for improvement in the hardware, at least for a project involving these kind of problems, can be made taking into account these effects in performance.

As a final issue of importance to attaining optimal communication efficiency, we examine briefly two features of our interconnections – firstly the switch configuration and secondly, the communication pairings. The first of these is rather more obvious. The cluster is expected to function most efficiently when it is fully switched. In this case there is no blocking of messages due to contention of traffic on the network. This is the idealized case represented above in our performance studies. Indeed we

have seen above that for fully switched interconnections, ideal performance is very nearly achieved. However it is worthwhile to examine the degradation of the interconnect performance as the level of switching is decreased. This can be achieved experimentally by reconfiguring the cluster's private network, employing hubs instead of switches for each set of several nodes. The hubs cause the nodes to share bandwidth, increasing the frequency of network contention and the possibility of message packet collisions. Results of such a test are shown in Fig. 7. For an $N = 128^3$ test the results for the fully switched configuration are compared with results for interconnect setups with two nodes per hub, and with four nodes per hub. The results are shown for varying P. The more nodes per hub the greater the network contention. It is clear that the switched interconnect is the best performing and that network contention ruins the theoretical scaling, which agreed well with the switched interconnect case.

The final remark we wish to make about communications strategy is somewhat more subtle. In our first attempts at verification of the theoretical performance scalings, we employed the following communication strategy: a sequential list of processors is established. Each processor first passes data to the processor below it on the list, while receiving data from a processor just above it on the list. Next, processors send data to the second element down the list, and so on, wrapping the end of the list back to the top. One might expect this "cyclic" strategy to be optimal, but it is not. Comparisons with the theoretical scaling in this case imply a blocking factor $B > 1$ such that $t_c \to Bt_c$.

It turns out that a superior approach is to enforce pairwise communication, so that, each node is sending to the same node from which it is receiving. Suitable algorithms can be constructed to generate a list of such pairings so that each node forms a temporary pairing with every other one, keeping the full network bandwidth

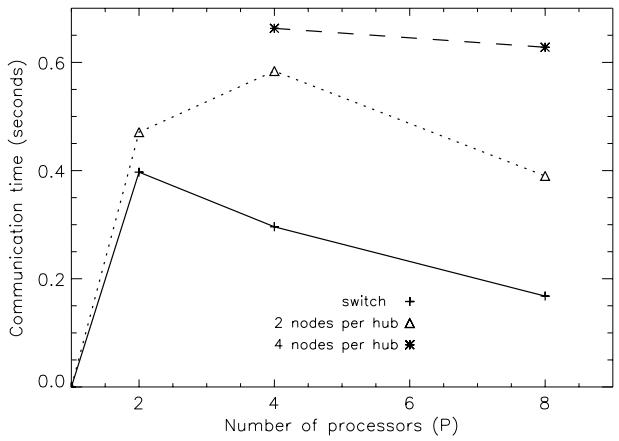

Fig. 7. Measured FFT communication times as a function of the number of processors at problem size 128^3 for different network configurations. Plus signs correspond to a fully switched interconnection, triangles and stars correspond to configurations with hubs that cause nodes to share bandwidth.

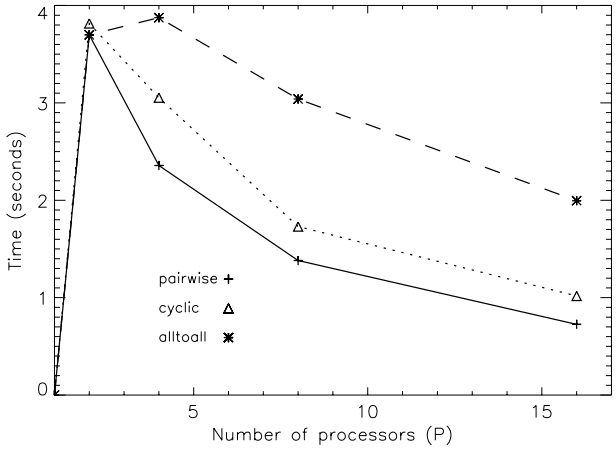

Fig. 8. Measured FFT communication times as a function of the number of processors at problem size 256^3 for different communication strategies. Plus signs correspond to a pairwise scheduling, triangles correspond to a cyclic scheme and stars are the communication times when using the MPI all-to-all directive.

occupied. An example of this would be an algorithm that generates "round robin" tournament pairings in a single elimination tournament. This round robin schedule can be established initially, keeping a list of "contenders" for each processor in a list array ready for use in the communication part of the FFT. We carried out a comparison of the cyclic and pairwise communication strategies, the results of which are portrayed in Fig. 8. Shown are communications time for $N = 256$ 3D FFT test runs for these two cases. A third set of runs is also shown that employs the MPI "all-to-all" operator [12] for the communications step. It is evident that the pairwise method gives the best performance. For smaller problem sizes ($N = 64$, 128) these differences still exist at intermediate number of processors ($P = 4$, 8), but tend to diminish as the number of processors increases, which indicates that differences in communication strategies tend to be less important when message packets are small.

4. Hydrodynamic passive scalar code

To demonstrate performance of an application that relies heavily upon the 3D FFT we present performance data for a hydrodynamic passive scalar code. The code addresses the standard homogeneous turbulence problem – solution of the 3D incompressible Navier–Stokes equations in cartesian periodic geometry. In addition to the incompressible Navier–Stokes equations, a passive scalar field is solved according to

$$\frac{\partial \psi}{\partial t} + \nabla(v\psi) = \kappa \nabla^2 \psi, \tag{8}$$

where κ is the scalar diffusivity. See, for example, [24] for additional details. The numerical method is a pseudo-spectral (or collocation) scheme, employing a second-order time integration. In the native Fourier representation employed by the code, spatial derivatives are evaluated algebraically by multiplication by factors involving the wave vector k. This property ensures both accuracy and efficiency. Advancing the velocity and scalar fields at each time step involves the computation of nonlinear terms such as $Z \equiv \nabla(v\psi)$ that appears in Eq. (8). Such terms are convolutions in the k-space. In the pseudo-spectral method they are evaluated efficiently by computing the product in the x-space but then performing the spatial derivatives in the k-space. Thus to evaluate Z, one transforms $\psi(k) \to \psi(x)$ and $v(k) \to v(x)$. The second transform is a part of the hydrodynamics simulation. The product $v(x)\psi(x)$ is computed in the x-space and then transformed back to the k-space. The divergence operation is performed in the k-space which completes the evaluation of $Z(k)$. The specific number of required 3D transforms per time step depends upon the particular way the equations are written. Following the method of [24], the number of 3D FFTs per time step is 13 (nine for the solution of the Navier–Stokes equations and four for the scalar field). The computational efficiency of the 3D FFT is crucial for the performance of the code.

Here we illustrate the code's performance using the block transpose FFT optimized for the current Beowulf cluster as we described above. Several runs were done, at resolutions varying from 64^3, 128^3 and 256^3. We ran the code on $P = 1, 2, 4, 8$ and 16 processors on the Beowulf cluster. Fig. 9 shows the code performance in terms of total time for 64^3 size (the only one for which runs with $P = 1$ can be done, due to single processor memory limitations). One can see that the scaling properties are very similar to those of the FFT itself. This method enables codes of this type, usually run on supercomputers and expensive shared memory computers, to be effectively run on small commodity class Beowulf clusters.

Further test data performed on a larger Beowulf cluster at ICASE of NASA Langley are shown in Fig. 10. Here each single CPU node is a Pentium II 400 MHz processor and has 384 MB of RAM, and therefore 128^3 simulation can also be conducted on a single node. As many as 32 CPU nodes with fast ethernet network

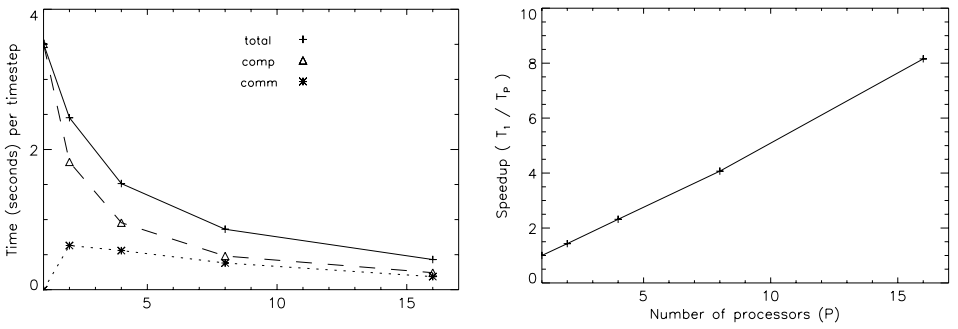

Fig. 9. Measured time per time step and speedup as a function of the number of processors for a hydrodynamic 3D pseudo-spectral simulation.

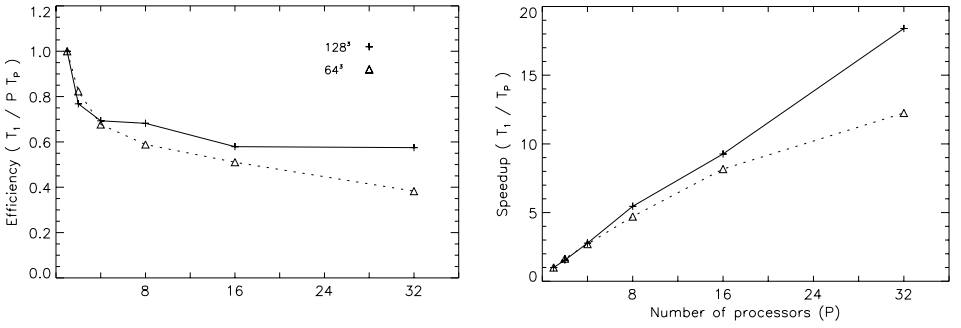

Fig. 10. Efficiency and speedup as a function of the number of processors for a hydrodynamic 3D pseudo-spectral simulation performed on a larger Beowulf cluster.

and switch were used. The overall computation efficiency increases with problem size (N) and decreases with P, as expected. An efficiency of 0.57 is achieved on 32 nodes for 128^3 simulation. A better efficiency is achievable for higher mesh resolutions.

For a problem requiring extensive non-local communications, this level of scalability is quite satisfactory and has not been demonstrated for Beowulf clusters.

Speedup for this test run is also shown in Fig. 10, indicating good performance for the 128^3 case up to the highest number of processors and some degradation for the 64^3 around 32 processors, due to latency effects as we mentioned in the FFT tests.

5. Conclusions

We have described a parallel FFT algorithm based on a block transpose approach using MPI on a distributed-memory multiple-computer system interconnected by fast ethernet. Simple tests quantitatively measured scalability of the computation time and communication time. We studied several questions related to the communication load: how does the communication time depend on the system and problem parameters – communication bandwidth, number of nodes, and problem size? How does the communication time depend on the communication scheme (non-overlapping pairing versus cyclic scheme)? What is the effect of communication hardware configuration? We also developed a model for scheduling non-overlapping pairings for the optimum communication mode. Finally we integrated the MPI FFT scheme into a spectral code and demonstrated its performance.

Our main conclusions are as follows. (1) The communication time and computation time can be described analytically. (2) For a given problem size, CPU speed, and communication bandwidth, there exists an optimum node number for the overall execution time (a single FFT or the simulation of full passive scalar turbulence). A model for this has been developed. This allows for projection of performance on future distributed memory multicomputer Beowulf systems. (3) Communication hardware configuration can substantially affect the performance.

Acknowledgements

This research was supported in part by the NSF Major Research Infrastructure program under grants PHYS-9601834, and ATM-9977692, and by NASA under grant NAG5-7164. Part of the test data were obtained using the computing facility at ICASE, NASA Langley. L.P. Wang thanks Dr. L.-S. Luo for arranging his visit to ICASE.

References

[1] M. Atiquzzaman, P.K. Srimani, Parallel computing on clusters of workstations (guest editorial), Parallel Comput. 26 (2000) 175–177.
[2] A. Averbuch, E. Gabber, Portable parallel FFT for MIMD multiprocessors, Concurrency: Pract. Exper. 10 (1998) 583–605.
[3] M.E. Brachet, M. Meneguzzi, H. Politano, P. Sulem, The dynamics of freely decaying two-dimensional turbulence, J. Fluid Mech. 194 (1988) 333–349.
[4] R. Brightwell, L.A. Fisk, D.S. Greenberg, T. Hudson, M. Levenhagen, A.B. Maccabe, R. Riesen, Massively parallel computing using commodity components, Parallel Comput. 26 (2000) 243–266.
[5] C. Calvin, Implementation of parallel FFT algorithms on distributed memory machines with a minimum overhead of communication, Parallel Computing 22 (1996) 1255–1279.
[6] C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, Spectral Methods in Fluid Dynamics, Springer, New York, 1988.
[7] S. Chen, X. Shan, High resolution turbulence simulations using the connection machine-2, Comput. Phys. 6 (1992) 643–646.
[8] A.T. Degani, G.C. Fox, Parallel multigrid computation of the unsteady incompressible Navier–Stokes equations, J. Comp. Phys. 128 (1996) 223–236.
[9] P.F. Fischer, A.T. Patera, Parallel simulation of viscous incompressible flows, Ann. Rev. Fluid Mech. 26 (1994) 483–527.
[10] I.T. Foster, Designing and Building Parallel Programs, Addison-Wesley, Reading, MA, 1995.
[11] D. Gottlieb, S.A. Orszag, Numerical Analysis of Spectral Methods: Theory and Application, SIAM, Philadelphia, 1977.
[12] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT Press, Cambridge, MA, 1995.
[13] A. Gupta, V. Kumar, The scalability of FFT on parallel computers, IEEE Trans. Parallel Distrib. Syst. 4 (1993) 922–932.
[14] W.H. Matthaeus, W.T. Stribling, D. Martinez, S. Oughton, D. Montgomery, Phys. Rev. Lett. 66 (1991) 2731–2734.
[15] P. Moin, J. Kim, Numerical investigation of turbulent channel flow, J. Fluid Mech. 118 (1982) 341–377.
[16] G.S. Patterson, S.A. Orszag, Spectral calculations of isotropic turbulence: efficient removal of aliasing interactions, Phys. Fluids 14 (1971) 2538–2541.
[17] R.B. Pelz, The parallel Fourier pseudospectral method, J. Comp. Phys. 92 (1991) 296–312.
[18] W. Press, B. Flannery, S. Teukolsky, W. Vetterling, Numerical Recipes, The Art of Scientific Computing, Cambridge University Press, Cambridge, UK, 1986.
[19] R.S. Rogallo, P. Moin, Numerical simulation of turbulent flows, Ann. Rev. Fluid Mech. 16 (1984) 99–137.
[20] T.L. Sterling, J. Salmon, D.J. Becker, D.F. Savarese, How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters, MIT Press, Cambridge, MA, 1999.
[21] P.N. Swarztrauber, Multiprocessor FFTs, Parallel Comput. 5 (1987) 199–209.
[22] C. Temperton, Self-sorting mixed radix fast Fourier transforms, J. Comp. Phys. 52 (1983) 1–23.

[23] L.-P. Wang, S. Chen, J.G. Brasseur, J.C. Wyngaard, Examination of hypotheses in the Kolmogorov refined turbulence theory through high-resolution simulations. Part 1. Velocity field, J. Fluid Mech. 309 (1996) 113–156.

[24] L.-P. Wang, S. Chen, J.G. Brasseur, Examination of hypotheses in the Kolmogorov refined turbulence theory through high-resolution simulations. Part 2. Passive scalar field, J. Fluid Mech. 400 (1999) 163–197.