COMMUNICATION PERFORMANCE OF PARALLEL 3D FFTS USING VARIOUS NETWORKS AND TRANSPOSITION ALGORITHMS

A. Adelmann, A. Bonelli[1] *W. P. Petersen*[2]*, C. W. Überhuber*[1]

Vienna University of Technology[1], ETH Zurich[2]

This contribution deals with empirical investigations of the behavior of communication-time to computation-time ratios of three-dimensional parallel Fast Fourier Transforms (FFTs). Different problem sizes, number of processes, as well as different types of communication structures such as: Ethernet, Fast Ethernet, Myrinet, and IBM SP Switch are considered. Preliminary results are given on algorithms developed at the Vienna University of Technology, the Paul Scherrer Institut, and at the ETH Zurich. All the transposition algorithms of this study were implemented in MPI, and are thus portable. Performance on specific network topologies therefore reflects the efficiency of these MPI implementations.

INTRODUCTION

FFTs are well known as tools for solving many computational problems. In particular, they are often used in solving partial differential equations. Among the many overview articles available, we recommend to interested readers a superb article by Henrici [6].

In the beam dynamic context, FFTs are used to solve the Poisson problem for a scalar Coulomb potential ϕ, from which the particle forces may be computed at each time step. This is done be solving a related integral equation for ϕ, namely

$$\phi(\vec{q}) = \int_{\Omega} G(\vec{q} - \vec{q}')\, \rho(\vec{q}')\, d\vec{q}', \quad \Omega \subset \mathcal{R}^3 \tag{1}$$

where $\vec{q}$ is any point in space, G is Green's function, and ρ is the charge density. After discretization, the complexity of solving this equation is $\mathcal{O}(M^2)$ where M is proportional to the problem size (number of grid points.) The convolution of Green's function with the charge density is computed by using the Conlvolution theorem, first by Fourier transforming the two quantities (G and ρ), computing the Hadamard product in $\vec{k}-$space, then inverse Fourier transforming the product. This gives a much more efficient algorithm of complexity $\mathcal{O}(M \log M)$. In three dimensions, an efficient parallel implementation of the discrete Fourier transform (as FFT) further enhances the performance of this procedure, see for example [1] and [7].

Our first concern is a three dimensional FFT on a cube. For $0 \leq p, q, r < n$, the transformation can be written:

$$y_{p,q,r} = \sum_{s=0}^{n-1} \sum_{t=0}^{n-1} \sum_{u=0}^{n-1} \omega^{\pm(ps+qt+ru)} x_{s,t,u} \tag{2}$$

where $n = 2^m$ (binary radix) and $\omega = e^{\frac{2\pi i}{n}}$ is the $n-$th root of unity. One dimension (z) is distributed as slabs over multiple processors.

The FFT computation on each component (G and ρ) in equation (1) is performed in three steps: first by the independent rows, then the independent columns

$$\begin{aligned}
\forall s,t: \quad Z^{[1]}_{s,t,r} &= \sum_{u=0}^{n-1} \omega^{ru} x_{s,t,u} \\
\forall s,r: \quad Z^{[2]}_{s,q,r} &= \sum_{t=0}^{n-1} \omega^{qt} Z^{[1]}_{s,t,r} \\
\forall q,r: \quad y_{p,q,r} &= \sum_{s=0}^{n-1} \omega^{ps} Z^{[2]}_{s,q,r}.
\end{aligned}$$

Since the $z-$directional data are distributed across processors, a data transpose must be performed to redistribute the $z-$direction vectors so that each vector is resident in local memory. In effect, the transpose is two dimensional, with the third dimension y forming *pencils* out of 2-D $x - z$ blocks within the *slabs*. A more detailed discussion of the transposition algorithm is given in [1]. Using the MPI command MPI_Sendrecv_replace, non-diagonal blocks are exchanged using an *exclusive or* exchange address computation. Every block, which is locally CPU memory resident, is then transposed. To preserve the input x, y, z order, this transpose is done **twice**: to do the $z-$direction, then inverse transposed back to the original order. The whole of the input data are replaced with the un-normalized transform in equation (1).

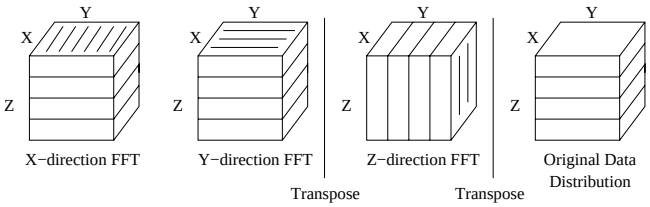

Fig. 1: Data distribution during the 3 steps

The aim of this paper is an analysis of the communication-time to computation-time ratio of these 3–D FFTs. Different FFT Kernels, problem sizes, and number of processors are studied on various machines. We are particularly interested in how the communication-computation time ratio changes on different systems and topologies.

NUMERICAL EXPERIMENTS

The Machines under Study

Experiments have been carried out on the following machines:

Asgard is a 500 processor Linux Cluster installed at ETH Zürich. The computing nodes are dual processor Pentium III Boards, each having 1 GB of memory. Some 192 Nodes have 500 MHz clocks and while 48 others have 650 MHz processors. A *frame* consists of 24 nodes connected by 100 MBit/s Ethernet Switches. The 10 frames of the system are connected to each other and to service and file server nodes by 1 Gbit/s optical links.

Seaborg is a large IBM RS/6000 SP-3 machine at the National Energy Research Scientific Computing Center (NERSC). It has 380 Compute Nodes for a total of 6080 CPUs. These nodes each have between 16 and 64 GB of memory and 166 IBM Power3 375 MHz Processors. Seaborg's network is the IBM *Colony* system having two *GX Bus Colony* network adapters per node.

Alvarez is a Linux cluster at NERSC with 80 two way SMP Pentium III nodes. The **Alvarez** network is Myrinet 2000.

zBox is a machine constructed at the University of Zürich. It has 144 dual AMD Athlon-MP 2200+ (1.8 GHz) nodes for 288 CPUs. The network is an SCI 2-dimensional 12×12 torus.

A summary of important data characterizing the networks is compiled in Table 1 .

Tab. 1: Network parameters.

Network	latency	pt.-pt. rate
Ethernet	175 μs	10 Mbit/s
Fast Ethernet	175 μs	100 Mbit/s
Myrinet	6 μs	3.9 Gbit/s
SCI (Scali)	5-6 μs	5.3 Gbit/s
Infiniband	2 μs	10-30 Gbit/s

The FFT Routines

The FFTs we tested are:

- FFTW [3]
- POOMA r1 [5]
- wpp3DFFT uses a generic implementation of Temperton's [2] in-place algorithm for the case $n = 2^m$.

The Process of Measurement

Measuring timing is not entirely trivial. Because the various machines used have different operating system parameters, codes must be instrumented in several ways to check consistency. One basic timer uses the clock routine provided in sys/times.h. Others use the MPI wallclock timer MPI_Wtime and the familiar command line system timer time.

Our experiments have running times between approximately 10^{-3} and 10^2 seconds. To assure consistent timing data, even for the very short runs, enough repeated forward/back transforms are performed to find a consistent average timing for one transform. This method is less sensitive to timer resolution. We measure wallclock time, system time, and user time.

PRELIMINARY RESULTS

Different FFTs show quite similar results if the number of processes is around the optimum for a given problem size (Fig. 2). On the other hand, if too many processors are designated for a certain problem size, the performance deteriorates because more communication is necessary. FFTW allows tuning for that and does not use all designated processors (see flat FFTW-graph between 128 and 256 processors in Fig. 2).

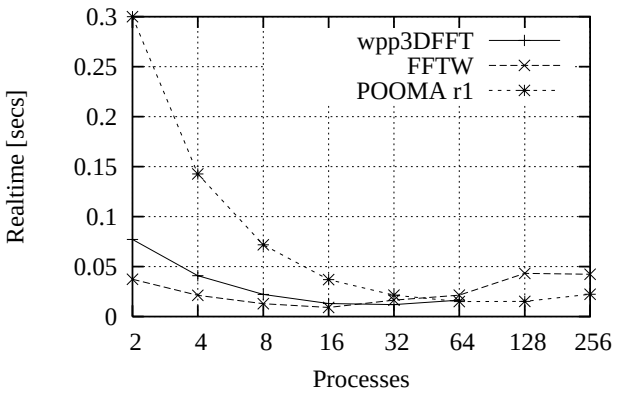

Fig. 2: Runtimes on *Seaborg*

Of particular interest to us is the ratio

$$r = \frac{\text{communication time}}{\text{total runtime}} \times 100. \qquad (3)$$

More precisely, communication time is the time for the two matrix pencil transpositions including local transposes. The *computation time* is the remainder of the total 3–D FFT time.

Our basic transpose operation is a simple case of matrix transposition. Few communication problems are in principle easier, but in practice more difficult to perform effectively. Both the linear systems software ScaLAPACK [4] and FFT package FFTW [3] require a variety of such transpositions. As shown in Fig. 3, the ration r varies considerably between the FFTs tested.

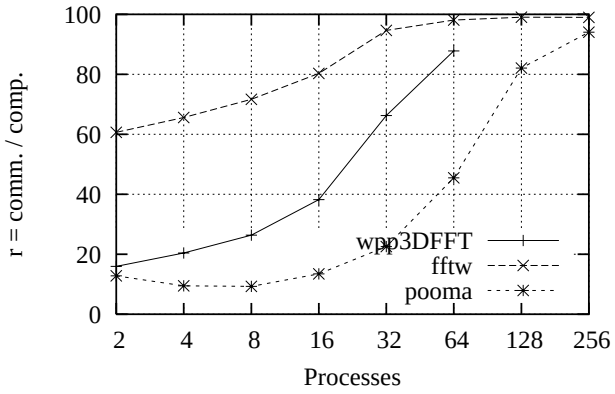

Fig. 3: r on *Seaborg*

Pooma spends a lower percentage of the execution time in the transpose function. The reason for this is that the communication time and the overall runtime for a small number of processors is higher than those of the other routines. The times spent in transpose functions (Fig. 4) show that pooma's transpose implementation is faster than the others but not as dramatically as r (3) indicates. Additionally, it is only faster on a large numbers of processors.

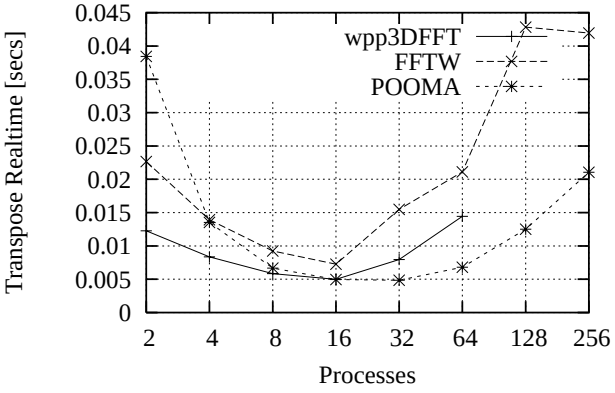

Fig. 4: Communication time on *Seaborg*

FFTW obviously has a very good implementation of the computation part. It is the fastest of the tested FFTs for small numbers of processes. The reason of this is probably that FFTW uses a *planner-function* which optimizes the algorithms used. For larger numbers of processors, the times become worse than its competitors. That seems to indicate that its transposition implementation has some tuning potential.

These results differ widely between tested machines as they have different network- and CPU-performance. Fig. 5 shows the r-values (3) for wpp3DFFT on some of the tested machine-configurations. More detailed results and conclusions will be published elsewhere [8].

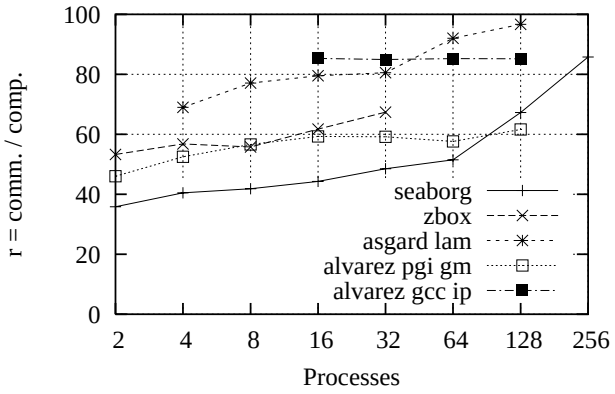

Fig. 5: r on chosen Machine configurations for wpp3DFFT, problem size = 256

REFERENCES

[1] W. P. Petersen and P. Arbenz, *Introduction to Parallel Computing*, Oxford University Press, 2004.

[2] C. Temperton, *Self-sorting In-place Fast Fourier Transforms*, SIAM J. Scientific and Statistical Computing, vol. 12, pp. 808-823, 1991.

[3] M. Frigo and S. G. Johnson, *Fast Fourier Transforms in One or More Dimensions*, available from NETLIB or http://fftw.org.

[4] L. S. Blackford et al., *ScaLAPACK Users' Guide*. SIAM Books, 1997, available from http://www.netlib.org/scalapack/.

[5] J.C. Cummings and W.F. Humphrey, *Parallel Particle Simulations using the POOMA Framework*, 8th SIAM Conf. Parallel Processing for Scientific Computing, 1997

[6] P. Henrici, Fast Fourier methods in computational complex analysis, SIAM Rev. 21 (1979), 481-527.

[7] Alan Edelman, Peter McCorquodale and Sivan Toledo, The Future Fast Fourier Transform, SIAM J. Sci. Comput. 20(3) (1999), 1094-1114

[8] A. Bonelli, A. Adelmann, W. P. Petersen and C. W. Überhuber, Communication Performance of Parallel 3D FFTs Using Various Networks and Transposition Algorithms accepted for presentation at VECPAR-04

ACKNOWLEDGMENTS

One of the authors (A.A) acknowledges the used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.