Communication Efficiency of Parallel 3D FFTs

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

Research Report No. 2004-04
May 2004

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

[1]Paul Scherrer Institut, Villigen, Switzerland
[2]Institute for Analysis and Scientific Computing, Vienna University of Technology

Communication Efficiency of Parallel 3D FFTs

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

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Research Report No. 2004-04 May 2004

Abstract

This paper describes the empirical investigation of the communication to computation time ratio of three-dimensional parallel FFTs. Different problem sizes, number of processes, and various communication networks are considered. Results are given on algorithms developed at the Vienna University of Technology, the Paul Scherrer Institut, and ETH Zurich. All transposition algorithms of this study were portably implemented in MPI. Thus, the communication performance on specific networks reflects the efficiency of these MPI implementations.

[1]Paul Scherrer Institut, Villigen, Switzerland
[2]Institute for Analysis and Scientific Computing, Vienna University of Technology

1 Introduction

Topic of this paper is the three-dimensional fast Fourier transform (FFT) on a cube. For $0 \le p, q, r \le n-1$, the transformation can be written as

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

where $n = 2^m$ (binary radix) and $\omega = \exp(2\pi i/n)$ is the nth root of unity. One dimension (z) is distributed over slabs corresponding with the processors on which the computation is carried out (see Fig. 1).

The computation of (1) is performed in three steps: first by the independent rows, then the independent columns:

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

Since the data in z-direction are distributed across processors, a transpose must be performed to redistribute the z-direction vectors so that each z-vector completely resides on one processing node. In effect, the transpose is two-dimensional, with the third dimension x forming *pencils* out of two dimensional y-z blocks within the *slabs*. A 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. Then every block, now CPU memory resident, is locally transposed. To preserve the input x, y, z order, this transpose is done *twice*: (i) to do the z-direction, and then (ii) inversely transpose back to the original order. Thus, the input data are replaced by the denormalized counterpart of transform (1).

The aim of this paper is to analyze the communication to computation time ratio of various 3D FFTs. Different FFT algorithms, problem sizes, and numbers of processes were run on various machines. To characterize *machine balance*, this paper investigates how the ratio of communication and computation time changes depending on communication infrastructure and topology.

2 Numerical Experiments

The Computer Systems. Numerical experiments were carried out using the following machines.

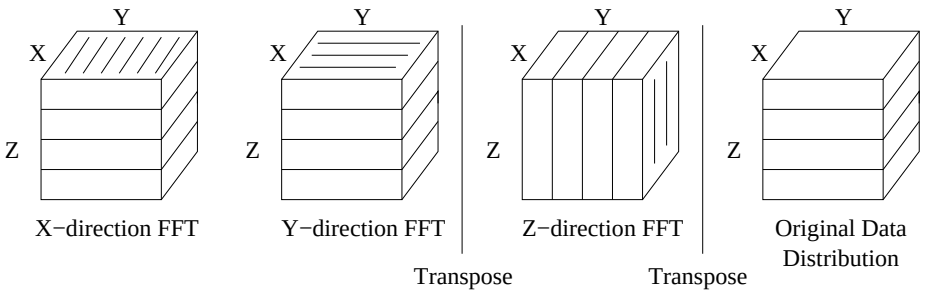


Figure 1: Data distribution during the three steps.

Alvarez is a Linux cluster installed at the National Energy Research Scientific Computing Center (NERSC). It consists of 80 two way SMP nodes equipped with Pentium III processors running at 866 MHz. The nodes of *Alvarez* are interconnected by Myrinet 2000.
(`http://www.nersc.gov/alvarez/`)

Asgard is a 480 processor Linux cluster installed at the ETH Zurich. The compute nodes are dual Pentium III boards, each provided with 1 GB memory. 192 nodes have 500 MHz clocks and 48 nodes run at 650 MHz. 24 nodes constitute a frame connected by a 100 MBit/s Ethernet switch. The 10 frames of the system are connected with each other and with the login and file servers by 1 Gbit/s optical links. (`http://www.asgard.ethz.ch/`)

Quad is a Linux cluster at the University of Bristol with 16 AMD Opteron 144 processors running at 2.2 GHz. The machine consists of four 4-way nodes with 4GB memory each. The network is Quadrics Elan4 with a QMS8A switch. (`http://quad.bris.ac.uk/`)

Regatta is an IBM machine at the Swiss Center for Scientific Computing (CSCS) in Manno which is part of ETH Zurich. It consists of 8 Regatta and 2 Nighthawk SMP nodes. Each Regatta node has 32 IBM Power4 processors (running at 1.3 GHz) and between 64 and 128 GB memory. The Nighthawk nodes are equipped with 16 IBM Power3+ processors running at 375 MHz and are provided with 16 GB memory. The network infrastructure is a double IBM Colony SP switch. Basically *Regatta* is the name of the architecture of this machine and not the machine itself. However, for simplicity the machine will be referred to as *Regatta* too.
(`http://www.cscs.ch/`)

Seaborg is an IBM RS/6000 SP computer system at NERSC. It has 380 compute nodes with a total of 6080 CPUs. The nodes have between 16 and 64 GB of memory and are equipped with IBM Power3 processors running at 375 MHz. The network switch is IBM's *Colony* connected to two *GX Bus Colony* network adapters per node.
(`http://hpcf.nersc.gov/computers/SP/`)

zBox was built at the University of Zurich. It consists of 288 CPUs of type AMD Athlon MP 2200+ (1.8 GHz) installed on 144 nodes. The network is an SCI 2D 12 × 12 torus.
(`http://krone.physik.unizh.ch/~stadel/zBox/`)

Table 1: Communication Network Parameters.

Network	Latency	Pt.-Pt. Rate
Ethernet	175 μsec	10 Mbit/s
Fast Ethernet	175 μsec	100 Mbit/s
Myrinet	6 μsec	3.9 Gbit/s
SCI (Scali)	5 – 6 μsec	5.3 Gbit/s
IBM *Colony* SP Switch	2 μsec	2.4 Gbit/s
Quadrics	2 μsec	7.2 Gbit/s
Infiniband	2 μsec	10 – 30 Gbit/s

The FFTs. The following FFT programs were tested: (*i*) FFTW 2.1.5 [3], (*ii*) POOMA r1 [5], and (*iii*) wpp3DFFT that uses a generic implementation of Temperton's in-place algorithm [2] for $n = 2^m$ and the simple `MPI` transpose in [1].

The Process of Measurement. Because of the different operating system parameters, the codes were instrumented manually using the timing routine provided in `sys/times.h`.

The observed run times vary between one millisecond and 10 seconds. To guarantee consistent timing even for very short run times, repeated forward/back transforms were run on a zero-matrix for a given time interval to calculate the average run time for one transform. This procedure has the advantage that timer resolution becomes less critical. In the experiments described in this paper, wallclock time, system time, and user time were measured.

3 The Results

Of particular interest is the ratio r of communication time to total run time. Communication time is the time required for the two matrix pencil transposes, including local transposition, needed to compute the FFT.

The basic transpose operation is a simple case of matrix transposition. Few communication problems are easier in principle, but more difficult to perform effectively in practice. Both the linear algebra package ScaLAPACK [4] and the FFT package FFTW [3] require a multitude of such transpositions.

Because of the huge amount of data gathered in more than 1000 runs, only a small number of significant results were singled out for the presentation in this paper.

3.1 A Comparison of FFT Routines

An interesting example showing the differences of various FFT routines running on *Seaborg* is a complex FFT of size 64^3 (Figs. 2 (a), (c), and (e)).

In this case the overall run times of wpp3DFFT and FFTW are quite similar. FFTW's computation part is more efficient because of the large effort that FFTW devotes to its *planner function* which optimizes the codelets (algorithm parts) used for a given machine. In particular, radix-2 transforms are usually less efficient than radix-4 or radix-8. Thus, computing most of the $n = 2^m$ transform as radix-4, radix-8, or even larger radices, is potentially much faster. A generic radix-2 procedure is less efficient as larger radices reduce the surface (input/output data traffic) to volume (actual floating point computation) ratio. Furthermore, if the number of processes is too large for a given job, the *planner function* does not use all of them but chooses a smaller number for efficiency reasons. In our case, the 256 processor job does not use all given processors because it needs approximately the same overall and transpose time as the 128 processor job.

For a small numbers of processors, POOMA is slower than the other FFT programs, but it does much better with an increasing number of processes. Fig. 2 (c) shows that POOMA's transpose implementation works well on this machine so in this case the bulk of the work is the calculation part. POOMA's times get better for larger numbers of processes as the calculation part becomes less important for overall run time.

Fig. 2 (e) shows the communication to computation time ratio r in percent. POOMA's values are low because of the reasons just described. FFTW spends at least 60 percent of its overall time in its transpose part. The computation part of FFTW is fast, but its transpose operations are slow.

Surprisingly, transposition time on *Seaborg* is relatively high, taking into consideration that it has many CPUs/board, a strong network, and comparatively slow processors (see Section 2). Generally, most of the massively parallel machines have slower networks than *Seaborg* and—especially on Beowulf clusters—faster processors.

To compare things, Figs. 2 (b), (d), and (f) show the same test cases investigated on *Asgard* which has slightly faster processors than *Seaborg* but a much slower network and dual-processor nodes (see Section 2).

If only two processors are used, the computations' on-board performance varies significantly. When off-board network communication is needed, run times become much worse on *Asgard* than on *Seaborg*. The time required for the transpose part increases by an order of magnitude. The ratio plot in Fig. 2 (f) shows that wpp3DFFT and POOMA spend much more time in the transpose parts when run on *Asgard* than on *Seaborg*. On *Asgard* at least 70 percent of the total run time is needed to transpose the data array when the network is involved.

Although POOMA's computational parts are inefficient, as compared with FFTW or wpp3DFFT, it is the fastest of the three tested FFTs on *Asgard*.

3.2 A Comparison of Machines

Fig. 2 (h) shows the communication to computation ratio r (in percent) and overall run times for wpp3DFFT doing a 256^3 FFT on a multitude of different machines.

The run time graphs in Fig. 2 (g) reflect the bandwidth of the network infrastructures (see Table 1). *Asgard* and *Alvarez*—both using Ethernet—are the slowest machines followed by *Alvarez* with Myrinet enabled. Next is *zBox* with Scali and *Seaborg* with SP Switch. The fastest machines are *Regatta* (SP Switch) and *Quad* (Quadrics).

Even on machines with slow processors and fast networks like *Seaborg* the program wpp3DFFT spends at least 40 percent of its overall run time for communication.

4 Conclusions and Outlook

The numerical experiments described in this paper show that on all the networks tested, the transposition needed to move the distributed z-direction vectors to be resident on one processor and back requires a substantial portion of the overall time needed to compute a 3D FFT. On a well balanced machine, the communication to computation ratio $r \approx 1$ or less on a powerful network. We found that r is never below 40 percent, and grows when the number of processors increases. Some networks are more efficient for transposition than others, cf. Fig. 2 (h). We also saw that some algorithms are more efficient than others. For example, the simple `MPI_Sendrecv_replace` procedure of wpp3DFFT is quite efficient. This procedure applies only to cases where both the dimension n and the number of processors are powers of two, and thus is too inflexible for most purposes. However, this algorithm can be modified for more general cases as follows.

From the discussion of the *slab* layout in Fig. 1 it follows that the transposition is essentially two-dimensional. One other dimension (the X-direction in Fig. 1) may be chosen to form *pencils* making up blocks. Let the size of the 2D array be $N_Y \times N_Z$ ($Y \times Z$ in Fig. 1), where N_Z is distributed into p slabs, where p is the number of processors, i. e., $p|N_Z$ (p divides N_Z). In addition, if $p|N_Y$, the block transposition of [1] can be done by an index digit permutation [6]. Let $0 \le b \le p-1$ count the blocks (of size $(N_Z/p) \times (N_Y/p) \times N_X$ `complex` data) in any slab numbered `CPU`, and B number the blocks in the y-z array:

$$B = \texttt{CPU} \times p + b,$$

where $0 \le \texttt{CPU} \le p-1$. Using `MPI_Sendrecv_replace`, block B must be exchanged with its corresponding block B', given by the permutation

$$B' = b \times p + \texttt{CPU}.$$

That is, block b on processor `CPU` is to be swapped with block $b' = \texttt{CPU}$ on processor $\texttt{CPU}' = b$. When $b = \texttt{CPU}$, the blocks are diagonal and only local

memory transposition is required. A total of $p(p-1)/2$ block pairs are swapped. For non-commensurate cases (when p does not divide N_Y), either the number of processors p must be changed, or padding data are required.

It has been demonstrated that the transposition time in 3D FFTs requires a significant share of the total computation time. On some networks, Ethernet or even Gigabit Ethernet, transposition even dominates calculation.

Much more work on general cases for N_X, N_Y, N_Z, as outlined above, has still to be done.

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/`.

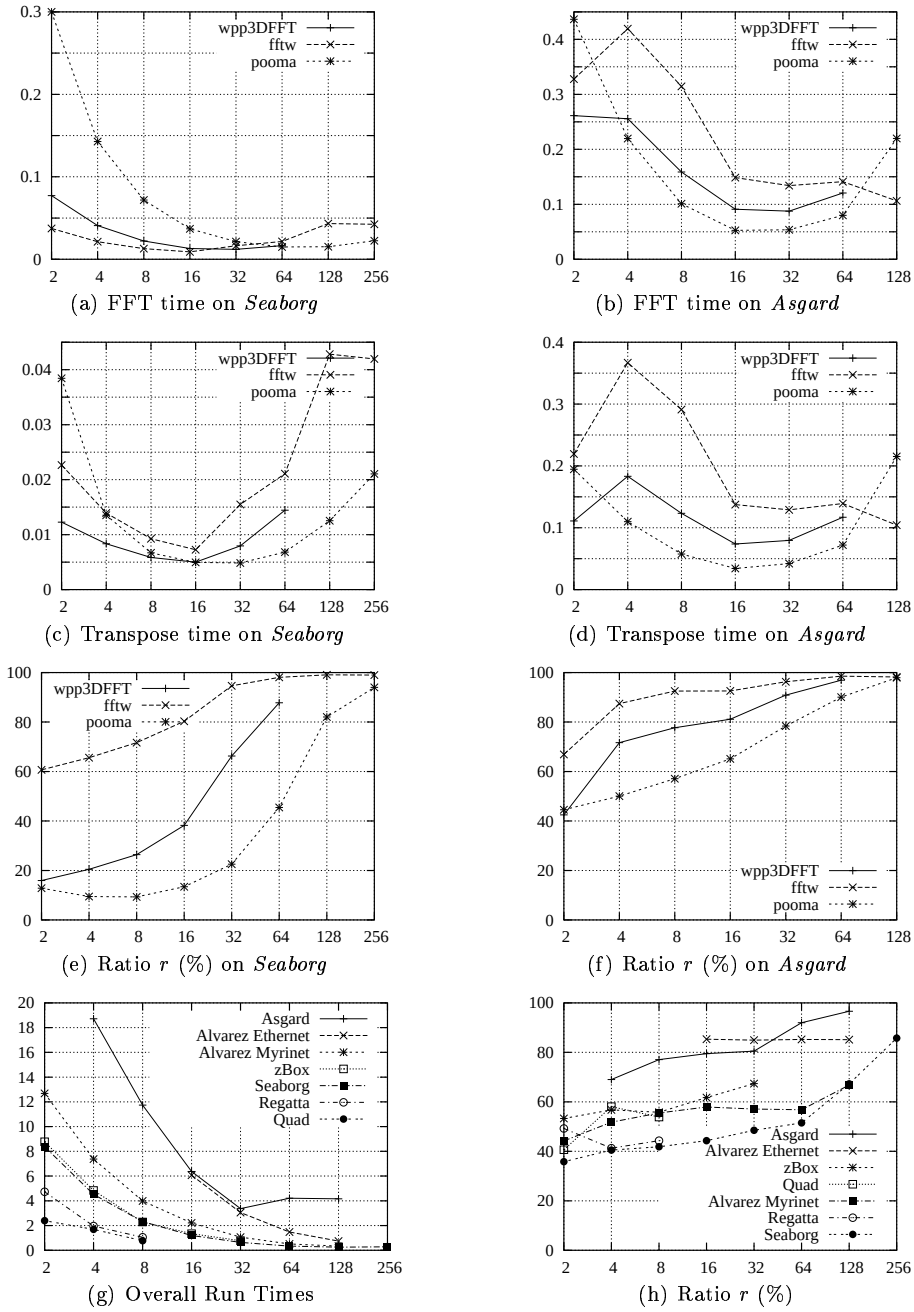
[5] J. C. Cummings and W. F. Humphrey, *Parallel Particle Simulations using the POOMA Framework*, 8th SIAM Conf. Parallel Processing for Scientific Computing, Mar. 14–17, 1997, CD available as ISBN 0-89871-395-1.

[6] H. W. Johnson and C. S. Burrus, *The Design of Optimal DFT Algorithms Using Dynamic Programming*, IEEE Trans. Acoust. Speech Signal Processing, vol. 31, pp. 378-387, 1983.

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.

Part of the work described in this paper was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund FWF.



(a) FFT time on *Seaborg* (b) FFT time on *Asgard*

(c) Transpose time on *Seaborg* (d) Transpose time on *Asgard*

(e) Ratio r (%) on *Seaborg* (f) Ratio r (%) on *Asgard*

(g) Overall Run Times (h) Ratio r (%)

Figure 2: (a), (b):Overall run times (in sec.); (c), (d): transpose times (in sec.); (e), (f): ratio r (in percent) vs. no. of processors for *Seaborg* and *Asgard* running FFTs of size 64^3; and (g),(h): run times (in sec.) and ratio r on various machines for wpp3DFFT computing FFTs of size 256^3.