

Gigaelement FFTs on x86-based Apple clusters

S. Noble

Advanced Computation Group, Apple Inc.

31 Jul 07

Abstract: This paper is an update to previous work [1] on the use of computational nodes for very large FFTs. Test code has been updated for Intel-based hardware—again using the OS X Accelerate framework [6] for all component-FFTs—while new performance measurements are provided for a canonical x86 hardware configuration. These more modern tests exhibit considerable speed advantages. A performance example is this: One can sustain > 2 gigaflops real-time for double-precision gigaelement (length- 2^{30} -complex) FFTs on a 4-machine cluster.

1 Motivation

The 2004 paper [1] addressed the question posed by researchers at Lawrence Livermore National Laboratory (LLNL), namely: How should Apple G5’s be configured to effect a 3-dimensional, billion-element FFT? We endeavor to answer this question for Apple clusters configured with Intel processors using the distributed FFT software developed for [1].

We remain concerned with each of these three scenarios:

- 1-dimensional FFT with length 2^{30} ,
- 2-dimensional FFT with length $2^{15} \times 2^{15}$,
- 3-dimensional FFT with length $2^{10} \times 2^{10} \times 2^{10}$,

The relevant signal-processing algebra, transposition mechanics, proper use of component-FFTs (i.e., smaller FFTs that are used to build up the gigaelement¹ variety), and bounding estimates on floating-point errors in the gigaelement region are described in that earlier paper [1]. For reader convenience we repeat here the algebraic definition of the 3-dimensional FFT in play; namely, the FFT in this case is S_{pqr} , based on the original $W \times H \times L$ signal s as

$$S_{pqr} = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} \sum_{z=0}^{L-1} s_{xyz} e^{-2\pi i(xp/W + yq/H + zr/L)}.$$

This formula reminds us a) that we are using the “engineer’s FFT” definition, b) that there is unit normalization for the forward direction, and c) that component FFTs of respective lengths W, H, L can—along with proper transpose operations—be used to effect the full FFT.

2 Configuration

The 2004 paper included a canonical G5 cluster configuration with four dual-processor 2.0 Ghz PowerMac G5 computers. Each machine was configured with 6.5 GB² of RAM and connected to a Netgear GS108 8-port gigabit ethernet switch. The machines ran MacOS X 10.3.4.

Our test cluster consists of four machines, each configured with two dual-core 2.66 Ghz Intel Xeon 5100 “Woodcrest” processors and eight gigabytes of RAM. The machines each have

¹“Gigaelement” is an approximate term meant to represent calculations on *roughly* a billion elements.

²In order to combat confusion in the engineering community, the International Electrotechnical Commission defined the prefix *gibi* to denote 2^{30} . Thus 2^{30} bytes should properly be denoted as a gibibyte, or GiB for short. Since RAM manufacturers have traditionally sold 2^{30} -byte modules of memory as 1 GB, we use that convention here.

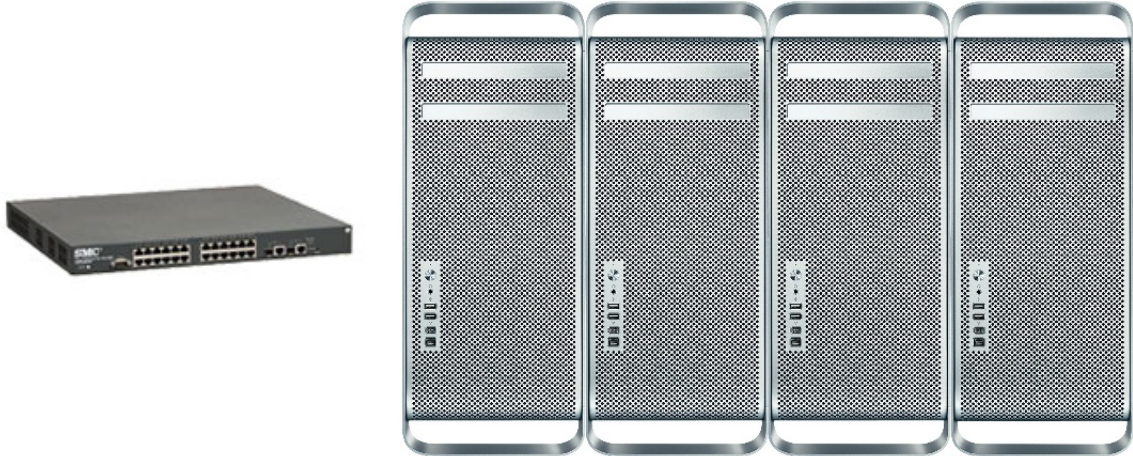


Figure 1: The canonical Mac Pro cluster consists of four dual-core 2.66 Ghz Mac Pro workstations with 8 GB RAM connected to an SMC8024L2 switch.

two connections to an SMC TigerSwitch 24 Port Gigabit Ethernet Switch (SMC8024L2) using gigabit ethernet and 802.3ad link aggregation. The machines run MacOS X 10.4.9 and Open MPI 1.2.1.[3]

The specific memory requirements of a cluster designed to run our FFT software depend on the length of the signal, the size of the cluster, and the precision of the data. The values in Table 1 are computed from the formula for required memory μ per machine:

$$\mu = \frac{3}{2} \cdot \frac{N}{M} \cdot b + W(N) + c$$

N	complex elements in the signal	
M	number of machines	
b	number of bytes per complex element	(1.1)
$W(N)$	space for FFT weights	
c	approximate system overhead	

The $3/2$ factor comes from the use of a transposition workspace buffer (described later in this paper) half the size again of the actual data. For single-precision complex elements, $b = 8$ bytes, while for double-precision complex elements, $b = 16$ bytes. Space for the FFT weights is proportional to the longest 1-dimensional FFT. As before, we observe that $c \approx 0.5$ GB.

N (signal length)	M (# of machines)	μ (single precision)	μ (double precision)
2^{28}	1	3.5 GB	6.5 GB
2^{28}	2	2.0 GB	3.5 GB
2^{30}	2	6.5 GB	N/A
2^{30}	4	3.5 GB	6.5 GB
2^{32}	8	6.5 GB	N/A
2^{32}	16	N/A	6.5 GB
2^{32}	16	6.5 GB	N/A

Table 1: Required physical memory μ , per machine, to run the FFT software on varying lengths of signals and varying numbers of computers.

Because we are interested in running both the single- and double-precision gigaelement transforms we choose the ($M = 4$)-machine cluster to be what we call our “canonical cluster.” Note that 3.5 GB and 6.5 GB are not valid configurations for Mac Pro workstations and so we have equipped our test machines with 8 GB of RAM.

3 Instructions for very large FFT cluster

1. Install Xcode developer tools. [4]
2. Install Open MPI. [3] The Open MPI project distributes a script which is capable of building four-way (PowerPC, PowerPC64, i386, and x86-64) universal binaries of all the MPI libraries and packaging these binaries with a Mac OS X installer. Look for `contrib/dist/macosx-pkg/buildpackage.sh`
3. Configure an Open MPI hosts file – an example hosts file for our canonical cluster might be as follows:

```
$ cat hosts
fft1.local. np=4
fft2.local. np=4
fft3.local. np=4
fft4.local. np=4
```

4. Configure ssh on the compute nodes to enable public key authentication. This can be as simple as the following commands:

```
$ ssh-keygen -t dsa
$ scp ~/.ssh/id_dsa.pub fft1.local:~/.ssh/authorized_keys
```

```
$ scp ~/.ssh/id_dsa.pub fft2.local:~/.ssh/authorized_keys
$ scp ~/.ssh/id_dsa.pub fft3.local:~/.ssh/authorized_keys
$ scp ~/.ssh/id_dsa.pub fft4.local:~/.ssh/authorized_keys
```

5. Build the software with Xcode or with make:

```
$ make
```

6. Run the software.

```
$ mpiexec --bootfile hosts dist_fft_split_double_out_of_place 30
```

One may also employ Apple's XGrid technology [5] for managing the distributed computation software, as in [1].

4 Software

Our present experiments use the same software implemented in [1] with modifications necessary to build on the x86 platform. The original implementation leveraged a hand-vectorized, in-place, single-precision, square-matrix transpose for the Velocity Engine. For the current tests, we worked out an SSE implementation of the same functionality. We stress that the mathematics and the algorithms have not changed, and so we refer the interested reader to [1].

The matrix transpositions are among the most time intensive operations performed on the data set. In our measurements, matrix transpositions are competitive with the 1-dimensional Fourier transform function in terms of computational overhead. The 3-dimensional FFT offers the opportunity to work on many square 2-dimensional matrices, for which there is a highly efficient transposition algorithm. These 2^{20} -element matrices are small enough to fit almost entirely within the processors shared four megabyte L2 cache. The benefits of this situation are apparent from the benchmarks which show a very clear performance advantage when moving from the 2-dimensional to the 3-dimensional transforms as well as a remarkable improvement in 3-dimensional performance versus the G5.

5 Performance measurements

The software described in [1] can be configured to allow single- or double-precision floating point arithmetic, split or interleaved complex data, using in-place or out-of-place matrix transpositions. The 2004 work presented performance data which we include here for easy

comparison. The x86 results presented in Table 2 represent the best performance over the entire matrix of available settings³ gathered by averaging the results of four forward and inverse FFT “runs.”

Important note: For convenience, we approximate throughput by assuming a theoretical complexity of $C \cdot N \cdot \log_2 N$ operations per FFT. N is the number of elements in the FFT and C is an algorithm-dependent constant—herein we simply assume Cooley–Tukey complexity equivalence, so that C takes herein the classical value 5. For our gigaelement FFTs, $N = 2^{30}$ and so the transform requires *approximately* 160×10^9 real multiply and add operations [1]. This manner of approximation avoids the delicate issues involving fused operations, FFT-butterfly structure, and so on.

Just as with the older G5 software, all the component-FFT routines used in these performance tests are provided by Apple’s Accelerate framework [6].

Signal length	Precision	Time per FFT (sec)		Throughput (Gflops)	
		Intel	G5	Intel	G5
2^{30}	single	48.7	35.4	3.3	4.5
$2^{15} \times 2^{15}$	single	48.5	35.1	3.3	4.6
$2^{10} \times 2^{10} \times 2^{10}$	single	42.6	43.5	3.8	3.7
2^{30}	double	76.8	85.8	2.1	1.9
$2^{15} \times 2^{15}$	double	76.4	86.4	2.1	1.9
$2^{10} \times 2^{10} \times 2^{10}$	double	71.1	120.8	2.3	1.3

Table 2: Average time per FFT and throughput for the canonical 4-computer cluster. Note that the gigaflop ratings are based, simply, on a classical Cooley–Tukey complexity (see in-text note above). Thus, gigaflops are approximated by $(5N \log_2 N)/(\text{nanosec. real-time})$.

Note that the Intel performance is considerably better on all double-precision modes. Future research will undoubtedly speed up these computations by virtue of explicit leveraging of newer multi-core hardware with multi-threaded transforms.

6 Related projects and future directions

Any exposition on the topic of distributed Fourier transforms would be incomplete without mentioning that the FFTW [2] project is introducing preliminary support for MPI FFTs in the alpha release of their next major update. We also note that Intel offers distributed Fourier transform support in the cluster edition of their Math Kernel Library.

³Our experimental cluster did not have enough memory to measure performance of interleaved double-precision data using out-of-place transpositions.

Turning to future research avenues, we remark that in any distributed computation, questions of communication overhead naturally arise. In our measurements of double-precision transforms, network communication accounted for approximately half the total clock time. One way to address concerns over network bottlenecks is to reduce the number of physical machines involved in each computation. We observed in [1] that distributed FFT software can be useful in a single machine “cluster” to overcome address space limitations imposed by a 32-bit memory model.

The 32-bit memory model constraint is no longer an issue for recent releases of Mac OS X, which support 64-bit memory addressing for computational processing. In a future paper, we intend to address the topic of multi-threaded fast Fourier transforms in a single 64-bit process.

Further, the FFT computation itself can be improved. In particular, as suggested by E. Postpischil, a more cache-friendly partial-transpose scheme would likely provide measurable performance benefits—particularly in the 3-dimensional case.

7 Acknowledgments

The author thanks E. Postpischil for comments and suggestions, A. Sazegari, I. Ollman for long-term support of the whole 3D-FFT project, and R. Ustach, E. Prabhakar who have kindly informed the Advanced Computation Group of Apple scitech-user needs. Thanks are also due to R. Crandall, E. Jones, J. Klivington, and D. Kramer for the first version of this paper as well as the original PPC implementation of the distributed FFT software.

References

- [1] R. Crandall, E. Jones, J. Klivington, D. Kramer, “Gigaelement FFTs on Apple G5 Clusters,” 2004. <http://www.apple.com/acg/>
- [2] FFTW, <http://www.fftw.org/>
- [3] Open MPI, <http://www.open-mpi.org/>
- [4] Xcode, <http://developer.apple.com/tools/macosxtools.html>
- [5] Xgrid, <http://www.apple.com/macosx/features/xgrid/>
- [6] Accelerate.framework, <http://developer.apple.com/performance/accelerateframework.html>