Auto-Tuning Distributed-Memory 3-Dimensional Fast Fourier Transforms on the Cray XT4

M. Gajbe [a] A. Canning, [b] L-W. Wang, [b] J. Shalf, [b] H. Wasserman, [b] and R. Vuduc, [a]
[a] *Georgia Institute of Technology, Atlanta, Georgia, USA*
[b] *Lawrence Berkeley National Laboratory, Berkeley, California, USA*

Abstract.
We present auto tuning, optimization, and performance modeling of 3 Dimensional Fast Fourier Transforms on Cray XT4 (Franklin) system. Spectral methods involving FFTs are a commonly used numerical technique with applications in engineering, chemistry, geosciences, and other areas of scientific computing. In the case of materials science the wavefunction of the electrons are expanded in spatial frequency components which is a natural basis since the wavefunction for a free electron is a plane wave. In this paper we study the performance of a 3D FFT specifically written for materials science applications. The problem with a parallel 3d FFT is that for a grid of N points the computational work involved is $O(NlogN)$ while the amount of communication is $O(N)$. This means that for small values of N (64 x 64 x 64 3D FFTs), the communication costs rapidly overwhelm the parallel computation savings. A distributed 3D FFT represents a considerable challenge for the communications infrastructure of a parallel machine because of the all-to-all nature of the distributed transposes required, and it stresses aspects of the machine that complement those addressed by other benchmark kernels, such as Linpack, that solves system of linear equations, $Ax = b$. Auto tuning can play a vital role in optimizing 3D FFT kernels for a diverse platforms as it can exploit features of the system-specific configuration characteristics to improve performance. We also depend on analytic performance models as a tool for predicting the idealized performance for the 3-dimensional Fast Fourier Transforms to set performance expectations for the target computational system. Overall, our methodology is able to achieve a substantial improvement in performance over conventional approaches to tuning 3D FFT performance.

Keywords. Fast Fourier Transform, Parallel Computing, Materials Science, Auto Tuning

Introduction

In electronic structure calculations in materials science first-principles methods based on Density Functional Theory (DFT) in the Kohn-Sham (KS) formalism [1] are the most widely used approach. In this approach the wave functions are usually expanded in plane waves (Fourier components) and pseudopotentials replace the nucleus and core electrons. This implementation requires parallel 3d FFTs to transform the electronic wavefunctions

from Fourier space to real space to construct the charge density. This gives a computationally very efficient approach with a full quantum mechanical treatment for the valence electrons, allowing the study of systems containing hundreds of atoms on modest-sized parallel computers. Taken as a method DFT-based codes are one of the largest consumers of scientific computer cycles around the world with theoretical chemists, biologists, experimentalists etc. now becoming users of this approach. Parallel 3d FFTs are very demanding on the communication network of parallel computers as they require global transpositions of the FFT grid across the machine. The ratio of calculations to communications for 3d FFTs is of order $\log N$ where N is the grid dimension (compared to a ratio of N for a distributed matrix multiply of matrix size N) which makes it one of the most demanding algorithms to scale on a parallel machine. A scalable parallel 3d FFT is critical to the overall scaling of plane wave DFT codes. In this work we have implemented five different versions of the communications routines used in the 3d FFT to determine which is the most efficient for different processor counts and different grid sizes.

In plane wave codes we have many electronic wavefunctions where each one is represented in fourier space so unlike spectral type codes we are typically performing many moderate sized 3d FFTs rather than one large 3d FFT. This has the disadvantage from the scaling point of view that it is difficult to efficiently scale up a moderate sized 3d FFT on a large number of processors but it has the advantage that for all-band codes we can perform many 3d FFTs at the same time to aggregate the message sizes and avoid latency issues. The wavefunctions are also represented by a sphere of points in Fourier space and a standard grid in real space where the sphere typically has a diameter about half the size of the grid. This means we can also reduce the amount of message passing and calculations required compared to using a standard 3d FFT where the number of grid points is the same in Fourier and real space. We have therefore written our own specialized 3d FFTs for plane wave codes that can run faster than using any public domain 3d FFT libraries such as FFTW or P3DFFT and have no restrictions on what grid sizes can be run. Our specialized 3d FFTs are used in many widely used materials science codes such as PARATEC, PeTOT, ESCAN [3].

DFT using the Local Density Approximation (LDA) for the exchange-correlation potential requires that the wavefunctions of the electrons $\{\psi_i\}$ satisfy the Kohn-Sham equations

$$[-\frac{1}{2}\nabla^2 + \sum_R v_{ion}(r-R) + \int \frac{\rho(r')}{|r-r'|}d^3r' + \mu_{xc}(\rho(r))]\psi_i = \varepsilon_i\psi_i \qquad (1)$$

where $v_{ion}(r)$ is the ionic pseudopotential, $\rho(r)$ is the charge density and $\mu_{xc}(\rho(r))$ is the LDA exchange-correlation potential. We use periodic boundary conditions, expanding the wavefunctions in plane waves (Fourier components),

$$\psi_{j,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{g}} a_{j,\mathbf{k}}(\mathbf{g})e^{i(\mathbf{g}+\mathbf{k})\cdot\mathbf{r}} \ . \qquad (2)$$

The selection of the number of plane waves is determined by a cutoff E_{cut} in the plane-wave kinetic energy $\frac{1}{2}|\mathbf{g}+\mathbf{k}|^2$ where $\{\mathbf{g}\}$ are reciprocal lattice vectors. This means that

the representation of the wavefunctions in Fourier space is a sphere or ellipsoid with each $\mathbf{g}$ vector corresponding to a Fourier component (see Figure 1). The $\mathbf{k}$'s are vectors sampling the first Brillouin Zone (BZ) of the chosen unit cell (or supercell). The Kohn-Sham equations are usually solved by minimizing the total energy with an iterative scheme, such as conjugate gradient (CG), for a fixed charge density and then updating the charge density until self-consistency is achieved (for a review of this approach see reference [2]). Some parts of the calculation are done in Fourier space and some in real space transforming between the two using 3d FFTs.

1. Parallel Data Decomposition 3d FFT

A 3d FFT consists of three sets of 1d FFTs in the x,y and z directions with transpositions of the data between each set of 1d FFTs. Only two transposes are needed if the final data layout is not required to have the same x,y,z order in both spaces. Since the $\mathbf{g}$ vectors (Fourier coefficients) are distributed across the processors these two transposes can require global communications across the parallel computer.

As mentioned in the previous section the data for a given wavefunction forms a sphere of points in Fourier space and a standard grid in real space (see Figure 1). The data distribution for the sphere is driven by 1) the need to have complete columns of data on a given processor to perform the first set of 1d FFTs 2) other parts of the code require intensive calculations related to the number of Fourier components each processor holds so to load balance this part of the calculation we require a similar number of Fourier components on each processor. The data layout we use in Fourier space is to order the columns of the sphere in descending order and then to give out the individual columns to the processors such that each new column is given to the processor with the fewest number of fourier components. In this way each processor holds sets of complete columns and approximately the same number of fourier components (see Figure 1 for an example of the layout on three processors). In real space we consider the grid as a one dimensional set of columns with (x,y,z) ordering and then give out contiguous sets of columns to each processor giving as closely as possible the same number of columns to each processor. In this way each processor will hold complete planes or sections of planes of the three dimensional grid (see 1). With this data layout we have no restrictions on the number of processors required for a given sphere or grid size. Also this data layout means that the first transpose in the 3d FFT typically requires all processors communicating with every other processor while the second transpose may require no communications (if the each processor has complete planes) or limited local communications if each processor has a section of a plane. In the case of SMP nodes complete planes can still reside on a node even if each processor is performing calculations on sections of a plane. A more detailed description of each step in our specialized 3d FFT can be found in reference [5].

2. Parallel Communication Structure for 3d FFT

In order to have a scalable parallel 3d FFT we therefore need to have as efficient as possible an implementation of the communications in the two parallel transposes. As

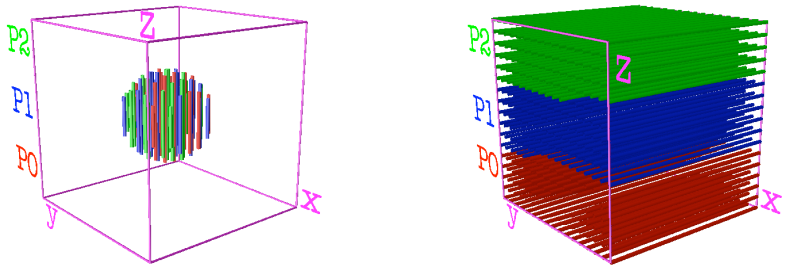

Figure 1. A three processor example of the parallel data layout for the wavefunctions of each electron in Fourier space (left) and real space (right). The different colors correspond to the data held by processors P0, P1 and P2.

mentioned above it is the first transpose that typically involves almost all the processors communicating with each other while the second transpose will typically involve limited local communications. We have implemented two basic types of MPI communications in our transposes. The first one uses MPI_ISEND and MPI_RECV and the second uses MPI_ALLTOALLV which is a collective operation allowing all processors to send distinct messages of different lengths to all the other processors. In the case of the ISEND and RECV version we have implemented different communication structures. In the first case all the processors send to processor one then processor two etc. at the same time while in the second case each processor sends to its processor number plus one, then plus two etc. as if in a ring, to avoid contention in communications. We also have what we will refer to as blocked versions of these 3d FFTs where we perform a number of 3d FFTs at the same time (typically 40) and so can aggregate the message sizes to avoid latency problems. In our particular application we are performing a large number of moderate sized 3d FFTs (our strong scaling tests will be for 512^3 grid) so it is important to take advantage of this blocking to avoid latency issues on large processor counts.

3. Motivation for Auto-Tuning

Achieving close to peak performance is important for all scientific applications. The performance is largely dependent upon computation kernels such as dense or sparse matrix operations, Fast Fourier Transforms, stencil computations etc. In general to obtain good performance on a specific platform these kernels require machine-dependent tuning by hand or using highly optimized machine specific scientific and mathematical libraries and compilers. A number of automatic tuning systems have been developed that typically uses - (*i*): Selecting an optimal implementation of a kernel (*ii*): developing multiple machine specific implementations of kernels (*iii*): Use of optimized system configurations.

However, conventional performance tuning by hand is time consuming, and can be error-prone. Moreover, the kernel should be tuned for a specific system configuration and processor. Another problem is that the hardware, system software, microprocessors

are continuously evolving. Even with intimate knowledge of the architecture, compilers and libraries, performance of a kernel is hard to predict. Modern high level compilers also face difficulty in optimizing the kernels for general use as they are not able to use the physics behind the computational kernel. Even performance of the best algorithms is dependent on problem size that is usually only determined at execution time.

Therefore, auto tuning plays a vital role in optimizing these kernels for a specific platform as it can make use of the domain or kernel specific knowledge as well as the system configuration. The best approach that can be taken towards auto tuning is, to identify a set of algorithms or methods and then find the fastest method for a given platform and problem size by collecting performance data. For testing, portability and maintenance of a code a modular design approach is required. Using such modular designs one can focus on the performance of a key subroutine or kernel. Tuning these kernels on several platforms automatically improves performance of an application that uses it.

4. Motivation for Studying 3D FFTs

One of the widely used kernels is 3 Dimensional FFT. It is a commonly used numerical technique in computational physics, engineering, chemistry, geosciences, and other areas of high performance computing. The 3D FFT is built on top of highly optimized versions of 1D FFT that are available on most modern systems. However, there is no optimized 3D FFT library or kernel available on today's modern supercomputers.

The most successful attempt to autotune FFTs are (*i*): FFTW [4](*ii*): Spiral [9].

4.1. FFTW :

The FFTW package [4] produces highly optimized Discrete Fourier Transform (DFT) kernels using a family of FFT algorithms. It is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). All fftw_mpi transforms are in-place. In particular, the data is divided according to the rows (first dimension) of the data: each process gets a subset of the rows of the data. (This is called a "slab decomposition.") One consequence of this is that you can't take advantage of more processors than you have rows (e.g. 64 x 64 x 64 matrix can at most use 64 processors). Hence, the 3D MPI FFT subroutine is not scalable as the maximum number of processors or cores that can be used is N for an N x N x N problem size.

4.2. SPIRAL :

Spiral generates platform-tuned implementations of digital signal processing (DSP) algorithms such as the discrete Fourier transform, discrete cosine transform, and many other numerical kernels. Spiral focuses on how to achieve optimal performance on numerical kernels using coding efforts. A high abstraction level for automation is used in Spiral that uses knowledge about available algorithms and algorithm optimizations to

generate a computer program.

Both FFTW and SPIRAL do not address automatic performance tuning of 3D MPI FFTs. FFTW is not scalable beyond the maximum of any of the 3 dimensions of 3D grid. In todays high performance world, 3 Dimensional FFT computations are widely used in various scientific applications and benchmarks. Hence, we decided to focus on automating this computational kernel.

4.3. Approach for Auto-tuning

In general, to auto tune a kernel involves two steps. (*i*): Identification of set of methods to perform 3D FFT. (*ii*): Identification of the fastest method after analyzing the collected data.

We have also chosen different methods to perform the transpose operation that is at the heart of 3D FFT. This is achieved using MPI library calls MPI_ISEND and MPI_RECV and MPI_ALLTOALLV as well as different techniques to perform the transpose operation as discussed above.

The steps involved in the auto tuning process are :

1. Overlapping of communication and computation using MPI_ISEND and MPI_RECV library calls.
2. The communications are staggered rather than all just sending to the same processor at the same time. The source and destination processors are used in a circular mechanism.
   - Process i will receive data from $mod(MyRank-i-1+NoNodes, NoNodes)$.
   - Process i will send data to $mod(MyRank + i - 1, NoNodes)$.
3. An optimal set of columns are packaged together and communicated.
4. The number of columns that are packaged together depends upon the optimal bandwidth of the network during the computation.

The domain decomposition of the 3D FFT's into component 1D FFTs is shown in figure 2. The results with the above tuning method and its comparison with different methods to perform the transpose operation are shown in table 1 and in the graph 3.

4.4. Performance Model of 3D FFT

The total execution time of an application on a platform is estimated as a linear combination of time spent by the processor, memory, communication subsystem, and I/O subsystem. The total estimated wallclock time is given by

$$T_{tot} = T_{init} + T_{comp} + T_{comm} + T_{overhead}$$

Where the wallclock time is composed of the computation time, communication time, time required for I/O and the overhead time, respectively. To estimate these terms, several executions of the application on the existing hardware platform are performed.

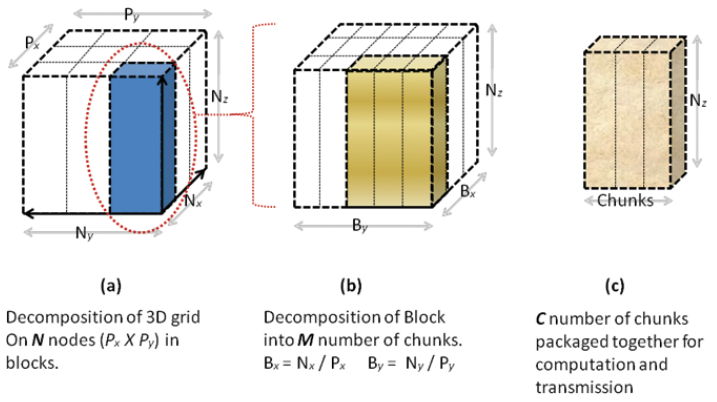

Figure 2. Pictorial representation of 3D FFT Auto tuning

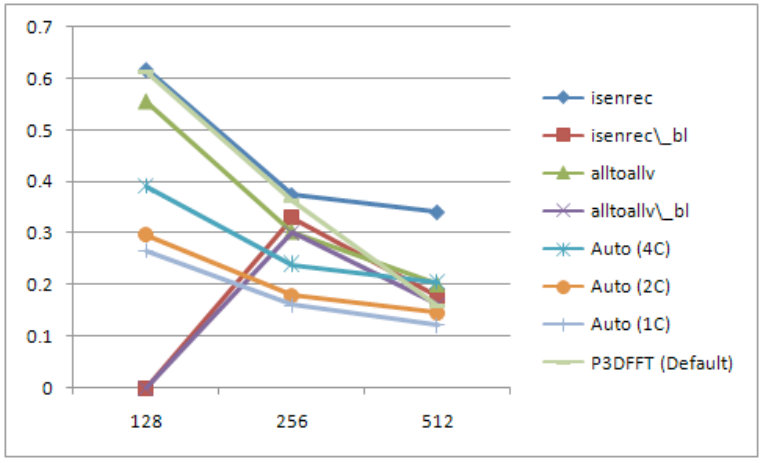

Figure 3. Auto tuning method and its comparison with different methods to perform a forward and reverse 3d FFT operation on a 512^3 grid. See Table 1 and text for an explanation of the different communication schemes. P3DFFT corresponds to the 3D FFT library version 2.2 developed by SDSC. It should be noted that this is for a real to complex 3d FFT so cannot be directly compared to our complex to complex 3d FFT which involves a dataset of double the size and over twice as many floating point operations

The performance data generated from these executions is used to estimate time required for each quantity.

The performance model of 3D FFT consists of 3 major components : problem size, computation, internode communication. The time for an iteration of 3D FFT can be given

by :

$$T_{fft}(N, P, Chunks) = T_{comp}(N, P) + T_{comm}(Chunks, P)$$

where
$T_{comp}(N, P)$: Computation Time
$T_{comm}(Chunks, P)$: Communication Time

Note that we excluded the data initialization time and data arrangement time from the actual 3D FFT computation.

4.5. Parallel Complexity

The parallel complexity for the block or cyclic FFT algorithm is :

$$T_p^{block/cyclic} = \frac{2mlogm}{p} + (\alpha + \frac{1}{\beta}\frac{m}{p})logp$$

$$= \frac{2mlogm}{p} + \alpha logp + \frac{1}{\beta}\frac{mlogp}{p}$$

The parallel complexity of transpose FFT algorithm is :

$$T_p^{transpose} = \frac{2mlogm}{p} + \alpha(p-1) + \frac{1}{\beta}\frac{m}{p}\frac{p-1}{p}$$

From the parallel complexity it shows that the block/cyclic algorithm is better than the transpose algorithms. Hence we can assume that in case of 3D MPI FFT instead of using alltoall operation to achieve transpose, non-blocking $ISend$ and $Recv$ would give more performance. We have observed the similar behavior after execution of the kernel. The results are shown in table **??** and 1. It is also observed that the overlapping of computation and communication further improves the performance of 3D FFT when certain the number of 1D FFTs performed together that fully utilise the bandwidth of the network.

Proc	isenrec	isenrec_bl	alltoallv	alltoallv_bl	Auto (4c)	Auto (2c)	Auto (1c)
128	0.6175		0.5560		0.39052	0.2961	0.2653
256	0.3752	0.3292	0.3033	0.3013	0.23993	0.1798	0.16033
512	0.3417	0.1770	0.2007	0.1609	0.2057	0.1466	0.1222
1024	6.3045	0.1913	0.1992	0.0772	1.65		

Table 1. Strong scaling test for 3d FFT on Cray XT4. Time given in seconds for one forward and reverse 3d FFT on a 512^3 grid. isenrec corresponds to the implementation using isend and recv for the communications. isenrec_bl is the same as isenrec except that it performs many 3d FFTs at the same time (40 unless otherwise stated) and aggregates the messages for all of them. alltoallv uses the MPI alltoallv collective routine for the communications and alltoallv_bl is a blocked version of that performing 40 3d FFTs at the same time and aggregating the message sizes by 40. Auto corresponds to the implementation for auto tuning. All tests were done using all four cores except isenrec and Auto where 2c and 1c corresponds to running in dual core and single core mode. The Cray XT4 has quad core 2.3GHz Opteron processors, where each node is one quad core processor and the nodes are connected in a 3d torus.

5. Future Research and Challenges

In this paper we have present an efficient implementation of a parallel 3D FFT specifically designed for plane wave electronic structure code. The limiting factor to scaling to larger processor count is the communications in the 3D FFT. The overlapping of computation and communication give good performance over alltoall communication primitives on moderate processor counts. The limiting factor to scaling to larger processor counts is the communications in the 3d FFTs and we are investigating different communication schemes. In future we would like to use one sided communication. We will also develop a robust performance model which can be useful in prediction of performance of application on future systems. We will also have performance results on different parallel computing platforms in order to make the model very robust.

In future, we are interested in autotuning of the 3D FFT so that it would be useful on different parallel systems as well as future computing platforms. We are also interested in developing a 3D FFT kernel that can be used in other scientific applications and benchmarks in which 3D FFT is the important computational and time consuming kernel.

Achieving peak performance from the computational kernels that dominate application performance often requires extensive machine-dependent tuning by hand. Automatic tuning of an application on a system typically performed by (1) generating a large number of possible, reasonable implementations of a kernel which produces best performance on the given system, and (2) selecting the fastest implementation by a combination of various machine dependent parameters, heuristic modeling etc (i.e. actually running the code).

One major challenge is finding efficient ways to select implementations at run-time when several known implementations are available. Our aim has been to discuss a possible framework, using sampling and statistical classification, for attacking this problem in the context of automatic tuning systems as well as development of performance model to predict the performance of scientific applications on the upcoming parallel systems.

We are facing a paradigm shift to multicore and manycore computing, and this paradigm shift imposes difficult challenges on application developers and algorithm designers. Another challenge is, portability of this kernel on the future parallel system that will be based on multicore and manycore architecture.

Acknowledgements

This research 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.

References

[1] W. Kohn and L.J. Sham, Phys. Rev. **140**, A1133 (1965).

[2] M. Payne, M.P. Teter, D.C. Allan, T.A. Arias and J.D. Joannopoulos, Rev. Mod. Phys. **64**, 1045 (1992).

[3] PARATEC (PARAllel Total Energy Code) www.nersc.gov/projects/paratec/ by B. Pfrommer, D. Raczkowski, A. Canning, S.G. Louie, Lawrence Berkeley National Laboratory (with contributions from F. Mauri, M. Côté, Y. Yoon, C. Pickard and P. Haynes).

[4] Matteo Frigo and Steven G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE 93 (2), 216-231 (2005). http://www.fftw.org/
[5] A. Canning, L.W. Wang, A. Williamson and A. Zunger, J. of Comput. Phys. **160**, 29 (2000).
[6] V. Kumar, A. Grama, A. Gupta, G. Karypis "Introduction to Parallel Computing", Publisher: Benjamin Cummings.
[7] www.sdsc.edu/us/resources/p3dfft.php
[8] A. Canning, High Performance Computing for Computational Science, Springer, J.M.L.M Palma et. al. (Eds.) proceedings of VECPAR 2008, p280 (2008).
[9] http://www.spiral.net/