Ultrascalable Fourier Transforms in Three Dimensions

Dmitry Pekurovsky
San Diego Supercomputer Center
UCSD
Introduction: Fast Fourier Transforms

• Heavily used in many areas of computational science
• Computationally demanding
  • Not a cache-friendly algorithm
    • Memory bandwidth is stressed
  • Communication intense
    • All-to-all exchange is an expensive operation, stressing bisection bandwidth of the host’s network
• The goal: providing a scalable, portable, easy to use implementation of 3D FFT
Three-dimensional Fast Fourier Transform (3D FFT): the algorithm

• 1D FFT is applied three times (for X, Y and Z)
• Use transpose approach:
  • call FFT on local data only
  • transpose where necessary so as to arrange the data locally for the direction of the transform
  • It is more efficient to transpose the data once than to exchange the data multiple times during a distributed 1D FFT
• At each stage, there are many 1D FFT to do
  • Divide the work evenly
1D decomposition     2D decomposition

z
x
y

P_1
P_2
P_3
P_4

P_1
P_2
P_3
P_4

SAN DIEGO SUPERCOMPUTER CENTER
UNIVERSITY OF CALIFORNIA, SAN DIEGO
Algorithm scalability

• 1D decomposition: concurrency is limited to N (linear grid size).
  • Not enough parallelism for $O(10^4)$-$O(10^5)$ cores
  • This is the approach of most libraries to date (FFTW 3.2, PESSL)

• 2D decomposition: concurrency is up to $N^2$
  • Scaling to ultra-large core counts is possible
  • The answer to the petascale challenge
3D FFT algorithm with 2D decomposition

Image courtesy of H. Jagode (ORNL)
P3DFFT

- Open source library for efficient, highly scalable 3D FFT on parallel platforms
- Uses 2D decomposition
  - Includes 1D option.
- Available at [http://code.google.com/p/p3dfft](http://code.google.com/p/p3dfft)
- Historically grew out of an Advanced User Support Project
P3DFFT: features

- Implements real-to-complex (R2C) and complex-to-real (C2R) 3D transforms
- Fortran and C interfaces
- Performance-optimized
- Single or double precision
- Arbitrary dimensions
  - Handles many uneven cases ($N_i$ does not have to be a factor of $M_j$)
- Can do either in-place or out-of-place transform
- Includes example programs in Fortran and C
P3DFFT implementation

- Baseline version implemented in Fortran90 with MPI
- 1D FFT: call FFTW or ESSL
- Transpose implementation in 2D decomposition:
  - Set up 2D cartesian subcommunicators, using MPI_COMM_SPLIT (rows and columns)
  - Two transposes are needed: 1. in rows 2. in columns
- Baseline version: exchange data using MPI_Alltoall or MPI_Alltoallv
- MPI Datatypes?
- No benefit is seen
Communication performance

- A large portion of total time (up to 80%) is all-to-all
- Highly dependent on optimal implementation of MPI_Alltoall (varies with vendor)
- Message sizes are medium to large
  - Mostly sensitive to network bandwidth, not latency
- Buffers for exchange are close in size
  - Good load balance, predictable pattern
- Performance can be sensitive to choice of \((M_1, M_2)\)
Performance dependance on processor grid aspect ratio $M_1/M_2$
Communication scaling and networks

• All-to-all exchanges are directly affected by *bisection bandwidth* of the interconnect

• Increasing P decreases buffer size
  • Expect $1/P$ scaling on fat-trees and other networks with full bisection bandwidth (until buffer size gets below the latency threshold).

• On torus topology (Cray XT) bisection bandwidth is scaling as $P^{2/3}$
  • Expect $P^{-2/3}$ scaling

• Process mapping?
  • No benefit so far
Computation performance

• 1D FFT, three times:
  1. Stride-1
  2. Small stride
  3. Large stride (out of cache)

• Strategy:
  • Use an established library (ESSL, FFTW)
  • An option to keep data in original layout, or transpose so that the stride is always 1
    • The results are then laid out as (Z,Y,X) instead of (X,Y,Z)
    • Use loop blocking to optimize cache use
Strong scaling on Cray XT5 (Kraken) at NICS/ORNL

4096$^3$ grid, double precision, best $M_1/M_2$ combination
Weak Scaling (Kraken)

N^3 grid, double precision
2D vs. 1D decomposition

![Graph showing comparison between 2D and 1D decomposition in terms of time vs. number of cores (N_{cores}). The graph illustrates the decrease in time as the number of cores increases, with 2D decomposition showing a steeper decrease compared to 1D.](image)
Applications of P3DFFT

P3DFFT has already been applied in a number of codes, in science fields including the following:

- Turbulence
- Astrophysics
- Oceanography

Other potential areas include

- Material Science
- Chemistry
- Aerospace engineering
- X-ray crystallography
- Medicine
- Atmospheric science
DNS turbulence

- Direct Numerical Simulations (DNS) code from Georgia Tech (P.K. Yeung et al.) to simulate isotropic turbulence on a cubic periodic domain
- Characterized by disorderly, nonlinear fluctuations in 3D space and time that span a wide range of interacting scales
- DNS is an important tool for first-principles understanding of turbulence in great detail
- Vital for new concepts and models as well as improved engineering devices
- Areas of application include aeronautics, environment, combustion, meteorology, oceanography
- One of three Model Problems for NSF’s Track 1 solicitation
DNS algorithm

• It is crucial to simulate grids with high resolution to minimize discretization effects, and study a wide range of length scales.
• Uses Runge-Kutta 2\textsuperscript{nd} or 4\textsuperscript{th} order for time-stepping
• Uses pseudospectral method to solve Navier-Stokes eqs.
  • 3D FFT is the most time-consuming part
• 2D decomposition based on P3DFFT framework has been implemented.
DNS performance (Cray XT5)
P3DFFT - Ongoing work
Part 1: Interface and Flexibility

1. Expanding the memory layout options
2. Adding other types of transform (e.g. complex-to-complex, Chebyshev)
3. Adding ability to isolate transposes so the user can design their own transform
4. Adding ghost cell support/halo exchange
P3DFFT - Ongoing work
Part 2: Performance improvements

1. One-sided communication
   • MPI-2
   • OpenSHMEM
   • Co-Array Fortran

2. Communication/computation overlap – requires RDMA
   • Coarse-grain
   • Fine-grain

3. Hybrid MPI/OpenMP implementation
Coarse-grain overlap

• Suitable for computing several FFTs at once
  • Independent variables, e.g. velocity components
• Overlap communication stage of one variable with computation stage of another variable
• Advantage: uses large send buffers due to message aggregation
• Uses pairwise exchange algorithm based on MPI-2 or SHMEM
Coarse-grain overlap, results on Mellanox ConnectX-2 cluster (64 and 128 cores)

Coarse-grain overlap, preliminary results at large scale on Cray XE6 (Hopper), 16k cores
Hybrid MPI/OpenMP preliminary results (Kraken)

4096 nodes Kraken, 8 cores/node

\[ P = (\text{Thr} \times M_1) \times M_2 \]

- 1 Thread
- 2 Threads
Conclusions

• Efficient, scalable parallel 3D FFT library has been developed (open-source download available at http://code.google.com/p/p3dfft)

• Good performance is achieved on leading platforms

• Great potential for enabling petascale science

• An example of project that came out of a Teragrid Advanced User Support Collaboration, now benefiting a wider community
  • Incorporated into a number of codes (13 citations as of today, hundreds of downloads)
  • A candidate for XSEDE community code program?

• Work under way to expand capability and improve ultra-scale performance even further
  • Best options still investigated; platform-dependent

• An excellent testing tool for future platforms’ capabilities
  • Bisection bandwidth
  • MPI implementation
  • One-sided protocols implementation
  • MPI/OpenMP hybrid performance
Acknowledgements

• P.K. Yeung
• D.A. Donzis
• G. Chukkappalli
• J. Goebbert
• G. Brethouser
• N. Prigozhina
• K. Tomko
• K. Kandalla
• H. Subramoni
• S. Sur
• D. Panda

Work supported by Teragrid, NSF grants OCI-0850684 and CCF-0833155
Benchmarks run on Teragrid resources Ranger (TACC), Kraken (NICS), and DOE resources Jaguar (NCSS/ORNL), Hopper (NERSC)
DNS performance