CSSI Element: Software: Multidimensional Fast Fourier Transforms on the path to Exascale

PI: Dmitry Pekurovsky
Institution: UC San Diego

Award #: 1835885

Motivation

- Efficient Fourier Transforms and related algorithms (Spectral Transforms) are in high demand in computational science and key to many simulations at large scale
- Inherent properties of Spectral Transforms limit their adoption at Exascale
- Existing implementations also suffer from limited functionality, such as data structures and usability options

WHAT CAN BE DONE TO RECONCILE SPECTRAL TRANSFORMS AND EXASCALE?

WHAT CAN BE DONE TO INCREASE THE RANGE OF USE BEYOND TRADITIONAL DNS TURBULENCE CODES?

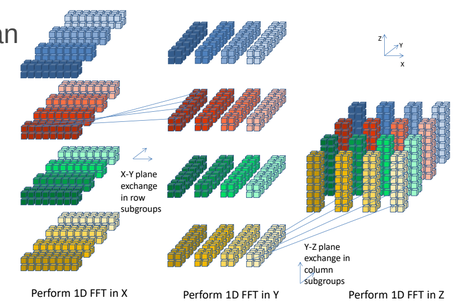
CAN PERFORMANCE AND FLEXIBILITY BE COMBINED?

Common Properties of Spectral Transforms in Multiple Dimensions

- Easily broken down into independent 1D transform components (example: 3D FFT)
- Each 1D component transform is non-local, i.e. uses heavily all data in given dimension
  - Best suited for node-local implementation, i.e. no inter-processor communication during the 1D transform
  - Algorithm typically reshuffles data among processors between consecutive 1D transforms
  - Limited by bisection bandwidth
  - Also limited by node memory bandwidth (see e.g. McClanahan
- **Recognized as a serious challenge for Exascale**

Existing work

- Many open source implementations available for 3D FFT (also many proprietary ones)
  - FFTW, FFTE, P3DFFT, OpenFFT, AccFFT, PFFT, NBFFT etc
- Each has areas of emphasis
- Traditionally focused on DNS turbulence style codes
- All have restricted data layouts
- Few support for overlapping communication
- Few support for GPUs
- Few support for pruned and/or sparse FFTs
- Few are optimal both in large and small grid range



3D FFT algorithm with 2D decomposition

Project Goals

1. Broader Impact through expanded functionality
2. Pushing performance towards Exascale

Expanding Functionality

- Spectral Transforms: a diverse ecosystem
  - Varied boundary conditions
  - Varied data types
  - Varied data layout
  - Varied processor decomposition
  - Special cases and potential for optimization:
    - Pruned/sparse transforms
    - Multivariable transforms
    - Higher-order operators: derivatives, convolutions, Laplacian etc
    - Large vs. small grid sizes

Performance Enhancements

Strategy 1: Reduce data volume in communication, memory access and computation, wherever possible.

Pruned Transforms: keep n < N Fourier modes. Used e.g. in dealiasing in CFD simulations.

Strategy 2: Combine multiple transforms into a single call

- Aggregate messages
- Overlap communication with computation

Strategy 3: Reduce number of memory reads/writes

- Reduce the number of local memory transpositions
- Optimize cache use for non-unit-stride reads/writes
- Reduce array copies
- Reuse cache by performing several operations in a single loop

Strategy 4: Utilize accelerators

- CUFFT/CUDA implementation

Strategy 5: Autotuning

P3DFFT++ Framework and design

- Open source package, with examples and documentation
- Distinct code from exiting P3DFFT package
- Object-oriented, modular design, hides platform and implementation details
- Written in C++, with C and Fortran interfaces
- Compute-intensive parts written in C-style code to optimize performance
- Highly flexible data layout
- Uses MPI, eventually combined with OpenMP and CUDA
- Supports multiple transform types (beyond FFT)
- Provides convenience functions for data manipulations
- (Work in progress) Autotuning to select fastest algorithm
- Similar to most libraries, offloads 1D transforms to a specialized library, such as FFTW

Features (current and future)

- Optimized for large core counts (1D, 2D and 3D decomposition)
- Flexible data layout
- Pruned/sparse transforms
- Multivariable transforms (with overlapping communication)
- GPU-enabled
- 3D and 4D transforms
- Autotuning for best algorithm
- Higher-order functions (derivative/Laplacian/convolution etc)

Some details of implementation

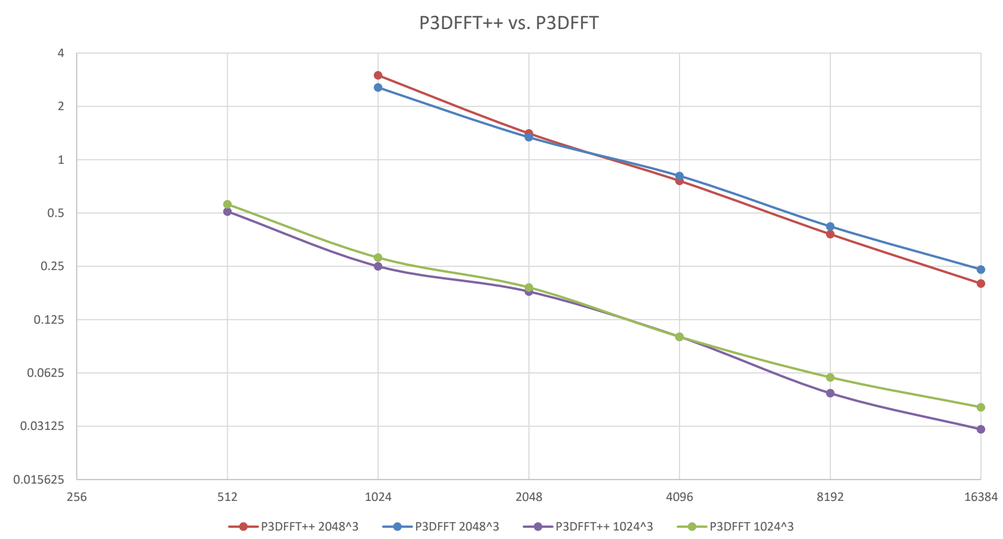
- Data types: real/complex, single/double precision
- Functions and classes expressed as C++ templates, with interfaces for C and Fortran
- Low level functions: combinations of 1D transform/derivative, local transposition, interprocessor transposition – accessible to the user.
- 1D transform types: designed as general as possible, with FFTW currently reference implementation.
- Higher level functions: 3D transforms (future: 4D transforms, correlation functions, Laplacian etc)
- Execution model: planner and execution functions, similar to FFTW

Data layout specification

Data descriptor `grid`:

- Data grid size and properties
- Processor grid definition (1D, 2D, 3D)
- Local memory layout (order of dimension storage)
  - E.g. (x,y,z), (z,y,x) etc
- Mapping of data grid onto processor grid
  - Example: Pgrid = P_1 x P_2 = 4 x 8. Data grid 32^3. Map Y dimension onto P_1, Z dimension onto P_2. Local dimensions: 32 x 8 x 4. Can be stored as array of A[4][8][32] (default layout (0,1,2)), or A[32][8][4] (transposed layout (2,1,0)), or …?

Preliminary performance study (Stampede2 @ TACC)



Conclusions and Future Work

- A new open source package P3DFFT++ aims to bridge high performance with ease of use and increased use range.
- New object-oriented design
- Preliminary performance on par with P3DFFT and similar libraries.
- Future work
  - GPU/CUDA
  - Pruned transforms
  - Overlap of communication
  - Higher order functions/combinations
  - 4D transforms