P3DFFT

Dmitry Pekurovsky
San Diego Supercomputer Center
UC San Diego
dmitry@sdsc.edu

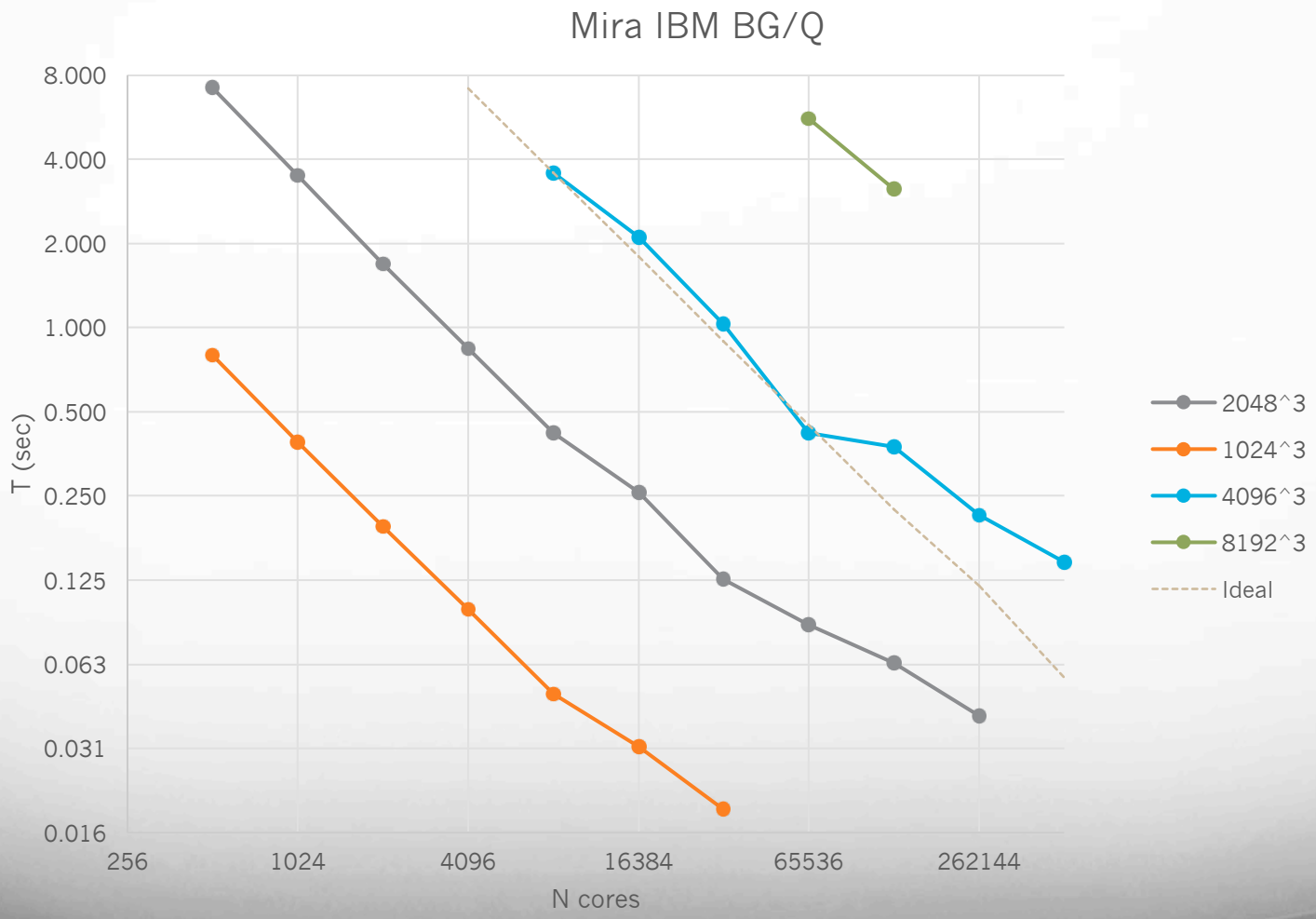
Web site: http://www.p3dfft.net
Github: https://github.com/sdsc/p3dfft

About P3DFFT

- Originally created in 2006, implementing real-to-complex 3D FFT.
  - Defining feature is the use of 2D domain decomposition
  - One of first open source libraries of this kind
  - Emphasis is on high performance and scalability
  - Written in Fortran 90/MPI, employing ESSL or FFTW for 1D FFT.
  - Single or double precision.
- Subsequently expanded to include:
  - Pruned transforms (incomplete set of frequencies): good for dealiasing in pseudospectral algorithms etc
  - Multivariable (batched) transforms. Overlap version.
  - MPI/OpenMP
  - Chebyshev transforms etc

P3DFFT current version 2.7.6

- Open source, GPL3 license
- Convenient build (autoconf package)
- Extensive documentation
- Example programs for a variety of use scenarios in Fortran an C
- Mailing list
- Github, regular pull requests
- Over 250 Citations
- Tested on many high-end platforms (Cray XC30, IBM BG/Q, IBM Power, Intel/ Infiniband)


Mira IBM BG/Q
8.000
4.000
2.000
1.000
0.500
0.250
0.125
0.063
0.031
0.016
T (sec)
256
1024
4096
16384
65536
262144
N cores
2048^3
1024^3
4096^3
8192^3
Ideal

P3DFFT++: Next generation library in the works

- Rationale: P3DFFT so far has predefined data structure that may not suit everyone. X-pencil: (x,y,z) ordering, Z-pencil: (x,y,z) or (z,y,x) ordering. Can only do 3D transforms between these two.
- Goal: expand the feature list while maintaining performance
- P3DFFT++: a while lot more than an FFT library
  - Modular structure, very flexible interface. Permits extensibility.
  - Written in C++, interfaces to C and Fortran.
  - Features:
    - 1D, 2D or 3D transforms and transposes
    - Very general assumptions about data layout
    - 1D, 2D or 3D domain decomposition
    - Expanded set of operators besides FFT: compact scheme; user-defined etc

Beta version available for testing by contacting dmitry@sdsc.edu

Questions

1) Why did you write your own FFT?

- Clear need to go from 1D to 2D decomposition to include larger grid sizes in simulations.
- No other open packages that could do 3D FFT with 2D domain decomposition at the time

2) What considerations are important for you in an FFT implementation?

Performance, scalability, ease of use, portability, extensibility

3) What might you look for if there were to be a unified FFT interface (similar to BLAS, LAPACK and SCALAPACK interfaces)?

Flexibility, portability, extensibility. Interfaces with C, C++, Fortran.

4) How important are performance, portability, and scalability for you?

Very important

Questions cont.

5) Will FFT be needed in exascale computing and if so how will it be achieved?

There is a high demand for FFT at the largest scale. It is one of the hardest algorithms to scale. Critically dependent on 1) bisection bandwidth and 2) memory bandwidth.

Possible solutions:

- Overlapping communication with computation
- Using a new generation of GPUs or FPGA's (still need to overcome the bandwidth bottleneck).
- Clever algorithms: pruned/sparse FFT, multivariable calls.
- Using lower precision whenever possible.

Questions cont.

6) What would be a good FFT benchmark or a good way to include the FFT in a high-performance computer benchmark?

Real-to-complex forward and back Fourier transform of a contiguous 3D array, single and batched (multiple transforms together).

Acknowledgements

Work supported by National Science foundation, grants OCI-0850684 and ACI-1339884, as well as XSEDE (The Extreme Science and Engineering Discovery Environment)

The author acknowledges Department of Energy centers Argonne and NERSC for use of compute resources