CHEMNITZ UNIVERSITY OF TECHNOLOGY 175 Years

Highly Scalable Fast Fourier Transforms

TU Chemnitz
Faculty of Mathematics
Applied Functional Analysis

Michael Pippig and Daniel Potts

SPONSORED BY THE Federal Ministry of Education and Research

Fast Fourier Transform

Consider a three-dimensional dataset of $n_0 \times n_1 \times n_2$ complex numbers $g_{k_0k_1k_2} \in \mathbb{C}$, $k_s = 0, \ldots, n_s - 1$ for all $s = 0, 1, 2$. With the definition of the roots of unity $\omega_s := \mathrm{e}^{+2\pi\mathrm{i}/n_s}$ for all $s = 0, 1, 2$ we can write the three-dimensional forward discrete Fourier transform (DFT) as

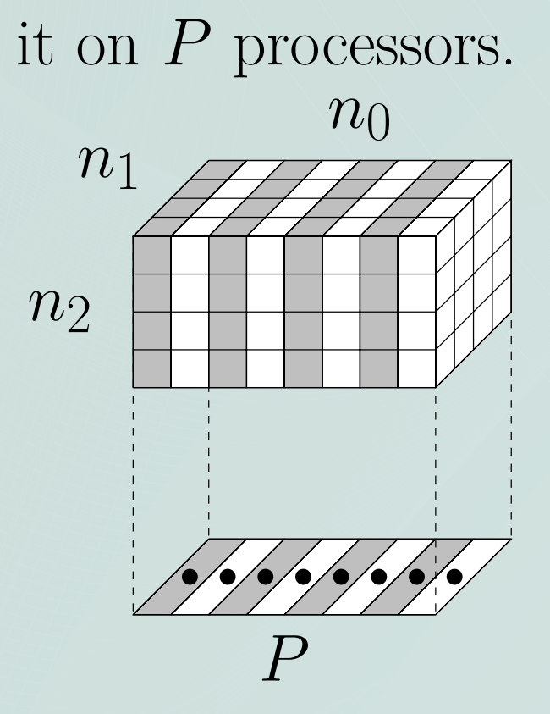
$$\hat{g}_{l_0l_1l_2} := \sum_{k_0=0}^{n_0-1} \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} g_{k_0k_1k_2}\, \omega_2^{-l_2k_2} \omega_1^{-l_1k_1} \omega_0^{-l_0k_0} \in \mathbb{C},$$

where $l_s = 0, \ldots, n_s - 1$ for all $s = 0, 1, 2$. It is well known, that a multi-dimensional DFT can be calculated efficiently by successive one-dimensional fast Fourier transforms (FFTs). The Fast Fourier transform plays a key role in scientific computing and has applications in nearly all fields of scientific research. Since the trend in computational science is to build parallel hardware architectures with billions of cores it is desirable to develop parallel FFT algorithms that benefit from this huge counts of cores. Focusing on the so-called transpose algorithm of three-dimensional FFTs, there are two strategies for parallelization.

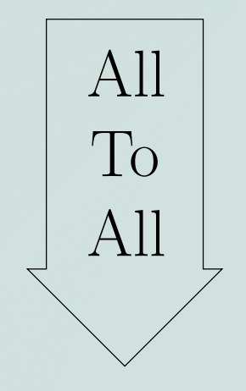
One-Dimensional Data Distribution

First parallel transpose algorithms were based on so-called slab decomposition, which means that the multi-dimensional data is split along one dimension to distribute it on P processors.

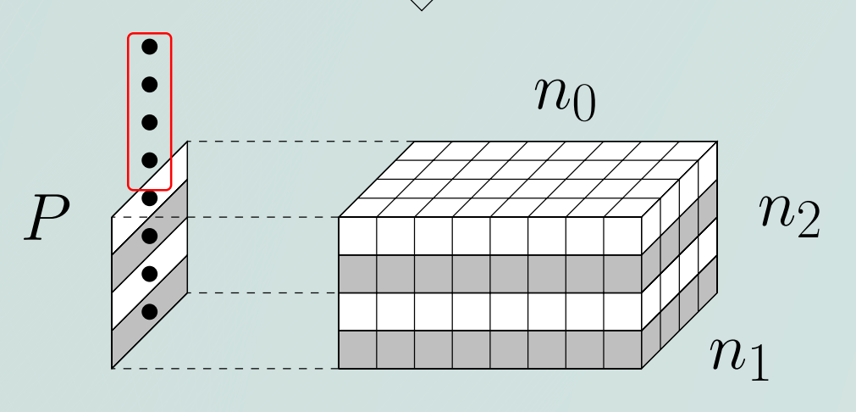
At the first step n_0 two-dimensional FFTs of size $n_1 \times n_2$ are calculated in parallel along the local available slices of the dataset. This can be performed efficiently with a standard serial FFT software library.



Next an all-to-all communication with all P processors is necessary to get the last dimension of the dataset local to the processors.



Finally the remaining $n_1 \times n_2$ one-dimensional FFTs of size n_0 can be computed. The red marked processors can not take part at this step since $n_2 < P$.



The main drawback of this algorithm is its limited scalability to at most

$$P_{\max}^{\mathrm{1D}} = n_0 + n_1 + n_2 - \max\{n_0, n_1, n_2\} - \min\{n_0, n_1, n_2\}$$

processors. Implementations are for example the IBM PESSL FFT [3], the Intel Math Kernel Library [6] and FFTW [5] by Matteo Frigo and Steven G. Johnson.

FFTW

The open source software library FFTW [4] is well known for its high performance FFT algorithms. Because of its easy to use interface and the concept of self hardware adapting codelets it is the FFT library of choice whenever good portability should be combined with high performance. Nevertheless the parallel FFTW based on MPI uses the one-dimensional data decomposition, which is not appropriate for large core counts.

FFTW Interface

PFFT

It naturally turns up to ask for a public available library, that unifies the advantages of FFTWs superior user interface and the highly scalable two-dimensional data decomposition approach. The key idea is to use the two-dimensional data transposition algorithms implemented in FFTWs parallel FFT library for three-dimensional remappings. There are some great advantages of using FFTWs parallel transposition algorithms instead of direct calls to corresponding MPI functions. FFTW does not only use one algorithm to perform array transpositions. Similar to the planing of FFTs different algorithms are compared to get the fastest one. This provides us with portable self hardware tuned communications. Furthermore all transpositions can be performed in place, which is impossible by MPIs standard all-to-all calls and hard to program in an efficient way with point to point communications.

The PFFT software library brings together the main aspects of modern high performance computing, namely easy usability, good portability, self hardware adaption and high scalability.

2D Data Decomposition

Features of PFFT

The extensive use of FFTWs well developed algorithms allows us straight forward transfer of many valuable features to our library. These are for example

- inplace FFTs and communications
- high performance algorithms
- self tuning ability for different hardware architectures
- support of planning flags
- generalization to d-dimensional parallel FFTs ($d \geq 3$)
- real-to-complex and complex-to-complex FFTs
- user friendly interface
- open source library
- good portability

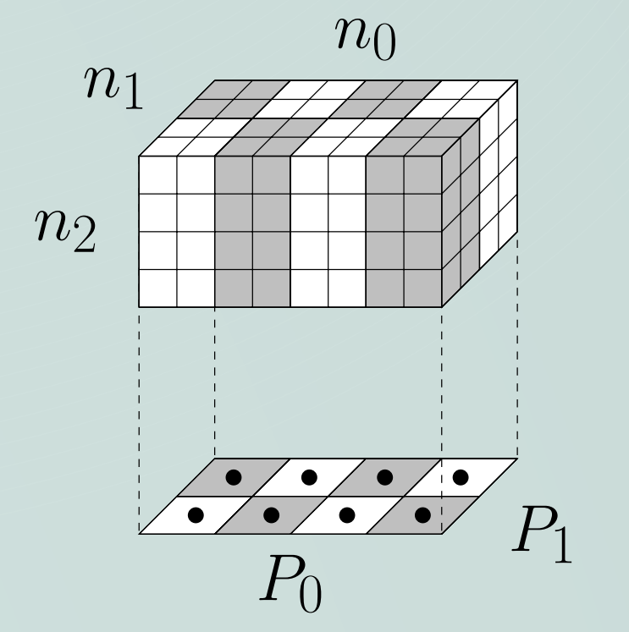
In addition we implemented the following features to increase performance and easy usability

- two-dimensional data decomposition
- ghost cell support
- truncated FFTs

Two-Dimensional Data Distribution

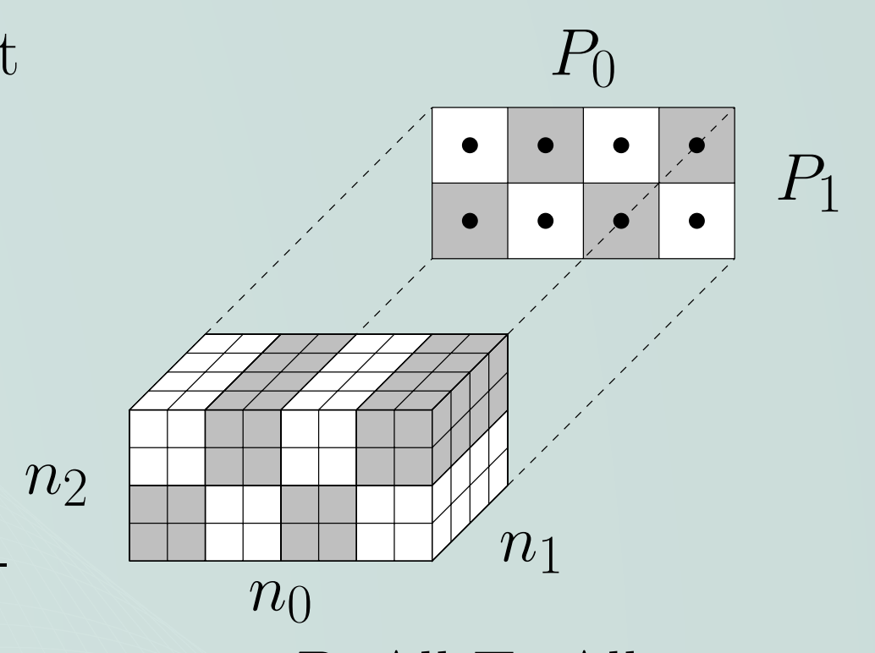
Eleftheriou et al. [2] proposed a volumetric domain decomposition to overcome the scalability bottleneck and implemented a software library [1] for power of two FFTs customized to Blue-Gene/L systems. They split the dataset along two dimensions and therefore were able to increase the number of processors quadratically.

At the first step $n_0 \times n_1$ one-dimensional FFTs of size n_2 are calculated in parallel by standard serial FFT software libraries.

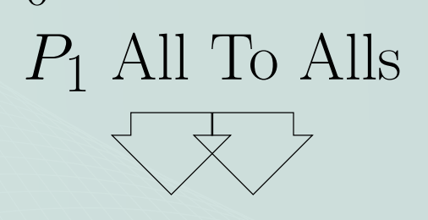


Next P_0 all-to-all communications are performed on processor subgroups of size P_1. Only processors within the same column of the processor mesh must communicate.

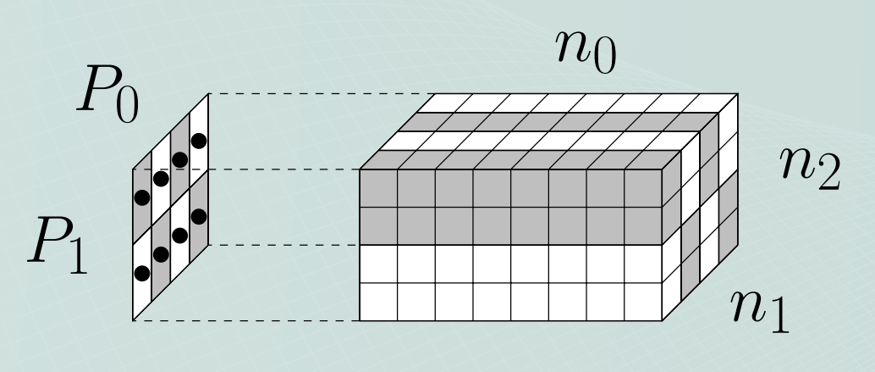
Now the one-dimensional FFTs along the second dimension of the dataset can be computed in parallel.



The next P_1 all-to-all communications only include processors within the same row of the processor mesh, namely P_0.



Finally the one-dimensional FFTs along the last dimension of the dataset can be computed in parallel.



The scalability of this algorithm is limited to at most

$$P_{\max}^{\mathrm{2D}} = \frac{n_0 n_1 n_2}{\max\{n_0, n_1, n_2\}}$$

processors.

Implementations

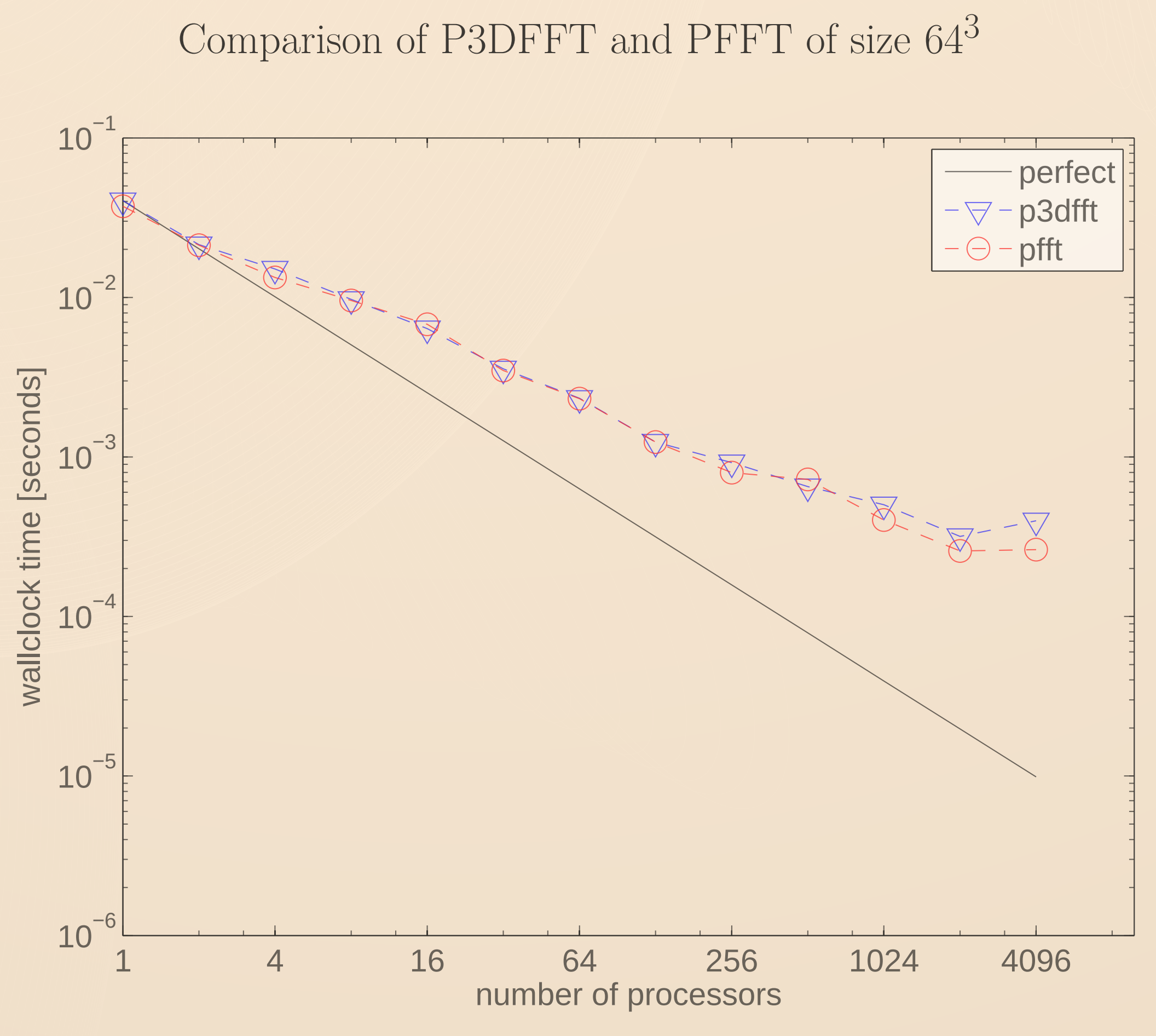
Portable implementations based on the two-dimensional data decomposition approach are the FFT package by Steve Plimpton (SandiaFFT) [8] from Sandia National Laboratories and the three-dimensional FFT library by Dmitry Pekurovsky (P3DFFT) [7]. While SandiaFFT is less restrictive on data distributions, runtime tests proved P3DFFT to perform better in most cases.

Scalability Comparison

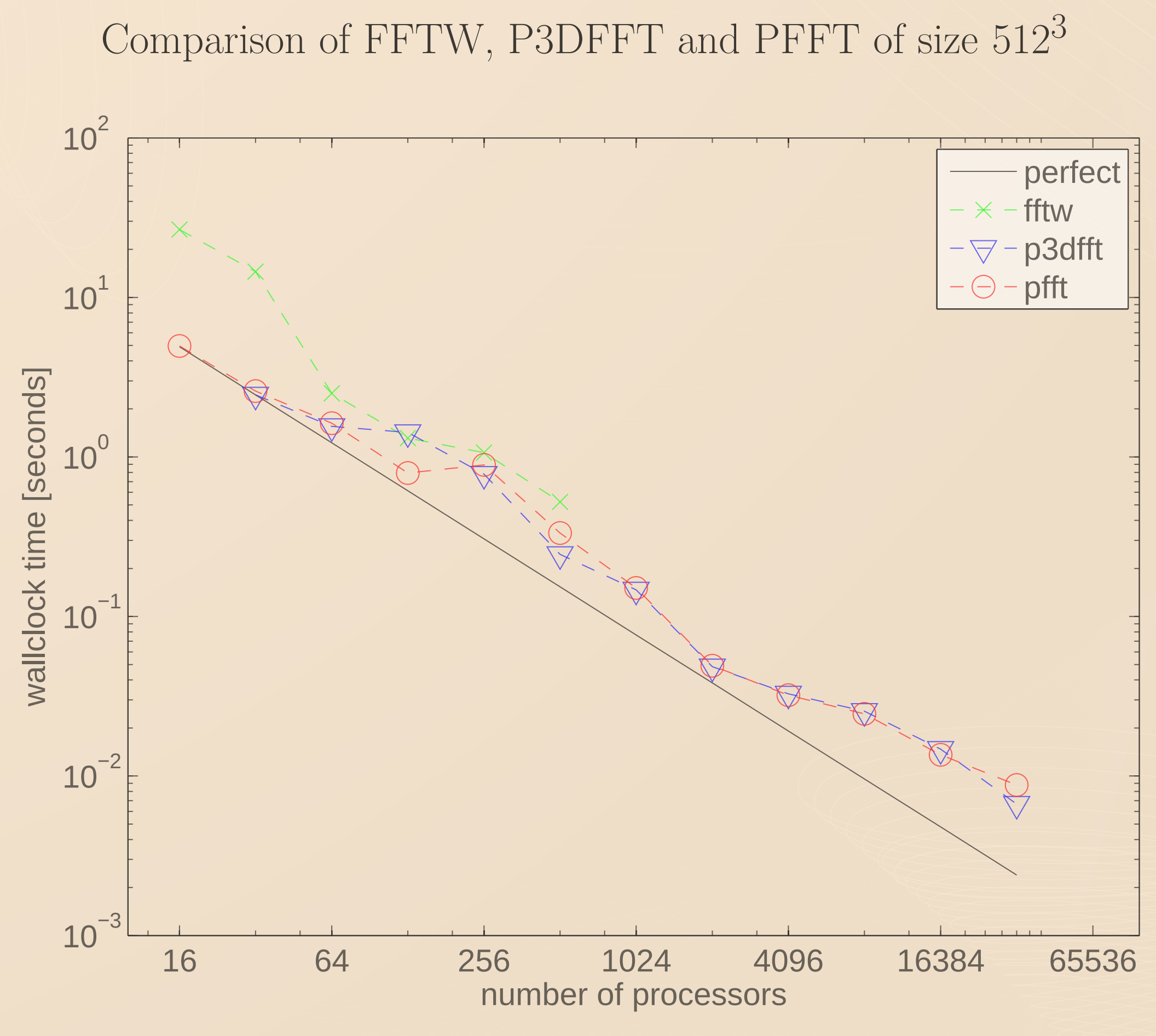
The table beside compares the maximum number of usable processors for a three-dimensional FFT of size n^3 with the two data decomposition algorithms. Note that the BlueGene/P supercomputer Jugene in Jülich already has 294912 cores.

n	$P_{\max}^{\mathrm{1D}} = n$	$P_{\max}^{\mathrm{2D}} = n^2$
64	64	4096
128	128	16384
256	256	65536
512	512	262144
1024	1024	1048576

Comparison of P3DFFT and PFFT of size 64^3



Comparison of FFTW, P3DFFT and PFFT of size 512^3



References

[1] M. Eleftheriou, J.E. Moreira, B.G. Fitch, and R.S. Germain: *Parallel FFT subroutine library.* `http://www.alphaworks.ibm.com/tech/bgl3dfft`.

[2] M. Eleftheriou, J.E. Moreira, B.G. Fitch, and R.S. Germain: *A volumetric FFT for BlueGene/L.* In T.M. Pinkston and V.K. Prasanna (eds.): *HiPC*, vol. 2913 of *Lecture Notes in Computer Science*, pp. 194–203. Springer, 2003.

[3] S. Filippone: *The IBM parallel engineering and scientific subroutine library.* In J. Dongarra, K. Madsen, and J. Wasniewski (eds.): *PARA*, vol. 1041 of *Lecture Notes in Computer Science*, pp. 199–206. Springer, 1995.

[4] M. Frigo and S.G. Johnson: *FFTW, C subroutine library.* `http://www.fftw.org`.

[5] M. Frigo and S.G. Johnson: *The design and implementation of FFTW3.* Proceedings of the IEEE, 93:216–231, 2005.

[6] Intel Corporation: *Intel math kernel library.* `http://software.intel.com/en-us/intel-mkl/`.

[7] D. Pekurovsky: *P3DFFT, C subroutine library.* `http://www.sdsc.edu/us/resources/p3dfft`.

[8] S. Plimpton: *Parallel FFT subroutine library.* `http://www.sandia.gov/~sjplimp/docs/fft/README.html`.

Contact

Michael Pippig `michael.pippig@mathematik.tu-chemnitz.de`
`http://www.tu-chemnitz.de/~mpip`
Daniel Potts `potts@mathematik.tu-chemnitz.de`
`http://www.tu-chemnitz.de/~potts`

Applied Functional Analysis
Faculty of Mathematics
Chemnitz University of Technology
D – 09107 Chemnitz