Parallelisation of the 3D Fast Fourier Transform Using the Hybrid OpenMP/MPI Decomposition

Vojtech Nikl(✉) and Jiri Jaros

Faculty of Information Technology, Brno University of Technology,
Bozetechova 2, 612 66 Brno, Czech Republic
{inikl,jarosjir}@fit.vutbr.cz

Abstract. The 3D fast Fourier transform (FFT) is the heart of many simulation methods. Although the efficient parallelisation of the FFT has been deeply studied over last few decades, many researchers only focused on either pure message passing (MPI) or shared memory (OpenMP) implementations. Unfortunately, pure MPI approaches cannot exploit the shared memory within the cluster node and the OpenMP cannot scale over multiple nodes.

This paper proposes a 2D hybrid decomposition of the 3D FFT where the domain is decomposed over the first axis by means of MPI while over the second axis by means of OpenMP. The performance of the proposed method is thoroughly compared with the state of the art libraries (FFTW, PFFT, P3DFFT) on three supercomputer systems with up to 16k cores. The experimental results show that the hybrid implementation offers 10-20% higher performance and better scaling especially for high core counts.

1 Introduction

The fast Fourier transform (FFT)[1] is the heart of many spectral simulation methods where it is used to calculate spatial gradients of various physical quantities. This approach eliminates the numerical dispersion that arises from the discretisation of the spatial derivative operators, and significantly reduces the grid density required for accurate simulations [2].

A recent application of spectral methods, we have been working on, is the k-Wave toolbox [3] oriented on the full-wave simulation of the ultrasound waves propagation in biological materials (both soft and hard tissues) intended for ultrasound treatment planning such as cancer treatment, neurostimulation, diagnostics, and many other. In many realistic simulations with domain sizes ranging from 512^3 to 4096^3, as much as 60% of the total computational time is attributed to the 3D FFTs. Reducing the 3D FFT compute time thus remains a challenge even in the petascale era [4].

Many libraries have been developed to compute the FFT in the massively parallel distributed memory environment, such as FFTW (Fastest Fourier Transform from West)[5], PFFT (Parallel FFT)[6] and P3DFFT (Parallel Three-Dimensional Fast Fourier Transforms)[7]. All of these libraries use the

P. Hliněný et al. (Eds.): MEMICS 2014, LNCS 8934, pp. 100–112, 2014.
DOI: 10.1007/978-3-319-14896-0_9

pure-MPI message passing approach to calculate the FFT in parallel. However, modern high-performance computer architectures usually consist of a hybrid of the shared and distributed paradigms: distributed networks of multicore processors. The hybrid paradigm marries the high bandwidth low-latency interprocess communication featured by shared memory systems with the massive scalability afforded by distributed computing.

In this work, we describe recent efforts to exploit modern hybrid architectures, using the popular MPI interface to communicate among distributed nodes and the OpenMP multi-threading paradigm to communicate among the individual cores of each processor to speed up the calculation of 3D Fast Fourier Transform. Moreover, we introduce a novel hybrid 2D pencil decomposition that allows us to employ more compute cores than the standard 1D slab decomposition implemented in the FFTW while keeping the communication burden significantly lower compared to PFFT and P3DFFT also based on pencil decompositions.

2 Parallel Implementations of the 3D Fast Fourier Transform

There are two main approaches for parallelising multidimensional FFTs; the first is binary exchange algorithms, and the second is transpose algorithms. An introduction and theoretical comparison can be found in [8]. In this paper, we restrict ourselves to transpose algorithms that need much less data to be exchanged [9] and have direct support in many software libraries, e.g. FFTW [5].

Regardless of decomposition, a Fourier transform in three dimensions is comprised of three 1D FFTs in the three dimensions (X, Y, and Z) in turn. When all of the data in a given dimension of the grid resides entirely in a processors memory (i.e., it is local) the transform consists of a 1D FFT done over multiple grid lines by every processor, which can be accomplished by a serial algorithm provided by many well-known FFT libraries and is usually a fairly fast operation. The transforms proceed independently on each processor with regard to its own assigned portion of the array. When the data are divided across processor boundaries (i.e., nonlocal), the array is reorganized by a single step of global transposition so that the dimension to be transformed becomes local, and then serial 1D FFT can be applied again. These global transpositions are known to be the main bottleneck of the 3D FFT since arithmetic intensity (computational work divided by communication work) grows only as a factor of $logN$ [5], [6], [7].

A general algorithm to calculate a distributed 3D FFT of size $Z \times Y \times X$ stored in C-like row major order follows this procedure:

1. Perform $Z \times Y$ one-dimensional FFTs along the X axis.
2. Perform $X \leftrightarrow Y$ data transposition.
3. Perform $Z \times X$ one-dimensional FFTs along the Y axis.
4. Perform $Z \leftrightarrow X$ data transposition.
5. Perform $Y \times X$ one-dimensional FFTs along the Z axis.
6. Transpose data back into the original order (optional).

2.1 Decomposition of the 3D Fast Fourier Transform

Solving the 3D FFT in parallel requires the compute grid to be partitioned and distrusted over processing cores. In the case of 3D FFT, there are three possible ways how to partition the grid; one-dimensional slab decomposition, two-dimensional pencil decomposition, and three-dimensional cube decomposition (see Fig. 1).

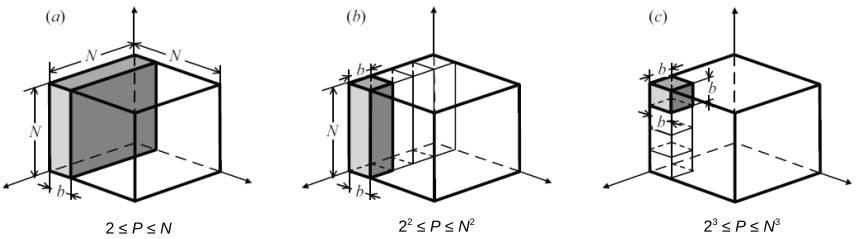

Fig. 1. Domain decompositions for three-dimensional grid over P processing cores. (a) slab decomposition, (b) pencil decomposition, (c) cube decomposition [10]. Data associated with a single processing core is shaded.

Most of the parallel 3D FFT libraries to date use the slab domain decomposition over the first dimension (Z in our case) [5], [11]. This decomposition is faster on a limited number of cores because it only needs one global transpose, minimizing communication. The main disadvantage of this approach is that the maximum parallelisation is limited by the largest size along an axis of the 3D data array used. At the age of petascale platforms more and more systems typically have numbers of processing cores far exceeding this limit. For example, cutting edge ultrasound simulations performed by the k-Wave toolbox [3] use 2048^3 grids and so with the slab decomposition would scale only to 2048 cores at most leading to the calculation time exceeding clinically acceptable time of 24 hours (here between 50 and 100 hours).

The second approach is the 2D pencil decomposition that further partitions the slabs into a set of pencils, see Fig. 1(b). This approach has recently been implemented in two novel FFT libraries PFFT[6] and P3DFFT[7]. Although this approach increases the maximum number of processor cores from N to N^2, it also requires another global communication. Nevertheless, these global transposition steps require communication only among subgroups of all compute cores. However according to Pekurovsky [7], attention must be paid to the pencil placement over the computing cores to keep good locality and efficacy.

The cube decomposition studied in [10] brings the highest scalability, however it requires one-dimensional FFTs to be calculated non-locally and thus fine-tuned FFT cores provided by FFTW cannot be used.

The parallel 3D FFT is usually implemented using a pure-MPI approach and one of the described decomposition techniques. However, many current supercomputers comprise of shared memory nodes typically integrating 16 cores. The use of shared memory significantly reduces the amount of inter-process communication and helps in exploiting local caches. The most sensible implementation of the hybrid decomposition bases on the pencil decomposition where a slab is assigned per compute node, and the cores within node take each their portion of pencils. One of the obvious advantages of exploiting hybrid parallelism is the reduction in communication since messages no longer have to be passed between threads sharing a common memory pool. Another advantage is that some algorithms can be formulated, through a combination of memory striding and vectorization, so that local transposition is not required within a single MPI node (while this is even possible for multi-dimensional FFTs, the recent availability of serial cache-oblivious in-place transposition algorithms appears to have tipped the balance in favour of doing a local transpose). The hybrid approach also allows smaller problems to be distributed over a large number of cores. This is particularly advantageous for 3D FFTs: the reduced number of MPI processes allows for a more slab-like than pencil.

Some authors object that this approach does not push the scaling significantly far [7]. However, for the grid of practical interest (1024^3 - 4096^3), the number of cores that can be employed lies between 16384 and 65536. These numbers of cores can only offer largest supercomputers in Europe accessible via the PRACE Tier-0 allocation scheme[1]. As the trend of integrating more cores within a node is going to continue, we consider the scaling to be good enough from the practical point of view. Although pure-MPI implementation may allow us to distribute the work over much more compute cores, the efficiency is then still very low anyway (less than 6% for 100k and more cores as presented in [6]).

2.2 Libraries for Distributed FFT

This section provides an overview of the most popular libraries for calculating the 3D FFT using both the slab and pencil decomposition and serves as a firm background for experimental comparison.

The Fastest Fourier Transform in the West (FFTW)[5] is probably the most popular library for calculating n-dimensional FFT over an arbitrary input size grid and still reaching the $NlogN$ time complexity. FFTW uses the so called *plan and execute* approach to select the most suitable implementation of FFT for the underlying hardware. This allows FFTW to be easily portable and still extremely fast. The FFTW supports both multi-threaded and memory distributed architectures. In case of distributed memory environment, the grid is decomposed using the slab decomposition. This feature is considered to be a significant drawback nowadays. Fortunately, FFTW allows to combine multi-threaded FFT kernels with custom grid decomposition and data exchange and is thus often used as

[1] PRACE: Partnership for Advanced Computing in Europe, http://www.prace-ri.eu

basis for advanced implementations (some of them are discussed later in this section).

The Parallel FFT library (PFFT) proposed by Michael Pippig [6] is one of a few FFT implementation using the pencil decomposition, unfortunately it is still in an alpha version. It builds on serial FFTW kernels applied on one-dimensional FFT and custom data exchange around the pure-MPI approach. PFFT has been tested on a BlueGene/P machine employing up to 256k PowerPC cores. However, the scaling with increasing number of cores becomes flat reaching only 6% for 256k cores.

The last library we took into account is the Parallel three-dimensional FFT (P3DFFT) by Dimitry Pekurovsky [7]. This library is specialised on calculating the 3D FFT using the pencil decomposition and the pure-MPI approach. The library employs one-dimensional kernels provided by FFTW or IBM ESSL[2]. This implementation allows to collapse the pencil decomposition into the slab one for low core counts preserving good efficacy. The implementation shows good performance for moderate core counts up to 65k. One of the main obstacles for us is the implementation language being Fortran and the support for only real-to-complex and complex-to-real transforms.

3 Proposed Method

The proposed implementation of the distributed hybrid OpenMP/MPI 3D FFT is called HyFFT. It is based on the modified pencil decomposition built on the top of the FFTW library. The 3D grid is first decomposed by MPI processes into slabs. The slabs are further partitioned into pencils assigned to threads to demand. This ensures the entire slab being always stored within the shared memory leading to the first transposition being local. In the corner case of small grids where the number of slabs is smaller than the number of cores, the decomposition naturally collapses into the original 1D slab decomposition and the pure-MPI implementation.

Exploiting full potential of modern clusters with multicore/multisocket nodes introduces some restrictions on the process/threads placement on nodes, sockets and cores. In the case of dual-socket x86 clusters, the best is usually to run a separate process per socket and spawn as many threads as cores per socket. This yields the advantage of the slab being stored in the socket's local memory with the fastest access. If a higher number of threads (higher scaling) is required, a single process per node can be run, instead. However, this implies the slab to be split over two memory islands leading in the non uniform memory access (NUMA) slowing down the local transposition. The situation is similar in the case of IBM PowerPC architectures, though the best is to spawn two threads per core to fully exploit all its HW resources.

The proposed HyFFT follows the diagram shown in Fig. 2. We can clearly see three series of 1D FFTs interleaved with local and global transpositions.

[2] http://www-03.ibm.com/systems/power/software/essl/

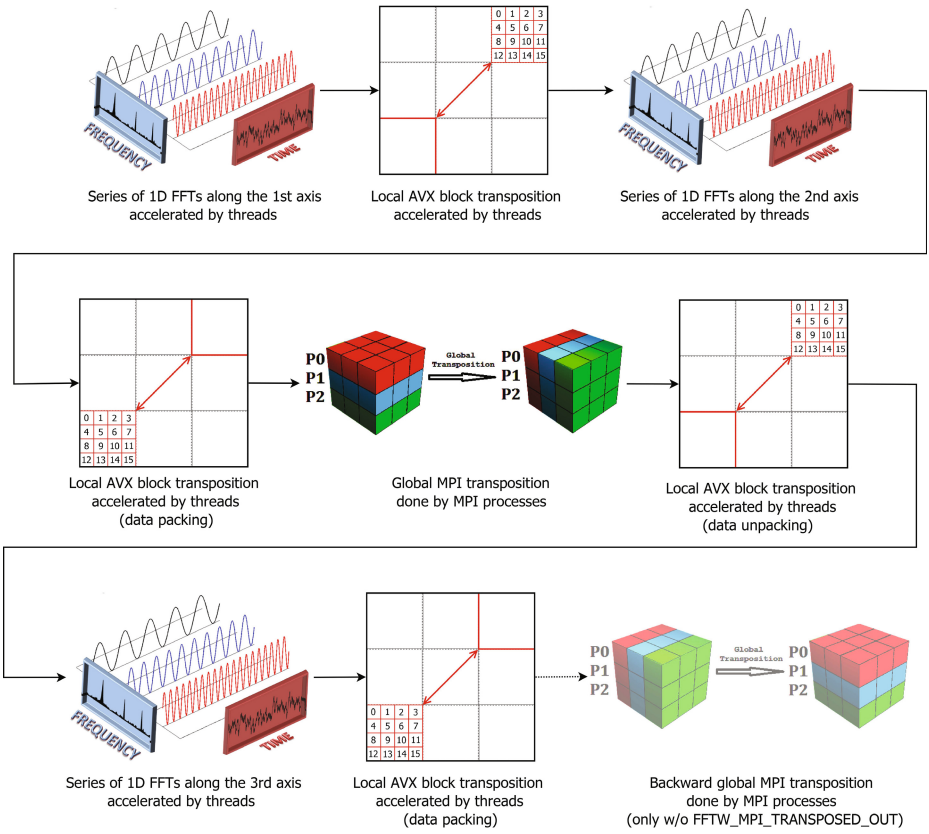

Fig. 2. The steps of HyFFT to be carried out to perform a forward 3D FFT

The first local transpositions rearranges data within a slab before the second FFT transform. The global transposition is wrapped by data packing and unpacking steps carried out as local transpositions. The last FFT transform is followed by a local transposition to get the output data compatible with the FFTW library under `FFTW_MPI_TRANSPOSED_OUT` flag omitting the second global transposition for the sake of performance. However, if the same shape of the grid is required after the 3D FFT, the global transpose has to be performed.

The calculation itself comprises of three main kernels as outlined in Section 2: series of 1D FFTs, local transpositions and a global distributed transposition:

1. **FFT kernels**: There are two different ways how to calculate FFTs over the slab in the shared memory. The one primarily used in this work distributes the pencils over the threads using OpenMP pragmas, calculates 1D FFTs in parallel using 1D FFTW kernels, performs the local transposition and continues over the second axis. If there are more pencils in the slab than threads, every thread is responsible for a bunch of pencils. These can be calculated

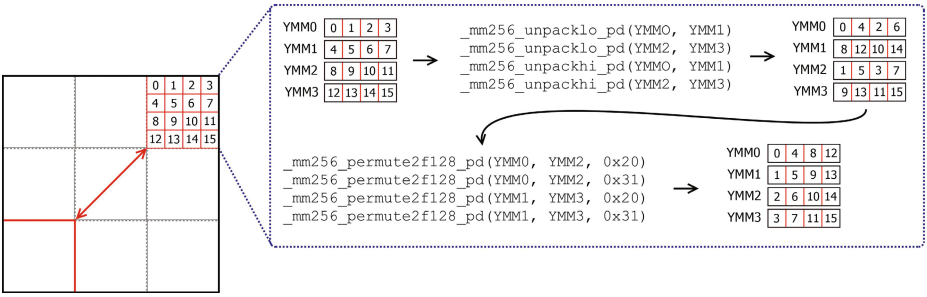

Fig. 3. The block based local transposition using the Intel AVX vector intrinsics

one by one (our approach) or simultaneously. Calculating a bunch of pencils sequentially is preferred for larger grid sizes due to a better utilisation of L1 cache (e.g. a complex single precision pencil of 1024 grid points occupies 8KB - one half of L1 cache) and because of only having a single implementation of the FFT kernel for all three calculation phases.

The second approach to calculate the FFT over the slab is a use a multi-threaded 2D FFT provided by FFTW instead of the doing the sequence of 1D FFTs, local transposition and 1D FFTs. This can increase the performance by a few percent in specific cases although it does not support multi-threaded transposition. That is why it is always considered by HyFFT as an alternative to the previous approach.

2. **Local transposition**: The local transposition is based on a multi-threaded, cache-friendly algorithm further accelerated by vector units (see in Fig. 3). The slab is first chopped into square blocks that can fit nicely into L1 or L2 cache. Threads then take pairs of blocks sitting symmetrically over the main diagonal, transpose the data inside and finally swap each other. In the case of square slabs, this can be done in-place. However, rectangular slabs enforce an out-of-place algorithm.

   The block being transposed is further divided into tiles of size 2x2 or 4x4 complex numbers depending on whether the SSE or AVX vector instruction set is available. A fast, vector register based, kernel is used to permute the grid elements within the tile yielding the transposed order. Since we work with single precision floating point numbers only, complex single precision values can be treated as double precision real ones leading in fewer instruction needed. In the case the size of the slab is not divisible by the size of the vector registers (2 or 4 for SSE and AVX, respectively), the reminders are treated separately using scalar kernels.
3. **Global transposition**: The distributed transposition getting the grid points over the last axis (Z) contiguous is done by a composition of two local transpositions and a global one. The FFTW library offers a fine-tuned routine to exchange data amongst the processes that is supposed to be faster than simple `MPI_Alltoall`. Let us note that this operation is performed only by the master thread (a single core per socket or node).

4 Experimental Results

Experiments were performed on 3 different clusters - Zapat[3], Anselm[4] and Fermi[5]. The performance and scaling were investigated on grid sizes ranging from 256^3 to 1024^3 and the core count from 128 to 16384. For the sake of brevity and similarity of plots, we only present the performance for the grid size of 1024^3. Each test consists of running 100 complex-to-complex forward single precision 3D FFTs in a loop to make sure everything settles down properly (branch predictors, etc.). The presented times are normalised per transform. Since P3DFFT does not support complex-to-complex transforms, they were simulated by calculating real-to-complex transforms on both real and imaginary parts of the input. Our code (HyFFT) runs one MPI process per socket and one OpenMP thread per core. Other libraries run one MPI process per core. In case of PFFT and P3DFF, the MPI processes are to be placed in a virtual 2D mesh by MPI routine `MPI_Cart_Create`. We used as squared process meshes as possible to minimise communication overhead since they reached the best performance. Execution times were measured by the `MPI_Wtime` routine. When possible, more accurate `FFTW_EXHAUSTIVE` planning flag was used (Zapat, Anselm). Since the exhaustive planning consumes a significant amount of time for high core counts, we had to roll back to less accurate `FFTW_MEASURE` on Fermi.

4.1 Experimental Supercomputing Clusters

The performance investigation was carried out on machines listed bellow. The first two are based on Intel x86 CPUs connected by a fat tree infiniband while the last machine is based on IBM BlueGene/Q architecture with a 5D torus topology.

1. **Zapat Cluster**
   Hardware configuration: 112 nodes (1792 cores), each node integrates 2×8-core Intel E5-2670 at 2.6GHz and 128GB RAM (14.3TB total), 2×600GB 15k hard drives, Infiniband 40 Gbit/s interconnection.
   Software configuration: GNU gcc 4.8.1 compiler (-std=c99 -O3), Open MPI 1.6.5, FFTW 3.3.4 (FFTW_EXHAUSTIVE only), PFFT 1.0.7 alpha, P3DFFT 2.6.1.
2. **Anselm Cluster**
   Hardware configuration: 209 nodes (3344 cores), each node integrates 2×8-core Intel E5-2665 at 2.4GHz, 64GB RAM (15.1TB total), Infiniband 40 Gbit/s QDR, fully non-blocking fat-tree interconnection.
   Software configuration: GNU gcc 4.8.1 compiler (-std=c99 -O3), Open MPI 1.6.5, FFTW 3.3.4 (FFTW_EXHAUSTIVE only), PFFT 1.0.7 alpha, P3DFFT 2.6.1.

[3] CERIT scientific cloud, CZ, https://www.cerit-sc.cz/en/Hardware/
[4] IT4Innovation Centre of Excellence, CZ, https://docs.it4i.cz/anselm-cluster-documentation
[5] CINECA consortium, IT, http://www.hpc.cineca.it/content/ibm-fermi-user-guide

3. **Fermi Cluster**
   Hardware configuration: IBM-BlueGene/Q, 10,240 nodes (163,840 cores), each node integrates a 16-core IBM PowerA2 at 1.6 GHz, 16GB RAM (163.8TB), 5D torus interconnection.
   Software configuration: GCC 4.4.6 compiler (-std=c99 -O3), FFTW 3.3.2 (FFTW_MEASURE only), PFFT 1.0.7 alpha.

4.2 Strong Scaling Investigation

The most important comparison of the HyFFT and other libraries involves the strong scaling, where the amount of work is fixed and the number of cores progressively increased by a factor of two. In an ideal case, any time the number of cores is doubled the execution time is be halved.

Fig. 4 shows the strong scaling for HyFFT, PFFT and the original FFTW library on the Fermi cluster. The results for P3DFFT has not been obtained yet due to difficulties while compiling the library on the BlueGene machine yet is expected to be very similar to PFFT. The most exciting observation is that both HyFFT and PFFT libraries scales very well even for very high core counts (the maximum number was limited by our allocation). Taking into consideration that each of 16k thread only processes 256KB of data, this is an extremely good result. The second favourable fact is that the curves remains steep without any flattening making us optimistic about further scaling. The average scaling factor is 1.87 while 2.0 would be optimal, with some superlinear drops attributed to cache effects (the slab/pencil is small enough to fit in cache).

The FFTW shows its superiority as long as there are enough slabs to employ all cores (slab decomposition has naturally lower overhead than the pencil one). The HyFFT is about 30% slower and the PFFT about 75% slower than the FFTW for low core counts. The advantages of the hybrid decomposition is clearly visible in this measurement (roughly 20-30% time reduction). The true potential of HyFFT and PFFT emerges when scaling beyond the number of slabs. Spreading the work over 16k cores can accelerate the calculation of 3D FFT over a 1024^3 grid by a factor of 7.8.

The strong scaling obtained on Anselm and Zapat shows the same tendency, thus only the plot for Anselm is presented, see Fig. 5. The first interesting observation is that the performance for all libraries almost matches for low and moderate core counts (up to 1024). Indeed, there is only about 10% difference between the fastest and slowest library. The difference becomes significant when running on 2048 cores where FFTW is not able to scale, the performance of PFFT and P3DFFT is almost identical and the HyFFT outperforms both by a factor of 1.27. The advantage of shared memory is again clearly visible. The average scaling factor reached by HyFFT is 1.9. Unfortunately, it was not possible to run the test on more cores as Anselm does only integrate 3.3k cores.

4.3 Comparison of Different Cluster Architectures

This section mutually compares the performance of the investigated libraries reached across different cluster architecture. Fig. 6 compares the performance

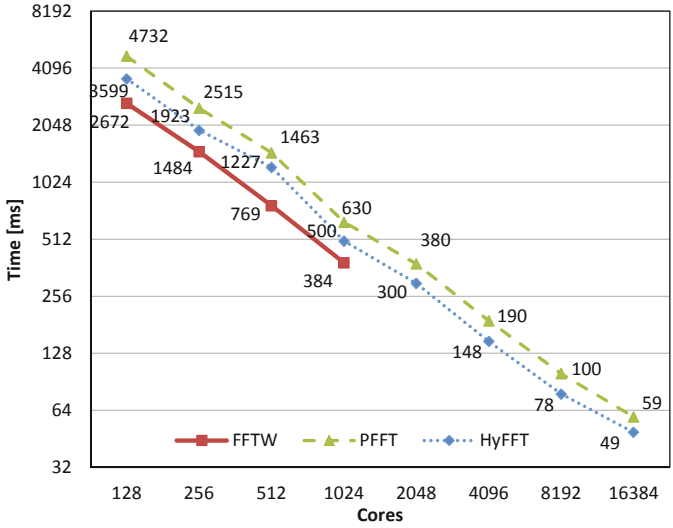

Fig. 4. Strong scaling for the grid size of 1024^3 on the Fermi cluster

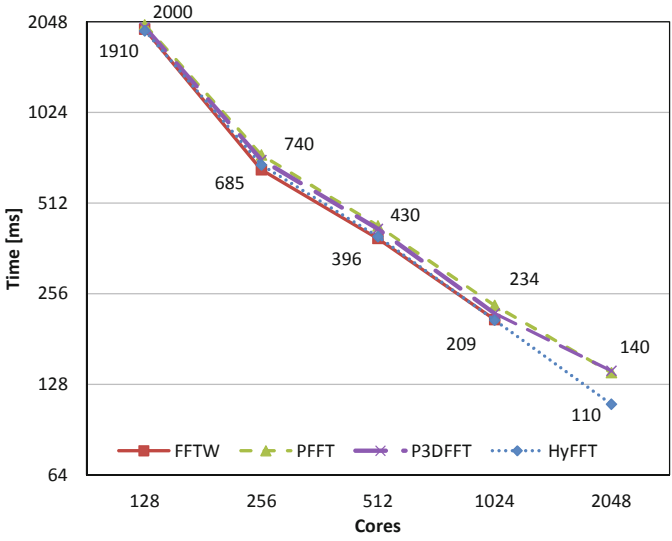

Fig. 5. Strong scaling for the grid size of 1024^3 on the Anselm cluster

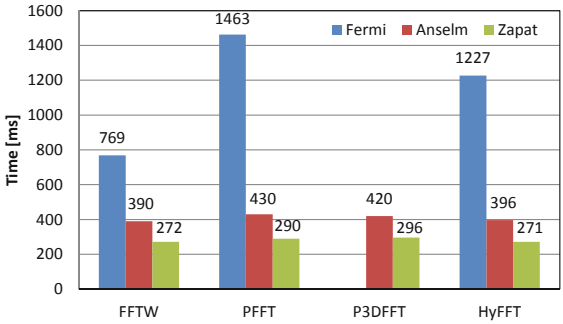

Fig. 6. The execution time of the 3D FFT over a 1024^3 grid distributed over 512 cores

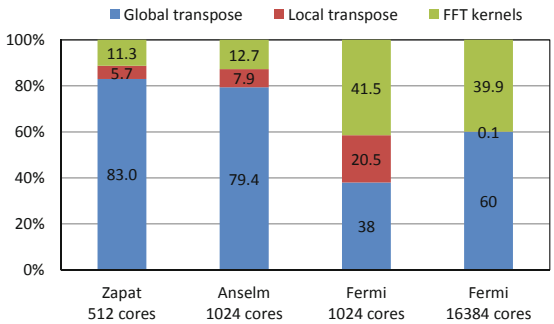

Fig. 7. Time distribution over the main components of HyFFT for a 1024^3 grid

provided by 512 cores because we did not have more cores at our disposal on Zapat.

It can be seen that the x86 based clusters (Anselm, Zapat) provide significantly higher performance than the BlueGene one (Fermi). It is caused by the joined factor of the lower raw performance per core as well as the different interconnection network. Fermi gives about 50% of FLOP/s per core compared to Anselm. The lower performance could also be caused by the less explorative FFTW planning flag. Interestingly, Zapat is approx. 1.4× faster than Anselm. Looking at the specification it is not obvious why there is such a big difference considering the interconnection is the same and the clock speed difference is less than 10%.

4.4 Time Distribution over HyFFT's Components

This section investigates the time distribution over the main components of HyFFT. Fig. 7 shows the time spent on calculating FFTs, local and global

transposition for a 1024^3 grid on different clusters with different core counts. The picture demonstrates that the global transposition remains the most time consuming part of the 3D FFT. It's overhead is highest for Zapat, closely followed by Anselm reaching up to 80%. The picture is a bit different for Fermi. For moderate core count, the compute time dominates, however with increasing number of cores, the compute part becomes smaller at the expense of communication. Finally, the time per local transpose seems reasonable.

5 Conclusions

The results have shown that the hybrid OpenMPI/MPI decomposition performs very well on current supercomputers. On Intel x86 clusters, HyFFT provides comparable performance to FFTW on low numbers of cores and outperforms the pure-MPI state-of-the-art libraries PFFT and P3DFFT by 10 to 20% for high core counts. Running HyFFT on a BlueGene machine reveals the true potential of the hybrid decomposition. Although being beaten by FFTW in situation where the 1D decomposition is enough to employ available cores, it further extends FFTW's scalability, reaching 8× higher performance on 16384 cores compared to the maximum number of employable cores (1024) of FFTW using a 1024^3 grid size. HyFFT also helps reduce communication overhead for high core counts leading in better execution times than other pure-MPI libraries.

This has a huge practical impact on many spectral simulations. Speaking about the k-Wave project, deploying the hybrid decomposition has the potential to decrease the simulation time by a factor of 8, bringing the simulation time within the clinically meaningful timespan of 24 hours and allowing patient specific treatment plans to be created.

In the future, we would like to add support for AVX-512 and ALTIVEC extensions to be able to vectorize the code on as many different machines as possible. We also plan to use non-blocking MPI communication to overlap some of the communication with computation. Finally, as the communication step is often dominant, we would like to focus our attention on low-power clusters.

Acknowledgments. The work was financed from the SoMoPro II programme. The research leading to this invention has acquired a financial grant from the People Programme (Marie Curie action) of the Seventh Framework Programme of EU according to the REA Grant Agreement No. 291782. The research is further co-financed by the South-Moravian Region. This work reflects only the authors view and the European Union is not liable for any use that may be made of the information contained therein. This work was also supported by the research project "Architecture of parallel and embedded computer systems", Brno University of Technology, FIT-S-14-2297, 2014-2016.

This work was further supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033). Computational resources were also provided by the MetaCentrum

under the program LM2010005 and the CERIT-SC under the program Centre CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, Reg. no. CZ.1.05/3.2.00/08.0144. We acknowledge CINECA and PRACE Summer of HPC project for the availability of high performance computing resources.

References

1. Cooley, J., Tukey, J.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 297–301 (1965)
2. Hesthaven, J.S., Gottlieb, S., Gottlieb, D.: Spectral Methods for Time-Dependent Problems. Cambridge University Press (2007)
3. Treeby, B.E., Jaros, J., Rendell, A.P., Cox, B.T.: Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method. The Journal of the Acoustical Society of America **2012**(131), 4324–4336 (2012)
4. Jaros, J., Rendell, A.P., Treeby, B.E.: Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. ArXiv e-prints (2014)
5. Frigo, M., Johnson, S.G.: The Design and Implementation of FFTW3. Proceedings of the IEEE **93**(2), 216–231 (2005)
6. Michael, P.: PFFT-An extension of FFTW to massively parallel architectures. Society for Industrial and Applied Mathematics **35**(3), 213–236 (2013)
7. Pekurovsky, D.: P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. SIAM Journal on Scientific Computing **34**(4), C192–C209 (2012)
8. Gupta, A., Kumar, V.: The scalability of FFT on parallel computers. IEEE Transactions on Parallel and Distributed Systems **4**(8), 922–932 (1993)
9. Foster, I.T., Worley, P.H.: Parallel algorithms for the spectral transform method. SIAM J. Sci. Comput. **18**(3), 806–837 (1997)
10. Sakai, T., Sedukhin, S., Tsuruga, I.: 3D Discrete Transforms with Cubical Data Decomposition on the IBM Blue Gene/Q. The University of AIZU, Fukushima, Japan, Technical report (2013)
11. Rahman, R.: The intel math kernel library and its fast fourier transform routines. Intel Corporation, Technical report (2012)