Large-scale Ultrasound Simulations Using the Hybrid OpenMP/MPI Decomposition

Jiri Jaros
Faculty of Information Technology
Brno University of Technology
Bozetechova 2
612 66 Brno, CZ
jarosjir@fit.vutbr.cz

Vojtech Nikl
Faculty of Information Technology
Brno University of Technology
Bozetechova 2
612 66 Brno, CZ
inikl@fit.vutbr.cz

Bradley E. Treeby
Dept. of Medical Physics and Biomedical Engineering
University College London
Malet Place Eng Bldg
London WC1E 6BT, UK
b.treeby@ucl.ac.uk

ABSTRACT

The simulation of ultrasound wave propagation through biological tissue has a wide range of practical applications including planning therapeutic ultrasound treatments of various brain disorders such as brain tumours, essential tremor, and Parkinson's disease. The major challenge is to ensure the ultrasound focus is accurately placed at the desired target within the brain because the skull can significantly distort it. Performing accurate ultrasound simulations, however, requires the simulation code to be able to exploit several thousands of processor cores and work with datasets on the order of tens of TB. We have recently developed an efficient full-wave ultrasound model based on the pseudospectral method using pure-MPI with 1D slab domain decomposition that allows simulations to be performed using up to 1024 compute cores. However, the slab decomposition limits the number of compute cores to be less or equal to the size of the longest dimension, which is usually below 1024.

This paper presents an improved implementation that exploits 2D hybrid OpenMP/MPI decomposition. The 3D grid is first decomposed by MPI processes into slabs. The slabs are further partitioned into pencils assigned to threads on demand. This allows 8 to 16 times more compute cores to be employed compared to the pure-MPI code, while also reducing the amount of communication among processes due to the efficient use of shared memory within compute nodes.

The hybrid code was tested on the Anselm Supercomputer (IT4-Innovations, Czech Republic) with up to 2048 compute cores and the SuperMUC supercomputer (LRZ, Germany) with up to 8192 compute cores. The simulation domain sizes ranged from 256^3 to 1024^3 grid points. The experimental results show that the hybrid decomposition can significantly outperform the pure-MPI one for large simulation domains and high core counts, where the efficiency remains slightly below 50%. For a domain size of 1024^3, the hybrid code using 8192 cores enables the simulations to be accelerated by a factor of 4 compared to the pure-MPI code. Deployment of the hybrid code has the potential to eventually bring the simulation times within clinically meaningful timespans, and allow detailed patient specific treatment plans to be created.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Technique—*Concurrent Programming*
I.6.5 [**Simulation and modeling**]: Types of Simulations—*Parallel, distributed*

General Terms

Algorithms, Performance, Design.

Keywords

Ultrasound simulations; 2D domain decomposition; OpenMP/MPI Hybrid programming; Performance evaluation; Supercomputing, k-Wave toolbox.

1. INTRODUCTION

The simulation of ultrasound wave propagation through biological tissue has a wide range of practical applications. Recently, high intensity focused ultrasound has been applied to functional neurosurgery as an alternative, non-invasive treatment of various brain disorders such as brain tumours, essential tremor, and Parkinson's disease. The technique works by sending a focused beam of ultrasound into the tissue, typically using a large transducer. At the focus, the acoustic energy is sufficient to cause cell death in a localised region while the surrounding tissue is left unharmed. The major challenge is to ensure the focus is accurately placed at the desired target within the brain because the skull can significantly distort it.

Performing accurate ultrasound simulations, however, requires the simulation code to be able to operate on large domains and deliver the results in a clinically meaningful time. Apart from the physical complexity, the main obstacle in implementing new ultrasound treatment planning procedures in clinical practice is the computational complexity. Considering the domain of interest encompassing the ultrasound transducer and the treatment area (normally on the order of centimetres in each Cartesian direction), and the size of the acoustic wavelength (on the order of hundreds of micrometers at the maximum frequency of interest), we have to simulate the wave propagation over hundreds or thousands of wavelengths. A sufficiently fine discretisation of the simulation domain which avoids numerical dispersion and instability can easily lead to grid sizes exceeding 10^{12} elements. Storing all the necessary acoustic quantities for such a large simulation domain in computer memory requires petabytes of memory and its processing reaches the order of exascale.

We have recently developed a pure-MPI pseudospectral simulation code using 1D domain decomposition that has allowed us to run reasonable sized simulations using up to 1024 compute cores [3]. However, this implementation suffers from the maximum parallelism being limited by the largest size of the 3D grid used. At the age of exascale, more and more systems will have numbers of processing cores far exceeding this limit. For example, a realistic ultrasound simulation performed by the k-Wave toolbox might use a grid size of 1024^3. Here, the 1D pure-MPI decomposition would only scale up to 1024 cores at most leading to calculation times exceeding clinically acceptable times (in this case between 30 and 72 hours). In contrast, top supercomputer facilities dispose with several hundred thousand compute cores and could provide the simulation result within an hour, if efficiently employed.

The second problem arising from limited parallelism is the total amount of memory that can be used to store simulation data. Not scaling the code to larger core counts holds the simulation domain size below 4096^3, which is not enough for some clinical applications (e.g., the use of shocked waves to vaporise a piece of tissue which can produce hundreds of harmonics).

This paper presents an improved implementation that exploits a 2D hybrid OpenMP/MPI decomposition. The 3D grid is first decomposed by MPI processes into slabs. The slabs are further partitioned into pencils assigned to threads on demand. This is supposed to (i) exploit shared memory within nodes and limit inter-process communication, (ii) employ 8 to 16 times more compute cores, (iii) increase the overall memory capacity while reducing the communication time.

2. DISTRIBUTED IMPLEMENTATION OF ULTRASOUND SIMULATIONS

The k-Wave toolbox [8] is designed to simulate ultrasound wave propagation in soft-tissues and bone, modelled as fluid and elastic media, respectively. In the k-Wave toolbox, the k-space pseudospectral method is used to solve the system of governing equations described in detail by Treeby in [9]. These equations are derived from the mass conservation law, momentum conservation law, and an empirically derived acoustic pressure-density relation that accounts for acoustic nonlinearity, absorption, and heterogeneity in the material properties [9].

The k-space and pseudospectral methods gain their advantage over finite difference methods due to the global nature of the spatial gradient calculations [4]. This permits the use of a much coarser grid for the same level of accuracy. However, the global nature of the gradient calculation, in this case using the 3D fast Fourier transform (FFT), introduces additional challenges for the development of an efficient parallel code. Specifically, the FFT requires a globally synchronising all-to-all data exchange. This global communication can become a significant bottleneck in the execution of spectral models. Fortunately, considerable effort has already been devoted to the development of distributed memory FFT libraries that show reasonable scalability of up to tens of thousands of processing cores [2], [5], [7].

The distributed implementation was written in C++ as an extension to the open-source k-Wave acoustics toolbox [8]. The standard message passing interface (MPI) was used to perform all interprocess communications, the MPI version of the FFTW library was used to perform the Fourier transforms [2], and the input/output (I/O) operations were performed using the HDF5 library [1]. To maximise performance, the code was also written to exploit single instruction multiple data (SIMD) instructions such as SSE or AVX. A detailed description can be found in [3]. The simulation time loop can be broken down into several phases:

1. The gradient of acoustic pressure is calculated by the Fourier collocation spectral method. This operation requires one forward 3D FFT and a few element-wise operations.

2. The acoustic particle velocity (a 3D vector) is calculated based on the acoustic pressure gradient using three inverse 3D FFTs and a few element-wise operations.

3. The gradients of particle velocity for each spatial dimension are calculated using three forward and three inverse 3D FFTs interleaved by several element wise operations.

4. The acoustic density is updated based on the particle velocity gradients using several element-wise operations.

5. The acoustic pressure field is updated based on the particle velocity gradients, acoustic density, and the non-linearity and absorption operators. This step includes two forward and two inverse 3D FFTs, and several elementary element-wise operations such as multiplication, addition, division, etc.

6. The desired acoustic quantities are sampled in regions of interest and either stored on the disk as time-varying series or further processed to calculate e.g. maximum, average, RMS, etc.

There are two important features of the time loop that should be highlighted. First, there are only two places where communication among MPI processes is required. It is within the 3D FFT while performing the distributed matrix transposition, and while the data is being sampled, collected, and stored using the parallel HDF5 library. To reduce the communication burden, pairs of forward and inverse FFTs do not bring the data into the original shape in between, instead a transposed shape is used to reduce the amount of communication to one half [3]. Moreover, the output data is collected and stored using chunks enabling buffering and staging of I/O operations. The second observation is that the simulation time loop is dominated by the FFT calculation. This accounts for nearly 60-80% (the higher number of processes, the higher proportion) of the execution time while the rest of the element-wise operations and the I/O only contribute by 40-20% [3]. Moreover, the FFT itself spends the vast majority of its time waiting for data being transmitted and transposed over the network.

The following subsections describe two different decompositions of the 3D simulation space we have developed: the 1D pure-MPI decomposition and the 2D Hybrid OpenMP/MPI decomposition.

2.1 Pure-MPI Decomposition

The pure-MPI decomposition is based on the 1D slab decomposition natively supported by the FFTW library. In this case, the 3D domain is partitioned along the z axis and every MPI process receives a given number of 2D slabs. In practice, all 3D matrices (acoustic pressure, velocity, density, etc.) are partitioned and distributed this way while several other support data structures are either partitioned and scattered or simply replicated [3]. The communication phase consists of one `MPI_Alltoall` communication performed as a part of the FFT, see Fig 1.

It has to be noted, that this decomposition provides reasonable scaling as long as the number of MPI processes is smaller than the z dimension size of the simulation domain. It also allows easy deployment on many supercomputing systems and eliminates problems with proper thread pinning, memory affinity, and so on. However, the disadvantage, apart from the limited number of processes

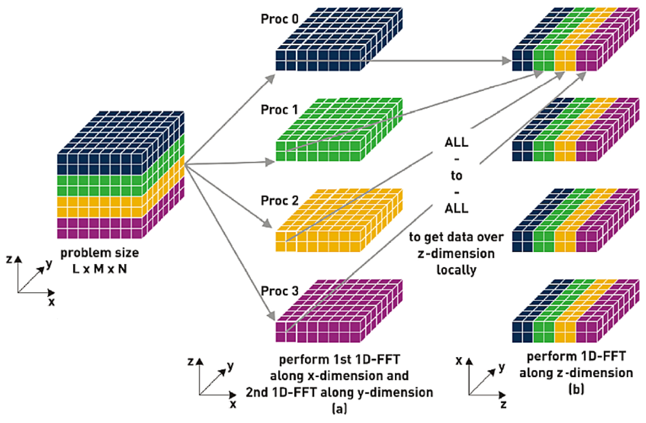

Figure 1: 1D domain decomposition and communication patterns within a 3D FFT.

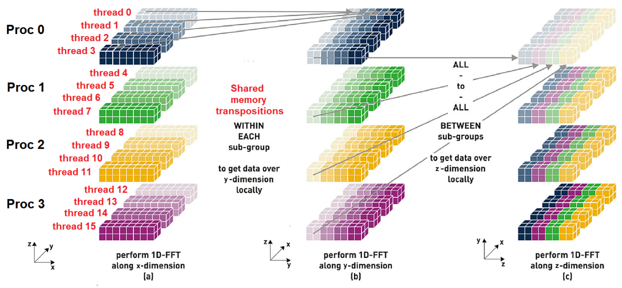

Figure 2: 2D domain decomposition and communication patterns within a 3D FFT.

to be used, is the communication overhead. With a growing number of MPI processes, the messages get smaller and smaller, while the number of messages grows with P^2. This eventually leads to network congestion and bandwidth decrease caused by the high latency of routing small messages.

2.2 Hybrid OpenMP/MPI Decomposition

The hybrid OpenMP/MPI decomposition tries to alleviate the disadvantages of the pure MPI decomposition by introducing a second level of decomposition and further breaking the 1D slabs up into pencils. In contrast to pure-MPI 2D decompositions, the smallest chunk an MPI process can receive still remains a 1D slab. Thus, the total number of MPI processes inherits the same limit as the 1D decomposition presented above. However, in this case, MPI processes are not mapped and bound to all compute cores, but only to one core per socket or node. Once a process is mapped on a socket/node, it spawns several OpenMP threads to process a given number of pencils from the allocated slab/slabs. Considering that many current supercomputers comprise of shared memory nodes typically integrating two sockets of 8 cores, we are able to scale the simulation up by a factor of 8 or 16. Moreover, the OpenMP threads can employ shared memory to significantly reduce the amount of inter-process communication and help in exploiting local caches.

It should be noted, that the 2D decomposition requires two communication phases to be carried out (one transpose along the y axis followed by another one along the z axis). Pure-MPI approaches typically implement this by a sequence of `MPI_Alltoall` communication over the y and z axis [7], [5]. Since the whole 1D slab is always placed on one socket/node, the hybrid implementation can efficiently employ the shared memory to perform the first transposition. The second transposition is carried out the same way as the 1D decomposition (see Fig 2), however, with a fewer number of processes (fewer and bigger messages, higher bandwidth, etc.).

The hybrid OpenMP/MPI simulation code was implemented in a very similar way to the pure-MPI one. The FFT calculation is based on the FFTW library tuned to be able to work with the 2D decomposition. We used our custom implementation presented in [6]. In a nutshell, it uses OpenMP FFTW kernels to perform series of 1D FFTs, a multi-threaded local transposition accelerated by SIMD instructions, and a distributed transposition offered by the FFTW library to carry out the communication part. This implementation has proved its superiority over pure-MPI approaches and enables better scaling than the original FFTW library (see [6] for more detail).

The element wise operations implemented in various steps of the simulation time loop were merged into a small number of kernels to maximize the temporal locality, written to utilise SIMD extensions, and run in parallel using the OpenMP library. To ensure correct thread and memory affinity, the First Touch Strategy was used.

3. EXPERIMENTAL RESULTS

The experimental evaluation of the hybrid decomposition was performed on two supercomputing systems, Anselm and SuperMUC. Anselm is a Czech supercomputer operated by the IT4Innovations National Supercomputing Center in Ostrava, Czech Republic. Anselm is an Intel-infiniband cluster based on Sandy Bridge processors (2x8 core Intel E5-2665 at 2.4GHz and 64GB RAM per node) interconnected by a 40Gb Fat-tree infiniband interconnection. The maximum number of cores we could use was 2048.

SuperMUC is a German supercomputer operated by Gauss Centre for Supercomputing and Leibniz Supercomputer Centre in Munich, Germany. SuperMUC is also an Intel-infiniband cluster based on similar Sandy Bridge CPUs (2x8 core Intel Xeon E5-2680 at 2.7 GHz and 32GB RAM per node) interconnected by a 40Gb Fat-tree infiniband network. The maximum number of cores we could use was 8192.

Comparing the hardware configuration, both systems are very similar and should produce very close results. The software stack on the other hand is different and allows us to check different compilers and MPI libraries. On Anselm, we used a GNU software stack comprising of a GNU C++ compiler (g++-4.8), the OpenMPI library in version 1.8.4, FFTW 3.3.3, and HDF5 1.8.13. The schedule manager is based on the OpenPBS software. SuperMUC on the other hand is based on an Intel software stack including an Intel Compiler 2015, Intel MPI in version 5.0, FFTW 3.3.3 and HDF5 1.8.12. The schedule manager is based on LoadLeveler.

3.1 Test configurations

One of the most important issues rising when working with a hybrid OpenMP/MPI code is the proper mapping of MPI processes and threads to cores, sockets and nodes. Improper setting can significantly deteriorate performance by allowing the threads to migrate among cores/sockets and losing the memory affinity. Since the default behaviour of MPI is to bind one process per core, spawning new threads by this process often leads to the threads being bound to the same core. As a consequence, one core is heavily overloaded while others are kept idle. The setting for three test configurations was as follows:

1. **Pure-C** (pure-MPI code, core level mapping) - This configuration uses the pure-MPI code implementing the 1D decomposition compiled without the OpenMP extension. This code

is the reference for comparison. No special care has to be taken to run this code.

2. **Hybrid-S** (hybrid code, socket level mapping) - This configuration uses the hybrid OpenMP/MPI code implementing the 2D decomposition compiled with the OpenMP library. The code starts one MPI process per socket and then spawns 8 threads per process. On Anselm, the code was launched with `mpirun -map-by socket -bind-to socket ./executable`, the number of threads was set by environmental variable `OMP_NUM_THREADS=8` pinned by `GOMP_CPU_AFFINITY="0-15"`. On SuperMUC, the LoadLeveler automatically sets all necessary environmental variables when specifying task per nodes equal to 2.

3. **Hybrid-N** (hybrid code, node level mapping) - This configuration uses the hybrid OpenMP/MPI code implementing the 2D decomposition. The code starts one MPI process per node and then spawns 16 threads per process. On Anselm, the code was launched with `mpirun -map-by node -bind-to none ./executable`, the number of threads was set by `OMP_NUM_THREADS=16` and thread binding by `GOMP_CPU_AFFINITY="0-15"`. On SuperMUC, the LoadLeveler automatically sets all necessary environmental variables when specifying task per nodes equal to 1.

The performance was investigated by a few simulation cases calculating the propagation of nonlinear waves in heterogeneous and absorbing media with a source driven by a sine wave. The domain sizes were chosen to equal 256^3, 512^3, and 1024^3 grid points. We did not test larger domains due to extensive simulation cost and the allocation limits. However, we expect better scaling with large simulation domains. The number of simulation timesteps varied from 100 to 1000 in order to get stable results and run the simulation for a reasonable timespan. The overall simulation run was, however, much longer due to the necessity of FFTW plan creation, which could take up to 30 minutes [3].

3.2 Strong Scaling

The strong scaling plots describe how the execution time decreases with increasing number of compute resources. The size of the problem is fixed. Fig. 3 and Fig. 4 show strong scaling for simulation domains of 256^3 and 512^3 grid points, respectively, and the number of compute cores growing from 16 (1 node) up to 2048 cores (128 nodes) on the Anselm supercomputer. The curves show the average execution time per one time step of the pure-MPI and two hybrid versions.

It can be seen that the simulation time decreases linearly, slowly reaching a plateau at the end (2048 cores). This is given by the size of the simulation grid, which is simply too small to keep all cores busy; one core only has 8k or 65k grid points to calculate. We can also conclude that the hybrid implementation is not so efficient for small core counts and the Pure-C code beats the hybrid ones almost twice. The clue is hidden in the communication part (the amount of computation is the same in all cases). In the Pure-C code, all cores participate in the communication transposing its part of the grid. However, the hybrid codes only use the master thread to communicate while the others are sleeping. Since the messages are quite big at low core counts, the loss in concurrency affects the performance by a great deal. For the smallest simulation domain size of 256^3, the hybrid decomposition seems to be inefficient. The Hybrid-S code offers a factor of two in performance, however, when using 8 times more resources. The efficiency is thus very low. For a bigger domain of 512^3, the hybrid codes scale much better and

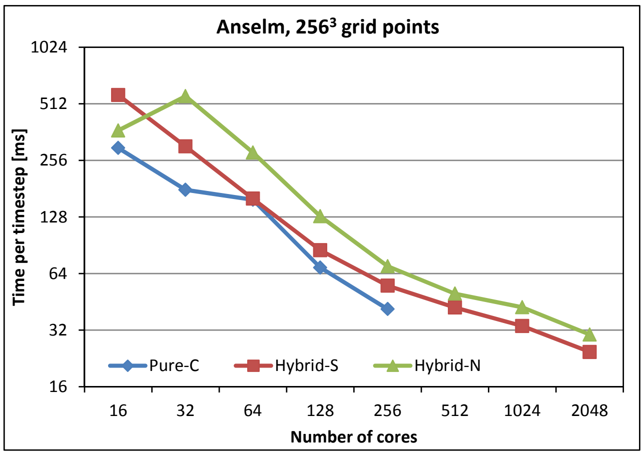

Figure 3: Strong scaling on Anselm, simulation grid of 256^3.

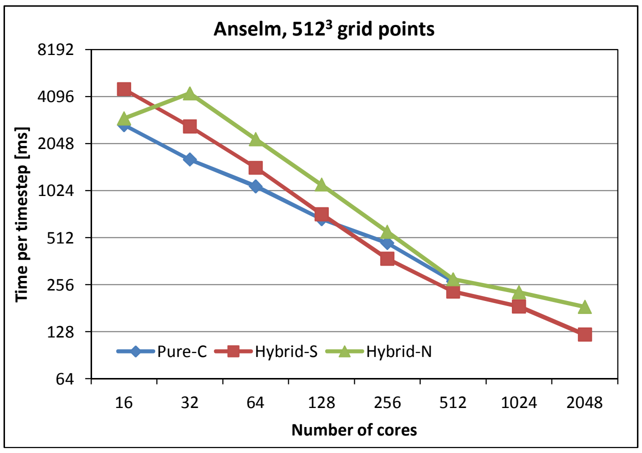

Figure 4: Strong scaling on Anselm, simulation grid of 512^3.

catches up with the Pure-C code at 128 cores (Hybrid-S version) or 512 cores (Hybrid-N version). The real strength of the hybrid code becomes evident beyond the scaling capability of the Pure-C code (512 cores). The Hybrid-S configuration offers more than 2.3 times higher performance when running on 2048 cores (efficiency of 57% compares to 512 cores).

The same test was also performed on SuperMUC, see Fig. 5. Since having a much bigger allocation here, we used a grid size of 1024^3 and executed the simulation with core counts ranging from 64 to 8192. Again, the Pure-C code is faster for lower core counts while the hybrid implementations win at the other side of the range. An interesting peak occurs for 2048 cores (Hybrid-S) and 4096 cores (Hybrid-N) where the performance is much lower than expected. This peak was also observed on other grid sizes always at the position where the number of cores is twice as high as the size of z dimension for Hybrid-S version, and four times higher for the Hybrid-N version. When investigating of this phenomenon, we tried different FFTW planning flags (patient and exhaustive), various compiler flags, MPI versions, and pinning strategies, however, we did not succeed in eliminating this behaviour. We suspect that it has something to do with the critical message size where MPI changes the policy of transmitting messages (sync. vs buffered), or that FFTW is unable to find a good communication plan.

To support this hypothesis we took a simulation flat profile, see

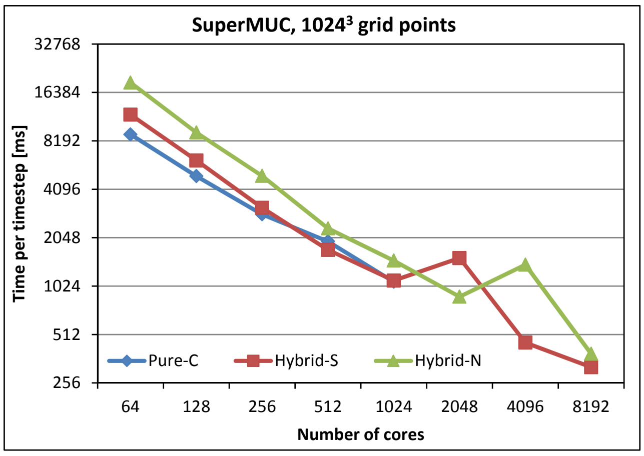

Figure 5: Strong scaling on SuperMUC, simulation grid of 1024^3.

Table 1: Communication share for various core counts and hybrid implementations on SuperMUC (grid 1024^3).

core count	Hybrid-S (MPI share)	Hybrid-N (MPI share)
1024	51.60%	46.24%
2048	71.48%	48.95%
4096	52.84%	74.38%

Table 1. The peaks in execution time directly correspond to the communication share. In a typical run, the communication share is about 50%, while in those exceptional cases the communication share springs up to 75%. The profile confirmed our hypothesis that the distributed transposition is not done optimally and a custom routine needs to be implemented to ensure the correct behaviour. This table also reveals that the hybrid OpenMP/MPI decomposition bounds the communication at a reasonable level of 50%, even for high core counts.

Fortunately, at least one of the hybrid versions works correctly at a given core count and the user has the ultimate choice. Finally, we would like to note that Hybrid-S version offers almost 4 times higher performance over Pure-C, which yields efficacy of almost 50%, which is not so bad considering the code is proven to be communication and memory bound.

4. CONCLUSIONS

This paper has presented our first attempt to improve scaling of large-scale ultrasound simulations using the hybrid OpenMP/MPI decomposition. The main goal was to enable the code to employ a number of compute cores exceeding the limit imposed by the standard 1D decomposition (the size of the z dimension). By introducing a second level of decomposition and breaking the 1D slabs assigned to MPI processes into pencils computed by OpenMP threads, as well as eliminating the need for another inter-process transposition by the shared memory, we have been able to accelerate the simulation by a factor of 4. This was achieved on SuperMUC when using 8192 compute cores to compute ultrasound wave propagation over a simulation domain discretised into 1024^3 grid points. We also managed to keep the communication overhead at an acceptable 50%.

We also observed curious behaviour for some configurations (number of processes and threads) where the simulation time abruptly increased. This may be attributed to the inability of the FFTW to find an optimal communication plan at this configuration. We can also conclude, that the scaling gets better for bigger simulation domains. While for domain sizes of 256^3 grid points, the hybrid decomposition does not bring much improvement due to the small amount of work, large domains of 1024^3 and bigger appear to benefit from the additional compute resources very well.

In our future work, we would like to test the code for bigger grid sizes, introduce custom communication plans, and further optimise the simulation code.

5. ACKNOWLEDGEMENTS

The project is 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 author's 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 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).

The authors gratefully acknowledge the assistance of Sebastian Lehrack from the LMU Faculty of Physics, and the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).

6. REFERENCES

[1] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson. An Overview of the HDF5 Technology Suite and Its Applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA, 2011. ACM.

[2] M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[3] J. Jaros, A. P. Rendell, and B. E. Treeby. Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. *The International Journal of High Performance Computing Applications*, 2015(2):1–19, 2015.

[4] T. D. Mast, L. P. Souriau, D.-L. D. Liu, M. Tabei, A. I. Nachman, and R. C. Waag. A k-space method for large-scale models of wave propagation in tissue. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, 48(2):341–354, 2001.

[5] P. Michael. PFFT-An extension of FFTW to massively parallel architectures. *Society for Industrial and Applied Mathematics*, 35(3):213–236, 2013.

[6] V. Nikl and J. Jaros. Parallelisation of the 3D Fast Fourier Transform Using the Hybrid OpenMP/MPI Decomposition. In *Mathematical and Engineering Methods in Computer Science*, LNCS 8934, pages 100–112. Springer International Publishing, 2014.

[7] D. Pekurovsky. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions.

SIAM Journal on Scientific Computing, 34(4):C192–C209, Jan. 2012.

[8] B. E. Treeby and B. T. Cox. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics*, 15(2):021314, 2010.

[9] B. E. Treeby, J. Jaros, A. P. Rendell, and B. T. Cox. 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.