DOI: 10.14529/jsfi160305

Spectral Domain Decomposition Using Local Fourier Basis: Application to Ultrasound Simulation on a Cluster of GPUs

J. Jaros[1], *F. Vaverka*[1], *B.E. Treeby*[2]

The simulation of ultrasound wave propagation through biological tissue has a wide range of practical applications. However, large grid sizes are generally needed to capture the phenomena of interest. Here, a novel approach to reduce the computational complexity is presented. The model uses an accelerated k-space pseudospectral method which enables more than one hundred GPUs to be exploited to solve problems with more than 3×10^9 grid points. The classic communication bottleneck of Fourier spectral methods, all-to-all global data exchange, is overcome by the application of domain decomposition using local Fourier basis. Compared to global domain decomposition, for a grid size of $1536 \times 1024 \times 2048$, this reduces the simulation time by a factor of 7.5 and the simulation cost by a factor of 3.8.

Keywords: domain decomposition, ultrasound simulation, spectral methods, GPU, FFT, local Fourier basis.

Introduction

Accurately simulating the propagation of ultrasound waves through biological tissue has a large number of practical applications, including the physics-based simulation of diagnostic ultrasound images [1] and treatment planning for ultrasound therapy [2] (a more comprehensive list is provided in [3]). However, ultrasound simulation for these applications is computationally demanding due to the length scales involved, where the propagation length can be hundreds or thousands of times longer than the acoustic wavelength. In turn, this leads to very large domain sizes, in some cases with more than 100 billion grid points [2]. This puts the simulation times beyond clinically useful time-limits, even when using significant computational resources [4]. The overarching goal of this work is to maximise computational efficiency to minimise the wall-clock time needed to solve such large scale problems.

One of the biggest challenges in performing large-scale ultrasound simulations is the accumulation of numerical dispersion. In general, this can be overcome through the application of spectral methods, which can be considered memory minimising due to their exponential error convergence with grid density [5]. For wave problems, the k-space pseudospectral method is particularly efficient. This combines the spectral calculation of spatial gradients (using the Fourier collocation spectral method) with a dispersion-corrected finite difference scheme to integrate forward in time. This approach was first proposed in [6] and further developed in [7–9]. The parallel implementation of the k-space pseudospectral method has previously been described by several groups [2, 10–13]. Aside from a number of element-wise matrix operations, the most significant operations performed at each time step are multiple real-to-complex and complex-to-real 3D fast Fourier transforms (FFTs). The parallel efficiency of the method therefore depends primarily on the parallel efficiency of the FFT. Note, in the conventional pseudospectral time domain method, the FFTs are 1D. However, for the k-space method, the FFTs are 3D, as the dispersion correction step is applied in the spatial Fourier domain.

The biggest challenge in the calculation of the 3D FFT is a globally synchronising all-to-all data exchange. This is required to transpose the 3D matrix data, as the FFT cannot

[1]Faculty of Information Technology, Brno University of Technology, Brno, CZ
[2]Department of Medical Physics and Biomedical Engineering, University College London, London, UK

stride across data belonging to multiple processes. Despite the large amount of progress on optimizing the implementation of distributed FFTs, the inherent communication bottleneck still limits scaling efficiency. The distributed FFT implementations deployed on CPU clusters usually achieve scaling factors between 1.5 and 1.8 when the number of processing elements is doubled. Pippig [14] reported a comparative study of FFTW [15], PFFT [14], and P3DFFT [16] using an IBM Blue Gene machine. Similar investigations using Intel-Infiniband clusters were reported in [13, 17, 18]. In general, the majority of the execution time is spent in communication. For typical ultrasound simulations with grid sizes ranging from 512^3 to 2048^3 grid points, when distributed over more than 512 CPU cores, 50-90% of the execution time is wasted waiting for data exchanges [2].

The imbalance between communication and computation is even more striking when graphics processing units (GPUs) are used, as the raw performance of GPUs is an order of magnitude above current central processing units (CPUs). In addition, transfers over the peripheral component interconnect express (PCI-E) bus have to be considered as another source of communication overhead. The implementation proposed by Gholami [17], which is currently one of the most efficient, reveals the fundamental communication problem of distributed GPU FFTs. For an 1024^3 FFT calculated using 128 GPUs, the communication overhead accounts for 99% of the total execution time. Although the execution time reduces by 8.6× for a 32× increase in the number of GPUs (giving a parallel efficiency of 27%), this overhead may be acceptable in many applications.

One way to overcome the global communication imposed by the Fourier spectral method is to use a *local* Fourier basis as proposed by Israeli, *et al* [19]. This allows the evaluation of derivatives to be splitted into multiple coupled subdomains, where the Fourier transforms for each subdomain are computed independently, followed by the exchange of data in an overlap or halo region. The spectral accuracy is maintained by forcing the local domains to be periodic through multiplication of the local data by a bell function. The bell function is equal to one within the physical domain, and tapers to zero within the overlap region [20]. Using local, Founer basis rather than global ones, FFTs can have a significant impact on the computational performance of Fourier spectral methods. For example, Ding & Chen implemented a solution to Maxwell's equations using local Fourier basis [21]. For the simulation with 1024^3 grid points running on 32 CPUs, they reported reduction in communication time from 9.49 seconds per time step when using global FFTs, to 1.46 seconds per time step when using local Fourier basis with 32 subdomains. Similarly, Garbey, *et al* reported close to ideal weak scaling when using local Fourier basis to solve a combustion problem using up to 16 processors [22].

In the current work, domain decomposition using local Fourier basis is combined with the k-space pseudospectral method to allow the highly efficient simulation of ultrasound propagation using a cluster of 128 GPUs with grid sizes up to $1024 \times 1536 \times 2048$. The governing equations and their discretisation are discussed in Sec. 2, with the local decomposition introduced in Sec. 3. Details of the parallel implementation are given in Sec. 4, with numerical experiments presented in Sec. 5. Summary and discussion are then given in Sec. 6.

1. Pseudospectral Ultrasound Model

The physical problem considered here is the propagation of small amplitude acoustic waves through a homogeneous and lossless fluid medium. In this case, the governing equations are

given by a set of coupled first-order partial differential equations [23]

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho_0}\nabla p \ , \quad \frac{\partial \rho}{\partial t} = -\rho_0 \nabla \cdot \mathbf{u} \ , \quad p = c_0^2 \rho \ . \tag{1}$$

Here $\mathbf{u}$ is the acoustic particle velocity, p is the acoustic pressure, c_0 is the sound speed, and ρ_0 and ρ are the ambient and acoustic density, respectively. The governing equations are solved using the k-space pseudospectral method, where spatial gradients are computed using the Fourier collocation spectral method, and time integration is performed using a dispersion-corrected finite difference scheme [8]. Written in discrete form, the governing equations in Eq. (1) become [2, 9]

$$\begin{aligned}
\frac{\partial}{\partial \xi} p^n &= \mathcal{F}^{-1}\left\{ ik_\xi \, \kappa \, e^{ik_\xi \Delta\xi/2} \mathcal{F}\left\{ p^n \right\} \right\} \ , \\
u_\xi^{n+\frac{1}{2}} &= u_\xi^{n-\frac{1}{2}} - \frac{\Delta t}{\rho_0} \frac{\partial}{\partial \xi} p^n \\
\frac{\partial}{\partial \xi} u_\xi^{n+\frac{1}{2}} &= \mathcal{F}^{-1}\left\{ ik_\xi \, \kappa \, e^{-ik_\xi \Delta\xi/2} \mathcal{F}\left\{ u_\xi^{n+\frac{1}{2}} \right\} \right\} \ , \\
\rho_\xi^{n+1} &= \rho_\xi^n - \Delta t \rho_0 \frac{\partial}{\partial \xi} u_\xi^{n+\frac{1}{2}} \ , \\
p^{n+1} &= c_0^2 \sum_\xi \rho_\xi^{n+1} \ .
\end{aligned} \tag{2}$$

The first four equations are repeated for each Cartesian direction, where $\xi = x, y, z$. Here, $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the forward and inverse FFT over all three spatial dimensions, i is the imaginary unit, k_ξ is the wavenumber vector in the ξ direction, $\Delta\xi$ is the grid spacing in the ξ direction, Δt is the time step, and κ is the so-called k-space operator used to correct for numerical dispersion introduced by the finite difference time step [8]. The acoustic density (which is physically a scalar quantity) is artificially divided into Cartesian components to allow an anisotropic perfectly matched layer (PML) to be applied to model free-field conditions [24]. The exponential terms are spatial shift operators that allow the particle velocity to be evaluated on a staggered grid [8]. The superscripts n and $n+1$ denote the function values at the current and next time points, and $n - \frac{1}{2}$ and $n + \frac{1}{2}$ at time staggered points.

The implementation of the discrete equations requires the storage of thirteen real 3D matrices defined in the spatial domain and three real and one complex 3D matrix defined in the Fourier domain. These are used to store the current values of the acoustic variables, their derivatives, and three temporary matrices. For a single precision shared memory implementation, the memory usage can be estimated as

$$\text{memory usage [GB]} \approx \frac{16.5 \times \text{Nx} \times \text{Ny} \times \text{Nz}}{1024^3/4} \ , \tag{3}$$

(neglecting scalars, 1D arrays, and the code itself). At each time step, the operations performed consist of four forward and six inverse 3D FFTs, and around 100 element-wise matrix operations.

Previously, this type of model has been implemented using C++ and parallelised using either OpenMP, with simulations reported up to 1024 × 1024 × 1024 grid points [9], or MPI, with simulations reported up to 4096 × 2048 × 2048 grid points [2, 10, 11]. In both cases, the 3D FFTs are calculated over the entire 3D domain, which requires an all-to-all global communication for each FFT. Although reasonable scaling is observed, particularly when using hybrid OpenMP/MPI decomposition [13], ten all-to-all communications are still required per time step, which is a major bottleneck in performance [2].

2. Local Fourier Basis decomposition

2.1. Formulation

As described in Sec. , the motivation behind domain decomposition using local Fourier basis is to replace the global FFTs by local FFTs computed independently on a series of subdomains. The general approach as applied to the k-space pseudospectral method can be described as follows. First, the field variables and material parameters are divided across the subdomains, including an overlap region (halo) with a specified width (see fig. 1). Here, the so-called non-overlapped decomposition is used [25], and the subdomains are assumed to always be of an equal size. Second, for each subdomain, an independent version of the complete k-space pseudospectral model is run. The spatial gradients are calculated as normal using local Fourier basis, but before taking the FFT, the halo is exchanged and function values are multiplied by a bell function. This tapers to zero within the overlap region to enforce periodicity. Here, the erf-like bell function defined by Boyd is used [25]. This is equal to 1 within the physical domain and given by $H(x) = \frac{1}{2}(1 + \mathrm{erf}(Lx/\sqrt{1-x^2}))$ within the overlap region, where L is a scaling parameter, which in this case is set to 2. The discrete values for x within the overlap region are given by $x = -1, -1 + 2/(N-1), \ldots, 1$, where N is the size of the overlap in grid points.

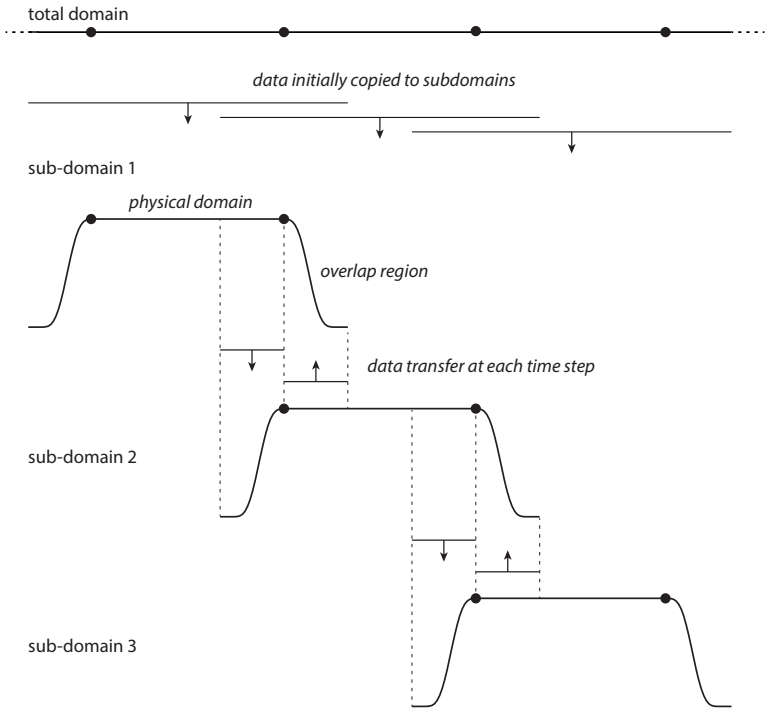

Figure 1. Schematic showing domain decomposition using local Fourier basis

2.2. Multidimensional decomposition

In 3D, there are several approaches to the decomposition of the global domain into subdomains, including 1D slab decomposition, 2D pencil decomposition, and 3D cube decomposition. The main differences are the number of interfaces a wave must travel through when traversing the grid in a given direction (this affects accuracy as discussed in Sec 2.3), the ratio between the halo and local subdomain size, and the number of data transfers that have to be performed [26]. The ratio between the halo and the local subdomain size improves with the dimensionality of

the decomposition. For a fixed number of subdomains, higher dimensionality implies a smaller amount of data that must be exchanged and consequently lower computational overhead. On the other hand, the number of direct neighbours grows with the dimensionality of the decomposition, i.e., there are 2, 8, and 26 direct neighbours for uniform 1D, 2D, and 3D decompositions, respectively. Since the interconnection bandwidth is not constant for all message sizes [27], in some cases it may be better to use 1D decomposition instead of 2D or 3D, even if a larger amount of data must be exchanged among a smaller number of neighbours. The impact of different decompositions on performance is discussed in Sec 4.3.

2.3. Accuracy

To test the accuracy of domain decomposition using local Fourier basis, a series of numerical experiments were conducted using a prototype CPU code. The tests consisted of propagating a plane wave along the grid axis (which reduces to a 1D problem) with the global domain divided into a given number of subdomains with a specified overlap width. The initial particle velocity was set to zero, and the initial pressure was set to be an impulsive pressure source. The spatial distribution of the pressure source was defined as a delta function (filtered by a Blackman window) positioned within the first subdomain. This generates a wave with broadband frequency content that smoothly decays up to the Nyquist limit [28]. For each test, a reference simulation using a global spectral method was also performed.

First, the dependence of the error on the width of the overlap region was examined. The total domain size was fixed at 512 grid points, and the overlap size varied from 8 to 32 grid points. The variation in L_∞ error compared to the reference simulation is shown in fig. 2(a). For an overlap width of 32 grid points, the error has not reached machine precision. However, the equivalent accuracy of the PML is only on the order of 10^{-3} to 10^{-4}, even with optimized parameters [29]. Given accuracy of the global solution is limited by the accuracy of the PML. It is sufficient to maintain a similar level of accuracy for the domain decomposition. Thus an overlap of 16 grid points was chosen, which gives an error less than 10^{-4} when using two subdomains. The change in the error for a fixed overlap size of 16 grid points with the total size of the local subdomain is shown in fig. 2(b). There is almost no change in the error for subdomain sizes from 32 to 1024 grid points, which means the size of the subdomains can be chosen to maximise computational efficiency.

Next, the dependence of the error on the number of domain cuts the wave must traverse was examined (for 1D decomposition, the number of domain cuts is one less than the number of subdomains). The total domain size was fixed at 2048 grid points with an overlap size of 16 grid points, and the number of subdomains was increased from 2 to 32 (in powers of 2). The variation in L_∞ error compared to the reference simulation is shown in fig. 2(c), and the error growth relative to using 2 subdomains is shown in fig. 2(d). The error increases linearly with the number of domain cuts the wave traverses, with a slope of $\sim$0.5. Thus, for typical sized problems (on the order of 2048 grid points in each dimension), up to 31 domain cuts (i.e., 32 subdomains if using 1D decomposition) can be used in each dimension with an overlap size of 16 grid points, and the error is still on the order of 10^{-3}. For 3D decomposition, this corresponds to 32,768 total subdomains (and in this case, GPUs). This means in practice, the level of achievable parallelism is not limited by the reduction in accuracy due to the use of local Fourier basis.

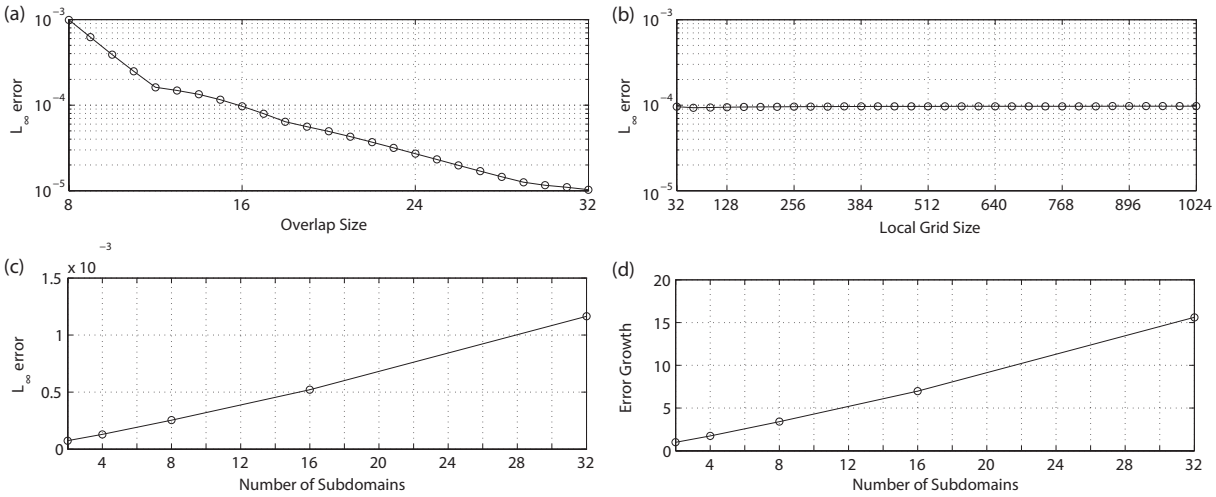

Figure 2. (a) Change in the L_∞ error with the overlap (halo) size for a fixed local domain size of 512 grid points. (b) Change in the L_∞ error with the local domain size for a fixed overlap (halo) size of 16 grid points. (c) Change in the L_∞ error with the number of domain cuts the wave must traverse for a total domain size of 2048 grid points and a fixed overlap size of 16 grid points. For 1D decomposition, the number of domain cuts is one less than the number of subdomains, where 2 subdomains corresponds to a local domain size of 1024 grid points, and 32 subdomains corresponds to a local domain size of 64 grid points. (d) Growth in the error with the number of subdomains relative to 2 subdomains (i.e., 1 domain cut)

3. Implementation

3.1. Communication framework

The numerical model described in Secs. 1 and 2.1 was implemented for multiple NVIDIA GPUs using MPI, C++, and CUDA. The implementation was divided into two components, the first responsible for the initial domain decomposition and periodic halo exchanges, and the second for performing calculations on each local subdomain. Starting with the communication framework, after the simulation is started, the number of subdomains and their organisation in 3D space is determined by parsing command line arguments. The framework supports any 1D, 2D, or 3D partition that fits into on-board GPU memory and meets the minimum size requirements. Each subdomain is assigned to an MPI process. In the case of 1D decomposition, the processes are grouped into an MPI communicator. If 2D or 3D decomposition is chosen, a virtual Cartesian topology is created and MPI is allowed to reorder ranks to preserve spatial locality between neighbouring subdomains. This is particularly useful while working on clusters with multiple GPUs per node.

The next step is GPU acquisition. Every MPI process (rank) inspects the configuration of the node being executed on, and chooses the first free GPU. This allows the framework to run on both slim and fat nodes with multiple GPUs, even in the case of non-uniform clusters (i.e., a mixture of nodes with a different number of integrated GPUs such as the Emerald cluster discussed in Sec. 4.1). The user (cluster batch scheduler) is responsible for assigning the correct number of ranks to individual nodes matching the number of integrated GPUs. The batch scheduler can also assign GPUs directly to ranks. If this feature is not supported by the scheduler and the GPUs are switched into exclusive process compute mode [30], the framework calculates the best assignment automatically and ensures mutual exclusion between ranks.

The execution proceed with the simulation setup. First, the simulation input file is opened and the simulation parameters are loaded. When the simulation domain size is determined, the framework calculates the size and position of the local subdomains, quantifies the size of the halo regions, and allocates all necessary data structures on both the CPU and GPU. The simulation data is then loaded from the input HDF5 file using parallel I/O and transferred into GPU memory.

The simulation time loop is divided into computation and communication phases. In total, there are four data exchanges per time step. These precede the gradient calculation for the acoustic particle velocity $\mathbf{u}$ in each Cartesian direction, and the gradient calculation for the acoustic pressure p (see Eq. (2)). The derivatives of the three spatial components of acoustic particle velocity can be calculated independently, which allows the MPI communications to be partially overlapped with the calculation (this is not possible for the calculation of the pressure gradient). During the data exchange, the halos are extracted from all three 3D matrices of velocity, packed into line-up buffers and downloaded to the CPU. This is done by repeated invocations of simple packing CUDA kernels followed by PCI-E transfers. This way the traffic over PCI-E is minimized. Next, the corresponding `MPI_Isends` and `MPI_Irecvs` are launched. The number of transfers depends on the decomposition chosen and varies between 4 (in the case of 1D decomposition) and 52 (in the case of full 3D decomposition). The execution only waits for the u_x halo to be delivered, uploads the halo back to the GPU, and replaces the appropriate data values. This is done by a PCI-E transfer followed by a CUDA unpack kernel. The calculation of $\frac{\partial}{\partial x} u_x$ is then started while the other four transfers proceed in the background. Thus this implementation partially hides two of three MPI communications.

3.2. Computation framework

The computation framework orchestrates all the necessary calculations in a simulation. It is divided into pre-processing, simulation time loop, data collection, and post-processing phases. The calculations are performed either as calls to the cuFFT library [31], or to custom CUDA kernels. Note, all calculations are performed by the GPU and the CPU only assists with the halo exchange and I/O operations. Since the size of the local subdomains has not been known in advance, several auxiliary variables are calculated during pre-processing, including the local wavenumber vectors, bell function, FFT shifts for staggered grids, etc [2]. The cuFFT library is then initialised and the FFT execution plans are created.

The simulation time loop then follows Eq. (2). The gradient calculations are performed by CUDA kernels which compute the required element-wise operations in both the spatial and Fourier domains. The kernels are organised into 1D CUDA grids composed of 1D CUDA blocks. The grid size is based on the actual number of CUDA multiprocessors (16 blocks per multiprocessor), and the block size is fixed to 256 threads. Every thread is responsible for processing multiple grid elements. The benefit of this solution is high occupancy and memory bandwidth. The same type of CUDA kernel is also used for halo packing and unpacking, as well as for sampled data aggregation and collection. The output data aggregation and post processing steps are described in more detail in [2].

4. Experimental results

4.1. Hardware description

The proposed implementation was deployed and evaluated on two GPU supercomputers, Emerald (e-Infrastructure South, UK) and Anselm (IT4Innovations national supercomputing center, CZ). Emerald is a heterogeneous GPU cluster consisting of several types of nodes equipped with different numbers and models of GPUs. Our allocation was limited to 128 NVIDIA Tesla M2090 cards with 6 GB of on-board memory. As error correction code (ECC) is switched on, the on-board memory capacity is reduced to approximately 5.4 GB. The GPUs are grouped in configurations of 3 or 8 per node, connected by PCI-Express 2.0. In the case of the 8-GPU configuration, pairs of GPUs share 16 PCI-E links. The CPU side is always comprised of two 6-core Westmere processors and 48 or 96 GB of RAM. The interconnection is provided by a 40 Gb/s half-duplex InfiniBand interconnect arranged into a fat-tree topology. The aggregated theoretical GPU performance is 170 TFLOPS in single precision, and the aggregated on-board memory is 768 GB.

Anselm consists of 209 compute nodes with a 40 Gb/s full-duplex InfiniBand interconnect arranged into a fat-tree topology. Each node integrates two 8-core Sandy Bridge CPUs and 64 GB of RAM. Twenty three nodes are equipped with one NVIDIA Kepler K20m GPU card with 5 GB of on-board memory and with ECC switched off. The GPUs are connected by PCI-Express 2.0. Our allocation was limited to 16 GPU cards. The aggregated GPU performance in single precision is 56 TFLOPS, and the aggregated on-board memory is 80 GB. For both systems, the GPU simulation code was compiled with the Intel compiler 2015, Intel MPI 5.0, NVIDIA CUDA 7.5, and HDF5 1.8.16. For comparison, a CPU implementation using global domain decomposition was used [2]. This code was compiled with the same tool chain, in addition to the FFTW 3.3.4 library.

4.2. Strong scaling

Several numerical experiments were performed to assess the performance of the multi-GPU implementation. First, strong scaling was assessed using domain sizes from 256^3 to $1024 \times 1024 \times 2048$ grid points with an overlap size (halo width) of 16 grid points. The problem sizes were limited by the aggregated amount of on-board memory and restrictions imposed by the smallest subdomain size. For Emerald, the scaling was investigated using up to 128 GPUs on the full range of simulation sizes (fig. 3(a)). Since our allocation on Anselm was limited to 16 GPUs, the largest domain size tested was $512 \times 512 \times 1024$ grid points (fig. 3(b)). The domain was partitioned over all three axes into a uniform number of subdomains starting from $1 \times 1 \times 1$ (i.e., running on a single GPU) up to $4 \times 4 \times 8$ and $2 \times 2 \times 4$ for the largest domain sizes on Emerald and Anselm, respectively.

Overall, the code achieves a reasonable scalability. On Emerald, for larger domain sizes the best parallel efficiency is approximately 27% when increasing from 8 to 128 GPUs, 34% from 16 to 128 GPUs, and 55% from 32 to 128 GPUs. In comparison, the strong scaling on Anselm reaches better values of parallel efficiency, reaching approximately 47% when increasing from 2 to 16 GPUs, 56% from 4 to 16 GPUs, and 85% from 8 to 16 GPUs. These levels of parallel efficiency are caused by a combination of multiple factors:

1. As the domain size grows, there is an increase in the number of neighbours the halo must be exchanged with. Since the number of GPUs is initially small, the decomposition is first done

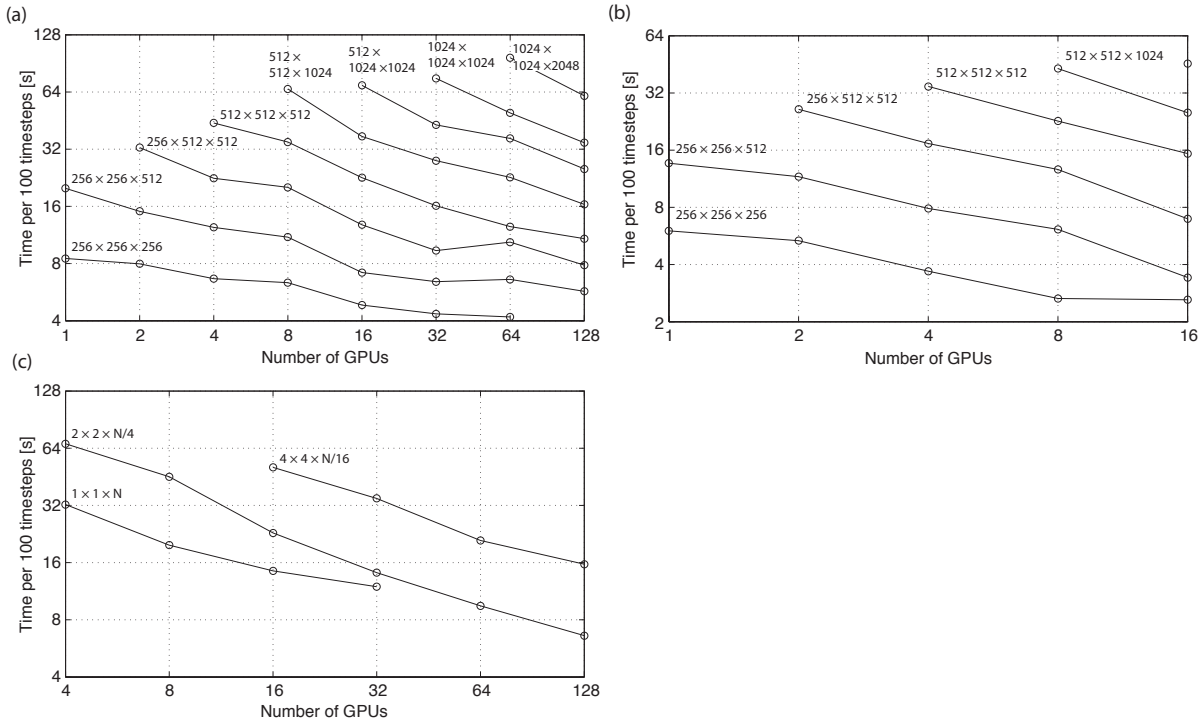

Figure 3. Strong scaling plots for (a) Emerald with 1-128 GPUs, and (b) Anselm with 1-16 GPUs. (c) The influence of different decomposition of the simulation domain of $256 \times 256 \times 2048$ on the strong scaling observed on Emerald. One-dimensional decomposition is compared with half and full 3D decomposition with 2, 11, and 26 neighbours, respectively

in 1D (2 GPUs), followed by 2D (4 GPUs) and 3D (8 GPUs) with only a single neighbour in each dimension. This situation will be referred to as half decomposition. For 16 and more GPUs, the decomposition turns into the full version with neighbours on both sides. This is done first in the x direction (16 GPUs), followed by the y direction (32 GPUs). The first complete full decomposition is employed for 64 GPUs, where the decomposition is $4 \times 4 \times 4$. The growing number of neighbours has a direct impact on the number of extraction/injection CUDA kernels, as well as the number of MPI communications to be performed.

2. On Emerald, the GPUs are packed into fat nodes sharing PCI-E links and the network adapter. The situation is worse in the case of 8-GPU nodes, where pairs of GPU cards share 16 links and the PCI-E bandwidth is halved. Moreover, 4 GPUs are connected to a single CPU socket creating contention in RAM.
3. Since the amount of on-board memory is limited, this also limits the total domain sizes. In the finest decomposition, the local subdomain only contains $64 \times 64 \times 64$ grid points, which makes the calculation time very small compared to the communication time. Moreover, for such a small subdomain, the halo accounts for 70% of the grid points in the local subdomain. The situation improves as the size of the local subdomains is increased. Unfortunately, the biggest subdomain that fits into on-board memory is approximately $256 \times 256 \times 512$. In this case, the halo accounts for around 25% of the local grid points.

Finally, comparing both systems, Anselm reaches almost twice the performance of Emerald. This is due to the combined effects of newer GPUs with higher performance and on-board memory bandwidth, ECC being switched off, and only a single GPU per node.

4.3. Decomposition comparison

Next, the influence of the domain decomposition shape on strong scaling was investigated on Emerald, using a global domain size of $256 \times 256 \times 2048$, a halo width of 16 grid points, and 4 to 128 GPUs. A 1D decomposition was used, where the domain is cut over the longest dimension into N partitions, with N corresponding to the number of GPUs. For comparison, a half 3D decomposition with 11 neighbours cut into $2 \times 2 \times N/4$, and a full 3D decomposition with 26 neighbours cut into $4 \times 4 \times N/16$ were also tested. As shown in fig. 3(c), the impact on performance is significant. The additional communication overhead arising from using a higher dimensionality decomposition with an increased number of neighbours directly translates into a performance drop. An obvious conclusion is that wherever possible, a 1D decomposition is preferred. When this is not possible due to an unacceptable level of numerical accuracy (the numerical error scales with the number of cuts along an axis as discussed in Sec. 2.3) or the subdomains being excessively small, a half 3D decomposition is preferred. Compared to the full 3D decomposition, the half decomposition reduces the simulation time by a factor of at least 2, which correlates with the reduced number of neighbours.

4.4. Simulation time breakdown

Next, the composition of the simulation time was investigated. Only the simulation loop was considered, as the initialisation, pre-processing, and post-processing phases usually take on the order of minutes, while realistic simulations may run for many hours. Figure 4 shows the simulation time breakdown for a domain size of $256 \times 256 \times 1024$, a halo width of 16 grid points, and 1D domain decomposition executed on Emerald and Anselm with 2 to 16 GPUs. Apart from faster execution on Anselm (due to faster GPUs), a difference in the PCI-E and MPI overhead may be observed. Although not clearly visible, the PCI-E latency on Emerald is almost 25% higher due to main memory congestion and shared PCI-E links. In addition, a higher MPI latency on Emerald is clearly noticeable. This is because Emerald only supports half-duplex, which decreases the MPI bandwidth by a factor of two. Furthermore, all inter-node communications are done via the main memory, which becomes the ultimate bottleneck. For the highest number of GPUs, where the local domain size is $256 \times 256 \times 64$, the percentage of time spent performing calculations, communications using PCI-E, and communications using MPI is 32%, 10%, and 58% for Emerald, and 40%, 17%, and 43% for Anselm, respectively. In comparison, previous implementations using global domain decomposition have reported the time spent performing calculations is below 1% on a GPU cluster [17], and 30% on a CPU cluster [13].

4.5. Influence of halo width

The influence of halo width (overlap size) on the communication overhead was investigated on Emerald using a simulation size of 512^3 grid points partitioned over all three dimensions with a halo width of 8, 16, or 32 grid points. The simulations were executed on 4 to 128 GPUs, with the time break down shown in fig. 5. The increase in the PCI-E and MPI overhead when the halo becomes larger is evident. When the halo size is 8 grid points, there is only a small overhead. However, when the overlap is increased to 32 grid points, the communication overhead prevents the code from scaling beyond 16 GPUs. Although the PCI-E latency remains at promising levels and scales with the number of GPUs, the MPI latency is the ultimate bottleneck. The

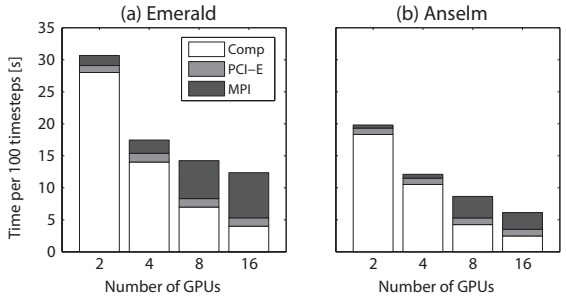

Figure 4. Breakdown of the execution time for a simulation domain size of $256 \times 256 \times 1024$ comprising of the computation part, MPI transfers between nodes, and PCI-E transfers between CPU and GPU

slight increase in the computation time across the three overlap sizes is due to the increase in local subdomain size with the halo size. The rise in MPI overhead as the number of GPUs is increased between 4 and 16 can be attributed to the transition from half 3D decomposition to full 3D decomposition. A halo width of 16 grid points was ultimately chosen as an acceptable compromise between performance and accuracy.

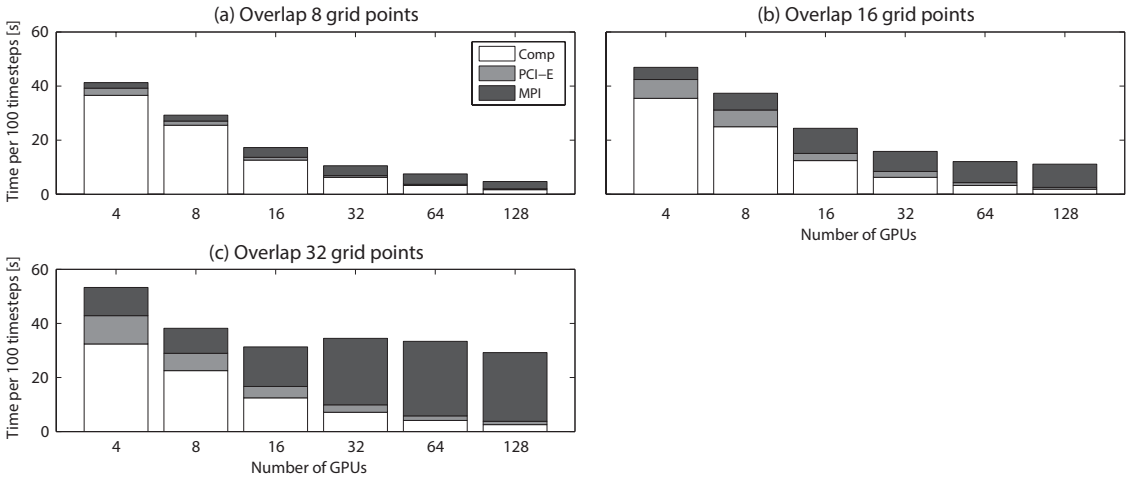

Figure 5. The influence of the overlap (halo) width on the overhead comprising of MPI and PCI-E transfers. The investigation was conducted on Emerald using a simulation size of 512^3, a halo width of 8, 16 and 32, and between 4 and 128 GPUs

4.6. Comparison of GPU and CPU

Finally, the GPU implementation using local Fourier decomposition was compared with an existing CPU implementation using global decomposition [13]. Several benchmark simulations were performed on Emerald and Anselm using three domain sizes (256^3, 512^3 and 1024^3) as shown in fig. 6. Here the horizontal axis corresponds to either the number of GPUs, or the number of CPU nodes (each of which integrates 16 processor cores). This grouping is used to estimate the simulation cost, which is charged per node on Anselm, regardless of whether the GPU is used. In comparison, Emerald has a different charge policy, with higher prices charged per GPU.

Figure 6(a) reveals that for small domain sizes, Anselm's GPUs are much faster than the CPU implementation when the number of nodes employed is small. Here, the simulation cost

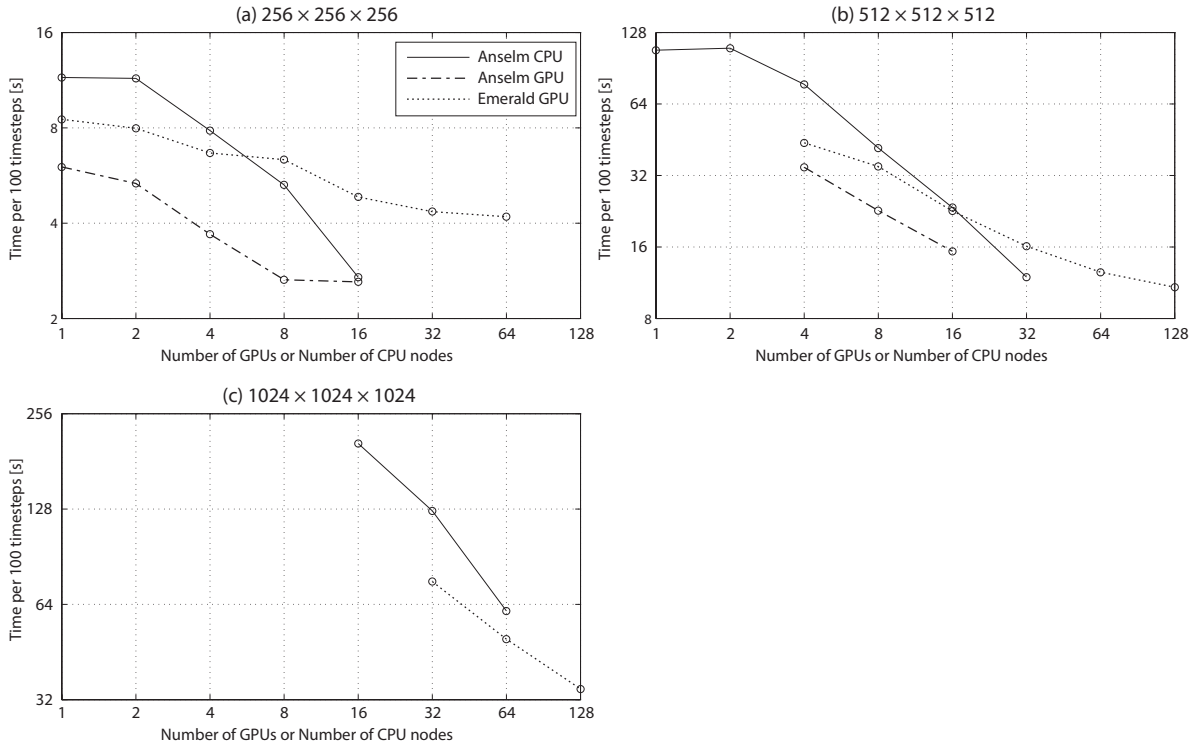

Figure 6. Performance comparison between local Fourier basis decomposition running on GPUs (Emerald and Anselm) and global domain decomposition running on CPUs (Anselm). In the case of the CPU implementation, the number of GPUs translates to the number of nodes, each of which contains 16 CPU cores

can be significantly reduced by utilising GPUs. The compute times for Anselm's GPUs and CPUs meet at 16 GPUs/nodes, where the local subdomains are extremely small. Emerald's GPUs seem to be too slow for such a small domain, where the communication is dominant. Figure 6(b) shows the same results for a larger domain size. Here, Anselm's GPUs outperform the comparable number of CPU cores. When the number of nodes is doubled (32 nodes or 512 CPU cores against 16 GPUs), the CPU cluster is faster by a factor of 1.28, however, for a doubled price. Emerald's GPUs beat Anselm when all 128 GPUs are used in the computation. The last benchmark shown in fig. 6(c) illustrates the benefits of the GPU implementation for larger domain sizes. Here 128 GPUs are faster than a cluster of 1024 CPU cores in 64 nodes by a factor of 1.76. Considering these 128 GPUs could be packed in 16 GPU nodes, this is a significant result. Note, the CPU code is limited by 1D slab decomposition so it is not possible to employ more cores than 256, 512 and 1024 for the domain sizes tested.

4.7. Production simulation

To show the impact of the proposed multi-GPU implementation on the type of ultrasound simulations used for treatment planning in focused ultrasound surgery, a comparison is given of the execution time and the financial aspects of running a single simulation using a grid size of $1536 \times 1024 \times 2048$ with 48,000 time steps performed as a part of the characterisation of a high-intensity focused ultrasound (HIFU) transducer used to treat prostate cancer. The output of such a simulation is given in fig. 7, which shows the maximum steady state acoustic pressure in a 2D plane in front of the transducer. Table 1 illustrates that a cluster of 128 GPUs is able

to deliver the simulation result in 9 hours and 29 minutes for the price of 426 USD (calculated based on the Emerald charge rate of 35.13c per GPU hour). Comparing this with the best price performance on the CPU cluster, the simulation can be completed in 3 days for 1,623 USD, or 2 days and 5 hours for 2,395 USD (calculated based on the Anselm charge rate of 8.8c per core hour). If price is the primary concern, as many tens of simulations are usually performed during any particular study, the GPU cluster can reduce the simulation time by a factor of 7.5 and the simulation cost by a factor of 3.8. The ultimate conclusion is that a GPU cluster is much a better solution for this type of simulation.

Table 1. Simulation time and cost when running a production simulation on Emerald with 96 and 128 GPUs, or Anselm with 128, 256, 512 CPU cores.

	Simulation Time	Simulation Cost
96 GPUs	14h 9m	$475
128 GPUs	9h 29m	$426
128 CPU cores	6d 18h	$1,826
256 CPU cores	3d 0h	$1,623
512 CPU cores	2d 5h	$2,395

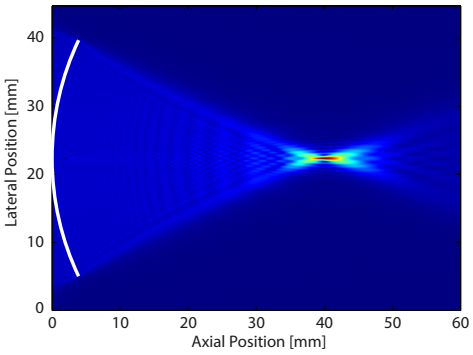

Figure 7. Pressure field from a prostate ultrasound transducer simulated using a domain size of $1536 \times 1024 \times 2048$ grid points ($45 \times 30 \times 60$ mm) with 48,000 time steps (60 μs) calculated in 9 hours and 29 minutes on 128 GPUs

Conclusion

This paper has presented a novel multi-GPU implementation of the Fourier spectral method using domain decomposition based on local Fourier basis [19]. The fundamental idea behind this work is the replacement of the global all-to-all communications introduced by the FFT (used to calculate spatial derivatives) by direct neighbour exchanges. By doing so, the communication burden can be significantly reduced at the expense of a slight reduction in numerical accuracy. The accuracy is shown to be dependent on the overlap (halo) size and independent on the local domain size. And to increase linearly with the number of domain cuts an acoustic wave must traverse. For an overlap (halo) size of 16 grid points, the error is on the order of 10^{-3}, which is comparable to the error introduced by the PML. Consequently, the level of parallelism achievable in practice is not limited by the reduction in accuracy due to the use of local Fourier basis.

Strong scaling results demonstrate that the code scales with reasonable parallel efficiency, reaching 50% for large simulation domain sizes. However, the small amount of on-board memory ultimately limits the global domain size for a given number of GPUs. 1D decomposition is shown to be the most efficient unless the local subdomain becomes too thin. Beyond, it is useful to exploit 2D or half 3D decomposition with only a single neighbour in a given direction to limit the number of MPI transfers. An overlap size of 16 grid points is shown to be a good trade off between speed and accuracy, with larger overlaps becoming impractical due to the overhead imposed by large MPI transfers. Compared to the CPU implementation using global domain decomposition, the GPU version is always faster for an equivalent number of nodes. For production simulations executed as part of ultrasound treatment planning, the GPU implementation reduces the simulation time by a factor of 7.5 and the simulation cost by a factor of 3.8. This is a promising result, given the GPUs utilized are now almost decommissioned.

In future, the code will be extended to model nonlinear wave propagation in heterogeneous media, as considered in [2]. The implementation could also be further improved by exploiting additional opportunities for overlapping communication and computation. First, the PCI-E and MPI communication could be overlapped. Second, the possibility of peer-to-peer communication among GPUs within the same node could be explored. This feature has the potential to eliminate expensive intra-node MPI communications. Third, the CPU could be utilized for additional executions, for example, assigning a subdomain to the idle CPU cores. Finally, multiple subdomains of different sizes could be executed on a single GPU, which might allow the communication on one subdomain to be overlapped while performing calculations on the others.

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), the Engineering and Physical Sciences Research Council, United Kingdom (grant numbers EP/L020262/1 and EP/M011119/1), and the Ministry of Education, Youth and Sports, Czech Republic (Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center - LM2015070"). The work presented here made use of Emerald, a GPU-accelerated High Performance Computer, made available by the Science & Engineering South Consortium operated in partnership with the STFC Rutherford-Appleton Laboratory. The authors would like to thank Alistair Rendell and Beau Johnston for useful contributions to early versions of the k-Wave GPU kernels.

References

1. G. F. Pinton, J. Dahl, S. Rosenzweig, and G. E. Trahey, "A heterogeneous nonlinear attenuating full-wave model of ultrasound," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 56, no. 3, pp. 474–488, 2009. DOI: 10.1109/TUFFC.2009.1066.

2. J. Jaros, A. P. Rendell, and B. E. Treeby, "Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound," *Int. J. High Perf. Comput. Appl.*, vol. 30, no. 2, pp. 137–155, 2016.

3. J. Gu and Y. Jing, "Modeling of wave propagation for medical ultrasound: a review," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 62, no. 11, pp. 1979–1992, 2015.

4. K. Okita, R. Narumi, T. Azuma, S. Takagi, and Y. Matumoto, "The role of numerical simulation for the development of an advanced HIFU system," *Comput. Mech.*, vol. 54, no. 4, pp. 1023–1033, 2014.

5. J. P. Boyd, *Chebyshev and Fourier Spectral Methods*. Mineola, New York: Dover Publications, 2001.

6. N. N. Bojarski, "The k-space formulation of the scattering problem in the time domain," *J. Acoust. Soc. Am.*, vol. 72, no. 2, pp. 570–584, 1982.

7. T. D. Mast, L. P. Souriau, D. L. 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*, vol. 48, no. 2, pp. 341–354, 2001.

8. M. Tabei, T. D. Mast, and R. C. Waag, "A k-space method for coupled first-order acoustic propagation equations," *J. Acoust. Soc. Am.*, vol. 111, no. 1, pp. 53–63, 2002.

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," *J. Acoust. Soc. Am.*, vol. 131, no. 6, pp. 4324–4336, 2012.

10. M. I. Daoud and J. C. Lacefield, "Distributed three-dimensional simulation of B-mode ultrasound imaging using a first-order k-space method.," *Phys. Med. Biol.*, vol. 54, no. 17, pp. 5173–5192, 2009.

11. J. C. Tillett, M. I. Daoud, J. C. Lacefield, and R. C. Waag, "A k-space method for acoustic propagation using coupled first-order equations in three dimensions," *J. Acoust. Soc. Am.*, vol. 126, no. 3, pp. 1231–1244, 2009.

12. J.-L. Vay, I. Haber, and B. B. Godfrey, "A domain decomposition method for pseudo-spectral electromagnetic simulations of plasmas," *J. Comput. Phys.*, vol. 243, pp. 260–268, 2013.

13. J. Jaros, V. Nikl, and B. E. Treeby, "Large-scale Ultrasound Simulations Using the Hybrid OpenMP/MPI Decomposition," in *Proceedings of the 3rd International Conference on Exascale Applications and Software*, pp. 115–119, Association for Computing Machinery, 2015.

14. M. Pippig, "PFFT-An extension of FFTW to massively parallel architectures," *SIAM J. Sci. Comput.*, vol. 35, no. 3, pp. C213–C236, 2013.

15. M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

16. D. Pekurovsky, "P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions," *SIAM J. Sci. Comput.*, vol. 34, no. 4, pp. C192–C209, 2012.

17. A. Gholami, J. Hill, D. Malhotra, and G. Biros, "AccFFT: A library for distributed-memory FFT on CPU and GPU architectures," *arXiv*, p. arXiv:1506.07933, 2015.

18. K. Czechowski, C. Battaglino, C. McClanahan, and K. Iyer, "On the Communication Complexity of 3D FFTs and its Implications for Exascale," in *Proceedings of International Supercomputing Conference*, ACM, 2012. DOI: 10.1145/2304576.2304604.

19. M. Israeli, L. Vozovoi, and A. Averbuch, "Spectral multidomain technique with local Fourier basis," *J. Sci. Comput.*, vol. 8, no. 2, pp. 135–149, 1993.

20. J. P. Boyd, "Asymptotic fourier coefficients for a C$^\infty$ bell (smoothed-"top-hat") & the Fourier extension problem," *J. Sci. Comput.*, vol. 29, no. 1, pp. 1–24, 2005. DOI: 10.1007/s10915-005-9010-7.

21. M. Ding and K. Chen, "Staggered-grid PSTD on local Fourier basis and its applications to surface tissue modeling.," *Optics Exp.*, vol. 18, no. 9, pp. 9236–9250, 2010.

22. M. Garbey and D. Tromeur-Dervout, "Parallel Algorithms with Local Fourier Basis," *J. Comp. Phys.*, vol. 173, pp. 575–599, 2001.

23. A. D. Pierce, *Acoustics: An Introduction to its Physical Principles and Applications*. New York: Acoustical Society of America, 1989.

24. J.-P. Berenger, "Three-dimensional perfectly matched layer for the absorption of electromagnetic waves," *J. Comput. Phys.*, vol. 127, no. 2, pp. 363–379, 1996.

25. J. P. Boyd, "A Comparison of Numerical Algorithms for Fourier Extension of the First, Second, and Third Kinds," *J. Comput. Phys.*, vol. 178, no. 1, pp. 118–160, 2002.

26. M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

27. HPC Advisory Council, "Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing," tech. rep., HPC Advisory Council, 2009.

28. B. E. Treeby and B. T. Cox, "A k-space Greens function solution for acoustic initial value problems in homogeneous media with power law absorption," *J. Acoust. Soc. Am.*, vol. 129, no. 6, pp. 3652–3660, 2011.

29. J. L. Robertson, B. T. Cox, and B. E. Treeby, "Quantifying numerical errors in the simulation of transcranial ultrasound using pseudospectral methods," in *IEEE Int. Ultrason. Symp.*, pp. 2000–2003, 2014.

30. NVIDIA, "CUDA Toolkit Documentation v7.5," tech. rep., NVIDIA, 2015.

31. NVIDIA, "cuFFT Library User's Guide," tech. rep., NVIDIA, 2015.