MPARD: A High-Frequency Wave-Based Acoustic Solver for Very Large Compute Clusters

Nicolas Morales
nmorales@cs.unc.edu
UNC Chapel Hill

Vivek Chavda
chavdav@cs.unc.edu
UNC Chapel Hill

Ravish Mehra
ravishm@cs.unc.edu
UNC Chapel Hill

Dinesh Manocha
dm@cs.unc.edu
UNC Chapel Hill

November 2, 2015

Abstract

We present a parallel time-domain acoustic wave solver designed for large computing clusters of tens of thousands of CPU cores. Our approach is based on a novel scalable method for dividing acoustic field computations specifically for large-scale memory clusters using parallel Adaptive Rectangular Decomposition (ARD). In order to take full advantage of the compute resources of large clusters, we introduce a hypergraph partitioning scheme to reduce the communication cost between cores on the cluster. Additionally, we present a novel domain decomposition scheme that reduces the amount of numerical dispersion error introduced by the load balancing algorithm. We also present a novel pipeline for parallel ARD computation that increases memory efficiency and reduces redundant computations.

Our resulting parallel algorithm makes it possible to compute the sound pressure field for high frequencies in large environments that are thousands of cubic meters in volume. We highlight the performance of our system on large clusters with 16000 cores on homogeneous indoor and outdoor benchmarks up to 10 kHz. To the best of our knowledge, this is the first time-domain parallel acoustic wave solver that can handle such large domains and frequencies.

1 Introduction

Modeling and simulating acoustic wave propagation is one of the leading problems in scientific computing today [12]. Challenges in this area vary from real time constraints in video games and virtual reality systems to highly accurate offline techniques used in scientific computing and engineering. The simulation environment also varies; acoustic propagation problems can vary from small room acoustics problems to large, complicated outdoor scenes.

The sound we hear is the results of small changes in air pressure traveling as a wave. The propagation of these pressure waves is governed by the linear, second order partial differential acoustic wave equation:

$$\frac{\partial^2}{\partial t^2} p(\vec{x}, t) - c^2 \nabla^2 p(\vec{x}, t) = f(\vec{x}, t), \qquad (1)$$

where $\vec{x}$ is a 3D position, t is time, $p(\vec{x}, t)$ is the pressure at point $\vec{x}$ and time t, f is a forcing term at point $\vec{x}$ and time t, and c is the speed of sound. In this paper, we assume that the environment is homogeneous and the speed of sound is constant in the media.

There are two classes of methods known for solving the acoustic wave equation. The first class encompasses geometric approaches to solving the equation. These methods include such approaches as ray

tracing techniques, image source methods, and beam tracing [2, 15, 16]. Geometric approaches are viable for real time applications, but do not provide sufficient accuracy for scientific or engineering applications.

The second class of methods, the numerical techniques, directly solve the wave equation. These techniques provide accurate solutions to the acoustic wave equation, suitable for scientific and engineering applications, but at a large computational cost. These methods include finite difference time domain (FDTD) [6, 28], finite element method (FEM) [35], and the boundary element method (BEM) [11, 8]. Most current implementations of these methods are limited to small domains or low frequencies (e.g., less than 2 kHz).

The main challenge is that the computational cost of computing the sound propagation in the environment scales with the 4th power of frequency and linearly with the volume of the scene, while memory use scales with the 3rd power of frequency and linearly with the scene volume. Since the human aural range scales from 20 Hz to 20 kHz, large scenes such as a cathedral (10 000 m^3 to 15 000 m^3) would require tens of Exaflops of computation and tens of terabytes of memory to compute using prior wave-based solvers. This makes high frequency acoustic wave simulation one of the more challenging problems in scientific computation [14].

Recently, there has been a lot of emphasis on reducing the computational cost of wave-based methods. These techniques, called *low dispersion techniques* use coarser computational meshes or decompositions in order to evaluate the wave equation. One example of a low dispersion algorithm is the Adaptive Rectangular Decomposition method (ARD)[24, 21], a solver for the 3D acoustic wave equation in homogeneous media. ARD is a domain decomposition method that subdivides the computational domain into rectangular regions. One of the main advantages of ARD is the greatly reduced computational and memory requirements over more traditional methods like FDTD [22].

Despite these computational advantages, the ARD method still requires a great deal of compute and memory in order to evaluate the wave equation for the full range of human hearing on large architectural or outdoor scenes. In order to deal with these requirements, a parallel distributed version of ARD was developed [22]. However, this approach was limited to smaller scenes and compute clusters and still had high computational requirements. The approach did not take into account the cost of communication between cores and suffered from numerical instability.

Main results

We present a parallel 3D wave-based acoustic solver capable of computing sound propagation through large architectural and outdoor scenes for pressure field computations at large frequencies (at least 10 kHz) and is designed to utilize tens of thousands of CPU cores.

Scaling to this level introduces many challenges, including communication cost, numerical stability, and reducing redundant computations. MPARD addresses all of these in a novel method that:

- Uses a hypergraph partitioning approach for load balance and communication reduction among multiple nodes
- Introduces a modified domain decomposition algorithm to improve the numerical accuracy of our wave-based simulator
- Uses a multi-stage pipeline for scene preprocessing and runtime computation

MPARD is capable of computing the acoustic propagation in large indoor scenes (20 000 m^3) up to at least 10 kHz. The algorithm has been tested to and scales up to at least 16000 cores. We have analyzed many aspects of our parallel algorithm including the scalability over tens of thousands of cores, the time spent in different computation stages, communication overhead, and numerical errors. To the best of our knowledge, this is one of the first wave-based solvers that can handle such large domains and high frequencies.

2 Previous work

There has been considerable research in the area of wave-based acoustic solvers, including the field of parallel solvers, domain decomposition approaches, and low-dispersion acoustic solvers. In this section, we provide a brief overview of some of these approaches.

2.1 Parallel wave-based solvers

Parallel wave solvers are used in a multitude of scientific domains, including the studying of seismic, electromagnetic, and acoustic waves. A large category of these solvers are parallel FDTD solvers either for large clusters [17, 36, 40, 41] or for GPUs [26, 29, 33, 34, 39]. A category of parallel methods are also based on finite-element schemes [13, 5].

Additionally, there are several parallel methods developed for specific applications of the wave equation. PetClaw [1] is a scalable distributed solver for time-dependent non-linear wave propagation. Other methods include parallel multifrontal solvers for time-harmonic elastic waves [38], distributed finite difference frequency-domain solvers for visco-acoustic wave propagation [23], discontinuous Galerkin solvers for heterogeneous electromagnetic and aeroacoustic wave propagation [4], scalable simulation of elastic wave propagation in heterogeneous media [3], etc.

2.2 Domain decomposition

MPARD, like ARD, is a domain decomposition method. Domain decomposition approaches subdivide the computational domain into smaller domains that can be solved locally. The solver uses these local solutions to compute a global solution of the scientific problem. Many of these approaches are designed for coarse grain parallelization where each subdomain or a set of subdomains is computed locally on a core or a node on a large cluster.

Domain decomposition approaches tend to fall into two categories: *overlapping subdomain* and *non-overlapping subdomain* methods [10].

Existing domain decomposition approaches in computational acoustics include parallel multigrid solvers for 3D underwater acoustics [27] and the non-overlapping subdomain ARD approach [24].

2.3 Low dispersion acoustic solvers

The goal of low dispersion methods is to reduce computation cost by using coarser computational meshes. A wide variety of these approaches exist including a multitude of waveguide mesh approaches [37, 30, 31] and an interpolated wideband scheme [19].

Our approach is primarily based on the ARD method [24] which partitions the computational domain into rectangular regions. It utilizes the property that the wave equation has a closed form solution in a homogeneous rectangular domain. Therefore, the only numerical dispersion originates from the interfaces between these rectangular regions, allowing a much coarser grid size.

3 Parallel Adaptive Rectangular Decomposition

In this section, we provide a brief overview of the ARD method and a description of the parallel ARD pipeline. More details about these methods are available in [24, 21, 22].

3.1 Adaptive Rectangular Decomposition

Adaptive Rectangular Decomposition (ARD) is a wave-based time domain method for solving the 3D acoustic wave equation (Equation 1). ARD is limited to homogeneous environments, where the speed of sound c does not vary.

ARD is a domain decomposition approach. It takes advantage of the fact that the analytic solution of a 3D rectangular domain in a homogeneous media is known [20]. The ARD solver computes this analytic solution inside the rectangular regions and patches the results across the boundaries between these regions using an FDTD stencil.

After the sound pressure in each rectangular subdomain is computed, sound propagation across the interfaces between subdomains must be calculated.

ARD uses a (6,2) FDTD stencil in order to patch together subdomains.

The pressure field computation assumes a perfectly reflective boundary condition for the rectangular regions and the interface stencils. However, this is an unrealistic assumption for real-world scenes, which have a variety of different absorbing materials and walls (the free space boundary condition can be modeled as a fully absorptive wall).

ARD implements the Perfectly Matched Layer (PML) scheme, a modified form the wave equation that absorbs propagating waves [25]. PML subdomains, using this modified wave equation, are generated from wall regions and the boundary of the scene and use the same interface scheme to transfer pressure waves between subdomains.

3.2 Parallel ARD

The parallel ARD method is an adaptation of the ARD method for distributed compute clusters and shared memory machines [22]. MPARD is directly based on parallel ARD, so this section will go over relevant implementation details and the parallel ARD pipeline, which MPARD also shares.

3.2.1 Parallel algorithm

The parallel ARD pipeline is similar to the serial implementation discussed in Section 3.1. Because the subdomains in the ARD decomposition are non-overlapping, the local computation for each rectangular subdomain can be run independent of any other subdomain. Therefore, each core can update a set of subdomains independently of any other core. These subdomains are referred to as *local subdomains* for a particular core and are fully in memory for that core. Subdomains not present on a core are referred to as *remote subdomains*; only metadata is stored for these rectangular regions. This metadata generally includes which rank (core id) owns the subdomain and what interfaces the subdomain is spanned by. The core that is responsible for the local update of a rectangular region is referred to as the *owner* of that subdomain.

However, interfaces between subdomains still need to be evaluated. Only one core needs to carry out the interface computation. This core is referred to as the *owner* of the interface. The owner of the interface is generally also the owner of one of the rectangular regions that the interface spans. Before interface evaluation is performed, the owner of the interface needs all the pressure terms from the subdomains spanned by the interface. Not all of these subdomains are local; some may be resident on other cores. Therefore, the core evaluating the interface needs to retrieve the pressure data from other cores. Similarly, once the interface is resolved, forcing terms need to be sent back to the cores owning the subdomains spanned by the interface. Finally global update for each rectangular region is computed with the results from the interface handling. This means that the final global pressure computation requires two data transfers per remote interface.

3.2.2 Load balancing

Due to the greedy nature of the rectangular decomposition algorithm, some rectangles can be quite large. Additionally, because of the stairstepping artifacts of curved geometric surfaces, the decomposition algorithm can generate some very small rectangles. Because the evaluation cost of the local update of a subdomain is linearly related to the volume of that subdomain, the discrepancy in subdomain sizes can cause load imbalance problems where all the cores wait for one core to finish computing the local update. In fact, even attempting some sort of balancing without modifying the rectangle sizes does not improve the scalability of parallel ARD [22].

Parallel ARD implements a load balancing scheme that modifies the sizes of the rectangles to be less than a certain threshold volume. Splitting is implemented by dividing any rectangle greater than a certain volume into two smaller rectangles. The splitting plane is chosen so that one of the rectangles is just below the threshold volume. This approach can minimize the interface area added under some circumstances, but does not take into account the shape of the subdomains. This can cause numerical stability issues, as discussed in Section 4.2.

Finally rectangular subdomains are assigned to cores using a simple bin-packing approach where the total volume assigned to each core is roughly equal. The bin-packing approach only considers the total volume assigned to each core and does not take into account communication cost of the core assignment.

3.2.3 Limitations of parallel ARD

While parallel ARD has been shown to work well for smaller clusters up to 1000 cores, it has many limitations when scaling to very large clusters with tens of thousands of cores:

- Parallel ARD does not take into account the cost of communication between cores
- Parallel ARD suffers from numerical instability when a large number of cores is used
- Parallel ARD computes interface information in the simulation itself, which can take hours on larger scenes

Large scale scenes on a large number of cores suffer from these issues. On some of our test scenes, hundreds of gigabytes of interface data are be created. This means a significant increase in communication cost that is dependent on the assignment of subdomains to cores. Additionally, the Parallel ARD splitting scheme creates thin and badly formed partitions when splitting the larger partitions. This occurs when the splitting threshold is lower than the smallest surface area of a given partition. Finally, the time ARD requires to compute hundreds of gigabytes of interface data can take hours to compute. On larger clusters where computation time is a valuable resource, this is time wasted.

4 MPARD

MPARD introduces a new parallel pipeline for the ARD method, designed to run on tens of thousands of cores. The pipeline includes a modified subdomain assignment scheme that takes into account the communication costs between cores, a modified domain decomposition and splitting stage that increases numerical stability, and new preprocessing stages for computing metadata for interfaces and PML subdomains.

4.1 Communication efficiency

Each interface in the scene covers two or more subdomains (for example, if the subdomains on either side are especially thin, the 6th order stencil may cover more than two). When subdomains on an interface are owned by different cores, the pressure terms required by the interface and the forcing terms generated by the interface need to be communicated.

MPARD uses asynchronous communication calls to avoid blocking while sending messages. When a core completes an operation (either a subdomain update or an interface update), it can send off a message to the cores that require the computed results. The sending core does not have to wait for the message to be received and can continue working on the next computation. When receiving data, a core can place the message on an internal queue until it needs the data for that operation. As a particular core finishes working on a subdomain, it sends off an asynchronous communication message to cores that require the pressure field of that subdomain for interface computation. As the message is being sent, that particular core begins to process the next subdomain, sending off a message upon completion. At the same time the receiving core is working on its own subdomains. The incoming message is stored on an internal queue for use when needed. The sending core therefore does not have to worry about when the message is received.

When a core finishes updating a rectangular region, it waits until it has the pressure fields from the remote subdomains available. It evaluates the first available interface, asynchronously sending back the forcing terms as soon as the stencil is evaluated. This allows us to effectively hide the communication costs. By using this system of queues and a "fire and forget" method of sending messages, the only explicit synchronization is the barrier at the end of the time step. This barrier is necessary to ensure the algorithm's correctness. Figure 1 shows a flow

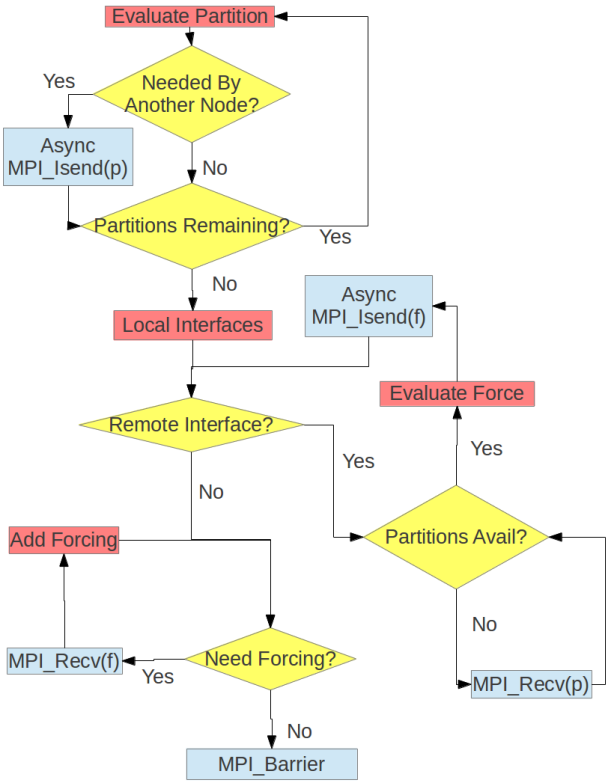


Figure 1: A flow-chart of one time step of the main loop of the parallel ARD technique implemented on each core of the CPU-cluster with asynchronous MPI calls.

chart of one time step of MPARD, laying out the asynchronous message passing calls for transferring pressure and forcing terms.

Another important thing to note is that in general ARD's communication cost of evaluating all the interfaces is proportional to the surface area of the rectangular region while the cost of evaluating each subdomain is proportional to the volume of the subdomain. This implies a rough $O(n^2)$ running time for interface evaluation (where n is the length of one side of the scene) while local update has an $O(n^3)$ running time. This means that as the size of the scene or the simulation frequency increases, the computation cost of local update dominates over the communication cost of transferring pressure and forcing values for interface evaluation. On the other hand the finer grid size of larger complex scenes can introduce many more interfaces which causes an increase in the cost of communication.

In order to evaluate these interfaces, communication is only required when resolving the interface between two or more subdomains that lie on different cores. As a result, minimizing the number of these interactions can greatly reduce the total amount of communicated data.

This can be performed by ensuring that neighboring subdomains that are part of the same interface are located on the same core. Due to the complexity of the scene and the interactions between different rectangular regions, this is not feasible. However, we can use a heuristic to minimize the number of interfaces that span across multiple cores.

This problem can be reworded as a hypergraph partitioning problem. Our hypergraph can be represented as the pair $H = (X, E)$ where X, the nodes of the hypergraph are the rectangular regions of our decomposition, while the hyperedges E represent the interfaces between the rectangular regions. Hyperedges are used rather than regular edges because an interface can actually cover up to six different subdomains.

The goal of the hypergraph partitioning algorithm is to divide the hypergraph into k regions such that the cost function of the hyperedges spanning the regions is minimized [9]. In ARD, the partitioning algorithm can be run to divide our computational elements (interfaces and rectangles) into k regions, where k is the number of processors used in the simulation. As a result, the interface cost between cores is minimized.

Additionally, because the hypergraph partitioning algorithm attempts to generate k regions of equal cost, the heuristic serves as a way of load balancing the assignment of work to cores. The cost of evaluating a rectangular region is linearly related to the volume of the region. Therefore, we can input the volume of each rectangular subdomain as the weight parameter for a node in the hypergraph.

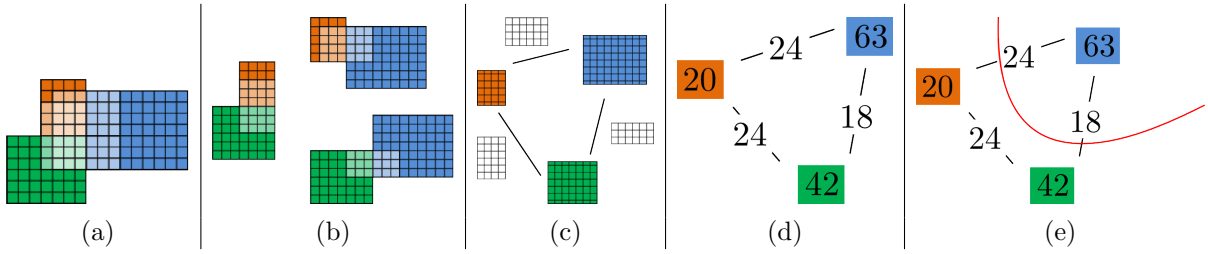


Figure 2: The relationship between computational elements of MPARD (rectangular subdomains and interfaces) and the hypergraph structure. (a) shows an example scene of three subdomains with three interfaces, shown in (b). The organization of the resulting hypergraph is shown in (c), while (d) shows the hypergraph with the node weights determined by the sizes of the rectangles and the hyperedge weights determined by the size of the interfaces. Subfigure (e) shows an example partitioning of the simple graph, taking into account both the computation cost of each node and the cost of each interface.

4.2 Load balancing and numerical stability

The splitting algorithm for load balancing introduced in parallel ARD minimizes the number of extra interfaces created by splitting in some cases. One of the resulting subdomains is exactly below the maximum volume threshold of the splitting algorithm.

However, on a cluster with a higher number of cores and where the volume threshold can be relatively small, this splitting algorithm can introduce a series of very small and thin rectangles. Small and thin rectangles can create numerical instability during interface resolution. The interface solver uses a (2,6) FDTD scheme. However, in the scenario where a series of thin interfaces with a width of one voxel are next to each other, our pressure computation devolves into the (2,6) FDTD scheme, but does not have a small enough grid size for the method to be stable. The lack of sufficient grid size for the FDTD stencil causes the numerical instability in the overall system.

In this particular case, it is more advantageous to have well-formed rectangular subdomains that may introduce more interface area rather than degenerate rectangles which can result in numerical issues.

We introduce a new splitting scheme for load balancing that is particularly useful for reducing numerical instability at interfaces for higher numbers of cores. Our new approach favors well-formed subdomains that are roughly cuboidal in shape rather than long and thin rectangles.

Each rectangular region that has a volume greater than some volume threshold Q is subdivided by the new algorithm. Q is determined by the equation

$$Q = \frac{V}{pf}, \tag{2}$$

where V is the total air volume of the scene in voxels, p is the number of cores the solver is to be run on, and f is the balance factor. It is usually the case that $f = 1$, although this can be changed to a higher value if smaller rectangles are desired for the hypergraph partitioning heuristic (5.1).

4.3 Interface and PML computation

An important step of the ARD and parallel ARD methods is the initialization of interfaces and PML regions. This generally involves determining subdomain adjacency and which voxels are in an interface. This computation is linear with respect to the number of voxels in the scene – that is, it scales linearly with the volume of the scene and with the 4th power of frequency. At higher frequencies, the interface and PML setup can consume several hours of time before any actual simulation steps are run.

In order to reduce the running time of the simulation, MPARD introduces a new preprocessing step in which the interfaces can be initialized offline. This preprocessing step occurs after any splitting and load

balancing, after the decomposition for the scene is final.

Additionally, this extra preprocessing step allows for further memory optimization in MPARD. With the kind of global metadata computed in the preprocessing, each core only needs to load the exact interfaces and PML regions it needs for its computations.

4.4 MPARD Pipeline

The MPARD pipeline introduces new preprocessing stages and a modified simulation stage. The pipeline overview can be found in Figure 3. The scene input (Figure 3(a)) is first voxelized (Figure 3(b)). Next, the rectangular decomposition fills the available air space with rectangular subdomains using a greedy algorithm (Figure 3(c)). Next, our new splitting algorithm that avoids degenerate subdomains splits rectangular regions that are larger than the volume threshold (Figure 3(d)). After interface regions are calculated (Figure 3(e)), we assign subdomains to cores using hypergraph partitioning (f). Finally, our solver reads the preprocessing data and runs the simulation for a set number of time steps.

5 Implementation

The following section discuss the implementation details of MPARD and the scalability problems introduced in section 4.

5.1 Hypergraph partitioning

We implemented a hypergraph partitioning scheme through the PaToH (Partitioning Tool for Hypergraphs) library in order to minimize core-to-core communication and equally distribute computation load across all cores[9]. There are many customizable parameters to adjust the details of how nodes are partitioned; we opted to use the "connectivity" (`PATOH_CONPART`) and "quality" metrics (`PATOH_SUGPARAM_QUALITY`), which sacrifice speed for better partitioning. We repeatedly execute this heuristic until a satisfactory subdomain assignment is achieved (no unassigned cores or subdomains).

We pass the rectangular regions into the PaToH partitioner as vertices with weights equal to the respective subdomain volumes. The hyperedges are defined by the interfaces connecting separate subdomains. By executing PaToH hypergraph partitioning onto this representational graph, the resulting core assignments will tend to prioritize low data transfer over cores, since it attempts to assign vertices in close proximity to a shared core. This also allows for a balanced load distribution by assigning each core roughly equal volume, which is linearly related to computation time.

5.2 Preprocessing pipeline

Both hypergraph partitioning and the subdomain splitting are implemented in the preprocessor stage. This is possible because the scenes targeted by MPARD are static. Since we know the 3D arrangement of subdomains and interfaces beforehand. As a result, we can avoid run-time simulation costs of dynamically updating the hypergraph or dynamically subdividing rectangular regions.

Our preprocessing pipeline in MPARD is separated into four stages: voxelization, decomposition, core allocation/splitting, and interface/PML preprocessing.

5.2.1 Voxelization

The voxelization stage takes in a triangle mesh representing the environment in which we want to compute the sound propagation. Because MPARD targets large and high frequency scenes that may consume a large amount of memory, we use a CPU method for voxelization. We implement an accurate and minimal method introduced by Huang et al. [18].

The spatial discretization for the voxelization is determined by the minimum simulated wavelength and the required number of spatial samples per wavelength, which is typically between 2 and 4 [24]. Therefore, the voxelization only needs to be run once per desired maximum frequency.

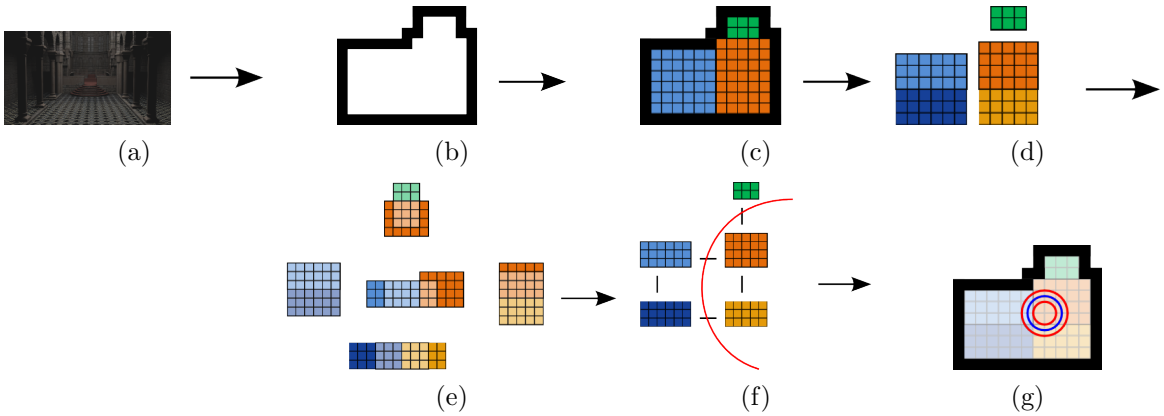


Figure 3: The MPARD pipeline. The input geometry (a) is voxelized in the first step (b). The rectangular decomposition step divides the domain into multiple non-overlapping subdomains (c). The splitting step then splits these subdomains when they are greater than the volume threshold Q (d). These partitions are then processed by the interface initialization stage which computes interface metadata (e). The final preprocessing stage allocates subdomains to nodes using the hypergraph partitioning (f). Finally, the simulation is run (g).

5.2.2 Decomposition

The decomposition stage then reads the voxel field and determines the location of the different cuboidal subdomains. The process is a greedy approach, attempting to expand each rectangular subdomain into as large a volume as possible under the constraints of the wall voxels.

At very high frequencies, such as 10 kHz, this process can take several days to complete but only needs to run once for a given voxel input.

5.2.3 Core allocation and subdomain splitting

The next stage of the preprocessing is the core allocation and subdomain splitting stage. In addition to a decomposition computed in the previous stage, this step also requires the number of cores the solver will run on. This stage uses the input values to compute a hypergraph partitioning for the decomposition in addition to splitting any rectangular regions that have volumes greater than the volume threshold Q.

The core allocation stage then determines the assignment of subdomains to cores by using the hypergraph partitioning assignment or alternatively a simple bin-packing algorithm. This load balancing step ensures that each core has a roughly equal amount of work to complete during the acoustic simulation.

The allocation and splitting stage must be run for each decomposition for each desired core configuration.

5.2.4 Interface and PML preprocessing

The final stage of preprocessing computes interfaces and creates PML regions from wall voxels. This stage takes as input a modified decomposition from the core allocation and splitting stage in addition to a refinement parameter r that can be used to subdivide voxels in the final acoustic simulation. This allows us to run at r times the frequency the decomposition was run at. However, this is at the expense of some accuracy where high frequency geometric features of the scene that may affect sound propagation cannot be accurately represented.

One additional caveat of the interface and PML preprocessing file is file read performance in the simulator. The interface file can be several GBs in size,

and thousands of CPU cores reading the file can cause a bottleneck. As a solution, we use the file striping feature of the Lustre file system [32] to increase file read performance over all cores.

The interface and PML initialization stage only needs to be run once for each core configuration.

6 Results and analysis

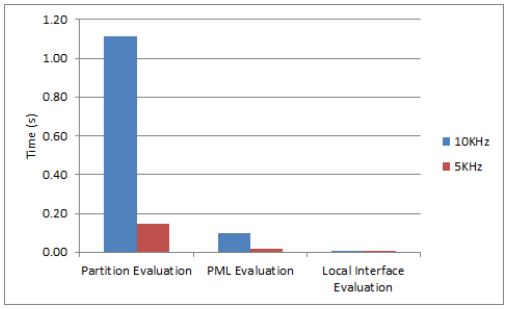
Our method was tested on two computing clusters: the large-scale Blue Waters supercomputer [7] at the University of Illinois and the UNC KillDevil cluster. Blue Waters is one of the world's leading compute clusters, with 362240 XE Bulldoze cores and 1.382 PB of memory. The KillDevil cluster has 9600 cores and 41 TB of memory.

(a) Sibenik Cathedral Scene

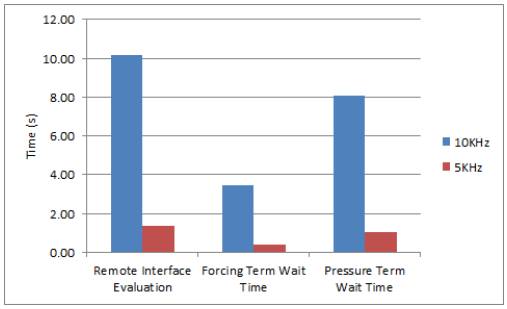
(b) Village Scene

Figure 4: The scenes used in our experiments.

Our primary experiments were performed on the Sibenik Cathedral scene and the Village scene (Figure 1). Both scenes provide a challenge for the underlying ARD solver. Cathedral has many curved surfaces, creating very small rectangular regions in the rectangular decomposition. Additionally, the large areas in the center of the cathedral creates a very large rectangular regions. Furthermore, the size of the scene is around $20\,000\,\mathrm{m}^3$, making communicating the sound propagation of the scene at high frequencies with wave-based methods very challenging. For example, a 10 kHz voxelization of the cathedral has almost 4 billion voxels. On the other hand, Village is mostly a large open area with a few buildings (Figure 4). Village is also much larger than cathedral. The size of the scene ($362\,000\,\mathrm{m}^3$) presents many computational challenges, particularly with computing interfaces. The scene contains over 100GB of interface data compared to 40GB in the 10 kHz Cathedral scene.



(a) Local Timings



(b) Communication Timings

Figure 5: The average running time of each stage of our solver on a 10 kHz scene compared to the 5 kHz scene. These results were obtained on 1024 cores.

We were able to run the MPARD solver on the cathedral scene up to 10 kHz. Figure 5 shows the average running time of each stage of our algorithm on this scene in comparison to the 5 kHz. The wait time for interface terms shows the necessity for optimizing communication at higher frequencies.

Scene Name	Volume	Frequency	Number of Triangles
Cathedral	19 177 m^3	5 kHz, 10 kHz	55415
Village	362 987 m^3	1.5 kHz	358

Table 1: Dimensions and complexity of the scenes used in our experiments.

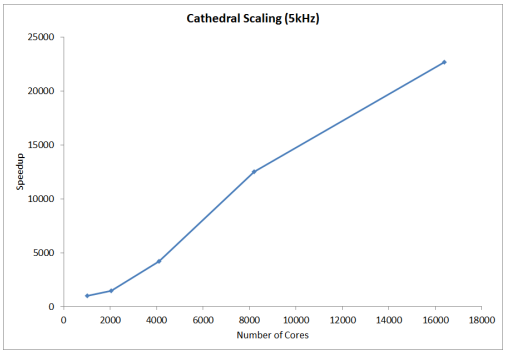


Figure 6: Scalability results from 1024 to 16384 cores on the 5 kHz cathedral scene. We obtain close-to-linear scaling in this result. The base speedup is 1024 on 1024 cores since the scene will not fit in memory on a lower number of cores.

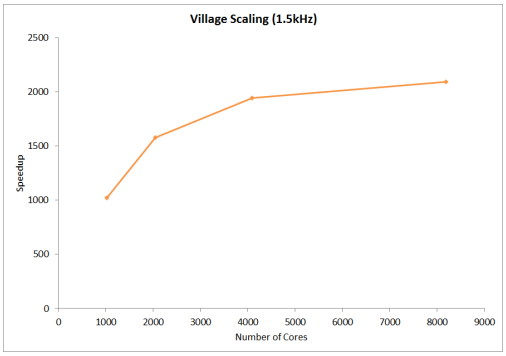


Figure 7: Scalability results from 1024 to 8192 cores on the 1.5 kHz village scene. We obtain sublinear scaling in this result. The base speedup is 1024 on 1024 cores since the scene will not fit in memory on a lower number of cores.

6.1 Scalability results

The primary scalability experiment was done on the cathedral scene at 5 kHz. The experiment was executed on the Blue Waters supercomputer up to 16384 cores. The main purpose of this experiment is to understand the performance of MPARD at very high number of cores. Figure 6 shows the performance of MPARD on the cathedral scene for 1024 cores all the way up to 16384 cores. With this kind of compute power, we are able to compute each time step on Cathedral in 0.193 75 s for a 5 kHz scene.

The Village scene also shows scalability up to 8192 cores. The challenge in the Village scene is the compute time of the interfaces. We show sublinear scaling in this case, with compute times as fast as 0.6761 s for a 1.5 kHz scene.

Figure 8 shows a plot of how the interface area generated by our splitting scheme increases as the number of cores used in the splitting algorithm increases. A comparison between three different test scenes was done.

6.2 Comparisons and benefits

In comparison to standard methods like FDTD, ARD does not require as fine of a computational grid. Traditional FDTD methods generally require a spatial discretization that is 1/10 the minimum wavelength (although the low-dispersion methods listed in Section 2.3 aim to lower this requirement). In comparison ARD can use a much coarser grid size, around 1/2.6 times the minimum wavelength [24, 21]. This means that ARD can inherently be $24-50$ times more memory efficient than FDTD and up to $75-100$ times faster.

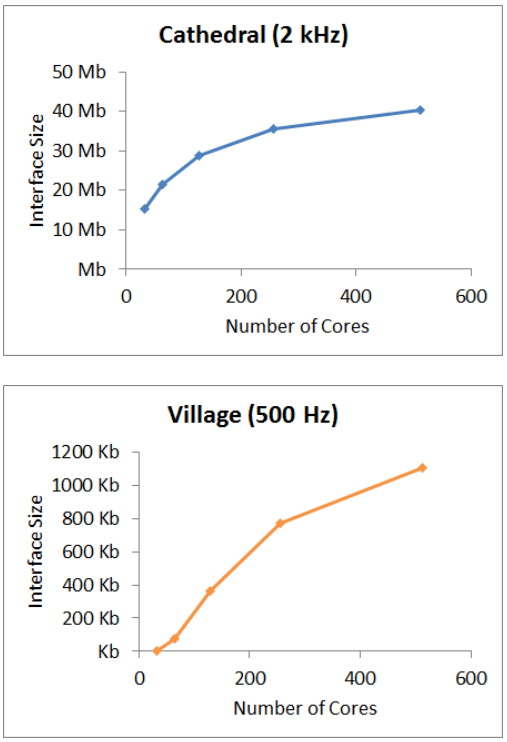


Figure 8: Interface area generated by our splitting scheme for three different scenes. The interface area tapers off as the number of cores increases.

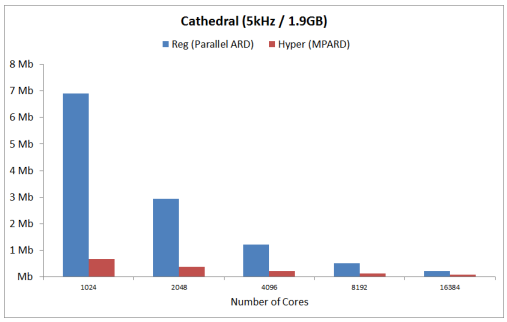


Figure 9: Communication cost comparison between the hypergraph and bin-packing approach on a 5 kHz cathedral scene.

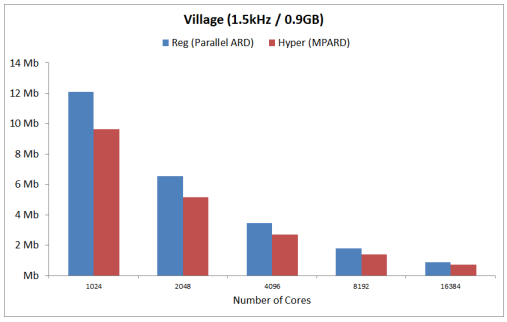


Figure 10: Communication cost comparison between the hypergraph and bin-packing approach on a 1.5 kHz village scene.

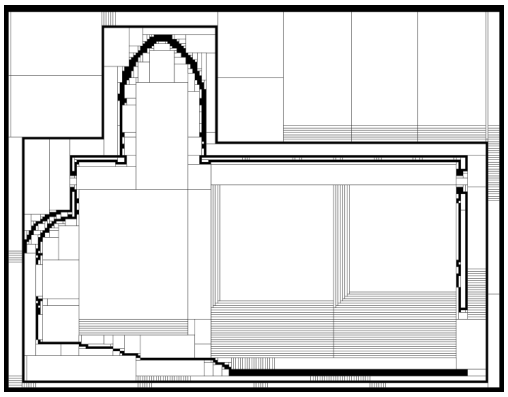
6.2.1 Comparsion with GPU ARD

MPARD provides advantages over previous GPU parallel ARD approaches. Large scenes at high frequencies require terabytes of memory that are easily available on large compute clusters but are not available on GPUs [21]. Secondly, MPARD scales over a much larger number of cores while GPU ARD is limited to a single machine and a shared memory architecture.

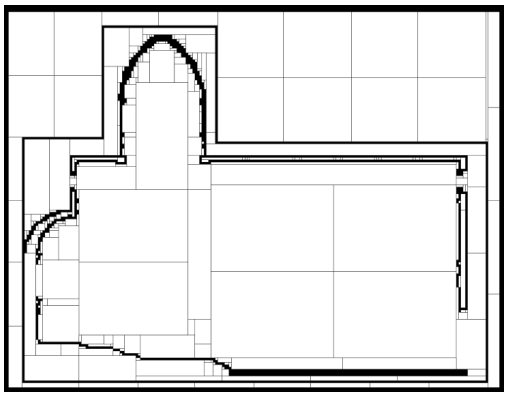
6.2.2 Comparison with Parallel ARD

Communication costs In order to examine the benefits of hypergraph partitioning, the total size of all messages sent during a single simulation time step of both scenes was computed. The scenes had cores assigned through the old bin-packing approach and the other had cores assigned through the new hypergraph partitioning approach [22]. Figure 9 shows the results of this experiment on a 5 kHz scene where the hypergraph partitioning approach has a 10x reduction in communication costs for 1024 cores and a 3x reduction for 16384 cores. The village scene, half as large in memory also had a reduction in communication costs (Figure 10.

Subdomain splitting Figure 11 shows the difference in decomposition between the old conservative approach and the new subdivision approach. Both scene decompositions were run on 1024 cores to show the difference between the two approaches at a high number of cores. The parallel ARD approach creates a series of very thin rectangles along the edge of the

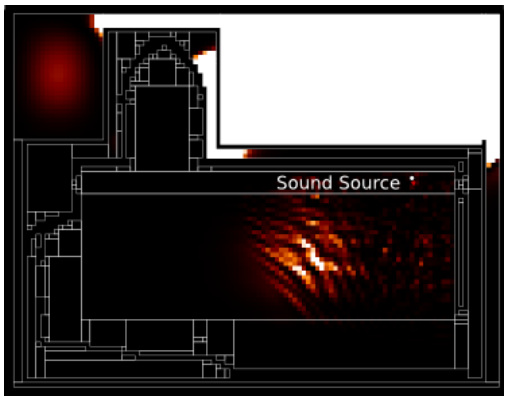


(a) Parallel ARD

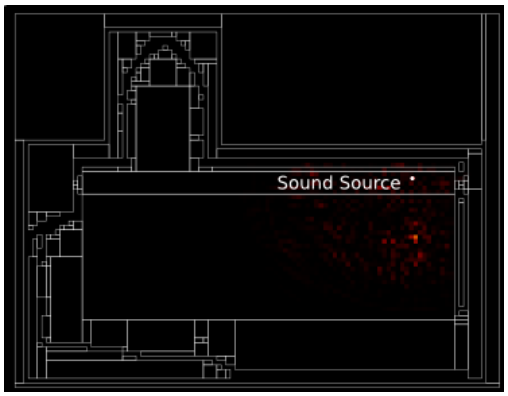


(b) MPARD

Figure 11: 2D slice comparison between parallel ARD and MPARD splitting methods. Notice the series of thin rectangles that are generated in parallel ARD [22] and can result in numerical stability problems. MPARD can overcome these issues. This result was computed on a decomposition for 1024 cores on a 500 Hz scene.



(a) Parallel ARD



(b) MPARD

Figure 12: Mean squared error comparison between the two splitting scenes as compared to a reference implementation that does not do splitting. The decomposition shown is the reference (non-split) decomposition.

interface while the new splitting scheme creates more cuboidal subdomains.

A comparison of the relative error amounts between parallel ARD and MPARD can be found in Figure 12. Each pressure field was compared to a reference field generated without splitting. The window of the slice shown was taken after a Gaussian pulse traversed the series of interfaces. It is clear that the new splitting approach reduces the amount of error generated by the interfaces.

7 Conclusion and future work

MPARD is a massively parallel approach showing scalability up to 16000 cores and is capable of calculating pressure fields for large scenes with frequencies up to 10 kHz. MPARD introduces several improvements over parallel ARD, reducing communication

cost by assigning cores with a hypergraph partitioning scheme, providing better numerical stability for higher numbers of cores, and implementing an extended preprocessing pipeline that reduces the usage of valuable cluster resources for redundant calculations.

In the future, we would like to evaluate the architectural acoustics for large indoor and outdoor scenes, and use them to improve their designs. The computational power of MPARD can be exploited to automatically calculate optimal scene geometry or surface materials for a desired acoustic configuration of a scene.

Additionally, although we have shown the ability to compute very high frequency scenes, we would like to test the method on more scenes including even larger architectural and outdoor scenes at high frequencies. Increasing the scalability of interface computations will help us do this. We also would like to adapt MPARD to work in heterogeneous environments with a spatially varying speed of sound.

Finally, although we examine the error impact of ARD's interfaces to some extent, a thorough error analysis is required. This analysis can better inform the splitting algorithm MPARD uses for load balancing.

Acknowledgments

This research was supported in part by ARO Contracts W911NF-13-C-0037, W911NF-14-1-0437 and the National Science Foundation (NSF awards 1345913 and 1456299). This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. We want to thank Allan Porterfield, Alok Meshram, Lakulish Antani, Anish Chandak, Joseph Digerness, Alban Bassuet, Keith Wilson, and Don Albert for useful discussions and models.

References

[1] A. Alghamdi, A. Ahmadia, D. I. Ketcheson, M. G. Knepley, K. T. Mandli, and L. Dalcin. Petclaw: a scalable parallel nonlinear wave propagation solver for python. In *Proceedings of the 19th High Performance Computing Symposia*, HPC '11, pages 96–103, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[2] J. B. Allen and D. A. Berkley. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*, 65(4):943–950, April 1979.

[3] H. Bao, J. Bielak, O. Ghattas, L. F. Kallivokas, D. R. O'Hallaron, J. R. Shewchuk, and J. Xu. Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers. *Computer methods in applied mechanics and engineering*, 152(1):85–102, 1998.

[4] M. Bernacki, L. Fezoui, S. Lanteri, and S. Piperno. Parallel discontinuous galerkin unstructured mesh solvers for the calculation of three-dimensional wave propagation problems. *Applied mathematical modelling*, 30(8):744–763, 2006.

[5] M. A. Bhandarkar and L. V. Kalé. A parallel framework for explicit fem. In *High Performance ComputingHiPC 2000*, pages 385–394. Springer, 2000.

[6] S. Bilbao. Modeling of complex geometries and boundary conditions in finite difference/finite volume time domain room acoustics simulation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(7):1524–1533, 2013.

[7] B. Bode, M. Butler, T. Dunning, W. Gropp, T. Hoe-fler, W.-m. Hwu, and W. Kramer. The blue waters super-system for super-science. In C. Jeffrey S . Vetter and H. 2013, editors, *Contemporary HPC Architectures*. Sitka Publications, 2012.

[8] C. A. Brebbia. *Boundary Element Methods in Acoustics*. Springer, 1 edition, Oct. 1991.

[9] Ü. Çatalyürek and C. Aykanat. Patoh (partitioning tool for hypergraphs). In *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer, 2011.

[10] T. F. Chan and O. B. Widlund. Domain decomposition algorithms. In *Acta Numerica*, pages 61–143. 1994.

[11] J.-T. Chen, Y.-T. Lee, and Y.-J. Lin. Analysis of mutiple-shepers radiation and scattering problems by using a null-field integral equation approach. *Applied Acoustics*, 71(8):690–700, 2010.

[12] M. J. Crocker. *Handbook of Acoustics*. Wiley-IEEE, 1998.

[13] R. Diekmann, U. Dralle, F. Neugebauer, and T. Römke. Padfem: a portable parallel fem-tool. In *High-Performance Computing and Networking*, pages 580–585. Springer, 1996.

[14] B. Engquist and O. Runborg. Computational high frequency wave propagation. *Acta Numerica*, 12:181–266, 2003.

[15] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West. A beam tracing approach to acoustic modeling for interactive virtual environments. In *Proc. of ACM SIGGRAPH*, pages 21–32, 1998.

[16] T. Funkhouser, N. Tsingos, and J.-M. Jot. Survey of Methods for Modeling Sound Propagation in Interactive Virtual Environment Systems. *Presence and Teleoperation*, 2003.

[17] C. Guiffaut and K. Mahdjoubi. A parallel fdtd algorithm using the mpi library. *Antennas and Propagation Magazine, IEEE*, 43(2):94–103, 2001.

[18] J. Huang, R. Yagel, V. Filippov, and Y. Kurzion. An accurate method for voxelizing polygon meshes. In *Volume Visualization, 1998. IEEE Symposium on*, pages 119–126. IEEE, 1998.

[19] K. Kowalczyk and M. van Walstijn. Room acoustics simulation using 3-d compact explicit fdtd schemes. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(1):34–46, 2011.

[20] H. Kuttruff. *Room acoustics*. CRC Press, 2009.

[21] R. Mehra, N. Raghuvanshi, L. Savioja, M. C. Lin, and D. Manocha. An efficient gpu-based time domain solver for the acoustic wave equation. *Applied Acoustics*, 73(2):83 – 94, 2012.

[22] N. Morales, R. Mehra, and D. Manocha. A parallel time-domain wave simulator based on rectangular decomposition for distributed memory architectures. *Applied Acoustics*, 97:104–114, 2015.

[23] S. Operto, J. Virieux, P. Amestoy, J.-Y. LExcellent, L. Giraud, and H. B. H. Ali. 3d finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study. *Geophysics*, 72(5):SM195–SM211, 2007.

[24] N. Raghuvanshi, R. Narain, and M. C. Lin. Efficient and Accurate Sound Propagation Using Adaptive Rectangular Decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):789–801, 2009.

[25] Y. S. Rickard, N. K. Georgieva, and W.-P. Huang. Application and optimization of pml abc for the 3-d wave equation in the time domain. *Antennas and Propagation, IEEE Transactions on*, 51(2):286–295, 2003.

[26] J. Saarelma and L. Savioja. An open source finite-difference time-domain solver for room acoustics using graphics processing units. *Acta Acustica united with Acustica*, 2014.

[27] F. Saied and M. J. Holst. Multigrid methods for computational acoustics on vector and parallel computers. *Urbana*, 51:61801, 1991.

[28] S. Sakamoto, A. Ushiyama, and H. Nagatomo. Numerical analysis of sound propagation in

rooms using the finite difference time domain metho d. *The Journal of the Acoustical Society of America*, 120(5):3008, 2006.

[29] L. Savioja. Real-time 3d finite-difference time-domain simulation of low-and mid-frequency room acoustics. In *13th Int. Conf on Digital Audio Effects*, volume 1, page 75, 2010.

[30] L. Savioja, J. Backman, A. Järvinen, and T. Takala. Waveguide mesh method for low-frequency simulation of room acoustics. 1995.

[31] L. Savioja and V. Valimaki. Reducing the dispersion error in the digital waveguide mesh using interpolation and frequency-warping techniques. *Speech and Audio Processing, IEEE Transactions on*, 8(2):184–194, 2000.

[32] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, volume 2003, 2003.

[33] J. Sheaffer and B. Fazenda. Wavecloud: an open source room acoustics simulator using the finite difference time domain method. *Acta Acustica united with Acustica*, 2014.

[34] P. Sypek, A. Dziekonski, and M. Mrozowski. How to render fdtd computations more effective using a graphics accelerator. *Magnetics, IEEE Transactions on*, 45(3):1324–1327, 2009.

[35] L. L. Thompson. A review of finite-element methods for time-harmonic acoustics. *The Journal of the Acoustical Society of America*, 119(3):1315–1330, 2006.

[36] A. Vaccari, A. Cala'Lesina, L. Cristoforetti, and R. Pontalti. Parallel implementation of a 3d subgridding fdtd algorithm for large simulations. *Progress In Electromagnetics Research*, 120:263–292, 2011.

[37] S. Van Duyne and J. O. Smith. The 2-d digital waveguide mesh. In *Applications of Signal Processing to Audio and Acoustics, 1993. Final Program and Paper Summaries., 1993 IEEE Workshop on*, pages 177–180. IEEE, 1993.

[38] S. Wang, M. V. de Hoop, J. Xia, and X. S. Li. Massively parallel structured multifrontal solver for time-harmonic elastic waves in 3-d anisotropic media. *Geophysical Journal International*, 191(1):346–366, 2012.

[39] C. J. Webb and S. Bilbao. Binaural simulations using audio rate fdtd schemes and cuda. In *Proc. of the 15th Int. Conference on Digital Audio Effects (DAFx-12), York, United Kingdom*, 2012.

[40] W. Yu, Y. Liu, T. Su, N.-T. Hunag, and R. Mittra. A robust parallel conformal finite-difference time-domain processing package using the mpi library. *Antennas and Propagation Magazine, IEEE*, 47(3):39–59, 2005.

[41] W. Yu, X. Yang, Y. Liu, L.-C. Ma, T. Sul, N.-T. Huang, R. Mittra, R. Maaskant, Y. Lu, Q. Che, et al. A new direction in computational electromagnetics: Solving large problems using the parallel fdtd on the bluegene/l supercomputer providing teraflop-level performance. *Antennas and Propagation Magazine, IEEE*, 50(2):26–44, 2008.