Solving 3D Anisotropic Elastic Wave Equations on Parallel GPU Devices[a]

[a]Published in Geophysics, 78, F7-F15, (2013)

Robin M. Weiss and Jeffrey Shragge**

ABSTRACT

Efficiently modeling seismic datasets in complex 3D anisotropic media by solving the 3D elastic wave equation is an important challenge in computational geophysics. Using a stress-stiffness formulation on a regular grid, we present a 3D finite-difference time-domain (FDTD) solver using a 2^{nd}-order temporal and 8^{th}-order spatial accuracy stencil that leverages the massively parallel architecture of graphics processing units (GPUs) to accelerate the computation of key kernels. The relatively small memory of an individual GPU limits the model domain sizes that can be computed on a single device. To circumvent this constraint and move toward modeling industry-sized 3D anisotropic elastic data sets, we parallelize computation across multiple GPU devices by using domain decomposition and, for each time step, employing an inter-device communication protocol to exchange data values falling within interior boundaries of each sub-domain. For two or more GPU devices within a single compute node, we use direct peer-to-peer (i.e., GPU-to-GPU) communication, while for networked nodes we employ message-passing interface (MPI) directives to route data over the network. Our 2D GPU-based anisotropic elastic modeling tests achieved a 10× speedup relative to an OpenMP CPU implementation run on an eight-core machine, while our 3D tests using dual GPU devices produced up to a 28× speedup. The performance boost afforded by the GPU architecture allows us to model seismic data for 3D anisotropic elastic models at lower hardware cost and in less time than previously possible.

INTRODUCTION

Efficiently calculating 3D elastic wavefields and data with algorithms capable of handling large-scale acquisition geometries (e.g., wide-azimuth surveys) and/or complex anisotropic media [e.g., horizontal transversely isotropic (HTI) or orthorhombic symmetries] remains a significant computational challenge. While there are a number of commercially available modelling packages that satisfy these requirements, generally they are aimed at users of high-performance computing (HPC) facilities and require significant dedicated cluster computing resources that remain too expensive for smaller-scale research and development groups. Based on these arguments, there

is a strong impetus for developing freely available, open-source, 3D elastic modeling solutions that can be run efficiently without the need for significant computing infrastructure.

Within the last half decade there has been a significant increase of interest in the exploration geophysics community of using general-purpose graphics processing units (GPUs) as accelerators for key seismic modelling, imaging and inversion kernels [e.g., Ohmer et al. (2005); Kuzma et al. (2007); Morton et al. (2008); Foltinek et al. (2009)]. Owing to their wider memory bandwidth and often two orders of magnitude more processors, albeit slower and lighter-weight than central processing unit (CPU) architectures, GPUs have emerged as an excellent parallel computing platform for problems characterized by a single-instruction multiple-data (SIMD) pattern. By allowing many thousands of GPU threads to run concurrently, significant speedups of SIMD-type problems on GPUs relative to CPUs have been documented in numerous studies in many branches of applied computer science (Pharr and Fernando, 2005; Nguyen, 2007).

For finite-difference time-domain (FDTD) solutions of wave equations (WEs) that form the basis of many seismic modeling, migration and velocity inversion applications, a number of studies have developed compact wave-equation FD stencils and algorithmic strategies that are well-suited for GPU implementation. Micikevicius (2009) and Abdelkhalek et al. (2009) discuss GPU implementations of the 3D acoustic WE. Komatitsch et al. (2010) discuss a GPU-based finite-element formulation of 3D anisotropic elastic wave propagation. Nakata et al. (2011) present results for solving the 3D isotropic elastic WE on multiple GPUs. These studies present impressive GPU runtimes of roughly one-tenth to one-twentieth of their corresponding multi-core CPU-based implementations.

When aiming to compute seismic data and/or wavefields for realistic 3D model sizes (i.e., $N^3 = 1000^3$ where N is sample number in one dimension), though, the relatively small global memory available on an individual GPU card relative to a multi-core CPU chip ($\leq$ 6 GBytes versus $\gg$ 6 GBytes, respectively) makes single-device GPU solutions of the 3D elastic WE intractable for realistic industry-sized models. This issue is compounded for 3D anisotropic media because the additional stiffness components (or equally anisotropic parameters) must also be held in memory. Fortunately, this issue can be addressed by parallel computing strategies that utilize domain decomposition to divide the computation across multiple GPU devices that work in concert through a communication protocol (Micikevicius, 2009; Nakata et al., 2011).

Starting with the *ewefd2d* and *ewefd3d* modeling codes that are freely available in the archives of the Madagascar project (Fomel, 2012), we develop 2D/3D GPU-based elastic wavefield modeling codes using NVIDIA's CUDA application programming interface (API). We similarly use a domain-decomposition strategy and present two different protocols for communicating between GPU devices. For individual nodes containing multiple GPUs (herein termed a consolidated node), we use direct peer-to-peer (P2P) communication that allows GPU devices situated on the same PCIe

bus to communicate without requiring intermediate data staging in system memory. In a distributed computing environment the P2P strategy does not work because GPUs do not share a common PCIe bus , and we must turn to the comparatively slower message-passing interface (MPI) to handle inter-device communication.

Our goals in communicating the results of our modeling efforts - and the code itself - are twofold. Firstly, to present a 3D FDTD elastic modeling code capable of handing various transversely isotropic (TI) symmetries that can scale to computational domains sufficiently large to model realistic 3D acquisition geometries without requiring massive CPU clusters to finish modeling runs in a "reasonable" duration of time. Secondly, to release a set of open-source GPU-based modeling codes and reproducible examples through the Madagascar project for both educational purposes and to facilitate innovation and collaboration throughout the geophysics community.

We begin by discussing the governing equations for 3D elastic-wave propagation in the stress-stiffness formulation, and presenting the discritization approach adopted for a regular computational mesh. We then highlight the numerical algorithm and discuss a number of issues regarding the GPU implementation strategy, including domain decomposition and how we target multiple devices within consolidated and distributed computing environments. We then provide a number of reproducible 2D/3D modeling examples for different TI media, and present GPU-versus-CPU runtime and speedup metrics that demonstrate the utility of the GPU-based modeling approach.

THEORY

The equations governing elastic wave propagation in 3D transversely isotropic media, when assuming linearized elasticity theory and a stress-stiffness tensor formulation, are fairly straightforward to implement in a numerical scheme. Our goal is to develop finite-difference (FD) operators of 2^{nd}- and 8^{th}-order temporal and spatial accuracy, respectively, that are well-suited for GPU hardware by virtue of being a compact stencil with a regular memory access pattern.

The linear theory of elasticity [e.g., Landau and Lifshitz (1986)] establishes a relationship between a vector seismic wavefield displaced infinitesimally from rest and the dimensionless linear strain tensor. In indicial notation we write

$$\epsilon_{kl} = \frac{1}{2}\left[\partial_k u_l + \partial_l u_k\right], \qquad k,l = 1,2,3, \tag{1}$$

where ϵ_{kl} is an element of the linear strain tensor, ∂_k is the spatial derivative in the k^{th} direction, and u_l is the l^{th} component of wavefield displacement. Herein, we assume Cartesian geometry where the x-, y- and z-axes are represented by indices $i = 1,2,3$, respectively, and use summation notation for repeated indices.

The linear strain tensor, ϵ_{kl}, is related to the Cauchy stress tensor, σ_{ij}, through a constitutive relationship that describes the elastic material properties through a

fourth-order stiffness tensor c_{ijkl}:

$$\sigma_{ij} = c_{ijkl}\epsilon_{kl}. \tag{2}$$

The above equations can be combined into the equations of motion, derivable from Newton's second law, that describe wave propagation through an anisotropic elastic medium:

$$\rho\partial_{tt}^2 u_i = \partial_j \sigma_{ij} + F_i, \tag{3}$$

where F_i is the body force per unit volume (that can be implemented as an equivalent stress source), ρ is material density, and ∂_{tt}^2 is the second-order temporal derivative.

Numerical approach

Any numerical implementation of equations 1-3 requires specifying both a computational mesh and a numerical discritization scheme. We define a $N_x \times N_y \times N_z$ computational grid and use N_t time steps such that a grid point in space and time can be represented by quadruplet $[x, y, z|t] = [p\Delta x, q\Delta y, r\Delta z|n\Delta t]$, where the integer counters have the ranges $p = 1, N_x$, $q = 1, N_y$, $r = 1, N_z$ and $n = 1, N_t$. Continuous wavefield displacements at a given location are represented on the discritized grid as

$$u_i \left|_{x,y,z|t} \approx u_i^{p,r,q|n}. \right. \tag{4}$$

We approximate the first derivative ∂_j by a compact centered difference operator $D_x[\cdot]$ of 8^{th}-order accuracy (Trefethen, 1996)

$$\partial_x u_j \approx D_x[u_j^{p,q,r|n}] = \frac{1}{\Delta x}\sum_{\alpha=1}^{4} W_\alpha \left(u_j^{p+\alpha,q,r|n} - u_j^{p-\alpha,q,r|n}\right), \tag{5}$$

where W_α are polynomial weights given by $\mathbf{W} = \left[\frac{+4}{5}\ \frac{-1}{5}\ \frac{+4}{105}\ \frac{-1}{280}\right]$. Spatial difference operators $D_y[\cdot]$ and $D_z[\cdot]$ are specified similarly for the $y-$ and $z-$directional derivatives, and all three are used for derivatives of the constitutive relationship in equation 3 (i.e., $\partial_j c_{ijkl}$). We use a standard second-order accuracy approximation for the second time derivative in equation 3

$$\partial_{tt}^2 u_j \approx D_{tt} u_j^{p,q,r|n} = \frac{1}{\Delta t^2}\left[u_j^{p,q,r|n+1} - 2u_j^{p,q,r|n} + u_j^{p,q,r|n-1}\right]. \tag{6}$$

Inserting the above difference operators into equations 1-3 and rearranging terms leads to a FD scheme for calculating the unknown wavefield solution at the forward time step, $u_j^{p,q,r|n+1}$, given wavefield values at the current and previous time steps, $u_j^{p,q,r|n}$ and $u_j^{p,q,r|n-1}$, and stencil points neighboring $u_j^{p,q,r|n}$ in the x-, y- and z-directions. Figure 1 depicts the FD stencil constructed from the points about $u_j^{p,q,r|n}$ at the current time step.

These difference operators allow us to specify a time-stepping scheme to calculate wavefield displacements in 3D elastic anisotropic media throughout the model domain.

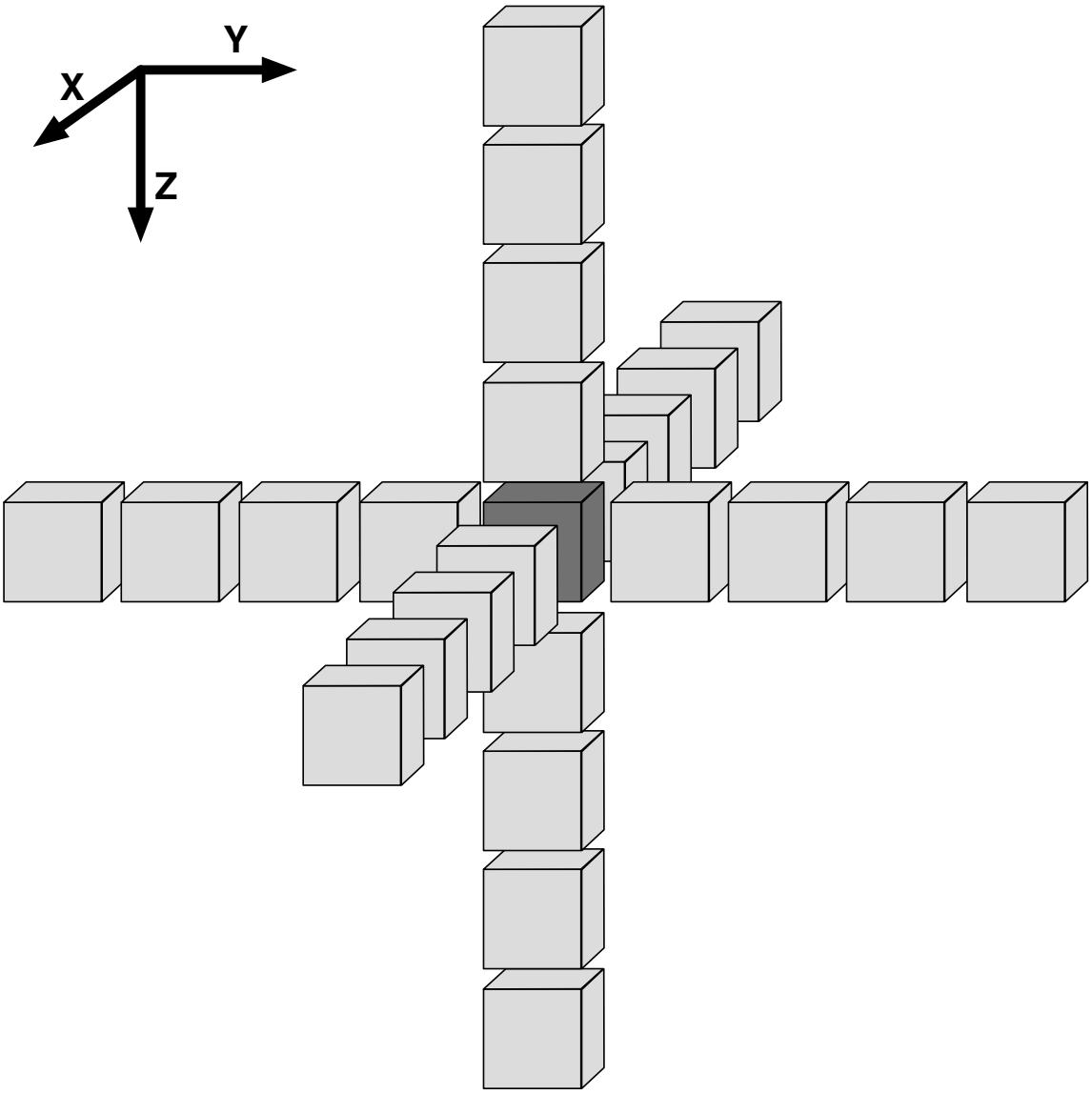


Figure 1: The 25 data points required to approximate a first derivative at the center shaded point using a FD stencil of 8^{th}-order accuracy.

However, additional work is required to treat both the free-surface boundary and to minimize the energy in non-physical exterior boundary reflections. Implementing free-surface boundary conditions is fairly straightforward on a uniform grid because a (topography-free) free surface can be placed directly on the boundary which, assuming the free-surface normal vector points in the z-direction, allows us to set $\sigma_{i3} = 0$ where $i = 1, 2, 3$. We treat the other boundaries using a cascade of two operators comprised of absorbing boundary conditions (ABCs) derived from a one-way wave equation (Clayton and Enquist, 1977) and an exponential-damping sponge layer (Cerjan et al., 1985) of at least 48 grid points.

Figure 2 presents the nine procedural steps in the 3D FDTD algorithm required for calculating each forward time-step: 1) Compute strains ϵ_{kl} from wavefield displacements according to equation 1; 2) Calculate stresses σ_{ij} from constitutive relationship c_{ijkl} and ϵ_{kl} according to equation 2; 3) Enforce the free-surface boundary condition (if required); 4) Inject a stress source (if not an acceleration source); 5) Compute acceleration from stress tensor (i.e., $RHS = \partial_j \sigma_{ij}$); 6) Inject an acceleration source (if not a stress source; $RHS \stackrel{+}{=} F_j$); 7) Compute the forward time step from current and previous wavefield values; 8) Apply boundary conditions through cascade of ABC and sponge operators; and 9) Output data/wavefield as required. The next section details our GPU implementation of these steps.

GPU IMPLEMENTATION

Each procedural step of our 3D FDTD algorithm is implemented as a GPU kernel function that is called at each time step by a CPU-based control thread. The computationally intensive steps of our algorithm are thus offloaded to the GPU (see Figure 2), where computation is greatly accelerated by concurrent execution of thousands of threads on the many hundred cores of the device.

Steps 1 and 5, the most computationally expensive steps of our algorithm, apply the FD stencil shown in Figure 1 at all points in the computational grid. The 8th-order stencil requires evaluating 25 data values per grid point, which potentially leads to a high number of transactions from the very high latency global memory. Minimizing the total number of these transactions is thus essential for maintaining high performance.

We adopt the method proposed in Micikevicius (2009) that minimizes global memory read redundancy and thereby mitigates the negative performance impact of high latency memory. This approach effectively optimizes the sharing of data between threads - thereby reducing the number of global memory transaction - by retrieving and storing a 2D plane of data in smaller but (roughly two orders of magnitude) faster shared memory and evaluating neighboring stencils concurrently. The 2D plane of threads is oriented perpendicular to the slowest-varying y-axis, thereby optimizing retrieval of values from global memory by reading from more contiguous data blocks. This 2D algorithm then repeats slice-by-slice as the computation progresses through

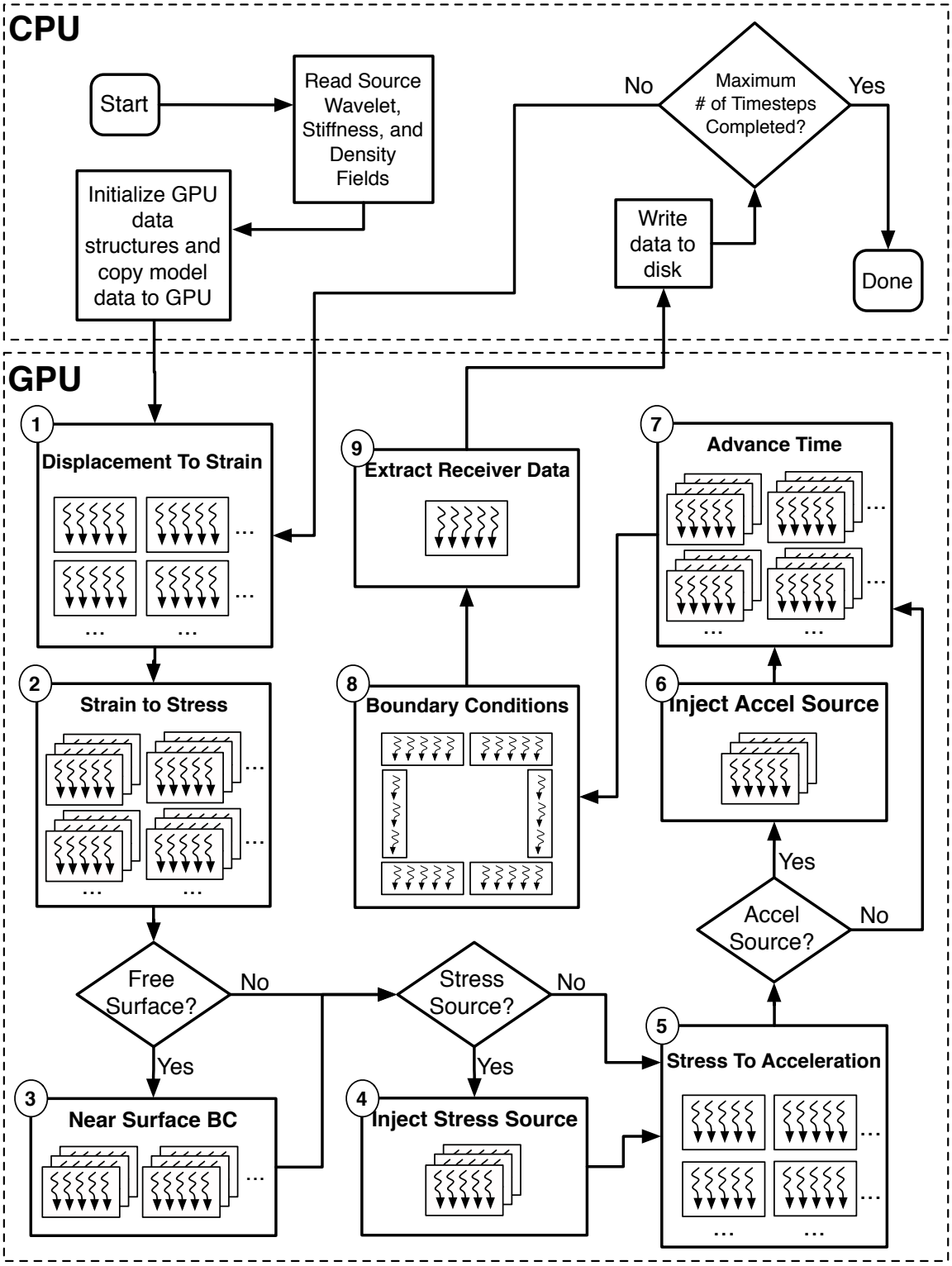


Figure 2: Control flow for FDTD algorithm implemented on a single GPU, including the division of tasks between the CPU host and GPU device.

the 3D volume.

The boundary condition kernel in Step 8 that applies a cascade of operators to treat the domain edge values also exhibits performance limitations due to memory access pattern. Wavefield values are stored in memory as a 1D array varying from continuous in the x-axis to a linear stride in the z-axis to a 2D plane offset in the y-axis. Accordingly, applying boundary conditions in the $z-y$ plane necessarily requires retrieving data from non-continuous memory locations leading to sub-optimal kernel performance. It remains to be determined whether this limitation can be circumvented through storing grid data in an alternative structure or in another form of GPU memory better optimized for 3D spatial locality.

The GPU kernels for Steps 2-4, 6-7 and 9 represent vector-vector operations that are highly efficient on GPU. We adopt a straightforward data-parallel scheme where one GPU thread per grid point is used to calculate the required values in a massively concurrent fashion. The high latency associated with global memory is hidden by the high degree of concurrency that allows computation to continue in some threads whilst others wait for memory transactions to complete. The use of shared memory for these kernels does not afford any acceleration because there is zero redundancy in memory transactions. Dimensions for thread blocks used in each kernel invocation were selected such that occupancy is maximized (as determined by NVIDIA's CUDA Occupancy Calculator). The selected values were then tuned experimentally with the NVIDIA Visual Profiler to identify the optimal configuration.

Multiple GPU Implementation

When solving the 3D elastic WE on large-scale model domains (i.e., $N_x \times N_y \times N_z > 300^3$), the limited global memory of an individual GPU precludes storage of the entire grid on a single device. To parallelize our 3D FDTD algorithm across multiple GPUs, we adopt a domain-decomposition scheme, illustrated in Figure 3, that divides the computational grid in the slowest varying y-axis direction and assigns the sub-domains to separate GPU devices. Each GPU individually executes Steps 1-9 on its assigned sub-domain, whilst CPU-based control threads coordinate the operations of the multiple devices, enable inter-device communication, and combine results to produce the output data.

Because the FD stencil in Figure 1 requires data from points extending four units in the forward and backward y-axis directions, the GPU threads that apply the stencil to points along the sub-domain edges may require data from a logically adjacent device. Therefore, boundary data from adjacent sub-domains must be exchanged between GPUs at every time step. This communication can be expensive due to the limited bandwidth of the PCIe bus within a node and/or the network connections between nodes in a distributed system.

In a consolidated multi-GPU computing environment where all devices share a common PCIe bus, the P2P protocol in NVIDIA's CUDA v4.0 (assuming a GPU with

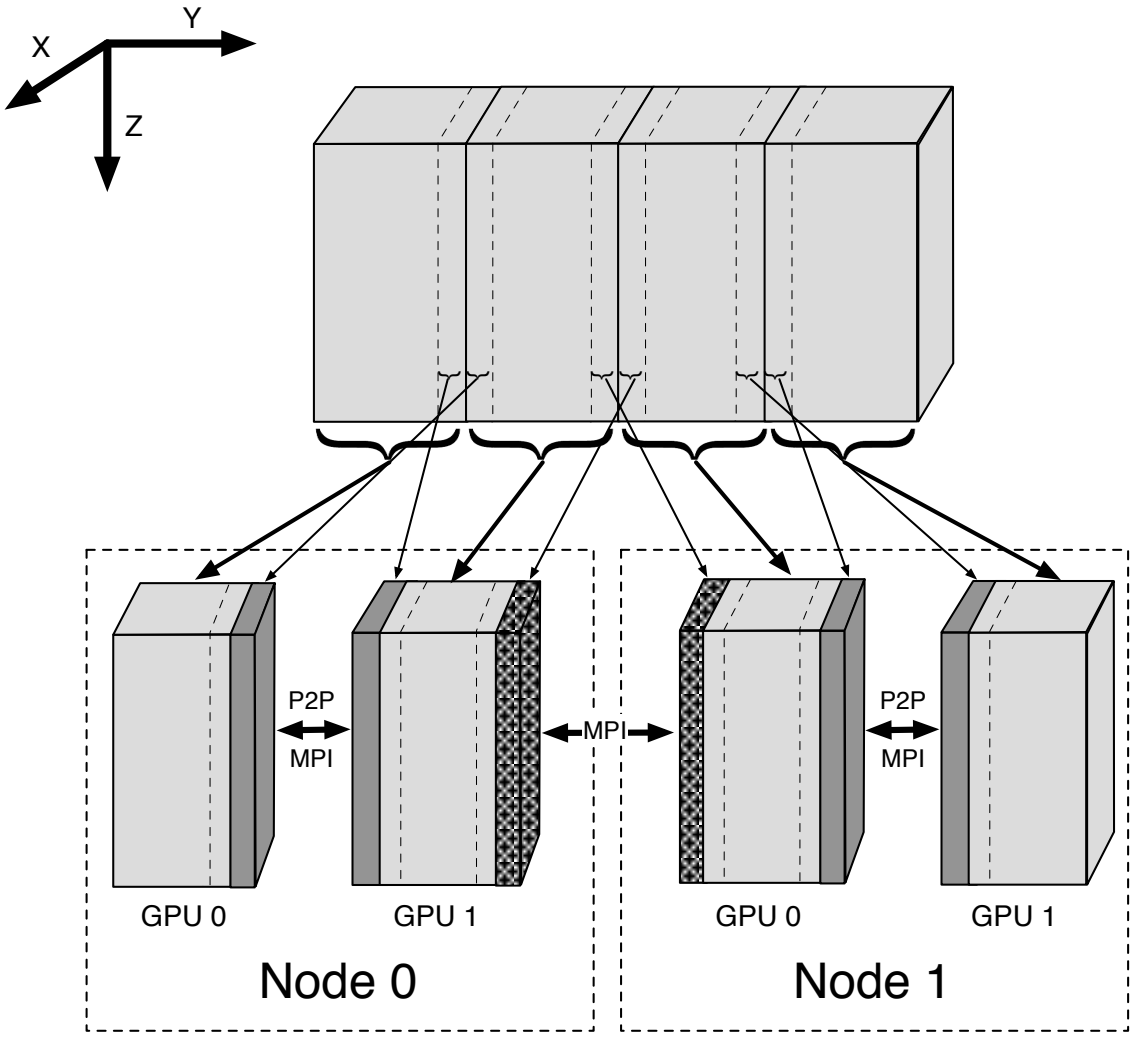


Figure 3: Schematic of the domain decomposition of the computational grid along the y-axis across four GPU devices shared equally in two compute nodes. Data in the interior boundary regions of each subdomain must be shared with their logical neighbor using either P2P or MPI in the gray area, and necessarily with MPI across the black hatched region.

compute capability of $\geq$2.0) can be used to directly exchange data between devices. For a distributed GPU environment, direct P2P communication is not possible as the devices neither share a common PCIe bus nor have direct access to the network. Therefore, as shown in Figure 4, the CPU-based control threads must use the MPI communication interface (or equivalent) to enable communication over the network.

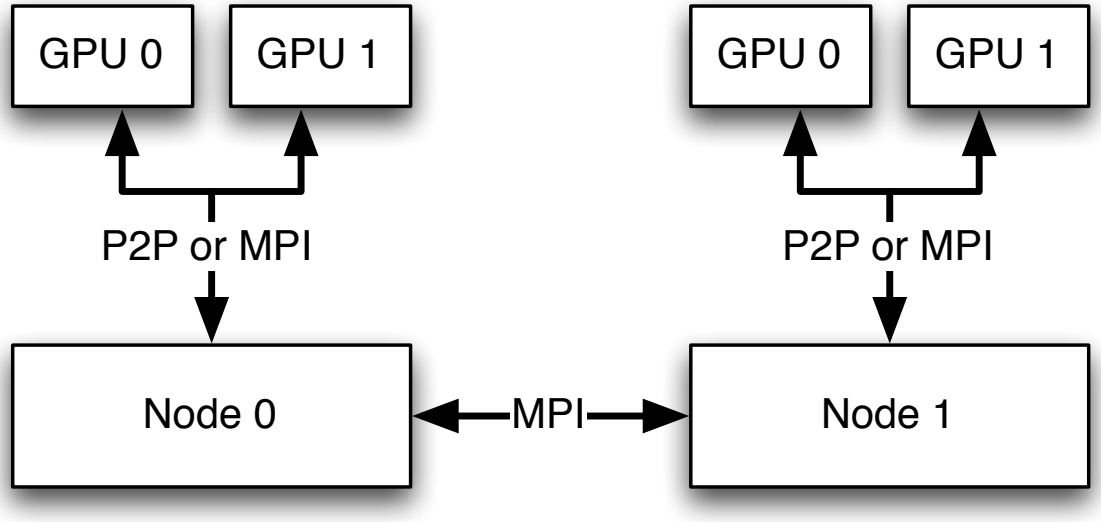


Figure 4: Schematic showing distributed computing environment where MPI is used to communicate between nodes, and either MPI or (hybrid) P2P communication is used within a node.

By eliminating system memory allocation and copy overhead, direct P2P memory transfer reduces total compute time by $10-15\%$ when compared to using MPI-based send/receive commands within a consolidated compute node. In a hybrid environment of distributed multi-GPU compute nodes, a hybrid communication scheme can be adopted where each node uses a single CPU control thread for managing local GPUs (utilizing P2P transfers between devices) while communicating when necessary with remote GPUs via the MPI sub-system. However, for the sake of simplifying our released codes and reproducible examples, we refrain from discussing this situation in more detail herein.

MODELING EXAMPLES

In this section we demonstrate the utility of the GPU-based modeling approach by presenting reproducible numerical tests using the 2D/3D elastic FDTD codes for different TI models. The first set of tests involve applying the FDTD code to 2D homogeneous isotropic and VTI media. We define our test isotropic medium by P- and S-wave velocities of v_p=2.0 km/s and v_s=$v_p/\sqrt{3}$, and a density of ρ=2000 kg/m^3. The VTI medium uses the same v_p, v_s and ρ, but includes three Thomsen parameters (Thomsen, 1986) of $[\epsilon_1, \delta_1, \gamma_1] = [0.2, -0.1, 0.2]$. Figures 5a-(b) present the vertical

and horizontal components, u_z and u_x respectively, of the 2D isotropic impulse response tests, while Figures 5c-(d) show the similar components for the VTI model. Both tests generate the expected wavefield responses when compared to the CPU-only code results.

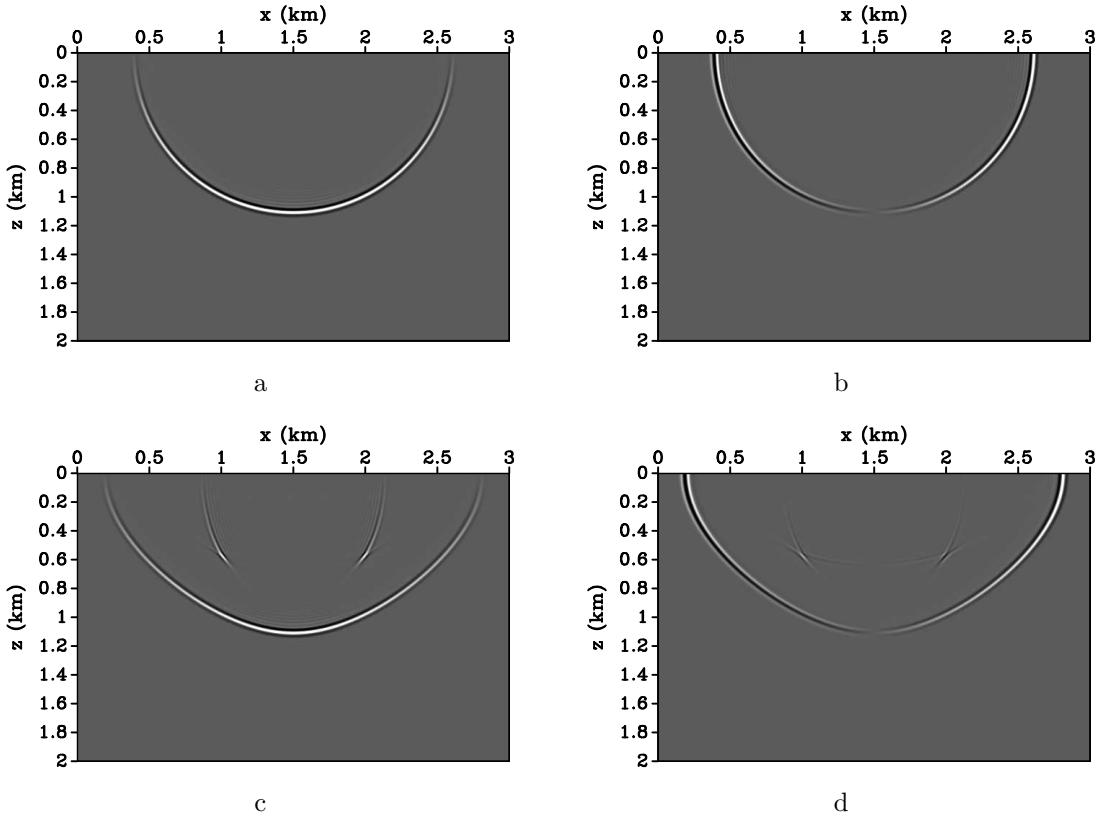
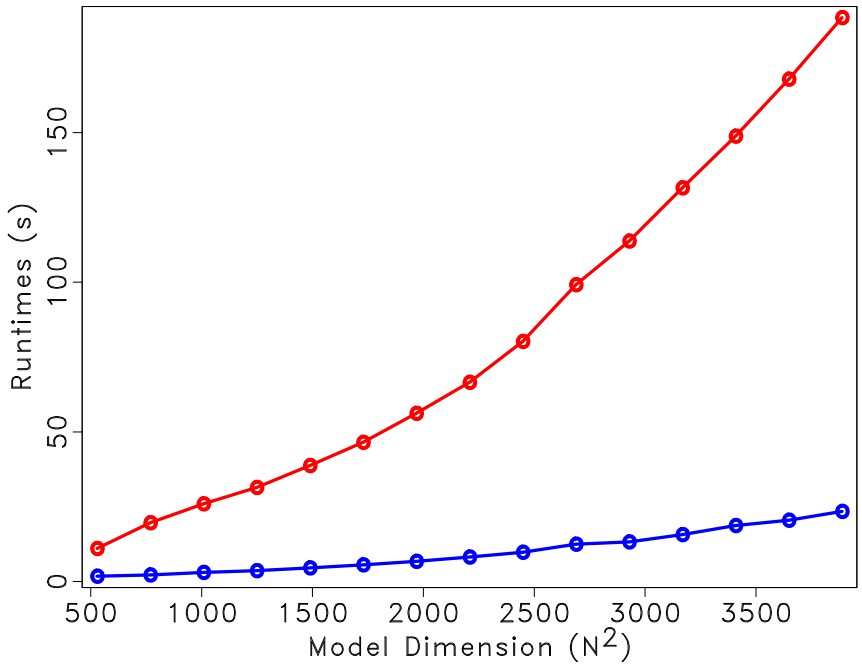


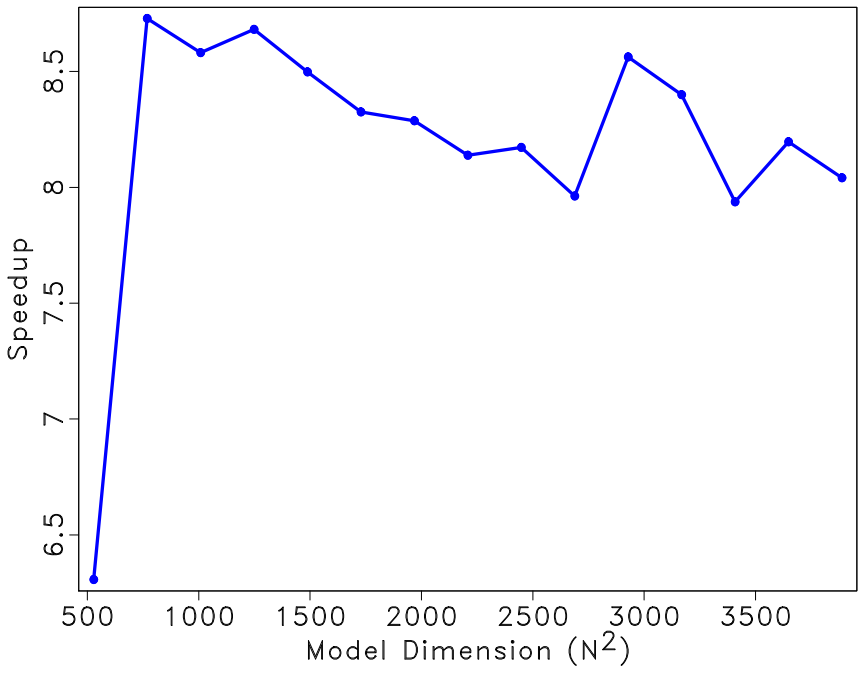
Figure 5: 2D Impulse response tests with the *ewefd2d_gpu* modelling code. (a) Isotropic model u_z component. (b) Isotropic model u_x component. (c) VTI model u_z component. (d) VTI model u_x component.

Figure 6 presents comparative GPU versus CPU metrics for a number of squared (N^2) model domains and runs of 1000 time steps. We ran the OpenMP-enabled CPU *ewefd2d* code on a dedicated workstation with a dual quad-core Intel Xeon chipset, and computed the corresponding GPU benchmarks on the 480-core NVIDIA GTX 480 GPU card. Because we output receiver data at every tenth time step, the reported runtimes involve both parallel and serial sections, which hides some of the speedup advantage of the GPU parallelism. Figure 6a presents computational runtimes for the CPU (red line) and GPU (blue line) implementations. The reported runtime numbers are the mean value of ten repeat trials conducted for each data point. Figure 6b shows the 10× speedup of the GPU implementation relative the CPU-only version.

Our second example tests the relative accuracy of the two implementations on a heterogenous isotropic elastic model. We use the publicly available P-wave velocity and density models of the 2004 BP synthetic dataset (Billette and Brandsberg-Dahl, 2005), and assume a S-wave model defined by $v_s = v_p/\sqrt{3}$. We use temporal and



a



b

Figure 6: GPU (blue line) versus CPU (red line) performance metrics showing the mean of ten trials for various square (N^2) model domain using the *ewefd2d* code. (a) Computational run time. (b) Speedup.

spatial sampling intervals of $\Delta t = 0.5$ ms and $\Delta x = \Delta y = 0.005$ km and inject a 40 Hz Ricker wavelet as a stress source for each wavefield component.

Figure 7 shows a snapshot of the propagating wavefield overlying the P-wave velocity model. Figures 8(a)-(b) presents the corresponding data from the CPU and GPU implementations, respectively, while Figure 8(c) shows their difference clipped to the same scale. The L_2 energy norm in the difference panel is roughly 2.0e^{-5} of that in the CPU/GPU versions, indicating that the GPU version is accurate to within a modest amount above floating-point precision. This slight discrepancy is expected due to the differences in treatment of math operations between the GPU and CPU hardware (Whitehead and Fit-Florea, 2011); however, we assert that this will not create problems for realistic modeling applications.

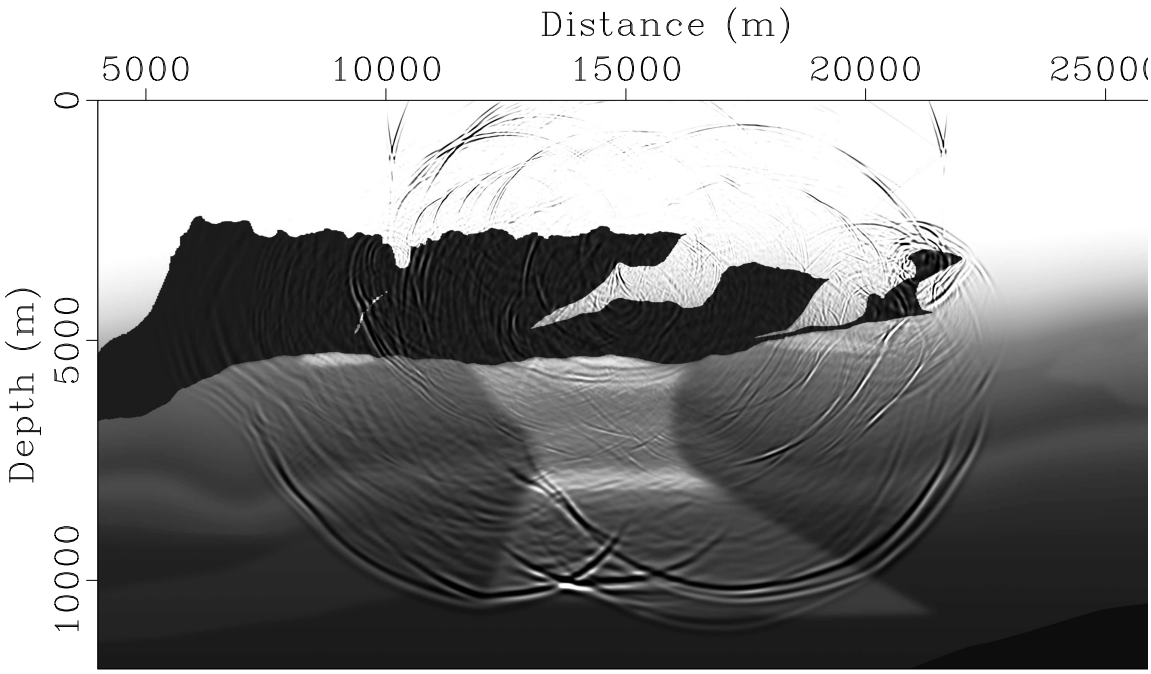


Figure 7: Wavefield snapshot overlying part of the P-wave model of the realistic BP velocity synthetic.

3D examples

We test the 3D multi-GPU implementation by computing impulse responses for 3D elastic media with different TI symmetries: isotropic, VTI, HTI and orthorhombic. Each example again uses the isotropic parameter set of v_p=2.0 km/s, $v_s = v_p/\sqrt{3}$, and $\rho = 2000$ kg/m^3, but incorporates different Thomsen anisotropy parameters. We define our VTI medium by $[\epsilon_1, \delta_1, \gamma_1] = [0.2, -0.1, 0.2]$, our HTI model by $[\epsilon_2, \delta_2, \gamma_2] = [0.2, -0.1, 0.2]$, and our orthorhombic medium by $[\epsilon_1, \epsilon_2, \delta_1, \delta_2, \delta_3, \gamma_1, \gamma_2] = [0.2, 0.25, -0.1, -0.05, -0.075, 0.2, 0.5]$. These parameters are transformed into stiff-

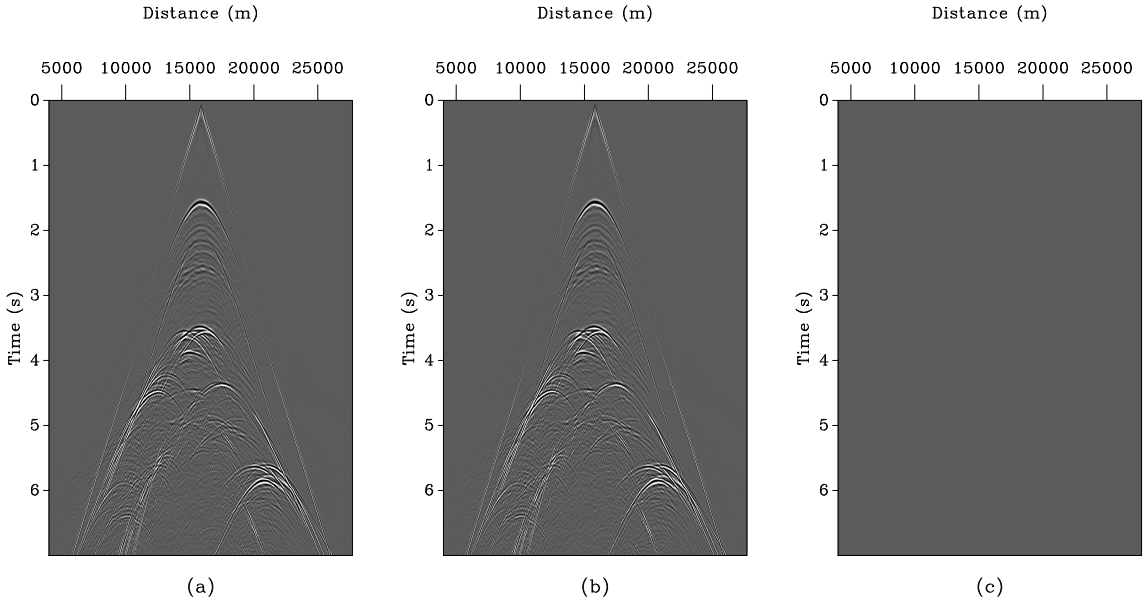


Figure 8: Data Modeled through for BP velocity synthetic model. (a) GPU implementation. (b) CPU implementation. (c) Data difference between GPU and CPU implementations clipped at the same level as (a) and (b).

ness tensor values using appropriate transformation rules (Thomsen, 1986). Figures 9a-(d) present a color-coded representation of the 6x6 C_{ij} Voigt representation of stiffness matrix c_{ijkl} for the isotropic, VTI, HTI and orthorhombic models used in the 3D impulse response tests, respectively.

We model seismic data on a $N_x \times N_y \times N_z = 204^3$ mesh at uniform $\Delta x = \Delta y = \Delta z = 0.005$ km spacing, assuming a 35 Hz Ricker wavelet stress source that we inject in each wavefield component. Figures 10a-(d) present the 3D impulse responses for the vertical component (u_z) for isotropic, VTI, HTI and orthorhombic media, respectively. Again, the GPU modeled wavefields for each TI medium are as expected when compared to results from the corresponding CPU code (not shown).

Figure **??** presents performance metrics for a number of different cubic (N^3) model dimensions. Figure **??** shows the runtimes for four different *ewefd3d* implementations: eight-core CPU (green line), single GPU (blue line), two GPUs with MPI communication within a single consolidated node (red line), and two GPUs with P2P communication within a single consolidated node (magenta line). Each reported runtime number is the mean value of ten repeat trials. The speedup metric shown in Figure **??** documents up to a 16× improvement over CPU benchmarks when using a single GPU device (blue line), and up to 28× improvement when using two GPU devices and P2P communication (magenta line). Generally, we observe increasing speedups when moving to larger model domains. Future multi-GPU tests will determine where this trend levels off. Figure 11c presents the P2P versus MPI speedup benchmark. We note that the MPI-based communication has a 10-15% overhead cost, which is expected due

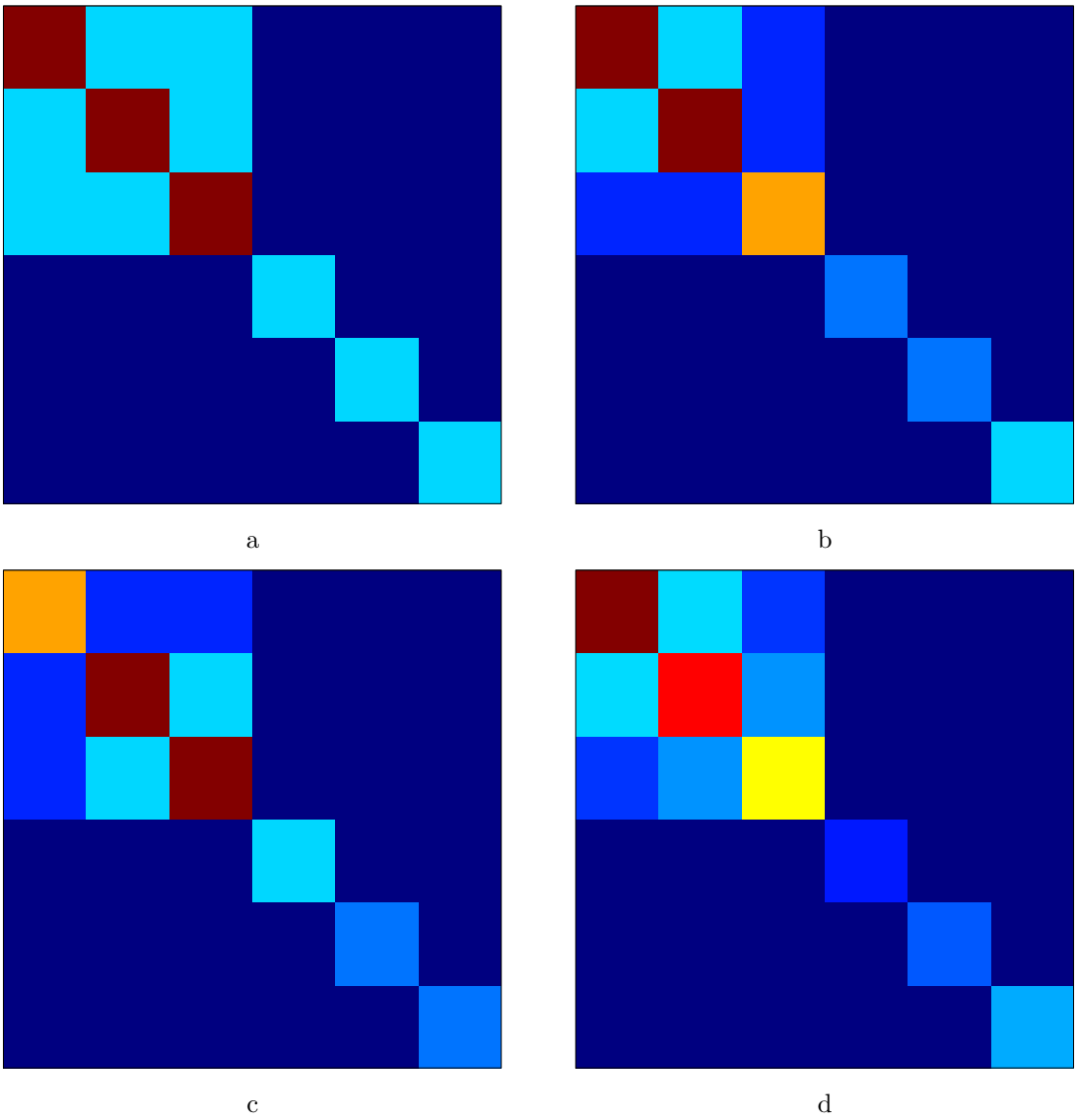


Figure 9: Elastic stiffness moduli in 6x6 Voigt notation for four elastic models with TI different symmetry. (a) Isotropic. (b) VTI. (c) HTI. (d) Orthorhombic.

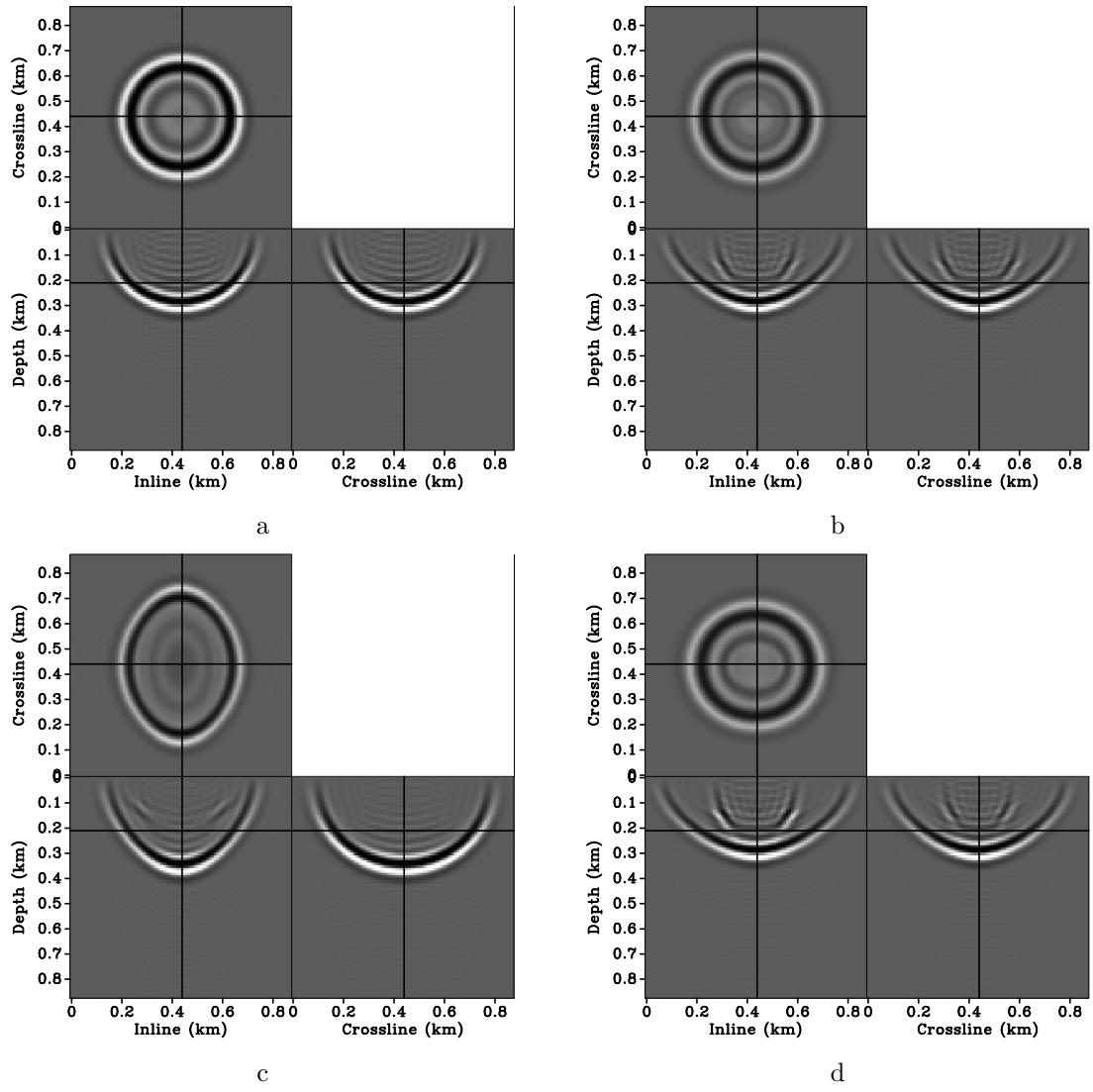


Figure 10: 3D Impulse responses (u_z shown) for four elastic models with different TI symmetry. (a) Isotropic. (b) VTI. (c) HTI. (d) Orthorhombic.

to the time required for repeatedly writing to a pinned memory location during the numerous MPI Send/Receive transfers required at each time step; this effect, though, diminishes with increasing model size. Note that the test results are benchmarks for a single consolidated node, and that in a true distributed compute environment where GPUs are located in networked nodes, network bandwidth and latency will have a significant impact on total compute time.

Our last test illustrates the utility of the 3D FDTD code for modeling wavefields through 3D heterogeneous anisotropic elastic media. Figures 12a-(b) present the C_{44} coefficient showing a layered earth model with a single dipping interface for the isotropic and HTI media, respectively. We superposed a snapshot of the propagating wavefield to demonstrate the complexity caused by the HTI media relative to an isotropic medium. The evident differences between the two wavefields indicates the importance of modelling realistic 3D anisotropic behavior, especially for velocity and anisotropic parameter estimation applications.

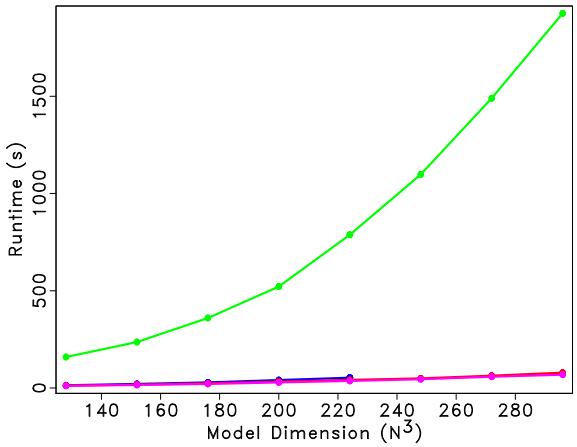
CONCLUSIONS

We present a GPU-based FDTD solution to the 3D elastic wave equation in a stress-stiffness formulation on a regular computational mesh that allows rapid modeling of data sets for 3D anisotropic TI media. We present a FD formulation of 2^{nd}-order temporal and 8^{th}-order spatial accuracy that leads to a compact stencil with a regular memory access pattern that is well-suited for running wavefield simulations on GPU devices. For the 3D algorithm we follow a loop unrolling approach over the slowest varying axis to minimize read redundancy from the GPU global memory. To circumvent the relatively limited memory on a GPU card, we use a domain decomposition strategy and employ CUDA's native P2P communications between multiple GPU devices housed within a single node. For situations involving a network of GPU-enabled compute nodes, we use the MPI instruction set to enable communication between GPUs.

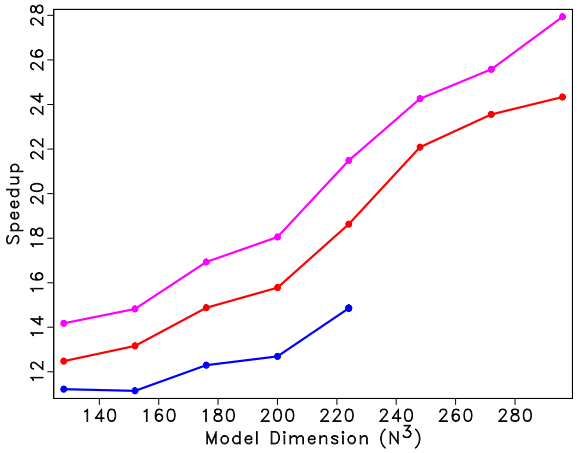
For 2D elastic modeling we achieved a 10× GPU speedup relative to an eight-core CPU version, while 3D anisotropic elastic modeling tests indicate up to a 16× improvement for a single GPU and a maximum 28× speedup when using two GPU devices relative to CPU benchmarks. These GPU-based speedup improvements allow us to efficiently model 3D elastic anisotropic phenomena and compute data sets for velocity and anisotropic parameter estimation and migration at lower hardware cost and with fewer total compute resources than heretofore possible.

ACKNOWLEDGMENTS

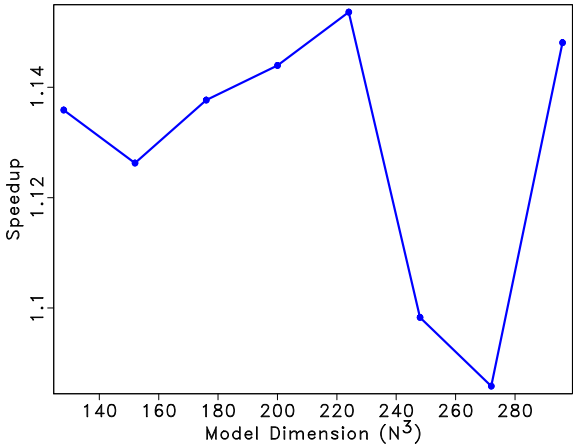
This research was partly funded by the sponsors of the UWA:RM consortium. We thank Paul Sava for the CPU versions of the *ewefd2d* and *ewefd3d* modeling codes, and David Lumley and Matt James for constructive conversations. We thank NVIDIA



a



b



c

Figure 11: Performance metrics showing the mean of ten trials for various cube (N^3) model domains using the *ewefd3d* code. (a) Computational run time for CPU (green line), a single GPU (blue line), two GPU with MPI communication (red line), and two GPU with P2P communciation (magenta line). (b) Speedup relative to CPU for single GPU (red line), and two GPUs with MPI (blue line) and P2P communication (magenta line). (c) Relative speed for the P2P versus MPI communication.

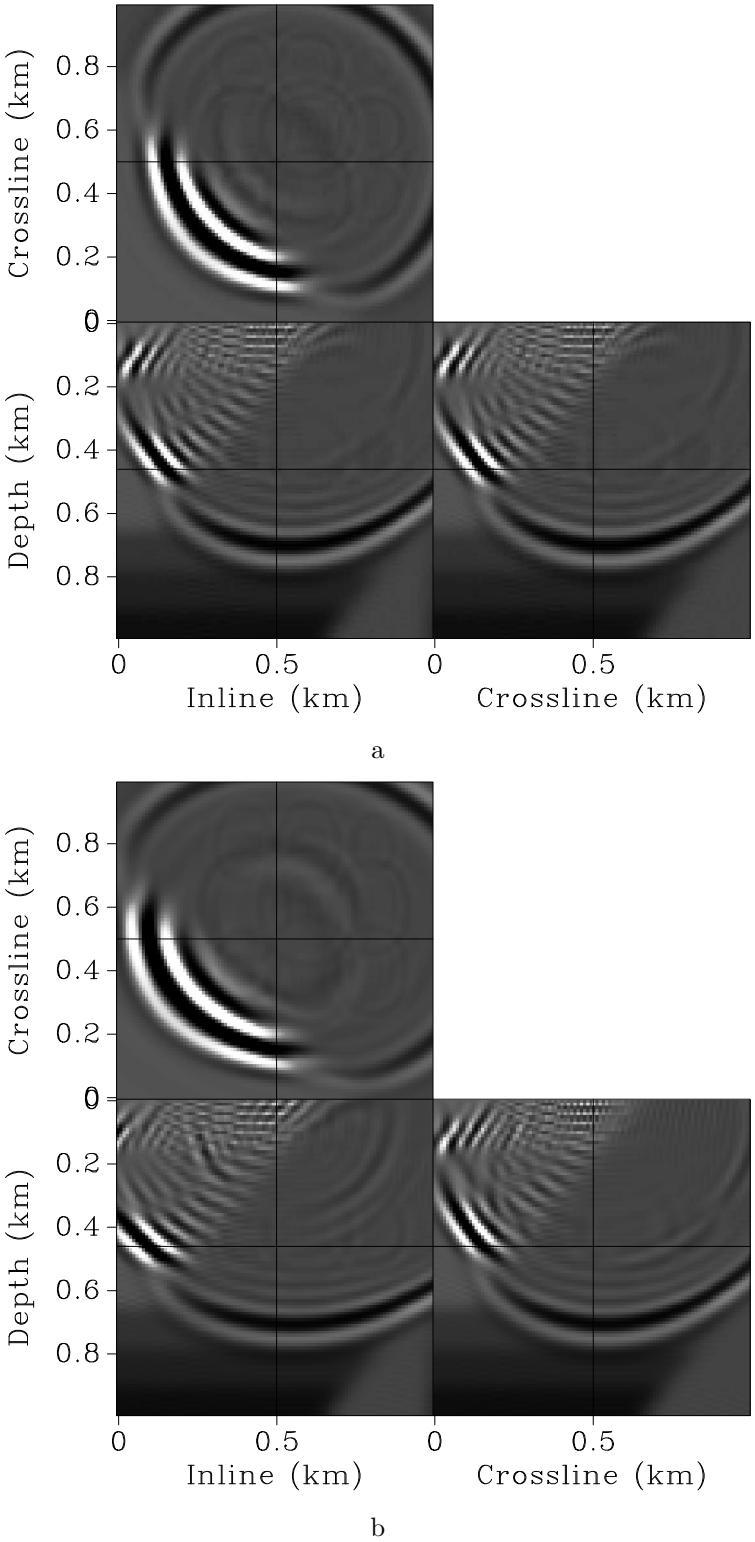


Figure 12: 3D impulse responses calculcated through a layered earth model with a single dipping layer. (a) Isotropic. (b) HTI.

for the GTX 480 and C2070 GPU cards used for research and development through the CUDA Research Center Program and a Professor Partnership Grant. We thank Joe Dellinger, Chris Leader and two anonymous reviewers for insightful comments and for helping to verify the modeling codes. The reproducible numeric examples use the Madagascar open-source package (http://www.reproducibility.org). Shragge acknowledges WAERA support through a Research Fellowship. We also acknowledge Professor David A. Yuen from the University of Minnesota for assistance in testing our software.

REFERENCES

Abdelkhalek, R., H. Calandra, O. Couland, G. Latu, and J. Roman, 2009, Fast seismic modeling and reverse time migration on a GPU cluster: HPCS09, Proceedings of the 2009 High Performance Computing & Simulation, 3643.

Billette, F., and S. Brandsberg-Dahl, 2005, The 2004 BP velocity benchmark: 67th Annual Meeting and Convention, EAGE, Expanded Abstracts, B035–B038.

Cerjan, C., C. Kosloff, R. Kosloff, and M. Reshef, 1985, A non-reflecting boundary condition for discrete acoustic and elastic wave equation: Geophysics, **50**, 705–708.

Clayton, R., and B. Enquist, 1977, Absorbing boundary conditions for acoustic and elastic wave equations: Bulletin of the Seismological Society of America, **67**, 1529–1540.

Foltinek, D., D. Eaton, J. Mahovsky, P. Moghaddam, and R. McGarry, 2009, Industrial-scale reverse time migration on GPU hardware: SEG Technical Program Expanded Abstracts, **28**, 2789–2793.

Fomel, S., 2012, Madagascar web portal: http://www.reproducibility.org (accessed 9 Feb 2012).

Komatitsch, D., G. Erlebacher, D. Göddeke, and D. Michéa, 2010, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster: Journal of Computational Physics, **229**, 7692–7714.

Kuzma, H. A., D. Bremer, and J. W. Rector, 2007, Support vector machines implemented on a graphics processing unit: SEG Technical Program Expanded Abstracts, **26**, 2089–2093.

Landau, L. D., and E. M. Lifshitz, 1986, Theory of elasticity, 3rd edition: Pergamon Press.

Micikevicius, P., 2009, 3D Finite Difference computation on GPUs using CUDA: Presented at the GPGPU2.

Morton, S., T. Cullison, and P. Micikevicius, 2008, Experiences with seismic imaging on gpus: 70th EAGE Conference & Exhibition, Expanded Abstracts, W08.

Nakata, N., T. Tsuji, and T. Matsuoka, 2011, Acceleration of computation speed for elastic wave simulation using a Graphic Processing Unit: Exploration Geophysics, **42**, 98–104.

Nguyen, H., 2007, GPU Gems 3: Addison-Wesley Professional.

Ohmer, J., F. Maire, and R. Brown, 2005, Implementation of Kernel Methods on the GPU: Digital Image Computing: Techniques and Applications, DICTA'05,

Expanded Abstracts, 78.

Pharr, M., and R. Fernando, 2005, GPU Gems 2: Programming techniques for high-performance graphics and general-purpose computation: Addison-Wesley Professional.

Thomsen, L., 1986, Weak elastic anisotropy: Geophysics, **51**, 1954–1966.

Trefethen, L. N., 1996, Finite difference and spectral methods for ordinary and partial differential equations: Cornell University.

Whitehead, N., and A. Fit-Florea, 2011, Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs: Technical report, NVIDIA Corporation, Santa Clara, California, USA.