Optimization of Forward Wave Modeling on Contemporary HPC Architectures

Jens Krueger*†, Paulius Micikevicius§, Samuel Williams*

*Lawrence Berkeley National Laboratory, Berkeley, CA, USA
§NVIDIA, Santa Clara, CA, USA
†Fraunhofer ITWM, Kaiserslautern, Germany

jtkrueger@lbl.gov, swwilliams@lbl.gov, pauliusm@nvidia.com

July 20, 2012

Abstract

Reverse Time Migration (RTM) is one of the main approaches in the seismic processing industry for imaging the subsurface structure of the Earth. While RTM provides qualitative advantages over its predecessors, it has a high computational cost warranting implementation on HPC architectures. We focus on three progressively more complex kernels extracted from RTM: for isotropic (ISO), vertical transverse isotropic (VTI) and tilted transverse isotropic (TTI) media. In this work, we examine performance optimization of forward wave modeling, which describes the computational kernels used in RTM, on emerging multi- and manycore processors and introduce a novel common subexpression elimination optimization for TTI kernels. We compare attained performance and energy efficiency in both the single-node and distributed memory environments in order to satisfy industry's demands for fidelity, performance, and energy efficiency. Moreover, we discuss the interplay between architecture (chip and system) and optimizations (both on-node computation) highlighting the importance of NUMA-aware approaches to MPI communication. Ultimately, our results show we can improve CPU energy efficiency by more than 10× on Magny Cours nodes while acceleration via multiple GPUs can surpass the energy-efficient Intel Sandy Bridge by as much as 3.6×.

1 Introduction

Modern oil or gas drilling operations, which have very high cost (over 100 million dollars) and risk, fuel the strong interest of the seismic industry in more accurate and higher performance computational kernels for subsurface imaging methods to support making drilling decisions. In recent years, Reverse Time Migration (RTM) has become a popular choice for subsurface layer modeling. RTM applies the discretized acoustic wave equation to each point in order to propagate waves through a given subsurface velocity model. The most physically accurate results would require the fully elastic wave equation to account for all elastic properties of the rock layers. The high computational requirements of such kernels make them impractical on current computing clusters for industry-sized seismic surveys. To address that problem, L.Thomson derived the so-called Thomson parameters in 1986 [24]. He reduced the 21 elastic properties to only 5, which are sufficient for the transverse isotropy that appears in many subsurface structures. This report discusses wave equation kernel implementations for isotropic (ISO), vertical transverse isotropic (VTI) and titled transverse isotropy (TTI) media.

With the ever increasing diversity of processor architectures and memory and interconnect topologies, it is essential to quantify and analyze the attainable performance on today's architectures in order to provide feedback to architects and applied mathematicians. Moreover, today's compute clusters are limited by energy consumption [12]. This motivates us to examine energy efficiency as well.

In this report we compare three variations of kernels used in RTM on x86 and GPU architectures that are available on the market. Our study provides direct performance and energy efficiency comparisons of single node and multi-node implementations and discusses the significance of kernel as well as inter-node communication optimization.

2 Reverse Time Migration

The Reverse Time Migration (RTM) [3] algorithm is composed of three main steps. In the first, the source wavefield is propagated forward in time starting from a synthetic simulated source signal at time zero. Next, the receiver wavefield is computed by propagating the receiver data backwards in time. Finally, the imaging condition cross-correlates the source and receiver wavefield to create the final subsurface image. Our study focuses on the wavefield propagation kernel in the first two steps that requires most of the overall RTM computation time. Specifically, the wave propagation kernel consumes more than 90% of execution time during the forward propagation step based on benchmark timings of current optimized RTM implementations [2]. In this report we optimize and analyze the three most common implementations of wave equation kernels on a variety of CPU and GPU-based systems.

2.1 Isotropic Wave Equation (ISO)

An isotropic medium represents the simplest case in which the wave propagates uniformly in all directions. The advantages of using a wave propagation kernel that assumes isotropy is the low computational and memory requirements but has strong disadvantages in accuracy when anisotropy properties are present in the medium. The partial differential equation (PDE) for isotropic media is presented in Equation 1. The finite-difference method is an approximation to the analytical solution. The error, or numerical dispersion, compared to the analytical solution decreases with the order of the derivative operator or the number of terms used in the Taylor series representation of the function, respectively [6]. The optimal order is a trade-off between computational complexity, numerical stability and quality requirements. Numerical analysis shows that above 8^{th}-order, the improvement in accuracy does not justify the increase of computational demand [16].

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right) p \tag{1}$$

in which c is velocity, t is time and p is the pressure wavefield.

The explicit, 8^{th}-order, finite-difference approximation produces the stencil shown in Figure 1. This stencil is applied to every point in the volume to calculate the amplitude for the next time step.

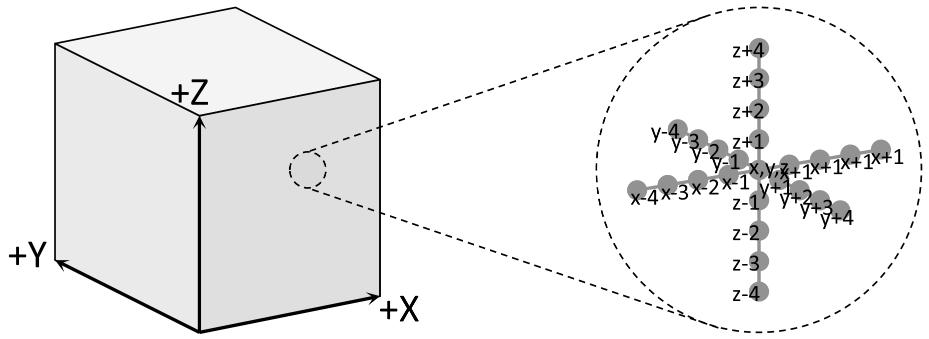


Figure 1: The figured shows an 8^{th} order in space star stencil how it appears in ISO and VTI kernel implementations.

2.2 Vertical Transverse Isotropic Wave Equation (VTI)

Vertical transverse isotropy (VTI) applies anisotropy by deriving varying velocities along a the vertical and horizontal axis using Thomson parameters ϵ and δ. An example PDE for VTI is presented by [29]:

$$\begin{aligned}
\frac{\partial^2 p}{\partial t^2} &= v_{px}^2\left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}\right) + v_{pz}v_{pn}\frac{\partial^2 q}{\partial z^2} \\
\frac{\partial^2 q}{\partial t^2} &= v_{pz}v_{pn}\left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}\right) + v_{pz}^2\frac{\partial^2 q}{\partial z^2}
\end{aligned}$$

where v_{pz} is the vertical P-wave velocity, $v_{px} = v_{pz}\sqrt{1+2\epsilon}$ is the horizontal pressure wave velocity and $v_p n = v_{pz}\sqrt{1+2\delta}$ is the P-wave moveout velocity. In contrast to ISO, VTI produces a coupled pair of stencils — a cross-shaped stencil in x and y-direction applied to the pressure wavefield p, coupled with a 1D stencil applied to the auxiliary wavefield q.

2.3 Tilted Transverse Isotropic Wave Equation (TTI)

For more accurate modeling of anisotropic properties for dipping subsurface layers the wave equation implementation for tilted transverse isotropy (TTI) media is used. TTI implementations account for the dip angle of a given medium and the azimuth describing the dipping angle along the third axis. Fletcher et al. presented a PDE for such media [21]:

$$\begin{aligned}
\frac{\partial^2 p}{\partial t^2} &= v_{px}^2 H_2 p + v_{pz}^2 H_1 q + v_{sz}^2 H_1 (p-q) \\
\frac{\partial^2 q}{\partial t^2} &= v_{pn}^2 H_2 p + v_{pz}^2 H_1 q - v_{sz}^2 H_2 (p-q)
\end{aligned}$$

where

$$\begin{aligned}
H_1 = & \; sin^2\theta cos^2\phi \frac{\partial^2}{\partial x^2} + sin^2\theta sin^2\phi \frac{\partial^2}{\partial y^2} \\
& + cos^2\theta \frac{\partial^2}{\partial z^2} + sin^2\theta sin 2\phi \frac{\partial^2}{\partial x \partial y} \\
& + sin 2\theta cos\phi \frac{\partial^2}{\partial x \partial z} + sin 2\theta sin\phi \frac{\partial^2}{\partial y \partial z} \\
H_2 = & \; \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - H_1
\end{aligned}$$

In the TTI case, the mixed partial derivatives create coupled stencils consisting of three intersecting planes. The result is a very complex and computationally-intensive stencil calculation.

2.4 Crosscutting Analysis of Kernels

All three kernel implementations have different memory requirements and work with single precision 32bit accuracy. The ISO kernel requires the pressure wavefield at two time steps t and t_{-1} (second order in time) as well as the velocity for every point in space. Thus, the reference implementation operates on four arrays. A common optimization to save memory capacity is the use of only two time step arrays. As points from t_{-1} are accesses only once, results for t_{+1} can be directly written to the same addresses. Such an optimization also obviates the need to cache bypass optimizations.

The VTI implementation requires the same volumes as ISO but adds the auxiliary wavefield Q and the spatially-varying Thomson parameters ϵ and δ, resulting in an increase in the total number of grids to nine. Commensurate with the increase in memory capacity is a corresponding increase in data movement computation.

For the TTI kernel we have to include two more grids for ϕ and θ. Optionally, we may replace these two arrays with four arrays that use pre-computed sine and cosine of ϕ and θ. This has the potential of increasing working set size and bandwidth pressure, but reduces the computational requirements of the code on machines lacking fast trigonometric function evaluation. Table 1 summarizes the memory requirements for the different kernels.

3 Related Work

There have been numerous previous studies that examined the performance of seismic algorithms [5, 15, 19] primarily focusing on optimization of the mathematical approximation scheme.

Various approaches have also been examined to address the bandwidth limitations of seismic migration simulations [10, 23, 25]. Reorganizing these stencil calculations to take full advantage of memory hierarchies has been the subject of much investigation over the years. These have principally focused on tiling optimizations [7, 8, 14, 20, 22], which attempt to reorganize stencil calculations to exploit locality and reduce capacity misses; such optimizations are effective when the problem size is larger than the cache's ability to exploit temporal locality. Alternately, researchers have examined design

array	ISO	VTI	TTI	TTI*
1	P(t-1)	P(t-1)	P(t-1)	P(t-1)
2	P(t)	P(t)	P(t)	P(t)
3	P(t+1)	P(t+1)	P(t+1)	P(t+1)
4	Velocity	Velocity	Velocity	Velocity
5		Q(t-1)	Q(t-1)	Q(t-1)
6		Q(t)	Q(t)	Q(t)
7		Q(t+1)	Q(t+1)	Q(t+1)
8		ϵ	ϵ	ϵ
9		δ	δ	δ
10			θ	$sin(\phi)$
11			ϕ	$cos(\phi)$
12				$sin(\theta)$
13				$cos(\theta)$
number of arrays accessed per point	4	9	11	13
average number of Flops per point	34	53	≈ 800	290
average number of Bytes moved per point	16.2	36.5	44	52.6
Overall Flop:Byte	2.1	1.6	18	5.6

Table 1: **Memory and Bandwidth requirements for the different kernels. All arrays are roughly the same size and must be accessed at least once per stencil sweep. *Assumes the sine and cosine of ϕ and θ are pre-computed and stored in 4 additional arrays.**

space exploration to tailor processor architecture to the needs of RTM in order to improve energy efficiency [13]. Since 2008, researchers in both industry and academia have explored the use of GPUs for seismic processing [1,4,11,17,18].

As canonical wave equations, this report uses mathematical representations of a VTI scheme introduced by [29] and the TTI scheme from [21].

4 Experimental Methodology

4.1 Evaluated Systems

In order to demonstrate the broad applicability of our performance optimization techniques as well as quantify the resultant performance and energy efficiencies, we use a broad range of CPU and GPU architectures. The pertinent details are summarized in Table 2.

AMD Opteron 6172 (Magny Cours): The AMD Opteron 6172 is a dual-socket $\times$ 12-core (24 total) compute node. In reality, each socket (multichip module) has two dual 6-core chips with individual memory controllers. As such, the compute node is conceptually a four-chip NUMA SMP. Each Opteron chip has six cores running a 2.1GHz. In single-precision (the focus of this report), each core can complete one (four-slot) SIMD add and one SIMD multiply per cycle. This provides a peak performance of 403 GFlop/s per node. Additionally, each core has private 64 KB L1 and 512 KB L2 caches. The six cores on a chip share a 6MB L3 cache and dual DDR3-1333 memory controllers capable of providing an average STREAM [28] read bandwidth of 12GB/s per chip.

The Magny Cours is used in “Hopper”, the Cray XE6 at NERSC. It is built from over 6000 Opteron 6172 compute nodes. Each pair of compute nodes shares one Gemini network chip and collectively form a 3D torus.

Intel Xeon X5550 (Nehalem): The X5550 is a dual-socket × quad-core node. Like the Opterons, the cores are capable of executing one SIMD add and one SIMD multiply per cycle, but are running at slightly faster 2.66 GHz. Each core owns a private 32 KB L1 and 256 KB L2 cache. The four cores on a chip share a 8 MB L3 cache and three DDR3-1333 memory controllers capable of providing up to 17.6 GB/s per chip resulting in a peak performance of 170 GFlop/s. We utilize the "Carver" Infiniband cluster at NERSC which is comprised of over 400 X5550 compute nodes. Within a rack, nodes are connected via a QDR InfiniBand fat tree network, while racks are arranged into a torus.

Intel Xeon E5-2687W (Sandy Bridge): The Xeon E5-2687W Sandy Bridge is Intel's newest commodity CPU architecture and is a dual-socket × 8-core (16 total) processor node running at 3.1 GHz. The microarchitecture has been enhanced from Nehalem and now supports the 256-bit AVX SIMD instruction set. Thus, each core is capable of executing one (eight-slot) SIMD add and one SIMD multiply per cycle. This provides a peak node performance of 793.6 GFlop/s (SP). With HyperThreading each core is capable of running 2 threads at a time, which results in a total of 32 threads per node. Unlike Nehalem, Sandy Bridge has two load ports to the L1, thereby doubling L1 read bandwidth — a clear benefit when locality cannot be maintained in the register file. Sandy Bridge uses a ring-based network-on-chip and has four memory controllers connected to DDR3-1333 DIMMs. This provides an aggregate DRAM pin bandwidth of 85.3 GB/s. The Sandy Bridge system evaluated here was a single-node configuration.

Multi-GPU Tesla M2090 Accelerated Node GPU performance was measured on a server with four M2090 GPUs. PCIe switches are used to arrange the GPUs into a tree topology. Each PCIe link has theoretical duplex bandwidth of 16 GB/s (8 GB/s per direction), in practice 12 GB/s can be achieved.

M2090 is a Fermi-architecture GPU with 6 GB of GDDR5 memory and 16 Streaming Multiprocessors (SMs). Each multiprocessor is clocked at 1.3 GHz and contains 32 fp32 pipelines (for an aggregate of 512 for the entire GPU), 64 KB of SRAM that can be partitioned by the programmer to be used as shared memory or L1 cache, and a 128KB register file. While an SM can track up to 1536 threads, it implements the Single-Instruction-Multiple-Threads (SIMT) execution model: any instruction is issued for a group of 32 threads (referred to as a warp), hardware predication ensures correctness when threads in a warp take divergent control paths. Instructions are issued in-order, execution is pipelined. Memory-access and arithmetic latencies are hidden by switching execution among warps. Context switching between warps is free since the register file and other state is partitioned among the active threads. Theoretical shared memory bandwidth, aggregated across all the SMs, is 1331 GB/s. Theoretical DRAM bandwidth is 177 GB/s, of which 150 GB/s can be sustained with a stream copy. Turning ECC on reduces the amount of bandwidth available to data. While the exact amount of bandwidth consumed by ECC depends on the application, typically it consumes around 20% (note that impact on application performance can be lower if the memory bus isn't continuously saturated). We provide performance numbers with and without ECC.

4.2 Programming Models

On the CPUs, we use posix threads for multicore parallelism. The C-code is optimized to exploit SSE and AVX via software intrinsics. Inter-node parallelism is realized by MPI via disjoint computation and communication phases. Nominally, we run one MPI process per node (with multicore parallelism extracted via Pthreads). However, to quantify the impact of NUMA and network contention, we also conduct experiments with one process per NUMA node. OpenMP is a viable solution, but we selected pthreads based on its ability to better control data placement for NUMA.

GPUs were programmed using CUDA C (version 4.0 of the toolkit and driver). A single CPU

Core Architecture	Intel Nehalem	NVIDIA Fermi	Intel Sandy Bridge	AMD Opteron
Clock (GHz)	2.66	1.30	3.40	2.10
Single-Precision Gflops/s	21.3	83.2	54.4	16.8
L1 Data Cache	32 KB	16/48 KB	64 KB	64 KB
L2 Data Cache/Local Store	256 KB	N/A	256 KB	512 KB
Threads/core	2	1536 (max)	2	1
Socket Architecture	**Xeon X5550**	**Tesla M2090**	**Xeon E5 2687W**	**Opteron 6172**
Cores/chip	4	16 SMs	8	6
Shared Last Level Cache/chip	8 MB	768 KB	20 MB	6 MB
Chips/socket	1	1	1	2
Single-Precision Gflops/s	85.3	1331.2	386.8	100.8
DRAM pin GB/s	32.0	177.4	42.4	21.33
TDP	95W	225W	150W	115W
SMP Architecture	**Xeon X5550**	**Tesla M2090**	**Xeon E5 2687**	**Opteron 6172**
Chips/SMP	2	1	2	4
memory parallelism	HW prefetch	Multi-threading	HW prefetch	HW prefetch
Single-Precision GFlops/s	170.66	1331	793.6	403.2
DRAM Pin GB/s	64.0	177	85.3	85.33
STREAM GB/s	38	150 (no ECC)	75.2	47
Power under load	298W	200W (GPU-only)	350W	455W

Table 2: Details of the evaluated architectures. Note, XE6 power derived from Top500 submission [26].

thread is used to control all GPUs, domain is decomposed among GPUs along the slowest-varying dimension, GPUs on a node exchange halos using peer-to-peer PCIe protocol.

4.3 Compact Benchmarks

We separate the experiments presented in this report into singe- and multi-node experiments. Single-node experiments use the same 512^3 single-precision problem size for ISO, VTI, and TTI as well as across all CPUs and GPUs. This allows the reader to directly compare the impact on time-to-solution of increased fidelity, performance optimization, or variations in architecture. All multi-node experiments use a 1536^3 volume decomposed among 27 compute nodes in a $3\times3\times3$ topology communicating via MPI. This results in a 512^3 subdomain per node. Using three nodes in each dimension makes sure that all neighboring nodes are distinct from each other for the applied periodic boundary conditions. Per-shot production runs are typically weakly-scaled larger by a small factor. As the code is an explicit time-stepping method with nearest-neighbor communication, performance will scale well.

5 Performance Expectations

Table 1 presented a detailed analysis of the kernel requirements. The highly divergent kernel characteristics result in different performance expectations. Based on a maximum bandwidth achieved with the Stream benchmark [28] and the Roofline model [27], we may calculate a performance bound for each kernel and system. Thus, a Sandy Bridge node should be able to deliver up to 4100 MPoints/s for ISO, 1800 MPoints/s for VTI and 1350 MPoints/s for TTI — twice the performance of Carver nodes. Due to the high-performance GDDR5 used by NVIDIA's M2090, the GPU should be able to deliver more than twice the performance of a Sandy Bridge node based on memory bandwidth limitations.

We expect the performance of ISO and VTI to be memory-bound due to their low arithmetic intensity (flop:byte ratio). Conversely, the high arithmetic intensity of TTI leads to the conclusion that we are likely not to be bound by memory bandwidth for TTI but may be constrained by the performance of the on-chip memory hierarchy.

6 CPU Optimizations

6.1 Cache Blocking and Parallelization

The reference implementation is parallelized over the z-dimension of the 3D grid. This has a severe impact on locality. The requisite cache working set scales as $O(threads \times X - Dim \times Y - Dim)$. As ISO, VTI, and TTI each operate on multiple arrays, the requisite cache working set can quickly exceed the cache capacity on any of these machines. The impact is that a large number of capacity misses squander memory bandwidth. The solution is to simultaneously parallelize and cache block in the Y-dimension. The code is restructured to operate on cache blocks of different sizes. These blocks are parallelized across the cores in a round robin fashion on the compute node. Note, parallelization among cores in the X-dimension was not attempted on the CPUs as CPUs rely on hardware stream prefetchers to hide memory latency. Parallelization in X will not expose long unit strides essential to effective hardware stream prefetching. We will show that the GPU's reliance on multithreading (instead of prefetching) obviates this restriction. For Intel based nodes, it is beneficial to block ISO and VTI kernels for L2 cache sizes and TTI for L3. Hopper showed a different behavior and all kernels performed best when blocked for the L3 cache capacity.

6.2 NUMA

The Xeon and Opteron are NUMA architectures. Failure to properly initialize the grids in a NUMA-aware fashion will result in data being allocated on only one of the NUMA nodes and would underutilize the memory controllers on the node — 50% of them on the Xeon and 75% of them on the Opteron. We modify the parallelization strategy to decompose the problem in the Z-dimension among NUMA nodes and parallelize the blocks in the Y-dimension among only the cores within a NUMA node. We then modified the initialization routines to allocate the grids in a NUMA-aware fashion. Although this optimization can be avoided by allocating one MPI process per NUMA node, we will show it has performance penalties.

6.3 SSE/AVX

Compilers are notoriously challenged to effectively exploit SIMD instructions. For these single-precision calculations we must ensure 4-way SSE SIMDization and 8-way AVX SIMDization. To maximize performance, we unroll the code, group the operations within a stencil and map them to SIMD intrinsics. To facilitate high-performance SSE code, when allocating the grids, we align to 32-byte boundaries. When coupled with problem sizes that are multiples of 8, we ensure most accesses are aligned and do not cross cache line boundaries. However, we observe that high-order derivatives in the unit-stride (X-dimension) are particularly challenging as they require 8 unaligned accesses per stencil. To minimize this effect, we modify the code to maintain locality within the register file and use shuffles to create the unaligned data we need. This significantly reduces the number of accesses to the L1.

6.4 Common Subexpression Elimination

As discussed in Section 5 the ISO and VTI kernels are likely to be memory-bound and the TTI implementation highly compute-bound. Hence, once we've maximized performance (flop/s), the only remaining option to improve performance is to reduce reduce the number of floating-point operations. We may exploit Common Subexpression Elimination (CSE) to eliminate redundant differences (subtracts) between adjacent stencils inspired by the success of Datta et al. [9]. The nature of second-order mixed derivates requires applying the second-order derivates in two steps. At first, first derivates are computed for each point along the axis in all three dimensions, followed by summing of such values and multiplying a coefficient. Since, the first derivatives do not alter for consecutive points, given x as fastest unit-stride direction, causes redundant computations in the $x-y$, $x-z$ and $y-z$ planes. To avoid redundant computations an additional array is allocated that holds the first derivates for all points and all space dimensions in the two fastest unit strides. The size of the arrays depends on the finite-difference approximation order-in-space. In x-direction these arrays are small enough to be kept in L1 cache, in y-direction the size of the array depends on the unit-stride in y. Hence, when the stencil is applied to the subsequent point along the x-axis, only three new first derivates need to be calculated. Each derivative requires $\delta = 3r$ floating-point operations with r as radius of the stencil. The three applied operations are the subtraction of two points and the multiplication of a coefficient. The result is summed to receive the final derivative. A total of $(2r+2)*\delta$ flops are required for the partial derivatives, and $\delta * r$ flops for the second-order derivatives along the axis for wavefield P and Q. When CSE is applied, the first derivative needs to be attained per wavefield only twice per axis. In total, a reduction of $6r^2$ flops is achieved for the mixed, partial derivatives. The second-order derivates along the axis cannot be avoided, which adds another $3r*\delta$ flops per wavefield.

With CSE, the number of floating-point instructions per point is dependent on the plane size in x and y. A larger plane improves the flops-per-point ratio because of an improved volume-to-surface ratio. The result is a reduction to about 290 flops per point for 512^3, 8^{th}-order stencil scheme. Assuming one captures all temporal locality and pre-computes sine/cosine of ϕ and θ, approximately 53 bytes must be accessed from DRAM per point. The resultant decrease in computational requirements lowers the flop-to-byte ration from approximately 19 to 5.6. This is still sufficiently high to prevent DRAM bandwidth from impeding performance.

6.5 Precomputation of Trigonometric Functions

As described in 6.4, one may improve TTI performance by reducing the number of instructions per point. As ϕ and θ do not vary in time, one may precompute their sine and cosine and store them in 4 additional arrays. This increases memory bandwidth pressure but significantly decreases the computational requirements (even over using SVML).

7 GPU Optimizations

7.1 Parallelization

GPU implementation follows the approach described in [17]. Specifically, the face of the volume defined by the two fastest-varying dimensions (let these be x and y) is tiled with thread blocks, each thread block then marches along the slowest-varying (z) dimension producing a pencil of output.

The "pencil" approach enables the use of registers and shared memory for storing the input values necessary to compute discrete spatial derivatives: values along the z-dimension are kept in registers (no sharing is needed among threads), values in the xy-plane are stored in shared memory (several threads need to access each value). Other input values (Thompson parameters, etc.) are read when needed, effectively streaming through those arrays. The use of shared memories comparable to cache-blocking on CPUs, since it ensures that stencil values are read once from the off-chip global memory (177 GB/s peak) and subsequently are read repeatedly from the on-chip shared memory (1331 GB/s aggregate peak). The use of shared memory is facilitated by GPU memory and programming models, which provide the programmer explicit control of the on-chip memory.

7.2 Limited On-chip Memory

Both ISO and VTI computations can be done in a single pass over the domain, using 2D thread blocks (32×16 and 32×8 threads, respectively). TTI is implemented as two passes: the first pass computes the first and second y-derivatives for the P and Q wavefields, the second pass computes the remaining derivatives and output.

Since GPUs contain hardware dedicated to transcendental functions the sines and cosines of dip and azimuth angles are computed on the fly. This provides a two-fold benefit: reduces bandwidth pressure during execution and lowers the size of the working data set.

7.3 Array padding

Array padding is used to ensure optimal memory access pattern. As with any memory system, Fermi memory throughput is maximized when memory-bus transactions (cache lines) carry only application-requested data. Since GPUs don't generally rely on caches for temporal coherence, this means that performance is highest when a warp (32 consecutive threads) requests a contiguous, preferably aligned, address region. Two memory quanta are available: 128 bytes and 32 bytes.

We use two types of padding - row and lead. We pad each row of an array so that its size is a multiple of 128 bytes thereby ensuring that all rows are equally aligned. Furthermore, we add padding at the beginning of the array so that the address of the first output element is a multiple of 128 bytes (the amount of this "lead-pad" is 28 floats: 32 floats minus half the order in space). This ensures that warps access all non-halo elements (both wavefields and earth-model property arrays) in a perfectly-aligned pattern.

7.4 Array interleaving

Both VTI and TTI GPU implementations interleave a pair of earth-property arrays into a single array of float2 structures. For example, δ and ϵ are stored as a single array of 2-element vectors, where components are δ and ϵ values, respectively. TTI implementation also interleaves the dip and azimuth angles (θ and ϕ) as well as all the P and Q wavefields. P and Q wavefields are not interleaved for VTI code since they are read at different offsets due to the fact that different derivatives are computed for each field (Section 2.2).

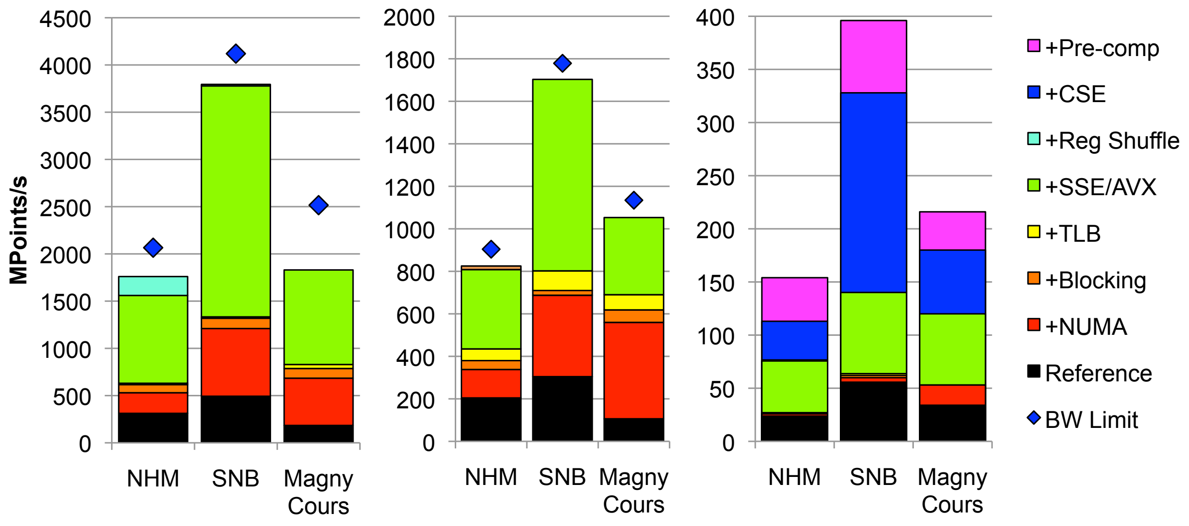
Since Fermi architecture provides 64-bit memory-access instructions, compared to accessing two separate float elements, accessing a single float2 reduces the number of load/store as well as pointer-arithmetic instructions. Array interleaving is used purely for storage - once data is read from memory, the respective components are treated as scalar floats and no vector arithmetic is performed.

When applied individually, padding or interleaving improve performance by 13-15%. Performance improves by 39% when they are applied concurrently.

7.5 Inter-GPU Communication

For multi-GPU experiments we use 1D domain-decomposition along the slowest-varying (z) dimension. Direct, peer-to-peer (P2P) PCIe DMA communication (available in CUDA 4.0 and later) is used to exchange halos between GPUs. As GPUs contain two DMA engines, which, together with duplex PCIe links, enable concurrent sending and receiving of halos. We implement the exchange in two phases. In the first phase each GPU sends its data to its neighbor in the positive-z direction. In the second, the halo is sent to the GPU below. We observe an aggregate throughput using a PCIe tree topology of 15 GB/s.

Inter-GPU communication is hidden by overlapping it with GPU execution. This is done by first computing the halo regions that have to be communicated to neighbors, then issuing the asynchronous memory copies and at the same time computing in the remaining internal region of the subdomain. Synchronization at the end of each step avoids data hazards.



(a) ISO (b) VTI (c) TTI

Figure 2: RTM wave propagation kernel performance performance across CPU architectures as a function of progressive levels of optimization of the wave equation. Note, "NHM" is the 2P Nehalem, "SNB" is the 2P Sandy Bridge, and "Magny Cours" is the 4P Opteron (XE6). The blue diamond is the STREAM bandwidth limit.

8 Single-Node Results and Analysis

In this section we present performance across platforms and scale with a keen eye on performance, energy efficiency, potential scalability, and the fundamental architectural bottlenecks that constrain them.

8.1 CPU Performance

We begin by examining single node performance across a variety of machines. In the case of GPU-accelerated nodes, we examine variants with 1, 2, and 4 GPUs per node collaborating on a single grid. In all cases, we evaluate performance on a 512^3 grid per node and express performance in millions of points updated per second (MPoints/s).

Figure 2 presents performance as progressively more optimizations are included in the mix. Clearly, the reference parallel implementation delivers poor performance across machines with Sandy Bridge delivering the best performance for all kernels.

As all nodes are NUMA architectures, domain decomposition in z-direction for each NUMA node and in y-direction for the number of cores within a NUMA node avoids the performance pitfalls (particularly sever ones on the 4P Magny Cours) of poor data placement. It should be noted that NUMA effects are amortized when an implementation is heavily compute-bound. Tuning for the appropriate block size further improves performance for ISO and VTI but has no effect on TTI kernels.

To improve floating-point performance, we manually SIMDize the code using SSE or AVX intrinsics including Intel's SVML sine and cosine functions. The benefit we realize is highly dependent on the underlying machine's potential memory or cache bottlenecks as well as the starting point. One can see that Sandy Bridges's dual ported L1 cache mitigates that cache bottleneck.

The novel common subexpression elimination scheme shows great performance improvements and more than doubles TTI performance on Sandy Bridge nodes. It was not applied to the ISO and VTI due to the bandwidth-bound nature and little potential. The use of cache bypass has limited potential when compute-bound and is detrimental when only two arrays are shared for the three time steps.

Finally, to reduce redundant computation in TTI to a minimum, we pre-compute the sine and cosine of the spatially varying azimuth and dip constants. This requires we increase the number of arrays by two, but completely avoid expensive trigonometric calculations in the inner loop. On Magny Cours, this provides a moderate increase in performance of 12.5% and up to 20% on Sandy Bridge.

Comparing the overall performance in Figure 2, one can see that the Sandy Bridge nodes deliver the highest total performance of our x86-based systems due its high bandwidth and compute capability. Although we obtain a more than 6× increase in TTI performance, we observe that TTI is still compute-bound on all architectures.

To qualify our results, we calculate a bandwidth performance bound for the ISO and VTI implementations. This limit is simply the ratio of STREAM bandwidth to compulsory data movement to and from DRAM. Figure 2 shows this limit as the blue diamond. The limit is not shown for TTI because it is compute-bound.

One should not that the fixed cache block sizes may not be ideal for each architecture. In fact experimentation with larger (non power-of-two) volume sizes shows one might improve performance by as much as 19% on the Magny Cours system.

8.2 GPU Performance

As seen in Figure 3, a single M2090 node setup achieves 4,972 MPoints/s for ISO kernels, 3,170 MPoints/s for VTI and 1,130 MPoints/s for TTI kernel implementations, respectively. One can observe that GPU performance scales nearly linear with the number of GPUs since communication time is lower than internal computation. It should be noted that turning off ECC can further improve performance by as much as 30%. Users must therefore carefully weigh the performance benefits with the risks of memory errors. As evaluated x86-based architectures rely on ECC buffered DRAM ECC is kept turned on for further benchmarks comparisons.

Interestingly, on the more bandwidth-bound ISO and VTI kernels, we observe that a single M2090 with enabled ECC is only about 1.3× to 1.45× faster than the Sandy Bridge system. Conversely, on the more compute-intensive TTI, the M2090 is about 2.85× faster than Sandy Bridge.

8.3 Energy Efficiency

While noting the performance differences between the architectures, we must also acknowledge the commensurate difference in power to address modern energy efficiency challenges for compute clusters. To normalize the differences in machines we examine the energy efficiency of calculating these RTM kernels as a function of machine and configuration.

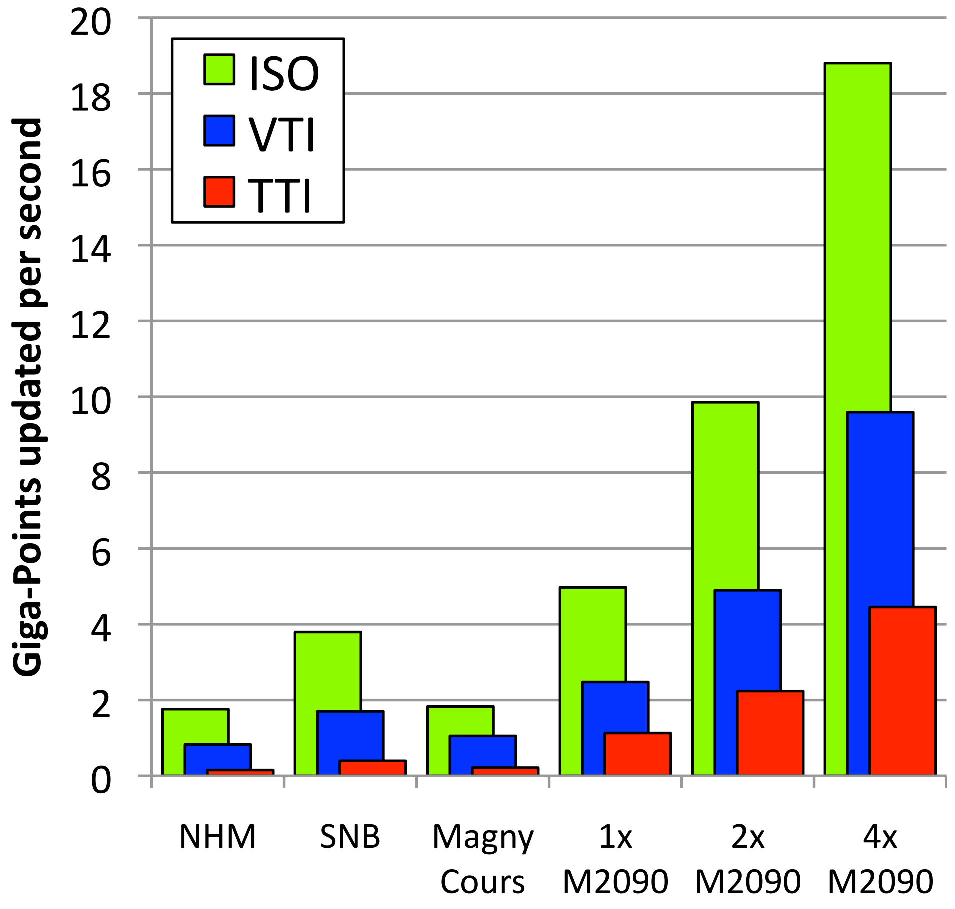


Figure 3: Optimized performance of different architectures running optimized kernel for ISO, VTI and TTI

Sustained system power consumption for the Nehalem and Sandy Bridge machines was measured with an in-line power meter. Using an in-line power meter was not possible for the Magny Cours system due to how power is distributed within a XE6 rack. Therefore, for the Magny Cours system we estimate the power by averaging per node the power NERSC measured for their Top500 submission.

GPU-power consumption was measured using nvidia-smi tool (included with the driver), which reports the power GPU draws from the host system. The highest measured power consumption was 180W. However, since this does not include the system power-supply overhead, we adjust this to 200W assuming a power-supply with 90% efficiency, common in modern servers. Since GPUs need a host system to control them, we add estimated host power consumption. To be conservative, we assume that the host CPU system is consuming the same power when GPUs are running as when all CPU cores are executing, which is 298W for a dual-Nehalem server. Thus, we estimate that power consumption for servers with 1, 2, and 4 GPUs to be 500W, 700W, and 1100W, respectively, and use these estimates in Figure 4.

Figure 4 shows energy efficiency (MPoints/s per Watt) across the various platforms. We observe that on ISO, the 1-GPU system with ECC enabled is slightly less energy-efficient than the 2P Sandy Bridge. Conversely, a system with 1 GPU is more than twice as energy-efficient as the Magny Cours system. Note, as the host is not performing computations in GPU systems, the host's idle

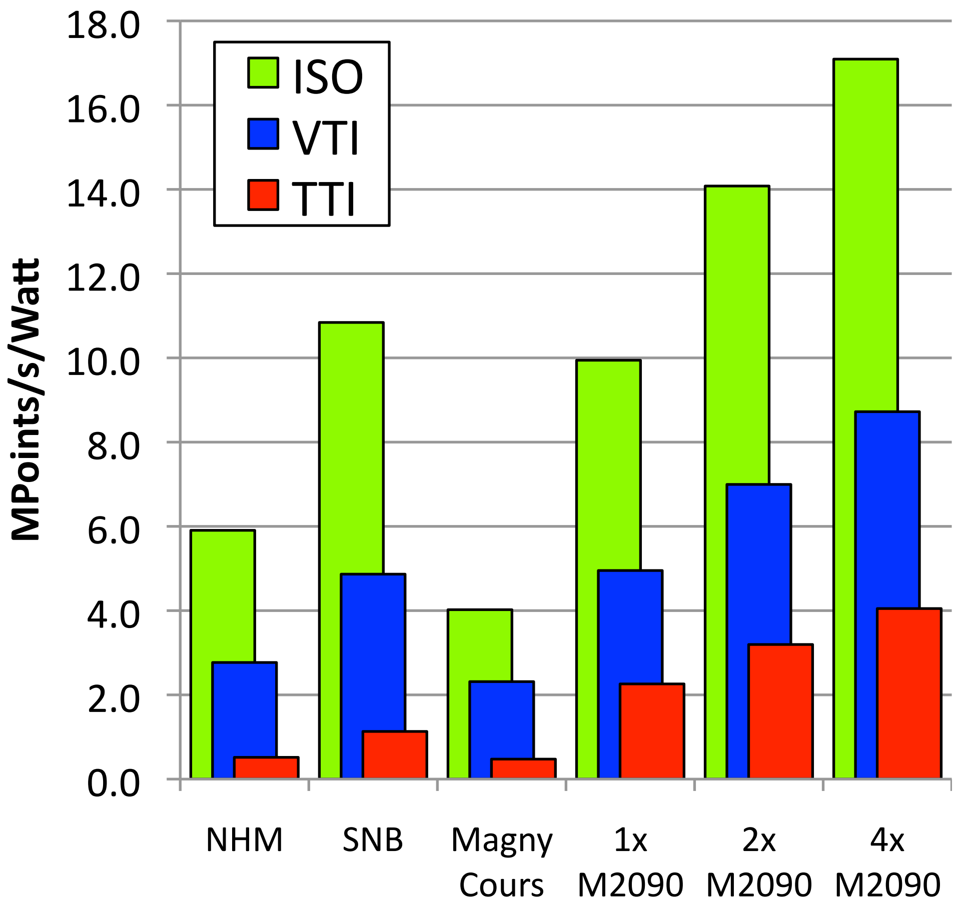


Figure 4: Energy efficiency of different architectures running optimized kernel for ISO, VTI and TTI

power consumption is amortized as more GPUs are added to the node (system-power consumption ratio approaches the ratio between single-GPU and single-socket power ratio). As a result, the 4-GPU solution is about 1.5× more energy-efficient than a Sandy Bridge system. For the even higher bandwidth-bound VTI kernel, the GPU's energy efficiency improves and results in almost 2× for a 4-GPU setup compared to the Sandy Bridge node.

As one moves to computationally-intensive kernels like TTI, we see the GPU's compute density and energy efficiency exploited. In fact, for TTI kernels, a single-GPU and 4-GPU systems are now more than 2× and 4× more energy-efficient than the Sandy Bridge system, and up to 8× as efficient as the Nehalem or Magny Cours systems.

9 Distributed-Memory Results

Due to the high communication requirements (thick halos induced from the high-order method), single node performance may not be entirely predictive of distributed memory performance at the moderate scale required per shot. Therefore, it is important to quantify how much the substantial performance improvements discussed in Section 6 and 7 are amortized by inter-node communication. In this section, we present distributed memory performance for the Cray XE6 (Hopper) at NERSC. Sandy Bridge and M2090 distributed memory experiments were not possible due the the lack of access to large clusters built from these components. In our experiments, each node works on a subdomain size of 512^3 and we report the performance per node.

9.1 MPI Communication

Most distributed memory implementations use the Message Parsing Interface (MPI) to share data between the nodes. Send and Receive MPI buffers are allocated and used to marshal data extracted from on-node data structures before sending in bulk to neighboring processes. In the case of 3D problem decomposition among MPI processes, the wave equation kernels typically require the 4 neighboring points in each direction. Unfortunately, the corresponding X-Z and Y-Z slabs do not lie consecutive in memory. As such, data must be gathered and packed into the MPI buffers.

We subdivide each time step of our MPI implementation into four steps. First, we extract the surface (six 4-deep slabs) of our local 3D grids and pack into six separate send buffers. TTI requires we also extract the 4x4-deep edges of the volume and pack into 12 corresponding buffers. ISO requires communicating the current wavefield while VTI and TTI require communicating the auxiliary grid as well. Second, we execute an MPI send/recv pair so that we exchange halo data and receive the incoming messages into receive buffers. Third, after receipt of the messages, we unpack the buffers and scatter data to the halo of the local 3D grids. Finally, we perform the wave equation stencils advancing the state by one time step.

9.2 Optimization for Communication

As a great deal of data is moved at each time step, optimization of data packing and unpacking is important for multicore, NUMA node architectures. As the multi- and manycore processors continue to improve the performance of the wave propagation kernel, optimization of the time spent in communication will become critical.

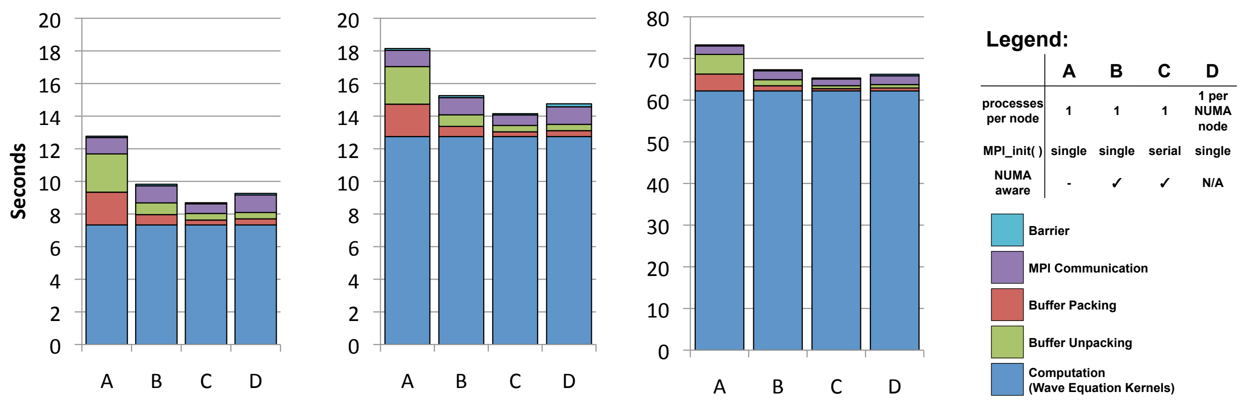
We implement four versions of the communication phase, labeled "A"–"D". Implementations "A"–"C" instantiate one process per node (thereby incurring NUMA challenges) while optimization "D" attempts to obviate NUMA by using one process per NUMA node and partitioning the 512^3 volume into slabs of size $512^2 \times 128$. In optimizations "B" and "C", we attempt to allocate buffers in a NUMA-aware fashion so that we may parallelize the buffer packing/unpacking in a manner that guarantees threads with affinity to the buffer also have affinity to the grid points they are responsible for accessing. Nominally, the use of `MPI_THREAD_SINGLE` demands the lead thread perform all communication. As this would require that thread to potentially access buffers on remote NUMA nodes, we explore the use of `MPI_THREAD_SERIALIZED` in which the thread responsible for a packing/unpacking a buffer is also responsible for initiating messages from that buffer. These combinations of optimizations are highlighted in the legend of Figure 5.

9.3 Results

Figure 5 presents how much time is spent for each of these steps as a function of progressive levels of communication optimization (labeled "A"–"D") on Hopper.

We observe that the naive approach of using one process per compute node can have performance penalties. The MPI overhead for the ISO and VTI implementations constitutes roughly 40% and 30% of the time. As VTI must communicate disjoint parts of two grids, the MPI time remains quite similar to ISO as the total volume of data movement remains similar.

NUMA-aware allocation and the inherently increased number of threads (one per NUMA node), which access MPI buffers (optimization "B"), reduces the time spent in buffer packing and unpacking by a factor of $3.2\times$.



	A	B	C	D
processes per node	1	1	1	1 per NUMA node
MPI_init( )	single	single	serial	single
NUMA aware	-	✓	✓	N/A

(a) ISO (b) VTI (c) TTI

Figure 5: Breakdown of time spent in each phase of the 4 different MPI implementations for the ISO (a), VTI (b) and TTI (c) kernels on Hopper (Opteron-based Cray XE6).

Elimination of the NUMA effect observed by the thread initiating MPI_Send/Recv (implementation "C") can reduce the time spent in MPI communication by a factor of 1.9× on ISO, and by 20% on VTI.

Improved communication time is achieved for the ISO kernel. VTI still sees a respectable improvement of 20% but with only minor improvements to the MPI time for TTI. Moreover, fully threading within a NUMA node can improve buffer packing and unpacking by a factor of 1.8× for all kernels.

It has been suggested that one may obviate NUMA by placing one MPI process on each socket (implementation "D"). Unfortunately, this causes additional overhead and a worse surface:volume ratio due to data replication and halo communication. Although performance is maximized by avoiding NUMA and threading buffer copies, the net overheads still increase run time significantly.

Overall, we attain an improvement of 30%, 18% and 10% for ISO, VTI and TTI, respectively. Unfortunately, a 17%, 7% and 5% MPI overhead exists for these kernels, which reduce the overall energy efficiency by 15%, 10% and 5% compared to the single node benchmarks presented in Section 8.

10 Conclusions

In this report, we examined the performance, optimization, and energy efficiency of three progressively more challenging wave propagation kernels used in Reverse Time Migration. We evaluated these metrics across three modern CPU architectures as well as NVIDIA's M2090 GPU. Moreover, we examined the performance of distributed memory versions on an Opteron-based system.

We found that user management of data locality and parallelism (both coarse- and fine-grained) were key on both CPUs and GPUs to attaining the up to 11× speedups. Not surprisingly, we found that optimization for NUMA is increasingly important on dual- and quad-chip single-node experiments. However, our distributed memory experiments highlight the subtitles in MPI performance that should deter programmers from simply obviating NUMA by running one process per NUMA node. Such an approach actually leads to substantial performance penalties on ISO and VTI.

This report presented a novel common subexpression elimination optimization to TTI kernels, which is able to reduce flop count by 2.5× and hence proved highly beneficial to the overall throughput. The mixed partial derivatives of TTI made it quite unique with respect to optimization and performance analysis. Although such characteristics increase the average arithmetic intensity, optimizing code for deep memory hierarchies was found to be quite challenging. The relatively simple, inverted memory hierarchy of the GPUs greatly simplifies the user's management of data locality. However, the relatively small shared memories within each streaming multiprocessor limit attainable performance.

As one contemplates usage of such optimized routines now and on future CPUs and GPUs, we observe there are two major issues that must be confronted. First, ISO and VTI are heavily memory-bound. Although increases in bandwidth result in a proportional increase in performance, increases in bandwidth result in disproportional increases in full system power (bandwidth is currently a minor contributor to full system power). This suggests a great deal of energy is wasted. To avoid such waste, systems must either adapt and reduce clock frequency, or we must contemplate implementation of time-skewing for these high-order wave equation stencils. Second, although we observe all three hardware vendors have dramatically improved on-node computational capabilities, sustained MPI bandwidth across a large network struggles to keep up. One should be mindful that progressively accelerating kernel performance (either through technology, architecture, or implementation) will have asymptotic benefits given as much as 20% of the runtime may be spent in communication.

Nevertheless, when confronted with the daunting and ever increasing demands for high-performance and accurate subsurface seismic survey analysis, it is imperative both academia and industry continue to explore computation and communication optimization techniques as they are a key component in reducing both corporate and consumer energy cost.

Acknowledgments

Authors from Lawrence Berkeley National Laboratory were supported by the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. Jens Krueger was financially supported by Fraunhofer ITWM while a visitor at LBNL.

References

[1] R. Abdelkhalek, H. Calandra, O. Coulaud, J. Roman, and G. Latu. Fast seismic modeling and reverse time migration on a gpu cluster. In *Proceedings of International Conference on High Performance Computing and Simulation, HPCS '09*, pages 36–43. IEEE Press, 2009.

[2] M. Araya-Polo, F. Rubio, M.Hanzich, R. de la Cruz, J.M. Cela, and D.P. Scarpazza. High-performance seismic acoustic imaging by reverse-time migration on the cell/b.e. architecture. In *ISCA2008*, 2008.

[3] Edip Baysal, Dan D Kosloff, and John W.C. Sherwood. Reverse time migration. In *Geophysics*, volume 48(11), 1983.

[4] Javier Cabezas, Mauricio Araya-Polo, Isaac Gelado, Nacho Navarro, Enric Morancho, and José María Cela. High-performance reverse time migration on gpu. In *SCCC*, pages 77–86, 2009.

[5] Robert G. Clapp, Haohuan Fu, and Olav Lindtjorn. Selecting the right hardware for reverse time migration. *The Leading Edge*, 29(1), 2010.

[6] G.C. Cohen. *Higher-order numerical methods for transient wave equations*. Scientific computation. Springer, 2002.

[7] Kaushik Datta, Shoaib Kamil, Samuel Williams, Leonid Oliker, John Shalf, and Katherine Yelick. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Review*, 51(1):129–159, 2009.

[8] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, et al. Stencil Computation Optimization and Auto-Tuning on State-of-the-art Multicore Architectures. In *Proceedings SC '08*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[9] Kaushik Datta, Samuel Williams, Vasily Volkov, Jonathan Carter, Leonid Oliker, John Shalf, and Katherine Yelick. Auto-tuning the 27-point stencil for multicore. In *In Proc. iWAPT2009: The Fourth International Workshop on Automatic Performance Tuning*, 2009.

[10] Eric Dussaud, William W. Symes, Paul Williamson, Larent Lemaistre, Paul Singer, Bertrand Denel, and Adam Cherrett. Computational strategies for reverse-time migration. In *SEG*, Las Vegas, 2008.

[11] Darren Foltinek, Daniel Eaton, Jeff Mahovsky, Peyman Moghaddam, and Ray McGarry. Industrial-scale reverse time migration on gpu hardware. In *SEG Houston International Exposition*, 2009.

[12] Peter Kogge. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, September 2008.

[13] Jens Krueger, David Donofrio, John Shalf, Marghoob Mohiyuddin, Samuel Williams, Leonid Oliker, and Franz-Josef Pfreund. Hardware/software co-design for energy-efficient seismic modeling. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 73:1–73:12, New York, NY, USA, 2011. ACM.

[14] A. Lim, S. Liao, and M. Lam. Blocking and array contraction across arbitrarily nested loops using affine partitioning. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 2001.

[15] Wei Liu and et al. Anisotropic reverse-time migration using co-processors. In *SEG Houston International Exposition*. SEG, 2009.

[16] Yang Liu and Mrinal K Sen. Advanced finite-difference methods for seismic modeling. *GEOHORIZONS*, December 2009.

[17] Paulius Micikevicius. 3D finite difference computation on GPUs using CUDA. In *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, 2009.

[18] S. Morton, T. Cullison, and P. Micikevicius. Experiences with seismic imaging on gpus. In *EAGE W08: Computing Trends in Oil and Gas*, 2008.

[19] Francisco Ortigosa, Mauricio Araya-Polo, Felix Rubio, Mauricio Hanzich, Raul de la Cruz, and Jose Maria Cela. Evaluation of 3d rtm on hpc platforms. In Barcelona Supercomputing Center, editor, *SEG*, 2008.

[20] G. Rivera and C. Tseng. Tiling optimizations for 3D scientific computations. In *Proceedings of SC'00*, Dallas, TX, November 2000. Supercomputing 2000.

[21] Xiang Du Robin Fletcher and Paul J.Fowler. Stabilizing acoustic reverse-time migration in tti media. In *SEG*, 2009.

[22] S. Sellappa and S. Chatterjee. Cache-efficient multigrid algorithms. *International Journal of High Performance Computing Applications*, 18(1):115–133, 2004.

[23] William W. Symes. Reverse time migration with optimal checkpointing. In *SEG*, 2007.

[24] Leon Thomsen. Weak elastic anisotropy. In *Geophysics*, volume 51, pages 1954–1966, 1986.

[25] Reverse time migration with random boundaries. Reverse time migration with random boundaries. In *SEG*, 2009.

[26] Top500 Supercomputer Sites. `http://www.top500.org`.

[27] S. Williams, A. Watterman, and D. Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. *Communications of the ACM*, April 2009.

[28] www.streambench.org. The stream benchmark: Computer memory bandwidth, 2011. `http://www.streambench.org/`.

[29] R.P. Fletcher X.Du and P.J. Fowler. Pure p-wave propagators versus pseudo-acoustic propagators for rtm in vti media. In *EAGE*, 2010.