Fast Scalable Reverse Time Migration Seismic Imaging on Blue Gene/P

Michael Perrone, Lurng-Kuo Liu, Ligang Lu, Karen Magerlein, Changhoan Kim
IBM TJ Watson Research Center
1101 Kitchawan Rd. Yorktown Heights, NY 10598
{mpp,lkliu,lul,kmager,kimchang}@us.ibm.com

Irina Fedulova, Artyom Semenikhin
IBM Russia Systems and Technology Lab
30 Obrucheva St., Moscow
{irina,artyom}@ru.ibm.com

ABSTRACT
We present an optimized Blue Gene implementation of Reverse Time Migration, a seismic imaging algorithm widely used in the petroleum industry today. Our implementation is novel in that it uses large communication bandwidth and low latency to convert an embarrassingly parallel problem into one that can be efficiently solved using massive domain partitioning. The success of this seemingly counterintuitive approach is the result of several key aspects of the imaging problem, including very regular and local communication patterns, balanced compute and communication requirements, scratch data handling, multiple-pass approaches, and most importantly, the fact that partitioning the problem allows each sub-problem to fit in cache, dramatically increasing locality and bandwidth and reducing latency. This approach can be easily extended to next-generation imaging algorithms currently being developed. In this paper we present details of our implementation, including application-scaling results on Blue Gene/P.

Categories and Subject Descriptors
G.1.6 [**Numerical Analysis**]: Optimization. D.2.8 [**Software Engineering**]: Metrics – *performance measures*. J.2 [**Physical Sciences and Engineering**]: Earth and atmospheric sciences. G.1.8 [**Numerical Analysis**]: Partial Differential Equations – *Finite difference methods.*

Keywords
Seismic Imaging, Reverse Time Migration.

1. Introduction
This paper represents a dramatic departure from common high performance computing practice: we show that in certain circumstances, recasting embarrassingly parallel problems as domain partitioned ones can dramatically improve performance. This statement might border on sacrilege for some because of the many advantages embarrassing parallelism has to offer, including better Amdahl's Law scaling, minimal inter-process communication and synchronization, freedom from frequent checkpointing and ease of recovery from the loss of individual processes. These advantages make embarrassingly parallel implementations extremely compelling when possible. Why then would we choose to go against this received wisdom? The answer is that, due to various architectural constraints, the benefits of data locality and massive local bandwidth can sometimes be crafted to outweigh the benefits of embarrassing parallelism.

In this paper, we explore these ideas in the context of production seismic imaging: physics-based signal processing used by the energy industry to find oil and gas reservoirs. We begin with an introduction to the problem of seismic imaging and a description of the widely used Reverse Time Migration (RTM) method, comparing and contrasting our approach to the one commonly used in the industry. We then describe our implementation of RTM, the optimizations that were performed, the experimental setup and the performance results, comparing where appropriate to other RTM implementations.

SC'11, Nov. 12-18, 2011, Seattle, WA, USA.

2. Seismic Imaging
Seismic imaging is the process of converting acoustic measurements of the Earth into images of the Earth's interior, much like ultrasound for medical imaging. It is widely used in oil and gas exploration and production to identify regions that are likely to contain hydrocarbon reservoirs and to help characterize known reservoirs to maximize production. These methods have become critical to the energy industry as known reserves are used up and new reserves become increasingly difficult (and expensive) to find and are increasingly in technically challenging areas, like the deep sea.

For the past several decades, the energy industry has tried to balance the need to image quickly and the need to image accurately. The need for accuracy is driven by the high cost of drilling a "dry" well due to poor imaging (a deep sea well can cost over $100 million) and the need for quick imaging is driven by the cost of not finding new reserves (i.e., bankruptcy). To minimize these costs, the industry relies on supercomputing clusters and regularly increases compute power, enabling both faster imaging on existing algorithms and the practical implementation of more accurate imaging. Thus, the development of fast, efficient methods for imaging is of high importance to the industry.

2.1 Seismic Data
Seismic imaging data varies widely depending on how and where the data is collected (e.g., on land, at sea, at the ocean surface, at the ocean floor, below ground, electromagnetically, etc). We focus here on the data collection method that is most relevant to the RTM algorithm analyzed in this paper: towed hydrophone receiver arrays for ocean seismic data collection. The basic idea is shown in Figure 1. A ship is shown towing a 2D array of hydrophones spaced about every 25m on 1 to 16 trailed streamers. Every 15 or so seconds, an air cannon is fired into the water,

creating an acoustic wave that propagates through the water and into the Earth. Reflections from various surface and subsurface boundaries cause echoes that reflect back and are recorded by each hydrophone in the array. The recording of a single hydrophone in time as a trace and the collection of traces for a single firing of the air cannon is called a **common shot gather**, or **shot**. As the ship moves, a large set of spatially overlapping shots is recorded. Depending on the size of the survey region to be imaged, this data collection can take a month or more and is designed to get the maximal coverage of the area to be imaged. For our purposes, we need to know that we have lots of shots, potentially hundreds of thousands, and that the receiver data collected is the result of some source data at a particular location. A sample of artificial shot data is shown in Figure 2.

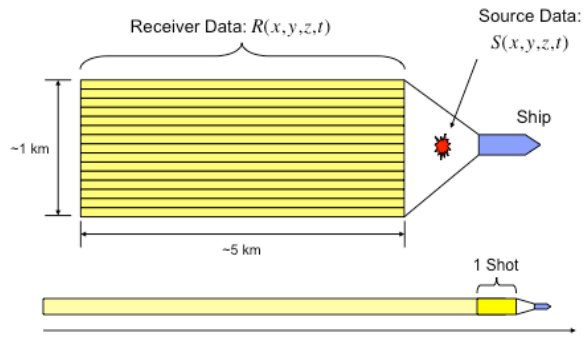

Figure 1: A ship collecting seismic data using a towed hydrophone receiver array

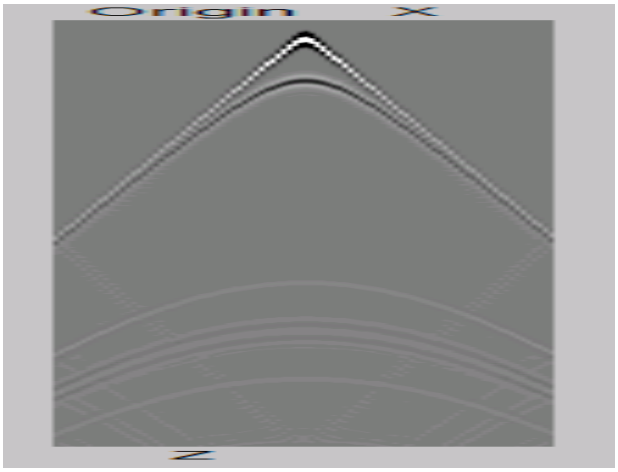

Figure 2: Sample shot data for a 1D array of hydrophones showing time on the Y-axis and spatial offset on the X-axis. The direct source signal propagates out linearly in time (from the center of the array) and appears as straight lines. The recorded reflections appear as curved lines.

2.2 The RTM Algorithm

The Reverse Time Migration (RTM) algorithm is widely used in the industry because of its superior imaging accuracy for difficult subsurface structures like salt domes which are poorly imaged by other algorithms but which are very effective at trapping oil and gas. Several variants of RTM exist with differing degrees of approximation to reality, all of which use single-precision arithmetic. For this paper we implemented isotropic, acoustic RTM which assumes the wave velocity is independent of wave direction and that no energy is absorbed by the medium.

The RTM algorithm arises from the observation that pressure waves should be correlated at reflection boundaries; so RTM proceeds by correlating two pressure waves (called the forward and backward waves) to find those boundaries. To generate the waves for correlation, RTM simulates wave propagation using the wave equation below for a wave U(x,y,z,t) with a source term S(x,y,z,t):

$$\frac{1}{c^2}\frac{\partial^2 U}{\partial t^2} = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} + S \quad (1)$$

The **forward wave** is the wave generated from the air cannon firing and propagating forward in time using a "**velocity model**" represented by C(x,y,z), which specifies the wave velocity at each point in space and represents the various material properties and boundaries of the volume being imaged. The air cannon firing is treated as a wavelet impulse localized in time and space. The **backward wave** is generated by using the shot data recorded by the hydrophone array as the source term for the wave equation and propagating that backward in time. These two waves are then multiplied point-wise at each time step to generate an image, using the following "**imaging condition**":

$$I(x,y,z) = \sum_t U_{Forward}(x,y,z,t) U_{Backward}(x,y,z,t) \quad (2)$$

This process is repeated for all shots in the seismic survey and the images generated are summed to create a final image of the reflecting boundaries, which represent the subsurface structure. It is important to note that the time summation in the imaging condition implies that the first time step of the forward wave needs to be correlated with the last time step of the backward wave. This constraint is typically handled in one of two ways: either the forward wave is saved to disk (called a "**snapshot**") every several time steps and read in for imaging when the backward wave is computed, or the forward propagation is run twice – once forward in time and once in reverse time using boundary data saved from the forward pass to recreate the forward pass in reverse – and then imaging proceeds with the backward wave and the reverse forward wave. The first method requires significant disk storage and can be bottlenecked on disk I/O, while the second requires 50% more computation and additional memory space to save the boundary data.

Following standard practice in the industry [2], we simulate the wave propagation of Equation (1) using the finite difference approximation in Equation (3) where we select the coefficients to implement 2nd order accuracy in time and 8th order accuracy in space. These coefficients are scaled to satisfy the CFL condition [5]. This approach gives rise to the 25-point stencil shown in Figure 3.

$$U_{i,j,k,t+1} = 2U_{i,j,k,t} - U_{i,j,k,t-1} + C^2_{i,j,k} \sum_{n=-4}^{n=4} (A_{xn} U_{i+n,j,k,t} + A_{yn} U_{i,j+n,k,t} + A_{zn} U_{i,j,k+n,t}) \quad (3)$$

In practice, the size of production RTM models varies widely, but the universal desire is to grow models larger to get more resolution and to run the models longer to enable deeper imaging since echoes take longer to reflect from deeper structures.

Typically, velocity models for individual shots are 512^3 to 1024^3 elements or larger and the number of time steps can be 10,000 or more in both the forward and backward propagation phases.

Seismic imaging is typically computed using single precision arithmetic and we take that approach here. Some practitioners believe that the RTM method described above, that avoids the need to save snapshots, must be run in double precision; however, we do not implement that version here.

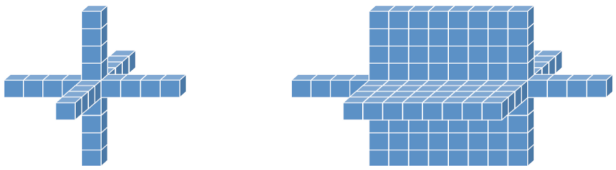

Figure 3: 25-Point spatial stencil with 8th order accuracy shown in isolation on the left and as it moves along the stride-1 dimension of the model

2.3 Embarrassingly Parallel RTM

Industrial implementations of RTM are embarrassingly parallel. They typically run individual shots on one to two nodes of a compute cluster and run many shots in parallel (see Figure 4). These clusters have minimal network connectivity because it is not needed: the individual shots run independently and asynchronously. A simple work queue is used to manage runs and if a run for a shot fails, it is simply re-run, as it doesn't impact any of the other runs. A master process of some kind is needed to manage the work queue and to merge the partial images that are generated from each shot. Additionally, other image processing might be included in this process, but for our purposes here we ignore these details as the RTM calculation is the main computational workload.

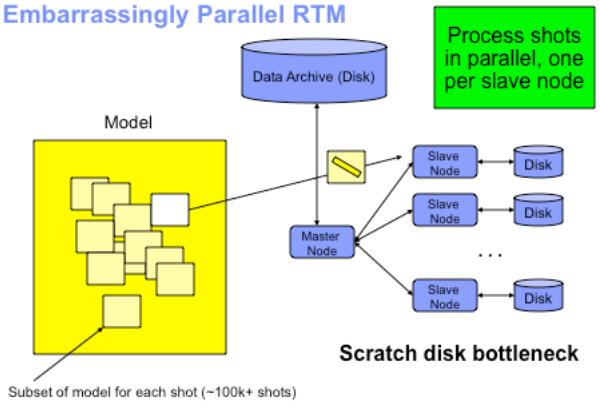

Figure 4: Embarrassingly parallel RTM implementation

RTM compute clusters have significant per-node scratch disk requirements for saving snapshot data, which for a 1024^3 model and 10,000 time steps would require 40TB of snapshot storage – per shot! In practice, snapshot subsampling is used to reduce both disk requirements and disk I/O bottlenecks; however subsampling results in image degradation and must be balanced with performance. Compression can be used to trade computation for disk I/O, but if lossy, compression can also degrade image quality.

As FLOPS per processor increase, the embarrassingly parallel implementations become disk I/O bound [2]. It is possible to improve performance by partitioning single shot gathers over multiple nodes; however, such implementations typically use only a handful of nodes. We have developed an RTM implementation that extends domain partitioning over thousands of nodes on a Blue Gene/P supercomputer and results in dramatic performance improvements.

2.4 Domain-Partitioned RTM

We have developed, implemented and tested a 3D isotropic RTM code that uniformly partitions the wave equation domain in blocks over thousands of nodes (see Figure 5). The partitioning over so many nodes means that the size of the model on each node is about 1000 times smaller than for standard RTM on a handful of nodes and provides five main benefits: (1) all forward wave snapshots can be stored in main memory, removing the need for disk and thereby improving per-node data bandwidth from hundreds of MB/s (for disk) to tens of GB/s (for main memory), effectively removing the disk I/O performance bottleneck; (2) the partitioned models can fit in processor cache, allowing processing to proceed at the speed of the cache memory bandwidth instead of main memory bandwidth which can be an order of magnitude larger on some systems; (3) we can load entire 3D seismic surveys into memory on one or more racks, enabling "in-memory" processing of algorithms that require multiple passes through the data set, such as Full Waveform Inversion (FWI), and thereby avoiding additional disk I/O bottlenecks and enabling better performance; (4) keeping the models in the processor's cache means that snapshot reading/writing has full access to main memory bandwidth and is not a bottleneck; and (5) this method can run an entire velocity model instead of a subset, as is typically done in standard RTM, allowing us to easily extend this method to include multisource processing with minimal code changes and significant potential performance gains [4].

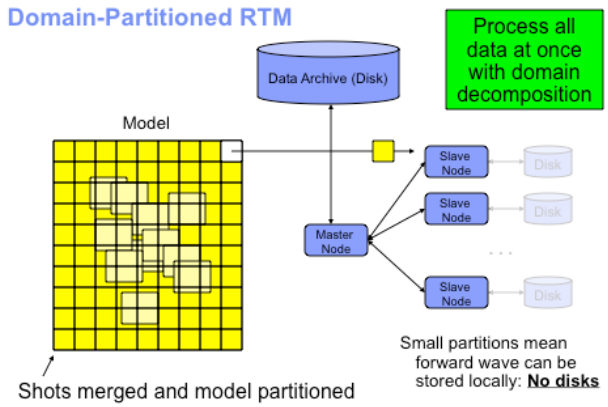

Figure 5: Domain-partitioned RTM Implementation

A critical aspect of domain-partitioned RTM is that current wave data from neighboring sub-domain boundaries is required for stencil calculations at each time step. Since this boundary data transfer grows with the amount of partitioning and with the size of the stencil used, it can easily become a performance bottleneck. To avoid communication bottlenecks, we implemented our partitioned RTM on a Blue Gene/P supercomputer, which is designed specifically for extremely efficient inter-node communication.

3. Blue Gene Architecture

Our performance measurements were all performed on subsets of two racks of Blue Gene/P. It is beyond the scope of this paper to give a full description of these machines. Instead we focus on those features that are relevant. More details can be found elsewhere [1].

The Blue Gene/P (**BGP**) supercomputer has 1024 nodes per rack running at 0.85GHz. Each node has 4 single-threaded cores, 4GB of RAM per node (4TB per rack) and an extremely high-bandwidth, low-latency, nearest-neighbor 3D torus topology network in which each node is connected to each of its 6 nearest neighbor nodes by 850MB/s of send+receive bandwidth (i.e., 5.1GB/s per node and 5.22TB/s of communication bandwidth per rack). Because of this massive bandwidth, BGP is ideally suited for physical modelling involving extensive nearest-neighbor communication and synchronization – like RTM. The nearest neighbor latency for 32B data transfers is about 0.1 microseconds and is essentially amortized away for larger block transfers required by RTM. Each compute node core has a 32KB L1 cache with a 32B cacheline and a shared 8MB L3 cache with a 128B cacheline. Each node has two memory channels with an aggregate bandwidth of 13.6 GB/sec to main memory. BGP compute notes are connected via dedicated I/O nodes to a GPFS file system based on three DDN S2A9900 couplets attached to the BGP I/O nodes via 10 Gigabit Ethernet connections, providing ~16GB/s of disk I/O bandwidth per rack. Each node can operate in **SMP** mode as a unit, or as four "virtual" nodes. The Virtual Node (**VN**) model avoids the need to explicitly use multithreading at the node level and thereby eases programmability. Each core has a 2-way SIMD unit.

4. Implementation Details

In this section we describe various implementation details that were important to our performance optimization.

4.1 Ping-Pong Buffering

Equation (3) uses four 3D data objects: the past, present and future waves and the velocity model. To increase the locality of our model, we use a ping-pong buffer pair, holding the current wave in one buffer and the future and past waves in the second buffer. This buffering is possible because once the future wave point is calculated, the past wave point is no longer needed and can be overwritten with the future value. This buffering reduces RTM's cache size requirements by 25% and thereby allows for processing larger models more efficiently.

4.2 Trade-Off Analysis

An analysis of the various trade-offs made in this implementation of RTM is helpful in guiding the choice of operational parameters. This analysis shows that various system constraints prevent us from running at the theoretically optimal operational parameters.

Consider a cubic velocity model of size N^3 elements which is uniformly partitioned over K^3 nodes such that each node is responsible for processing a sub-volume of size $V=N^3/K^3$. For any sub-volume, we can estimate the time required to compute the stencil over all the elements of the corresponding sub-volume and the time required to communicate boundary data to and from its nearest neighbors. An ideal balanced implementation would have equal time for these tasks so as to efficiently utilize all the machine's resources. In practice this is not possible for a variety of reasons; however we can still use this goal to guide our system design.

The 2^{nd} order in time and 8^{th} order in space finite difference method used in Equation (3) to approximate wave propagation gives rise to ~32 floating-point operations for each stencil calculation (depending on the details of the assembly implementation), if one precomputes the spatial and temporal deltas into the stencil parameters. This precomputation is possible here since the deltas are constant for RTM. If we let F be the peak number of FLOPS per node, then the total time to compute each sub-volume is bounded below by $T_{Compute} = 32(N/K)^3/F$.

For each pass of the stencil over a wave sub-volume, the boundary regions need to be communicated between nearest neighbor nodes. Since the Blue Gene torus network allows nodes to send and receive simultaneously, and since it has independent paths for each of the spatial dimensions, we can assume that these transfers all happen at approximately the same time for each node. Further, since the algorithm sends the same amount of data between all nearest-neighbor nodes, we only need to consider the time of a single boundary transfer to characterize the communication behavior of the node.

The amount of data transferred for each finite difference time step is 4 bytes per element, 4 elements per stencil calculation and one stencil calculation for each element on a face of the sub-volume. Dividing this data by the peak torus send bandwidth, D, between each node gives a total time of $T_{Data} = 16(N/K)^2/D$.

This analysis shows that $T_{Compute} / T_{Data} = 2N/KFD$. For an ideal balanced system, this ratio would be one, and getting there would simply be a matter of choosing appropriate N and K. However, we have additional constraints that prevent us from choosing N and K arbitrarily. In particular, we would like to store all of the RTM models (velocity and two time steps of wave volume) in cache because complete cache locality gives a dramatic performance advantage over systems that need to use main memory. This means that for this RTM implementation, we need to fix 3 sub-volumes of size V in cache. This means $V<8/3$ Mbytes for BGP. Since $V=N^3/K^3$, we see that $N/K<89$ which implies $N/K<56$ per core. For a rack, this means a model of size 880^3 fits in cache.

In practice, there are several additional constraints on the block dimensions. The L1 cache line length imposes a preferred granularity in the stride-one (z) dimension of 8 floats (32B). The cache can be used more effectively if the number of cache lines in the z dimension is not a factor or multiple of the number of sets in the set-associative cache (16 for BGP), since otherwise memory accesses will be concentrated in some portions of the cache while other portions remain unused. Cache tiling, described below, makes it convenient to have each dimension of the block be a multiple of the corresponding tile dimension. Such considerations suggest that better performance may be achieved with a block size of 54x54x56 rather than 55^3. Choices of this nature trade kernel performance and MPI performance since asymmetry to favor stride-one dimension efficiency leads to higher communication requirements. We can use this choice to help balance our implementation.

Additionally, we have designed our RTM implementation to save snapshots to main memory to avoid disk I/O bottlenecks. This choice imposes another constraint on the data: the model in cache has to be small enough to allow a sufficient number of snapshots to be saved. Typically production RTM runs can be on the order of five to ten thousand forward iterations, however due to disk storage and bandwidth constraints, practitioners typically subsample the data in time, saving only a fraction of the wave fields according to a pre-specified "snapshot" frequency.

Common snapshot frequencies range from 3-10 iterations per snapshot, depending on the RTM imaging requirements. For Blue Gene, this implies that one can save at most 1500 snapshots (=memory size / one-third the cache size), which imposes a snapshot frequency range of 3-7 iterations per snapshot. If one wants to save more snapshots (e.g., for higher image quality or more time iterations) then one can reduce the size of V and run on more nodes; or reduce the size and/or number of snapshots (e.g., by sub-sampling and/or compressing snapshots); or save some of the snapshots to disk. Our implementation includes all of these options except compression; however, simple lossy 4x compression is easy to implement and can provide adequate image quality [2].

Note that this analysis also shows that for practical values of N and K, T_{Data} is much larger that the MPI latency of both Blue Gene systems. So we are not MPI latency bound.

Our domain partitioning allows us to partition the domain over different numbers of computing nodes and thus take advantage of the cache structure of the platform. When the partitioned sub-volume can fit in processor cache, it allows processing to proceed at the speed of the cache memory bandwidth instead of main memory bandwidth. We evaluate the effect of the sub-volume sizes on the main memory bandwidth requirement on BGP, which has 32KB L1 Cache and 8MB L3 Cache. As shown in Figure 6, the main memory bandwidth requirement is minimized when the sub-volume is 64^3 or less in size. With a larger sub-volume size, data will spill to main memory and thus data access will be conducted at main memory bandwidth and latency. Note that Figure 6 is for a single core, assuming linear scaling to 4 cores per node brings the bandwidth requirement to only ~1GB/s, which is well within the 13.6GB/s peak bandwidth available. Thus we are very far from being bandwidth bound. This result implies that we can easily extend our isotropic model to more accurate RTM versions which increase the model size by a factor of 4 or more due to additional auxiliary data volumes.

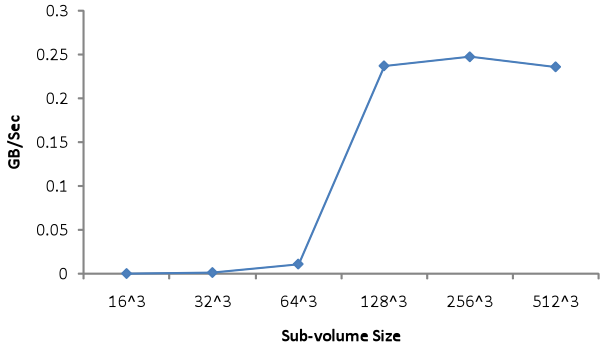

Figure 6: Effect of sub-volume size on memory bandwidth requirements

An important implication of the observation in Figure 6 is that we can extend our isotropic RTM to more sophisticated versions (e.g., VTI [2] and TTI [8]) which require larger data models without becoming bandwidth bound.

4.3 Stencil Kernel Analysis

Stencil computation is the core computational kernel in RTM seismic imaging. As the stencil computation kernel is computationally expensive, it is important to have a good understanding of the workload characteristics for performance optimization. The RTM compute kernel, which is an 8^{th} order in space and 2^{nd} order in time FDTD for advancing the pressure wave, was analyzed for various combinations of model size (256^3, 512^3 and 1024^3), BGP node count (64 and 128), and operation mode (VN and SMP). The full application was run but only the RTM computational kernel was analyzed. We will use the number of stencils computed per unit time as our RTM application performance metric because it normalizes for a variety of run-dependent configuration parameters (e.g., size of the velocity model, number of time steps, number of shots, etc.) which would otherwise complicate the comparison of raw run times. As shown in Figure 7, we find that our RTM application (end-to-end, not just the stencil kernel) achieves 6-7B stencils/sec on 128 BGP nodes using virtual node mode, which is ~22% of peak flops on BGP. Figure 7 shows baseline and optimized performance. Our optimization techniques are described below.

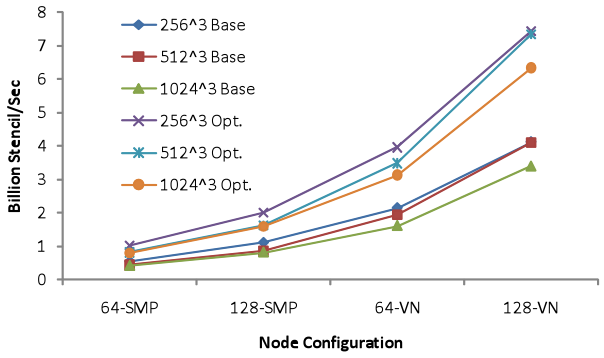

Figure 7: RTM performance

4.3.1 Cache Tiling

We studied the memory bandwidth requirement and L1 cache miss ratio from the RTM runs with different configurations. A cache tiling technique was then used to improve the reuse of data already available in cache. As shown in Figure 8, we studied the cache tiling performance using different tile sizes on a 512^3 model using 128 BGP nodes. We see that small tiles, such as 4x4x4, have a negative impact on performance. There are two major reasons for this: 1) the cache reuse is not effectively realized in the stride-one dimension; 2) the overhead added from the additional tiling *for* loops diminishes the benefit from a small tile size. The study also shows that, given a fixed cache tile, it is better to have a longer tile length on the stride-one dimension in order to make effective use of the prefetched cacheline, e.g., a tile of size 8x8x64 is better than 16x16x16. Figures 9 and 10 show the L1 miss ratio and bandwidth requirement. The kernel requires less than 10% of peak memory bandwidth and has an L1 miss ratio of ~9%.

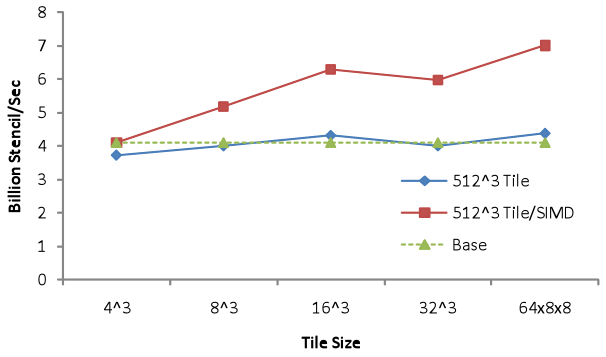

Figure 8: Tiling performance

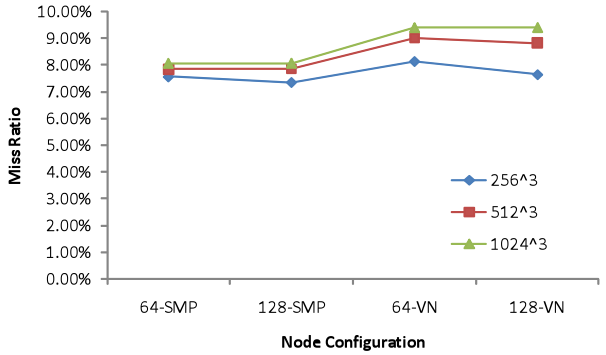

Figure 9: L1 Miss Ratio

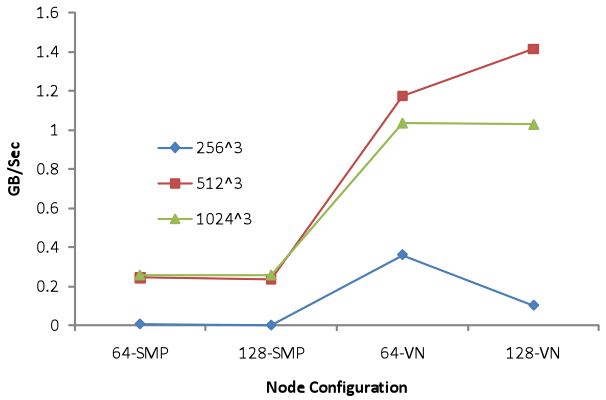

Figure 10: Memory bandwidth requirement

To further analyze the stencil kernel for optimization, we studied the instruction mix of the stencil kernel, focusing on floating point operations, as they form the core of the stencil kernel. The study showed that the stencil kernel employed around 45% of instructions in load operations, which indicated that data is not reused effectively at the instruction level. This brings us to the next level of candidates for performance optimization: instruction level optimization. We looked into the technique of loop unrolling as well as the use of SIMD operations to take advantage of the architecture's processing power. As shown in Figure 11, we achieve about 22% of instructions using fused multiply-add for better processor utilization. The percentage of load operations also dropped from 45% to 36%, achieving better data reuse.

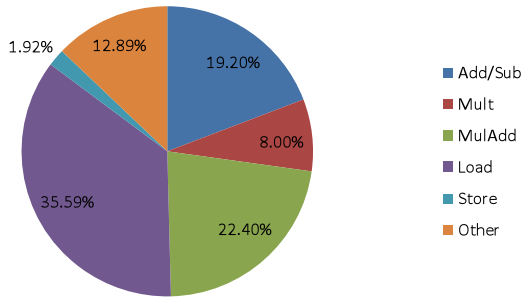

Figure 11: Floating point operation usage

4.3.2 Absorbing Boundary Conditions

In production runs, RTM algorithms use various processing methods to minimize the effect of wave reflection from the (non-physical) model boundaries. In our study, we employed Higdon's absorbing boundary condition [3], which predicts pressure field values at the absorbing boundaries based on the known pressure field values from the previous time step. As seen in Equation (4), these conditions are applied per dimension and can be tuned absorb plane waves incident with the boundary at various angles, alpha. These constraints are discretized in the usual way and for our purposes we used a single angle per dimension chosen to minimize image noise.

$$\left[\prod_{j=1}^{p}\left(\cos\alpha_j\frac{\partial}{\partial t}-c\frac{\partial}{\partial x}\right)\right]U=0 \tag{4}$$

These methods add overhead for stencil computations at the model boundaries, leading to load imbalance between nodes. This imbalance can be significant, causing the maximum and minimum kernel time to vary by as much as 1.5x. Future work will attempt workload balancing with non-uniform partitioning – making nodes working on the model boundary process less data.

4.3.3 SIMD Considerations

Exploitation of BGP's 2-way SIMD instructions is a key element in the efficiency of our code. To use the SIMD instructions efficiently, the z (stride-one) dimension of each partition must contain a multiple of 2 elements, with the beginning of each z line aligned on at least a 8-byte boundary. In preparation for the next generation of Blue Gene technology, we have designed our partitioning code to be slightly more restrictive by assuming that the problem size is a multiple of 4 in the z dimension. The addition of some special-case code to pad along one edge could remove this limitation. The stencil-processing loop in the kernel was then rewritten to use SIMD operations.

For the four-byte-thick boundaries we are using, SIMD can also be applied to the absorbing boundary condition computations. Some additional improvement was achieved by pre-calculating the coefficients used in updating the outer boundary layer. These are functions of position; they are independent of the time step. We pre-calculated seven coefficients for each point (the eighth is a constant). This gives us an improvement in speed at a small cost in storage. It replaces somewhat scattered references to the velocity field with sequential reads of a (larger) array of coefficients, improving storage locality; and it removes computations and the operands they require from the loop, saving time and freeing up registers. On BGP, these two changes

combined give a 29% improvement in the time required for boundary processing in the 256x256x256 case on 64 cores, which translates into an overall improvement of 5.8% for that case.

4.3.4 Loop Unrolling

In our first attempt of stencil kernel SIMDization, we computed two elements simultaneously using the BGP two-way SIMD unit for each inner loop iteration. To effectively use the instruction pipeline and reduce loop management overhead, we employed loop unrolling inside the inner loop iterations. In our study, we unrolled the inner loop so that four elements can be computed in each inner loop using SIMD operations. Based on the instruction mix analysis, we were able to increase the fused multiply-add instructions from 19% to 22% and reduce the other non-computation-related instructions from 20% to 12%. This enables better efficiency in stencil kernel computation from instructions.

We optimized our RTM stencil kernel computation using a variety of techniques, including partitioning, cache tiling, SIMDization, loop unrolling, etc. Figure 12 shows the contribution of performance improvement from different optimization techniques. Overall, these techniques gave a 1.7-1.9x performance gain.

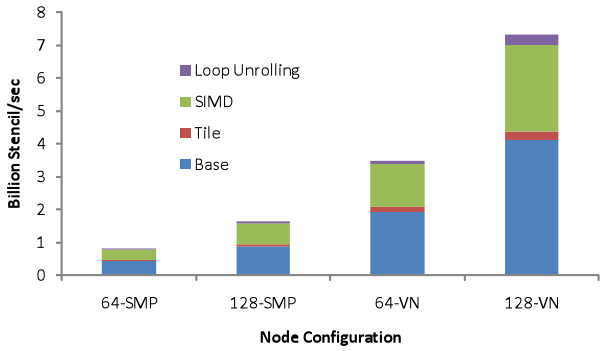

Figure 12: RTM performance improvement from different techniques

5. Experimental Setup and Results

We implemented a full, 3D isotropic RTM using a 25-point (8^{th}-order) stencil in the compute kernel. Absorbing boundary conditions were applied to the model boundaries to reduce artificial reflections. RTM imaging performance analysis was conducted on two different velocity model sizes, 512^3 and 1024^3, using 1408 trace samples in each receiver trace and 1408 time steps in both the forward and backward passes. The velocity models were partitioned evenly on the X, Y and Z axes so that each node processed an equal-size, contiguous, 3D sub-block of the whole. Due to the 25-point stencil, each node had to read and write four 2D sheets of data from and to the set of nearest neighbor sub-block nodes. Our code has been SIMDized, can be run with one core per node (SMP mode) or four cores per node (VN mode) and does not overlap MPI with computation.

We demonstrate the performance of our method with emphasis on scalability. Performance was measured on one and two racks of BGP varying the number of compute nodes (from 128 to 2048 nodes or, equivalently, 512 to 8196 virtual nodes). We also compare the performance impact of SIMD versus non-SIMD as well as one core per node versus virtual node (VN) mode (four cores per node).

Previously [6], we have shown that storing the snapshots in main memory provides a significant performance advantage over storing them to disk. In this paper, we continue to use main memory storage for snapshots.

5.1 Profiling Analysis

For performance profiling purposes, the total run time of the application falls into several categories: (1) A one-time initialization phase (which was not optimized) and which in practice would be amortized over potentially hundreds of thousands of shots; (2) a per-shot forward processing time composed of boundary data exchange, core computation and boundary condition computation; (3) a per-shot backward processing time which is similar to the forward but with the addition of an imaging condition computation; and (4) a per-shot image writing time. Our profiling analysis measured the critical components of the code, including the forward pass loop, the backward pass loop, and the final image write. We further break down the forward pass into forward stencil calculation, boundary calculation, and boundary exchanges. Likewise, we break down the backward pass into backward stencil calculation, boundary calculation, boundary exchanges, and imaging condition calculation. Figure 14 shows an example of the profiling output, which is well balanced with the exception of the absorbing boundary condition calculations (FwdABC and BwdABC).

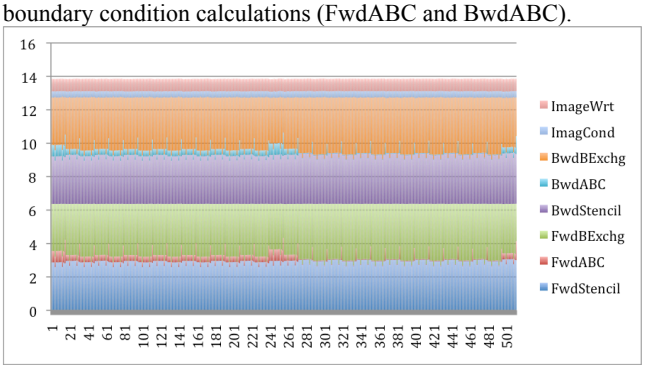

Figure 14: Per core profiling time (in seconds) for the first 512 cores of a 4096 core RTM run on a 512^3 model

Figure 14 shows that compute time and communication time are well balanced. We are currently working to implement overlapping between computes and communication. Optimally, overlapping will result in another performance factor of ~2x. Note that the code for image writing (to disk) has not been optimized as it represents a checkpointing operation and can be amortized over multiple RTM imaging runs. Furthermore, there is an initialization time (not shown in Figure 14) which grows with the number of nodes (~12 seconds for 1024 nodes) but which has not been optimized as it is only invoked once for an entire RTM survey (hundreds of thousands of shots) and is therefore negligible.

5.2 Impact of Snapshot Frequency

We have previously shown [6] that relying on disk I/O for RTM snapshot storage becomes a bottleneck that prevents scaling. For the purposes of this paper, we therefore consider only snapshot storage in main memory. Note that our RTM implementation is largely insensitive to snapshot frequency because BGP has sufficient main memory bandwidth to save a snapshot every time step without it becoming a bottleneck even for the largest models that fit in cache. This fact can be seen by comparing the snapshot size and compute time for a single iteration with the main memory bandwidth, and it has been measured experimentally.

5.3 Scaling Performance

To analyze our optimization results and evaluate the performance of our RTM implementation, we conducted tests varying velocity model size, number of compute nodes used and the number of cores per node. We tested our RTM code on two sizes of velocity models: a 512^3 cubic model and a 1024^3 cubic model. To evaluate the scaling performance of our RTM implementation, we conducted the tests on different numbers of computing nodes ranging from 128 nodes up to 2048 nodes. We also conducted tests and evaluated corresponding performances on SIMD vs. non-SIMD on different number of cores per node.

Figures 15 to 18 provide the time performance plots and scaling performance of test results. For example, our end-to-end RTM implementation can process about 3000 iteration of a 512^3 model in 9.71 seconds, a 1024^3 model in 42.97 seconds, achieves 40M stencils per node per second, and can easily handle models as big as 2048^3 or larger. The figures also show that our RTM implementation scales very well across the number of computing nodes, especially in the larger model case. Experimentally, on-node scaling of the RTM computational kernel (with and without SIMD and 1 to 4 cores) approaches the theoretical maximum of 4x.

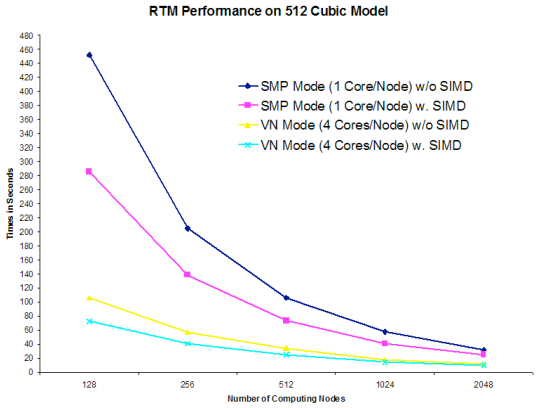

Figure 15: RTM performance for a 512^3 model

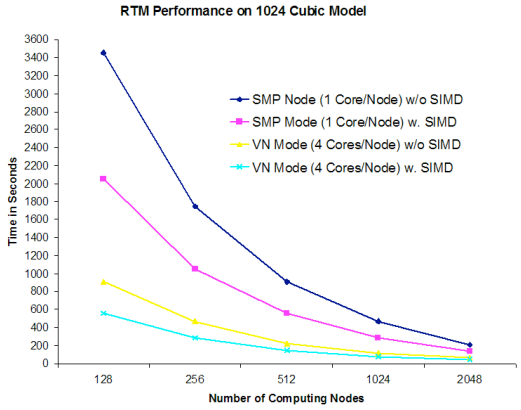

Figure 16: RTM performance for a 1024^3 model

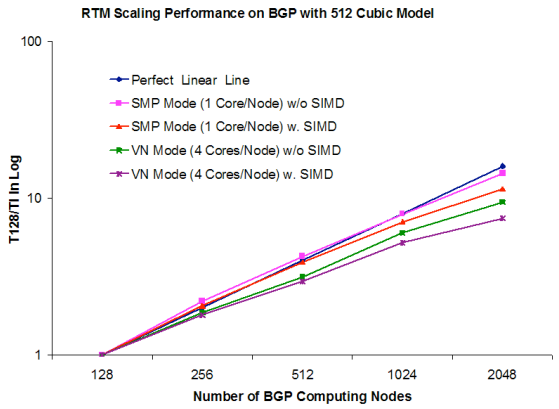

Figure 17: RTM scaling performance for a 512^3 model

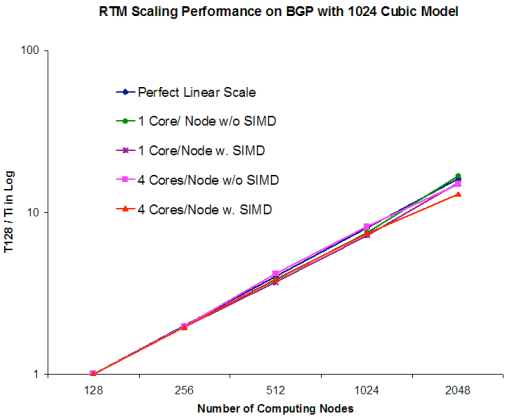

Figure 18: RTM scaling performance for a 1024^3 model

5.4 Performance Comparison

In [7], a rack of 8 Tesla S1070 (32 GPUs) and 16 Harpertown CPU nodes achieved a processing rate of 232 shots per day per rack on a velocity model of size 560x560x905 with 22760 time steps (forward+backward). For comparison, the GPU rate is equivalent to about 17B stencil calculations per second (=(232x560x560x905x22760 stencils/day) / (24x60x60 sec/day)) while the equivalent number for BGP is 40B stencils/second (=1024x1024x1024x2816 stencils / 75 sec, where the 75 sec run time comes from a one rack BGP measurement). Note that our implementation includes the additional computation of absorbing boundary conditions while the GPU implementation does not. Since we are currently well balanced between computes and communication on BGP and we have yet to optimize our MPI implementation, we are optimistic that performance can be improved further.

In [9], an efficient 25-point, RTM kernel implementation achieved an impressive 10.8B stencils per second on a Tesla S1070 with four GPUs. However, this result is not for an end-to-end application. It has no imaging condition, no snapshot I/O, no image write, no absorbing boundary condition, no disk I/O and requires host CPU systems to operate the GPUs. If we assume 1U for the S1070 server, 1U for the host systems and 1U for the disks required to keep the process from becoming disk I/O bottlenecked, then we could put 14 of these in a 42U rack (e.g., compare to the 8 S1070's used in [7]). Such a rack using embarrassingly parallel scaling might achieve 151G stencils per

second. An apples-to-oranges comparison of this kernel estimate to our end-to-end RTM application on a much larger data model (40B stencils/second) shows a 4x performance advantage. Using [7] as a guide, it is likely all of this advantage and more disappears when moving from kernel to full application.

Furthermore, GPU implementations are more limited by their available memory. This problem becomes especially important when advancing to VTI and TTI RTM, which require considerably larger models. The option is to continue scaling to more GPUs; however this is not easily done. In [9], scaling stopped at 4 GPUs on a single S1070. In [10], a heroic effort was made to scale to 120 GPUs using S1070's and 3D domain partitioning; however, it was found that communication became a bottleneck. The 56GF/s measure on 1 GPU scaled to 2.2 TF/s on 120 – equivalent to 30% of theoretical maximum. This compares poorly to the excellent BGP scaling shown in Figure 18.

6. Conclusions

This paper presents what at first sight might seem like a counter-intuitive idea: replace a straightforward embarrassingly parallel implementation of a difficult real-world problem with a more sophisticated, synchronized, communication intense, massive domain-partitioned approach. However, we have shown that on BGP, this new approach can lead to performance that is competitive with today's industry standard approach. Clearly, this is a commentary on the ability of the BGP system, which is over 4 years old; but it is equally a commentary on the risk of leaving established industrial practices unquestioned. In this case, we have demonstrated that at some point the advantages of cache and communication locality can overcome the advantages of embarrassingly parallel implementations.

The results presented here are especially encouraging when one considers the potential performance boost possible when moving to the next generation of Blue Gene technology.

7. Acknowledgements

The Blue Gene/P project has been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy under Lawrence Livermore National Laboratory subcontract no. B554331. We also acknowledge the support and collaboration of Columbia University and Edinburgh University.

8. REFERENCES

[1] Sosa, C. and Knudson, B. 2009. *IBM System Blue Gene Solution: Blue Gene/P Application Development, IBM Redbooks*, DOI= http://www.redbooks.ibm.com/abstracts/sg247287.html

[2] Zhou, H., Fossum, G., Todd, R. and Perrone, M. 2010. Practical VTI RTM. In *Proceedings of 72nd EAGE Conference*.

[3] Higdon, R. L. 1987. Numerical Absorbing Boundary Conditions for the Wave Equation. *Mathematics of Computation*, Vol. 49:179 July,1987, pps. 65-90.

[4] Boonyasiriwat, B. and Schuster, G. 2010. 3D Multisource Full-Waveform Inversion using Dynamic Quasi-Monte Carlo Phase Encoding. *Geophysical Research Abstracts,* Vol. 12, EGU2010-7298, 20.

[5] Trefethen, L. and Bau, D. 1997. *Numerical Linear Algebra*, SIAM.

[6] Perrone, M., Liu, L., Lu, L. and Magerlein, K. 2010. High Performance RTM Using Massive Domain Partitioning, In Proceedings of *EAGE'2011 Conference*, May, 2010.

[7] Abdelkhalek, R., Calandra, H., Coulaud, O., Roman, J., Latu, G. 2009. Fast Seismic Modeling and Reverse Time Migration on a GPU Cluster. In *International Conference on High Performance Computing & Simulation*, 2009. HPCS'09.

[8] Fletcher, R. P., Du, X. and Fowler, P. J. 2009. Reverse time migration in tilted transversely isotropic (TTI) media. *Geophysics,* Vol 74:6, 2009.

[9] Micikevicius, P. 2009. 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, Washington, D.C., 79-84.

[10] Okamoto, T., Takenaka, H., Nakamura, T. and Aoki, T. 2009. Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition. In *Earth Planets Space*, November, 2010.