Asynchronous Task-Based Execution of the Reverse Time Migration for the Oil and Gas Industry

Amani Alonazi, Hatem Ltaief, David Keyes, Issam Said, Samuel Thibault

Asynchronous Task-Based Execution of the Reverse Time Migration for the Oil and Gas Industry

A. AlOnazi, H. Ltaief, and D. Keyes
Extreme Computing Research Center
King Abdullah University of Science and Technology
Thuwal, Jeddah 23955 Saudi Arabia
{amani.alonazi, hatem.ltaief, david.keyes}@kaust.edu.sa

I. Said
NVIDIA
Oil and Gas Department
Paris, France
isaid@nvidia.com

S. Thibault
Univ. Bordeaux, CNRS, Bordeaux INP,
Inria, LaBRI, UMR 5800
Talence, 33400 France
samuel.thibault@u-bordeaux.fr

***Abstract*—We propose a new framework for deploying Reverse Time Migration (RTM) simulations on distributed-memory systems equipped with multiple GPUs. Our software, TB-RTM, infrastructure engine relies on the StarPU dynamic runtime system to orchestrate the asynchronous scheduling of RTM computational tasks on the underlying resources. Besides dealing with the challenging hardware heterogeneity, TB-RTM supports tasks with different workload characteristics, which stress disparate components of the hardware system. RTM is challenging in that it operates intensively at both ends of the memory hierarchy, with compute kernels running at the highest level of the memory system, possibly in GPU main memory, while I/O kernels are saving solution data to fast storage. We consider how to span the wide performance gap between the two extreme ends of the memory system, i.e., GPU memory and fast storage, on which large-scale RTM simulations routinely execute. To maximize hardware occupancy while maintaining high memory bandwidth throughout the memory subsystem, our framework presents the new out-of-core (OOC) feature from StarPU to prefetch data solutions in and out not only from/to the GPU/CPU main memory but also from/to the fast storage system. The OOC technique may trigger opportunities for overlapping expensive data movement with computations. TB-RTM framework addresses this challenging problem of heterogeneity with a systematic approach that is oblivious to the targeted hardware architectures. Our resulting RTM framework can effectively be deployed on massively parallel GPU-based systems, while delivering performance scalability up to 500 GPUs.**

***Index Terms*—Reverse Time Migration, Task-Based Programming Model, Out-Of-Core Algorithms, Asynchronous Executions, Overlapping I/O with Computation, StarPU OOC**

I. Introduction

Seismic imaging is the process of mapping out subsurface deposits of crude oil, natural gas, and minerals by sensing the response of waves sent into the earth to map its reflectivity distribution. Reverse time Migration (RTM), initiated in the early 1980s [1], is a seismic imaging method that maps the subsurface reflectivity using recorded seismic waveforms. RTM is one of the major workhorse simulations in the oil and gas industry. It calculates high-resolution images by solving the three dimensional acoustic wave equation using seismic datasets recorded during a shot, a particular source location creating an acoustic wave. Applying finite-difference time-domain, the time integration follows an adjoint-state formulation with two successive phases, i.e., forward modeling and backward integration. These phases are mostly composed of stencil computational kernels and I/O operations. RTM belongs to the class of out-of-core algorithms, since its computation requires to offload to disk snapshots of the domain solution at many time intervals during the forward modeling phase. During the backward time integration, these snapshots need to be read back at the same intervals. The backward solution at the current time step is then differenced with its freshly fetched-in counterpart from the forward modeling in order to incrementally calculate the image condition, until the subsurface image from this single shot is eventually obtained.

We are interested in deploying large-scale RTM simulations on massively parallel systems equipped with multiple GPUs. The hardware technology as well as the development of high storage capacities allow to design and implement advanced optimizations for enhancing RTM performance [2]. RTM usually suffers from I/O performance bottlenecks due to unnecessary synchronizations, which prevents overlapping data traffic with computations. Moreover, the RTM simulation in a production mode actually operates several thousand shots running simultaneously in an embarrassingly parallel fashion, until the shot gather phase is initiated and generates the final subsurface image. These inner and outer shot synchronization points in the RTM may create idle time and make the overall RTM simulation running at the speed of the slowest shot. The RTM stresses all components of the underlying hardware systems, i.e., vector units, memory/storage bandwidth, and point-to-point hardware interconnect. Although the workload between shots is usually identical, load imbalance may occur between shots, which impedes performance scalability.

We propose a novel software framework based on a task-based programming model to tackle asynchronous RTM (TB-RTM) for seismic imaging that leverages the performance of multicore and GPU-based systems. The main idea is to taskify the RTM by considering its various kernels as a monolithic task. The resulting TB-RTM can then be translated into a directed acyclic graph (DAG), where nodes represent tasks and edges correspond to data dependencies. The StarPU [3] dynamic runtime system is then employed to map the scheduling of these tasks onto the system' resources. StarPU abstracts the hardware complexity from end-users and provides portability across heterogeneous shared/distributed-memory machines. Once StarPU is integrated as the driving engine, TB-RTM

enables decoupling the I/O operations from the computational kernels. The overall TB-RTM application with StarPU can then foster asynchronous executions, bringing to the fore new opportunities for overlapping primary with non-critical operations. In particular, the write and read I/O operations may be overlapped with tasks that belong to the critical path, such as compute-bound GPU stencil kernels during the forward and backward modeling, respectively. TB-RTM promotes a systematic approach to exploit parallelism at all computational stages, while weakening the strong synchronization points of the native RTM and mitigating the overhead of load imbalance due to on-node shared hardware resources.

All in all, StarPU orchestrates at runtime the asynchronous scheduling of various computational tasks of TB-RTM, including stencil kernel computations (compute-bound), data transfers between host/device main memory (memory-bound) and data transfers between CPU memory and disk storage (IO-bound). Last but not least, TB-RTM takes advantage of a new out-of-core (OOC) feature from StarPU, which supports prefetching mechanisms to further reduce the overheads of data movement not only from/to CPU and GPU main memory but also from/to CPU main memory and fast storage drives. The OOC optimization technique is instrumental in maximizing the hardware occupancy. To our knowledge, TB-RTM is the first task-based software framework to tackle the full RTM application. We deploy TB-RTM on two world-class GPU-based HPC supercomputers, i.e., Tsubame 3.0 and Summit, ranked 22nd and 1st, respectively, in the Top500 from November 2018 [4]. We demonstrate the effectiveness of the OOC feature and scale the TB-RTM framework up to 500 GPUs. We run against 3D synthetic and real velocity models and assess the numerical property of the obtained seismic subsurface image.

The remainder of this paper is organized as follows. Section II provides related work in the context of out-of-core approaches for scientific applications. Section III presents our contributions. Section IV describes the seismic imaging applications and the related challenges. Section V highlights the components of our proposed software framework to deploy large-scale task-based RTM simulations using StarPU on heterogeneous machines. Section VI details our high performance implementation. Section VII reports the performance impact of OOC techniques and demonstrates the scalability of our task-based RTM on two leading-edge supercomputers, i.e., Tsubame 3.0 and Summit, using synthetic velocity models. Section VIII summarizes the contributions of this paper and discusses future work.

II. Related Work

With the development of advanced vector instruction sets, there have been many research studies addressing the challenges of a faster computational stencil kernels on homogeneous x86 and GPU-based systems using spatial [5]–[15] or temporal blocking [16]–[26]. These stencil kernel optimizations are key components to the Reverse Time Migration (RTM), but they usually rely on simple boundary conditions (e.g., Dirichlet) and do not consider the full RTM ecosystem and specifications. For instance, the temporal blocking stencil kernel optimization from Girih [25] may encounter performance issues [27] when incorporating the Convolutional Perfectly Matched Layer (CPML) [28], which corresponds to the RTM reference in terms of absorbing boundary conditions. This highlights the importance of developing a modular RTM framework, where high performance stencil kernels can be integrated, validated and evaluated directly from the RTM perspective.

There have been few attempts to provide a modular framework for the stencil computations. Devito [29], [30] generates optimized wave propagation kernels for use in seismic imaging. However, Devito relies on the fork-join paradigm from the OpenMP programming model [31] and SIMD optimizations for parallel performance. Besides not supporting asynchronous executions, it only runs on homogeneous x86 systems. Moreover, it does not provide a prefetching mechanism to mitigate the I/O overheads. There are other RTM frameworks solving 3D problems developed in the academic field [32], [33] and in the context of industrial seismic applications ([34]–[36]), but, here again, they do not adopt a dynamic runtime system to orchestrate the scheduling of the various RTM tasks.

Out-of-core (OOC) algorithms have been extensively studied in the literature [37]–[41]. Memory is typically a scare resource, especially on GPUs, for which the memory capacity may be at least an order of magnitude lower than the main memory of a CPU. This memory capacity limitation has been further exacerbated with the advent of big data problems. There have been some software tools development [42], [43] to address OOC algorithms but none provide seamlessly support for transferring data from GPU main memory all the way to fast storage drives.

All in all, our proposed software framework implementing Task-Based RTM (TB-RTM), powered by the StarPU dynamic runtime system, highlights a systematic approach for hardware-oblivious scheduling of RTM tasks, for data traffic overhead reduction thanks to the OOC feature, for mitigating load imbalance due to shared resources, and for rapid deployment on large-scale systems equipped with multiple GPUs.

III. Contributions

The contributions of this paper are as follows:

- Deploy a novel software framework for Task-Based Reverse Time Migration (TB-RTM) that leverages the performance of large-scale GPU-based systems.
- Integrate the StarPU dynamic runtime system to asynchronously schedule the various tasks of the RTM application, while maximizing hardware occupancy.
- Leverage the out-of-core feature from StarPU to further mitigate the I/O overheads during the RTM simulation.
- Promote a systematic approach within TB-RTM to ensure software portability and to address load imbalance.
- Demonstrate performance scalability up to 500 GPUs on two large-scale GPU-based systems, i.e., Tsubame 3.0 and Summit.

- Assess the TB-RTM numerical accuracy by using a synthetic velocity model and generating the corresponding seismic subsurface image.

To our knowledge, our TB-RTM software framework is the first task-based library designed holistically driven by a dynamic runtime system (i.e., StarPU) to tackle full RTM simulations.

IV. Reverse Time Migration for Seismic Imaging

The Reverse Time Migration (RTM) solves a two-way numerical PDE based on the first or second-order formulation of the 3D wave equation. The equation is discretized using a finite-difference time-domain method. RTM relies on the Convolutional Perfectly Matched Layer (CPML) [44] to set up the absorbing boundary conditions.

The workflow of the RTM is a collection of several shots, which accumulates the local data solution from individual shots to eventually generate the final seismic subsurface image. As shown in Fig. 1, the RTM workflow starts with a velocity model of the earth as an input for each shot experiment. It illustrates the kernels of the RTM with green circles for

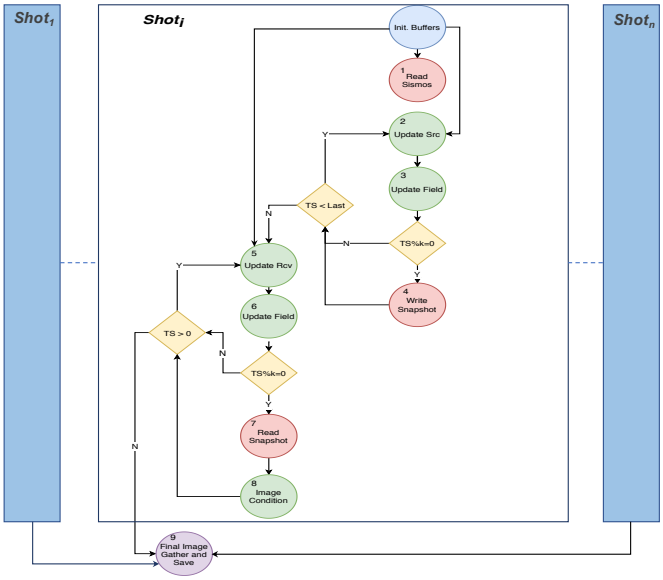


Fig. 1: Color-coded RTM flowchart for n shots: green circles represents GPU computation kernels, red (and blue) circles for I/O and memory kernels, purple circles for reduction kernels, and yellow diamonds for conditionals to trigger I/O operations.

GPU computation kernels, red (and blue) circles for I/O and memory kernels, purple circles for reduction kernel, and yellow diamonds for conditionals to trigger I/O operations. The time-stepping loop in each shot is independent of the others. The communication occurs after the backward phase when gathering and stacking the image per shot into the final image. In Fig. 1, Sismos refers to a set of data input required by each shot and obtained prior running RTM (in a seismic acquisition step). In modeling the seismic acquisition, we activate a source (S) to send artificially-generated seismic waves into the subsurface. The waves get reflected off layer boundaries called reflectors in the seismic literature. The arrival times and amplitudes of the reflected waves on the surface is recorded then detected by the receivers (R), these traces are stored for every shot independently.

Algorithm 1 Pseudo-code of the full RTM application.

```
1:  Init Buffers
2:  for Shot = 0 to Last − 1 do
3:      Read Sismos Traces
4:      for Timestep = 0 to LastTimestep FWD do
5:          Update Source Wavelet
6:          Compute Wavefield
7:          Write Snapshot if(Timestep%K = 0)
8:      end for
9:      for Timestep = LastTimestep to 0 BWD do
10:         Update Receivers Data
11:         Compute Wavefield
12:         Read FWD Snapshot if(Timestep%K = 0)
13:         Apply Image Condition if(Timestep%K = 0)
14:     end for
15: end for
16: for All Shots do
17:     Gather Images
18: end for
19: Save Final Image
```

The source wavefield, i.e., the wavefield whose origin is the seismic source, is propagated forward in time during a computational phase referred as the seismic modeling phase. The receiver wavefield, i.e., a wavefield that is incident from the receivers, is then propagated back in time, during the backward modeling phase. The stencil computational kernels, which include CPML, are the main substance of both aforementioned propagations. Finally, the imaging condition phase is incrementally calculated throughout the backward modeling phase. The latter phase requires the source wavefield, computed during the forward modeling, and the receiver wavefield, both at the same time step. Algorithm 1 provides the standard RTM pseudo-code and highlights both forward (i.e., lines 4-8) and backward (i.e., lines 9-14) computational phases of the RTM.

During the forward modeling, the source wavefield is stored every K^{th} time step. Given that the source and receiver wavefields are advanced along opposite time directions, this means that their values need to be stored at a specific time step period that does not exceed K, for some $K > 1$. The choice of the value of K is based on the Nyquist-Shannon sampling theorem [32]. During the backward propagation, the needed forward value is loaded every K^{th} time step and the imaging condition is performed at the same time step.

For example, if $K = 20$, a snapshot of the solution has to be saved in memory every 20 time steps during the forward phase and read back during the backward phase at identical time steps. Since RTM simulations run for thousands of time steps, this results in an enormous burden on memory,

which current memory technologies cannot accommodate. This makes the RTM an I/O-intensive application, in addition to being compute-intensive. Given that the compute workload may usually be executed faster than the I/O workload, the overall time to solution may be solely driven by the I/O bandwidth. For small time step period K, frequently hitting disk storage may not be an option, so a systematic approach to provide out-of-core feature in the context of RTM simulations is necessary.

V. Task-Based RTM Software Framework

Our software infrastructure of the TB-RTM framework depends on two main libraries: SIMWAVE and StarPU [3].

A. The SIMWAVE Stencil Computational Kernels

SIMWAVE is a core library that provides CPU and GPU stencil kernels to TB-RTM. It is an in-house implementation of RTM kernels, as introduced in Algorithm 2. It includes CPML absorbing boundary conditions and generated velocity models. The second-order formulation of the 3D acoustic wave equation is:

$$U^{n+1}_{i,j,k} = 2U^{n}_{i,j,k} - U^{n-1}_{i,j,k} + c^2_{i,j,k}\Delta t^2 \Delta U^{n}_{i,j,k} + c^2_{i,j,k}\Delta t^2 s^n. \quad (1)$$

The Equation 1, which is explicitly second-order in time, is discretized to eighth-order in space by replacing the $\Delta U^n_{i,j,k}$ term with a 25-point star stencil. This is a proxy discretization for seismic imaging in oil and gas industry. This high-order discretization scheme renders the stencil computational kernels compute-intensive. The algorithmic complexity is further exacerbated when considering CPML into the stencil kernels. Algorithm 2 highlights the SIMWAVE Pseudo-code for the stencil kernel. The 3D data is fetched slice by slice from GPUs global memory and stored in local memory where each CUDA worker thread sweeps along the Z dimension. The Z dimension is the fastest index since data solution is contiguously stored in memory along this dimension. While TB-RTM is currently tailored for SIMWAVE stencil kernels' APIs, integrating other stencil implementations is also possible thanks to the modularity of TB-RTM. The default value of K in SIMWAVE is given by the Nyquist limit, which is computed as $K_{N-S} = \left\lfloor ((2.0 \times Frequency_{max})/\delta t)^{-1} \right\rfloor + 1$.

Algorithm 2 SIMWAVE Pseudo-code for Stencil Computational Kernel

```
1: Input: U(N_x, N_y, N_z, t)
2: Output: U(N_x, N_y, N_z, t + 1)
3: for z = 0 to N_z do
4:   for y = 0 to N_y do
5:     for x = 0 to N_x do
6:       COMPUTE LAPLACIAN
7:       if z, x, y is inner field then
8:         UPDATE INNER FIELDS
9:       else
10:        UPDATE PML FIELDS
11:      end if
12:    end for
13:  end for
14: end for
```

B. The StarPU Dynamic Runtime System

StarPU is a dynamic runtime system of task-based scheduling on heterogeneous multicore and manycore architectures. It is perhaps one of the most mature, comprehensive dynamic task-based runtime systems. There are three central components: data, codelets and tasks. Application data is registered to StarPU so that the latter can manage its location at will between the GPU memory, the main memory, and the disk. Codelets group under the same name multiple implementations (CPU, CUDA, etc.) of the same computation function (e.g., stencil kernel). Tasks correspond then to the application of a codelet on data inputs and outputs. The runtime system is able to transparently handle the execution on multiple heterogeneous nodes in a distributed-memory environment through an MPI layer within StarPU-MPI [45]. During execution, the runtime system can automatically decide which task implementation is suited to achieve the highest performance based on cost models, which are automatically computed by StarPU when executing the application. A unique feature from StarPU is the support of Out-Of-Core (OOC) data management. The OOC feature enables StarPU to monitor the GPU/CPU memory usage at runtime. In case the defined memory capacity is reached on GPU and CPU, StarPU may start flushing in and out to CPU and disk storage the data registered in StarPU radar, respectively. StarPU exposes environment variables, which enables end-users to control and monitor internal settings and tunable parameters.

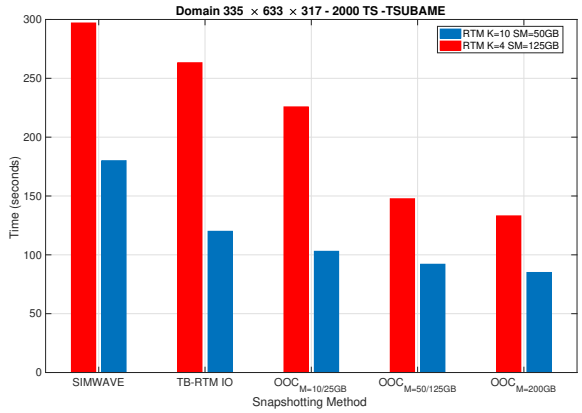
Every task in RTM (see Fig. 1) needs to be defined as a StarPU codelet, along with data directions (i.e., IN, OUT, and INOUT), which are used as a basis by StarPU to track data dependencies among the various computational tasks.

VI. Implementation Details

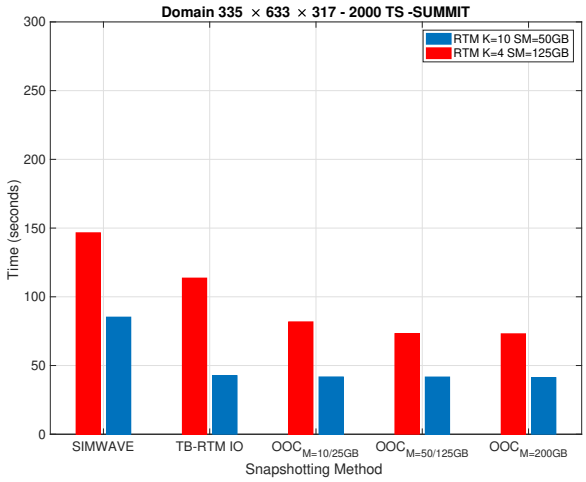
In TB-RTM, we matricize the three dimensional domains (or tensor) so that we can leverage the Chameleon library [46], designed for dense linear algebra tile algorithms, in addition to the StarPU dynamic runtime system. Computational tasks can then operate on data that is contiguous in memory to reduce cache misses.

A. Tasks and Concurrency

Indeed, TB-RTM supports two task granularities: at the level of a full domain (i.e., coarse granularity) and at the level of a subdomain within a single shot (i.e., fine granularity) in case of domain decomposition. For both granularities, we matricize the three dimensional domains (or tensor) so that we can leverage the Chameleon library [46], designed for dense linear algebra tile algorithms, in addition to the StarPU dynamic runtime system. Computational tasks can then operate on data, which is contiguous in memory to reduce cache misses. The tensor is registered in StarPU so that the runtime monitor the data location and task readiness for the prefetching mechanism



(a) Tsubame node: P100 GPU with SSDs.



(b) Tsubame node: V100 GPU with NVMe.

Fig. 2: Time to the solution of single shot RTM simulation on one node of Tsubame and Summit against different snapshotting methods supported in TB-RTM in addition to SIMWAVE as our baseline.

Algorithm 3 TB-RTM

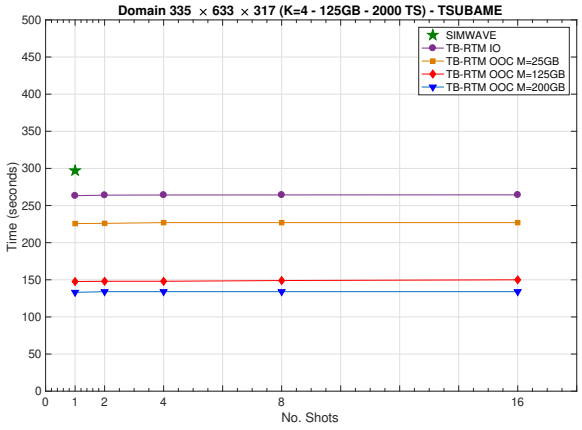
```
1:  Insert Task: Init Buffers
2:  for Shot = 0 to Last − 1 do
3:      Insert Task: Read Sismos Traces
4:      for Timestep = 0 to LastTimestep FWD do
5:          Insert Task: Update Source Wavelet(INOUT: U, IN: Src)
6:          Insert Task: Compute Wavefield(INOUT: U)
7:          if Timestep%K=0 then
8:              Insert Task: Copy(IN: U, OUT: Snap)
9:              Insert Task: Write Snapshot(IN: Snap)
10:         end if
11:     end for
12:     for Timestep = LastTimestep to 0 BWD do
13:         Insert Task: Update Receivers Data(IN: Sismos, IN-OUT: U)
14:         Insert Task: Compute Wavefield(INOUT: U)
15:         if Timestep%K=0 then
16:             Insert Task: Read FWD Snapshot(OUT: Snap)
17:             Insert Task: Copy(IN: Snap, OUT: FWD)
18:             Insert Task: Apply Image Condition(IN: U, IN: FWD, OUT: IMGSHOT)
19:         end if
20:     end for
21: end for
22: for All Shots do
23:     Insert Task: Gather Images(IN: IMGSHOT, OUT:IMG)
24: end for
25: Insert Task: Save Final Image(IN: IMG)
```

and hide the costly data movement. The difficulty resides in ensuring the data coherency between the three layers of memory/storage, i.e., GPU, CPU and disk. There are several scheduling heuristics to drive StarPU behavior at runtime and to make various compromises between data locality and task priorities. Privileging task priorities tends to execute first the tasks which will release other tasks, thus unfolding parallelism early, but this will usually come with more memory transfers for these tasks. On the other hand, privileging data locality tends to execute first the tasks for which data is already within the GPU, thus requiring no data transfer, before tasks which will release other tasks but require data transfers. In I/O-intensive situations, the latter approach becomes more and more critical, so the StarPU Scheduler heuristics $dmdar$ and lws are our choices of schedulers because they privilege that approach. More precisely, $dmdar$ sorts tasks in its queues by data availability over task priority, and lws only balances the load between GPUs when some of them are really idle. We map the computation of a shot to GPU where all the stencil computations are executed on GPU and the CPU workers manage and monitor data and I/O operations.

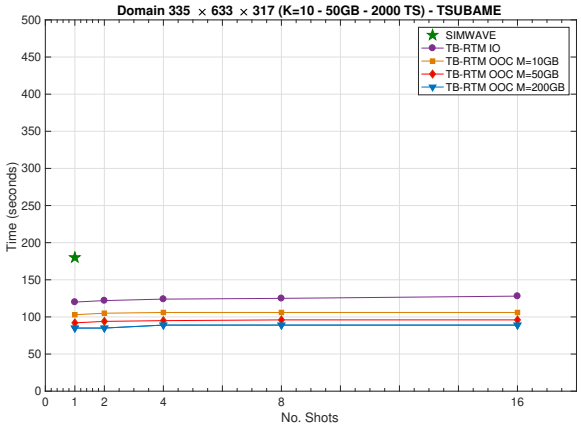
In this paper, we focus on the coarse granularity of a shot. We want to highlight two main features of TB-RTM: out-of-core and I/O overlap with the compute tasks using the StarPU runtime. Algorithm 3 shows the data flow of RTM and the additional copy kernels (line number 8 and 17). These additional copy kernels decouple the I/O from the compute tasks for asynchronous execution. The task dependency graph is rather skinny and has an odd shape where tasks are serialized within shots but embarrassingly parallel across shots. Task-based parallelism opens up a new avenue of possibilities for a systematic approach to overlap I/O with computation, and out-of-core algorithms.

B. Snapshotting

TB-RTM supports three snapshotting modes: asynchronous I/O, OOC, and in memory. The first one, TB-RTM IO,

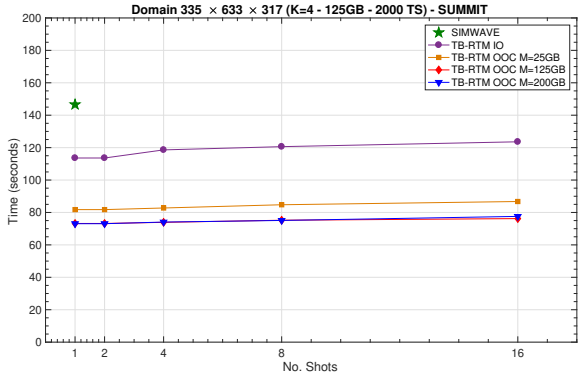


(a) K=4 Case.

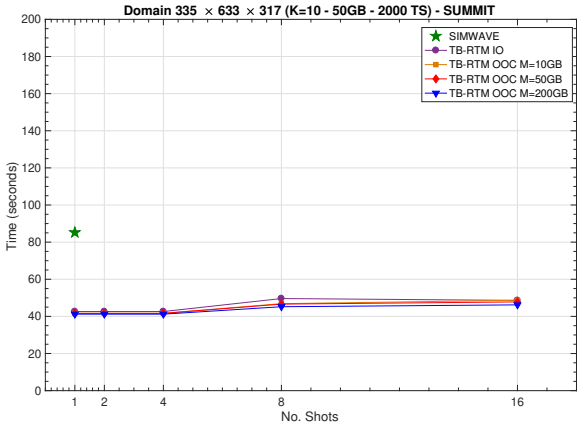


(b) K=10 Case.

Fig. 3: Weak scaling for TB-RTM simulating multiple shots for 2000 TS, on multiple nodes of Tsubame.



(a) K=4 Case.



(b) K=10 Case.

Fig. 4: Weak scaling for TB-RTM simulating multiple shots for 2000 TS, on multiple nodes of Summit.

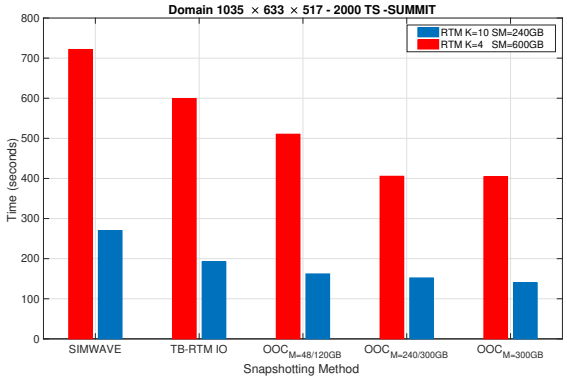
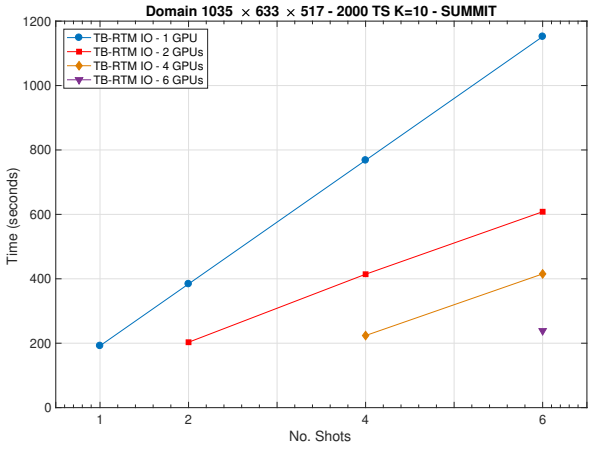


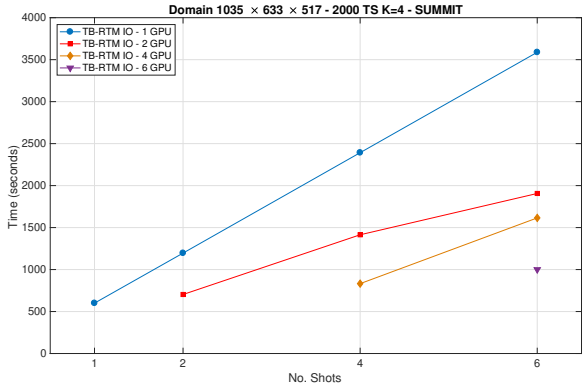
Fig. 5: Time to the solution of single shot RTM of larger size on one node of Summit (V100 GPU with NVMe) against different snapshotting methods supported in TB-RTM in addition to SIMWAVE as our baseline.

decouples the dependency between I/O tasks and compute tasks in the FWD and BWD phases and allows overlap of possible computations to avoid lack of parallelism and stalling the computation waiting for I/O. The second mode, TB-RTM OOC, the stencil kernel switches between in-core and out-of-core modes based on the scheduling heuristics of the task dependency graph. The I/O operation here is only a copy to a memory buffer which is monitored by the StarPU runtime. In OOC mode, Lines 9 and 16 in Algorithm 3 are removed, and instead StarPU will handle the data availability in the Snap buffer. The third mode is in memory; as its name implies all the snaps are alive in memory and it fails when there is no space available.

The OOC mode is controlled using a memory limit environment variable. This variable, STARPU_LIMIT_CPU_MEM, limits the memory visible to StarPU and when the application data reaches this threshold, StarPU OOC support starts evicting data to disk, and will reload it as appropriate.



(a) Case K=10.



(b) Case K=4.

Fig. 6: Weak and strong scaling for TB-RTM simulating multiple shots for 2000 TS, on single nodes of Summit.

VII. Performance Results

A. Environment Settings

1) Apparatus for TB-RTM and Test Cases: Our TB-RTM library is written in C, while the GPU stencil kernels are written in CUDA. It is compiled with CUDA 10 and StarPU 1.3 release. We test both single and multiple shots of full RTM simulation of a 3D Finite Difference Time Domain, 8th order in space, 2nd order in time, with PML damping $18\times18\times18$ for 2000 time-steps. We choose two domain sizes: $335\times633\times317$ (0.25 GB) and $1035 \times 633 \times 517$ (1.25 GB). We also include a synthetic velocity of a 3D SEG/EAGE salt model [47], [48] that includes Sismos traces for 109 shots for good quality image. The final test case is extended with the 1.25 GB model to 1000 shots.

2) Platforms:

- **Tsubame**
  Tsubame 3.0 supercomputer at GSIC Center, Tokyo Institute of Technology, Japan, a SGI ICE XA with 540 nodes each with two Xeon E5-2680v4 (14 Cores, 2.4GHz), and four NVIDIA Tesla P100 SXM2. Each compute node has 4 ports of Intel Omni-Path interface. The nodes are linked in a fat tree topology by an Omni-Path switch.
- **Summit**
  Summit, the top ranked supercomputer system in November 2018 Top500 list, an IBM supercomputer running at the Department of Energy's Oak Ridge National Laboratory, consisting of $4,356$ nodes, each one equipped with two 22-core Power9 CPUs, and six NVIDIA Tesla V100 GPUs.

B. Performance Impact of OOC

In Fig. 2a, we compare different snapshotting methods supported in TB-RTM in addition to SIMWAVE as our baseline. TB-RTM IO can be considered as the task-based SIMWAVE, where each kernel in the time stepping loop is a StarPU task. For $K = 10$, the application need, for 2000 time-steps, is a storage of 200 snaps each of size 0.25 GB, for a total application need is 50 GB.

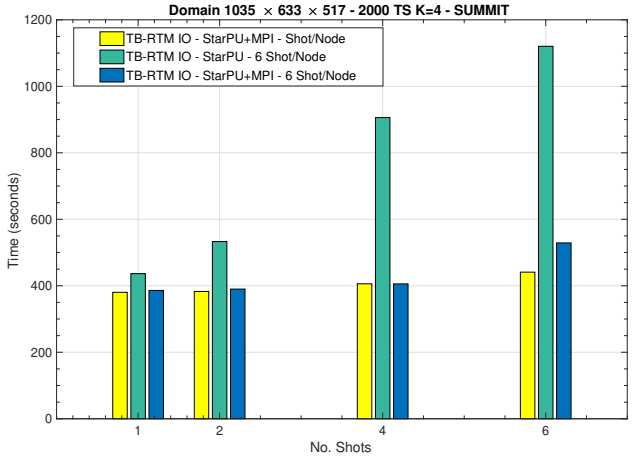


Fig. 7: Plotting StarPU +MPI and StarPU implementations of TB-RTM for simulating multiple shots RTM for K=4.

TB-RTM IO offers 50% performance gain from overlapping computation tasks with I/O tasks. TB-RTM OOC offers another level of prefetching and hiding the I/O cost. The performance gain varies from 1.8x, 2x, to 2.25x speedup based on the available memory (M) 10, 50, and 200 GB, respectively. For $K = 4$, the required storage is 125 GB. In this case, TB-RTM IO has fewer computation tasks (only 4) in order to hide the I/O penalty, therefore, the performance gain is only 15%. The performance gain varies from 1.3x, 2x, to 2.15x speedup based on the available memory (M) 25, 125, and 200 GB, respectively.

We repeat the same set of experiments on Tsubame. In Fig.

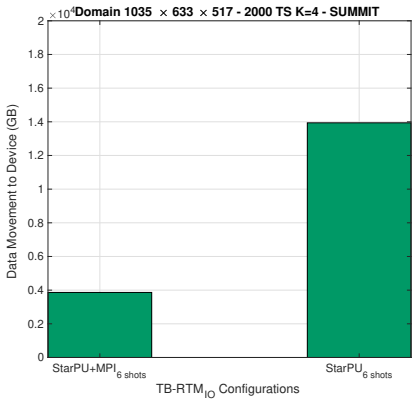


Fig. 8: Data movement from memory to device of TB-RTM IO for simulating multiple shots RTM for K=4 on one node.

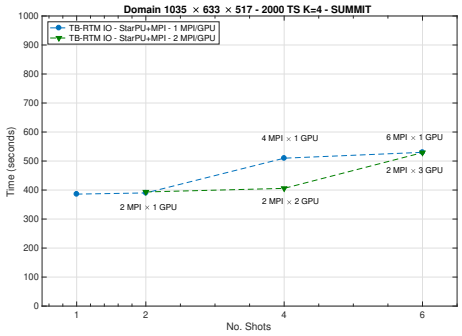


Fig. 9: Impact of pinning on time to solution TB-RTM IO for simulating multiple shots RTM for K=4 on one node.

2b, for $K = 10$, TB-RTM IO and TB-RTM OOC offers 2x speedup as with K=10 there are enough computation tasks that can hide the I/O cost. For $K = 4$, the performance gain is 1.4x, 1.7x, 2x, and 2x for TB-RTM IO, OOC M=25, 125, and 200 GB, respectively. Comparing Summit with Tsubame results, there is more than 2x speedup from Summit due to V100 GPUs and NVLINK between Power9 and GPUs. Each Summit node consists of two sockets of POWER9 each has NVLINK of 50 GB/s bandwidth connection to three V100 GPUs.

C. Weak Scalability

Figures 3 and 4 show the weak scaling of multiple shots RTM on Tsubame and Summit. As expected, TB-RTM weak scales perfectly. Dedicating each shot to a separate node is the typical way of executing multiple shots RTM simulation. However, at the node level, not all resources may be utilized. There are two ways to utilize the underlying resources: to go with domain decomposition or to oversubscribe the node with multiple shots. The first option, domain decomposition, is good for a very large domain size that exceeds one GPU memory, i.e., 16 GB. The second option can scale down/up to small and very large domain sizes. The challenge is utilizing the resources without performance degradation due to oversubscription.

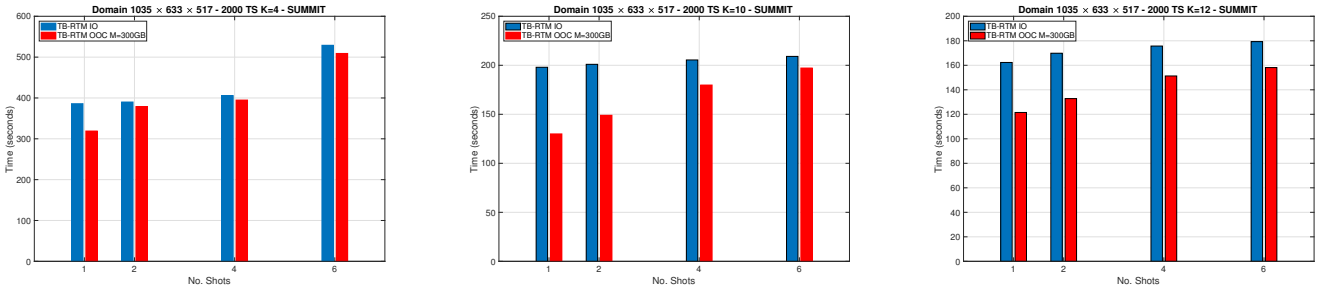
D. Performance Impact and Profiling with Larger Domain Size

Fig. 5 shows the execution time of a single shot RTM simulation applying different TB-RTM snapshotting methods and SIMWAVE of a larger domain size 1.25 GB for 2000 time-steps and two different period of snapshotting. For $K = 10$, total application need (SM) is 240 GB, the performance gain is 1.4x, 1.8x, 1.9x, and 2x for TB-RTM IO, OOC M=48, 240, and 300 GB, respectively. For $K = 4$, SM is 600 GB, the performance gain is 1.2x, 1.4x, 1.8x, 1.8x for TB-RTM IO, OOC M=120, 300 GB, and all memory, respectively.

Fig. 6a where $K = 10$, we can see that StarPU helps schedule and fuse the computation of multiple shots using the available GPUs showing both strong (vertical points) and weak (horizontal points) scalability with negligible penalty. Figure 6b where $K = 4$, the performance degradation is more pronounced as it goes to more than 50% due to resource contention.

Sharing the resources is one reason for this degradation of performance. The other reason is suboptimal scheduler decisions. We investigate the performance of using different scheduling context instead of the default *lws* StarPU scheduler. *lws* uses a Locality-aware Work-Stealing approach: whenever a worker is idle it steals a task from the neighbor workers. *dmdar*, instead, takes the task-execution and data-transfer performance models into account for a Heterogeneous Earliest Finish Time (HEFT) scheduler [49], that makes a compromise between task duration and data transfer cost. *dmdar* thus requires preliminary calibration runs for the performance models to converge. *dmdar* additionally privileges tasks whose data buffers are already available on the GPU memory. Figure 7 shows the case of $K = 4$ and compares the three different configurations of simulating multiple shots: classical Shot/Node using StarPU +MPI, the same implementation StarPU +MPI but Shot/GPU, and StarPU shared-memory with multiple GPUs.

The resource contention is more pronounced in this case of $K = 4$ and this becomes clear by comparing the StarPU +MPI using multiple nodes with the one in one node. This degradation of performance is exacerbated further with StarPU shared-memory. In the StarPU shared memory configuration, StarPU has to decide the data mapping over GPUs as shown in Fig. 8. The *dmdar* scheduling heuristic however only has limited sight of the task graph to keep its cost reasonable. This prevents it from making proper data placement taking future use into account. Consequently, the additional costs of blindly moving the data across GPUs degrade the performance dramatically in this case. A static grid of processors as in the StarPU +MPI case improves the performance. In Fig. 9, dedicating two MPI processors and varies the number of GPUs per MPI improves the performance further of StarPU +MPI for the 4 shots case where each 2



(a) Case K=4. (b) Case K=10. (c) Case K=12.

Fig. 10: Time to the solution of multiple shots RTM simulations on one node of Summit against different snapshotting methods supported in TB-RTM: IO and OOC.

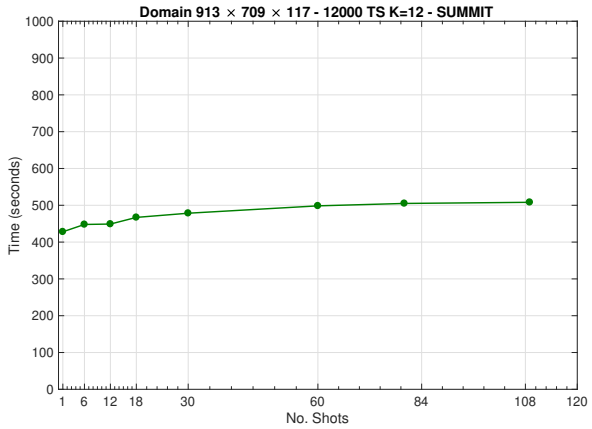


Fig. 11: Weak Scaling of Salt 3D model with K=12 using TB-RTM OOC on Summit (1 Shot/GPU - 6 GPUs/Node).

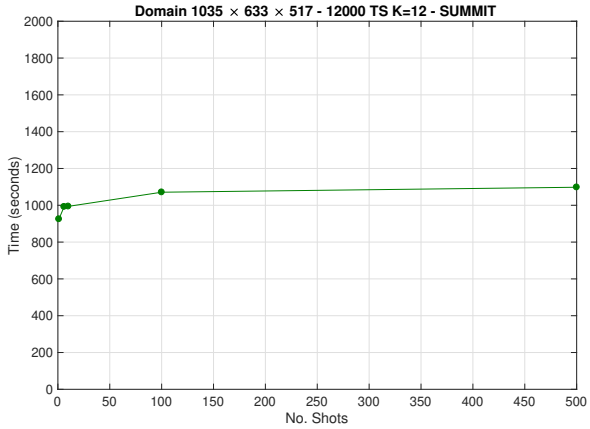


Fig. 12: Weak Scaling of a synthetic 3D model with K=12 using TB-RTM OOC on Summit (1 Shot/GPU - 6 GPUs/Node).

shots are pinned to one socket (instead of three in socket 0 and one in socket 1).

Fig. 10 compares the TR-RTM IO and TB-RTM OOC simulating 1 to 6 shots using $K = 4$, 10, and 12 on one node of Summit. As expected, thanks to the memory available for prefetching, there is speedup in computation of the 1 to 6 shots. This is important in terms of energy saving as the performance of using 6 nodes each with 1 GPU to execute 6 shots can be done in one node with 6 GPUs. TB-RTM with the shot granularity demonstrated a clear path toward a modular RTM framework unleashing the abilities and productivity of using StarPU runtime system particularly and task-based parallelism in general. TB-RTM opens up this path to take advantage of highly heterogeneous computing architecture brought by current and coming exascale systems.

E. Realistic Velocity Model Simulation

We use the synthetic velocity model 3D SEG/EAGE salt model to simulate the wave propagation in an isotropic medium with a constant density. This model was built by the SEG research committee, and created as part of the Advanced Computational Technology Initiative [48]. Fig. 11 shows the weak scalability of TB-RTM OOC with all memory available running this model: 109 shots, computational size: $913 \times 709 \times 117$, and image dimension: $841 \times 673 \times 99$. Fig. 12 shows the weak scalability with efficiency 85% a synthetic 3D model with $K = 12$ using TB-RTM OOC on Summit of the large domain up to 500 shots. Finally, Fig. 13 depicts the migrated image of the 3D SEG/EAGE salt model [48] of stacking 15 shots after modeling the propagation a 25 Hz Ricker wavelet source in an isotropic medium. We can clearly recognize the synthetic reflectors of the model.

VIII. Conclusion

We propose a novel Task-Based framework of RTM, TB-RTM, for seismic imaging that leverages the performance of multicore and GPU-based systems by relying on a dynamic runtime system, StarPU, to schedule the various tasks of the RTM. TB-RTM ultimately can serve as a unified framework implementation that is modular and easily adapts high-performance kernels. It enables asynchronous execution of RTM tasks based on the data-flow of the application. It provides a systematic approach for out-of-core execution with data prefetching. We plan in the future to add sup-

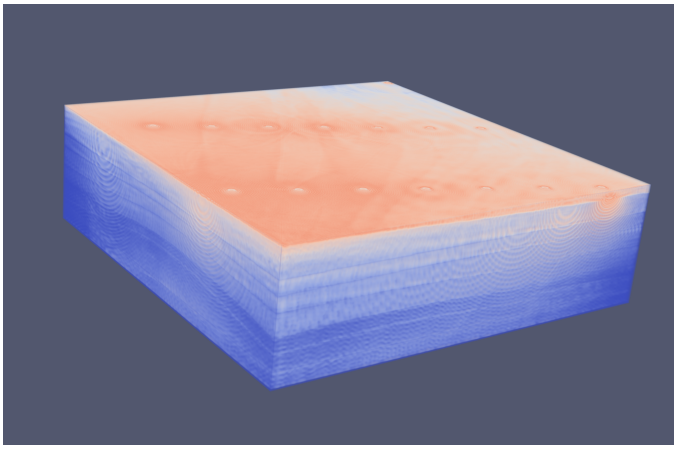


Fig. 13: The final image of stacking 15 shots of the 3D SEG/EAGE salt model.

port in StarPU OOC for two stages of storage paths for NVMe/SSDs and parallel file systems. Currently, TB-RTM supports only SIMWAVE CUDA-based kernels. We plan in the future to include Girih [25] CPU-based kernels. Also, we plan to activate domain decomposition and investigate the performance of this granularity. Our main goal is to provide TB-RTM a robust library-quality open-source task-based RTM framework.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. We are grateful to ORNL's HPC Engineer George Markomanolis and Prof. Rio Yokota of Tokyo Institute of Technology, Japan for their assistance with the runs on Summit and Tsubame 3.0, respectively. We are also grateful to Dr. Rached Abdelkhalak from the Extreme Computing Research Center, KAUST for the fruitful discussions.

References

[1] E. Baysal, D. D. Kosloff, and J. W. C. Sherwood, "Reverse time migration," *GEOPHYSICS*, vol. 48, no. 11, pp. 1514–1524, 1983. [Online]. Available: https://doi.org/10.1190/1.1441434

[2] S. Brandsberg-Dahl, *High-performance computing for seismic imaging; from shoestrings to the cloud*, 2017, pp. 5273–5277. [Online]. Available: https://library.seg.org/doi/abs/10.1190/segam2017-17795566.1

[3] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[4] The Top500 List. Available at http://www.top500.org.

[5] D. Michéa and D. Komatitsch, "Accelerating a 3D finite-difference wave propagation code using GPU graphics cards," *Geophysical Journal International*, vol. 182, no. 1, pp. 389–402, 2010. [Online]. Available: https://hal.inria.fr/inria-00528487

[6] K. Datta, "Auto-tuning stencil codes for cache-based multicore platforms," Ph.D. dissertation, EECS Department, University of California, Berkeley, Dec 2009.

[7] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-D blocking optimization for stencil computations on modern CPUs and GPUs," in *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–13.

[8] V. Etienne, T. Tonellot, T. Malas, H. Ltaief, S. Kortas, P. Thierry, and D. Keyes, "High-Performance Seismic Modeling with Finite-Difference Using Spatial and Temporal Cache Blocking," *3rd EAGE Workshop High Performance Computing for Upstream*, 2017.

[9] S. Titarenko and M. Hildyard, "Hybrid multicore/vectorisation technique applied to the elastic wave equation on a staggered grid," *Computer Physics Communications*, vol. 216, pp. 53–62, 2017.

[10] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," *ACM SIGPLAN Notices*, vol. 43, no. 6, pp. 101–113, 2008.

[11] D. Orozco, E. Garcia, and G. Gao, "Locality optimization of stencil applications using data dependency graphs," in *Languages and Compilers for Parallel Computing*. Springer Berlin Heidelberg, 2011, pp. 77–91.

[12] T. Henretty, R. Veras, F. Franchetti, L. N. Pouchet, J. Ramanujam, and P. Sadayappan, "A stencil compiler for short-vector SIMD architectures," in *27th ACM Int. Conf. on Supercomputing*. ACM, 2013, pp. 13–24.

[13] J. Holewinski, L. N. Pouchet, and P. Sadayappan, "High-performance code generation for stencil computations on GPU architectures," in *26th ACM Int. Conf. on Supercomputing*. New York, NY, USA: ACM, 2012, pp. 311–320.

[14] D. G. Wonnacott and M. M. Strout, "On the scalability of loop tiling techniques," in *3rd Int. Workshop on Polyhedral Compilation Techniques*, Berlin, 2013, pp. 3–11.

[15] T. Grosser, A. Cohen, J. Holewinski, P. Sadayappan, and S. Verdoolaege, "Hybrid hexagonal/classical tiling for GPUs," in *IEEE/ACM Int. Symposium on Code Generation and Optimization*. ACM, 2014, p. 66.

[16] D. Orozco and G. Gao, "Diamond tiling: A tiling framework for time-iterated scientific applications," CAPSL Technical Memo 091, Tech. Rep., 2009.

[17] G. Wellein, G. Hager, T. Zeiser, M. Wittmann, and H. Fehske, "Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization," in *33rd Annual IEEE Int. Computer Software and Applications Conference*, vol. 1, July 2009, pp. 579–586.

[18] R. Strzodka, M. Shaheen, D. Pajak, and H.-P. Seidel, "Cache accurate time skewing in iterative stencil computations," in *Int. Conf. on Parallel Processing*. IEEE Computer Society, Sep. 2011, pp. 571–581.

[19] D. G. Wonnacott, "Using time skewing to eliminate idle time due to memory bandwidth and network limitations," in *International Parallel and Distributed Processing Symposium*, 2000, pp. 171–180.

[20] V. Bandishti, I. Pananilath, and U. Bondhugula, "Tiling stencil computations to maximize parallelism," in *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov 2012, pp. 1–11.

[21] X. Zhou, "Tiling optimizations for stencil computations," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2013.

[22] T. Grosser, S. Verdoolaege, A. Cohen, and P. Sadayappan, "The relation between diamond tiling and hexagonal tiling," *Parallel Processing Letters*, vol. 24, no. 03, 2014.

[23] T. Malas, G. Hager, H. Ltaief, H. Stengel, G. Wellein, and D. Keyes, "Multicore Optimized Wavefront Diamond Blocking for Optimizing Stencil Updates," *SIAM Journal on Scientific Computing*, vol. 37, no. 4, pp. 439–464, 2015.

[24] S. Moustafa, W. Kirschenmann, F. Dupros, and H. Aochi, "Task-Based Programming on Emerging Parallel Architectures for Finite-Differences Seismic Numerical Kernel," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, pp. 764–777.

[25] T. M. Malas, G. Hager, H. Ltaief, and D. E. Keyes, "Multidimensional intratile parallelization for memory-starved stencil computations," *TOPC*, vol. 4, pp. 12:1–12:32, 2017.

[26] L. Yuan, Y. Zhang, P. Guo, and S. Huang, "Tessellating stencils," in *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 49:1–49:13.

[27] R. Abdelkhalak, K. Akbudak, V. Etienne, H. Ltaief, T. Tonellot, and D. E. Keyes, "Application of high performance asynchronous acoustic wave equation stencil solver into a land survey," 2019. [Online]. Available: http://hdl.handle.net/10754/631705

[28] D. Komatitsch and R. Martin, "An Unsplit Convolutional Perfectly Matched Layer Improved at Grazing Incidence for the Seismic Wave Equation," *GEOPHYSICS*, vol. 72, no. 5, pp. SM155–SM167, 2007.

[29] M. Louboutin, M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko, and G. J. Gorman, "Devito: an embedded domain-specific language for finite differences and geophysical exploration," *CoRR*, vol. abs/1808.01995, Aug 2018. [Online]. Available: https://arxiv.org/abs/1808.01995

[30] F. Luporini, M. Lange, M. Louboutin, N. Kukreja, J. Hückelheim, C. Yount, P. Witte, P. H. J. Kelly, G. J. Gorman, and F. J. Herrmann, "Architecture and performance of Devito, a system for automated stencil computation," *CoRR*, vol. abs/1807.03032, jul 2018. [Online]. Available: http://arxiv.org/abs/1807.03032

[31] *OpenMP Application Program Interface, Version 5.0*, OpenMP Architecture Review Board, 2018, https://www.openmp.org/wp-content/uploads/OpenMPRef-5.0-111802-web.pdf.

[32] W. W. Symes, "Reverse time migration with optimal checkpointing," *GEOPHYSICS*, vol. 72, no. 5, pp. SM213–SM221, 2007. [Online]. Available: https://doi.org/10.1190/1.2742686

[33] T. Okamoto, H. Takenaka, T. Nakamura, and T. Aoki, *Accelerating Large-Scale Simulation of Seismic Wave Propagation by Multi-GPUs and Three-Dimensional Domain Decomposition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 375–389.

[34] D. Imbert, K. Imadoueddine, P. Thierry, H. Chauris, and L. Borgues, "Tips and trick for Finite Difference and I/O less FWI," *SEG Annual Meeting, Expanded Abstracts*, pp. 3174–3178, 2011.

[35] A. Sena, A. Nascimento, C. Boeres, V. Rebello, and A. Bulcao, "An Approach to Optimise the Execution of RTM Algorithm in Multicore Machines," *2011 Seventh IEEE International Conference on eScience*, pp. 403–410, 2011.

[36] V. Etienne, T. Tonellot, P. Thierry, V. Berthoumieux, and C. Andreolli, "Optimization of the seismic modeling with the time-domain finite-difference method," *SEG Annual Meeting, Expanded Abstracts*, pp. 3536–3540, 2014.

[37] S. Toledo, "External memory algorithms," J. M. Abello and J. S. Vitter, Eds. Boston, MA, USA: American Mathematical Society, 1999, ch. A Survey of Out-of-core Algorithms in Numerical Linear Algebra, pp. 161–179. [Online]. Available: http://dl.acm.org/citation.cfm?id=327766.327789

[38] J. S. Vitter, "External memory algorithms and data structures: Dealing with massive data," *ACM Computing Surveys*, vol. 33, p. 2001, 2001.

[39] ——, "Algorithms and data structures for external memory," *Found. Trends Theor. Comput. Sci.*, vol. 2, no. 4, pp. 305–474, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1561/0400000014

[40] T. Joffrain, E. S. Quintana-Ortí, and R. A. van de Geijn, "Rapid development of high-performance out-of-core solvers," in *Applied Parallel Computing. State of the Art in Scientific Computing*, J. Dongarra, K. Madsen, and J. Waśniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 413–422.

[41] K. Kabir, A. Haidar, S. Tomov, A. Bouteiller, and J. Dongarra, "A framework for out of memory svd algorithms," in *High Performance Computing*, J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, Eds. Cham: Springer International Publishing, 2017, pp. 158–178.

[42] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout, "PIDX: Efficient parallel I/O for multi-resolution multi-dimensional scientific datasets," in *2011 IEEE International Conference on Cluster Computing*, Sep. 2011, pp. 103–111.

[43] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson, "The TileDB Array Data Storage Manager," *Proc. VLDB Endow.*, vol. 10, no. 4, pp. 349–360, Nov. 2016. [Online]. Available: https://doi.org/10.14778/3025111.3025117

[44] R. Kosloff and D. Kosloff, "Absorbing Boundaries for Wave Propagation Problems," *J. Comput. Phys.*, vol. 63, no. 2, pp. 363–376, Apr. 1986. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(86)90199-3

[45] E. Agullo, O. Aumage, M. Faverge, N. Furmento, F. Pruvost, M. Sergent, and S. P. Thibault, "Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[46] E. Agullo, O. Aumage, M. Faverge, N. Furmento, F. Pruvost, M. Sergent, and S. P. Thibault, "Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, Dec 2018.

[47] Sandia Natonal Laboratories,SEG/EAGE Salt C3 Dataset. Available at http://www.cs.sandia.gov/c̆cober/seismic/salt_c3.html.

[48] 3D SEG/EAGE Salt Model. Available at https://wiki.seg.org/wiki/SEG/EAGE_3D_modeling_Salt_Model_Phase-C_1996.

[49] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *Proceedings of the Eighth Heterogeneous Computing Workshop*, ser. HCW '99. Washington, DC, USA: IEEE Computer Society, 1999, p. 3.

NOTES