Optimizing Complex Spatially-Variant Coefficient Stencils For Seismic Modeling on GPU

Jiarui Fang*†§, Haohuan Fu*
He Zhang*†, Wei Wu‡, Nanxun Dai‡, Lin Gan*†, Guangwen Yang*†
*Ministry of Education Key Lab. for Earth System Modeling, Center for Earth System Science, Tsinghua University
†Department of Computer Science & Technology, Tsinghua University
‡BGP R&D Center, China National Petroleum Corporation
§Tsinghua National Laboratory for Information Science and Technology (TNList)

***Abstract*—The Explicit Time Evolution (ETE) method is an innovative Finite-Difference (FD) type method to simulate the wave propagation in acoustic media with higher spatial and temporal accuracy. However, different from FD, it is difficult to achieve an efficient GPU design because of the poor memory access patterns caused by the off-axis points and spatially-variant coefficients. In this paper, we present a set of new optimization strategies for ETE stencils according to the memory hierarchy of NVIDIA GPU. To handle the problem caused by the complexity of the stencil shapes, we design a *one-to-multi* updating scheme for shared memory usage. To alleviate the performance damage resulted from the poor memory access pattern of reading spatially-variant coefficients, we propose a stencil decomposition method to reduce un-coalesced global memory access. Based on the state-of-the-art GPU architecture, combining with existing spatial and temporal stencil blocking schemes, we manage to achieve 9.6x and 9.9x speedups compared with a well-tuned 12-core CPUs version for 37-point and 73-point ETE stencils, respectively. Compared with a well-tuned MIC version, the best speedups for the 2 type stencils are 3.7x and 4.7x. Our designs leads to an ETE method that is 31.2x faster than conventional CPU-FD method and make it a practical seismic imaging technology.**

***Index Terms*—GPU, MIC, spatially-variant coefficient stencil, Explicit Time Evolution, forward modeling, seismic imaging, RTM, FWI**

I. Introduction

Seismic forward modeling is the most computationally expensive part of high quality seismic imaging methods for hydrocarbon exploration, such as reverse-time migration (RTM)[16] and full waveform inversion (FWI)[17]. Due to the extremely high computational cost of forward modeling, not till recently have the RTM and FWI been widely adopted in oil exploration industry, despite of its high accuracy in imaging of subsurface areas. Recently, the fast development of High Performance Computing (HPC) technologies has brought some new architectures that are able to provide such huge computing power. For example, the Graphic Processing Unit (GPU) designed by NVIDIA, and the Xeon Phi based on the Intel Many Integrated Core (MIC) architecture are two popular HPC platforms that have been widely applied in many key applications [1],[2]. The advances of these new architectures result in renewed attention from the seismic community to the high-performance forward modeling methods.

On the one hand, improvement of computing power from these new HPC architectures has led geologists to design more complex forward modeling methods to improve the accuracy. The Explicit Time Evolution (ETE) method, proposed by [10] in recent years, is the one of the most promising methods. By adopting a FD-style stencil in the discrete spatial-time domain to explicitly extrapolate wavefields in time, it can achieve high spatial and temporal accuracy in acoustic media without using Fourier transforms. Numerical tests indicated that ETE can achieve similar waveform accuracy as FD with five times larger discrete time steps by involving more off-axis points and adopting spatially-variant coefficients. The implementations of ETE on multi-core architectures have already demonstrated higher efficiency than the conventional Finite-Difference (FD) method [3], which is one of the most widely used methods in industry. In a real world application, every node in a computer cluster is working in parallel to image a small part of 3D work domain, which is an embarrassing parallel workload. As a result, it becomes attractive to exploit the computing power on a single node by accelerating the ETE method through the new accelerators like GPU and MIC.

On the other hand, there exist tough challenges for computer scientists to design highly efficient implementations of these complex forward modeling methods on the new architectures. Although a lot of existing efforts have already performed thorough optimization studies on FD on GPU [4],[6],[5],[7], efficient parallel solutions to the FD-like ETE method on GPU are still less to be seen. Our experiments demonstrate that GPU implementation with conventional optimization techniques merely achieves 60% performance of a CPU version with 12 cores. It is mainly due to the following two reasons: First, the coefficients of ETE stencils are spatially-variant while the coefficients of FD stencils are constant. The variant coefficients can shift the dominant workspace from the grid points being updated to the corresponding coefficients, which leads to poor locality for efficient cache usage. In addition, the memory access to the variant coefficients are largely un-coalesced, which brings dramatic efficiency damage to vectorized memory access operations on GPU. Second, ETE stencils involving off-axis points are more complicated in shape than FD stencils. It requires a careful design to increase the data reuse in the GPU on-chip fast buffer. Similar challenges also exist in a variety of FD-like forward modeling methods, such as [9], [12], [13].

To bridge the gap between the huge computing power provided by these new HPC platforms and poor memory access patterns resulted from the new forward modeling method, in this paper, we design a highly-efficient ETE method on GPU according to its memory hierarchy. Using our proposed GPU-based ETE design, the practical RTM forward modeling processing time on a single node can be reduced from 1.38 hours one shot to 3 minutes compared with CPU-FD implementation used in practical applications, making ETE a favorable design in many cases.

Our main contributions are:

- As far as we know, this is the first work that manages to accelerate the ETE method kernels through the state-of-the-art GPU platforms. Our work demonstrates promising potential to finally integrate the proposed design into the real-world seismic imaging applications.
- We propose a set of novel optimization methods on GPU for stencils with spatially-variant coefficients and complex shapes involving off-axis points, which have not been comprehensively investigated before. Based on GPU, combined with existing 2.5D spatial blocking and 1D temporal blocking strategies, we propose a *one-to-multi* updating scheme to use shared memory for stencils involving off-axis points. In addition, we design stencil decomposition schemes for stencils with spatially-variant coefficients. We gain approximately 10x speedup compared with a CPU version with 12 cores.
- We also propose some insights on GPU and MIC architectures for optimizations of spatially-variant coefficients stencils, which can provide a guidance for a board of FD-like forward modeling methods.

This paper is organized as follows. In Sec. II, we review the ETE method and describe the computational challenges to implement it on the GPU platform. In Sec. III, we introduce our GPU-based method. I/O optimizations between disk and GPU are investigated in Sec. IV. We provide the corresponding experimental results about performance and correctness in Sec. V and conclude our work in Sec. VI.

II. Backgrounds

A. *The Explicit Time Evolution method*

The analytical solution of acoustic wave equation in the wavenumber domain assuming a constant velocity is:

$$\hat{p}\left(\mathbf{k}, t+\Delta t\right) = 2\cos\left(v\Delta t\left|k\right|\right)\hat{p}(k,t) - \hat{p}\left(\mathbf{k}, t-\Delta t\right) \quad (1)$$

Where $\mathbf{k} = (k_x, k_y, k_z)$ is the wavenumber vector. For variable velocities, we consider velocity v as a ”local constant” and convert equation (1) to the time-space domain:

$$p\left(\mathbf{x}, t+\Delta t\right) = 2c\left(\mathbf{x}|v\Delta t\right) * p\left(\mathbf{x}, t\right) - p\left(\mathbf{x}, t-\Delta t\right) \quad (2)$$

Where $c\left(\mathbf{x}|v\Delta t\right)$ is a spatially variable 2D or 3D filter corresponding to $\cos(v\Delta t|k|)$ in the wavenumber domain and * denotes a spatial convolution. An ETE operator for wavefield simulation based on equation (2) is formulated in a discrete space. For a selected stencil in a neighborhood of any given

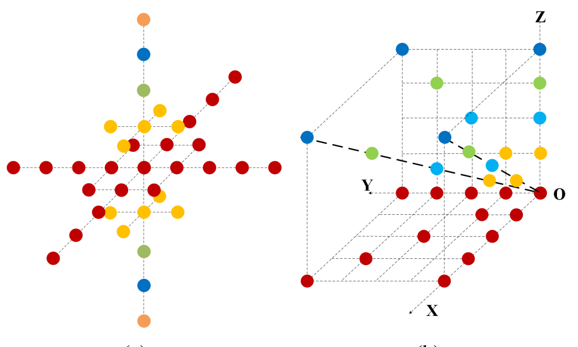

Fig. 1. Shapes of 2 type ETE stencils. (a) is the 1st type 8th order ETE stencil involving 37 points in total. (b) is one eight of the 2nd type ETE stencil, which is symmetric about XOY, XOZ and YOZ plane. It involves 73 points in total.

point in the space domain, we seek coefficients that minimize the difference between their discrete Fourier Transform and $\cos(v\Delta t|k|)$ in the wavenumber domain. The accuracy of the time evolution of the wavefield will be solely determined by the fitting of the cosine function at each wavenumber. We seek c_j, the coefficients of each stencil point, in least square sense:

$$MIN\left|\sum_j c_j e^{i\Delta\mathbf{x}_j\Delta\mathbf{k}} - \cos\left(v\left(\mathbf{x}\right)\Delta t\left|k\right|\right)\right|^2 \quad (3)$$

where $\Delta\mathbf{x}_j = \mathbf{x}_j - \mathbf{x}$ is the distance vector from updating point to a stencil point. Once the coefficients c_j are determined, the ETE operator applied to wavefield simulation is similar to FD schemes, as described in the following formula:

$$p\left(\mathbf{x}, t+\Delta t\right) = 2\sum_{j=1}^{N_e} c_j\left(\mathbf{x}\right)p(x+\Delta x_j, t) - p\left(\mathbf{x}, t-\Delta t\right) \quad (4)$$

As for the shapes of ETE stencils, it has been proven that a 8th-order stencil including off-axial stencil points appears to be optimal to reduce the misfit of cosine function, which can reach a magnitude of 10^{-6}. Numerical experiments from [11] indicate that in order to achieve similar accuracy the FD scheme requires time steps about five times as fine as ETE. Therefore, ETE operator is superior to the conventional FD scheme in terms of accuracy. Figure 1 shows two types of ETE stencils, the 1st type is used in isotropic media, while the 2nd type is commonly used in tilted transversely isotropic media. The stencils are traversed through the entire 3D grid multiple times to update each grid point with calculations involving its nearest neighbors. The updating rule for one time sweep is indicated in equation (5).

37 and 73 coefficients are required for the 1st and the 2nd type ETE stencil operations. However, in forward modeling, the 3D gird is regular, which results in symmetric distribution of the stencil’s coefficients. For a regular grid, as is shown in Fig. 1(b) the coefficients are distributed symmetrically to XOY, YOZ and XOZ planes. Thus 11 and 17 different coefficients are required to perform the 1st and the 2nd type of ETE stencil, respectively.

B. PARALLEL COMPUTING ARCHITECTURES

We explore the performance of our parallel ETE methods on 3 different parallel platforms: (1) a node with two 6-core Intel Xeon E5-2697 v2 (Intel Sandy Bridge architecture) CPUs. (2) Intel MIC 5110P (Knight Corner architecture) and MIC 7120P (Knight Corner architecture) coprocessors. (3) NVIDIA Tesla M2090c (Fermi architecture), Tesla K20c (Kepler architecture) and Titan X (Maxwell architecture) GPUs. The details of these platforms are illustrated in Table I.

C. COMPUTATIONAL CHALLENGES IN ETE METHOD

There are mainly three challenges to implement ETE stencil operations on GPU.

The first challenge is in the design of data reuse of grid points in stencil kernels. Computing 3D ETE stencils involves additions on a number of adjacent off-axis points, which do not exist in FD stencils. These points can not be accommodated into one slice of shared memory and lead to a huge registers usage (may up to 168 registers) and additional global memory access.

The second challenge comes from the complex memory access pattern of variant coefficients. Different from FD stencils, the coefficients of ETE stencil points are not constant values. They are calculated before spatial traverses of ETE stencils using the least-square method. A 2D coefficient matrix and a 3D coefficient index array have to be stored in the memory before the stencil sweeps. For the coefficient matrix (c in equation (5)), the number of rows is the number of spatial points involved in the stencil, and the number of columns is the number of different discretized velocities that we use to calculate the ETE coefficients in the model. The size of the index array is the same as the grid size. As shown in Fig. 2, when conducting the stencil operation, we access the coefficient index array according to its coordinate to acquire the index and then access a group of coefficients according to the index. The Flop to Byte Ratio of ETE's CPU implementations are 0.48 and 0.94 for the 37-point and 73-point stencils, which are far behind the theoretical Flop to Byte Ratio of all architectures in Table I. Our experiment demonstrates that the performance of a 73-point stencil implementation with conventional methods on K20c GPU only achieves 60% performance of a 12-core CPUs version.

The third challenge comes from communication between coprocessors and host disk. After a few time steps, a snapshot of 3D grid is required to be written into host disk for imaging through the PCI-E bus. The bandwidth of PCI-E is merely 6 GB/s,which will become a bottle-neck for ETE's execution on coprocessors.

III. GPU-BASED ETE STENCIL

In this section we describe our parallelization strategies for the 3D 8th order ETE stencils computation to fully utilize the memory hierarchy of GPU. A smart usage of the fast memory to increase the data reuse in stencils has been proven to be the key for memory bound stencil operation. Shared memory and registers are two kinds of fast memory on GPU. Shared

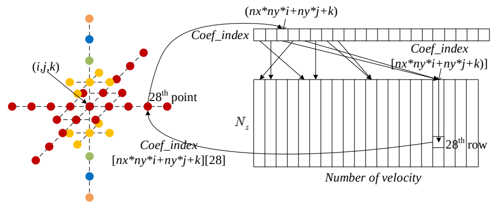

Fig. 2. Memory access pattern of spatially variable coefficients in a ETE stencil

memory can be accessed by all threads in a thread block, while registers are private resource of a thread. However, since on-chip fast memory available per thread block is not sufficient to store a large 3D subdomain of the problem, blocking techniques are proposed to maintain a chunk of data in fast buffer in turn. For our design, we adopt a 2.5D blocking technique [8],[19], which has been proven to be efficient for FD stencils. The grid points in one xy plane are computed in parallel by threads in a GPU thread block, whereas the computation over the z-direction is swept sequentially. Based on 2.5D blocking scheme, we develop our customized optimizations for stencils with spatially-variant coefficients and a large number of off-axis points. First, we propose a *1-to-multi* grid points updating scheme to reuse the grid points through fast memory on GPU in 3D domain. Then, we design stencil decomposition schemes to issue the variant coefficients problem. Finally, we reduce global memory accesses by a 1D temporal blocking based on 2.5 spatial blocking.

A. *one-to-multi updating scheme*

To resolve the problem brought by off-axis points, a fast memory utilization method different from traditional stencil implementations is introduced in this subsection. We first introduce two traditional shared memory usage methods:

(1) [14] suggested to store 9 xy-planes in shared memory for stencil in the same shape of ETE but with constant coefficients. However, shared memory is a significantly precious resource for variant coefficient ETE stencils, as mentioned in next subsection.

(2) Another popular method suggested by [4] is that when the stencil is traversing through z axis according to the 2.5D blocking scheme, stencil points of the current xy-plane are stored in one slice of shared memory and the rest points of stencil are stored in registers. As shown in Fig. 3, we need to store 12 points in registers for 37-point stencil. At each step, 5 points are required to be loaded into the registers of each thread from global memory. Five global memory loadings are required for calculation of one point. Afterwards, The grid points of next time step at current xy-plane are updated by scaling and summing up points from upper and lower 4 slices stored in registers of each thread in addition to the points stored in the shared memory. However, applying their design to the 73-point ETE stencil, 168 registers (9 register queues of length 8, 8 register queues of length 6, 8 register queues of length 4 and 8 register queues of length 2) are required

TABLE I
THE PERFORMANCE AND BANDWIDTH OF THE EVALUATED PLATFORMS.

Architectures	Clock	Peak Performance TFlops		Cache Size per core (KB)		Memory bandwidth(GB/s)		ARCH-FBR[‡]	
	GHz	float	double	L1	L2	theoretical	measured	single	double
E5-2697 v2[*] Ivy bridge	2.7	0.512	0.256	32	256	119	114	4.49	2.24
MIC 5110P Knights Corner	1.053	2.002	1.011	32	512	320	130	15.4	7.7
MIC 7120P Knights Corner	1.238	2.416	1.208	32	512	352	150	16.11	8.05
Tesla 2090c Fermi	1.3	1.331	0.665	64[†]	768	177	120	11.0	5.54
Tesla K20c Kepler	0.732	3.52	1.17	64[†]	1536	208	144	24.4	8.13
Titan X Maxwell	1.0	6.14	0.192	64[†]	3072	336	241	25.4	0.80

[*] E5-2697 v2 has a 30M L3 cache shared among 12 cores;
[†] 64KB can be configured as 48KB shared memory + 16KB L1 or 16KB shared memory + 48KB L1
[‡] is the ratio between peak computation performance and the measured memory bandwidth of the architecture.

for each thread. 33 times global memory access are required to load points from global memory into registers at each step. We call this stencil calculation method a *multi-to-one* updating scheme.

The *multi-to-one* scheme suffers from 2 performance concerns. 1. Additional global memory access leads to high Byte/Flops ratio and reduces the performance. 2. Register spilling caused by overusing of registers can slow down the computation. To reduce the register usage and global memory access times, we introduce a *one-to-multi* scheme which updates 9 xy-planes with current xy-plane in shared memory. Updating a register indicates accumulating values of grid points in shared memory to current values of the registers after scaling with corresponding coefficients. In our ETE stencil kernels, only one register queue of length 9 is allocated for each thread and one time global memory access is needed for the calculation of one point. When the points of xy-plane Z are loaded into shared memory, we read points from shared memory to update every register in the register queue. The rule for updating is illustrated in Fig. 3. After 9 steps, register 0 is accumulated with all points in the stencil and contains the value of grid point at next time step. In 2.5D scheme, register 0 is written to global memory and reset to zero. Then the register queue is circularly shifted for the next xy-slice updating.

In the *multi-to-one* scheme, grid points are stored in registers, while in *one-to-multi* scheme intermediate results are stored in registers. The number of intermediate results is the same as the order of stencil (L) for any stencil in complicated shape with off-axis points. As a result, only one register queue with length L is required for each thread. Our *one-to-multi* scheme reduces nearly 95.3%, 40% register usage and 4 and 32 times global memory access for the 2nd and the 1st type ETE stencil, respectively.

B. Stencil decomposition schemes

Different from FD stencils, the coefficients of ETE stencils are not constant values. The coefficients of different velocities are calculated using the least-square method (equation 2)

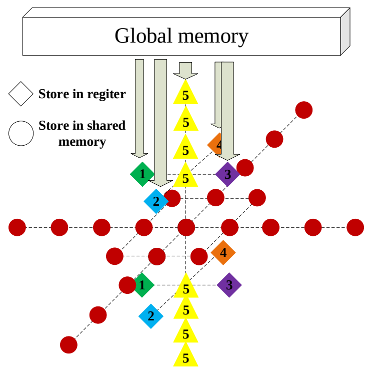

Fig. 3. *multi-to-one* scheme for the 1st ETE stencil in one CUDA thread

before spatial traversal of ETE stencils. As mentioned in the previous section, a 2D coefficient matrix and a coefficient index array have to be stored in the GPU global memory before the sweeps. We first access the index array to know the required column number of coefficient matrix and then access the corresponding column of coefficient matrix to get a group of coefficients. The real geological structures usually come with sharp velocity discontinuities. Such velocity distribution leads to discontinuous access to the coefficient matrix, which results in un-coalesced memory access of neighboring threads on GPU (described in Fig. 2). GPU devices provide a very high off-chip memory bandwidth (up to 336 GB/sec on Titan X), which is only achievable with coalesced access (data from the global memory is transferred to the GPU device in contiguous blocks) and high bandwidth can only be achieved when requests by concurrent 32 adjacent threads fall within such continuous blocks. When non-continuous memory locations are accessed by threads, the achieved bandwidth is much lower than the peak, leading to stalling and wasted compute cycles.

To issue this problem, we plan to load the coefficient matrix into shared memory initially and then access the shared memory for the coefficients. The memory access for initial shared memory loading is coalesced. However, even if taking the symmetry of coefficients into consideration, the coefficient

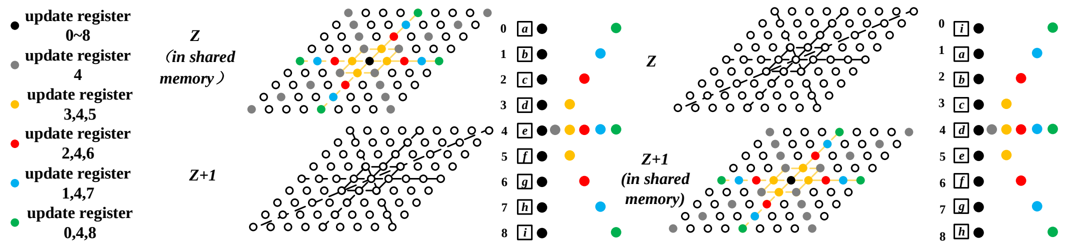

Fig. 4. *one-to-multi* scheme for the 2nd ETE stencil in one CUDA thread. Diamonds represent the register queue. The letters in the diamonds identify the registers. The colors of the grid point indicates which registers in the queue to be updated. The left picture indicates the register updating rule when the xy-slice Z reside in shared memory. The right picture indicates the updating rule for xy-slice Z+1. The register queue is shifted after updating of xy-slice Z.

matrix cannot always be accommodated into shared memory on GPU. A large number of coefficients also have to be loaded from global memory with un-coalesced access pattern. In order to completely eliminate the un-coalesced global memory access, we design decomposition schemes for ETE stencils. A ETE stencil is reconfigured into two sub-stencils. The rows of the coefficient matrix are broken down by the sub-stencils. Each of sub-stencils can perform independent scaling and sum calculation to update the grid points with corresponding rows of matrix coefficients. By using such a decomposition scheme, we can then fit the coefficients of one sub-stencil into the shared memory to execute one time sweep and reload the coefficients of the other sub-stencil into shared memory for another sweep. For Kepler and Maxwell architectures, the hardware provides a 48KB read-only cache, which can to some extent alleviate the damage from un-coalesced memory access with a larger cacheline. If the overhead cost of one time sweep is less than the benefits gained from replacing read-only cache with shared memory, we can fit the coefficients of the other component into 48KB read-only cache. Under this situation, we can execute two sub-stencil simultaneously in one time sweep. The decomposition schemes for two types of ETE stencils are illustrated in Fig. 5.

C. 3.5D temporal blocking

Since sweeps of 3D grid with ETE stencil will be executed for multiple time steps, we can further combine the 2.5D spatial blocking scheme with an additional 1D temporal blocking scheme by executing 2 time steps of blocked data together so that intermediate data can reside in shared memory and registers. A 3.5D scheme based on *one-to-multi* scheme can be applied to ETE method with no stencil decomposition optimizations. Details of our 3.5D implementation are illustrated in Fig. 6. We rearrange the calculation order of xy-planes so that the 3.5D ETE-stencil can work in pipeline. After the initial filling of the pipeline, at each calculation step a xy-plane is loaded into fast memory from the grid of time step t and a xy-plane is written to the grid of time step $t + 2\Delta t$. The grid points of time step $t + \Delta t$ are stored in registers and shared memory on GPU. The xy-planes of 3D grid are written to global memory every 2 time steps.

This scheme saves all global memory transfers to and from gird at time step $t+\Delta t$, leaving one load and one store eviction

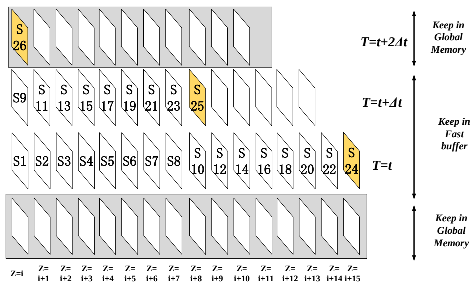

Fig. 6. Temporal 3.5D ETE stencil with *one-to-multi* updating scheme. S# indicates the order of execution. The values of grid points of time step $t + \Delta t$ are written to shared memory, which is used to updating another register queue storing intermediate results and output points of time step $t + 2\Delta t$ to global memory.

for two point updates. In addition, because we load coefficients into shared memory initially, the coefficient matrix loaded at time step t can be reused by 2 time steps afterwards. However, one drawback of the scheme is that even though we load planes of 40 $\times$ 40 elements, we can write back only planes of 32 $\times$ 32 elements because we cannot update the border elements due to missing neighbor elements. In addition, more registers are allocated in the 3.5D ETE kernel, which may brings registers spilling to damage efficiency.

IV. Writing snapshots to disk from coprocessors

To simulate the wave propagation on GPU, we need to output the snapshots of 3D domain timely into disk at set intervals. According to GPU architectures, data is first written to host memory through PCI-E bus and then written from host memory to disk by CPU. The bandwidth of the PCI-E interface is merely 6GB/s, which is not enough for conveying the data exchanges between the host CPU and the GPU cards. To hide the communication time, we overlap the communication and computation by using the streams and asynchronous GPU-host data transfer function. On NVIDIA GPU, Stream is a sequence of operations that execute in issue-order on the GPU. In Fig. 7(a) two streams are launched: one is in charge of ETE stencil computation, the other is in charge of data transfers. As shown in Fig. 7(b), with 4 or 5 discrete time intervals, we can largely hide the communication with calculation.

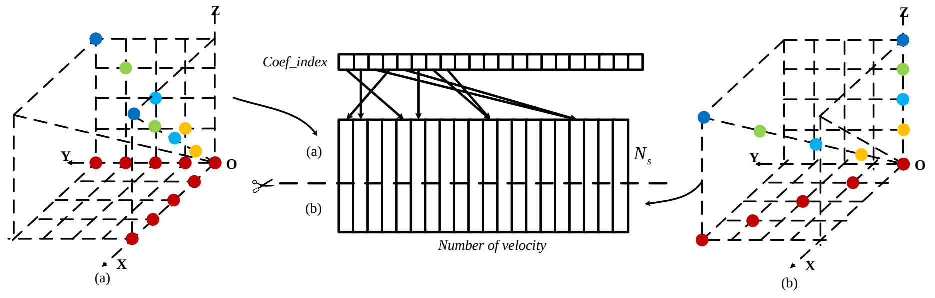

Fig. 5. Stencil decomposition for one eight of the 2nd type ETE stencil (The shapes can be arbitrary). The coefficients table is cut into 2 subtables, each of which can be accommodated into shared memory or read-only cache.

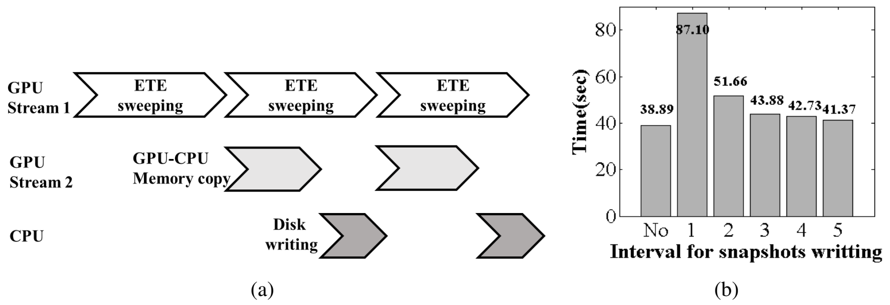

Fig. 7. Overlapping communication and computation. (a) The workflow of multiple streams on the GPU. (b) is the time performance of 1000 times wave propagations with different disk writing intervals.

V. Performance Discussions

We implement ETE methods on 5 parallel architectures mentioned in Sec. II-B. In order to make a fair comparison, we optimize the implementations of ETE stencils on multi-core CPUs and MIC.

A. MIC-based ETE stencil

Although optimization techniques on multi-core CPU such as multi-threading with OpenMP, vectorization with aligned load instructions, loop unrolling, etc. are also useful for MIC, simple porting parallel code from CPU to MIC seldom achieves satisfying performance. It is mainly because of the difference of memory hierarchy between MIC and multi-core CPUs.

On one hand, for MIC platforms, as we have a larger number (60) of physical cores (4 hardware threads per core), and the instructions from the same thread can not be executed in two consecutive cycles, there would be more frequent context switches than CPUs. When we combine the L2 caches in different cores to a large global cache, the memory hierarchy would demonstrate a strong NUMA effect. Due to the NUMA effect, specifying a suitable affinity configuration for the threads and the deciding the number of active threads are important for achieving good performance.

On the other hand, because there is no large L3 cache that can be shared by different cores on the MIC architecture, we are prompted to consider an efficient blocking plan. Different from 2.5D or 3D blocking, a heuristic blocking scheme is adopted for ETE stencil operation on MIC. [15] proposed a general blocking scheme to reduce the cache capacity misses for 3D stencil code. Assuming threaded blocking is of size $NBlock_1 \times NBlock_2 \times NBlock_3$, it is suggested rectangular blocks of shape $NBlock_1$ = $(N-2)$, $NBlock_2$=s , $NBlock_3$ = (s $\times$ $L/2B$) is best for an $N{\times}N{\times}N$ 3D domain, where L is the cacheline size, B is the size of data type of gird points, and s is the blocking factor. On MIC, we can expand the idea to make an initial estimate for the blocking factor and then adjust the size of $NBlock_2$ and $NBlock_3$ to search for the blocking size with best performance. We design the the initial guess of blocking size as $N_1{\times}s{\times}s\ {\times}L/2B$. This also implies no blocking in the slowest direction, and leads to long continuous memory read streams on this particular choice. We can determine the value of s by following equation:

$$N_1 \times s \times s \times L/2 <= C \tag{5}$$

where L is the size of cacheline which is 64B in MIC. C is the size of cache which is 512KB in MIC. The size of 3D domain is $N_1 \times N_2 \times N_3$.

We choose the best blocking size around the theoretical best blocking size to achieve the best performance.

B. Performance Metrics of different architectures

The performance is evaluated as number of floating-point operations per second (Flops), which can be measured by PAPI [18]. For our 3D problem model, the dimension size is set to be 320×320×912. Our test velocity model is a complete random distribution of 902 different velocity values.

Fig. 8 demonstrates the recorded performance for the 2 types of ETE stencil kernels on 5 different platforms. For the CPU and MIC platforms, we show the base performance (with only OpenMP and basic optimization techniques), and the optimized performance after we apply SIMD vectorization and blocking. For the GPU platforms, we show the performance of accessing coefficients directly from global memory, from read-only cache and from shared memory. As with the 37-point ETE stencil we fit the entire coefficient matrix into shared memory in this test case, so we can also adopt a 3.5D blocking scheme with 1D temporal blocking. In addition, we compare the performance of *one-to-multi* and *multi-to-one* updating scheme. For the 73-point stencil, we implement two kinds of stencil decomposition schemes. *Shared_Mem 2kernels* indicates we execute two spatial sweeps and store coefficients of two components in shared memory in turn. For Kepler and Maxwell architectures, *Shared_Mem & Read-only cache 2kernels* means we decompose the stencil into two sub-stencils and store the corresponding coefficient sub-matrices into shared memory and read-only cache separately. For the 73-point stencil, we implement all optimization techniques with the *one-to-multi* updating scheme.

The results are: for the 37-point ETE stencil we obtain up to 9.6x and 9.9x speedup while for the 73-point ETE stencil we obtain 3.7x and 4.7x speedup compared with well-tuned 12-core CPUs version on MICs and GPUs, respectively.

C. Performance Analysis

1) Analysis of our optimization strategies: We achieve more than 5 times performance boost on GPU by applying our optimization strategies. For the 37-point ETE stencil, our *one-to-multi* scheme brings 10% performance boost as the global memory access is reduced by three times. As the *multi-to-one* scheme will definitely slow down the stencil operation with registers spilling and too much global memory access, we implement all 73-point stencil operations with the *one-to-multi* scheme. Compared with 37-point stencil, the number of off-axis points in 73-point stencil is increased from 12 to 48. However, the GFlops values of two stencil operations are actually the same with the same optimization methods, which indicates our *one-to-multi* scheme can eliminate the performance damage caused by off-axis points.

Decomposition of stencil and storing the coefficient matrix into shared memory offer 80% to 130% performance improvement, which primarily gains from the reduction of un-coalesced memory access. When utilizing decomposition plan to the 73-point stencil, the coefficients of the second sub-stencil cannot fully fill the 48KB shared memory for the second sweep. Although reading coefficients from read-only cache spends more time than reading from shared memory, it reduces the overhead costs caused by launching another stencil sweep. As a result, in this case, the *Shared_Mem 2kernels* scheme is inferior to *Shared_Mem & Read-only 2kernels* scheme. However, with more different velocities values, the second scheme will gain more benefits from using the shared memory instead of the read-only cache.

2) Comparison of Fermi, Kepler and Maxwell: Titan X (Maxwell) delivers around 200% performance boost compared with K20c (Kepler). Besides the improvement in computation capacity, the increased memory bandwidth and more parallelism from Fermi to Maxwell bring the main performance boost for memory bound ETE stencil operations: 1. Memory Bandwidth: the measured global memory bandwidth increased from 120 to 241 GBps. 2. More Parallelism: (1) Maxwell has 24×128 CUDA cores. Kepler K20c has 13×192 CUDA cores while Fermi has only 16×32 cores. (2) In Kepler and Maxwell The warp-schedule number of each SM increases from 2 to 4.

Fermi M2090 has 32K registers, while Kepler K20c and Maxwell Titan X have 64K, which indicates more register pressure on Fermi. As a result, 3.5D blocking brings no benefit to Fermi architecture due to the register pressure primarily brought by allocating another register queue.

D. Comparison of ETE and FD method

To evaluate the effectiveness of our GPU design, we compare the GPU-based ETE with FD method implemented on 12-core CPUs and GPU. To gain similar imaging accuracy, we simulate the wave propagation process with $\Delta t = 0.2$ ms for FD and with $\Delta t = 1$ ms for ETE in isotropic media. The time performance for 10s wave propagation simulation is illustrated in Table II. Our ETE-based propagation is 31.21x faster than CPU-FD method and 2.41x faster than GPU-FD method provided by NVIDIA CUDA samples library.

TABLE II
COMPARISON OF GPU-ETE AND FD METHODS

Method	*CPU-FD*	*GPU-FD*	*GPU-ETE*
Time(s)	5503.1	426.90	176.31

VI. CONCLUSIONS

In this paper, we present our work on parallel GPU-based ETE solution, which makes ETE method a competitive forward modeling implementation for RTM and FWI. Our work proposes a better way to use GPU memory architectures for ETE stencils which are characterized with spatially-variant coefficients and more complex shapes. We optimize our ETE method on 6 different architectures, including Ivy bridge 12-core CPUs, NVIDIA Fermi, Tesla and Maxwell GPU and Knight Corner MIC. Our GPU-ETE achieves 385 and 434 GFLOPs for 37-point and 73-point ETE stencils, respectively. After optimizing of data transfer between GPU device memory and CPU disk, our GPU-ETE is 31.21x faster than traditional FD method to achieve similar imaging accuracy. Our designs can also server as a guidance for a broad range of FD-like forward modeling methods with variant coefficients. In the near future, we will integrate our code into the *lightning* seismic exploration software developed by CNPC.

VII. ACKNOWLEDGMENTS

The authors would like to thank BGP and CNPC for supporting the project.This work is also supported in part

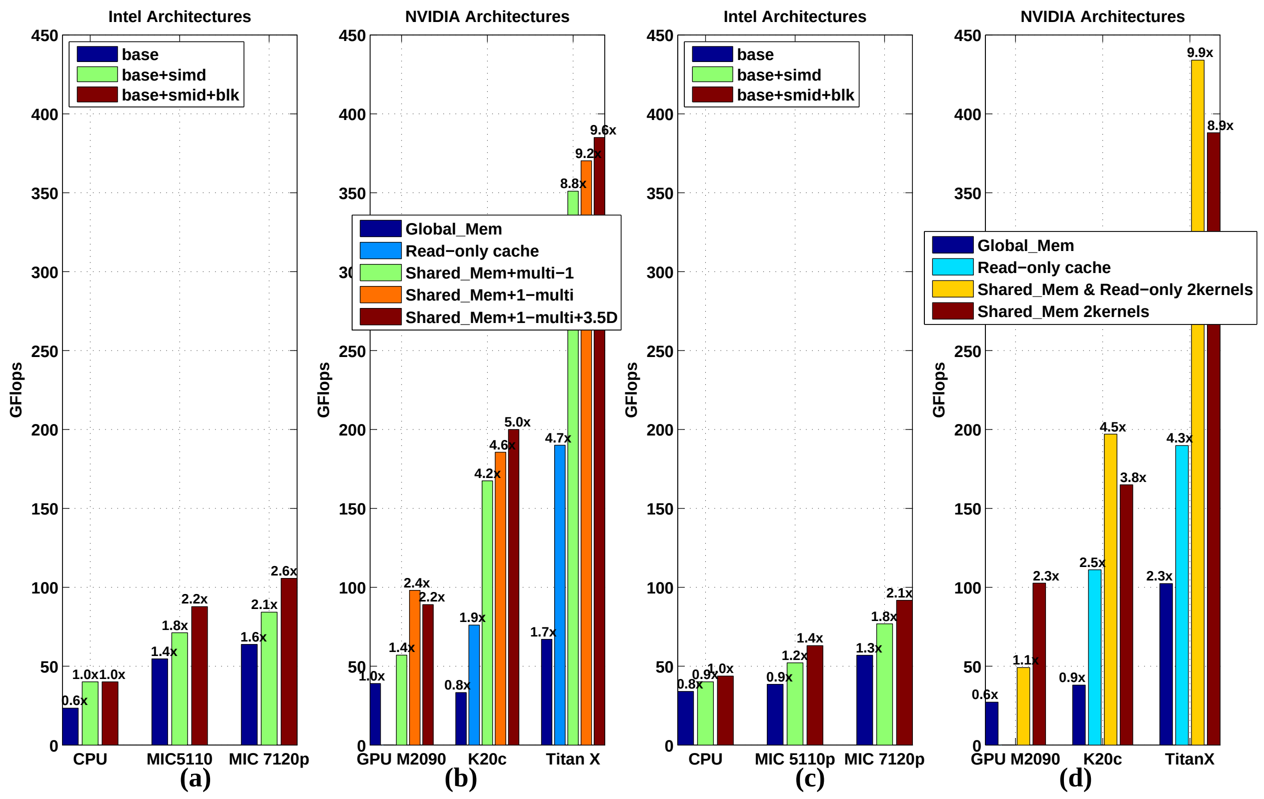

Fig. 8. Performance of ETE stencil. (a)(b) illustrate results on the 37-point ETE stencil. (c)(d) illustrate results on the 73-point ETE stencil.

by National Natural Science Foundation of China (Grant No. 61303003, 41374113), and by National High-tech R&D (863) Program of China (Grant No. 2013AA01A208).

References

[1] http://www.nvidia.com/object/gpu-applications-domain.html
[2] https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-applications-and-solutions-catalog
[3] Moczo P, Robertsson J O A, Eisner L. The finite-difference time-domain method for modeling of seismic wave propagation[J]. Advances in Geophysics, 2007, 48: 421-516.
[4] Micikevicius P. 3D finite difference computation on GPUs using CUDA[C]//Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2009: 79-84.
[5] R. Abdelkhalek, H. Calandra, O. Coulaud, J. Roman, and G. Latu. Fast Seismic Modeling and Reverse Time Migration on A GPU Cluster. In High Performance Computing & Simulation, 2009.
[6] Micha D, Komatitsch D. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards[J]. Geophysical Journal International, 2010, 182(1): 389-402.
[7] Komatitsch D, Erlebacher G, Goddeke D, et al. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster[J]. Journal of Computational Physics, 2010, 229(20): 7692-7714.
[8] 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUsExamples:
[9] Liu Y., 2013, Globally optimal finite-difference schemes based on least squares: Geophysics, 78(4): T113-T132.
[10] Liu H, Dai N, Niu F, et al., 2014, An explicit time evolution method for acoustic wave propagation: Geophysics, 79(3): T117-T124.
[11] Dai N, Liu H, Wu W, 2014, Solutions to numerical dispersion error of time FD in RTM: 84th Annual International Meeting, SEG, Expanded Abstracts, SEG-2014-0858.
[12] Tan S, Huang L, 2014, A staggered-grid finite-difference scheme optimized in the timeCspace domain for modeling scalar-wave propagation in geophysical problems: Journal of Computational Physics, 276: 613-634.
[13] Wang Y, Liang W, Nashed Z, et al., 2014, Seismic modeling by optimizing regularized staggered-grid finite-difference operators using a time-space-domain dispersion-relationship-preserving method: Geophysics, 79(5): T277-T285.
[14] You, Y., Fu, H., Huang, X., Song, G., Gan, L., Yu, W., & Yang, G. (2013, May). Accelerating the 3D Elastic Wave Forward Modeling on GPU and MIC. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International (pp. 1088-1096). IEEE.
[15] Leopold C. Tight bounds on capacity misses for 3D stencil codes[M]//Computational SciencełICCS 2002. Springer Berlin Heidelberg, 2002: 843-852.
[16] Baysal, Edip, Dan D. Kosloff, and John WC Sherwood. "Reverse time migration." Geophysics 48.11 (1983): 1514-1524.
[17] Virieux, Jean, and Stphane Operto. "An overview of full-waveform inversion in exploration geophysics." Geophysics 74.6 (2009): WCC1-WCC26.
[18] Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A portable programming interface for performance evaluation on modern processors. International Journal of High Performance Computing Applications 14(3) (2000) 189-204
[19] Micikevicius P. 3D finite difference computation on GPUs using CUDA[C]//Proceedings of 2nd workshop on general purpose processing on graphics processing units. ACM, 2009: 79-84.