Cache-friendly Design for Complex Spatially-variable Coefficient Stencils on Many-core Architectures

Jiarui Fang*†‡§, Haohuan Fu*§, Guangwen Yang*†§
†*Department of Computer Science & Technology, Tsinghua University*
**Ministry of Education Key Lab. for Earth System Modeling, Center for Earth System Science, Tsinghua University*
‡*Tsinghua National Laboratory for Information Science and Technology (TNList)*
§*National Supercomputing Center in Wuxi*

***Abstract*—Many-core architectures, such as the NVIDIA graphics processing unit and Intel Xeon Phi, which are characterized by high computation resources but limited on-chip memory capacity, have been used to significantly accelerate various computationally demanding tasks. Stencil operators are naturally suitable for such architectures because of their parallel calculation patterns. However, only simple stencils with points distributed along the axes and with constant coefficients have been fully investigated. This study first provides insights into optimization strategies for stencils with complex shapes, including off-axial points and spatially variable coefficients. Through our proposed stencil-decomposition schemes, we maintain read-only coefficients in on-chip caches to avoid unvectorized memory access. To alleviate the resulting severe cache-starvation situation, a generalized cache-friendly design for many-core architecture is proposed. It can reduce cache miss times and cache space consumption. The proposed methodology significantly improves the performance of stencil operations in a real seismic imaging application and introduces a new option to write highly efficient memory-bound stencil-like loops.**

***Keywords*-Many core architecture, GPU, Intel Xeon Phi, Spatially-variable coefficient stencil, Seismic imaging**

I. Introduction

Stencil computations are ubiquitous in scientific computing problems, such as those in fluid dynamics, seismic imaging, ocean modeling, and climate simulation. A stencil computation [4] is used to formulate approximate numerical solutions for partial differential equations by performing linear weighting of a small neighborhood in a discretized grid. To obtain an accurate and efficient approximation, scientists have shifted interest from simple constant-coefficient stencils generated from low-order equations to complex shapes and variable coefficients stencils (CSVC stencils) discretized from high-order equations with higher accuracy and longer discrete time intervals. Spatially variable coefficients generally correspond to the constitutive parameters of a physical problem, such as velocity, elastic moduli, and conductivities, depending on space or time. Complex shapes indicate that additional off-axial points are included in the stencils, which are used to reduce the misfit between the analytical solution and the numerical solution.

Before the 21st century, multi-core processors were the only solution for the problem caused by the situation in which the clock rate gains of a single processor cannot be continued anymore. Recently, the rapid development of high-performance computing technologies has resulted in many-core architectures, which can provide huge computing power higher than that of multi-core architectures. By deploying numerous computing cores and different memory hierarchies on a single chip, such platforms can generally achieve improved performance through high parallelism, and thus, are promising candidates to satisfy the large computing demands of real-world applications. For example, the graphics processing unit (GPU) designed by NVIDIA and the Xeon Phi based on the Intel Many Integrated Core (MIC) architecture are two popular many-core architectures that have been widely used in many key applications [1],[2]. Given their natural parallel calculation patterns, stencil operators are the first batch of applications that have been ported to many-core architectures.

To date, most optimization processes have been directed to design operators on many-core architectures for stencils with simple shapes generated from heat, diffusion, and Poissons equations, etc., which are characterized by constant coefficients and shapes that only include points on the axes. However, a comprehensive investigation towards implementations of CSVC stencils remains unavailable: On one hand, variable coefficients can shift the dominant working space from the grid points being updated to the coefficients themselves; Complex shapes including off-axial points can make cache reuse more difficult. CSVC stencil operations therefore can cause problems on cache locality and are featured with extremely low flop-per-byte ratios. On the other hand, in many-core architectures such as GPU and Xeon Phi, more resources are allocated for computation; consequently, less resources are allocated for on-chip cache. It is hard to unleash the computing power of many-core architectures with existing caching strategies on traditional architectures like CPU and multi-core CPU.

In this study, we propose a cache-friendly design for CSVC stencils to bridge the gap between limited on-chip storage resources on many-core architectures and the cache-starved situation arising from CSVC stencil computations. We have selected two stencils generated via the explicit time evolution (ETE) method in seismic forward modeling [10], which has been developed as a powerful alternative to the finite difference (FD) method [5], as real application cases for evaluating performance. Two ETE stencils, including 12 and 48 off-axial grid points with coefficients varying with the velocity values of that position, are presented to evaluate our design. We demonstrated that it is not possible

to achieve highly-efficient implementations of the 73-point ETE stencil using conventional optimization techniques for some memory hierarchies of many-core architectures, such as GPU.

The main contributions of this study are as follows.

- A stencil decomposition scheme for caching read-only spatially variable coefficients is proposed to alleviate unvectorized memory access.
- A cache-friendly design is introduced to reduce the cache consumption for grid point reuse, which is also highly efficient in terms of reducing cache miss rate and boundary memory traffic.
- Our designs have already been integrated into a real-world seismic imaging application. We achieve considerably better performance for the stencils with more off-axial grid points on Xeon Phi than GPU. Some insights into these two architectures for optimizing CSVC stencils are presented.

The remainder of this paper is organized as follows. In Sec. II, we present a real case that can generate CSVC stencils. In Sec. III, we introduce the many-core platforms that we used in this study. In Sec. IV, we present our solutions for spatially variable coefficients. Some insights are investigated and a cache-friendly design is proposed in Sec. V. We provide the corresponding experimental results on performance in Sec. VI. Lastly, we present related works and conclude our study in Secs VII. and VIII.

II. Generations of CSVC stencils

We present a real application for the seismic forward modeling method that generates CSVC stencils. Seismic forward modeling is the most computationally expensive part of high-quality seismic imaging methods for hydrocarbon exploration, such as reverse-time migration [11] and full waveform inversion [12].

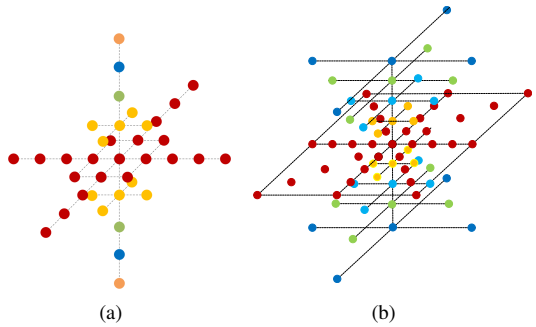
The analytical solution for the acoustic wave equation in the wavenumber domain that assumes a constant velocity is as follows:

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

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) into 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 that corresponds 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 discrete space. For a selected stencil in a neighborhood of any given point in the space domain, we search for the 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 search for c_j, i.e., the coefficients



(a) (b)

Figure 1. Shapes of two type 8th order ETE stencils. (a) is 37-point ETE stencil. (b) is 73-point ETE stencil.

of each stencil point, in the least square sense, as follows:

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

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

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

For the shapes of the ETE stencils, an 8th-order stencil, including off-axial stencil points, has been proven to be optimal for reducing the misfit of the cosine function, which can reach a magnitude of 10^{-6}. Numerical experiments from [10] indicate that the FD scheme requires time steps that are approximately five times as fine as that in ETE to achieve a similar accuracy. Therefore, the ETE operator is superior to the conventional FD scheme in terms of efficiency. Figure 1 shows two types of ETE stencils: the 37-point stencil is used in isotropic media, whereas the 73-point stencil is commonly used in tilted transversely isotropic media. The stencils are traversed multiple times through the entire 3D grid to update each grid point with calculations that involve its nearest neighbors. The updating rule for one time sweep is indicated in Equation (4).

III. Parallel computing architectures

We explore the performance of our designs on 5 platforms mentioned in Table I, where ARCH-FBR indicates the flop-to-byte ratio[1] of the architecture. We briefly introduce two many-core architectures, i.e., NVIDA GPU and Intel Xeon Phi, in this section.

A GPU card consists of a set of stream multiprocessors (SMs). Each SM has its own stream processors (SPs). Tesla

[1]Flop to Byte Ratio is the ratio between peak computation performance and the measured memory bandwidth (measured from the STREAM benchmark on CPU/MIC and the bandwidthTest facility on GPU) of the above different platforms

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 C2070 Fermi	1.15	1.03	0.515	64	768	144	100	10.3	5.15
Tesla K40c Kepler	0.75	4.29	1.43	64	1536	288	204	21.0	7.00
E5-2697 v2 has a 30M L3 cache shared among 12 cores; 64KB L1 cache on Tesla GPUs can be configured as 48KB shared memory + 16KB L1 or 16KB shared memory + 48KB L1									

C2070 has 14 SMs, each of which has 32 SPs, and K40c has 15 SMs, each of which has 192 SPs. Each SM has an on-chip memory of 64 KB that can be accessed both explicitly, as a shared memory, and implicitly, as an L1 cache. An SM also has a register file, which is 256 KB on Kepler architecture and 128 KB on Fermi architecture. Each Kepler SM is equipped with an easily accessible read-only cache of 48 KB per SM, which is an extension of the texture cache available in Fermi architecture.

The Intel Xeon Phi (Knights Corner) coprocessor features a high number of simple cores (61), with support for 512 bit wide vector processing units (VPUs). In Xeon Phi, all the cores are connected via a ring bus that transports data between caches, memories, and the outside memory space via the PCIe bus. This ring bus plays an important role in creating the L2 cache coherency of Xeon Phi. Each core has its own directly attached 512KB L2 cache. If a desired data segment is absent from the L2 cache of a core, then the L2 caches of the other cores will be probed before finally retrieving the data from the comparatively slow main memory as the last resort. Data found in the L2 cache of another core will be sent back to the requesting core through the ring bus. Xeon Phi is designed to run a maximum of four independent threads per core, where each in-order core can execute up to two instructions per cycle.

IV. DECOMPOSITION SCHEME FOR STENCILS WITH VARIABLE COEFFICIENTS

For a stencil with variable coefficients, particularly N_e different coefficients, it seems that we need to maintain a 4D coefficient array with size $N_X \times N_Y \times N_Z \times N_e$. Given that the coefficients are describing one type of property with this position in a 3D grid domain and with limited diversity, the number of distinctive coefficient values N_d is relatively small compared with the size of the 3D grid. To reduce memory consumption, we only need to store a read-only coefficient matrix with size $N_e \times N_d$ and a coefficient index array with size $N_X \times N_Y \times N_Z$ in the memory before the stencil sweeps. In the ETE method, the index array represents the velocity of the position. As shown in Fig. 2, when conducting stencil operation using the ETE method, 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.

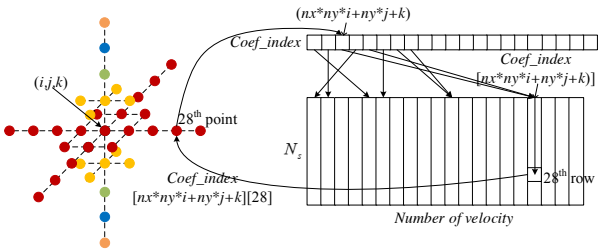


Figure 2. Memory access pattern of spatially variable coefficients of the 37-point ETE stencil.

The parameter distributions, on which the coefficients depend, generally exhibit sharp spatially discontinuities. Such distributions lead to discontinuous access to the coefficient matrix of neighbor stencil operations, which results in a non-vectorized (un-coalesced) memory access pattern on many-core architectures. Performance loss resulting from such an access pattern for multi-core CPU architectures is insignificant given that the coefficients can be maintained in a shared L3 cache (30 MB+ in Sandy Bridge 12-core CPU). However, such access pattern will lead to performance loss on many-core architectures without a large shared last-level cache. On GPU, neighbor threads (Fig. 2) access coefficients in noncontinuous global memory positions un-coalescedly. Un-coalesced accesses are an order of magnitude slower than coalesced memory accesses. The achieved global memory bandwidth is considerably lower than the peak (over 100 GB/s), which leads to stalling and wasted computation cycles. The reason is because that memory loads of 32 neighboring threads in a warp can be finished in one transaction with coalesced access, but they will be executed in many transactions with un-coalesced address patterns including large strides between. On Xeon Phi, unvectorized access patterns require more cache lines loaded into L2 or L1 cache, which will also result in noticeable bandwidth waste.

To address this problem on many-core architectures, we intend to initially prefetch and load the read-only coefficient

matrix into the on-chip cache and then to access the cache for the coefficients. Memory access for initial cache loading can be vectorized. Xeon Phi has a 512 KB L2 + 32KB L1 cache each core, which is always sufficiently large for the read-only coefficient matrix. For the GPU architecture, the shared memory with bandwidth around 1.5-2 TB/s is user-controlled for memory prefetching. However, the shared memory can be set to a maximum 48KB, which is sometimes insufficient for caching entire coefficient matrix. As a result, a large number of coefficients also have to be loaded from global memory with an un-coalesced access pattern.

To eliminate un-coalesced global memory access completely, we design stencil decomposition schemes for ETE stencils. It is based on the associative law of addition. The equ. (4) can be reformed into two equations with an equivalent effect.

$$p\left(\mathbf{x}, t+\Delta t\right) = 2\sum_{j=1}^{N_{split}} c_j\left(\mathbf{x}\right) p(x+\Delta x_j, t) - p\left(\mathbf{x}, t-\Delta t\right) \tag{5}$$

$$p\left(\mathbf{x}, t+\Delta t\right) += 2\sum_{j=N_{split}+1}^{N_e} c_j\left(\mathbf{x}\right) p(x+\Delta x_j, t) \tag{6}$$

A variable coefficient stencil can be reconfigured into two sub-stencils and the rows of the coefficient matrix are broken down by the sub-stencils. Each sub-stencil can perform independent scaling and sum calculation to update the grid points with the corresponding rows of the matrix coefficients. By using this decomposition scheme, we can then fit the coefficients of one sub-stencil shared memory on GPU to execute one sweep (complete update of all points) and reload the coefficients of the other sub-stencil into shared memory for another sweep. The decomposition scheme for the 37-point ETE stencils is illustrated in Fig. 3. Performing another sweep requires to reload grid points into the on-chip cache from global memory. On Kepler architecture, we can store one part of coefficients in shared memory and the other part of coefficients in the 48 KB read-only cache. The memory access to the read-only cache will also result in global memory access but with a longer cacheline. It can reduce wasted global memory access bandwidth but the latency for un-coalesced memory access is still far higher than shared memory access. However, with read-only cache, we perform one-time sweep instead of twice, which can reduce the overhead of reloading grid points. Thus a tradeoff should be considered between the overhead of the sweep and performance loss for un-coalesced memory access. Applicability of this scheme is discussed in Sec. VI.

V. Cache-Friendly Design for Stencils with Complex Shapes

Stencil computations for two neighboring positions require a large number of common points. Caching grid points for data reuse is a primary method to alleviate the constraints of memory bandwidth. Storing read-only coefficients in cache has already exacerbated the cache-starved situation. In this section, we describe our cache-friendly design to reduce the cache use of cacheless many-core architectures for stencils with off-axial points. Sometimes we use the 8th-order ETE stencils as an example for simplicity, although our method can be generalized for stencils with any shape.

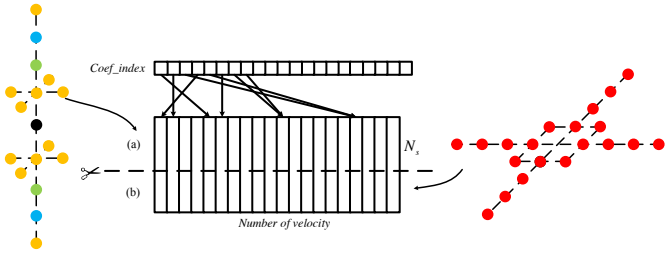


Figure 3. Stencil decomposition for the 37-point ETE stencil (The shapes can be arbitrary). The coefficients table is cut into 2 subtables, each of which can be accommodated into shared memory or read-only cache.

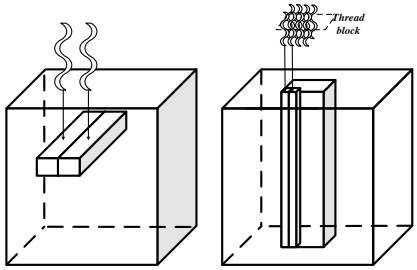
A. Insights

We first present our insights into the cache usage of CSVC stencils before introducing our design. For a simple stencil operation with points on axes, the traditional cache usage scheme [16] [23] suggests keeping all the grid points of the current time step required in the on-chip cache to update the grid points of the next time step and neighbor stencil operations can reuse these data in the cache. CSVC stencils are always handled using similar techniques. This scheme ensures good CSVC stencil performance on multi-core CPU architectures because the shared large L3 cache of multi-core architectures can maintain sufficient required grid points. However, now we need to present several shortcomings caused by this cache usage scheme on many-core architectures.

For Xeon Phi architecture, although the L2 cache can be accessed by remote cores, the latency is considerably larger than the local cache access. To guarantee that the processing work set is a resident in the local L2 cache, we consider an efficient blocking plan, shown in Fig. 4 (a), to split a large 3D grid into small blocks and then assign the calculation of one block to one thread. Applying this blocking scheme to 8th-order CSVC stencils, if nine successive layers of grid points in this block fit in the caches, then the only load operation that can cause cache miss is the last point on the bottom layer. Assuming C is the cache size (as a rule of thumb, only C/2 is available as a result of caching both data and instructions); M_{coef} is the size of the read-only coefficient matrix required to be stored in the cache; $Size_{point}$ is the storage space occupied by a grid point element; N_x is the block size of the x -axis, and N_y is the block size of the y-axis; $n_{nthread}$ is the number of threads in one core. The layer condition [17] for the traditional method is:

$$n_{threads} \times N_x \times N_y \times Size_{point} \times 9 < C/2 - M_{coef} \tag{7}$$

As discussed in Sec. III, 512 KB + 32 KB local cache space is shared by a maximum of four threads. To leverage



(a) MIC blocking (b) GPU blocking

Figure 4. The blocking strategies for many core architectures. (a) one thread is in charge of one block. (b) one thread is in charge of one line. One thread block is in charge of a column.

the wide VPU for vectorization efficiently, no blocking in the x direction is applied. This conclusion is consistent with [20]. To cache the upper layer points of the stencil, block size y should be set as a small value. Small blocks lead to numerous boundary memory traffic for the entire 3D grids. As a second choice, we violate the layer condition to adopt a large block in order to reduce boundary memory traffic and allow cache miss to occur at a position far from the center points. However, for CSVC stencils with too many points on the top and bottom layers, such as the 73-point ETE stencil, this choice results in many cache misses. To address this problem, we can consider a stencil decomposition scheme that combines appropriate blocking techniques to reduce the occurrence of cache miss. For the 37-point ETE stencil, we can split a stencil into two sub-stencils. Most of the points of the first sub-stencil are distributed on the xy-plane. The stencil is swept in the order of $x \rightarrow y \rightarrow z$ on the 3D grid. By contrast, most of the points of the other sub-stencil are distributed on the xz-plane and sweeps the 3D grid in the order of $x \rightarrow z \rightarrow y$. We observe a performance boost in this scheme. However, selecting the blocking techniques is highly inflexible, and the decomposition plan is restricted to certain stencil shapes. We also need to adopt different best blocking parameters for each substencil.

For GPU architecture, a different blocking scheme is adopted according to the architecture shown in Fig. 4 (b). The 3D grid is split along z axis by blocks. A GPU thread is in charge of the computation of a series of grid points along the z-axis. A block is calculated by a thread block. Several studies [6][18] have already implemented this concept by utilizing registers and shared memory to cache grid points. According to their designs, when the computation is traversing through the z-axis, the stencil points of the current xy-plane are stored in the shared memory, whereas the remaining points are stored in the registers. We need to store 12 points in the registers for the 37-point stencil. At each step, 5 points are required to be loaded into the registers of each thread from the global memory. That is, 5 global memory loadings for 4 off-axial columns and 1 center column are required to calculate 1 point. Afterward, the grid points of the next time step at the current xy-plane are updated[2] by points from the upper and lower 4 slices stored in the registers of each thread in addition to the points stored in the shared memory. Expanding the traditional stencil calculation design to the 73-point ETE stencil, 168 registers (9 register queues with length 8, 8 register queues with length 6, 8 register queues with length 4, and 8 register queues with length 2) are required for each thread. That is, 33 instances of global memory access are required to load points into the registers at each step. A solution for GPU is proposed in [14], where the researchers optimized a constant stencil with off-axial points and stored all grid points in the shared memory at each time. However, shared memory is a considerably precious resource for variable coefficient ETE stencils, as mentioned in the previous subsection.

In conclusion, the traditional cache usage scheme suffers from three performance concerns. (1) A large stencil radius in the outer grid dimension leads to high pressure on cache for many-core architectures because these architectures have to hold a large number of layers in caches. (2) More extra boundary memory traffic or on-chip memory allocation are required to avoid cache miss. (3) Data transfer between off-chip memory and on-chip cache is concentrated on the read operation. Most of the time, the store unit is idle.

B. Cache-friendly design

To reduce cache consumption, we introduce a cache-friendly design. We observe that when a layer is cached, a part of the cached grid points are also required by stencil operations for the upper and lower layers. As indicated in Fig.5, the solid points filled with different colors in the 37-point ETE stencil are useful to obtain the next grid points of 9 stencil operations. Instead of caching all points required for one complete stencil operation, our cache-friendly design divides one stencil operation into 9 partial operations along the xy-plane. Calculation of each part only requires the data in one xy-layer. Through rearranging the calculation order, the partial stencils requiring the data of the same xy-layer are calculated together in each thread. By caching one current xy-layer, we can update partial results of 9 layers and sum them together to obtain the final results.

$$n_{threads} \times N_x \times N_y \times Size_{point} < C/2 - M_{coef} \quad (8)$$

We now apply the cache-friendly design to the order-L CSVC stencil implementations on Xeon Phi. When sweeping from top layer to bottom layer in one block, we load current xy-layer grid points into the local caches. These points can be reused to execute partial stencil operations as shown in Fig.5. Each stencil operation can produce L intermediate results. Vector registers in Xeon Phi are used to collect intermediate results, which are accumulated to final results in the memory. Compared with traditional cache usage scheme, this design can reduce the cache space consumption to $1/L$ because only one slice of the block grid is required to be stored in the cache. The layer condition for

[2]updating indicates accumulating the values of the grid points to current values after scaling with the coefficients.

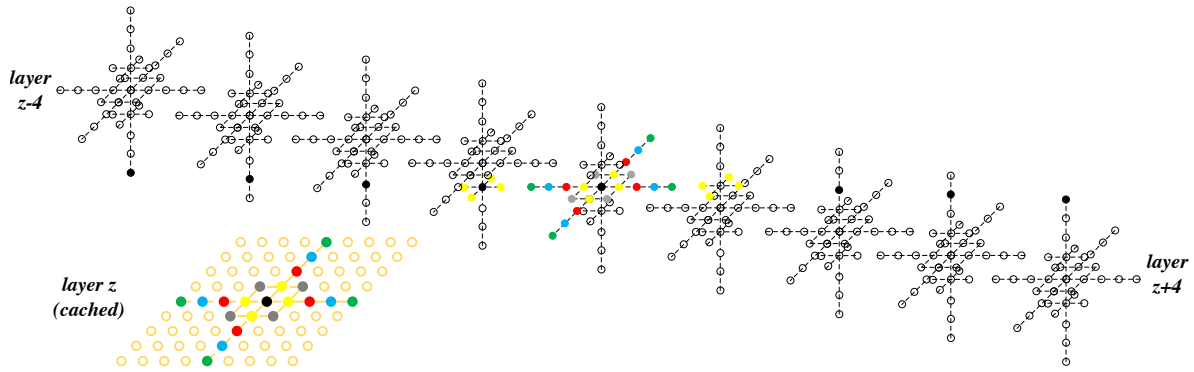


Figure 5. Colored solid grid points can be reused in cache to update the gird points of upper and lower four layers.

CSVC stencils now becomes Equ. (8) to ensure that only 1 cache miss occurs (the newest point) in the read operation for the current layer. Compared with that of the layer condition, i.e., Equ.(7), of traditional cache usage, due to N_x is the same, the N_y can be L times larger theoretically. We can adopt a larger blocking size, which indicates less memory traffic for boundary grid point loading. Most importantly, the cache-friendly design also enables less cache assess times. Grid points taken out from the cache can be involved in multiply-add operations for at most L times. However, in traditional scheme, each points taken out from the cache only take part in one multiply-add operation. Less different points in cache are required to be accessed for one stencil operation with our cache friendly design. This will finally lead to less memory access times when the theoretical layer condition can not be met. Take the 73-point ETE stencil for example, with traditional scheme, we require reading 73 grid point from 9 layers in cache and writing 1 grid point to memory. With cache-friendly, We require reading 33 grid points from the current layer in the cache and writing 9 grid points to the memory space of the next grid. Although, our cache-friendly design will result in more writing cache misses on Xeon Phi, write operations can overlap with read operations and the latency can be hidden under read operations. The cache-friendly scheme can rearrange read and write memory traffic to reduce reading pressure.

For GPU architecture, we assign a thread to compute output values for a given column along z axis and threads of a given threadblock coherently traverse the 3D grid block along z axis. When sweeping from top to bottom layer, because the points in the current slice are needed for computation by multiple threads of the same layer, input points in the current slice are stored in shared memory. When the points of the xy-layer are loaded into the shared memory, we use points from the shared memory to execute partial stencil operations of preceding and succeeding $L/2$ layers as indicated in Fig.5. The intermediate results are stored in registers rather than written directly into the off-chip global memory. In order-L CSVC stencil kernels, 1 register queue with length L is allocated for each thread and one-time global memory access is required to calculate a single point. After $L+1$ steps, register 0 is accumulated with all the points in the stencil; it contains the value of the grid point at the next time step. Afterwards, register 0 is written to the global memory and reset to zero. Then, the register queue is circularly shifted for the next xy-slice updating. While the traditional scheme requires maintaining off-axial points not in the current layer in the registers, in our cache-friendly scheme, only one register queue with length L is required to maintain the intermediate results for each thread. As a result, our cache-friendly design leads to less register consumption and less global memory access times. For the 73-point ETE stencil, compared with 168 registers required for grid points caching with traditional scheme, cache friendly only requires 8 registers regardless of the shape of stencil. In addition, the cache-friendly design requires only one global memory read access for each stencil operation, which can be regarded as a cache miss on GPU, for grid points to load them into the shared memory. However, the traditional scheme requires memory access for all the off-axial points at bottom for each stencil operation (i.e., 4 times and 17 times global memory access for the 37-point and the 73-point ETE stencils, respectively).

VI. Performance Evaluations and Discussions

We implement CSVC stencil kernels on 5 parallel architectures mentioned in Sec. III with single- and double-precision floating-point data. The Intel CPU and Xeon Phi versions are implemented with icc compiler of version 15.0.2. GPU versions are complied with CUDA driver of version 7.0. Performance is evaluated as number of floating-point operations per second (Flops), which can be measured by PAPI [15]. According to the real situation, the dimension size of 3D grid is set to be $328\times328\times936$. Our index array, which represents the velocity model in forward modeling, is a completely random distribution of 902 different velocity values. Stencil operations are performed 800 times iteratively.

A. Performance Metrics

1) Xeon Phi: Fig. 6 demonstrates the recorded performance of the 37-point and 73-point CSVC stencils generated from ETE method on Intel Xeon Phi 5110p and 7120p. *base+simd*[3] is a baseline implemented with data-alignment, OpenMP multi-threading and SIMD vectorization. *base+simd+blk* indicates that we optimize the baseline with blocking on the *y* and *z* axes. Given that Xeon Phi has a high number (61) of physical cores (4 hardware threads per core), and the instructions from the same thread cannot be executed in two consecutive cycles, context switches will occur more frequently compared with that in CPUs. The affinity mode, which directs how threads or processes are bound to cores, and the multi-threading schedule scheme, which directs how calculation assignments are bound to threads, play significant roles in optimizing system performance. In this work, configurations of the affinity mode and the schedule scheme are adjusted carefully, to ensure communication between tasks placed onto the closest adjacent cores, as much as possible, which increases the sustained intra-cache bandwidth. We observe that the implementation of $balance$ affinity and $static$ schedule scheme with 180 threads always achieves best performance. *Cache-friendly design* is implemented with our cache-friendly design mentioned in Sec. V, for which we perform a search to determine the optimal block size for the best performance. Because the cache-friendly design requires the initial values of the next grid to be reset to zeros, we also include resulting overhead in our results.

2) GPU: Fig. 7 demonstrates the recorded performance of the 37-point and 73-point CSVC stencils on Tesla C2070 and K40c cards. When it comes to the 37-point stencil: *GMem* illustrates the performance of accessing coefficients directly from global memory. *RO* shows the performance of accessing coefficients from the read-only cache. *SMem+traditional* presents the performance of the traditional cache usage scheme. *SMem+cache-friendly* shows the performance implemented with our cache-friendly design. In the later two cases, coefficients are directly accessed from the shared memory; if exceeding, remaining coefficients are accessed from the read-only cache.

Given that the traditional cache scheme will definitely slow down the stencil operations with register spilling (excessive allocated variables are stored in L1 caches) and excessive global memory accesses, we implement all the 73-point stencil operations with the cache-friendly design. The coefficient matrix size (over the 122KB for double precision) exceeds the available shared memory space. Two kinds of stencil decomposition schemes are presented in case the coefficient matrix is significantly larger than the 48KB shared memory. *SMem+2kernels(SM+SM)* indicates that we execute two spatial sweeps and store the required coefficients by either component in the shared memory in turn. For Kepler architectures, *SMem+2kernels(RO+SM)* indicates that we decompose the stencil into two sub-stencils and store the corresponding coefficient sub-matrices separately into the shared memory and the read-only cache.

[3]Blocking techniques are useless for multi-core CPU due to the large L3 cache

B. Performance Analysis

1) Stencil Decomposition Scheme: On GPU, stencil decomposition scheme is applied to 73-point ETE stencil operations. For Fermi, *SMem+2kernel(SM+SM)* scheme results in 3.56× (single-precision) and 1.58× (double-precision) performance boost compared with reading coefficients directly from global memory. For Kepler architecture, by using the read-only cache, *SMem+2kernels(SM+RO)* is superior to *SMem+2kernels(SM+SM)* version. It results in 5.18× (single-precision) and 4.39× (double-precison) performance boost compared with reading coefficients directly from global memory. For 37-point stencil, coefficients are capable to be cached in shared memory. Caching coefficients has already brought significant performance boost compared with fetching coefficients from global memory.

2) Cache-friendly Design: For the 73-point CSVC stencil, compared with the traditional scheme, the cache-friendly design enables 3.38×, 4.02× (single precision), and 2.86× and 4.29×(double precision) for Xeon Phi 5110p and Xeon Phi 7120p, respectively. For the 37-point CSVC stencil, our cache-friendly design enables 1.70× and 1.80× (single precision), and 1.68× and 1.30× (double precision) for Xeon Phi 5110p and 7120p, respectively.

Table II indicates the performance-monitoring events of cache on Xeon Phi, which including the L1 and L2 cache miss times of 800 times stencil operations. Because each Xeon Phi core can access local L2 cache and remote L2 caches, the missed data from L1 cache can be hit at local L2 cache or filled at remote L2 caches and memory. We treat the latter two cases as a L2 cache miss. We use *L2_DATA_READ/WRITE_MISS_CACHE_FILL* to indicate counts of data loads that missed the local L2 cache, but were serviced by a remote L2 cache and use *L2_DATA_READ/WRITE_MISS_MEM_FILL* to indicate counts of data loads were serviced by the memory. We can see our cache-friendly design can significantly reduce the L1 cache misses and achieve higher L1 hit ratios. It can also be inferred that the total L1 cache access times of cache-friendly designs, which can be achieved by *L1_CACHE_MISSES/(1-L1_HIT_RATIO)*, is far less than the traditional schemes. The cache-friendly design results in less L2 cache read miss times than the traditional scheme, however results in more write cache misses. Moreover, except for the 37-point double-precision case, cache friendly designs have less L2 cache read+load miss times.

Fig. 8 illustrates performance distributions with different blocking plans for the cache-friendly design and the traditional scheme. The cache-friendly design always achieves best performance with larger block sizes than the traditional scheme. A small block results in more extra memory traffic for the boundary grid points and finally results in more overhead.

On GPU, compared with versions implemented with the traditional scheme, for the 37-point stencil, our cache-

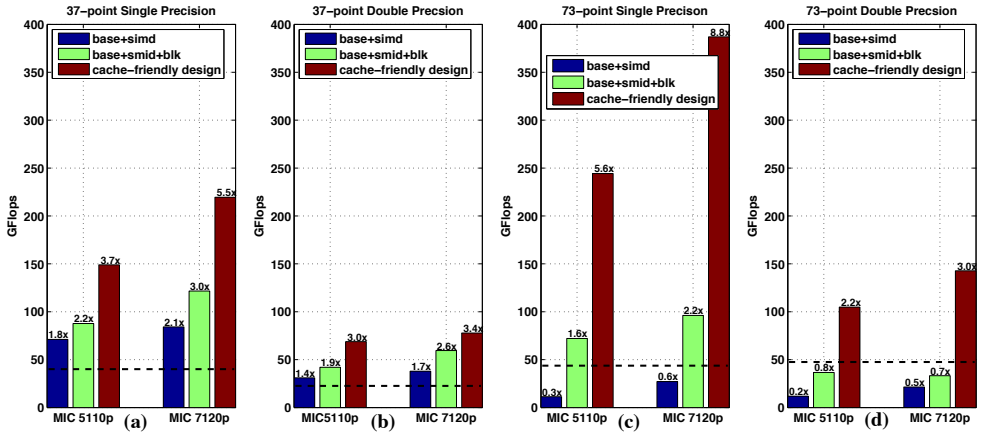


Figure 6. Performance of ETE stencils. (a)(b) illustrate results on the 37-point ETE stencil. (c)(d) illustrate results on the 73-point ETE stencil. Dashed lines indicate the performance of a 12-core CPU version. The number above each bar indicates the speedup compared with a 12-core CPU version.

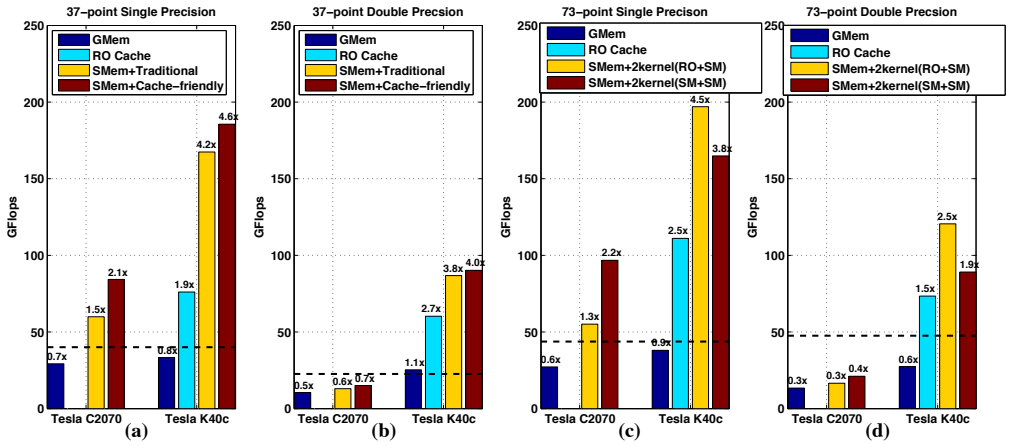


Figure 7. Performance of ETE stencils. (a)(b) illustrate results on the 37-point ETE stencil. (c)(d) illustrate results on the 73-point ETE stencil. Dashed lines indicate the performance of a 12-core CPU version. The number above each bar indicates the speedup compared with a 12-core CPU version.

Table II
THE PERFORMANCE-MONITORING EVENTS OF CACHE ON XEON PHI 5110P WITH INTEL VTUNE 2016.

Methods	BEST BLKING($y \times z$)	L1 CACHE MISS	L1 Hit Ratio	L2 DATA READ MISS CACHE FILL	L2 DATA WRITE MISS CACHE FILL	L2 DATA READ MISS MEM FILL	L2 DATA WRITE MISS MEM FILL
37-single-traditional	1×8	73,376,750,000	0.945	18,324,000,000	4,227,500,000	8,398,000,000	3,240,500,000
37-single-cache-friendly	8×16	29,011,000,000	0.949	5,033,000,000	6,521,000,000	12,329,500,000	8,996,000,000
73-single-tradition	4×16	81,137,250,000	0.865	3,853,000,000	5,090,500,000	11,161,500,000	8,818,000,000
73-single-cache-friendly	8×32	36,616,500,000	0.945	3,597,500,000	4,989,500,000	11,532,500,000	8,379,000,000
37-double-tradition	1×8	190,261,500,000	0.884	34,558,500,000	9,055,000,000	18,841,500,000	6,847,000,000
37-double-cache-friendly	8×16	85,145,000,000	0.894	7,258,000,000	11,730,500,000	41,343,500,000	47,169,500,000
73-double-tradition	2×16	1,033,688,500,000	0.739	86,751,000,000	21,951,000,000	44,123,500,000	11,161,000,000
73-double-cache-friendly	4×16	76,509,500,000	0.934	10,381,500,000	13,125,500,000	27,778,500,000	34,574,500,000

friendly design results in 1.41× (on Fermi), 1.11× (on Kepler) performance boost implemented with single-precision and 1.16× (on Fermi), 1.05× (on Kepler) performance boost implemented with double-precision.

The Table. III illustrates the register spilling situation on C2070 (Fermi). When variables can not be assigned to registers, the spilled variables are located in the L1 or even in the L2 cache. Register spilling will hurt the performance

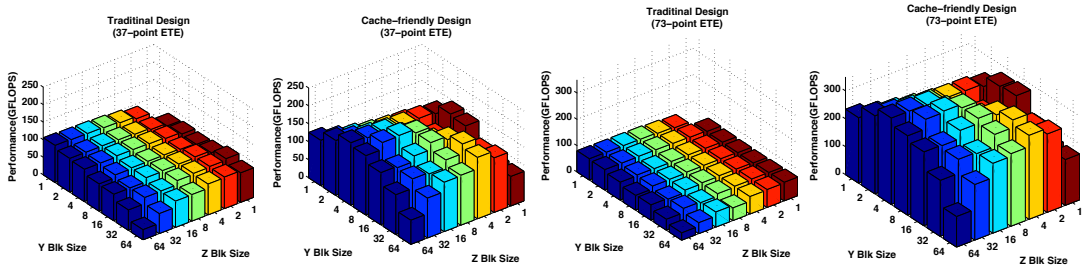


Figure 8. Searches to find the optimal cache block size using cache-friendly design for 37-point ETE with single-precision floating-point on Intel Xeon Phi 5110. Each cache blocks x-dimension (contiguous in memory) is uncut.

because memory traffic that leaves register (goes to L1 or even L2) is much more expensive. We can observe that our cache-friendly scheme always results in less register spilling for data load and store on GPU architecture.

Table III
REGISTER SPILLING SITUATION ON NVIDIA C2070.

Methods	SPILL STORE	SPILL LOAD
37-single-traditional	280	196
37-single-cache-friendly	200	144
73-single-traditional	—	—
73-single-cache-friendly	328	262
37-double-traditional	440	356
37-double-cache-friendly	206	152
73-double-traditional	—	—
73-double-cache-friendly	532	432

3) Comparison between Intel Xeon Phi and GPU: Although Xeon Phi architectures actually have superior peak performance and memory bandwidth than GPU architectures for the same generation, they have long been considered an inferior option to GPU for implementing stencil calculations. We believe that the superior performance of Xeon Phi obtained in our works is attributed to the local cache being considerably larger than GPU (There are 2048 KB L1 cache and 32768 KB L2 data cache in total on KNC Xeon Phi, while only 960 KB L1 cache and 1536 KB L2 data cache in total on Kepler GPU.); having a good mechanism to use the cache efficiently will significantly unleash the power of Xeon Phi.

We observe the substantial performance improvement from the 37-point to the 73-point CSVC stencil operations on Xeon Phi. It is because that the flop-to-byte ratios are increased by our cache-friendly design and the memory bound issues are largely alleviated. However, no such trend is observed on GPU. For the 73-point stencil, although memory traffic for grid points remains the same with the 37-point stencil, the coefficient matrix cannot be accommodated into the L1 cache anymore. Overhead has to be paid for the caching coefficients with remedies such as stencil decomposition and read-only cache compensation. Numerous off-axial points result in a large coefficient matrix, which will cause problems in the small user-controlled L1 cache of GPU. Consequently, the cache-friendly design enables Xeon Phi to outperform GPU for CSVC stencils, especially those with more off-axial points.

C. Application Scenarios

We have already integrated our GPU-based CSVC stencils with our cache-friendly design into the *GeoEast-Lightning* seismic exploration software [3] developed by China National Petroleum Corporation (CNPC). We ported the 37-point ETE stencil single-precision operations for the 12-core Intel Xeon X5675 CPU to the NVIDIA Tesla M2090 GPU. After optimizing data transfer between the GPU device memory and the CPU disk with CUDA streams, our design reduces the entire seismic imaging time from 6195s to 1935s with 256 nodes for SEG salt model, a widely used benchmark the field of geophysics.

VII. RELATED WORKS

In [16], the researchers first achieved an order of magnitude speedup for a constant coefficient 3D-8th order FD stencil with a spatially blocking scheme on Tesla-10 GPU. A series of works [7] [8] [9] based on this research has been published. Considering architecture improvement, [18] presented a series of optimization techniques for the 7-point 3D stencil with constant coefficient computations on NVIDIA Kepler GPUs. Their method achieves approximately 80% of the estimated peak performance of the roofline model. Moreover, [13] [18] [21] [22] applied temporal blocking (time skewing) techniques to GPU stencil kernels. However, it has been reported [21] [22] that the temporal blocking was merely suitable for stencils with a limited range, like 3D 7-point stencil. Current stencil optimization techniques for GPU narrow down their work to simple constant coefficient shapes with points on axes. Although, several works, such as [21], have already encountered complex shape stencils, they do not place special consideration for their optimization techniques. As an exception, [14] discussed the optimization techniques for the 3D Lax-Wendroff stencil, which includes off-axial points with constant coefficients. However, their methods consumed a huge amount of shared memory and were actually not unsuitable for a shared-memory starved situation resulting from the caching variable coefficients.

For Intel Xeon Phi architectures, [19] outlined an approach to adapt stencils from 3D MPDATA algorithm to the Xeon Phi architecture. Meanwhile, [20] focus on key

issues that should be considered in order to achieve optimal performance on the Xeon Phi architecture for 3D stencil codes. However, no CSVC stencil optimization has yet been discussed for the Xeon Phi. Xeon Phi optimization has also been taken in [14]; however, only several conventional optimization methods were performed and no cache usage was optimized.

VIII. Conclusions

In this study, we present a cache-friendly design to use on-chip fast memory for the operation of stencils, which are characterized with spatially variable coefficients and complex shapes. Such an approach gives us the possibility to ease memory bounds by reducing the cache miss, cache consumption, cache access times and the boundary memory traffic. We report our comprehensive experimental results for two CSVC stencils generated from seismic forward modeling on several popular many-core architectures, including NVIDIA Fermi, Tesla GPU, and Knight Corner Xeon Phi. On Xeon Phi, Over $4\times$ speedup achieved for a 73-point CSVC stencil. On GPU, Our design enables a highly efficient implementation of the 73-point CSVC, which has been previously impossible using the traditional scheme. We provide a new option to write highly efficient memory-bound stencil-like loops, and demonstrate the final results to integrate our code into a real oil exploration application.

IX. acknowledgements

This work was supported in part by National Natural Science Foundation of China (Grant No. 61303003, 41374113) and Tsinghua University Initiative Scientific Research Program (no. 20131089356).

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] http://www.bgp.com.cn/Technology/GeoEast-Lightning.html
[4] Meis T, Marcowitz U. Numerical solution of partial differential equations[M]. Springer Science & Business Media, 2012.
[5] 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.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[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] Baysal, Edip, Dan D. Kosloff, and John WC Sherwood. "Reverse time migration." Geophysics 48.11 (1983): 1514-1524.
[12] Virieux, et al.An overview of full-waveform inversion in exploration geophysics[J]. Geophysics, 2009, 74(6): WCC1-WCC26.
[13] Nguyen A, Satish N, Chhugani J, et al. 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs[C]//Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society, 2010: 1-13.
[14] You Y, Fu H, Song S L, et al. Evaluating multi-core and many-core architectures through accelerating the three-dimensional LaxCWendroff correction stencil[J]. International Journal of High Performance Computing Applications, 2014, 28(3): 301-318.
[15] 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
[16] 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.
[17] H. Stengel, J. Treibig, G. Hager and G. Wellein, Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model. http://arxiv.org/abs/1410.5010
[18] Maruyama N, Aoki T. Optimizing stencil computations for NVIDIA Kepler GPUs[C]//Proceedings of the 1st International Workshop on High-Performance Stencil Computations, Vienna. 2014: 89-95.
[19] Szustak L, Rojek K, Wyrzykowski R, et al. Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture[J]. Proce. HiStencils, 2014, 14: 51-56.
[20] José, Mario Hernández1 Juan M Cebrián and Garcıa, M Cecilia3 José M. Evaluating 3-D Stencil codes on Intel Xeon Phi: Limitations and Trade-offs[J] XXVI Jornadas de Paralelismo, Córdoba (Spain), pp. 568-573..
[21] G. Zumbusch. Vectorized Higher Order Finite Difference Kernels. In State-of-the-Art in Scientific and Parallel Computing (PARA), 2012.
[22] J. Holewinski, L. N. Pouchet, and P. Sadayappan. Highperformance code generation for stencil computations on GPU architectures. In Proceedings of the 26th ACM international conference on Supercomputing, ICS 12, pages 311C320. ACM, 2012.
[23] Datta K, Murphy M, Volkov V, et al. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures[C]//Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008: 4.