doi:10.3772/j.issn.1006-6748.2017.02.010

A case study of 3D RTM-TTI algorithm on multicore and many-core platforms①

Zhang Xiuxia (张秀霞)②*** , Tan Guangming* , Chen Mingyu* , Yao Erlin*
(*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, P. R. China)
(**University of Chinese Academy of Sciences, Beijing 100049, P. R. China)

Abstract

3D reverse time migration in tiled transversly isotropic (3D RTM-TTI) is the most precise model for complex seismic imaging. However, vast computing time of 3D RTM-TTI prevents it from being widely used, which is addressed by providing parallel solutions for 3D RTM-TTI on multicores and many-cores. After data parallelism and memory optimization, the hot spot function of 3D RTM-TTI gains 35.99X speedup on two Intel Xeon CPUs, 89.75X speedup on one Intel Xeon Phi, 89.92X speedup on one NVIDIA K20 GPU compared with serial CPU baseline. This study makes RTM-TTI practical in industry. Since the computation pattern in RTM is stencil, the approaches also benefit a wide range of stencil-based applications.

Key words: 3D RTM-TTI, Intel Xeon Phi, NVIDIA K20 GPU, stencil computing, many-core, multicore, seismic imaging

0 Introduction

3D reverse time migration in tiled transverse isotropy (3D RTM-TTI) is the most precise model used in complex seismic imaging, which remains challenging due to technology complexity, stability, computational cost and difficulty in estimating anisotropic parameters for TTI media[1,2]. Reverse time migration (RTM) model was first introduced in the 1983[3] by Baysal. However, the 3D RTM-TTI model is more recent[1,2,4], which is much more precise and intricate in complex seismic imaging. Normally, RTM-TTI needs thousands of iterations to get image data in particular precision. In our practical medium-scale data set, it takes around 606 minutes to iterate 1024 times with five processes on Intel Xeon processors. It will cost more when dealing with larger dataset or iterating more times in order to get more accurate result in future experiments. Enormous computing time prevents 3D RTM-TTI from being widely used in industry.

The limitations of current VLSI technology resulting in memory wall, power wall, ILP wall and the desire to transform the ever increasing number of transistors on a chip dictated by Moore's Law into faster computers have led most hardware manufacturers to design multicore processors and specialized hardware accelerators. In the last few years, specialized hardware accelerators such as the Cell B. E. accelerators[5], general-purpose graphics processing units (GPGPUs)[6] have attracted the interest of the developers of scientific computing libraries. Besides, more recent Intel Xeon Phi[7] also emerges in Graph500 rankings. High performance energy efficiency and high performance price ratio feature these accelerators. Our work is trying to address the enormous computing time of 3D RTM-TTI by utilizing them.

The core computation of RTM model is a combination of three basic stencil calculations: x-stencil, y-stencil and z-stencil as explained later. Although the existing stencil optimization methods could be adopted on GPU and CPU, it's more compelling than ever to design a more efficient parallel RTM-TTI by considering the relationship among these stencils. Besides, there is not much performance optimization research on Intel Xeon Phi. Fundamental research work on Intel Xeon Phi is needed to find their similarity and difference of the three platforms.

In this paper, implementation and optimization of 3D RTM-TTI algorithms on CPUs, Intel Xeon Phi and GPU are presented considering both architectural features and algorithm characteristics. By taking the algo-

① Supported by the National Natural Science Foundation of China (No. 61432018).
② To whom correspondence should be addressed. E-mail: zhangxiuxia@ict.ac.cn
Received on Apr. 16, 2016

rithm characteristics into account, a proper low data coupling task partitioning method is designed. Considering architecture features, a series of optimization methods is adopted explicitly or implicitly to reduce high latency memory access and the number of memory accesses. On CPU and Xeon Phi, we start from parallelization in multi-threading and vectorization, kernel memory access is optimized by cache blocking, huge page and loop splitting. On GPU, considering GPU memory hierarchy, a new 1-pass algorithm is devised to reduce computations and global memory access. The main contributions of this paper can be summarized as follows:

1. Complex 3D RTM-TTI algorithm is systematically implemented and evaluated on three different platforms: CPU, GPU, and Xeon Phi, which is the first time to implement and evaluate 3D RTM-TTI on these three platforms at the same time.

2. With deliberate optimizations, the 3D RTM-TTI obtains considerable performance speedup which makes RTM-TTI practical in industry.

3. Optimization methods are quantitatively evaluated which may guide other developers and give us some insight about architecture in software aspect. By analyzing the process of designing parallel codes, some general guides and advice in writing and optimizing parallel program on Xeon Phi, GPUs and CPUs are given.

The rest of the paper is organized as follows: An overview of algorithm and platform is given in Section 1. Section 2 and 3 highlight optimization strategies used in the experiments on CPU, Xeon Phi and GPU respectively. In Section 4, the experimental results and analysis of the results are presented. Related work is discussed in Section 5. At last, conclusion is done in Section 6.

1 Background

To make this paper self-contained, a brief introduction is given to 3D RTM-TTI algorithm, then the architecture of Intel MIC and NVIDIA GPU K20 and programming models of them are described respectively.

1.1 Sequential algorithm

RTM model is a reverse engineering process. The main technique for seismic imaging is to generate acoustic waves and record the earth's response at some distance from the source. It tries to model propagation of waves in the earth in two-way wave equation, once from source and once from receiver. The acoustic isotropic wave can be written as partial differential functions[8]. Fig. 1 shows the overall 3D RTM-TTI algorithm, which is composed of shots loop, nested iteration loop and nested grid loop. Inside iteration, it computes front and back propagation wave field, boundary processing and cross correlation. In timing profile, most of the computation time of 3D RTM-TTI algorithm is occupied by the wave field computing step. Fig. 2 shows the main wave updating operations within RTM after discretization of partial differential equations. Wave updating function is composed of derivative computing, like most finite differential computing, they belong to stencil computing. Three base stencils are combined to form xy, yz, xz stencils, as Fig. 3 shows. Each cell in wave field needs a cubic of $9 \times 9 \times 9$ to update as Fig. 4 shows. All these three stencils have overlapped memory access.

```
1: function 3D-RTM-TTI
2: /* Input:shots */
3: /* Ouput: image */
4: Read parameters, compute parameters
5: for All shot points do
6:   for All iterations do
7:     for All main grids do
8:       call Tstep-2DC //update wave P,Q
9:     end for
10:    for Boundry area do
11:      Spong boundray absorbing for P, Q
12:    end for
13:    Update P and Q
14:    Cross-correlation and update image
15:  end for
16: end for
```

Fig. 1 Overall 3D-RTM-TTI algorithm

```
1: function Tstep-2DC
2: /* Input:p, q */
3: /* Ouput:pp, qq(new p and q) */
4: for All main grids do
5:   compute dpx, dpy
6:   compute dqx, dqy
7:   compute dpx2, dpy2, dpxy, dpyz, dpxz
8:   compute dqx2, dqy2, dqxy, dqyz, dqxz
9:   combine them to get pp, qq
10: end for
```

Fig. 2 Wave updating function

1.2 Architecture of Xeon Phi

Xeon Phi (also called MIC)[7] is a brand name given to a series of manycore architecture. Knight Corner is the codename of Intel's second generation manycore architecture, which comprises up to sixty-one processor cores connected by a high performance on-die bidirectional interconnect. Each core supports 4 hardware threadings. Each thread replicates some of the architectural states, including registers, which makes it very fast to switch between hardware threads. In addi-

tion to the IA cores, there are 8 memory controllers supporting up to 16 GDDR5 channels delivering up to 5.5GT/s. In each MIC core, there are two in-order pipelines: scalar pipeline and vector pipeline. Each core has 32 registers of 512 bits width. Programming on Phi can be run both natively like CPU and in offload mode like GPU.

1.3 Kepler GPU architecture

NVIDIA GPU[6] is presented as a set of multiprocessors. Each one is equipped with its own CUDA cores and shared memory (user-managed cache). Kepler is the codename for a GPU microarchitecture developed by NVIDIA as the successor to the Fermi. It has 13 to 15 SMX units, as for K20, the number of SMX units is 13. All multiprocessors have access to global device memory. Memory latency is hidden by executing thousands of threads concurrently. Registers and shared memory resources are partitioned among the currently executing threads, context switching between threads is free.

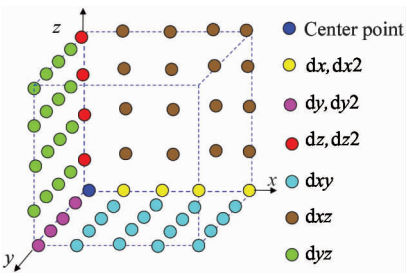

Fig. 3 One wave field point updating

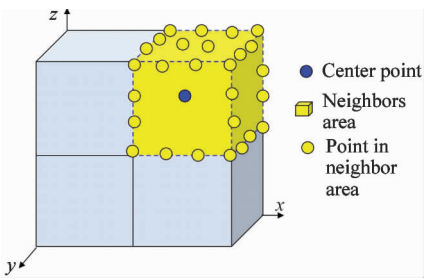

Fig. 4 Stencil in a cubic

2 Implementation and optimization on Intel Xeon Phi and CPU

Optimizing RTM on Intel Xeon Phi and CPU is similar due to similar programming model, the optimization methods of these two platforms are proposed in detail in this section.

2.1 Parallelization

2.1.1 Multi-threading

Intel threading building blocks (TBB) thread library is used to parallelize 3D RTM-TTI codes on CPU and Xeon Phi. Since grid size is much larger than the thread size, the task is partitioned in 3D dimension sub-cubic. Fig. 5 demonstrates TBB template for 3D task partition, and the task size is (bx, by, bz). On CPU and Xeon Phi platforms, each thread computes deviations in the sub-cubic. An automatic tuning technique is used to search the best number of threads. For RTM application, the optimal number of threads on Xeon Phi is 120, the best threads number of Intel Xeon CPU NUMA-core is 12 threads.

```
parallel for(blocked range3dhinti(0, (nz-9)/bz+1, 1, 0, (ny-
9)/by+1, 1, 0, (nx-9)/bx+1, 1),
[=](const blocked range3dhintir)
for int k = r. pages(). begin(); k! =r. pages(). end(); ++k do
for int j = r. rows(). begin(); j! =r. rows(). end(); ++j do
  for int i = r. cols(). begin(); i! =r. cols(). end(); ++i do
    int ii = (k×bz+4) × ny × nx + (j × by +4) × nx + i × bx +4;
      int ni =(k ×((ny-9)/by+1) ×((nx-9)/bx+1) +j×((nx
        -9)/bx+1) +i) × (bz × by × bx);
    //stencil computing
    end for
  end for
end for
);
```

Fig. 5 Parallel template of stencil function using TBB

2.1.2 Instruction level parallel: SIMDization

One of the most remarkable features of Xeon Phi is its vector computing unit. Vector length is 512 bits, which is larger than CPU's vector 256 bits AVX vector. One Xeon Phi vector instruction can be used to compute 512/8/4 = 16 single float type data at once. Vector instruction is used by unrolling the innermost loop and using *#pragma simd* intrinsic.

2.2 Memory optimization

2.2.1 Cache blocking

Cache blocking is a standard technique for improving cache reuse, because it reduces the memory bandwidth requirement of an algorithm. The data set in a single computing node in our application is 4.6GB, whereas cache size for the processors in CPU and Xeon Phi is limited to a few MBs. The fact that higher performance can be achieved for smaller data sets fitting into cache memory suggests a divide-and-conquer strategy for larger problems. Cache blocking is an effect way to improve locality. Cache blocking is used to increase spatial locality, i.e. referencing nearby memory addresses consecutively, and reduce effective memory access time of the application by keeping blocks of future array references at the cache for reuse. Since the

data total used is far beyond cache capacity and non-continuous memory access, a cache miss is unavoidable. It's easier to implement cache blocking on the basis of our previous parallel TBB implementation, because TBB is a task based thread library, each thread can do several tasks, so a parallel program can have more tasks than threads. The task size (bx, by, bz) is adjusted to small cubic that could be covered by L2 cache.

2.2.2 Loop splitting

Loop splitting or loop fission is a simple approach that breaks a loop into two or more smaller loops. It is especially useful for reducing the cache pressure of a kernel, which can be translated to better occupancy and overall performance improvement. If multiple operations inside a loop body reply on different inputs and these operations are independent, then, the loop splitting can be applied. The splitting leads to smaller loop bodies and hence reduces the loop register pressure. The data flow of P and Q are quite decoupled. It's better to split them to reduce the stress of cache. Iterate on dataset P and Q respectively.

2.2.3 Huge page table

Since TLB misses are expensive, TLB hits can be improved by mapping large contiguous physical memory regions by a small number of pages. So fewer TLB entries are required to cover larger virtual address ranges. A reduced page table size also means a reduction memory management overhead. To use larger page sizes for shared memory, huge pages must be enabled which also locks these pages in physical memory. The total memory used is 4.67GB, and more than 1M pages of 4kB size will be used, which exceeds what L1 and L2 TLB can hold. By observation of the algorithm, it is found that P and Q are used many times, huge pages are allocated for them. Regular 4kB page and huge page are mixedly used together. The using method is simple. First, interact with OS by writing our input into *the proc* directory, and reserve enough huge pages. Then use *mmap* function to map huge page files into process memory.

3 Implementation and optimizations on GPU

3.1 GPU implementation

The progress of RTM is to compute a serials of derivatives and combine them to update wave field P and Q. In GPU implementation, there are several separate kernels to compute each derivative. Without losing generality, we give an example how to compute dxy in parallel. The output of this progress is a 3D grid of dxy. Task partition is based on result dxy. Each thread compute nz points, each block compute $bx \cdot by$ panel, and lots of blocks will cover the total grid.

3.2 Computing reduction and 1-pass algorithm optimization

Fig. 3 shows several kinds of derivatives. The traditional 2-pass computation is to compute 1-order derivative dx, dy, dz, and then compute dxy, dyz, dxz based on it. This method will bring additional global reads , global writes and storage space. A method to reduce global memory access is devised by using shared memory and registers:1-pass algorithm. Similar to 2-pass algorithm, each thread computes a z-direction result of dxy. The 1-order result xy-panel is stored in shared memory, and register double buffering is used to reduce shared memory reading. Fig. 6 shows a snapshot of register buffering.

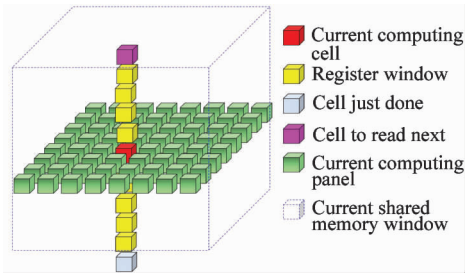

Fig. 6 1-pass computing window snapshot

4 Evaluation

4.1 Experiment setup

The experiment is conducted on three platforms. The main parameters are listed in Table 1. The input of RTM is single pulse data with grid dimension of 512 × 312 × 301. The algorithm iterates 1000 times. The time in this section is the average time of one iteration.

Table 1 Architecture parameters

Item	Xeon E5-2670	Xeon Phi 7120P	TESLA K20
# of cores	8	61	2496
Hardware Threads	16	240	2496
Clock Frequency	2.6GHz	1.238GHz	706 MHz
Bandwidth (GB/s)	51.2	352	208
Peakflops (Single, Gflops)	31.4	1010(225w)	1170

4.2 Overall performance

Fig. 7 shows performance comparison of three platforms. Our optimized 3D RTM-TTI gains considerable performance speedup. The hotspot function of 3D RTM-TTI gains 35.99X speedup on two Intel Xeon CPUs,

89.75X speedup on one Intel Xeon Phi, 89.92X speedup on one NVDIA K20 GPU compared with serial CPU baselines. Our work makes RTM-TTI practical in industry. The result also shows obviously that accelerators are better at 3D RTM-TTI algorithm than traditional CPUs. The hotspot function gains around 2.5X speedup on GPU and Xeon Phi than that on two CPUs. On one hand, because the data dependency in RTM algorithm is decoupled, plenty of parallelism could be applied. Accelerators have more cores, threads, and wider vector instructions. For example, Xeon Phi has 60 computing cores. Besides that, it has 512-bit width vector instruction. Tesla K20 GPU has 2496 cores. Hence, accelerators are good at data parallelism computing. RTM algorithm is a memory bounded application. Accelerators like Xeon Phi and GPU have 7X and 5X more theoretical memory bandwidth than CPU as shown in Table 1.

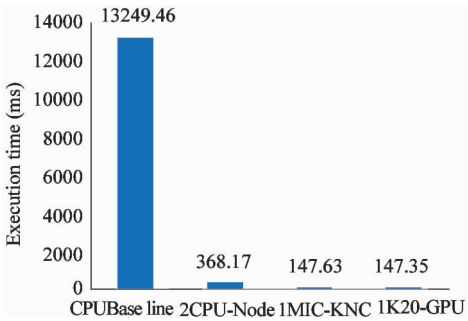

Fig. 7 Performance evaluations of three platforms

4.3 Performance analysis

On CPU, the wave updating function gains 35.99X speedup compared with single thread CPU baseline. 20.12X speedup comes from parallelism of multi-threading and vector instruction as 1.96X comes from memory optimization, such as cache blocking, loop splitting and huge page configuring, as Figs 8 and 9 show.

Fig. 10 and Fig. 11 show the parallelism and memory optimization performance of Xeon Phi respectively. RTM gains 13.81X for using 512-bit vector instruction

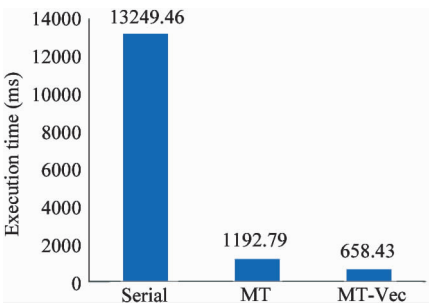

Fig. 8 Parallelism evaluation on CPU (MT: multi-threading, Vec: vectorization)

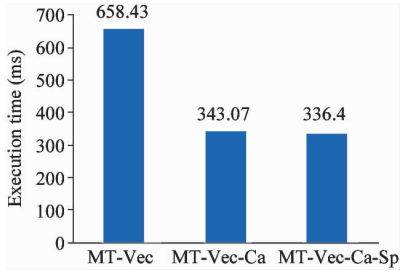

Fig. 9 Memory optimization on CPU (Ca: cache blocking, Sp: splitting)

on Phi. From Table 1, it is seen that the ideal speedup for single float SIMD on Xeon Phi is 16X. SIMD is near to the ideal limit. It's due to cache miss which will make the pipeline stalled. The multi-threading on Xeon Phi gains 40.13X speedup, there are 60 cores on Xeon Phi. Xeon Phi has very good scalability in multi-threading and wide vector instruction. RTM gains 2.08X speedup due to cache blocking, because cache blocking reduces cache miss rate and provides good memory locality which will benefits SIMD and multi-threading. RTM gains 1.44X by using huge page for reducing L2 TLB miss rate. Loop splitting gains 1.69X speedup to reduce cache pressure in advance. When compared on the same platform, 2806.13X speedup is gained compared with the single thread Xeon Phi baseline. Of this, 554.53X is from parallelism of multi-threading and vector instruction, 5.06X is achieved from memory optimization. Here Intel Phi is more sensitive to data locality according to more speedup gains from explicit memory optimization.

As Fig. 12 shows, RTM gains 1.23X speedup by using 1-pass algorithm on GPU, and 1.20X speedup by using texture memory in 1-pass algorithm. In total,

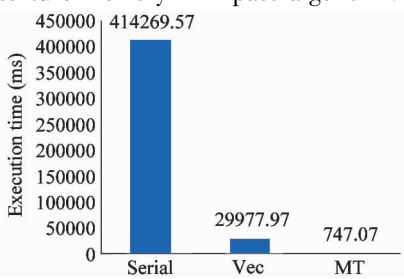

Fig. 10 Parallelization on Phi

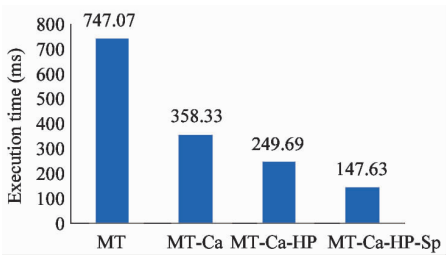

Fig. 11 Memory optimization on Phi (HP: huge page)

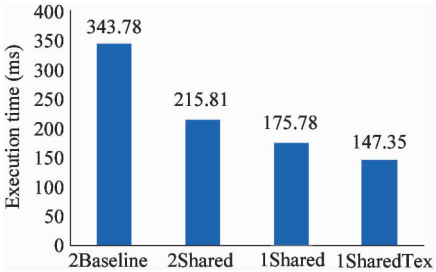

Fig. 12 Memory optimization on GPU evaluation (2:2-pass, 1:1-pass, shared:using shared memory, Tex: using texture memory)

the hot spot function gains 2.33X speedup compared with the baseline parallel GPU implementation. Threads block and grid selection are very important to the performance of application. Making full use of fast memory, such as shared memory and texture memory, will benefit application a lot. Explicit data locality plays an important role in application performance on GPU.

5 Related work

Araya-Polo[9] assessed RTM algorithm in three kinds of accelerators: IBM Cell/B. E., GPU, and FPGA, and suggested a wish list from programming model, architecture design. However they only listed some optimization methods, and didn't evaluate the impact quantitatively on RTM performance. Their paper was published earlier than Intel Xeon Phi, so performance about Xeon Phi is not included in that paper. In this paper, we choose much more popular platforms, and we evaluated each optimization method quantitatively. Heinecke[10] discussed performance of regression and classification algorithms in data mining problems on Intel Xeon Phi and GPGPU, and demonstrated that Intel Xeon Phi was better at sparse problem than GPU with less optimizations and porting efforts. Micikevicius[11] optimized RTM on GPU and demonstrated considerable speedups. Our work differs from his in that the model in his paper is average derivative method, our's model is 3D RTM-TTI, which is more compelling.

6 Conclusion and Future work

In this paper, we discussed the enormously time-consuming but important seismic imaging application 3D RTM-TTI by parallel solution, and presented our optimization experience on three platforms: CPU, GPU, and Xeon Phi. To the best of our knowledge this is the first simultaneous implementation and evalution of 3D RTM-TTI on these three new platforms. Our optimized 3D RTM-TTI gains considerable performance speedup. Optimization on the Intel Xeon Phi architecture is similiar to CPU due to similar x86 architecture and programming model. Thread parallelization, vectorization and explicit memory locality are particularly critical for this architecture to achieve high performance. Vector instruction plays an important role in Xeon Phi, and loop dependence should be dismissed in order to use them, otherwise, performance will be punished. In general, memory optimizations should be explicaed such as using shared memory, constant memory etc. To benefit GPU applications a lot, bank conflicts should be avoided to get higher practical bandwidth. In future, we will evaluate our distributed 3D RTM-TTI algorithm and analysis communications.

References

[1] Alkhalifah T. An acoustic wave equation for anisotropic media. *Geophysics*, 2000, 65(4):1239-1250
[2] Zhang H, Zhang Y. Reverse time migration in 3D heterogeneous TTI media. In: Proceedings of the 78th Society of Exploration Geophysicists Annual International Meeting, Las Vegas, USA, 2008. 2196-2200
[3] Baysal E, Kosloff D D, Sherwood J W. Reverse time migration. *Geophysics*, 1983, 48(11):1514-1524
[4] Zhou H, Zhang G, Bloor B. An anisotropic acoustic wave equation for modeling and migration in 2D TTI media. In: Proceedings of the 76th Society of Exploration Geophysicists Annual International Meeting, San Antonio, USA, 2006. 194-198
[5] Gschwind M, Hofstee H P, Flachs B, et al. Synergistic processing in cell's multicore architecture. *IEEE Micro*, 2006, 26(2):10-24
[6] NVIDIA Cooperation, NVIDIA's next generation cuda compute architecture: Fermi. http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf, White Paper, 2009
[7] Intel Cooperation, Intel Xeon Phi coprocessor system software developers guide. https://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-system-software-developers-guide.html, White Paper, 2014
[8] Micikevicius P. 3D finite difference computation on GPUs using CUDA. In: Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units, Washington, D.C., USA, 2009. 79-84
[9] Araya-Polo M, Cabezas J, Hanzich M, et al. Assessing accelerator-based HPC reverse time migration. *IEEE Transactions on Parallel and Distributed Systems*, 2011, 22(1):147-162
[10] Heinecke A, Klemm M, Bungartz H J. From GPGPU to many-core: NVIDIA Fermi and Intel many integrated core architecture. *Computing in Science & Engineering*, 2012, 14(2): 78-83
[11] Zhou H, Ortigosa F, Lesage A C, et al. 3D reverse-time migration with hybrid finite difference pseudo spectral method. In: Proceedings of the 78th Society of Exploration Geophysicists Annual Meeting, Las Vegas, USA, 2008. 2257-2261

Zhang Xiuxia, born in 1987, is a Ph. D candidate at Institute of Computing Technology, Chinese Academy of Sciences. Her research includes parallel computing, compiler and deep learning.