An MPI-CUDA Implementation for the Compression of DEM

Zeng Fei
Wuhan University
61363 Troops
Wuhan, China

Chen Yumin
Wuhan University
Wuhan, China
Tony2fish@163.com

Abstract—**A high performance terrain data compression method is proposed based on discrete wavelet transform (DWT) and parallel run-length code. But the implementation of the schemes to solve these models in realistic scenarios imposes huge demands of computing power. Compute Unified Device Architecture (CUDA) programmed, Graphic Processing Units (GPUs) are rapidly becoming a major choice in high performance computing. Hence, the number of applications ported to the CUDA platform is growing high. Message Passing Interface (MPI) has been the choice of high performance computing for more than a decade and it has proven its capability in delivering higher performance in parallel applications. CUDA and MPI use different programming approaches but both of them depend on the inherent parallelism of the application to be effective. In this approach, MPI functions as the data distributing mechanism between the GPU nodes and CUDA as the main computing engine. This allows the programmer to connect GPU nodes via high speed Ethernet without special technologies. We tackle the acceleration of the compression of digital elevation models (DEM) by exploiting the combined power of several CUDA-enabled GPUs in a GPU cluster. This implementation overlaps MPI communication with CPU-GPU memory transfers and GPU computation to increase efficiency. Several numerical experiments, performed on a cluster of modern CUDA-enabled GPUs, show the efficiency of the distributed solver. Our speed-up was over 20 compared to two thread CPU version.**

I. INTRODUCTION

With the development of technology of remote sensing, the size and resolution of DEM has quickly increased. Today, spatial resolutions as high as 1m are available for some areas, and data at only slightly lower resolution is currently being acquired for ever larger regions worldwide. The storage and transmission of high-resolution elevation information can consume amounts of resources, and with the increased interest in mapping the earth and having maps for real time navigation, the development of compression techniques to help in these tasks is becoming very important.

At first people use the traditional Fourier transform, although the transform is simple and fast in data compression, but it also loss a lot of information of the data. Therefore, there need fast lossless compression in order to satisfy application requirement. Different compression schemes have been developed to exploit this property. Wavelet transform have been applied for image lossless compression or at least near-lossless way which can be used for image compression [1]. Since wavelet transform possesses resolution both in space domain and in frequency domain, it is very suitable for dealing with the data with instability, less relativity or less redundancy. DEM data inheres in the characteristics of instability, fragmentation and less relativity, here by discrete wavelet transform is rather suitable for DEM data compression [2, 3, 4]. However, for the huge data, the traditional application of wavelet transform is less considering parallelism, which will cause long time for transform. The GPU is an attractive platform for a broad field of applications, because it still remains a significant high arithmetic processing capability and is often less utilized. Therefore it can be used as a powerful accelerator without extra cost [5]. These platforms make it possible to achieve speedups of an order of magnitude over a standard CPU in many applications and are growing in popularity [6, 7].

Moreover, several programming toolkits such as CUDA [8] have been developed to facilitate the programming of GPUs for general purpose applications. There are previous works to compress data by using a graphics-specific programming language, but currently most of the proposals to compression on a single GPU are based on the CUDA programming model. Although the use of single GPU systems makes it possible to satisfy the performance requirements of several applications, many applications require huge meshes, large numbers of time steps and even real time accurate predictions. These characteristics suggest to combine the power of multiple GPUs.

One approach to use several GPUs is based on programming shared memory multi-GPUs desktop systems. These platforms have been used in fluid dynamic [9] and shallow water [10, 11] simulations by combining shared memory programming primitives to manage threads in CPU and CUDA to program the GPU. However, this cost-effective approach only offer a reduced number of GPUs (2-8 GPUs) and more flexible systems are desirable. A more flexible approach involves the use of clusters

of GPUs-enhanced computers where each node is equipped with a single GPU or with a multi-GPUs system. The computation on GPUs clusters could make it possible to scale the runtime reduction according to the number of GPUs. Thus, this approach is more flexible than using a multi-GPUs desktop system and the memory limitations of a GPUs-enhanced node can be overcome by suitably distributing the data among the nodes, enabling us to simulate significantly larger realistic models and with greater precision. The use of GPUs clusters to accelerate intensive computations is gaining in popularity [12, 13, 14, 15, 16, 17]. Most of the proposals to exploit GPUs clusters use MPI [18] to implement inter process communication and CUDA [19] to program each GPUs. A common way to reduce the remote communication overhead in these systems consists in using non-blocking communication MPI functions to overlap the data transfers between nodes with GPUs computation and CPU-GPU data transfers.

In this work, an implementation of DWT compression to the raster DEM is developed for a GPUs cluster by using MPI and CUDA. The outline of the article is as follows: Section 2 summarizes the background to DWT compression and provides an introduction to MPI-CUDA. In Section 3 we provide the main details of our parallelization strategy for the DWT compression using MPI-CUDA. Experimental results are analyzed in Section4. Finally, Section 5 summarizes the work and concludes the paper.

II. THE COMPRESSION OF DEM

A. DWT by CUDA

The DWT of image signals produces a non-redundant image representation, which provides better spatial and spectral localization of image information as compared to other multi-resolution representation [20]. For an input represented by a list of 2n numbers, the Discrete wavelet transform may be considered to simply pair up input values, storing the difference and passing the sum. This process is repeated recursively, pairing up the sums to provide the next scale: finally resulting in 2n−1 differences and one final sum. The describe of the DWT is in Eq.1 and Eq. 2. $a_{j-1}(n)$ is the original pending signal data; $a_j(n)$ is the approximation coefficient; $d_j(n)$ is the accurate coefficient; h_0 is the low-frequency filter; h_1 is the high-frequency filter.

$$a_j(n) = \sum_{n=-\infty}^{\infty} a_{j-1}(n) h_0(n-2k) \qquad (1)$$

$$d_j(n) = \sum_{n=-\infty}^{\infty} a_{j-1}(n) h_1(n-2k) \qquad (2)$$

The DWT for an image as a 2D signal will be obtained from 1D DWT. We get the scaling function and wavelet function for 2D by multiplying two 1D functions. This may be represented as a four channel perfect reconstruction filter bank as shown in Fig. 1. Now, each filter is 2D with the subscript indicating the type of filter (HPF or LPF) for separable horizontal and vertical components. By using these filters in one stage, an image is decomposed into four bands. There exist three types of detail images for each resolution: horizontal (HL), vertical (LH), and diagonal (HH).

The operations can be repeated on the low low (LL) band using the second stage of identical filter bank. Thus, a typical 2D DWT, used in image compression, generates the hierarchical structure shown in Fig. 2.

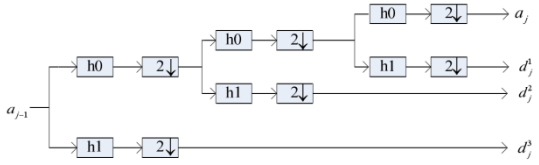

Fig. 1. One Filter Stage in 2D DWT

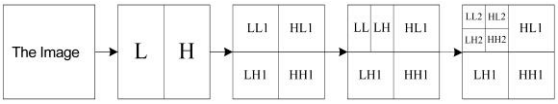

Fig. 2. Structure of wavelet decomposition

B. Rarallel Run-length Code by CUDA

This section presents the parallel VLE (Virtual Learning Environment) algorithm for General Purpose GPU with hardware support for atomic operations. The parallel variable-length encoding consists of the following parallel steps: (1) assignment of code words to the source data, (2) calculation of the output bit positions for compressed data (code words), and finally (3) writing (storing) code words to the compressed data array. In the first step, variable-length code words are assigned to the source data. The code words can be either computed using an algorithm such as Huffman, or they can be predefined, e.g. as it is frequently the case in image compression implementations. Without loss of generality, we can assume that the code words are available and stored in a table. This structure will be denoted as the code word look-up table (code word LUT). Each entry in the table contains two values: the binary code for the code word, and code word length in bits, denoted as a (cw, cwlen) pair. Our implementation uses an encoding alphabet of up to 256 symbols, with each symbol representing one byte. During compression, each source data symbol (byte) is replaced with the corresponding variable-length code word.

C. MPI-CUDA

NVIDIA SLI technology can be used to connect multiple GPUs that are in one computer and as of the latest release of the CUDA sdk, all those SLI (Scalable Link Interface) connected GPU cards can only be seen as one single GPU by the programmer. But we can connect GPU cards in different computers using Ethernet and exploit CUDA+MPI model so that it enables the user to see different GPUs in different computers as separate processing engines. Hence the programmer can execute different kernels in one application on different GPUs at the same time.

CUDA is the programming language provided by NVIDIA to run general purpose applications on NVIDIA GPUs. The CUDA incorporates an Application Programmer Interface, a runtime, couple of higher level libraries and a device driver for the underline GPU.

MPI provides a standard set of subprogram definitions which allow parallel programs to be written using a distributed memory programming model to allow more than one process to perform computations on a given set of data copies of this data must be sent to any process which requires it (to be saved on that process's memory). This is referred to as message passing.

D. System design

We successively overlap computations with inter-node and intra-node data exchanges to better utilize the cluster resources. All the implementations have much in common, with differences in the way data exchanges are implemented. We show that implementation details in the data exchanges have a large impact on performance.

For all implementations, one MPI process is started per GPU. Since we must ensure that each process is assigned a unique GPU identifier, an initial mapping of hosts to GPUs is performed. A master process gathers all the host names, assigns GPU identifiers to each host such that no process on the same host has the same identifier, and scatters the result back. At this point the cuda SetDevice()call is made on each process to map one of the GPUs to the process which assures that no other process on the same node will map to the same GPU.

These kernels implement the computation steps of the solver in the GPU, and do not require any modification for use in the multiple GPU implementation. We added the use of constant memory to support runtime model configuration while maintaining efficient GPU memory accesses to this common data. A temperature kernel was added and the momentum kernel changed to apply the buoyancy effect.

III. RESULTS AND DISCUSSION

A. Experimental Platform

All experiments are done using both CPU and *MPI-CUDA*. The configurations for them are listed in Table 1 and Table 2. The experiment requires some software and tools for programming and documenting purpose: CUDA SDK 4.2\VS 2010\ Nsight Visual Studio Edition 2.2.

TABLE I. DEVICE CONFIGURATION AT EXPERIMENT

Feature	Specification
Name	GeForce GT 650M
CUDA Driver Version	4.2
MPI	MPICH2-1.0.1
Total Global memory	2048M
#Multiprocessor	2
#Cores	384

TABLE II. TABLE 2: HOST MACHINE CONFIGURATION AT EXPERIMENT

Feature	Specification
System Model	Y480 Notebook
Operating Systerm	Windows xp
Manufacture	Lenovo
Processor	Intel(R) Core(TM) i5-3210M
Memory	4096MB RAM

B. Experimental Data

For test the different performance of the CUDA-based DWT compression, we use srtm_57_05 images (Fig.3) ranging from 256*256 to 6000*6000.

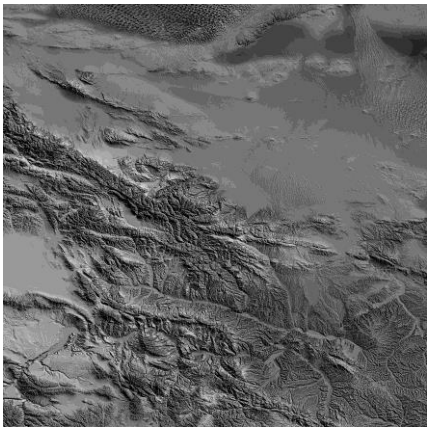

Fig. 3 srtm_57_05 Image

C. Experimental Result

To evaluate the proposed method, we examined each execution time of the DWT on a CPU and *MPI-CUDA*. The program for the CPU is implemented in C++ language. Every image makes the DWT on a CPU and *MPI-CUDA*. The execution time of the GPU is much less than the time taken on the CPU: we can see that when the image is small (256*256) the time spends on the both devices is almost the same. It is due to the fact that the data transfer overhead (from CPU to *MPI-CUDA* and vice versa) in case of small image size mitigating the parallel execution effects. For larger data sizes, the speedup obtained by the parallel operations take over the data transfer overheads and hence the performance gain becomes more obvious. With resolution's increase, the *MPI-CUDA*'s accelerator is more obvious (from 1.2 to 20.8). The speed-up ratio increase, when the image size becomes larger. It shows that the larger image the

more efficiency of the *MPI-CUDA* DWT compression. So the experiment has proved the *MPI-CUDA* can be used to improving the speed of DWT compression to dem. By using the GPU for decoding, the CPU is free for other tasks like prefetching and data management. We validated these statements by integrating the compression and decoding schemes into a terrain rendering system, and we showed that high visual quality on high-resolution displays is possible at interactive frame rates.

D. Discussion

In this paper, we present a *MPI-CUDA* accelerated DWT compression for images method using the CUDA and MPI. We realized significant improvement in runtime (speedup to 20.8), with imperceptible degradations of quality. Although this experiment has verified that *MPI-CUDA* parallel computation on GPUs significantly increases the speed of the discrete wavelet transform, much work remains to be done. The DWT method based on CUDA has many places to improve, for example, using the texture memory (it will be fast) to translate the data. Moreover, the DWT compression is only one step of DEM processing, the method should be used in some DEM processing, such as DEM realtime display. By using the GPU for decoding, the CPU is free for other tasks like prefetching and data management. We will validate these statements by integrating the compression and decoding schemes into a terrain rendering system, and show that high visual quality on high-resolution displays is possible at interactive frame rates.

ACKNOWLEDGMENT

The research is supported by the National Nature Science Foundation of China (No: 41171347).

REFERENCES

[1] Wan Gang, Zhu Changqiang, 1999. “Application of multi-band wavelet on simplifying DEM with lose of feature information”. Acta Geodaetica et Cartographica Sinica, 28 (1): 36-40p.

[2] Wu Lun, Liu Yu, Zhang Jing, 2001. “Principle of geographic Information system”. Beijing: Science Press, in press.

[3] Chang Zhanqiang, Wulixin. Montanic, 2004, “grid DEM data compression based on wavelet transform and mixed entropy coding”. Geography and Geo-Information, 20(1): 24-27p.

[4] NVIDIA Tutorial at PDP08, “Cuda: A new architecture for computing on the gpu”, 2008.

[5] J.D. Owens, S. Sengupta, and D. Horn, “Assessment of Graphic Processing Units for Department of Defense Digital Signal Processing Applications”, Technical Report, ECE-CE-2005-3, Computer Engineering Research Laboratory, University of California, Davis, October 2005.

[6] M. Rumpf, R. Strzodka, 2005, “Graphics Processor Units: New Prospects for Parallel Computing”, in: Numerical Solution of Partial Differential Equations on Parallel Computers, Vol.51 of Lecture Notes in Computational Science and Engineering, Springer, 89–134p.

[7] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips, 2008, GPU computing, Proceedings of the IEEE 96 (5), 879–899p.

[8] NVIDIA Corporation, NVIDIA CUDA C Programming Guide 3.2, 2010.

[9] W.-Y. Liang, T.-J. Hsieh, M. T. Satria, Y.-L. Chang, J.-P. Fang, C.-C. Chen, C.-C. Han, 2009, “A GPU-Based Simulation of Tsunami Propagation and Inundation”, in: Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP ’09, Springer-Verlag, Berlin, Heidelberg, 593–603p.

[10] J. Thibault, I. Senocak, 2010, “Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU platforms”, The Journal of Supercomputing, 1–27p.

[11] M. Geveler, D. Ribbrock, S. Mallach, D. G, 2010, “A Simulation Suite for Lattice-Boltzmann based Real-Time CFD Applications ExploitingMulti-Level Parallelism on modern Multi- and Many-Core Architectures”, Journal of Computational Science In Press, Accepted Manuscript.

[12] M. L. Saetra, A. R. Brodtkorb ,2010, “Shallow water simulations on multi-ple GPUs”, Proceedings of the Para 2010 Conference, Lecture Notes in Computer Science.

[13] D. A. Jacobsen, J. C. Thibault, I. Senocak, 2010, “An MPI-CUDA Implementa-tion for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters”, in: The 2009 High Performance Computing & Simulation- HPCS’09.

[14] D. Komatitsch, G. Erlebacher, D. G ̈oddeke, D. Mich ̈ea, 2010, “High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster”, J. Comput. Phys. 229, 7692–7714p.

[15] D. Komatitsch, 2011, “Fluid-solid coupling on a cluster of GPU graphics cardsfor seismic wave propagation”, High Performance Computing, 125–135p.

[16] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover, 2004, “GPU Cluster for High Performance Computing”, in: Proceedings of the 2004 ACM/IEEE con-ference on Supercomputing, SC’04, IEEE Computer Society.

[17] Y. Zhang, F. Mueller, X. Cui, T. Potok, 2011, “Data-intensive document clustering on graphics processing unit clusters”, J. Parallel Distrib. Comput. 211–224p.

[18] R. Abdelkhalek, H. Calendra, O. Coulaud, J. Roman, G. Latu, 2009, “Fast Seis-mic Modeling and Reverse Time Migration on a GPU Cluster”, in:The 2009 High Performance Computing & Simulation-HPCS’09, Leipzig Allemagne, Best Paper Award at HPCS’09 Total.

[19] NVIDIACorporation,CUDAZone,http://www.nvidia.com/object/cudahome new.html.

[20] M.Atallah, S.Kosaraju, L.Larmore, G.Miller, S.Teng, 1989, Constructing trees in parallel. In: Proceedings of the first annual ACM symposium on Parallel algorithms and architectures, pp. 421–431. ACM, New York.