Computers and Mathematics with Applications 72 (2016) 1076–1095

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Preconditioned GMRES solver on multiple-GPU architecture

Bo Yang *, Hui Liu, Zhangxin Chen

University of Calgary, Canada

ARTICLE INFO

Article history:
Received 20 September 2015
Received in revised form 2 May 2016
Accepted 18 June 2016
Available online 12 July 2016

Keywords:
GMRES
Large-scale sparse linear system
Parallel computing
GPU
Preconditioner

ABSTRACT

In this paper, we analyze the preconditioned GMRES algorithm in detail and decompose it into components to implement on multiple-GPU architecture. The operations of vector updates, dot products and Sparse Matrix–Vector multiplication (SpMV) are implemented in parallel. In addition, a specific communication mechanism for SpMV is designed. The preconditioner is established on the host (CPU) and solved on the devices (GPUs). Validated by a series of numerical experiments, the GPU-based GMRES solver is effective and favorable parallel performance is achieved.

1. Introduction

Numerical simulations have been widely used in many scientific and industrial fields, such as reservoir simulations, finance, and bio-computing. Ultimately, most of them need to solve sparse linear systems to obtain numerical solutions. The process of solving such linear systems is always time consuming. For example, over 70% of time is spent on the solution of linear systems derived from the Newton methods for the black oil model in reservoir simulation [1]. In addition, the solution of linear systems requires more simulation time when geological models are large and highly heterogeneous. The coefficient matrices of large-scale sparse linear systems are nonsingular and they have two distinctive characteristics. The first one is that the size of linear systems is large. Many of them have millions of rows. The second one is the matrices from FEM (Finite Element Method), FVM (Finite Volume Method) and FDM (Finite Difference Method) are sparse, whose pattern is determined by equations, methods and meshes. Usually each cell of a mesh has limited neighbors and the average number of non-zero entries in each row is relatively small.

Iterative methods have been studied for many years. Krylov subspace methods are a strand of most commonly used iterative methods [2–4]. The GMRES (Generalized Minimal Residual) method was developed by Saad et al. and used for unsymmetric linear systems [2,3]. The BiCGSTAB (Bi-conjugate Gradient Stabilized) method is also widely used for the numerical solution of nonsymmetric linear systems, which was designed by H.A. van der Vorst [5,2]. Coupled with preconditioners, Krylov subspace methods have enhanced robustness and efficiency [2,6]. Many advanced preconditioners have also been developed, such as domain decomposition methods [7], multi-stage preconditioners [1,8,9], constrained pressure residual (CPR) preconditioners for black oil model and compositional model [10–12] and a fast auxiliary space preconditioning method (FASP) [9]. The ILU preconditioner has been deeply studied and extensively applied [2,3,13,6,4].

GPU (Graphics Processing Unit) computing has advanced greatly in various scientific applications based on its superiority over conventional CPU (Central Processing Unit) computing. For example, NVIDIA Tesla K80 GPU Accelerator, which is a two-GPU architecture, owns a peak performance of 2910 GFlops in double precision while a latest CPU, Intel Core i7-5960X

* Corresponding author.
E-mail address: yang6@ucalgary.ca (B. Yang).

http://dx.doi.org/10.1016/j.camwa.2016.06.027
0898-1221/

processor Extreme Edition, has only a typical peak performance of 385 GFlops [14,15]. In addition, the memory speed of a GPU accelerator is also much faster than that of a CPU architecture. Using the same example, NVIDIA Tesla K80 GPU accelerator has 480 GB/s but the Intel Core i7-5960X Processor Extreme Edition has only 68 GB/s [14,16]. Thereby it is possible to design an algorithm that is faster on a GPU architecture than on a CPU architecture. Zhang et al. studied dense matrix–matrix multiplication on GPU [17]. Krylov linear solvers have been developed on GPUs [4,6,18,12]. Naumov [19] and Chen et al. [20] studied parallel triangular solvers for GPUs which concentrate on solving triangular systems derived from linear solvers or preconditioners. Haase et al. developed a parallel AMG (Algebraic Multigrid) solver using a GPU cluster [21], which is a multi-GPU implementation of preconditioned Conjugate Gradient algorithm with an AMG preconditioner (PCG-AMG). Bolz, Buatois, Goddeke, Bell, Wang, Brannick, Stone also studied GPU implementation of algebraic multigrid solvers, and details can be found in the references [22–26]. NVIDIA provided a matrix format HYB (Hybrid of ELL and COO) for general matrices [27,28]. Saad et al. studied a JAD matrix for GPU computing [6,4]. Chen et al. designed a matrix format HEC (Hybrid of ELL and CSR) [29]. HEC is friendly to design GPU-based algorithms and it is used in this research.

We focus on studying the multiple-GPU architecture in a single node (in one computer), which contains a two-level parallelization. They are the coarse-grained level composed by GPUs and the fine-grained level formed by CUDA cores on each GPU. It is unavoidable to communicate among GPUs when designed some algorithms. Though the communication speed of PCIe is always high, its latency is a virtual speed bottleneck. Based on these characteristics, we design our algorithms and try to make the most use of multiple-GPU parallel capability and minimize its communication defect. A preconditioned GMRES algorithm can be divided into some components to implement, which are vector updates, dot products, SpMV, preconditioner setup and preconditioning operations [2]. The vector updates and dot products can be relatively easily decomposed into subtasks and executed on GPUs in parallel. The rest three operations are complex and crucial to the GMRES, which need specific design and implementation. For SpMV, we divide the original matrix and vector into domain matrices and domain vectors firstly. Each pair of domain matrix and domain vector is distributed onto a GPU. We established a specialized communication mechanism on the multiple-GPU architecture, through which domain vectors can exchange data with each other. A series of typical large-scale matrices are used to test the performance of the SpMV algorithm and favorable results are shown on our four-GPU workstation. For preconditioner setup, this is a more complicated process, which is very difficult to design to run in parallel. Fortunately, this process only needs to be executed only once before the outer loop of the GMRES algorithm begins, which occupies a small part of whole running time. We complete this setup process on the CPU because CPU is adept at dealing with complex logic. The ILU preconditioner can be implemented coupling with a RAS (Restricted Additive Schwarz) structure [30]. As mentioned above, the multiple-GPU architecture owns a two-level parallelization. Thus we design a two-level RAS to maximize the usage of GPU's parallel capacity, which is named nested RAS. The original preconditioner matrix are divided into small matrices by nested RAS. Then these matrices are factorized into lower and upper triangular matrices by ILU(0) because triangular matrices are very easy to be solved. After preconditioner setup, all the triangular matrices are distributed onto GPUs. The preconditioning operations are responsible for solving these matrices. We have developed a parallel triangular solver for GPU and the solution process can be implemented based on it [20]. Numerical experiments are employed to test the accomplished preconditioned GMRES algorithm. For each matrix, a set of parameter combinations is designed to evaluate the performance characteristics from various aspects. The results verifies the algorithm is effective and has good prospect in application.

The follows gives the layout of this paper: In Section 2, the implementation of algorithm components is presented firstly. Then the preconditioned GMRES algorithm mechanism is analyzed. In Section 3, a series of numerical experiments are performed to test the performance of the preconditioned GMRES on our four-GPU workstation. In Section 4, conclusions are provided.

2. GPU computation

2.1. Multiple GPUs

Fig. 1 gives an illustration of multiple-GPU architecture in a single node. It contains a host (CPU) and several devices (GPUs), which are all installed on the same motherboard. The CPU controls each device through an OpenMP thread independently. This architecture provides two levels of parallelization. The first level is formed by the devices. The original task is divided on the host and distributed onto each device. After the devices received their tasks, the second level plays a part. Each device contains hundreds or even thousands of cores. Each core works as an individual thread. Thus the task on each GPU can be executed by fine cores in parallel.

In our research, we select NVIDIA GPU. The processor of a typical NVIDIA GPU is categorized into two levels: a streaming multiprocessor (SM) and a streaming processor (SP or CUDA core). A GPU may have a different number of SMs. A SM consists of 32, 128 or 192 SPs due to concrete GPU architecture. For instance, the Tesla C2070 has 14 SMs and each SM contains 32 SPs. So the total CUDA cores on a GPU is 448. The Tesla K40 has 15 SMs and 192 SPs in each SM. It can provide 2880 CUDA cores running simultaneously, which is a very strong parallel capability. A NVIDIA GPU has a hierarchical memory architecture. It contains register, L1 cache, L2 cache, shared memory and global memory. The speed of shared memory is faster than that of the global memory but the size of it is much small. For instance, a multiprocessor has only 48 KB shared memory. Thereby it is mainly used for communication among threads in a block. We use the global memory to store our matrices