Computers & Fluids 92 (2014) 160–171

Contents lists available at ScienceDirect

Computers & Fluids

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

An efficient GPU implementation of cyclic reduction solver for high-order compressible viscous flow simulations

Vahid Esfahanian, Behzad Baghapour *, Mohammad Torabzadeh, Hossain Chizari

School of Mechanical Engineering, Engineering College, University of Tehran, North Kargar Ave., P.O. Box 14395-1335, Tehran, Iran

ARTICLE INFO

Article history:
Received 19 March 2012
Received in revised form 8 September 2013
Accepted 9 December 2013
Available online 24 December 2013

Keywords:
Cyclic reduction
GPU computing
High-order compact finite-difference scheme
Compressible viscous flow

ABSTRACT

In this paper, the performance of the Cyclic Reduction (CR) algorithm for solving tridiagonal systems is improved with the aid of efficient global memory transactions on Graphics Processing Units (GPU). To achieve maximum memory throughput with a lower computational runtime, two different *Sort* algorithms are introduced for reordering the initial system of equations: *direct* and *step-by-step*. It is shown that the latter method is well-fitted to modern GPUs and achieves speedup of up to 3.47× in single precision and 2.1× in double precision compared to the CPU Thomas algorithm. By benefiting from the new global memory implementation, the CR solver could run 2×–100× faster compared to previous works on parallel tridiagonal solvers. The CR solver is also applied to 2D & 3D compressible viscous flow simulations using the high-order compact finite-difference scheme. In this matter, the procedure of filtering, primitive variables, and flux derivative calculations are carried out by using the parallel tridiagonal solver on the GPU device. The GPU-accelerated calculations achieve speedups between 1.9×–15.2× in 2D and 6.4×–20.3× in 3D simulations for different grid sizes compared to CPU computations. The computations are performed on the NVIDIA GTX480 GPU. The obtained results are compared to those achieved on a single core of Intel Core 2 Duo (2.7 GHz, 2 MB cache) in terms of calculation runtime.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Over the past few years, the GPU has evolved into a processor with high performance of floating-point arithmetic operation and wide memory bandwidth. Massive hardware multi-threading by GPUs has experienced an order of magnitude more performance over a single CPU for CFD applications [1–4].

Solving tridiagonal systems is nearly a primary task in many CFD applications. It serves as a kernel for famous numerical algorithms like Alternating Direct Implicit (ADI) and Line Successive Over Relaxation (LSOR), which are required for solving the linear system of equations and preconditioners. The conventional method to solve tridiagonal systems is the sequential Thomas algorithm, which is based on Gaussian elimination. Two alternative parallel algorithms for the solution of tridiagonal equation systems are Cyclic Reduction (CR) [5] and Recursive Doubling (RD) [6]. These parallel algorithms have led many researchers to present various GPU implementations to boost computational performance.

Zhang et al. [7] first discussed the applicability of these algorithms on modern GPUs and suggested a hybrid technique amongst the Thomas algorithm, CR, Parallel CR (PCR) and RD with a comprehensive performance examination of each algorithm. They concluded that CR suffers from shared memory bank conflicts and poor thread utilization. Göddeke et al. [8] reported matrix size limitation due to shared memory usage. They declared that performing multiple transfers to global memory would significantly reduce the performance of the algorithm. Moreover, Davidson et al.[9] proposed a multi-stage method for solving large tridiagonal systems. Their GPU implementation was much slower than the CPU Thomas algorithm. Davidson and Owens [10] presented a register packing optimization method for improving the scalar tridiagonal CR solver. This method enabled them to solve larger systems and achieve higher performance compared to the results achieved by [7,8]. Egloff [11] presented a PCR implementation for solving large tridiagonal matrices over finite difference PDE solvers and reported 60% performance degradation in the case of using global memory. Kim et al. [12] proposed a hybrid of the tiled PCR and Thomas algorithms to overcome the size limitation of shared memory without causing extra data loads.

In GPU architecture, the fast on-chip shared memory has two orders of magnitude lower latency than global memory, however, it confronts memory limitation in large system of equations. This has led Davidson et al. [9] and Kim et al. [12] to split the initial large system into smaller ones on global memory, pass each segment into the local shared memory and then solve each piece of

* Corresponding author. Tel./fax: +98 21 88020741.
E-mail addresses: evahid@ut.ac.ir (V. Esfahanian), baghapor@ut.ac.ir (B. Baghapour), torabzadeh@ut.ac.ir (M. Torabzadeh), hossainchizari@yahoo.com (H. Chizari).

0045-7930/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compfluid.2013.12.011

the problem by the Stream Multiprocessors (SM) of the GPU device. However, their proposed algorithm did not reach a considerable speedup compared to the CPU Thomas algorithm.

A recent research on high-order simulation of fluid flow on GPU was carried out in [13]. They simulated the vortex advection on 3D Cartesian grids using sixth-order compact and tenth-order filtering scheme. On the GPU platform, the required flux derivatives for compact finite difference was obtained by inversion of the tridiagonal system once at the beginning of the simulation. Although this procedure results in faster computations, it suffers from lack of efficiency especially in large matrix sizes. This is related to the large share of memory allocation and explicit matrix inversion in the total runtime procedure. They concluded that employing an efficient parallel algorithm for solving tridiagonal systems has an important role in high-order numerical simulations.

In the present study, an efficient cyclic reduction solver that can be applied to CFD codes is developed. The current work uses a global memory-based algorithm which matches well with GPU architecture by proper thread utilization, computing intensive operations, and coalesced memory access in order to overcome some of the difficulties that existed in the previous hybrid and shared-memory-bound methods. This paper starts by briefly presenting the CR algorithm. Then, the GPU implementation of the CR algorithm is explained and the performance of the tridiagonal solver is analysed. Finally, the compressible viscous flow simulations around the NACA 0012 airfoil and the circular cylinder are considered to demonstrate the performance of the solver.

2. The cyclic reduction algorithm

CR consists of two phases: forward reduction and backward substitution. The algorithm flowchart is shown in Fig. 1.

1. *Forward reduction*
   This phase is based on a divide and conquer algorithm. It means that the initial $N \times N$ matrix splits into two independent tridiagonal matrices for two sets of odd and even unknowns in the first step. The two smaller matrices can be split again in the same way into two submatrices. For the final step, submatrices of two unknowns are produced. At this point, the solution of odd-indexed equations is trivial. The CR algorithm consists of $\log_2 N$ steps for a $N \times N$ tridiagonal matrix. The diagonals (a, b, and c) and right-hand-side (d) elements in each step of the algorithm can be obtained by the following formulas.

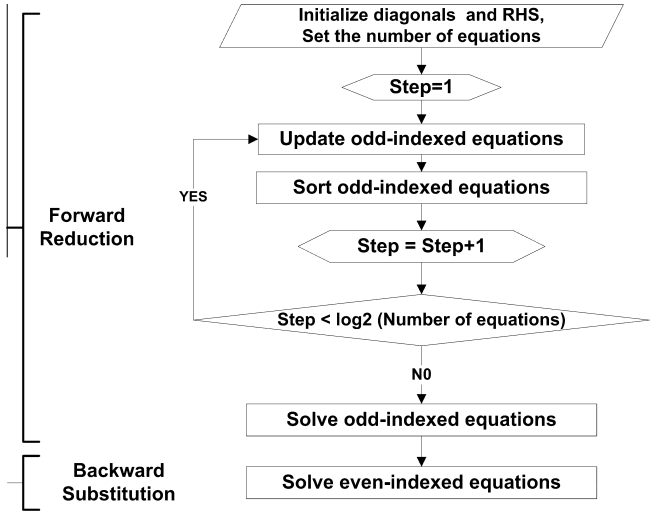

Fig. 1. CR algorithm flowchart.

$$
\begin{aligned}
d_i^{step+1} &= d_i^{step} - d_{i-1}^{step} k_1^{step} - d_{i+1}^{step} k_2^{step} \\
b_i^{step+1} &= b_i^{step} - c_{i-1}^{step} k_1^{step} - a_{i+1}^{step} k_2^{step} \\
c_i^{step+1} &= -c_{i+1}^{step} k_2^{step} \\
a_i^{step+1} &= -a_{i-1}^{step} k_1^{step} \\
k_1^{step} &= \frac{a_i^{step}}{b_{i-1}^{step}} \\
k_2^{step} &= \frac{c_i^{step}}{b_{i+1}^{step}}
\end{aligned}
\tag{1}
$$

2. *Backward substitution*
   The backward substitution phase determines the other half of the unknowns using the previously solved values. The following equation determines each even-indexed unknown (x_i) by substituting previously solved odd-indexed values (x_{i-1} and x_{i+1}).

$$x_i = (d_i - a_i x_{i-1} - c_i x_{i+1})/b_i \tag{2}$$

The cyclic reduction requires $17N$ arithmetic operations while the Thomas algorithm needs $8N - 1$ operations to solve a tridiagonal system of size N. Although CR costs $\frac{17}{8}$ times more than the Thomas algorithm in terms of computational operations for large tridiagonal systems, the present study indicates that the proposed CR solver runs faster than the Thomas solver on GPU devices.

3. GPU implementation of the CR algorithm

The proposed CR algorithms are implemented via Compute Unified Device Architecture CUDA [14], tested on NVIDIA GTX480 GPU and compared to Intel Core 2 Duo (2.7 GHz, 2 MB cache) CPU.

3.1. CUDA architecture and GPU memory hierarchy

In CUDA architecture, a kernel execution is distributed among an array of parallel threads, which are grouped in thread blocks. The thread blocks are then mapped to the stream multiprocessors (SM) of the GPU device [14]. The parallel threads can access data from a variety of memory resources during the kernel execution. Each thread can have low latency access to 48 KB of shared memory for inter-thread communication within a thread block. All threads of the grid have access to the large global memory with high transaction latency. The read-only memories visible to all threads are constant and texture memory spaces [14].

Each SM contains on-chip register memory resources, which are allocated to active threads of the kernel. The number of registers that are available per multiprocessor is fixed. The extra register memory consumption limits the number of threads that can be simultaneously mapped to the GPU multiprocessor during kernel execution, which results in memory latency [15]. Local memory is exploited when a kernel runs out of SM register memory resources. In case of high register pressure, the extra amount of registers is spilled in the private local memory of the thread [15]. However, the data access to off-chip local memory reduces the performance due to high latency and low bandwidth [16].

The NVIDIA GTX 480 used in this research is built upon the GF100 Fermi architecture. The 1536 MB of frame buffer memory runs through a 384-bit bus and delivers 177.4 GB/s of memory bandwidth. The GPU consists of 4 Graphics Processing Clusters (GPC) that house the 480 CUDA cores, 60 Texture units, 15 Streaming Multiprocessors (SM), and 48 Render Outputs (ROP). Every SM core in each GPC is comprised of 32 CUDA cores, with 48/16 KB of shared memory, 16/48 KB of L1, 4 texture units, and 1 PolyMorph Engine.

Download English Version:

https://daneshyari.com/en/article/761873

Download Persian Version:

https://daneshyari.com/article/761873

Daneshyari.com