Journal of Computational Physics 352 (2018) 246–264

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A GPU-accelerated semi-implicit fractional-step method for numerical solutions of incompressible Navier–Stokes equations

Sanghyun Ha, Junshin Park, Donghyun You [*]

Department of Mechanical Engineering, Pohang University of Science and Technology, 77 Cheongam-ro, Nam-gu, Pohang, Gyeongbuk 37673, Republic of Korea

ARTICLE INFO

Article history:
Received 6 February 2017
Received in revised form 25 September 2017
Accepted 26 September 2017
Available online 29 September 2017

Keywords:
GPU (Graphics Processing Unit)
CUDA (Compute Unified Device Architecture)
Navier–Stokes equations
Semi-implicit method
Tridiagonal matrix
Fast Fourier transform

ABSTRACT

Utility of the computational power of Graphics Processing Units (GPUs) is elaborated for solutions of incompressible Navier–Stokes equations which are integrated using a semi-implicit fractional-step method. The Alternating Direction Implicit (ADI) and the Fourier-transform-based direct solution methods used in the semi-implicit fractional-step method take advantage of multiple tridiagonal matrices whose inversion is known as the major bottleneck for acceleration on a typical multi-core machine. A novel implementation of the semi-implicit fractional-step method designed for GPU acceleration of the incompressible Navier–Stokes equations is presented. Aspects of the programing model of Compute Unified Device Architecture (CUDA), which are critical to the bandwidth-bound nature of the present method are discussed in detail. A data layout for efficient use of CUDA libraries is proposed for acceleration of tridiagonal matrix inversion and fast Fourier transform. OpenMP is employed for concurrent collection of turbulence statistics on a CPU while the Navier–Stokes equations are computed on a GPU. Performance of the present method using CUDA is assessed by comparing the speed of solving three tridiagonal matrices using ADI with the speed of solving one heptadiagonal matrix using a conjugate gradient method. An overall speedup of 20 times is achieved using a Tesla K40 GPU in comparison with a single-core Xeon E5-2660 v3 CPU in simulations of turbulent boundary-layer flow over a flat plate conducted on over 134 million grids. Enhanced performance of 48 times speedup is reached for the same problem using a Tesla P100 GPU.

1. Introduction

Reduction of computational time is a major challenge in numerical simulations of fluid flow. At high Reynolds numbers, the three-dimensional Navier–Stokes equations require a very large number of grid points to resolve broadband scales of interest. Particularly in a direct numerical simulation (DNS) of turbulent fluid flow, the computational grid is required to be dense enough to resolve the entire spectrum of turbulent scales in space and time. As a result, parallel computing based on multi-core Central Processing Units (CPUs) has long been used to overcome such computational cost and still remains as the mainstream methodology for solving large-scale problems [2,17]. Nevertheless even for moderate Reynolds numbers, DNS

[*] Corresponding author.
E-mail address: dhyou@postech.ac.kr (D. You).

https://doi.org/10.1016/j.jcp.2017.09.055
0021-9991/© 2017 Elsevier Inc. All rights reserved.

Table 1
Comparison of CPU and GPU used in the present study.

	Tesla K40c GPU	Xeon E5-2660 v3 CPU
Cache	1.5 MB[a]	**25 MB**[b]
Core frequency	745 MHz	**2.6 GHz**
Memory bandwidth	**288 GB/s**	68 GB/s
DP Peak throughput	**1430 GFlops**	416 GFlops
Computation units	15 SMX	10 cores
Performance per Watt	**6.09 GFlops/W**	3.96 GFlops/W

[a] GPU L2 cache shared by multiprocessors.
[b] CPU L3 cache shared by cores.

requires a few hundred million number of grid points and thereby calls for compute nodes occupying considerable footprint in terms of space and energy consumption.

Graphics Processing Units (GPUs) on the other hand, offer new opportunities for accelerating computational solutions of the Navier–Stokes equations. As shown in Table 1 in which distinct characteristics of two hardwares are compared, GPUs are generally characterized by energy-efficiency and high devotion to throughput and memory bandwidth rather than latency enhancement. In other words, GPU is more suited for handling large amount of data in parallel rather than fast processing of a single operation. Thus GPU computing can become an attractive candidate for large-scale problems, many of which can be formulated into massively parallel tasks. A comprehensive review by Niemeyer et al. [18] introduces a number of recent studies which have used GPUs to successfully accelerate flow solvers.

Although a GPU is known to have high compute throughput and memory bandwidth, it may not always deliver impressive performance gains depending on the numerical scheme used for temporal integration of Navier–Stokes equations. GPUs in general become effective when the given problem is decomposed into tasks operating on independent data set. Regarding the methods of time advancement, fully explicit schemes are examples with such data independence. For this reason, researchers have employed GPUs to accelerate flow solvers based on fully explicit temporal integration of compressible as well as incompressible Navier–Stokes equations [1,25,27].

In contrast to fully explicit schemes which are usually used for compressible flows, semi-implicit fractional-step methods with second-order finite-volume or finite-difference spatial discretization schemes [13] have been commonly used for solutions of wall-bounded incompressible flows. In this method, Navier–Stokes equations are integrated using a combination of explicit and implicit schemes for convective and viscous terms, respectively. The main advantage of the method for wall-bounded flows is that the implicit treatment of viscous terms allows a stable solution even at a larger time-step size, which is limited in fully explicit schemes due to the small grid size near the wall. Owing to stability and savings in computation time, this method has widely been used for solving incompressible Navier–Stokes equations [9,12,16,28].

Unfortunately, the semi-implicit fractional-step method has limited scalability due to the serial nature of its algorithms. Fractional-step methods in general divide the original three-dimensional incompressible Navier–Stokes equations into momentum equations for solving intermediate velocities, and a Poisson equation for correcting the velocities. When the viscous terms are discretized using a commonly used second-order central-difference scheme and are integrated implicitly, the resulting momentum equations require inversion of multiple tridiagonal matrices (TDMAs). A classic way of inverting TDMAs is the Thomas algorithm, which performs $O(n)$ operations; yet it is inherently difficult to parallelize. Similar reasoning applies to the Poisson equation; despite the efficiency of direct solution of matrices after Fourier transform in homogeneous directions and finite-difference discretization in the wall-normal inhomogeneous flow direction, this method has not received much attention from prior studies using GPUs because of its additional complexity coming from inverting TDMAs. Alfonsi et al. [1] used a modified version of Thomas algorithm on GPUs to invert multiple TDMAs in the Poisson equation, but parallelism was limited because each thread solved one linear system at a time. Deng et al. [6] implemented a TDMA solver for accelerating the Alternating Direction Implicit (ADI) method on GPUs, but parallelism was likewise limited in that each thread swept each line in the z-direction. For this reason, recent studies have simplified fractional-step methods by using fully explicit time integration when GPUs were adopted for acceleration [1,10].

A few studies have assessed the performance of semi-implicit fractional-step methods on parallel machines. Borrell et al. [2] developed a high-resolution DNS code for a solution of the flat-plate boundary layer under a zero pressure gradient up to $Re_\theta = 6800$ using a semi-implicit fractional-step method, which treated only the wall-normal viscous terms implicitly. In addition to Message Passing Interface (MPI), OpenMP was used for an additional level of parallelism, achieving weakly linear scalability up to 32768 cores of CPU. The authors mention about the difficulty in parallelization of the TDMA solver and fast Fourier transform (FFT) when using a hybrid MPI-OpenMP approach. Another semi-implicit fractional-step method using ADI for the Helmholtz-type momentum equations and partial diagonalization for the Poisson equation was implemented using MPI [29]. The implementation showed that the discretized momentum equations which consist of nine tridiagonal matrices (three for each velocity component) were the major source of reduced scalability. The work was further extended to a hybrid CPU–GPU environment [30] and for the same method, Thomas algorithm was ported onto GPUs. However, the approach exhibits coarse-grain parallelism, providing insufficient amount of work needed for effective GPU acceleration.

The present study proposes a new implementation of a semi-implicit fractional-step method coupled with ADI and Fourier-transform methods which is designed particularly for a GPU-accelerated computation of incompressible Navier–