11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

A CFD SOLVER ON GRAPHICAL PROCESSING UNITES FOR TURBULENCE SIMULATIONS

CAO WENBIN, LI HUA,TIAN ZHENGYU AND PAN SHA

College of Aerospace Science and Engineering
National University of Defense Technology
Changsha, 410073 Hunan, China
e-mail: caowenbin08@163.com

Key Words: *Graphics Processing Unit (GPU), Turbulence, Computational fluid dynamics.*

Abstract. The focus of the current paper is the development of a finite volume computational fluid dynamics solver on the Graphical Processing Unites (GPUs) using CUDA Fortran for turbulence simulations. The solver is implemented with an AUSM scheme for the spatial discretization and the SST κ-ω model for turbulence model. Several test cases, such as 2D NASA 0012 airfoil, a supersonic mixing layer, and hypersonic inlet flow are chosen to demonstrate the acceleration performance of GPU on multi-million grid cells. Results show that the computational expense can be reduced by 20-32 times when using a NVIDIA Tesla K20c GPU as compared to a single core of an Intel Xeon 2670 CPU.

1 INTRODUCTION

Graphics Processing Unit (GPU) computing technologies are nowadays a widely adopted choice to perform scientific and engineering computations[1-2]. GPUs are based on the stream processing architecture that is suitable for compute-intensive parallel tasks,such as fluid dynamics solutions. In 2010, the National University of Defense Technology in China created a new world's fastest supercomputer Tianhe-1A which epitomized modern heterogeneous computing by coupling massively parallel GPUs with multi-core CPUs,enabling significant achievements in performance,size and power.

Modern GPUs can access memory bandwidth and floating-point performances that are orders of magnitude faster than a standard CPU. However, to run on GPUs with high performance, existing codes for CPU need to be redesigned and optimized, a procedure which must take care of the GPU hardware characteristics. For eliminating the barriers to use of GPUs for general-purpose computing, NVIDIA released CUDA (Compute Unied Device Architecture) in 2007, allowing the scientic community to apply NVIDIA GPUs to large complex problems, reducing the computational cost of these numerical simulations.The algorithms for CUDA-enabled devices are very different from the traditional parallel algorithms used in MPI or OpenMP protocols. To create an effective GPU algorithm one has to take into account the following. The GPU computations are carried out simultaneously by a large number of threads which are grouped into thread blocks. A thread block is executed on a multiprocessor. The number of multiprocessors on GPU depends on GPU model and it is of

the order of hundreds for modern GPUs.

The scope of this work is to show the potential of GPU devices for the computation of steady, three-dimensional turbulent flows. Specifically, we describe the porting and optimization of an highly efficient compressible finite-volme code, on CUDA programming model.

2 GOVERNING EQUATIONS

The governing equations for two-dimensional compressible viscous flow neglecting the source term can be written in terms of generalized coordinates as

$$\frac{\partial \tilde{Q}}{\partial t}+\frac{\partial(\tilde{F}-\tilde{F}_v)}{\partial \xi}+\frac{\partial(\tilde{G}-\tilde{G}_v)}{\partial \eta}=S \tag{1}$$

the vector of conserved variables consists of the following four components

$$\tilde{Q}\equiv\frac{Q}{J}=\frac{1}{J}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho E\end{bmatrix} \tag{2}$$

Where J represents the Jacobian of the transformation between the Cartesian variables and the generalized coordinates, ρ is the density, u,v,w are the velocity component in Cartesian coordinates system, E is the total energy per unit mass and S is turbulent source terms .

The vector of convective fluxes is

$$\tilde{F}=\frac{1}{J}\begin{bmatrix}\rho U\\ \rho uU+\xi_x p\\ \rho vU+\xi_y p\\ \rho HU\end{bmatrix} \tag{3}$$

with the static pressure P and the contravariant velocity U given by

$$U=\xi_x u+\xi_y v \tag{4}$$

The total enthalpy H is given as

$$H=h+\frac{|v|^2}{2}=E+\frac{p}{\rho} \tag{5}$$

The vector of viscous fluxes is

$$\tilde{F}_v=\frac{1}{J}\begin{bmatrix}0\\ \xi_x\tau_{xx}+\xi_y\tau_{xy}\\ \xi_x\tau_{yx}+\xi_y\tau_{yy}\\ \xi_x\Theta_x+\xi_y\Theta_y\end{bmatrix} \tag{6}$$

where

$$\begin{aligned}\tau_{xx}&=-\frac{2}{3}\mu\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+2\mu\frac{\partial u}{\partial x}\\ \tau_{yy}&=-\frac{2}{3}\mu\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+2\mu\frac{\partial v}{\partial y}\\ \tau_{xy}&=\tau_{yx}=\mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\end{aligned} \tag{7}$$

in which μ denotes the dynamic viscosity coefficient and

$$\Theta_x = u\tau_{xx} + v\tau_{xy} + k\frac{\partial T}{\partial x}$$
$$\Theta_y = u\tau_{yx} + v\tau_{yy} + k\frac{\partial T}{\partial y} \tag{8}$$

T represents the static temperature k anddenotes the thermal conductivity coefficient.

In order to close the equation, the calorically perfect gas assumption is adopted, for which the equation of state takes the form

$$p = \rho RT \tag{9}$$

where denotes the specific gas constant. The laminar dynamic viscosity can be obtained by the Sutherland formula. The result for air is

$$\frac{\mu}{\mu_0} \approx \left(\frac{T}{T_0}\right)^{1.5} \left(\frac{T_0 + T_s}{T + T_s}\right) \tag{10}$$

with $\mu_0 = 1.7161\times10^{-5}\,\mathrm{kg/(m\cdot s)}$, $T_0 = 273.16\mathrm{K}$, $T_s = 124\mathrm{K}$. The relationship

$$\kappa = \frac{\mu c_p}{Pr} \tag{11}$$

is generally used for air, where denotes the laminar Prandtl number. In addition, it is commonly assumed that the Prandtl number and the specific heat ratio are constant in the entire flow field. For air $Pr = 0.72$ and $\gamma = 1.4$.

For turbulence flow,the shear stress terms τ_{ij} is composed of a laminar and a turbulent component as

$$\tau_{ij} = \tau_{ij}^{\ L} + \tau_{ij}^{\ T} \tag{12}$$

For SST κ-ω model,the turbulence N-S equations are identical to identical to the laminar equations with the exception that μ is replaced by $\mu + \mu_T$ and $\frac{\mu}{pr}$ is replaced by $\frac{\mu}{pr} + \frac{\mu_T}{pr_T}$,where μ_T is the eddy viscosity value and the turbulent Prandtl number $pr_T = 0.9$.

For the explicit scheme, the time derivative is approximated by a forward difference and the residual is evaluated at the current time level. Thus, a new solution is given by the relation

$$Q^{n+1} = Q^n - \frac{\Delta t}{V} RHS^n \tag{13}$$

where the superscripts n and (n+1) denote the current and next time level respectively, means the cell volume and is the residual function, represents the local time step evaluated from

$$\Delta t = CFL\frac{V}{(\rho_A^c + \rho_B^c) + 2(\rho_A^v + \rho_B^v)} \tag{14}$$

In which CFL denotes the CFL number, ρ_A^c, ρ_B^c represent the inviscid spectral radii and ρ_A^v, ρ_B^v are the viscous spectral radii.

The AUSMpw+ scheme[3] is implemented in spatial discretization. For second order accuracy, the primitive variable interpolation of MUSCL approach with Van Albada limiter is used.

3 GPU CODE OPTIMIZATION

In the CPU source code, flow variables are stored in four-dimensional arrays where the first component steps through primitive variables and the other components refer to spatial coordinates, i.e. Q(7,i,j,k) stands for the turbulent kinetic energy at the spatial coordinate indexes i,j and k. This is a good approach for the CPU version program to optimize the cache usage, because the processor execute sequential access to all the variables at the same grid point. Another possible alternative is to have the index for the primitive variables as the last component, i.e. Q(i,j,k,7), which could be more convenient when executing finite difference derivative for the first component since it involves values residing in contiguous memory locations. When porting the CPU code to CUDA, the second alternative is largely preferable as we are going to clarify.

In CUDA architecture, global memory accesses are scheduled on each multiprocessor by threads of a half-warp (for devices of compute capability 1.x) or of a warp (for devices of compute capability 2.0 and higher). The best performance is achieved when threads in a warp (or half-warp) access data in as few memory transactions as possible. For optimizing to only one transaction when reading or writing data, CUDA kernels must perform coalesced memory accesses to global memory.This leads to neighboring threads in a warp have to be mapped to contiguous memory regions ,thus we suggesting a layout with grid coordinate indexes as the three first indexes.

Another important optimization is minimizing the number of divergent warps. Because a warp of threads executes concurrently, if a branch of a conditional is satisfied by any thread in a warp, all threads in a warp must execute that branch. The various execution paths are serialized and the instruction count per warp increases accordingly. If there are many branches to an if or case construct or multiple levels of nesting of such control flow statements, warp divergence can become a problem. On the other hand, if a condition evaluates uniformly over a warp of threads, then at most a single branch is executed. Therefore,in our CUDA FORTRAN code, all branch statements in AUSMpw+ sheme have been removed by the sign function, for example:

$$\begin{array}{l} if \quad Ma > 0 \quad V = U_L \\ if \quad Ma \le 0 \quad V = U_R \end{array} \text{ replaced by } V = 0.5\,[1 + sign(Ma)]U_L + 0.5\,[1 - sign(Ma)]U_R$$

The registers in a Stream Multiprocessor (SM) are the fastest memories on a GPU. However they are distributed to all the threads of the blocks loaded on this SM at the same time and then a register becomes private for the thread it is attached to. The more the threads have been assigned to this SM, the less the number of registers per thread are available. This small amount of registers available per thread is a strong limitation. If this amount is overpassed by the request of the code executed in the thread, the contains of the registers are spilled in local (or global) memory inducing a high latency. In the GPU code, the number of intermediate variables which consume a large amount of registers should be redused as few as possible. Moreover, we can make the best use of the shared memory to relieve the register pressure and optimize global memory coalescing.

4 GPU ACCELERATION

An important aspect of GPU programming is the speed-up, or computational speed increase, when compared to similar CPU codes. This section compares the speeds of the CFD solver on GPU and CPU. To compare the two devices, only one CPU core will be run, and compared to one entire GPU. The CPU code is written in FORTRAN, while the GPU code is in CUDA FORTRAN. Both codes are compiled with the same optimization level to ensure best performance for comparison. In all cases, the CPU used is an Intel Xeon-E5 2670 running at 2.6 GHz without hyper threading. A Tesla K20c GPU is employed for the cases, and uses 32 bits floating points instructions (single precision) with ECC off.

4.1 Nasa 0012 airfoil

The first test case for this paper is the supersonic flow over a nasa 0012 airfoil. The airfoil length is 1.0m. The computational grids is C type with size A (128 × 64 × 5) and size B (2048 × 512 × 5). Mesh spacing for the first cell above the body surface is 1.0×10^{-5} m. The free stream conditions as follows: calorically perfect gas, temperature 200.0K , Mach number 3.0, density 0.1 kg/m3. The computational conditions as follows: angle of attack 5.0, the wall temperature 300.0K. Note that we adopt 3D computations to assess the computational performance of GPU for all the computational problems. For grid size A, GPU's runtime per step is 5.56×10^{-3} s and CPU is 0.113. For grid size B, GPU's runtime is 0.138s and CPU is 4.3s. GPU version code performs 20.3 and 31.2 times faster than the CPU code with the two size grids. We can see that GPU give poor performance for small grid size. Fig.1 presents the pressure field and streamlines around the airfoil.

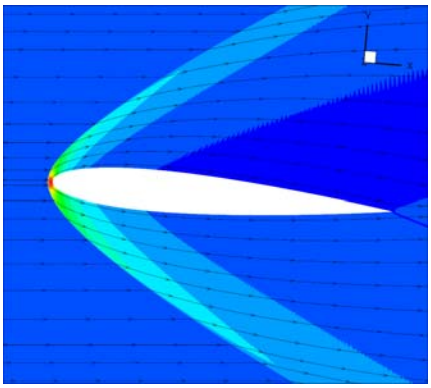

Figure 1: pressure field of NASA 0012 airfoil for grid size B

4.2 Supersonic mixing layer

The compressible mixing layer is the flow deriving from the turbulent mixing of two streams with different velocities. The flow experiments have been accomplished by Goebel and Dutton[4]. In our numerical test, the computational domain is discretized with a Cartesian grid consisting of 4000 × 1124 × 5 points,which are uniformly spaced in the streamwise and lateral direactons. For this case, the GPU's runtime per step is 0.566s and CPU is 17.89s, showing 31.6 × speedup for GPU. An overview of the flow development can be gained by Fig.2, where an instantaneous field of vorticity magnitude is report in a longitudinal plane.

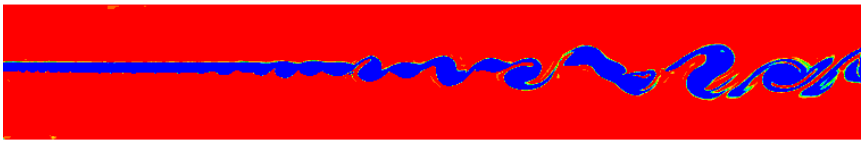

Figure 2: mixing layer instantaneous field of vorticity magnitude

4.3 Hypersonic inlet flow

The investigated inlet model is a two-dimensional conguration with a width-to-height ratio of 3.5. A detailed wind-tunnel experiment test data is given by Herrmann and Koschel[5]. The computational conditions as follows: inflow Mach numnber 2.41, angle of attack -10.0, the wall temperature 295.0K.Grid size is $3600 \times 1280 \times 5$. For this case, the GPU's runtime per step is 0.572s and CPU is 18.47s , attaining 32.3 × speedup for GPU. Fig.3 shows the Mach number field in the inlet.

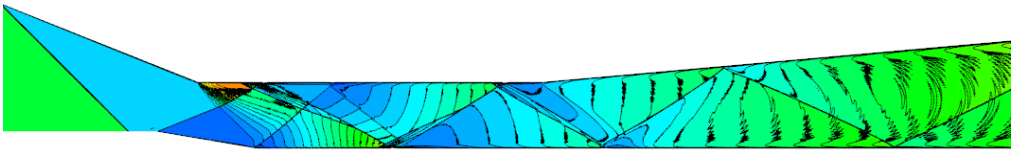

Figure 3: Mach number field of hypersonic inlet flow

5 CONCLUSIONS

We have demonstrated GPU accelerated implementations of the CFD solver on structured meshes for 3D turbulence flow. From the test cases results,we can see that GPU appears to be a very promising solution to decrease the computational cost and time of CFD solvers. GPU computing is becoming a viable solution for large-scale CFD simulation. Extensions of this work will include developing a MPI version code which will allow us to handle more complex CFD cases.

REFERENCES

[1] F. Salvadore, M. Bernardini, et al. *Journal of Computational Physics*. McGraw Hill, Vol. 235, pp.129-142, 2013.
[2] F.A. Kuo, M.R. Smith, et al. *Computers & Fluids*, Vol. 45,pp.147-154, 2011.
[3] Kim K H, Kim C. *Journal of Computational Physics*, 2005, 208.
[4] S. Goebel, J. Dutton, *AIAA Journal*. 29 (1990) 538–546.
[5] C. D Herrmann, and Koschel. *AIAA Paper* 2002-4130.