Advanced Materials Research Vol. 320 (2011) pp 335-340
Online available since 2011/Aug/16 at www.scientific.net
© (2011) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/AMR.320.335

A GPU Accelerated Red-Black SOR Algorithm for Computational Fluid Dynamics Problems

Jitang Liu[1, a], Zhaosong Ma[1, b], Shihai Li[1, c], Ying Zhao[1, d]

[1] Institte of Mechanics, Chinese Academy of Sciences, Beijing 100190, China

[a] ljt_1987624@163.com, [b] marze@163.com,

[c] shli@mail.imech.ac.cn, [d] zhaoying@imech.ac.cn

Keywords: (Graphics Processing Units) GPU, (Computational Fluid Dynamics) CFD, Red-Black SOR, Memory hierarchy, Shared Memory Allocation.

Abstract. GPUs are high performance co-processors of CPU for scientific computing including CFD. We present an optimistic shared memory allocation strategy to solve 2D CFD problems using Red-Black SOR method on GPU with CUDA (Compute Unified Device Architecture). Lid-driven results are compared with the benchmark data. The speed up ratio of same problem size by using NVDIA GTX480 and Intel Core-Dual 3.0GHz processor is discussed, the performance of GPU is 120 times faster than the sequential code on CPU with the problem size of 756×756. Based on this work, we conclude that using the memory hierarchy properly has a key role in improving the computational performance of GPU.

1. Introduction

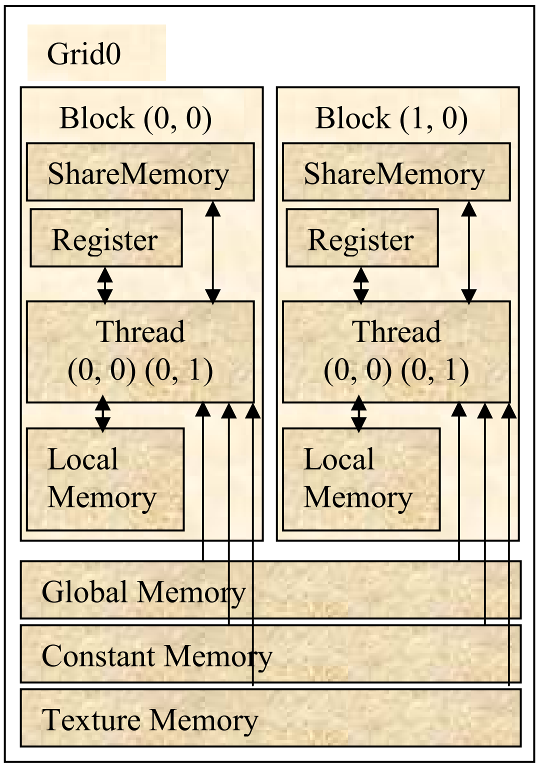
Simulating CFD problems efficiently and accurately is of great importance for scientific computing and engineering applications. GPUs that are originally designed for graphics rendering have become massively-parallel "co-processors" of the Central Processing Unit (CPU). In recent years, GPU technology develops quickly and modern GPU can provide memory bandwidth and floating-point performance that are orders of magnitude faster than a standard CPU [1]. Researchers in CFD field have done a lot of work in parallel computing algorithms and applications on GPU and gotten great achievements. In the aspect of algorithms of CFD, Senocak [2] presented a 3D Navier-Stokes solver on GPU for incompressible flows using Jacobi iteration method, Serban Georgescu [3] developed a Conjugate Gradient solver for 3D Poisson's equation on GPU and reported up to 22 times acceleration when using three GPUs compared with CPU, Jonathan M. Cohen [4] implemented 3D Boussinesq code with Red-Black Gauss-Seidel on GPU and got an acceleration of up to 8 times faster than a CPU.

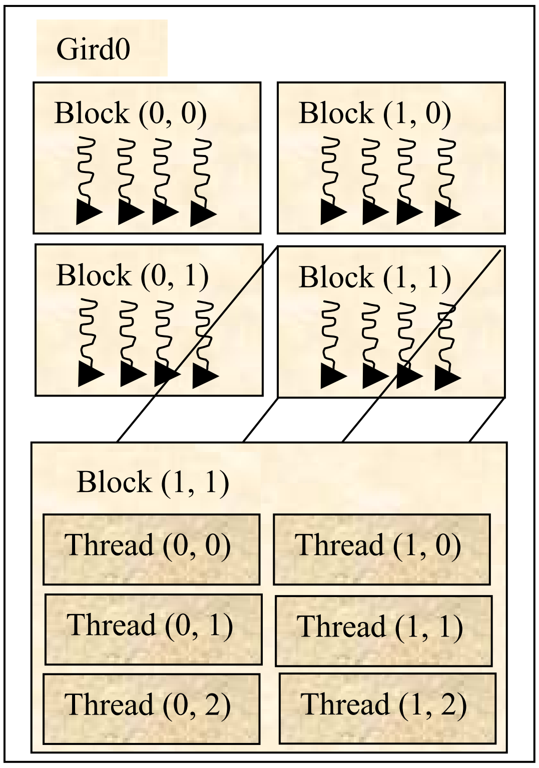
Red-Black SOR which is a high efficiency, yielding simple, inexpensive and fully parallelizable method [5] is widely used in parallel computing both on CPU and GPU. Chih-Wei Hsieh [6] implemented Red Black method for solving 2D parabolic partial differential equations on GPU was 11 times faster compared with CPU with the problem size of 400x400, Sheng-Hsiu Kuo [7] solved 2D nonlinear Burgers' equation by using Red-Black SOR method on GPU and got a speed-up ratio of 12 times at mesh size 1026×1026 on GPU compared with CPU, Jonathan M. Cohen [4] and Aaron F. Shinn [8] implemented the Red-Black SOR iteration method to solve 3D CFD problems on GPU with multi-grid relaxation schemes and achieved speed up ratio of 8 times and 15times respectively. As a highly parallel computational method, Red-Black SOR method is suitable for GPU computing and can achieve a high speed up ratio if we use the memory hierarchy properly and allocate memory efficiently according to our experience.

The rest of this paper is organized as follows. Section 2 introduces the GPU hardware architecture and CUDA programming model. Section 3 briefly shows the governing equation and numerical method of incompressible fluid flows. The acceleration strategy of solving CFD problems on GPU is described in section 4 and the result verification and speed up discussion is shown in section 5. Finally, Section 6 gives the conclusion and future work.

2. GPU Hardware Architecture and CUDA Programming Model

GPU Hardware Architecture. GPU which is originally built for graphics rendering has become a highly parallel, powerfully programmable, suitable for general purpose computing device owing to its' unique hardware architecture. As is shown in Fig.1 (a), GPU is an example of a Single Instruction Multiple Data (SIMD) multiprocessor [11]. Each thread reads data in different memory locations when executes. Each thread has its own registers and local memory, each block has the same shared memory of its own, all threads in a grid can access the data in global memory. Besides, there are two kinds of read only memory: constant memory and texture memory [12].





a) GPU SIMD multiprocessor architecture b) CUDA thread organization

Figure 1. GPU Hardware Architecture and CUDA Programming Model[12]

CUDA Programming Model. CUDA is a general purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU. CUDA C extends C by allowing the programmer to define C functions which are called kernels. Each kernel is mapped to threads on GPU. Threads in the same block can communicate with each other and synchronize together while threads from different blocks can't. Blocks which execute the same kernel function can be batched together into a grid and be executed in parallel [12], which are illustrated by Fig.1 (b).

3. Governing Equation and Numerical Method of Incompressible Fluid Flows

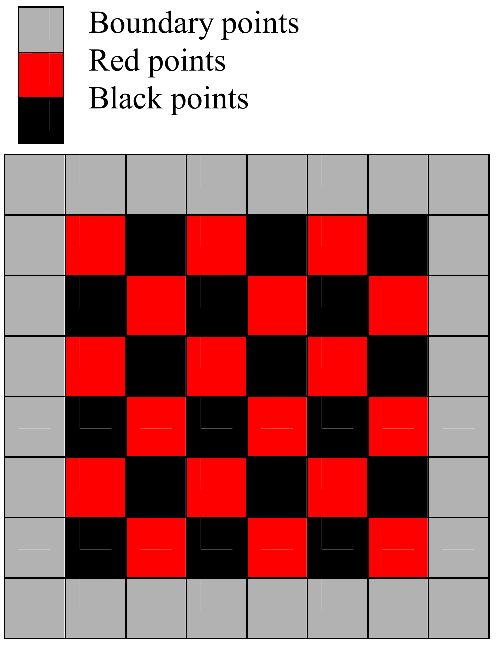
The Navier-Stokes equations for 2D incompressible fluid flows can be written as follows:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\mathrm{Re}}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\mathrm{Re}}\frac{\partial p}{\partial y}+\nu(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}) \quad (3)$$

Where *u, v* is the velocity, *p* is the pressure, ν is the kinematic viscosity and Re is Reynolds number.
To solve the equations above in numerical method, we choose first-order, explicit Euler scheme for time term and second-order central difference scheme for the diffusion and advection terms. The projection method is used to solve Navier-Stokes equations for incompressible flows [9].
To solve the Pressure Poisson equation on GPU in high speed, we choose the Red-Black SOR as iteration method which has the same convergence rate as the Gauss-Seidel method. As is shown in Fig.2, the red (black) points are surrounded by each other. We update the red points use the black points' previous values firstly and then update the black points use the red just solved. The Red-Black SOR iteration number was set to be 20 for each time step to get convergence results for problems of Re=100. The flow diagram of the code procedure is shown in Fig.3.



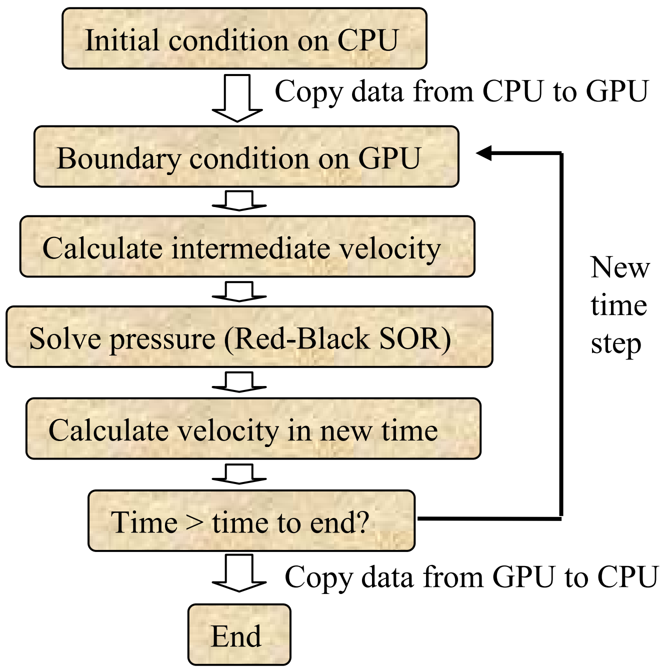
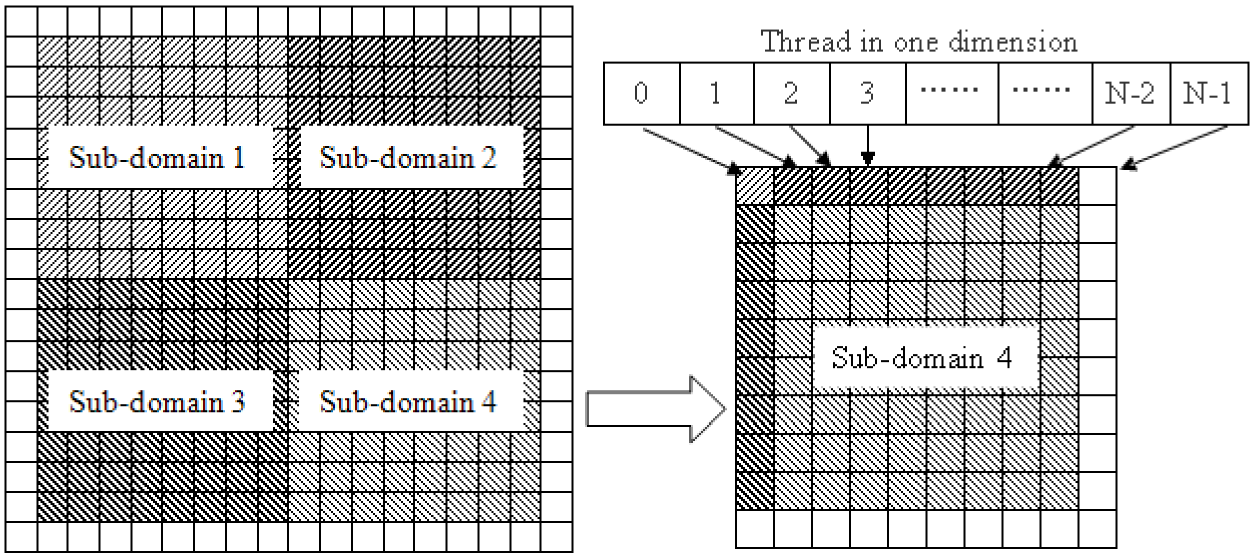

Initial condition on CPU
Copy data from CPU to GPU
Boundary condition on GPU
Calculate intermediate velocity
Solve pressure (Red-Black SOR)
Calculate velocity in new time
Time > time to end?
New time step
Copy data from GPU to CPU
End

Figure 2. Red and black point distribution   Figure 3. The flow diagram of executing code on GPU

4. Acceleration Strategies of Solving CFD Problems on GPU

As is mentioned above, solving CFD problems using Red-Black SOR is an optimal choice because of its highly parallel capability and efficiency. But the dependency on neighboring points reduces the extent of parallelism, which is also the bottle-neck of computing performance enhancement on GPU. Compared with the floating-point performances, the performance of computing is limited by the memory bandwidth of GPU when the data on Global memory was accessed by the execution units. Shared memory is chosen as the cache to improve computing performance in this paper. Firstly, the data (including boundary element data) is loaded into shared memory from global memory. Secondly, all the operations on data are completed in shared memory. Lastly, the data is written to global memory.
The domain decomposition method is used for GPU implementation. The grids in computational domain are divided into sub-domains and mapped on GPU thread as is shown in Fig. 4. The block dimension is chosen 1D and threads number is times of 32 (64 or 128, will be discussed later) to utilize the shared memory properly and efficiently (Fig. 4), hence the coalescence is obtained during the data access. Firstly, the threads in the block read the data (including the boundary data) from global memory to shared memory in the form of lines, the data in columns will be read successively

by the same block thread. Secondly, the points of inner grid are calculated by the 1D thread in the same block which can change data with each other via the shared memory. Lastly, the solved inner points are copied from shared memory to global memory. The height of the sub-domain in which condition to get highest performance will also be discussed later.



a) Computation domain decomposition b) Sub-domain mapped on 1D GPU thread

Figure 4. Domain decomposition method and thread allocation strategy

The code snippet is composed of two loops, the outer loop for time advance and the inner for iterations of Red-Black solver to solve Poisson equations numerically. Firstly, the intermediate velocity is solved using the momentum equations. Secondly, the boundary condition is computed by the next two kernel functions. Thirdly, the divergence of every element is solved by the next kernel which will be used in the iterations. Fourthly, the Poisson equations are solved using Red-Black SOR method. Lastly, the new velocity at time n+1 is calculated with the intermediate velocity and pressure solved justly.

5. Result Verification and Speed Up Discussion

Result Verification. The lid-driven flow was chosen as a benchmark for validation of our numerical method. Fig. 5 shows the contour distribution of velocity *u* and *v*, the stream line of lid-driven problem of Reynolds number 100 at steady state respectively.To numerically validate the GPU code, results of computed on GPU are compared with data from Ghia [10], which is shown in Fig. 6. As is shown in Fig. 6, the two velocity components *u* and *v* along the vertical and horizontal lines through the geometric center are in excellent agreement with results of Ghia [10] both for Reynolds numbers 100 and 1000.

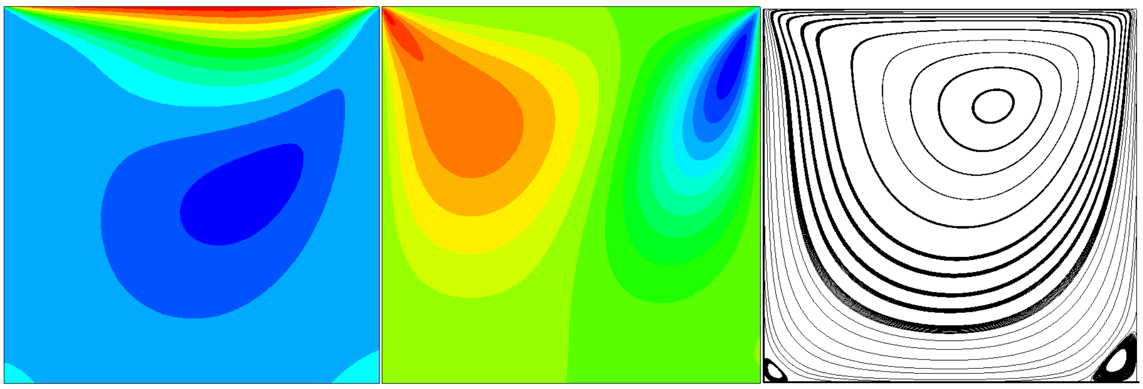
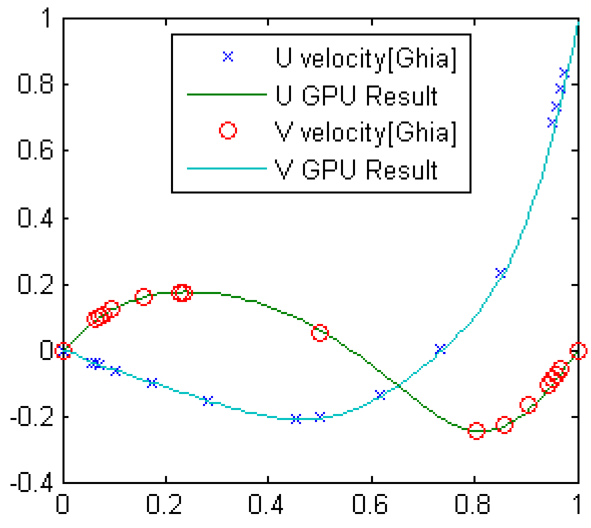
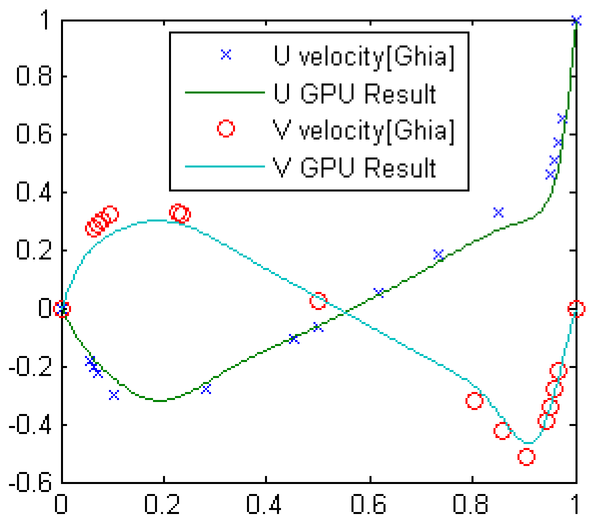


Figure. 5 Steady state Contour distribution of *u, v* and stream line at Reynolds number 100





a) Results comparison for Re=100 b) Results comparison for Re=1000

Figure. 6 Comparison of GPU results with benchmark data from Ghia for Re=100 and Re=1000

Speed Up Discussion. Next we will present the GPU computing performance compared with CPU . The lid-driven flow of Reynolds number 100 is chosen as the test case. We compared the time of the code executes 1000 time steps on different platforms. Table 1 shows the time of two codes for problems of different sizes with different thread allocation methods and the same sub-domain height (of 8), the speed up ratio can get 120 for 756×756 size when allocate 1D 128 threads in the block. The same thread allocation method (1D 128 threads in the Block) and different sub-domain heights are shown in table 2, we can get a higher speed up ratio for small problem size (120× for 756×756) when the height of the sub-domain is 8. The codes are complied by Microsoft Visual Studio 2008. The GPU we choose is GTX 480 and CPU is Intel 3.0GHz processor.

Table 1. Time comparison for problems of different sizes with different thread allocation methods and sub-domain height of 8.

Grid number	CPU time (s)	GPU time (s)	Speed up
372×372	46.437	0.531	87.45
744×744	189.25	1.641	115.33

a) problems of different sizes with 64 threads in a block

Grid number	CPU time (s)	GPU time (s)	Speed up
378×378	47.578	0.516	92.21
756×756	199.046	1.657	120.12

b) problems of different sizes with 128 threads in a block

Table 2. Time comparison for problems of different sizes with 128 thread in a block and different sub-domain heights.

Grid number	CPU time (s)	GPU time (s)	Speed up
504×504	87.375	0.922	94.77
1008×1008	355.172	3.219	110.34

a) problems of different sizes with sub-domain height of 6

Grid number	CPU time (s)	GPU time (s)	Speed up
378×378	47.578	0.516	92.21
756×756	199.046	1.657	120.12

b) problems of different sizes with sub-domain height of 8

Grid number	CPU time (s)	GPU time (s)	Speed up
504×504	87.375	0.829	105.40
1008×1008	355.172	2.937	120.93

c) problems of different sizes with sub-domain height of 10

6. Conclusion and Future Work

We found a new strategy to allocate the memory hierarchy of CUDA programming model properly to improve the overall bandwidth utilization and thus to hence the performance of computing on GPU. Based on the algorithm, we have gotten a CFD solver for incompressible fluid problems with Red-Black SOR on GPU with higher speed up ratio than ever reported. Overall, the numerical solver of incompressible fluid flow equations was accelerated by a factor 120 by using the NVDIA GTX 480 compared with the serial code on CPU 3.0 GHz processor when computing problems size of 756×756. It's found that using the memory hierarchy properly has a key role in improving the computational performance of GPU.

In our future work, we will execute the free surface code on GPU using the method offered above and investigate the speed up performance of our method.

Acknowledge

Gratefully acknowledge the support from Natural Science Foundation of China (Grant No. 11002146) and the Chinese National 973 Project (Grant No. 2010CB731500).

References

[1] J.C. Thibault, I. Senocak: 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition (2009).

[2] I. Senocak, J. Thibault, M. Caylor: Eighth Symposium on the Urban Environment, Phoenix Arizona (2009).

[3] S. Georgescu, H. Okuda: Numer. Meth. Fluids. Vol. 64(2010), p. 1254.

[4] Information on http://www.nvidia.com/content/cudazone/CUDABrowser/ downloads/papers/ DoublePrecision-CFD-Cohen-parCFD09.pdf.

[5] I. Yavneh: SIAM J. Sci. Comput., Vol. 17 (1996), p. 1.

[6] C.W. Hsieh, S.H. Kuo, F.A. Kuo, C.Y Chou: International Symposium on Parallel and Distributed Processing with Applications (2010).

[7] S.H. Kuo, C.W. Hsieh, R.K. Lin, W.H. Sheu: Computer Science. Vol. 6082 (2010), p. 297.

[8] Information on http://www.greatlakesconsortium.org/events/manycore/files/CaseStudy_CFD _GPU.pdf.

[9] A. J. Chorin: Math. Comput., Vol. 22(1968), p. 745.

[10] U Ghia, K.N. Ghia, C.T. Shin: J. Comput. Phys., Vol. 48 (1982), p. 387.

[11]. Information on http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/ NVIDIA _CUDA_ProgrammingGuide.pdf.

[12] Shu Zhang, Yanli Chu, Kaiyong Zhao, Yubo Zhang: GPU High Performance computing – CUDA (China WaterPower Press, 2009).

Key Engineering Materials and Computer Science
10.4028/www.scientific.net/AMR.320

A GPU Accelerated Red-Black SOR Algorithm for Computational Fluid Dynamics Problems
10.4028/www.scientific.net/AMR.320.335

DOI References
[3] S. Georgescu, H. Okuda: Numer. Meth. Fluids. Vol. 64(2010), p.1254.
http://dx.doi.org/10.1002/fld.2462
[5] I. Yavneh: SIAM J. Sci. Comput., Vol. 17 (1996), p.1.
doi:10.1137/0917013
[6] C.W. Hsieh, S.H. Kuo, F.A. Kuo, C. Y Chou: International Symposium on Parallel and Distributed Processing with Applications (2010).
doi:10.1109/ISPA.2010.48
[9] A. J. Chorin: Math. Comput., Vol. 22(1968), p.745.
http://dx.doi.org/10.1090/S0025-5718-1968-0242392-2
[10] U Ghia, K.N. Ghia, C.T. Shin: J. Comput. Phys., Vol. 48 (1982), p.387.
doi:10.1016/0021-9991(82)90058-4
[12] Shu Zhang, Yanli Chu, Kaiyong Zhao, Yubo Zhang: GPU High Performance computing – CUDA (China WaterPower Press, 2009).
doi:10.1109/ICC.2009.5199483