Parallel GPU algorithms for alternate-triangular finite difference schemes

Bohaienko V.O.[1]

[1]V.M. Glushkov Institute of cybernetics of NAS of Ukraine, Glushkov ave., 40, Kyiv, Ukraine

sevab@ukr.net

Abstract. Parallel algorithms for modern high performance computing systems are required for fast modelling of high dimensional convection-diffusion processes in air. Such algorithms, designed for alternate-triangular finite difference splitting schemes applied to convection-diffusion equation, have been considered. An algorithm for single GPU systems and an algorithm for clusters with graphical processors has been described, algorithms' performance theoretical estimations has been given and results of their testing on SKIT-4 cluster of Institute of Cybernetics has been presented. Obtained testing results about developed algorithms' performance with sufficient accuracy agree with theoretical estimations.

Keywords

finite elements method, alternate-triangular schemes, convection-diffusion equations, parallel algorithms, GPU, OpenCL.

1 Introduction

Problem of diffusion-convection processes modelling in air has been considered on the base of equation

$$\frac{\partial\varphi}{\partial t}+\frac{\partial u(\vec{x},t)\varphi}{\partial x}+\frac{\partial v(\vec{x},t)\varphi}{\partial y}+\sigma\varphi=div(\mu\nabla\varphi)+\sum_{i=1}^{N}q_i\delta(\vec{x}-\vec{r}_i), \tag{1}$$

using alternate-triangular finite difference splitting schemes [1]. Henceforth φ is a migrating substance's concentration; $u(\vec{x},t)$, $v(\vec{x},t)$ are an air velocity vector's components; μ is a diffusion coefficient; σ is a coefficient of migrating substance's concentration change over time; q_i is an intensity of point source; $\vec{r}_i$ is coordinates of point source placement; $\vec{x}\in\Omega\subset R^2$, Ω is a domain of solution. When convection term dominates, difficulties arise during numerical solution of equation, so special schemes that use physical processes decomposition and are based on finite difference [2,3], finite element [4,5] or finite volume [6] discretization were developped. Alternate-triangular schemes are explicit schemes based on the alternate directions approach [7,8,9] and have natural parallelism which can be used during their program implementation on clusters, GPU and GPU clusters. While GPU algorithms for implicit alternate directions schemes that results in independent solving of many tridiagonal linear equation systems [10,11,12] are widely studied, little research concerning GPU implementation of explicit schemes [13] is done.

2 Parallel algorithm for GPU

Alternate-triangular schemes' natural parallelism can be exposed by applying loop skewing to two of highest loops in loop nest that must be executed on every splitting step [1,14,15] after that computations in inner loop iteration become independent. On each of inner loop iterations, one grid cell value is being changed using values in neighbour cells. Increase of granularity results in the need of doing computations over a block of cells while distance vectors of iteration's dependences don't change. Such computational scheme's feature make it possible

to use similar parallel schemes for distributed and shared memory systems, particularly, GPU, and to combine them with little algorithms' modifications. However, such schemes do not completely use processor resources.

Parallel algorithm for clusters based on algorithms from [14] and described in [15] uses overlapping block-row data distribution, cutting latter row blocks into column blocks inside each process. While processing first and last P blocks, where P is a number of processes, some of them stand idle. Optimal block size can be calculated using theoretical estimation of algorithm's running time and measured parameters of computational system [15]. GPU parallel algorithm for alternate-triangular schemes must be constructed taking graphical processor memory organization specificity into account. Following algorithm for splitting steps with positive velocity components values [1] is proposed:

- data are defined on a $n \times m$ grid, split into square, except, perhaps, boundary, blocks of fixed size $l \times l$;
- on each of $(n/l) + (m/l) - 1$ iterations, independent computations inside from 1 to n/l blocks with cell coordinates from $(il, (j-i)l)$ to $((i+1)l, (j-i-1)l)$, where j is an iteration number and i is a block number, is being carried out. Computations inside a block is being done only if the block lies inside the grid. This explains different quantity of processed blocks that depends on iteration index. On each of iterations, CPU program sets input parameters and runs GPU kernel;
- GPU kernel that executes one computation's iteration does per row data processing that is one thread is responsible for one cell row inside a block. One block's rows processing threads are grouped for acceleration of computations by using local GPU memory. Before computations start, all needed data are simultaneously loaded into local memory using request coalescing, after that only local memory is used. Later, results are written back into global memory;
- the same scheme is used while carrying out computations inside each thread group - $2l - 1$ iterations on unit size blocks must be completed with some threads idle and barrier synchronization operation after each iteration.

Running time of such algorithm (hereinafter without considering initialization and results gathering procedures) can be estimated as below.

Time spent by l threads for one $l \times l$ block processing can be estimated as

$$t_1 = (5l + 16)t_g + (2l - 1)(13t_l + t_c + t_b), \tag{2}$$

where t_g and t_l are execution times of read or write operations done by all threads, correspondingly, from or in local or global memory and which, in case of full coalescing, are equal to operation's execution time then it is done by one thread; t_c is single cell computations execution time; t_b is barrier synchronization execution time.

Assuming that GPU can execute $b * l$ threads simultaniously that $m/l > n/l$, total running time can be not deeply considering GPU architecture estimated as

$$t(n, m, l) = (2\sum_{i=1}^{n/l}(\left\lfloor \frac{i}{b} \right\rfloor + 1) + (\frac{m}{l} - \frac{n}{l} - 1)(\left\lfloor \frac{n/l}{b} \right\rfloor + 1))t_1. \tag{3}$$

Assuming that local memory access time is essentially less that time of access in global memory and therefore can be neglected; that GPU is able to execute all created threads simultaneously ($b > (n/l)$); that grid is square (n=m), estimation (3) turns into

$$\bar{t}(n, l) = (2\frac{n}{l} - 1)((5l + 16)t_g + (2l - 1)(t_c + t_b)). \tag{4}$$

While l is in range $l = [1, ..., n]$, $\bar{t}(n, l)$ function have single optimum that is a maximum and it's minimal value is achieved when $l = 1$ or $l = n$. When $n > 13$, local minimum will be achieved at $l = n$ and will be equal to $(5n + 4)t_g + (2n - 1)(t_c + t_b)$. Since l is limited to GPU computational resources number and $n > 13$ condition always meets for problems that needs parallel computations, selecting maximal permissible l will be the best strategy for computations acceleration.

3 Parallel algorithm for GPU clusters

GPU algorithm of alternate-triangular schemes can be used while doing distributed program's single processes' local computations. The only needed modification for such usage is CPU-GPU data exchange addition. Such scheme's running time can be estimated as

$$t_3(n,m,g,P,l) = \left(\frac{m}{g} + P - 1\right)\left(t(\frac{n}{P},g,l) + 2gt_{s2} + T_s(2g)\right), \quad (5)$$

where g is columns' block size which optimal value can be found using estimation given in [15], $T_s(s)$ is an s data units exchange time, t_s is a network performance coefficient, t_{sc} is a constant time spent on auxiliary data exchange operations, t_{s2} is an estimation of time spent on transmitting a unit of data between GPU and CPU.

Usage of estimation (4) with $l = n = m$ and $g = \frac{n}{P}$ instead of estimations (3) gives

$$t_4(n,P) = (2P-1)\left(\frac{n}{P}(4t_g + 2(t_c+t_b) + 2(t_{s2}+t_s)) + 16t_g - (t_c+t_b) + t_{sc}\right) \quad (6)$$

Order of optimal processor's number subject to grid size here is $O(\sqrt{n})$ while for CPU algorithm with running time estimation as

$$t_5(n,P) = (2P-1)\left(\frac{n}{P}(\frac{n}{P}t_{cpu} + 2t_s) + t_{sc}\right), \quad (7)$$

where t_{cpu} is a CPU performance coefficient, this order is $O(n)$. Efficiency of first algorithm has $O\left(\frac{n}{P^2}\right)$ order, efficiency of the second — $O\left(\frac{n^2}{P^2}\right)$. But, on the other hand, GPU algorithm running on single GPU in optimal conditions has $O(n)$ order of complexity while CPU algorithm — $O\left(n^2\right)$.

4 Parallel algorithms performance testing

Algorithms' software implementations have been tested on SKIT-4 NASU Institute of Cybernetics' cluster (12 nodes with 4 Intel Xeon E5-2600, 3 NVidia Tesla M2075; CentOS 5.9, Cuda toolkit 4.2, OpenMPI 1.6.5). As a testing problem, modelling of pollution transport in air from point source has been chosen as below:

$$\frac{\partial\varphi}{\partial t} + \frac{\partial\varphi}{\partial x} - 0.5\frac{\partial\varphi}{\partial y} = div(0.5\nabla\varphi) + \delta(\vec{x} - \overline{(20,10)}, t), \vec{x} \in \Omega_1 = \overline{([0..40],[0..20])}, \quad (8)$$

$$\varphi(\vec{x} \in \overline{\Omega_1}, t) = 0, \varphi(\vec{x}, 0) = \delta(\vec{x} - \overline{(20,10)}, 0). \quad (9)$$

Here Ω_1 is a domain of solution and $\overline{\Omega_1}$ is it's boundary. Execution time of one time step with $\tau = 0.002$ measured while running software on single GPU is given in table 1.

Table 1. One time step computations' execution time (single GPU)

Grid size	Block size l	Local memory usage	Time, msec	One block's processing time, msec	Acceleration from local memory usage	Estimation of one block processing time using formula (2), ms	Relative accuracy of estimation
3600x1800	4	+	2960	1,1945117	43%	1,242	7%
3600x1800	4	-	4240	1,7110573		1,6656	4%
3600x1800	16	+	2110	3,39228296	9%	3,468	3%
3600x1800	16	-	2300	3,6977492		3,6624	1%
2400x1200	8	+	1050	1,75			
1800x900	8	+	625	1,38888889			

One block processing time has been calculated using formula (3) using information about NVidia Tesla M2075 parallel processing capabilities and assuming that maximal block size is limited by local memory size.

According to theoretical estimations, measured time slightly relates on grid size and linearly depends on block size. Assuming that auxiliary kernel execution operations take constant time $t_{kc} = 0.5$, such linear dependencies have been found for algorithms with and without local memory usage: $t_1(l) = 0.1855l + 0.1032$ and $t_1(l) = 0.1664l + 0.5408$, correspondingly. On the base of these dependencies, coefficient values from formula (2) have been found and estimations of one block's processing time have been done using this formula.

Execution time for problem with 3600x1800 grid size while running on the GPU cluster is given in table 2.

Table 2. One time step computations' execution time (several GPU)

Number of GPUs	Block size l	Local memory usage	Time, msec	Time estimation using formula (5), msec	Relative error of estimation
1	8	+	2230		
12	8	+	2840	2632,13893	7,32%
1	16	+	2110		
12	16	+	2710	2941,64947	8,55%
1	32	-	1980		
12	32	-	3800	2719,39866	28,44%

Estimating execution time using formula (5), one $l \times l$ block's processing time measured upon single GPU testing was used. Processing time of one $\frac{n}{P} \times g$ cell block was estimated using formula (3) where a mean of values measured in each run was used as t_s coefficient value. Value of t_{sc} was obtained upon testing algorithm for clusters without GPU.

5 Conclusions

The results show that usage of GPU local memory gives bigger speedup for lesser block size. This can be explained by complication of local memory access synchronization that isn't taken into account in estimations (2),(3) and results in estimation error increase.

Minimal running time was reached at $l = 16$ for algorithm that uses GPU local memory and at $l = 32$ for algorithm that doesn't use it. In optimal conditions the second is 6% faster. Acceleration comparing with CPU algorithm was equal to 9.6. Theoretical estimations and measured running times shows lesser efficiency of algorithm for GPU clusters comparing with algorithm for CPU clusters when running on equal number of computational units, particularly, maximal acceleration comparing with single GPU algorithm was 17% while running on 4 GPU. Another expected consequence was a speedup fall when block size increases. But in optimal conditions parallel algorithm for GPU clusters was found to be 3.19 times faster than CPU cluster algorithm.

References

[1] Згуровский М.З., Скопецкий В.В., Хрущ В.К., Беляев Н.Н. Численное моделирование распространений загрязнения в окружающей среде.- К.: Наук. думка 1997. - 368 с.

[2] H.Y. Xu, M.D. Matovic A. Pollard Finite Difference Schemes for Three-dimensional Time-dependent Convection-Diffusion Equation Using Full Global Discretization // Journal of Computational Physics, Volume 130, Issue 1, 1 January 1997, Pages 109–122

[3] E. Sousa, I. Sobey A family of finite difference schemes for the convection-diffusion equation in two dimensions // Numerical Mathematics and Advanced Applications 2003, pp 179-188

[4] Rajeev Kumar, Brian H. Dennis, Bubble-Enriched Least-Squares Finite Element Method for Transient Advective Transport // Differential Equations and Nonlinear Mechanics, vol. 2008, Article ID 267454, 21 pages, 2008

[5] Allen J. Toreja, Rizwan-uddin Hybrid numerical methods for convection–diffusion problems in arbitrary geometries // Computers & Fluids, Volume 32, Issue 6, July 2003, Pages 835–872

[6] C. G. Mingham, D. M. Causon A simple high-resolution advection scheme // International Journal for Numerical Methods in Fluids Volume 56, Issue 5, pages 469–484, 2008

[7] J. Douglas, Jr Alternating direction methods for three space variables // Number.Math.4(1962), pp.41-63

[8] J. Douglas, Jr, D. Peaceman Numerical solution of two-dimensional heat flow problems // American Institute of Chemical Engineering Journal, 1 (1995), pp.505-512

[9] T. Arbogast, Chieh-Sen Huang, Song-Ming Yang Improved Accuracy for Alternating-Direction Methods for Parabolic Equations Based on Regular and Mixed Finite Elements // Math. Models Methods Appl. Sci. 17, 1279 (2007)

[10] Y. Zhang, J. Cohen, J.D. Owens Fast tridiagonal solvers on the GPU // PPoPP '10 Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 127-136

[11] A. Davidson, Y. Zhang, and J. D. Owens An auto-tuned method for solving large tridiagonal systems on the GPU // Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, pages 956-965, May 2011.

[12] D. Goddeke and R. Strzodka Cyclic reduction tridiagonal solvers on GPUs applied to mixed precision multigrid // IEEE Trans. Parallel Dist. Syst., vol. 21, 2010, pp.22-32

[13] Y. Inoue, M. Unno, S. Aono, H. Asai GPGPU-based ADE-FDTD method for fast electromagnetic field simulation and its estimation // Microwave Conference Proceedings (APMC), 2011 Asia-Pacific, pp. 733 - 736

[14] Кучеров А.Б., Николаев Е.С. Параллельные алгоритмы итерационных методов с факторизированным оператором для решения эллиптических краевых задач // Дифференц. ур-ния. 1984. Т.20. №7. С. 1230-1236.

[15] Скопецкий В.В., Богаенко В.А. Моделирование прямых и обратных задач распространения загрязнений в воздушной среде с помощью кластерной системы СКИТ // Управляющие системы и машины. – 2007. - №5. – С. 86-92.