Parallel algorithms for fast air pollution assessment in three dimensions

Bohaienko V.O.[1]

[1]Glushkov Institute of Cybernetic of NAS of Ukraine, Kyiv, Ukraine

sevab@ukr.net

Abstract. Parallel algorithms for graphical processors have been developed for three-dimensional alternate-triangular finite-difference explicit splitting scheme with regard to air pollution modelling problems. Theoretical estimations of proposed algorithms execution time have been obtained. Results of algorithms testing while modelling industrial premise pollution process after accidental gas release have been presented.

Keywords

Three-dimensional problems, alternate-triangular schemes, convection-diffusion equation, Poisson equation, parallel algorithms, GPU, OpenCL.

1 Introduction

Modelling of air pollution process after accidental gas release is considered. Gas transfer process in an industrial premise is described by gradient flow model

$$\frac{\partial C}{\partial \mathrm{t}} + \frac{\partial \mathrm{uC}}{\partial \mathrm{x}} + \frac{\partial \mathrm{vC}}{\partial \mathrm{y}} + \frac{\partial w\mathrm{C}}{\partial \mathrm{z}} = \mu\left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} + \frac{\partial^2 C}{\partial z^2}\right) + \sum \mathrm{Q_i}\left(t\right)\delta\left(r - r_i\right), \tag{1}$$

where C is substance concentration in the premise; u, v, w – air velocity vector components; μ – diffusion coefficient; Q – substance emission intensity; $\delta\left(r - r_i\right)$ – Dirac delta function; $r_i = (x_i, y_{i,}, z_i)$ – emission sources coordinates.

Air motion is considered to be potential and air velocity vector components is defined as

$$u = \frac{\partial P}{\partial x}, \quad v = \frac{\partial P}{\partial y}, \quad w = \frac{\partial P}{\partial z}, \tag{2}$$

where P is a velocity potential that can be found from

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} = 0. \tag{3}$$

For the equations (1),(3) following boundary conditions are set:

- $C = C_E$ and $\frac{\partial P}{\partial n} = V_n$ where V_n is a given velocity and C_E - pollutant concentration, are set on ventilation inlets;
- $\frac{\partial C}{\partial n} = 0$ and $\frac{\partial P}{\partial n} = 0$ where n is a unit normal to a surface are set on the walls of the premise;
- $\frac{\partial C}{\partial n} = 0$ and $P = P_0 + const$ where P_0 is a given velocity potential are set on the outlets.

For the equation (1) in the case of convection term domination special schemes that use physical processes decomposition were developped. Among them, finite-difference splitting schemes based on the alternate directions approach [1,2] are one the fastest. Most parallel algorithms for such schemes were developed for implicit

schemes and are primary focused on parallel solution of three-diagonal linear equation systems [4-6] while parallel algorithms for faster explicit schemes are less studied [7,8]. To solve the system (1),(3) alternate-triangular explicit finite difference splitting scheme [3] is used. Used scheme comparing with implicit schemes and to finite elements method has lesser computational complexity but also lesser accuracy thus can be efficiently used for quick assessment of process evolution. Developing of parallel algorithms, especially for graphical processors (GPU), here gives more performance while preserving accuracy.

2 Parallel algorithms for GPU

For three-dimensional problem (1),(3) solving using alternate-triangular scheme [3] in grid domain

$$\overline{\omega}_h = \omega_h \bigcup \gamma_h = \{(x_i = ih_1,\ i = \overline{0, N}_1\ ;\ y_j = jh_2,\ j = \overline{0, N}_2; z_j = kh_3,\ k = \overline{0, N}_3), h_\alpha = l_\alpha / N_\alpha\} \qquad (4)$$

two algorithms which extend the two-dimensional one described in [8] into three-dimensional case are proposed.

Let's algorithmically consider alternate-triangular splitting scheme [3] as three nested loops. Loop skewing procedure [9] can be applied to complete loop nest or only to two innermost loops.

In the first case, algorithm (further designated as algorithm 1) can be described as follows:

- 1) Set of internal nodes $(x_i, y_j, z_k) \in \omega_h$ is split on cubic (perhaps with exception of boundary) blocks of size$L \times L \times L$.
- 2) Let $K = \max(N_1 - 1, N_2 - 1, N_3 - 1)$, $N = \min(N_1 - 1, N_2 - 1, N_3 - 1)$, $M = \begin{cases} N_1 - 1, (N_2 \geq N_1 \geq N_3) \ \vee \ (N_3 \geq N_1 \geq N_2), \\ N_2 - 1, (N_1 \geq N_2 \geq N_3) \ \vee \ (N_3 \geq N_2 \geq N_1), \\ N_3 - 1, (N_1 \geq N_3 \geq N_2) \ \vee \ (N_2 \geq N_3 \geq N_1), \end{cases}$ then on each of $\lfloor N/L \rfloor + \lfloor M/L \rfloor + \lfloor K/L \rfloor - 2$ iterations, independent computations using formulae given in [3] are made inside up to$\lfloor N/L \rfloor \ \lfloor M/L \rfloor$ blocks. When $N_1 \geq N_2 \geq N_3$, block nodes' coordinates are within the range from $(iL + 1, jL + 1, (k - i - j)L + 1)$ to $((i + 1)L, (j + 1)L, (k - i - j + 1)L)$, where $k = 0, ..., \lfloor N/L \rfloor + \lfloor M/L \rfloor + \lfloor K/L \rfloor - 3$ is the step index, $(i, j),\ i = 0, ..., \lfloor N/L \rfloor - 1,\ j = 0, ..., \lfloor M/L \rfloor - 1$ is the block index. Computations are being done only if block lies inside the grid. Algorithm can be easily adopted for cases other than $N_1 \geq N_2 \geq N_3$.
- 3) Each of step 2 iterations is being executed by GPU kernel and each block is being processed by a group of L^2 threads.
- 4) While carrying out computations within each thread group, $k = 0, ..., 3L - 3$ iterations are made with thread $(i, j),\ i = 0, ..., L - 1,\ j = 0, ..., L - 1$ processing node $(i, j, (k - i - j))$ of $L \times L \times L$-sized block and barrier synchronization operation executed after each iteration.

Regarding algorithm 1, time spent by L^2 threads for processing of one block can be estimated as

$$T_{31}(L) = (5L + 16)t_g + (3L - 1)(13t_l + t_c + t_b), \qquad (5)$$

where t_g and t_l are execution time of global and local memory access operations done by all threads, t_c is a time spent on doing computations for one node, t_b is a barrier synchronization execution time.

Total time (if GPU can execute bL^2 threads simultaneously) can be estimated as

$$\begin{aligned} T_{32}(N, M, K, L) = & \left(2 \sum_{i=1}^{\lfloor N/L \rfloor} \left(\left\lfloor \frac{i(i+1)/2}{b} \right\rfloor + 1 \right) + \right. \\ & + 2 \sum_{i=1}^{\lfloor M/L \rfloor - \lfloor N/L \rfloor} \left(\left\lfloor \frac{i\lfloor N/L \rfloor + \lfloor N/L \rfloor (\lfloor N/L \rfloor + 1)/2}{b} \right\rfloor + 1 \right) + \\ & + \sum_{i=1}^{\lfloor N/L \rfloor} \left(\left\lfloor \frac{(\lfloor M/L \rfloor - \lfloor N/L \rfloor) \lfloor N/L \rfloor + \lfloor N/L \rfloor (\lfloor N/L \rfloor + 1)/2) + (1 - i)/2}{b} \right\rfloor + 1 \right) + \\ & \left. + \left(\left\lfloor \frac{K}{L} \right\rfloor - \left\lfloor \frac{M}{L} \right\rfloor - 1 \right) \left(\left\lfloor \frac{\lfloor M/L \rfloor \lfloor N/L \rfloor}{b} \right\rfloor + 1 \right) \right) T_{31}(L). \end{aligned} \qquad (6)$$

In the second case of skewing only two innermost loops, three-dimensional problem is regarded as a sequence of two-dimensional ones that can be solved by algorithm described in [8]. Such algorithm will be further designated as algorithm 2.

Within algorithm 2, $N \times M \times K$ grid is split into $L \times L \times K$ blocks. $\lfloor N/L \rfloor + \lfloor M/L \rfloor - 1$ iterations are done with independent computations inside up to $\lfloor N/L \rfloor$ blocks. Each block is being processed by a group of L threads that sequentially handles its' $L \times L$ layers.

Time spent by L threads to process $L \times L$ layer can be estimated as

$$T_{21}(L) = (15L + 48)t_g + (2L - 1)(14t_l + t_c + t_b), \tag{7}$$

and total execution time (if GPU can execute bL threads simultaneously) will be

$$T_{22}(N, M, K, L) = K\left(2\sum_{i=1}^{\lfloor N/L \rfloor}\left(\left\lfloor \frac{i}{b} \right\rfloor + 1\right) + \left(\left\lfloor \frac{M}{L} \right\rfloor - \left\lfloor \frac{N}{L} \right\rfloor + 1\right)\left(\left\lfloor \frac{N/L}{b} \right\rfloor + 1\right)\right) T_{21}(L). \tag{8}$$

3 Parallel algorithms performance testing

Modelling of air pollution after accidental gas release in industrial premise depicted on figure 1 is considered as testing problem. Premise has $8m$ length, $6m$ width and $8m$ height with $1.6m$ height and $1.2m$ width ventilation inlet on the left wall and the same sizes outlet on the opposite wall. Incoming air velocity was chosen to be $1m/s$, diffusion coefficient of a gas – to be equal to $0.2m^2/s$ with initial dimensionless gas concentration of $C = 1$ in a $2 \times 2 \times 4m$ parallelepiped centred in point $(2m, 3m, 3m)$. Solution result for $t = 40s$ in $y = 3m$ plane is presented on figure 2.

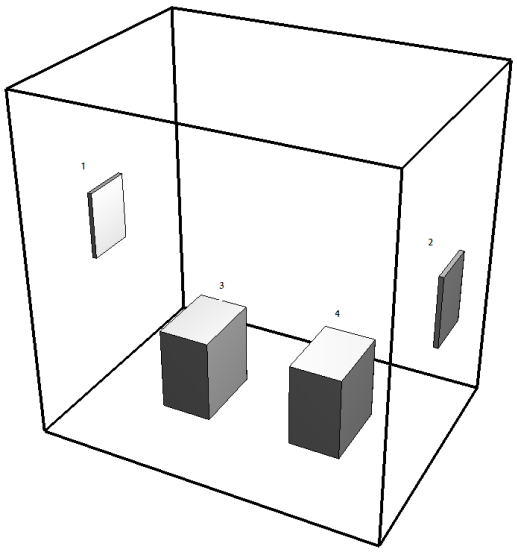

Figure 1. Industrial premise model (1 – ventilation inlet; 2 – outlet; 3,4 – industrial equipment)

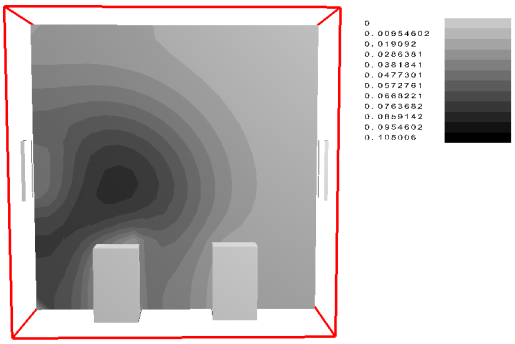

Figure 2. Gas concentration

Algorithms were tested on SKIT-4 NASU Institute of Cybernetics' cluster (28 nodes with 4 Intel Xeon E5-2600 and 3 NVidia Tesla M2075 per node). Both algorithms were applied to solve both convection-diffusion

equation (1) and Poisson equation (3). Time spent for solution of equations on one time step is given in table 1. Block size L in all cases was equal to 8.

Table 1. Execution time, ms.

Grid size	Equation (1), algorithm 1	Equation (1), algorithm 2	Equation (3), algorithm 1	Equation (3), algorithm 2	Equation (1) solved using CPU	Equation (3) solved using CPU
40x40x30	17	110	3	19	100	20
80x80x60	44	480	8	86	1070	107
160x160x120	200	2130	41	436	6130	966
320x320x240	1493	9700	294	1909	45810	8389

Experimental data shows that algorithm 1 is more efficient that algorithm 2. Solving problems using algorithm 2 gives 4.72 speedup for convection-diffusion equation (1) and 4.39 – for Poisson equation (3) while algorithm 1 acquires 30.66 speedup for equation (1) and 28.53 speedup for equation (3). It's worth noting that bigger speedup was achieved for more computationally complex problem.

Boundary conditions accounting is a significant part of total execution time in the case of algorithm 1. Each thread of GPU kernel that implements this procedure reads marker value from an array of boundary conditions markers and makes changes in solution values array. Such procedure is inefficient while executing on GPU because of significantly larger number of memory access operations comparing with number of arithmetical operations.

Execution time estimations (6),(8) and relative errors of such estimations are given in table 2.

Table 2. Execution time estimations, ms.

Grid size	Equation (1), algorithm 2	Relative error of estimation for equation (1), algorithm 2	Equation (3), algorithm 2	Equation (3), algorithm 1	Relative error of estimation for equation (3), algorithm 1	Relative error of estimation for equation (3), algorithm 2
40x40x30	110,92	-4,64%	20,86	2,28	-14,37%	-9,80%
80x80x60	499,14	-6,88%	93,87	5,80	3,23%	-10,45%
160x160x120	2107,48	0,26%	396,37	27,97	3,53%	6,51%
320x320x240	8651,77	9,59%	1627,24	187,56	5,27%	10,25%

Relative errors of execution time estimations lie in an acceptable range of below 15%. Factors that influence aforementioned estimations' accuracy can be revealed while analysing execution time of iterations done by GPU kernels.

Time spent for executing iterations of algorithms' 1 step 2 is presented on figure 3.

Regions that correspond to formula (6) terms can be observed on figure 3. Iterations $1-30$ and $81-110$ correspond to term with quadratic growth of processed blocks number, iterations $31-40$ and $71-80$ – to term with linear growth and iterations $41-70$ – to term with almost constant processed blocks number. Measured time dependence on iteration index is in concordance with obtained theoretical estimation and its' incorrectness can be explained by influence of such factors as caching. In addition, experimental results show that one blocks' processing time here, as it was assumed, does not depend on grid size.

Time spent for processing one $L \times L$ layer in $L \times L \times K$ block when solving different problems using algorithm 2 is presented on figure 3.

Execution time estimations for algorithm 2 were built under assumption that global memory access time is constant and so is one $L \times L$ layer processing time. But, as it follows from figure 4, this time isn't constant and varies more when solving convection-diffusion equation that involves more memory access operation comparing with solution of Poisson equation.

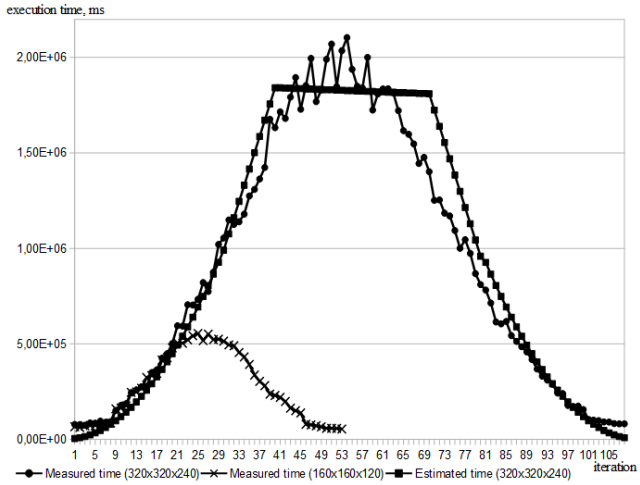

Figure 3. Execution time of algorithms' 1 step 2 iterations

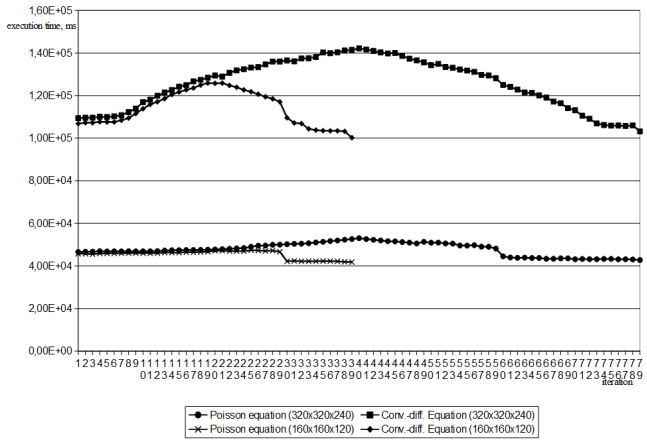

Figure 4. One LxL layer processing time on iterations of algorithm 2

4 Conclusions

Comparison of two GPU parallel algorithms execution times shows that algorithm 1 with three-dimensional grid decomposition achieves 6-times better performance than algorithm 2 with two-dimensional decomposition and 30 speedup comparing with CPU algorithm. Performance of algorithm 1 appears to be less dependent on specific factors that affect memory access time but for small grids it is severely influenced by execution time of poorly parallelizable boundary conditions accounting procedure. The last-mentioned feature can be one the factors that leads to lesser accuracy of algorithms' 1 execution time estimation. Performance models of the proposed algorithms have been built mostly taking algorithm structure into consideration while giving less attention to hardware. Further studies will be conducted to build more accurate performance models.

References

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

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

[3] Gladky A.V. Study of splitting algorithms for convection-diffution problems // Cybernetics and system analysis. – Volume 50, Issue 4, 2014. –pp.76-88. (in russian)

[4] 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.

[5] 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

[6] Y. Zhang, J. Cohen, A. A. Davidson, and J. D. Owens, "A hybrid method for solving tridiagonal systems on the GPU," in GPU Computing Gems, vol. 2, Morgan Kaufmann, Aug. 2011.

[7] 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

[8] Bohaienko V.O. Parallel GPU algorithms for alternate-triangular finite-difference schemes // Third International Conference "High Performance Computing" HPC-UA 2013, 2013, pp.64-68

[9] David F. Bacon, Susan L. Graham, Oliver J. Sharp, Compiler transformations for high-performance computing // ACM Computing Surveys (CSUR), Volume 26 Issue 4, Dec. 1994, pp 345-420.