

## GPU implementation of Jacobi method for data arrays that exceed GPU-dedicated memory size

Kochurov A.V., Vorotnikova D.G., Golovashkin D.L.

Image Processing Systems Institute, Russian Academy of Sciences,  
Samara State Aerospace University

**Abstract.** The paper proposes a method to extend the dimension of grids that GPU-aided implicit finite difference method is capable to work with. The approach is based on the pyramid method. A predictive mathematical model for computation duration is proposed. This model allows to find optimal algorithm parameters. The paper provides computation experiment results that has shown the model to be accurate enough to predict optimal algorithm parameters

**Keywords:** pyramid method, finite-difference method, parallel computing, GPU computing, Jacobi method

**Citation:** Kochurov A.V., Vorotnikova D.G., Golovashkin D.L. GPU implementation of Jacobi Method for Data Arrays that Exceed GPU-dedicated Memory Size. Proceedings of Information Technology and Nanotechnology (ITNT-2015), CEUR Workshop Proceedings, 2015; 1490: 414-419. DOI: 10.18287/1613-0073-2015-1490-414-419

Issues related to general-purpose computing on graphics processing units (GPU) and particularly GPU-aided finite-difference method implementations are discussed in a number of publications, such as [1, 2].

A major drawback of the methods discussed is the requirement for the video memory capacity to be big enough to contain the entire grid domain. Known open-source libraries that solve such problems, such as OpenCurrent (<https://code.google.com/p/opencurrent/>), have the same limitation. Note that modern GPUs do not provide memory upgrade capability. Besides, the video memory is much smaller than RAM.

Meanwhile, practical large-scale tasks require the grid domains as large as thousands of nodes in a single direction. Such problems may arise in many areas where partial differential equations (PDE) are applied, such as designing micron-size optical elements with nanoscale features [3], simulating light propagation with FDTD, seismic wave modelling, etc. There's clearly a need for PDE solving methods that do not require the entire grid to fit into the GPU memory.

There are few known approaches for explicit finite difference method that allow to apply GPU for such problems, such as: using multiple GPU or GPU-aided computing cluster [4]; algorithmic tricks allowing to reduce GPU-memory requirements by storing the main data array in RAM and dynamically copying the necessary data to



Obviously the diagonal of A matrix contains only non-zero elements. Then the Jacobi method iteration for this system takes a form:

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left( - \sum_{j \neq i} a_{i,j} x_j^k + b_i \right), \quad (1)$$

where  $i = 1, \dots, N$ ,  $k = 0, 1, \dots, N$ ,  $x_i^k$  is approximation of the solution at  $k$ -th iteration,  $x_i^0$  is an initial approximation of the solution.

Taking into account that A has only five non-zero diagonals there are only four summands under the sum sign in Eq.1.

Replacing  $x_i$  back with corresponding  $u_{i,j}$  turns the Jacobi iteration into the following form:  $u_{i,j}^{k+1} = \frac{1}{4} \left( u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k + \frac{h^2}{D} f_{i,j} \right)$ .

The same idea of Jacobi iteration could be also applied to the stationary 3D heat equation  $\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} = \frac{-f(x,y,z)}{D}$ . The Jacobi iteration for this equation is very similar to the one for 2D heat equation:

$$u_{i,j}^{k+1} = \frac{1}{6} \left( u_{i+1,j,k}^t + u_{i-1,j,k}^t + u_{i,j+1,k}^t + u_{i,j-1,k}^t + u_{i,j,k-1}^t + u_{i,j,k+1}^t + \frac{h^2}{D} f_{i,j,k} \right). \quad (2)$$

The Eq. 2 describes the basic iteration of Jacobi method. In order to perform a single iteration it is necessary to store matrices  $u^t$ ,  $u^{t+1}$  and  $f$ . The required videomemory size limits the dimension of problems that GPU might solve. However as the Eq. 3 is almost identical to the iteration of a regular explicit finite-difference method for a non-stationary heat equation, the previously used pyramid method may also be used for this iteration [5].

This method assumes RAM as a main storage for data arrays while the grid is tiled into intersected blocks which are processed sequentially, each of them is copied to videomemory from RAM, a computing CUDA-kernel is invoked at GPU few times to perform a few time-steps of finite-difference method and then the resulting data is copied back to RAM. The number of time-steps performed in a row for a single block is called 'the pyramid height'  $n$ . The intersection between neighbouring tiles has  $2n$  grid nodes in width. Obviously the values that fall into this intersection will be computed independently in both tiles.

The pyramid height is a parameter of decomposition. If it is too small then the algorithm will spend too much time on data transfers. If the pyramid height is too large then the intersection between tiles will lead to an excessive amount of duplicating computations. It seems reasonable to choose the pyramid height that would lead to minimal overall execution time.

As it has been shown (see Eq. 3, page 752 of [5]), the run time in a case of 1D tiling could be estimated as  $\tau \approx K(I-1)^3 \frac{R-n}{R-2n} \left( \frac{2}{n} \tau_c + \tau_a \right)$ , where  $\tau$  is an overall execution time,  $K$  is the number of time steps,  $n$  is pyramid height,  $N$  is grid size in

one dimension (the grid is assumed to have equal size in all dimensions),  $R$  is a tile-basement width,  $\tau_c$  is an average data-transfer duration per a single value from RAM to GPU memory,  $\tau_a$  is an average duration of computing a single value. The last two constants could be measured empirically by performing computations for a small-dimension problems and measuring results with a tool like NVidia Visual Profiler.

Taking into account that the Jacobi iteration employs 3 matrices ( $u^t$ ,  $u^{t+1}$  and  $f$ ) instead of 2 used in explicit finite-difference method, the estimation should be modified as follows:

$$\tau \approx K(I-1)^3 \frac{R-n}{R-2n} \left( \frac{3}{n} \tau_c + \tau_a \right).$$

Finding an optimal pyramid height  $n$  is a non-linear optimisation problem. As the number of possible  $n$  values doesn't exceed  $R/2$ , this problem could be solved fast enough by bruteforcing all the possible  $R/2$  values. However it is possible to find minimum with algebraic solution but brute-force is much simpler in practice. The computational complexity of this brute-force is  $O(R)$ , with  $R$  in practice being in range from 10 to 500. Thus the computation of an optimal pyramid height would not exceed few milliseconds which is absolutely minor in comparison with the total computation duration (minutes and hours).

A significant difference between the Jacobi iteration and the explicit finite-difference iteration should be mentioned: for a non-stationary heat equation there was a fixed number of time-steps (iterations)  $K$ , while Jacobi method termination condition has a form of  $\|u^{k+1} - u^k\| < \varepsilon$ . It means that the number of Jacobi method iterations is not predefined.

Practically the optimal pyramid height is a relatively small number (hundreds) while the number of iterations for Jacobi method to reach a reasonable error is an order of magnitude higher. The authors propose to limit the value of  $n$  to a reasonable maximal number (e.g. 100) and to check the termination condition every  $n$  iterations (see line 2 of the algorithm 1).

**Algorithm 1.** Pyramid method for Jacobi

1.  $u := u_0$
2.  $n := \min(\text{maxPyramidHeight}; \text{optimalPyramidHeight})$
3.  $\text{blocks} := \text{buildPyramids}(n; N)$
4. *repeat*
5.  $u_n := \text{zeroMatrix}(N)$
6. *for*  $i = 1 : \text{size}(\text{blocks})$  *do*
7.  $u_{gpu} = \text{copyRamToGpu}(u(\text{addGhostZone}(\text{blocks}(i); n)))$
8.  $f_{gpu} = \text{copyRamToGpu}(f(\text{addGhostZone}(\text{blocks}(i); n)))$
9. *for*  $i = 1 : n$  *do*
10.  $\text{tmp}_{gpu} := \text{JacobiKernel}(u_{gpu}; f_{gpu})$

11.  $swap(tmp_{gpu}; u_{gpu})$
12. *endfor*
13.  $u_n(blocks(i)) := copyGpuToRam(u_{gpu})$
14. *endfor*
15.  $swap(u; u_n)$
16. *until*  $\|u_n - u\|_\infty < \varepsilon$

Proposed algorithm 1 uses three temporary GPU arrays of size  $R \times N \times N$ . JacobiKernel is a GPU kernel that performs Jacobi iteration at a given data arrays. All other operations are performed at CPU and all other variables are stored in RAM.

The main kernel JacobiKernel is a typical implementation of iterative stencil loop (ISL), its algorithm is listed in algorithm 2. Each GPU thread executing this kernel processes  $R - 2$  values. This kernel has high theoretical occupancy on the hardware authors use and tends to be arithmetic-bound according to NVidia Visual Profiler data.

## Results

Authors have developed a program for above-described algorithm and performed few series of experiments with different data sets. The experiments were carried out on a node of K100 cluster 1 equipped with  $3 \times$  NVidia Tesla C2050 GPU (448 CUDA cores, 2.5 GB of memory), 96 GB RAM and  $2 \times$  6-core CPU Xeon X5670. The testing stand was running CentOS 5.5, CUDA toolkit 6.5 was used.

According to the results in Table 1, the pyramid method gives a significant performance benefit over pure-CPU performance even though the GPU-dedicated memory doesn't allow to fit  $u$  and  $f$  matrices at once.

**Table 1.** Results of the Pyramid method for Jacobi

| Grid size N | Data size, GB | # iterations | Method duration, sec |                |
|-------------|---------------|--------------|----------------------|----------------|
|             |               |              | CPU, 12 threads      | Pyramid method |
| 800         | 5.7           | 8300         | 9687                 | 1353           |
| 900         | 8.1           | 11200        | 20003                | 2671           |
| 1000        | 11.2          | 12300        | 33543                | 4218           |

The pyramid method allows to apply GPU for explicit finite-difference method for stationary PDEs with a grid size exceeding the available GPU-dedicated memory size. At the same time its performance is significantly better than that of pure-CPU implementation (the speedup is about 7–8 times).

**Acknowledgments**

This work has been supported by the grants of Russian Foundation of Basic Research 14-07-00291, 14-01-31305 and 14-07-31178.

**References**

1. **Golovashkin DL, Vorotnikova DG, Kochurov AV and Malisheva SA.** Solving finite-difference equations for diffractive optics problems using graphics processing units. *Opt. Eng.*, 2013; 52 (9): 091719. doi: 10.1117/1.OE.52.9.091719.
2. **Yakimov PYu.** Preprocessing of digital images in systems of location and recognition of road signs. *Computer Optics*, 2013; 37(3): 401-405. [in Russian]
3. **Pavelyev VS, Karpeev SV, Dyachenko PN, Miklyaev YV.** Fabrication of three-dimensional photonics crystals by interference lithography with low light absorption. *J. Modern Optics*, 2009; 9 (56): 1133–1136.
4. **Micikevicius P.** Multi-GPU Programming for Finite Difference Codes on Regular Grids, Stanford AHPARC/iCME Colloquium Series, 2012 <http://www.stanford.edu/dept/ICME/docs/seminars/Micikevicius-2012-01-23.pdf>
5. **Golovashkin DL, Kochurov AV.** Solution of difference equations on GPU. Pyramids method. *Computational Technologies*, 2012; 17(3): 39-52. [In Russian]