Mokslas – Lietuvos ateitis / Science – Future of Lithuania
ISSN 2029-2341 / eISSN 2029-2252
2020 Volume 12, Article ID: mla.2020.13500, 1–5
https://doi.org/10.3846/mla.2020.13500

Information technologies and multimedia
Informacinės technologijos ir multimedija

COMPARISON OF GPU AND CPU EFFICIENCY WHILE SOLVING HEAT CONDUCTION PROBLEMS

Julija SEMENENKO, Aliaksei KOLESAU, Vadimas STARIKOVIČIUS, Artūras MACKŪNAS*, Dmitrij ŠEŠOK

Vilnius Gediminas Technical University, Vilnius, Lithuania

Received 03 September 2020; accepted 11 November 2020

Abstract. Overview of GPU usage while solving different engineering problems, comparison between CPU and GPU computations and overview of the heat conduction problem are provided in this paper. The Jacobi iterative algorithm was implemented by using Python, TensorFlow GPU library and NVIDIA CUDA technology. Numerical experiments were conducted with 6 CPUs and 4 GPUs. The fastest used GPU completed the calculations 19 times faster than the slowest CPU. On average, GPU was from 9 to 11 times faster than CPU. Significant relative speed-up in GPU calculations starts when the matrix contains at least 400^2 floating-point numbers.

Keywords: CUDA, GPU, Jacobi iterative algorithm, parallel computing, heat conduction problem.

Introduction

The first generation of Graphics Processing Units (GPUs) was created at the end of the 20th century to fulfil the demands of computer games. Starting from shadowing algorithms, like Shadow Mapping (Williams, 1978) and Shadow Volume (Crow, 1977), devices and coding possibilities were becoming more and more sophisticated to give birth for the second generation of GPUs with shaders, small programs, consisting of 20 lines of GPU assembler code. Loops, branching, etc., made their way to the GPU programming with the understanding that GPU can not only be used for game graphics but is also a powerful calculation tool that allows reducing the execution time of computationally intensive applications.

General-Purpose computing on Graphics Processing Units (GPGPU) is now supported by many platforms. GPU manufacturers, NVIDIA, and AMD provide necessary functions and libraries to enable GPU calculations. These calculations can be performed on GPU only if it's possible to split a problem into smaller parts which can be solved concurrently. Also, it is important to mention that on average CPUs are usually more efficient than GPUs when data size isn't big enough to effectively use all GPU cores. The primary task of this paper is to compare GPU and CPU calculations efficiency while solving heat conduction problems with different amount of data.

1. Prior and related works

GPUs are widely used in Machine Learning because they allow teaching the models concurrently.

Kuckuk and Köstler (2018) used GPU to model shallow water equation which allowed to calculate large, time-consuming systems via Piz Daint supercomputer. Filonenko et al. (2018) applied GPU to detect fumes from a real-time camera. Lu et al. (2019) used GPU to serve the medical Drug-Drug Interaction (DDI) system, which collects information from 150,000 publication-wide PubMed database. Warrena et al. (2019) enhanced Finite-Difference Time-Domain (FDTD) electromagnetic modelling. Fambrini et al. (2018) used GPU calculations for the JSEG algorithm optimization.

Bohacek et al. (2019) used CUDA to solve the inverse heat conduction problem. They suggested 3 solutions and made a comparison with classical OpenFOAM (FDIC) and ANSYS Fluent (AMG). GPU solution appeared to be the best one and increases the calculations speed up to 15 times.

*Corresponding author. E-mail: *amackunas.research@gmail.com*

2. Heat conduction problem

The heat conduction problem arises when a body isn't equally heated. The heat equation allows us to find the temperature at each point of the observed body at the specified point in time. This problem has 3 types based on the dimensions of the body:

- One-dimensional, where only x coordinate of a uniform rod and time are used: $T = f(x, t)$;
- Two-dimensional, where the heated object is planar and x and y coordinates are used accordingly: $T = f(x, y, t)$ (Figure 1);
- Three-dimensional, where space bodies are heated: $T = f(x, y, z, t)$.

The heat conduction problem has 2 more types:

- Stationary – temperature does not depend on time (when the thermal field does not evolve over time). The main purpose of those problems is to find the temperature at each point of the body;
- Non-stationary – temperature is not constant over time. The task is to determine how temperature is changing for each point of the body.

In this paper, the stationary two-dimensional problem is solved.

2.1. Heat equation

The following equation was named after Poisson and is widely used in Physics to calculate different potential fields, for example, electric, pressure, etc.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \qquad (x,y) \in (0,1)\times(0,1), \tag{1}$$

$$u(x,y) = \mu(x,y) \qquad (x,y) \in \gamma, \tag{2}$$

where $u(x, y)$ is the temperature at the point (x, y), γ marks a boundary, $\mu(x, y)$ is the temperature at the boundary point (x, y), $f(x, y)$ defines a heat source.

The equation is solved by the finite-difference method. To use approximation by finite differences, a uniform discrete grid for this problem has been chosen:

$$w_h = \{(x_i, y_j) : x_i = ih, y_j = jh, \ \ 0 \le i, j \le N\},$$

where $h = 1/N$ is a grid step.

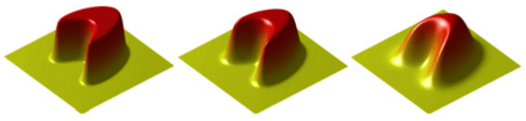

Figure 1. Two-dimensional example of the heat conduction problem. Here, the height and the colour define the temperature at that point: high and red means hot, low and green means cold. Notice, how the displayed object is "melting" from the state a) to the state c) – the heated area is cooling and passing the heat to the nearby area (the images are taken from https://en.wikipedia.org/wiki/Heat_equation#/media/File:Heat_eqn.gif)

A discrete solution $U_{ij} = U(x_i, y_j)$ needs to be found. The temperature at the boundary grid points is calculated by the (2) equation. In order to calculate the temperature at the inner points, the (1) differential equation at each point is replaced by the algebraic equation. It is achieved by approximating the derivatives with finite differences which are calculated by using three-point stencil method in vertical and horizontal directions (Figure 2).

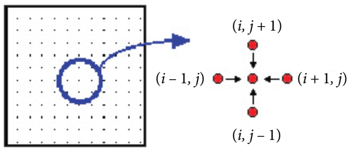

Figure 2. Discrete grid and scheme stencil

Thus, the system of linear equations is the following:

$$\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h^2} = f_{i,j},$$

$$1 \le i, j \le N-1. \tag{3}$$

The system is made up of $(N - 1)^2$ equations.

2.2. Jacobi method

Eventually, to solve the heat conduction problem there is a need to solve the system of linear equations. This can be done in many ways, but the Jacobi method has been selected for this paper.

Jacobi method is an iterative algorithm for determining the solutions of a diagonally dominant system of linear equations. Each element is calculated approximately by using this equation:

$$U_{i,j} = \frac{1}{4}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} - h^2 f_{i,j}),$$

$$1 \le i, j \le N-1. \tag{4}$$

The process is iterated until it converges.

Jacobi method converges slower than, for example, Krylov or Gauss-Seidel (Amador & Gomes, 2012). On the other hand, a big advantage of this algorithm is its suitability for concurrent calculations (Margaris et al., 2014) and thus is an effective option for GPU calculations. Created by Jacobi (2009), the algorithm started to be used only a hundred years later when computers were invented.

3. Experiments

The code has been written in Python, using Numpy and TensorFlow GPU libraries. Mainly, NVIDIA GPUs were used. To run TensorFlow GPU on NVIDIA, Compute Capability of processing unit must be 3.0 or higher, CUDA and cuDNN (NVIDIA CUDA Deep Neural Network) have to be installed.

One of the TensorFlow GPU advantages is that code is suitable not only for NVIDIA but also for the AMD devices. If the TensorFlow ROCm library is installed instead of

TensorFlow GPU, the same code can be run without making any changes. Same with CPU and GPU – the selected device is passed to the function as a parameter and no code changes are needed.

Two experiments were conducted: a heat conduction problem with and without the heat source.

3.1. Equipment

GPUs used to conduct the experiments:

1. AMD Radeon™ RX VEGA 56;
2. NVIDIA GeForce GTX 1060;
3. NVIDIA GeForce GTX 860M;
4. NVIDIA Tesla M40.

CPUs used to conduct the experiments:

1. Intel® Core™ i7-3630QM;
2. Intel® Core™ i7-4720HQ;
3. Intel® Core™ i7-6700K;
4. Intel® Core™ i7-7700;
5. Intel® Core™ i7-7820HQ.

The source code used in the experiments can be found at https://github.com/kolesov93/tf_jacobi/tree/master.

In order to get the most from a vectorized computation both on CPU and GPU, the matrix form of the computation was chosen. The form of a single equation (4) can be reformulated in the matrix form if we "extract" 4 submatrices from *u* (*L*: without a left column, *R*: without a right column, *T*: without a top row, *B*: without a bottom row). Then the equation becomes

$$u' = \frac{1}{4}(L + R + T + B - f \cdot h^2)$$

with appropriate padding. See *build_iteration* method in the source code for implementation details.

Note that TensorFlow doesn't recompute $f \cdot h^2$ each iteration due to the constant propagation technique.

3.2. First experiment: heat conduction problem without a heat source

First experiment parameters:

1. N, where N^2 is a grid size. $N = 100i$, where $i = [1 .. 10]$, later $N = 1000j$, where $j = [1 .. 10]$;
2. Boundary condition: $\mu(x, y) \equiv 1$;
3. Without a heat source: $f(x, y) \equiv 0$;
4. Accuracy of the calculations: $\varepsilon = 2 \cdot 10^{-5}$;
5. Device: CPU or GPU.

We see in Table 1 and Figure 3 that if $N < 400$, the CPU solves this problem faster than the GPU as there is no need for the CPU to transfer data and the calculations are faster up to 2 times but it is just several seconds. The bigger the *N*, the faster the GPU solves this problem compared to the CPU. Maximum GPU speed up for $N = [100; 1000]$ is up to 6.4 times. Figure 4 shows the CPU and GPU calculation speed comparison when N is between 500 and 1000.

Experiment with $N = 1000$ for the first experiment was conducted with several different devices. Out of these, even the slowest NVIDIA GTX 860M GPU solves the problem 1.3 times faster than the fastest Intel i7-6700K CPU. The fastest GPU solves the problem 19.6 times faster than the slowest CPU. On average, GPU devices solve this problem 5.9 times faster than CPU devices. Tables 2 and 3 show average calculation time for CPU and GPU devices when $N = 1000$.

Table 1. First experiment. NVIDIA GeForce GTX 1060 GPU and Intel i7-7700 CPU

N	CPU, s	GPU, s
100	2.6	4.5
200	9.2	11.9
300	18.7	20.7
400	19.7	17.6
500	23.4	20.6
600	105.7	24.9
700	147.9	29.0
800	186.2	35.3
900	237.7	40.2
1000	302.6	46.8

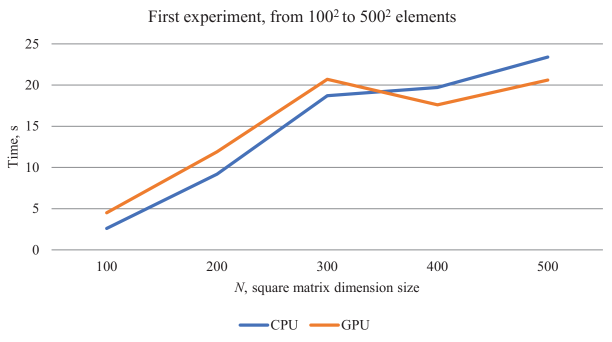

Figure 3. GPU and CPU solving times for the first experiment with $N = [100; 500]$

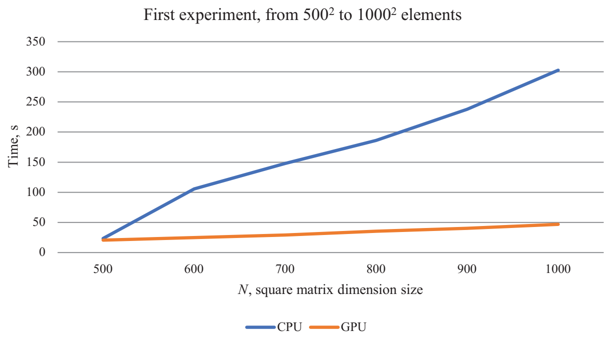

Figure 4. GPU and CPU solving times for the first experiment with $N = [500; 1000]$

Table 3. First experiment calculation time comparison with different GPU devices when $N = 1000$

GPU	NVIDIA Tesla M40	NVIDIA GTX 1060	AMD RX VEGA 56	NVIDIA GTX 860M
Time, s	42.00	46.82	79.12	85.00

If the grid size is further increased to $N = 5000$, the problem is solved in ~3 hours with the CPU and in ~16 minutes with the GPU. Thus, the GPU is 11 times faster than the CPU with this grid size (Table 4 and Figure 5).

Maximum calculations speed increase for [1000; 10000] range is 11.7 times. With $N = 10000$ CPU needed ~12h and GPU ~1h to solve the problem (Figure 6).

NVIDIA GTX 1060 belongs to gaming series. The calculation would be faster with a specialized unit from Tesla or Quadro series, for example, Tesla M40.

Table 4. First experiment. NVIDIA GTX 1060 GPU and Intel i7-7700 CPU for N = [1000; 10000]

N	CPU, s	GPU, s
1000	302.6	46.8
2000	1507.1	166.5
3000	4039.6	356.3
4000	7215.7	617.8
5000	11157.3	957.0
6000	15920.6	1384.1
7000	21837.9	1867.9
8000	28295.8	2445.0
9000	35861.9	3091.2
10000	44406.1	4297.4

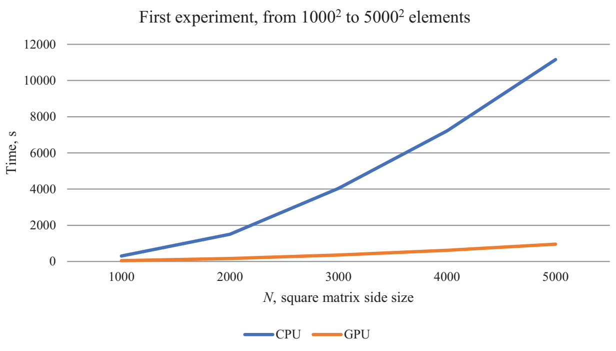

Figure 5. GPU and CPU solving times for the first experiment with N = [1000; 5000]

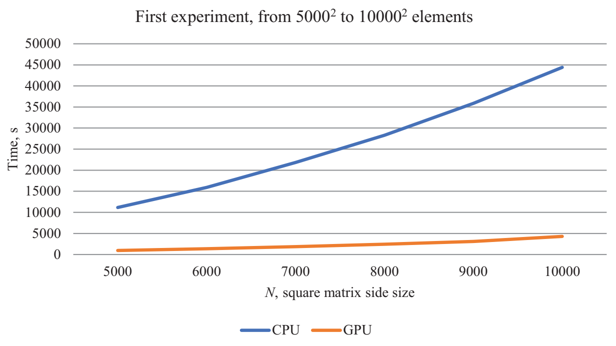

Figure 6. GPU and CPU solving times for the first experiment with N = [5000; 10000]

3.3. Second experiment: heat conduction problem with a heat source

Second experiment parameters:

- N, where N^2 is a grid size. $N = 100i$, where i = [1; 10];
- Boundary condition: $\mu(x, y) \equiv 0$;
- With a heat source: $f(x, y) = -exp(-10((x - 0.5)^2 + (y - 0.5)^2))$;
- Accuracy of the calculations: $\varepsilon = 10^{-7}$;
- Device: CPU or GPU.

The second experiment is very similar to the first one. However, now a heat source function is added. This increases the number of calculations per step. Also, the accuracy of the calculations is lower so that the calculations run longer and the comparison can also be run more precisely.

We see in Table 5 and Figure 7 that the CPU is faster than the GPU only for $N < 300$. The higher amount of actions means that calculation time is longer for both CPU and GPU. For example, when $N = 100$ the calculations of the second experiment run ~1.5 times longer on both devices compared to the first experiment.

The proportion of CPU/GPU calculation duration of the first and second experiments is very similar. For example, if $N = 1000$ then the calculations on NVIDIA GTX 1060 are 6.3 times faster than on Intel i7-7700 even if the calculations ran 8.5 times longer compared to the first experiment (Figure 8).

Table 5. Second experiment. GTX1060 GPU and Intel i7-7700 CPU for N = [100; 1000]

N	CPU, s	GPU, s
100	3.8	7.2
200	16.5	20.9
300	42.5	39.6
400	81.1	75.5
500	181.6	121.0
600	765.7	187.5
700	1185.3	248.5
800	1598.2	297.7
900	2077.8	362.9
1000	2584.5	407.4

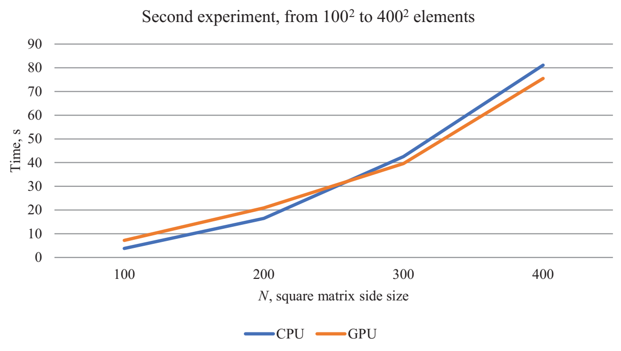

Figure 7. GPU and CPU solving times for the second experiment with N = [100; 400]

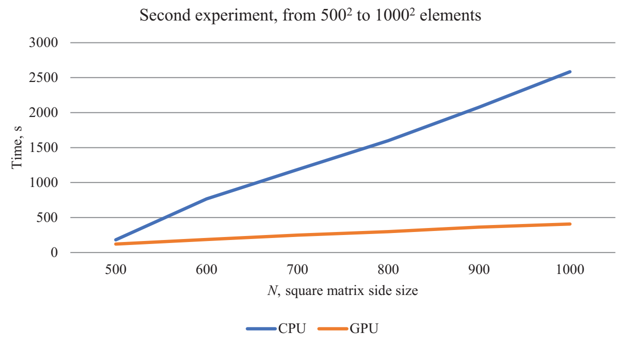

Figure 8. GPU and CPU solving times for the second experiment with N = [500; 1000]

4. Results comparison

The results of these experiments are similar to (Bohacek et al., 2019) – while solving small matrices CPU was 2 times faster than GPU. In the paper they also mention that by using commercial packages like ANSYS FLUENT 14.5 or OpenFOAM for small matrices the calculations can be performed from 40 to 50 times faster than by using simple CPU code. Since these packages support parallelism, they are also great for bigger grids. We have compared GPU and CPU calculation speed without using the specialized packages and achieved an average increase of 10 times in calculation speed for the same code with different amount of data and different processing units.

Conclusions

The more data has to be processed, the more efficient GPU calculations will be compared to CPU. After increasing the size of the grid, the calculations on NVIDIA GTX 1060 ran up to 11.7 times faster than on Intel i7-7700. For more accurate results, more actions have to be performed with data like uploading it to GPU memory before the calculations and moving the results back after the calculations. While using a heat source function GPU calculations became more efficient than CPU when the number of matrix elements reached 300^2 contrary to 400^2 without the heat source. With a small amount of data, CPU calculations might be a better option because CPU usually consists of several cores and support parallelism.

References

Amador, G., & Gomes, A. (2012). *Linear solvers for stable fluids: GPU vs CPU.* https://www.it.ubi.pt/17epcg/Actas/artigos/17epcg_submission_39.pdf

Bohacek, J., Kharicha, A., Ludwig, A., Wu, M., Holzmann, T., & Karimi-Sibaki, E. (2019). A GPU solver for symmetric positive-definite matrices vs. traditional codes. *Computers & Mathematics with Applications*, *78*(9), 2933–2943. https://doi.org/10.1016/j.camwa.2019.02.034

Crow, F. (1977). Shadow algorithms for computer graphics. *ACM SIGGRAPH Computer Graphics*, *11*(2), 242–248. https://doi.org/10.1145/965141.563901

Fambrini, F., Iano, Y., Caetano, D. G., Rodriguez, A. A. D., Moya, C., Carrara, E., Rangel, A., Cabello, F. C., Zubem, J. V., del val Cura, L. M., Destro Filho, J. B., Campos, J. R., & Saito, J. H. (2018). GPU Cuda JSEG Segmentation Algorithm associated with Deep Learning Classifier for Electrical Network Images Identification. *Procedia Computer Science*, *126*, 557–565. https://doi.org/10.1016/j.procs.2018.07.290

Filonenko, A., Hernández, D. C., & Jo, K.-H. (2018). Fast smoke detection for video surveillance using CUDA. *IEEE Transactions on Industrial Informatics*, *14*(2), 725–733. https://doi.org/10.1109/TII.2017.2757457

Jacobi, C. G. (2009). Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. *Crelle's Journal*, *1846*(30), 51–94. https://doi.org/10.1515/crll.1846.30.51

Kuckuk, S., & Köstler, H. (2018). Whole program generation of massively parallel shallow water equation solvers. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)* (pp. 78–87). IEEE. https://doi.org/10.1109/CLUSTER.2018.00020

Lu, Y., Ramachandra, A. C., Pham, M., Tu, Y.-C., & Cheng, F. (2019). CuDDI: A CUDA-based application for extracting drug-drug interaction related substance terms from PubMed literature. *Molecules*, *24*(6), Article 1081. https://doi.org/10.3390/molecules24061081

Margaris, A., Souravlas, S., & Roumeliotis, M. (2014). *Parallel implementations of the Jacobi linear algebraic systems solve* [Conference presentation]. Balkan Conference of Informatics (BCI2007), Sofia, Bulgaria.

Warrena, C., Giannopoulos, A., Gray, A., Giannakis, I., Patterson, A., Wetter, L., & Hamrah, A. (2019). A CUDA-based GPU engine for gprMax: Open source FDTD electromagnetic simulation software. *Computer Physics Communications*, *237*, 208–218. https://doi.org/10.1016/j.cpc.2018.11.007

Williams, L. (1978). Casting curved shadows on curved surfaces. *ACM SIGGRAPH Computer Graphics*, *12*(3), 270–274. https://doi.org/10.1145/965139.807402

GPU IR CPU EFEKTYVUMO PALYGINIMAS SPRENDŽIANT ŠILUMOS LAIDUMO UŽDAVINIUS

J. Semenenko, A. Kolesau, V. Starikovičius, A. Mackūnas, D. Šešok

Santrauka

Šiame straipsnyje apžvelgtas GPU taikymas įvairiems inžineriniams uždaviniams spręsti, palyginti skaičiavimai naudojant CPU ir GPU, aprašytas šilumos laidumo uždavinys. Įgyvendintas Jakobio metodas naudojant „Python“, „TensorFlow GPU“ biblioteką ir NVIDIA CUDA technologijas. Atlikti skaitiniai eksperimentai naudojant šešis CPU ir keturis GPU įtaisus. Greičiausias nagrinėtas GPU įvykdė skaičiavimus 19 kartų greičiau negu lėčiausias CPU. Naudojant GPU, vidutiniškai skaičiavimai buvo atliekami nuo 9 iki 11 kartų greičiau nei su CPU. Didelis santykinis GPU pagreitėjimas vyko, kai lygiagrečiai buvo apdorojama bent 400^2 realiųjų skaičių.

Reikšminiai žodžiai: CUDA, GPU, Jakobio metodas, lygiagretieji skaičiavimai, šilumos laidumo uždavinys.