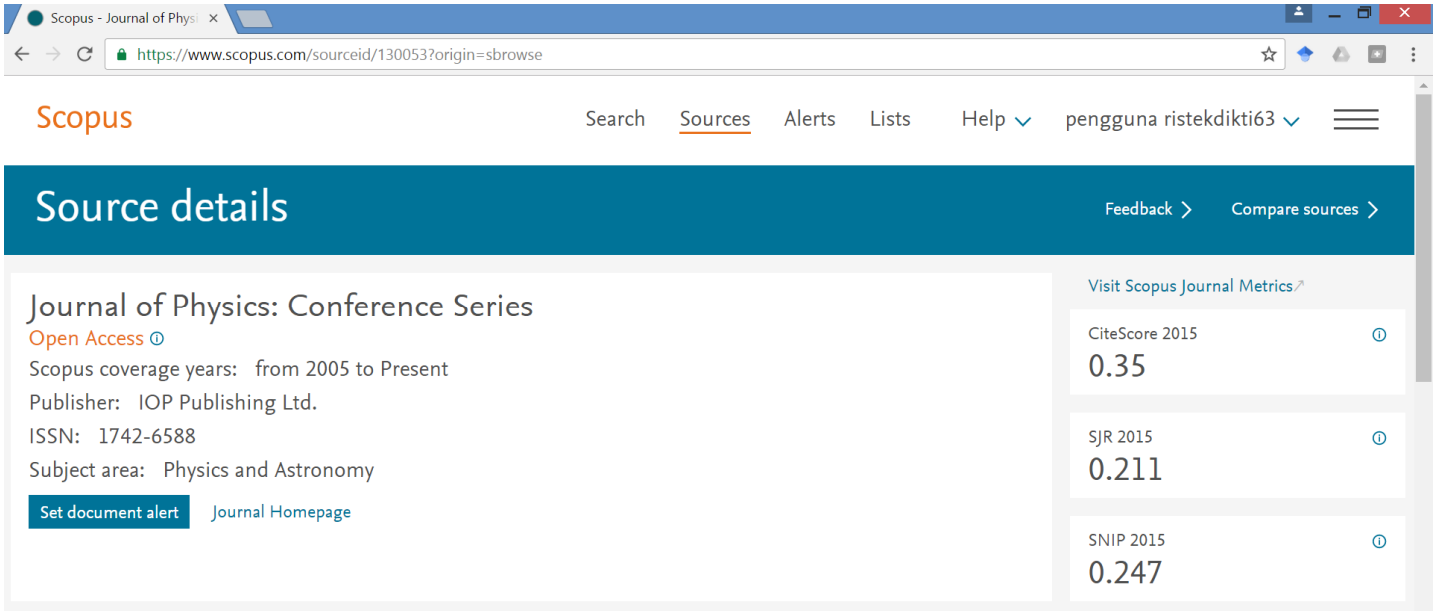
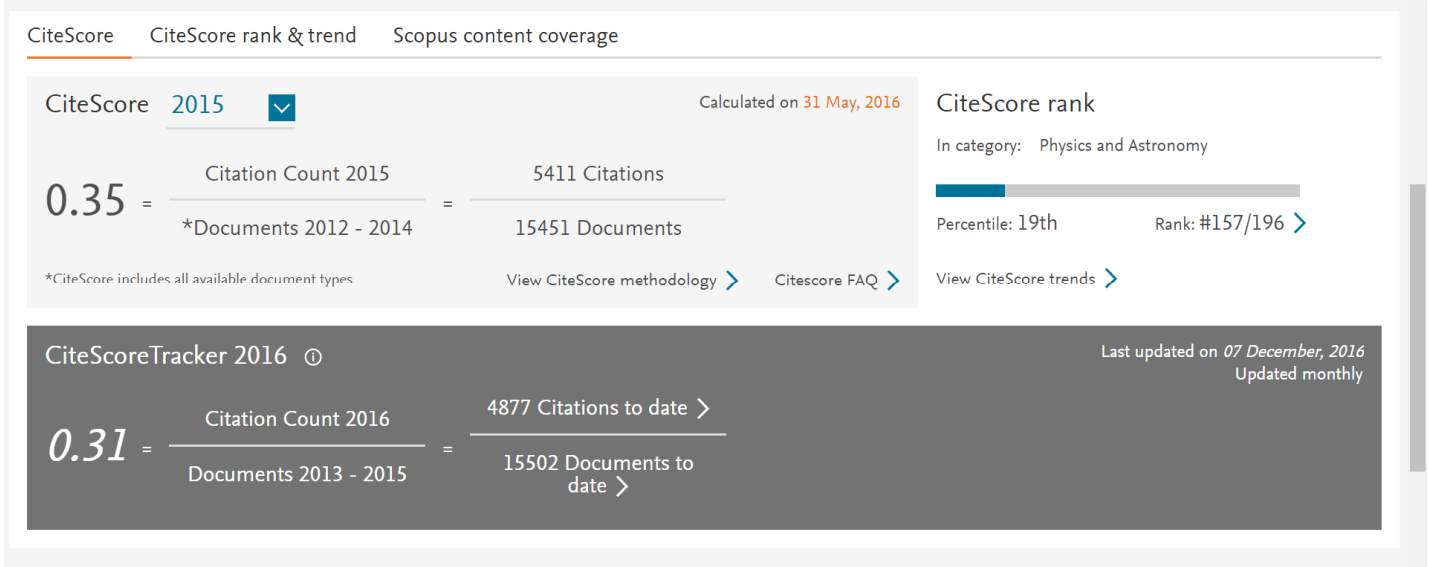

Scopus - Journal of Phys
https://www.scopus.com/sourceid/130053?origin=sbrowse
Scopus
Search
Sources
Alerts
Lists
Help
pengguna ristekdikti63
Source details
Feedback
Compare sources
Journal of Physics: Conference Series
Open Access
Scopus coverage years: from 2005 to Present
Publisher: IOP Publishing Ltd.
ISSN: 1742-6588
Subject area: Physics and Astronomy
Set document alert
Journal Homepage
Visit Scopus Journal Metrics
CiteScore 2015
0.35
SJR 2015
0.211
SNIP 2015
0.247


CiteScore
CiteScore rank & trend
Scopus content coverage
CiteScore 2015
Calculated on 31 May, 2016
0.35 = Citation Count 2015 / *Documents 2012 - 2014 = 5411 Citations / 15451 Documents
*CiteScore includes all available document types
View CiteScore methodology
Citescore FAQ
CiteScore rank
In category: Physics and Astronomy
Percentile: 19th
Rank: #157/196
View CiteScore trends
CiteScoreTracker 2016
Last updated on 07 December, 2016
Updated monthly
0.31 = Citation Count 2016 / Documents 2013 - 2015 = 4877 Citations to date / 15502 Documents to date

Performance of parallel computation using CUDA for solving the one-dimensional elasticity equations

2017 J. Phys.: Conf. Ser. 801 012080

(http://iopscience.iop.org/1742-6596/801/1/012080)

Performance of parallel computation using CUDA for solving the one-dimensional elasticity equations

J B B Darmawan[1] and S Mungkasi[2]

[1]Department of Informatics, Faculty of Science and Technology,
Sanata Dharma University, Yogyakarta, Indonesia
[2]Department of Mathematics, Faculty of Science and Technology,
Sanata Dharma University, Yogyakarta, Indonesia

E-mail: b.darmawan@usd.ac.id, sudi@usd.ac.id

Abstract. In this paper, we investigate the performance of parallel computation in solving the one-dimensional elasticity equations. Elasticity equations are usually implemented in engineering science. Solving these equations fast and efficiently is desired. Therefore, we propose the use of parallel computation. Our parallel computation uses CUDA of the NVIDIA. Our research results show that parallel computation using CUDA has a great advantage and is powerful when the computation is of large scale.

1. Introduction

A large number of real world problems can be modelled mathematically. Solutions to mathematical models are representatives of the solutions to the real problems. Some mathematical models are in the form of differential equations.

In this paper, we solve a system of partial differential equations. In particular, we consider the nonlinear elasticity equations. These equations were derived from engineering problems relating to elastic wave propagation through heterogeneous media [1].

Some work has been done previously regarding the nonlinear elasticity equations. Supriyadi and Mungkasi [2] solved the nonlinear elasticity equations in a sequential computation. Solving the problem in sequential is not appropriate for a large-scale problem, as the computation is tedious for the large-scale problem. Darmawan and Mungkasi [3] solved the nonlinear elasticity equations in a parallel computation using the MPI with a cluster of workstations applying MPI_Send and MPI_Recv techniques. However, parallel programming with MPI_Send and MPI_Recv techniques in the MPI standard is not appropriate to be used in the one-dimensional elasticity equations. This is because the total time needed in the data exchange is considered more dominant compared with the total amount of time to conduct the basic elasticity computation using the finite volume method [4].

Nowadays, Graphic Processing Units (GPUs) have better performance than CPUs in both floating point operation and memory bandwidth. One of computing platforms and programming models in GPUs is Compute Unified Device Architecture (CUDA) [5]. GPUs with CUDA offer high performance at a very low cost, and they can also be integrated into high performance computer systems [6-7]. This paper investigates if we obtain high speed in parallel computations using CUDA to solve elasticity problems.

This paper is organized as follows. We recall the mathematical model concerning elasticity problems and present the numerical scheme to solve the model in Section 2. We describe the

numerical method in parallel to solve elasticity problems in Section 3. Computational results and discussion are provided in Section 4. Finally, we draw some concluding remarks in Section 5.

2. Mathematical model and numerical scheme

In this section, we present the mathematical model to be solved and the numerical scheme used to solve the model.

The nonlinear elasticity equations are given by

$$\varepsilon_t - u_x = 0, \tag{1}$$

$$u_t - \sigma(\varepsilon)_x = 0. \tag{2}$$

In this model, the free variables are time t and the space x. In addition, the notation $\varepsilon = \varepsilon(x,t)$ is the strain, $u = u(x,t)$ represents the velocity, and $\sigma(\varepsilon)$ denotes the stress. The space domain that we consider in this paper is $0 \le x \le 100$. The initial condition for our test problem is

$$\varepsilon(x,0) = 0, \tag{3}$$

$$u(x,0) = 0, \tag{4}$$

for all x. The boundary condition at $x = 0$ and $x = 100$ is

$$\varepsilon(0,t) = 0, \tag{5}$$

$$u(0,t) = \begin{cases} -0.4\left(1 + \cos\left(\dfrac{(t-30)\pi}{30}\right)\right) & \text{if} \quad t \le 60, \\ 0 & \text{if} \quad t > 60, \end{cases} \tag{6}$$

$$\varepsilon(100,t) = 0, \tag{7}$$

$$u(100,t) = 0. \tag{8}$$

The system of equations (1)-(2) have the form of a conservation law

$$q_t + f(q)_x = 0, \tag{9}$$

where $q = q(x,t)$ is the conserved quantity and $f(q)$ is the flux function. One of numerical methods that can be used to solve conservation laws is the finite volume method. The finite volume method itself is conservative, as the numerical quantity is conserved at any time.

The finite volume scheme for equation (9) in the fully discrete version is

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^n - F_{i-1/2}^n), \tag{10}$$

where Q_i^n is the approximate quantity computed in the finite volume frame work at the i-th cell at the n-th time step. In addition, the variable $F_{i+1/2}^n$ denotes the approximate flux at the $(i+1/2)$ interface from the n-th time step to the $(n+1)$-th time step. The notations Δt and Δx are the time step and the

spatial cell width respectively. We use a uniform time step as well as a uniform spatial cell width. All fluxes are computed using the Lax-Friedrichs formulation. Lax-Friedrichs fluxes for equation (9) relating the finite volume scheme (10) are given by

$$F^n_{i+\frac{1}{2}} = \frac{1}{2}[f(q^n_{i+1}) + f(q^n_i)] - \frac{\Delta x}{2\Delta t}(q^n_{i+1} - q^n_i), \tag{11}$$

and

$$F^n_{i-\frac{1}{2}} = \frac{1}{2}[f(q^n_i) + f(q^n_{i-1})] - \frac{\Delta x}{2\Delta t}(q^n_i - q^n_{i-1}). \tag{12}$$

3. Method for parallel computations

The method for our parallel computations is described as follows.

NVIDIA developed the CUDA programming model and computing platform to let programmers write scalable parallel codes [5]. CUDA is an extension of the C and C++ programming languages. The programmer writes a serial program that calls parallel kernels, which may be functions or programs. A kernel executes a set of parallel threads. The threads must be organized into hierarchy of grids of thread blocks. A thread block is a set of concurrent threads that share access to a memory space privately to the block and cooperate among themselves through barrier synchronization. A grid is a set of thread blocks that each of them may be executed independently in parallel [8]. In this work, we use CUDA on a Personal Computer and a GPU for parallel programming with the C programming language.

Foster [9] mentioned that the execution time (which is varied with problem size) as well as the efficiency (which is independent of problem size) can be the metrics to evaluate parallel algorithm performance. The relative speed up S_{relative} is calculated as

$$S_{\text{relative}} = \frac{T_1}{T_p}, \tag{13}$$

where the execution time T_1 is used on one processor and the time T_p is used on p processors. The factor by which execution time is reduced on p processors is actually the relative speedup. Relative efficiency E_{relative} is calculated as

$$E_{\text{relative}} = \frac{T_1}{p\,T_p}. \tag{14}$$

Notice that from equations (13)-(14), the relative speedup is in relation with the relative efficiency.

However, according to Tan [6] in GPU parallel computing the execution times are tested on hardware platforms with totally different architectures, namely, the CPU and the GPU. The efficiency is not as a useful metric as in CPU parallel analysis. Therefore, in this paper we evaluate computational results using the execution time and the speedup defined by equation (13) with a fixed p processor equal to the number of GPU cores plus the number of CPU cores.

4. Computational results

In this section we present main results of our research.

Computations are conducted in a Personal Computer with one Core 2 Quad processor with RAM 8 GB and a GPU using GT 730 from NVIDIA with 96 cores, memory 2 GB and maximum of 1024 threads per block. We use the global memory in the GPU to share data between threads. The operating system that we use is Windows 7 on 64 bit. The parallel computing environment is CUDA Toolkit 7.5.

Table 1. Total time (in seconds) and speedup for elasticity simulation scenarios with 1001 threads per block conducted in parallel.

Size of arrays (N)	Sequential execution time (T_1)	Parallel execution time ($T_{p=96\ GPU + 1\ CPU}$)	Speedup ($S_{relative}$)
10001	1.36267	0.76433	0.56091
20001	2.00200	1.52400	0.76124
30001	2.33433	2.28767	0.98001
40001	2.86500	3.08367	1.07632
50001	3.42167	3.81200	1.11408
60001	3.92067	4.60200	1.17378
70001	4.46700	5.39700	1.20819
80001	5.01800	6.16633	1.22884
90001	5.54867	6.94733	1.25207
100001	6.13567	7.73200	1.26017
110001	6.33533	8.51267	1.34368
120001	6.88533	9.29733	1.35031
130001	7.43600	10.11400	1.36014
140001	7.97700	10.87833	1.36371
150001	8.38233	11.66367	1.39146
160001	8.82933	12.46967	1.41230

We have conducted 16 simulations of parallel computations and 16 simulations of sequential computations for running the program in order to solve the elasticity problem mentioned in Section 2. The elasticity problem is based on one-dimensional elasticity equations implemented in one-dimensional array. We use scenarios with increasing size of the arrays from 10001 to 160001 with the step is 10000. We choose 1001 threads per block and calculate the number of blocks to obtain the number of threads at least equal to the dimension of the arrays.

For a basic elasticity problem we record the total time for each simulation in Table 1. As shown in Figure 1 for the total time of both sequential and parallel executions, we observe that more number of size arrays leads to slower computation. For the size of arrays between 10001 and 20001, sequential execution is faster than parallel execution. Surprisingly for the size of arrays between 30001 and 40001 we observe that parallel execution exceeds sequential execution, and parallel execution time continue to increase linearly exceeding the sequential execution time. As shown in Figure 2, for the speedup less than 1, the speedup increases linearly and sharply. However, for the speedup more than 1, the speedup increases linearly and gradually. This indicates that more size of arrays will gain better speedup. The limitation is the number of memory available. It means that for one-dimensional elasticity computations, more size of arrays in the CUDA parallel programming will gain more speedup.

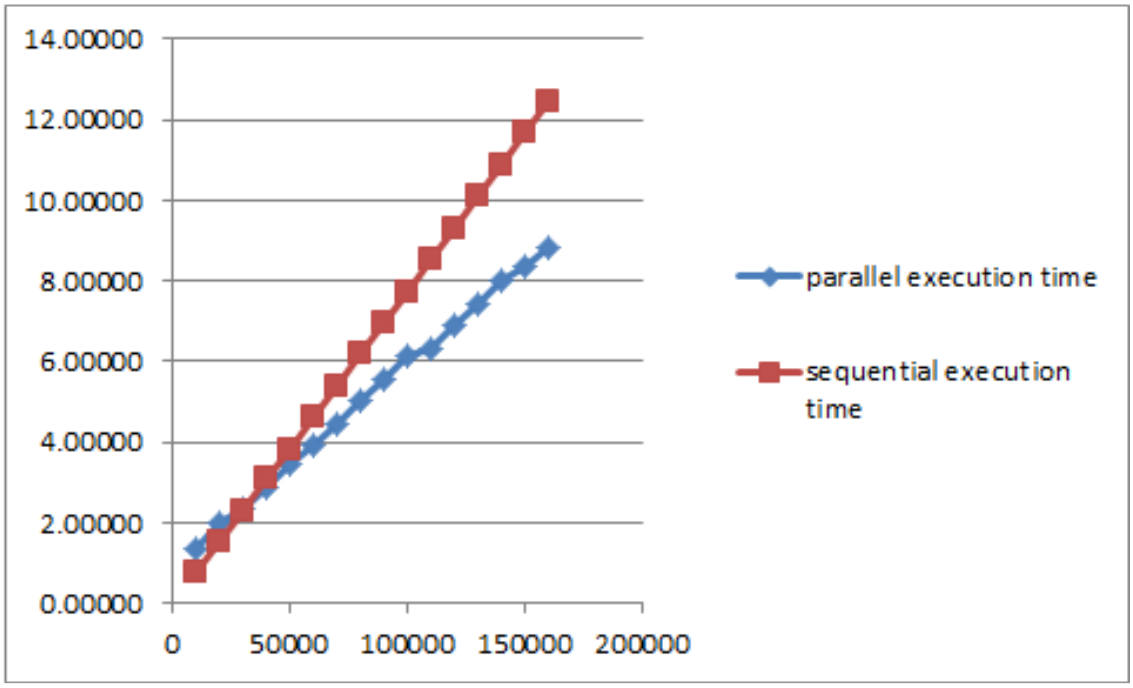


Figure 1. Total time elapsed in a elasticity simulation scenario for several computational settings. The horizontal axis is the size of arrays. The vertical axis is the total time used for computations.

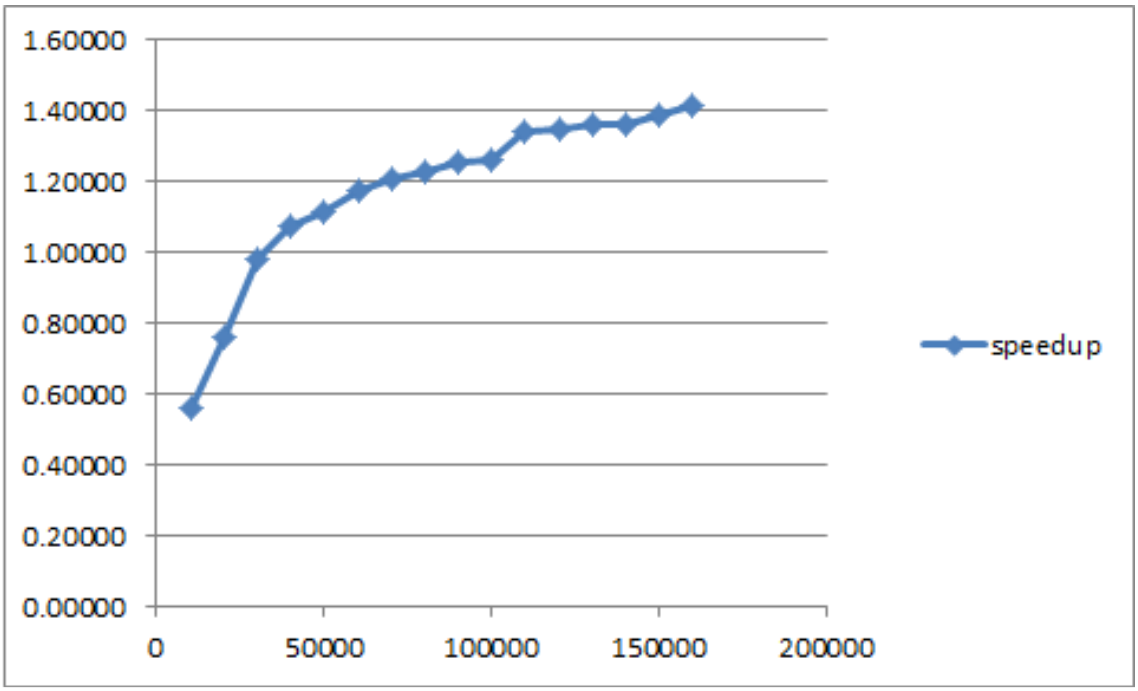


Figure 2. Speedup in an elasticity simulation scenario for several computational settings. The horizontal axis is the sizes of arrays. The vertical axis is the speedup.

5. Conclusion

We have simulated several scenarios of parallel computations for elasticity problems. We obtain that parallel programming with CUDA can be used to improve execution time of computation to solve the one-dimensional elasticity equations within an appropriate array dimension. The improvement of speedup can be obtained when the size of arrays are more than 40001 and it will continue to increase linearly until 160001. Without loss of generality, we recommend that the sizes of arrays in the one-dimensional elasticity computation must be appropriate to obtain improvement of speedup in parallel computations using CUDA.

Acknowledgments

This work was financially supported by the Institute for Research and Community Services of Sanata Dharma University (LPPM USD). The LPPM USD internal research grant year 2016 is gratefully acknowledged by both authors.

References

[1] LeVeque R J 2002 Finite-volume methods for non-linear elasticity in heterogeneous media *International Journal for Numerical Methods in Fluids* **40** 93
[2] Supriyadi B and Mungkasi S 2016 Finite volume numerical solvers for non-linear elasticity in heterogeneous media *International Journal for Multiscale Computational Engineering* **14** 479
[3] Darmawan J B B and Mungkasi S 2016 Parallel computations using a cluster of workstations to simulate elasticity problems *Journal of Physics: Conference Series* accepted
[4] Mungkasi S and Darmawan J B B 2015 Fast and efficient parallel computations using a cluster of workstations to simulate flood flows *Communications in Computer and Information Science* **516** 469
[5] NVIDIA Corp 2016 *CUDA C Programming Guide*. Available online (accessed on 21 November 2016) https:docs.nvidia.com/cuda/cuda-c-programming-guide
[6] Tan Y and Ding K 2016 A survey on GPU-based implementation of swarm intelligence algorithms *IEEE Transactions on Cybernetics* **46** 2028
[7] Coates A, Huval B, Wang T, Wu D J and Ng A Y 2013 Deep learning with COTS HPC systems *Proceedings of the International Conference on Machine Learning (ICML)* (Atlanta, GA, USA) 1337
[8] Nickolls J, Buck I, Garland M and Skadron K 2008 Scalable parallel programming with CUDA *ACM Queue* **6** 40
[9] Foster I 1995 *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering* (Boston: Addison-Wesley)