SEBASTIAN IWASZENKO

Using GPU acceleration in solving selected kinetic coal gasification models

Coal gasification is recognized as one of clean coal technologies. Though it has been known for a relatively long time, its complexity still challenges scientists all over the world. One of the tools used in the research is simulation. The presented work investigates the capabilities of using GPGPU in modeling coal gasification. The selected set of models is used (volumetric, non-reactive core and Johnson's). The models as well as numeric solution methods were implemented as a sequential and parallel code. The execution time for both methods was investigated and the speedup for the parallel code determined. The influence of mathematical function call in the GPU code was also checked. The results show that for all models the parallel code gives significant speedup over the serial counterpart, as long as a reasonably large equation set is used. Therefore, using a dedicated GPU code for coal gasification simulations is highly recommended, whenever big systems of ODE have to be solved.

Keywords: *GPGPU, coal gasification modeling, parallel calculations*

1. INTRODUCTION

Coal gasification and underground coal gasification are examples of technologies allowing to use coal as an energy source or substrate for the chemical industry with relatively low environmental impact. Though the idea of the technology has been known since the 18th century, the process itself is complicated and much effort is required for its efficient and safe use. Computer simulation is one of research tools used for process investigation. However, the calculations are usually demanding and time consuming [16]. The possibilities given by parallel calculations are believed to help to overcome the mentioned problem.

In the presented work the capabilities of using GPGPU in modeling coal gasification are investigated. Three commonly used models were selected for calculations. They are known as volumetric, non-reactive core and Johnson's models [12]. Each model describes a change of char conversion in time in the form of ODE. They all belong to the group of kinetic models. The most interesting application of the parallel approach is to solve not one equation but a relatively large set of equations. It is common when a set of different parameters has to be used for calculation or when the simultaneous advance of the process for a few hundred of char particles is necessary. It is also the case when in a slab of char a division into a set of elements is applied. In all mentioned cases, the set of model equations has to be solved. The equations are independent of one another.

The rest of the article is organized as follows: in section 2 general information of GPGPU and its programming is provided, section 3 presents used models and numerical methods, section 4 describes equipment used and test cases, section 5 presents results and discussion. Finally, section 6 summarizes the research and forms conclusions.

2. USING GPU FOR GENERAL PURPOSE CALCULATIONS

Rapid development of semiconductor circuits resulted in incredible growth of their complexity. Moore's law, formed in the 1970s, predicted dou-

bling of the number of transistors on the integrated circuit every two years [13]. The processing power of the equipment followed the same pattern. However, the increase in integration and efficiency came with more and more effort. At the beginning of the 21st century it became obvious that further development at that speed would hardly be possible. The dramatic slowdown was observed after 2004, due to factors which have been summed up with the term "brick wall" [2]. The term wraps up all "walls" arising in front of further increment of the processing power and integration level. Increase in the processing power is nowadays more connected with parallel processing than boosting the speed of a particular component. Modern GPU units are an example of that strategy. They are composed of hundreds of processing units and they are particularly suitable for solving computationally intensive problems which can be represented in a parallel form.

One of most popular environments for using GPU as a general processing unit is offered by NVIDIA. The CUDA toolset allows the users to develop a parallel code using an extension of C/C++. The nvcc compiler is capable of converting the code into the form runnable by graphic streaming multiprocessors [3]. The CUDA architecture can be spotted as a kind of a SIMD machine. The program is processed by units called multiprocessors. Each multiprocessor is capable of running hundreds of threads set up in blocks. There are three levels of memory accessible for each thread. First, a local memory, private for each thread, can be used. Though it is extremely fast, it cannot be exchanged between the threads. On the other hand, the global memory can be accessed by all threads but is relatively slow. Somewhere in between there is a shared memory, which is common for a set of threads operating within the same block. It is slower than the local memory but still much faster than the global one. All kinds of memory have been located at the GPU board.

The procedure of setting up the calculations on GPU using CUDA consists of a few steps. First of all, CUDA distinguishes between codes and data dedicated to CPU and GPU. The first is called 'host' and does not require any other treatment than usually in C or C++ programs. The second is called 'device' and here support from CUDA extensions and libraries comes in help. To launch the code on the device a special function called 'kernel' has to be implemented. The function is marked with __global__ prefix. The function can be provided with parameters. It is important that all data submitted as parameters, especially pointers to the kernel, should to be located in any of GPU managed memories. Therefore, the usual call to the kernel function has the following general structure:

1. Allocation of memory on the device;
2. Copying data from the host to the device;
3. Launching the kernel and waiting for results;
4. Copying data (results) from the device to the host;
5. Releasing memory on the device.

In the kernel function itself, there is a bunch of predefined variables available. They can be used, for example, for identification of the thread currently processing the kernel. CUDA gives no warranty of order in which the threads processing kernels are started nor does it provide any default mechanisms for thread synchronization.

General processing on GPU and CUDA has been attracting more and more attention for the last few years. There are many fields of application where accelerated performance can be beneficial. One of them is definitely image processing. Most of used algorithms can be efficiently parallelized. There is some research reported which is focused on a new approach for filtering and improving the performance with the use of 'scatter-based' kernels instead of more naturally 'gather' ones [15]. Some scientists investigated the improvement of performance after the application of GPU to image processing, which shows that 10 to 20 times faster processing can be expected [4. Another efficiency results show that after the application of graphic processors to reconstruction algorithms for electron tomography, an increase in performance of 60 to 80 times has been observed [5]. GPU possibilities were also used for 3 body interaction computation (applied in molecular simulations) [18]. There are also many attempts to use GPU acceleration in calculations and high performance computing. Starting from research aimed at the development of dedicated high efficient methods for matrix multiplication [11], through algebraic solvers [10], to solving various problems of different types of flows [7] [8] [14] and finite methods applications [6]. Other applications include optimization [1] or database applications [17].

3. CHAR GASIFICATION MODELS

For calculation efficiency tests three of char gasification models were used. The models form a series from the simplest to the most complex. In the following sections each model is described.

The volumetric model is a representative of the homogenous models. It is assumed that the gasification goes in the whole volume of the char particle.

The process advances until the whole char is used. The mathematical representation of the model is described by the following equation [12]:

$$\frac{dx(t)}{dt} = k(1 - x(t)) \tag{1}$$

The k coeficient is a reaction rate constant. It is usually assumed that k is a function of temperature according to the Arrhenius law.

The non-reactive core model is based on an assumption that a reaction takes place only at the coal/char and gasous phase interface. A non-reactive core is formed from coal which does not participate in the reaction. The non reacting core becomes smaller as the reactions advance. The model equation can be written as follows [8]:

$$\frac{dx(t)}{dt} = k \cdot (1 - x(t))^{\frac{2}{3}} \tag{2}$$

The most complex equation used for tests is that of Johnson's model. The model extends the non-reactive core model by taking into consideration the resistance caused by a porous medium. The resistance influences the substrates and products transport to and from the particle surface. It is a result of the porous nature of coal and ashes. During the reaction, there is a significant change of the surface available for chemical processes. The Johnson's model equation is defined as follows:

$$\frac{dx(t)}{dt} = k \cdot (1 - x(t))^{\frac{2}{3}} \exp\left(-\alpha x(t)^2\right) \tag{3}$$

The k symbol stands for a reaction constant. It is assumed that in Johnson's model it represents the char type and its characteristics. The αx^2 factor describes the influence of the changing surface and transport resistance on the reaction rate. The following figures show typical shapes of char conversion-time dependence as predicted by each model [9].

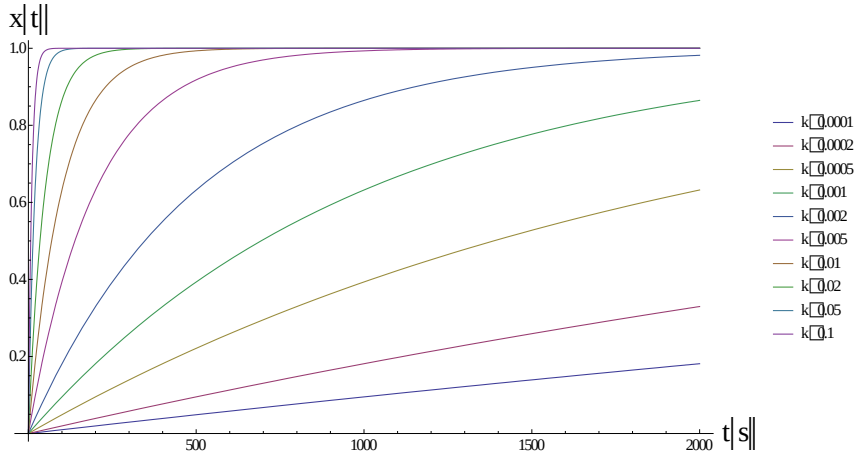

Fig. 1 Volumetric model predictions of char conversion in time

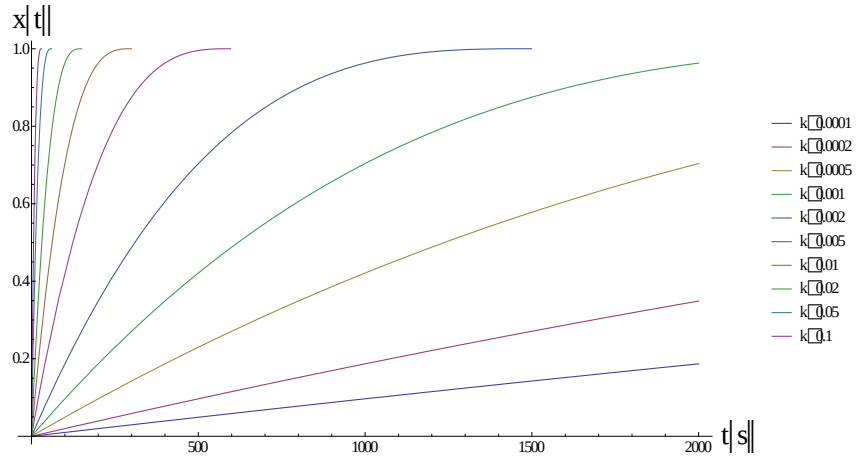

Fig. 2 Non-reactive core model predictions of char conversion in time

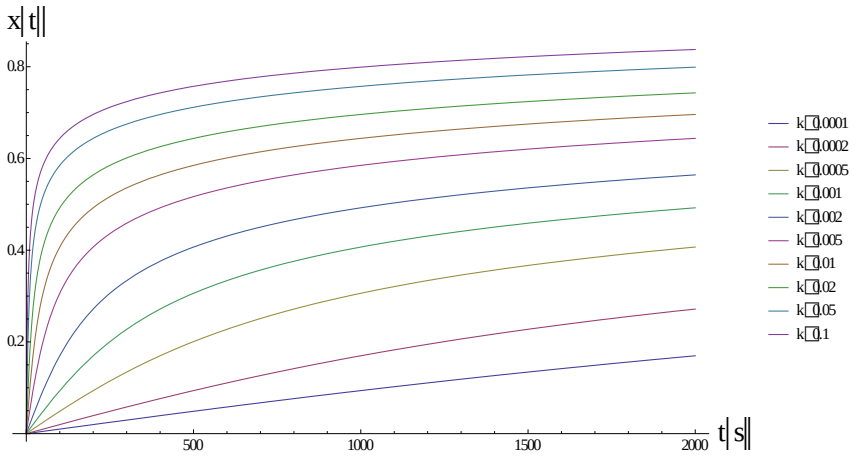

Fig. 3 Johnson's model predictions of char conversion in time

All presented equations were solved with the use of the Euler forward method:

$$x(t_{n+1}) = x(t_n) + hf\left(x(t_n), t_n\right) \tag{4}$$

The symbol f represents the right side of ODE, while h is the size of a time step. The Euler method is the simplest differential scheme for ODE solving. It is known of its limited accuracy as well as dependence on the chosen step size h value. However, it is quite sufficient for testing the effectiveness of GPU acceleration. It is worth to mention that the right side of the ODE equations is calculated only once during the computations.

4. TEST CASES

The testing code was implemented with the use of C++ and the Visual Studio integrated development environment. The code was designed as a console application. The procedure responsible for solving the model equations takes a pointer to the function representing the right side of ODE. During the implementation of serial and parallel codes, the focus was to keep both versions as similar as possible. CUDA Toolkit 7.5 was used as a platform for the CUDA code. All calculations were performed by means of single precision real numbers representation.

The calculations were set up in three series with a varying number of steps and equations in a set (equation set size). It was assumed that for each model the calculations were done with system sizes changing from 1 to 1,000 equations. The steps number was changing from 1 to 10,000. The calculations for each mix of system size and steps number were performed 30 times, and then the average time was recorded as a representative result. The results were collected in text csv files. Time measurements were performed using std::chrono::high_resolution_clock. All measurements were recalculated with the precision to milliseconds. The tests were performed on a workstation running Windows 8.1 Pro operating systems. Intel Core i5-3579K CPU running at 3.4 GHz was used for the serial code execution. The parallel CUDA code was executed with the use of an NVIDIA GeForce 660Ti graphic card, which is equipped with 1,344 CUDA cores. All tests were performed with a clean system which ran a testing console application and saved the results to a text file. The gathered data were then transferred to an Excel spreadsheet for further analysis.

5. RESULTS AND DISSCUSION

The selected gathered data are presented in Table 1, Table 2 and Table 3. It can be easily observed that using mathematical functions in calculations had great influence on calculation time. Taking as an example the case for 100 equations and 1,500 steps, the calculation times for the non-reactive core model and Johnson's model are, respectively, more or less two and three times longer than for the volumetric model. It is worth mentioning that the CUDA code takes even a greater amount of computation time for using mathematical functions. For the same case, 100 equations and 1,500 steps, the non-reactive core model takes five times as much time than the volumetric one. However, the difference between the non-reactive core model and Johnson's model is much

smaller, and equals about 10%. It is probable that loading the mathematical functions library to the CUDA device takes a significantly long time. Once loaded, it can be used more effectively for computations.

Another interesting pattern that can be observed is the calculation time dependence on the equation set size and number of steps. As one can expect, for a serial calculation either the increase in equations set size or the number of calculation steps leads to the increase of the calculation time. The dependence is nearly linear, provided that the number of equations and steps are reasonably large (Fig. 4 and Fig. 5). For smaller values, the influence of the system related tasks (e.g. context switching) is probably big enough to hinder the linearity. The situation with the parallel CUDA code is quite different. It shows almost no dependence on the equations set size. Despite 100 or 500 calculated equations, the calculation time does not differ more than 5%. In addition, the mentioned difference between the calculation times for the non-reactive core model and Johnson's model lies within the 10% boundary. The observed dependence allows to formulate a practical rule for using CUDA for calculations – whenever large sets of equations, calling mathematical functions are concerned, the parallel code is worth considering. It is less sensitive to the number of mathematical functions calls and changes in the equations set size. Having said that, it has to be mentioned that the CUDA calculation overhead can be unacceptable for a small system with low number calculation steps. Though the correctness of the remark can be deducted by analyzing the tables from Table 1 to Table 3, it is clearly depicted in Table 4 to Table 6, where the speedups of the CUDA code are presented (speedup is calculated as the CUDA code execution time to the CPU code execution time). The results are also presented in Fig. 6 and Fig. 7.

Table 1

Measured calculation times in [ms] for the volumetric model (cases when the parallel code is quicker than the serial one are marked in bold)

Equations	Steps (iterations)							
	Serial				CUDA			
	1000	2500	5000	10000	1000	2500	5000	10000
100	5.95	6.02	12.03	24.65	**1.75**	**4.01**	**7.51**	**15.01**
200	4.84	12.01	24.01	48.10	**1.51**	**3.77**	**7.53**	**15.05**
300	7.21	18.03	36.00	71.97	**1.51**	**3.77**	**7.54**	**15.08**
400	9.60	23.98	48.04	96.01	**1.51**	**3.78**	**7.55**	**15.09**
500	11.99	29.99	59.98	119.92	**1.52**	**3.78**	**7.56**	**15.12**
1000	24.15	60.35	120.47	241.45	**1.53**	**3.81**	**7.62**	**15.24**

Table 2

Measured calculation times in [ms] for the non-reactive core model (cases when the parallel code is quicker than the serial one are marked in bold)

Equations	Steps (iterations)							
	Serial				Parallel CUDA			
	1000	2500	5000	10000	1000	2500	5000	10000
100	8.57	14.62	29.14	58.14	11.03	24.41	48.82	97.64
200	11.72	29.35	58.13	116.11	**9.77**	**24.42**	**48.84**	**97.68**
300	17.53	43.62	87.15	174.07	**10.25**	**25.63**	**51.27**	**102.55**
400	23.35	58.14	116.09	232.28	**10.50**	**26.26**	**52.51**	**105.02**
500	29.18	72.91	145.09	289.92	**10.60**	**26.50**	**52.98**	**105.96**
1000	58.18	145.06	289.94	579.61	**10.75**	**26.86**	**53.71**	**107.39**

Table 3

Measured calculation times in [ms] for Johnson's model (cases when the parallel code is quicker than the serial one are marked in bold)

Equations	Steps (iterations)							
	Serial				Parallel CUDA			
	1000	2500	5000	10000	1000	2500	5000	10000
100	12.24	22.23	44.28	88.47	12.29	27.69	55.38	110.77
200	17.85	44.33	88.48	176.84	**11.08**	**27.70**	**55.41**	**110.82**
300	26.80	66.45	132.65	265.10	**11.58**	**28.95**	**57.90**	**115.80**
400	35.55	88.52	176.85	353.35	**11.83**	**29.56**	**59.12**	**118.25**
500	44.42	110.62	220.90	441.67	**11.91**	**29.78**	**59.55**	**119.10**
1000	88.64	221.07	441.78	883.75	**12.06**	**30.15**	**60.28**	**120.54**

Table 4

Parallel code speedup for the volumetric model (cases when the parallel code is quicker than the serial one are marked in bold)

Equations	Steps (iterations)								
	1000	2000	3000	4000	5000	6000	7000	8000	9000
100	**3.40**	**1.38**	**1.57**	**1.61**	**1.60**	**1.61**	**1.61**	**1.61**	**1.61**
200	**3.21**	**3.19**	**3.20**	**3.20**	**3.19**	**3.19**	**3.19**	**3.23**	**3.19**
300	**4.77**	**4.78**	**4.78**	**4.78**	**4.77**	**4.78**	**4.77**	**4.77**	**4.78**
400	**6.35**	**6.35**	**6.36**	**6.36**	**6.36**	**6.38**	**6.37**	**6.36**	**6.36**
500	**7.91**	**7.93**	**7.93**	**7.93**	**7.93**	**7.93**	**7.93**	**7.94**	**7.93**
1000	**15.81**	**15.83**	**15.82**	**15.80**	**15.80**	**15.83**	**15.85**	**15.86**	**15.97**

Table 5

Parallel code speedup for the non-reactive core model (cases when the parallel code is quicker than the serial one are marked in bold)

Equations	Steps (iterations)								
	1000	2000	3000	4000	5000	6000	7000	8000	9000
100	0.78	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60
200	**1.20**	**1.19**	**1.19**	**1.19**	**1.19**	**1.19**	**1.19**	**1.19**	**1.19**
300	**1.71**	**1.70**	**1.70**	**1.70**	**1.70**	**1.70**	**1.70**	**1.70**	**1.70**
400	**2.22**	**2.21**	**2.21**	**2.21**	**2.21**	**2.21**	**2.21**	**2.21**	**2.21**
500	**2.75**	**2.75**	**2.75**	**2.74**	**2.74**	**2.74**	**2.74**	**2.74**	**2.74**
1000	**5.41**	**5.40**	**5.40**	**5.40**	**5.40**	**5.40**	**5.40**	**5.40**	**5.40**

Table 6

Parallel code speedup for Johnson's model (cases when the parallel code is quicker than the serial one are marked in bold)

Equations	Steps (iterations)								
	1000	2000	3000	4000	5000	6000	7000	8000	9000
100	1.00	0.82	0.80	0.80	0.80	0.80	0.80	0.80	0.80
200	**1.61**	**1.60**	**1.60**	**1.60**	**1.60**	**1.60**	**1.60**	**1.60**	**1.60**
300	**2.31**	**2.30**	**2.29**	**2.29**	**2.29**	**2.29**	**2.29**	**2.29**	**2.29**
400	**3.01**	**3.00**	**2.99**	**2.99**	**2.99**	**2.99**	**2.99**	**2.99**	**2.99**
500	**3.73**	**3.72**	**3.71**	**3.71**	**3.71**	**3.71**	**3.71**	**3.71**	**3.71**
1000	**7.35**	**7.34**	**7.33**	**7.33**	**7.33**	**7.33**	**7.33**	**7.33**	**7.33**

It can be observed that for the volumetric model the CUDA code has an advantage over the serial one when the system size exceeds 30 equations and so does the number of iterations. For smaller calculation tasks it is faster to run it on CPU as a serial code. The maximum speedup recorded for the volumetric model was 15.97 for 1,000 equations with 9,000 iterations.

The speedup for the non-reactive core model is lower than for the volumetric model. It seems justifiable to say that 200 equations are a limit up to which the CUDA code has a reasonable advantage over the serial one. The maximum speedup recorded for the non-reactive core model was 5.41 for 1,000 equations with 1,000 iterations.

The Johnson's model equations behave similarly to those of the non-reactive core model, still the former offered better speedup when calculated on the CUDA device. The maximum speedup recorded for Johnson's model was 7.35 for 1,000 equations with 1,000 iterations.

It is worth mentioning that speedup values became constant when the number of steps was relatively large. The dependence is true for all tested models (Fig. 7).

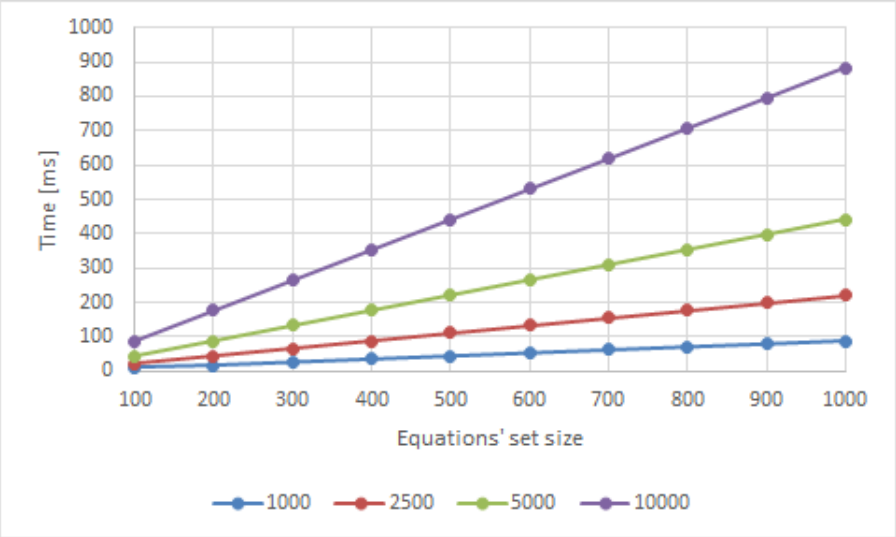

Fig. 4 CUDA code speedup as a function of equations set size

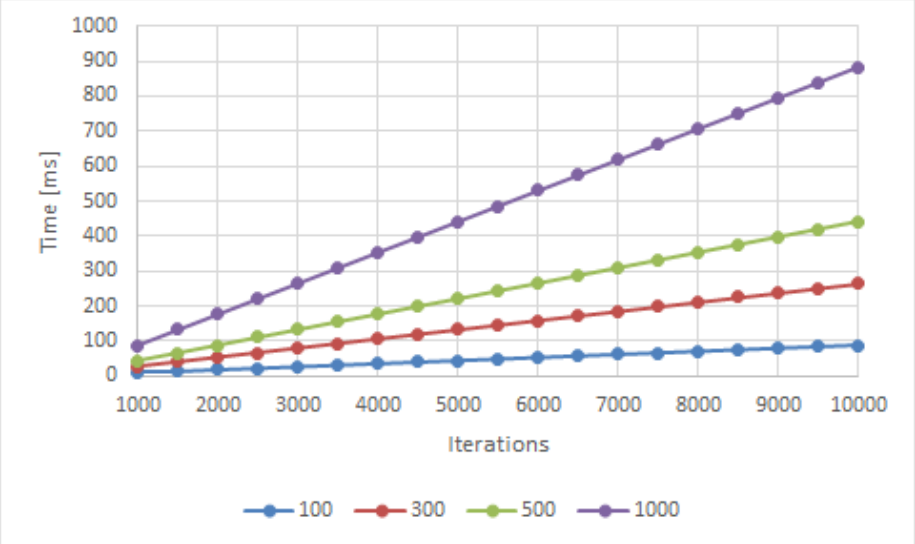

Fig. 5 CUDA code speedup as a function of steps number

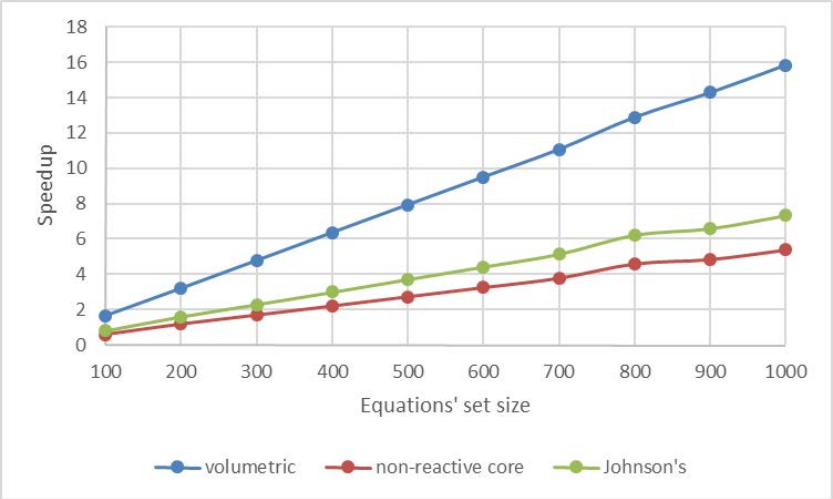

Fig. 6 CUDA code speedup as a function of equations set size

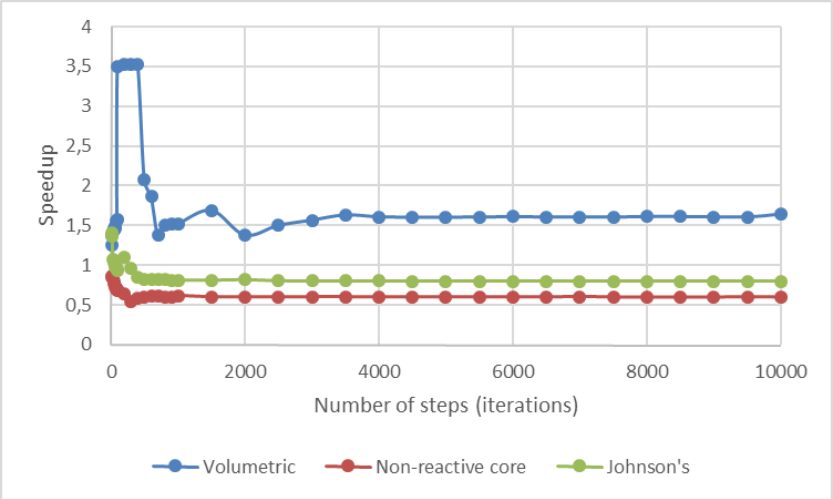

Fig. 7 CUDA code speedup as a function of steps (iterations) number

6. CONCLUSIONS

In the paper the application of serial and parallel implementation for selected char gasification models was presented. The volumetric, non-reactive core and Johnson's models were selected for test cases. The models and numerical solver were implemented as serial and parallel codes. The serial code was run on a CPU unit with the use of a single thread while GPU was used for the parallel code. The results show that the parallel code runs significantly faster than the serial one, as long as there are no mathematical function calls within the code or there is a reasonable large set of equations solved. It was observed that a speedup value became constant for a given number of equations, provided that the number of iterations number was sufficiently large. It was also noticed that the parallel code was less sensitive to mathematical function calls than to functions representing the right side of ODE models.

Upon that the following conclusions were formulated:

- For a small number of equations it is usually better to use the CPU serial code than the parallel GPU versions. There are a certain number of equations and iterations below which the serial code is more advantageous than the parallel one;
- The parallel CUDA code performs significantly better for simple equations. The performance of CUDA code drops more than serial one when a library mathematics function is called. However, additional calls slows CUDA code much less than serial one;
- There is a limiting speedup value possible for each model and equations set size.

Acknowledgements

The works presented in the paper have been supported by the statutory activity of the Central Mining Institute: Research on hardware system architecture influence on calculations efficiency for coal gasification modeling – No. GIG: 11420255-350.

Bibliography

1. Arca B, Ghisu T, Trunfio GA.: GPU-accelerated multi-objective optimization of fuel treatments for mitigating wildfire hazard. Journal of Computational Science 11, 2015 pp. 258-68.
2. Asanovic K., Bodik R., Catanzaro B.C., Gebis J.J., Husbands P., Keutzer K., Patterson D. A., Plishker W. L., Shalf, J., Williams S. W., Yelick K. A.: The landscape of parallel computing research: A view from Berkeley, Tech. Rep. UCB EECS-2006-183. Electrical Engineering and Computer Sciences. University of California Berkeley 2006.
3. Brodtkorb AR, Hagen TR, Saetra ML.: Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing 73(1), 2013 pp.4-13.
4. Castaño-Díez D, Moser D, Schoenegger A, Pruggnaller S, Frangakis A.S.: Performance evaluation of image processing algorithms on the GPU. Journal of structural biology 164(1), 2008 pp.153-60.
5. Díez DC, Mueller H, Frangakis AS.: Implementation and performance evaluation of reconstruction algorithms on graphics processors. Journal of Structural Biology 157(1), 2007 pp.288-95.
6. Fialko S.: Parallel direct solver for solving systems of linear equations resulting from finite element method on multi-core desktops and workstations. Computers & Mathematics with Applications 70(12), 2015 pp.2968-87.
7. Fu L, Gao Z, Xu K, Xu F.: A multi-block viscous flow solver based on GPU parallel methodology. Computers & Fluids 95, 2014 pp.19-39.
8. He X, Wang Z, Liu T.: Solving Two-Dimensional Euler Equations on GPU. Procedia Engineering 61, 2013 pp.57-62.
9. Iwaszenko S.: Using Mathematica software for coal gasification simulations–Selected kinetic model application. Journal of Sustainable Mining 14(1), 2015 pp.9-21.
10. Liu H, Yang B, Chen Z.: Accelerating algebraic multigrid solvers on NVIDIA GPUs. Computers & Mathematics with Applications 70(5), 2015 pp.1162-1181.
11. Matsumoto K, Nakasato N, Sakai T, Yahagi H, Sedukhin SG.: Multi-level optimization of matrix multiplication for GPU-equipped systems. Procedia Computer Science 4, 2011 pp.342-351.
12. Molina, A., Mondragón, F.: Reactivity of coal gasification with steam and CO_2. Fuel 77(15), 1998 pp.1831–1839. doi:10.1016/S0016-2361(98)00123-9.
13. Moore G.E.: Progress in Integrated Electronics. Technical Digest 1975. International Electron Devices Meeting. IEEE, 1975 pp. 11-13.
14. Oyarzun G, Borrell R, Gorobets A, Lehmkuhl O, Oliva A.: Direct numerical simulation of incompressible flows on unstructured meshes using hybrid CPU/GPU supercomputers. Procedia Engineering 61, 2013 pp.87-93.
15. da Silva J, Ansorge R, Jena R.: Efficient scatter-based kernel superposition on GPU. Journal of Parallel and Distributed Computing 84, 2015 pp.15-23.
16. Wachowicz, J., Janoszek, T., Iwaszenko, S.: Model tests of the coal gasification process. Archives of Mining Sciences 55, 2010 pp.249–262.
17. Walkowiak S, Wawruch K, Nowotka M, Ligowski L, Rudnicki W.: Exploring utilisation of GPU for database applications. Procedia Computer Science 1(1), 2010 pp.505-513.
18. Yaseen A, Ji H, Li Y.: A load-balancing workload distribution scheme for three-body interaction computation on Graphics Processing Units (GPU). Journal of Parallel and Distributed Computing 87, 2016 pp.91-101.

SEBASTIAN IWASZENKO, PhD Eng.
Central Mining Institute GIG
e-mail: siwaszenko@gig.eu