Heat Transfer Simulation using GPUs

Sándor Szénási
John von Neumann
Faculty of Informatics
Óbuda University
Budapest, Hungary
Email: szenasi.sandor@nik.uni-obuda.hu

Imre Felde
John von Neumann
Faculty of Informatics
Óbuda University
Budapest, Hungary
Email: felde.imre@nik.uni-obuda.hu

***Abstract*—Several real-world applications involve simulation of the heat transfer within a given workpiece when placed into an environment with a different temperature. This requires calculation of the temperature in a particular location and at a given moment in time according to the available input parameters (shape and thermal characteristics of the given object and the environment). There are several computer-assisted numerical methods available for solving this type of problem, but these usually have a high computational demand. This paper presents a way to re-design an already known method as a data-parallel one, which makes it possible to use graphics accelerators to speed up the simulation process. According to the test results, the CUDA implementation of the parallel algorithm offers the same accuracy, but 4-5x lower runtime, as the original sequential method.**

I. Introduction

Heat transfer is the exchange of thermal energy between physical systems. There are more than one fundamental mode of heat transfer [1], [2], but this paper deals only with transient conduction when the temperature within the examined object changes as a function of time. In this case, the analysis of heat transfer is quite complex and often calls for the application of approximation methods and computer-assisted numerical analysis.

There are several applications in which it is necessary to simulate the heat transfer within a given workpiece when placed into some cooling liquid. In these cases, the input parameters of the algorithm are the shape and the thermal characteristics of the object and the parameters of the surrounding environment. The required results are the temperature values of the object in a given spatial location and at a given time. The most commonly-used methods for solving such an algorithm are based on a discretized numerical analysis. Computer programs can speed up the usage of such an analysis , but in cases involving a large number of experiments, the required runtime can be unacceptably high.

This paper presents a redesigned algorithm for this purpose, which has been adapted for data-parallel systems, especially for graphics accelerators. The rest of this article is structured as follows: Section 2 contains the mathematical background of the heat transfer model used; Section 3 presents the idea and implementation of the parallelization of the necessary calculations; Section 4 contains the results of our experiments (in both speed and accuracy); and finally, the last section contains the conclusions and plans for further development.

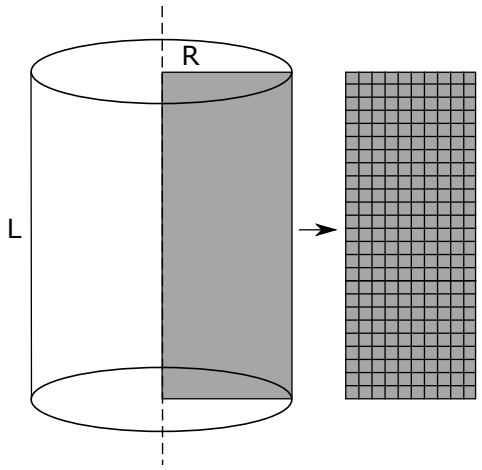

Fig. 1: Cross-section of the work item and the used finite grid.

II. The Heat Transfer Model

In real-world applications [3], we often need to simulate the heat transfer inside a given workpiece. There are several methods available for doing so, and improvements and adaptations of the already existing algorithms are still a very active area of research. It is quite complicated to calculate heat transfer in the case of irregularly shaped objects; therefore, this paper deals only with regular cylindrical objects. This restriction greatly simplifies the model, because it is enough to simulate one of the middle cross-sections passing through the centre line of the workpiece, reducing the three-dimensional problem to a two-dimensional one. This two-dimensional axis-symmetrical heat conduction model [4] is considered to estimate the distribution of temperature (Fig. 1).

The input parameters of the simulation are:

- R = the radius of the cylinder;
- L = the length of the cylinder;
- $HTC(t, z)$ = the heat transfer coefficient. The actual value depends on the local coordinate (z) and the time (t);

978-1-5090-1216-9/16/$31.00 ©2016 IEEE

- $k(T)$ = thermal conductivity (varying with the temperature);
- $C_p(T)$ = heat capacity (varying with the temperature).

Based on the above, we can describe the transient heat conduction problem with the following nonlinear mathematical formulation (Eq. 1):

$$\frac{\partial}{\partial r}(k\frac{\partial T}{\partial r}) + \frac{k}{r}\frac{\partial T}{\partial r} + \frac{\partial}{\partial z}(k\frac{\partial T}{\partial z}) + q_v = \rho C_p \frac{\partial T}{\partial t} \quad (1)$$

and the given initial and boundary conditions (Eq. 2 and 3):

$$T(r, z, 0) = T_0 \quad (2)$$

and

$$k\frac{\partial T}{\partial z}|_{0 \le z \le Z, r=R} = HTC(z,t)[T_q - T(r,z,t)] \quad (3)$$

where r and z = local coordinates of the examined point; t = time; ρ = density of the object; T_0 = initial temperature of the workpiece; and T_q = temperature of the cooling liquid. We have used the weighted Schmidt explicit finite difference method to discretize the equations mentioned above. Using this method we can directly solve the problem.

To calculate the temperature values in given locations and at given times, we use the finite difference method [5]. To use this method to approximate the solution to a given problem, one must first discretize the problem domain. This is usually done by dividing the domain into a uniform grid. We substitute the cross section of the object with an $N \times M$ sized matrix, where N is the number of the finite items radially, and M is the number of finite items longitudinally. N and M are constant input parameters; larger values lead to more accurate results but also increase the runtime.

Using the finite difference method, we can calculate the heat transfer between the given areas for one time step. To ensure accuracy it is also important to select a sufficiently small time interval (dt) for this step. The simulation needs to run a cycle of the calculations mentioned above to generate the heat transfer values for each time step. The above-mentioned method is well-suited for several situations; however, the calculation demand is quite high. We have to calculate the given formulas for all finite items for all given time steps. In a typical application, N could be 10, M could be 40 and dt should be 0.02 secs. If we needed to simulate 120 seconds, we have to calculate the given equations for $10 \times 40 \times (120/0.02) = 2.4M$ times.

In normal circumstances, this is not an issue, because an average CPU can calculate the final results in less than one second. But there are several applications, in which a lot of heat transfer simulations are necessary to solve the problem. The various inverse problems are a typical example in this class. For example, in the case of the inverse heat conduction problem (IHCP) [6], [7], the temperature values of some given points of the workpiece are known, but the HTC function is not. To find the HTC values, the only presently available methods are based on heuristic searches. In such cases, we systematically generate HTC functions and run the above-mentioned simulation using these functions as the input. During the simulation, we then compare the measured temperature values to the created ones. The goal is to minimize the difference between measured and generated values, but this usually requires millions of simulations, and consequently the runtime of the heat transfer calculation becomes a very limiting factor.

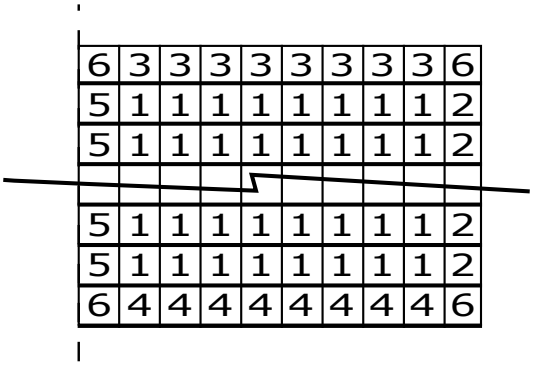

Fig. 2: Representation of item classes.

III. Parallelization of the algorithm

Fig. 2 depicts a graphical representation of the half cross-section of the cylinder divided into $N \times M$ cells. Six separate item classes can be distinguished:

1) Inner items, surrounded by other items in all four directions (number of elements: $(N-2) \times (M-2)$);
2) Items at the side surface of the cylinder (number of elements: $M-2$);
3) Items at the top surface of the cylinder (number of elements: $N-2$);
4) Items at the bottom surface of the cylinder (number of elements: $N-2$);
5) Items at the centre line of the cylinder (number of elements: $M-2$); and
6) Items at the corners (top of centre line, the bottom of centre line, the outermost point of the top surface, outermost point of the bottom surface) (number of elements: 1+1+1+1).

Calculations are more or less different from each other in the different classes, but it is clearly visible that there is a high potential for parallelization. Within each group, the parallelization can be done in a pure data-parallel fashion, because the algorithm is the same for all elements, only the arguments used are different.

It is worth implementing a GPU kernel to solve the equations mentioned above [8]. It is important to run these threads in one block because the algorithm requires an explicit synchronization. Without providing a detailed description of the discretized equations, it is evident that the output of each element (the temperature of the given point at a given time) is based on the values of its surrounding elements (the temperature of the neighbouring elements at the previous moment in time). Based on this, threads are not allowed to write back their results to the global memory until all

neighbouring threads read their input values in the same time phase.

Practical experiences have shown that the ideal number of finite elements $N \times M$ is about 400 (10×40). This configuration leads to an acceptable accuracy and a tolerable runtime. This configuration is also ideal for GPU implementation, because according to the CUDA specifications [9], from Compute Capability 1.0 the maximum number of threads per block is 512. Therefore, it is possible to assign one thread to one finite element of the grid, and we can run 400 threads in parallel to solve the equations at the same time. In the case of Compute Capability 2.x or more, the maximum number of threads is 1024. This provides a significant amount of reserves; thus, it is possible to increase the resolution of the grid so as to enchance the accuracy of the heat transfer simulation, if necessary.

IV. Evaluation

Execution of 400 threads in one block does not utilize the full processing power of an average graphics card at all [10]. As our preliminary tests showed, if running one full simulation (where the size of the grid is 400 and the number of time intervals is 16,000), it is not worthwhile to use a GPU, because CPU implementation is faster. But there are several applications in which a lot of heat transfer simulations with different input parameters are needed (typically in the case of heuristic search-based solutions for some inverse problem [11], [12]). These are entirely independent processes; therefore, we can run them in a parallel fashion. Using this method, we can run $N \times M \times K$ parallel threads in K thread blocks (where K is the number of simulations), and this can significantly increase the occupancy of the multiprocessors.

We used the following hardware configurations for the tests:

- CPU configuration
  - Processor: Intel® Core™ i7-2600
  - Architecture: Sandy Bridge
  - Number of cores: 4
  - Memory: 16GB DDR2
- GPU configuration
  - Graphics accelerator: NVIDIA Tesla K40c
  - Architecture: Kepler (GK110B)
  - Number of shaders: 2880
  - SMX Count: 15
  - Memory: 12GB GDDR5

To decrease uncertainty, we ran 52 tests for all parameter sets and calculated the average of the last 50 results.

Table I and Figure 3–4 show the results of the first runtime test, in which we change the value of K and run more and more parallel heat transfer simulations at the same time. As is evident, in the case of low K values, the runtime of the CPU is lower. In the case of higher K values, the two implementations become identical (at about $K = 15$), and after that, the GPU becomes faster.

Table II and Fig. 5 show the results of the second runtime test. In this case, we changed the resolution of the grid. As is also visible, in the case of low $N \times M$ values, the advantage

TABLE I: Detailed runtime values for CPU and GPU runtime tests based on the number of parallel simulations.

Parallel simulations	Runtime (ms) GPU	CPU	Runtime/p.s. (ms) GPU	CPU	Speedup
1	77.10	18.17	77.10	18.17	0.24
2	74.05	20.46	37.03	10.23	0.28
3	74.33	23.45	24.78	7.82	0.32
4	74.77	30.58	18.69	7.65	0.41
5	75.04	33.10	15.01	6.62	0.44
6	75.27	33.79	12.55	5.63	0.45
7	75.22	40.56	10.75	5.79	0.54
8	75.93	52.81	9.49	6.60	0.70
9	74.98	57.72	8.33	6.41	0.77
10	75.48	64.00	7.55	6.40	0.85
11	75.32	67.70	6.85	6.15	0.90
12	75.47	66.41	6.29	5.53	0.88
13	75.56	67.46	5.81	5.19	0.89
14	75.89	74.57	5.42	5.33	0.98
15	75.98	86.63	5.07	5.78	1.14
16	80.76	100.44	5.05	6.28	1.24
17	80.81	102.95	4.75	6.06	1.27
18	81.34	111.02	4.52	6.17	1.36
19	81.62	111.95	4.30	5.89	1.37
20	81.20	115.50	4.06	5.77	1.42
30	88.75	167.94	2.96	5.60	1.89
40	92.27	226.38	2.31	5.66	2.45
50	104.00	279.91	2.08	5.60	2.69
100	208.62	544.46	2.09	5.44	2.61
200	470.12	1087.11	2.35	5.44	2.31
300	567.25	1636.27	1.89	5.45	2.88
400	874.30	2142.26	2.19	5.36	2.45
500	989.43	2671.10	1.98	5.34	2.70
600	1289.54	3236.25	2.15	5.39	2.51
700	1521.90	3770.28	2.17	5.39	2.48
800	1771.62	4271.60	2.21	5.34	2.41
900	1940.57	4762.51	2.16	5.29	2.45
1000	2167.54	5362.34	2.17	5.36	2.47
1500	3252.75	7994.97	2.17	5.33	2.46
2000	3856.91	10646.80	1.93	5.32	2.76
2500	4822.39	13320.90	1.93	5.33	2.76

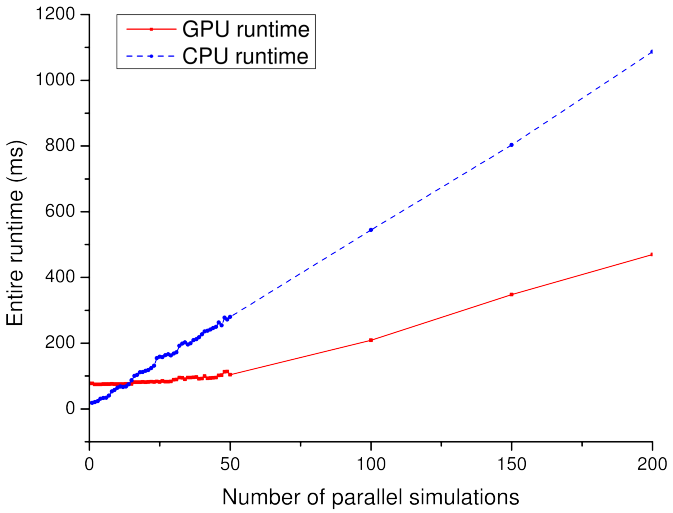

Fig. 3: CPU and GPU runtime values according to the number of parallel simulations.

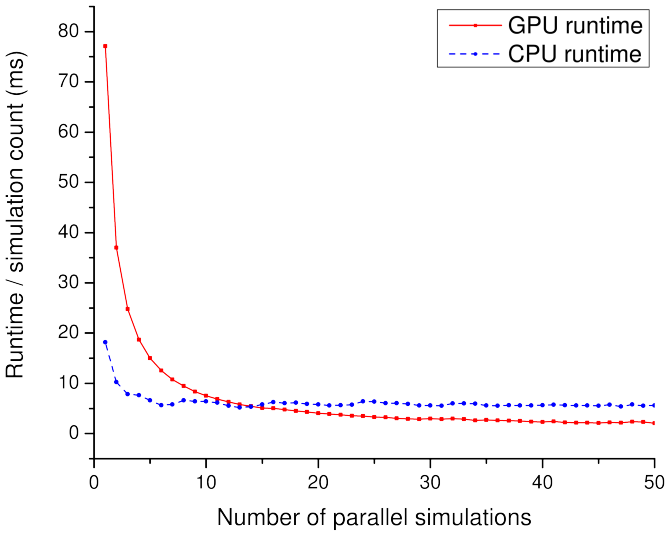

Fig. 4: CPU and GPU runtime values of one simulation according to the number of parallel simulations.

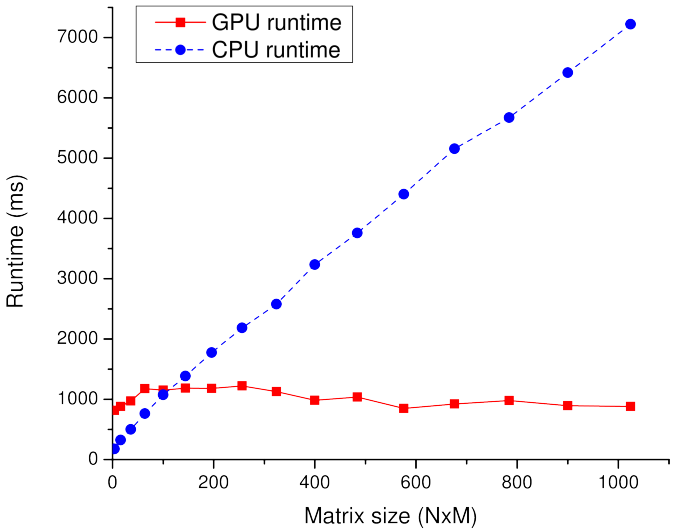

Fig. 5: CPU and GPU runtime values of one simulation according to the size of the finite matrix.

TABLE II: Detailed runtime values for CPU and GPU runtime tests based on the size of the finite matrix.

N	M	NxM	GPU runtime (ms)	CPU runtime (ms)	Speedup
1	4	4	814.89	177.78	0.22
2	8	16	879.55	324.71	0.37
3	12	36	970.07	500.40	0.52
4	16	64	1176.16	760.75	0.65
5	20	100	1153.12	1072.73	0.93
6	24	144	1185.27	1386.11	1.17
7	28	196	1181.77	1775.00	1.50
8	32	256	1221.46	2185.83	1.79
9	36	324	1128.36	2575.96	2.28
10	40	400	984.28	3234.19	3.29
11	44	484	1035.43	3759.17	3.63
12	48	576	849.96	4399.99	5.18
13	52	676	920.89	5156.49	5.60
14	56	784	980.70	5671.42	5.78
15	60	900	894.79	6419.29	7.17
16	64	1024	882.85	7221.50	8.18

of using the GPU is smaller. Increasing the number of finite elements leads to better GPU performance compared to the CPU.

We ran several further tests of accuracy. Comparing the results of the CPU and GPU simulations shows that the results achieved were always identical. Because of this, while this paper does not present the detailed results, we can simply conclude that the accuracy of the two methods is equal.

V. Conclusions

The main purpose of this research was to develop a novel heat transfer simulation algorithm that is capable of being executed in a data-parallel environment, in particular for graphics accelerators. In the current phase, the algorithm and its CUDA implementation have been completed. As demonstrated by the test results, the accuracy of the new method is the same as that of the original sequential one; therefore, it can serve as a replacement for CPU implementation.

As shown by further testing, the execution time for the parallel method is only about 20-25% of the time required by the original sequential implementation; however, this value depends on the resolution of the discretization and the number of executed simulations.

As a further development, it may be worthwhile to store the grid containing the temporary temperature values in the shared memory of the multiprocessor. Preliminary calculations show that this is possible, as the typical shared memory size of current NVIDIA architectures (16KB – 112KB according to the Compute Capability [9] specifications) is large enough to store all necessary data for the independent blocks.

Acknowledgements

We acknowledge the financial support of this work by the Mexican-Hungarian S&T bilateral project TÉT MX-1-2013-0001 and "Cooperative research on control of heat treatment stress and distortion for high value added machinery products" (2014DFG72020) projects. The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching Center program. The authors also wish to thank the members of the CUDA Teaching Center at Óbuda University for their constructive comments and suggestions. The content of the paper however does not necessarily express the views of these companies and persons, the authors take full responsibility for any errors or omissions.

References

[1] M. N. Ozisik, *Heat Conduction*. Wiley-Interscience, 1993.

[2] P. Oksman, S. Yu, H. Kytönen, and S. Louhenkilpi, "The Effective Thermal Conductivity Method in Continuous Casting of Steel," *Acta Polytechnica Hungarica*, vol. 11, no. 9, 2014.

[3] A. Bogárdi-Mészöly, A. Rövid, H. Ishikawa, S. Yokoyama, and Z. Vámossy, "Tag and Topic Recommendation Systems," *Acta Polytechnica Hungarica*, vol. 10, no. 6, pp. 171–191, 2013.
[4] K. W. Kim and S. W. Baek, "Inverse Surface Radiation Analysis in an Axisymmetric Cylindrical Enclosure Using a Hybrid Genetic Algorithm," *Numerical Heat Transfer, Part A: Applications*, vol. 46, no. 4, pp. 367–381, aug 2004.
[5] J. Crank, P. Nicolson, and D. R. Hartree, "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 43, no. 01, p. 50, oct 2008.
[6] M. N. Özisik and H. R. B. Orlande, *Inverse Heat Transfer: Fundamentals and Applications*. Taylor & Francis, 2000.
[7] O. M. Alifanov, *Inverse Heat Transfer Problems*. Springer, 1994.
[8] S. Szénási, I. Felde, and I. Kovács, "Solving One-dimensional IHCP with Particle Swarm Optimization using Graphics Accelerators," in *10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics*. Timisoara, Romania: IEEE, 2015, pp. 365–369.
[9] NVIDIA, "CUDA C Programming Guide," 2014.
[10] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, and W.-m. W. Hwu, "Program optimization space pruning for a multithreaded gpu," in *Proceedings of the sixth annual IEEE/ACM international symposium on Code generation and optimization - CGO '08*. New York, New York, USA: ACM Press, apr 2008, p. 195.
[11] S. Sergyán and L. Csink, "Automatic Parametrization of Region Finding Algorithms in Gray Images," in *4th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2007, pp. 199–202.
[12] S. Das, A. Abraham, and A. Konar, "Particle swarm optimization and differential evolution algorithms: Technical analysis, applications and hybridization perspectives," *Studies in Computational Intelligence*, vol. 116, no. 2008, pp. 1–38, 2008.