

Using CUDA Architecture for the Computer Simulation of the Casting Solidification Process

Grzegorz Michalski, and Norbert Sczygiol

Abstract—This paper presents a simulation of the casting solidification process performed on graphics processors compatible with nVidia CUDA architecture. The new approach shown in this paper allows the process of matrix building to be divided into two independent phases. The first is independent from the nodal temperature values computed in successive time-steps. The second is performed on the basis of nodal temperature values, but does not require a description of the finite element mesh. This phase is performed in each time step of the simulation of the casting solidification process. The separation of these two phases permits an effective implementation of the simulation software of the casting solidification process on the nVidia CUDA architecture or any other multi-/manycore architecture. In this paper the authors present the results of a computer simulation conducted on a GPU and CPU. The results of the computer simulations presented in this article were obtained with the use of authorial software (CPU-based and GPU-based).

Index Terms—casting, computations on many and multi-core processors, computer simulation, parallel processing, solidification processing.

I. INTRODUCTION

IN recent years the dynamic development of multicore processors has had a direct impact on the availability of advanced high performance solutions for engineers. A few years ago, computers with a high-end graphics card were intended primarily for computer game players and computer graphics designers. Nowadays, the situation has dramatically changed. Graphic processors (GPUs – Graphics Processing Unit) are increasingly being used in high performance computations. A single GPU has a theoretical computational power several times higher than the fastest general purpose processors available today. Fig. 1 shows the increase in the theoretical computational power of graphic processors and general purpose processors in recent years. Computations using a GPU working as a computing accelerator do not require any additional equipment, such as specialized workstations, and can be made on an ordinary personal computer equipped with a graphic card that supports CUDA or OpenCL. As graphic processors, in accordance with their original purpose, have been designed to efficiently perform mathematical operations on two-dimensional matrices, they should be capable of performing numerical simulations. The application of modern multi- and many-core architectures, such as graphic processors, for computational purposes allows huge systems of equations, which may consist of many

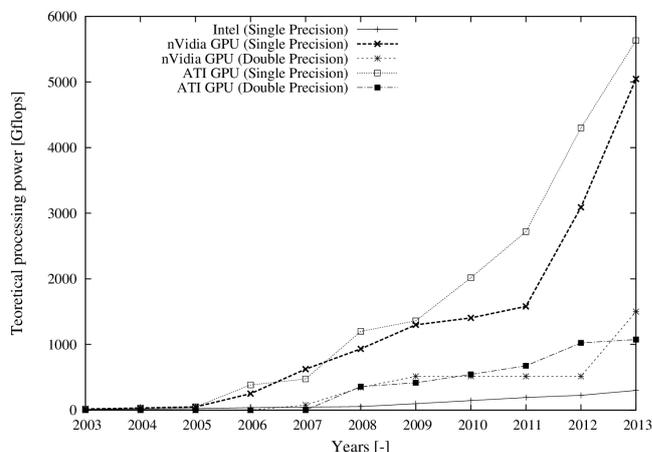


Fig. 1: An increase in theoretical computational power of the GPU and CPU in the recent years

millions of variables, to be solved very fast. The algorithms used for solving such systems of equations are based primarily on standard arithmetic operations on matrices and vectors. Such algorithms permit efficient parallelization and implementation for many-cores architecture like graphic processors. [1].

In this article, the authors present a computer simulation of the casting solidification process (enthalpy formulation), one that belongs to a group of unsteady processes. For processes of this type, a system of equations is built from scratch in each time step. The matrix of the coefficients of this system of equations and a vector of right sides are built on the basis of several factors: a description of the finite elements mesh, the boundary conditions, the material properties and the results obtained from previous time steps. These operations require a large amount of data in each time step of the simulation to be processed. Repeatedly sending huge data sets to the memory of the graphic card may create a real problem. The specific character of this data (mainly the finite element mesh) makes it difficult to efficiently parallelize the matrix building process. This difficulty derives from the lack of regularity of the data that are to be processed. This problem can be clearly seen in the case of GPUs and SIMT architecture (Single Instruction Multiple Thread), where the conditional execution of certain parts of the program code has a strong negative impact on the efficiency of computations. An additional problem is the need to synchronize the write operations performed during the construction of the system of equations [2], [3], [4].

Manuscript received December 20, 2013; revised January 08, 2014.

Grzegorz Michalski is with the Czestochowa University of Technology, Dabrowskiego 69, PL-42201, Czestochowa, Poland. (corresponding author: Grzegorz Michalski; phone: (+48 34) 3250589; fax: (+48 34) 3250589; email: grzegorz.michalski@icis.pcz.pl).

Norbert Sczygiol is with the Czestochowa University of Technology, Dabrowskiego 69, PL-42201, Czestochowa, Poland. (email: norbert.szczygiol@icis.pcz.pl)

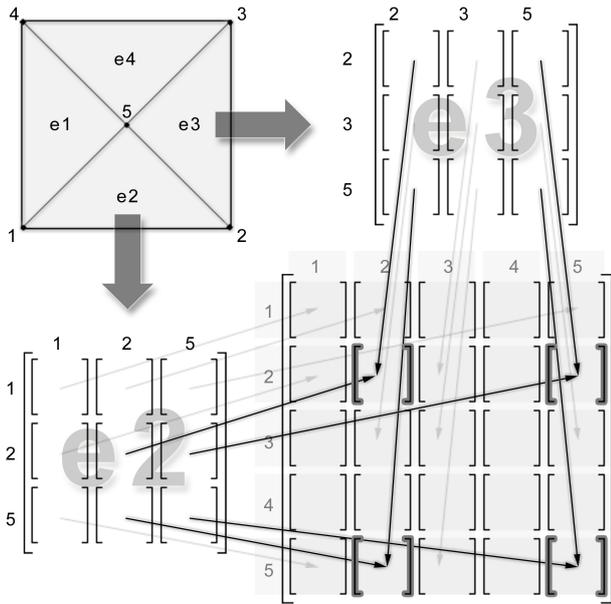


Fig. 2: Process of building the global coefficient matrix

II. BUILDING THE SYSTEM OF EQUATIONS ON GRAPHIC PROCESSORS

While performing simulations of unsteady processes the system of equations has to be built many times (for each time step). Owing to the specific nature of the graphic processors architecture, this can constitute a real problem and can significantly decrease the efficiency of computations. This relates to the slow data transfer from the global memory of the graphic devices (device memory) to the system memory (host memory), synchronize mechanism and conditional instructions.

As the data transfer from the system memory to the global memory of the graphic devices should be reduced to a minimum, a good solution is to transfer the description of the finite element mesh to the graphic device global memory and build the system of equations using the graphic processor [4], [5], [6]. However, it should be remembered that the finite element meshes that are used nowadays often consist of a few million nodes. Transferring such a large amount of data is a very time-consuming process and, moreover this data can reduce the limited resources of the device.

The parallelization operation of the matrix building process on the basis of the finite element mesh is very complicated. This process requires a lot of conditional instructions and synchronize blocks, which causes a significant decrease in the performance of computations made on the graphic processor (nVidia GT200 in this case).

The values of elements in i -row of the coefficient matrix depend on the finite elements, which include the node connected with the node with index i . A single node in the finite element mesh can belong to several finite elements. No direct method exists to identify these finite elements solely on the basis of the node index. These indices can only be read from the finite element mesh. Fig. 2 shows the process of building the global coefficient matrix. Non-zero elements of the global coefficients matrix are determined as the sum of several values which depend on those finite elements

which include the pair of nodes with indexes corresponding to the indexes of the row and column of those elements. Constructing the global matrix of coefficient in this way definitely makes the parallelization process more difficult.

The approach presented in this paper uses a modified way of building the system of equations which divides this process into two separate phases. The first requires information from the finite element mesh. This phase is performed only once at the beginning of the computer simulation. An appropriate transformation of the matrix obtained from the Finite Element Method makes this part of the matrix independent from the nodal temperatures and enthalpy values, determined in subsequent time steps. The second phase is performed in each time step and is dependent on the nodal temperature values and enthalpy from previous steps.

A. Two-step building of the system of linear equation

The system of equations resulting from the transformation of a differential equation, which describes the process of solidification, into the numerical model using the Finite Element Method can be finally written in matrix formulation (1) [7], [8], [9].

$$\mathbf{M}\mathbf{H}^{n+1} + \Delta t\mathbf{K}^n\mathbf{T}^{n+1} = \mathbf{M}\mathbf{H}^n + \Delta t\mathbf{b}^{n+1} \quad (1)$$

The elements of (1), which are built on the basis of the finite element mesh description are: matrix \mathbf{M} (2) and conductivity matrix \mathbf{K} (3) [10].

$$\mathbf{M}^e = \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega \quad (2)$$

$$\mathbf{K}^e(T) = \int_{\Omega^e} \lambda(T) \nabla^T \mathbf{N} \cdot \nabla \mathbf{N} d\Omega \quad (3)$$

Matrix \mathbf{M} is independent from the nodal temperature and enthalpy values in subsequent time steps. This matrix may be determined at the beginning of the simulation, before the first time step is executed. This matrix can be used in the same form in the subsequent time steps. Matrix \mathbf{M} is a diagonal matrix, and it is preferable to store it in the form of a simple vector, as this helps to save priceless graphic device memory.

In contrast, conductivity matrix \mathbf{K} is a temperature function. The element which causes this dependency is thermal conductivity coefficient λ . The value of the thermal conductivity coefficient is determined during the building of the matrix of coefficients for each finite element. If thermal conductivity coefficient λ is not dependent on the spatial coordinates, it can be taken out before the integral. After completing these steps, equation (3) assumes the form (4).

$$\mathbf{K}^e = \lambda(T) \int_{\Omega^e} \nabla^T \mathbf{N} \cdot \nabla \mathbf{N} d\Omega \quad (4)$$

After integration, conductivity matrix \mathbf{K} (for triangular elements) takes the form (5), where \mathbf{K}^e is a local coefficient matrix for the finite element, A the surface area of the finite element, C_{ij} are coefficients which depend on the spatial coordinates of the nodes belonging to the finite element.

The value of thermal conductivity coefficient λ for the finite element is calculated as the average value of the thermal conductivity coefficients determined for the nodes belonging to that finite element. After building the local

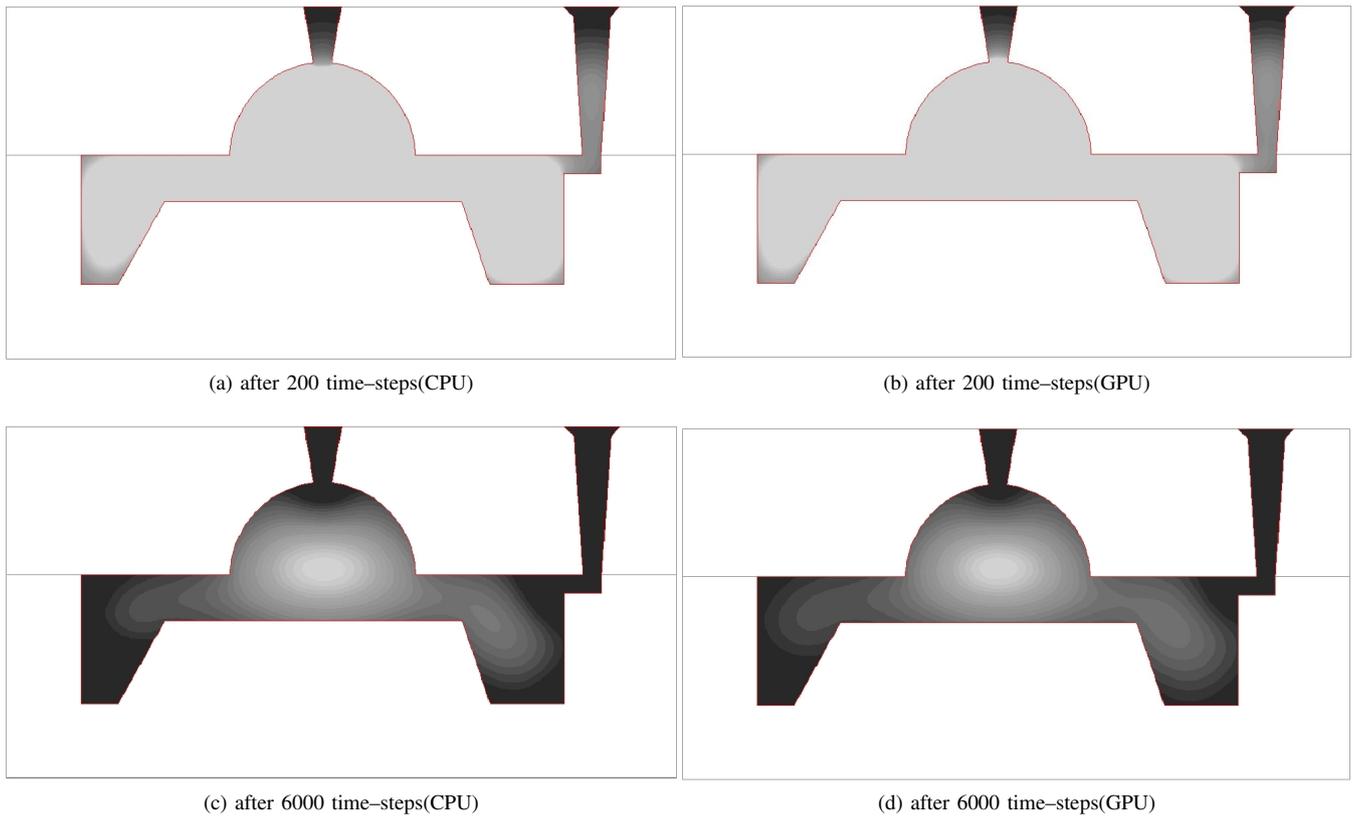


Fig. 4: Distribution of the solid phase

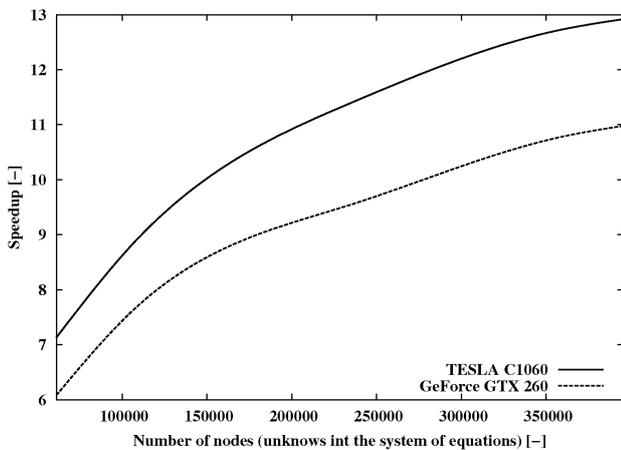


Fig. 5: Speed up obtained with the use of GPU-based simulation software

- 1) nVidia TESLA C1060 (4 GiB GDDR),
- 2) GeForce GTX 260 (896 GiB GDDR).

Both the above are equipped with an nVidia GT 200 processor with 240 CUDA cores which has a peak performance of about 1 Tflops in single float operations. The authors used CUDA version 3.2 in the graphic processor implementations.

The casting material is aluminum alloy with the addition of copper and the mold is assumed to be made from steel. The material properties are listed in Table I. The initial temperature of pouring was equal to 960 K and this value was the value of the initial condition temperature for the region of the cast in the performed calculations. The initial temperature

TABLE I: Physical properties of cast material (Al-2%Cu), and mold

Quantity	Unit symbol	Value
Thermal Conductivity of solid phase	$\frac{W}{m \cdot K}$	262
Thermal Conductivity of liquid phase	$\frac{W}{m \cdot K}$	104
Density of solid phase	$\frac{kg}{m^3}$	2824
Density of liquid phase	$\frac{kg}{m^3}$	2498
Specific heat of solid phase	$\frac{J}{kg \cdot K}$	1077
Specific heat of liquid phase	$\frac{J}{kg \cdot K}$	1275
Solidus temperature	K	853
Liquidus temperature	K	926
Melting temperature of pure metal	K	933
Latent heat of solidification	$\frac{J}{kg \cdot K}$	390 000
Thermal conductivity of mold	$\frac{W}{m^2 \cdot K}$	40
Density of mold	$\frac{kg}{m^3}$	7500
Specific heat of mold	$\frac{J}{kg \cdot K}$	620

of the mold was set at 560 K. Newton's boundary condition is assumed on all the surfaces of the mold, assuming heat exchange with the environment with a coefficient equal to $100 \frac{W}{m^2 \cdot K}$. The ambient temperature has a value of 300 K in all boundary conditions. The heat exchange between the casting and the mold is obtained from a type IV boundary condition, which assumes the heat exchange through the insulation layer of the conductivity coefficient of the separating layer to be $800 \frac{W}{m^2 \cdot K}$.

Fig. 5 shows that speed up depended on the size of the task (number of nodes in the finite element mesh). In small tasks the speed up is minor. This can be explained by the time

taken to copy the data from the system memory to GPU memory and to copy the results into the system memory from the graphic device memory (first and last time-step). No differences were noted in the process of the computer simulation implemented on a general purpose processor and graphics processor. The simulation results (temperature and part of the solid phase) show that there is a slight difference between the results obtained with the GPU-based and CPU-based software. These differences are minor and do not exceed 0.1% and have no effect on the simulation process. This is illustrated in Fig. 4 which shows a part of the solid phase after 200 and 6 000 time-steps of computer simulation of the casting solidification process realized on CPU (Fig. (4a) and Fig. (4c)) and GPU (Fig. (4b) and Fig. (4d)). Comparing these figures it can be seen that the simulation process in both cases is the same.

IV. CONCLUSION

In this article, the authors present a new method of parallelization for the computer simulation of the casting solidification process and its implementation on graphics processors compatible with CUDA architecture. The proposed method divides the process of building the system of equations into the two phases. This solution improves the efficiency with which the available system resources are used. The great advantage of the developed solution is that it is easily adapted to the different architectures of multicore processors.

The speed up observed during the computer simulation of the casting solidification process confirms that the use of graphic processors in engineering simulations brings significant benefits. It was also noted that with an increase in the number of unknowns in the system of equations, the time needed to solve such a system increases linearly. At the same time, as the size of the task increases so does the speed up observed in the computations with the graphic processor.

Having regard to the the above results it can be stated that using graphic processors in engineering simulations seems to be a viable solution as this approach can significantly reduce the time needed for research.

REFERENCES

- [1] R. Strzodka, M. Dogger, and A. Kolb, "Scientific computation for simulations on programmable graphics hardware," *Simulation Modeling Practice and Theory*, vol. 13, pp. 667–680, 2005.
- [2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using cuda," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, 2008.
- [3] nVidia, *CUDA C Best Practices Guide v3.2*, 20 August 2010.
- [4] P. Pospíchal, J. Schwarz, and J. Jaroš, "Parallel genetic algorithm solving 0/1 knapsack problem running on the GPU," in *16th International Conference on Soft Computing MENDEL 2010*. Brno University of Technology, 2010, pp. 64–70.
- [5] C. Lee, X. Wei, J. W. Kysar, and J. Hone, "Measurement of the Elastic Properties and Intrinsic Strength of Monolayer Graphene," *Science*, vol. 321, pp. 385–388, 2008.
- [6] nVidia, *Optimization. OpenCL Best Practices Guide*, 27 May 2010.
- [7] N. Sczygiol, *Numerical modelling of thermo-mechanical phenomena in a solidifying casting and mold*. Wydawnictwo Politechniki Czestochowskiej, Czestochowa 2000. (in Polish).
- [8] N. Sczygiol and G. Szwarz, "Application of enthalpy formulations for numerical simulation of castings solidification," *Computer Assisted Mechanics and Engineering Sciences*, vol. 8, no. 1, pp. 99–120, 2001.
- [9] O. Zienkiewicz, *The finite element method, Volume I, the Basis, 5th ed.* Oxford: Butterworth-Heinemann, 2003.

- [10] N. Sczygiol, G. Szwarz, and R. Wyrzykowski, "Numerical modelling of equiaxed structure formation during solidification of a two-component alloy," in *2nd European Conference on Computational Mechanics, Krakow*, 2001, pp. 820–821.