Date of publication JUN-28, 2021, date of current version MAY-28, 2021.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.20.2.2177

Multithreaded Acceleration of 3D Mathematical Model for Ore Sintering

KYRYLO S. KRASNIKOV
Department of Systems Software, Dniprovskyi State Technical University, Kamianske, Ukraine,
(e-mail: kir_kras@ukr.net) http://www.scitensor.com/lab/metallurgy/sintering

Corresponding author: Kyrylo S. Krasnikov (e-mail: kir_kras@ukr.net).

ABSTRACT One of the widely used methods to accelerate a numerical solver is implementation of multithreading. The problem of thread allocation on-demand at runtime is latency, caused by periodical instantiation of threads. The article is devoted to parallelization of solver for 3D mathematical model of ore sintering, based on software threads reusing them during computation. Computational domain is equally shared among available threads. Each thread writes only to own data partition. A looped barrier is proposed for guaranteed synchronization of all threads after iteration. The method allows scaling performance without recompilation of the solver by using similar CPU with more cores. Measurement of solver performance with 2^{20} nodes using different thread count confirms scalability around 95% for double and single precision arithmetics. Presented pictures of perspective view with three slices of temperature field show influence of heat loss from pallets walls. A cross section of temperature field in layer after 16 minutes of sintering is calculated with appearance of two high-temperature regions inside. Comparison of temperature field with literature data gives good correspondence. The computer model takes into account important chemical reactions, such as, coke burning, carbonate dissolution, water vaporization, as well as mass-heat transfer inside the sinter layer and can be used in metallurgical plants to increase effectiveness of sintering.

KEYWORDS multithreading; numeric solver; 3D model; ore sintering.

I. INTRODUCTION

TODAY ore sintering is actively used in metallurgical plants as preparation stage before getting liquid iron in furnace. At the stage, the main resources (ore, coke, and limestone) are fused with collected dust or fines (remaining after previous sintering) at temperature around 1500K. Then the product (sinter) is cooled and broken into pieces for smelting in furnace. The sintering helps plant to reduce waste and to save some number of resources. Also sinter has properties, which make furnace to work longer.

So sintering is complex process with nonlinear dependencies. Thus, to precisely predict it a math model needs time-consuming calculation of a large number of numbers. In addition to minimize approximation errors of a numerical scheme and provide a sufficient accuracy for a complex mathematical model a numerical solver of partial differential equations (PDEs) needs a small spatial and time step. The small step and minimization of computation time leads to the necessity of a high-performance optimization based on existing computer technologies. That's why acceleration of computing is an important problem.

Historically the development of a central processing unit (CPU) goes in the course of increasing CPU cores number instead of CPU frequency, because the latter requires a nonlinear growing of electric power consumption. Multi-core CPU gave an ability to scale up running performance with acceptable power consumption at the cost of increasing complexity of hardware and software implementation.

Talking about hardware: nowadays a computer market has a lot of multi-core processors with an interesting technology, called the simultaneous multithreading (SMT), which in an appropriate situation allows a single core to simultaneously process two threads, increasing CPU performance like if it would have more cores than it has.

In our time, operating systems (OSes) offer effective instruments for programmers to simplify parallelization of

their applications. Such widely used instrument in a software development is multithreading. According to it an each thread uses an available core of CPU to complete a task. So to accelerate a program execution, one needs to accurately distribute the execution of time-consuming code across the threads. The OS job is to optimally associate physically available CPU cores with created software threads for a maximum load of the processor.

Today, the multithreading is implemented in many free parallel software frameworks and standards, such as Boost.Thread or OpenMP, which are often used by scientists for a solving of partial differential equations [1, 2].

In this paper, performance benefits of the multithreading will be counted during a numerical solution of a mathematical model with PDEs.

II. RELATED WORKS

In the previous paper [3], the author considered a problem of heat and mass transfer in a two-dimensional statement. However, a three-dimensional statement better fits agglomeration process due to simultaneous movement of a sinter layer on an agglomeration machine and the air through the layer. Also a heat loss from pallet walls, which affects a temperature distribution in the sinter layer, should be taken into account. The usage of multicore architecture of present CPUs will speed up calculation of the mathematical model.

The author of the paper [4] discusses about the researcher's need for a high performance computing, since it is an instrument to get a new knowledge and a good solution to their scientific problems. Also majority of researchers don't know computer science well and doesn't have enough time to implement their mathematical models. So they search for a released high performance programs. The author gives interesting examples of parallelization and concurrency.

The authors of the paper [5] successfully parallelized different variants of iterative Conjugate Gradient methods for solving finite-difference implicit problem using an OpenMP API with 228 threads and Intel Xeon Phi with 57 cores. A good refining of a numeric grid significantly increases effectiveness of the OpenMP implementation.

Another paper [6] is devoted to parallelization of wave equation computation using MATLAB/ Octave, OpenMP and CUDA parallel implementation of mathematical model. The author of the paper concludes about good fitting of CPU implementation to a small problem and GPU based solution – to large and very large data processing.

In the thesis [7], the author tests two CPU architectures from the same manufacturer using OpenMPI. At the page 79, he shows a table with maximum of 12.11 speedup of tiled Jacobi solver using 15 threads comparing to single thread performance.

Paper [8] considers parallelization of vector and matrix multiplication using processor with two cores and four threads. The authors present results about a small difference in run-times of two and four threads. A four cored CPU could give a faster result for the latter case.

In the thesis [9], the author tries to get benefits from hyperthreading technology to maximize load of CPU cores. Regrettably he does not have a big effect with it. Also the other authors [10] recommend turning hyperthreading off to avoid performance gaps.

The article [11] is devoted to parallelized solution of shallow water equations using memory access patterns. The performance comparison using 28-64 threads shows 1.3-1.4x gain of speed.

In the paper [12], the authors use multithreading to achieve 2-3x performance speedup using CPU, and 4-7x – by using GPU for a mathematical model of heat and mass transfer during polymer flooding.

Other authors [13] present a numerical scheme for solving 2D Sine-Gordon equations and its parallelization using OpenMP and FORTRAN. The results given by cluster, which has two CPUs and 24 cores with SMT, show almost constant speedup using 30 threads or lower. However, every addition of 6 threads over mentioned 30 ones gives significantly smaller profit (30x speedup using 48 threads).

The authors of the paper [14] use optimized thread parallelism of GPU with CUDA framework for the fluid flow problem. The comparison with CPU performance shows a large advantage of GPU with 136x gain using single precision arithmetic. Nowadays, a GPU performs a lot slower when it uses double precision arithmetic, but nevertheless it is becomes faster than a CPU using double precision arithmetic.

The authors of the paper [15] present a multithreading library for C++11, which depends on Eigen and ThreadPool libraries and has following features: easy-to-use C++ code constructs; scalability dependency on CPU cores number. The authors consider an influence of hyperthreading on the resulting figures and state about parallelization benefits from using only large data to process.

The article [16] is devoted to a method of multithreading optimization using OpenMP applied for two problems: wave equation and linearized Einstein equations. The presented pictures show good speedup scaling.

The authors of the paper [17] present an interesting pyramidal three-level parallel framework for cluster, in which master processor sends messages to defined constant number of processors – submasters. Each submaster has a list of slave CPUs, sends data to them and waits for data changes from slaves to form a message for master. The framework shows mostly linear performance gain with CPU count ranging from 1 to 40 for the implementation of an ant colony algorithm.

In the paper [18], it is considered a parallelized numeric simulation of the water flow in an open channel based on the Saint-Venant PDEs and single precision arithmetic. The authors reveal certain GPU performance advantage over CPU.

The authors of the paper [19] present object-oriented C++ framework for implementing parallelization, communication and synchronization patterns. In addition, authors compare it with MPI (message passing library) when solving of unsteady incompressible Navier-Stokes problem.

The presented framework approaches a golden center between flexibility and performance of PDE solvers.

As demonstrated in [20], multithreading is successfully applied to accelerate solving of two-dimensional nonlinear Burgers' equation using 8 CPU cores. The proposed method is surprisingly faster than single core by 20 times (one could theoretically expect only 8 times).

The scalability of code performance on a processor cluster can decrease with growing of CPUs number as discovered in [21], which can be explained by time-consuming synchronization between many CPUs.

III. PURPOSE OF THE RESEARCH

The aim of the article is to implement multithreading with thread reusing and synchronization. It is necessary to compare performance of single-threaded app with its multithreaded version during calculation of improved 3D mathematical model of sintering. The improved model takes into consideration a heat loss from the walls of the pallets, where sinter is located. Also, it is appreciated to model the process using three spatial dimensions and getting visualization of developed sintering on the cross sections of layer.

IV. MATHEMATICAL MODEL

A. DESCRIPTION

A porous sinter layer, in which a temperature distribution is simulated, is 30 meters long and starts under the area of gas burners. The area has a width of 2 meters and a length of 4 meters. The layer has a constant height of half a meter (Fig. 1). A computational domain for the mathematical model is rectangular. Z-axis is mirrored down to match the direction of air flow.

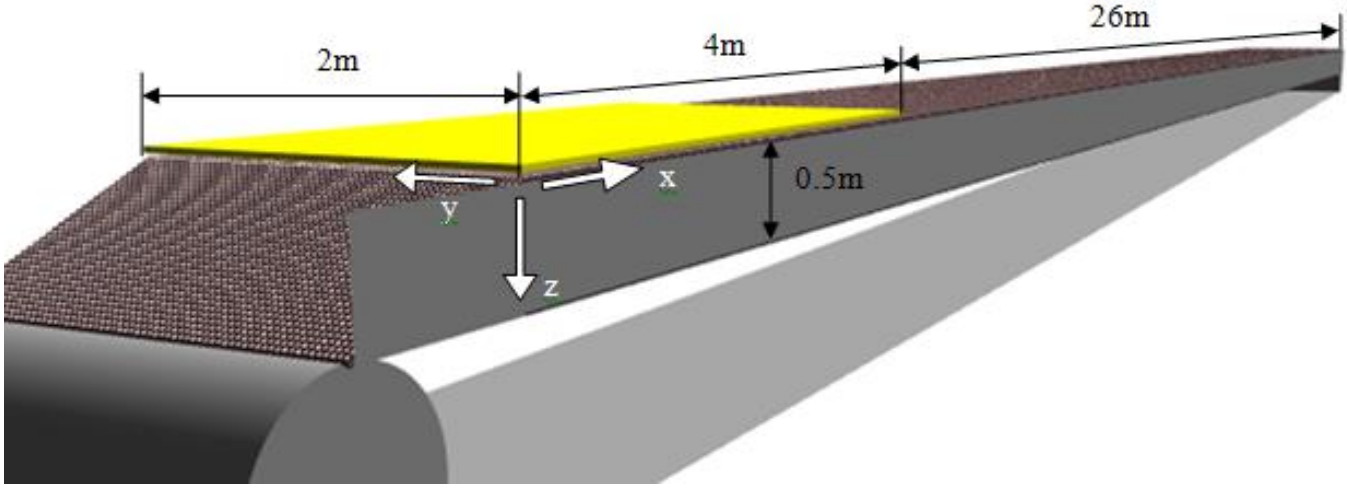

Figure 1. The schematic view of a sintering machine for an agglomeration modeling

The system of equations, that represents essential transfers in layer [3], is following:

$$\frac{\partial M_{\mathrm{O2}}}{\partial t} + v_z \frac{\partial M_{\mathrm{O2}}}{\partial z} = -m_{r\mathrm{O2}}, \quad (1)$$

$$\frac{\partial M_{\mathrm{FeO}}}{\partial t} = -m_{\mathrm{rFeO}} \quad (2)$$

$$\frac{\partial M_{\mathrm{C}}}{\partial t} = -m_{\mathrm{rC}} \quad (3)$$

$$\frac{\partial M_{\mathrm{H2Og}}}{\partial t} + v_z \frac{\partial M_{\mathrm{H2Og}}}{\partial z} = m_{\mathrm{riH2O}} - m_{\mathrm{rkH2O}} \quad (4)$$

$$\frac{\partial M_{\mathrm{H2O}}}{\partial t} = m_{\mathrm{rkH2O}} - m_{\mathrm{riH2O}} \quad (5)$$

$$\frac{\partial M_{\mathrm{CaCO3}}}{\partial t} = -m_{\mathrm{rCaCO3}} \quad (6)$$

$$\frac{\partial T_{\mathrm{gas}}}{\partial t} + v_z \frac{\partial T_{\mathrm{gas}}}{\partial z} = -Q_{\mathrm{tob}} \quad (7)$$

$$\frac{\partial T_{\mathrm{s}}}{\partial t} - a\Delta T_{\mathrm{s}} = Q_{\mathrm{tob}} + Q_{\mathrm{gC}} + Q_{\mathrm{gFeO}} - Q_{\mathrm{dCaCO3}} - Q_{\mathrm{iH2O}} + Q_{\mathrm{kH2O}} - Q_{\mathrm{pl}} + Q_{\mathrm{kr}} \quad (8)$$

$$Q_{\mathrm{tob}} = h(T_{\mathrm{gas}} - T_{\mathrm{s}}) \quad (9)$$

where respectively M_{O2} and M_{H2Og} oxygen and water vapor concentration in air inside layer, M_{FeO}, M_{H2O}, M_{CaCO3} and M_{C} Fe oxide, water, calcium carbonate and carbon concentration in layer, T_{gas} and T_{s} – temperature of air and sinter inside layer, Q_{tob} – heat transfer between sinter and air.

Boundary conditions on the top surface of layer are known inlet temperature of air, concentration of oxygen and vapor. At the layer bottom – free passing fluxes – outflow boundary conditions.

B. ENHANCEMENT

The mathematical model [3] is considered at 3-dimensional statement and is improved by adding a boundary condition, that corresponds to the Newton's law of cooling and heat radiation from the walls of the pallets, because a temperature of the wall often can be greater than 800 °C [22]:

$$\left.\frac{\partial T_s}{\partial y}\right|_{wall} = h(T_{env} - T_s) + \varepsilon\sigma\left(T_{env}^4 - T_s^4\right). \quad (10)$$

In the above formula: T_s and T_{env} are temperatures of the sinter layer and surround atmosphere; h is an empirical coefficient of the heat outflow, which depends on surround convection; σ is Stefan-Boltzman constant; ε is emissivity of the wall.

Mathematical models, like proposed above, have almost 100% parallelizable numeric calculations and sequential parts of computation can't notably influence solver's performance when it implements multithreading.

V. ACCELERATION OF THE SOLVER

For a multithreading computation of the mathematical model, the computational domain is divided on equally sized data chunks along z-axis. There is a local z-index to access a particular number (i.e. temperature) inside a single data chunk. Each chunk is only processed by associated thread, so the calculation is spatially separated and is concurrently done, accelerating overall performance of the calculation. The equality of the chunks also provides a time minimization of the one thread waiting for the other threads. However some workload unbalancing can occur when the first thread or the last one needs to set up a boundary condition for the top or the bottom of layer at the beginning of the next timestep.

The axis of time is divided by timesteps, so all time-variable fields (i.e. temperature field) on the next timestep are computed on the basis of the previous ones. To avoid data races it is needed the guarantee that all the threads have finished a calculation for the current timestep. The guarantee is satisfied by a synchronization of all the threads with a help of a barrier with a loop for the waiting. The looping is meant to be very short because of a small waiting time of the threads, mentioned above. It's worth noting that order of the threads, in which barrier is reached, does not matter in this case. Let *max* is maximum number of the threads passed barrier. Let *current* is count of threads currently reached barrier. And let *iteration* is monotonically incremented index of current timestep then algorithm of the barrier is follow:

1) Add 1 to the *current* and compare it with the *max*.

2) If the result of the comparison is not equal then start a loop which exits when *iteration* changes with going to 1. Else go to 3.

3) Assign 1 to *current* and add 1 to the *iteration*. Also, the latter releases other threads from the loop at 2.

The *iteration* variable of the barrier can be used for a debugging purpose.

Saving computation results to hard disk can be performance bottleneck, so output to disk is done by another thread.

A. IMPLEMENTATION DETAILS

Standard C++ library (namespace std::*) has excellent support among integrated development environments accelerating a realization of mathematical model. Also it provides cross-platform definitions and patterns, which are useful especially for multithreading programming. Thus, one can avoid usage of unsafe C++ pointers or unnecessary implementation popular algorithms (finding maximum, minimum and other). There are classes and structures, which replace raw C++ arrays for numbers (scalar fields) and chars (strings). They help to minimize memory leaks.

Computation of mathematical model is long-running task with repeated iterations. Series of iterations can be performed by single thread. So, during mathematical computation any threads are not created, which can hurt performance. Threads for computation are spawned at the beginning and exited when model time reaches a predefined end. Moreover each thread can be assigned to particular CPU core to minimize context switches by operational system.

Let's consider a start of computation on CPU with four cores and let *z_max* be a number of z-chunks. Spawned threads receive individual constants *z_start* and *z_end*, which are used for looping own z-chunks of data. Then algorithm of thread spawning will be following:

Algorithm: main thread spawns other four ones

```
1. t_index = 0, part = z_max / 4
2. while t_index < 4 do
3.    z_start = (index * part + 1)
4.    z_end = (index * part + 1 + part)
5.    Spawn a thread, which computes part of
   data between z = z_start and z < z_end
   (next algorithm)
6.    Detach spawned thread (joining to the
   main thread is not used in this case)
7.    t_index ++
8. end while
9. Loop for saving computation results
  incoming from spawned threads
```

Let *time_layer* be an index of current tick on the time axis and *saves_per_second* be a frequency of outputting results to a hard disk. Threads are synchronized after each computation of new fields:

Algorithm: time loop in computational thread

```
1. time_layer = 0, save_interval =
saves_per_second / model_time_step
2. while time_layer < end do
3.    Compute new fields using z_start and
z_end
4.    Waiting all threads using
synchronization barrier, mentioned at the
previous paragraph
5.    time_layer ++
6. if time_layer % save_interval == 0
```

```
7. Compute maximum, minimum and average
   value from numbers in computed fields
8. Push results into thread safe queue to
   save them on the main thread
9. end if
10. end while
11. Thread exit
```

Values computed on different threads are combined on the main thread and saved together with computed field.

VI. RESULTS

Parameters (width, height and depth) of sintering layer are taken for real machine. Firstly, let's analyze results of the mathematical model computing. There are three slices of temperature field on the Fig. 2: at 7.5m of length, at 1m of width, and 0.01m of height (z-axis heads from the top to the bottom of the sinter). The Fig. 2 shows the beginning of coke burning around the point of 2m along x-axis. The heat loss from the pallets wall smoothes the temperature field, which can be seen on the y-z slice at the point of 7.5m along x-axis. Without the heat loss the field is isolated by the pallets wall and has a sharp edge as demonstrated on the figure.

The Fig. 3 shows two high-temperature (about 1300˚C) regions diagonally connected by the path of sinter with about 1100 ˚C: the first one is a small around point (4m, 0m), where coke starts burning, and the second one is a large around point (10m, 0.4m), where fire mostly horizontally spreads to the neighborhoods.

Comparison of Figure 2 with results in the paper [23] (Fig. 4) leads to conclusion about similar temperature field with no more 20% difference. On the both figures high-temperature region begins under gas burners and develops to the bottom of layer diagonally.

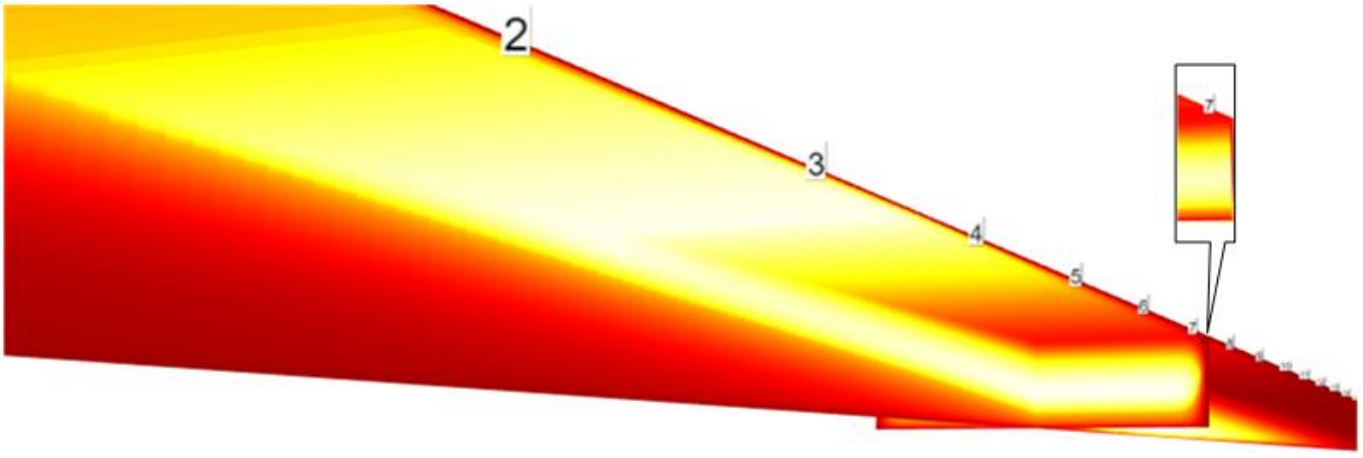

Figure 2. The three slices of temperature field (x=7,5m, y=1m, z=0.01m). Starting ignition area is clearly seen between 2 and 4 meters. Without heat loss effect temperature field has sharp edge (at small frame above)

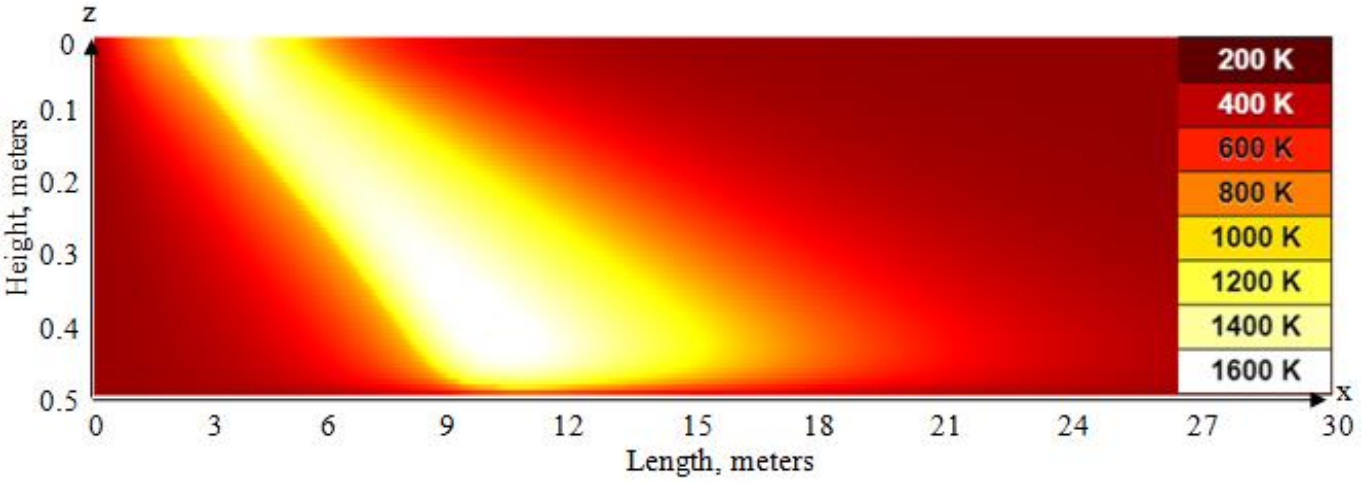

Figure 3. The cross section of the developed temperature field at half a width of layer after 16 min of modeled sintering

The set of experiments are conducted to measure performance of the multithreaded numeric solver. Calculations are done on the CPU with hyperthreading technology: i9-9900T (8 cores, 16 threads) with base clock rate 2,1 GHz. The amount of timesteps is 20 000 (200 seconds of modeling time). For the each timestep the amount of nodes for the calculation is 1 048 576. The frequency of all CPU cores is fixed at 2,1 GHz. A profiler shows that after

start of the experiment more than 90% of running time the CPU spends for a computation of the fields, that is a good load.

Rising of threads count from 1 to 8 (Fig. 4) gives acceleration of approximately 5 times with hyperthreading turned on, and almost 8 – without it, so the hyperthreading slows speed down on a small number of threads. Achieved speed up corresponds to other one in the paper [24] at Fig.C.1 for 1-8 threads, where finite difference method also used. The computation using double precision arithmetic for numbers costs a small performance overhead, because the amount of nodes is sufficiently small for optimizations or for the CPU cache size.

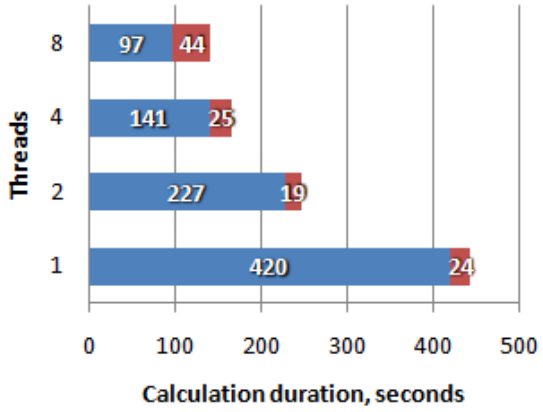

Figure 4. The scalability of the solver's acceleration

Also, effectiveness of proposed algorithm is confirmed in the experiments with different node counts (Fig. 5). The node count grows exponentially, and calculation time is not exactly corresponding to it, however, a general pattern exists.

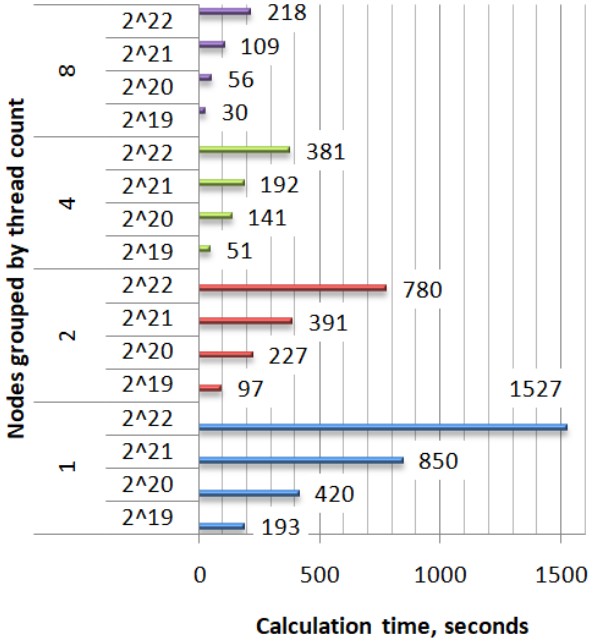

Figure 5. Dependency of computation time on node count using 8 threads (lower is better)

Initial temperature change from 300 K to 500 K gives insignificant performance difference (smaller than 1%).

VII. CONCLUSION

Numerical experiments with single-threaded implementation demonstrate that the time needed to compute mathematical model with interconnected partial differential equations can be very large – from hours to days. Moreover, scientists need to conduct a lot of numerical experiments to determine rational parameters of process. Created computer model with multithreaded implementation significantly reduces calculation time of ore sintering (to around 23%) on CPU with 8 cores.

Presented figures show successfully simulated heat distribution inside sinter layer influenced by set of important chemical and physical processes and heat loss from the walls. Two high-temperature regions are established after 16 minute of model time. The further simulation gives very small changes of the temperature field. Thus, variables approaches state, where they don't depend on time.

Comparing calculated temperature field to the field on the figures in the paper [23], a good correspondence is stated. The presented mathematical model can be used to optimize ore sintering at the plants. Also, the evolution of the temperature field can be viewed at the website URL, provided at the beginning of the paper.

The parallelization of the numeric solver by software threads has confirmed scalability over 95% utilizing 1-8 CPU cores (Fig. 4-5) and has a good perspective to accelerate a calculation of other similar mathematical models.

It will be useful to test this parallelization on a much higher number of cores in the future. Also, further research can include vectorization using SIMD instructions implemented in a majority of CPUs.

References

[1] Q. Wang, J. Liu, X. Cui, G. Fu, C. Gong and Z. Xing, "Accelerating FDTD simulation of microwave pulse coupling into narrow slots on the Intel MIC architecture," *Proceedings of IEEE Pacific RIM Conference on Comunications, Computers, and Signal Processing*, Victoria, Canada, August 24-26, 2015, pp. 263-268, https://doi.org/10.1109/PACRIM.2015.7334845.

[2] A. Valles, W. Zhang, *Optimizing for Reacting Navier-Stokes equations*, in: J. Reinders, J. Jeffers (Eds.), High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, Morgan Kaufmann, Waltham, USA, 2015, pp. 69-85, https://doi.org/10.1016/B978-0-12-802118-7.00004-2.

[3] K.S. Krasnikov, "Computation of heat and mass distribution in sinter layer based on PDEs," *International Journal of Computing*, vol. 17, issue 4, pp. 226-233, 2018, https://doi.org/10.47839/ijc.17.4.1144.

[4] S. Almeida, "An Introduction to High Performance Computing," *International Journal of Modern Physics A*, vol. 28, issue 22, pp. 1-9, 2013, https://doi.org/10.1142/S0217751X13400216.

[5] L.F. Werneck, M.M. de Freitas, H.G. da Silva Junior, G. de Souza, H.P.A. Souto, "An OpenMP parallel implementation for numerical simulation of gas reservoirs using Intel Xeon Phi coprocessor," *Revista Interdisciplinar De Persquisa Em Engenharia (RIPE)*, vol. 2, issue 21, pp. 37-56, 2016, https://doi.org/10.26512/ripe.v2i21.21697.

[6] F.M. Arrayas, *Parallelization of Finite Difference Methods: Nodal and Mimetic solutions of the wave equation*, Polytechnic University of Catalonia, Barcelona, 2016, 116 p.

[7] F. Luporini, *Automated Optimization of Numerical Methods for Partial Differential Equations*, thesis for the degree of Doctor of Philosophy in Computing, Imperial College, London, 2016, 213 p.

[8] X. Liang, A.A. Humos, and T. Pei, "Vectorization and parallelization of loops in C/C++ code," *Proceedings of the 13th International Conference on Frontiers in Education: Computer Science and Computer Engineering*, Las Vegas, USA, July 17-20, 2017, pp. 203-206.

[9] A. Roussel, *Parallelization of iterative methods to solve sparse linear systems using task based runtime systems on multi and many-core architectures: application to Multi-Level Domain Decomposition methods*, Universite Grenoble Alpes, France, 2018, 123 p.

[10] J. Jeffers, J. Reinders, A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*, Elsevier Science, 2016, 662 p., https://doi.org/10.1016/B978-0-12-809194-4.00002-8.

[11] B.M. Ginting, R.-P. Mundani, "Comparison of shallow water solvers: Applications for Dam-Break and Tsunami cases with reordering strategy for efficient vectorization on modern hardware," *Water (Switzerland)*, vol. 11, issue 4, pp. 1-31, 2019, https://doi.org/10.3390/w11040639.

[12] V.M. Konyukhov, I.V. Konyukhov, A.N. Chekalin, "Numerical modeling and parallel computations of heat and mass transfer during polymer flooding of non-uniform oil reservoir developing by system of producing and injecting wells," *Journal of Physics*, vol. 1158, issue 3, pp. 1-8, 2019, https://doi.org/10.1088/1742-6596/1158/3/032018.

[13] I. Hristov, R. Hristova, S. Dimova, "Parallelization of a finite difference scheme for solving systems of 2D Sine-Gordon equations," *Proceedings of the 7th International Conference Distributed Computing and Grid-technologies in Science and Education*, Dubna, Russia, July 4-9, 2016, pp. 250-255.

[14] N. Tran, M. Lee, and S. Hong, "Performance OPTIMIZATION of 3D Lattice Boltzmann flow solver on a GPU," *Scientific Programming*, vol. 2017, article ID 1205892, pp. 1-16, 2017, https://doi.org/10.1155/2017/1205892.

[15] I. Bell and M. Kunick, "NISTfit: A natively multithreaded C++11 framework for model development," *Journal of Research of National Institute of Standards and Technology*, vol. 123, article ID 123003, pp. 1-12, 2018, https://doi.org/10.6028/jres.123.003.

[16] R. Alfieri, S. Bernuzzi, A. Perego, and D. Radice, "Optimization of finite-differencing kernels for numerical relativity applications," *Journal of Low Power Electronics and Applications*, vol. 8, issue 2, article ID 15, pp. 1-13, 2018, https://doi.org/10.3390/jlpea8020015.

[17] M. Craus, L. Rudeanu, "Multi-level parallel framework," *International Journal of Computing*, vol. 3, issue 3, pp. 20-28, 2004, https://doi.org/10.47839/ijc.3.3.301.

[18] W.W. Meng, Y.G. Cheng, J.Y. Wu, Z. Y. Yang, S. Shang and F. Yang, "GPU parallel acceleration of transient simulations of open channel and pipe combined flows," *IOP Conference Series: Earth and Environmental Science*, vol. 240, issue 5, article ID 052025, pp. 1-11, 2019, https://doi.org/10.1088/1755-1315/240/5/052025.

[19] A. Ben-Abdallah, A.S. Charao, I. Charpentier, B. Platea, "Ahpik: a parallel multithreaded framework using adaptivity and domain decomposition methods for solving PDE problems," *Proceedings of the 13th International Conference on Domain Decomposition Methods*, Lyon, France, October 9-12, 2000, pp. 295-301.

[20] S.H. Kuo, C.W. Hsieh, R.K. Lin, W.H. Sheu, "Solving Burgers' equation using multithreading and GPU," *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, Busan, Korea (Republic of), May 21-23, 2010, pp. 297-307, https://doi.org/10.1007/978-3-642-13136-3_31.

[21] S. Titarenko, I. Kulikov, I. Chernykh, M. Shishlenin, O. Krivorot'ko, D. Voronov and M. Hilyard, "Multilevel parallelization: Grid methods for solving direct and inverse problems," *Proceedings of Communications in Computer and Information Science: Supercomputing*, Moscow, Russia, September 26-27, 2016, pp. 118-131, https://doi.org/10.1007/978-3-319-55669-7_10.

[22] Y. A. Cengel, A. J. Ghajar, *Heat and Mass Transfer: Fundamentals & Applications*, fifth ed., McGraw-Hill Education, New York, 2015, 968 p.

[23] J.A. Castro, L.M. Silva, G.A. Medeiros, E.M. Oliveirab, H. Nogami. "Analysis of a compact iron ore sintering process based on agglomerated biochar and gaseous fuels using a 3D multiphase multicomponent mathematical model," *Journal of Materials Research and Technology*, vol. 9, issue 3, pp. 6001-6013, 2020, https://doi.org/10.1016/j.jmrt.2020.04.004.

[24] K.M. Gerke, R.V. Vasilyev, S. Khirevich, D. Collins, M.V. Karsanina, T.O. Sizonenko, D.V. Korost, S. Lamontagne, D. Mallants, "Finite-difference method Stokes solver (FDMSS) for 3D pore geometries: Software development, validation and case studies," *Computers and Geosciences*, vol. 114, pp. 1-61, 2018, https://doi.org/10.1016/j.cageo.2018.01.005.

KYRYLO S. KRASNIKOV *in 2009 graduated from Faculty of Electronics and Computer Engineering, Dniprovskyi State Technical University (DSTU) with M.S. in Software Engineering. At the end of 2016 in National metallurgical academy of Ukraine he defended his thesis for the degree of PhD in Mathematical simulation and methods of calculation. Present job: a senior lecturer at DSTU, Department of systems software. The scientific interests are focused on the application of Computational Fluid Dynamics, System of Solids Dynamics and Thermo-dynamics to the various problems in the ferrous metallurgy and other fields. Professional skills include software projecting using UML as well as programming using modern versions of C++, C#, Java and popular frameworks.*