

A multilevel approach to algorithm and software design for exaflops supercomputers^{*}

B. Glinskiy, I. Kulikov, A. Snytnikov, I. Chernykh, D. Weins

Institute of Computational Mathematics and Mathematical Geophysics SB RAS

A strategy is proposed for the development of algorithms and software for Exaflops supercomputers. The strategy contains three stages. The first stage is the co-design, which is defined as considering the architecture of the supercomputer at all stages of the development of the code. The second is the forward-looking development of algorithms and software for the most promising Exaflops supercomputers. The forward-looking development is based on the simulation of the algorithm behavior within the given supercomputer architecture. The third stage is the estimation of the energy efficiency of the algorithm with different implementation for one single architecture or with different supercomputer architectures. The strategy is shown by the example of two problems from astrophysics and plasma physics.

1. Introduction

The bottleneck in the use of the Exaflops supercomputers will be the incredible degree of parallelism, hundreds of millions or even billions of simultaneously working computation cores. Software development for supercomputers is closely connected with the evolution of their architectures. Thus the study of scalability of the parallel algorithms with the future Exaflops supercomputers is an urgent question. One must note that the computational algorithms evolve slowly compared to the evolution of hardware. So, even for Exaflops supercomputers the present-day computational algorithms will be applied since there is no special mathematics that takes the peculiarities of Exaflops machines into account. Therefore it is important to study right now the perspective of the implementation of the present-day algorithms on future supercomputers and to investigate their parallelism, scalability and energy consumption.

In [1] the co-design concept is introduced, which is the considering the architecture of the supercomputer at all stages of the development of the code. In the present case co-design begins with the physical analysis of the problem. The last stage of co-design is the improvement of the parallel numerical algorithm taking the peculiarities of the architecture into account.

The problem of simulation of the algorithms scalability is not something new; there is a lot of work throughout the world [2]. A good example of earlier work in this field is the PARSIT package [3]. This package is capable to simulate the behavior of a parallel program considering the supercomputer architecture. In [4, 5] programs for algorithm simulation are also proposed. Unfortunately, they are too simple and thus they are not fit for simulation of supercomputer algorithms. Finally, the BIGSIM project must be mentioned [6]. The project is devoted to the development of the simulation workbench that enables to develop, test and tune software by means of simulation of the future computers. This also enables the hardware engineers to improve their solutions with the applications being tested.

In the Institute for System Programming RAS a model of a parallel program was developed under the supervision of Academician V.P.Ivannikov. The model can be efficiently interpreted by an instrumental computer thus providing the precise prediction of the real execution of a parallel program within the given computer. The model is developed for the parallel programs with explicit message interchange using Java together with MPI. Therefore the model is included into ParJava [7,8]. The prediction of execution time for certain spots of the parallel program is performed considering the MPI work time. It means that the model time is corrected with the average time of the RTS (Run Time System)

^{*} This work was partially supported by Russian Foundation for Basic Research (grants 15-31-20150, 15-01-00508, 13-07-00589, 14-01-31199 and 14-01-00392) and by Grant of the President of Russian Federation for the support of young scientists number MK – 6648.2015.9. This project was partially supported by the Russian Ministry of Education and Science Grant 3.961.2014/K.

thread. In such a way the ParJava project enables to solve the wide range of the problems connected with execution efficiency estimate with future supercomputers. Unfortunately, its capabilities are not very high because it is attached to a certain programming language.

In [8, 9] an approach is being developed which is based on multi-agent simulation. This approach suits for simulation of computations perfectly. The node of the supercomputer together with the code being executed on this node is selected as the atomic, independent particle in the model of the computations. Every functional agent simulates the behavior of the node of the supercomputer and the code working on this node. The computations are represented as the set of primitive operations (computations with the core, read/write to the memory, peer-to-peer data exchange, data synchronization between the cores) and of temporal characteristics of each operation. Thus it is possible to estimate the behavior of the algorithms and to develop the improved computational schemes right now by means of the implementation of the algorithms within a simulation model. Here the model displays millions and tens of millions of computational cores. The simulation model shows the bottlenecks of the algorithms and gives the ideas how to modify and improve them and what parameters to change in order to achieve good scalability.

Since the architecture of the exaflops supercomputer is an open question, the energy efficiency of the algorithms becomes a question of great importance. Energy efficiency may be dividing into 3 main parts [10-14]: the efficiency of interaction with the CPU and the RAM, the efficiency of working with network devices and also the efficiency of working with peripherals, for example, the input-output system. In order to increase the energy efficiency while working with the CPU and the RAM it is recommended to use the hardware capabilities at maximum extent. It is also recommended to simplify the code, facilitating its optimization by the compiler. The control of the memory use is necessary. The compiler option «Reduce memory leaks» prevents using the swap file. The use of high performance libraries (e.g. Intel IPP, Intel MKL) enables to improve the program and the energy efficiency of the program. During the development of the program it is recommended to reduce the amount of the data transmitted between the processes and to minimize the number of data transfer operations. Processor load balance can improve the energy efficiency in a great extent. When it is necessary to put a process to the standstill, for example, to collect the data from all the nodes it is recommended to reduce the frequency of the processors/cores that perform these operations. The energy consumption of various peripherals may be equal or even greater than the energy consumption of processors or computation accelerators. It is necessary to reduce addressing the file system, in particular if it is the network file system. The final version of the code must not contain any debug information, all the input/output must be maximally reduced. The use of the latest versions of the compilers with maximum optimization level for the target platform also improves the energy efficiency of the algorithms. For example, the latest Intel compilers enable to obtain recommendations on code optimization. This may result in 30% performance increase with latest Intel processors and accelerators.

2. Co-design of methods for solving of complex computational problems

Co-design of the parallel methods for the solution of large-scale problems is a process complex enough and hard to formalize. This is because every problem has its own features. Even if two problems are from the same area of science, they will still have some differences. The careful consideration of these features and differences may alter not only the parallel computational method, but the mathematical model itself and of course, the program implementation and its efficiency.

It is impossible, of course, to make a «collection of recipes» for efficient solution of all the large-scale problems. However, some general approach is possible to be proposed. In any case, the concept of co-design consists of the following:

1. The formulation of the physical statement of the problem.
2. The mathematical formulation of the physical problem.
3. The development of the numerical method for the solution of the mathematical formulation of the physical problem.
4. The selection of data structures and of the parallel algorithm.
5. The consideration of supercomputer architecture.
6. The use of the development tools.

At the stage of the formulation of the physical statement of the problem it is possible to define the main physical process. This will be the basis on the following stages. For example, the simulation of gas flows may be reduced to the solution of hyperbolic equations (gas dynamics equations) in the case of jet pipe simulation [15]. And in the case spacecraft flying in rarefied atmosphere the gas flow is governed by the kinetic equations solved by either particle methods [16] or by Monte Carlo methods [17]. All these methods are of different nature and thus require different parallel implementation.

Even at the stage of the formulation of the physical statement of the problem it is possible to begin using some parallel formalism and to define the type of parallelism that will use in the solution of the problem. To our opinion the most natural division of the types of parallelism is the following: distributed problems and the problems with dependence parallelism. With no loss of generality all the distributed problems are integration problems. It means that the problem is decomposed into the set of independent subproblems, the results of which are accumulated in some shared object. In the case of numerical integration or summation in quantum chemistry problems [18] and of Monte Carlo methods [17] the result is a number of some scalar characteristics. In the problems of spectral analysis the main objective is to obtain the dynamic process matrix or the 2D plot of the matrix specter [19,20]. In both cases the results are accumulated in some shared object. The same is true with image processing [21]. One must note that the algorithm will require tuning for exaflops supercomputer even the in the case of an integration problem [8].

The problems with dependence parallelism may be divided into three types: explicit operator inversion problem; implicit operator inversion problems solved by grid method; implicit operator inversion problems solved by grid-less methods. Operator inversion problems are primarily the solution of either elliptic equations [23, 24] or parabolic equations reduced to elliptic equations and hyperbolic [25] equations. In this case the main task is the inversion of a sparse matrix by iteration [26] or direct [27] methods. Also some other methods can be applied, namely the methods based on the transformation of the matrix to a simpler form: either two-diagonal [19] or three-diagonal [28] form. The same is true about the ODE system solution methods [29].

Implicit operator inversion by the grid methods is the solution of hyperbolic equations within a compact stencil for most of the continuous medium models: gas dynamics equations [30], magnetic gas dynamics equations [31], first moments of the Boltzmann equation for the description of the stellar component of galaxies[32,33], the equations of relativistic gas dynamics[34], the equations of relativistic gas dynamics with electric field [35,36], the equations of the theory of elasticity [37], the equations of multi-phase hydrodynamics [38], of gravitational gas dynamics [39 – 41] etc. In this case the main strategy of the efficient parallel implementation is the use of the local computational stencil [42]. Implicit operator inversion by grid-less methods is the solution of the equations (in particular, hyperbolic) by representation of the continuous medium of gas [43] or solid body [44] as a group of particles with limited interaction radius or as rarefied medium.

2.1 Co-design of parallel numerical methods for astrophysical simulation

The present authors develop the Lagrangian-Eulerian approach for solving astrophysical problems on the basis of Fluids-In-Cells (FIIC) method and Godunov method for several years. Using of this combination enables to eliminate the Galilean non-invariance and the use of Godunov method at the Eulerian stage enables to simulate the discontinuities correctly.

GPU architecture is a set of concurrently executing threads, combined in a multi-level 2d topology. The maximum performance on this architecture can be obtained in the absence of synchronization between threads. In this case the strategy to co-design a numerical method is to use the computational mesh topology, which directly projected on the topology of GPU architecture (e.g. use of regular mesh), and independent computation of the values at the next step in each cell of the computational domain. The Riemann problem for Euler stage original method satisfies this property, but for the Lagrangian stage needed a modification. This modification allows independently computing advective transport for each cell of the computational domain.

Are traditionally, collisionless component is described using N-body models. Nevertheless, this model has disadvantages as a spurious generation of entropy, increased communication overhead and poor load balancing. Therefore, in co-design algorithms to describe collisionless components in the context of modeling the galaxies collision was selected «Collisionless Stellar Hydrodynamics» ap-

proach. This approach is based on the equations for the first moments of the collisionless Boltzmann equation. This approach is adequate for importance of a movement particles cluster and high kinetic energy. In addition, for describing «Collisionless Stellar Hydrodynamics» models can be used an unified numerical methods and parallel algorithms, that are used to solve the hydrodynamic equations. Consequently, they can be efficiently implemented on GPU-based supercomputers.

2.2 Co-design of parallel numerical methods for plasma physics

In the present case co-design begins at the stage of the physical consideration of the problem. It is known from experiment that the plasma density modulation cannot exceed 300 \%. It means that the number of model particles cannot be increased without a limit. Thus there is no need for dynamic load balancing, and the absence of dynamic balancing improves the reliability and efficiency of the parallel implementation.

At the stage of the numerical method design the field evaluation method was chosen that is built on the basis of Faraday and Ampere laws. In such a way there is no need to solve Poisson equation. Instead, the equations that represent the Faraday and Ampere laws in the numerical form are solved by the Langdon-Lasinski scheme. This results in the field solver with virtually unlimited scalability.

At the stage of supercomputer architecture selection the PIC method details are taken into account. In order to evaluate the new values of position and impulse of a particle it is necessary to know the values of electric and magnetic fields at the present position of the particle. Each of the three components of both electric and magnetic field is stored in a separate 3D array. In such a way six 3D arrays are accessed at each time step for each particle. Since the particles are situated randomly in the domain, the access to the arrays is also unordered. It means that the use of the cache memory can not reduce the computation time. If a part of the field array was fetched to the cache in the process of computation of the particle movement, it would be impossible to use this part of the field array for the computation with the next particle, because it is (most likely) situated in a completely different part of the subdomain. Since the cache memory cannot store all the six arrays for fields, one has to access the RAM (Random Access Memory) for computation of the particle movement. And since the performance of the processor is usually limited by the memory bandwidth, it is the memory bandwidth that determines the speed of the computation with particles and the performance of the program as a whole (particles take from 60% to 90% of the total time).

This fact determines the transition to the supercomputers equipped with GPUs because CUDA has a lot of tools to accelerate memory use for the PIC method.

At the stage selection of software design tools is the following. For the PIC method with a very big number of independently processed elements (the model particles) the use of CUDA technology is very efficient. Other parallel technologies for hybrid supercomputers such as OpenCL, OpenMP, OpenACC could be also used but it is CUDA that gives the possibility to use the highest number of parallel processes and to get the highest performance.

The last stage of co-design is the adaptation of the algorithm to the GPU architecture. The traditional PIC method implementation — all the particles stored in one very big array is unacceptable for GPUs because it prevents the use of the advantages of GPU memory. Thus the particles are distributed between the cells and the computation is conducted as follows: one block of threads for one cell.

3. Scalability simulation of algorithms

The multi-agent simulation system AGNES [45] is used for studying the behavior of the algorithms. There are simulations agents (AstroGrid and PlasmaGrid) were created for scalability simulation the astrophysics and plasma physics problems. All of these agents are simulating computations for each of the problems and sending data between neighbors that simulating computational nodes.

AstroGrid Agent sending messages to two neighbors and expects to receive the messages from both neighbors as implemented in software. A calculation continues only after this cycle.

PlasmaGrid agents are summarizing data for all agents before the next iteration of the loop.

All data sending delays are collected and analyzed according to real run tests of astrophysics and plasma physics codes.

3.1 Astrophysics problem

The model of interacting processes is defined here as the set of threads (not Operating System threads), being concurrently executed on a single computational node. These threads interact with each other by means of information interchange, for example, with MPI. The main parameter of the thread is the execution time and also the time spent on passing the data to another thread. Considered this, let us give the scheme of interactions between threads for the problems of astrophysics and plasma physics.

The peculiarity of parallel grid methods is the possibility for geometrical decomposition of computational domain and further exchange by the boundary values by the adjacent computational nodes only. These methods are the main tool for the solution of hyperbolic equations, not only in astrophysics. Here, if the computational domain contains N^3 cells, then the number of element transmitted to the adjacent computational node is N^2 elements. The interaction scheme may be shown by the following figure:

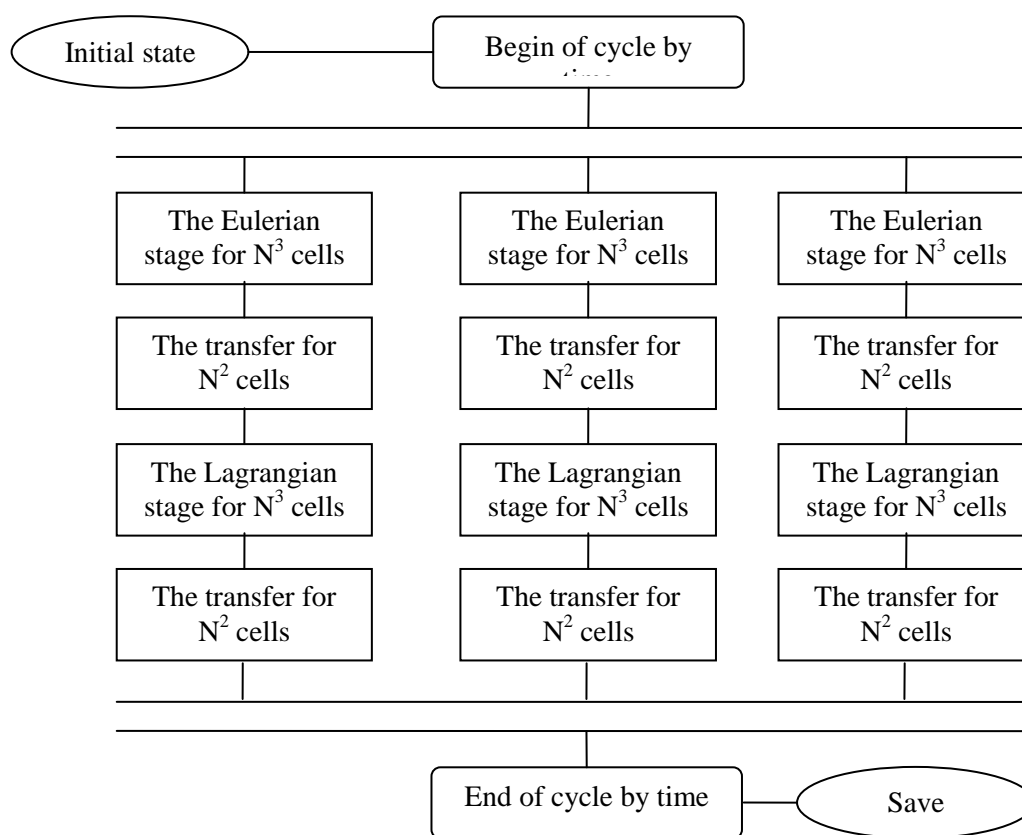


Fig. 1. The scheme of interaction between the computational nodes.

In order to investigate the scalability of grid methods with CPUs, GPUs and Intel Xeon Phi accelerators let us consider the following:

1. Execution time for Eulerian and Lagrangian stage of the Fluids-In-Cells method are given by the formulas:

$$T^{Euler} = T_E \times N^3 \qquad T^{Lagrange} = T_L \times N^3$$

It means that for each type of the computational device the optimal computation time is known for each stage. The total computation time is proportional to the number of cells.

2. The scalability and speedup within a single CPU or accelerator will not be simulated. The reason is that the number of cores of CPUs and accelerators is fixed and also, this number is much less than the prospective number of computational nodes in peta- or exaflops supercomputers. Therefore the execution time for the stages $T_{E,L}$ is optimal for:

- 12 cores of Intel Xeon E5-2697 processor
- 1024 cores of the Nvidia Kepler GPU
- 60 cores of the Intel Xeon Phi 7110 accelerator in the offload mode
- 240 cores of the Intel Xeon Phi 7120 in the native mode

In the case of the CPU the speedup value was 12 within one Intel Xeon (12 cores compared to 1 core), for the GPU the speedup was 55 (1024 cores of Kepler compared to 1 core) and for the Intel Xeon Phi accelerator the speedup was 27 in offload mode (60 cores compared to 1 core) and 54 in native mode (240 core compared to 1 core). This speedup is because of the architecture of the task manager of the computational device. The speedup data was obtained by the computational experiments with GPUPEGAS and AstroPhi codes.

3. The communication time is considered to be a linear function of the number of the transmitted element with the latency taken into account:

$$T^{Comm} = \Lambda + T_C \times N^2$$

where Λ is the latency and T_C is the average time for transmitting of one data element.

4. The feature of the numerical method is that the number of the elements being transmitted is the same for both Eulerian and Lagrangian stages. Consequently, the communication time is also equal.

5. The evaluation and comparison of the communication times is impossible since the computational experiments were conducted with different hardware and different architectures, and in some cases it was just one node of or even one single GPU. Thus it will be presumed that the network of the Siberian Supercomputer Center (SSCC) is being used.

6. In the general case for the solution of the 3D problems with the large number of computational nodes the most efficient geometrical decomposition is the multi-dimensional (among others 3D) decomposition. Most likely, the multidimensional Cartesian topology will be used in Exaflops supercomputers. In figure 1 the linear topology is shown that corresponds the real SSCC architecture. The use of such topology for the interaction between the nodes simplifies the simulation of the execution of threads. At the same time it does not limit the use of multidimensional topologies. This is because the data exchange between threads will grow proportionally to the number of dimensions. The communication time for the K-dimensional decomposition will be connected to the communication time along one dimension according to the formula $T_{KD}^{Comm} = K \times T_{1D}^{Comm} = K \times T^{Comm}$ which does not break the generality of the model presented by figure 1.

7. It must be noticed that in the present model only the extensivity of the computer system is being considered. It is quite natural that both computation performance and data exchange speed will grow with Petaflops computing becoming more common and while moving towards Exaflops. But along with this the computational complexity will also be growing, since the physical model of the astrophysical processes will become more complex, resulting in the increasing number of operations per one cell. Due to this reason we investigate the scalability of concurrently executed processes considering the current hardware parameters and consequently, current level of complexity of the computational models. At present, the characteristic number of cell per one accelerator is $N = 256^3 \approx 16 \times 10^6$.

In order to describe the model of interacting processes let us write the table of the computation and communication times per one element (one cell).

3.2 Physics of plasma problem

The goal of this section is to show the parallelization strategy that must be supported by the PIC implementation template since it is quite clear that the plasma physics problems of interest will never fit into a single CPU or GPU memory.

The program was parallelized by the domain decomposition method. The computational domain is divided into parts along the direction orthogonal to the direction of the beam (along the Y axis, the beam moving along the X axis). The computational grid in the whole domain is divided into equal parts (sub-domains) along the Y axis. Each sub-domain is assigned to a group of processors (in the

case of a multi-core system a single core would be called a processor, since no hybrid parallelization like MPI+OpenMP is employed, just mere MPI).

Furthermore, the particles of each sub-domain are distributed uniformly between processors of the group with no regard to their position, as it is shown in Figure 2.

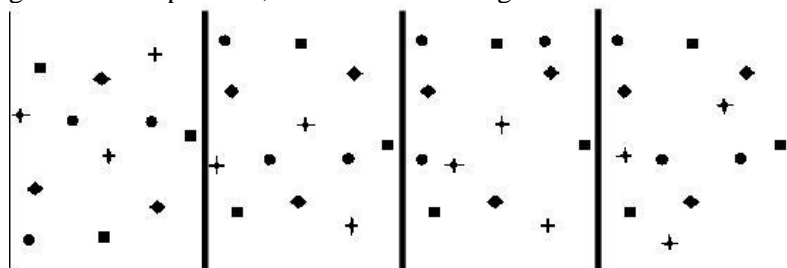


Fig. 2. The scheme of domain decomposition.

The computational domain is divided into 4 subdomains. The particles of each subdomain are distributed between four processors uniformly with no regard to their position. Different symbols (circle, square, diamond, star) denote particles belonging to different processors in the same sub-domain.

Every processor in the group solves the Maxwell equations in the whole sub-domain, and exchanges boundary values of the fields with processors assigned to adjacent sub-domains. Then the movement equations for the particles are solved, and the 3D matrix of the current density and the charge density are evaluated by each processor. But since the processor has only a part of the particles located inside the sub-domain, it is necessary to sum the matrices through all the processors of the group to obtain the whole current density matrix in the sub-domain.

Interprocessor data exchange is performed by the MPI subroutines. The overall communication scheme generally corresponds the scheme given in figure 1 for the astrophysical code. There is only one important difference. The plasma physics code at present has no peer-to-peer communications, just the summation along the whole domain performed by MPI_Allreduce.

3.3 The simulation results

The AGNES simulations of astrophysics code was done for computational nodes with Intel Xeon E5-2697 (5632 nodes with 67584 cores), Nvidia Kepler K40 GPUs (1024 nodes with 1048576 cores) and with Intel Xeon Phi 7110 (4096 nodes with 983040 cores). The AGNES simulations of plasma physics code was done for computational nodes with Intel Xeon E5-2697 (3072 nodes with 36864 cores), Nvidia Tesla2090M GPUs (1536 nodes with 786432 cores) and with Nvidia Kepler K40 GPUs (1536 nodes with 4.4 million cores).

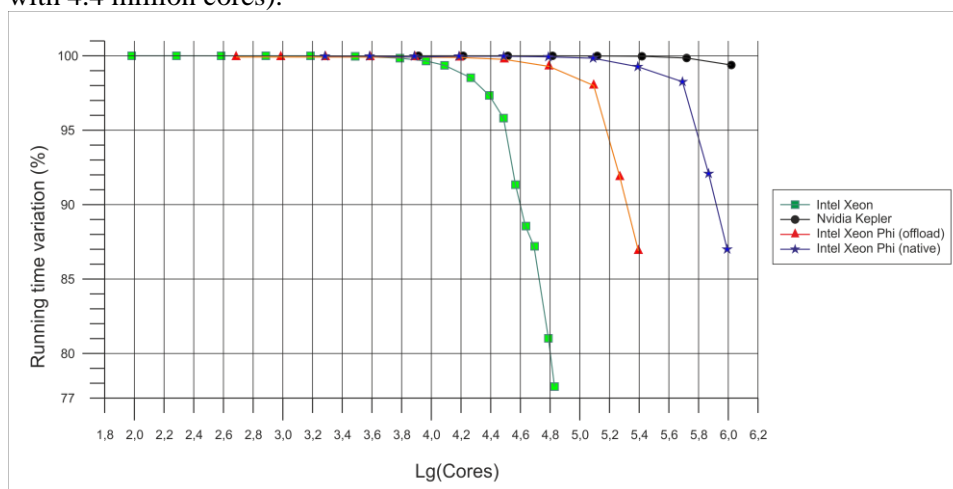


Fig. 3. AGNES scalability simulation result for astrophysical code. Mesh size increases with number of cores.

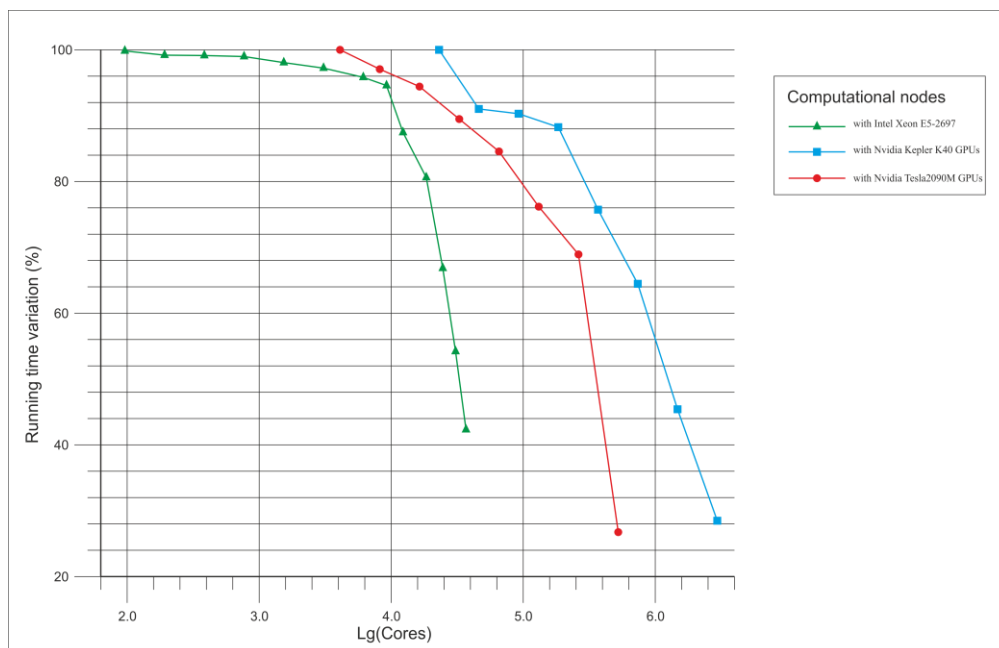


Fig. 4. AGNES scalability simulation result for physics of plasma code. Mesh size increases with number of cores.

The results for astrophysical code simulation are the following:

- 1) Running time increases insignificantly (about 20%) up to the 5120 computing nodes;
- 2) The best scalability was demonstrated for nodes with Nvidia Kepler K40 and Intel Xeon Phi (native mode) accelerators.

The results for plasma physics code simulation are the following:

- 1) The substantial increase of communication messages gives the 58% growth of code execution time for 3072 nodes with Intel Xeon CPUs;
- 2) The plasma physics code has the best scalability on Nvidia Kepler K40 GPUs nodes;

4. Energy efficiency of the algorithms

In order to improve the energy efficiency of the algorithms for the problems of astrophysics and plasma physics each code was being optimized along the following three directions: the efficiency of interaction with the CPU and the RAM, the efficiency of working with network devices and also the efficiency of working with peripherals, for example, the input-output system. The interaction with the CPU and RAM was facilitated with Intel Parallel Studio XE 2016 Beta package, including Intel Vectorization Advisor. The latter analyzes the code and gives offers for code optimization for maximally efficient use of vector instructions. The use of this package gave the 30% performance gain for the astrophysical code for Intel Sandy Bridge CPU. For both astrophysics and plasma physics problems presented here the most efficient methods were selected. Efficiency here means the minimal number of network data exchanges.

Name	TSelf	TSelf	TTotal	#Calls	TSelf/Call
▲ Group Application	78.174 s		80.0835 s	4	19.5435 s
Process 0	19.2928 s	<div style="width: 75%;"></div>	20.021 s	1	19.2928 s
Process 1	19.8552 s	<div style="width: 75%;"></div>	20.0281 s	1	19.8552 s
Process 2	19.9098 s	<div style="width: 75%;"></div>	20.0216 s	1	19.9098 s
Process 3	19.1162 s	<div style="width: 75%;"></div>	20.0128 s	1	19.1162 s
▷ Group MPI	1.90951 s		1.90951 s	384	4.97269e-3 s

Fig. 5. The profile of the astrophysical code with Intel Core-i7 3820 Sandy Bridge processor.

Name	TSelf	TSelf	TTotal	#Calls	TSelf/Call
▲ Group Application	154.396 s		157.082 s	4	38.5989 s
Process 0	38.4594 s		39.2725 s	1	38.4594 s
Process 1	38.9564 s		39.2733 s	1	38.9564 s
Process 2	38.9702 s		39.2756 s	1	38.9702 s
Process 3	38.0096 s		39.2609 s	1	38.0096 s
▷ Group MPI	2.68675 s		2.68675 s	384	6.99675e-3 s

Fig. 6. The profile of the astrophysical code for the computation node SMP-G7 (Intel E7-4870) at the Siberian Supercomputer Center.

Name	TSelf	TSelf	TTotal	#Calls	TSelf/Call
▲ Group Application	44.1507e+3 s		45.0657e+3 s	16	2.75942e+3 s
Process 0	2.75706e+3 s		2.81676e+3 s	1	2.75706e+3 s
Process 1	2.7155e+3 s		2.81672e+3 s	1	2.7155e+3 s
Process 2	2.73732e+3 s		2.81674e+3 s	1	2.73732e+3 s
Process 3	2.73277e+3 s		2.81675e+3 s	1	2.73277e+3 s
Process 4	2.73852e+3 s		2.81672e+3 s	1	2.73852e+3 s
Process 5	2.70039e+3 s		2.81675e+3 s	1	2.70039e+3 s
Process 6	2.73803e+3 s		2.81674e+3 s	1	2.73803e+3 s
Process 7	2.75461e+3 s		2.81677e+3 s	1	2.75461e+3 s
Process 8	2.78787e+3 s		2.8166e+3 s	1	2.78787e+3 s
Process 9	2.78898e+3 s		2.81653e+3 s	1	2.78898e+3 s
Process 10	2.78098e+3 s		2.81635e+3 s	1	2.78098e+3 s
Process 11	2.78907e+3 s		2.81662e+3 s	1	2.78907e+3 s
Process 12	2.78067e+3 s		2.81623e+3 s	1	2.78067e+3 s
Process 13	2.79598e+3 s		2.8167e+3 s	1	2.79598e+3 s
Process 14	2.77751e+3 s		2.81614e+3 s	1	2.77751e+3 s
Process 15	2.77545e+3 s		2.81654e+3 s	1	2.77545e+3 s
▷ Group MPI	914.946 s		914.946 s	1776	515.172e-3 s

Fig. 7. The profile of the astrophysical code for the node G8-K40 (Intel Xeon E5-2650v2) at the Siberian Supercomputer Center.

The optimization of the numerical method of the astrophysical code and the use of the last version of the Intel Parallel Studio XE 2016 Beta package resulted in the excellent result for the load balancing for the processor cores. The time for the MPI operations does not exceed 2-3% of the total execution time. Figures 5-7 present the profiles of the codes.

The figures 7-9 show the profile for plasma physics code. The co-design of the numerical algorithm and the use of latest CUDA 7.0 optimization techniques together with Intel Parallel Studio resulted in good load balancing with MPI operation time not exceeding 10%.

Name	TSelf	TSelf	TTotal	#Calls	TSelf/Call
▲ Group Application	128.395 s		136.26 s	10	12.8395 s
Process 0	12.8576 s		13.7542 s	1	12.8576 s
Process 1	13.0029 s		13.6415 s	1	13.0029 s
Process 2	13.1126 s		13.6327 s	1	13.1126 s
Process 3	12.6506 s		13.5972 s	1	12.6506 s
Process 4	13.1373 s		13.5917 s	1	13.1373 s
Process 5	12.703 s		13.5807 s	1	12.703 s
Process 6	12.3619 s		13.6444 s	1	12.3619 s
Process 7	12.1864 s		13.6153 s	1	12.1864 s
Process 8	13.1687 s		13.6079 s	1	13.1687 s
Process 9	13.2136 s		13.5945 s	1	13.2136 s
▷ Group MPI	7.86551 s		7.86551 s	3200210	2.45781e-6 s

Fig. 8. The profile of plasma physics code at the G8-K20 nodes (Intel Xeon E5-2650v2 and 3 Nvidia Tesla M2090) at the Siberian Supercomputer Center.






























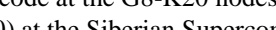
Name	TSelf	TSelf	TTotal	#Calls	TSelf/Call
▲ Group Application	389.82 s		422.531 s	30	12.994 s
Process 0	12.5916 s		13.9837 s	1	12.5916 s
Process 1	12.941 s		14.2177 s	1	12.941 s
Process 2	12.9323 s		14.2493 s	1	12.9323 s
Process 3	12.5263 s		14.037 s	1	12.5263 s
Process 4	13.4026 s		14.132 s	1	13.4026 s
Process 5	13.6495 s		14.1396 s	1	13.6495 s
Process 6	12.9546 s		14.0106 s	1	12.9546 s
Process 7	13.4088 s		14.1386 s	1	13.4088 s
Process 8	13.5501 s		14.1408 s	1	13.5501 s
Process 9	13.0014 s		14.0147 s	1	13.0014 s
Process 10	12.6144 s		14.1758 s	1	12.6144 s
Process 11	12.828 s		14.1234 s	1	12.828 s
Process 12	12.3164 s		14.0278 s	1	12.3164 s
Process 13	12.6281 s		14.159 s	1	12.6281 s
Process 14	13.1659 s		14.0234 s	1	13.1659 s
Process 15	13.3675 s		14.0839 s	1	13.3675 s
Process 16	13.1959 s		14.1117 s	1	13.1959 s
Process 17	13.3358 s		14.168 s	1	13.3358 s
Process 18	13.4274 s		14.1031 s	1	13.4274 s
Process 19	13.2678 s		14.1091 s	1	13.2678 s
Process 20	12.3406 s		14.0329 s	1	12.3406 s
Process 21	12.8743 s		14.08 s	1	12.8743 s
Process 22	12.3167 s		13.9653 s	1	12.3167 s
Process 23	12.314 s		14.0282 s	1	12.314 s
Process 24	13.3187 s		14.1198 s	1	13.3187 s
Process 25	13.4332 s		14.023 s	1	13.4332 s
Process 26	12.8908 s		13.992 s	1	12.8908 s
Process 27	13.1831 s		14.1562 s	1	13.1831 s
Process 28	13.09 s		14.0077 s	1	13.09 s
Process 29	12.9528 s		13.9768 s	1	12.9528 s
▷ Group MPI	32.7115 s		32.7115 s	9600660	3.40721e-6 s

Fig. 9. The profile of plasma physics code at the G8-K20 nodes (2 Intel Xeon E5-2650v2 and 3 Nvidia Tesla M2090) at the Siberian Supercomputer Center.

5. The conclusion

In this paper, we proposed a methodology for the development of algorithms and software for exaflops supercomputers. Our approach has three related stages: co-design stage, scalability simulation, energy efficiency improvements. The co-design technique is based on the idea that algorithm and software design takes into account supercomputer's architecture. One of the most important stages in our approach is the scalability simulation. We use AGNES software package to predict the scalability of algorithms on possible exaflops supercomputer's architecture. Energy efficiency improvements stage consists of different code optimizations such as amplification of CPU or GPU usage, load balancing and MPI operations reduction. Our approach is illustrated by two examples: astrophysics and plasma physics codes. As a result of our approach for software development we got highly parallelized codes with suitable load balancing and MPI operations time.

References

1. Boris Glinskiy, Igor Kulikov, Alexey Snytnikov, Alexey Romanenko, Igor Chernykh, VitalyVshivkov. Co-design of Parallel Numerical Methods for Plasma Physics and Astrophysics Supercomputing frontiers and innovations. Vol 1, No 3, pp. 88-98 (2014).

2. Anand Sivasubramaniam, Aman Singla, Umakishore Ramachandran, H. Venkateswaran: A Simulation-based Scalability Study of Parallel Systems, *Journal of Parallel and Distributed Computing*, Vol 22(3), pp. 411–426 (1994).
3. G. Racherla, S. Killian, L. Fife, M. Lehmann, and R. Parekh. Parsit: A parallel algorithm reconfiguration simulation tool. In: *Proc. of International Conference on High Performance Computing* (1995).
4. Roshan M. D’Souza, Mikola Lysenko, Simeone Marino, and Denise Kirschner: Data parallel algorithms for agent-based model simulation of tuberculosis on graphics processing units. In: *Proceedings of the 2009 Spring Simulation Multiconference (SpringSim ’09)*. Society for Computer Simulation International, San Diego, CA, USA (2009).
5. Michael T. Goodrich: Simulating Parallel Algorithms in the MapReduce Framework with Applications to Parallel Computational Geometry CoRR, <http://dblp.unitrier.de/db/journals/corr/corr1004.html#abs-1004-4708>.
6. Peter Kogge (Ed.): Peter Kogge(Ed.). ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, DARPA report. <http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf>.
7. Ivannikov, V., Avetisyan, A., Padaryan, V.: Evaluation of dynamic characteristics of a parallel program on a model. *Programming*, vol. 4, pp. 21–37 (in russian) (2006).
8. Glinsky, B., Rodionov, A., Marchenko, M., Podkorytov, D., and Weins, D.: Scaling the Distributed Stochastic Simulation to Exaflop Supercomputers. *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICSS)*, 2012 IEEE 14th International Conference on, *Proceedings*, pp. 1131--1136, IEEE (2012).
9. B.M. Glinskiy, A.S. Rodionov, M.A. Marchenko, D.I. Podkorytov, D.V. Vins. Agent oriented approach to imitation of distributed statistical modeling in application of exaflops supercomputer // *Vestnik YURGU*, No. 18 (277), Vol. 12., pp. 94-99 (2012).
10. Petter Larsson. Energy-Efficient Software Guidelines. White Paper for the Intel Software Solutions Group (2008).
11. Susanne Albers , Hiroshi Fujiwara, Energy-efficient algorithms for flow time minimization, *ACM Transactions on Algorithms (TALG)*, v.3 n.4, p.49-es (2007).
12. Ricardo Bianchini , Ram Rajamony, Power and Energy Management for Server Systems, *Computer*, v.37 n.11, p.68-74 (2004).
13. Elaine J. Weyuker , Filippos I. Vokolos, Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study, *IEEE Transactions on Software Engineering*, v.26 n.12, p.1147-1156, (2000).
14. Eugenio Capra, Chiara Francalanci, Sandra A. Slaughter, Is software “green”? Application development environments and energy efficiency in open source applications, *Information and Software Technology*, Volume 54, Issue 1, p. 60-71 (2012).
15. V. P. Stulov. The lecture about gas dynamics. Moscow: PHYSMATHLIT. (1997).
16. Ye. Bondar, A. Shevyrin, Y. Chen, A. Shumakova, A. Kashkovsky, M. Ivanov. Direct Monte Carlo simulation of high-temperature chemical reactions in air // *Thermophysics and Aeromechanics*. v. 20, I. 5. p. 553-564 (2013).
17. M. A Marchenko. Study of a parallel statistical modelling algorithm for solution of the nonlinear coagulation equation // *Russian Journal of Numerical Analysis and Mathematical Modelling*. v. 23, I. 6. p. 597–613 (2008).
18. S. Ono, Y. Noguchi, R. Sahara, Y. Kawazoe, K. Ohno. TOMBO: All-electron mixed-basis approach to condensed matter physics // *Computer physics communications*. v. 189. p. 20-30 (2015).

19. S. Godunov. The lecture about modern aspects of linear algebra. Novosibirsk: Scientific Book (2002).
20. N. Trefethen, M. Embree. Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators. Book in Mathematics and Statistics (2005).
21. R. Gonzales. Digital Image Processing. Prentice Hall (2007).
22. J. Parker. Algorithms for Image Processing and Computer Vision. Wiley (2010).
23. S. Goreinov, I. Oseledets, D. Savostyanov, E. Tyrtysnikov, N. Zamarashkin. How to find a good submatrix // Matrix methods: theory, algorithms and applications. p. 247-256 (2010).
24. V. Ilin. The parallel processes on exaflops modeling stages // Numerical Methods and Programming. v. 12, i. 1.p. 103–109 (2011).
25. J. Guryeva, V. Ilin. A some parallel methods and technologies of domain decomposition // Scientific Seminar Letters. v. 428. p. 89-106 (2014).
26. V. Ilin. The biconjugate methods in Krylov subspace // Journal of Applied and Industrial Mathematics. v. 4, i. 1. p. 65–78 (2010).
27. Kalinkin, Y. Laevsky, S. Gololobov. 2D Fast Poisson Solver for High-Performance Computing // Lecture Notes in Computer Science. v. 5698. p. 112-120 (2009).
28. Terekhov. Parallel Dichotomy Algorithm for solving tridiagonal system of linear equations with multiple right-hand sides // Parallel Computing. v. 36, i. 8. p. 423-438 (2010).
29. N. Guglielmi, E. Hairer. Implementing Radau IIA methods for stiff delay differential equations // Computing. v. 67. p. 1-12 (2001).
30. S. Godunov, I. Kulikov. Computation of Discontinuous Solutions of Fluid Dynamics Equations with Entropy Nondecrease Guarantee // Computational Mathematics and Mathematical Physics. v. 54, i. 6. p. 1012-1024 (2014).
31. Roe P., Balsara D. Notes on the Eigensystem of Magnetohydrodynamics // SIAM Journal of Applied Mathematics. – 1996. – V. 56, I. 1. – P 57–67.
32. Mitchell N., Vorobyov E., Hensler G. Collisionless Stellar Hydrodynamics as an Efficient Alternative to N-body Methods // Monthly Notices of the Royal Astronomical Society. – 2013. – V. 428. – P. 2674-2687.
33. Kulikov I. GPUPEGAS: A New GPU-accelerated Hydrodynamic Code for Numerical Simulations of Interacting Galaxies // The Astrophysical Journal Supplements Series. – 2014. – V. 214, I. 12. – P. 1-12.
34. J. Ibanez, I. Cordero-Carrion, J. Miralles. On numerical relativistic hydrodynamics and barotropic equations of state // Classical and Quantum Gravity. v. 29, i. 15. ID 157001 (2012).
35. S. Godunov. About inclusion of Maxwell's equations in systems relativistic of the invariant equations // Journal of Computational Mathematics and Computational Physics. v. 53, i. 8. p. 1356-1359 (2013).
36. S. Godunov. Thermodynamic formalization of the fluid dynamics equations for a charged dielectric in an electromagnetic field // Journal of Computational Mathematics and Computational Physics. v. 52, i. 5. p. 916-929 (2012).
37. Godunov S., Romensky E. The elements of continuum mechanics and conservation laws. Novosibirsk: Scientific Book (1998).
38. Romankov, E. Romensky. The Runge–Kutta/WENO method for solving equations for small-amplitude wave propagation in a saturated porous medium // Siberian Journal of Computational Mathematics. v. 17, i. 3. p. 259-271 (2014).

39. V. Springel. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh // *Monthly Notices of the Royal Astronomical Society*. v. 401. p. 791-851 (2010).
40. J. Murphy, A. Burrows. BETHE-Hydro: An Arbitrary Lagrangian-Eulerian Multidimensional Hydrodynamics Code for Astrophysical Simulations // *The Astrophysical Journal Supplement Series*. v. 179. p. 209-241 (2008).
41. N. Chuev. The construction of 3D evolutionary model of polytropic self-gravitational gas dynamics // *The Bulletin of URGUPS*. v. 9, i. 1. p. 14-21 (2011).
42. M.A. Kraeva, V.E. Malyskin. Assembly technology for parallel realization of numerical models on MIMD-multicomputers // *Future Generation of Computing System*. v. 17, i.6. p. 755-765 (2001).
43. R. Gingold, J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars // *Monthly Notices of the Royal Astronomical Society*. v. 181. p. 375-389 (1977).
44. X. Li, F. Mo, X. Wang, B. Wang, K. Liu. Numerical study on mechanism of explosive welding // *Science and Technology of Welding and Joining*. v. 17, i. 1. p. 36-41 (2012).
45. D. Podkorytov, A. Rodionov, O. Sokolova, A. Yurgenson. Using Agent-Oriented Simulation System AGNES for Evaluation of Sensor Networks. LNCS, v. 6235, p. 247-250 (2010).