Peculiarities of numerical algorithms parallel implementation for exa-flops multicomputers

Victor E. Malyshkin

Supercomputer Software Department,
Institute of Computational Mathematics and Mathematical Geophysics,
Russian Academy of Sciences,
6, Pr. Lavrentieva, 630090, Novosibirsk, Russia
E-mail: malysh@ssd.sscc.ru

Abstract: Peculiarities of numerical algorithms parallel implementation for exa-flops multicomputers were considered and the appropriate examples were given. The problems of such big data processing were analysed and also, a solution was suggested. Problems of parallel implementation of large scale numerical models on a rectangular mesh were demonstrated by the parallelisation of the particle-in-cell method (PIC). For similar problems solution, the fragmentation of big data and computations were suggested. Fragmentation automatically provides different useful properties of a target program including dynamic load balancing on the basis of fragments migration from overloaded into underloaded processor elements of a multicomputer.

Keywords: numerical big data processing; parallel programming; large scale numerical models; exa-flops computations; numerical algorithms fragmentation; fragmented programs; FP; dynamic load balancing; processes migration.

Reference to this paper should be made as follows: Malyshkin, V.E. (2014) 'Peculiarities of numerical algorithms parallel implementation for exa-flops multicomputers', *Int. J. Big Data Intelligence*, Vol. 1, Nos. 1/2, pp.65–73.

Biographical notes: Victor E. Malyshkin received his MS in Mathematics from the National University of Tomsk (1970), PhD in Computer Science from the Computing Center RAS (1984), and Doctor of Sciences degree from the National University of Novosibirsk (1993). He founded and currently heading the Chair of Parallel Computing at the Novosibirsk National University. He is one of the organisers of the Parallel Computing Technologies series of international conferences. He has published over 110 scientific papers. His current research interests include parallel computing technologies, parallel programming languages and systems, methods of parallel implementation of large-scale numerical models, dynamic load balancing, etc. His main project now is devoted to the development of fragmented programming technology.

1 Introduction

Current multicomputers are mostly loaded by large-scale numerical simulation. Oncoming peta- and exa-flops multicomputers also seemingly will mostly execute numerical models. Parallel implementation of large-scale realistic numerical models involves big data in processing. Big data processing influences on the choice of necessary processing algorithms. This is the case where the use of new system algorithms like resources and computation allocation, new design principles and new technique for the development as application as well system algorithms and programs are demanded. In this paper, the problems of big data processing in large-scale numerical models only were described, analysed and new solution was suggested.

Large scale numerical models are usually remarkable for irregularity and even dynamically changing irregularity of the data structures (adoptive mesh, variable time step, particles, etc.). For example, in the particle-in-cell method (PIC) (Hockney and Eastwood, 1981; Berezin and Vshivkov, 1980) the test particles are in the bottom of such an irregularity. Hence, with conventional programming systems up to now these models are very difficult for effective parallelisation and high performance execution especially if a multicomputer contains hundred thousand or more nodes.

In order to formulate and to analyse the main problems of parallel implementation of large-scale numerical models on exa-flops multicomputer, let us consider the parallel implementation of numerical models based on the use of the popular now PIC. Many works describe its parallel implementation with the use of conventional programming languages (Kireev et al., 2007; Kraeva and Malyshkin, 2001, 1997; Kireev, 2009; Walker, 1990).

2 The PIC method and the problems of its parallel implementation

Description of algorithms of PIC method parallel implementation is given below, in order to demonstrate and to argue the necessity to develop and to use new system and application algorithms and programs for parallel implementation of large scale realistic numerical models. Only scheme of PIC application parallel implementation is given, mathematical model and computational scheme of a numerical algorithm are not discussed.

2.1 The main idea of PIC implementation

The particle simulation is a powerful tool for modelling of the behaviour of complex non-linear phenomena in plasma physics, hydrodynamics, astrophysics, etc. In the PIC method, applied to simulation of a plasma physics phenomenon, trajectories of a huge number of test particles are calculated as these particles are moved under the influence of the electromagnetic fields computed self-consistently on a discrete mesh. These trajectories represent a desirable solution of the system of differential equations in 6D space (three coordinates, three velocities + time) describing a physical phenomenon under study (Kraeva and Malyshkin, 2001).

2.2 Data representation

A real physical space is represented by a model of simulation domain called the space of modelling (SM). The electric *E* and magnetic *B* fields are defined as vectors and discretised upon rectangular mesh (or several shifted meshes, as shown in Figures 1 and 2). Thus, as distinct from other numerical methods on the rectangular mesh, in the PIC method there are two different data structures – particles and meshes. None of the particles affects another particle. At any moment of modelling a particle belongs to a certain cell of each mesh.

Figure 1 A cell of the SM with the electric E and magnetic B fields, discretised upon shifted meshes

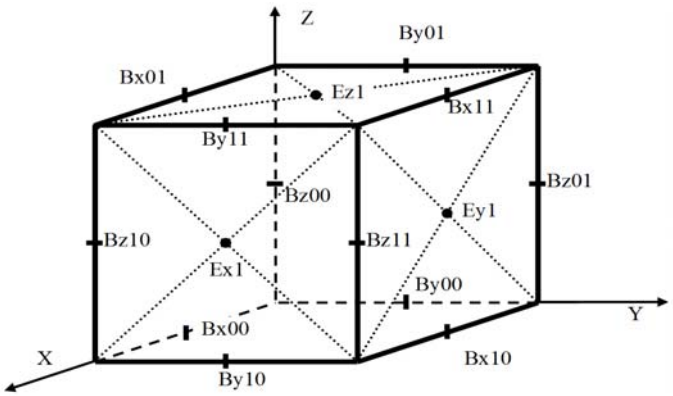
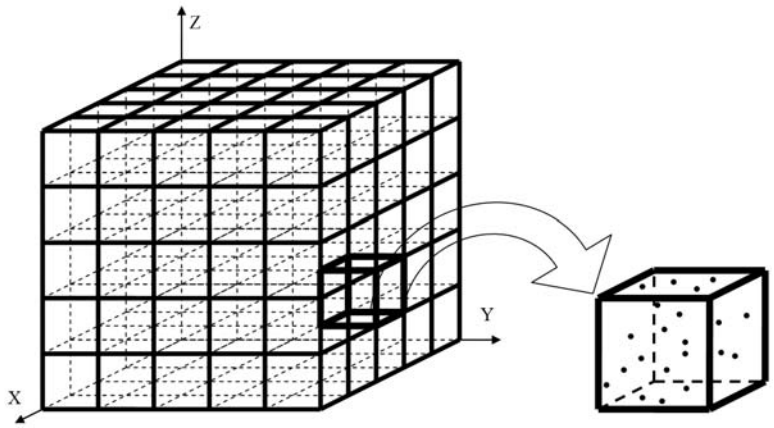


Figure 2 The whole SM assembled out of cells



Each charged particle is characterised by its mass, coordinates and velocities. Instead of solution of equations in the 6D space of coordinates and velocities, the dynamics of the system is determined by integrating the equations of motion of every particle in the series of discrete time steps. For solution of a problem in astrophysics the data might consume practically unlimited resources.

2.3 Basic theoretical algorithm of modelling

At each time step $t_{k+1} = t_k + \Delta t$ the following is done:

1 for each particle, the Lorentz force is calculated from the values of electromagnetic fields at the nearest mesh points (*gathering phase*)

2 for each particle the new coordinates and velocity of a particle are calculated; a particle can move from one cell to another (*moving phase*)

3 for each particle the charge carried by a particle to the new cell vertices is calculated to obtain the current charge and density, which are also discretised upon the rectangular mesh (*scattering phase*).

Maxwell's equations are solved to update the electromagnetic field (*mesh phase*).

This theoretical algorithm cannot be efficiently implemented in parallel on multicomputers (see Section 2.5). In order to provide good quality of parallel PIC implementation another technological algorithms[1] should be developed.

2.4 Two basic decompositions

The sizes of a time step and of a cell are chosen in such a manner that a particle cannot migrate farther than into the adjacent cell at one time step of modelling. The number of time steps depends on the conditions of a physical experiment. A more detailed description of the PIC method can be found in Hockney and Eastwood (1981), Berezin and Vshivkov (1980) and Kireev et al. (2007).

The PIC algorithm has the great possibility for the parallelisation, because all the particles are moved independently. The volume of computations at the first three phases of each time step is proportional to the number of particles. About 90% of multicomputer resources are spent for the particle processing. Thus, in order to implement the PIC code on multicomputers with high performance, the equal number of particles should be assigned for processing to each processor element (PE). However, on the MIMD distributed memory multicomputers, the performance characteristics of the PIC code crucially depends on how the mesh and particles are distributed among the PEs. In order to decrease the communication overheads at the first and the third phases of a time step, it is required that a PE contains as the cells (values of the electromagnetic fields at the mesh points) as the particles located inside them. Unfortunately, in the course of modelling, some particles might migrate from one cell to another. To satisfy the previous requirement, two basic decompositions can be used (Kraeva and Malyshkin, 2001).

In the so-called Lagrangian decomposition, the equal numbers of particles are assigned to each PE with no regard for their position in the *SM*. In this case, the values of the electromagnetic fields, the current charge and density at all the mesh points should be copied in every PE. Otherwise, the communication overheads at the first and the third phases will decrease the effectiveness of parallelisation. Disadvantages of the Lagrangian decomposition are the following:

- strict memory requirements
- communication overheads at the second phase (to update the current charge and the current density in each PE).

In the Eulerian decomposition, each PE contains a fixed rectangular sub-domain of the SM, for example parallelepipeds, including electromagnetic fields at the corresponding mesh points and particles in the corresponding cells. If a particle leaves its sub-domain and migrate to another sub-domain in the course of modelling, then this particle should be transferred to the PE containing this latter sub-domain. Thus, even with an equal initial workload of the PEs, in several steps of modelling, some PEs might contain more particles than the others. This results in the load imbalance. The character of the particles motion does not fully depend on equations, but also on the initial particles distribution, the value of their velocities, mass and value of the electromagnetic fields.

2.5 Typical phenomena

Let us consider some examples of particles distribution, which correspond to the different real physical experiments.

- *Uniform distribution* of the particles in the entire SM.
- *The case of a plate.* The SM has $n1 \times n2 \times n3$ size. The particles are uniformly distributed in $k \times n2 \times n3$ size space ($k << n1$).
- *Flow*. The set of particles is divided into two subsets: the particles with zero initial velocity and the active particles with the initially non-zero velocity. Active particles are organised as a flow crossing the space along a certain direction.
- *Explosion*. There are two subsets of particles. The background particles with zero initial velocities are uniformly distributed in the entire SM. All the active particles form a symmetric cloud ($r << h$, where r is the radius of the cloud, h is the mesh step). The velocities of active particles are directed along the radius of the cloud. In this case initially most of the particles are concentrated inside a cell and as a rule, cannot be located in one PE. Therefore, basic theoretical algorithm cannot be directly used for modelling such phenomenon.

The main problem of programming is that the data distribution among the PEs depends not only on the volume of data, but also on the data properties (particle velocities, configuration of electromagnetic field, etc.). With the same volume of data but different particles distributions inside the space the data processing is organised in different ways.

As the particles distribution is not stable in the course of modelling, the program control and data distribution among the PEs should be dynamically changing. It is clear that the parallel implementation of the PIC method on a distributed memory multicomputer strongly demands the dynamic load balancing.

3 Parallelisation of PIC

The line, the 2D and 3D grid structures of interprocessor communications of a multicomputer are very suitable for the effective parallel implementation of PIC and generally numerical algorithms on the rectangular meshes.

A cell is natural *atomic fragment*[2] of computation for PIC parallel implementation. It contains both data (particles inside the cells and values of electromagnetic fields, current density at their mesh points) and the procedures, which operate with these data. For the PIC method, when a particle moves from one cell to another, it should be removed from the former cell and added to the latter cell. Thus, we can say that the Eulerian decomposition is implemented.

3.1 PIC parallelisation on the line of PEs

Let us consider first in what way the PIC is parallelised for the multicomputers with the line structure of interprocessor communications. The three-dimensional modelling domain is initially partitioned into N blocks (where N is the number of PEs). Each block_i consists of several adjacent layers of cells and contains approximately the same number of particles (Figure 3). When the load imbalance is crucial, some border layers of the block located into an overloaded PE migrate to another less loaded neighbour PE. In the course of modelling, the adjacent blocks are located in the linked PEs. Therefore, the adjacent layers are located in the same or in the linked PEs. It is important for the second phase, at which some particles can migrate from one cell into another and for the fourth phase, when for recalculation of values of electromagnetic fields in a certain cell, values in the adjacent cells are also used. All this means that a layer should be not dividable data fragment and implemented in a program as separate process (P_fragment) able to migrate. Processing of every block_i consists of processing of all its layers.

3.2 PIC parallelisation on the 2D grid of PEs

Let us consider now the PIC method parallelisation for the 2D grid of PEs. Let the number of PEs be equal to $l \times m$, NP is the number of particles. Then SM is divided into l blocks orthogonal to a certain axis. Each block consists of several adjacent layers and contains about NP/l particles (in the same way as it was done for the line of PEs). The block_i is assigned for processing to the i-th row of the 2D grid of PEs (Figure 4). Blocks are formed in order to provide an equal total workload of every PE's row of the processor grid. Then every block_i is divided into m sub-blocks *block_i_j*, which are distributed for processing among m PEs of the row. These sub-blocks are composed in such a way in order to provide an equal workload of every PE of the row. If overload of at least one PE occurs in the course of modelling, this PE is able to recognise it at the moment when the number of particles substantially exceeds $NP/(l \times m)$, i.e., the permitted threshold of the imbalance. Then this PE initiates the balancing procedure.

Figure 3 Decomposition of *SM* for implementation of the PIC on the line of PEs

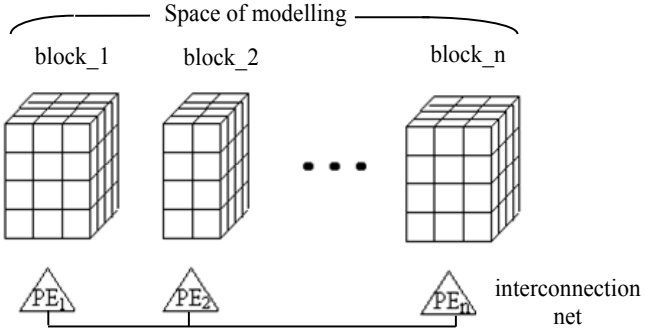
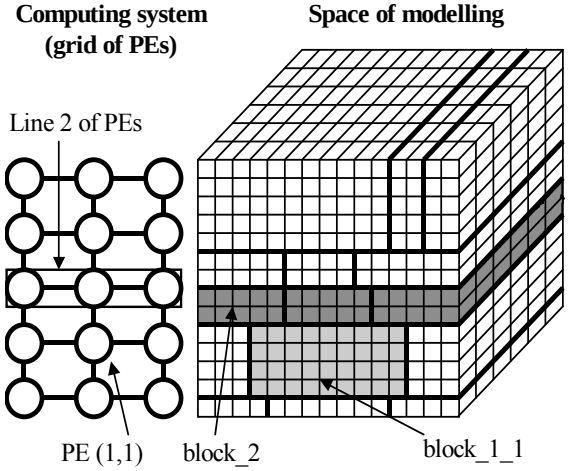


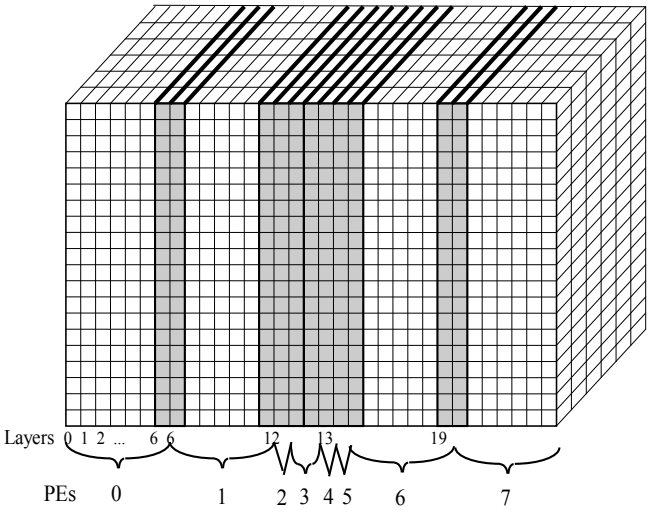
Figure 4 Decomposition of *SM* for implementation of PIC on the 2d grid of PEs



If the number of layers $k \times N$ (or $k \times l$ in the case of the grid of PEs), it is difficult or even impossible to divide the SM into blocks with the equal number of particles. Also, if particles are concentrated inside a single cell, it is definitely impossible to divide SM into the equal sub-domains. In order to attain the better load balance, the following modified domain decomposition is used. A layer containing more than the average number of particles is copied into two or more neighbouring PEs (Figure 5) – these are *virtual layers*. A set of particles located inside such a layer is distributed among all these PEs. In the course of the load balancing, particles inside the virtual layers are the first to be redistributed among PEs, and only if necessary, the

layers are also redistributed. For the computations inside a PE, there is no difference between virtual and non-virtual layers. Any layer could become virtual in the course of modelling; a virtual layer can stop to be virtual.

Figure 5 Virtual layers for implementation of PIC on the 2d grid of PEs



Any data distribution and resources allocation should keep the neighbourhood relation on the set of minimal fragments (adjacent fragments should be assigned for processing in the same PE or in neighbouring PEs, i.e., connected physical link).

3.3 General analysis

We can see, that there is no a necessity to provide a cell migration. A cell is too small fragment; the set of cells of SM is too big. Therefore, many resources should be spent to provide cell's migration. Thus, there is a necessity to use bigger indivisible fragments on the step of execution (not on the step of a problem/program description!). In the case of the line of PEs a layer of the *SM* should be chosen as indivisible fragment of a certain parallel implementation of PIC. Such indivisible fragment is called a *minimal* fragment. For PIC implementation on the grid of PEs a column can be taken as minimal indivisible fragment. Procedure for minimal fragment implementation is composed statically out of P_fragments, before the whole program assembling. This essentially improves the performance of an executable code.

In such a way, a cell is used as atomic fragment at the step of numerical algorithm description. At the step of execution of a numerical algorithm, different minimal fragments should be constructed.

Division of an SM into layers and columns is not desirable because their size is not restricted at least in one dimension. As result such a layer or column cannot be located in the memory of any multicomputer node. Universal parallelisation can be constructed dividing SM into the blocks of constant size, for instant, into parallelepipeds (indivisible fragments). In the case of imbalance, a parallelepiped migrates from overloaded into neighbouring underloaded PE, equalising the workload of PEs.

3.4 Problems of PIC parallel implementation

The main problems of numerical algorithms parallel implementation on exa-flops multicomputers can now be clearly formulated.

1 Exa-flops multicomputers might consist of several hundred thousand and even millions of processing elements (PE). In a large-scale model, the data distribution is changing in the course of simulation. Therefore, a program should not contain the numbers of concrete resources; mostly it should not contain fixed decisions on the resources allocation. Generally, all the resources, used for program execution, should be dynamically assigned and re-assigned.

2 At any moment of simulation, data distribution and resources allocation, including initial resources distribution, should keep the neighbourhood relation on a set of minimal fragments (the adjacent fragments should be assigned for processing in the same PE or in the neighbouring PEs, i.e., connected by a physical link).

3 All the communications, excluding, possibly a few individual cases, should be carried out between the neighbouring processes and PEs.

4 An application parallel program of the large-scale numerical modelling should not contain any central control. All the control and resources allocation decisions should be made in the course of simulation locally in any PE with the use of the data, located in this PE and/or in the neighbouring PEs.

5 The workload imbalance that might occur in the course of simulation should be equalised by the dynamic load balancing procedure, which is executed in every PE. The decision on workload balancing is made locally in overloaded/underloaded PE. The workload balancing involves overloaded/underloaded PE and its neighbouring PEs.

6 The algorithms of resources allocation and re-allocation should follow the behaviour of a numerical model in the course of simulation. For example (see Figures 4 and 5), if the particles migrate to the right, then the minimal fragments should migrate to the left decreasing in advance the workload of those PEs, that should accept at next time step of simulation the migrated minimal fragments.

To a greater or to a lesser extent, these problems are intrinsic of numerical models and should be overcome in the process of the development of parallel programs of numerical modelling. As a result, the development of parallel programs implementing a numerical algorithm for an exa-flops multicomputer has high complexity comparable with the complexity of system functions

development. Therefore, their development should be supported by a proper system of parallel programming (Malyshkin, 2007, 2010; Kireev and Malyshkin, 2011; Malyshkin and Perepelkin, 2012; Kaleet et al., 2008; Charm++; Shu and Kale, 1991). For large-scale numerical models parallel implementation including PIC applications, here the use of technology of fragmented programming is suggested, that is supported by the LuNA language and programming system. The ways how to attain high performance of simulation on exa-flops multicomputers are suggested and discussed.

4 Programming large-scale numerical models

Currently, several parallel programming systems support programming the large-scale numerical models for multicomputers (Kireev and Malyshkin, 2011; Malyshkin and Perepelkin, 2012; Kaleet et al., 2008; Charm++; Shu and Kale, 1991). The idea of supporting large-scale models programming for exa-flops multicomputers is demonstrated by the LuNA fragmented programming system (Kireev and Malyshkin, 2011; Malyshkin and Perepelkin, 2012), which is now under development in the National University of Novosibirsk.

4.1 The main idea of the fragmented programming technology

The idea of the fragmented technology of parallel programming can be trivially formulated:

1 Fragmented programming technology supports assembling the whole program out of fragments of computations (FC) and data fragments and its parallel execution. Therefore, for solving an application problem its algorithm as well data should be initially fragmented similar to fragmentation of the above PIC algorithms.

2 Each FC is an independent unit of a program, it contains descriptions of input/output variables (data fragments) and code (module, procedure) (Kraeva and Malyshkin, 1997).

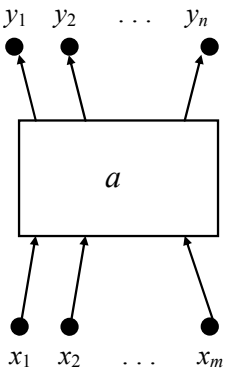
3 A fragmented program (FP) is a recursively countable sets of data fragments and FCs.

4 As opposed to module programming, the fragmented structure of an FP is kept in the course of computation. In the course of computation each executed FC independently generates an executed process interacting with the other processes. This provides the possibility of implementing dynamic properties of an FP automatically (its dynamic tune in all the available resource, dynamic load balancing).

5 Following Kleene (1962) and Rogers (1967), the basic algorithm representation in LuNA is a recursively computable set of functional terms.

The fragmented programming technology is based on the theory of parallel program synthesis (Valkovskii and Malyshkin, 1988). The LuNA programming system is designed the system of parallel programs construction.

Figure 6 A fragment of computation



Omitting many theoretical and technological details of the description of the fragmented programming technology in general and LuNA implementation in particular, let us consider a couple of simple examples: matrices multiplication and PIC method fragmented implementation.

4.1.1 Matrices multiplication fragmented algorithm

Initial algorithm:

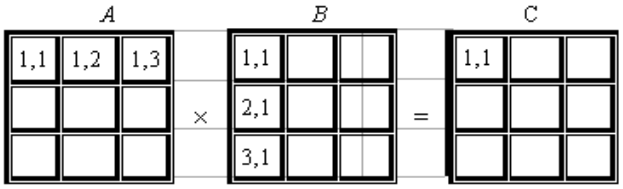
$$C = A \times B,\ A = \left(a_{ij}\right)_{i,j=\overline{1,N}},\ B = \left(b_{ij}\right)_{i,j=\overline{1,N}},\ C = \left(c_{ij}\right)_{i,j=\overline{1,N}}$$

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} b_{k,j}, \quad i, j, k = \overline{1, N}$$

Algorithm fragmentation: At the first step, the initial algorithm should be fragmented, i.e., divided into data and computation fragments. Most of numerical algorithms have a regular structure of data and computations, which can be divided into the parts. Division of the algorithm's data into parts is called the data fragmentation. For example, if an initial algorithm data structure is an N-dimensional array, then it can be fragmented into the set of N-dimensional sub-arrays. So, a fragmented algorithm will process the N-dimensional array of sub-arrays, whose entries are called data fragments.

In the fragmented matrices multiplication algorithm, the square matrices A, B and C of $N \times N$ size are divided into sub-matrices of $M \times M$ size, which are data fragments (Figure 7), i.e., data are aggregated. For the sub-matrices multiplication operation (fragment of computation) the same code is applied as for the multiplication of the initial matrices.

Figure 7 Fragmentation of the matrices multiplication algorithm



Thus, a set of data fragments is as follows:

$$DF = \left\{A_{i,j} \,|i,j = \overline{1,K}\right\} \cup \left\{B_{i,j} \,|i,j = \overline{1,K}\right\} \cup \left\{C_{i,j} \,|i,j = \overline{1,K}\right\} \cup \left\{D^k_{i,j} \,|i,j,k = \overline{1,K}\right\},$$

where $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, $D^k_{i,j}$ are matrices.

Here $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, $D^k_{i,j}$ are the names of data fragments. $A_{i,j}$, in particular, denotes a sub-matrix in the i^{th} row and the j^{th} column of $K \times K$ array of data fragments. In Figure 7, the matrix A fragmentation for $K = 4$ is presented. The output data fragments $C_{i,j}$ are computed by the following fragmented algorithm:

1 $D^k_{i,j} = A_{i,k} B_{k,j}$

2 $C_{i,j} = \sum_{k=1}^{K} D^k_{i,j}$.

The product $D^k_{i,j} = A_{i,k} B_{k,j}$ is a partial sum of $C_{i,j}$. The data fragments have the same structure as the initial data structure. The sub-matrices multiplication can be implemented by a ready-made sequential procedure, for example, the BLAS subroutine SGEMM. Thus, the matrices multiplication algorithm is defined by the set of functional terms, depicted in Figure 8. The operations of the functional terms are the fragments of computation.

Sometimes algorithms fragmentation is very difficult work. For example, the work on sweep method fragmentation (Terekhov, 2010) was lasting more than two years.

In order to compute the matrix C, these sub-matrices $C_{i,j}$ (data fragments) should be properly multiplied with the use of FC, as it is defined by a set of functional terms (Figure 8).

4.1.2 The PIC method parallel implementation

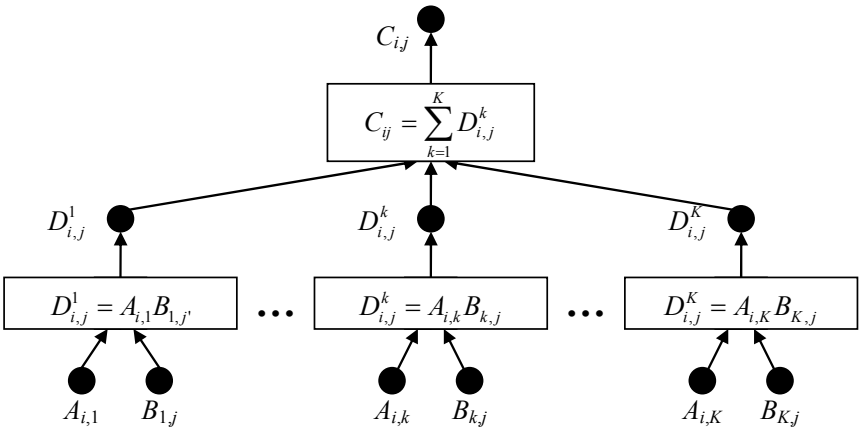
In Section 3 we have discussed the PIC algorithms and data fragmentation. The application of the PIC to the astrophysics problem of dust cloud simulation and the idea of automatic load balancing are shown in Kireev (2009). Processing all the cells of a layer, a column and a parallelepiped define the FCs.

A simplest way to execute an FP is to develop a run-time system that starts the FC's execution in any order not contradicting to the information dependences between FC. The basic algorithm of run-time system checks whether the values of all the input variables of a FC are available, then this operation can begin its execution. If several FC might begin their execution, then all the FCs can begin the execution or a part of them or none of them. The performance of fragmented algorithm execution dramatically depends on this choice.

4.2 How to attain high performance of PIC parallel implementation

LuNA applications are large-scale numerical models. Regular structure of mesh numerical algorithms essentially simplifies the algorithms of the LuNA run-time subsystem implementation. As result, the algorithms of LuNA run-time subsystem provide high enough performance execution of numerical algorithms and do not guarantee good execution of the algorithms from the different application areas. In order to reach high performance execution of a fragmented algorithm the proper algorithm of FP execution should be developed and also additional information on the fragmented algorithm should be prepared by a compiler or given by a programmer.

Figure 8 Functional terms defining the fragmented matrices multiplication algorithm



4.2.1 Relations

First relation is the neighbourhood data fragments relation which is defined on the set of data fragments and reflects the neighbour of data fragments in the whole data structure of a problem.

The second relation is the neighbourhood relations on the set of FCs. Neighbour FCs have information dependence. They should be mostly executed on the same or neighbour PEs (connected by physical links).

The third neighbourhood relation is defined on the set of PEs and reflects the communication structure of a multicomputer. In order to not schedule the resources allocation in details, the initial resources allocation is made by the algorithm that imitates the behaviour of a liquid in the system of communicating vessels. This algorithm can be executed in any PE locally (locality property), for example, equalising the workload of a PE and its neighbour PEs in the course of execution. Regularity of mesh numerical algorithms provides this possibility. This algorithm provides also addressing of resources in the terms of neighbourhood, not in physical addresses of the resources. Diffusive algorithms (Kraeva and Malyshkin, 1999; Hu and Blake, 1999; Corradi et al., 1997) also can be used.

The fourth relation is the relation on the set of FC that defines a control in the FP, i.e., the order of FCs execution.

All these relations are mostly kept in the course of FP execution.

4.2.2 Communications

The above algorithm of resources allocation provides that all the communications are made between FCs located in the same or in the neighbour PEs. Regularity of mesh numerical algorithms is very substantial for providing of this property.

4.2.3 Control

An application parallel program of the large-scale numerical modelling should not contain any central control. All the control and resources allocation decisions should be made in the course of simulation locally in any PE with the use of the data, located in this PE and/or in the neighbouring PEs. Data and computation fragmentation and the chosen algorithm of resources allocation provide this property.

4.2.4 The workload imbalance

The workload imbalance is also overcome by the local algorithm that imitates the behaviour of a liquid in the system of communicating vessels. In the case of explosion or similar phenomena modelling not serious modifications of the algorithm should be done (http://ssd.sscc.ru/en/dlb).

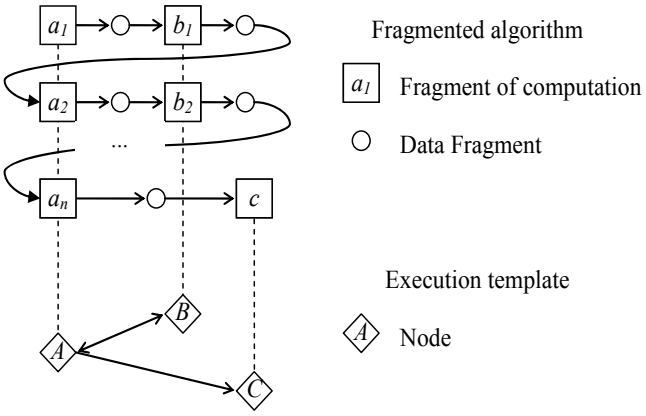
4.2.5 The priority

A programmer possesses of important information that can be used for improvement an FP execution, in particular, priority. Priority is a real-valued function, defined on the set of FCs. If there is ability for run-time subsystem to fetch for execution a number of different FCs, then the FCs with the highest priority are fetched. Definition of the priorities allows controlling the flow of FCs execution in order to optimise performance (see profiling subsection below). Execution of the FP according to the priorities is not mandatory. Depending on different factors of a certain situation, run-time subsystem can accept or reject the priority recommendations.

4.2.6 Execution template (ET)

To optimise the performance of the most time- and resource-consuming parts of the FP an ET can be defined. The ET is an oriented graph, ET = < N, E >, where N is the set of multicomputer nodes, and E is the set of oriented links, $E = \{< n_1, n_2 > \mid n_1, n_2 \in N\}$. The nodes N are execution units, connected with each other by links, which transfer values from one node to another. A number of FCs can be mapped to ET nodes. Consider an example (Figure 9). FCs a_i are mapped to node *A*, b_i- to node B and FC c is mapped to node C. Execution of the part of FP, mapped to the ET is organised by run-time system as follows. An FC, located in a node, is executed if its predecessors in relation E finished their execution without any additional checks. After FC execution is finished, the values of output variables of FC are transferred to the other nodes. In such a way, ET specifies the scheme of FCs execution, which reduces the run-time subsystem overhead of choosing FC for execution.

Figure 9 Example of executive template



4.2.7 Profiling

Profiling is a process of gathering a profile, which is the run-time information about an FP execution. This information includes real order of FCs execution, FCs execution times, PEs workload, etc. The profile is used by run-time subsystem in order to optimise next executions of the FP. For example, if in process of the FP execution some PEs were idle because the value of a data fragment x was not computed in time, than this information is extracted from the profile, the highest priority is assigned to FCs yielding the value of x and in process of the next FP

execution the FCs, which provide yielding the value of x will be assigned for execution earlier. In such a way, each next execution of the FP will be more efficient, if the FP is run on the same multicomputer and with the similar input data.

4.2.8 Behaviour of the model

The suggested in Section 4.2.1 local algorithm of resources allocation and re-allocation can also be used in order to prepare the workload balancing in advance in the case the workload is changing in the course of simulation depending on model input data.

Also this algorithm is used substantially in the case if heterogeneous GRID or Cloud is used for simulation.

5 Conclusions

The fragmentation of numerical model algorithms and programs is universal mechanism for large scale models parallel implementation. It provides automatically the high enough performance of the FP execution, high FP flexibility, dynamic tunability to all the available resources of a multicomputer, portability of an FP and all the other dynamic FP properties. Unified approach to automatic construction of parallel programs of numerical modelling substantially simplifies the design, implementation and exploitation a programming system.

Also this approach demands to re-design as application as well system algorithms in order FP execution could be organised locally. Re-design should provide the locality property of all the developed and used algorithms.

References

Berezin, Y.A. and Vshivkov, V.A. (1980) *The Method of Particles in Rarefied Plasma Dynamic*, in Russian, Nauka, Novosibirsk.

Charm++ [online] http://charm.cs.uiuc.edu (accessed 15 April 2013).

Corradi, A., Leonardi, L. and Zambonelli, F. (1997) 'Performance comparison of load balancing policies based on a diffusion scheme, Paper presented at the *Proc. of the Euro-Par '97*, LNCS Vol. 1300.

Hockney, R. and Eastwood, J. (1981) *Computer Simulation Using Particles*, McGraw-Hill, New York, USA.

http://ssd.sscc.ru/en/dlb.

Hu, Y.F. and Blake, R.J. (1999) 'An improved diffusion algorithm for dynamic load balancing', *Parallel Computing*, Vol. 25, No. 4, pp.417–444.

Kaleet, L.V. et al. (2008) 'Programming petascale applications with Charm++ and AMPI', *Petascale Computing: Algorithms and Applications*, pp.421–441, Chapman & Hall, CRC Press, USA.

Kireev, S. and Malyshkin, V. (2011) 'Fragmentation of numerical algorithms for parallel subroutines library', *The J. of Supercomputing*, Vol. 57, No. 2, pp.161–171, Springer.

Kireev, S.E. (2009) 'A parallel 3D code for simulation of self-gravitating gas-dust systems', Paper presented at the *PaCT-2009 Proceedings*, Springer, LNCS 5698, pp.406–413.

Kireev, S.E., Kuksheva, E.A., Snytnikov, A.V., Snytnikov, N.V., Vshivkov, V.A. (2007) 'Strategies for development of a parallel program for protoplanetary disc simulation', Paper presented at the *PaCT-2007 Proceedings*, Springer, LNCS 4671, pp.128–139.

Kleene, S.C. (1962) *Introduction to Metamathematics*, North-Holland Publishing Co., Amsterdam.

Kraeva, M. and Malyshkin, V. (1997) 'Implementation of PIC method on MIMD multicomputers with assembly technology', *Proc. of HPCN Europe, High Performance Computing and Networking, Int. Conference*, Springer Verlag, LNCS, Vol. 1255, pp.541–549.

Kraeva, M.A. and Malyshkin, V.E. (1999) 'Algorithms for dynamic load balancing in implementation of the particle-in-cell method on MIMD multicomputers', *Programming and Computer Software*, Vol. 25, No. 1, pp.39–45.

Kraeva, M.A. and Malyshkin, V.E. (2001) 'Assembly technology for parallel realization of numerical models on MIMD-multicomputers', *Int. J. on Future Generation Computer Systems*, Vol. 17, No. 6, pp.755–765, Elsevier Science.

Malyshkin, V. (2007) 'Fragmented programming of library parallel numerical subroutines', *Proceedings of the 7th Int. Conference on New Trends in Software Methodologies, Tools and Techniques*, 28–30 September, 2007, IOS Press, Dubai, Vol. 193, pp.425–430.

Malyshkin, V. (2010) 'Assembling of parallel programs for large scale numerical modeling', *Handbook of Research on Scalable Computing Technologies*, 1021p, Chapter 13, pp.295–311, ISBN 978-1-60566-661-7, IGI Global, USA.

Malyshkin, V. and Perepelkin, V. (2012) 'Optimization methods of parallel execution of numerical programs in the LuNA fragmented programming system', *The Journal of Supercomputing*, Vol. 61, No. 1, pp.235–248, Springer, Special issue on Enabling Technologies for Programming Extreme Scale Systems, DOI:10.1007/s11227-011-0649-6.

Rogers, H. (1967) *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York.

Shu, W. and Kale, L.V. (1991) 'Chare kernel – a runtime support system for parallel computations', *J. Parallel Distrib. Comput.*, Vol. 11, No. 3, pp.198–211.

Terekhov, A.V. (2010) 'Parallel dichotomy algorithm for solving tridiagonal system of linear equations with multiple right-hand sides', *Parallel Computing*, Elsevier, Vol. 36, No. 8, pp.423–438.

Valkovskii, V.A. and Malyshkin, V.E. (1988) Synthesis o*f Parallel Programs and Systems on the Basis of Computational Models*, in Russian, Nauka, Novosibirsk.

Walker, D.W. (1990) 'Characterising the parallel performance of a large-scale, particle-in-cell plasma simulation code', *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp.257–288.

Notes

1 Technological algorithm should have the polynomial complexity and possesses the necessary pragmatic properties.

2 Not dividable fragment.