ELSEVIER

Future Generation Computer Systems 17 (2001) 755–765

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.nl/locate/future

Assembly technology for parallel realization of numerical models on MIMD-multicomputers

M.A. Kraeva*, V.E. Malyshkin

Supercomputer Software Department, Institute of Computational Mathematics and Mathematical Geophysics (ICMMG), Russian Academy of Sciences, pr. Lavrentieva 6, 630090 Novosibirsk, Russia

Abstract

The main ideas of the assembly technology (AT) in its application to parallel realization of big size numerical models on a rectangular mesh are considered and demonstrated by the parallelization of the particle-in-cell (PIC) method. The realization of the numerical models with the assembly technology is based on the construction of a fragmented parallel program. This provides different useful properties of the target program including dynamic load balance on the basis of the fragments flying from overloaded to underloaded processor elements of a multicomputer.

Keywords: Assembly technology; Dynamic load balancing; Particle-in-cell (PIC) method

1. Introduction

Parallel realization of realistic numerical models, using direct modeling of a physical phenomenon on the basis of description of the phenomenon behavior in the local area, usually requires high performance computations. However these models are also remarkable for irregularity and even dynamically changing irregularity of the data structure (adoptive mesh, variable time step, particles, etc.). For example, in the particle-in-cell (PIC) method the test particles are in the bottom of such an irregularity. Hence, these models are very difficult for effective parallelization and high performance realization with conventional programming systems.

The assembly technology (AT) [1,2] has been especially created in order to support the development of fragmented parallel programs for multicomputers. Fragmentation and dynamic load balancing are the key features of programming and program execution under the AT. In order to provide high performance of target programs the AT is implemented in different ways for different application areas. In this paper, the application of the AT to realization of big size numerical models is demonstrated on the example of parallel realization of the PIC method [3].

2. The PIC method and the problems of its parallel realization

The particle simulation is a powerful tool for modeling of the behavior of complex non-linear phenomena in plasmas and fluids. In the PIC method, 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 describing a physical phenomenon under study [4,5].

* Corresponding author.
E-mail addresses: kraeva@ssd.sscc.ru (M.A. Kraeva), malysh@ssd.sscc.ru (V.E. Malyshkin).

0167-739X/01/$ – see front matter
PII: S0167-739X(00)00058-3

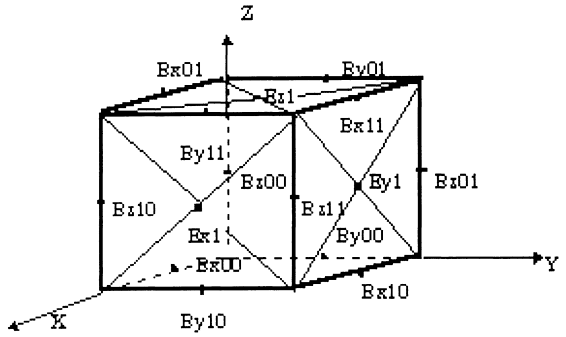

Fig. 1. A cell of the *SM* with the electric E and magnetic B fields, discretized upon shifted meshes.

A real physical space is represented by a model of simulation domain called the space of modeling (SM). The electric *E* and magnetic *B* fields are defined as vectors and discretized upon rectangular mesh (or several shifted meshes, as shown in Figs. 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 modeling a particle belongs to a certain cell of each mesh.

Each charged particle is characterized by its mass, co-ordinates and velocities. Instead of solution of equations in the 6D space of co-ordinates 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. 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 co-ordinates 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 discretized upon the rectangular mesh (*scattering phase*).
4. Maxwell's equations are solved to update the electromagnetic field (*mesh phase*).

The sizes of a time step and of a cell are chosen in such a manner that a particle cannot fly farther than into the adjacent cell at one time step of modeling. The number of time steps depends on a physical experiment. A more detailed description of the PIC method can be found in [4,5].

The PIC algorithm has the great possibility for the parallelization, because all the particles 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 multicomputer with high performance, 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

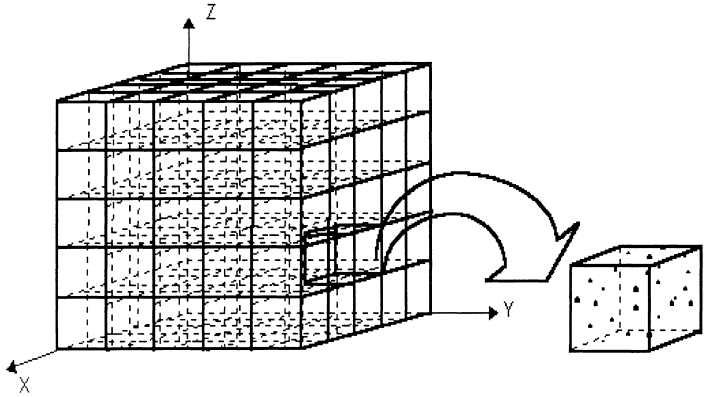

Fig. 2. The whole space of modeling (*SM*) assembled out of cells.

to decrease the communication overhead 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) the particles located inside them. Unfortunately, in the course of modeling, some particles might fly from one cell to another. To satisfy the previous requirement, two basic decompositions can be used [6].

In the so-called Lagrangian decomposition, equal number 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 overhead at the first and the third phases will decrease the effectiveness of parallelization. Disadvantages of the Lagrangian decomposition are the following:

- strict memory requirements;
- communication overhead 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, including electromagnetic fields at the corresponding mesh points and particles in the corresponding cells. If a particle leaves its sub-domain and flies to another sub-domain in the course of modeling, 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 modeling, some PEs might contain more particles than the others. This results in the load imbalance. The character of the particles motion does not only fully depend on equations, but also on the initial particles distribution and the initial value of the electromagnetic field. Another disadvantage of the Eulerian decomposition is that particles displacement on every time step results in frequent memory reallocation.

Many researchers studied parallel implementation of the PIC method on different multicomputers. Several methods of the PIC parallelization were developed. The big list of references to articles devoted to PIC parallelization could be found in [6,7]. In order to reach high performance these methods take into account the particles distribution and/or depend on the architectures and resources.

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 $n_1 \times n_2 \times n_3$ size. The particles are uniformly distributed in $k \times n_2 \times n_3$ size space ($k \ll n_1$).
- *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 organized 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 ($r \ll h$, where r is the radius of the cloud, and h is the mesh step). The velocities of active particles are directed along the radius of the cloud.

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 particle distributions inside the space the data processing is organized in different ways.

As the particles distribution is not stable in the course of modeling, 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.

In the papers [8,9] the dynamic load balancing for particle methods is discussed. The package described in [9] runs only on the distributed memory system of p processors, where p is the power of 2.

3. Substantial features of AT

Numerical methods, generally, and the PIC method, in particular, are very suitable for the application of AT. The fragmented representation of an algorithm affects different stages of an application program development. Here, we would like to briefly list some peculiarities of the AT.

3.1. Algorithm and program fragmentation

In accordance with the AT, the description of an application problem should be divided into a system of reasonably small atomic fragments, representing the realization entities of a problem description. Fragments might be represented in programming languages by variables, procedures, subroutines, macros, nets, notions, functions, etc., depending on the method of the AT realization. The program, which realizes an atomic fragment (P_fragment), contains both data and code. In other words, a program, realizing an application problem, is assembled out of such small P_fragments of computations, which are connected through variables for data transfer.

The fragmented structure of an application parallel program is kept in the executable code and provides the possibility for organization of flexible and high performance execution of the assembled program. The general idea is the following. An assembled program is represented in a programming language, and is composed as a system of executable P_fragments. This program is executed inside every PE, looping over all the P_fragments, loaded to the PE. If these fragments are small enough, then initially for each PE of a multicomputer the equal workload is assembled out of these P_fragments. This is of course a general idea only. Actually, in order to attain high performance of a target program, its assembling is realized in different ways.

The workload of PEs can be changed in the course of computation, and if at least one PE becomes overloaded, then a part of P_fragments (with the processing data), which were assigned for execution into the overloaded PE, should fly to the underloaded PEs equalizing the workload of multicomputer PEs. Dynamic load balancing of a multicomputer is based on such a fragmentation [2,3]. The algorithms of a target program assembling out of atomic fragments are not discussed here in detail.

3.2. Assembling vs. partitioning

Our basic keyword is *assembly*. Contrary to partitioning, the AT supports explicit assembling of a whole program out of ready-made fragments of computations, rather than dividing a problem, defined as a whole, into the suitable fragments to be executed on the different PEs of a multicomputer. These fragments are the elementary blocks, the “bricks”, to construct a whole program (a whole problem description, there is no difference here). An algorithm of a problem assembling is kept and used later for dynamic program/problem parallelization. Assembling defines the “seams” of a whole computation, the way of the fragments connection. Therefore, these seams are the most suitable way to cut the entire computation for parallel execution. For this reason program parallelization can be always done if the appropriate size of atomic fragments is chosen.

3.3. Explicitly two-level programming

Under the AT, first, suitable atomic fragments of computation are designed, programmed and debugged separately. Then the whole computation (problem solution) is assembled out of these fragments.

3.4. Separation of the fine grain and the coarse grain computations

The fine and the coarse grain computations are executed on different levels of hardware and software. Under AT the fine grain computations are encapsulated inside a module (P_fragment) that realizes the computations bound up with atomic fragment. Such a module can be implemented very effectively on a PE. The whole parallel program is assembled out of these ready-made modules. The system of modules of a program defines a system of interacting processes (coarse grain computations). Encapsulation of the fine grain computations, their complexity inside an atomic fragment provides the possibility of formalization of a parallel program construction and the use of explicit two-level representation of an algorithm: the programming level inside an atomic fragment and the scheme level of an application program assembling.

3.5. Separation of semantics and scheme of computation

The fine grain computations define a sufficient part of semantics (functions) of computations. They are realized within an atomic fragment. Therefore, on the coarse grain level, only a scheme (non-interpreted or semi-interpreted) of computations is assembled. It

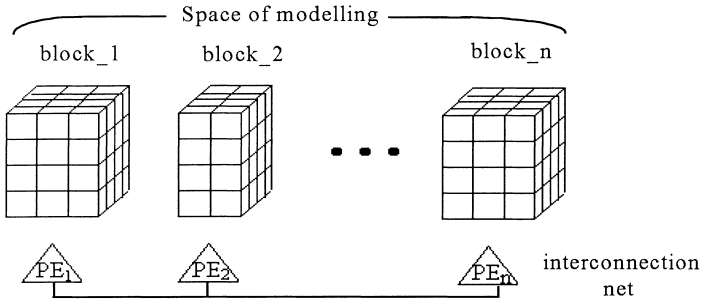

Fig. 3. Decomposition of *SM* for implementation of the PIC on the line of PEs.

means that formal methods of automatic synthesis of parallel programs [1] can be successfully used and the standard schemes of parallel computations can be accumulated in libraries.

4. Parallelization of numerical methods with AT

Let us consider the assembly approach to the numerical methods parallelization on the example of the PIC method parallel implementation.

The line and the 2D grid structure of interprocessor communications of a multicomputer are sufficient for the effective parallel realization of numerical methods on the rectangular meshes. In [3], the algorithm of the 2D grid mapping into the hypercube keeping the neighborhood of PEs is given.

A cell is natural atomic fragment of computation for a numerical method realization. It contains both data (particles inside the cells of a fragment 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 with the AT the Eulerian decomposition is realized.

4.1. PIC parallelization on the line of PEs

Let us consider first in what way the PIC is parallelized for the multicomputers with the line structure of interprocessor communications. The three-dimensional simulation 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 (Figs. 3 and 4). When the load imbalance is crucial, some layers of the block located into an overloaded PE are transferred to another less loaded PE. In the course of modeling, 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 fly 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.

4.2. PIC parallelization on the 2D grid of PEs

Let us consider now the PIC method parallelization for the 2D grid of PEs. Let the number of PEs be equal

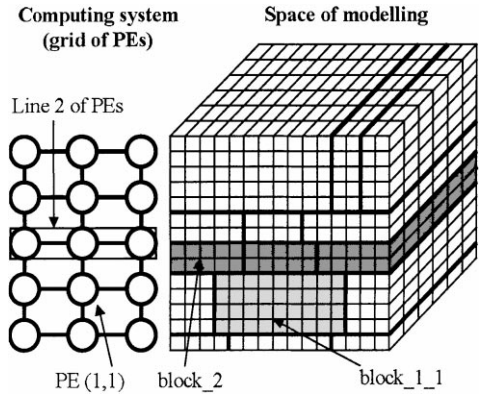

Fig. 4. Decomposition of *SM* for implementation of PIC on the 2D grid of PEs.

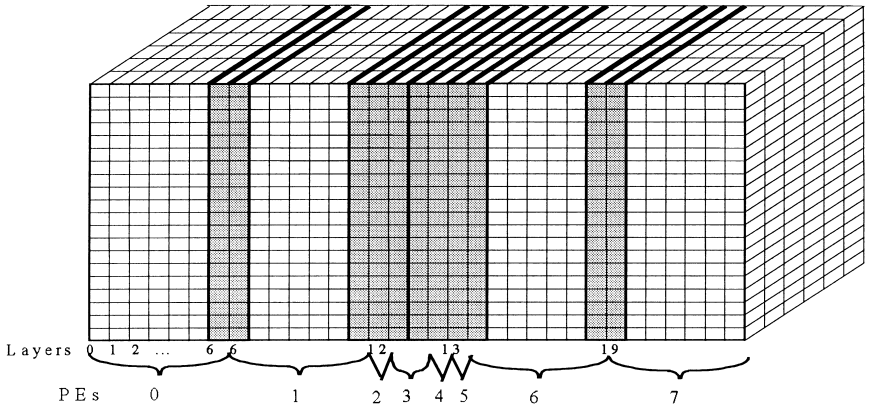

Fig. 5. Virtual layers for implementation of PIC on the 2D grid of PEs.

to $l \times m$. 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 ith row of the 2D grid (Fig. 4). Blocks are formed in order to provide an equal total workload of every PEs 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 modeling, this PE is able to recognize it at the moment when the number of particles substantially exceeds NP/($l \times m$). Then this PE initiates the re-balancing procedure.

4.3. Virtual layers

If the number of layers $k \approx N$ (or $k \approx l$ in the case of the grid of PEs), it is difficult or even impossible to divide the SM into blocks with 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 at least into 2 or more neighboring PEs (Fig. 5) — these are *virtual layers*. A set of particles located inside such a layer is distributed among all these PEs. In the course of 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 layer after redistribution. Conversely, virtual layer could stop being virtual. Virtual layers are always set up between adjacent PEs whether or not there are particles in those virtual layers.

4.4. Minimal fragments

We can see that in both cases (for the line and for the 2D grid of PEs) there is no necessity to provide flying of a cell. A cell is a very small fragment, therefore large resources should be spent to provide its flying. Thus, there is a necessity to use bigger indivisible fragments on the step of execution (not on the step of a problem/program assembling!). In the case of the line of PEs a layer of the SM should be chosen as indivisible fragment of concrete realization of PIC. Such a fragment is called a *minimal* fragment. For PIC implementation on the grid of PEs a column is taken as minimal indivisible fragment. Procedure realizing minimal fragment 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

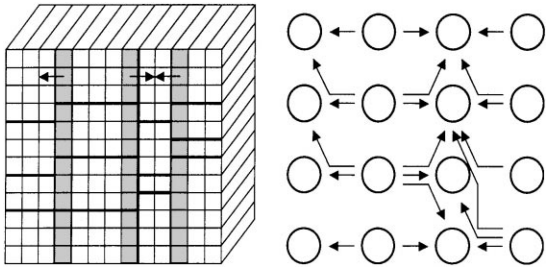

Fig. 6. Direction of data transfer for implementation of the PIC on the 2D grid.

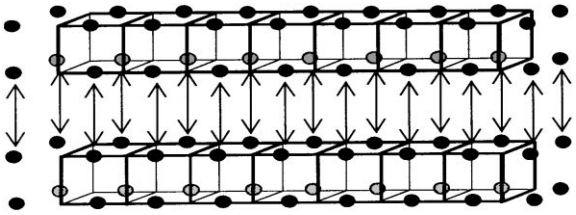

Fig. 7. Unification of the Hx mesh variables for two minimal fragments (layers).

minimal fragments assembled out of atomic fragments are chosen depending on the architecture of a multicomputer.

4.5. Realization of the PIC method on multicomputers

Using AT PIC method has been implemented on different multicomputer systems (MBC-100, multitransputer system, Hypercube Intel iPSC/860, Parsytec PowerXplorer, Cray J90, IBM SP2). In order to provide a good portability, the language C was chosen for the parallel PIC code implementation. Since different message passing systems are used for different multicomputers, the special communication functions were designed. In the compilation time the actual functions of a target multicomputer are substituted instead of these communication functions. In particular, the MPI functions might be substituted. For the dynamic load balancing of PIC several algorithms were developed [10,11]. In the case of the grid communication structure (Fig. 6) and the virtual fragments (particles might fly not only to the neighboring PEs) special tracing functions are used.

According to the AT the array of particles is divided into N parts, where N is the number of minimal fragments (layers of SM in the case of the line of PEs, and columns in the case of the 2D grid). When elements of the mesh variables (electromagnetic fields, current charge and density) of different minimal fragments hit upon the same point of SM, they are unified (Fig. 7). The elements of the mesh variables of one minimal fragment are not added to the data structure for this fragment, but are stored in the 3D arrays with elements of the mesh variables of other minimal fragments in the PE. This appears possible due to the rectangular shape of blocks *block_i* (*block_i_j* in the case of the 2D grid). Such a decision allows us to decrease the memory requirements and to speed-up computations during the forth phase of the PIC algorithm.

In the case of dynamic load balancing, when some minimal fragments are transferred from one PE to another, the size of the 3D arrays of elements of the mesh variables is dynamically changed. This demands special realization of such dynamic arrays.

5. Dynamic load balancing

To attain a high performance of parallel PIC implementation, a number of centralized and decentralized load-balancing algorithms were especially developed.

5.1. Initial load balancing

A layer of cells is chosen as minimal fragment for the PIC implementation on the line of PEs. Each minimal fragment has its own weight. The weight of a minimal fragment is equal to the number of particles in this fragment. The sum of weights of all the minimal fragments in some PE determines the *workload of PE*. The 3D simulation domain is initially partitioned with a certain algorithm into N blocks (where N is the number of PEs). Each block consists of several adjacent minimal fragments and contains approximately the average number of particles. For the initial load balancing two heuristic centralized algorithms were designed [10,11]. These algorithms employ the information about the weights of all the minimal fragments. Each PE has this information and constructs the workload card by the same algorithm. The workload card contains the list of minimal fragments that should be

loaded to the PEs. If the number of minimal fragments is much greater than the number of PEs, it is usually possible to distribute the minimal fragments among the PEs in such a way that every block would contain approximately the same number of particles.

If considerable portion of particles is concentrated inside a single cell, it is impossible to divide the SM into the blocks with quite an equal workload. To solve this problem, a notion of virtual layer is introduced. The centralized algorithm was modified for the case of virtual fragments.

If overloading of at least one PE occurs in the course of modeling, this PE is able to recognize it at the moment when the number of particles substantially exceeds NP/*N*. Then this PE initiates the re-balancing procedure.

5.2. Dynamic load balancing

If a load imbalance occurs, the procedure BALANCE is called. In this procedure, the decision about the direction of data transfer and the volume of data to be transferred is taken. For any load-balancing algorithm there exists its own realizing procedure BALANCE. The procedure TRANSFER is used for the data transfer. There are two realizations of this procedure: for the line of PEs and for the grid of PEs. The procedure TRANSFER is the same for any load-balancing algorithm on the line of PEs. Parameters of the procedure are the number of particles to be exchanged and the direction of data transfer. Such separation of decision making and data transfer itself allows to add new dynamic load-balancing algorithms easily.

Let us consider algorithms of the dynamic load balancing of the PIC. All the PEs are numerated. In the case of the line of PE, each PE has number i, where $0 \le i < number\_of\_PEs$. In the case of the $l \times m$ grid of PEs, the number of PE is the pair (i,j), where $0 \le i < l$, $0 \le j < m$. In the same way, layers and columns of the SM are numerated.

5.3. Centralized dynamic load-balancing algorithm

For the dynamic load-balancing the initial load-balancing algorithm could be used. One of the PEs collects the information about weights of the minimal fragments and broadcasts this information to all the other PEs. All the PEs build the new load card. After that neighboring PEs exchange minimal fragments according to the information in the new workload card.

Imbalance threshold. If centralized algorithms are used for the dynamic load balancing, the PEs exchange the information about load balancing, therefore every PE has information about the number of particles in all the PEs. In any PE, the difference mnp−NP/*N* (where mnp is the maximum number of particles in PE, NP is the total number of particles, *N* is the number of PEs) is calculated. If the difference is more than the threshold Th, the procedure BALANCE is called. The threshold can be a constant, chosen in advance, or an adaptive number. In the latter case, initially Th=0. In the course of modeling, the time *t_part*, which is required to implement steps (1)–(3) of the PIC algorithm for one particle, is calculated. After every BALANCE call the time of balancing t_bal is calculated. Th is assigned to be equal to t_bal/t_part (how many particles could be processed for the same time as one balancing requires). After each subsequent step of the PIC algorithm, Th is decreased by mnp−NP/*N*. When the value of Th is negative, BALANCE is called. If the threshold is always equal to zero, the procedure BALANCE is called after each time step of modeling.

5.4. Decentralized dynamic load-balancing algorithm for the constant number of particles

The use of centralized algorithms is good enough for multicomputers containing a few PEs. But if the number of PEs is big enough, the communication overhead could neutralize the advantages of the dynamic load balancing. In this case it is more preferable to use decentralized algorithms. Such algorithms use information about the load balance in the local domain of PEs only.

If the number of model particles is not changed in the course of modeling, a simple decentralized algorithm could be suggested. Each PE has the information on how many particles were moved to/from its neighboring PEs. To equalize the load balance it is sufficient just to receive/send the same number of particles from/to the neighboring PEs. It should be noted that this algorithm works in the case of virtual fragments only.

5.5. Specialized decentralized algorithm

To equalize the load balance for the PIC implementation, the following specialized algorithm was designed. In the course of modeling in every PE the main direction of particles motion is calculated from the values of particle velocities. In the load balancing, particles are delivered to the direction opposite to the main direction. As in the previous algorithm it is assumed that the number of particles is not changed in the course of modeling. The number of particles to be transferred from overloaded PEs to their neighbors in the direction opposite to the main direction is calculated from the average number of particles and the number of particles in the PE. Some particles are transferred in advance in order to reduce the number of calls of the dynamic load-balancing procedure.

5.6. Diffusive load-balancing algorithms

The *basic diffusive* load-balancing algorithm was implemented and tested for the PIC parallel realization [10–12]. The size of the local domain is equal to 2. Any diffusive algorithm is characterized by the number of steps. Actually the number of steps defines how far the data from one PE could be transferred in the course of load balancing. For every step the procedure TRANSFER is called. The more the number of steps of diffusive algorithm, the better load balance could be attained, but also more time is required for load balancing. The tests have shown that the total program time does not decrease with the growth of the number of steps.

5.7. Algorithms comparison

In Tables 1–4 the results of different dynamic load-balancing algorithms testing for the problem of plasma cloud explosion modeling are shown. The size of the problem is the following:

- The size of the SM is $24\times24\times36$.
- The number of background particles is $24\times24\times36\times27=559872$.
- The number of cloud particles is 240128.
- The total number of particles is 800000.

If the number of PEs is greater than 2, the load balancing is required. The results of the tests confirm the

Table 1
Dynamic load balancing is absent

	2 PE	3 PE	4 PE	5 PE	6 PE	7 PE	8 PE	20 PE
Maximum number of particles in PE	400000	353256	223190	207663	213364	153948	152389	79930
Duration of particles processing	110190	97462	62012	57375	59072	42821	46980	22267
Total time	121768	107878	68801	63481	65332	47470	46716	24687

Table 2
Centralized load-balancing algorithm

	2 PE	3 PE	4 PE	5 PE	7 PE	8 PE
Maximum number of particles in PE	400000	266787	200102	160091	114388	100091
Duration of particles processing	110625	73862	55722	44498	32189	27579
Total time	122294	81900	61902	49453	35826	30744

Table 3
Diffusive load-balancing algorithm (the number of steps is equal to 2)

	2 PE	3 PE	4 PE	5 PE	6 PE	7 PE	8 PE	20 PE
Maximum number of particles in PE	400000	266977	200001	160484	133679	111125	100727	42092
Duration of particles processing	110459	75116	56246	45036	37270	30900	28260	11887
Total time	122048	83191	62606	50083	41465	34451	31508	13353

Table 4
Specialized decentralized algorithm

	2 PE	3 PE	4 PE	5 PE	6 PE	7 PE	8 PE	20 PE
Maximum number of particles in PE	400000	328325	200000	161630	133473	115489	100238	41388
Duration of particles processing	110329	90333	56203	45926	36821	32669	28504	11647
Total time	121952	100022	62337	50946	40980	36419	31801	13093

fact that for the small number of PEs (<10), the utilization of a centralized algorithm is more preferable, though decentralized algorithms also give satisfactory results.

6. Automatic generation of parallel code

The PIC method is applied to simulation of many natural phenomena. In order to facilitate the parallel PIC realization, a special system of parallel program automatic generation was designed. This system consists of the VISual system of parallel program Assembling (VisA) and a parallel code generator in C language.

The process of the generation of a parallel program for the PIC and for the other numerical algorithms on the rectangular meshes consists of three steps.

At the first step, a user defines the atomic fragment of computation — a cell of the SM. This cell contains elements of the mesh variables at several mesh points, an array of particles (for PIC method) and procedures in C language, which describe all the computations inside the cell — $\{procedure_1, \ldots, procedure_k\}$ (Fig. 8).

On the next step the user chooses the target multicomputer and the topology of the communication net for which the parallel code should be generated.

After that the generator constructs a minimal fragment (a layer or a column) from the atomic fragments according to the chosen communication topology. The particle arrays of atomic fragments merge to a single particle array for the minimal fragment. The elements of the mesh variables, which hit upon the same point of SM, are unified. In such a way, for every mesh variable, only one 3D array of its elements is formed.

At the third step, the decision on the PEs workload is made. The generator creates a parallel program for a target multicomputer. This program includes data initialization and a time loop. At each iteration of the time loop, k loops (where k is the number of procedures in the description of an atomic fragment) over all the minimal fragments of PE were run. After each kth loop, those elements of the mesh variables, which are copied in several PEs, are updated (if necessary).

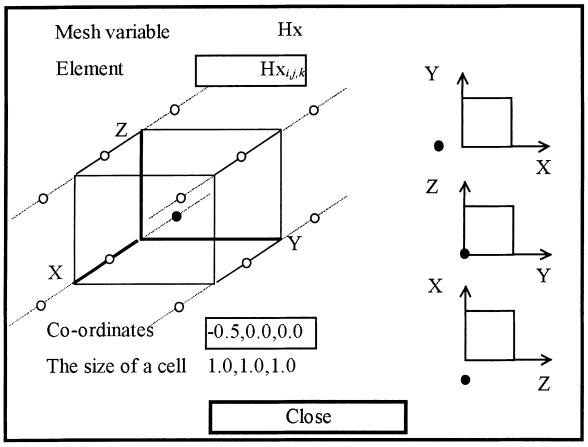

Fig. 8. Review window for Hx mesh variable of an atomic fragment of computation.

All the particles are stored in m arrays (where m is the number of minimal fragments in a certain PE). However, similar to the case of the minimal fragment assembling, the elements of a mesh variable in all the minimal fragments of a PE form one 3D array.

The user develops procedures (computations inside a cell) in C language, using also several additional statements for defining the computations over the mesh variables.

7. Conclusion

The described algorithms of the numerical models assembling provide high performance of the assembled program execution, its high flexibility in recon-

struction of the code and dynamic tunability to available resources of a multicomputer.

High performance of the program execution provides the modeling of big size problems such as the study of a cloud plasma explosion in the magnetized background, modeling of interaction of a laser impulse with plasma, etc.

We apply the assembly technology to realization of different numerical methods and hope to create a general tool to support realization of mathematical approximating models.

Acknowledgements

The work presented in this paper was partially supported by the grants INCO-COPERNICUS-97-7120, RFBR 99-07-90442, RFBR 99-07-90418.

References

[1] V.A. Valkovskii, V.E. Malyshkin, Synthesis of Parallel Programs and Systems on the Basis of Computational Models, Nauka, Novosibirsk, 1988 (in Russian).
[2] V. Malyshkin, Functionality in ASSY system and language of functional programming, in: Proceedings of the First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis, Aizu-Wakamatsu, Fukushima, Japan, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 92–97.
[3] M. Kraeva, V. Malyshkin, Implementation of PIC method on MIMD multicomputers with assembly technology, in: Proceedings of the (HPCN) High Performance Computing and Networking International Conference, Europe, Lecture Notes in Computer Science, Vol. 1255, Springer, Berlin, 1997, pp. 541–549.
[4] R. Hockney, J. Eastwood, Computer Simulation Using Particles, McGraw-Hill, New York, 1981.
[5] Yu.A. Berezin, V.A. Vshivkov, The Method of Particles in Rarefied Plasma Dynamic, Nauka, Novosibirsk, 1980 (in Russian).
[6] D.W. Walker, Characterising the parallel performance of a large-scale, particle-in-cell plasma simulation code, Concurrency: Practice and Experience 2 (4) (1990) 257–288.
[7] E.A. Carmona, L.J. Chandler, On parallel PIC versatility and the structure of parallel PIC approaches, Concurrency: Practice and Experience 9 (12) (1997) 1377–1405.
[8] G.A. Kohring, Dynamic load balancing for parallelized particle simulations on MIMD computers, Parallel Comput. 21 (4) (1995) 683–693.
[9] X. Yuan, C. Salisbury, D. Balsara, R.G. Melhem, A load balancing package on distributed memory systems and its application to particle–particle particle-mesh (P3M) methods, Parallel Comput. 23 (10) (1997) 1525–1544.
[10] M.A. Kraeva, V.E. Malyshkin, Algorithms of parallel realization of PIC method with assembly technology, in: Proceedings of the Seventh (HPCN) High Performance Computing and Network, Europe, Lecture Notes in Computer Science, Vol. 1593, Springer, Berlin, 1999, pp. 329–338.
[11] M. Kraeva, V. Malyshkin, Dynamic load balancing algorithms for implementation of PIC method on MIMD multicomputers, Programmirovanie, Vol. 1, 1999, pp. 47–53 (in Russian).
[12] A. Corradi, L. Leonardi, F. Zambonelli, Performance comparison of load balancing policies based on a diffusion scheme, in: Proceedings of the Euro-Par'97, Lecture Notes in Computer Science, Vol. 1300, Springer, Berlin, pp. 882–886.

Marina Kraeva received her MS degree in Mathematics from the Novosibirsk State University, Russia. From 1993 she is working in the Institute of Computational Mathematics and Mathematical Geophysics of the Russian Academy of Sciences. Her research interests are concentrated in the area of parallel programming. Her research work covers the adaptation of assembly technology of parallel program construction for realization of mathematical models on parallel computers. In 1999, she received her PhD in Computer Science at the State Technical University of Novosibirsk, Russia. She is currently a member of the High Performance Research Group at Iowa State University, USA.

Victor Malyshkin received his MS degree in Mathematics form the State University of Tomsk in 1970, PhD degree in Computer Science from the Computing Center of the Russian Academy of Sciences in 1984 and Doctor of Sciences degree from the State University of Novosibirsk in 1993. From 1970, he worked in software industry. In 1979 he joined the Computing Center RAS, where he is presently the head of Supercomputer Software Department. He also found and is presently heading the Chair of Parallel Computing Technologies at the State Technical University of Novosibirsk. He is one of the organizers of the PaCT (Parallel Computing Technologies) series of international conferences that were held each odd year. He published over 70 scientific papers on parallel and distributed computing, parallel program synthesis, supercomputer software and applications, parallel realization of big size numerical models. His current research interests include parallel computing technologies, parallel programming languages and systems, methods of parallel realization of big size numerical models, dynamic load balancing.