Functionality in ASSY System and Language of Functional Programming

V.E.Malyshkin

Supercomputer Software Department
Computing Center of Russian Academy of Sciences
Novosibirsk, Russia 630090

Abstract

The main features of integrated system to support technology of application problem parallelization, development (assembly) of parallel programs, tuning to available resources of specific multiprocessor system in the course of their execution are presented, the notion of functionality is discussed. Contrary to partitioning assembly approach supports synthesis (in a broad sense) of the whole problem solution on the basis of elementary ready made fragments. It enable us to use the unified technology for solution of wide range application problems (in seismic data and image processing, nuclear physics, modeling of natural phenomena etc.) in the framework of the same parallel programming system. This approach is a generalization of our experience in solution of big-size problems.

1 Introduction.

Assembly approach is a generalization of our experience in development of parallel software and solution of big size problems in seismic data and image processing, modeling of natural phenomena (particle-in-cell method), nuclear physics, geophysics, quantum chaos etc. The main objective of our investigations is a solution of wide range of big size problems in science and engineering, using unified technology of problem parallelization and parallel program creation. This approach was first realized in the course of development of scalable high performance multiprocessor system (MPS) SIBERIA [1,2] and its parallel software. Above mentioned problems were solved on this MPS.

Development of ASSY (ASsembly SYstems) language and system is based first on the analysis of results of big size problems solution including their parallelization, design of parallel algorithm, data transferring and communications, programming and debugging. An another source of design information is the analysis of current high level parallel programming languages and systems such as Linda, Strand-88, Occam, PVM etc. [3-7]. The third source is the theory of parallel program synthesis [8].

Certainly, simple analysis shows that the use of MPS can be organized on the basis of programming language of high nonprocedurality [17] which must provide the generation of high performance and portable parallel programs. But high nonprocedurality of the language means its high specialization for effective solution of problems from specific area. This is a result of usage in program of specific large-scale objects of application area each of which defines ready made fragment of computations. Usually, these fragments can not be used in the other application areas for algorithms definition. In such a way, we can not hope to develop an universal programming language of high nonprocedurality eligable to be used in different application areas.

This being the case, the decision was made to develope and to support a general unified technology of parallel program creation suitable for a wide range of applications and architectures. This is the technology of a computation assembly out of preliminary made fragments.

One of the key element of assembly technology is a program synthesis. We are not going here to solve the problems instead of users, but we are going to support automatic program creation for effective solution of problems from the good studied areas. It means that knowledge of good studied areas (the algorithms of problem solution) can be "canonised" and used in the mode of calculator.

We proceed from the assumption that we need and we can assemble specific MPS installation oriented to a very effective solution of a specific problems from a chosen area. We believe, that user should use the same

0-8186-7038-X/95 $4.00

or similar facilities to create his programs. Our favourite architecture is now linear hierarchical MPSs (like cluster architecture, fig.1,2) for which ASSY approach was developed first [1].

In this paper the main principal features of ASSY are presented.

This work is being done in the framework of the academic research project *Assembly Systems,* which is funded by Russian Academy of Sciences. It is intended for studying assembling phenomena in parallel processing.

2 The elements of assembling.

The following main constituents of assembling can be pointed. They must be supported from ASSY software.

2.1. Assembling and Partitioning

Our main key word is ***assembly***. Contrary to partitioning, the assembly technology supports explicit assembling of a whole program out of ready made fragments of computations like variables, procedures, macros, nets, notions, functions rather then dividing of problem, defined as a whole, into the suitable fragments to be executed on the different processor elements (PE). These fragments are the elementary blocks, the "briks", to build the whole program (the description of whole problem, there is not a difference in ASSY here). An algorithm of problem assembling (APA) is kept by ASSY and used later for program/problem parallelization. APA 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.

2.2 Separation of fine grain and coarse grain computations

Fine and coarse grain computations are executed on the different level of hardware and software. Fine grain computations are encapsulated inside a module. This module can be implemented very effectively on specialized processor element. Parallel program is assembled out of these ready made modules. The system of modules of a program will define a system of interacting processes (coarse grain computations). It means, for solution of specific problem a specific nonhomogeneoue MPS with specialized PEs should be assembled. Encapsulation of fine grain computation provides the possibilities to use formal methods of parallel program construction [8] and to use explicitly two level representation of an algoritm: a scheme level and a program level.

2.3 Separation of semantics and scheme of computation

Fine grain computations define a sufficient part of semantics (functions) of computations, they are realized within a module. Therefore, on the coarse grain level only a scheme (non-interpreted or semi-interpreted) of computations is assembled. It means, that formal method of automatic synthesis of parallel programs [8] can be successfully used and standard schemes of parallel computations can be accumulated in the libraries.

2.4 The regular method of MPS assembling

Diversity of different structures of MPSs and algorithms (structures of data transferring between PEs and between processes) is too big and their structures do not always match each other. The consequence of unmatched structures of algorithm and MPS is a slump of MPS performance. Thus, restrictions must be imposed on the structure of MPS to fix minimum communications between PEs and to define conceivable extensions of communication lines. Knowledge of main peculiarities of MPS structure provides the possibility to create parallel program tuning on the available resources of MPS in the course of its execution. These restrictions are described as a set of rules for MPS assembling [1].

2.5 Parallel program is assembled out of ready made modules

Application program must be assembled out of ready made modules as a wall is laid from the bricks. It is known an entire computation can be automatically divided into fragments (partitioning) for very simple regular classes of algorithms usually represented as a systems of recurrent equations with linear dependencies between indexes[9,10]. Assembled computation keeps the "***seams*** " of assembling. They give to compiler an information for automatic dividing of entire computation along the "seams" in suitable fragments (as data as program code) which can be executed on separate PEs .

2.6 Mapping assembling

Parallel program must be tunable on the available resources of the specific installation. To provide this property first of all the structure of application algorithm must be restricted in order to be matched to MPS structure. Our experience have shown that such reasonable restrictions can be found [1] and wide class of numeric algorithms can be transformed to satisfy to the restrictions. In this case it will be possible to accumulate a library of standard mappings and to assemble a mapping of entire computation out of them.

3 Functions: Analysis and Synthesis.

ASSY program is assembled out of functions. It means really in SSY the following.

We will abide by the following notation. Function $F:D \rightarrow B$ is a computable function, A^iF, $i=1,2,...$, are the algorithms computing function F, A_F is used to identify any algorithm computing F. P_A is a certain procedure realizing an algorithm A_F and computing the corresponding function F, maybe, in a subdomain $D1 \subseteq D$.

3.1 Function definition

In real program we can not use a definition of function F as a mapping from D set into/onto B set, $F : D \rightarrow B$. Certainly, rigorous mathematical definition of F contains the definitions of all the conceivable algorithms computing, this function F, but we are not actually able in programming system to derive a suitable algorithm A_F from the mathematical definithion of a function F (suitable in the sense that this algorithm is one of the best to be executed on a chosen multiprocessor or PE) and to build a suitable procedure P_A, realising A_F. On the other hand, this is impossible to say, that a function F is defined by a single algorithm A_F presented by a certain procedure P_A, i.e., the value $x=F(y)$, $x \in B$, $y \in D$, and x is computed by a procedure call $x=P_A(y)$. We would loose in this last case the possibility to choose a suitable algorithm and program for execution and, as consequence, ASSY program will not be portable and high performance one. Thus, we had to find some acceptable compromise.

In ASSY a pragmatic definition of function is used, which can be really implemented in a program. In our reasoning we abided by the source proposition that in a system of functional programming a function definition must be supported from the program synthesis method. It must guarantee the successful choice/derivation of suitable algorithm for realization and support by doing so the portability of ASSY program, the diversity in architectures of multiprocessors and processor elements (PE) on which ASSY program can be executed.

Any ASSY function F is roughly defined as a finite set of different algorithms A^iF, $i=1,2,... n$, computing F. Every elementary algorithm A^iF is presented by a ready-made procedure (module) P^i_A. Thus, elementary algorithm is presented in ASSy as procedure, and function F is presented as a set of algorithms, computing F. We proceed from the assumption, that this set contains all the necessary algorithms. In particilar, there are many algorithms for matrices multiplication, but in practice this is enough, usually, to have about 20 of them in the library to cover a wide range of pragmatic applications. Practically it means, that suitable algorithm to compute F, can be chosen among the finite set of ready-made algorithms. Certainly, this choice is done as derivation. If in some specific case there is not such suitable algorithm in F definition, it must be added in the definition of the function F. Usually the set of algorithms is represented as a net (a bi-partite graph, fig.3), which defines in compact form a big number of algorithms and their compositions/superpositions. As variables the arrays can be used, it provides the use of infinite set of algorithms in function F definition. Knowledge is accumulated as a library of functions, this is a first stage (stage of analysis) of ASSY application. On this stage the basic functions of application area are analyzed and designed.

Drawing analogy to the geometry we can compare the ASSY and logic programming approaches in the following way. Logic approach uses Euclid system of axioms as a compact description of geometry (all the algorithms for solution of any geometric problem are presented here), but there is a problem to derive suitable algorithm. These axioms constitute the knowledge base of geometry. In our approach knowledge base accumulates the best algorithms for solution of the concrete problems. Therefore, if a requested algorithm is absent in the knowledge base, it can not be derived and must be appended into the knowledge base.

More exactly, but nevertheless roughly, a function F is defined as a tuple F=(X, G, V, W), where $X=\{x,y,...,z\}$ is a finite set of variables (simple variable, arrays, data structures), $G=\{a,b,...,c\}$ is a finite set of functional symbols (operations). To each operation $a \in F$ the tuples of input $in(a)=(x_1,x_2,...,x_n)$ and output $out(a)=(y_1,y_2,...,y_m)$ variables from X are associated. Function definition is depicted as a bi-partite graph (fig.3).

The set of terms T(V,G) is induced by a function definition:

1) if $x \in V$, then $x \in T(V,G)$, $in(x)=out(x)=\{x\}$.

2) if $a \in G$, $in(a)=(x_1,x_2,...,x_n)$, $out(a)=(y_1,y_2,...,y_m)$ and $t_1, t_2,...,t_n$ are the terms from T(V,G) such that $\forall i \in \{1,2,...,n\}(x_i \in out(t_i))$, then $t=a(t_1,t_2,...,t_n)$, $t \in T(V,G)$.

The terms from T(V,G) may contain superfluous computations, Thus, the set T^1 is formed which includes only such terms of T(V,G) in whose graphical representation each operation on any of term's path from the leaves to the root occurs not more than once to compute each of its output variables. This condition is enough to delete insufficient computations.

The set $T(V,W)=\{t \in T^1 | in(t) \subseteq V \& out(t) \subseteq W \neq \emptyset\}$ can be specified. The terms from T(V,W) define the set of all the algorithms computing the function F. In the case the arrays are used the sets T(V,G), T^1 and

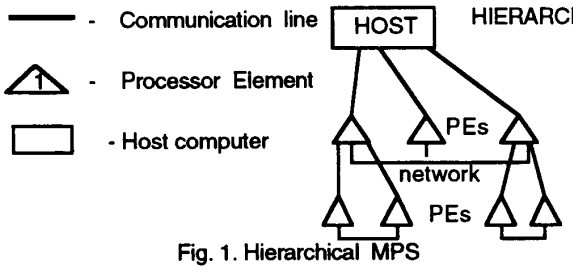


Fig. 1. Hierarchical MPS

HIERARCHICAL MULTIPROCESSOR SYSTEMS

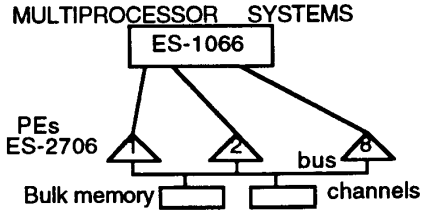


Fig. 2. Hierarchical MPS SIBERIA

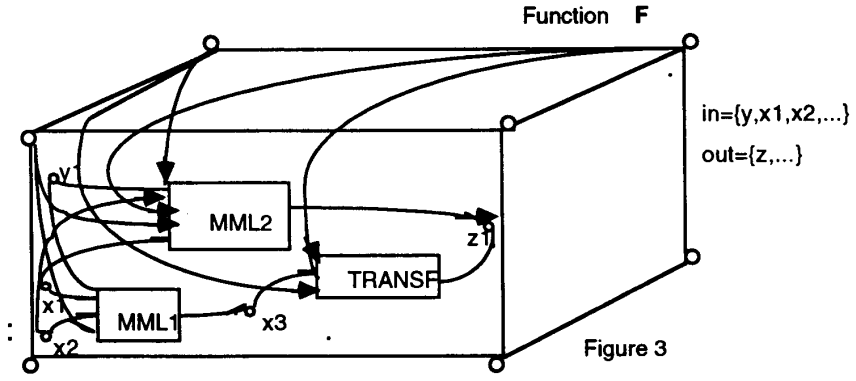


Figure 3

big_box

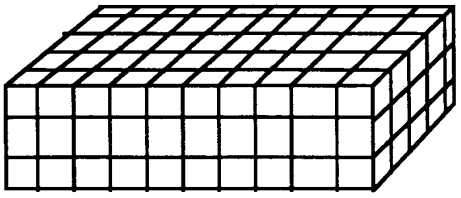
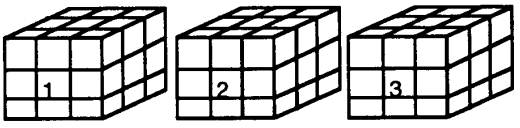


Figure 4



small_box

Figure 5

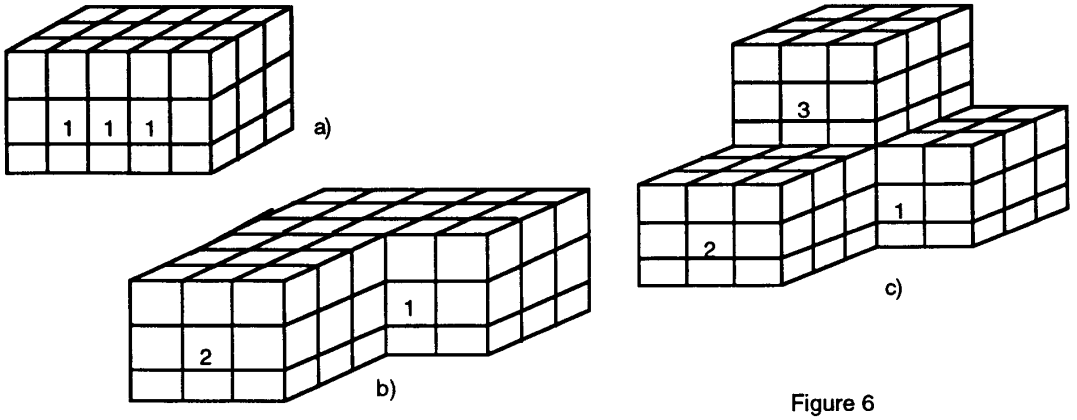


Figure 6

T(V,W) are infinite sets, they are built by specific derivation algorithm.

The interpretation I in the domain D bind each operation $a \in F$ to a function $f_a = I(a)$, each variable $x \in X$ to an element (value of x) $d_x = I(x)$, $d_x \in D$, $d_x \neq \Omega$, were Ω, $\Omega \in D$, is a symbol of indefiniteness. Each variable $y \in X$, such that $y \in X \setminus V$ & $\exists t \in T(V,W)$ $(y \in out(t))$, $t = b(t_1,...,t_n)$, $in(b) = \{x_1,...,x_n\}$, is bound to an element $d^t_y \in D$, $d^t_y = val(t,y) = f_b(val(t_1,x_1),...,val(t_n,x_n))$, $val(x,x) = d_x$.

The function $f_a = I(a)$ is assumed to be finally computed by the program module mod_a. This module mod_a is either a ready made module from the library or, in the case the function $f_a = I(a)$ presented as a function definition, mod_a must be synthesised from the definition. In ASSY mod_a can be developed with any suitable language. Some information on peculiarities of mod_a (language, estimation of consumed time and memory, properties of input data etc.) is kept on scheme level and used during algorithm derivation.

If $y \in X \setminus V \& \neg\exists t \in T_V$ $(y \in out(t))$, then $\forall t \in T_V$ $(val(t,y) = \Omega)$. The interpretation I is called correct if for any terms t_1 and t_2 from T^W_V such that $x \in (out(t_1) \cap out(t_2))$ the condition $val(t_1,x) = val(t_2,x) = d_x$ is satisfied. It means, that the function F definition is a single assignment scheme and the value of variable z (fig.3), computed by MML2 or MML1;TRANSF, will be the same (input data are the same too). Any way of variable z computation can be chosen depending on specific MPS or other significant factors, definition of F function is always excessive to provide a good choice.

ASSY approach to program synthesis is a very simple and very restricted approach, but with the growth of number of the algorithms in the knowledge base it provides, in numeric computations for example, a good choice and possibilities for derivation of suitable algorithms to be executed on specific installation. Moreover, the bigger flexibility in algorithm derivation can be harmful. Only expected algorithms must be derived in numeric computations! Additionally, it enable us to accumulate in libraries the standard schemes of computations, which are in common use together with the best ways of their execution. We hope there exist not many different schemes.

3.2 Specialized language assembling

One of the key peculiarity of our approach is the synthesis (composition) of new computations on the basis of the standardized fragments and knowledge accumulation. ASSY supports the accumulation of the standard schemes of computation as a new elements of the language [11,12]. Actually, it means, that specific languages for application in specific area are developed and ASSY supports this process. The greater is the knowledge base more diversity of the language facilities is.

3.3 Explicitly two-level description of a program

First basic level of program constitutes a noninterpreted scheme of computation with a single assignment. Every algorithm or function are presented here by their names, input/output variables and some additional information . On this level the different necessary formal optimizing transformations of scheme are done including algorithm derivation/choice, based on the architecture and the configuration of MPS installation. Many transformations can be done successfully as a scheme is a far more simple object for analysis and transformation in comparison to a program. Program synthesis (derivation of suitable algorithm) as a choice of the best algorithm among all the algorithms of requested function definition is done here. This choice is made with due regards for the properties of specific installation on which the program will be executed, its available resources, communication network structure, types of PEs etc. Here an analysis of algorithm is provided to map it on MPS in the best way. The peculiarities of input data are taken into account too, for different input data the different algorithms to compute a requested function can be built.

The second level is the level of computation semantic definition, the program is generated in chosen imperative language of parallel programming, resources are assigned, procedures are substituted instead of names of algorithms, program is compiled, data are input and a program is executed.

4 Example

As example of ASSY approach application the particle-in-cell method is regarded[13-15] for simulation of generation of magnetosonic disturbances in magnetospheric plasma[16]. Apart from the description of particle-in-cell method and ASSY language facilities the technology will be demonstrated by the series of pictures.

The 3-D space of modeling is represented as a big_box (fig.4). This *big_box* is divided into small-boxes. Data processing inside small_boxes are defined by the same or different fragments of computations. Every fragment is defined as a function, an example of simple function definition is shown in fig. 3. Certainly, the fragments of computations for the particle-in-cell method are defined by far more sophisticated scheme. Among the variables of computation fragments there are the space variables, which define the structure of small_box.

On the second stage the whole computation (big_box) is assembled out of computation fragments, i.e., small_boxes. Let us assume, that there exist 3 different fragments (fig. 5). The whole computation is laid sequentially out of small_boxes of different type (fig. 6a, b, c), while the whole big_box is assembled. ASSY executive system knows the "seams" of assembling and has full information for different types of partitioning (in different boxes and layers). The knowledge of these seams provides partitioning as code as data. The whole parallel program is assembled automatically. If ASSY executive system makes decision to locate adjacent fragments to the same PE, then communications between these fragments are canceled (they are not generated in the whole parallel program).

This scheme of particle-in-cell method realization is standardized and can be used with different fragments of computations (inside small_boxes) to be applied for the modeling of different natural phenomena.

5 Conclusions.

Utilization of assembly approach in development of MPS SIBERIA and solution of big size problems in seismic data and image processing, modeling natural phenomena, electrical physics, quantum chaos etc. shows this method can be used as a basis for creation and accumulation of knowledge on solution of big size problems on different architecture multiprocessors.

We schedule to implement ASSY as a general technology of problems solution on multiprocessor computer systems oriented, first of all, for solution of big size numeric problems.

References

[1] V.E.Malyshkin. "Linearized mass computation", *Proceedings of Parallel Computing Technologies International conference*, Novosibirsk, Russia, 339-353,1991.

[2] V.A. Anisimov, V.E.Malyshkin. "Assemble Parallel Programming System INYA", *Proceedings of Parallel Computing Technologies International conference* Novosibirsk, Russia, 339-353, 1991.

[3] D.Gelernter, A. Bernstein. "Distributed communications via global buffer", *Proc. Symp. on Proncoples of DistributednComputing*, August, 1982.

[4] D.Gelernter. "Generative Communication in Linda", *ACM Trans. on Prog. Languages and Systems 7, 1* 10-18, 1985.

[5] D.May. "OCCAM".*SIGPLAN Notices*, 18, 4 (April), 1983, 69-79.

[6] A.Burns. *Programming in Occam.* Addison-Wesley, Reading, MA, 1988.

[7 I.Foster, S.Taylor. *Strand: New Concepts in Parallel Programming*, Prentice Hall, New Jersey 1990

[8] V.Valkovskii, V.Malyshkin. *Parallel Program Synthesis on the basis of Computational Models*. Nauka, Novosibirsk, 1988 (in Russian).

[9] R.Karp, R.Miller, S.Winograd. "The Organization of Computations for Uniform Recurrent Equations". *J. of ACM*, V.14, No.3 (1967), pp.563-590.

10. A.Pnuely,R.Zashi."Realizing an equational specifications" *Lect. Notes in Comp. Science*, V.115 (1981) pp.459-478.

[11] V.Malyshkin. "Assembly Parallel Programming: some Examples", *Proceedings of Parallel Computing Technologies International conference* (1993), Obninsk, Russia.

[12] V.Malyshkin. "Assembly Environment for Development of Application Parallel Programin", *Proceedings of International conference on High-Performance Computing and Networking, LNCS No. 797,* Munich, Germany (1994).

[13] D.Walker. "Characterizing the Parallel Performance of a Large-scale, Particle-In-Cell Plasma Simulation Code", *Concurrency: Practice and Experience*. Vol. 2(4) (1990) pp. 257-288.

[14] D.Walker. "Particle-In-Cell Plasma Simulation Codes on the Connection Machine". *Computer Systems in Engineering*, Vol. 2, No. 2/3 (1991), pp. 307-319.

[15] A.Maccabe. "Performance of a Particle-In-Cell Plasma Simulation code on the BBN TC 2000". *Concurrency: Practice and Experience*, Vol. 4(1) (1992), pp. 1-18.

[16] G.Dudnikova and others. "Laboratory and computer simulation of generation magnetosonoc disturbances in magnetospheric plasma". *Prog. XX ICPIG*, Vol. 2, Pisa, 1991.

[17]V.Valkovskii, V.Malyshkin. "On more detailed definition of the notion of programming languages nonprocedurality". *Kibernetika*, No.3, 1981 (in Russian).