Bull. Nov. Comp. Center, Comp. Science, 3 (1995), 69–75

On parallel recursive mapping algorithm for pyramidal multiprocessor systems*

O.G.Monakhov

A problem of mapping of an information graph of a complex algorithm into the pyramidal interprocessor network of a parallel computer system is considered. The parallel recursive algorithm for optimal or suboptimal solution of the mapping problem, the objective functions for mapping and experimental results for the pyramidal multiprocessor system MEMSY are presented.

1. Introduction

There is considered a problem of mapping of the information graph of a complex algorithm into the pyramidal interprocessor network of the parallel computer system (CS), which consists of processor nodes (PN) with distributed shared memory. The mapping problem [1–4] is known as NP-complete problem, and it is reasonable to develop a parallel algorithm for solving the problem. In this paper the parallel recursive algorithm is presented for optimal or suboptimal solution of the mapping problem, the objective functions for mapping are developed. It is shown by experiments on multiprocessor system MEMSY that the proposed algorithm is faster as compared to the sequential centralized algorithm.

2. Optimal mapping problem

Let a model of a parallel program be the graph $G_p = (M, E_p)$, where M is a set of modules (processes), E_p be a set of edges, representing information connections between modules. Let t_i be defined as weight of the module $i \in M$, representing the execution time (or the number of computational steps) of the module i. Let v_{ij} be defined as weight of the edge $(i, j) \in E_p$, representing the number of information units passed from the module i to the module j.

A model of the multiprocessor system with distributed memory is an undirected graph $G_s = (P, E_s)$ representing the network topology (struc-

*Supported by the Russian Foundation for Basic Research under Grant No. 94-01-00682.

ture) of the system, where P is a set of PN, and edges E_s represent interconnection links between PN.

The distance between the nodes i and j of the graph G_s (G_p) is denoted as d_{ij}. The neighborhood of the node i with the radius $\rho \geq 1$ is the set $L_\rho(i) = \{j \in M \mid d_{ij} \leq \rho\}$. Let $L(i) = L_1(i)$.

Let $\varphi : M \to P$ be the mapping of an information graph of the parallel program G_p into the structure G_s of CS. Let the mapping φ be represented by the vector $X = \{x_{ik};\ i \in M, k \in P\}$, where $x_{ik} = 1$ if $\varphi(i) = k$ and $x_{ik} = 0$ if $\varphi(i) \neq k$.

Let the quality of the mapping parallel program graph into the structure of CS for the given vector X be described by the functional: $F(X) = F_E(X) + F_C(X)$, where $F_E(X)$ represents computational cost (the overall module execution time of a parallel program on the system or the load balancing of PN for given X), and $F_C(X)$ represents the interprocessor communication cost (the overall interaction time between modules, which are distributed in different PN, or the distance between adjacent modules of the program graph on CS structure for given X). The optimal mapping problem of a parallel program graph into CS structure consists in optimization of the functional $F(X)$ by means of the parallel recursive algorithm.

3. Cost functionals of mapping

Now let us describe two examples of the objective cost functions which represent the computational load balance of PN and the communication cost for the given mapping X

$$F_1(X) = \sum_{k=1}^{n}\left[\left(\sum_{i=1}^{m} x_{ik} - Mx_k\right)^2 + \sum_{p=1}^{n}\sum_{i=1}^{m}\sum_{j\in L(i)} d_{kp}x_{ik}x_{jp}\right],$$

where $Mx_k = \sum_{p\in L_\rho(i)}\sum_{i=1}^{m} x_{ip}/|L_\rho(k)|$.

The first term in this expression describes deviation of PN_k load from average load in the neighborhood of PN_k with the radius $\rho \geq 1$, the second term describes the distance between PN_k and processors, which contains the adjacent modules to the modules embedded into PN_k, $1 \leq k \leq n$.

$$F_2(X) = \sum_{k=1}^{n}\left[\sum_{i=1}^{m}\left(t_i x_{ik} + \sum_{p=1}^{n}\sum_{j\in L(i)} c_{ij}(v_{ij})d_{kp}x_{ik}x_{jp}\right)\right],$$

where $c_{ij}(v_{ij})$ is the time needed to transfer v_{ij} data units between the modules i and j, when they are located in the neighbouring PN. The first term in this expression describes the overall module execution time, the second term describes the overall interprocessor communication time, $m = |M|$, $n = |P|$ and $\sum_{k=1}^{n} x_{ik} = 1$.

Thus, the optimization of the mapping φ consists in minimizing the nonlinear function $F(X)$ with linear restrictions and integer variables. Let Z denote this task. The optimal solution of the task Z will be found by the following parallel recursive mapping algorithm.

4. Parallel recursive mapping algorithm

In this section, there is presented a hierarchical recursive parallel mapping algorithm for partitioning the modules of a parallel program, described by the graph G_p, into $n \geq 4$ sets and allocating each set on its own processor of the pyramidal multiprocessor system. The goal of the algorithm is to produce an allocation with the minimal communication cost and the computational load balance of processors.

At the first step algorithm divides a given parallel program graph into four parts and finds the optimal partition according to the given cost function. Then, each part of the graph is allocated to each node on the upper plane of the system. Each node on the upper plane divides its part of the program graph into five parts and allocates these parts to its own node and to four down nodes on the lower plane. Then, this algorithm is recursively repeated on the lower plane and on each other plane of the system until the bottom plane. In the end each node of the system has its own part of the program graph (Figure 1).

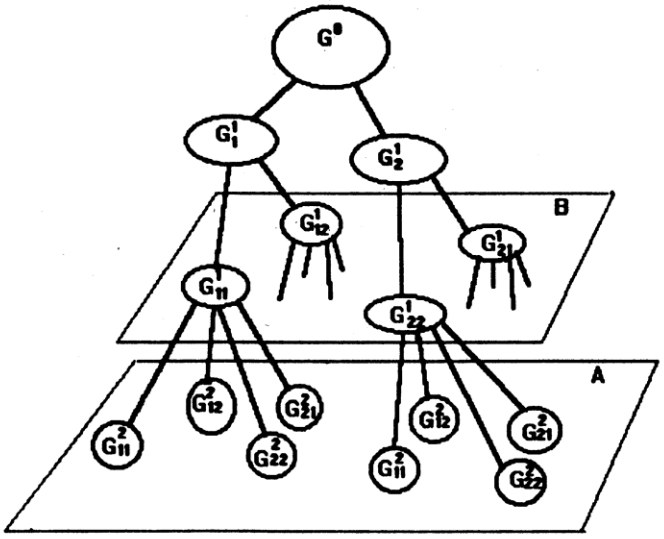

Figure 1. Graph of the recursive mapping algorithm

The algorithm contains a basic procedure – bisection, which divides the modules of the program graph into two sets with the minimal communica-

tion cost between them and roughly equal computation load. The bisection algorithm consists of the following steps.

1. All modules are divided into two sets: $|P_1| = n - 1$ and $|P_2| = 1$.
2. A module $i \in P_1$ is removed to P_2, where the module i has the maximal gain $D_i = \sum_{j \in P_2} d_{ij} - \sum_{j \in P_1} d_{ij}$.
3. If $\sum_{j \in P_2} t_j < \sum_{j \in P_1} t_j$, then go to Step 2, else end.

The hierarchical recursive mapping algorithm divides the given parallel program graph and allocates the parts of the program graph on processors of a system with pyramidal topology. The upper level (plane) of the system contains 4 nodes and other levels contain 4^k nodes, where k is the number of the levels. The recursive algorithm consists of the following steps.

1. The modules of the program graph are divided by bisection algorithm into four parts (at first – into two parts, then each part – again into two parts) and are allocated to four processors of the upper plane.
2. Each processor of the upper plain divides its part of the program graph into five parts and allocates these parts to its own node and to four down nodes on the lower plane connected with this node. Each processor $p_j \in \{p_1, p_2, p_3, p_4\}$ of the upper plane executes (in parallel with others) the following algorithm.

   2.1. All modules allocated to p_j are divided into two sets: S_1 and $S_2 = \emptyset$. Let $M_2 = M \setminus S_1$.
   2.2. A module $i \in S_1$ is removed to S_2, where the module i has the maximal gain $D_i = \sum_{j \in M_2} d_{ij} - \sum_{j \in S_1} d_{ij}$.
   2.3. While $\sum_{j \in S_2} t_j < \sum_{j \in S_1} t_j$ go to Step 2.2.
   2.4. Modules of the set S_2 are allocated to the processor p_j and modules of the set S_1 are divided by the bisection algorithm into four parts and are allocated to four processors of the lower plane (as at Step 1).
   2.5. In parallel, each processor of lower plain applies the algorithm recursively from Step 2.1.

The algorithm is recursively repeated until the bottom plane of the system and can map of a program graph into the pyramidal system with a given number of levels.

5. MEMSY – pyramidal multiprocessor system

MEMSY (Modular Expandable Multiprocessor System) [5] is an experimental multiprocessor system with a scalable architecture based on locally

shared memory between a set of adjacent nodes and other communication media. The MEMSY system continues the line of systems which have been built at the University of Erlangen, Nurnberg (Germany) using distributed shared-memory and pyramidal topology.

The MEMSY structure consists of the two planes with 4 nodes in the upper plane and 16 processor nodes in the lower plane. In each plane the processor nodes form a rectangular grid. Each node has a shared-memory module, which is shared with its four neighbouring nodes. Each grid is closed to a torus. One processing element of the upper plane has access to the shared memory of the four nodes directly below it, thereby forming a small pyramid (Figure 2).

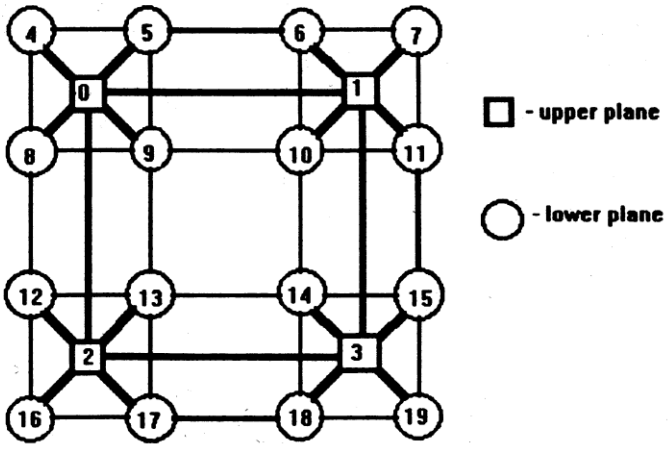

Figure 2. Structure of MEMSY system

The MEMSY consists of the following functional units: 20 processor nodes, one shared-memory module (communication memory – 4 Mbytes) at each node, the interconnection network between processor nodes and communication memories, a special optical bus (FDDI net) connecting all nodes, a global disk memory (1.57 Gbytes). Each node of the MEMSY consists of four processors MC88000 with 25 Mflops performance, 32 Mbytes local memory, 500 Mbytes local disk memory.

The programming model of the MEMSY was designed to give a direct access to the real structure and the power of the system. The application programmer can use a variety of different mechanisms for communication and coordination defined as a set of system library calls which can be called from C and C++ languages. There are the following mechanisms for communication and coordination: shared communication memory between neighbouring nodes, message passing mechanisms, semaphores and spinlocks, EDDI net for fast transfer of high volume data. The operating system of MEMSY (MEMSOS) is based on Unix adapted to the parallel

hardware. The multitasking/multiuser feature of Unix and traditional I/O library calls for local and global data storage are supported. The MEMSOS allows different applications (single user parallel program) to run simultaneously and shields from one another.

6. Experimental results on MEMSY system

The proposed parallel mapping algorithm has hierarchical recursive structure and suits for the pyramidal topology of MEMSY. This algorithm was implemented on MEMSY for one, two and four nodes with communication via shared memory. In experiments there were allocated simple program graphs – square grids with a given size, but arbitrary weighted graphs of parallel programs can be also allocated. There were obtained mapping of the given graphs on all MEMSY nodes, the computing time, speed up and efficiency of the mapping algorithm and values of the objective cost function. The results are presented in Tables 1 and 2.

Table 1. Computing time (T_N), speed up ($S_N = T_1/T_N$) and efficiency ($E_N = S_N/N$) of the parallel mapping algorithm executed on N nodes for mapping of square grid ($a \times a$) into MEMSY architecture

Size of grid	T_1 sec.	T_2 sec.	T_4 sec.	S_2	S_4	E_2	E_4
10 × 10	8	5	4	1.6	2.0	0.8	0.5
15 × 15	97	57	40	1.7	2.43	0.85	0.61
20 × 20	529	310	201	1.7	2.63	0.85	0.66
25 × 25	2058	1190	874	1.72	2.35	0.86	0.59
30 × 30	6145	3558	2510	1.72	2.45	0.86	0.61
35 × 35	15839	9421	5931	1.68	2.67	0.84	0.66
40 × 40	35253	20359	13162	1.73	2.68	0.865	0.67

Table 2. The objective cost function values (F_N) obtained by the parallel mapping algorithm on N nodes for mapping of square grid ($a \times a$) into MEMSY architecture

Size of grid	F_1	F_2	F_4	F_1/F_2	F_1/F_4
10 × 10	840	840	844	1.0	0.995
15 × 15	1778	1770	1676	1.004	1.06
20 × 20	2488	2488	2546	1.0	0.997
25 × 25	4326	4230	4184	1.02	1.033
30 × 30	5786	5786	5726	1.0	1.01
35 × 35	7884	7716	7628	1.02	1.033
40 × 40	9600	9600	9546	1.0	1.005

Thus, the results show that the proposed parallel recursive algorithm

is efficient and produces an optimal or good suboptimal solution of the mapping problem. The algorithm has hierarchical recursive structure and suits well for application on the multiprocessor system MEMSY.

Acknowledgements. I would like to thank Professor H. Wedekind for supporting this work. I also thank MEMSY system group and especially T. Thiel and S. Turowski for helpful consultations and providing access to MEMSY system.

References

[1] N.N. Mirenkov, *Parallel programming for multimodule computer systems*, Radio i svyas, Moscow, 1989.

[2] F. Berman, L. Snyder, *On mapping parallel algorithms in parallel architectures*, J. Parallel Distrib. Comput., 4, 1987, 439–458.

[3] D. Fernandez-Baca, *Allocating modules to processors in a distributed system*, IEEE Trans. Software Eng., 15, 1989, 1427–1436.

[4] O.G. Monakhov, *Parallel mapping of parallel program graphs into parallel computers*, Proc. Internat. Conf. "Parallel Computing 91", Elsevier Science Publishers, Amsterdam, 1992, 413–418.

[5] F. Hofman, M. Dal Cin, A. Grygier, H. Hessenauer, U. Hildebrand, C.-U. Linster, T. Thiel, S. Turowski, *MEMSY: a modular expandable multiprocessor system*, Technical report, University of Erlangen-Nurnberg, 1992.