Bull. Nov. Comp. Center, Comp.Science, 17 (2002), 31–38

High-level communication functions for parallel programs

V.D. Korneev

In this paper, the functions, raising the level of communication operations and mapping operations of the solution space of applied tasks on the multicomputer memory in the MPI-programs, are offered.

1. Introduction

At the present time, the number of the multicomputer systems, assembled on a different microprocessors basis, such as: Intel Paragon (Intel, the USA), SP2 (IBM, the USA), Cray3D and Cray3E (Cray Research, the USA), PowerXplorer (Parsytec, Germany), MVS-1000 (Quantum, Russia), is rapidly increasing. The number of computers in such systems also grows. The multicomputer system with with 2048 computers has been created by the Hitachi company (Japan). The ASCI Red (the USA) consists of more than 9,000 Pentium PRO/200 computers. The ASCI Wait (the USA) contains 8,192 computers PowerPC 3-III with the peak processing power 12.2 Tflops. An important feature of such a computing system is the possibility to concentrate all the computing resources on the solution of one applied task.

Along with the development of multicomputer systems, the parallel programming systems are being developed as well. One of such systems is the MPI system [1, 2] being one of the most advanced from the existing systems. The MPI is the basic programming means of the above-mentioned multicomputers and many other types of computers not mentioned here. The MPI works on the most different multicomputer architectures, both with distributed, and shared memories. In addition, the MPI works on homogeneous and heterogeneous computer networks. An important characteristic of the MPI is that for the user programs it creates a virtual environment: a virtual multicomputer with distributed memory and virtual networks of this virtual multicomputer. Virtual computers and their communication structures are mapped by the MPI system onto a concrete physical system automatically, i.e., the user is not obliged to take architectural peculiarities of a certain multicomputer into account when writing his own parallel programs. In the program start operator, the user orders the number of computers for solving his task, and in the executed program he defines a

communications topology between these computers. The MPI realizes the user's order on a concrete physical system. In this case, restriction is an operative memory size of a physical multicomputer. A virtual environment ensures the possibility to transfer the user programs, thus providing the possibility to create libraries and the parallel program packages.

The computing system architectures with distributed memory are basically oriented to the two computing models: the MPMD (Multiple programs – Multiple Data) and the SPMD (Single program – Multiple Data). In the first model, the MPI-program represents a set of autonomous processes functioning under the control of their own programs and interacting with help of a standard set of library procedures. I the second model, all the branches are executed under the same program. Since the MPI creates a virtual multicomputer system with distributed memory, then accordingly the MPI is oriented to these computing models.

Since multicomputers are basically oriented to solving one problem, the SPMD-model is used in most cases on such systems. In other words, this model is identified as data parallelization [3–11]. Problems of the linear algebra, problems, solved by difference methods and many others are sufficiently effectively parallelized by this method. The above problems are intrinsic of such application areas as: nuclear physics, geophysics, weather forecasting, and many others. Such a broad use is provided by the following characteristics of this model. First, logical schemes of tasks from the above-specified application areas correspond to the given computing model. Second, with the help of this parallelization method it is possible to provide the optimum timing relationships between computations in branches and communications between these parallel branches. And third, the algorithms parallelized by this model can be readily easy adapted to the number of computers in the computing system.

When parallelizing algorithms according to the SPMD model, the following problems arise:

1. The mapping of the computing space of applied tasks onto the multicomputer memory with overlapping of the domain boundaries or without it.
2. The exchange of the overlapping domain boundaries between branches of a parallel program.

The problem of data mapping onto a parallel computing system is in a general case, a difficult task. A special case of such a mapping is considered here. It is expected that a computing space is presented as n-dimensional arrays of values, i.e., this is a mapping of n-dimensional array onto a multicomputing system memory. Such a mapping corresponds to a large class of practical tasks.

The MPI system has all necessary means for solution of the problems in question. First, the MPI has the sufficiently developed means for the creation of the MPI-types of data. The MPI-types essentially raise a programming level, in particular, in exchanging data between computers as compared to the programming of exchanges by standard types of data. Second, the MPI has a large set of communication functions of a sufficiently high level. However if we look, for example, at the MPI-program as product of two matrices on a three-dimensional computer grid, we will see that the computing part has 5 operators, while the other part of the program having about 90. The latter are operators of the creation of the MPI-types of matrix domains sent to computers from the root computer, as well as of the creation of the MPI-types of results collected in the root computer, and also, of sending and receiving data. This means that the main efforts and a great deal of time needed for writing a parallel program is spent on the creation of the MPI-types and data communications. In this case, the parallel program becomes bulky and poorly debugged. At the same time, the problem of data distribution between the parallel system computers and the subsequent collection of results from all the computers arises for the system programmers, and each programmer has to solve this problem anew. Therefore, transferring the distribution, collection and data exchange functions to the MPI library of functions seems to be an urgent task.

2. Communication functions

In this section, the functions, substantially saving the user, the trouble and expense of the above-mentioned difficulties, are offered. Such functions are the following:

1. The functions of mapping n-dimensional arrays onto multicomputer memory;
2. The functions for the exchange of overlapping domain boundaries between parallel program branches.

For the efficient solution of a problem, the structure of its parallel algorithm should be brought nearer to the computing system architecture. The virtual topologies allow providing an optimum "approximation" of a problem to the system architecture with a good transfer of programs within the framework of different computing systems. Since we consider n-dimensional arrays for solving a problem, the virtual topology of a computing system should be similarly defined, i.e., we mean the Cartesian topology (n-dimensional grids and torahs). For example, for a three-dimensional data array, such computer structures as: one-dimensional, two-dimensional, three-dimensional grids, and torahs are acceptable.

The function realizing such a data decomposition is the following:

```
Rmas_mpi(void* IBUF, MPI_Datatype datatype, int ndims,
         int* dims, int p, MPI_Comm com, int p, void* PBUF,
         int* rdims, MPI_Datatype* gran, int* disp,
         int* left_right);
```

Further we denote the input parameters IN and output – OUT.

IN `IBUF` is the name of a data array, specifying the initial computing space. The array is stored in the root computer. This array needs to be "cut" into domains, and these domains should be distributed among the computers.

IN `datatype` is the type of the `IBUF` array.

IN `ndims` is the dimension of the `IBUF` array.

IN `dims` is the one-dimensional integer `ndims` array, whose elements are the dimensionality of `IBUF` array along the corresponding coordinates: `IBUF[0]` is the size along the coordinate 0, `IBUF[1]` – along the coordinate 1, etc.

IN `com` is the name of a communication topology. The name of topology is necessary for the communication between branches of a parallel program.

IN `p` is the number of overlappings of boundary domains.

OUT `PBUF` is the name of the pointer to an array. In each computer, this name is associated with a memory space, dynamically selected by the function `Rmas_mpi()` for the distributed domains.

OUT `rdims` is the one-dimensional integer `ndims` array, whose elements are the dimensionality of the `IBUF` array along the corresponding coordinates: `IBUF[0]` is the size along the coordinate 0, `IBUF[1]` – along the coordinate 1, etc.

OUT `gran` is the one-dimensional `ndims` array, whose elements are the MPI-types of overlapped boundaries of the domains: `gran[0]` – the MPI-types of the boundary along the coordinate 0, `gran[1]` – along the coordinate 1, etc.

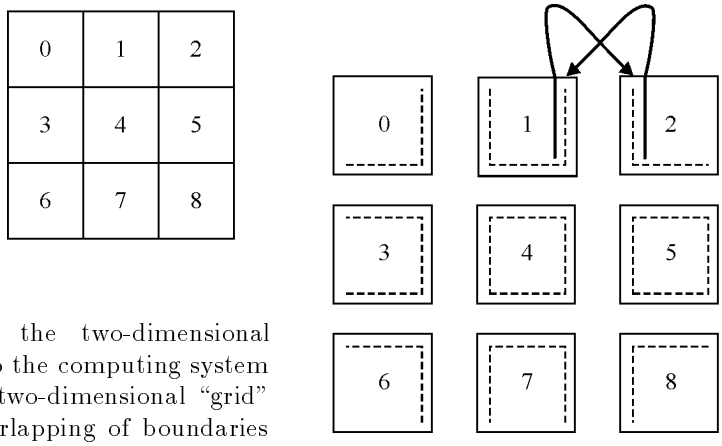
OUT `disp` is the one-dimensional integer `2*ndims` array, whose elements are the displacements of the boundaries within the domain. This parameter is necessary for communication functions for an exchange of domain boundaries between branches of a parallel program.

OUT `left_right` is the one-dimensional integer `2*ndims` array, whose elements are the serial numbers of the neighboring computers along each coordinate in the Cartesian structure. For example, `left_right[0]`

and `left_right[1]` are the neighboring computer numbers along the coordinate 0, towards decreasing and increasing computer serial numbers (number `MPI_PROC_ NULL` if there is no neighboring computer); `left_right[2]` and `left_right[3]` are the neighboring computer numbers along the coordinate 1, towards decreasing and increasing computer serial numbers, etc.

The values of the output array elements – `gran`, `disp` and `left_right` are sought by the function `Rmas_mpi()` for each computer individually. These arrays are necessary for communication functions to carry out an exchange of domain boundaries.

Decomposition of the `IBUF` array to domains according to the parameters ndims, com and p and distribution of these domains among the computers are executed by this function. An example of the decomposition of the two-dimensional array to domains and distribution of the domains throughout the two-dimensional computer grid is shown in the figure below. The overlapping domains are marked with dotted lines.



Decomposition of the two-dimensional computing space to the computing system memory with the two-dimensional "grid" topology, with overlapping of boundaries

Further we describe a communication function, realizing an exchange of the overlapping boundaries domains between the computers. This exchange is realized at the end of each computing iteration. This function takes into account a corresponding computer location in the Cartesian topology.

```
Mgr_mpi(void* PBUF, int ndims, int* rdims, MPI_Datatype* gran,
        int* disp, int p, int* left_right, MPI_Comm com,
        int* f_obp)
```

The parameters of this function: `PBUF`, `rdims`, `gran`, `disp`, `left_right`, `com` are output parameters of the function `Rmas_mpi()`. Parameters `ndims` and `p` are the same as input parameters of the function `Rmas_mpi()`. In the given communication function, all the parameters, except for the last one,

are the input parameters (IN) and their description is given in the function `Rmas_mpi()`.

OUT `f_obp` is the logical variable, whose value is assigned by the function `Mgr_mpi()` depending on the generalized condition of termination of loops (in all the computers). If in all the parallel program branches `f_obp` = 1, a parallel program can terminate loops, otherwise the computing iterations should be continued. However the condition of termination of work of each computer is an unconditional execution of the condition `f_obp` = 1 in all the computers.

An example of a program fragment of the solution to the Poisson equations in the two-dimensional domain by the Seidel method using the above-described communication functions is as follows.

```
#define Mndims 2 // The dimension of computing space and domains
#define Pndims 2 // The dimension of the Cartesian system topology
#define Mn 100   // The size of computing space along the coordinate X
#define Mm 200   // The size of computing space along the coordinate Y
#define P0 4     // The size of the Cartesian system topology along
                 // the coordinate 0
#define P1 4     // The size of the Cartesian system topology along
                 // the coordinate 1
double *IBUF;    // The pointer to the initial computing domain
double *PBUF;    // The pointer to the computing domain in each
                 // computer
main (int argc, char **argv)
{ int Mdims[Mndims]; // The sizes of the initial domain along
                           // the coordinates
  MPI_Comm grid;     // The name of a computing system structure
  MPI_Datatype gran[Mndims]; // Types of overlapping boundaries
                             // of domains
  int disp[2*Mndims];    // Displacement of boundaries
  int left_right[2*Mndims];  // Serial numbers of the neighboring
                             // computer
  int Rpdims[Mndims];    // The sizes of distributed domains along
                         // the coordinates
  int p = 1;             // The number of overlappings
  int flag;              // The logical variable for a generalized
                         // conditional transition
  int Pdims[Pndims];     // The sizes of the Cartesian structure
                         // along each coordinate
  int Perid[Pndims];     // The ringed Cartesian structure along
                         // the coordinates
```

```
int size;                // The size of a computing system
int rank;                // The serial computer number
int i, j, F1, Fi, Fj;
double x = 0.01, y = 0.01; // The grid steps of computations
Mdims[0] = Mn;
Mdims[1] = Mm;
Init(&argc, &argv);      // The initialization of the MPI library
/* The determination of the computing system size */
MPI_Comm_size(MPI_COMM_WORLD, &size);
/* dims array is zeroed and Perid array is filled in */
for(i=0;i<Pndims;i++) { Pdims[i]=0; Perid[i]=0; }
/* dims array is filled in, where the size of a network is determined */
MPI_Dims_create(size, Pndims, Pdims);
/* The creation of the topology "grid" with the communicator-grid */
MPI_Cart_create(MPI_COMM_WORLD,Pndims,Pdims,Perid,1,&grid);
/* Each branch of the parallel program defines a serial number in
   the grid*/
MPI_Comm_rank(grid, &rank);
/* Zero branch of the parallel program determines the memory for the
   initial computing domain and initiates its boundary values */
if(rank == 0)
{ IBUF = (double *)calloc(Mn*Mm, sizeof(double));
  /* The boundary values initialization of the IBUF domain */
  .........
}
/* Further all parallel program branches work again. */
/* Recall that Rmas_mpi() is a collective function.
   All the parameters are given. */
Rmas_mpi(IBUF, MPI_DOUBLE, Mndims, Mdims, grid, p, PBUF,
         Rpdims, Gran, disp, left_right);
/* Further follows the main iterative loop. All the parallel program
   branches do computations in these domains */
flag = 0;
while (flag == 0)
  { for (i = 1; i < Rpdims[0]-1; i++)
    for (j = 1; j < Rpdims[1]-1; j++)
    { F1 = PBUF[i][j];
      Fi = ((PBUF[i+1][j] + PBUF[i-1][j]))/x_2;
      Fi = ((PBUF[i][j+1] + PBUF[i][j-1]))/y_2;
    }
    /* The condition of the iterative process convergence is checked */
    if ("Was the convergence condition executed?") flag = 1;
    else flag = 0;
```

```
        /* Exchange of the domain boundaries is realized between parallel
           program branches and a generalized condition of termination
           of the parallel program execution is done */
        Mgr_mpi(PBUF, Mndims, Rpdims, gran, disp, p, left_right,
                Grid, &flag);
    }
}
```

From the considered example it becomes clear that the decomposition of the initial computing space and the exchange of boundary domains have been simplified in a maximum possible way.

References

[1] Snir M., Otto S.W., Huss-Lederman S., Walker D., Dongarra J. MPI: The Complete Reference. – Boston: MIT Press, 1996.

[2] Dongarra J., Otto S.W., Snir M., Walker D. An Introduction to the MPI Standard. – January 1995. – (Technical report / University of Tennessee; CS-95-274);

[3] Malyshkin V.E., Vshivkov V.A., Kraeva M.A. About realization of the method of particles on multiprocessors. – Novosibirsk, 1995. – (Preprint / RAS. Sib. Branch. Comp. Cent; 1052) (in Russian).

[4] Evreinov E.V., Kosarev Yu.G. High Efficiency Homogeneous Universal Computing Systems. – Novosibirsk: Nauka, 1966.

[5] Mirenkov N.N. Parallel Programming for Multimodular Computing Systems. – Moscow: Radio i Svyaz, 1989 (in Russian).

[6] Malyshkin V.E. Linearization of mass calculations // System Computer Science / Ed. V.E. Kotov. – Novosibirsk: Nauka, 1991. – № 1. – P. 229–259.

[7] Valkovskiy V.A., Kotov V.E., Marchuk A.G., Mirenkov N.N. Elements of parallel programming. – Moscow: Radio i Svyaz, 1983 (in Russian).

[8] Valkovskiy V.A., Malyshkin V.E. The Synthesis of the Parallel Programs and Systems on Computing Models. – Novosibirsk: Nauka, 1988 (in Russian).

[9] Korneev V.D. A system and methods of programming of multicomputers on an example of the computer complex PowrXplorer. – Novosibirsk, 1998. – (Preprint / Russ. Acad. Sci. Sib. Branch. Inst. Comp. Math. and Math. Geoph; 1123) (in Russian).

[10] Korneev V.D. Parallel algorithms deciding the task of linear algebra. – Novosibirsk, 1998. – (Preprint / Russ. Acad. Sci. Sib. Branch. Inst. Comp. Math. and Math. Geoph; 1124) (in Russian).

[11] Korneev V.D. Parallel Programming in MPI. – Novosibirsk: Russ. Acad. Sci. Sib. Branch, 2000.