An interface for a mapping software under PVM

J.M. Lépine and Franck Rubi

rubi@labri.u-bordeaux.fr

LaBRI, Laboratoire Bordelais de Recherche en Informatique, Université Bordeaux I, 351 Cours de la Libération, 33405 Talence, France.

1 Introduction

The work described is this paper is carried out at the "Laboratoire Bordelais de Recherche en Informatique" which is taking part in the LHPC[1] software development projects [4]. The aim of this work is to provide a mapping software. This mapping tool is based upon the experience gained in developing the ADAM programming environment [2].

For several classes of applications, it is possible to know or to compute information about the parallel execution before the execution. These applications are modeled as an *interprocess communication graph* (ICG) where the vertices represent the parallel processes and their weights represent known or estimated computation costs of the processes. The edges represent communication requirements between processes and their weights reflect the relative amounts of interprocess communication involved. We can use this graph to obtain a "good" mapping, i.e. a mapping which tends to balance the computing load of the processors and to share out communication among the physical links still using minimal length paths.

PVM [3] maps processes on request (spawn function) without knowing the future communications between processes. To take into account this communication information, the user must write a mapping program which specifies to the PVM spawn function the target processor for each process. We propose an extension of PVM allowing an algorithmic description of the *interprocess communication graph* (ICG) and the use of static mapping functions. After the mapping, the application runs normally under PVM.

The organization of the paper is as follows. Section 2 is a brief survey on static mapping. We describe the added functions allowing the description of the interprocess communication graph in Section 3. Section 4 contains information about the technique used to map and run the application under PVM. We give a code example in Section 5. This is followed by a conclusion in Section 6.

2 The static mapping problem

A parallel program can be modeled by the *interprocess communication graph* (ICG). A parallel architecture is also modeled by an undirected connected graph where vertices represent processors and edges represent communication links between processors. The mapping problem can be defined by a function wich assigns each process to a processor. A cost function which associates a real value to each mapping must be defined to compare different possible solutions. Two contradictory allocations have been considered:

- the first minimizes the sum of the total communication costs between processors;

- the second minimizes the load imbalance accross the parallel system.

It is well known that even simplified versions of this problem fall into the NP-hard complexity category. Several heuristics approaches have been proposed which tend to obtain satisfactory sub-optimal solutions in a reasonable amount of time [8] : greedy algorithms, iterative algorithms, simulated annealing technique, genetic algorithms, graph theory heuristics, ...

[1] Laboratory for High Performance Computing set up in Lyon between the "Laboratoire de l'Informatique du Parallélisme" and Archipel.

Mainly, we focus on a two phases iterative algorithm [7] for both its speed and the quality of the results. In the first phase, the processes are grouped so that the number of groups equates the number of processors. Then groups are mapped in the last phase. We use a *mapping cost function* that allows to take several criteria into account: the computation load of the processors, the cost of communication between processes on the same processor, the communication load of physical links (link contention), the length of the routage.

Our mapping software allows the use of two phases method, but also of other heuristics like greedy or Bokhari algorithms. It is easy to add news heuristics. The user can choose an adequate mapping method for his (her) application.

3 Algorithmic description of the Interprocess Communication Graph

The structure of an application is divided in two parts. The first part contains the description of the code of the processes (C and PVM functions). The second part, called the initialization part, contains the algorithmic description of the ICG. This description uses two new functions: one function for the edge creation and another for node processes.

- Process declaration:

  Each node process of the ICG must be declared with the *map_process* function. This function has same paramaters as *pvm_spawn*. Only a cost parameter is added; this value is used by the mapping cost function. The *pvm_spawn* fonction returns for each process a PVM process identifier (**tid**). The *map_process* function returns a map process identifier (**mid**). The **mid** and the **tid** are different. When the *map-process* function returns, the **tid** is not known; the process is only created under PVM after the full description of the ICG and the mapping computation.

  With the flag parameter, the user may choose the target processor for a process or leave the mapping free to compute it.

- Edge declaration:

  Each edge of the ICG must be declared with the new *map_connect* function which has three parameters: the **mids** of the two node process edge ends and a **cost edge** which will be used during the computation of the *mapping cost function* value.

With these two functions, multiple occurrences of the same process with different neighbours and any ICG can be created. This graph is not constrained by the topology of the processor network.

4 Mapping and execution under PVM

After fully describing the ICG, the user can launch the mapping software by calling a new function (*pvm_map*). He chooses a heuristic method (by example the two phases method). The mapping is computed and the processes are automatically launched under PVM with the *pvm_spawn* function. The mapping software provides statistics so that the user can esteem the quality of the resulting mapping.

To communicate, a node process needs to know the tids of its neighbour processes. Most applications begin by exchanging their tids for future communication. Generally, for large applications, this critical part is not very easy to write. In our case, the software knows the ICG and the tid of each node process. It is possible with a minimum of PVM communication to automatically send to each node process the tids of its neighbours. At the beginning of the execution of the node process, we call a new function (*pvm_neighbour*) to receive these tids. This method seems to be the simplest. All information is taken in the ICG. The only thing to write is a sequential code for the ICG description (easier to debug than the classical exchange tid part of a parallel execution).

5 A sample application

In this section, we give an illustration of the constuction of an ICG in the case of a fictional application. The ICG to describe is a toric grid. The length and the width of the grid are parameters of the main function. All the node processes are similar and they only print the tids of their neighbours. The value used for the mapping cost function is the same for each object (node or edge).

We show below the essential part of the C codes. Source 1 is the ICG description and Source 2 is the simplest node code.

— SOURCE 1 : ICG.c —

```
/* ICG toric grid description  */
/* usage : grid <length> <width> */

main(argc,argv)
{
  int   length,width, number;
  int   i,j;
  int   COST = 1;
  char  **argv;

/* set the length and width variables*/
  sscanf(argv[1],"%d",&length);
  sscanf(argv[2],"%d",&width);

/* compute the number of node */
  number = length * width ;

/* allocate an array for mid identifier*/
/* mid[i][j] for the node process i,j  */
  mid = (int *) malloc (number * (sizeof(int *)));
  [...]

/* declare node process line by line */
  for (i=0;i<length;i++)
   for (j=0;j<width;j++) {
     /* set argv for this node */
     argv = [...]
    /* call map_process for the node i,j */
     map_process(execfile,argv,0,NULL,1,&mid[i][j],COST);
   }

/* declare horizontal edges  */
/* connect (i,j) with (i,j+1)*/
  for (i=0;i<length;i++)
    for (j=0;j<width;j++)
       map_connect(mid[i][j],mid[i][(j+1)%width], COST);

/* declare vertical edges    */
/* connect (i,j) with (i+1,j)*/
 for (i=0;i<length;i++)
    for (j=0;j<width;j++)
       map_connect(mid[i][j],mid[(i+1)%length][j], COST);

/* ask software mapping        */
/* choosen heuristic : Bokhari */

  pvm_map (Bokhari);

/* when pvm_map returns, mapping is achieved */
/* node processes are running under PVM      */

[...]

}
```

— SOURCE 1 : ICG.c —

— SOURCE 2 : node.c —

```
/*    node process   */
/* receive in argv
/* - the length and width of the grid   */
/* - its location (i,j) in the grid     */
main(argc,argv)

{
int   length,width;
int   my_tid;
int   i,j;
int   neighbour[4];

/* enroll process in PVM */
my_tid = pvm_mytid();

/* set length and width  */
sscanf(argv[1],"%d",&length);
sscanf(argv[2],"%d",&width);

/* set the location      */
sscanf(argv[3],"%d",&i);
sscanf(argv[4],"%d",&j);

/* receive in an array         */
/* neighbour tids              */
/* East, South, West and North */

pvm_neighbour(neighbour);


printf("my tid is      : %d \n",my_tid);
printf("my location is : (%d,%d) \n",i,j);
printf("East  tid : %d \n",neighbour[0]);
printf("South tid : %d \n",neighbour[1]);
printf("West  tid : %d \n",neighbour[2]);
printf("North tid : %d \n",neighbour[3]);

/* leave pvm */
pvm_exit();

}
```

— SOURCE 2 : node.c —

6 Conclusion

In this paper, we propose an extension to PVM allowing the use of static mapping technics in a PVM environment.

Today, many heuristics for the static mapping problem exist and give "good" solution in a reasonable amount of time. Taking into account the load of the processors but also the communication dependencies between processes allows a better mapping.

The class of problems where it is possible to describe the Interprocess Communication Graph is large. For example, we use our mappping software for a parallel implementation of a block Cholesky factorisation based on nested dissection method [5].

The description of the Interprocess Communication Graph is not a hard work because, in most cases, the user had already the knowledge of this graph. Furthermore, this graph can be used for other tools like debugging mechanisms [6]. An extension to other message passing environments like MPI (Message Passing Interface [1]) is also possible.

References

[1] Document for a standard Message Passing Interface (MPI). Technical report, Message Passing Forum Interface, November 1993.

[2] M. Alabau, S. Chaumette, M.C. Counilh, J.M. Lépine, J. Roman, F. Rubi, and B. Vauquelin. A programming environment dedicated to a model of explicit parallelism. In *Workshop on environments and tools for parallel scientific computing in Parallel Computing Series*, volume 6, pages 193–212. North-Holland, 1993.

[3] A. Beguelin, J. Dongarra, G.A. Geist, R. Manchek, and V. Sunderam. A users's guide to PVM Parallel Virtual Machine. Technical Report TM–11826, Oak Ridge National Laboratory, August 1993.

[4] L Brunie, S. Chaumette, M. Cosnard, F. Desprez, L. Lefevre, M. Loi, M. Pouzandi, B. Tourancheau, and X. Vigouroux. The LHPC Programming Environment. In *Workshop on Environments and Tools for Parallel Scientific Computing*, 1994. To be published by SIAM.

[5] P. Charrier and J. Roman. Partitioning and mapping for parallel nested dissection on distributed memory architectures. In *Parallel processing: Conpar92-VAPP V, LNCS 634*, pages 295–306. Springer Verlag, 1992.

[6] S. Chaumette. Neutral observation of logical interprocess communication links in PVM applications. In *1994 PVM Users' Group Meeting, Oak Ridge, Tennessee*, May 1994. Also available on netlib.

[7] J. M. Lépine and F. Rubi. Allocation de ressources pour les applications massivement parallèles. Technical Report 92-79, LaBRI, Université Bordeaux 1, September 1992.

[8] T. Muntean and E. Talbi. Static allocation of communicating processes on parallel architectures. *TSI : Technique et Science Informatiques*, 10(5):365–370, 1991.