Commission of the European Communities

****

ESPRIT III

PROJECT NB 6756

****

CAMAS

COMPUTER AIDED MIGRATION OF APPLICATIONS SYSTEM

****

CAMAS-TR-2.1.3.1
MAP Position Paper

****

Date: March 1994

Review 3.0

ACE - U. of Amsterdam - ESI SA - ESI GmbH - FEGS - PARSYTEC - U. of Southampton.

Technical Report CAMAS-TR-2.1.3.1

MAP Position Paper

J.F. de Ronde, M. Bergman and P.M.A. Sloot
University of Amsterdam

March 1994

1 Introduction

An important issue in parallel computing is the efficient distribution of a set of tasks (i.e. data and instructions) over a set of processors. An optimal solution would be that each of the processors is kept busy doing useful work as much as possible, while at the same time the communication between the processors is minimized. This is equivalent to minimizing the overall execution time of the set of tasks on a set of processors. The problem of mapping a set of tasks onto a set of processors is known to be *NP* complete. This implies that in general it is not possible to find the optimal solution within polynomial time by deterministic methods such as recursive bisection.

For an efficient distribution of a collection of tasks over a collection of processors two requirements are to be met. On the one hand a situation is desired in which the workload is equally divided over the processors. When for a distribution only a few processors are active while the rest is idle, the capabilities of a parallel computer are not fully utilized. On the other hand the communication between the processors should be minimal. When a lot of communication has to take place between processors this will slow down the execution of an application. Situations in which processors are frequently waiting for results from other processors should be avoided.

For many parallel applications the above mentioned requirements are conflicting goals. Distributing tasks over a lot of processors automatically involves communication between the processors.

An extreme example of minimizing the communication is to map all tasks to a single processor. In this case there is no communication at all, but no parallelism is exploited either. It is therefore necessary to find a good trade-off between the amount of communication and calculation within the task mapping.

In general a program is a set of processes that have a partial ordering. The precedence between these processes can be represented by an acyclic directed graph that connects the processes (precedence or temporal process graph). The load balancing problem in general is the following optimization problem: schedule the processes of the temporal process graph on the processors of the parallel machine so as to minimize the overall execution time.

A data parallel algorithm is based on distributing the problem's data items among the processors. The algorithm consists of a sequence of calculations, each of which is executed by every processor on the data assigned to it. The values of variables that have been modified in calculations have to be communicated to the processors that need them as input to the next calculation phase. For this situation one can define the load balancing problem in a simpler manner: distribute the processes (or the corresponding data) among the processors in order to minimize the execution time for one calculation phase. This is referred to as

the *restricted load balancing problem*. This means that the more general load balancing definition is replaced by a series of simpler load balancing problems. This formulation is essential in adaptive problems (where the data decomposition changes throughout the program execution) where the precedence graph is not known in advance.

The SPMD (single program multiple data) programming paradigm is the framework in which we consider the implementation of data parallel programs. This paradigm has enough generality to be able to express homogeneous as well as heterogeneous problems in it.

It must be noted that the overall execution time that is obtained using a restricted load balancing technique can only be equal to or larger than the minimal time for the general load balancing problem since the space of feasible solutions is restricted.

In the CAMAS project we are essentially interested in the restricted load balancing problem, i.e. the mapping problem for data parallel programs. The DDT(see [1]) subtask in CAMAS deals with the problem of decomposition of an irregular mesh into a set of partitions that correspond to equal workload size. The MAP subtask will be concerned with the problem of mapping partitions on a processor topology in an optimal manner. The output that is generated by the DDT tool may serve as input to MAP.

For an excellent overview of the mapping problem see for example the overview article of Norman *et al.* [2].

2 Data parallel computations modelled

When one wants to apply or test a mapping technique one needs two abstract models, namely, a model of the target multicomputer and a model of the parallel computation. Obviously an abstraction is not able to emphasize every feature within the multicomputer or application. Consequently quantitative results obtained using the models strictly apply if the actual multicomputer and the actual computation correspond to the models. In general it needs to be validated if one is allowed to apply results that have been obtained using models to realistic problems. PARASOL and SAD respectively serve as the abstractions for machine and application within the CAMAS workbench.

The load balancing problem for data parallel applications can be approached in the following manner. A given data structure can be subdivided into units. These units are considered to be atomic during the parallel execution. The correspondence of an abstract "unit" and some set of data elements within a real application is defined by the user. For example in grid oriented applications units can correspond to grid partitions (of arbitrary size). The data dependencies between the units can be represented by a unit interconnection graph. Given a process that is responsible for performing the calculations corresponding to a unit. It must have access to the information associated with the calculations in this unit. Therefore, the execution of this process must be preceded by the receipt of the boundary information from neighbouring units.

Any data parallel program can be written as a sequence of calculation and communication steps. In terms of a pseudo description:

```
start data parallel program
        for every calculation step
            for all units send neighbour information
            for all units receive neigbour information
            for all units calculate
        end for
end program
```

It should be noted that only boundary information is exchanged between neighbouring units that reside on separate processors.

3 Mapping algorithms and decomposition

The restricted load balancing problem that must be solved in each phase consists of finding a good distribution of the problem data. This distribution procedure can be seen as consisting of two separate phases. The first phase is concerned with the decomposition of the data domain in a set of units that each represent a specific workload and which have connectivities with respect to necessary information exchange. The second phase can be seen as the procedure of assigning units to processors. This is referred to as the mapping procedure.

Many heuristic approaches to partitioning exist. A well-known strategy is based on recursive 2-way partitioning of the data set. The application must provide one primitive: the split-operation or partitioner, which splits a unit in several new ones. Suppose that the data decomposition is given. A good load balancing distribution is found by minimizing the estimated parallel execution time of a phase. This can be done by explicitly modelling the program execution time with an (approximate) cost function (like SAD) [5]. A graph that represents the units in one phase and their mutual dependencies can be represented by an interconnection matrix. The diagonal elements in this matrix are equivalent to the workload associated with individual units. The off diagonal elements represent message sizes that have to be exchanged between neighbouring units due to data dependencies.

An equivalent connectivity matrix can be created for a processor interconnection graph. Now, the off diagonal elements represent the average cost of sending a standard size message from one processor to another. This information can be obtained using PARASOL [6]. A cost function can then be derived that estimates the execution time given a specific data distribution. In the MAP task the procedure sketched above will be investigated explicitly dedicated to irregular mesh systems (Finite Element grids).

4 MAP in CAMAS

4.1 Grid partioning

In the above section the mapping problem in general and restricted to data parallel problems has been described. In the MAP task in CAMAS research is planned on a specific subject, namely, Finite Element grids. A Finite Element grid that consists of shells generally consists of quadrilaterals and trilaterals. The workloads that are associated to calculations concerning one element can be expressed in terms of a set of high level language statements (Fortran generally).

The most obvious manner to establish a data decomposition for Finite Element calculations is to partition the grid along element boundaries (figure 1). This has as a result that only infromation at the boundaries of subdomains would have to be exchanged after each calculation phase for units that reside on separate processors (since the element calculations are of local nature).

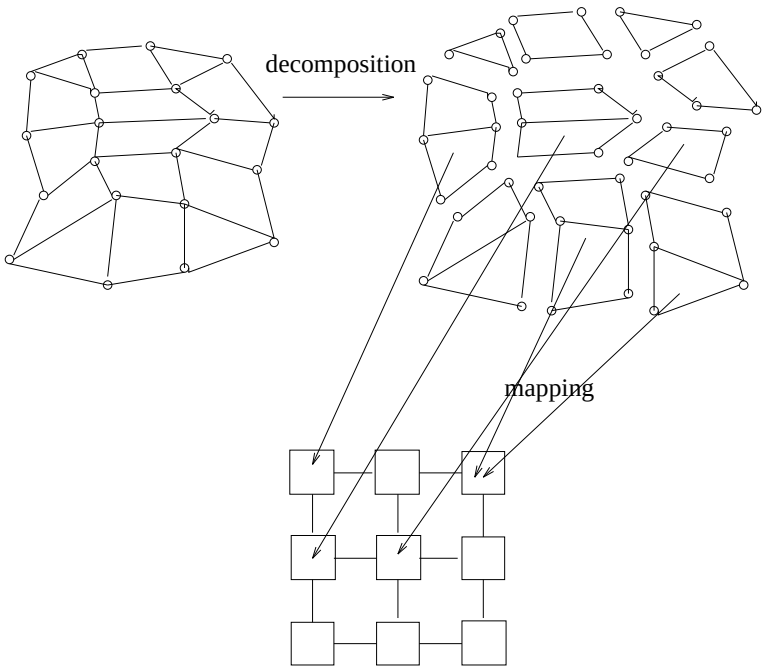

Figure 1: the process of decomposition and mapping for a mesh.

To keep consistent with the terminology introduced above with respect to Finite Element grids we define the following:

- A mesh can be subdivided into several units (as in figure 1). The minimum size of a unit is one element. The maximum size is the whole data structure.
- The sizes of messages (e.g. for nodal point data in Finite Element grids) that have to be sent between elements on separate units are directly proportional to the size of the boundaries between them. (Obviously only in the case that separate units are stored on different processors actual messages will have to be sent.)

4.2 Workload graph

From a Finite Element grid (given element connectivities) one can derive a workload graph in which every vertex corresponds to the workload stuck to a specific element whereas the edges point out the amount of knowledge that neighbouring elements have to exchange in a communication phase.

In a decomposition procedure one divides up such a graph in a number of units (see figure 3).

The connectivity graphs can be stored in connectivity matrices. On the diagonal of these matrices the work that will be done on specific elements or units is stored. This workload can be stated in terms of actual costs on a real machine when the time complexities of the calculations are known (derived with help from SAD and PARASOL). The figure 2 shown below explains this pictorially for a simple situation. There are 3 tasks of equal load W. Between some tasks equal messages have to be exchanged of size m. From this the connectivity matrix next to the graph can be derived.

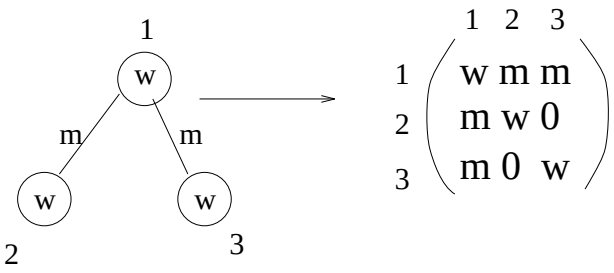

Figure 2: a simple workload graph with corresponding connectivity matrix

The edges in the graphs are stored on the off diagonal elements of the connectivity matrix. They correspond to actual message sizes that have to be sent between elements or units and can for real problems be specified in number of bytes. For example for an explicit Finite Element code like PAM-CRASH in one message phase per nodal point on the boundaries the velocity in three cartesian directions has to be sent. For a boundary of size 10 (10 nodal points) this would mean that a message of size 10 * sizeof(real) * 3 bytes has to be exchanged between units that share this boundary.

An edge in a connectivity graph generally has to be handled as uni-directional. Therefore actually two edges exist between neighbouring elements or units. One edge weight corresponds to the message size in one direction the other edge corresponds to the message size in the opposite direction. Very often the weight of the edges in both directions is equal and can therefore be represented as single value. The corresponding connectivity matrix then is symmetric.

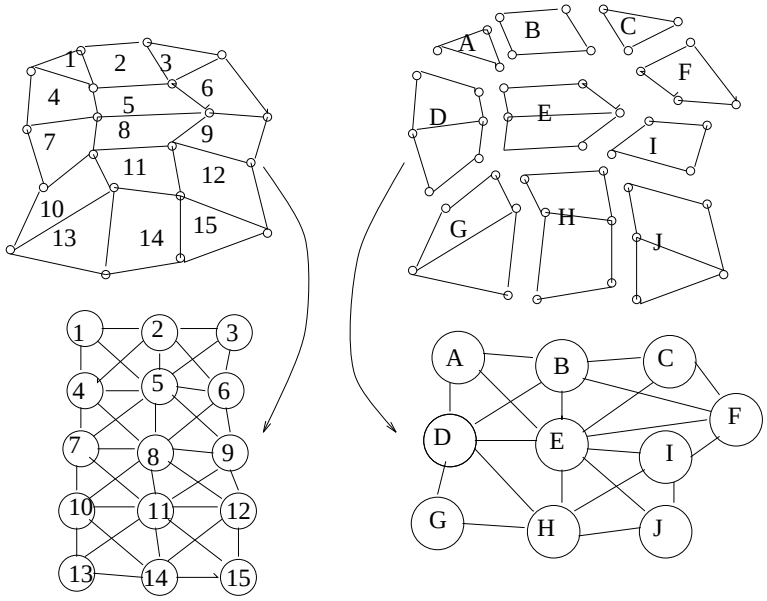

Figure 3. Connectivity graph for original grid vs. connectivity graph for units.

4.3 Processor graph

For a multiprocessor system a same type of connectivity graph can be drawn in which the vertices denote the relative performance of corresponding processors and the edges represent a cost for sending a standard amount of information (e.g. a byte) across the link between processors. The way that message size influences the cost of sending a message over a processor to processor link is highly dependent on things such as network contention,

routing (random or deterministic) and network type.

In a first approximation the costs associated to sending a message grows linearly with message size. PARASOL can give a more detailed description of the parallel machine of our interest. Using this the processor connectivity matrix can be filled with realistic values.

5 Mapping Algorithms

Finding a good mapping of elements of a FE grid is a difficult task. One may e.g. consider the problem of mapping individual elements to processors using stochastic optimization algorithms such as Simulated Annealing or Genetic Algorithms.

Alternatively, one may use a recursive method to obtain a decomposition in a set of units. Then an stochastic optimization algorithm can be applied to the graph of units.

The assigment of N units to P processors is a discrete optimization problem. The solution space contains in the order of P^N solutions. It is clear that such a solution space very soon grows out of proportion for exact solution techniques (e.g. branch and bound algorithms).

In MAP the applicability of deterministic and nondeterministic mapping algorithms for Finite Element grid to processor mappings described by connectivity matrices expressed in SAD and PARASOL will be investigated.

5.1 Costfunctions

Given two matrices W_{ij} (representing workloads and connectivities) and P_{kl} (representing processor performances and processor to procesor links). It is important to define suitable costfunctions that as accurately as possible model the total execution time of the parallel computation on the multicomputer.

Examples of such costfunctions are (see for example [4]:

$$C = \max_{q \in \mathcal{Q}} \sum_{u_i \in \mathcal{U}^q} W_{ii}$$

where $\mathcal{Q}$ is the set of processors and $\mathcal{U}^q$ is the set of units u_i residing on processor q.

This costfunction only takes workloads into account. Thus basically it represents a model of a fully connected machine with infinite communication bandwidth.

A more realistic costfunction might be:

$$C = \max_{q \in \mathcal{Q}} \left(\sum_{u_i \in \mathcal{U}^q} W_{ii} + \max_{u_i \in \mathcal{U}^q, u_j \in \mathcal{A}(u_i)} P_{ij} W_{ij} \right)$$

where $\mathcal{A}(u_i)$ is the set of neighbouring units for unit u_i.

In this case the costfunction takes into account the connectivities in the workload graph as well as the processor topology via the factor P_{ij}. This will force the maximum distance between the processors onto which connected units are mapped to be minimal while at the same time for optimizing the workload distribution.

It should be noted that the multiprocessors in the above two models are assumed to consist of processors that have identical performance rate. That's why the factor describing this rate P_{ii} can be omitted in the costfunctions.

The form of the costfunctions can be made as complex as desired. But it is necessary to find a model that on the one hand has an appropriate accuracy for the mapping purpose as well as that it's evaluation is relatively cheap.

6 Time schedule for MAP

The central issue in MAP is the development of optimization techniques for mapping work associated with irregular finite element grids on a processor topology in a near optimal way. For this purpose non deterministic optimization algorithms will be investigated. Furthermore investigation on parallelization strategies for the such optimization techniques and an implementation of a parallel optimization algorithm for the mapping problem is intended. The parallelization becomes specifically important when systems are concerned that need adaptive remeshing and thus redistribution of data elements during execution. For such problems an efficient mapping algorithm has to be embedded in the application itself.

The optimization techniques will be applied to purely finite element grids where the connectivities are element-element based as well as to unit connectivity graphs. Such graphs can be obtained by applying an initial decomposition algorithm to a finite element grid which splits up the grid in units. Such an algorithm may be based on a deterministic method such as recursive bisection.

A study is planned that looks at the differences in efficiency behaviour for combination of deterministic and indeterministic methods (decompostion and mapping). Furthermore the quality of results obtained from combination methods can be compared to direct application of non deterministic methods.

- Months 18 - 24 Development of an optimization algorithm (either SA or GA) for the mapping problem of finite element graphs on processor graphs. Firstly mechanisms have to be developed that allow for the generation of mappings. The algorithm will be implemented as a black box module that accepts connectivity graphs in a certain format and provides as output a near optimal mapping for the specific system. Examples of candidate finite element grids can be supplied by the industrial partners in the CAMAS project.

- Months 24 - 30 Interfacing MAP with FAM, PAM, DDT and PARASOL-SAD. Parallelization of the selected non deterministic solver for the mapping problem. Furthermore the applicability of combination of deterministic and non deterministic methods will be investigated. When useful a specific combination of such methods will be implemented.

- Months 30 - 33 are dedicated to the release of a module that takes as input (according to a specified format) a finite element mesh and a processor topology (specified with the workbench tools) and provides a near optimal mapping.

7 References

1. N. Floros *et al.* Domain Decomposition Tool (DDT). CAMAS Workpackage 2.2.2. (Year 1 Report). September 1993.
2. M. G. Norman *et al.* Models of Machines and Computation for Mapping in Multicomputers. ACM Computing Surveys, Vol. 25, No. 3, pp 263-302. September 1993.
3. H. D. Simon. Partitioning of Unstructured Problems for Parallel Processing. Computer Systems in Engineering, Vol. 2, No. 2/3, pp. 135-148, 1991.
4. J. de Keyser *et al.* Load Balancing data parallel programs on distributed memory computers. Parallel Computing, Vol. 19, pp 1199-1219, 1993.
5. J.F. de Ronde *et al.* CAMAS-TR-2.2.4.2 SAD/PARASOL II progress report. September 1993.
6. P.M.A. Sloot *et al.* CAMAS-TR-2.1.1.4. PARASOL I progress report. March 1994.