Linear Programming for Mesh Partitioning under Memory Constraint : Theoretical Formulations and Experimentations

Sébastien Morais[1,2], Eric Angel[1], Cédric Chevalier[2], Franck Ledoux[2], and Damien Regnault[1]

[1] IBISC Laboratory, University of Evry Val d'Essonne, France
[2] CEA, DAM, DIF, F91297 Arpajon, France

In many scientific areas, the size and complexity of numerical simulations lead to make intensive use of massively parallel simulations on High Performance Computing (HPC) architectures. Such architectures are mainly modelized as a set of processing units (PU) where memory is distributed. Distribution of simulation data is crucial: it has to minimize the computation time of the simulation while ensuring that the data allocated to every PU can be locally stored in memory.

In this work, we focus on numerical simulations using finite elements or finite volumes methods, where physical and numerical data are carried on a mesh. The computations are then performed at the cell level (for example triangle and quadrilateral in 2D, tetrahedron and hexahedron in 3D). More specifically, computing and memory cost can be associated to each cell. Depending on the numerical scheme to apply, a graph or an hypergraph representation of the mesh is built to perform partitioning. Such a representation is then used by tools like Metis, Patoh, Scotch or Zoltan to distribute the mesh. Traditional approaches consist in balancing the computing load between PUs while minimizing:

- either the edge cut, or hyper-edge cut, between parts ;
- or the size of a vertex separator.

Such objective functions do not rely on an important characteristic of the numerical methods used in simulation codes. The computation performed on cell i requires data from adjacent cells. As a consequence, an usual approach is to duplicate some cells between PUs. Such cells are commonly called "ghost cells" and we obtain a partitioning with covering. Although it is assumed that edge cuts is proportional to the total communication volume, it is not [Hen98]. And, for identical reason the memory footprint of ghost cells is not explicitly taken into account while minimizing edge cuts.

Load balancing is usually achieved in two steps:

1. Cells are distributed according to a balance criterion, without taking care of ghost cells, to be reached;

2. Then ghost layers are built to allow the resolution of numerical scheme locally to every PU.

However, the partitioning obtained after phase (1) does not take into account the memory footprint of ghost cells added in phase (2). For 2D meshes the size of the edge cut between two parts grows as $O(n^{\frac{1}{2}})$ and for 3D meshes it grows as $O(n^{\frac{2}{3}})$, with n the number of cells. Then, distributing a mesh on a large number of processors can bring the simulation to break off due to a lack of physical memory on a PU.

In this context, we propose a new approach for the mesh partitioning on k parts, which takes into account ghost cells and the memory constraint on each PU. We formalize this new problem by means of integer linear programming [Dan63]. In this way, we obtain a problem similar to `make_span` minimization in scheduling [CPW98], where the variables are: $x_{i,p}$ which is equal to 1 if the computations associated to the cell i are performed on part p and 0 otherwise; $y_{i,p}$ which is equal to 1 if the cell i is stored in memory on part p and 0 otherwise. The problem is:

$$
\begin{array}{llll}
\text{Function:} & \texttt{make\_span} & \min \; C_{max} & \\[1em]
\text{Constraints:} & \text{Assignment} & \displaystyle\sum_{p=1}^{k} x_{i,p} = 1 & \forall i \in \text{Mesh} \\[1em]
 & \text{Time} & \displaystyle\sum_{i \in \text{Mesh}} x_{i,p} \; c_{i,p} \leq C_{max} & \forall p \in [\![1,k]\!] \\[1em]
 & \text{Memory} & \displaystyle\sum_{i \in \text{Mesh}} y_{i,p} \; \omega_{i,p} \leq Mem_p & \forall p \in [\![1,k]\!] \\[1em]
 & \text{Ghost} & x_{i,p} \leq y_{i',p} & \forall (i,i') \text{ adjacent} \\
 & & & \text{and } \forall p \in [\![1,k]\!]
\end{array}
$$

This model optimizes the computation time and obtains the resulting partition, through the variables $x_{i,p}$, while ensuring that the memory size including ghost cells does not exceed the memory bounds of each processor, through the auxiliary variables $y_{i,p}$.

To compare our approach with existing solutions, we also modelize some classical combinatorial problems such as graph partitioning, hypergraph partitioning, and graph partitioning with vertex separator by using integer linear programming. Having these various problems in the same formalism supplies first elements of comparison that we complete by comparing their solutions on benchmarks.

References

[CPW98] B. Chen, C.N. Potts, and G.J. Woeginger. A review of machine scheduling : Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization*, 1998.

[Dan63] George.B. Dantzig. *Linear Programming and Extensions*, 1963.

[Hen98] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? *Lecture Notes in Computer Science*, 1998.