Current challenges for parallel graph partitioning and static mapping

François Pellegrini
INRIA Bordeaux Sud-Ouest and LaBRI
Université de Bordeaux
351, cours de la Libération, 33405 TALENCE, FRANCE
pelegrin@labri.fr

May 15, 2011

1 Introduction

Graph partitioning is an ubiquitous technique which has applications in many fields of computer science and engineering. It is used to help in solving domain-dependent optimization problems modelled in terms of weighted or unweighted graphs, where finding good solutions amounts to computing small vertex or edge cuts that balance evenly the weights of the graph parts. In the context of intensive numerical simulations, it is mostly used for domain decomposition, used by iterative or hybrid solvers, as well as for the computation of fill-reducing or block-preserving orderings required by direct or hybrid solvers.

For some problems that cannot be adequately represented in the form of graphs, the hypergraph model, although heavier to implement, allows one to define more accurate cut cost metrics. For instance, hypergraphs allow one to represent non-symmetric or non-square matrix patterns and, in the case of sparse matrix-vector products, hypergraph partitioning models more accurately data exchange than graph partitioning. Yet, the run time of hypergraph partitioning tools is currently much higher than the one of graph partitioning tools, which makes the latter still useful in this context.

Because problem sizes keep increasing, large problems graphs cannot fit in the memory of sequential computers, and cost too much to partition, leading to the development of parallel graph partitioning tools such as ParMeTiS or Jostle, or parallel hypergraph partitioning tools such as Zoltan. The Scotch project, carried out within the Bacchus team of INRIA Bordeaux – Sud-Ouest, is yet another attempt to address the parallel graph partitioning problem.

The advent of massively parallel, NUMA machines, represents a new challenge for software designers, in order for partitioning tools to scale up to hundred thousands of processing elements.

2 Design constraints

While the sequential version of the SCOTCH library makes very few assumptions on the nature of graphs to be handled, the only limitation regarding the fact that vertex and edge weights should be strictly positive integers, the parallel version required some early design decisions to be taken, which condition the ability of the software to handle some types of graphs. In particular, we assumed that distributed graphs are of reasonably small degree, that is, that graph adjacency matrices have sparse rows and columns.

Also, we want our software to run on any number of processes P, and produce any number of parts k, irrespective of the values of k and P. Tackling these issues in PT-SCOTCH requires data structures to decouple the target and execution architectures: a process can handle several target domains, and conversely a given target domain can be managed concurrently by several processes.

2.1 Challenges

For the sizes of problems that we consider, parallel processing is mandatory, for application software as well as for partitioning tools. In the context of Exascale computing, the roadmap of parallel partitioning tools is to be able to partition (hyper)graphs of more than a trillion vertices, distributed across a million processing elements, in any number of parts. The number of parts must be decoupled from the number of processors as partitioning tools are not likely to ever be run on the GPU units of hybrid architectures, because of their irregularity and lower computational complexity, while target computations are more likely to be so. Consensus is not yet achieved on the best methods to use in this problem space. It is most likely that current state-of-the-art local optimization algorithms will be replaced by more global algorithms such as diffusion-based and/or genetic algorithm methods, which are heavier albeit more scalable. Moreover, several specific challenges have to be overcome, which are discussed below.

2.1.1 Heterogeneity

The advent of massively parallel, heterogeneous machines imposes a higher burden on software development: all parallel codes must take into account the heterogeneity of the machines they are going to run on, and parallel partitioning software must also provide partitions (that is, domain decompositions) that reflect this heterogeneity.

The problem of assigning the communicating processes of a parallel program onto the processing elements of some heterogeneous computer system is referred to as static mapping in the literature (graph partitioning can therefore be seen as a subproblem of static mapping where the target architecture is homogeneous). In the SPMD context, it is equivalent to the distribution across processors of the data structures of parallel programs so as to maximize load balance with respect

to the different compute powers, and to maximize data locality by favoring local over remote communication.

In order to adapt to heterogeneous parallel architectures, partitioning tools must offer mapping and remapping capabilities, by integrating some knowledge on the hierarchy and topology of the parallel machines for which partitioning is requested. Sequential tools already exist, parallel tools have to be created and/or adapted.

While graph mapping is already well studied, hypergraph mapping has not yet been considered to our knowledge. This should be a direction for future research.

2.1.2 Dynamicity

Since large scale simulations are most likely to involve remeshing and/or data redistribution, dynamic data migration and load balancing features are mandatory for application software designers. It is therefore necessary to provide parallel repartitioning capabilities, which should be available both for graphs and hypergraphs. While parallel graph repartitioning tools already exist, also, to our knowledge, parallel hypergraph repartitioning and/or remapping has not yet been addressed.

2.1.3 Synchronization avoidance

Large heterogeneous machines may also pose a synchronicity problem. Most partitioning algorithms make use of halo exchanges, that is, some form of all-to-all communication between neighbouring vertices held by different processes, as well as of parallel reduction, to inform all processes of the result of a distributed computation (e.g. the global sum of distributed values, etc.). With the advent of machines having several hundred thousand processing elements, and in spite of the continuous improvement of communication subsystems, the demand for more asynchronicity in parallel algorithms is likely to increase. In this respect, genetic algorithms may be good candidates, as computations can take place asynchronously within independent sub-populations, called demes, with some "champions" being asynchronously exchanged between neighbouring demes, much like what happens on Earth. When enough demes exist, convergence to a good local optimum is likely to be achieved, even if some demes may lag behind in terms of number of generations. Yet, as said above, genetic algorithms are too expensive, so that new algorithms have also to be investigated in this respect.

Static mapping aims at improving the locality of communications in the target machine, by taking into account the topology of the latter when partitioning the problem graph. This additional constraint poses a problem to the computation of initial mappings. While initial k-way partitions can be easily computed in parallel by means of recursive bipartitioning, this is not possible for recursive bi-mapping, because every bipartition at some level must take into account the

shapes of neighboring partitions. This information is available in a sequential context, but not in parallel, when $k \approx |V|$.

Also, the increasing number of processing elements hinders the convergence of partition refinement algorithms which are commonly used in k-way multilevel schemes. The more processing elements there are, the more vertices can be moved independently by each of them from overloaded domains to a presumed underloaded neighboring domain, overloading it even more than its neighbors. Computing exactly diffusion matrices prior to data movement may not always be possible, as these structures are in k^2. Using iterative diffusion-based algorithms may also lead unbalance, due to rounding artifacts when deciding which domain owns some vertex.

Future parallel static mapping software will therefore have to rely on a combination of these methods to preserve partition quality.