C. R. Mecanique 339 (2011) 90–95

Contents lists available at ScienceDirect

Comptes Rendus Mecanique

www.sciencedirect.com

High Performance Computing / Le Calcul Intensif

Current challenges in parallel graph partitioning

Les défis actuels du partitionnement parallèle de graphes

François Pellegrini

Université de Bordeaux, IPB, LaBRI & project Bacchus, INRIA Bordeaux – Sud-Ouest, 351, cours de la Libération, 33405 Talence, France

A R T I C L E   I N F O

Article history:
Available online 3 January 2011

Keywords:
Computer science
Graph partitioning
Parallel architecture

Mots-clés :
Informatique, algorithmique
Partitionnement de graphes
Architecture parallèle

A B S T R A C T

Graph partitioning is a technique used for solving many problems in scientific computing, such as the decomposition of a mesh into domains so as to evenly balance the compute load on the processors of a parallel architecture. Because of the ever increasing size of the meshes to handle, partitioning tools themselves had to be parallelized. The parallel versions of these software provide good results for and on several thousands of processors, but the advent of architectures comprising more than a million processing elements raises new problems. Not only do the partitioning results produced by these software have to take into account the heterogeneity of these architectures, but also does the efficient execution of the partitioning software on these architectures require much more sophisticated algorithms. The purpose of this note is to present the challenges to overcome in order to reach these goals.

© 2010 Published by Elsevier Masson SAS on behalf of Académie des sciences.

R É S U M É

Le partitionnement de graphes est une technique utilisée pour la résolution de nombreux problèmes en calcul scientifique, tels que la décomposition d'un maillage en sous-domaines pour répartir la charge de calcul sur les processeurs d'une architecture parallèle. La taille des maillages à traiter augmentant sans cesse, les logiciels de partitionnement ont eux-mêmes dû être parallélisés. Les versions parallèles de ces logiciels fournissent de bons résultats sur et pour plusieurs milliers de processeurs, mais l'arrivée imminente d'architectures hétérogènes comprenant plus d'un million d'unités de traitement pose de nouveaux problèmes. Non seulement les résultats de partitionnement produits par ces logiciels doivent-ils tenir compte de l'hétérogénéité de ces architectures, mais l'exécution efficace du logiciel de partitionnement sur cette même architecture doit-elle nécessiter une algorithmique bien plus sophistiquée. L'objet de cette note est de présenter les défis à surmonter afin d'atteindre ces objectifs.

© 2010 Published by Elsevier Masson SAS on behalf of Académie des sciences.

1. Introduction

Graph partitioning is an ubiquitous technique which has applications in many fields of computer science and engineering, such as domain decomposition for parallel iterative linear system solvers, VLSI circuit layout, or image segmentation, among others. It is used to help solving domain-dependent optimization problems modeled in terms of weighted or unweighted

E-mail address: pelegrin@labri.fr.

1631-0721/$ – see front matter © 2010 Published by Elsevier Masson SAS on behalf of Académie des sciences.
doi:10.1016/j.crme.2010.11.004

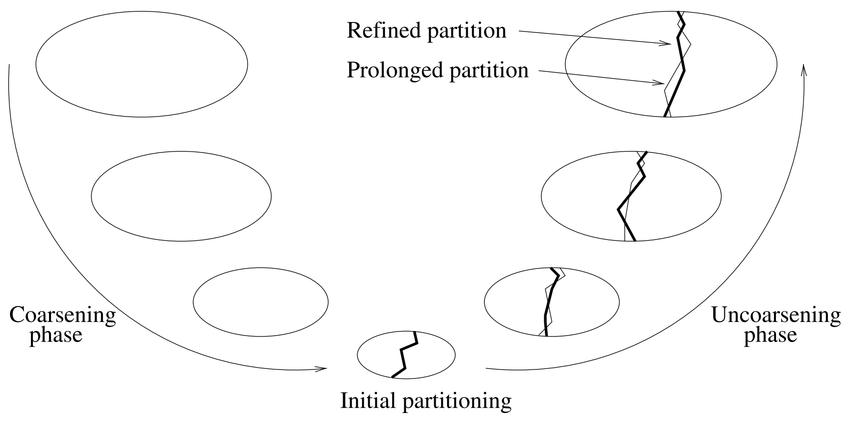


Fig. 1. The multilevel partitioning process, here in the case of bipartitioning. In the uncoarsening phase, the light and bold lines represent for each level the prolonged partition obtained from the coarser graph, and the partition obtained after refinement, respectively.

graphs, where finding good solutions amounts to computing, eventually recursively in a divide-and-conquer framework, small vertex or edge cuts that balance evenly the weights of the graph parts.

Save for very constrained subproblems, the computation of a balanced partition with minimal cut is NP-complete, even in the bipartitioning case [1]. Consequently, for problems of sizes above several thousands of vertices, heuristics are the only way to provide acceptable solutions in reasonable time.

Over the past decades, many methods have been proposed to compute graph partitions with small cut size: evolutionary algorithms (comprising simulated annealing [2,3], genetic algorithms [4], ant colonies [5], greedy iterative algorithms [6,7]), graph algorithms [8,9], geometry-based inertial methods [10], spectral methods [11], region growing [12], etc. While taking their inspiration from radically different fields such as genetics or statistical physics, analogies between these methods are numerous, as well as cross-fertilization in their implementations, so that it is sometimes difficult to categorize them unambiguously. While some of them are specifically designed to minimize cut size, that is, the number of cut edges or of separator vertices, others can handle more general *cost functions*, so as to provide partitions which represent trade offs between multiple constraints.

1.1. The multilevel framework

Experience has shown that, for many graphs used in scientific computations, the best partition quality is achieved when using a multilevel framework. This method, which derives from the multi-grid algorithms used in numerical physics, repeatedly reduces the size of the graph to partition by finding matings which collapse vertices and edges, computes an initial partition for the coarsest graph obtained, and prolongs[1] the result back to the original graph [11,13,14] (see Fig. 1).

Multilevel methods are most often combined with partition refinement methods, in order to smooth the prolonged partitions at every level so that the granularity of the solution is the one of the original graphs and not the one of the coarsest graphs.

In the sequential case, the most popular refinement heuristics in the literature are the local optimization algorithms of Kernighan–Lin [15] (KL) and Fiduccia–Mattheyses [16] (FM), either on edge-based [13] or vertex-based [17] forms depending on the nature of the desired separators. All of these algorithms base on successive moves of frontier vertices to decrease the current value of the prescribed cost function. Vertices to be moved are chosen according to their *gain value*, that is, the amount by which the current value of the cost function would be modified if the given vertex was moved to some other part; vertices of negative gain values are consequently the most interesting to move. Moves which adversely affect the cost function may be accepted, if they are further compensated by moves which result in an overall gain, therefore allowing these algorithms to perform *hill-climbing* from local minima of the cost function.

1.2. The need for parallelism

Based on the multilevel framework, several successful general-purpose sequential tools for graph partitioning have been developed in the past decades, such as CHACO [18], METIS [19], JOSTLE [20] and our own project, SCOTCH [21]. However, 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. The PT-SCOTCH project (for "*Parallel Threaded* SCOTCH"), currently carried out within the BACCHUS team of INRIA Bordeaux – Sud-Ouest, is yet another attempt to address this problem.

[1] While a *projection* is an application to a space of lower dimension, a *prolongation* refers to an application to a space of higher dimension.

In order to anticipate the expected growth rate of problem and machine sizes, our initial design goal for PT-Scotch was to be able to partition graphs of above one billion vertices, distributed over a thousand processors. This goal has been successfully achieved by mid-2010, with revision 5.1.10 of PT-Scotch being able to partition a graph over 2.4 billion vertices, distributed across 2048 processors of the platine machine located at the French CCRT computer center.

The advent of massively parallel NUMA[2] and heterogeneous[3] machines represents a new challenge for graph partitioning software designers, because partitioning tools will have to be scalable up to hundred thousand of processing elements. The purpose of this paper is to present the key issues that are currently tackled within the Scotch project for the development of highly scalable parallel graph partitioning algorithms suitable for these new architectures.

2. Three main challenges

The advent of massively parallel, heterogeneous machines impacts the design of the software to be executed on these machines, both for applications and for service tools.

For the Scotch project, it results in three interwoven challenges, which we are going to discuss below. These challenges are a consequence of our new roadmap, which consists in being able to handle graphs of a trillion vertices distributed across a million processing elements. While, in our previous roadmap, scalability was mostly a matter of handling graphs with a large number of vertices, implying the definition of suitable data structures, the new roadmap focuses on the ability to run efficiently on a large number of processing elements, implying the definition of suitable algorithms.

The figures of our first roadmap were coined because it amounted to storing about one million vertices per processing element, which matched the memory capacity of the latter. We reiterated this assumption in our second roadmap, because we assume that the continuous increase in the number of processing elements will tend to match the continuous increase in overall memory capacity of the machines, so that the amount of memory per processing element will remain constant. Consequently, the critical algorithmic issue in the years to come will be the need to hide the latency of a larger amount of communication, which are likely to involve a larger fraction of the local data borne by the processing elements, without any possibility to increase the size of the local data. Moreover, latency is likely to increase because of the increasing heterogeneity of the machines, making synchronous algorithms more prone to communication bottlenecks and stalling. These three challenges of scalability, heterogeneity and asynchronicity have to be addressed concurrently.

2.1. The challenge of scalability

Even without taking into account architectural concerns, which will add new constraints, our data structures and algorithms must be able to scale up to the sizes prescribed in our new roadmap. After recalling the design choices that we did in the context of our first roadmap, we will discuss some issues related to the algorithms.

2.1.1. Design choices

In order to reach our first goal, some early design decisions have been 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. Unlike more robust approaches [22], we presumed that vertex adjacencies can be stored locally on every process, without incurring too much memory imbalance or even memory shortage.

Also, we wanted our software to run on any number of processes p, and produce any number of parts k, to avoid limitations which existed in other packages. For instance, ParMeTiS can compute orderings only on a number of processes which is a power of two ($k = 2^i$), and Jostle produces only as many domains as the number of processes it is running on ($k = p$).

In order for parallel partitioning algorithms to be able to scale on an arbitrary number of processing elements, graph data must never be duplicated, across all of the processing elements, more than a (small) factor of the vertex and edge set sizes $|V|$ and $|E|$. The distributed graph structure which we implemented, and which we describe below, fulfills this goal. Also, all of our algorithms should never use data structures that are super-linear in k, p and/or $|V|$. For instance, in the case of graph repartitioning, where the number of vertices to be moved between every pair of parts has to be determined so as to rebalance graph parts while minimizing data transfers, centralized flow matrices must never be used (it is a known problem that the current version of ParMeTiS builds such an array of size p^2).

In terms of execution architecture, we considered a distributed memory model, since upcoming, very large machines are not likely to ever implement efficiently a shared-memory paradigm, were it be NUMA. The distributed memory paradigm has a strong impact on algorithms, because mutual exclusion and locking issues, which are implicitly resolved by doing memory accesses in the sequential case, have to be reformulated in terms of undeterministic, latency-inducing, message exchanges.

[2] *Non-Uniform Memory Access*. These machines are categorized by the fact that, while their memory can be accessed from all of their processing elements, the cost of access may differ depending on the location of the memory with respect to the processing element which makes the request.

[3] Heterogeneous machines are even more generic than NUMA ones, as they can also consist in clusters of machines of different kinds and compute powers, linked by networks of different bandwidths and latencies, e.g. a grid computer system.

According to the above assumptions, in our PT-Scotch software, like in other packages, distributed graphs are represented by means of adjacency lists [23]. Vertices are distributed across processes along with their adjacency lists and with some duplicated global data. Since many algorithms require that local data be attached to every vertex, and since using extensively global indices would be too expensive in our opinion, all vertices owned by some process are also assigned local indices, suitable for the indexing of compact local data arrays. This local indexing is extended so as to encompass all non-local vertices which are neighbors of local vertices, which are referred to as ghost vertices. Since only local vertices are processed by the distributed algorithms, the adjacency of ghost vertices is never stored on the processes, which guarantees the scalability of the data structure as no process will store information of a size larger than its number of local outgoing arcs.

2.1.2. *Algorithmic scalability*

It is common knowledge in the parallel computing community that the best parallel algorithm to solve a given problem is most often not the parallel transposition of the best sequential algorithm for this purpose. Graph partitioning algorithms by no means escape this rule. In particular, the FM and KL state-of-the-art sequential methods for local optimization, which are iterative by nature, bear strong sequentiality constraints which prevent their direct transposition into scalable parallel formulations.

This is why several of our contributions specially targeted the conception of highly scalable algorithms for the two critical issues of the multilevel framework: graph coarsening, and the optimization of prolonged partitions during the uncoarsening phase. Regarding graph coarsening, we proposed a probabilistic matching algorithm that does not incur any bias with respect to initial graph data distribution [24], and converges in a small number of iterations, as only 5 rounds of collective communication are required in practice to match more than 80% of the vertices. Regarding the optimization of prolonged partitions, we decided to turn to global optimization algorithms, such as diffusion-based algorithms [25], which are much more suitable for parallelization than local optimization algorithms. Yet, as these algorithms would have been much too expensive when applied to the finer graphs, we proposed to reduce problem space to a small band of vertices around the prolonged separators [26].

The idea of diffusion-based algorithms is to model the graph to partition as a set of barrels linked by pipes. Liquids are flowed in from the centers of each of the domains, and create fronts when they meet, tending to minimize the surface tension energy in the domain interfaces. This why such algorithms are also referred to as “bubble-growing” algorithms.

While our data structures are likely to scale nicely up to the fulfillment of the goal of our second roadmap, and while our probabilistic coarsening algorithm is independent of graph size and of data distribution, the current version of our diffusion-based optimization algorithm is susceptible to coarsening and rounding artifacts, so that the produced domains become too imbalanced when too many parts, having only a few vertices per part, are sought for. Moreover, this algorithm is not suitable for creating domains for heterogeneous architectures, since the diffusion process cannot account for the fact that neighbors may belong to domains which are more expensive to reach than others. This would require to dynamically change the weights of the edges, depending on the domains to which their end vertices are currently assigned, which has no meaning in terms of physical analogy. Suitable algorithms have therefore to be designed, as we are going to see in the next section.

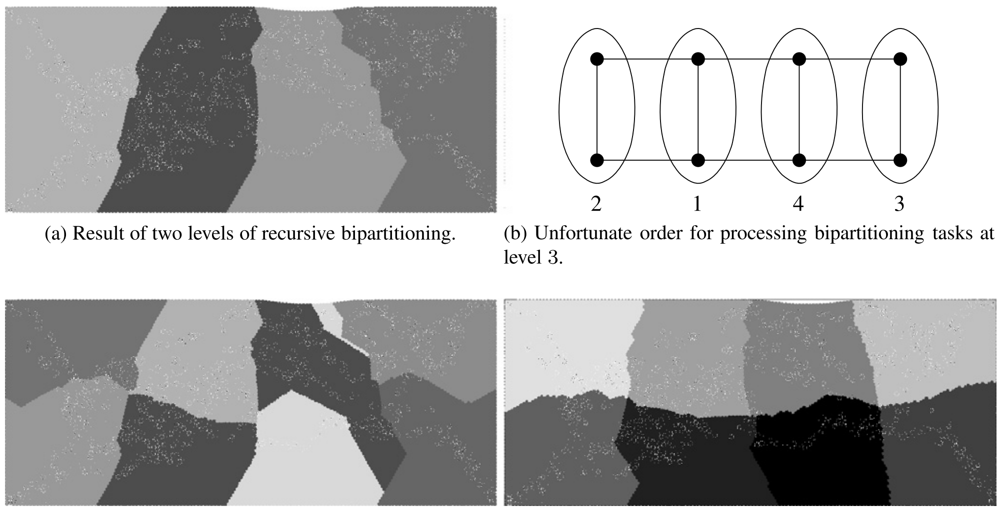
2.2. *The challenge of heterogeneity*

The problem of assigning the communicating processes of a parallel program onto the processing elements of some computer system is referred to as *mapping* in the literature. In the SPMD[4] context described above, it is equivalent to the *distribution* across processors of the data structures of parallel programs; in this case, all pieces of data assigned to some processor are handled by a single process located on this processor. This is for instance the case when computing domain decompositions of large meshes, so that each of their domains is handled by a different processing element. In this context, we will talk about *processes* rather than about *processors* or *processing elements* to represent the entities across which data will be distributed.

The parallel program to be mapped onto the target architecture is modeled by a valuated unoriented graph, called source graph or process graph, the vertices of which represent the processes of the parallel program, and the edges of which represent the communication channels between communicating processes or data dependencies. The target machine onto which the parallel program is mapped, is also modeled by a valuated unoriented graph, called target graph or architecture graph. The *partitioning* problem can therefore be seen as a subproblem of the mapping problem, where all processing elements are identical and all equally interconnected, that is, the target graph is a complete graph with identical edge weights.

The *a priori* computation of efficient mappings requires some knowledge on the dynamic behavior of the target machine with respect to the programs which are run on it. This knowledge is synthesized in a cost function, the nature of which determines the characteristics of the desired mappings. One of the most widely used cost functions for communication minimization is the sum, for all edges of the source graph to be mapped onto the target graph modeling the target architecture,

[4] *Single Process, Multiple Data*. This is a programming paradigm in which all processing elements of some parallel machine run the very same code, the behavior of the latter being conditioned by the data which have been loaded onto it.



(a) Result of two levels of recursive bipartitioning. (b) Unfortunate order for processing bipartitioning tasks at level 3.

(c) Resulting "twist" in task 4 if tasks are processed as in (b). (d) Result if tasks are processed from left to right.

Fig. 2. Impact of the order in which sequential recursive bipartitioning tasks are processed within the same level when mapping the 2D rectangular graph BUMP onto a 2×4 grid architecture. When bipartitioning tasks are processed as in figure (b), if parts computed by tasks 1 and 3 are not assigned in the same way, task 4 will be forced to produce two "twisted" parts so as to minimize overall edge dilation in figure (c). When tasks are processed in judicious order, jointed frontiers can be produced, as in figure (d).

of their dilation[5] multiplied by their weight [13,27]. In the case of plain graph partitioning, where all edge dilations are equal to 1, this metric simplifies into the sum of the weights of all cut edges. This function is widely used, because it is easy to compute and because its minimization favors the mapping of intensively intercommunicating processes onto nearby processors [27,28].

In the last decades, when advances in computer architectures allowed vendors to build parallel architectures comprising up to several hundreds of processing elements and that still exhibited a quasi-UMA behavior, the domain decomposition problem could be efficiently solved by means of regular graph partitioning tools. With the advent of much more heterogeneous target architectures, either hierarchical NUMA (cores tightly interconnected within processor chips, the latter being clustered on shared-memory boards, which are stacked into cabinets, interconnected with high-speed links) or loosely-interconnected grids, it becomes more and more important to take topology into account when splitting process graphs.

The sequential Scotch software possesses static mapping capabilities since its inception. It bases on an algorithm called "dual recursive bipartitioning" [27], which proceeds by recursive allocation of subsets of processes to subsets of processors. Yet, this algorithm cannot be transposed in parallel, because in order for a recursive bipartitioning task to maximize local communication, it must have knowledge of the outcome of its already computed neighboring bipartitioning tasks. While careful ordering of the bipartitioning tasks can lead to good results in the sequential case, as evidenced in Fig. 2, this is not possible at all in the parallel case, because all tasks at the same recursion level are supposed to run in parallel on distinct processes.

A solution to this is to adapt our parallel multilevel framework so as to compute a sequential static mapping of the coarsest graph, and use a parallel local optimization algorithm that allows for such complex cost functions. As seen before, diffusion-based algorithms cannot easily be adapted. This is why we have also experimented with genetic algorithms, but they are way too expensive, even on reduced problem spaces. New algorithms have to be investigated.

2.3. The challenge of asynchronicity

Yet, large heterogeneous machines may also pose a synchronicity problem. Most algorithms that we presented here make use of halo exchanges, that is, some form of all-to-all communication between neighbor vertices borne 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 thousands of 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 neighboring 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

[5] The dilation of a source graph edge is the length of the shortest path, within the target graph, linking the two target graph vertices onto which are placed the two source graph vertices which are the ends of the considered edge.

above, genetic algorithms are too expensive, so that new algorithms have also to be investigated in this respect. The study of parallel asynchronous graph partitioning algorithms is therefore, in our opinion, to become an active way of research in the years to come.

3. Conclusion

Traditional parallel programming techniques, based on message-passing and synchronous algorithms, have allowed us to create a parallel graph partitioning software able to partition graphs of more than a billion vertices on a thousand processing elements.

However, the design of parallel graph partitioning software for the heterogeneous, massively parallel architectures to come requires the tackling and solving of three interwoven challenges, namely scalability, heterogeneity and asynchronicity. In the context of the proved multilevel framework, new coarsening and partition refinement algorithms have to be developed, which is the subject of our current and future research for the years to come.

References

[1] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, 1979.

[2] S.W. Bollinger, S.F. Midkiff, Processor and link assignment in multicomputers using simulated annealing, in: Proc. 11th Int. Conf. on Parallel Processing, The Penn. State Univ. Press, August 1988, pp. 1–7.

[3] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (May 1983) 671–680.

[4] T.N. Bui, B.R. Moon, Genetic algorithm and graph partitioning, IEEE Trans. Comput. 45 (7) (1996) 841–855.

[5] A.E. Langham, P.W. Grant, Using competing ant colonies to solve k-way partitioning problems with foraging and raiding strategies, in: ECAL'99: Proc. 5th European Conference on Advances in Artificial Life, in: LNCS, vol. 1674, Springer, 1999, pp. 621–625.

[6] R. Battiti, A.A. Bertossi, Greedy prohibition, and reactive heuristics for graph partitioning, IEEE Trans. Comput. 48 (4) (1999) 361–385.

[7] M. Laguna, T.A. Feo, H.C. Elrod, A greedy randomized adaptive search procedure for the two-partition problem, Oper. Res. 42 (August 1994) 677–687, doi:10.1287/opre.42.4.677.

[8] C. Fahrat, A simple and efficient automatic FEM domain decomposer, Comput. Struct. 28 (5) (1988) 579–602.

[9] G. Karypis, V. Kumar, Multilevel graph partitioning schemes, in: Proc. 24th Intern. Conf. Par. Proc. III, CRC Press, 1995, pp. 113–122.

[10] B. Nour-Omid, A. Raefsky, G. Lyzenga, Solving finite element equations on concurrent computers, in: A.K. Noor (Ed.), Parallel Computations and Their Impact on Mechanics, ASME Press, 1986, pp. 209–227.

[11] S.T. Barnard, H.D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, Concurrency: Practice and Experience 6 (2) (1994) 101–117.

[12] R. Diekmann, R. Preis, F. Schlimbach, C. Walshaw, Aspect ratio for mesh partitioning, in: Proc. Euro-Par'98, in: LNCS, vol. 1470, Springer, 1998, pp. 347–351.

[13] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, in: Proc. ACM/IEEE Conference on Supercomputing (CDROM), December 1995, 28 pp.

[14] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.

[15] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, BELL System Technical Journal 49 (2) (February 1970) 291–307.

[16] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proc. 19th Design Automation Conference, IEEE, 1982, pp. 175–181.

[17] B. Hendrickson, E. Rothberg, Improving the runtime and quality of nested dissection ordering, SIAM J. Sci. Comput. 20 (2) (1998) 468–489.

[18] Chaco: Software for partitioning graphs, http://www.sandia.gov/~bahendr/chaco.html.

[19] metis: Family of multilevel partitioning algorithms, http://glaros.dtc.umn.edu/gkhome/views/metis.

[20] jostle: Graph partitioning software, http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/.

[21] scotch: Static mapping, graph partitioning, and sparse matrix block ordering package, http://www.labri.fr/~pelegrin/scotch/.

[22] B. Vastenhouw, R.H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix–vector multiplication, SIAM Rev. 47 (1) (2005) 67–95.

[23] C. Chevalier, F. Pellegrini, PT-Scotch: A tool for efficient parallel graph ordering, Parallel Comput. 34 (2008) 318–331.

[24] J.-H. Her, F. Pellegrini, Efficient and scalable parallel graph partitioning by recursive bipartitioning, http://www.labri.fr/~pelegrin/papers/scotch_parallelbipart_parcomp.pdf, Parallel Computing (2011), in press.

[25] F. Pellegrini, A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries, in: Proc. Euro-Par'07, in: LNCS, vol. 4641, Springer, August 2007, pp. 191–200.

[26] C. Chevalier, F. Pellegrini, Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework, in: Proc. Euro-Par'06, Dresden, in: LNCS, vol. 4128, Springer, September 2006, pp. 243–252.

[27] F. Pellegrini, Static mapping by dual recursive bipartitioning of process and architecture graphs, in: Proc. SHPCC'94, IEEE, May 1994, pp. 486–493.

[28] C. Walshaw, M. Cross, M.G. Everett, S. Johnson, K. McManus, Partitioning and mapping of unstructured meshes to parallel machine topologies, in: Proc. Irregular'95, in: LNCS, vol. 980, Springer, 1995, pp. 121–126.