The PT-Scotch project: purpose, algorithms, first results

Cédric Chevalier and François Pellegrini
LaBRI and INRIA Futurs
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, FRANCE
{cchevali|pelegrin}@labri.fr

I. Introduction

Graph partitioning is an ubiquitous technique which has applications in many fields of computer science and engineering. It is mostly used to help solving domain-dependent optimization problems modeled in terms of weighted or unweighted 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.

Because there always exists large problem graphs which cannot fit in the memory of sequential computers and cost too much to partition, parallel graph partitioning tools have been developed. PT-Scotch is another attempt to provide a simple and efficient library for parallel graph partitioning and ordering. We present in this paper the main research topics that we want to cover in this project in order to achieve our goals, as well as some early results.

II. Some paths for efficient parallel graph partitioning

The purpose of the PT-Scotch software ("*Parallel Threaded* Scotch", an extension of the sequential Scotch software), developed at LaBRI within the ScAlApplix project of INRIA Futurs, is to provide efficient parallel tools to partition graphs with sizes up to a billion vertices, distributed over a thousand processors. This deliberately ambitious goal aims at tackling frontally scalability and efficiency issues.

In order to achieve good efficiency on a large number of processors, our project focuses on several key issues:

- Multi-threading: although the MPI API has been designed so as to enable thread-safe implementations (by passing relevant data structure handles whenever necessary, which allow implementors not to use global context variables), it is only recently that robust multi-platform MPI implementations have been provided (such as MPIch2 or OpenMPI), specifically designed to support multi-threading. All of the routines of Scotch and PT-Scotch are also designed so as to be reentrant, and we want to explore in PT-Scotch the use of multi-threading to cover communications with useful computations, both within individual methods and on independent computations performed on the same graph data.
- Parallel multi-level: all state-of-the-art graph partitioning tools are based on a multi-level approach, several parallel implementations of which have been proposed by the creators of parallel graph partitioning tools. The matching algorithm used to recursively coarsen graphs is critical, as it requires much communication to take place, to match pairs of vertices located on distant processors. This communication cannot be avoided, because the simple reduction of communication by privileging local matchings tends to lower matching quality, as it biases the mating process. Based on the use of multi-threading, we are currently investigating an asynchronous multi-buffered approach, to determine in what respect delaying matching requests can bias mating and reduce coarsening quality.
- Local optimization algorithms: in a multi-level context, local optimization algorithms are critical, as they must, at every uncoarsening step, refine the frontier project back from the previous level. The most efficient sequential algorithms known to date are like of the Fiduccia-Mattheyses algorithm. Since this algorithm is intrinsically sequential, most techniques that have been proposed to optimize it rely on finding independent moves that can be done in parallel without requiring synchronization. The approach that we investigate is based on two axes: the first one aims at keeping using the traditional sequential algorithm as much as possible, because it does not incur communication overheads. We have explored a pre-constrained banding approach that has proven very efficient in this respect. The second axis is based on radically different algorithms, that are intrinsically parallel, such as evolutionary algorithms, which we hope will scale better when the number of processors increase.

III. Early results on graph ordering

Parallel graph ordering has been chosen is the first target application of the PT-Scotch project. Graph ordering is a critical problem for the efficient factorization of symmetric sparse matrices, not only to reduce fill-in and factorization cost, but also to increase concurrency in the elimination tree, which is essential in order to achieve high performance when solving these linear systems on parallel architectures. We outline in this abstract the algorithms which we have implemented in PT-Scotch to parallelize the Nested Dissection ordering method.

Our implementation takes advantage of three levels of concurrency. The first level is the implementation of the nested dissection method itself, which is straightforward thanks to the intrinsically concurrent nature of the algorithm. Starting from the initial graph, arbitrarily distributed across p processors, the algorithm proceeds as follows: once a separator has been computed in parallel, by means of a method described below, each of the p processors participates in the building of the distributed induced subgraph corresponding to the first separated part. This subgraph is then folded on the first $\lceil \frac{p}{2} \rceil$ processors, such that the average number of vertices per processor, which guarantees efficiency as it allows the shadowing of communications by a subsequent amount of computation, remains constant. The same procedure is used to build, on the $\lfloor \frac{p}{2} \rfloor$ remaining processors, the folded induced subgraph corresponding to the second part. These two constructions being completely independent, each of the computations of an induced subgraph and of its folding can be done in parallel, thanks to the temporary creation of an extra thread per processor. At the end of the folding process, the nested dissection process can recursively proceed independently on each subgroup of $\frac{p}{2}$ processors, until each of the subgroups is reduced to a single processor. From then on, the nested dissection process will go on sequentially, using the nested dissection routines of the Scotch library.

The second level of concurrency regards the computation of separators in a multi-level framework. The matching of vertices is performed in parallel by means of an asynchronous probabilistic multi-threaded algorithm. At the end of each coarsening step, the coarser graph is folded onto half of the processors that held the finer graph, in order to keep a constant number of vertices per processors, but it is also duplicated on the other half of the processors too. Therefore, the coarsening process can recursively proceed independently on each of the two halves, which results in an improvement of the quality of the separators, as only the best separator produced by the two halves is kept at the upper level.

The third level of concurrency regards the refinement heuristic which is used to improve the separators. We have successfully tested a multi-sequential approach, where at every distributed uncoarsening step, a distributed band graph is created and then centralized on all of the processors. These copies can be used collectively to run a scalable parallel multi-deme genetic optimization algorithm, or fully independent runs of a full-featured sequential FM algorithm. The best refined band separator is projected back to the distributed graph, and the uncoarsening process goes on. This scheme allows us to even gain in quality when the number of processors increases, as the problem space can be explored by more independent agents in the same amount of time.

Graph	**Size** ($\times 10^3$)		**Average degree**
	V	E	
audikw1	944	38354	81.28
conesphere1m	1055	8023	15.21
coupole8000	1768	41657	47.12
thread	29	2220	149.32

TABLE I
Some of our test graphs.

Test case		**Number of processes**		
		2	16	128
audikw1	S	5.59e+12	5.32e+12	5.31e+12
	P	5.98e+12	7.42e+12	1.52e+13
conesphere1m	S	1.86e+12	1.86e+12	1.94e+12
	P	2.14e+12	3.05e+12	3.48e+12
coupole8000	S	7.44e+10	7.41e+10	7.41e+10
	P	8.14e+10	8.21e+10	9.19e+10
thread	S	3.70e+10	4.32e+10	4.65e+10
	P	4.38e+10	1.12e+11	–

TABLE II
Cholesky operation count (OPC) for PT-Scotch (S, top lines) and ParMeTiS (P, bottom lines).

Table II presents the operation count of Cholesky factorization (OPC) yielded by the orderings computed with PT-Scotch and ParMeTiS. The improvement in quality yielded by PT-Scotch is clearly evidenced, and increases along with the number of processes, as our local optimization scheme is not sensitive to the number of processes.

More results will be given in the presentation, regarding time and memory occupation issues. We will also discuss how these results, which are interesting in the context of parallel graph ordering, can or cannot be extended in the field of parallel graph ordering, where some communication is still needed between all of the branch of the recursive bipartitioning process, and how the problem of k-way partitioning can be addressed.