SCOTCH : A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs*

François Pellegrini and Jean Roman

LaBRI, URA CNRS 1304
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, FRANCE
{pelegrin|roman}@labri.u-bordeaux.fr

Abstract. The combinatorial optimization problem of assigning the communicating processes of a parallel program onto a parallel machine so as to minimize its overall execution time is called *static mapping*.
This paper presents SCOTCH, a software package for static mapping based on the recursive bipartitioning of both the source process graph and the target architecture graph. SCOTCH can map any weighted source graph onto any weighted target graph in a time linear in the number of source edges and logarithmic in the number of target vertices. We give brief descriptions of the algorithm and its bipartitioning methods, and compare its performance to other mapping and partitioning programs.

1 Introduction

The efficient execution of a parallel program on a parallel machine requires that the communicating processes of the program be assigned to the processors of the machine so as to minimize its overall running time. When processes are assumed to coexist simultaneously for the duration of all the program, this optimization problem is called *mapping*. It amounts to balancing the computational weight of the processes among the processors of the machine, while reducing the communication overhead induced by parallelism by keeping intensively intercommunicating processes on nearby processors. A mapping is called *static* if it is computed prior to the execution of the program and is never modified at run-time.
Static mapping is NP-complete in the general case [4]. Therefore, many studies have been carried out in order to find sub-optimal solutions in reasonable time. Specific algorithms have been proposed for mesh [10] and hypercube [2, 5] topologies. When the target machine is assumed to have a communication network in the shape of a complete graph, the static mapping problem turns into the partitioning problem, which has also been intensively studied [1, 7, 8, 13].

SCOTCH is a project carried out at the *Laboratoire Bordelais de Recherche en Informatique* (LaBRI) of the Université Bordeaux I, by the ALiENor (ALgorithmics and ENvironments for parallel computing) team. Its goal is to study static

* This work was supported by the French GDR PRS

mapping by the means of graph theory, using a "divide and conquer" approach. This work has resulted in the development of the Dual Recursive Bipartitioning (or DRB) mapping algorithm and the analysis of several graph bipartitioning heuristics.

This paper describes the principles and capabilities of the Scotch software package for static mapping, which embodies all the algorithms and graph bipartitioning heuristics developed within project Scotch. This package allows the user to map efficiently any weighted source graph onto any weighted target graph, or even onto disconnected subgraphs of a given target graph, in a time linear in the number of source edges and logarithmic in the number of target vertices. The rest of the paper is organized as follows. Section 2 presents some definitions, and section 3 outlines the most important aspects of the Dual Recursive Bipartitioning algorithm. Section 4 defines some of the bipartitioning algorithms that we use, and section 5 describes Scotch itself. Section 6 gives some performance results. Then follows the conclusion.

2 Static Mapping and Cost Functions

A *mapping* of a process (or *source*) graph S onto an architecture (or *target*) graph T consists of two applications $\tau_{S,T} : V(S) \longrightarrow V(T)$ and $\rho_{S,T} : E(S) \longrightarrow \mathcal{P}(E(T))$, where $\mathcal{P}(E(T))$ denotes the set of all simple loopless paths which can be built from $E(T)$. $\tau_{S,T}(v_S) = v_T$ if process v_S of S is mapped onto processor v_T of T, and $\rho_{S,T}(e_S) = \{e_T^1, e_T^2, \ldots, e_T^n\}$ if communication channel e_S of S is routed through communication links e_T^1, e_T^2, ..., e_T^n of T. $|\rho_{S,T}(e_S)|$ denotes the dilation of edge e_S, *i.e.* the number of edges of $E(T)$ used to route e_S.

The computation of efficient static mappings requires an *a priori* knowledge of 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 optimal mappings. The goal of our mapping algorithm is to minimize some communication cost function, while keeping the load balance within a user-specified tolerance. The communication cost function f_C that we have chosen is the sum, for all edges, of their dilation multiplied by their weight:

$$f_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{e_S \in E(S)} w(e_S) \, |\rho_{S,T}(e_S)| \ .$$

This function is easy to compute, can be updated incrementally, and its minimization favors the mapping of intensively intercommunicating processes onto nearby processors. The strong positive correlation between its values and effective execution times has been experimentally verified by several authors [5, 7].

3 The Dual Recursive Bipartitioning Algorithm

Our mapping algorithm is based on a *divide and conquer* approach. It proceeds by recursive allocation of subsets of processes to subsets of processors. The algorithm starts by considering a set of processors, also called *domain*, containing all the processors of the target machine, and to which is associated the set of all the processes. At each step, the algorithm bipartitions a domain into two disjoint subdomains, and calls a graph bipartitioning algorithm to map the subset of processes associated with the domain onto the two subdomains. Whenever a domain is restricted to a single processor, the processes of the set associated with the domain are assigned to this processor, and recursion stops. All routines that handle domains and bipartition graphs are seen as black boxes by the mapping program, which can thus accept any kind of target architecture and process graph bipartitioning functions.

The association of a subdomain with every process defines a *partial mapping* of the process graph. As bipartitionings are performed, the subdomain sizes decrease, up to give a complete mapping when all subdomains are of size one. The production of efficient complete mappings requires that all graph bipartitionings favor the criteria that we have chosen. Therefore, the bipartitioning of a subgraph S' of S should maintain load balance within the user-specified tolerance, and minimize the *partial* communication cost function f'_C, defined as

$$f'_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{\substack{v \in V(S') \\ \{v,v'\} \in E(S)}} w(\{v,v'\}) \, |\rho_{S,T}(\{v,v'\})| \ ,$$

which accounts for the dilation of edges internal to subgraph S' as well as for the one of edges which belong to the cocycle of S'. Dilation of edges are computed by estimating the distance in the target graph between the domains to which edge ends belong. Using such a distance function amounts to considering that all routings will use shortest paths in the target architecture. The computation of edge dilations as distances between subdomains amounts to taking into account partial mappings issued by previous bipartitionings, and makes it possible to avoid local choices that might prove globally bad.

4 Domain and Graph Bipartitioning Methods

Since, in our approach, the recursive bipartitioning of the target graph is fully independent with respect to that of the source graph (however, the opposite is false), the recursive decomposition of a given target architecture needs only be computed once, in order to store the resulting data in look-up decomposition tables which will be used in the mapping process. Decomposition tables can be easily computed with our mapper, by mapping the considered target graph onto the complete graph with same number of vertices. Mapping onto the complete graph zeroes the contribution of cocycle edges, so that only cut minimization is considered. In the resulting decomposition, strongly-connected clusters of processors will be kept uncut as long as possible, and strongly-connected clusters of processes will therefore tend to be mapped onto them.
The mapping of a source graph onto any target graph can be performed in two steps. The mapping engine is called a first time to map the target graph onto

the complete graph, in order to build the decomposition table for this architecture. Then, it is called a second time to map the source graph onto the target architecture, using the decomposition table that has just been computed.

The core of our recursive mapping algorithm uses process graph bipartitioning methods as black boxes. It allows us to run any type of graph bipartitioning method compatible with our criteria. Moreover, it is possible to apply several different methods in sequence, each one starting from the result of the previous one, and to select these methods with respect to the job characteristics, leading to the definition of *mapping strategies*.

Several graph bipartitioning methods have been implemented yet: random and greedy algorithms to compute initial bipartitions and refine them, a backtracking method, and an improved version of the Fiduccia-Mattheyses heuristics [3]. The Fiduccia-Mattheyses graph bipartitioning heuristics tries to reduce the cut of a partition by moving vertices between subsets. To achieve fast convergence, vertices the moving of which would bring the same *gain* to the cut are linked into lists which are indexed in a *gain array*. The almost-linearity in time of this algorithm is based on the assumption that the range of gain values is small, so that the search in the gain array for the vertices of best gain takes an almost-constant time. To handle the huge gains generated by the weights and dilations of edges, we have implemented a logarithmic indexing of gain lists, which keeps the gain array a reasonable size guaranteeing an almost-constant access time. Results obtained with linear and logarithmic data structures are equivalent in quality, which shows that the approximation induced by logarithmic indexing is of same order of magnitude as the one inherent to the Fiduccia-Mattheyses algorithm (see [11, page 91] for a full argumentation).

5 The Scotch Software Package

Scotch is a software package for static mapping, which embodies the algorithms developed within the Scotch project. Apart from the mapper itself, the Scotch package contains programs to build and test source graphs, compute target graph decompositions, and visualize mapping results. Advanced command-line interface and vertex labeling capabilities make them easy to interface with other programs. See [12] for details.
The mapper can map any weighted source graph onto any weighted target graph, or even onto disconnected subgraphs of a given target graph, which is very useful in the context of multi-user parallel machines. On these machines, when users request processors in order to run their jobs, the partitions allocated by the operating system may not be regular nor connected, because of existing partitions already attributed to other people. With Scotch, it is possible to build a target decomposition corresponding to this partition, and therefore to map processes onto it, automatically and regardless of the partition shape.

The Scotch 3.0 academic distribution may be obtained from the Scotch WWW page at `http://www.labri.u-bordeaux.fr/~pelegrin/scotch/`, or by `ftp` at `ftp.u-bordeaux.fr` in directory `/pub/Local/Info/Software/Scotch`. It is labeled `scotch_3.0A.tar.gz`. It contains the executables for several machines and operating systems, along with documentation and sample files.

6 Performance Evaluations

When mapping onto the complete graph, our program behaves as a standard graph partitioner. Table 1 summarizes edge cuts that we have obtained for classical test graphs, compared to the ones computed by CHACO [6] and METIS [9]. SCOTCH produces partitions the cut of which is within a few percents of the ones computed by the two other programs. It can therefore be used as a state-of-the-art graph partitioner. However, not accounting for the target topology generally leads to worse performance results of the mapped applications [5, 7], due to long-distance communication, which makes static mapping more attractive than strict partitioning for most communication-intensive applications.

Graph	CHACO-ML			METIS-ML			SCOTCH		
	64	128	256	64	128	256	64	128	256
4ELT	2928	4514	6869	2965	4600	6929	2941	4604	6900
BCSSTK30	241202	318075	423627	190115	271503	384474	194539	277122	382375
BRACK2	34172	46835	66944	29983	42625	60608	30270	42743	60583
PWT	9166	12737	18268	9130	12632	18108	9286	12887	18366
ROTOR	53804	75140	104038	53228	75010	103895	55511	77136	105006

Table 1. Edge cut produced by CHACO, METIS, and SCOTCH for partitions with 64, 128, and 256 blocks (CHACO and METIS data extracted from [8]).

The Cyclic Pairwise Exchange (CPE) algorithm developed by Hammond [5] maps graphs onto hypercubes by computing an initial balanced partition of low cut by means of a recursive spectral bisection (RSB) algorithm, and refining it by exchanging vertices between neighbor processors along the hypercube dimensions. In table 2 are given mapping results extracted from performance curves in [5], expressed with respect to $\overline{\Lambda} = \frac{f_C}{2048}$, and equivalent results computed by our mapper. SCOTCH produces mappings the quality of which is consistently more than 10% better than the ones computed by RSB+CPE.

Graph	$\overline{\Lambda}$ RSB+CPE	$\overline{\Lambda}$ SCOTCH	f_C SCOTCH
BUMP	> 4	3.964	8119
4ELT	> 5	4.686	9600
BRACK2	> 50	41.48	84968
ROTOR	> 90	83.43	170869

Table 2. Value of f_C and $\overline{\Lambda}$ for mappings onto the hypercube with 256 vertices using RSB+CPE and SCOTCH.

A complexity analysis of the DRB algorithm shows that, provided that the running time of all graph bipartitioning algorithms is linear in the number of edges of the graphs, the running time of the mapper is linear in the number of edges of the source graph, and logarithmic in the number of vertices of the target graph. This is verified in practice [11]. For instance, on a 150MHz Indigo 2 machine with 32 Mb of main memory, the BCSSTK30 graph with 28924 vertices and 1007284 edges is mapped in 81, 137, and 208 CPU seconds onto hypercubes with 4, 16, and 256 vertices, respectively.

7 Conclusion

In this paper, we have presented the principles and capabilities of SCOTCH, a software package for static mapping which implements the Dual Recursive Bipartitioning algorithm, and allows the user to map any weighted source graph onto any weighted target graph. Due to the graph bipartitioning algorithms used, its running time is linear in the number of source edges and logarithmic in the number of target vertices. Work in progress includes the development of new graph bipartitioning methods, and in particular of multi-level schemes, as in [8], which should further speed-up the mapping process.

References

1. S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.
2. F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitionning. *JPDC*, 10:35–44, 1990.
3. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Autom. Conf.*, pages 175–181. IEEE, 1982.
4. M. R. Garey and D. S. Johnson. *Computers and Intractablility: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, 1979.
5. S. W. Hammond. *Mapping unstructured grid computations to massively parallel computers.* PhD thesis, Rensselaer Polytechnic Institute, feb 1992.
6. B. Hendrickson and R. Leland. The CHACO user's guide. Technical Report SAND93–2339, Sandia National Laboratories, nov 1993.
7. B. Hendrickson and R. Leland. An empirical study of static load balancing algorithms. In *Proceedings of SHPCC'94, Knoxville*, pages 682–685. IEEE, may 1994.
8. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. TR 95-035, University of Minnesota, jun 1995.
9. G. Karypis and V. Kumar. METIS – *Unstructured Graph Partitioning and Sparse Matrix Ordering System – Version 2.0.* University of Minnesota, jun 1995.
10. D. M. Nicol. Rectilinear partitioning of irregular data parallel computations. *Journal of Parallel and Distributed Computing*, 23:119–134, 1994.
11. F. Pellegrini. *Application de méthodes de partition à la résolution de problèmes de graphes issus du parallélisme.* Thèse de Doctorat, Université Bordeaux I, jan 1995.
12. F. Pellegrini. SCOTCH 3.0 User's guide. Technical Report 1095-95, LaBRI, Université Bordeaux I, oct 1995.
13. A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis*, 11(3):430–452, jul 1990.

This article was processed using the LaTeX macro package with LLNCS style