

Serial and Parallel Multilevel Graph Partitioning using Fixed Centers

Kayhan Erciyeş^{1,2}, Ali Alp³ and Geoffrey Marshall¹

¹ California State University San Marcos
333 S.Twin Oaks Valley Rd., San Marcos, CA 92096, U.S.A.
kerciyes@csusm.edu, marsh021@csusm.edu

² İzmir Institute of Technology, Urla, İzmir, TR-35340, Turkey

³ Ege University International Computer Institute
Bornova, İzmir, TR-35100, Turkey
alpali@bornova.ege.edu.tr

Abstract. We present new serial and parallel algorithms for multilevel graph partitioning. Our algorithm has coarsening, partitioning and uncoarsening phases like other multilevel partitioning methods. However, we choose fixed nodes which are at least a specified distance away from each other and coarsen them with their neighbor nodes in the coarsening phase using various heuristics. Using this algorithm, it is possible to obtain theoretically and experimentally much more balanced partitions with substantially decreased total edge costs between the partitions than other algorithms. We also developed a parallel method for the fixed centered partitioning algorithm. It is shown that parallel fixed centered partitioning obtains significant speedups compared to the serial case. ...

1 Introduction

The aim of a graph partitioning algorithm is to provide partitions such that the number of vertices in each partition is averaged and the number of edge-cuts between the partitions is minimum with a total minimum cost. Graph partitioning finds applications in many areas including parallel scientific computing, task scheduling, VLSI design and operation research. One important area of research is on searching algorithms that find good partitions of irregular graphs to map computational meshes to the high performance parallel computer processors for load balancing such that amount of computation for each processor is roughly equal with minimum communication among them. Solution of sparse linear systems where the graph representing the coefficient matrix is partitioned for load balancing among processors is one area that research is directed [1][2]. Recently, graph partitioning algorithms are used in mobile ad-hoc networks to form clusters for dynamic routing purposes [3][4]. In *multi-constraint approach* [5], each vertex is assigned a vector of k weights that represent the work associated with that vertex in each of the k computational phases. The aim is to partition the graph so that each of the k weights is balanced as well as the sum of edge weights are minimized. In the related *multi-objective model* [6], the partition tries to minimize several cost functions at the same time. Each edge is given a

vector of j weights where different cost functions are an element of this vector. The partitioning then tries to balance vertex weights by minimizing the cost functions. In *skewed partitioning* [7], each vertex can have k preference values showing its tendency to be in each one of k sets which is taken into consideration by the partitioning algorithm. Partition refinement is an important step to improve partition quality. Kernighan and Lin (KL) [8] provided a greedy method to swap a subset of vertices among partitions to reduce edge connectivity further. During each step of KL algorithm, a pair of vertices, one from each partition are selected and swapped to give a reduced total edge weight among the partitions if possible. *Multilevel graph partitioning* is a comparatively new paradigm for graph partitioning and consists of coarsening, initial partitioning and refinement steps [9][10]. In the coarsening phase, a set of vertices is selected and are collapsed to form a coarser graph. This step is performed sufficient times to get a simple graph which can be divided into the required partitions by a suitable algorithm. The obtained partitions are projected back by uncoarsening and refinement by algorithms such as KL along this process. Chaco[11], METIS[12] and SCOTCH[13] are popular multilevel partitioning tools used in diverse fields.

In this study, we propose new serial and parallel multilevel algorithms for graph partitioning. We compare these methods with the multilevel graph partitioning method of [10]. In Section 2, background including the multilevel graph partitioning method is reviewed. Section 3 describes our serial graph partitioning algorithm which is called *Fixed Centered Partitioning* (FCP). Parallel implementation of the FCP algorithm on a cluster of workstations is given in Section 4. Finally, experimental results obtained so far are presented and comparison of the algorithm with other algorithms are outlined in Section 5 and the conclusions are given in Section 6.

2 Background

The multilevel graph partitioning model has proven to be very robust, general and effective. The idea is simply approximation of a large graph by a sequence of smaller and smaller graphs after which the smallest graph can be partitioned into p partitions by a suitable algorithm. This partition is then brought back to the original graph by refinements. Consider a weighted graph $G_0=(V_0, E_0)$ with weights both on vertices and edges. A multilevel graph partitioning algorithm consists of the following phases.

Coarsening Phase : In the coarsening phase, the graph G_0 is transformed into a sequence of smaller graphs G_1, G_2, \dots, G_m such that $|V_0| > |V_1| > |V_2| > \dots > |V_m|$. In most coarsening schemes, a set of vertices of G_i is combined to form a single vertex of the next level coarser graph G_{i+1} . Let $V=\{v_i\}$ be the set of vertices of G_i combined to form vertex v of G_{i+1} . The weight of vertex v is set equal to the sum of the weights of the vertices in $V=\{v_i\}$. Also, in order to preserve the connectivity information in the coarser graph, the edges of v are the union of the edges of the vertices in v_i . The *maximal matching* of the graph is a set of edges such that there is not a pair adjacent

on the same vertex. Various approaches to find maximal matching exist. The *heavy edge matching* (HEM) computes the matching M_i such that the weight of the edges in M_i is high. The vertices are visited in random order, but the collapsing is performed with the vertex that has the heaviest weight edge with the chosen vertex. In *Random Matching* (RM), vertices are visited in random order and an adjacent vertex is chosen in random order as well [10]. During the successive coarsening phases, the weight of vertices and edges increase.

Partitioning Phase : The second phase of the multilevel algorithm computes a high-quality partition P_m of the coarse graph $G_m=(V_m, E_m)$ by a suitable algorithm such that each partition contains roughly equal vertex weights and the sum of the weights of the edge cuts between the partitions is minimum.

Uncoarsening/Refinement Phase : During this phase, the partition P_m of the coarser graph G_m is projected back to the original graph, by going through $G_{m-1}, G_{m-2}, \dots, G_1$. Since each vertex of G_{i+1} contains a distinct subset of vertices of G_{m-1} , obtaining P_i from P_{i+1} is done by simply assigning the set of vertices collapsed in G_{i+1} to the partition $P_{i+1}[v]$. Algorithms such as KL are usually used to improve partition quality.

3 Serial Fixed Centered Partitioning

The method we propose has coarsening, partitioning and uncoarsening phases as in the other multilevel partitioning methods. We however choose fixed initial nodes called *centers* and collapse the vertices around these centers which must have at least a fixed distance to the other selected center nodes. The FCP algorithm is described in Section 3.1, the formal analysis of the algorithm is stated in Section 3.2. and an example partition is given in Section 3.3.

3.1 Serial FCP Algorithm

The Serial FCP algorithm can be formally described as in Fig.1. Inputs to the algorithm are the initial graph G_0 , number of partitions required and the two heuristics, HC and HM. Since we have fixed centers that do not change as the graph gets coarsened, a way to allocate these centers initially is needed. The first approach we employed is to run *Breadth-First-Search* (BFS) algorithm for all the nodes in the graph and find p center nodes which have the maximum distance between them. BFS, however, is time consuming as it has a runtime of $O(n^3)$. Secondly, we may choose the centers randomly with the constraint that each center has at least some predetermined distance among them. The third approach chooses the centers randomly with no constraints. The minimum distance heuristic h_1 between any two centers may be associ-

ated to the diameter value of the graph and the number of partitions by $h_1 = 2d/p$ where d is the diameter of the graph and p is the number of partitions required. The possible heuristics used to locate the centers initially could be summarized as follows:

- HC_1 : Apply Breadth-First-Search (BFS) to G_0 and find p centers that are $2d/p$ distance from each other
- HC_2 : Choose centers randomly with the condition that they are at least $2d/p$ distance from each other
- HC_3 : Choose the centers at random with no constraints

Once the centers are chosen, FCP proceeds by collapsing a neighbor vertex at each iteration to the fixed centers as shown in Fig.1 using a second heuristic, HM. Two possible values for HM are the *Heaviest Edge Neighbor* (HEN) or *Random Neighbor* (RN). Based on the heuristic chosen, the *Collapse* function performs the collapse operation of a marked neighbor node with the center which simply merges the marked vertex to the center by adding its weight to the center, removing the edge between them and inserting any previously coexisting edges between them by adding the weights of the edges and representing them as a single edge with this weight.

```

Procedure Serial_FCP
  Input :  $\tilde{G}_0$  : initial graph
          $p$  : number of partitions
          $HC$  : heuristic to allocate initial centers
          $HM$  : heuristic to mark neighbor nodes
  1. Locate_centers( $G_0$ ,  $HC$ );
  2. for  $i=1$  to  $\lfloor n/p \rfloor$  do
  3.   for each center  $c$  do
  4.     Collapse( $G_i$ ,  $c$ ,  $HM$ );

```

Fig. 1. Serial FCP Algorithm

3.2 Analysis of Serial FCP

The time complexity and the quality of the partitions of the Serial FCP can be stated in the following theorems:

Theorem 1 : FCP performs partitioning of $G(V,E)$ in $O(\lfloor n/p \rfloor)$ steps where $|V| = n$ and p is the number of partitions required. The time complexity of the total collapsing of FCP is $O(n)$.

Proof : FCP simply collapses p nodes with its heaviest edges at each step resulting in $\lfloor n/p \rfloor$ steps. Since there are p collapsing at each step, total time complexity is $O(n)$.

Corollary 1 : FCP performs partitioning of $G(V,E)$ such that the final partitions have $O(\lfloor n/p \rfloor + 1)$ vertices

Proof : The FCP collapses one node to each partition and the total number of steps is $O(\lfloor n/p \rfloor)$ by Theorem 1. In the last step, there will be $O(p \text{ MOD } n)$ nodes to collapse which means that the final partitions will have a maximum of $n/p+1$ nodes.

3.3 FCP examples

Let us illustrate RM, HEM and FCP by the example shown in Fig. 1 where (a) is RM, (b) is HEM, (c) is FCP and heavy lines are used to show matchings. The initial graphs and outputs of RM and HEM are reproduced from [14]. The output graphs are formed after $\Theta(5)$ collapses for RM and HEM but $\Theta(6)$ for FCP after two steps. For this particular example, we see that FCP performs much better with a total edge cost of 16 compared to RM (30) and HEM (24). We also get 3 vertices per partition with respect to 2 vertices in RM and HEM. If three partitions were required, we would have stopped for FCP but continue with matching for CM and HEM. Moreover, FCP does not have a matching phase, therefore it has much better runtimes than RM and HEM.

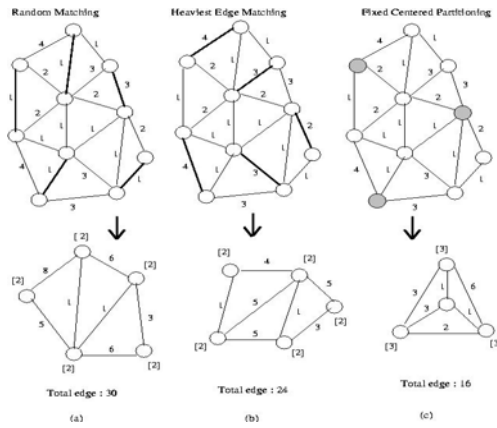


Fig. 2. Comparison of RM (a), HEM (b) and FCP (c) in an arbitrary network

4 Parallel Fixed Centered Partitioning

The proposed parallelization of FCP consists of three phases. In the first phase, the determination of the diameter of the network is done in parallel. The coordinator sends the adjacency list of the graph to individual workers and each worker then estimates its local diameter of the graph by performing BFS on its local partition. The coordinator gathers all of the local diameters and estimates the total diameter. It then locates the centers based on this diameter and sends the identities of the centers to each processor. In the second phase, each processor collapses the graph around its designated center independently until a predetermined $h2$ times such that no overlap would occur. The heuristic $h2$ used is set as d/p^2 for the implementation. In the third

phase, each processor attempts to collapse possibly overlapping regions with others. Therefore, every time a number of nodes are to be collapsed, acknowledgment from the coordinator is searched to check that these nodes have not been collapsed before. The coordinator and worker pseudocodes are shown in in Fig. 3 and Fig. 4.

```

Process Parallel_FC_Coordinator

Input :  $G_0$  : initial graph
         $p$  : number of partitions
         $HC$  : heuristic to allocate initial centers
         $HM$  : heuristic to mark neighbor nodes

1.  /* Locate Centers */
2.  Send adjacency list and their identities to slaves
3.  Receive local diameters from all slaves
4.  estimate diameter of the graph and determine center nodes
5.  Send center nodes to the slaves.
6.  /* Wait for local collapses */
7.  forall workers
8.    Receive the collapsed nodes from the worker
9.    while there are nodes to be collapsed /* Check Overlaps */
10.     Receive node identities from slaves
11.     Send COLLAPSED or NOT_COLLAPSED to workers
12.     mark NOT_COLLAPSED nodes as collapsed

```

Fig. 3. Coordinator pseudocode for Parallel FCP

```

Process Parallel_FC_Worker

Input :  $G_0$  : initial graph
         $p$  : number of partitions
         $HM$  : heuristic to mark neighbor nodes

1.  Determine Local Diameter;
2.  Receive adjacency list and my_index;
3.  Find the local diameter;
4.  Send local diameter to coordinator;
5.  Receive diameter and my_center from master
6.   $h_2 = d/p^2$  /* Local Collapses */
7.  Collapse nodes until distance  $h_2$  from my_center;
8.  Send the identities of the collapsed nodes to the master;
9.  /* Overlapping Collapses */
10. while any uncollapsed neighbor of mycenter exists
11.   Send neighbor to master for checking;
12.   Receive check from master;
13.   if (check== neighbor not collapsed)
14.     Collapse neighbor and mark it as collapsed;

```

Fig. 4. Worker Pseudocode for Parallel FCP

5 Results

5.1. Results for Serial Centered node Matching

We implemented the graph partitioning using HEM, RM and FCP (alternatively called Centered Matching - CM) for various randomly created matrix sizes (128*128, 256*256, 512*512, 1024*1024, 2048*2048). The graphs represented by the matrices are partitioned on Ultra 5 Sun Sparc servers which run Solaris7 as operating system and runtimes of partitioning algorithms are compared. Center nodes in FCP are found by two different heuristics as HC_1 and HC_2 . by running the BFS algorithm for all nodes or randomly choosing and checking for a distance between as described in Section 3.1. As shown in Fig. 5, the first FCP (CM with random center) method is the fastest as expected since FCP does not have a matching phase. The second FCP method (CM with BFS) is the slowest because BFS is executed on all nodes of the graph to find center nodes. In Fig. 6, the total edge costs between the partitions are plotted for FCP (with HC_2 and HC_3) and HEM and RM. It may be seen that both FCP methods have a significant decreased total edge costs between the partitions.

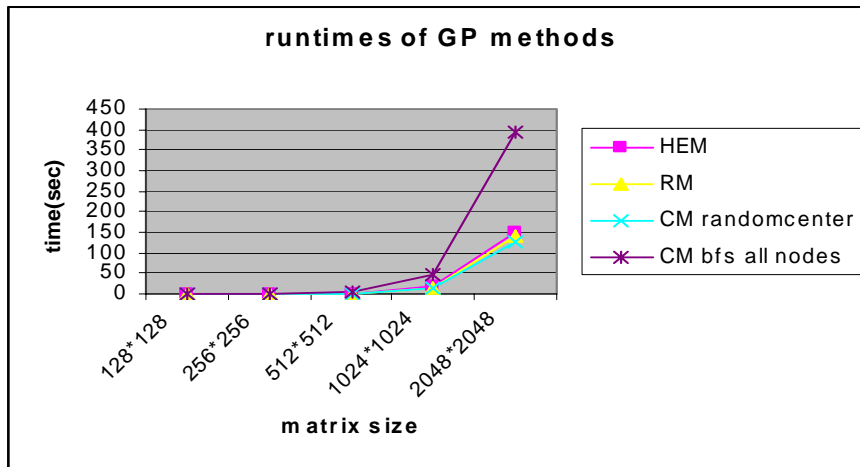


Fig. 5. Execution Times for the Graph partitioning methods

Fig. 7 depicts the number of nodes in each partition with FCP (HC_2), HEM and RM. As shown, FCP generates much more balanced partitions than other methods. This is because of the operation principle of FCP where the centers are visited in sequence as stated in Corollary 1.

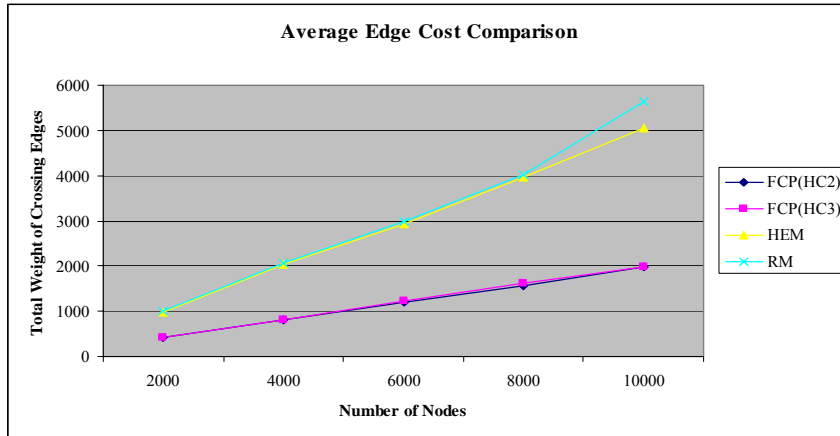


Fig. 6. Edge Cost Comparison of the Four Algorithms

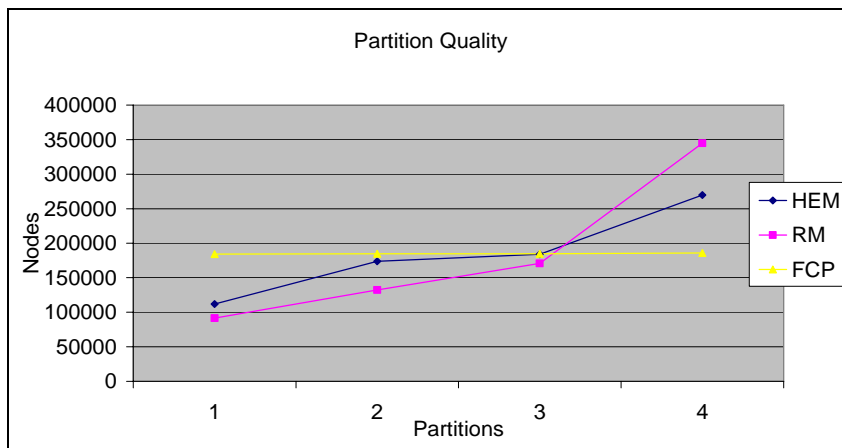


Fig. 7. Number of nodes in each part for all graph partitioning methods(2048*2048 matrix).

5.2. Results for Parallel Centered node Matching

We implemented the parallel FCP method on a network of workstations running Parallel Virtual Machine (PVM) v3.4.4. The processors find their local diameters and coarsen their neighbor nodes around their local centers to partition the graph. A coordinator and 2,4,5,6 worker workstations are used in the same VLAN over a Gigabit Ethernet. All of the servers are Ultra 5 Sun Sparc running Solaris 7. Workers communicate with the coordinator using a point-to-point protocol. For various graph sizes, the computational run times are recorded. Fig. 8 displays the results of the Parallel FCP for various number of processors ranging from 1 to 6. It may be seen

that after a threshold value of the number of workers, the communication costs become dominant.

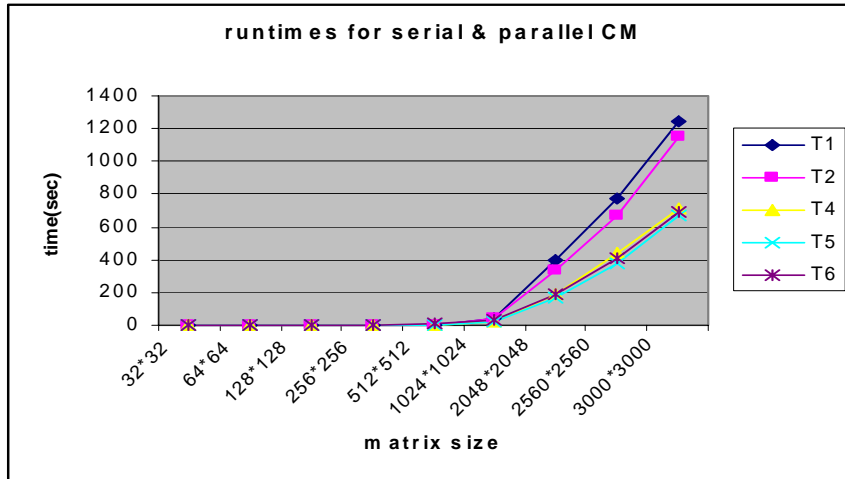


Fig. 8. Comparison of serial and parallel runtimes of FCP. T_1 : serial runtime. T_i represents the execution time with i workers.

6. Conclusions

We proposed a new graph partitioning algorithm called FCP and compared this method with other methods such as HEM and RM. FCP provides more favorable partitions than the other methods theoretically and experimentally. However, FCP needs to assign some nodes in the graph as center nodes which have at least a certain distance to each others. The experimental results confirmed the theoretical FCP properties in terms of the runtime, total edge cost between the partitions and the partition quality. FCP using any heuristic performed much better than HEM and RM in terms of total edge cost and partition quality. For the runtime, FCP with HC2 and HC3 resulted in lower times than HEM and RM but FCP with HC1 proved to be the slowest as expected. FCP does not have a matching phase which results in a faster execution time and divides the graph into almost equal partitions as stated in Corollary 1. We also developed a parallel version of FCP. In this case, the diameter of the graph is estimated after each processor finds their local diameters and then the centers can be found by the coordinator. After finding centers where each of is assigned to different workers, each workstation continues collapsing the neighbor nodes of its center until there are no uncollapsed nodes left by getting acknowledgement from the coordinator after some predetermined value. The efficiency of the parallel algorithm rises for larger graphs until a threshold value where the communication costs start to dominate.

Our general conclusions can be summarized as follows. FCP provides improvement over the other graph matching algorithms such as RM and HEM in three aspects. Firstly, it does not have a matching phase, therefore it is faster. Secondly, it provides almost equal partitions with significantly lower total edge costs between the partitions than the other methods and thirdly it is suitable for parallel processing as each center collapses independently. One negative aspect of FCP is the initial marking of the fixed centers and random allocation could be a solution in this case. Also, the parallel algorithm employed requires a heuristic to be able to perform collapsing without any supervision initially. We are looking into more efficient ways of finding centers and performing FCP in parallel. Another interesting research direction would be modifying the FCP to perform multi-constraint, multi-objective graph partitioning. In this case, several cost functions would need to be minimized when choosing the vertices to collapse.

References

1. Hendrickson, B. and Kolda, T., G. : Partitioning Rectangular and Structurally Nonsymmetric Sparse Matrices for Parallel Processing, *SIAM J. Sci. Comput.* 21(6), (2000), 2048-2072.
2. Turhal, B., Solution of Sparse Linear Systems on a Cluster of Workstations Using Graph Partitioning Methods, Msc. Thesis, Ege University, Int. Computer Institute, (2001)
3. P. Krishna et al, A Cluster-based Approach for Routing in Dynamic Networks, *ACM SIGCOMM Computer Communication Review*, 27 (2), (1997), 49 – 64.
4. Erciyes, K., Marshall, G. : A Cluster based Hierarchical Routing Protocol for Mobile Networks, *LNCS, Springer-Verlag, ICCSA(3)*, (2004), 528-537.
5. Yuanzhu P. C., Liestman, A., L. : A Zonal Algorithm for Clustering Ad Hoc Networks, *Int. J. Foundations of Computer Science*, 14(2), (2003), 305-322.
6. Schloegel, K, Karypis, G., Kumar, V. : A New algorithm for Multi-objective Graph Partitioning, Tech. Report 99-003, University of Minnesota, Dept. of CS, (1999).
7. Hendrickson, B., Leland, R., Driessche, R., V. : Skewed Graph Partitioning, *Proc. of 8th SIAM Conf. Parallel Processing for Scientific Computing*, SIAM (1997).
8. Kernighan, B., and Lin, S., An Effective Heuristic Procedure for Partitioning Graphs, *The Bell System Technical Journal*, (1970), 291-308.
9. Hendrickson, B., Kolda, T., G. : Graph Partitioning Models for Parallel Computing, *Parallel Computing*, 26, (2000), 1519-1534.
10. Karypis, G., Kumar, V. : A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, Tech. Report 95-035, University of Minnesota, Dept. of CS, (1995).
11. Hendrickson, B., Leland, R. : The Chaco User Guide, V. 2.0, Technical report SAND94-2692, Sandia National Labs., (1994).
12. Karypis, G., Kumar, J. : METIS,1.5 : A Hypergraph Partitioning package. Tech. Report, Univ. of Minnesota, Dept. of CS, (1998).
13. Pellegrini, F., Roman, J. : SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of process and Architecture Graphs. *HPCN-Europe, LNCS, Springer-Verlag*, 1067, (1996), 493-498.
14. Schloegel, K., Karypis, G., Kumar, V. : Graph Partitioning for High Performance Scientific Simulations, Tech. Report 00-018, University of Minnesota, Dept. of CS, (2000).