A k-way Graph Partitioning Algorithm Based on Clustering by Eigenvector

Tae-Young Choe[1] and Chan-Ik Park[2]

[1] School of Computer Engineering, Kumoh National Institute of Technology, 188, ShinPyung Dong, Gumi, KOREA 730-701. choety@kumoh.ac.kr
[2] Department of Computer Science and Engineering, Pohang University of Science and Technology, San 31, HyoJa Dong, Pohang, KOREA 790-784. cipark@postech.edu

Abstract. The recursive spectral bisection for the k-way graph partition has been underestimated because it tries to balance the bipartition strictly. However, by loosening the balancing constraint, the spectral bisection can identify clusters efficiently. We propose a k-way graph partitioning algorithm based on clustering using recursive spectral bisection. After a graph is divided into a partition, the partition is adjusted in order to meet the balancing constraint. Experimental results show that the clustering based k-way partitioning generates partitions with $83.8 \sim 108.4\%$ cutsets compared to the strict recursive spectral bisections or multi-level partitions.

1 Introduction

Given a graph $G = (V, E)$, where V is the set of $|V| = n$ vertices and E is the set of $|E| = e$ edges, the k-way graph partition is composed of disjoint k subsets, where the union of the subsets is V. We assume all vertices and edges are same size one. An edge whose endpoints are located in different subsets is called the *cut* edge. The *cutset* of a partition is the set of cut edges. The graph partitioning problem is to divides V into a balanced partition with the minimum cutset size. The graph partitioning problem is known as NP-hard even in 2-way partitioning [1]. Therefore, the partitioning algorithms heuristically find approximated solutions within acceptable computation time.

The multi-level partitioning algorithm, one of the clustering-based algorithms, partitions a graph in three phases: coarsening phase, partitioning phase, and uncoarsening phase [2]. They rely on local information like adjacent vertices and edges to determine candidates to be merged. The spectral method partitions graph using eigenvectors of the Laplacian matrix of the graph. It utilizes global information and have been widely used [3]. Because many graphs are not balanced, balanced partitions by the spectral method can be local optima.

If we do not stick on the balancing constraint, the spectral method is especially efficient for clustering. From the position of vertices expressed by the Fiedler vector, some relations between vertices are revealed: dense areas and parse areas. Based on these facts, we divide a graph into clusters by setting a

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3037, pp. 598–601, 2004.

sparse area as a border between clusters. The remainder of the paper is organized as follows; Section 2 presents the clustering methods and a partitioning algorithm. Section 3 provides experimental results. Finally, Section 4 concludes the paper.

2 The Proposed Algorithm

Our partitioning algorithm is just composed of three sequential steps: given graph is clustered considering vertices connectivity in clustering step, the clusters are fitted to balanced blocks in balancing step, and finally, a simple partitioning algorithm refines the balanced blocks in refinement step.

2.1 Clustering Step

The base architecture of the clustering step is the recursive spectral method [4]. The main difference between the proposed step and other recursive spectral bisection algorithms is that our clustering step attaches more importance to clustering with smaller cutset rather than balanced partition.

Figure 1 shows function Clustering() for the clustering step. Element f_i in the Fiedler vector F_C is considered as an one-dimensional coordinate of vertex v_i in G_C. By projecting the vertices into one-dimension, we can see relative positions and connectivities between vertices. That is, as two vertices are more strongly connected each other, they are laid more closely in the one-dimension, and vise versa.

Function Clustering(G, k)
  //INPUT: graph $G = (V, E)$ to be clustered and k the number of clusters
  //OUTPUT: a set of clusters $\mathcal{C}$
  //CONSTANT l: the number of subregions
  $\mathcal{C} \leftarrow \{V\}$;
  while $|\mathcal{C}| < k$ **do**
    $C \leftarrow$ the largest cluster in $\mathcal{C}$;
    $G_C \leftarrow$ subgraph composed of C;
    compute Fiedler vector F_C of G_C;
    divide section $[\min_{f_i \in F_C} f_i, \max_{f_i \in F_C} f_i]$ to l subregions;
    partition G_C into $C_1, C_2, \ldots, C_k$ according to partition policy;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_1, C_2, \ldots, C_k\} - \{C\}$;
  end while
  return ($\mathcal{C}$);

Fig. 1. Clustering step

In order to find clusters in a graph, we simplify the method proposed by Hagen and Kahng [5]. Our algorithm divides the entire region of the vertices coordinates into same sized subregions and counts the number of vertices in

Table 1. The cutset size and execution time of result partition of each algorithm

measure		cutset				run time	
No. subset	graph	Chaco	Sanchis	Metis	Proposed	Metis	Proposed
2	hammond	105	233	109	104	0.02	1.25
	barth5	164	204	149	**147**	0.09	5.28
	brack2	**747**	1886	779	1006	0.62	23.85
8	hammond	590	414	**376**	383	0.03	4.90
	barth5	**733**	970	744	823	0.09	44.00
	brack2	8986	10934	8802	**8341**	0.68	111.03
32	hammond	1174	1250	1066	**1047**	0.05	8.46
	barth5	1853	2160	1823	**1716**	0.13	66.34
	brack2	21674	24705	**18898**	20690	0.77	243.75

each subregion. The routine sets the initial number of subregions as the number of vertices $|C|$. Adjacent two subregions are merged until every subregions has one or more vertices. After the merge operations, we identify a cluster boundary which is a subregion with the least vertices If there are two or more subregions with the least vertices, a border that generates most balanced clusters is selected.

2.2 Balancing Step

In general, clusters generated during clustering step are unbalanced, even some cluster could be splited. Thus balancing step is indispensable for partition to meet the balancing constraint. We modify and combine Sanchis algorithm [6] and P-P algorithm [7]. Gain of a movement for a vertex to a subset is the number of removed cutsets and improvement in degree of balance [7]. The gain is managed by bucket data structure [6].

2.3 Refinement Step

The refinement step runs on the partition that satisfies the balancing constraint. The remainder is to reduce the cutset size further. Thus, Sanchis algorithm is a good refinement algorithm to reduce the cutset size. Since the partition becomes globally near optimal by the clustering step, it is not needed to consider escape from local optima. Thus movement of cells with negative gain is not necessary.

3 Experimental Results

The algorithms are implemented in C language. Since the programming code uses the standard C libraries, it can run in almost all systems. The Chaco, Sanchis algorithm, and the Metis algorithms are executed to compare the performance with our algorithms. Among the many options of the Chaco partitioning algorithm, recursive spectral octal-partition and K-L refinement are used. The Metis

runs with default options. The balancing constraint is 3% of $|V|/k$. Three test graphs are used: hammond with 4720 vertices and 13722 edges, barth5 with 15606 vertices and 45878 edges, and brack2 with 62631 vertices and 366559 edges.

The results of the algorithm executions are shown in Table 1. The data in Sanchis column show partition without clustering step. That is, it starts with a partition that all vertices are in one subset, runs the min-bal balancing steps, and finishes with Sanchis refinement. Each bold fonted number means that the number is the smallest cutset given mesh and the number of subsets. Execution times of Metis and the proposed algorithm are compared. A large proportion the execution times in the proposed algorithm is the refinement step. It shows that the refinement step need some performance tuning as future works.

4 Conclusions

We proposed an effective partitioning algorithm that generates a partition with a relatively small cutset. The algorithm clusters the graph using the recursive unbalanced bipartition.

Acknowledgments. The authors would like to thank the Ministry of Education of Korea for its support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program. This research was also supported in part by HYSDR IT Research Center and in part by grant No. R01-2003-000-10739-0 from the Basic Research Program of the Korea Science and Engineering Foundation.

References

1. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Information Processing Letters **42** (1992) 153–159
2. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20** (1999) 359–392
3. Simon, H.D., Sohn, A., Biswas, R.: Harp: A dynamic spectral partitioner. Journal of Parallel and Distributed Computing **50** (1998) 83–103
4. Hsieh, S.H., Paulino, G.H., Abel, J.F.: Recursive spectral algorithms for automatic domain partitioning in parallel finite element analysis. Comput. Methods Appl. Mech. Engrg. (1995) 137–162
5. Hagen, Kahng: New spectral methods for ratio cut partitioning and clustering. IEEETCAD: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **11** (1992)
6. Sanchis, L.: Multiple-way network partitioning. IEEE Trans. Computers **38** (1989)
7. Park, C.I., Park, Y.B.: An efficient algorithm for vlsi network partitioning problem using a cost function with balancing factor. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **12** (1993)