International Journal of Current Trends in Engineering & Research (IJCTER)
e-ISSN 2455–1392 Volume 2 Issue 3, March 2016 pp. 12 - 16
Scientific Journal Impact Factor : 3.468
http://www.ijcter.com

Multithreaded Graph Partitioning Based Efficient Network Optimization

Himanshi Ganatra[1], Twinkle Pardeshi[2] , Priyanka Wagh[3], Prajakta Wailkar[4]

[1,2,3] *Department of Information Technology*
MET's Institute of Engineering
Nashik, India

Abstract—In this paper we explore the design space of creating a multi-threaded graph partitioner. We present and compare multiple approaches for parallelizing each of the three phases of multilevel graph partitioning: coarsening, initial partitioning, and uncoarsening. We also explore the differences in thread lifetimes and data ownership in this context. We show that despite the options for fine grain synchronization and task decomposition offered by current threading technologies, the best performance is achieved by preserving data ownership and minimizing synchronization. In addition to this we also present an unprotected approach to generating a vertex matching in parallel with little overhead. We use these findings to develop an OpenMP based implementation of the Metis algorithms and compare it against MPI based partitioners on three different multicore architectures. Our multithreaded implementation not only achieves greater than a factor of two speedup over the other partitioners, but also uses significantly less memory.

Keywords— Graph Partitioning, coarsening, initial partitioning, uncoarsening, optimize graph

I. INTRODUCTION

Graph partitioning is used as a first step in divide & conquers approaches to reduce complexity, task decomposition for parallel systems, and data decomposition for distributed systems. As the capacity to collect information and to model complex systems has soared, so has the size and diversity of the graphs that are being generated and need to be processed and analyzed. This growth in scale and scope presents new challenges to partitioning these graphs quickly and efficiently. Graph Partitioning comes with various outcomes in today world.

Graph partitioning is used as a first step in divide & conquers approaches to reduce complexity, task decomposition for parallel systems, and data decomposition for distributed systems. There are various phases included in graph partitioning those are namely coarsening, initial partitioning and uncoarsening. In coarsening phase, the original graph is partitioned into number of finer graphs so that individual graph is executed by different threads ion parallel. The next phase that is initial partitioning uses GGP and GGGP techniques for bisection of graph. In last uncoarsening phase, the finer graphs are projected back to original but optimized graph.

However, graph partitioning is a well-studied problem and as a result, many fast heuristic algorithms exist for finding high-quality partitioning. The multilevel paradigm is a widely used approach in graph partitioning libraries including Chaco, Metis, Jostle, and Party. In multilevel a graph is divided into sub graph appropriate to original graph. The sub graphs are then processed to get the original but optimal original graph that would reduce the access time and increase the speed.

In this paper design space of creating a multi-threaded graph partitioner is explored. Various algorithms are studied and compare for the key steps of the multilevel partitioning paradigm that take different strategies in terms of synchronization frequency, task granularity, data ownership, and thread lifetime. Experiments will be on three different multicore architectures using the threading functionality provided by OpenMP, even though multicore architectures allow for fine grain task

decomposition and frequent synchronization, the best performance is achieved when the task decomposition is coarse and synchronization is infrequent. In addition to this it was explore that data locality, which when using OpenMP can only be controlled implicitly by having threads operate on the data they generate, is also crucial to achieving performance on a large number of cores. These findings are to a large extent consistent with the best practices of high performance parallel algorithms used on distributed memory machines.

II. SYSTEM ANALYSIS

The *k-**way*** graph partitioning problem is given as follows: Given a graph $G = (V, E)$ with $|V| = n$, partition V into k subsets, $V1, V2, \ldots, Vk$ such that $Vi \cap Vj = \emptyset$ for $i \neq j$, $|Vi| = n/k$, and Ui $Vi = V$, and the number of edges of E whose incident vertices belong to different subsets is minimized. A k-way partition of V is commonly represented by a partition vector P of length n, such that for every vertex $v \in V$, $P[v]$ is an integer between 1 and k, indicating the partition at which vertex v belongs. Given a partition P, the number of edges whose incident vertices belong to different subsets is called the edge-cut of the partition.

The graph G can be bisected using a multilevel algorithm. The basic structure of a multilevel algorithm is very simple. The graph G is first coarsened down to a few hundred vertices, a bisection of this much smaller graph is computed, and then this partition is projected back towards the original graph (finer graph), by periodically refining the partition. Since the finer graph has more degrees of freedom, such refinements usually decrease the edge-cut.

KMetis stores the graph in a structure with the addition of a vector storing the vertex weights, the partition vector P, and the vertex mapping vector C. The coarsening phase in KMetis is made up of two parts those are matching and contraction. The vertices are visited in ascending order according to their degrees. Vertices with the equal degree are visited in random order. If a visited vertex v has already been matched, it is skipped. Otherwise the neighbors of v are searched for the unmatched neighbor u connected by the heaviest edge and is recorded in the C vector. Then in contraction, two matched vertices v, u will share the same mapping to the coarse vertex c. The vertex weight of c is equal to the sum of the weights of v and u. The coarse edges connected to c are the fine edges connected to

v and u minus the contracted edge $\{u,v\}$. Where multiple edges connect the same two coarse vertices, they are reduced to a single edge with a weight equal to the sum of all the weights of the combined edges.

ParMetis is an MPI based parallel formulation of KMetis. In this the initial partition is made through recursive bisection. The processes perform an all-to-all broadcast resulting in each process having all of Gm. Then using the same random seed they each generate a branch of the bisection tree required to create the k-way partitioning. That is, all processes generate the initial bisection of the graph, then half of the processes generate a bisection of the left partition and half of the processes generate a bisection of the right partition, and so forth. These partitionings are then assembled so that each process has all k partitions applied to its local part of Gm.

In Uncoarsening phase, each process projects the partition information from the coarse vertices $Vi+1$ that it owns to the fine vertices Vi that it owns. It also transmits the partition information of coarse vertices that it owns and contain fine vertices from other processes.

1. Coarsening Phase

In coarsening phase original graph is divided into smaller graphs. There are two approaches for coarsening phase first one is based on random matching and collapsing the matched vertices into a

multimode and second based on creating multinodes that are made of groups of vertices that are highly connected. In the graph *Gi = (Vi , Ei)*, the edge between two vertices is collapsed and a multinode consisting of these two vertices is created. For this purpose matching is done. It is nothing but set of edges, no two of which are incident on same vertex. Thus, the next level coarser graph *Gi*+1 is constructed from *Gi* by finding matching of *Gi* and collapsing the vertices being matched into multinodes.

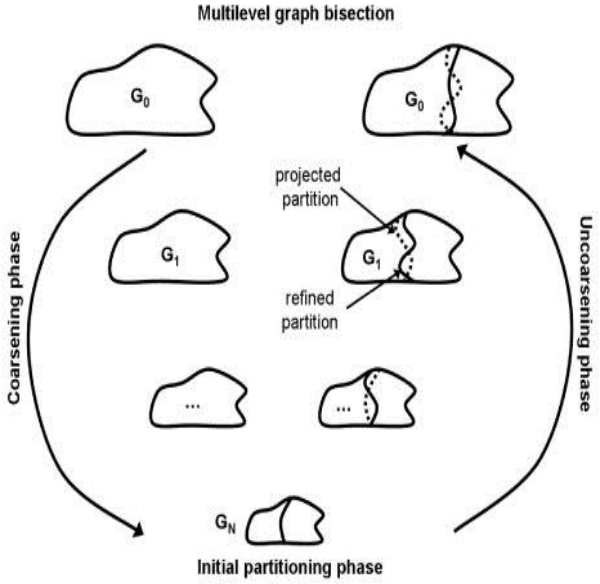

Figure1. System Architecture

2. Coarsening Phase

In coarsening phase original graph is divided into smaller graphs. There are two approaches for coarsening phase first one is based on random matching and collapsing the matched vertices into a multimode and second based on creating multinodes that are made of groups of vertices that are highly connected. In the graph *Gi = (Vi , Ei)*, the edge between two vertices is collapsed and a multinode consisting of these two vertices is created. For this purpose matching is done. It is nothing but set of edges, no two of which are incident on same vertex. Thus, the next level coarser graph *Gi*+1 is constructed from *Gi* by finding matching of *Gi* and collapsing the vertices being matched into multinodes. The unmatched vertices are copied over to *Gi*+1. The matching of maximal size is called maximal matching.

Random Matching (RM)

In this matching algorithm, the vertices are visited randomly .If any vertex stays unmatched then we randomly then one of its unmatched adjacent vertices
are selected. Matched vertices are marked as matched. If for example if an unmatched adjacent vertex is not present *v*, then vertex *u* remains unmatched.

Heavy Edge Matching (HEM)

In this, vertices are visited in random order and vertex *u* is matched with vertex *v* such that the weight of the edge (*u*,*v*) is maximum.

Light Edge Matching (LEM)

Instead of selecting the edge with maximum weight, edge with minimum weight is selected here. In this, vertices are visited in random order and vertex *u* is matched with vertex *v* such that the weight of the edge (*u*,*v*) is minimum.

Heavy Clique Matching (HCM)
Here a ***clique*** of an unweighted graph *G = (V, E)* is a fully connected subgraph of *G*. In this scheme matching is performed by collapsing vertices that have high edge density. In short, this scheme computes a matching whose edge density is maximal.

3. Partitioning Phase
The second phase of graph partitioning is partitioning phase. Various algorithms are used obtain a partitioned graph.
There are three algorithms for partitioning phase, first and foremost algorithm uses the spectral bisection and second and third algorithm uses graph growing heuristics. In first GGP, it firstly select a vertex randomly *v* and creates a region around it in breadth-first fashion till half of the vertex weight has been included. In second GGGP, it also select a vertex *v* randomly but it include only those vertices that leads to the smaller increase in edge cut. The quality depends upon the choice of vertices *v* and different partitions are computed.

4. Uncoarsening Phase
In coarsening phase, the partition *Pm* of coarser graph is projected back to original graph that is the optimized graph. In this phase different refinement algorithms are used to get the original graph.

Kernighan-Lin Refinement
This refinement algorithm decreases the edge cut with a minimum number of iteration. In KL algorithm a single iteration stops as soon as *x* swaps are performed that do not decrease the edge cut. Here te number of vertices swapped in each iteration are very small as compare to in other algorithms. KL algorithm terminates after a small percentage of vertices have been swapped which results in significant savings in total execution time. This algorithm has been found to be effective in finding locally optimal partitions when it starts with a fairly good initial partition.

Greedy Refinement
KL algorithm terminates when there are no further improvement seen in the iteration. Much time is required in KL algorithm to insert the vertices in appropriate data structures. Even though we significantly reduced the number of vertices that are swapped, the overall complexity does not change in asymptotic terms. Hence, Greedy refinement algorithm is use which takes the advantage of only the first iteration of KL algorithm. To overcome the complexity drawback of KL algorithm, Greedy refinement algorithm is developed.

Boundary Refinement
Boundary refinement algorithm is mostly used for boundary vertices. Initially, only the gains of boundary vertices are inserted in the data structures. In KL algorithm after the swap is performed, gain of the adjacent vertices not yet been swapped are updated in the data structures. Since if any of the vertices during swapping becomes the boundary vertices, it is to be inserted in the data structure if it has a positive gain. The main advantage of boundary refinement algorithm is that the vertices are inserted into data structures as needed and no work is wasted. This is the major advantage over KL algorithm.

III. CONCLUSION

Many automatic approaches were used for code reorganization in the last several years. Code reorganization is the division of the software into modules, publication of the API (Application Programming Interface) for the modules, and then requiring that the modules access each other's resources only through the published interfaces. Here, the metrics that measure the quality of modularization of a software system are used. A set of design principles are proposed to capture the

concept of modularity and defined metrics centered on these principles. The values of metrics are used to decide whether the software system is properly modularized or not.

REFERENCES

[1] Dominique LaSalle and George Karypis, Multi-Threaded Graph Partitioning.
[2] G. Karypis and V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, in Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM), ser. Supercomputing 96. Washington, DC, USA: IEEE Computer Society, 1996. [Online].
[3] C. Walshaw and M. Cross, Optimisation Algorithms for Multilevel Mesh Partitioning, Parallel Compution, vol. 26, no. 12, pp. 16351660, 2000.
[4] Tanveer Zia, Albert Zomaya, "Security Issues in Wireless Sensor Networks", it icsnc, pp.40, International Conference on Systems and Networks Communication (IC-SNC'06), 2006.
[5] P. Sanders and C. Schulz, "Distributed evolutionary graph partitioning", in ALENEX, D. A. Bader and P. Mutzel, Eds. SIAM / Omnipress, 2012, pp. 1629.
[6] G. Karypis and V. Kumar," Multilevel graph partitioning schemes", in ICPP (3), 1995, pp. 113122.

