

Computing Breadth First Search in Large Graph Using hMetis Partitioning

Sahar Idwan

*Computer Science and application Department
Hashemite University, Zarqa, Jordan*
E-mail: sahar@hu.edu.jo, sahar700@hotmail.com
Tel: +96253903333; Fax: +96253826613

Wael Etaiwi

*Computer Science and application Department
Al-Balqa' Applied University, Al-Salt, Jordan*
E-mail: waelcis@yahoo.com
Tel: +962795744288

Abstract

In this paper, we present a new algorithm for the Breadth first search traversing based on hMetis partitioning. hMetis is a hyper graph partitioning algorithm that divides the massive graph into sub-graphs. Our heuristic approach of computing the breadth first search (h_BFS) starts at the partition that contains the source node then processing the other fragments piece by piece based on the border edges. Experimental results show that proposed algorithm is simpler and faster of traversing the large graph that does not fit in the memory based on the time and the number of I/O operations.

Keywords: h_BFS, BFS, hMetis, Graph

1. Introduction

Numerous applications such as networks, data mining, and web applications represented as a graph. These applications need graph search; checking thoroughly all nodes in a graph with the goal of finding a specific node or one with a given property. Different techniques are adopted for graph searching such as Breadth first Search (BFS) and Depth First Search (DFS). It is easy to apply the BFS on small graph since it fits in the memory, as for large graph, BFS poses many challenges related with its storage in the memory and the I/O operations. Different parallel and distributed algorithms are introduced to overcome the difficulties related with BFS on massive graph which move all computation to the set of processors.

This paper introduces a heuristic approach to compute the BFS (h_BFS) on large graph based on the RAM model. The main idea is dividing the large graph into set of sub-graphs that fit in the memory by using hMetis. The partition techniques reduce the number of crossing edges between the sub-graphs, which reduce the number of I/O operations.

This paper is organized as follows. In Section 2, we briefly review the related work on computing the breadth first search for large graph. Section 3 describes the proposed h_BFS algorithm.

The implementation and the experimental results are discussed in Section 4 and section 5 concludes the paper

2. Previous Research

The first external memory algorithm for breath-first search (BFS) was given by Ulrich (2001), which achieves $o(n)$ I/Os on arbitrary undirected graphs with n nodes and maximum node degree d .

Parallel algorithms for breadth first search for directed and undirected graph on the MTA-2 was presented by David (2006). MTA-2 is the first parallel machine they used to implement their algorithms.

D. Ajwani, and others (2005), (2006), and (2007) discussed different algorithms to compute the breadth first search on large graph such as MR_BFS, and MM_BFS. They implemented the deterministic variant of MM_BFS, which outperforms the randomized variants.

A distributed breadth first search is discussed by A. Yoo (2005). Their algorithm was based on 2D (edge) partitioning of the graph that helps to reduce the communication overhead. They tested their algorithm on IBM BlueGene/L at the Lawrence Livermore National Laboratory.

Rong Zhou, Eric A. Hansen (2004) and 2006 adopting a reduced memory approach that stores only the search frontier and relies on a divide-and-conquer method of solution recovery. They showed that the breadth first strategy can be more efficient than a best-first strategy.

Our proposed algorithm guarantees the optimality in computing the breadth first search on large graph. The main idea is to divide the massive graph into smaller sub-graphs by using hMetis, that guarantee the number of nodes in each partitioning is equal and minimizes the number of crossing edges. We apply the BFS on one partition at a time, the algorithm traversing all nodes in the fragment. Then we use the crossing edge to select another fragment. We finished after traversing all nodes in the graph.

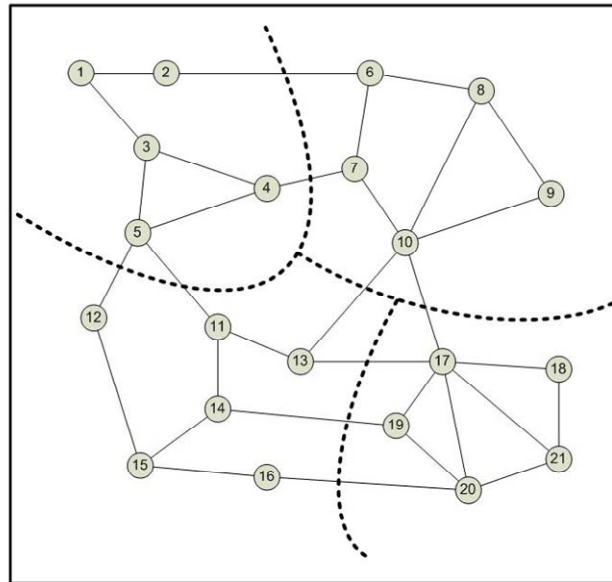
3. Proposed Breadth First Search

This section, addresses how to compute the breadth first search for large graph (h_BFS). Our approach consists of pre-processing the original (large) graph by partitioning it into sub-graphs using hMetis. In order to compute the BFS, it requires transferring the fragment to the RAM, and applies BFS on it, continuing until it completes all fragments.

3.1. Graph Partitioning:

We used the hMetis software package that was developed for partitioning the VLSI circuits by G. Karypis and V. Kumar (1997), (1998) and (1999) to partition the original graph. Graph partitioning is NP-complete and consequently the VLSI design automation community has explored several approaches for practical graph partitioning. In a traditional graph, each edge connects two vertices. hMetis permits to specify a parameter K , which represents the desired number of fragments. It partitions the graph into K fragments such that each fragment contains approximately the same number of vertices, while minimizing the number of edges that cross the fragments. Figure 1 illustrates the graph that has been partitioned. There are other methods to partition the graph that depends on spatial proximity. The G. Karypis and V. Kumar (1996) sort all edges by the x-coordinates of their nodes then apply a plane-sweep technique to sweep all x-sorted edges from left to right. The sweeping process stops periodically to sort edges swept since the last storage by y-coordinated of their origin nodes. This technique is applicable only if coordinates are available for the vertices of the graph, and usually yields partitions that are worse than those obtained by other partitioning techniques.

Figure 1: Graph Partitioning: the graph consisting of 21 vertices. The dashed lines divide them into four fragments consisting of 5 five vertices each

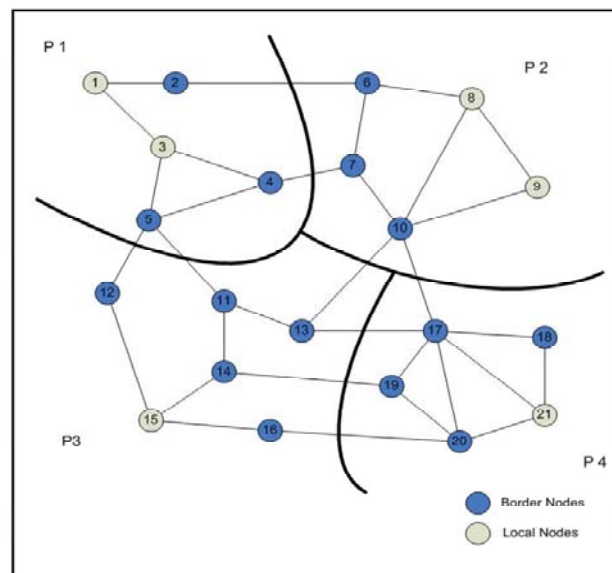


3.2. Heuristic approach to compute BFS (h_BFS)

Our algorithm contains two steps: the first step is the pre-processing step. It uses hMetis. The second one is when we apply our heuristic approach to compute the BFS algorithm for all fragments.

The hMetis algorithm partitions the vertices of a graph into fragments. The edges of the graph are of two types: local and border. A local edge is one whose vertices belong to the same fragment, whereas a border edge is one both vertices of which belong to different fragments. Two fragments are said to be adjacent if they have edges with endpoints belonging to different fragments. The endpoints of the border edges are border nodes other nodes are local nodes. Figure 2 illustrates these concepts. The pre-processing step considers specifying the type of each node in the fragment.

Figure 2: Local, and Border nodes

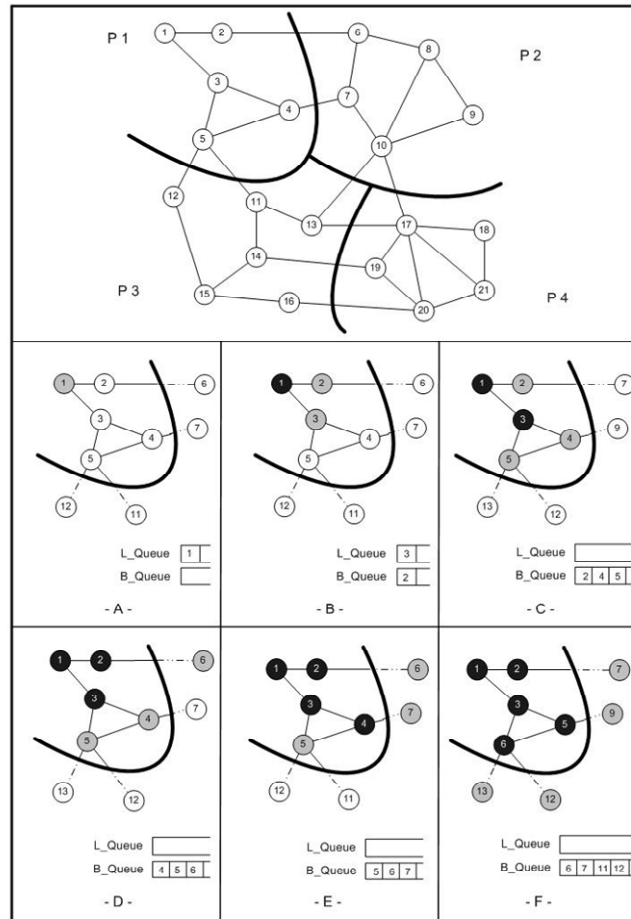


The heuristic h_BFS algorithm is presented in Figure 3. It starts by specifying the fragment number that contains the source node (step 2) and transferring it to the memory (step 4.3). We define two queues: L_Queue and the priority queue B_Queue. Insert nodes in L_Queue if it is a local node otherwise we will add it to the priority queue B_Queue this includes the source node (step 3 and step 4.4.1.4). The function Local() takes two parameters: the node number and the partition number and it returns true if the node is local to that partition number otherwise it returns false. When L_Queue is empty then we remove a node from priority B_Queue that its fragment in the memory, otherwise we can select any node from it and transfer the fragment to the memory. The main idea of the h_BFS is to minimize the number of nodes in the memory and the number of I/O's that enhance the performance of the traditional BFS algorithm. We transfer the fragment from the hard disk to the memory to process its local and border nodes and move to the other fragment based on any one of its border edges. We continue until we traverse all nodes in all fragments. The memory contains a part of the graph instead of the whole graph.

Figure 3: The heuristic Breadth First Search (h_BFS)

H_BFS(G,s)	
1.	Initialize_G(G,V,s)
2	fragment_no=fragment_number(s)
3	if Local(s,fragment_no)=True
3.1	Then Enqueue(L_Queue,s)
3.2	Else Enqueue(B_Queue,s)
4	while isnotempty(L_Queue) or isnotempty(B_Queue)
4.1	do if isnotempty(L_Queue)
4.1.1	Then selected_node=Dequeue(L_Queue)
4.1.2	Else selected_node=Dequeue(B_Queue,fragment_no)
4.2	fragment_no=fragment_number[selected_node]
4.3	if fragment_no in not in memory
4.3.1	then transfer fragment_no in memory
4.4	for each x -> adj[selected_node]
4.4.1	do if color[x]=white
4.4.1.1	then color[x]=gray
4.4.1.2	D[x]=d[selected_node]+1
4.4.1.3	predecessor[x]=selected_node
4.4.1.4	if Local(x,fragment_no)=true
4.4.1.4.1	then Enqueue(L_Queue,x)
4.4.1.4.2	else Enqueue(B_Queue,x)
5	color[selected_node]=black

In figure 4, the source node which is node 1 is inserted to the L_Queue since it is a local node (part A). Next we traverse all the nodes that are adjacent to node 1. Node 3 is inserted to L_Queue and node 2 to B_Queue (part B). When L_Queue is empty then we start processing all nodes in B_Queue, which belongs to that fragment (part C). At the time, we finish traversing all nodes belonging to the fragment then select a new node from the B_Queue to move its fragment to the memory and repeat the process (part F).

Figure 4: Traverse the h_BFS

4. Implementation and Experimental Results

Our algorithm was developed in Visual C++® 6.0 Enterprise Edition in D. S. Malik (2006), and Harvey Deitel (2005) were implemented on an Intel® Pentium® M with speed 1.73 GHz processor and 504 MB RAM, running on the Microsoft Windows XP Professional Service Pack 2. In all experiments, the results contain the following:

1. running time
2. number of I/O operations
3. number of partitioning.

We used two types of data: grid data (this models the road network structure in urban areas) and a random graph with different number of nodes. We ran our experiments on 100X100, 200X200, and 300X300 undirected grids. Table 1 shows these results. Table 2 shows results of our algorithm on undirected random graph with 10000 and 30000 nodes. We make the following observations about the results of our experiments:

1. Tables show that our h_BFS are significantly superior to the BFS under either measure the run time and number of I/O operations. The BFS takes longer time due to the large number of I/O operations.
2. The number of I/O operations increase as we increase the number of partitions but still less than the number of I/O of BFS

Table 1: Results on Grid with different partitioning

Graph size	BFS		h_BFS							
	Time	I/O	4 Partitions		8 Partitions		16 Partitions		32 Partitions	
			Time	I/O	Time	I/O	Time	I/O	Time	I/O
100*100	3.14 hours	445	49.71 Sec	8	20.34 Sec	16	84.29 Sec	32	115.5 Sec	64
200*200	5 days	844	15.1 Min	8	17.9 Min	16	18.0 Min	32	17.67 Min	64
300*300	15 days	1266	63.0 Min	8	59.0 Min	16	67.35 Min	32	88.5 Min	64

Table 2: Results on random graph with different partitioning

Graph size	BFS		h_BFS							
	Time	I/O	4 Partitions		8 Partitions		16 Partitions		32 Partitions	
			Time	I/O	Time	I/O	Time	I/O	Time	I/O
10000	3.14 hours	126	26.78 Min	8	32.91 Min	16	33.63 Min	32	32.6 Min	64
30000	4 days	378	4.6 Hour	8	4.71 Hour	16	4.47 Hour	32	4.96 Hour	64

5. Conclusion

We present a heuristic algorithm for traversing the breadth first search (h_BFS) for large graph by using the hMetis partitioning technique that divide the large graph into fragments where the number of nodes in each fragments are equal and reduces the number of the crossing edges. The implementation shows that we have one fragment in the memory at a time after exploring its nodes then we use one of its border edges to select another fragment. The experimental results show that we enhance the running time and reduce the number of I/O operations. Our future work includes the deployment of the h_BFS on sparse graph and introduces a new h_BFS which can work in parallel.

References

- [1] A. Yoo, E. Chow, K. Henderson, W. Mcledon, B. Hendrickson, and Ü. Çatalyürek, 2005, 2005. A scalable distributed parallel breadth-first search algorithm on Bluegene/L. In *Proc Supercomputing (SC 2005)*, Seattle, WA.
- [2] David A. Bader; Kamesh Madduri, 2006. Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2, Parallel Processing ICPP 2006. International Conference on Parallel Processing, Page(s):523 – 530 Deepak Ajwani, R. Dementiev, U. Meyer, and V. Osipov, 2006. Breadth First search on massive. Graphs. The Ninth DIMACS Implementation Challenge:
- [3] The Shortest Path Problem. November 13 - 14, DIMACS Center, Rutgers University, Piscataway, NJ
- [4] Deepak Ajwani, Ulrich Meyer, Vitaly Osipov, 2007. Improved external memory BFS implementations In Workshop on Algorithm engineering and experiments (ALENEX 07), pages 3–12. New Orleans, USA.
- [5] Deepak Ajwani, 2005. Design, implementation and experimental study of external memory BFS algorithms, M.Sc. Thesis, Universität des Saarlandes
- [6] D. S. Malik, 2006. C++ Programming: From problem analysis to program design, 3rd Edition, Thomson.
- [7] G. Karypis and V. Kumar, 1999. “MultiLevel k-way hypergraph partitioning,” in design Automation Conference, pp. 343-348.
- [8] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, 1997. “MultiLevel hypergraph partitioning:application in vlsi domain,”in DAC 97: Proceedings of the 34th annual conference on Design Automation, (New York, NY, USA), pp. 526-529, ACM Press.
- [9] G. Karypis and V. Kumar, 1998.hMETIS Ahypergraph Partitioning Package version 1.5.3 University of Minnesota.
- [10] G. Karypis and V. Kumar, N. Jing, and E. A., 1996. Rundensteiner, “Effective graph clustering for path queries in digital map databases,” in CIKM 1996: Proceedings of the fifth international conference on Information and knowledge management, (NewYork, NY, USA), pp. 215-222, ACM Press.
- [11] Harvey Deitel and Paul Deitel, 2005. C++ How to Program, 5th edition,Prentice Hall,2005
- [12] Ulrich Meyer, 2001. External memory BFS on undirected graphs with bounded degree. In Proc. 12th Annual ACM-SIAM Symp. On Discrete Algorithms, pages 87-88, Philadelphia, PA, USA.
- [13] Zhou, R., Eric A. Hansen, 2006. A Breadth-First Approach to Memory-Efficient Graph Search, 21st National Conference on Artificial Intelligence (AAAI-06), Boston, MA.
- [14] Zhou, R., and Hansen, E. 2004. Breath-First heuristic search. In Proce. Of the 14th International conference on Automated Planning and Scheduling (ICAPS-04),92-100
- [15] Zhou, R., and Hansen, E. 2006. Breath-First heuristic search. Artificial Intelligence 170:385-408