A MPI Implementation of Gusfield's Algorithm to Generate Cut Trees

Luiz Antonio Rodrigues

Federal University of Paraná, Departament of Informatics, Curitiba, Brazil
Western Paraná State University, Departament of Computer Science, Cascavel, Brazil

luiz.rodrigues@unioeste.br

***Abstract*—This paper presents experimental results of a parallel implementation of Gusfield's algorithm to generate cut trees. Cut trees are a compact representation of edge-connectivity between every pair of nodes of an undirected graph. Cut trees have a vast number of applications in combinatorial optimization and analysis of graphs originated in a variety of fields, including network connectivity. We describe the implementation of Gusfield's algorithm using MPI. The results achieved a significant speed-up for all real and synthetic graphs in our dataset.**

I. Introduction

Parallel processing has become an important tool to improve the solution of computing problems that require intensive processing power or deal with large volumes of data. Parallel computers have become cheaper and are available even as advanced personal computers, including laptops. Morover, it is possible to construct clusters using these processors, further increasing the opportunities for parallelism. Within each architecture, different software solutions exist, which allow parallelism to be exploited easily: with a limited set of adjustments, one can obtain parallel versions of sequential applications. Among these solutions, some libraries such as MPI (Message Passing Interface) stand out.

In this paper we present a MPI implementation of a well-known cut tree algorithm. Cut trees are compact representations of minimum cuts between all pairs of vertices of an undirected graph and have numerous applications, such as routing, graph partitioning and connectivity ([1], [2], [3], [4]). A cut tree of a weighted graph can be found using one of two well-known algorithms: Gomory and Hu[5] and Gusfield[6].

Although many existing cut tree applications involve large graphs, few works have been published on the practical applicability of these algorithms for very large graphs. In particular, heuristics that could make these algorithms more efficient have hardly ever been studied for graphs with particular properties. Futhermore, to the best of our knowledge, there are no parallel or distributed algorithms for computing cut trees available.

In this paper we present a parallel implementation based on MPI for finding cut trees. Experimental results are presented for different graphs and a variable number of processors.

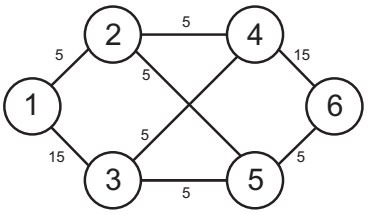
The rest of the paper is organized as follows. In Section II we give preliminary definitions, several related to the maximum flow problem which is the basis for computing cut trees. Section III describes Gusfield's algorithm, including both the sequential and parallel implementations. The Section IV describes the environment and parameters used in the executions. The results are presented and discussed in Section V. Finally, in Section VI we present conclusions and future work.

II. Preliminary Definitions

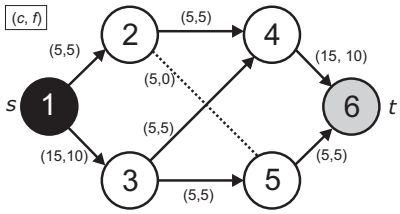
A graph $G = (V, E)$ is composed by a finite set of vertices V and a set of edges E of unordered pairs $\{u, v\}$ with $u, v \in V$. A network can be modeled as a graph, where each vertex is a node and each edge is a logical link between two nodes. In this model, an edge has a capacity $c : E \rightarrow R^+$ which represents the flow that can be send through it. As a example, consider the undirected graph in Fig. 1(a). The capacities are indicated on the edges. Expanding this concept, consider a communication between two different nodes s (source) and t (destination), not necessarily adjacent. For any node s and t, there is a flow value f between them which is maximum when the flow that leaves s and reaches t is the maximum possible. Fig. 1(b) illustrates the flow values, considering the same graph in Fig. 1(a) and using $s = 1$ and $t = 6$. Each edge was represented by two values (c, f), where c is the capacity and f the final flow passing through the edge. The maximum flow in this case is 15, which is the sum of the flow $\{1, 2\}$=5 and $\{1, 3\}$=10. For each edge, f must satisfy two conditions: (1) it cannot exceed the capacity and (2) the flow is conserved at each vertex $v \neq s, t$, i.e., the flow on edges converging to v is equal to the flow leaving from v [7].

A cut of a graph $G = (V, E)$ is a bipartition $\{X, V - X\}$ of V. The capacity of the cut is the sum of the capacities of the edges with one vertex in X and other in $V - X$. A s-t-cut is a cut $\{X, V - X\}$ such that $s \in X$ and $t \in V - X$. A minimum s-t-cut is a s-t-cut of minimum capacity. The local connectivity between s and t is defined as the capacity of a minimum s-t-cut. The local connectivity between two vertices is equals the maximum flow between them [7].

Although the maximum flow problem is usually defined for directed graphs, here we will describe it in terms of undirected graphs, which are our objects of interest. However, note that any algorithm for the problem of maximum flow can be used to find the local connectivity between two vertices of an



(a) Example of an undirected weighted graph



(b) Maximum flow, $s = 1$ and $t = 6$

Fig. 1. Undirected graph and Maximum flow.

undirected graph by replacing each edge by two directed edges with same original capacity, one in each direction.

The main algorithms for computing the maximum flow are divided into two categories: those based on augmenting paths and those based on the operations *push* and *relabel*. An example of algorithm in the first category is *Ford-Fulkerson* [8]. The second category is formed by Algorithm *Push-relabel* and its variations [9], [10].

III. On Cut Trees

A cut tree $T = (V, F)$ is formally defined as a weighted undirected tree defined on the vertices of the graph according the following properties [11]: (a) for any pair of vertices $s, t \in V, \lambda(s, t)$ is the minimum weight on an edge $e(s, t)$ in the unique path connecting s to t in T; (b) the bipartition of vertices in T produced by removing $e(s, t)$ corresponds to the minimum $s - t$ cut in the original graph G.

Cut trees were introduced by Gomory and Hu [5] as a structure that represents all $s - t$ cuts of an undirected graph. They used a maximum flow algorithm and graph contractions to solve the problem. Many years later, Gusfield [6] proposed a solution that also uses maximum flow, but it works on the input graph without contractions. Goldberg and Tsioutsiouliklis [12] presented an experimental study of cut trees algorithms that propose heuristics to make Gomory-Hu and Gusfield faster. According their results, a modified version of Gomory-Hu is more robust than Gusfield's algorithm. Duarte, Santini and Cohen [2] used cut trees to compute a connectivity measure to classify network nodes. They used this approach to identify highly connected nodes in the network in order to find reliable routing detours.

The two following subsections describe the implementations of Gusfield's algorithm in its sequential and parallel versions.

A. Gusfield's Algorithm: Sequential Version

The sequential version of Gusfield's algorithm consists of $n-1$ iterations of a Maximum Flow algorithm. A pseudocode

Algorithm 1 Sequential Gusfield's Algorithm

Input: $G = (V, E, c)$
Output: $T = (V, E, f)$, where T is a cut tree of G
1: $V(T) \leftarrow V(G)$; $E(T) \leftarrow \emptyset$
2: **for** $tree_i, flow_i, 1 \leq i \leq N$ **do**
3: $tree_i \leftarrow 1$; $flow_i \leftarrow 0$
4: **end for**
// $n - 1$ maximum flow iterations
5: **for** $s \leftarrow 2$ to N **do**
6: $flow_s \leftarrow$ MaxFlow(s, $tree_s$)
7: adjust the $tree$ with Cut(s,$tree_s$)
8: **end for**
// Generate T
9: **for** $s \leftarrow 1$ to N **do**
10: $E(T) \leftarrow E(T) \cup \{s, tree_s\}$
11: $f(\{s, tree_s\}) \leftarrow flow_s$
12: **end for**
13: **return** T

is illustrated on Algorithm 1. For each iteration (lines 6-9), a different vertex is chosen as source. This choice determines the destination vertex. Initially, all vertices of the tree point to node 1. After the first iteration, the nodes on the source side of the cut point to the source (node 1) and the nodes on the destination side point to the destination (node 2). In the second iteration, node 3 is chosen as source and node 1 or 2 is the destination, depending on which side node 3 was on the first iteration. This process continues for each node until $n - 1$ iterations are completed. The implementation of the algorithm is simple and requires no changes in the maximum flow algorithm.

Our choice for the maximum flow algorithm to use was the *Push-Relabel* algorithm [9]. Basically it starts the process by pushing the maximum flow allowed by the edges connected to source s. This flow is propagated through the other edges to find the target t. If the flow received by a vertex is greater than the sum of the capacities of edges attached to that vertex, the excess flow is returned before completion. The minimum s-t-cut is defined by the nodes reachable from s without using saturated edges, that is, those edges where the flow is equal to maximum capacity.

B. Gusfield's Algorithm: Parallel (MPI) Version

MPI has emerged in the early 1990's as a set of libraries for process management and message exchange in distributed memory architectures. Its main advantage is scalability, a consequence of the fact that it uses independent computers that can be easily connected across the network. MPI usually requires a set of precise modifications on the sequential version of the algorithm to obtain the parallel solution. Usually, the parallel computing model employed is the master/slave.

The Algorithm 2 illustrates the implementation of Gusfield's algorithm in MPI. The master process is $proc_0$ and the slaves are $proc_1, .., proc_{p-1}$. A copy of the graph is maintained by each process. The master creates the tasks and sends them to slaves. Each task contains the source and destination nodes to be used in the maximum flow computation. When a slave finishes its task, it sends the result back to the master. The

Algorithm 2 MPI Version of Gusfield's Algorithm

Input: $G = (V, E, c)$, $proc_j$ processors $(0 \leq j < P)$
Output: $T = (V, E, f)$, where T is a cut tree of G

1: $V(T) \leftarrow V(G)$; $E(T) \leftarrow \emptyset$
2: **for all** $tree_i, flow_i, 1 \leq i \leq N$ **do**
3: $tree_i \leftarrow 1$; $flow_i \leftarrow 0$
4: **end for**
// $n - 1$ maximum flow iterations
5: **if** $proc_j = 0$ **then**
6: **for** $s \leftarrow 1$ To $P - 1$ **do**
7: send Task(s, $tree_s$) to $proc_s$
8: **end for**
9: **while** $s < N$ **do**
10: receive a result from $proc_j$
11: **if** result is valid **then**
12: adjust the $tree$ with Cut(s,$tree_s$)
13: $s \leftarrow s + 1$
14: send a new Task(s, $tree_s$) to $proc_j$
15: **else**
16: send a Task($result_s$,$tree_{result_s}$) to $proc_j$
17: **end if**
18: **end while**
19: **else**
20: **while** more tasks **do**
21: executes $flow_s \leftarrow$ MaxFlow(s, $tree_s$)
22: send result to $proc_0$
23: **end while**
24: **end if**
// Generate T as sequential algorithm
25: **return** T

TABLE I
GRAPHS USED IN THE TESTS.

Graphs	#Vertices	#Edges
CA-CondMat	21,363	182,572
GeoComp	3,621	9,461
PowerGrid	4,941	6,594
P2P-Gnutella	10,876	39,994
BA	10,000	49,995
ER	10,000	49,841
DBLCYC	1,024	2,048
NOI	1,500	562,125
PATH	2,000	21,990
TREE	1,500	563,625

result contains the cut and maximum flow used by the master to update the tree.

IV. Experimental Setup

Our experiments with MPI used a cluster with 14 Intel Core 2 Quad (2.4 GHz processors with 2 Gbyte of RAM memory and 4096 Kbyte cache) interconnected by a Gigabit Ethernet network. The code was written in C and compiled with gcc and optimization option -O3. For the tests, we chose 10 graphs from different families, shown in Table I. The first four were obtained using real data [13], [14], [15]. BA is a power law graph generated by the Barabási-Albert model. ER was generated by the classical Erdös-Rényi model (Gnp). The last four are synthetic graphs from classes defined in [12].

In the execution of the MPI program we tested several different configurations that could allow better performance.

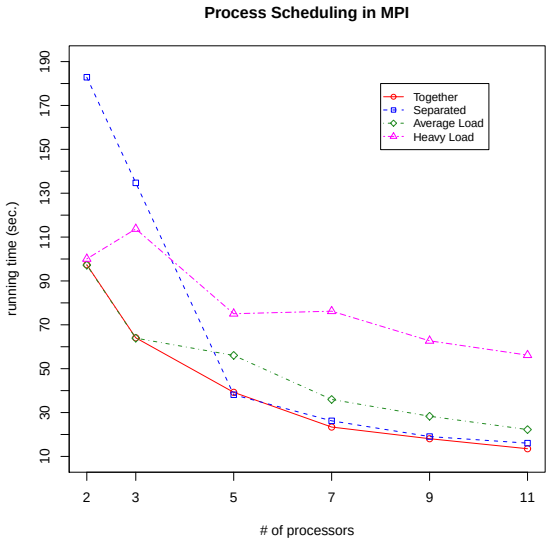


Fig. 2. Initial tests with MPI

Figure 2 shows the results of tests for executing master and slaves in different ways. In *Together*, the master and one slave run in the same machine and the others slaves run separated. In *Separated*, master and slaves run in different machines. *Average Load* uses 2 processes on each machine and *Heavy Load* uses 4 processes on each machine. Results showed that *Together* generated the best results. So, we employed this configuration for all remaining experiments.

V. Experimental Results

The results are shown in Table II. The first line contains the sequential time (in seconds) from the execution of Gusfield's algorithm in a single processor. Results from 2 to 15 processors were obtained with the parallel execution. Excluding the sequential execution, we show for each test the mean time (in seconds), the speed-up (S) and the efficiency (E)[1].

Results of all experiments show a significant speed-up and efficiency. The best results were achieved with the BA graph, for which there is no waste of computation. The worst results happened during the tests with the Powergrid graph. While the computation of cuts of BA are 100% recovered, recomputations on Powergrid (with 1,500 nodes) achieved 4,000 with 15 processors. This seriously reduces the efficiency of the solution and is the main point on which we are working to improve results.

VI. Conclusion

In this paper we presented experimental results obtained with the parallelization of Gusfield's algorithm using MPI. The results show a significant gain in terms of speed-up and efficiency. We believe that it is possible to devise heuristics

[1]The speed-up was computed as $S = T_S/T_P$, where T_S is the execution time of the sequential version and T_P is the execution time with MPI on P processes. The efficiency was computed using $E = S/P$.

TABLE II
MPI RESULTS. TIMES ARE IN SECONDS.

Number of processors	CA-HepPh time	S	E	Geocomp time	S	E	Powergrid time	S	E	P2P-Gnutella time	S	E	BA time	S	E
sequential	864.93	-	-	3.85	-	-	7.26	-	-	142.00	-	-	204.72	-	-
2	886.82	0.98	0.49	7.39	0.52	0.26	10.48	0.69	0.35	137.82	1.03	0.52	177.82	1.15	0.58
3	513.63	1.68	0.56	2.54	1.52	0.51	4.50	1.61	0.54	73.90	1.92	0.64	88.98	2.30	0.77
4	433.16	2.00	0.50	2.06	1.87	0.47	3.48	2.09	0.52	64.13	2.21	0.55	65.54	3.12	0.78
5	353.81	2.44	0.49	1.74	2.21	0.44	3.00	2.42	0.48	51.79	2.74	0.55	54.02	3.79	0.76
6	214.53	4.03	0.67	1.45	2.66	0.44	2.47	2.94	0.49	31.81	4.46	0.74	42.08	4.87	0.81
7	161.27	5.36	0.77	1.19	3.24	0.46	2.22	3.27	0.47	23.41	6.07	0.87	29.02	7.05	1.01
8	129.25	6.69	0.84	1.10	3.50	0.44	2.01	3.61	0.45	18.76	7.57	0.95	22.50	9.10	1.14
9	107.52	8.04	0.89	0.92	4.18	0.46	1.95	3.72	0.41	15.64	9.08	1.01	18.18	11.26	1.25
10	92.70	9.33	0.93	0.85	4.53	0.45	1.84	3.95	0.39	13.10	10.84	1.08	15.35	13.34	1.33
11	81.33	10.63	0.97	0.79	4.87	0.44	1.99	3.65	0.33	11.66	12.18	1.11	13.54	15.12	1.40
12	73.26	11.81	0.98	0.76	5.07	0.42	1.69	4.30	0.36	10.39	13.67	1.14	11.79	17.36	1.45
13	65.92	13.12	1.01	0.71	5.42	0.42	1.90	3.82	0.29	9.25	15.35	1.18	11.17	18.33	1.41
14	62.80	13.77	0.98	0.63	6.11	0.44	1.76	4.13	0.29	8.65	16.42	1.17	10.54	19.42	1.39
15	60.46	14.31	0.95	0.70	5.50	0.37	1.89	3.84	0.26	8.36	16.99	1.13	10.02	20.43	1.36
Number of processors	**DBLCYC** time	S	E	**ER** time	S	E	**NOI** time	S	E	**PATH** time	S	E	**TREE** time	S	E
sequential	10.49	-	-	191.26	-	-	657.50	-	-	5.90	-	-	410.07	-	-
2	13.10	0.80	0.40	203.52	0.94	0.47	703.64	0.93	0.47	9.80	0.60	0.30	434.23	0.94	0.47
3	6.49	1.62	0.54	103.21	1.85	0.62	409.89	1.60	0.53	3.91	1.51	0.50	292.41	1.40	0.47
4	4.46	2.35	0.59	88.37	2.16	0.54	301.97	2.18	0.54	3.53	1.67	0.42	200.02	2.05	0.51
5	3.40	3.09	0.62	68.97	2.77	0.55	209.52	3.14	0.63	2.97	1.99	0.40	135.61	3.02	0.60
6	2.75	3.81	0.64	42.23	4.53	0.75	138.65	4.74	0.79	2.04	2.89	0.48	96.51	4.25	0.71
7	2.37	4.43	0.63	32.10	5.96	0.85	104.65	6.28	0.90	1.68	3.51	0.50	72.16	5.68	0.81
8	2.04	5.14	0.64	25.30	7.56	0.94	81.83	8.03	1.00	1.40	4.21	0.53	59.76	6.86	0.86
9	1.82	5.76	0.64	20.66	9.26	1.03	68.84	9.55	1.06	1.16	5.09	0.57	49.01	8.37	0.93
10	1.66	6.32	0.63	17.83	10.70	1.07	59.51	11.05	1.10	1.07	5.51	0.55	41.98	9.77	0.98
11	1.51	6.95	0.63	15.44	12.40	1.13	53.60	12.27	1.12	0.92	6.41	0.58	37.27	11.00	1.00
12	1.37	7.66	0.64	14.01	13.70	1.14	50.38	13.05	1.09	0.85	6.94	0.58	35.12	11.68	0.97
13	1.28	8.20	0.63	12.41	15.40	1.19	47.21	13.93	1.07	0.76	7.76	0.60	32.08	12.78	0.98
14	1.23	8.53	0.61	11.70	16.30	1.17	45.12	14.57	1.04	0.71	8.31	0.59	30.57	13.41	0.96
15	1.16	9.04	0.60	11.34	16.90	1.12	42.40	15.51	1.03	0.69	8.55	0.57	28.09	14.60	0.97

to improve this gain. One of the possible optimizations is to determine how to properly choose the source and destination vertices so that less computations are lost, thus thus increasing the efficiency for the the generation of the tree.

Furthermore, as multicore machines become popular, we are also planning as future work to compare a parallel solution to this problem using OpenMP (Open Multi-Processing), that is an interface designed for parallel programming on shared memory architectures.

ACKNOWLEDGMENT

The research was supervised by Elias P. Duarte Jr. and Jaime Coher, from Federal State University of Paraná. This work was supported by Fundação Araucária/SETI, (proj. 19836/2010) and by FINEP (proj. CT-INFRA/UFPR).

REFERENCES

[1] L. Wu and P. K. Varshney, "On survivability measures for military networks," *Military Communications Conference*, pp. 1120–1124, 1990. [Online]. Available: http://dx.doi.org/10.1109/MILCOM.1990.117586

[2] E. P. Duarte Jr., R. Santini, and J. Cohen, "Delivering packets during the routing convergence latency interval through highly connected detours," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 495–.

[3] R. Engelberg, J. Könemann, S. Leonardi, and J. S. Naor, "Cut problems in graphs with a budget constraint," *J. of Discrete Algorithms*, vol. 5, pp. 262–279, Jun. 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1240580

[4] K. Y. Kamath and J. Caverlee, "Transient crowd discovery on the real-time social web," in *Proceedings of the 4th ACM Web Search and Data Mining Conference (WSDM)*, 2011. [Online]. Available: http://faculty.cs.tamu.edu/caverlee/pubs/kamath11wsdm.pdf

[5] R. E. Gomory and T. C. Hu, "Multi-terminal network flows," *Journal of the Society for Industrial and Applied Mathematics*, vol. 9, no. 4, pp. 551–570, 1961.

[6] D. Gusfield, "Very simple methods for all pairs network flow analysis," *SIAM J. Comput.*, vol. 19, pp. 143–155, February 1990.

[7] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 2nd ed. Germany: Springer, 2002.

[8] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1955.

[9] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, pp. 921–940, October 1988.

[10] B. V. Cherkassky and A. V. Goldberg, "On implementing the push-relabel method for the maximum flow problem," *Algorithmica*, vol. 19, no. 4, pp. 390–410, 1997.

[11] D. Panigrahi, *Encyclopedia of Algorithms*. Springers, 2008, ch. Gomory-Hu Trees.

[12] A. V. Goldberg and K. Tsioutsiouliklis, "Cut tree algorithms," in *SODA '99: Proceedings of the tenth annual ACM-SIAM Symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999, pp. 376–385.

[13] J. K. J. Leskovec and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data (ACM TKDD)*, 2007.

[14] V. Batagelj and A. Mrvar, "Pajek datasets," 2006.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, 1998.