Computer and Information Science; Vol. 12, No. 3; 2019
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Modeling the Parallelization of the Edmonds-Karp Algorithm and Application

Amadou Chaibou[1,4], Ousmane Moussa Tessa[2] & Oumarou Sié[3,4]

[1] Département d'Informatique, École des Mines, de l'Industrie et de la Géologie (EMIG), Niamey, Niger

[2] Département de Mathématiques et d'Informatique, Université Abdou Moumouni, Niamey, Niger

[3] Département d'Informatique, Université Ouaga I Pr Joseph KI-ZERBO, Ouagadougou, Burkina Faso

[4] Laboratoire de Mathématiques et Informatique, Université Ouaga I Pr Joseph KI-ZERBO, Ouagadougou, Burkina Faso

Correspondence: Amadou Chaibou, Département d'Informatique, École des Mines, de l'Industrie et de la Géologie (EMIG), Niamey, Niger.

Received: May 20, 2019 Accepted: June 17, 2019 Online Published: July 25, 2019

doi:10.5539/cis.v12n3p81 URL: https://doi.org/10.5539/cis.v12n3p81

Abstract

Many optimization problems can be reduced to the maximum flow problem in a network. However, the maximum flow problem is equivalent to the problem of the minimum cut, as shown by Fulkerson and Ford (Fulkerson & Ford, 1956). There are several algorithms of the graph's cut, such as the Ford-Fulkerson algorithm (Ford & Fulkerson, 1962), the Edmonds-Karp algorithm (Edmonds & Karp, 1972) or the Goldberg-Tarjan algorithm (Goldberg & Tarjan, 1988). In this paper, we study the parallel computation of the Edmonds-Karp algorithm which is an optimized version of the Ford-Fulkerson algorithm. Indeed, this algorithm, when executed on large graphs, can be extremely slow, with a complexity of the order of $O|V|.|E|^2$ (where |V| designates the number of vertices and |E| designates the number of the edges of the graph). So why we are studying its implementation on inexpensive parallel platforms such as OpenMp and GP-GPU. Our work also highlights the limits for parallel computing on this algorithm.

Keywords: Edmonds-Karp algorithm, Maximum flow problem, Parallel computation, OpenMP, GP-GPU, Modeling

1. Introduction

In this paper, we study the parallelization of the Edmonds-Karp algorithm. Originally, this algorithm was proposed by Ford-Fulkerson (Ford & Fulkerson, 1962), but implemented by Edmonds-Karp (Edmonds & Karp, 1972). A formal approach of this algorithm was also proposed by Lammich P. and Sefidgar S. R. (2016). This algorithm consists of finding paths from the source vertex S to the terminal vertex T, with a capacity available on all their arcs. The process stops when all possible paths with nonzero capability are exhausted. This algorithm, as simple as it looks, is very slow when run on large graphs. Thus, different versions have been proposed for parallel architectures, to obtain better performances. Edmonds and Karp had already proposed an algorithm parallelization and optimization approach for shared memory systems. Other parallel versions of this algorithm have also been developed (Karger & Stein, 1996; Vineet & Narayanan, 2008). In this work, we analyze the complexity of the parallelization of this algorithm, then we propose a parallel solution approach that we codify and test.

2. Preliminary

The Edmonds-Karp algorithm is used to solve problems of optimization, planning, operational research, artificial vision or maximizing the flow of packets in a computer network, etc.

2.1 Maximum Flow and Minimum Cut

2.1.1 Maximum Flow

The problem of the maximum flow is to find, in a flow network, a maximum flow that can be realized from a single source S to a single destination T. In practice, this problem consist to determining the value of this

maximum flow.

Example

As an illustration, the network of figure 1 consisting of 7 nodes (Wikipedia, 2014) can represent an electrical circuit.

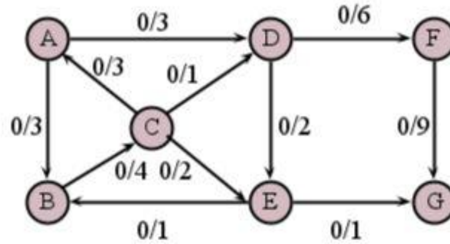

Figure 1. Example of a maximum flow problem: A designates the source and G the destination

Description: On the edges are represented the pairs f/c where f is the flow of current and c is the capacity. The residual capacity of u to v is CF(u, v)=c(u, v)-f(u, v). It is equal to the total capacity less the flow already used. If the net flow from u to v is negative, it contributes to the residual capacity. When Edmonds-Karp algorithm is applied on this network, the maximum flow rate is 5.

2.1.2 Minimum Cut

A cut is a set of vertices, and the cardinal of the cut is the number of edges having one end inside this set and the other outside (see figure 2). A cut is minimum if its cardinal is minimum.

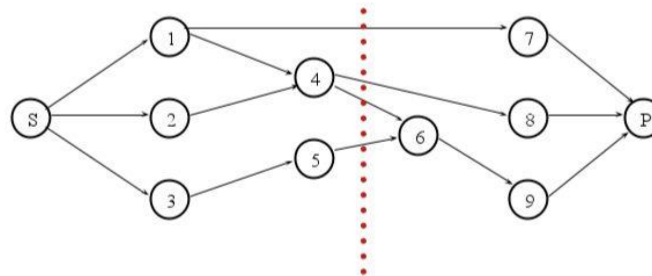

Figure 2. Example of a graph cut with 4 as value

The problem of the minimum cut (Min-Cut) is equivalent to the maximum flow problem, according to the flow-max/min-cut theorem (Fulkerson & Ford, 1956) that we recall below.

Theorem 1 *In optimization, the flow-max / min-cut theorem states that given a flow graph, the maximum flow that can go from the source to the destination is equal to the minimum capacity that must be removed from the graph in order to prevent that any flow cannot go from the source to the destination.*

2.1.3 Case of a Complete Undirected and Unweighted Graph

Complete graph: A complete graph G is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.

Let N be the number of nodes of a undirected complete graph representing a network. The weight (or flow) in each edge is assumed equal to 1. If CN denotes the minimum cut of this graph, we obtain the explicit relations in figure 3.

Number of vertices	Dummy graphs	Minimum cut: CN
2		1
3		2
4		3
5		4
6		5

Figure 3. Examples of complete graphs and their minimum cuts

Proposition 1 *The maximum flow (or minimum cut) of a complete undirected and unweighted graph of N vertices is N-1.*

Proof. Consider G_N an complete undirected and unweighted graph of N vertices. Each vertex is linked to the other N-1 vertices by an edge (according to the definition of the complete graph). So, the total number of edges of G_N is: $C_N^2 = \frac{N!}{2!(N-2)!}$ (WhereC_N^2 designates the combination of N taken 2 to 2)

Let be any partition of G_N, in two sets of x elements and N-x elements. The number of edges common to x elements is C_x^2 and the number of edges common to (N-x) other elements is C_{N-x}^2. Hence the number of edges that have a source and a destination in the two sets that form the partition of G_N is given in equation (1).

$$C_N^2 - (C_x^2 + C_{N-x}^2) = \frac{N!}{2!\,(N-2)!} - (\frac{x!}{2!\,(x-2)!} + \frac{(N-x)!}{2!\,(N-x-2)!}) = Nx - x^2 \qquad (1)$$

Thus the values of the cut are given by the function $C(x) = Nx - x^2$, x ϵ [1, N-1].

This function is strictly increasing for $x \epsilon\ [1, E\left(\frac{N+1}{2}\right)]$ (Note 1) then decreasing in the $[E\left(\frac{N+1}{2}\right),\ N-1]$ range as shown in variation table 1.

Table 1.Variation of C

X	1	$E\left(\frac{N+1}{2}\right)$	N-1
C'(x)=N-2x	+	0	-
$C(x)=Nx-x^2$	N-1 ↗	$E\left(\frac{N^2}{4}\right) \mid E\left(\frac{N^2-1}{4}\right)$	↘ N-1

$$\text{The maximum values of C are:} \begin{cases} E\left(\frac{N^2}{4}\right) \text{ if N is even} \\ E\left(\frac{N^2-1}{4}\right) \text{ if N is odd} \end{cases} \quad (2)$$

The minimum values are reached for x=1 and x=N-1 which correspond to the limits of its domain of definition. C(1)=C(N-1)=N-1. Hence the result.

2.2 Algorithms Used

In this section, we assume:

- f: Value of the flow in the network
- x, z: Edges
- C(x): Capacity of edge x
- e: Maximum flow that can be send on a path
- G_f : Graph for searching maximum flow
- G: Graph
- Q: Queue
- Marked(x): x is marked as visited

2.2.1 Edmonds-Karp Algorithm

The Edmonds-Karp algorithm (Edmonds & Karp, 1972) is an implementation of the Ford-Fulkerson method for calculating the maximum flow in a flow network, with a time of order $O|V|.|E|^2$.

The difference between the two algorithms is that in Edmonds-Karp algorithm (see algorithm 1 for description) the search is done in an orderly way: a search criterion of the path that improves the solution is defined. The solution path must be as short as possible (number of edges) and with a non-zero capacity of the flow. This path can be found with the Breadth-first search (BFS) algorithm, assuming all edges are of unit capacity.

Algorithm 1 : Edmonds-Karp(G)(Edmonds J. & Karp R. M., 1972)

```
BEGIN
    f <-- 0; Gf<-- G;
    While Gf contains a path from the source S to the terminal T do
        Let C be that path with a minimum of edges
        Increase f using C
        Update Gf
    EndWhile
    return f
END
```

Principle of the algorithm

The Edmonds-Karp algorithm is based on 3 steps:

1. Search for an unexplored path from source S to terminal T. This step is performed using the Breadth-First Search (BFS) algorithm (algorithm 2) which ensures that the next path from S to T is minimal in number of edges.
2. Find the maximum flow that can be send on this path. This flow noted e is given by the equation (3).

$$e = min_{x \epsilon E}\big(C(x) - f(x)\big) \quad (3)$$

3. Send the flow e on this path.

These operations are repeated until there is no augmenting (Note 2) path. The maximum flow is equal to the sum of flows coming from distinct paths from S to T. This can be described algorithmically as follows:

- Initialize the flow f to 0 value on each edge.
- while there is a path from S to T unsaturated, increase the flow with the residual capacity e calculated until having a complete flow (all paths are saturated);
- the maximum flow value of the Network is $|f| = \sum e$.

The Edmonds-Karp algorithm determines the maximum flow by starting from a zero flow ($\forall x \in E; f(x) = 0$) and then increasing its flow along acceptable paths S = s_1; s_2; s_3;…; s_{n-1}; s_n = T such: ($\forall i\ (s_i; s_{i+1}) \in V$ and the flow $max_i\ C(s_i; s_{i+1})$ is strictly positive.

2.2.2 Breadth First Search (BFS) Algorithm

This algorithm allows to browse a graph as follows: it starts by exploring a source node, then its successors, then the unexplored successors of successors, etc. Nodes already visited are marked to prevent the same node from being explored more than once. It can be summarized in the following steps:

1. Put the source node in the queue (Q).
2. Remove the source node from the queue to process it.
3. Put all unexplored neighbors in the queue.
4. If the queue is not empty, return to step 2.

Algorithm 2 : BFS (G,s)

```
BEGIN
    Q<-- CreateQueue(); Gf<-- G; Marked(s); Q<-- AddQueue(Q; s);
    While NotEmptyQueue(Q) Do
        X <-- PopQueue(Q);
        While ExistSuccessor(x) Do
            z <-- NextSuccessor(x);
            If Not(Marked(z)) then
                Marked(z);
                Q <-- AddQueue(Q; z);
            EndIf
        EndWhile
    EndWhile
END
```

3. Proposed Parallel Computation Approach

As we saw above, the Edmonds-Karp algorithm consists of three main steps:

1. The determination of the augmenting path. This step is performed sequentially. The paths are generated one by one, to avoid their duplication. On the other hand, the operation search for unvisited neighbors of a vertex will be performed in parallel.
2. The calculation of the residual capacity *e*. This operation is performed in parallel. Each computing unit (core) will process a group of edges assigned to it. The unit calculates the residual capacities of these arcs and then determines the relative minimum e_i. Once these relative minima are calculated, the global minimum *e* is obtained by a reduction operation in parallel.
3. Updating the values of the edges. The previous step permit to calculate the minimum overall residual capacity e. It remains now to update, the values of the edges diminished by e. This operation is a

classical, depending only to each edge. It can also be done in parallel, by distributing the edges to the different cores.

These three steps are repeated in a main loop which stop condition is “no augmenting path”.

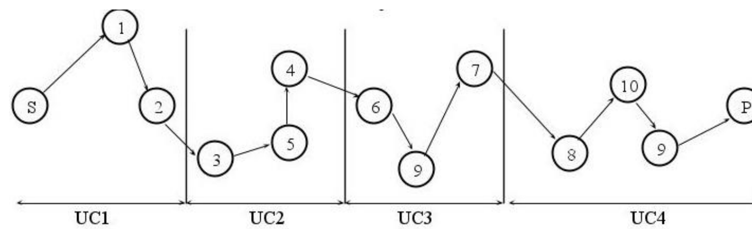

Figure 4. Assigning edges to calculation units

In Figure 4 each computing unit (CU) determines the local residual capacity of its assigned edges. After determining the global residual capacity, the CU updates their residual capacities.

3.1 General Principle of the Solution

The main loop of the Edmonds-Karp algorithm ends when there is no augmenting path unexplored from S to T. The Boolean function BFS (given in algorithm 2) performs this test. Where there is a path to explore, our approach is to create a list of vertices that make up that path and another one for their predecessors. The tasks of calculating the residual capacity of the path and the update of the values of the flows are parallelizable because they concern only this list of vertices.

3.2 Complexity of the Algorithm

We assume:

$|V|=N$: number of nodes of the graph

$|E|=M$: number of edges of the graph

C_{max}: maximum capacity of the edges

We use two methods for estimating the complexity of the Edmonds-karp algorithm.

1st method

The maximum number of paths from S to T is in the order of $O(N)$.

Assuming that at each iteration, the flow increases by one, there are at most $O(M.C_{max})$ improvements. Thus, the total number of operations for the Edmonds-Karp algorithm is $O(NM.C_{max})=O(NM^2)$.

2nd method

The search for a path from the source S to the terminal T, with a minimum of edges using the algorithm BFS (algorithm 2) is of the order of $O(M)$ (because $N \leq M$).

Once the path C is found, we have a time of the order of O (N) for the increase of the flow f and O (N) to update the residual capacities. As M= O (N), we have finally a time of the order of O (M).

Each iteration of the ”while” loop identifies an increasing path, and the increase can only be stopped when there is zero unsaturated edge. There is therefore at most M*N iterations.

In conclusion, the complexity of the Edmonds-Karp algorithm is of the order of O (M^2N) identical to the 1st method.

3.3 OpenMP Implementation

OpenMP (**OpenM**ulti-**P**rocessing) uses memory sharing to perform data transfer (Eigenmann & Voss, 2001; Milos et al., 2008; Chapman et al., 2008; Gonzalez et al., 2001).

The simulations carried out concern networks ranging in size from 100 vertices (3323 edges) to 1000 vertices (499500 edges).

Principle of the solution

The use of the OpenMP directives concerns only two steps of the algorithm, trivially parallelizable: the calculation of the residual capacities and the update of the values of the edges.

During the search phase of the residual capacity, each thread calculates a local residual capacity in relation to the nodes that it processes then in a critical area "#**pragma** omp critical" these threads contribute to the determination of the overall residual capacity.

Modeling of treatment time

In addition to the assumptions made in paragraph 3.2, we assume:

α: OpenMP initialization time;

β: Latency or delay for all threads to finish their tasks;

T_e: execution time of a sequential iteration;

T_{it}: execution time of a OpenMP iteration.

From the assumptions made in section 3.2, we deduce the sequential execution time (T_{Seq}) of the computation of the maximal flow at relation (4).

$$T_{Seq} = 3T_e * M^2 \quad (4)$$

But when these calculations are done with OpenMP directives and using a number of threads equal to nb_threads we obtain:

Duration of the path reconstruction operation: $T_e M$

Search duration of the minimum residual capacity: $\alpha + T_e(\frac{M}{nb_threads}) + \beta$

Duration of edges capacities update: $\alpha + T_e(\frac{M}{nb_threads}) + \beta$

Hence OpenMP calculation time for an iteration (T_{it}) is:

$$T_{it} = T_e M + (\alpha + T_e(\frac{M}{nb_threads}) + \beta) + (\alpha + T_e(\frac{M}{nb_threads}) + \beta) = 2\alpha + 2\beta + T_e M + 2T_e(\frac{M}{nb_threads})$$

As these computations are made in a loop which is carried out M times, the expression of the total time T is deduced in relation (5)

$$T = (2\alpha + 2\beta + (1 + \frac{2}{nb_threads})T_e M) M \quad (5)$$

Optimal value of nb_threads

By deriving the expression of T in equation (5) with respect to nb_threads, we obtain:

$$\frac{\partial T}{\partial nb_treads} = -2T_e M^2 (nb_threads)^{-2} = -\frac{2T_e M^2}{(nb_threads)^2}$$

$$\frac{\partial T}{\partial nb_treads} = 0 \Rightarrow -\frac{2T_e M^2}{(nb_threads)^2} = 0$$ $\Rightarrow$ no value of nb_threads cancels the derivative.

Theoretical acceleration

$$\frac{T_{sequential}}{T_{parallel}} = \frac{3T_e M^2}{(2\,\alpha\ + 2\beta + (1\ + \frac{2}{\text{nb_threads}})\ \text{Te M})\ \text{M}}$$

$$\text{From where} \lim_{M\to\infty} \frac{T_{sequential}}{T_{parallel}} = \frac{3}{(1+\frac{2}{nb_threads})} = \frac{3*nb_threads}{nb_threads+2} \quad (6)$$

The relation (6) shows that the acceleration of this algorithm for very high values of M is majored to 3

3.4 Implementation on GPGPU

To port the Edmonds-Karp algorithm on GP-GPU (**G**eneral-purpose **P**rocessing on **G**raphics **P**rocessing **U**nits), we have as well as when it comes to the OpenMP, identify the parallelizable zoned: the calculation for the search for the minimum residual capacity and the update of edges capabilities. This is what we will discuss below.

Principle of the solution

The adjacency matrix is represented in vector form. Then we choose enough block sizes so that parallel calculations can be run at the same time. The calculation of the minimum capacity is done by a conventional reduction operation and the updates are done in parallel by the calculation units of the GPU.

GPU execution time modeling

In addition to the previous hypotheses, we assume:

λ: kernel initialization time;

Nb: number of blocks per multiprocessor;

Nwp: number of wraps;

Ntw: number of threads in a warp;

E(x) integer part of the number x.

The theoretical calculation time for the Edmonds-Karp algorithm on GPU is given by expression (7). The number of blocks does not influence because all the blocks execute at the same time, as a single block. On the other hand, as $Nwp = \frac{N}{Ntw}$, it follows that $T_{GPU}=O\ (M^2 + M*lb(Nwp))$, where lb() (Note 3) is the binary logarithm (or logarithm in base 2).

$$T_{GPU} = ((\lambda + T_e M) + (\lambda + \text{lb(Nwp)} T_e) + (\lambda + T_e))M = (3\lambda +\ (\text{M}\ +\ \text{lb(Nwp)}\ +\ 1)T_e)\text{M} \quad (7)$$

The theoretical time of execution of the same calculations on a sequential machine is given in equation (4); it is of the order $3T_eM^2$.

Estimation of theoretical acceleration

Thus, the theoretical acceleration achieved is:

$$\gamma = \frac{3T_e M^2}{(3\lambda +\ (\text{M}\ +\ \text{lb(Nwp)}\ +\ 1)T_e)\text{M}} = \frac{3T_e \text{M}}{(3\lambda +\ (\text{M}\ +\ \text{lb(Nwp)}\ +\ 1)T_e)} \approx \frac{3}{1 + \frac{\text{lb(Nwp)}}{M}} \quad (8)$$

Note that the acceleration γ is majored by 3.

Optimal value of Nwp

$T_{GPU} = (3\lambda +\ (\text{M}\ +\ \text{lb(Nwp)}\ +\ 1)T_e)\text{M}$

By deriving the expression of T_{GPU} with respect to Nwp, we obtain:

$$\frac{\partial T_{GPU}}{\partial \text{Nwp}} = \frac{\partial(\text{lb(Nwp)}T_e\text{M})}{\partial \text{Nwp}} = \frac{T_e\text{M}}{\text{Nwp} * \ln(2)}$$

⇒no value of Nwp cancels the derivative. T_{GPU} is strictly monotonous and increasing because ln(2) > 0.

4. Results

After having coded the Edmonds-Karp algorithm in OpenMP then in Cuda C, we carry out some tests to measure the impact of our approach.

4.1 Code of the Algorithm Used

For our simulations, we took an optimized implementation of the Edmonds-Karp algorithm, written in C++. The graph representation used is the adjacency matrix. Graphs can be real-world representations such as road networks, networks for distribution of water, electricity, gas, etc. The Edmonds-Karp algorithm comes in various versions depending on the programming language, particularly because of its practical use. We have made changes to highlight the parallel parts of the code without impacting its initial effectiveness.

4.2 Characteristics of the Machine Used

CPU	Pentium ® Dual-Core CPU E5500 @ 2.80Ghz 2.80 Ghz, RAM 3 GiB, OS i686_Ubuntu 18.04 LTS	
GPU	Graphic card:	1:00.0 VGA compatible controller nvidia GeForce GTX 670, 2048Mo
	Number of CUDA cores :	1344
	Version of the gcc :	4.5.2
	Nvcc :	NVIDIA
	Constant memory :	65536
	Number of Multiprocessor Streaming Units:	7
	Multiprocessor shared memory:	49152
	Number of threads per warp:	32
	Maximum number of threads per block:	1024

4.3 Basic Data

The data used to test the different codes come from various sources such as:

- 9th DIMACS Implementation Challenge website (DIMACS, 2015);
- GTgraph, a program generating random graphs of type (n, m)(Bader & Madduri, 2006);
- GTgraph but of the type RMAT which uses a law of probability for the generation of the graphs;
- RandomGraph: RandomGraph[m,n] returns a random pseudo- graph with n vertices and m edges;
- or with Graph Generator - v3.1

4.4 Recorded Performance

Table 2. Execution times experienced with OpenMP

Numb. of vertices	Numb. of edges	Average Deg. of the graph	CPU (ms)	OpenMP (ms)			Acceleration
				4 threads	8 threads	16 threads	
100	3323	33.23	2.23	1.65	1.55	1.54	1.44
250	20727	82.91	36.63	20.74	20.77	20.74	1.76
500	109990	219.98	340.53	160.01	159.87	159.70	2.13
800	319600	399.50	1332.68	621.14	619.70	621.86	2.15
1000	499500	499.50	2598.60	1191.65	1196.15	1195.73	2.18

Table 3. Execution times experienced on GPU

Numb. of vertices	Numb. of edges	Average Deg. of the graph	CPU (ms)	GPU(ms)	Acceleration
100	3323	33.23	2.23	1.73	1.29
250	20727	82.91	36.63	19.59	1.87
500	109990	219.98	340.53	151.35	2.25
800	319600	399.50	1332.68	548.43	2.43
1000	499500	499.50	2598.60	995.63	2.61

Description: Table 2 shows datas recorded on OpenMp platforms and table 3 shows GP-GPU datas.

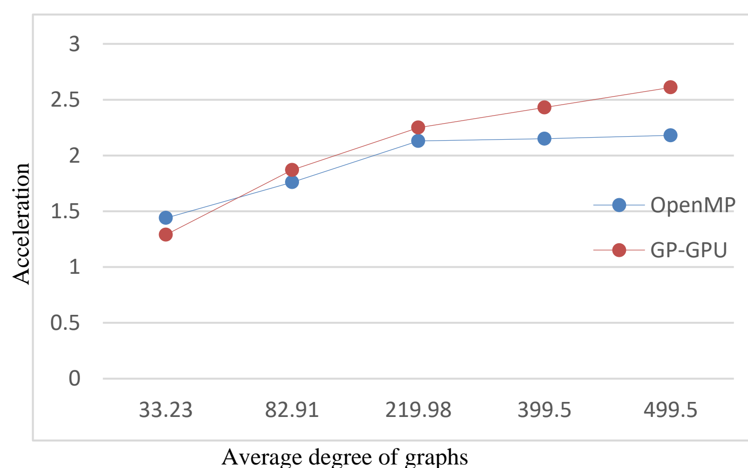

Figure 5. Performance on OpenMP and GP-GPU depending on the size of the graph

Remarks

- ✓ For OpenMP, the acceleration is quasi-stationary from 4 threads.
- ✓ For both cases (OpenMP and GP-GPU), the acceleration increases with the density of the graph.
- ✓ In our study, the GP-GPU presents a better performance than the OpenMP.

5. Conclusions

In this paper, we studied the possibilities of parallelization of the Edmonds-Karp algorithm and modeled the computation times of its sequential and parallel versions. We have estimated the theoretical acceleration then we proposed the corresponding parallel codes on inexpensive architectures: OpenMP and GPU. In both cases, the basic idea is to treat the nodes of the network in parallel:

- when looking for the maximum flow that can cross a path;
- when updating the residual capacities of the nodes.

The tests we have done on these parallel versions show that they improve the calculation time of the algorithm compared to its sequential version. The acceleration ranges from 1.4 to 2.2 for the OpenMP and from 1.3 to 2.6 for the GP-GPU. This variation is correlated with the average degree of the graph. The acceleration is also bounded by 3 because of the limitation of the parallelizable zone.

References

Bader, D. A., & Madduri, K. (2006). *GTgraph: A suite of synthetic graph generators.* Retrieved March 12, 2015 from: http://www.cc.gatech.edu/~kamesh/GTgraph/2006

Chapman, B., Jost, G., & Van der Pas, R. (2008). Using OpenMP Portable Shared Memory Parallel Programming. *In IEEE Distributed Systems Online, 9*(1), 3-3, Jan. 2008. https://doi.org/10.1109/MDSO.2008.1

Demetrescu, C., Goldberg, A. V., & Johnson, D. (2005). *9th DIMACS Implementation challenge –Shortest Paths.* Retrieved February 17, 2015 from: http://www.dis.uniroma1.it/challenge9/ (2005).

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (Association for Computing Machinery), 19*(2), 248-264. https://doi.org/10.1145/321694.321699

Eigenmann, R., & Voss, M. (2001). OpenMP Shared Memory Parallel Programming. *Lecture Notes on Computer ScienceLNCS 2104,* 155-169, Springer.

Ford, L, R., & Fulkerson, D. R. (1962). Flows in Networks. Princeton University Press. https://doi.org/10.1063/1.3051024

Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *Journal of the ACM, ACM Press, 35*(4), 921-940. https://doi.org/10.1145/48014.61051

Gonzalez, M., Ayguad, E., Martorell, X., & Labarta, J. (2001). Defining and Supporting Pipelined Executions in OpenMP. *2nd International Workshop on OpenMP.*2104, 155-169. https://doi.org/10.1007/3-540-44587-0_14

Karger, D. R., & Stein, C. (1996). A new approach to the minimum cut problem. *Journal of ACM, 43*(4), 601-40. https://doi.org/10.1145/234533.234534

Lammich, P., & Sefidgar, S. R. (2016). Formalizing the Edmonds-Karp Algorithm. In: Blanchette J., Merz S. (eds)Interactive Theorem Proving. ITP 2016. *Lecture Notes in Computer Science, 9807*. Springer, Cham. https://doi.org/10.1007/978-3-319-43144-4_14

Milos, M. R. F., Vladimir, G., Osman, S. U., Adrin, C., Ayguad, E., & Valero, M. (2008). Nebelung: Execution Environment for Transactional OpenMP. *International Journal of Parallel Programming, 36*(3), 326-346. ISSN: 0885-7458, Springer. https://doi.org/10.1007/s10766-008-0073-6

Vineet, V., & Narayanan, P. J. (2008). CUDA cuts: Fast graph cuts on the GPU. *In CVPR Workshop on Computer Visionon the GPU,* 1-8, 2008. https://doi.org/10.1109/CVPRW.2008.4563095

Wikipedia (2014). Algorithme d’Edmonds-Karp. Retrieved March 17, 2015 from https://fr.wikipedia.org/wiki/Algorithme_d%27Edmonds-Karp

Notes

Note 1. E(x) denotes the integer part of x.

Note 2. Augmenting path is any path from the source S to the terminal T that can currently take more flow.

Note 3. This is the notation recommended by ISO. lb(x)=ln(x)*lb(e) and lb(e)= 1,442695041.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).