International Journal of
Science and Engineering Investigations vol. 6, issue 71, December 2017
ISSN: 2251-8843

Algorithmic Presentation in Order to Resource Scheduling on Graph According to Time and Cost Parameters with High Computing Power of Graphic Processor

Naghmeh Kamyar
Department of Computer Engineering, Central Tehran Branch, Islamic Azad University, Tehran, Iran
(Naghmeh_kam@yahoo.com)

Abstract- Using CUDA computer programing and performing this algorithm on graphics processor increase the speed considerably. According to high computing power of graphics processors, this algorithm was turned to CUDA language to be able to be calculated by graphics processor unit and increase the speed of calculations.

After implementing and performing different tests we witnessed the considerable difference in performance time and manifold speed of algorithm on graphics processing unit toward central process unit. The main constrain during designing optimization is time constraint. There are other constraints such as size, energy consumption and so on during scheduling. Time budget management is employed for different designing operations such as cables and gates sizing and providing library map. This paper aims at presenting an algorithm which can perform the resources scheduling on graph according to time and cost parameters in a reasonable period in a way to have the fastest form with the lowest cost. So, at first the cheapest maximum flow is introduced which is one of the most applicable discussions. The important point in using this algorithm is to find and replacing the parameters of flow and cost so that this replacement can solve the considered problem absolutely optimal. The next step is to implement the algorithm in an acceptable way. First we implement this algorithm in C++ language and the next step is to implement this algorithm on graphics processing unit.

Keywords- *Graphics Processor, Cost Parameters, Time Resources, Time Parameter*

I. Introduction

Today, graphics processing units have better computation power in comparison with new central processors. The programming models written for central processors are in fact, s serial model i.e. semi codes of a program hieratically were read and performed after translation and rarely used paralleling in data processing. The considered architecture is based on this model and has no important capabilities for performing several similar instructions simultaneously. While programming model of graphic processing programs called flow is a parallel model and has the capability of using paralleling techniques. However, new central processors with capabilities such as Hyper threading, SSE and applying multi core architecture make paralleling possible but their paralleling rate is much less than a graphic processor with 320 processing unit. Based on previous studies, the computing power of a Teslla graphic processing unit is more than 500 times of the power of an Intel 4-core processor. According to the basic concepts of paralleling, the graphic processing unit can be called optimization processors along paralleling duties and data. Graphic processing units are capable because of their special architecture and their architecture is not suitable for implementing a central processing unit. According to current architecture of x86 platform, there are different devices and equipment in system that processor managed them and in order to protect suitable communication and comprehensive management, the central processing unit needs this kind of structure and finally slower processing. The reason of naming graphs is that they can be shown graphically and also this graphic demonstration helps us understand many graph properties.

II. Literature

The maximum flow problem was presented by Harris and Ross as a simplified model of Russian railway in 1954. In 1955, Ford and Fulkerson created the first known algorithm called Ford-Fulkerson. During time, many improved solution has been found for maximum flow problem, particularly Edmonds Shortest Way algorithm and Karp and independent from Dinitz, Dinitz's cold flow algorithm, Goldenberg and Tarjan's push-relabel algorithm, Goldenberg and Rao's blocking Binary flow algorithm. Madry et al.'s electrical flow algorithm finds a relative optimal maximum flow but it only works in undirected graphs.

III. Problem analysis and identifying defaults

As it was explained before, different algorithms are used in order to find the cheapest maximum flow in graphs which it will be explained in detail later. This problem is a combination of two following problems:

If there is no more capacity constrain, it will turn to the shortest path. If all costs are zero, it will turn to maximum flow problem. According to the point above, one of the challenging problems is to find the shortest path in order to find the lowest cost that both Bellman-ford and dikstra algorithm are similar and optimal.

The shortest path problem: in graph theory, the problem of finding the shortest path is in fact finding a path between two heads (or nodes) so that the sum of weights of wings minimize. For example, the problem of finding the fastest way for moving from one place to another on a map can be considered. In this situation, the heads demonstrate places and wing show the parts of path that are weighted based on essential time for passing them. If we have a weighted graph (including V set of heads, E set of wings and weight function of f : E → R) and an element like v, we aim to find a path like P from v to v' so that $\sum_{p \in P} f(p)$ would be the shortest path among all existing paths from v to v'. This problem is sometimes called finding the shortest path between two heads to be distinguished from other general forms as follow:

The problem of finding the shortest path from origin in which the goal is to find the shortest path form origin head v to other heads in graph. The problem of finding the shortest path to destination in which the goal is to find the shortest path from all graph heads to destination head v. the problem of finding the shortest path between heads in which the goal is to find the shortest way between each head couple of v and v' in graph. These general manners have more efficient algorithm meaningfully. Assume that we have given to each wing e of G a real number of w(e) which is called its weight. Here, G is called a weighted graph with the weights on its wings. Weighted graphs appear in many theory applications of graphs. For example, in friendship graph, the weights can show the friendship among people. In communication graph, weights can show the construction weight or communication maintain. If H is a sub-graph of a weighted graph, the weights on its wings are $\sum_{e \in E(H)} w(e)$. Many optimization problems end up finding a certain sub-graph with the least (or the most) weight in a weighted graph. An example of these problems is the shortest path problem which is defined as so: a railway network connects several cities. Find the shortest path between cities. What we have to look for is a path with the shortest weight which connects two $\upsilon_{\circ}$ *and* $u_{\circ}$ heads in a weighted graph. Weights shows the rail distance between couples of cities that are directly connected then have non-negative weights. It is obvious that studying the shortest path problem is enough for simple graphs. So we will assume that G is simple. Also, we will assume that all weights are positive, however it isn't considered a serious constrain because if the weight of a wing is zero, two heads can be put together. We accept that if $u\upsilon \notin E$ then $w(u\upsilon) = \infty$. The most important algorithms for solving this problem are:

Dijkstra algorithm: it solves the problem of finding the shortest path between heads from origin to destination unit. Bellman-Ford algorithm: it solves the shortest path from origin in a way that the weights on wings can be negative. Floyd-Warshall algorithm: it solves the shortest path between heads.

IV. Dijkstra Algorithm

Dijkstra algorithm is one of the graph survey algorithm which presented by a Dutch computer scientist named Dijkstra in 1959. Dijkstra algorithm is famous as single-source shortest path and is similar to Prim's algorithm. If the graph has negative weight, this algorithm won't work and other algorithms such as Bellman-Ford having more time complexity should be applied. The policy of Dijkstra algorithm is similar to the greedy approach used in Prim's algorithm for finding an optimal subtree. This algorithm is named after the scientist presented it, Dijkstra. In addition to the shortest path $-(u_{\circ}, \upsilon_{\circ})$, this algorithm finds the shortest path between $\upsilon_{\circ}$ and other heads of G. the main idea is: assume that S is a subgroup of V, then $\overline{s}, u_{\circ} \in s$. If $p = u_{\circ} ... \overline{u}\,\overline{\upsilon}$ is one of the shortest paths between $u_{\circ}$ and $\overline{s}$, then $-(u_{\circ}, \overline{u}), \overline{u} \in s$ and p should be the shortest path $-(u_{\circ}, \overline{u})$. In conclusion:

$$d(u_{\circ}, u) + w(u, \upsilon) \qquad (1)$$

And the distance between $u_{\circ}$ and $\overline{s}$ can be calculated with the formula below:

$$d(u_{\circ}, \overline{s}) = \min_{u \in s, v \in s} - \{d(u_{\circ} u) + w(u\upsilon)\}) \qquad (2)$$

This formula is considered as the basis of Dijkstra algorithm, beginning from $s_{\circ} = \{u_{\circ}\}$ we have $s_{\circ}, s_1, ... s_{v-1}$ from v subgroups that at the end of ith level, the shortest path from $u_{\circ}$ to all s_i heads have to be identified. The first step is to determine the closest head to $u_{\circ}$ and it can be done by computing $d(\upsilon_{\circ}, \overline{s}_{\circ})$ and selecting a head such as $u_1 \in \overline{s}_{\circ}$ so that $d(u_{\circ}, u_1) = d(\upsilon_{\circ}, \overline{s}_{\circ})$. According to (2) we have:

$$d(u_{\circ}, \overline{s}_{\circ}) = \min_{u \in s, v \in s} - \{d(u_{\circ}, u) + w(u\upsilon)\} = \min_{v \in s.} - \{w(u_{\circ}\upsilon)\} \qquad (3)$$

Therefore, $d(u_{\circ}, \overline{s}_{\circ})$ can be easily computed. Now we consider $s_1 = \{u_{\circ}, u_1\}$ and assume that p_1 shows $u_{\circ}u_1$ path which is obviously the shortest path $-(u_{\circ}.u_1)$. Generally, if $S_k = \{u_{\circ}.u_1, ...\}, u_k$ and the shortest path $p_k, ..., p_2, p_1$ has been identified, $d(u_{\circ}, \overline{s}_k)$ has been computed by (1) and we select the $u_{k+1} \in \overline{s}_k$ head that $d(u_{\circ}, u_{k+1}) = d(u_{\circ}, \overline{s}_k)$. Based on (1) per $j \leq k$ we have:

$$d(u_{\circ}, u_{k+1}) = d(u_{\circ}, u_j) + w(u_j, u_{k+1}) \quad (4)$$

By adding $u_j u_{k+1}$ to p_j path we obtain the shortest path $-(u_{\circ}, u_{k+1})$. In each step, the paths form a connected graph called tree. So, this algorithm is considered as a "tree growing" process and final tree has the characteristic to be the shortest path $-(u_{\circ}, \upsilon)$ for each head of υ for the distance between υ *and* $u_{\circ}$. Dijkstra algorithm only determines the distance between $u_{\circ}$ and all other heads and identifies the shortest path. On the other hand, the shortest path can be easily achieved by maintaining the past heads of tree and following them. Dijkstra algorithm is an example of what Edmonds has called a good algorithm. A graph theory algorithm will be good if the number of computational levels for implementing on each G graph is limited by ε *and* ν, for example $3\nu^2\varepsilon$. An algorithm whose implementing needs an exponential number of levels (example 2^{ν}), is not efficient for big graphs. In order to show that an algorithm is good, we precise that the computations in 2 and 3 of diagram need to $\frac{\nu(\nu-1)}{2}$ and $\nu(\nu-1)$. A question about flow diagram is how we can understand that a head belongs to $\bar{S}$ or not. Driphas (1969) presented a technique which needs $(\nu-1)^2$ on the whole. So, if we consider every sum or comparison as a main computational unit, the required computational sum for this algorithm is almost $\frac{5\nu^2}{2}$ and the result will be ν^2 . it is said that $\int(\nu,\varepsilon)$ function is of $g(\nu,\varepsilon)$ level, if there is a positive constant number such as c. so that per any ν,ε we have $\frac{\int(\nu,\varepsilon)}{g(\nu,\varepsilon)} \le c$.

Although the problem of the shortest path can be solved by a good algorithm but there are many other problems in graph theory that there is no good algorithm for them.

Dijkstra algorithm process is as follows:

1. Selection of origin head
2. Identifying S group including graph heads. At the beginning this group is empty and by improving algorithm, this group contains the heads with the shortest path.
3. It puts the origin head with zero index inside S.
4. For heads out of S, the index is considered as the wing length + index of previous wing. If head has index out of group, new index will be the least amount of last index and wing length + index of previous head.
5. A head with fewer indexes is selected from out of group and added to S.
6. This job will be continued until the destination head enters S.

At the end if the destination head has an index, its index shows the distance between origin and destination. Otherwise, there is no path between origin and destination.

Also, another index can be considered for each head in order to find the path. Then, after ending algorithm, the shortest path will be found between two points by following the previous heads.

Bellman-Ford algorithm is a graph survey algorithm that solves the shortest path from origin for weighted graphs with negative weights. Dijkstra algorithm solves the similar problems in less time but in that algorithm the weight should be non-negative. So, practically Bellman-Ford algorithm is used for graphs with negative weight. It has to be mentioned that if a flow graph has negative weight that can be obtained from origin, the shortest path problem will have no answer, because there will be paths with less and less weight by surveying the flow. Performing |v|-1 algorithm will be defined for each v head, dv at the end of ith is equal to weight of shortest path from origin to v with this condition that the number of wings has to be i. therefore, at the end of |v|-1 level, dv is equal to the weight of shortest path from origin to v. the important point is that because there is no negative weight, the shortest path with maximum |v|-1 wing from origin to v is the shortest path from origin to v in graph.

The basis of algorithm is relaxation of all graph wings in each level. (u,v) wing relaxation means if du + weight(u,v) < dv then du + weight(u,v) = dv. Then if the relaxation of all wings repeated for |V|th time and d changes after this level, it may be concluded that graph has a negative flow obtainable from origin. Then Bellman-Ford algorithm has the capability of identifying negative flow. This algorithm is so similar to Dijkstra algorithm with only difference that it starts from one head and gives index to all neighbor points. Continuing this process and keeping smaller index, the shortest distance from origin head to all graph point will be calculated and the shortest path will be identified between origin and destination.

V. Applications

The problem of finding the shortest path is used for finding path between real locations including traffic on internet maps example Google maps. If a virtual machine is considered graphically that heads express manners and wings expresses traffic, the algorithm can be used to find the shortest path as tools for finding an optimal tail of selection in order to reach a particular manner. This algorithm can be used in order to find a lower bound of needed time for reaching a particular manner. For example, if heads express the manners of a puzzle like cube of a head and each directed wings expresses a manner or a flow, the shortest path algorithms can be used so that these algorithms may lead to a solution with the least number of flow. Sometimes in communication network structure or telecom networks, this problem is called the problem of finding the least cost or the least delayed path that often have a close relationship with finding the widest path. The widest path can be as the same of band width. This problem can be applied in Robotic, traffic and designing integrated circuits.

VI. The maximum flow

In maximum flow problem we consider pushing the maximum flow of an origin head to a destination in a graph by considering this constrain that flow never can go beyond its capacity. In optimization theory, the maximum flow includes finding a practical maximum flow inside a single-origin and single-destination flow network.

VII. Edmonds- Karp algorithm

This algorithm wants to find the maximum form origin s to t destination. This algorithm is similar to Ford-Fulkerson algorithm and the only difference is that in this algorithm constrains of Ford-Fulkerson algorithm improves because the computation of increasing path is implemented by breadth-first search (bfs). If we perform bfs in Ford-fulkerson algorithm, it needed more time constrain. The path should be the shortest path that has the capacity to be found by bfs search and the wings are allowed to have the same size. The maximum charge is equal to bfs. The other characteristic of this algorithm is that the shortest path increases each time. What this algorithm can do is to find increased path by bfs and if there is increasing path, it will find the lowest capacity and adds the minimum capacity to the flow of all wing. In this case, it will find the maximum charge between origin and destination that is equal to bfs. The performance of this algorithm is demonstrated in code below. The input of algorithm: the input of this algorithm is a graph which has an s origin head and a t destination head and each capacity is written on each wing. The output of algorithm: the maximum flow from s to t when there is an increasing path with characteristics above. The time of performing this algorithm is o(VE)^2. In each time at least one of the E wings is full and the distance from full wing to origin should be longer than the last time it was full and this length is equal to V number of heads.

VIII. Implementation description

There is a charge network in which directed graph is $G(V,E)$ with origin $s \in V$ and destination $t \in V$ and the wing $(u,v)\in E$ has $c(u,v) > 0$ capacity, $f(u,v) \geq 0$ flow and $a(u,v)$ cost. Our problem is to find the maximum flow which has the lowest cost among maximum flows that can be called the lowest cost maximum flow. A graph is considered including capacity and cost parameters per each flow unit in wing.

On the left, the first number is pass flow (f), the second number is the capacity of each wing for flow (CP) and the third number is the cost of each flow unit (C) in that wing (f/cp/c). By writing an optimal algorithm and in some parts by using optimal algorithms such as the shortest path algorithm and modifying them, the lowest cost maximum flow can be achieved.

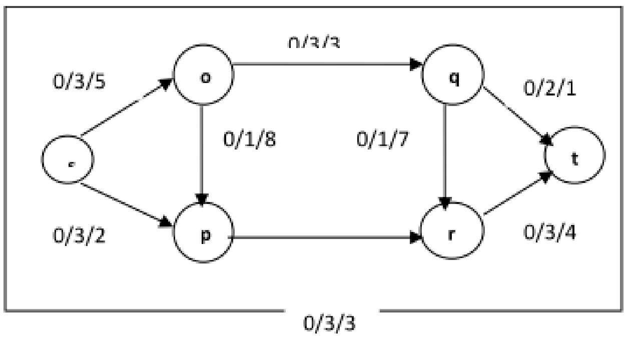

Figure 1. An example graph in order to solve the lowest cost maximum flow

The conditions of G graph

a) Directed

b) There is one resource and one destination $s,t \in V$

c) Each wing includes a positive capacity $c(u,v) \geq 0$

d) Each wing has a cost that is a function of flow. $a(u,v) > 0$

e) Along with the resource and destination node, the sum of input flow to a node is equal to the sum of output flows from the node.

$$\sum_{w \in V} f(s,w) = d \quad \text{and} \quad \sum_{w \in V} f(w,t) = d \tag{5}$$

f) The maximum flow passing a wing cannot go beyond the capacity of that wing $f(u,v) \leq c(u,v)$.

On the basis of definition and conditions mentioned before, we solve this problem as: the paths between origin and destination are found. In other words, between maximum flows, the shortest path (Bellman-Ford algorithm) is the response. Practically Bellman-ford algorithm can obtain the lowest cost being a function of flow and an example of the shortest path according to maximum flow paths. According to the remained capacity of each wing (difference between pass flow and capacity of each wing) the flow that can be pass which is practically equal with the minimum remained capacity between wings of path is computed with cost.

$$\min_{(u,v)\in E} (c(u,v) - f(u,v)) \tag{6}$$

If there is no other path (even one wing cannot pass the considered flow, it practically gets out of path) the algorithm will practically end up. The sum of input flow to destination gives the maximum flow.

$$\sum_{w \in E_{in}} f(w,t) \tag{7}$$

The cost of graph is concluded:

$$\sum_{(u,v)\in E} f(u,v).a(u,v) \tag{8}$$

If there is more than several form to reach the maximum flow, the form with lowest cost is the solution.

The lowest cost of transmitting a flow unit from origin node to any node is calculated by Bellman-Ford algorithm.

$$\mathrm{dist}(s) = 0, \mathrm{dist}(v) = \min(\mathrm{dist}(v), \mathrm{dist}(u) + \mathrm{cost}(u,v)) \quad (9)$$

Then by using the obtained amounts from Bellman-Ford we began to calculate the maximum flow. On any wing

$$\mathrm{pr}(v) = \min(\mathrm{pr}(v), \mathrm{pr}(u) + \mathrm{cost}(u,v) + \mathrm{dist}(u) - \mathrm{dist}(v)) \quad (10)$$

We obtain the maximum pass flow through considered path.

$$\mathrm{maxflowtrans}(v) = \min(maxflowtrans(u), cap - f) \quad (11)$$

In each flow the cost is obtained according to the formula below:

$$\mathrm{cost}_{\mathrm{path}} = \sum_{i,j\in V}^{m} f(i,j) * \mathrm{cost}(i,j) \quad (12)$$

And finally the sum of all pass flows through each path is equal to maximum flow:

$$\mathrm{maxFlow} = \sum_{i\in \mathrm{path}}^{n} \mathrm{flow}_i \quad (13)$$

And the sum of all paths is equal to the lowest cost of flow:

$$\mathrm{minCost} = \sum_{i\in \mathrm{path}}^{n} \mathrm{cost}_i \quad (14)$$

For example in this graph, maximum 5 unit flow is pushed from S to T that the achieved paths are as follow:

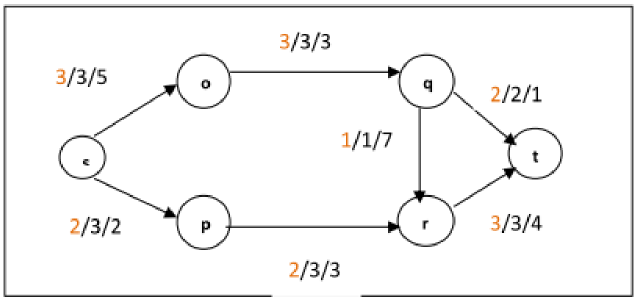

Figure 2. The first path with the maximum flow

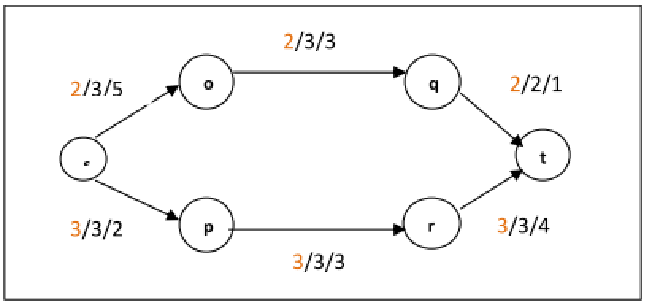

Figure 3. The second path with the maximum flow

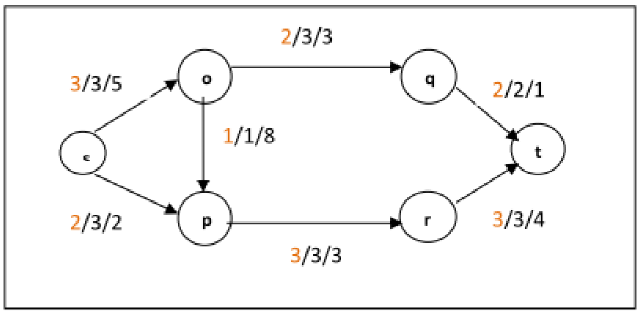

Figure 4. The third path with the maximum flow

Then, according to the formula below the cost of each path is:

$$\mathrm{total}_{\mathrm{cost}} = \sum_{i,j\in V}^{m} f(i,j) * \mathrm{cost}(i,j) \quad (15)$$

In first path the sum of pass cost is 55.

In second path the sum of pass cost is 45.

In third path the sum of pass cost is 56.

In the following V, E, CP, C, Flow and TotalCost expresses heads, wings, capacity of flow in each wing, the cost unit per flow in each wing, the maximum rate of flow to destination node and the minimum cost of flow to destination, respectively. In this test, two systems are considered called system A and system B.

Characteristics of system A:

Cpu: core i3, 3.3 GHZ

Ram : 2GB

Graphic: Nvidia Gforce 210, 1GB

Characteristics of system B:

Cpu: core i7, 3.6 GHZ

Ram : 8GB

Graphic: Nvidia Gforce 620, 1GB

T_{c++} is the time of performing algorithm C++ and T_{cuda} is the time of performing CUDA algorithm.

In this chapter the algorithm with C++ language is considered as first algorithm and algorithm with CUDA language is called second algorithm.

The performance of the cheapest maximum flow with input and graphs with less than 500 heads, in this part the effect of the number of nodes and wings on speed and precision of considered algorithm and objective function (cost) with other characteristics constant were investigated.

First performance

In first performance the characteristics of example performance is as follows:

V=4, E=5

TABLE I. THE COST PER FLOW UNIT OF WINGS IN FIRST EXAMPLE GRAPH

Node	1	2	3	4
1	-	4	8	-
2	-	-	-	5
3	-	3	-	6
4	-	-	-	-

TABLE II. THE MAXIMUM CAPACITY OF FLOW FROM WING IN FIRST EXAMPLE GRAPH

Node	1	2	3	4
1	-	1	3	-
2	-	-	-	3
3	-	2	-	2
4	-	-	-	-

These amounts are obtained for graph by performing algorithm:

Total Cost = 53, Flow = 4

A: $T_{c++} = 16, T_{cuda} = 16$

B: $T_{c++} = 4, T_{cuda} = 6$

Time complexity of algorithm

The diagram will be as below according to the performance complexity function of algorithm.

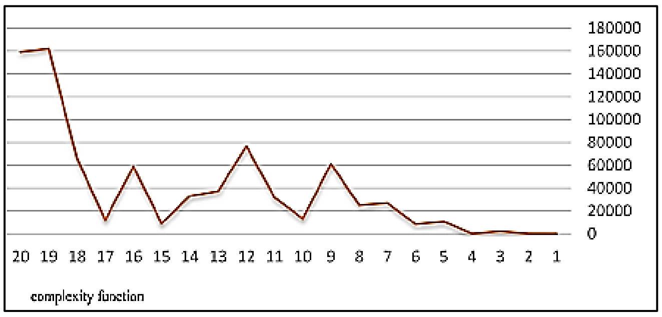

Figure 5. Diagram based on performance complexity function of the time performance of algorithm in first system according to the time achieved in first system with C++ and CUDA algorithm performance

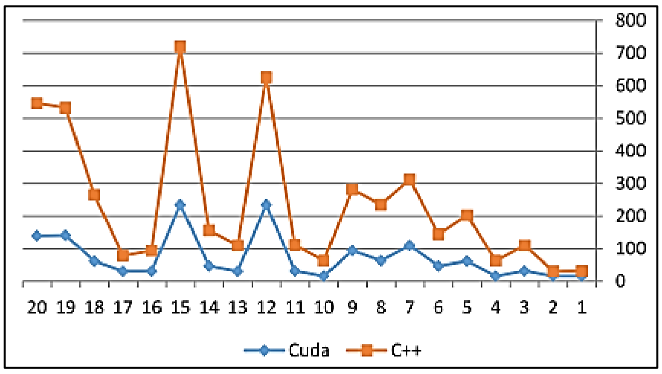

Figure 6. Linear diagram based on performance time on first system (less than 500 heads)

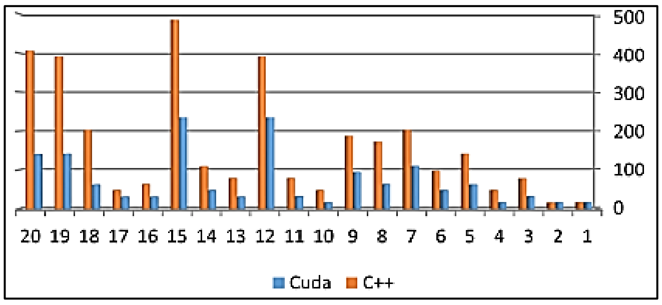

Figure 7. Column chart based on performance time on first system (less than 500 heads)

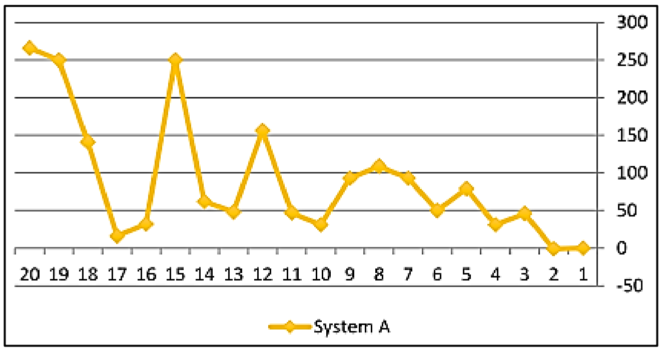

Figure 8. Difference rate of CUDA and C++ algorithm based on the performance time on first system (less than 500 heads). The comparison of two systems in graphs less than 500 heads

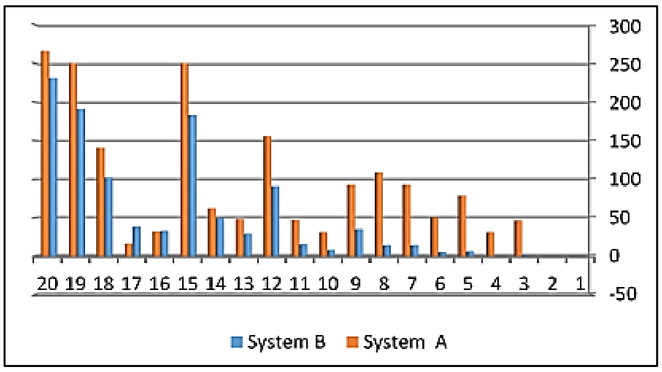

Figure 9. The comparison of difference of performance rate on both systems (less than 500 nodes)

As it can be seen, the results in performances with more complex data are more satisfying. Toward first tests and measured output in larger data, the performing time difference and efficiency of graphic processing unit is obvious. Lower cost performances of maximum flows with inputs and graphs with more than 1000 heads resulted in that the difference between the speed of algorithm on more and more complex inputs are so considerable. Now in this section, the tests are conducted on data more than 1000 head.

IX. First performance

In first performance the characteristics of example performance are as follows:

V = 1000

E = 5000

By performing algorithm, the below amounts are achieved for graph:

Total Cost = 4456

Flow = 51

A: $T_{c++} = 4891$, $T_{cuda} = 1297$

B: $T_{c++} = 5208$, $T_{cuda} = 1174$

The algorithm performance time in first system

According to time achieved in first system by performing C++ and CUDA algorithm the resulted diagram is as:

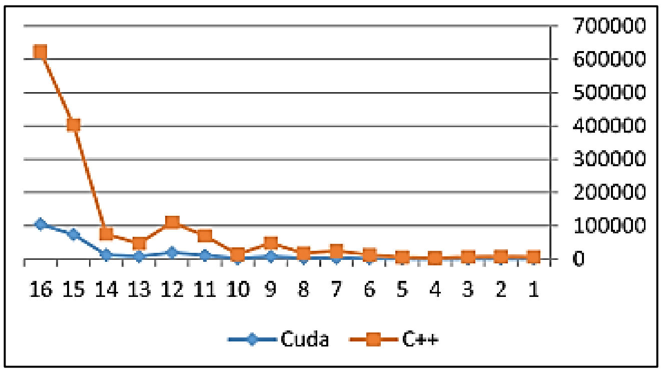

Figure 10. Linear diagram based on the performance time on first system (more than 1000 head)

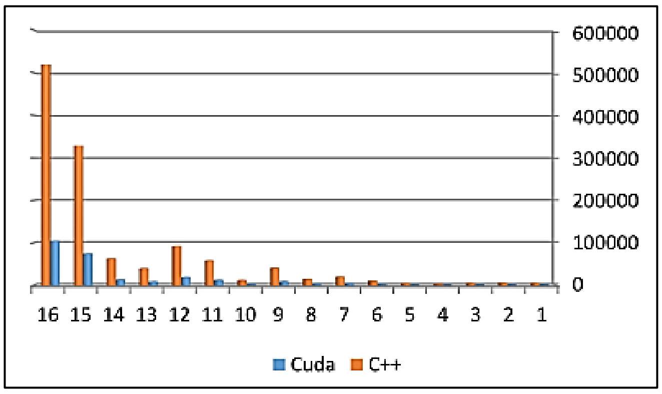

Figure 11. Column chart based on performance time on first system (more than 1000 head)

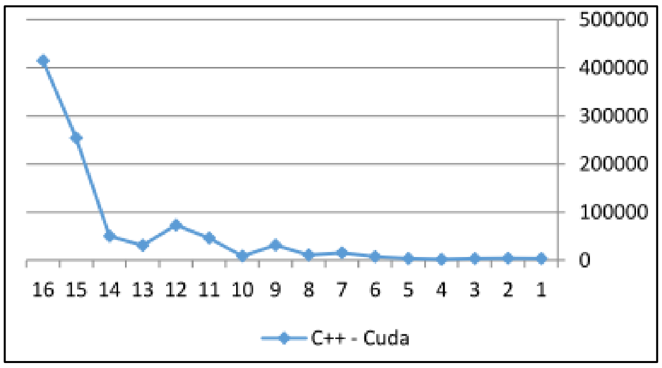

Figure 12. The difference rate of CUDA and C++ algorithm based on performance time in first system (more than 1000 heads)

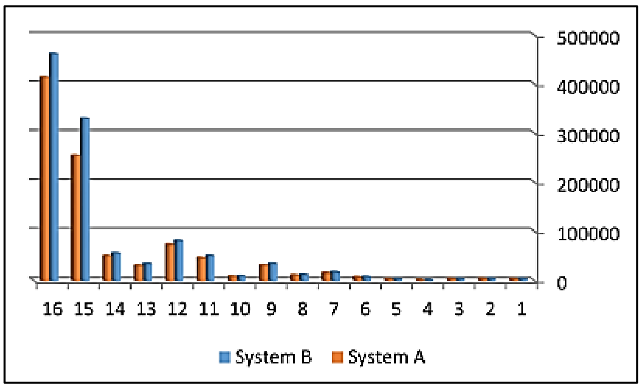

Figure 13. The comparison between the differences of performing time on both systems (more than 1000 heads)

X. CONCLUSION

The main constrain in designing optimization is time constrain. During scheduling, there are other constrains such as size, energy consumption and so on. Budget management is applied for different operations such as gates and cable sizing and providing library map. The aim of this paper is to present an algorithm which can do a resource scheduling on graph in a reasonable time level according to cost and time parameters in order to have the fastest manner with the lowest cost. At first, we introduced the lowest cost maximum flow being one of the most applicable. A very important point in using this algorithm is to find and replace the parameters with cost and flow so that this problem can be solved completely optimal. The next step is to implement the algorithm with acceptable level. First this algorithm was implemented to C++ language and then implemented on graphic processing unit. In recent years, the increasing applications of graphic cards have the researchers implement processing power on non-graphic applications. As a result, a new branch in computer science has been created called computations with all-purpose aims on graphic processing unit. This field aims at using graphic card as a computational coprocessor in non-graphic and public programs. Furthermore, there was a broad modification from computational industry to parallel computations and almost all computers by 2010 along with multi core processors have done transmission operation. Computer industry faces a revolution of parallel programing to be able to be absolutely effective in computer science. In conclusion, CUDA nividia has operated as one of the most important languages designed for parallel computations.

According to the high computational power of graphic processor unit, this algorithm is turned to CUDA language to be able to be computed by graphic processor unit and computation speed increases. After implementing and performing different tests we witnessed a considerable difference in performance time and speed of algorithm on graphic processing unit toward central processing unit. As it can be seen from the results, using CUDA programming and performing this algorithm on graphic processing unit has increased the speed considerably.

The results from performing algorithm

According to amounts obtained from algorithm we have witnessed a considerable effect of graph size and flow on speed and precision of algorithm.

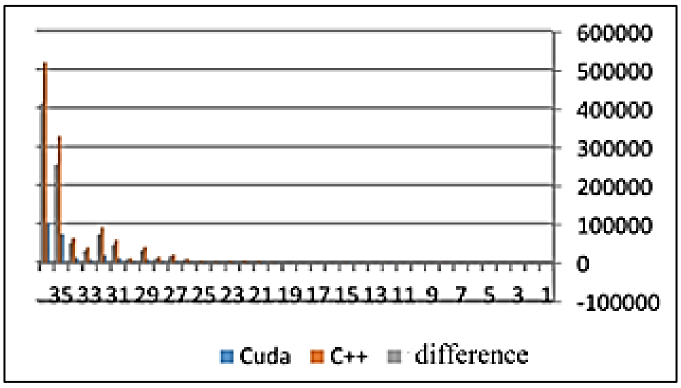

Figure 14. Performance time of algorithm in C++ and CUDA and their differences in system A

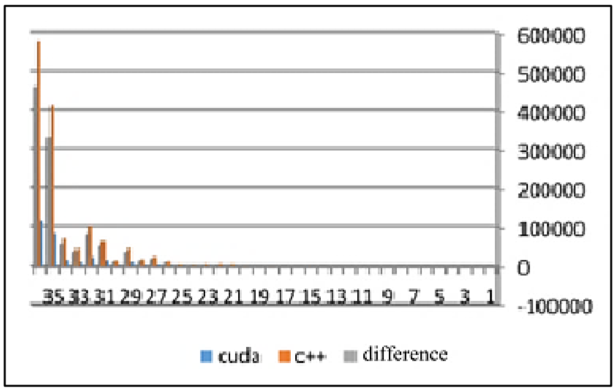

Figure 15. Performance time of algorithm in C++ and CUDA and their difference in system B

REFERENCES

[1] A. Kahng, S.Mantik, and I.L. Markov. (2002) "Min-Max Placement for Large- Scale Timing Optimization", In the proceedings of *ACM International Symposium on Physical Design*, pp. 143-148.

[2] Barron J, Fleet DJ, Beauchemin S. (1994) Performance of Optical Flow Techniques. Int'l J Comp Vision. 12:43–77.

[3] C. Chen, E. Bozorgzadeh, A. Srivastava, and Majid Sarrafzadeh.(2002) "Budget Management with Applications". In *Algorithmica*, vol 34, No. 3, pp. 261- 275.

[4] C. Chen, X. Yang, M. Sarrafzadeh. (2000) "Potential Slack: An Effective Metric of Combinational Circuit Performance. In the proceedings of *ACM/IEEE International Conference on Computer-Aided Design*, pp. 198-201.

[5] C. Kuo and A. C.-H Wu. (2000)"Delay Budgeting for a Timing-Closure-Design Method", In the proceedings of *International Conference on Computer- Aided Design*, pp. 202 207.

[6] Chetverikov D. (2003) Applying feature tracking to Particle Image Velocimetry. International Journal of Pattern Recognition and Artificial Intelligence. 17:487–504.

[7] E. Bozorgzadeh, S. Ghiasi, A Takahashi, and M. Sarrafzadeh. (2004) ”Optimal Integer Delay Budget Assignment on Directed Acyclic Graphs”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,Vol. 23, No.xx,.*

[8] Ford, L.R., Jr.; Fulkerson, D.R. (1962), *Flows in Networks*, Princeton University Press.

[9] Ford,L.R.; Fulkerson, D. R.(1956) "Maximal flow through a network". *Canadian Journal of Mathematics* **8**: 399, 1956.

[10] Gass, Saul I.; Assad, Arjang A. (2005) "Mathematical, algorithmic and professional developments of operations research from 1951 to 1956". *An Annotated Timeline of Operations Research*. International Series in Operations Research & Management Science **75**. pp. 79–110.

[11] Goldberg, A., Tarjan, R.,(1988) A New Approach to the Maximum-flow Problem, Journal of the Association for Computing Machinery 35, 4, 921–940.

[12] H. R. Lin and T. Hwang.(1995) “Power Reduction by Gate Sizing with Path- Oriented Slack Calculation”. In the proceedings of ‘*IEEE ASP-DAC,* pp. 7 12.

[13] H. Szymanski,(2013) "Max-Flow Min-Cost Routing in a Future-Internet with Improved QoS Guarantees", In the proceedings of *IEEE Transactions on Communication, Vol. 61, No.4*.

[14] H. Youssef, E. Shragowitz. (1955) “Timing Constraints for Correct Performance”, In the proceedings of *ACM/IEEE International Conference on Computer-Aided Design*, 1990.

[15] Harris, T. E.; Ross, F. S. "Fundamentals of a Method for Evaluating Rail Net Capacities". *Research Memorandum* (Rand Corporation).

[16] http://www.wikipedia.com

[17] Hu W, Tan T, Wang L, Maybank S. (2004) A Survey on Visual Surveillance of Object Motion and Behaviors. IEEE Transactions on Systems, Man, and Cybernetics. 34:334–352.

[18] J. Luo and N. Jha. (2001) ”Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems”. In the proceedings of *IEEE/ACM Design Automation Conference*.

[19] Kelner, J. A.; Lee, Y. T.; Orecchia, L.; Sidford, A. (2014). "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations". *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. p. 217.

[20] Knight, Helen. (2014) "New algorithm can dramatically streamline solutions to the ‘max flow’ problem". MIT News. Retrieved 8 January.

[21] M. Sarrafzadeh, D. Knol, and G. Tellez.(1997) “Unification of Budgeting and Placement” In the proceedings of *ACM/IEEE Design Automation Conference*, June 1997.

[22] M.Hulkkonen.(2011), "Graphics Processing Unit Utilization in Circuit Simulation ",.Master's Thesis. Aalto University School of Electrical Engineering, August.

[23] M.Sarrafzadeh, D. A. Knol, G.E. Tellez. (1997)“A Delay Budgeting Algorithm Ensuring Maximum Flexibility in Placement In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, , Vol. 16, No. 11 , pp. 1332 -1341, Nov.

[24] *Matov, M. Edvall, Ge Yang, and G. Danuser. (2011*) "Optimal-Flow Minimum-Cost Correspondence Assignment in Particle Flow Tracking". *Comput Vis Image Underst.; 115(4): 531–540, Apr.*

[25] Medioni G, Cohen I, Bremond F, Hongeng S, Nevatia R.(2001) Event Detection and Analysis from Video Streams. IEEE Transactions on Pattern Analysis and Machine Intelligence. 23:873–889.

[26] Micheli ED, Torre V, Uras S. (1993) The accuracy of the computation of optical flow and of the recovery of motion parameters. IEEE Trans Pattern Anal Mach Intell. 15:434–447.

[27] Morris BT, Trivedi MM. (2008) A Survey of Vision-Based Trajectory Learning and Analysis for Surveillance. IEEE Transactions on Curcuits and Systems for Video Technology. 18:114–1127..

[28] S. Bakshi and D. Gajski. (1996) “Component Selection for High-Performance Pipelines”. In *IEEE Transactions on Very Large Scale Integrated Systems*, pp. 181-194, Vol. 4, No. 2.

[29] S. Chen, C, Chern, (2000) "Max-Flow Min-Cost Algorithm for A Supply Chain Network", In the proceedings of *APDSI 2000 Full Paper, July*..

[30] S. Ghiasi, K. Nguyen, E. Bozorgzadeh, and M. Sarrafzadeh. (2003) ”On Computation and Resource Management in an FPGA-based Computing Environment”. a poster presentation in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, February.

[31] S. L. Lin and J. Allen. (1986) “MinPlex- A Compactor that Minimizes the Rounding Rectangle and Individual Rectangles in a Layout”. In the proceedings of *ACM/IEEE Design Automation Conference*, pp. 123-130.

[32] Schrijver, A. (2002) "On the history of the transportation and maximum flow problems". *Mathematical Programming* **91** (3): 437–445.

[33] Schwartz, B. L. (1966). "Possible Winners in Partially Completed Tournaments". *SIAM Review* **8** (3): 302.

[34] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.(2006) "Maximum Flow". *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill. pp. 643–668. ISBN 0-262-03293-7.

[35] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte, and Y. Tsai.(2001) ”Exploiting VLIW schedule slacks for dynamic and leakage energy reduction ”. In the proceedings of *IEEE International Symposium on Microarchitecture*, 2001.

[36] Moavenian, K. (2012) "implementing several examples of parallel algorithm using multi-core graphic cards", Payam Noor University, Mashhad.