The partitioning technique of directed cyclic graph for task assignment problem

Cite as: AIP Conference Proceedings **1750**, 020010 (2016); https://doi.org/10.1063/1.4954523
Published Online: 21 June 2016

Wan Nor Munirah Ariffin and Shaharuddin Salleh

The Partitioning Technique of Directed Cyclic Graph for Task Assignment Problem

Wan Nor Munirah Ariffin[1, a)] and Shaharuddin Salleh[2, b)]

[1]*Institute of Engineering Mathematics, Universiti Malaysia Perlis, Pauh Putra Campus, 02600 Arau, Perlis.*
[2]*Department of Mathematical Sciences, Faculty of Science, Universiti Teknologi Malaysia,81300 Skudai, Johor.*

[a)]Corresponding author: munirah@unimap.edu.my
[b)]ss@utm.my

Abstract. The scheduling and mapping of task graph to processors is considered to be the most crucial NP-complete in parallel and distributed computing systems. In this paper, the theoretical graph application using simple partitioning technique is presented to assign a number of tasks onto two processors. This paper addresses a directed-weighted cyclic graph. The effort is to reduce the graph onto directed acyclic graph. A Kernighan-Lin algorithm is applied to obtain the partition of tasks. Combining the technique of reduction and partitioning lead to an efficient graph-mapping concept.

INTRODUCTION

Graph theory has many practical applications in various disciplines such as engineering, medical, economics, computer science and social science. Informally, a graph can be stated as a collection of edges and vertices together with a rule on how the vertices are connected to one another with the edges. Graph is an excellent tool. It can represent various real life applications with regard to the establishment and maintenance of systems. We can read all information from a graph and record the abstract definition from the graph in a unique way. Hence, the practical applications can be reduced into graphs.

Research on task allocation and scheduling problems began in the 1960's, and has become a popular research topic in the past few decades. The produced schedule is judged based on the performance criterion we are trying to optimize. To solve the problem of the application, a graph which represents the problem can be transformed or reduced into a simpler form of graph rather than solved them directly. As the number and complexities of graph based applications increase, rendering these graphs more compact, easier to understand, and navigate through are becoming crucial tasks. One approach to graph simplification is to partition the graph into smaller parts, so that instead of the whole graph, the partitions and their inter-connections need to be considered. Common approaches to graph partitioning involve identifying sets of edges (edge-cuts) or vertices (vertex-cuts) whose removal partitions the graph into the target number of disconnected components.

One of the performance measures in scheduling is to obtain a minimum total completion time. In order to achieve the goal, a good partitioning algorithm and assignment strategy of tasks onto processors should be implemented. Most of the previous research done by other researchers focused on undirected graph and bipartite matching [1,2]. If a directed graph or directed cyclic graph is given, how it can be mapped onto processors? Motivated by the work done by researches [2,3,4], a matching technique for a small number of nodes for directed cyclic task graph (DCG) is constructed. In this paper, the nodes are partitioned using Kernighan-Lin algorithm (hereafter, the acronym KL-algorithm will be used throughout this paper).

The organization of the paper is as follows:

Section 1 will discuss some related definitions on directed graph. The model of task scheduling problem is presented in Section 2. The reduction of directed cyclic graph and KL-algorithm also will be explained in Section 2. Conclusions will be drawn in the last section.

Advances in Industrial and Applied Mathematics
AIP Conf. Proc. 1750, 020010-1–020010-8; doi: 10.1063/1.4954523
Published by AIP Publishing. 978-0-7354-1407-5/$30.00

DEFINITIONS

It is important to know few types of graph. A directed graph G is an (ordered) pair (V, E) such that V is a finite set of nodes and E is a set of ordered pairs of nodes. That is, if $e \in E$, then $e = (X_i, X_j)$ and $\{X_i, X_j\} \subseteq V$. Figure 1 shows an example of the directed graph.

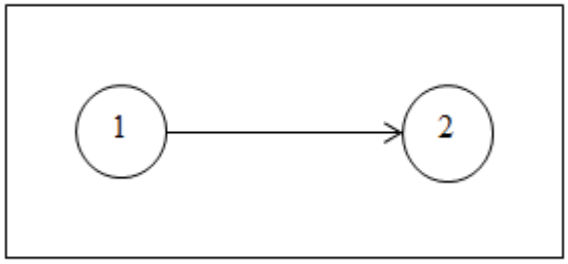
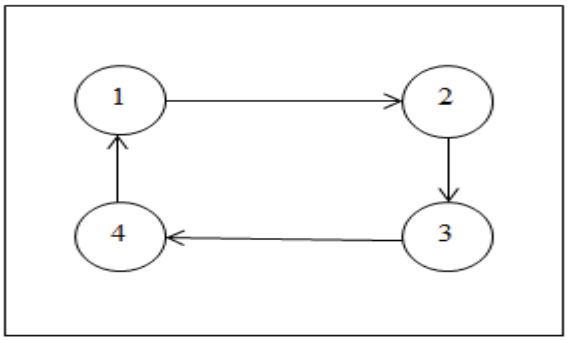
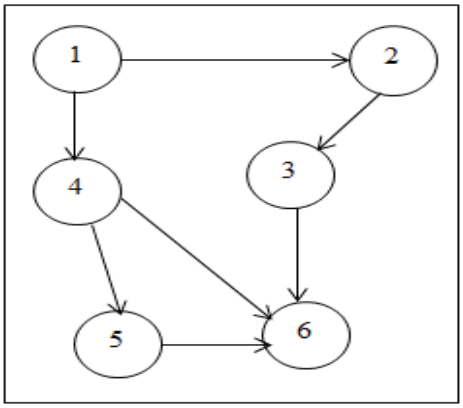


FIGURE 1. A directed graph with $V = \{1, 2\}$ and $E = \{(1, 2)\}$.

If G is a directed graph, then a directed cyclic graph (DCG) is a cycle in G consists of a path from a node to itself. However, a directed acyclic graph (DAG) is a graph which does not contain any directed cycle. If $G = (V, A)$, is a DAG containing n nodes, then a topological (or ancestral) ordering $(X_1, ..., X_n)$ of the nodes in V is any ordering such that, if $(X_i, X_j) \in A$, then $i < j$. Both DCG and DAG are shown in Figure 2.



(a)



(b)

FIGURE 2. In this graph (a), $[1,2,3,4,1]$ is a directed cyclic and in (b) is a directed acyclic graph.

Let a task graph G be a DCG composed of N nodes $n_1, n_2, n_3, ..., n_N$. Each node is termed a task of the graph which in turn is a set of instructions that must be executed. A node has one or more inputs. A node with no parent is called source node while a node with no child is called destination node. The graph also has E directed edges representing the communication between the nodes. The weight on an edge is called the communication cost of the edge and denoted by $c(n_i, n_j)$.

THE MODEL FOR TASK SCHEDULING PROBLEM

In this paper, a directed cyclic task graph is presented. The main goal of our study is to obtain a minimum total completion time of a task scheduling. In order to achieve the goal, a good partitioning algorithm and mapping strategy of task that will be assigned to processor should be implemented. The developed algorithm in this paper is very simple and it is constructed using the following steps:

Step1: Identify the existence of cycle(s) in the task graph.
Step2: All nodes are arranged in two columns like wrapped-butterfly topology, inspired by DPillar [5].
Step3: A source and destination nodes are chosen from the graph.
Step4: Obtain the path from source to destination nodes such that they do not share common intermediate nodes. The produced graph now is a directed acyclic graph.
Step5: A KL-algorithm is applied to the current task graph.
Step6: The nodes are then being mapped onto two processors to obtain a schedule of the task.

These steps are constructed to solve the problem, named Model 1 [6].

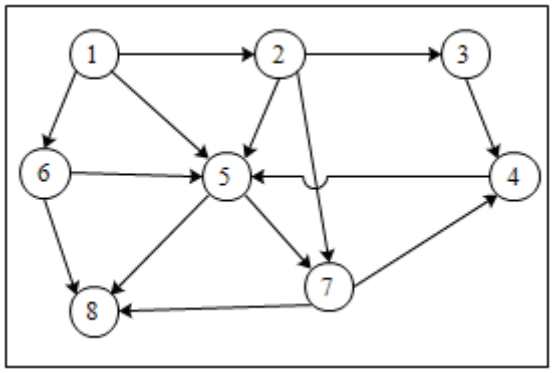


FIGURE 3. Model 1: A DCG with eight number of nodes.

The Reduction of Directed Cyclic Task Graph

The directed cyclic graph is reduced to a directed acyclic graph. This is done by arranging the nodes in two-column. The purpose of doing this is as alternative way of finding a directed acyclic graph. The nodes of graph as shown in Figure 3 is then arranged as in Figure 4. The communication (edges) between the nodes also presented in Fig. 4.

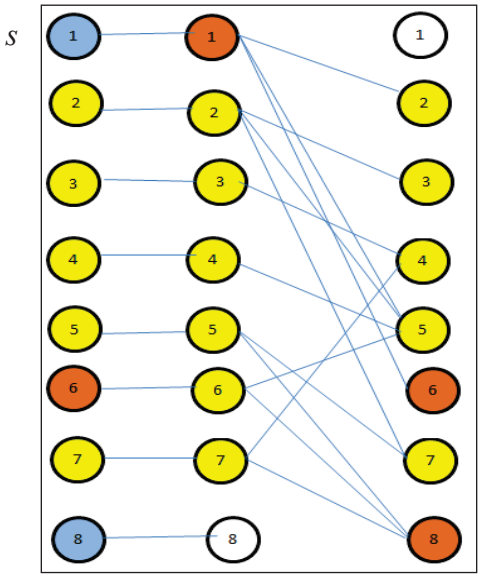


FIGURE 4. Nodes are arranged in two-column.

The next step requires finding the directed acyclic graph. As an example, the shortest path from the source node 1, denote as *s* to the destination node 8, denote as *t*. So, the path from *s* to *t* must be found such that all nodes must be chosen and they must not share common intermediate nodes. As can be seen from Fig. 4, there are three nodes can be chosen from the source node 1. Let say, node 2 is chosen from node 1. Next, from node 2, node 3, 5 and 7 can be reached. Next, node 3 is chosen. From node 3, only node 4 can be chosen. Next, node 5 is chosen from node 4. After that, node 7 is chosen from node 5. Note that there are two nodes that are not chosed yet. So, again from the source node, node 6 is chosen. From node 6, either node 5 or node 8 can be chosen. Notice that, same intermediate nodes are not allowed. Thus, node 5 is strictly not available to be chosed. Next, from node 6, node 8 can be reached, which is the destination node. Now, the directed acyclic graph is obtained. It is illustrated as in Figure 5.

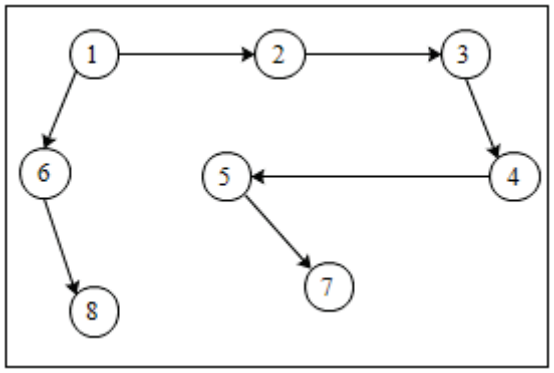


FIGURE 5. The reduced directed cyclic graph to directed acyclic graph.

KL-algorithm to the Problem

The next step requires to partition the task graph. an initial partition of graph into two parts, named S_1 and S_2 is created as shown in Figure 6.

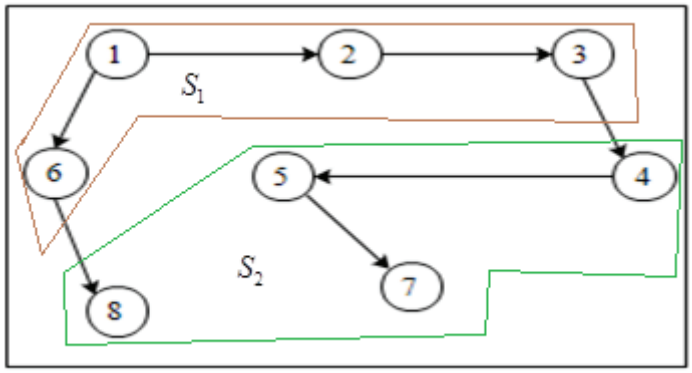


FIGURE 6. An initial partition.

The KL- algorithm is applied to the proposed problem. The algorithm involves swapping a set of nodes, then go back and do the whole thing again, recomputing improvements, counting cuts, and so forth. (This algorithm has several obvious optimizations.) The entire algorithm stops when an iteration produces no improvement. This is typical of iterative-improvement algorithms.

Table 1 lists the cut-count, uncut-count and improvement for each node.

TABLE 1. Node improvements, Step 1.

Node	Cut-Count	Uncut-Count	Improvement
1	0	2	-2
2	0	2	-2
3	1	1	0
4	1	1	0
5	0	2	-2
6	1	1	0
7	0	1	-1
8	1	0	1

The pair improvements for each of pair nodes is shown in Table 2.

TABLE 2. Pair improvements, Step 1.

Pair	Improvement
1,4	-2
1,5	-4
1,7	-3
1,8	-1
2,4	-2
2,5	-4
2,7	-3
2,8	-1
3,4	0
3,5	-2
3,7	-1
3,8	1
6,4	0
6,5	-2
6,7	-1
6,8	1

There are two pairs with maximum improvement in Table 2, (3,8) and (6,8). The total number of cut nets is 2. The pair (3,8) is selected and a tentative swap is made. Table 3 shows the initial state of the list of tentative swaps.

TABLE 3. First tentative swap.

Pair	Cut-count
Zero swap	2
(3,8)	1

The changes in the gate improvements are given in Table 4.

TABLE 4. Node improvements, Step 2.

Node	Cut-Count	Uncut-Count	Improvement
1	0	2	-2
2	1	1	0
4	0	2	-2
5	0	2	-2
6	0	2	-2
7	0	1	-1

The changes in the list of pairs is given in Table 5.

TABLE 5. Pair improvements, Step 2.

Pair	Improvement
1,4	-4
1,5	-4
1,7	-3
2,4	-2
2,5	-2
2,7	-1
6,4	-4
6,5	-4
6,7	-3

At this point, there are no more positive improvements to be made. Some enhanced versions of the KL-algorithm are capable of detecting this condition and stopping at this point. These are not that smart yet, so these will persevere by swapping (2,7), then (1,6) and the last one is (4,5) giving the tentative swap list of Table 6.

TABLE 6. Tentative swap.

Pair	Cut-count
Zero swap	2
(3,8)	1
(2,7)	2
(1,6)	2
(4,5)	2

After running few steps of KL-algorithm, finally the optimal partition of task graphs is obtained. It is shown in Figure 7.

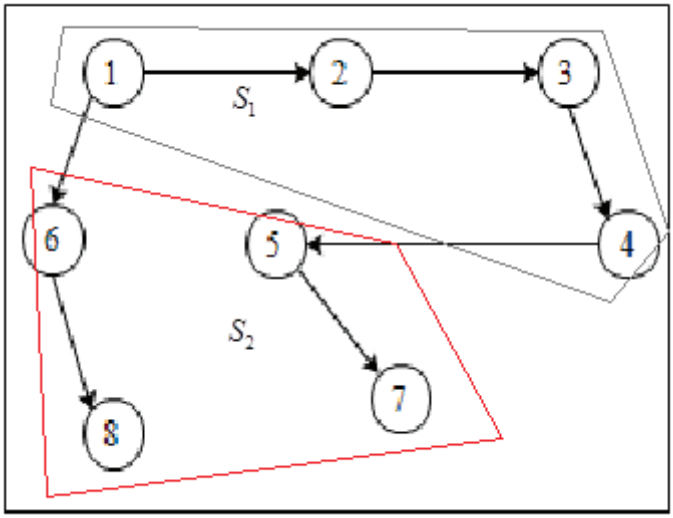


FIGURE 7. Optimal partition of task graph.

Next, the nodes (tasks) are mapped onto two processors as shown in Table 7.

TABLE 7. The schedule length without communication.

Length (Unit)	P_1	P_2
1	Node 1	
2	Node 2	Node 6
3	Node 3	
4	Node 4	Node 5
5		Node 7
6		Node 8

In this problem, the length of the schedule is found to be six units.

CONCLUSION AND FUTURE WORK

Based on the work done in this paper, it can be concluded that the technique of partitioning a directed cyclic graph onto two processors is easy to be implemented. The solution is found initially from the directed cyclic graph. The quality techniques used for finding the acyclic graph from the directed cyclic graph and apply the KL-algorithm to obtain the partitioning of the task graph has led to an efficient graph-mapping concept. The next paper will discuss a larger number of nodes and map it onto *n* number of processors and verify our work through simulation model using programming language.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Education, Malaysia for Research Acculturation Grant Scheme (RAGS) under project code 9018-00036.

REFERENCES

1. Z.Linhong, Wee K.N., J. Cheng, Information Systems **36**, 958-972 (2011).

2. Y. Tian, J.M. Patel, "*Tale: A tool for approximate large graph matching*," in *Proceedings of the 24th International Conference on Data Engineering,* (IEEE Computer Society, Washington, DC, USA, 2008), pp. 963-972.
3. L. Shuli, Y. Weigen, Discrete Applied Mathematics **162**, 415-420 (2013).
4. R. Mohan, A. Gupta, International Journal of Computer Science, Engineering and Applications (IJCSEA) **1(6)**, (2011).
5. Y. Liao, J. Yin, D. Yin, L. Goa, Computer Networks **56**, 2132-2147 (2012).
6. W.N.M. Ariffin, S. Salleh, Contemporary Engineering Sciences **8(17)**, 773 – 788 (2015).