List Scheduling Algorithms for Heterogeneous Systems

Bhavana Kurra  
*Dept. of Computer Science   
University of Colorado-Denver*Denver, United States  
bhavana.kurra@ucdenver.edu

Vijayasimha Bheemireddy  
*Dept. of Computer Science*  
*University of Colorado-Denver*Denver, United States  
vijayasimha.bheemireddy@ucdenver.edu

RohanSomisetty  
 *Dept. of Computer Science*  
 *University of Colorado-Denver*Denver, United States  
 rohan.somisetty@udenver.educ

Rohith Reddy Venna  
*Dept. of Computer Science*   
 *University of Colorado-Denve*rDenver, United States  
rohithreddy.venna@ucdenver.edu

***Abstract*— The performance of a heterogeneous computing system can be improved by the implementation of an efficient application scheduling algorithm. For static application scheduling, list scheduling is a widely used classic algorithm that has low time complexity. Two list-based scheduling algorithms called Predict Earliest Finish Time(PEFT) which is based on optimization cost table(OCT) and pre-scheduling-based list scheduling(PSLS) are implemented. OCT is used to rank tasks and select the processors. Both the algorithms PEFT and PSLS have a time complexity of O(v2.p) for p processors and v tasks while offering makespan improvements. PSLS has pre-scheduling, task prioritizing and processor selection phases. The schedule length ratio and speedup are the metrics of measure for the PEFT and PSLS algorithms which are implemented over graphs.**

***Keywords- Heterogeneous computing system, Application scheduling, DAG scheduling, static scheduling, List scheduling***

# Introduction

A system that contains various types of computational units such as GPUs, multi-core CPUs, DSPs etcetera is called a heterogeneous computing system which is used for the implementation of applications that require a lot of processing power. The general-purpose processors which run an operating system are the computational units of a heterogeneous computing system. A computationally intensive application is divided into a set of tasks and a Directed Acyclic Graph(DAG) which consists of the various properties of an application such as task execution time, task dependencies, and data size to communicate between tasks etcetera is used to represent the application[1]. The problem of task scheduling can be divided into two categories: static scheduling and dynamic scheduling[2]. All information regarding tasks, such as execution and communication costs for each task, as well as their relationships with other tasks, is known ahead of time in the static scheduling; however, such information is not available in the dynamic scheduling, and decisions are determined during runtime. Heuristic-based algorithms and guided random search-based algorithms are the two categories of static scheduling algorithms. List scheduling, cluster scheduling, and duplicate scheduling are the three categories of heuristic-based scheduling algorithms[1].

The goal of this project is to implement and examine the performance of two list scheduling algorithms over the heterogeneous systems Predict Earliest Finish Time(PEFT) and Pre-Scheduling-based List Scheduling(PSLS). Task prioritizing and processor selection are the two main phases of list scheduling algorithms. There is an additional phase called the pre-scheduling phase for the PSLS algorithm. The objective of both algorithms is to minimize the makespan which is the total completion time of an application.

The input for these list scheduling algorithms consists of two data structures. The first one is a Directed Acyclic Graph(DAG) whose vertices denote the set of tasks. The edges between the vertices of the Directed Acyclic Graph(DAG) denote the communication cost between the tasks. The second data structure is a matrix which is called a “computation cost table”. After obtaining the results using both the algorithms, we use Scheduling Length Ratio (SLR) and Speedup as the metric of evaluation and comparison of both algorithms. Makespan is used to compute SLR and Speedup.[1][2]

# Relation to coursework

Over the years, the various types of process scheduling algorithms and their importance has been discussed as a vital part of operating systems. While we always discuss the scheduling algorithms that are there for the homogeneous systems, in this project we are understanding and implementing the scheduling algorithms for heterogeneous systems.

# Motivation

The primary goal of a scheduling algorithm is to reduce resource scarcity and promote fairness among those who use the resources. Cloud computing, grid computing, edge computing, etcetera are the environments where heterogeneous computing can be applied. The demand for the acceleration of data processing has increased with the rise of big data, the Internet of things, and artificial intelligence which increased the requirement for heterogeneous computing. Task scheduling for heterogeneous systems involves more complexity compared to that task scheduling for homogeneous systems. Heuristics for list scheduling frequently produce high-quality schedules at a low cost. They have a lower time complexity than clustering algorithms, and their solutions utilize fewer processors than task duplication strategies, resulting in more efficient schedules. The implementation of task scheduling for a heterogeneous system is a complex problem with low cost using the list scheduling algorithms has motivated us to work on this project.

# Software And Cloud Platform Tools

The software that are used in our project is as follows -

1. Anaconda Navigator
2. Jupyter notebook
3. Google Colab
4. Spark in Google Cloud Platform(GCP)
5. Anaconda Navigator - Anaconda Navigator is a desktop application that is used for launching python programs while providing packages, channels, and environments. It also avoids the necessity for the usage of command-line commands.
6. Jupyter Notebook - Jupyter Notebook is a web application that is used for the creation of code, visualizations, and text.
7. Google Colab - Google Colab is a version of the jupyter notebook which is provided by Google and runs on the cloud. It is used for the creation of python programs while allowing developers to share notebooks and to use free cloud services.
8. Spark in Google Cloud Platform(GCP) - Google Cloud Platform's Spark is used for the analysis and processing of large-scale data. It also allows users to build machine learning models over large-scale data.

# Implementation Design

The implementation design of our project is shown in Figure 5.1. The following are three main steps in our implementation design

1. Data Generation.
2. Implementation of PEFT and PSLS Algorithms.
3. Evaluation and Analysis of Results.

Initially, we generated inputs by the data generation code that we have written. Later, we implemented both the PEFT and PSLS algorithms. After the implementation of the algorithms, we have run these algorithms over the data generated. As a part of our last phase, we have noted, analyzed, and evaluated the results that are obtained.

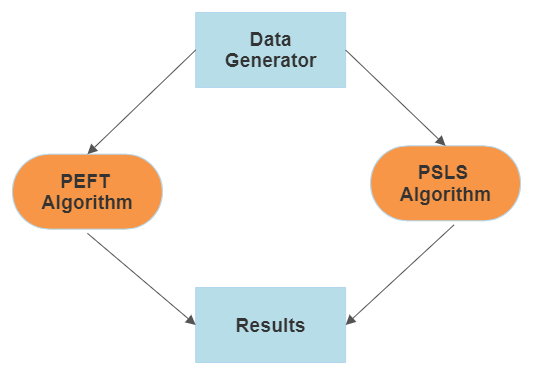


Figure 5.1

# Methodology

1. **Data generation -**

Both the list scheduling algorithms PEFT and PSLS have Directed Acyclic Graph(DAG)and Computation time matrix as the inputs.The number of tasks, number of processors, alpha, beta, and Communication to Computation Ratio(CCR) are the inputs for the generation of data. Alpha is used for the generation of Directed Acyclic Graphs whereas both the beta and CCR are used for the generation of computation cost tables. Beta is a heterogeneity parameter. The definitions for alpha and CCR are as follows -

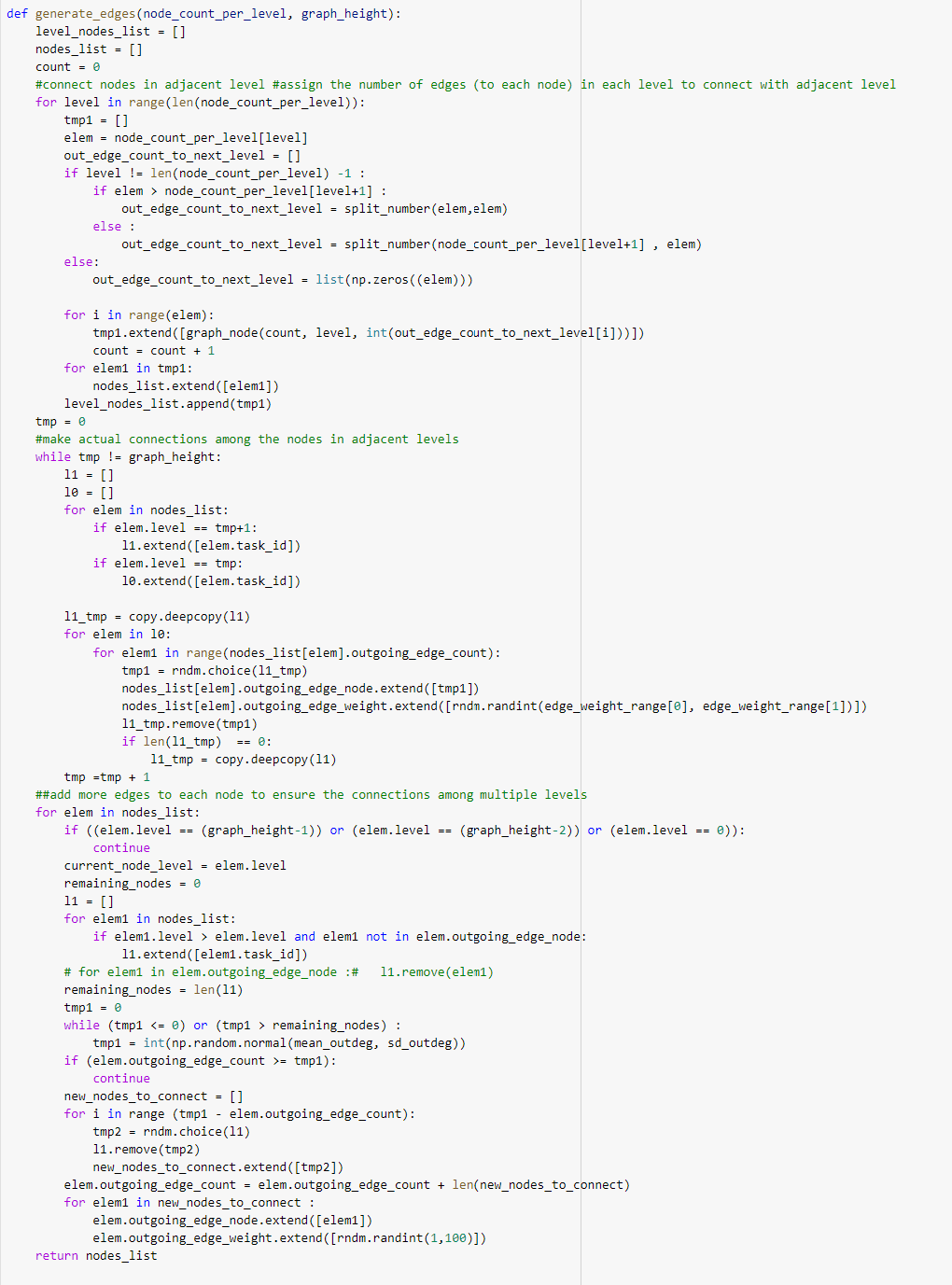
* Alpha - It is a parameter that determines the shape of a DAG by affecting the height and width of a DAG. The width of the graph is the maximum number of tasks that can be executed concurrently. The higher value of alpha results in higher-level task parallelism resulting in a fat DAG.
* Communication to Computation Ratio(CCR) - It is the average communication cost divided by the average computation cost.

For a given CCR value, beta, and task T, we calculate the average communication cost from that task to its successors using the DAG. Later we compute the average computation cost for task T. Based on the average computation cost of the task and beta values we randomly generate a value that will be used for the creation of the computation cost table. Networkx library of python is used for the representation of graphs[3][4][5]. Approximately 28000 Directed Acyclic Graph(DAG) and computation cost tables are generated.

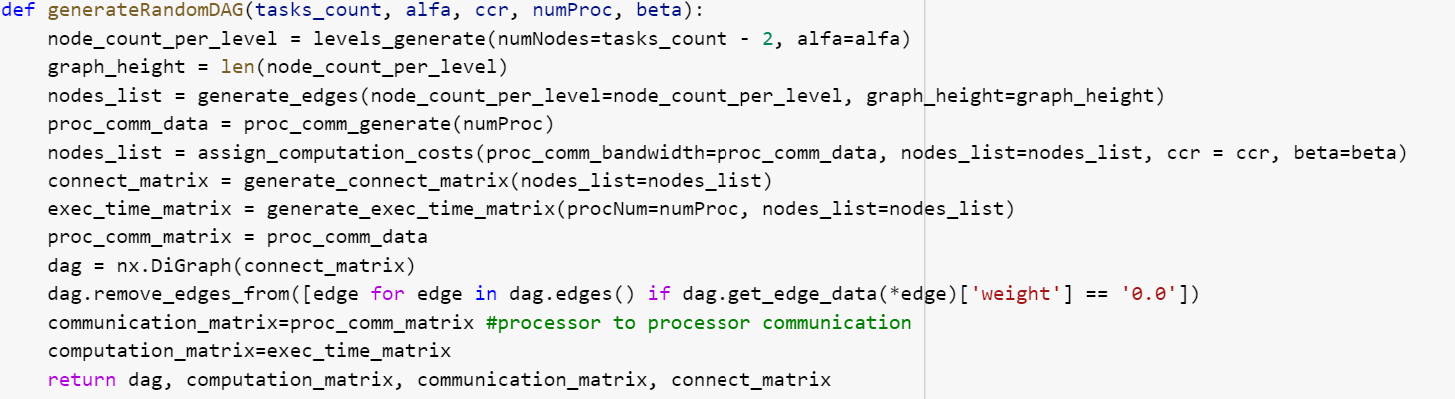
The following is the code that is written for the generation of data -



Data Generation CodeScreenshot 1



Data Generation CodeScreenshot 2



Data Generation CodeScreenshot 3

1. **Predict Earliest Finish Time(PEFT) -**

PEFT algorithm has two phases - task prioritizing and processor selection phase.

* As a part of the task prioritizing phase we calculate Optimistic Cost Table(OCT). The formula for OCT is as mentioned below-

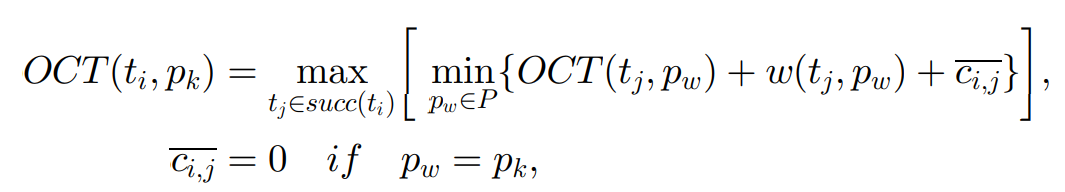
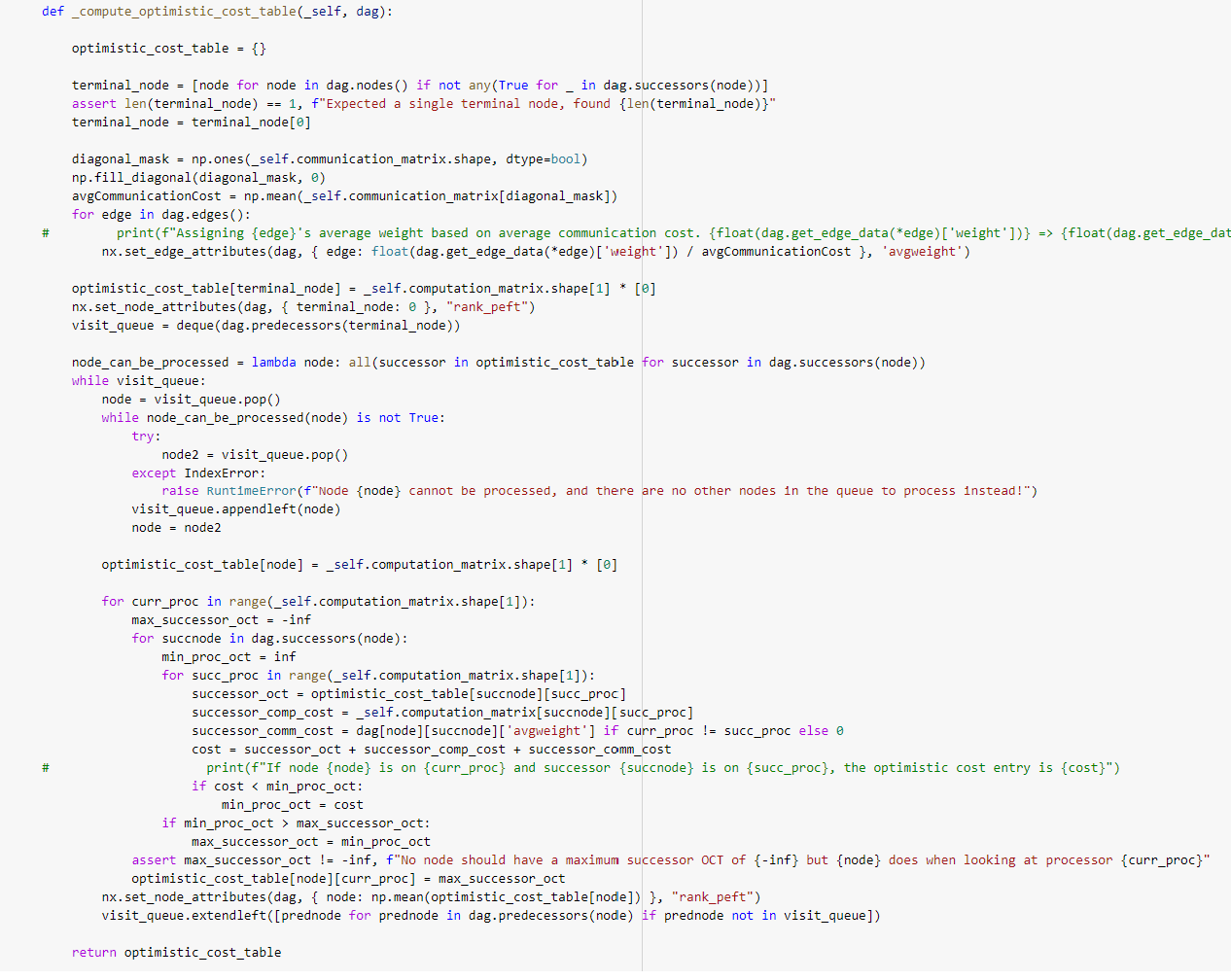


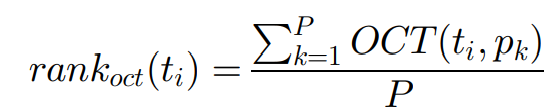
Figure 6.1[1]

Here, succ(ti) denotes the set of immediate successors for task ti. A task with no successors is called an exit task and denoted by nexit. ci,j is the communication cost between the task i and task j. ci,j is zero when both the i and j tasks are being executed on the same processor and has a non-zero value otherwise. ci,j can be obtained by looking at the given weighted Directed Acyclic Graph(DAG). The code for the implementation of OCT is below:



PEFT CodeScreenshot 1

* Later we calculate rankOCT as a part of the task prioritization phase. RankOCT for a task “ti” is the average Optimistic Cost Table of that task for all the processors. The formula for the rankOCT is as below:

 Figure 6.2[1]

* Based on the rankOCT we create a queue of tasks. The task with the highest rankOCT will be given more priority.
* We calculate Earliest Finish Time(EFT) as the first step of our processor selection phase. EFT is the sum of Earliest start time(which is defined in the PSLS algorithm) and the computation cost of task ni on processor pj. The formula for EFT is as defined below:

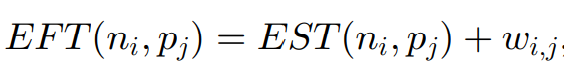


Figure 6.3[1]

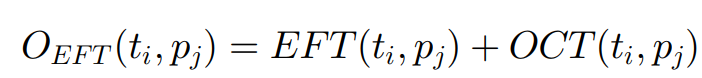
* As the second step of our processor selection phase we calculate the Optimistic Earliest Finish Time(OEFT). OEFT for a ti task on processor pj is the sum of the EFT for task ti on processor pj and OCT of task ti on processor pj.The formula for Optimistic Earliest Finish Time(OEFT) is as defined below:

Figure 6.4[1]

* For a given Task t, we choose the processor with the lowest OEFT value.
* The code for the implementation of PEFT algorithm is as below:



PEFT CodeScreenshot 2



PEFT CodeScreenshot 3

1. **Pre-Scheduling-based List Scheduling(PSLS) -**

PSLS algorithm has three phases -

1. Pre-scheduling phase
2. Task prioritizing phase
3. Processor selection phase

As a part of Pre-scheduling phase the following parameters are derived and used-

1. Processor chosen for a task using the PEFT algorithm is denoted using map1
2. Actual start Time of a task derived using PEFT algorithm is denoted as AST1.
3. Makespan derived using PEFT algorithm is denoted as makespan1.

* As a part of task prioritizing phase we calculate Downward Length Table(DLT). DLT is a n×m matrix where n denotes the number of tasks and m denotes the number of processors. It is calculated in the task prioritizing phase. DLT is calculated based on the total makespan and the Actual Start Time of task nj which are calculated as a part of the pre-scheduling phase. The formula for the DLT is as shown below -

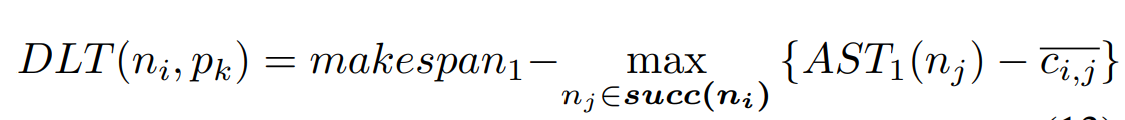


Figure 6.5[2]

The code for the implementation of DLT is below:



PSLS CodeScreenshot 1

* Later we calculate rankDLT as a part of the task prioritizing phase. RankDLT for a task “ni” is the average Down Length Table(DLT) of that task for all the processors. The formula for the rankDLT is as below:

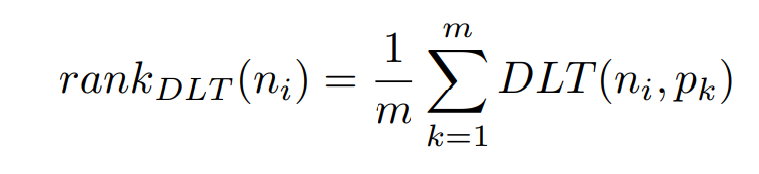


Figure 6.6[2]

* Based on the rankDLT we create a queue of tasks. The task with highest rankDLT will be given more priority.
* As a part of the processor section phase we calculate Earliest Finish Time(EFT). Earliest Finish Time for a given task ni on the processor pk is calculated as the sum of Earliest Start Time for a task ni on processor pk with the computational cost of task ni on processor pk. The formula for the EFT is as below:



Figure 6.7[2]

* As a second step of the processor selection phase we calculate Optimization Objective Function(OPT). OPT for a given task ni on the processor pk is calculated as the using the below mentioned formula:

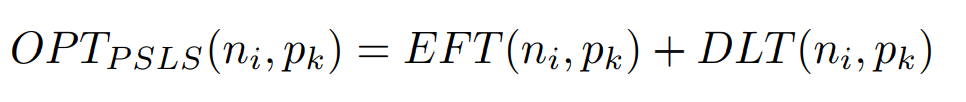


Figure 6.8[2]

* For a given Task t, we choose the processor with the lowest OPT value. The code for the implementation of PSLS algorithm is as below:



PSLS CodeScreenshot 2

* The main function is the crucial function where we call all the functions such as random DAG generator function, peft algorithm function, psls algorithm function etcetera. The code for our main function is as follows:



Main Function CodeScreenshot 1

* The following screenshot shows us the range of values that we have taken for the number of tasks, number of processors, alpha, beta and Communication to Computation Ratio(CCR) in order to generate random Directed Acyclic Graphs(DAG) and how we are written the results into a file using the data frame data structure available in pandas library of python.



Main Function CodeScreenshot 2

# How to run the code

The code is written in python and is saved as an ipynb file. For the creation of code we used the jupyter notebook service provided by anaconda navigator, google colab and jupyter notebook web application. We can use any of these tools in order to run the code. To run the code numpy, pandas, matplotlib, networkx libraries should be installed in the system using the pip package-management system of python.

# Measurements

Makespan is the most common metric used for the evaluation of a single DAG. In order to evaluate our list scheduling algorithms, we have chosen Scheduling length ratio and speedup as the metrics to compare the performance of the algorithms[1][2].

For the calculation of Scheduling Length Ratio and speedup we require two parameters called makespan and critical path. The definition of makespan and critical path is as follows -

* Makespan **-** It is also called as schedule length which is the maximum of Actual Finish Time of all the exit tasks(nexit), where the task which doesn’t have any successors is called as nexit. The formula for makespan is as follows:



Figure 8.3[1]

* Critical Path - The longest path from the entry tasks to the exit tasks in a Directed Acyclic Graph(DAG) is called Critical Path.

The definition of Scheduling Length Ratio(SLR) and speedup is as follows -

* Scheduling Length Ratio - Scheduling length Ratio(SLR) is also called normalized schedule length(NSL). Makespan and minimum computation cost of the critical path tasks(CPMIN) are used for calculating the Scheduled Length Ratio.[1] Lower Scheduling Length Ratio value means a better performance of scheduling algorithms. The formula used for the calculation of the scheduled length ratio is as shown in the figure below:

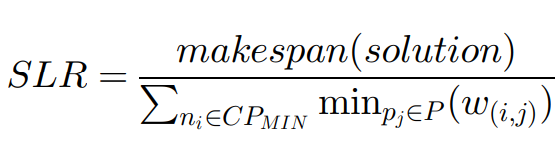


Figure 8.1[2]

Here, ni indicates the ith task, pj indicates the jth processor, w(i,j) indicates the computation cost of ni on pj.

* Speedup - For the calculation of speed up, makespan and minimum of computation time taken for all the tasks to run over a processor are used. Higher speedup value means a better performance of scheduling algorithms. The formula used for the calculation of the speedup is as shown in the figure below:

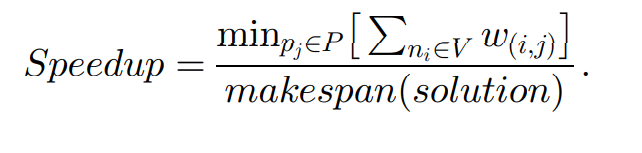


Figure 8.2[1]

Here, ni indicates the ith task, pj indicates the jth processor, w(i,j) indicates the computation cost of ni on pj.

# Evaluation Methodology

Over the DAG and computation cost table simulations that we generated, we run both the PEFT and PSLS algorithms. The metrics that are mentioned in section VIII are used for evaluation. We have noted down the makespan, scheduling length ratio and speed up resulted by both the algorithms over different input configurations which we later used for the comparison and evaluation of results.

# Results and Analysis

We have run both the Predict Earliest Finish Time(PEFT) and Pre-Scheduling List Scheduling(PSLS) algorithms over the test scenario which has the below mentioned Directed Acyclic Graph(Figure 9.1) and Computation cost table(Figure 9.2) as inputs. There are ten tasks and three processors in our test scenario.

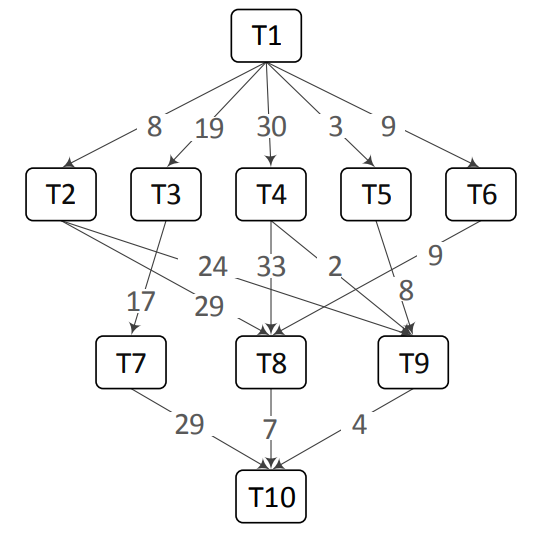


Figure 9.1[2]

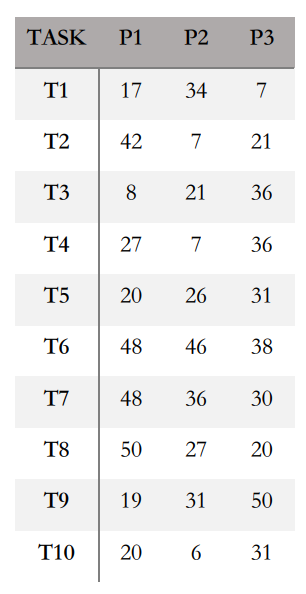


Figure 9.2[2]

The results of PEFT and PSLS when run over the above mentioned test scenario are shown in Figure 9.3 and Figure 9.4. Both the figures are gantt charts which helps us in interpreting which processor is assigned for a given task. We also can interpret start and end times of each task using these gantt charts. From Figure 9.3 we can say that the total time taken for execution of all the tasks or makespan when using PEFT algorithm is 120. From Figure 9.4 we can say that the total time taken for execution of all the tasks or makespan when using the PSLS algorithm is 114.

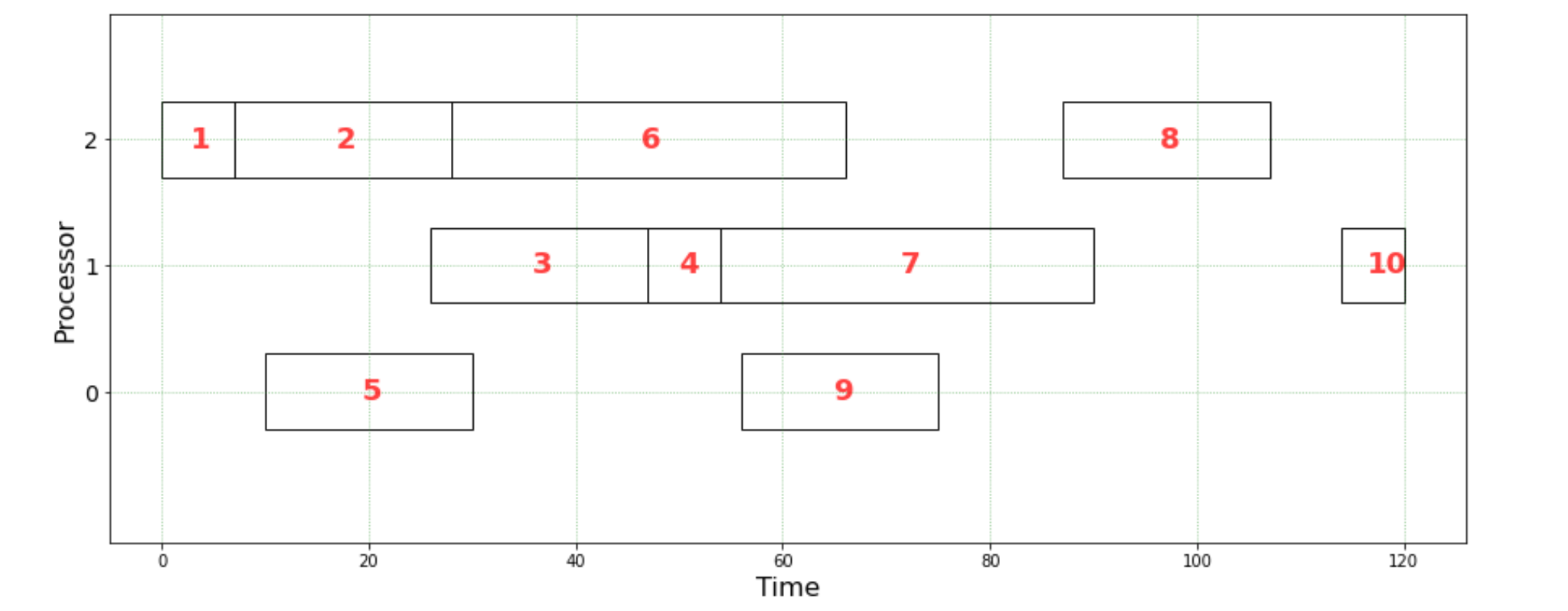


Figure 9.3

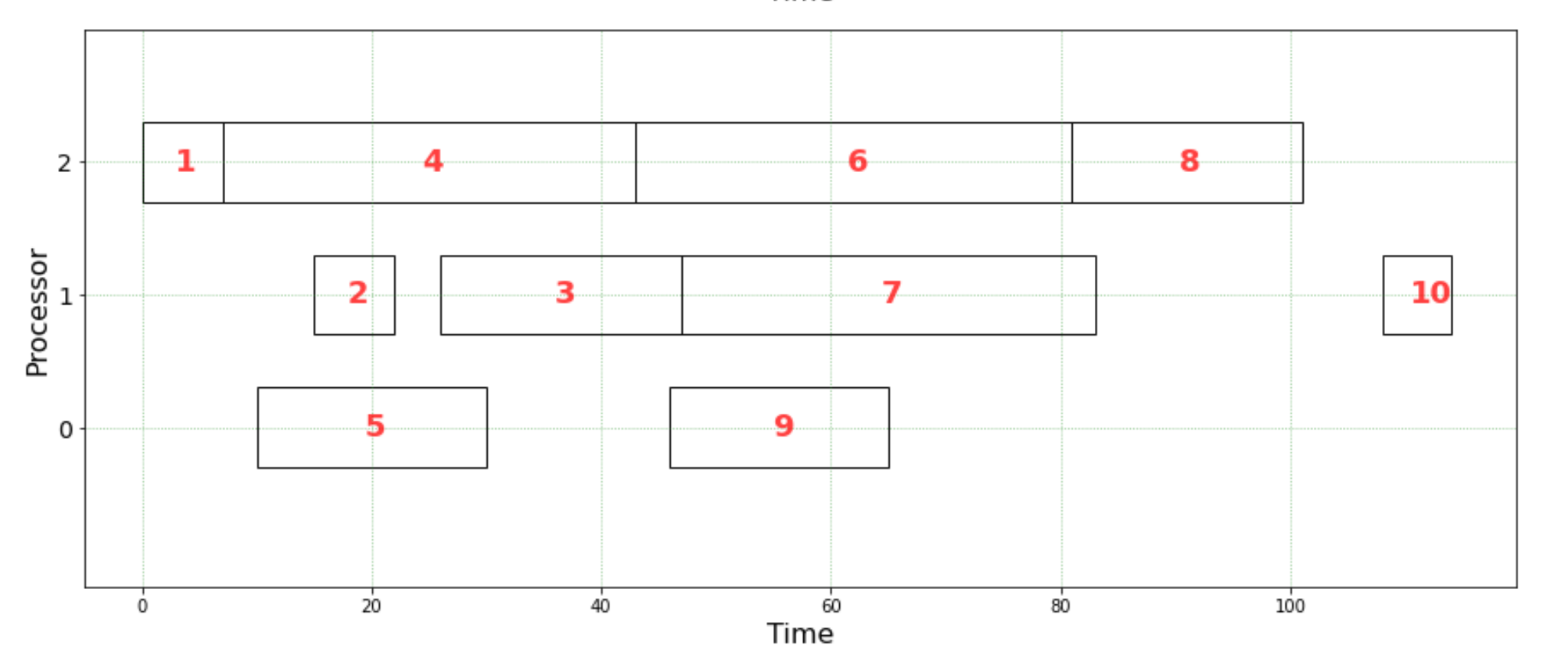


Figure 9.4

Over approximately 28000 simulations that are generated using the DAG generation code, we have run both the PEFT and PSLS algorithms while noting the performance of both the algorithms. The performance of PEFT and PSLS algorithms as the number of tasks increases and the Scheduling Length ratio(SLR) evaluation metric is considered are as shown in Figure 9.5.

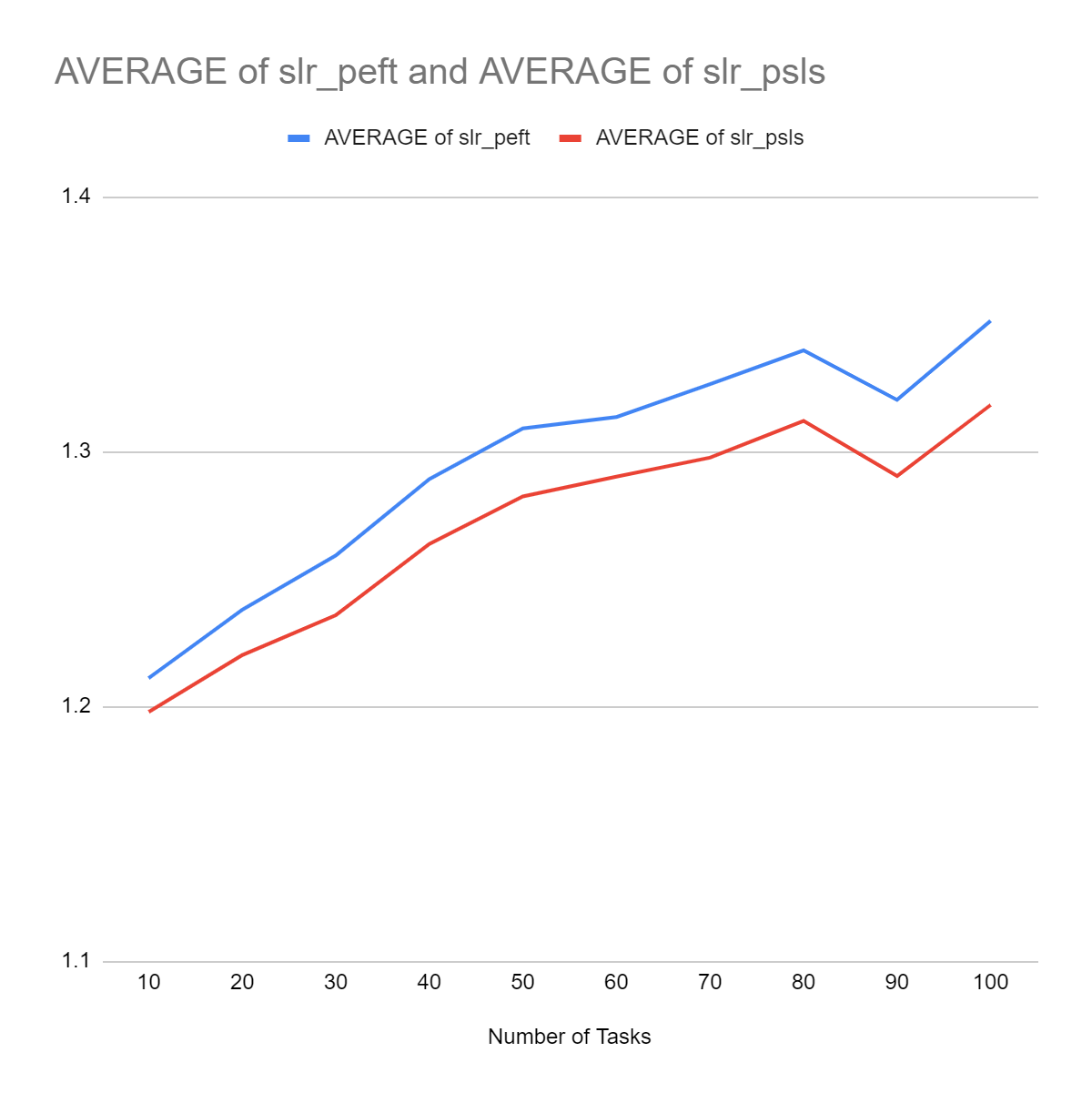


Figure 9.5

The performance of PEFT and PSLS algorithms as the number of processors increases and the Scheduling Length ratio(SLR) evaluation metric is considered are as shown in Figure 9.6.

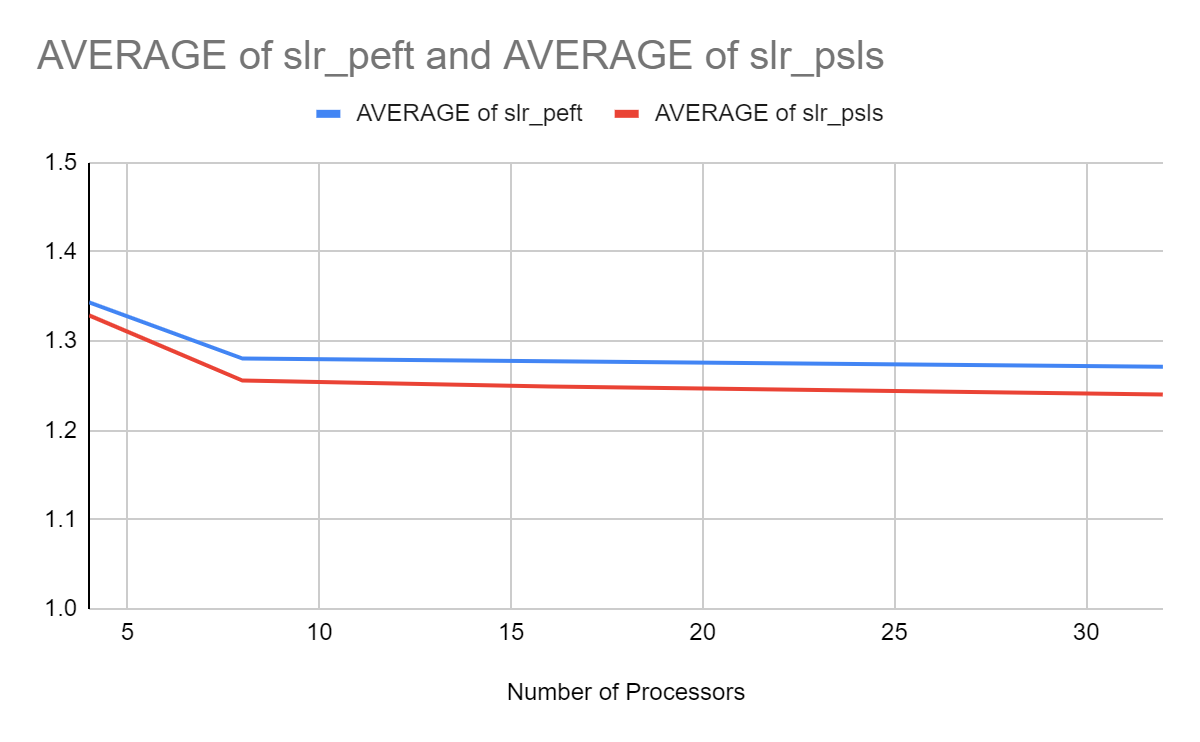


Figure 9.6

The performance of PEFT and PSLS algorithms as the number of tasks increases and the speedup evaluation metric is considered are as shown in Figure 9.7.

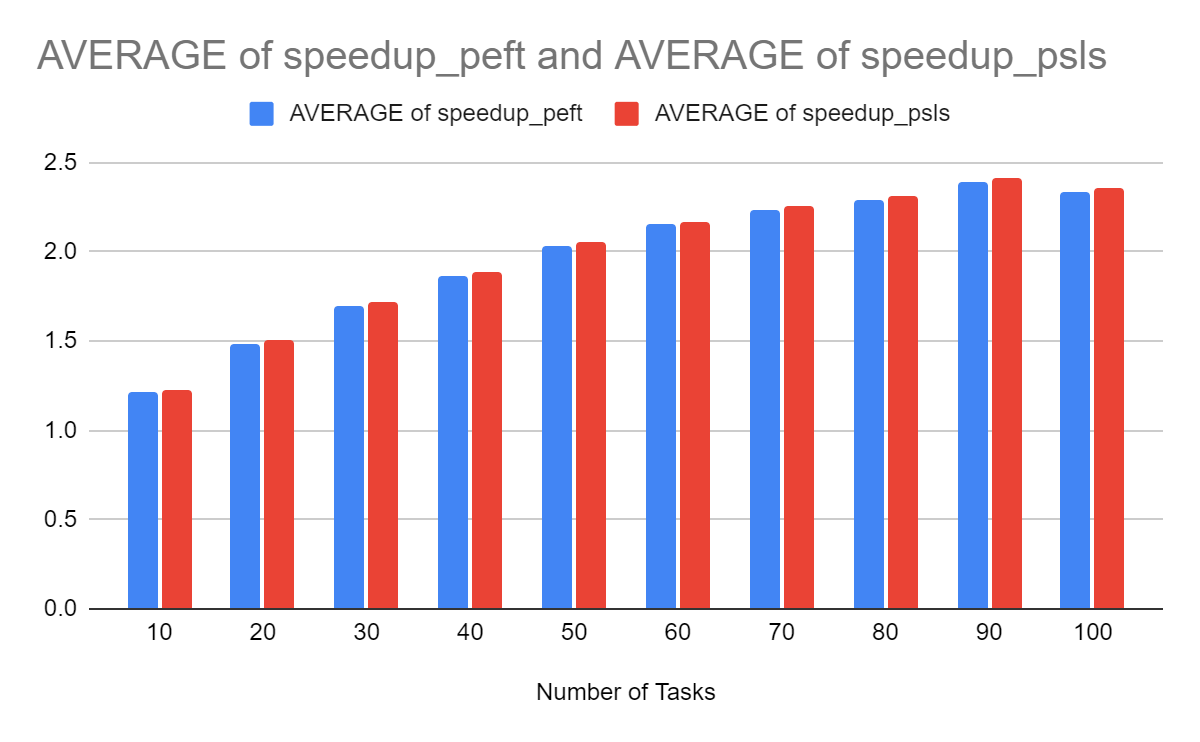


Figure 9.7

The performance of PEFT and PSLS algorithms as the number of processors increases and the speedup evaluation metric is considered are as shown in Figure 9.8.

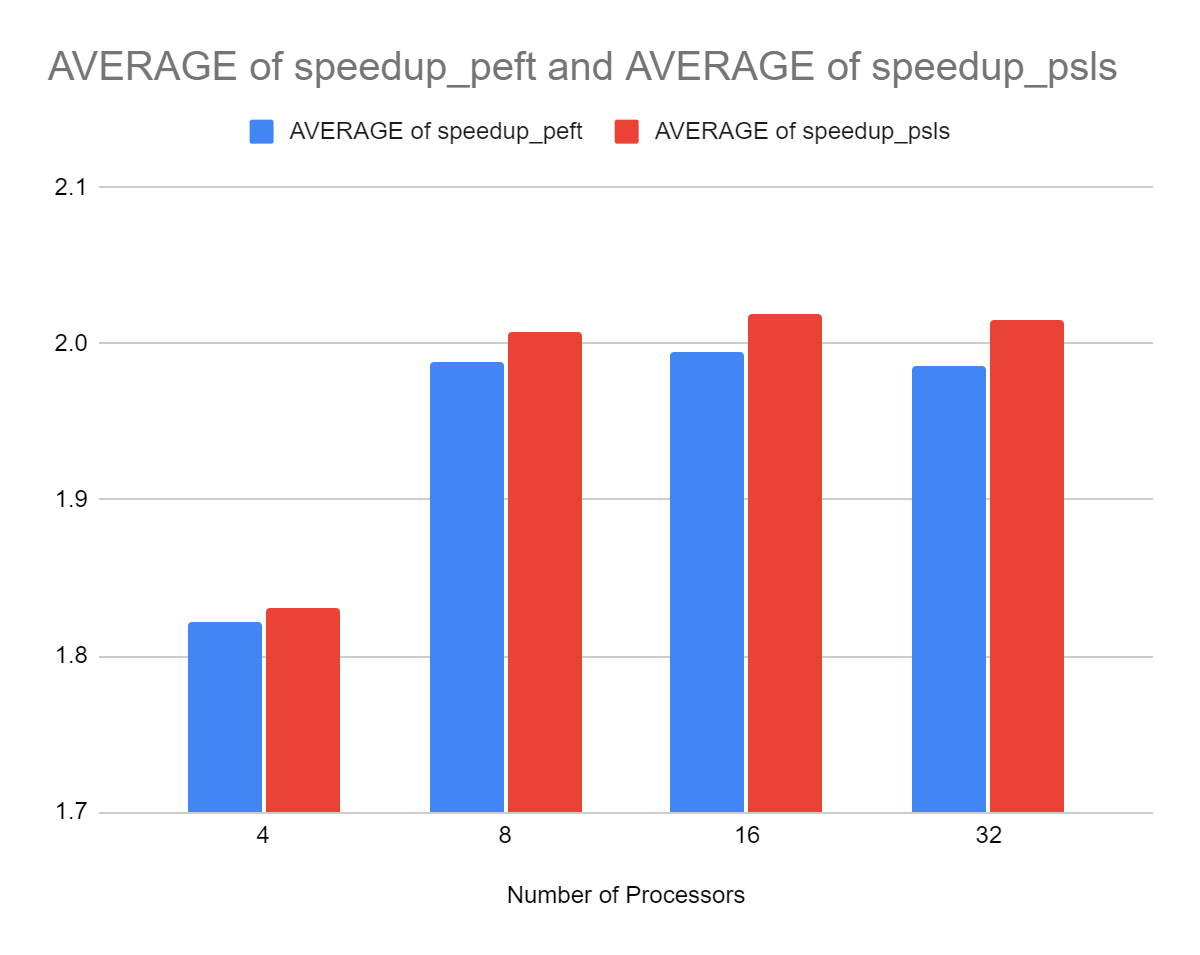


Figure 9.8

# Conclusion

Pre-Scheduling List Scheduling(PSLS) has performed better than Predict Earliest Finish Time(PEFT) when run over the test scenario that we have discussed at the beginning of section VIII. PSLS resulted in a makespan of 114 over the test scenario which is less than the makespan generated by PEFT. From observing Figure 9.5 and Figure 9.6 we can say that as the number of processors or number of tasks increases, the Scheduling Length Ratio(SLR) of the PSLS algorithm is less compared to that of the SLR of PEFT which is desirable. From observing Figure 9.7 and Figure 9.8 we can say that as the number of processors or number of tasks increases, the speedup of the PSLS algorithm is greater compared to that of the speedup of the PEFT algorithm which is desirable. Therefore, we can say that the PSLS algorithm performs better than the PEFT algorithm.

The hypothesis outcome based on the research papers that we have referred for implementation[1][2] is that the Pre-Scheduling List Scheduling(PSLS) algorithm should perform better than the Predict Earliest Finish Time(PEFT) having an additional implementation scheduling phase called the Pre-scheduling phase. We are successful in deriving the results that are similar to that of the hypothesis outcome.

# Future work

We have implemented both the Pre-Scheduling List Scheduling(PSLS) and Predict Earliest Finish Time(PEFT) algorithms over the Directed Acyclic Graphs(DAG) that we have generated. Further, we can implement these algorithms over real-world graph applications such as gaussian elimination, fast fourier transform montage workflow, epigenomic workflow etcetera.

##### References

1. H. Arabnejad and J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682-694, March 2014, doi: 10.1109/TPDS.2013.57.
2. H. Arabnejad and J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp. 682-694, March 2014, doi: 10.1109/TPDS.2013.57.
3. <https://mungingdata.com/python/dag-directed-acyclic-graph-networkx/>
4. <https://anush-venkatakrishna.medium.com/directed-acyclic-graph-dag-2baeb325b883>
5. <https://ipython.org/ipython-doc/3/parallel/dag_dependencies.html>