Implementing Different CPU Scheduling Algorithms in Parallel Programming Structure.

Parallelizing CPU Scheduling Algorithms

A PROJECT REPORT

Submitted by

ROHAN BARSAIYA	**18BCE2206**
PRANAV KHURANA	**18BCE2513**
SHASHANK SHUKLA	**18BCE2522**
MIHIR AGARWAL	**18BCE2526**

Submitted to:
PROF. PREETHA

CSE4001 PARALLEL DISTRIBUTED AND COMPUTING
Slot – C1

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Computer Science and Engineering
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT®

Vellore Institute of Technology

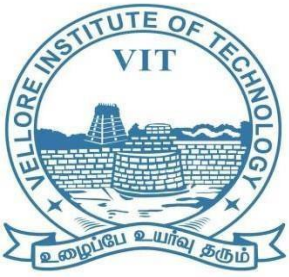
(Deemed to be University under section 3 of UGC Act, 1956)

We hereby declare that the project entitled **"Implementing Different CPU Scheduling Algorithms in Parallel Programming Structure"** submitted by us to the School of Computer Science and Engineering, VIT University, Vellore in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a record of bonafide work carried out by us under the supervision of **PROF. PREETHA**

We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

SIGNATURE:

1. **ROHAN BARSAIYA -18BCE2206**
2. **PRANAV KHURANA -18BCE2513**
3. **SHASHANK SHUKLA -18BCE2522**
4. **MIHIR AGARWAL -18BCE2526**



VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

The project report entitled **"Implementing Different CPU Scheduling Algorithms in Parallel Programming Structure"** is prepared and submitted by our group members. It has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering in VIT University, India.

Guide
PROF. PREETHA

ACKNOWLEDGEMENT

We would like to express my gratitude to all those who have helped me in the successful completion of this project. Without their support, we would not have been able to achieve the goal of the project successfully.

We would like to take this opportunity to thank my guide**, PROF. PREETHA**, for her constant support, guidance and mentorship without which it would have been really difficult to complete the project on time.

We would like to thank our Dean, Dr. SARAVANAN R., who provided us with the facilities required and conductive conditions for the project.

Finally, we would like to express my sincere gratitude to VIT University, which provided me with a platform to hone my skills over a period of four years.

CONTENTS

AIM/OBJECTIVE

To prove the performance improvement for parallelizable serial codes with the help of parallelizing various serial scheduling algorithms and comparing them.

ABSTRACT

Task scheduling in parallel processing is a technique in which processes are assigned to different processors. Task scheduling in parallel processing uses different types of algorithms and techniques which are used to reduce the number of delayed jobs. Nowadays there is a different kind of scheduling algorithms and techniques used to reduce the execution time of tasks scheduling of jobs in parallel is becoming the subject of much research. The problem of job scheduling is to determine how resources should be shared in order to maximize the system's utility. This problem has been extensively studied for well over a decade. In this paper, we will first discuss the execution time of scheduling algorithm (FIFO, ROUND ROBIN, SJF, PRIORITY SCHEDULING) in series and parallel, we discuss how deployed scheduling policies can be improved to meet existing requirements, specific research challenges and future scope.

INTRODUCTION

The Central Processing Unit (CPU) is the brain of the computer system so it should be utilized efficiently. CPU Scheduling is one of the most important concepts of Operating System. Sharing of computer resources among multiple processes is called scheduling.

The various CPU scheduling algorithms are: -

1. FCFS CPU scheduling: - In this scheduling the process that requests the CPU first is allocated to CPU first. This algorithm is easy to understand and implement but poor in performance as average wait time is very high.

2. SJF CPU scheduling: - In this scheduling the process with the shortest CPU burst time is allocated to CPU first. It is the best way to reduce time. Here the actual time taken by the process to execute is already known by the processor

3. Priority scheduling: - In this scheduling the process with high priority is allocated to CPU first, and process with the same priority are executed in FCFS manner. Priority can be decided on the basis of the memory allocation, or on the basis of the time or on the basis of some resource requirements.

4. Round Robin scheduling: - It is same as FCFS scheduling with pre-emption is added to switch between processes. A static Time Quantum (TQ) is used in this CPU Scheduling. Once a process is executed for given time period that process is preemptied and other process executes for given time period. A process known as Context switching is used to save states of preemptied processes while execution.

The various scheduling parameters for the selection of the scheduling algorithm are :

1. Context Switch: A context switch is process of storing and restoring context (state) of a pre-empted process, so that execution can be resumed from same point at a later time. Context switching is wastage of time and memory that leads to the

increase in the overhead of scheduler, so the goal of CPU scheduling algorithms is to optimize only these switches.

2. Throughput: Throughput is defined as number of processes completed in a period of time. Throughput is less in round robin scheduling. Throughput and context switching are inversely proportional to each other.

3. CPU Utilization: It is defined as the fraction of time CPU is in use. Usually, the maximize the CPU utilization is the goal of the CPU scheduling

4. Turnaround Time: Turnaround time is defined as the total time which is spend to complete the how long it takes the time to execute that process.

5. Waiting Time: Waiting time is defined as the total time a process has been waiting in ready queue.

6. Response Time: Respond Time is better measure than turnaround time. Response time is defined as the time used by the system to respond to the any particular process. Thus the response time should be as low as possible for the best scheduling.

7. Load average: It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

The various characteristics of good scheduling algorithm are [3]:

1. Minimum context switches.

2. Maximum CPU utilization.

3. Maximum throughput.

4. Minimum turnaround time.

5. Minimum waiting time

ADVANTAGES AND DISADVANTAGES OF THE ALGORITHMS

1. First Come First Serve (FCFS):

Advantages	Disadvantages
This algorithm puts the process requests in a Queue and executes it one by one.	There is no option for pre-emption of a process. If a process is started, then CPU executes the process until it ends.
simple and easy to implement	Because there is no pre-emption, if a process executes for a long time, then other process needs to wait a bit long for execution.
Every process will get a chance to run, so starvation doesn't occur.	

2. Shortest Job First (SJF):

Advantages	Disadvantages
executed first process and then followed by longer processes.	The time taken by a process must be known by the CPU beforehand, which is not possible.
more processes can be executed in less amount of time.	Longer processes will have more waiting time, eventually they'll suffer starvation.

3. Round Robin (RR):

Advantages:	Disadvantages:
Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.	The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS.
Starvation doesn't occur	If time quantum is shorter than needed, than this can lead to decrease in CPU efficiency.
No process is left behind.	

4. Priority based Scheduling:

Advantages:	Disadvantages:
be selected based on memory requirement, time or user preference.	A second scheduling algorithm is required to schedule the processes which have same priority.
For example, a high end game will have better graphics,that means the process which updates the screen in a game will have higher priority so as to achieve better graphics performance.	In preemptive priority scheduling, a higher priority process can execute ahead of an already executing lower priority process.

LITRATURE SURVEY

[1]This paper is about scheduling jobs on distributed memory massively parallel processors.

It talks about various Scheduling models used in study of scheduling algorithms which can be classified according to the following criteria:

Partition Specification,Job Flexibility,Level of Pre-emption supported,Amount of job and workload knowledge available ,Memory Allocation

[2] This paper compares various scheduling algorithms such as-

1. FCFS
2. SJF pre-emptive
3. SJF, Non pre-emptive
4. Round Robin
5. LJF
6. Priority Scheduling

And mentions the advantages and disadvantages with respect to various parameters like implementation complexity, starvation, pre-emption. [3] This paper discusses majorly two topics:

To present some of the major technological changes and to discuss the additional dimensions they add to the set of JSSPP challenges.

And to promote and suggest research topics inspired by these dimensions in the JSSPP community.

(JSSPP stands for Job Scheduling strategies for Parallel Processing).[4]This paper mainly emphasizes on the comparison criteria of different CPU Scheduling algorithms.[5]This paper proposes a simple on-line algorithm employing job pre-emption without migration and derives theoretical limits for the performance of the algorithm. The algorithm is experimentally evaluated with data from a large

computing facility.

The treatment of shortest process in SJF scheduling tends to result in increased waiting time for long processes.[6]In this paper, various scheduling techniques for parallel computing are discussed like longest processing time, list scheduling and approximation techniques like heuristic algorithms and some other techniques like load balancing and thread scheduling.

[7]This paper introduces a scheduling algorithm TMDC (Task Mapping Based On Dependence And communication) for assigning tasks to processors and minimizes the communications volume to optimize the total execution.

Problem Analysis Chart (PAC)

Data	Processing	Output	Solution Alternatives
N processes to be assigned to M processors in the CPU	CPU scheduling algorithms like SRTF,SJF, FCFS,RR etc	Execution time, waiting time	Efficient parallelization for lowered execution time.

CODE
SERIAL CODE

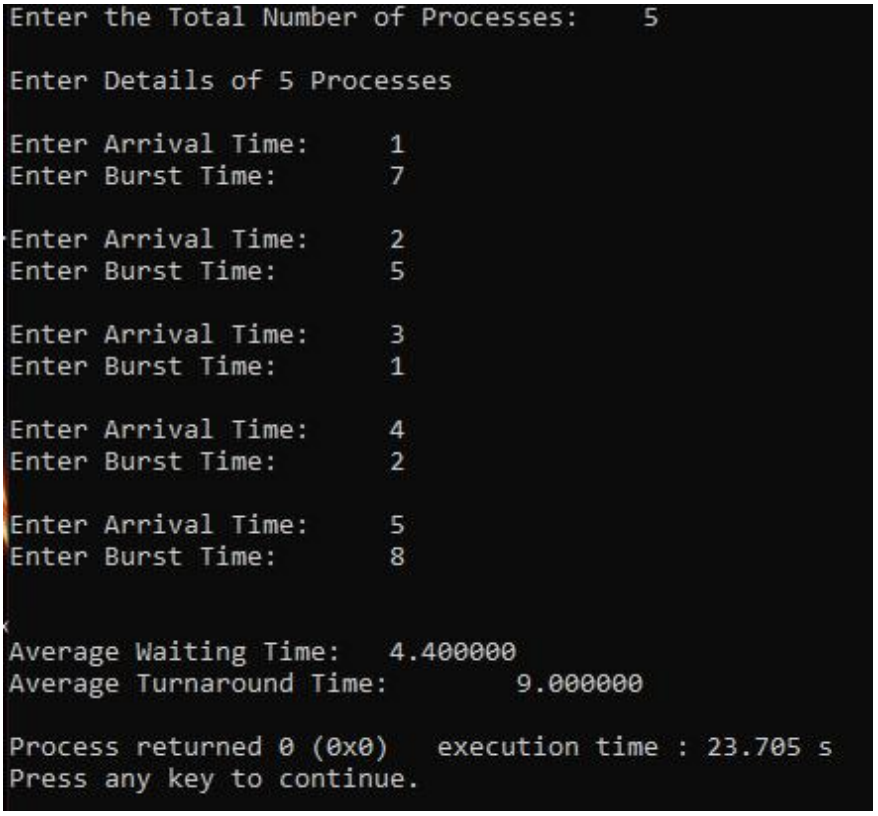
1. SJF serial

CODE:

```
#include <stdio.h>

int main()
{
    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    printf("\nEnter the Total Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Details of %d Processes\n", limit);
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] &&
burst_time[i] > 0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
```

```
            turnaround_time = turnaround_time + end - arrival_time[smallest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
    printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
    return 0;
}
```

OUTPUT:



2. LJF serial

CODE:

```
#include <stdio.h>

int main()
{
    int arrival_time[10], burst_time[10], temp[10];
    int i, biggest, count = 0, time, limit, Limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time=0, average_turnaround_time=0;
    printf("\nEnter the Total Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Details of %d Processes\n", limit);
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }

    burst_time[9] = -1;
    for(time = 0; count != limit; time++)
    {
        biggest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] > burst_time[biggest] &&
burst_time[i] > 0)
            {
                biggest = i;
            }
        }
        burst_time[biggest]--;
        if(burst_time[biggest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[biggest] - temp[biggest];
```

```
            turnaround_time = turnaround_time + end - arrival_time[biggest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
    printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
    return 0;
}
```

OUTPUT:

```
Enter the Total Number of Processes:    5

Enter Details of 5 Processes

Enter Arrival Time:     1
Enter Burst Time:       7

Enter Arrival Time:     2
Enter Burst Time:       5

Enter Arrival Time:     3
Enter Burst Time:       1

Enter Arrival Time:     4
Enter Burst Time:       2

Enter Arrival Time:     5
Enter Burst Time:       8


Average Waiting Time:   14.400000
Average Turnaround Time:        19.000000

Process returned 0 (0x0)   execution time : 15.951 s
Press any key to continue.
```

3. Priority scheduling algorithm

CODE:

```
#include<stdio.h>
 int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);
   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;          //contains process number
   }

   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }
```

```
    temp=pr[i];
    pr[i]=pr[pos];
    pr[pos]=temp;
    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;    //waiting time for first process is zero
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
 avg_wt=total/n;      //average waiting time
total=0;

printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];     //calculate turnaround time
    total+=tat[i];
    printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
```

```
    }
    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);
    return 0;
}
```

OUTPUT:

```
Enter Burst Time and Priority

P[1]
Burst Time:7
Priority:1

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:1
Priority:3

P[4]
Burst Time:3
Priority:4

P[5]
Burst Time:8
Priority:5

Process     Burst Time          Waiting Time    Turnaround Time
P[1]              7                 0                 7
P[2]              2                 7                 9
P[3]              1                 9                 10
P[4]              3                 10                13
P[5]              8                 13                21

Average Waiting Time=7
Average Turnaround Time=12

Process returned 0 (0x0)   execution time : 76.479 s
Press any key to continue.
```

PARALLEL CODE

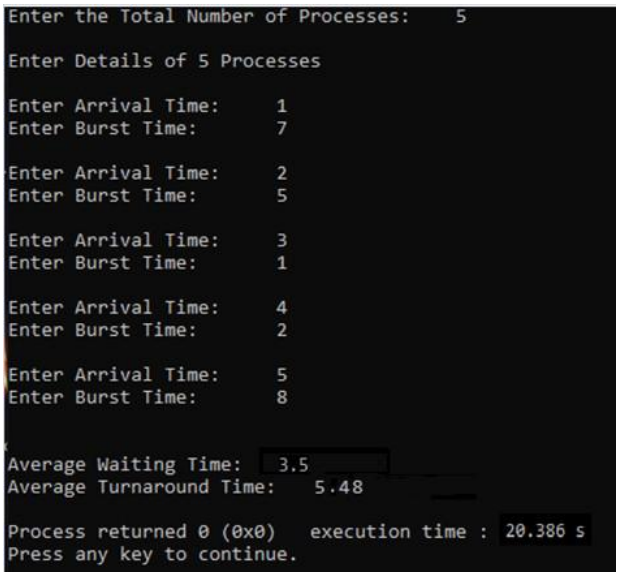
1.SJF parallel

CODE:

```
#include <stdio.h>
#include<omp.h>
int main()
{
   double arr[10];
   omp_set_num_threads(4);
   double min_val=9999;

    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    printf("\nEnter the Total Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Details of %d Processes\n", limit);
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;

   #pragma omp parallel for shared(i) reduction(min: min_val)
   for( i=0; i<limit ; i++)
   {
      if(burst_time[i] < min_val && arrival_time[i] <= time)
      {

         smallest=i;
      }
   }

        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
```

```
                count++;
                end = time + 1;
                wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                turnaround_time = turnaround_time + end - arrival_time[smallest];
            }
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
        printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
        return 0;
    }
```

OUTPUT:



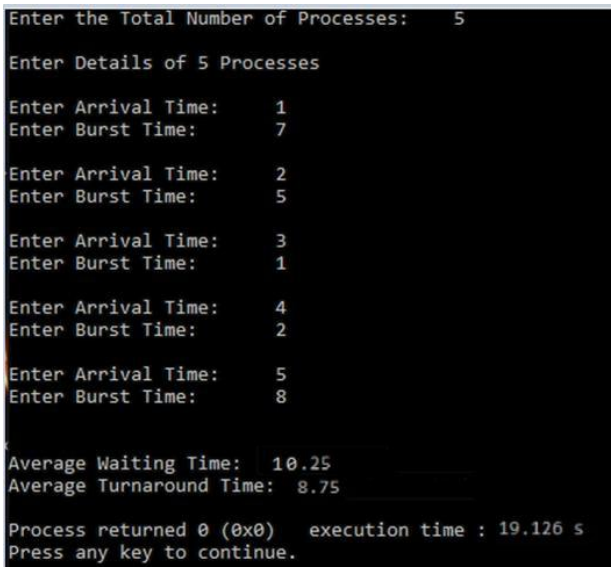
2. LJF parallel

CODE:

```
#include <stdio.h>
#include<omp.h>
int main()
{
   double arr[10];
   omp_set_num_threads(4);
   double min_val=0;

    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    printf("\nEnter the Total Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Details of %d Processes\n", limit);
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;

   #pragma omp parallel for shared(i) reduction(max: max_val)
   for( i=0; i<limit ; i++)
   {
     if(burst_time[i] > max_val && arrival_time[i] <= time)
     {

        smallest=i;
     }
   }

        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
            turnaround_time = turnaround_time + end - arrival_time[smallest];
```

```
            }
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
        printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
        return 0;
    }
```

OUTPUT:



3. Priority scheduling algorithm(parallel)

CODE:

```
#include<stdio.h>
#include<omp.h>
int k=0;

int partition(int arr[], int low_index, int high_index)
{
int i, j, temp, key;
key = arr[low_index];
i= low_index + 1;
j= high_index;
while(1)
{
while(i < high_index && key >= arr[i])
   i++;
while(key < arr[j])
   j--;
if(i < j)
{
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
else
{
temp= arr[low_index];
arr[low_index] = arr[j];
arr[j]= temp;
return(j);
}}}
void quicksort(int arr[], int low_index, int high_index)
{
int j;

if(low_index < high_index)
{
j = partition(arr, low_index, high_index);
printf("Pivot element with index %d has been found out by thread %d\n",j,k);
```

```
#pragma omp parallel sections
{
   #pragma omp section
   {
      k=k+1;
      quicksort(arr, low_index, j - 1);
   }

   #pragma omp section
   {
      k=k+1;
      quicksort(arr, j + 1, high_index);
   }}}}
int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;          //contains process number
   }

   //sorting burst time, priority and process number in ascending order using
selection sort
   quicksort(pr, 0, n - 1);

   wt[0]=0;    //waiting time for first process is zero

   //calculate waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
```

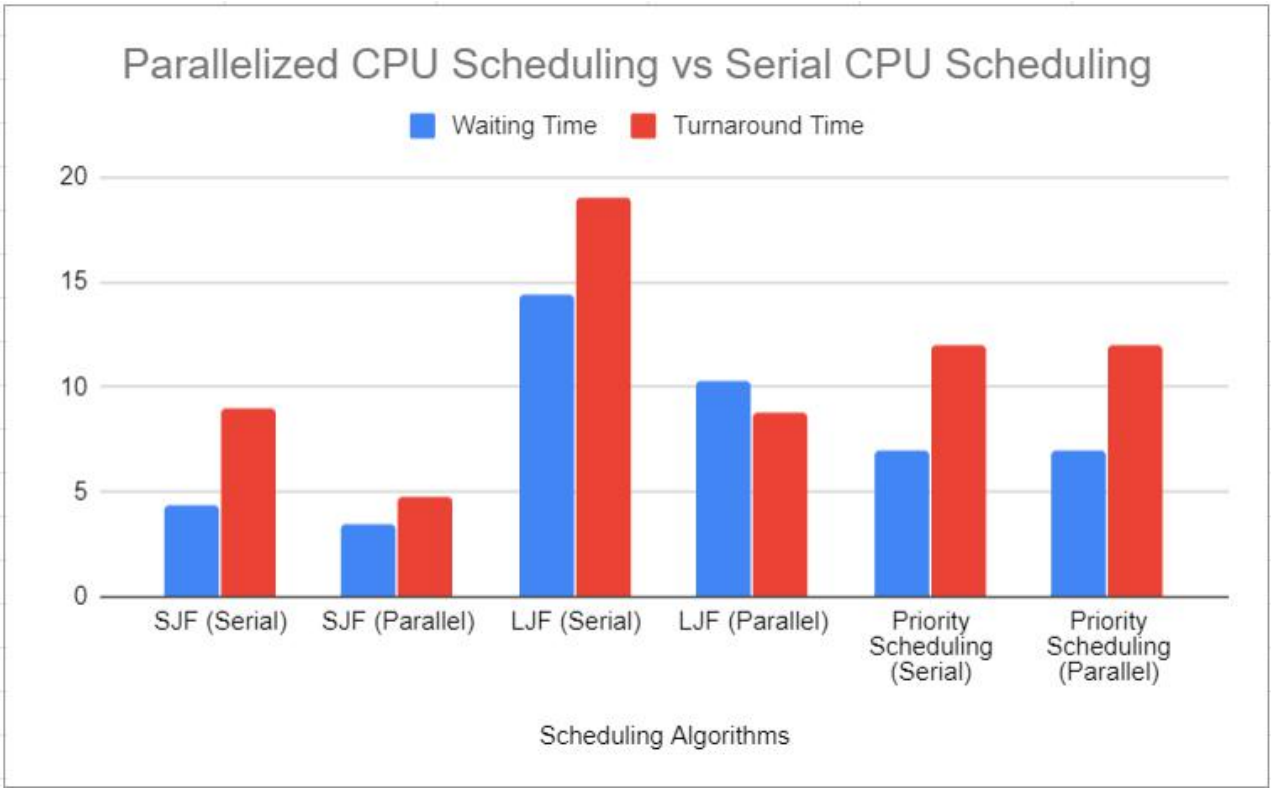
```
        total+=wt[i];
    }

    avg_wt=total/n;      //average waiting time
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

    return 0;
}
```

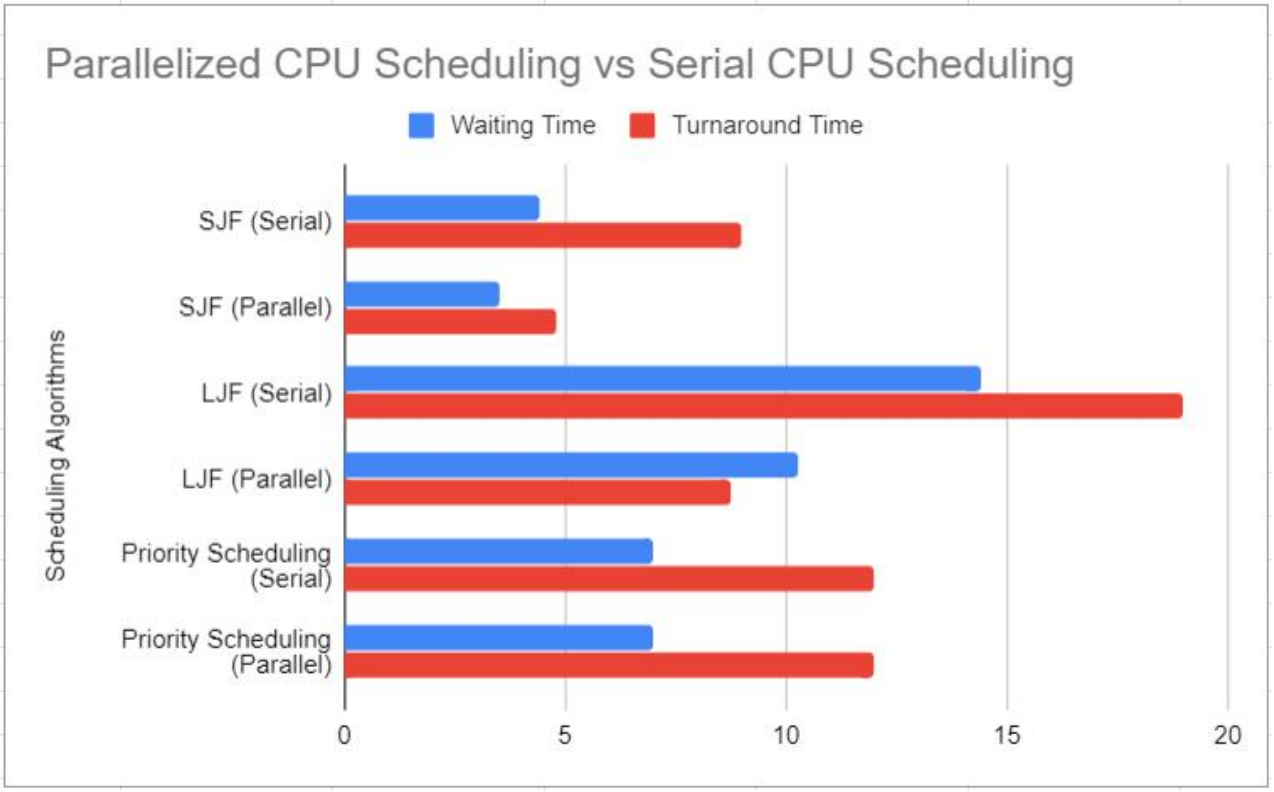
OUTPUT:

```
Enter Total Number of Process:5

Enter Burst Time and Priority

P[1]
Burst Time:7
Priority:1

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:1
Priority:3

P[4]
Burst Time:3
Priority:4

P[5]
Burst Time:8
Priority:5
Pivot element with index 0 has been found out by thread 0
Pivot element with index 1 has been found out by thread 2
Pivot element with index 2 has been found out by thread 4
Pivot element with index 3 has been found out by thread 6

Process     Burst Time          Waiting Time    Turnaround Time
P[1]              7                 0                 7
P[2]              2                 7                 9
P[3]              1                 9                 10
P[4]              3                 10                13
P[5]              8                 13                21

Average Waiting Time=7
Average Turnaround Time=12
```

OBSERVATIONS

Scheduling Algorithms	Waiting Time	Turnaround Time
SJF (Serial)	4.4	9
SJF (Parallel)	3.5	4.8
LJF (Serial)	14.4	19
LJF (Parallel)	10.25	8.75
Priority Scheduling (Serial)	7	12
Priority Scheduling (Parallel)	7	12




Parallelized CPU Scheduling vs Serial CPU Scheduling
Waiting Time
Turnaround Time
Scheduling Algorithms
SJF (Serial)
SJF (Parallel)
LJF (Serial)
LJF (Parallel)
Priority Scheduling (Serial)
Priority Scheduling (Parallel)
0
5
10
15
20

CONCLUSION

Task scheduling in parallel processing is a technique in which processes are assigned to different processors in series task are assigned to a single processor. PROCESSOR ONLY Nowadays there is a different kind of scheduling algorithms and techniques used to reduce the execution time of tasks scheduling of jobs in parallel is becoming the subject of much research. The problem of job scheduling is to determine how resources should be shared in order to maximize the system's utility. This problem has been extensively studied for well over a decade. In this paper, we discuss the execution time of scheduling algorithm (LJF, SJF, PRIORITY SCHEDULING) in series and parallel, and we also discussed how deployed scheduling policies can be improved to meet existing requirements, specific research challenges and future scope. In job scheduling, the most important thing need to be considered while executing are throughput, utilization, and response time.

REFRENCES

1. By- Uwe Schwiegelshohn & Ramin Yahyapour –"Analysis of First-Come-First-Serve Parallel job Scheduling" JUNE-1998
2. By- Sumit Kumar Nager & Naseeb Singh Gill –"Comparative study of various CPU Scheduling algorithms" MARCH -2001
3. By- Eitan Frachtenberg and Uwe Schwiegelshohn –"New Challenges of Parallel Job Scheduling" JAN-2007
4. By- Arvind Seth, Vishaldeep Singh –"Types of Scheduling Algorithms in Parallel computing" 2007 :
5. By- Yang Rui, Zhang Xinyu –"An Algorithm for assigning tasks in parallel computing" JAN-2010
6. By- Dror G.Feitelson1, Larry Rudolph1, Uwe Schwiegelshohn2, Kenneth C. Sevcik3 and Parkson Wong41- "Theory and Practice in Parallel Job Scheduling"- 12-JULY-2015
7. Comparison Study of Processor Scheduling Algorithms:
8. https://www.tutorialspoint.com
9. https://en.wikipedia.org