Algorithm Design and Analysis Project Report

**Parallel Computing vs Serial Computing**

horizontal line

# 



**DELHI TECHNOLOGICAL UNIVERSITY**

**PROJECT REPORT**

**Submitted By: Submitted To:**

Abhishek Kumar Singh (2K19/CO/021)Mr. Sanjay Kumar

Amit Kumar Jha (2K19/CO/053)

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi, 110042

**CANDIDATE’S DECLARATION**

We, Abhishek Kumar Singh, Amit Kumar Jha Roll No(s). 2K19/CO/021, 2K19/CO/053, students of B.Tech. in Computer Science & Engineering, hereby declare that the project Dissertation titled **Parallel Computing vs Serial Computing** which is submitted by us to the Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the mid-semester component evaluation, semester-4 of Bachelor of Technology is original and not copied from any source without proper citation. This work has not previously formed a basis for the award of any degree, Diploma Associateship, Fellowship, or any similar title or recognition.

Place: Delhi Abhishek Kumar Singh

Date: 25/05/2021 Amit Kumar Jha

**Department of Computer Science & Engineering**

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi, 110042

**CERTIFICATE**

We hereby declare that the project Dissertation titled “ **Parallel Computing vs Serial Computing**” which is submitted by Abhishek Kumar Singh, Amit Kumar Jha Roll No(s). 2K19/CO/021, 2K19/CO/053 Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the mid-semester component evaluation, semester-4 of Bachelor of Technology, is the record of the project work carried out by the students under my supervision.

Place: Delhi **Mr. Sanjay Kumar**

Date: 25/05/2021 Delhi Technological University

(Formerly DCE)

Bawana Road, Delhi,110042

**ACKNOWLEDGEMENT**

At the very outset of this report, we would like to extend our sincere and heartfelt obligation towards all the personages who have guided us with the project.

s

A special thanks to **Mr. Sanjay Kumar** for teaching us the subject “**Algorithm Design and Analysis**”. He helped us visualize the subject and to find its applications in real life. He supervised us with the intricacies of this project. He also offered many relevant and productive recommendations for the project, for which we are very grateful. We would also like to extend sincere gratitude towards our Vice-Chancellor Mr. Yogesh Singh for allowing the students to improve their theoretical and practical skills with practical and crucial subjects like Algorithm Design And Analysis.

Finally, a thank you to all our friends who helped us with the project, gave worthy ideas.

**ABSTRACT**

In computers, serial processing is the processing of program instructions just one at a time whereas parallel processing is the processing of program instructions by dividing the program into multiple fragments and processing these fragments simultaneously. The objective of parallel processing is running a program in less time. Parallel computing has been around for many years. Recently, the interest in parallel computing has grown due to the introduction of multi core processors at a reasonable price for the common people. The goal of this paper is to analyze and compare serial computing with parallel computing.

**INTRODUCTION**

**Serial Computing**

Serial computing is a type of computation where instructions are executed sequentially one after another. In serial computing, a problem is broken into a series of instructions and the instructions are then executed sequentially one after another. Only one instruction may execute at any moment in time.

A serial processor is a processor type used by systems where the central processing unit (CPU) carries out just one machine-level operation at a time. The term is often used in contrast to a parallel processor, which features more than one CPU to perform parallel processing.

In 2005 Intel launched the first dual-core processor for end users; prior to that, every computer processor used the serial processing technology.

**Parallel Computing**

Parallel computing is a type of computation where many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved at the same time. There are several different forms of parallel computing: bit-level, instruction-level, data, and task parallelism. Parallelism has long been employed in high-performance computing, but has gained broader interest due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.

Parallel programming involves the concurrent computation or simultaneous execution of processes or threads at the same time. In parallel programming, we have multiple processes executed at the same time. Parallel Processing Systems are designed to speed up the execution of programs.

Parallel computing is an evolution of serial computing where the jobs are broken into discrete parts that can be executed concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different CPUs. Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated and synchronized.

A few years ago, parallel computers could be found only in research laboratories and they were used mainly for computation intensive applications like numerical simulations of complex systems. Today, there are a lot of parallel computers available in the market; used to execute both data intensive applications in commerce and computation intensive applications in science and engineering.

Today, new applications arise and demand faster computers. Commercial applications are the most used on parallel computers. A computer that runs such an application; should be able to process large amounts of data in sophisticated ways. These applications include graphics, virtual reality, and decision support, parallel databases, medicine diagnosis and so on. We can say with no doubt that commercial applications will define future parallel computers architecture but scientific applications will remain important users of parallel computing technology.

**LITERATURE REVIEW**

Serial vs Parallel Computing

Traditionally, software has been written for serial computation:

* To be run on a single computer having a single Central Processing Unit (CPU).
* A problem is broken into a discrete series of instructions.
* Instructions are executed one after another.
* Only one instruction may execute at any moment in time.

Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem:

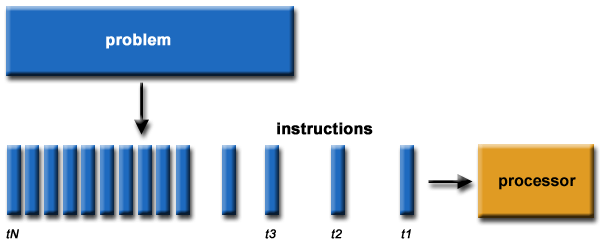
* To be run using multiple CPUs.
* A problem is broken into discrete parts that can be solved concurrently.
* Each part is further broken down to a series of instructions.
* Instructions from each part execute simultaneously on different CPUs or cores.

Serial processing was the best way of computing data sets until hardware and software technologies finally caught up and made true parallel processing a reality.

**THEORY**

**Serial Computing**

Traditionally software has been written for serial computation:- run on a single computer-instructions are run one after another-only one instruction executed at a time.



All the Instructions that are entered first in the system will be executed first and the instructions that are entered later Will be executed later. Serial processing is purely sequential. A system using standard serial processing techniques lets every object take exactly the same average time frame for processing. Moreover, the subsequent object starts processing only after the completion of the previous one.

Various single-core processors can be used together for handling serial processing by means of parallel computer clusters that are networked, or by operating multiple processors on a single motherboard.

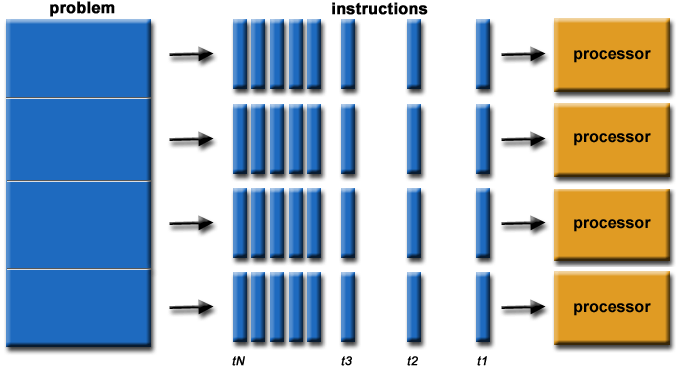
Programs intended for serial processing can make use of just a single core at a time, where the tasks are processed in a sequential order. Functions of a serial processor can be compared with a grocery store cashier who single-handedly handles different lanes, looking over every customer simultaneously. The cashier (like the CPU) switches from lane to lane to check out a number of items at a time prior to addressing the next one, with the objective of completing every order concurrently.

**Limits to serial computing**

* Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
* Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
* Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

**Parallel Computing**

Parallel computing implies simultaneous processing on various objects or subsystems. The processing, however, may complete at different times. Individual as well as overall processing periods can be random in either type of processing. That is, the time periods essential for processing an item or executing an operation may differ from trial to trial.



Parallel operating systems are used to interface multiple networked computers to complete tasks in parallel. The architecture of the software is often a UNIX-based platform, which allows it to coordinate distributed loads between multiple computers in a network. Parallel operating systems are able to use software to manage all of the different resources of the computers running in parallel, such as memory, caches, storage space, and processing power. Parallel operating systems also allow a user to directly interface with all of the computers in the network.

A parallel operating system works by dividing sets of calculations into smaller parts and distributing them between the machines on a network. To facilitate communication between the processor cores and memory arrays, routing software has to either share its memory by assigning the same address space to all of the networked computers, or distribute its memory by assigning a different address space to each processing core.

Sharing memory allows the operating system to run very quickly, but it is usually not as powerful. When using distributed shared memory, processors have access to both their own local memory and the memory of other processors; this distribution may slow the operating system, but it is often more flexible and efficient.

**Advantages of Parallel computing**

* Save time and/or money: In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. Parallel computers can be built from cheap, commodity components.
* Solve larger problems: Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory.
* Provide concurrency: A single compute resource can only do one thing at a time. Multiple computing resources can be doing many things simultaneously

**PROPOSED APPROACH & SOLUTION**

**Serial Computing**

In this problem, we will simulate a program that processes a list of jobs in serial. Jobs arrive in some order. There is only one processor, and it processes the incoming jobs in the order of their arrival. If the processor starts to process some job, it doesn’t interrupt or stop until it finishes the processing of this job. The computer processing the jobs has a buffer of fixed size 𝑆. When jobs arrive, they are stored in the buffer before being processed. However, if the buffer is full when a job arrives, it is dropped and won’t be processed at all. If several jobs arrive at the same time, they are first all stored in the buffer (some of them may be dropped). The computer processes the jobs in the order of their arrival, and it starts processing the next available job from the buffer as soon as it finishes processing the previous one. If at some point the computer is not busy, and there are no jobs in the buffer, the computer just waits for the next job to arrive. Note that a job leaves the buffer and frees the space in the buffer as soon as the computer finishes processing it.

**Algorithm / Code**

Response Process(const Request &request) {

while (!finish\_time\_.empty()) {

if (finish\_time\_.front() <= request.arrival\_time)

finish\_time\_.pop();

else

break;

}

if (finish\_time\_.size() == size\_)

return Response(true, -1);

if (finish\_time\_.empty()){

finish\_time\_.push(request.arrival\_time + request.process\_time);

return Response(false, request.arrival\_time);

}

int last\_element = finish\_time\_.back();

finish\_time\_.push(last\_element + request.process\_time);

return Response(false, last\_element);

}

vector <Response> ProcessRequests(const vector <Request> &requests, Buffer \*buffer) {

vector <Response> responses;

for (int i = 0; i < requests.size(); ++i)

responses.push\_back(buffer->Process(requests[i]));

return responses;

}

**Parallel Computing**

In this problem, we will simulate a program that processes a list of jobs in parallel. Operating systems such as Linux, MacOS or Windows all have special programs in them called schedulers which do exactly this with the programs on your computer.

**Algorithm / Code**

void ParallelJobs() {

assigned\_workers\_.resize(jobs\_.size());

start\_times\_.resize(jobs\_.size());

priority\_queue<Worker, vector<Worker>, WorkerCompare> pq;

for(int i = 0; i < num\_workers\_; i++) {

pq.push(Worker(i));

}

for (int i = 0; i < jobs\_.size(); i++) {

Worker freeThread = pq.top();

pq.pop();

assigned\_workers\_[i] = freeThread.id;

start\_times\_[i] = freeThread.nextFreeTime;

freeThread.nextFreeTime += jobs\_[i];

pq.push(freeThread);

}

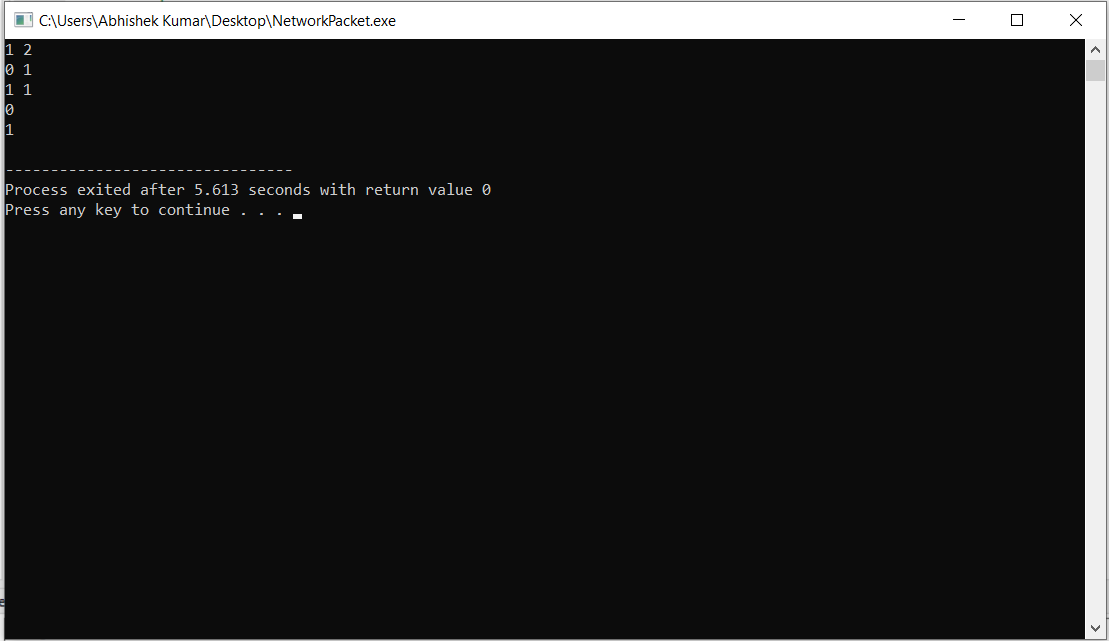
}

**RESULTS AND ANALYSIS**

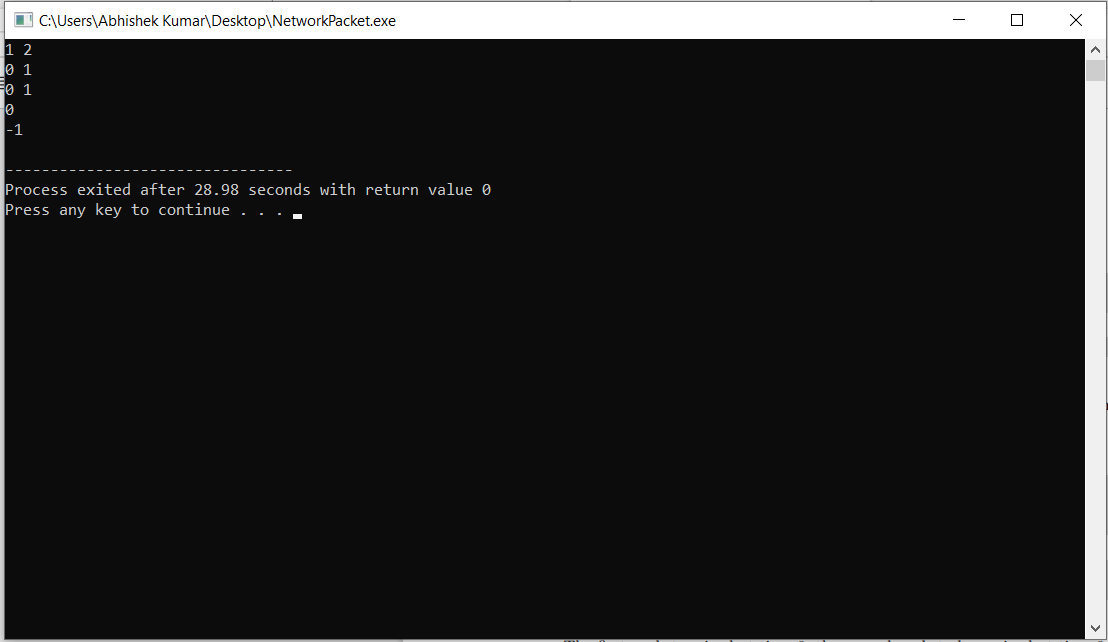
**Serial Computing**

Input : The first line of the input contains the size 𝑆 of the buffer and the number 𝑛 of incoming jobs. Each of the next 𝑛 lines contains two numbers. 𝑖-th line contains the time of arrival 𝐴𝑖 and the processing time 𝑃𝑖 (both in milliseconds) of the 𝑖-th job. It is guaranteed that the sequence of arrival times is non-decreasing (however, it can contain the exact same times of arrival in milliseconds — in this case the job which is earlier in the input is considered to have arrived earlier).

Output : For each job output either the moment of time (in milliseconds) when the processor began processing it or −1 if the job was dropped (output the answers for the jobs in the same order as the jobs are given in the input).



Explanation : The first job arrived at time 0, the computer started processing it immediately and finished at time 1. The second job arrived at time 1, and the computer started processing it immediately.

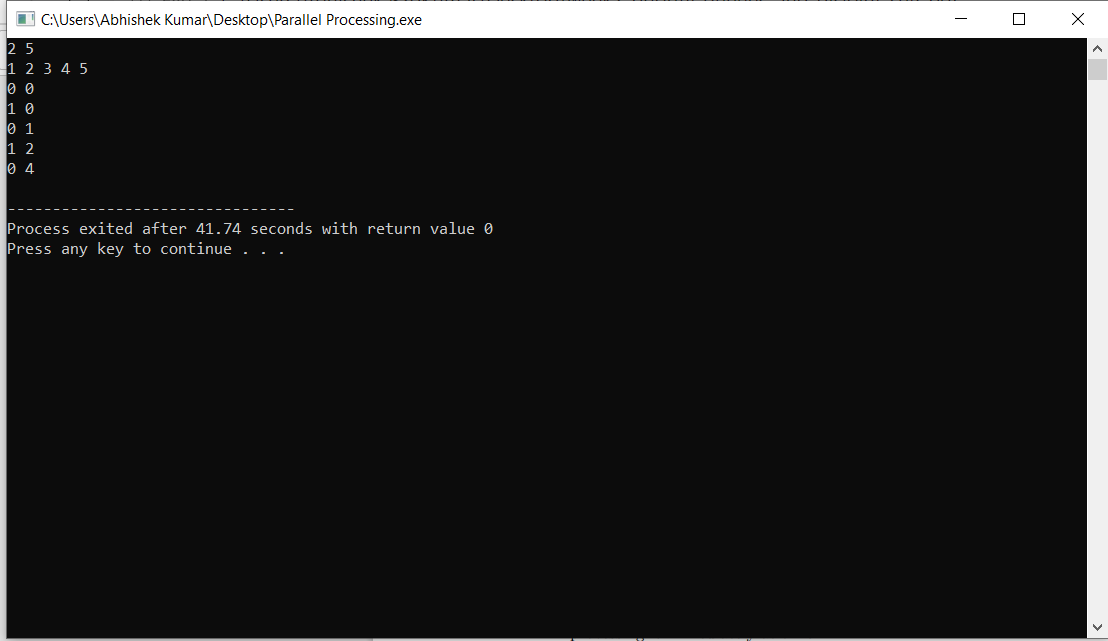


Explanation : The first job arrived at time 0, the second job also arrived at time 0, but was dropped, because the buffer has size 1 and it was full with the first job already. The first job started processing at time 0, and the second job was not processed at all.

**Parallel Computing**

Input : The first line of the input contains integers 𝑛 and 𝑚. The second line contains 𝑚 integers 𝑡𝑖 — the times in seconds it takes any thread to process 𝑖-th job. The times are given in the same order as they are in the list from which threads take jobs. Threads are indexed starting from 0.

Output : Output exactly 𝑚 lines. 𝑖-th line (0-based index is used) should contain two space separated integers — the 0-based index of the thread which will process the 𝑖-th job and the time in seconds when it will start processing that job.



Explanation :

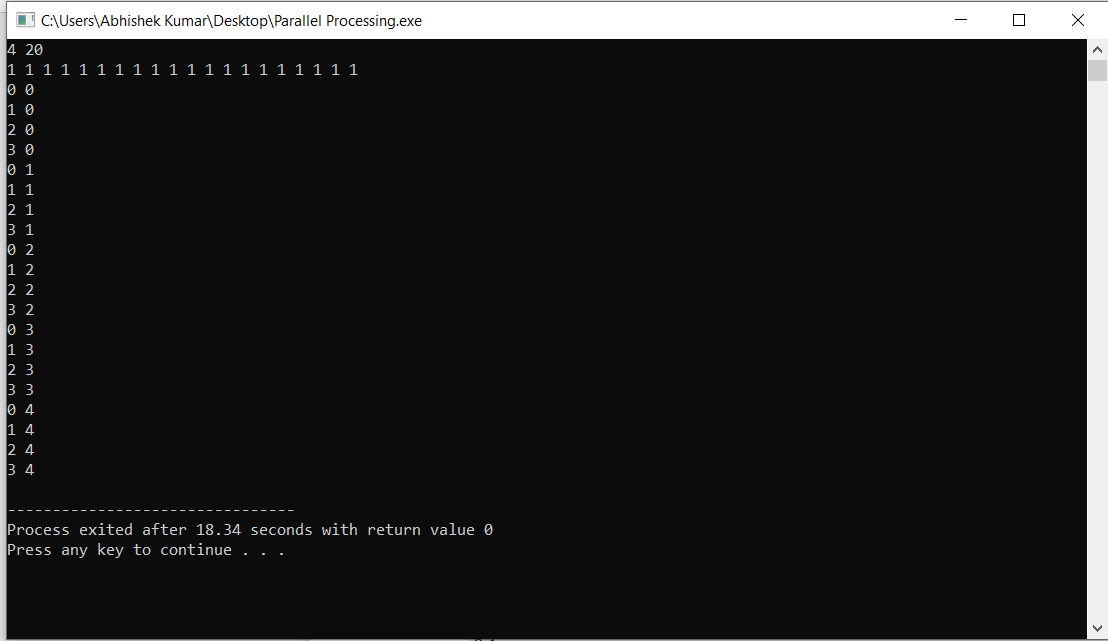
1. The two threads try to simultaneously take jobs from the list, so thread with index 0 actually takes the first job and starts working on it at the moment 0.

2. The thread with index 1 takes the second job and starts working on it also at the moment 0.

3. After 1 second, thread 0 is done with the first job and takes the third job from the list, and starts processing it immediately at time 1.

4. One second later, thread 1 is done with the second job and takes the fourth job from the list, and starts processing it immediately at time 2.

5. Finally, after 2 more seconds, thread 0 is done with the third job and takes the fifth job from the list, and starts processing it immediately at time 4.



Explanation : Jobs are taken by 4 threads in packs of 4, processed in 1 second, and then the next pack comes. This happens 5 times starting at moments 0, 1, 2, 3 and 4. After that all the 5 × 4 = 20 jobs are processed.

**CONCLUSION**

We have successfully implemented and simulated the serial and parallel processing of jobs. The parallel processing saves time and money as many resources working together will reduce the time and cut potential costs. Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key. It can be impractical to solve larger problems on serial computing. Complex, large datasets, and their management can be organized only and only using parallel computing’s approach. Serial computing ‘wastes’ the potential computing power, thus Parallel Computing makes better work of hardware.

**REFERENCES**

1. Ms. Rupa Yashwanta Nagpure and Prof Sandhya Dahake, “Research Paper on Basic Parallel Processing”, IOSR Journal of Engineering (IOSR JEN), ISSN (e): 2250-3021, ISSN (p): 2278-8719, PP 77-83
2. Abhay B. Rathod, Rajratna Khadse and M Faruk Bagwan, “SERIAL COMPUTING vs. PARALLEL COMPUTING: A COMPARATIVE STUDY USING MATLAB”, International Journal of Computer Science and Mobile Computing, Vol. 3, Issue. 5, May 2014, pg.815 – 820