



## PARALLEL COMPUTING:

Parallel computing is a type of computation in which many calculations or the execution of processes are carried out concurrently. Large problems can often be divided into smaller ones, which can then be solved at the same time.

Parallel computing, uses multiple processing elements concurrently to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm concurrently with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Types of Parallelism:

1. Bit-level parallelism is a form of parallel computing based on increasing processor word size, depending on very-large-scale integration (VLSI) technology. Enhancements in computers designs were done by increasing bit-level parallelism. Increasing the word size reduces the number of instructions the processor must execute in order to perform an operation on variables whose sizes are greater than the length of the word.

2. Instruction-level parallelism (ILP) is a measure of how many of the instructions in a computer program can be executed simultaneously.

There are two approaches to instruction level parallelism:

1. Hardware
2. Software

Hardware level works upon dynamic parallelism whereas, the software level works on static parallelism.

3. Task parallelism (also known as function parallelism and control parallelism) is a form of parallelization of computer code across multiple processors in parallel computing environments. Task parallelism focuses on distributing tasks—concurrently performed by processes or threads—across different processors. A common type of task parallelism is pipelining which consists of moving a single set of data through a series of separate tasks where each task can execute independently of the others.

## Classes of parallel computers

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. This classification is broadly analogous to the distance between basic computing nodes. These are not mutually exclusive; for example, clusters of symmetric multiprocessors are relatively common.

### Multi-core computing:

A multi-core processor is a processor that includes multiple processing units (cores) on the same chip. This processor differs from a superscalar processor, which includes multiple execution units and can issue multiple instructions per clock cycle from one instruction stream (thread); in contrast, a multi-core processor can issue multiple instructions per clock cycle from multiple instruction streams. IBM's Cell microprocessor, designed for use in the Sony PlayStation 3, is a prominent multi-core processor. Each core in a multi-core processor can potentially be superscalar as well—that is, on every clock cycle, each core can issue multiple instructions from one thread.

### Symmetric multiprocessing:

A symmetric multiprocessor (SMP) is a computer system with multiple identical processors that share memory and connect via a bus. Bus contention prevents bus architectures from scaling. As a result, SMPs generally do not comprise more than 32 processors. Because of the small size of the processors and the significant reduction in the requirements for bus bandwidth achieved by large caches, such symmetric multiprocessors are extremely cost-effective, provided that a sufficient amount of memory bandwidth exists.

### Distributed computing:

A distributed computer (also known as a distributed memory multiprocessor) is a distributed memory computer system in which the processing elements are connected by a network. Distributed computers are highly scalable. The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel.

### Cluster computing:

A cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single computer. Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, load balancing is more difficult if they are not.

### Massively parallel computing:


A massively parallel processor (MPP) is a single computer with many networked processors. MPPs have many of the same characteristics as clusters, but MPPs have specialized interconnect networks (whereas clusters use commodity hardware for networking). MPPs also tend to be larger than clusters, typically having "far more" than 100 processors. In an MPP, each CPU contains its own memory and copy of the operating system and application. Each subsystem communicates with the others via a high-speed interconnect.

### Grid computing:

Grid computing is the most distributed form of parallel computing. It makes use of computers communicating over the Internet to work on a given problem. Because of the low bandwidth and extremely high latency available on the Internet, distributed computing typically deals only with embarrassingly parallel problems.

## Specifications of System:

Processor

Name	Intel Core i7 6700HQ				
Code Name	Skylake	Max TDP	45.0 W		
Package	Socket 1440 FCBGA				
Technology	14 nm	Core VID	0.807 V		
Specification	Intel® Core™ i7-6700HQ CPU @ 2.60GHz				
Family	6	Model	E	Stepping	3
Ext. Family	6	Ext. Model	5E	Revision	R0
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX				

Clocks (Core #0)

Core Speed	898.46 MHz
Multiplier	x 9.0 ( 8 - 35 )
Bus Speed	99.73 MHz
Rated FSB	

Cache

L1 Data	4 x 32 KBytes	8-way
L1 Inst.	4 x 32 KBytes	8-way
Level 2	4 x 256 KBytes	4-way
Level 3	6 MBytes	12-way

Selection

Socket #1

Cores

4

Threads

8

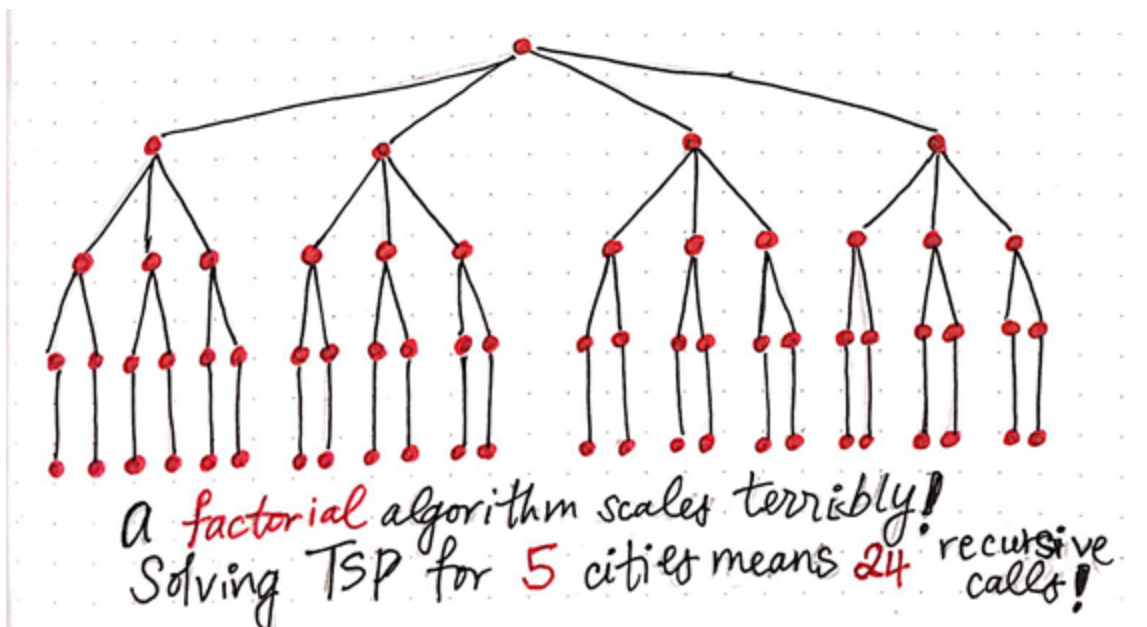
# Travelling Salesman Problem:

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Naive Solution:

- 1) Consider city 1 as the starting and ending point.
- 2) Generate all  $(n-1)!$  Permutations of cities.
- 3) Calculate cost of every permutation and keep track of minimum cost permutation.
- 4) Return the permutation with minimum cost.

Time Complexity:  $O(n!)$



However, this approach is not feasible for even a medium sized input as  $n!$  growth is very large and it will take large amount of time to compute the answer.

The part generating new permutations is parallelized. Each thread generates a part of the solution which is later combined using reduction clause.

A better approach to the travelling salesman problem is through dynamic programming( Branch and bound method).

The appropriate sub-problem for the TSP in this case is the initial portion of a tour. Suppose we have started at city 1 as required, have visited a few cities, and are now in city  $j$ . In order to extend this partial tour we certainly need to know  $j$ , since this will determine which cities are most convenient to visit next. And we also need to know all the cities visited so far, so that we don't repeat any of them.

For a subset of cities  $S \subseteq \{1,2,\dots,n\}$  that includes 1, and  $j \in S$ , let  $C(S,j)$  be the length of the shortest path visiting each node in  $S$  exactly once, starting at 1 and ending at  $j$ .

This algorithm(bottom up dynamic programming) reduces the time complexity of the problem to  $O(n^2 2^n)$ . This approach starts with the smallest possible subproblems, figures out a solution to them, and then slowly builds itself up to solve the larger, more complicated subproblem.

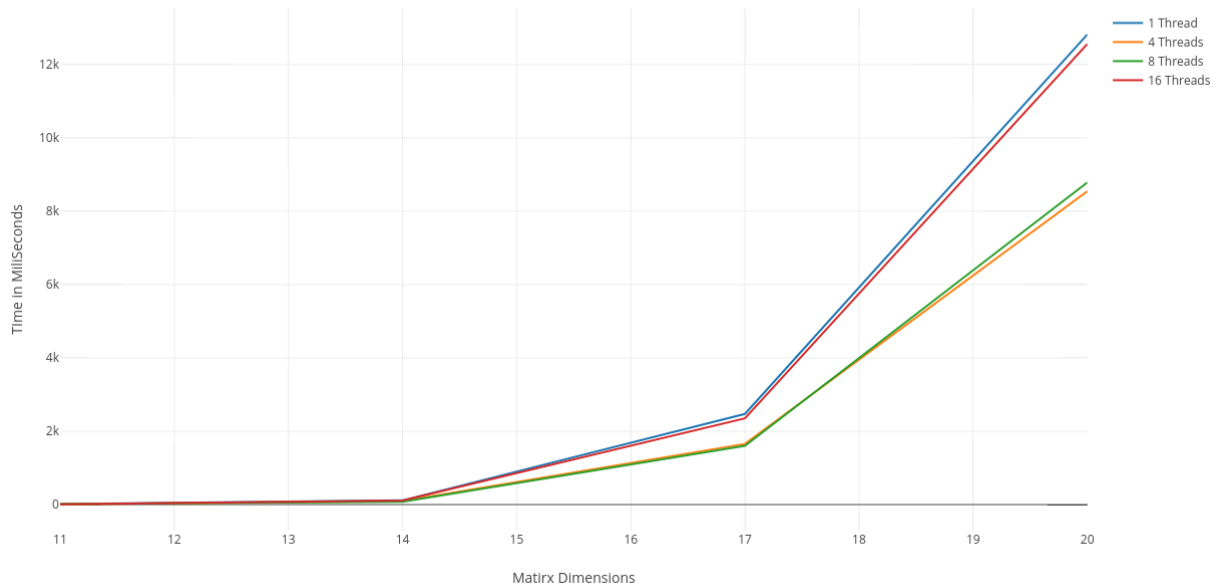
There are at most  $2^n \cdot n$  sub-problems, and each one takes linear time to solve. The total running time is therefore  $O(2^n n^2)$ .

For each level or recursion when the optimal route is found by the algorithm it is stored in an array containing the best route. This action is performed under the omp pragma critical section to prevent race condition by making it accessible to only 1 thread to write.

```
#pragma omp critical
    if (curr_res < final_res) {
        copyToFinal(curr_pat);
        final_res = curr_res;
    }
```

## Results

Ciites	Execution Time in ms			
	1 Thread	4 Threads	8 Threads	16 Threads
11	15	6	14	6
14	114	94	77	113
17	2472	1653	1601	2352
20	12818	8546	8783	12562



# Word Count Problem:

This involves of counting words from large number of files presents the sequential program of counting task and parallel program of counting task. Along with counting task it also shows the time taken to execute the programs indicates the speedup of the programs. For performing the work of counting words from large files both the programs have used the system calls and functions related to file management.

## Working Model

1. Open specified file and start reading contents
2. If a delimiting character is encountered then the buffer is treated as a word and added into the structure, and its count is initialized to 1.
3. If a previously occurring word is encountered then, its count is incremented by 1 and we continue reading input.
4. Input is taken until one of the following occurs, either end of file is reached or limit of number of words is touched.
5. Execution time for the above computation is calculated and emitted

```
#pragma omp parallel for schedule(static,200) private(returned_index,token,check) shared(histo)
for(j=0; j < NUM_LINES ; j++){

    token = strtok(lines[j], " ");

    while (token != NULL){

        check = check_word(token,histo,NUM_WORDS);

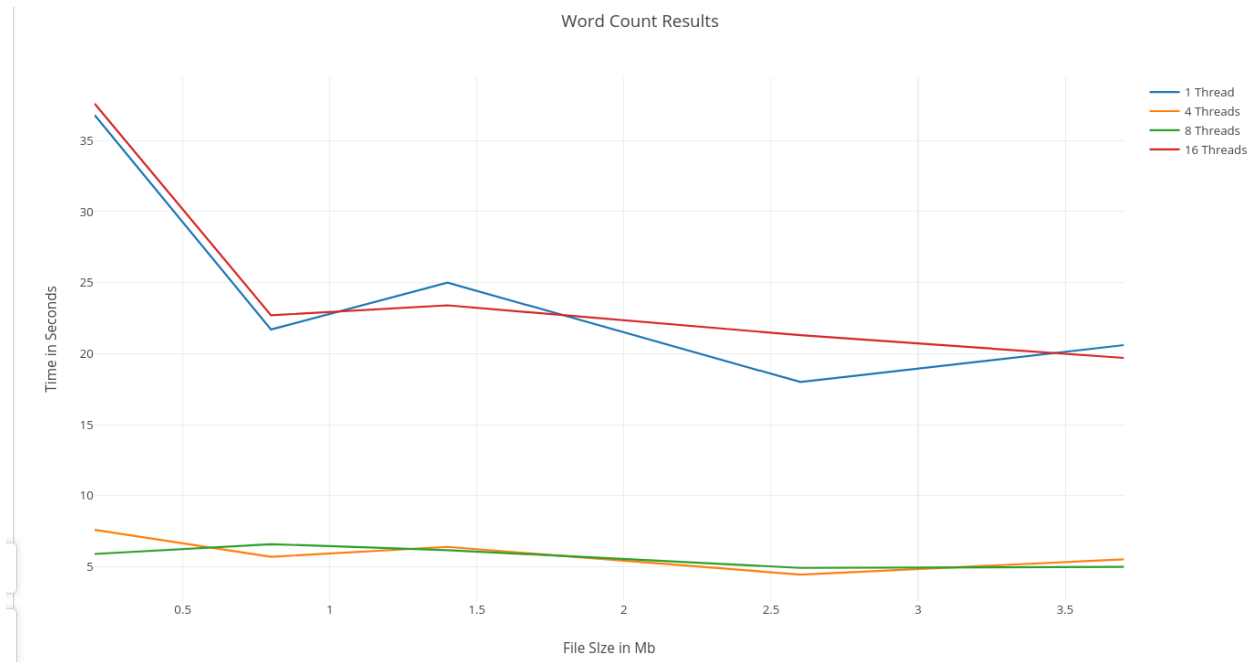
        if(!check){
            #pragma omp critical
            {
                strcpy(histo[i].word, token);
                histo[i].freq=1;
            }
        }
        else
            #pragma omp critical
            histo[check].freq++;

        token = strtok(NULL, " ");
        i++;
    }
}
```



## Results

File Size	Execution Time in seconds			
	1 Thread	4 Threads	8 Threads	16 Threads
0.2	36.8	7.6	5.9	37.6
0.8	21.7	5.7	6.59	22.7
1.4	25	6.4	6.17	23.4
2.6	18	4.45	4.92	21.3
3.7	20.6	5.52	4.99	19.7



## Conclusion:

From the above graphs and tables in case of both TSP and Word count problems we can conclude that the most optimal solution was found was when we used 4 or 8 threads. If the number of threads were increased to 16 the performance suffered due to high context switching(overhead of maintaining the threads).