Hw1

1. What are the two main styles of parallelism? Explain.

A: Parallelism is a form of computation in which many tasks/instructions are carried out simultaneously. It is basically breaking down the complex problem into smaller pieces and assigning each smaller components to the processes running in parallel. The main two styles of parallelism are:

1. Data level parallelism
2. Task level parallelism

1.Data level parallelism:

Data level parallelism, focuses on distributing data across multiple processing nodes in parallel computing environment. In a multiprocessor system, if each processor performs the same task on different smaller sets of distributed dataset, we can parallelize the program based on data level parallelism. Sometimes this is done with single execution thread which controls all the operations done on all individual sets of distributed data. It is also done by multiple threads that execute the same piece of code.

Example:

```
for(i=0;i<100;i++){

x[i] = y[i]+z[i]; }
```

In this example we have to perform addition of array elements. There is a loop which executes 100times on same dataset. So in parallel computing, this dataset is split across multiple processor, say 100 iterations on 100 data elements are split equally between 10 processors, so that each processor executes only 10times on 10 data elements. Data is split and computation time is reduced.

2.**Task level parallelism:**

Task level parallelism is focuses on distributing execution task/functions across multiple processing nodes in parallel computing environment. In a multiprocessor system, task parallelism is achieved when each processor executes a different thread on the same or different data. The threads may execute the same or different code. Threads communicate with other threads as they work. Communication usually takes place by passing data from one thread to the next as part of a workflow.

Example:

```
for(i=0;i<100;i++){

 x[i+1] = x[i]+y[i];

 }

 for(j=0;j<1000;j++){

 m[i+1] = m[i]+n[i];

 }
```

In this example we can run these loops separately in multiple processing nodes. i.e one processor will work on one FOR loop and other processor will work on the other FOR loop. The entire task is split into multiple process and processed separately, thus reducing computation time.

Data level parallelism mostly emphasizes the parallelized nature of the data, whereas Task level parallelism emphasizes the parallelized nature of the processing.

2. What are the two main types of locality? Explain.

A: Basically Principle of Locality is the phenomenon of the same value or related storage locations being frequently accessed, i.e. Program access a relatively small portion of the address space at any instant of time.

Example: 90% of time in 10% of the code

There are two main types of Locality:

1. Temporal locality (Time)

2. Spatial locality (Space).

1. Temporal locality

Temporal Locality refers to the reuse of specific data or resources within relatively small time durations or accessed in the near future.

Example: code in loops, top of stack.

In Temporal locality, that two instructions reference the same location within a relatively short timeframe. For example, in the code given, a[i] is referenced frequently, with instructions like a[i] = a[i] + 2 and a[i] = a[i] - 3 being executed very close together. Here References to i are also temporally local, because every time i is referenced to a[i] .

2. Spatial locality

Spatial Locality refers to the use of data elements within relatively close storage locations. That means, Data elements at addresses close to the addresses of recently accessed items will be accessed in the near future

Example: Sequential code, Elements of arrays

Sequential Locality is special case of Spatial locality, occurs when data elements are arranged and accessed Sequentially. For example, it occurs on when we traverse the elements in a one dimensional array. When two instructions reference contiguous memory locations.

```
for(i = 0; i < 10; i++)
  { for(j = 0; j < 10; j++)
      a[i] = a[i]*j;
```

}

Here References to a[i] are a good example of this, we can assume that most of the time that a[0] and a[1] will be next to each other in memory.

3.What are the three basic programming paradigms for parallel processing? Explain.

A: The three basic programming paradigms for parallel processing are :

1. Shared memory programming model: This program has a collection of threads of control. In this model of parallel programming, a single process can have multiple, concurrent execution paths. These threads can be created dynamically, or statically either in some languages or using any user libraries. Each thread has a set of private variables. Threads communicate implicitly by writing and reading shared variables through global memory. Threads coordinate by synchronizing on shared variables to ensure that more than one thread is not updating the same global address at any point of time.

2. Message passing programming model: This program consists of a collection of named processes. These processes are usually fixed at program startup time. Every processes named and thread of control has local address space. These does not have shared data. Every process communicate with each other by sending and receiving messages. Each processor has its own memory and cache but cannot directly access another processor's memory. In this model, logically shared data is partitioned over local processes.

3. Data parallel model: This program contains a single thread of control which in turn consists of parallel operations. Parallel operations is applied to all elements of a data structure used. In this model of communication and coordination is done implicitly. And it's easy and elegant to understand. Each task is a duplicate of a single program and works on a different partition of the same data structure.

4.Discuss the difference between shared address space machines and distributed address space machines. Discuss the advantages and disadvantages of both architectures.

A:Shared address space machines : This platform provides a shared data space between parallel tasks. Read/write operations are carried out through a shared address space that is accessible to all processors. Processors interact by modifying data objects stored in this shared address space. Memory in shared address space can be local or global.
1. A shared memory machine has a single address space shared by all processors. The time taken by a processor to access any memory word in the system is identical in Uniform Memory Access.
2. The time taken by a processor to access any memory word in the system is differs for NUMA
Advantages:
1. It provides a user-friendly programming environment to access the memory among the processes.
2. Data sharing will be fast and uniform.

Disadvantages: 1. Scalability issue exists. 2.
Synchronization issue also exists if it is not handled properly.

2.Distributed address space machines: In this machine, each processor has its own memory and can access its own memory faster than it can access the memory of a remote processor. Communication among the processors are done by passing message. Messages are used to transfer data among the processes working at different nodes, and also in sharing the work and synchronizing actions between the processes. These machines are commonly referred to as multicomputer.

Advantages: 1.Every processor has its own memory and it can never have overhead of the underlying network and its traffic.
2. Cost effective as the number of processors increases the size of memory increases in proportionate to it.
Disadvantages: 1.Message passing has to be handled carefully and it takes extra effort of coding it for programmers.
2.Based on global memory very hard to map data structures already present.

5 .What is parallel I/O? Why do we need parallel I/O?

A: The basic definition of Parallel I/O is multiple processes or threads of a parallel program accessing data concurrently from a common file. Parallel I/O is a parallel computing that performs multiple input/output operations simultaneously. Rather than process I/O requests serially, one at a time, parallel I/O accesses data on disk simultaneously. So that system to achieve higher write speeds.

With parallel I/O, some of the logical cores on the multicore chip are dedicated to processing I/O. This allows the processor to handle multiple read and write operations concurrently. Parallel I/O helps to eliminate I/O bottlenecks.

Example: AMD and Intel

We needed parallel I/O for not wasting the resources, prevent affecting the other users and we can spend more time on scientific applications. We will get more scalable performance with parallel I/O by allowing more cores.

6.Give an example of antidependence and give a corresponding solution to remove the dependence.

A: Anti-Dependence(Write after Read): Two statements S1 and S2, have anti-dependence between them, if in S1 a value is read first and then the write operation is done on that value in S2.

For example:
```
x = 3;      // S1
y = x * 5; // S2
x = z * 4; // S3
m = x + y; // S4
```

In this example, In order to calculate the value of m , we need to first execute statement 2 before statement 3, else the output will be incorrect. so, Statement 3 has an anti-dependence on Statement 2.

To Remove anti-dependence from above code , Substitute the variable ‘x’ in S3 and S4, with a new variable ‘n’.

7.a)If a sequential search of the tree is performed using the standard depthfirst search (DFS) algorithm, how much time does it take to find the solution if traversing each arc of the tree takes one unit of time?

A: If a sequential search of the tree is performed using the standard depth-first search (DFS) algorithm, they are 11 edges need to be traversed to find the dark node.

Traversing each arc of the tree takes one unit of time, then the running time will be 11.

b) Assume that the tree is partitioned between two processing elements that are assigned to do the search job, as shown in figure b. If both processing elements perform a DFS on their respective halves of the tree, how much time does it take for the solution to be found? What is the speedup? Is there a speedup anomaly? If so can you explain the anomaly?

A: If a sequential search of the tree is performed using the standard depth-first search (DFS) algorithm by two processing element, they are 4 edges need to be traversed to access the dark node.

Traversing each arc of the tree takes one unit of time, then the running time will be 4.

We need to find the speedup.

Speedup = Ts/Tp, where,

Ts =time for the best serial algorithm,

Tp=time for parallel algorithm using p processors

Speedup = 11/4 = 2.75

Speedup is greater than the number of processors (2.75>2) i.e, 2 processors.

There is a speed anomaly, because the work performed by serialization is different from parallelization. There is super linearity due to more efficient utilization of resources by multiprocessor and also due to the better work performed in parallel algorithm.

8.Derive the formula for calculating the average access time for a word in a system with three levels of cache. Assume the following values for a theoretical system containing an L1, L2, and L3 cache.

A: We know that, when processor attempts to read a word from memory, cache is checked first.

CPU requests contents of memory location, then Check cache for this data.

If present, get from cache (fast), then Average memory access time is:

Average memory access time= Cache Hit Time

If not present, read required block from main memory to cache, then deliver from cache to CPU.

Average memory access time = Cache Hit Time*Hit Rate + (1-Hit Rate) x Miss Penalty)

Average memory access time for one level cache ,

⇨ **Average memory access time = h × C + (1 – h) × M**

Where, h = hit rate,C = Access time required to access data from the cache,M = Access time of memory

If data not present in first level cache, read required block from second level cache.

Average memory access time for two level cache = h1C1 + (1-h1)(h2C2 + (1-h2)M))

Where,h1 = Hit rate of first level cache,h2 = Hit rate of second level cache.,C1 = Access time of first level cache,C2 = Access time of second level cache which includes C1,M = Access time of memory which includes C1 and C2.

Similarly, it will read required block from third level cache.

Average memory access time for third level cache = h1C1 + (1 - h1) (h2C2 + (1 - h2) (h3C3 + (1 - h3) M))

Where,h1 = Hit rate of first level cache,h2 = Hit rate of second level cache.,h3 = Hit rate of third level cache,C1 = Access time of first level cache,C2 = Access time of second level cache which includes C1,C3 = Access time of second level cache which includes C1, C2,M = Access time of memory which includes C1, C2 and C3.

We need to find Average memory access time for a memory word :

Given,

h1 – 45% = 0.45

h2 – 75% = 0.75

h3 – 90% = 0.9

C1 = 5ns

C2 = 10ns

C3 = 35ns

M = 100ns

Average memory access time for a memory word = h1C1 + (1 - h1) (h2C2 + (1 - h2) (h3C3 + (1 - h3) M))

= (0.45*5) +(1-0.45)((0.75*10)+(1-0.75)((0.9*35)+(1-0.9)100

= 2.25 + 0.55(7.5+0.25(31.5+10))

= 2.25 + 0.55(7.5+10.375)

= 2.25 +0.55(17.875)

= 2.25+ 9.83

= 12.0812 ns

9. Explain how a queue, implemented in hardware in the CPU, could be used to improve the performance of a write through cache.

A: Queue is the data structure has implemented in hardware in the CPU, because it improves the performance of a write through cache. Since we know that all writes go to main memory as well as cache Multiple CPUs can monitor main memory traffic to keep local cache up to date, it results lots of memory traffic, slow down writes.

Basically queue is First in first out (F IFO) Data structure, it has stored latest written in the cache are on Top. It continuously writes most frequently used data, reduces lot of traffic and increases writes. It results improve the performance of a write through cache. This enables faster I/O performance if an application needs to read the data that was recently written.

10. Recall the example involving cache reads of a two-dimensional array (figure below). How does a larger matrix and a larger cache affect the performance of the two pairs of nested loops? What happens if MAX = 8 and the cache can store four lines? How many misses occur in the reads of A in the first pair of nested loops? How many misses occur in the second pair?

A: We have a 2-dimensional array, but memory is inherently 1-dimensional. Computer will stores it in memory like:

A[0,0] | A[0,1] | A[0,2] | A[0,3] | A[1,0] | A[1,1] | A[1,2] | A[1,3] | A[2,0] | A[2,1] | A[2,2] | A[2,3] | A[3,0] | A[3,1] | A[3,2] | A[3,3].

So, if array size increases, cache must be loaded more data and cache line will no longer to store entire row of matrix .This results running of pair of loops 1 and 2 gets slow down. if the cache size is increases loops, it results increases performance due to the small amount of cache misses.

Cache would be slower because each cache misses large amount of data that would have loaded into cache.

If MAX = 8 and the cache can store 4 lines with each storing 4 elements of the array. Then, the first pair of loops misses 16 cache, 2 for each row of the matrix.

The second pair of loops misses 64 cache because even with the increased number of lines that the cache can hold the second pair of loops still have one cache miss for each element read from the array.

11. Explain why the performance of a hardware multi¬threaded processing core might degrade if it had large caches and it ran many threads.

A: We have seen that the first and second thread can almost double the performance of the core but as the number of threads increases, performance will degraded because all hardware threads are running different software threads with different instructions and different data. If the L1 cache is fully dynamic, one thread can use its major part, it results switching to the next thread will start with instruction and/or data misses causing significant performance degradation.

One improvement is to load L1 caches automatically by the hardware when switching to another thread, but cache line loads takes some time, that the thread switching time becomes longer and multiple processes could be even worse than multiple threads, because multiple processes have separated memory blocks bringing stress on one more shared resource – TLB tables. There are other things to check when considering multithreaded cores. One could be the pipeline implementation, thread switching time and others.

12. a) A planar mesh is just like a toroidal mesh, except that it doesn't have the "wraparound" links. What is the bisection width of a square planar mesh?

A: Let d-dimensional mesh:

N = kd-1 x kd-2 ...x k1 x k0 processes, Described by d-vector of coordinates (id-1 ... i0) – Where 0 < ij ≤ kj

$k = \sqrt[d]{\boxed{N}}$

Bisectional width= pow(k,d-1);

For square planar mesh,d=2

Bisectional width = pow(k,2-1)

=k

$= \sqrt[d]{\boxed{N}}$

$= \sqrt[2]{\boxed{N}}$

The Bisectional width of square planar mesh is Sqrt(N).where N is the number of processes.

b) A threedimensional mesh is similar to a planar mesh, except that it also has depth. What is the bisection width of a three-dimensional mesh?

A: Let d-dimensional mesh and N be the number of processes, N = kd-1 x kd-2 ...x k1 x k0 processes, Described by d-vector of coordinates (id-1 ... i0) – Where 0 < ij ≤ kj

$k = \sqrt[d]{\boxed{N}}$

Bisectional width= pow(k,d-1);

For square planar mesh,d=3

$k = \sqrt[3]{\boxed{N}}$

Bisectional width = pow(k,3-1)

$= K^2$

The Bisectional width of square planar mesh is K^2 ($k = \sqrt[3]{\boxed{N}}$).where N is the number of processes.

13. Modify the trapezoidal rule so that it will correctly estimate the integral even if comm_sz doesn't evenly divide n (you can still assume that n >= comm_sz).

A: We can't use the correct number of trapezoids if comm_sz doesn't evenly divide n .

Let us take example,

If n=15 and comm_sz =4.

Trapezoids for each process= N/comm_sz

=15/4

=3

The total number of trapezoids that processes use is 4*3=12 Trapezoids. Here, we are not using the remaining three trapezoids. In order to use 15 trapezoids, we need to assign 4 trapezoids for each process. So, we need to modify the trapezoidal rule so that it will correctly estimate the integral even.

All the processes will get N/comm_sz trapezoids, while n%comm_sz will get ann extra trapezoidal.

Let q be the quotient,r be the remainder and n be the number of trapezoids.

```
int q,r,n,comm_sz, local_n;;

double local_a, local_b;

q = n / p;

 r = n %p;

 if(my_rank< r)

{       local_n = q+ 1;

        local_a = a + my_rank*local_n*h ;

        local_b = local_a + local_n*h;

}

 else

 { local_n = q;
```

```
  local_a = a + my_rank*local_n*h + r*h;
  local_b = local_a + local_n*h
}
```