Assignment #1
CS5379: Parallel Processing

Name: Ashritha Puradamane Balachandra

R number: R11613952

1.

Rank	System	Rmax (TFlop/s)	Rpeak (TFlop/s)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	148,600.0	200,794.9
2	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	94,640.0	125,712.0
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	93,014.6	125,435.9
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	61,444.5	100,678.7
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	23,516.4	38,745.9
6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray/HPE Swiss National Supercomputing Centre (CSCS) Switzerland	21,230.0	27,154.3
7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray/HPE DOE/NNSA/LANL/SNL United States	20,158.7	41,461.2

Rank	System	Rmax (TFlop/s)	Rpeak (TFlop/s)
8	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	19,880.0	32,576.6
9	**SuperMUC-NG** - ThinkSystem SD650, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path , Lenovo Leibniz Rechenzentrum Germany	19,476.6	26,873.9
10	**Lassen** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Tesla V100 , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	18,200.0	23,047.2

2. Two examples of "parallel processing" are:
    - multiple checkout lines in supermarket
    - multiple query response in call center.

3. The two reasons that I would argue for having parallel processing are:
    - To solve grand challenging problems that cannot be solved easily by a single person or computer
    - To reduce the time consumption (by doing millions of transactions per second).

4. **SISD (Single Instruction Single Data stream):** It represents a serial Von Neumann machine. only one instruction stream is being executed by the CPU during any one clock cycle (single control processor). only one data stream is being used as input during any one clock cycle. Examples: older generation mainframes, minicomputers and workstations.

   SIMD (Single Instruction Multiple Data stream) : It is suitable for problems with high degree of regularity such as graphics/image processing. All processing units execute the same instruction at any given clock cycle. Each processing unit operate on a different data element. It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an "activity mask", which determines if a processor should participate in a computation or not. Examples: IBM 9000, Cray X-MP, Y-MP &C90

   MISD (Multiple Instruction Single Data stream) : A single data stream is fed into multiple processing units. Each processing unit operates on the data independently via independent instruction streams. Few actual examples of this class of parallel computer have ever existed.

MIMD (Multiple Instruction Multiple Data stream) : Most common type of parallel computer. Every processor may execute different instruction stream. Examples: IBM SP, TMC's CM-5, Cray T3D & T3E, SGI Origin, Tera MTA

SPMD (Single Program, Multiple Data) : It is a variant of MIMD, executes the same program (multiple instances) on different processors. It is widely used by many parallel platforms. Example: executions of MPI (message passing interface) programs

5. **UMA (Uniform Memory Access):** It is a type of share-address-space machine. Processors in this architecture have same uniform access latency when accessing memory.

   NUMA (Non-Uniform Memory Access): It is also a type of share-address-space machine. Processors in this architecture have different access latency when accessing memory. NUMA machines require locality from underlying algorithms for better performance, as a processor access data from a close memory would be fast than accessing data from remote memory.

   ccNUMA (cache coherent Non-Uniform Memory Access): It is a type of NUMA machine. Cached data copies in this type of machine are consistent with each other.

   DSM (Distributed Shared Memory): It is also a type of share-address-space machine, as even though physical memory is distributed, but either a hardware or a software solution provides logical shared-address-space from distributed memory. Particularly, DSM is a NUMA machine, as memory is distributed, and processors have different access latency when accessing memory.

6. Shared-address-space:

   Advantages:

   - It provides Global address space view for programmers, easier to program usually.
   - They have implicit communication.

   Disadvantages:

   - They need hardware/software to support view and hence there exist complexity in system design.
   - Not easy to be scalable.

   Distributed-address-space:

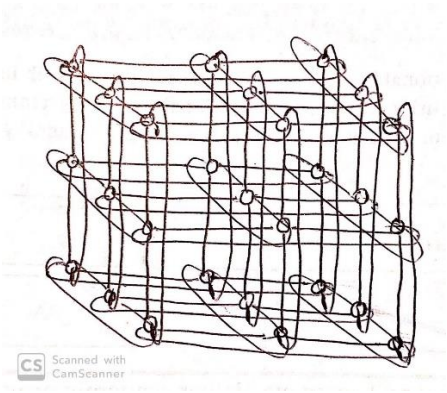
   Advantages:

   - Requires little hardware support other than a network
   - Easy to scale up

   Disadvantages:

   - Distributed-address-space parallel computers are harder to program because of their lack of a global address-space view.
   - They need explicit communication.

7. 3D-torus with 27 nodes:



8. Based on how a *d*-dimensional hypercube is constructed, we say d-that dimensional hypercube can be constructed by connecting corresponding nodes of (d-1)- dimensional hypercube and we always get even number of nodes in d-dimensional hypercube except 0-dimensional hypercube.
Also, based on how the binary number of the processor label changes when traversing through the hypercube network, we can say that from one node (processor) to its connected node - number of ones in the label should change by one binary bit. So, in cycle pattern the number of changes occurred should be even. Therefore, **there are no odd-length cycles in a *d*-dimensional hypercube**.
Example: 2-D hypercube is constructed by connecting two 1-D hypercube and its processors are labeled as 00,01,10 and 11. Any connected pair of nodes changes its bit by only one. Cycle pattern is created when we end the traverse of the nodes at the begin node. So, we will get 4 links for cycle pattern in a 2-D hypercube.

9. 1. **Bisection width**: When the mesh with p processors is required to divided into two equal parts of p/2 processors, the minimum number of links required to cut is $\sqrt{p}$. Therefore, bisection width of a mesh tree is $\sqrt{\mathbf{p}}$.

   2. **Diameter**: The diameter is the maximum distance between any two processing nodes. The corner processors that are located diagonally need maximum number of links to communicate.

   We know that in tree topology, the distance between any 2 nodes is no more than 2 log p.----------(1)

   Also given that fig. (b) and (c) is constructed on $\sqrt{p} \times \sqrt{p}$ grid----------(2)

   From (1) and (2), distance between diagonally corner nodes is $2 \log \sqrt{p} + 2 \log \sqrt{p} = 2 \log p$

   Therefore, diameter is **2log p**.

   3. **Number of Switches**: Since a complete binary tree is imposed on each row and each column of the grid of $\sqrt{p} \times \sqrt{p}$, each row has $(\sqrt{p}-1)$ switches and each column has $(\sqrt{p}-1)$ switches.

   Total number of Switches are:

   $p(\sqrt{p}-1) + p(\sqrt{p}-1)$

$= 2\sqrt{p}(\sqrt{p}-1)$

$=2(p-\sqrt{p})$

Therefore, total number of switches is **$2(p-\sqrt{p})$.**