Q1.

1. What is the impact of memory latency and bandwidth and how can we reduce it?
   Processor waits for fetching operand from memory (memory latency) and in case of accessing consecutive location memory operation are serialized due to limited bandwidth. Placing a cache (cache line size = 1 word) solves the memory latency problem. Memory bandwidth is increased by pre-fetching many consecutive words in cache (cache line size = 4 or 8 words).

2. Explain through an example how granularity of tasks may affect overall performance?
   In fine-grain case communication cost determines performance as many smaller computations (tasks) communicates with other tasks. Large frequency of communication between tasks is needed as each dependent task requires results (data) from previous tasks. High-speed network will improve performance. In course-grain case, computation cost per nodes determines performance as there is large task (more computation per task) and less communication (such as in reduction). High-ended super-scalar super-pipelined processor (which may use built-in and motherboard GPU) will improve performance.

3. How rescheduling/pipelining/superscalar / execution does improves the performance? Explain with example.
   A bunch of mathematical computation with a mix of floating point and integer operations are ideally suited for this architecture where multiple integer and floating point ALUs, and 8-10 pipelines can be used with large instruction cache and in-order or out-of-order scheduling to speed up execution. For example, giving 10 instructions per cycle instead of one.

4. A quick sort algorithm of a large array. Is it a data parallelism or task parallelism?
   Quicksort can be implemented iteratively or recursively. It partition data into two sets by selecting the pivot. Both implementations are possible. Marks based on written argument.

5. You need to perform a word count of a file size = 10GB. What do you think a shared memory system or distributed memory system will increase performance.
   In distributed memory, a master process on one node need to distribute suitable chunks of 10GB data to other n-1 nodes and collect computed results. If the file is already distributed on these N noes (see discussed in class example) this distribution can be avoid and only word count of individual chunks on nodes need to be collected. **Communication overhead is critical to performance.** In shared memory, the system should have sufficient RAM (shared memory) to hold 10GB data from the file otherwise performance can be lost due to excessive page from disk. **The configuration of shared memory system is critical to performance.**

Q2(a).

```
// corrections: REAL -> double and y[n] -> c[n]
// sum declared as shared.
double A[n][n], b[n], c[n], sum;
int i, j;
// we need to distribute a chunk of 100 iterations of upper for loop to each threads
// There will be 10 or more thread to distribute work of for loop
#pragma omp parallel for default (shared), private (sum,i,j), num_threads (10)
for (i=0; i<n;i++) {
   sum = 0.0; // private to each thread and initilized to zero for each i
   for (j=0; j<n; j++) sum += A[i][j] * b[j]; // no data race here
   c[i] = sum;
}
```

Q2(b).

In this setup, peak processor is 2GFLOPS, and reading from 100nsec memory makes if 10MFLOPS. As the question state 2 instructions per cycle instead of 4 as shown in the slides. Multiplication will take 32usec instead of 16usec. This means 64k/232 = 282.5 MFLOPS. So if a cache is placed between CPU and Memory which caches all three matrices (two given and third resultant) the performance will be increased from 10MFLOPS to 282.5MFLOPS.

Impact of Caches: Example

Consider the architecture from the previous example. In this case, we introduce a cache of size 32 KB with a latency of 1 ns or one cycle. We use this setup to multiply two matrices A and B of dimensions 32 × 32.

We have carefully chosen these numbers so that the cache is large enough to store matrices A and B, as well as the result matrix C.

```
for(i=0;i<r;i++) {                    O (N³)
  for(j=0;j<c;j++) {
    for(k=0;k<c;k++) {
      mul[i][j]+=a[i][k]*b[k][j];
  }}}
```

Action items:

1. Draw processor, cache and memory
2. Draw Cache of size 32 K
3. How three matrices of size 32 x 32 fits into this cache?
4. How cache data is reused during these calculations?

Impact of Caches: Example (continued)

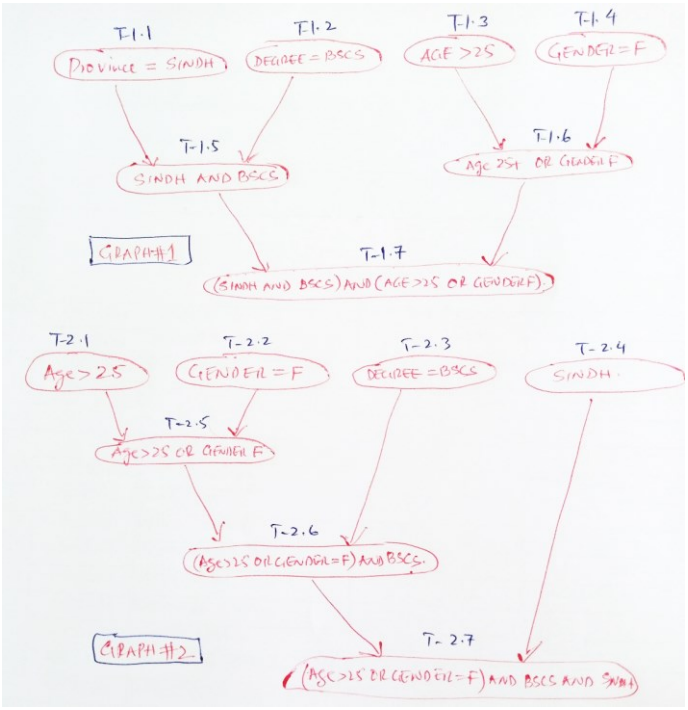
- The following observations can be made about the problem:
  - Fetching the two matrices (n x n = 32 x 32) into the cache corresponds to fetching 2K words, which takes approximately **200 µs**. HOW?
  - Multiplying two $n \times n$ matrices takes $2n^3$ operations. For our problem, this corresponds to 64K operations, which can be performed in 16K cycles (or 16 us) at four instructions per cycle. $n^2(n+(n-1)) = 2n^3 - n^2 = O(n^3)$.
  - The total time for the computation is therefore approximately the sum of time for load/store operations and the time for the computation itself, i.e., (200 + 16) µs.
  - This corresponds to a peak computation rate of 64K/216 or 303 MFLOPS.

A thirty-fold improvement over the previous example. However, it is still less than 10% of the peak processor performance. By placing a small cache memory, we are able to improve processor utilization considerably.

Q3(a).

- Graph # 1: Task 1.1, 1.2, 1.3 and 1.4 are independent and can be executed on processors P1, P2, P3 and P4 respectively.
- Graph # 1: If T 1.5 and T1.6 are executed on P1 and P3 respectively, they only need results on T1.2 and T1.4 from P2 and P4 respectively.
- Graph #1: If T 1.7 is executed on P1 it only need T 1.6 results on P3.

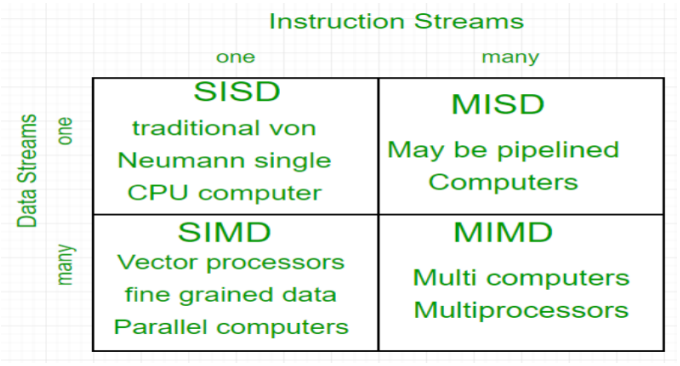
Same from Graph #1.



Q3(b).

Examples are:

- SISD -> Von Neumann single CPU computer
- SIMD -> Vector Processors
- MISD -> Pipelined Computers
- MIMD -> Modern Computers



----------(X)----------