Jiaxing Wu
CS451
September 4, 2019

1. The two main types of parallelism are data parallelism and task parallelism.

   For data parallelism, the program is split into groups by data and each group would be processed on separate cores.

   ---

   For example: You have ten cores
   Program:

   ```
   for i= 0; i < 10; i ++
   A[i] = b[i] + c[i]
   ```

   Split the program into ten parts with each part running on different cores
   Core0 → A[0] = b[0] + c[0]
   Core1 → A[1] = b[1] + c[1]
   ...
   Core9 → A[9] = b[9] + c[9]

   ---

   For task parallelism, the program is split into groups by functions/tasks and each task would be processed on separate cores.

   ---

   For example. You have two cores
   Program:

   ```
   Function 1:
       for i= 0; i < 10; i ++
       A[i] = b[i] + c[i]

   Function 2:
       for i= 0; i < 10; i ++
       d[i] = e[i] - f[i]
   ```

   Split the program by functions and run each function on each core
   Core0 → run function 1
   Core1 → run function 2

   *Data parallelism supports no dependency and some dependency(there are ways around it) between data. Task parallelism ONLY supports no dependency between data

2. The three parallel programming paradigms are shared memory, message passing, and data parallel.

   The shared memory model emphasises on sharing memory just like the name implies. It is the idea that all processors should have access to all the memory. There are two types of memory sharing: shared memory and distributed shared memory.

   -Shared memory: All processor has access to one big memory module
   -Distributed shared memory: All processors has its OWN private memory module.
   ---
   The message passing model is when processors/nodes communicate via explicit send/receive pairs with other processors. The private memory between each processor/node is not shared. When a processor needs data from another processor, they would communicate via send/receive pairs.

   Example of message passing model for processors: NUMA
   Example of message passing model for nodes: cluster

   ---
   Data parallel is when you divide the program into different group by data and each group is processed by a separate core. (This is the same explanation as Problem 1 with data parallelism.)

3. The difference between shared memory and distributed shared memory is the space of memory that each is allowed to access.

   -Shared memory: All processor has access ALL memory
   Advantage: Takes less money to make, simple
   Disadvantage: Slow because all processors have to access memory from the same place

   -Distributed shared memory: All processors has its only have access to its OWN private memory
   Advantage: faster than shared with less communication between processors
   Disadvantage:Takes more money to make

4. Anti Dependence Example:
   Given there is a global value x:
   x= 0

   Line 1: printf(x)

Line 2: x= 9

If the code is parallelized, Line 2 has anti dependence on Line 1 because it writes the value that Line 1 reads. It should print 0 but it can also print 9. The can be solved if Line 2 waits for line 1 to finish reading x.

5. a) time = 19
   b) time = 5, speedup = 19/5 = 3.8; The anomaly is that the speedup is greater than the amount of processors. This shows that the speedup is a super linear speedup. It is a super linear speedup because parallelizing DFS kinda makes it behave like a binary search.

6. Principle → $T_{avg}$ = hit time + miss rate * miss penalty
   *miss penalty = hit time of next level + miss rate of next level * miss penalty of next level
   It is a recursive formula until it hits a Last level cache(LLC) or Main memory in this case.

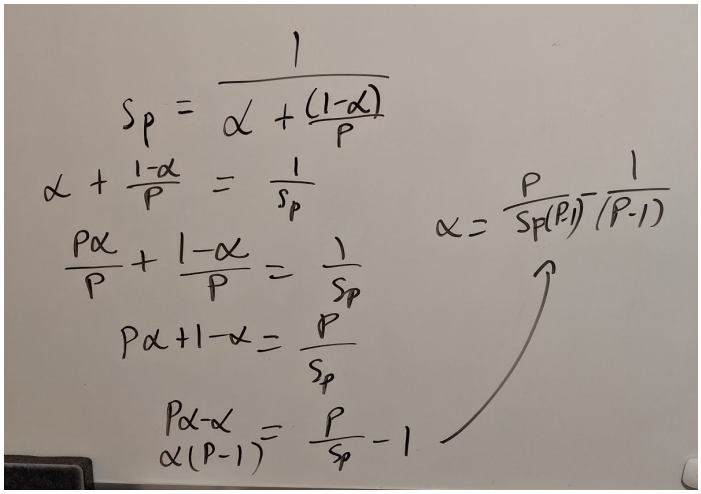
   $$T_{avg} = (P_{L1})(L_{L1}) + (1 - P_{L1})((P_{L2})(L_{L2}) + (1 - P_{L2})((P_{L3})(L_{L3}) + (1 - P_{L3})((P_{MM})(L_{MM}))))$$
   *L= latency, P=hit rate

   $T_{avg}$ = (0.4)(2ns) + (0.6)((0.7)(8ns) + (0.3)((0.9)(30ns) + (0.1)((1)(100ns))))
   $T_{avg}$ = 10.82ns

7. a) $t_s = n * m * 2$
   $t_s = 3 * 3 * 2$
   $t_s = 18$

   b) $t_p = m * p * 1 + (n*m*2)/p$
   $t_p = 3 * 3 + (3*3*2)/3$
   $t_p = 15$

   c) $S_p = t_s/t_p = 18/15 = 1.2$
   $S_p = 1/((\alpha + (1-\alpha)/p))$




$\alpha = \frac{p}{S_p(p-1)} - \frac{1}{p-1}$

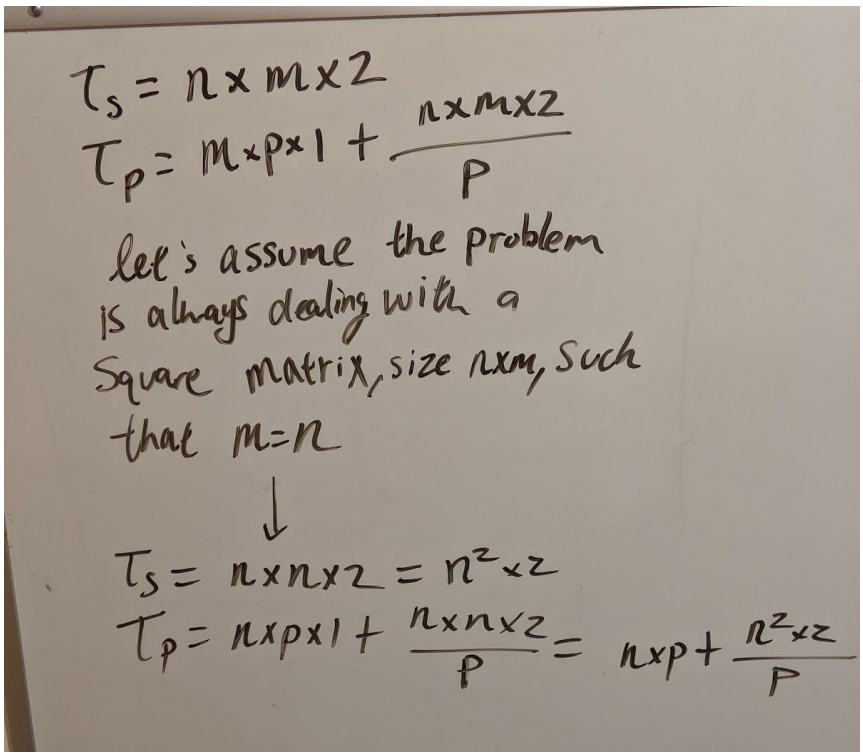
$\alpha = \frac{3}{1.2(3-1)} - \frac{1}{3-1}$

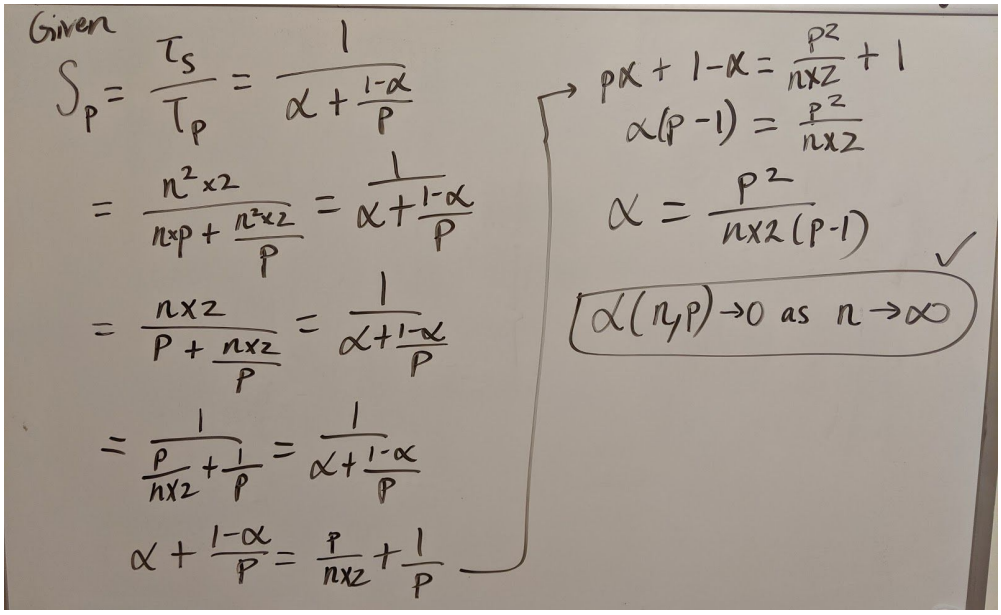
$\alpha = \frac{3}{1.2(2)} - \frac{1}{2}$

$\alpha = \frac{3}{2.4} - \frac{1}{2}$

$\alpha = \frac{3}{4} = 0.75$

d)






According to Amdahl's Law, the algorithm is efficient because **$\alpha(n, p) \to 0$ as $n \to \infty$** is true.