Homework 2

Md Ali A20439433
CS 546: Parallel and Distributed Processing

October 2, 2021

Exercise 1. Suppose a shared-memory system has two cores, core 0 and core 1, both have the variable x in their cache. After core 0 executes the assignment $x = 5$, core 1 tries to execute $y = x$. What value will be assigned to y? Why? Can you provide a solution to fix the problem?

Proof. Since we have a shared memory system this depends if this is write-through or write-back, either way we are assuming there is no cache coherency protocol, so when core 1 tries to execute $y = x$, the value of y could be anything that was in the cache of core 1 or some memory address where x was in core's 1 cache. Either way, the issue here is that $y \neq 5$ when core 1 executes $y = x$ as when core 0 does an assignment operation on $x = 5$, we won't know if the update comes to core 1's cache. This is due to that there is no cache coherency amongst the system so whatever value x has in it's cache will be assigned to y. To remedy this we need to implement a cache coherency protocol of some sort. There is one of two ways that we can assure this for when core 0 executes the assignment $x = 5$, (1) the other copies of x in the system must be invalidated also known as an invalidate protocol, or (2) the other copies must be updated also known as an update protocol. Some popular implementation of coherence protocols that could be used in snoopy systems, directory based systems, or a combination of both of these, either of these protocols would suffice in solving the issue and insure that $y = x = 5$ to be true.

□

Exercise 2. Consider a memory system with a level 1 cache of 32 KB and DRAM of 512 MB with the processor operating at 1 GHz. The latency to L1 cache is one cycle and the latency to DRAM is 100 cycles. In each memory cycle, the processor fetches four words (cache line size is four words). What is the peak achievable performance of a dot product of two vectors? Note: Where necessary, assume an optimal cache placement policy.

/* dot product loop */
for($i = 0; i < dim; i++$)
dot_prod $+= a[i] * b[i]$;

Proof. Let's consider each of the given information that we are given.

L1 cache 32 KB and latency of 1 cycle
DRAM 512 MB and latency of 100 cycles
Frequency on the processor 1 GHz or $1 \cdot 10^9$ Hz
Cache line is in the size of 4 words

Let's first figure out the time per cycle for the L1 cache. We will use the formula below.

$$Time\,per\,Cycle = \frac{Latency}{Frequency} = \frac{1}{1 \cdot 10^9} \rightarrow 1ns$$

Hence the L1 cache takes $1ns$, we will do the same for the DRAM.

$$Time\,per\,Cycle = \frac{Latency}{Frequency} = \frac{100}{1 \cdot 10^9} \rightarrow 100ns$$

Hence the DRAM takes $100ns$, now we can find the peak performance of the dot product of two vectors. We know each iteration involves 2 FLOPS and we know that the processor fetches four words, so we have a total of 8 FLOPS.

$$Peak\ Performance = \frac{8FLOPS}{\frac{2 \cdot 100}{1 \cdot 10^9}} = \frac{8 \cdot 10^9}{200} = 4 \cdot 10^7 FLOPS \rightarrow 40\,MFLOPS$$

Hence the peak achievable performance of a dot product of two vectors of this fashion is 40 MFLOPS.

□

Exercise 3. Now consider the problem of multiplying a dense matrix with a vector using a two-loop dot-product formulation. The matrix is of dimension 4K x 4K. (Each row of the matrix takes 16 KB of storage.) What is the peak achievable performance of this technique using a two-loop dot-product based matrix-vector product?

```
/* matix-vector product loop */
for($i = 0; i < dim; i++$)
for($j = 0; j < dim; j++$)
$c[i] += a[i][j] * b[j];$
```

Proof. Utilizing the same logic as we used in exercise 2, we can see that we have a matrix element instead of single row, so we have 16 words now. Nonetheless, the same technique is utilized but we have an added second loop so now it will onyl take one word to fetch from memory. In this our case for the matrix this means that every 4K entry, so it can only fetch 4 words at a time. Each iteration involes 2 FLOPS but since each row of the matrix take 16KB of storage the flops will be $2 \cdot 16FLOPS = 32FLOPS$ We can see this in the equation below

$$Peak\ Performance = \frac{32FLOPS}{\frac{4 \cdot 4 \cdot 100}{1 \cdot 10^9}} = \frac{32 \cdot 10^9}{1600} = 2 \cdot 10^7 FLOPS \rightarrow 20\,MFLOPS$$

Hence the peak achievable performance of a dot product of the matrix in this fashion is 20 MFLOPS.

□

Exercise 4. Enumerate the critical paths in the decomposition of LU factorization shown in the following figure (textbook figure 3.27).

Proof. A critical path is denoted by the longest directed path between any pair of start and finish nodes. We are asked to enumerate the critical paths in the decomposition of LU factorization from their tasks and we will be assuming that all the tasks have the same weight. The following sequences of tasks are shown below for each critical path taken.

Critical Path 1: 1,2,6,10,11,13,14
Critical Path 2: 1,2,6,10,12,13,14
Critical Path 3: 1,2,8,10,11,13,14
Critical Path 4: 1,2,8,10,12,13,14
Critical Path 5: 1,3,7,10,11,13,14
Critical Path 6: 1,3,7,10,12,13,14
Critical Path 7: 1,3,9,10,11,13,14
Critical Path 8: 1,3,9,10,12,13,14
Critical Path 9: 1,4,6,10,11,13,14
Critical Path 10: 1,4,6,10,12,13,14
Critical Path 11: 1,4,7,10,11,13,14
Critical Path 12: 1,4,7,10,12,13,14
Critical Path 13: 1,5,8,10,11,13,14
Critical Path 14: 1,5,8,10,12,13,14
Critical Path 15: 1,5,9,10,11,13,14
Critical Path 16: 1,5,9,10,12,13,14

These are all the critical paths as they all have the longest directed path from the start and end node, in our case task 1 to 14. Here, assuming all the nodes have equal weight, there is 16 critical paths.

□

Exercise 5. Show an efficient mapping of the task-dependency graph of the decomposition shown in the above figure (textbook figure 3.27) onto three processes, four processes. Prove informally that your mapping is the best possible mapping for three processes.

Proof. Below is the task-dependency graph of the decomposition of figure 3.27 of the textbook. We will discuss each step and the corresponding task in figure 1.

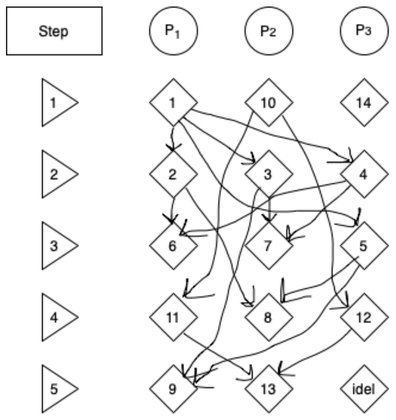

Figure 1: Three processes

Step 1: Tasks 1, 10, and 14 must start. Task 1 has four children with each of them having two dependencies, task 10 has two children, where they each just share one dependency. Lastly, task 14 has no dependencies.

Step 2: Do task 2, 3, and 4, since they have the most dependencies. We can't do task 5 since we only have 3 processors in this case so we will have to push that to step 3.

Step 3: Do task 5, 6, and 7. Task 5 was pushed on this step due to not having enough enough processes available. Task 6 and 7 will complete two paths.

Step 4: Do tasks 11 and 12 since they have a dependency and then task 8 to complete two paths.

Step 5: Do task 9 and task 13 to complete two paths each and have one processor idle.

The next part we will showcase the same issue but with four processors, much of the same logic is essentially used and described below the graph but we will have four steps instead of five which we can see in figure 2.

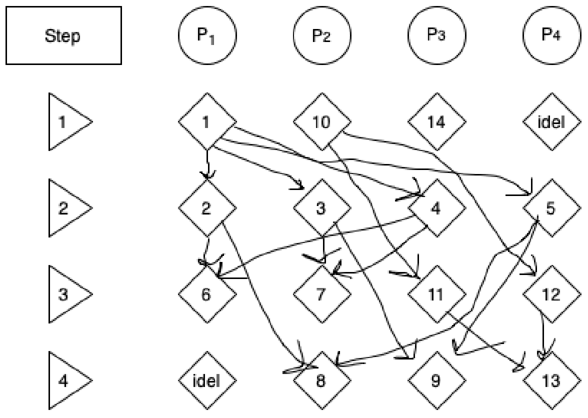

Figure 2: Four processes

Step 1: Do task 1, 10, 14 and the last processor is idle because we have to start with these three tasks. Task 1 has four children, each with two dependencies. Task 10 has two children that both have one dependency together. Lastly, task 14 has no dependencies.

Step 2: Do task 2, 3, 4, and 5 because they have the most dependencies each.

Step 3: Do task 6 and 7 to complete two paths and then do task 11 and 12 since they have one dependency.

Step 4: Processor 1 will be in idle but do task 8 and 9 to complete two paths, and 13 to complete the other path.

□

Exercise 6. For the task graphs given in the following figure, determine the following:

a) Maximum degree of concurrency.
b) Critical path length.
c) Maximum achievable speedup over one process assuming, that an arbitrarily large number of processes is available.
d) The minimum number of processes needed to obtain the maximum possible speedup.
e) The maximum achievable speedup if the number of processes is limited to 2, 4, and 8.

Proof. We will assume that all the weights are equal and take each graph one at a time as shown below.

Graph A:
a) Maximum degree of concurrency: 8
b) Critical path length: 4
c) Maximum achievable speedup: $\frac{15}{4}$
d) Minimum number of processes for max speedup: 8
e) 2 processors $\rightarrow \frac{15}{8}$; 4 processors $\rightarrow \frac{15}{5} = 3$; 8 processors $\rightarrow \frac{15}{4}$

Graph B:
a) Maximum degree of concurrency: 8
b) Critical path length: 4
c) Maximum achievable speedup: $\frac{15}{4}$
d) Minimum number of processes for max speedup: 8
e) 2 processors $\rightarrow \frac{15}{8}$; 4 processors $\rightarrow \frac{15}{5} = 3$; 8 processors $\rightarrow \frac{15}{4}$

Graph C:
a) Maximum degree of concurrency: 8
b) Critical path length: 7
c) Maximum achievable speedup: $\frac{14}{7} = 2$
d) Minimum number of processes for max speedup: 3
e) 2 processors $\rightarrow \frac{7}{4}$; 4 processors $\rightarrow 2$; 8 processors $\rightarrow 2$

Graph D:
a) Maximum degree of concurrency: 8
b) Critical path length: 8
c) Maximum achievable speedup: $\frac{15}{8}$
d) Minimum number of processes for max speedup: 2
e) 2 processors $\rightarrow \frac{15}{8}$; 4 processors $\rightarrow \frac{15}{8}$; 8 processors $\rightarrow \frac{15}{8}$

□

Exercise 7. We have introduced the Parallel Partition LU Tridiagonal (PPT) algorithm in class (see slide 16 of Lecture 9). An algorithm count and communication count analysis result of the PPT algorithm are also presented in Lecture 9. Please provide a detailed, step-by-step algorithm count and communication count analysis of the PPT algorithm to reach the result or reach your own analysis result.

Proof. My best effort on this is on the rest of the document from my iPad.

□

Exercise 8. We have introduced the Parallel Diagonal Dominant (PDD) algorithm in class (see slide 30 of Lecture 9). An algorithm count and communication count analysis result of the PDD algorithm are also presented in Lecture 9. Please provide a detailed, step-by-step algorithm count and communication count analysis of the PDD algorithm to reach the result or reach your own analysis result.

Proof. My best effort on this is on the rest of the document from my iPad.

□