Ira A. Fulton Schools of Engineering
School of Computing, Informatics and Decision Systems Engineering

SER 450 **COMPUTER ARCHITECTURE**

Your Name: Jacob Hreshchyshyn

When completing this worksheet, please use a different color text or typeface so that we can easily identify your work. Remember to show your work / give justification for your answers.

Portions of this material are derived from "Computer Organization and Design", Patterson & Hennessy.

PRACTICE PROBLEMS 6

Problem 1:

Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value X in a sorted N-element array A and returns the index of the matched entry:

```
BinarySearch(A[0..N-1], X) {
   low = 0
   high = N - 1
   while (low <= high) {
      mid = (low + high) /2
      if (A[mid] > X)
         high = mid -1
      else if (A[mid] < X)
         low = mid + 1
      else
         return mid  // found
   }
}
```

A. Assume that you have Y cores on a multi-core processor to run BinarySearch. Assuming that Y is much smaller than N, express the speedup factor you might expect to obtain for values of Y and N without refactoring the code.

This problem appears to be another application of the example from page 506 of the book. First, we should consider Amdahl's Law, which says the following:

Execution Time After Improvement = ((Execution Time Affected By Improvement)/Amount of Improvement) + Execution Time Unaffected.

We also need to consider what portion of the code is parallelizable. Given that we are not to refactor this code, and given that no pthreads exist to properly partition instructions to be parallelized, there would be no opportunities to parallelize any of the instructions in this code. Therefore, whether Y < N or Y = N, I would expect to see no speedup factor greater than 1.

B. Next, assume that Y is equal to N. How would this affect your conclusions on the previous answer? If you were tasked with obtaining the best speedup factor possible (i.e. strong scaling), explain how you might change this code to obtain it.

In the previous answer, I already indicated that the fact that no pthreads exist in this code would prevent any speedup factor greater than 1 since no portion of this code is parallelizable. Therefore, the code ought to be refactored to incorporate those pthreads. That way, a strong scaling solution could be implemented to supply cores to work on the problem while the problem size stays mostly fixed relative to the number of cores.

Problem 2:

Consider the following piece of C code:

```
for (j=2; j<1000; j++)
   D[j] = D[j - 1] + D[j - 2];
```

The MIPS code corresponding to the above fragment is:

```
      addiu $s2, $zero,7992
      addiu $s1, $zero,16
loop:
      l.d   $f0, -16($s1)
      l.d   $f2, -8($s1)
      add.d $f4, $f0,$f2
      s.d   $f4, 0($s1)
      addiu $s1, $s1, 8
      bne   $s1, $s2, loop
```

Instructions have the following associated latencies (in cycles)

add.d	l.d	s.d	addiu
4	6	1	2

A. How many cycles does it take for all instructions in a single iteration of the above loop to execute?

To find the number of cycles required to execute a single iteration, simply add up each number of cycles corresponding to the instruction. The result would be the following:

6 + 6 + 4 + 1 + 2 = 19 cycles.

B. When an instruction in a later iteration of a loop depends upon a data value produced in an earlier iteration of the same loop, we say that there is a loop carried dependence between iterations of the loop. Identify the loop carried dependencies in the above code. Identify the dependent program variable and assembly-level registers. You can ignore the loop variable j.

Looking at the C code, it's clear that D[j] depends on D[j – 1] and D[j – 2], both of which, after the first couple of iterations, are produced by earlier loop iterations. Looking at the assembly code, we can see that registers $f0 and $f2 store D[j – 2] and D[j – 1], respectively. These registers contain the values carried over from previous iterations. $f4 then contains D[j], which is dependent on registers $f0 and $f2.

C. **Loop unrolling was described in chapter 4. Apply loop unrolling to this loop and then consider running this code on a 2-node distributed memory message passing system. Assume that we are going to use message passing as described in the text, where we introduce a new operation send(x,y) that sends to node x, the value y, and an operation receive() that waits for the value being sent to it. Assume that the send operations take a cycle to issue (i.e. later instructions on the same node can proceed in the net cycle), but take 10 cycles to be received on the receiving node. Receive instructions stall execution on the node where they are executed until they receive a message. Compute the number of cycles it will take for the loop to run on the message passing system.**

It is unclear to me how to calculate the number of cycles it will take for the loop to run on the message passing system. However, given that distributed memory passing often results in worse performance, I would expect the latencies to increase despite the fact that the loop is now unrolled.

D. **The latency of the interconnect network plays a large role in the efficiency of message passing systems. How fast does the interconnect latency need to be in order to obtain a speedup from using the distributed system form the previous part of this problem?**

Again, this is difficult for me to determine given that I did not numerically define the answer in part C. However, the interconnect latency will likely need to be quite fast, though not as fast as would have been needed without the unrolled loop.

Problem 3:

Assume a quad-core computer system can process database queries at a steady state rate of requests per second.

A. **If we move to an 8-core system, ideally, what will happen to the system throughput (i.e. how many queries/second, will the computer process)?**

While there is no list of speeds of transaction latencies and processing rates, I would think that the throughput should increase by a factor of 2 due to the doubling of available processor cores.

B. **Discuss why we rarely obtain this kind of speedup by simply increasing the number of cores**

This kind of speedup is rare since there still exist bandwidth limitations in the amount of data that can be transferred. Additional data decomposition is needed to obtain better speedups.