On Multiplying Polynomials (OMP) (DAVE)

`void multiply1(double *A, double *B, double *C, int n) { // Outer Loop for (int i=0; i < n; i++) // Inner Loop for (int j=0; j < n; j++) C[i+j] += A[i] + B[j]; }`	`void multiply2(double *A, double *B, double *C, int n) { // Outer Loop for (int i=0; i < 2*n; i++) // Inner Loop for (int j = MAX(i - n + 1, 0); j < MIN(i + 1, n); j++) C[i] += A[j] + B[i-j]; }`

```
void multiply1(double *A, double *B,
          double *C, int n) {
   // Outer Loop
   for (int i=0; i < n; i++)
      // Inner Loop
      for (int j=0; j < n; j++)
         C[i+j] += A[i] + B[j];
}
```

```
void multiply2(double *A, double *B,
          double *C, int n) {
   // Outer Loop
   for (int i=0; i < 2*n; i++)
     // Inner Loop
      for (int j = MAX(i - n + 1, 0);
            j < MIN(i + 1, n); j++)
         C[i] += A[j] + B[i-j];          }
```

a) Suppose OpenMP pragmas are placed on the outer and inner loops according to the table below. Evaluate if multiply1 and multiply2 are guaranteed to return the correct answer with the given pragma placements. Circle your answer in each of the rightmost six cells. A, B, and C point to non-overlapping memory regions.

1 POINT PER CORRECT ANSWER

Outer Loop	Inner Loop	multiply1	multiply2
<leave empty>	parallel for	[✓ *correct*] *incorrect*	[✓ *correct*] *incorrect*
parallel for	<leave empty>	*correct* [✓ *incorrect*]	*correct* [✓ *incorrect*]
parallel for	parallel for	*correct* [✓ *incorrect*]	*correct* [✓ *incorrect*]

b) Suppose that we execute the multiply2 in the following configuration. We have parallelized only the outer loop, our machine has 64B cache lines, is running 8 threads, and n = 1024. In scheduling scheme A we allocate indices to threads round robin (ie, thread *p* takes care of indices *p*, *p* + 8, *p* + 16, etc). In B we allocate work in contiguous chunks. Which configurations exhibit false sharing?

2 POINTS FOR CORRECT ANSWER

- A and B
- [✓ A only]
- B only
- Neither A nor B

c) Circle all statements that are **true** for both multiply1 and multiply2 unless otherwise noted.

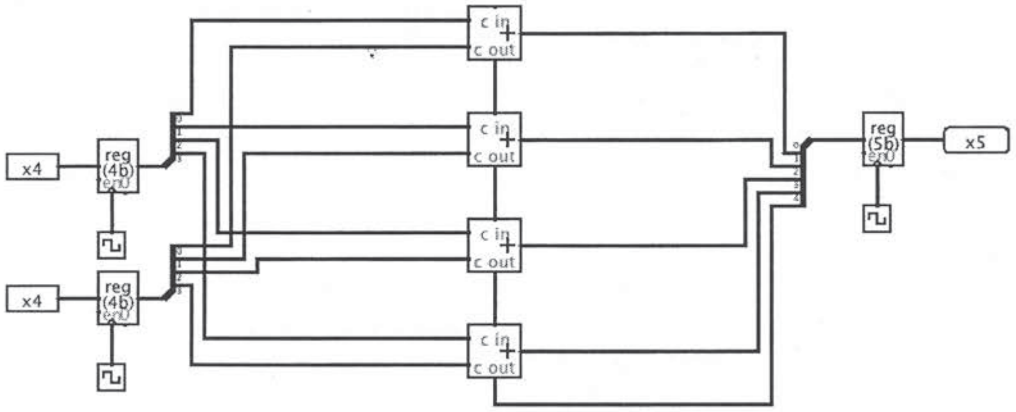
- Threads can see different "i" values even if the parallel for is only on the inner loop.
- A #pragma omp barrier on the innermost statement will fix all data races from part (a).
- Ignoring correctness, parallelizing both inner & outer loops will give a speedup for large n.
- [✓ The synchronization overhead of parallelizing an inner loop with a given number of threads is amortized for large n.]

2 POINTS FOR CORRECT ANSWER
-1 POINT PER INCORRECT ANSWER
MINIMUM SCORE IS 0

14 points Total

Additional Logic (IAN)

For this problem we will look at a simple carry bit adder. Each adder block is simply a one bit adder. This circuit will add a series of pairs of 4 bit inputs and output 5 bit outputs (instead of having an overflow bit). The inputs will change once each clock cycle.



2 points

a) We want to pick the maximum clock frequency such that we can guarantee this circuit works correctly. The adder block has a delay of time *a*. The registers have a clock-to-q delay of time *q*, as well as a setup time *s*, and a hold time *h* where $h \leq q$. All times given are in seconds.

Max frequency = 1/[$4a + q + s$] Hz

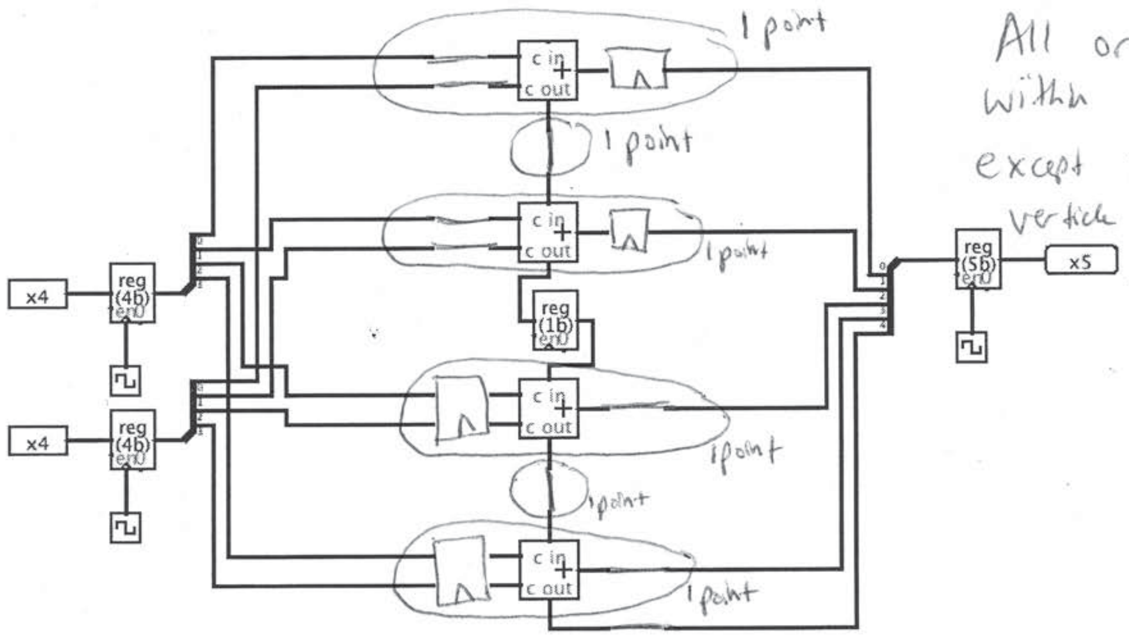
1 for $4a$

1 for $q + s$

6 points

b) Alyssa P Hacker realizes that she can improve this circuit by pipelining it. She starts by inserting a register between the carry out of the second adder and the carry in of the third adder (see next page).

Your job is to add additional registers to make this circuit correctly perform addition again. You are to either place a register in each empty space (any box will be taken to be a register) or place a wire across it.



All or nothing within circle, except if both vertical left blank -1 not -2

2 points c) Now what is the max frequency:

Max frequency = 1/[2a + q + s] Hz.

- 2 points for correct answer
- 1 point for dividing 4a by something other than 2
- 1 point for dividing the whole expression not just 4a by 2

2 points d) In terms of time, the latency of a single addition has [✓ increased] / decreased / stayed the same. Circle one.

All or nothing

2 points In terms of additions per time, the throughput of this circuit has [✓ increased] / decreased / stayed the same. Circle one.

All or nothing