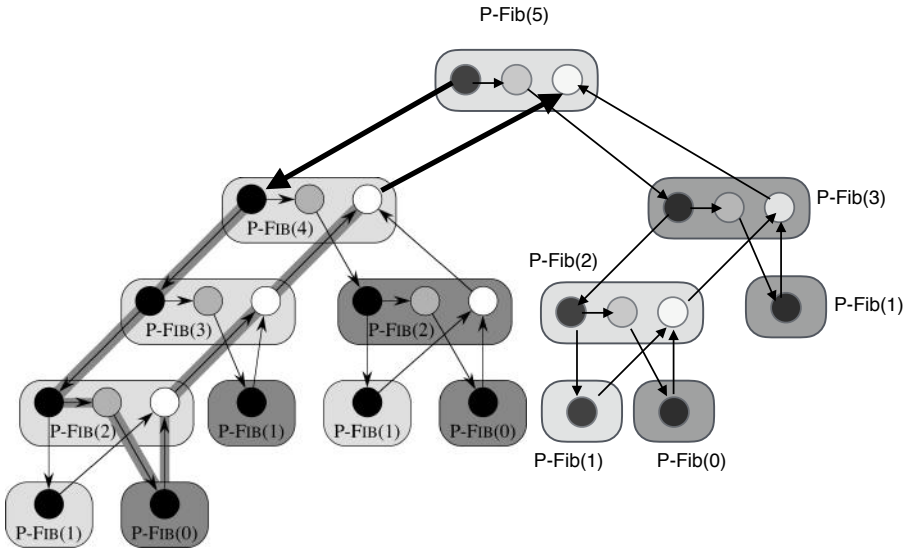
CMSC 441: Unit 4 Homework Solutions

November 19, 2017

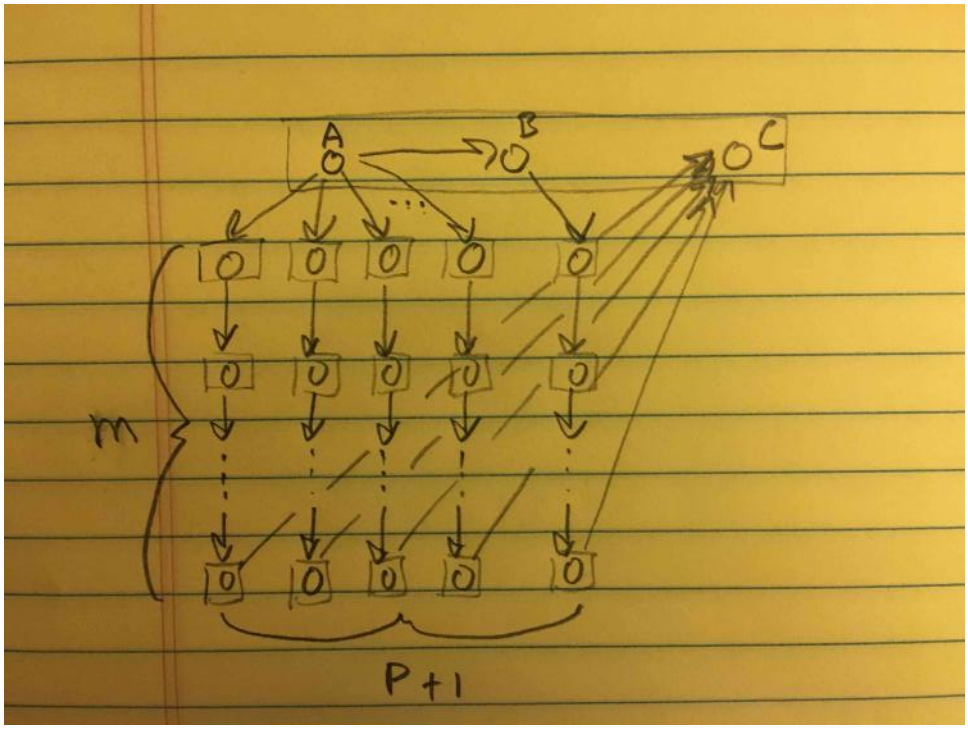
(27.1-1) If the call to P-FIB$(n-2)$ were replaced with a spawn, we would add a horizontal arrow from the center strand to the right strand in each call to P-FIB, indicating that the right strand is permitted to execute in parallel with the spawned computation. However, right strand would still need to wait for *both* spawned computations to complete before it could be executed, so there would be no change to the critical path length. Also, the work would be unchanged as the number of strands would not change. Thus, the work, span, and parallelism would be unchanged.

(27.1-2)



We can see that the length of the critical path is is nine, so $T_\infty = 10$. The work is just the number of strands, so $T_1 = 29$. Therefore, the parallelism is $T_1/T_\infty = 29/10 = 2.9$.

(27.1-4) The only non-determinism in greedy scheduling occurs on complete steps when there are more strands available for scheduling than processors. In this case, we can not be sure which strands will be scheduled. Consider the following DAG:



With P processors, suppose we execute strand A first, then execute the strands of the first P columns. This will take $m + 1$ steps. Executing B and the m strands in the $(P + 1)$st column will take an additional $m + 1$ steps, and the final execution of strand C is an additional step. Putting this all together gives $2m + 3$ steps.

Alternatively, suppose the scheduler always completes full rows of the computation as soon as possible:

1. Execute A.
2. Execute B and $P - 1$ strands from first row.
3. Execute remaining two strands from first row and $P - 2$ strands from second row.
4. Execute remaining two strands from second row and $P - 3$ strands from third row.
5. etc.

Then the total number of steps to complete the large parallel portion is no greater than

$$\left\lceil \frac{m(P+1)}{P} \right\rceil = \left\lceil m\frac{P+1}{P} \right\rceil .$$

As P becomes large, $(P+1)/P$ approaches 1, so the total number of steps for the parallel block is approximately m. Adding 3 additional steps for A, B, and C gives a total of $m+3$ steps which, for large m, is about half as many steps as with our first approach.

(27.1-7) We can assume the exchange on line 4 is $\Theta(1)$. The the number of iterations of the inner loop (line 3) are $1, 2, \ldots, n-1$ and the work T_1 is the sum, or $n(n-1)/2$, which is $\Theta(n^2)$. Therefore, $T_1(n) = \Theta(n^2)$. To analyze the span, we use the following formula from the textbook:

$$T_\infty(n) = \Theta(\lg n) + \max_{2 \le j \le n} iter_\infty(j)$$

where $iter_\infty(j)$ is the span of the j^{th} iteration. The $\Theta(\lg n)$ term is the cost of recursive spawning introduced by the parallel for construct. In this case, $iter(j)$ is the code on lines 3 and 4; since this is another parallel loop, and line 4 is $\Theta(1)$, we have that

$$iter_\infty(j) = \Theta(\lg n) + \Theta(1) = \Theta(\lg n).$$

and therefore,

$$T_\infty(n) = \Theta(\lg n) + \Theta(\lg n) = \Theta(\lg n).$$

Finally, the parallelism is

$$T_1/T_\infty = \Theta(n^2/\lg n).$$

(27.1-8) The work will be the serial running time. The nested loops take a total of $1+2+\cdots+(n-1) = \Theta(n^2)$ iterations, the cost of the exchange on line 4 is $\Theta(1)$, and the cost of line 1 is $\Theta(1)$ so $T_1 = \Theta(n^2)$.

The span is the parallel running time. Using the result on analysis of parallel for loops, we have that

$$T_\infty(n) = \Theta(\lg n) + \max_{2 \le j \le n} \text{iter}_\infty(j).$$

The iterations have different running times, but the maximum is the last iteration when $j = n$, with $\text{iter}_\infty(n) = \Theta(n)$. Therefore, $T_\infty(n) = \Theta(n)$.

The parallelism is $T_1(n)/T_\infty(n) = \Theta(n)$.

(27.2-3) The problem with the P-Square-Matrix-Multiply function in the textbook is that the innermost loop cannot be parallelized without inducing a

race condition. To reduce the span to $\Theta(\lg n)$, we need to find a way around this obstacle. The following code creates a separate array d to store the products $a_{ik} \cdot b_{kj}$, removing the race condition. However, it is still necessary to sum the elements of d, which can be done in $\Theta(\lg n)$ time using the parallel Divide-and-Conquor algorithm P-SUM.

```
P-SQUARE-MATRIX-MULTIPLY(A, B)
 1  n = A.rows
 2  Let C be a new n-by-n matrix
 3  Let d be a new n-long array
 4  parallel for i = 1 to n
 5       parallel for new j = 1 to n
 6            c_ij = 0
 7            parallel for new k = 1 to n
 8                 d_k = a_ik · b_kj
 9            sum = P-SUM(d)
10            c_ij = c_ij + sum
11  return C
```

```
P-SUM(x, p, r)
1  if p < r
2       q = ⌊(p + q)/2⌋
3       a = spawn P-SUM(x, p, q)
4       b = P-SUM(x, q + 1, r)
5       sync
6       return a + b
7  else return x[p]
```

(27.2-5) We use a block-submatrix approach, similar to the Divide-and-Conquer matrix multiplication algorithm discussed in class. Write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

where the A_{ij} are $n/2$-by-$n/2$ submatrices. Then the transpose A^T of A is

$$A^T = \begin{pmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{pmatrix}.$$

The transposes of the submatrices can be computed recursively.

```
P-TRANSPOSE(A)
 1  n = A.rows
 2  if n > 1
 3       Let A11 be the upper left n/2-by-n/2 submatrix of A
 4       Let A12 be the upper right n/2-by-n/2 submatrix of A
 5       Let A21 be the lower left n/2-by-n/2 submatrix of A
 6       Let A22 be the lower right n/2-by-n/2 submatrix of A
 7       Let C be a new n-by-n matrix
 8       Let C11, C12, C21, C22 be the n/2-by-n/2 block submatrices of C
 9       C11 = spawn P-TRANSPOSE(A11)
10       C12 = spawn P-TRANSPOSE(A21)
11       C21 = spawn P-TRANSPOSE(A12)
12       C22 = P-TRANSPOSE(A22)
13       sync
14       return C
15  else return A
```

The work of the algorithm satisfies the recurrence relation

$$T_1(n) = 4T_1(n/2) + \Theta(n^2),$$

where the $\Theta(n^2)$ represents the work of copying the matrices returned by the calls to P-TRANSPOSE to the submatrices of C. By MT (ii),

$$T_1(n) = \Theta(n^2 \lg n).$$

To compute the span, we first note that the copies for the return values can be performed in $\Theta(\lg n)$ time using nested parallel loops, so the recurrence relation is

$$T_\infty(n) = T_\infty(n/2) + \Theta(\lg n),$$

and by MT (iii) (assuming $f(n)$ satisfies the regularity condition),

$$T_\infty(n) = \Theta(\lg n).$$

Then the parallelism is given by

$$\frac{T_1(n)}{T_\infty(n)} = \Theta(n^2).$$