Department of Information and Computer Science
Utrecht University

INFOB3CC: Work & Span (solutions)

Trevor L. McDonell

January 22, 2024

Introduction

Use these tasks to practice the topics of the lectures. You may have to do some research to read up on terms or topics not (yet) covered in the lectures.

Efficient and optimal

A parallel algorithm has two asymptotic complexity parameters—work and span—both expressed as a function of the input size n. For parallel algorithms we sometimes accept that the work is higher than that of the best sequential algorithm. The ratio of work divide by the (best sequential) time is the parallelisation overhead. An algorithm is called *efficient* when the span is poly-logarithmic and the overhead is also poly-logarithmic. An algorithm is called *optimal* when the span is poly-logarithmic and the overhead is constant.

The Master Theorem

The Master Theorem immediately gives you the asymptotic solution for recurrent relations of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$. You should compare $b \log a$ with the power of n in $f(n)$, so ignore any log factors in f. If $n^{b \log a}$ or $f(n)$ has a greater n-power than the other, that is the answer. If the exponents are the same, than an extra log-factor is to be added (on top of any logs in f that you now do not ignore further). For further information refer to the college on recursion in the course Data Structures, or see the cheat sheet here:

- `https://ics.uu.nl/docs/vakken/ds/WerkC/WorkMasThm.pdf`

Questions

1. Parallel algorithms can often use recursion efficiently, for example using a divide-and-conquer strategy. We want to write a method `sum(A, p, q)` which calculates the sum of all numbers in A in the range $[p, q)$. Consider the following method which computes the sum of an array recursively:

```
1  sum(A, p, q)
2    if (p == q-1)
3      return A[p]
4
5    m = (p + q) / 2
6    s = async ( sum(A, p, m) )   // run in separate thread
7    t = async ( sum(A, m, q) )   // run in separate thread
8    return s+t                   // assume this waits for the results
```

   (a) What is the work of this algorithm?

Solution: To calculate work $W(n)$, note first that lines 2, 3, 5, and 8 cost $\Theta(1)$, then the two recursive calls each cost $W(n/2)$. Therefore the total work is $W(n) = \Theta(1)+2*W(n/2) = \Theta(n)$.

(b) What is the span of this algorithm?

Solution: To calculate span $S(n)$, we consider that lines 6 and 7 executed in parallel each now count only once, so we have $S(n) = \Theta(1) + S(n/2) = \Theta(\log n)$.

2. Given an array of n numbers sorted in ascending order, we want to compute the smallest difference between two consecutive numbers δ. For example, given the array:

```
xs = [1,3,5,7,8,12,14,25]
```

Then $\delta = 1$ (namely difference between 7 and 8). The best possible sequential algorithm costs $\Theta(n)$ steps.

(a) Give a parallel divide-and-conquer algorithm for computing the minimum difference of an array.

Solution: The minimum distance is either the δ from the left half, the δ from the right half, or the difference between the last value on the left and the first value on the right.

(b) Analyse the work and span of your algorithm using the Master Theorem

Solution: Calculating the work and span:

- *Work:* we make two recursive calls of the function at half size of the input, plus a constant amount of work, so $W(n) = 2*W(n/2)+\Theta(1)$. From the Master Theorem it follows that $W(n) = \Theta(n)$.
- *Span:* We make two recursive calls in parallel at half the input size, plus a constant amount of work for computing the difference between the values at the ends. So $S(n) = S(n/2) + \Theta(1)$ and by the Master Theorem $S(n) = \Theta(\log n)$.

(c) Is the algorithm efficient and optimal?

Solution: The span is logarithmic, and the overhead is constant (and thus also polylogarithmic). Efficient means [poly]logarithmic overhead and [poly]logarithmic span, which applies here. Optimal means [poly]logarithmic span and constant overhead, which also applies here.

3. Sara has developed a parallel algorithm with work $\Theta(n^3)$ and span $\Theta(\log^3 n)$.

(a) What is the (asymptotic) running time for this algorithm with a linear ($\Theta(n)$) number of processors?

Solution: With so few processors (namely $P < work/span$) the algorithm is work bound, so you can estimate the time as $T_P = work/P = n^3/n = n^2$.

(b) At how many processors will the algorithm become (asymptotically) span-bound?

Solution: The break even point between being work-bound and span-bound is $P = work/span = n^3/\log^3 n$ processors.

4. The length of a greedy schedule can vary depending on which ready steps the scheduler chooses in each round.

   (a) Give an example of a calculation graph and two greedy schedules, each on three cores, where one schedule takes at least one and a half times as long as the other.

   > **Solution:** This is an example; another answer may look completely different. Consider 18 tasks; 6 are sequentially dependent, while the other 12 are all completely independent.
   >
   > (a) Schedule 1: First finish all of the independent tasks in 4 rounds, then the 6 sequential tasks in 6 rounds (only one thread is active). This costs a total of 10 rounds.
   >
   > (b) Schedule 2: At reach round take one task from the sequentially dependent chain and two from the set of independent tasks. This takes a total of 6 rounds.

   (b) Is it possible to give an example where a bad greedy schedule is more than two and a half times as long as the other?

   > **Solution:** According to Brent's scheduling theory, a greedy schedule can be no longer than twice that of the best schedule, so no longer than twice the length of another greedy schedule.

5. A parallel calculation with work W and span S must be done on a machine with P processors and you want to do this according to a greedy schedule.

   (a) Describe how a greedy schedule is created

   > **Solution:** The greedy schedule chooses a maixmum number of ready steps in each round. There are P in a "full" round (core-bound, where every processor is busy), or less than this in an "empty" (ready-bound) round.

   (b) Why is the length T of this schedule limited by $(W/P) + S$?

   > **Solution:** Because a full round performs P steps, there is a maximum of W/P full rounds. Because an empty round decreases the span of the remaining calculation by at least one, there is at most S empty rounds. Thus the number of rounds is limited by $W/P + S$.

   (c) Your boss does not think this is fast enough. He claims your competitor is four times faster. Can you catch up with your competitor with better scheduling?

   > **Solution:** This will not work through better scheduling alone, because the greedy schedule is 2-optimal. Each schedule uses at least $\max(W/P, S)$ steps, and this is more than half of the number $W/P + S$. To become four times as fast, you need a better algorithm or more processors.

6. For a certain task, you have a parallel algorithm A with work W_A and span S_A. There is an alternative algorithm B with higher work $W_B > W_A$ but lower span $S_B < S_A$. For what number of processors P do you estimate algorithm B to have a lower (asymptotic) calculation time than A?

   > **Solution:** The answer is $P \geq W_B/S_A$. According to Brent's scheduling theory, the calculation time of A is about W_A/P as long as $P \leq W_A/S_A$, the work-bound phase, and S_A when $P \geq W_A/S$, the span-bound phase. By displaying this in a graph, you can see that A and B intersect at $P = W_B/S_A$.

7. Consider a function `maxOneRow` which computes the longest sequence of consecutive ones in a number, for example `maxOneRow 01011001111100111111110 = 7` (see the positions starting at index 7 and 14). You can calculate the `maxOneRow` function using a recursive function which returns three values from the row of bits:

   - `p`: the number of ones with which the row starts
   - `i`: the length of the longest sequence of ones in this segment
   - `s`: the number of ones with which the row ends

   (a) What is the best sequential time to calculate the `maxOneRow`?

   > **Solution:** $\Theta(n)$. It can not be sub-linear, but you don't need to prove this.

   (b) How do you find the values of `p`, `i`, and s of a row of digits given those values for the left half and the right half of the range?

   > **Solution:** Combining the results from the left and right halves requires only a constant amount of work to compute each of `p`, `i`, and `s`. You can write out the code for this but it's not necessary, so we leave it as an exercise.

   (c) Analyse the work of the resulting algorithm

   > **Solution:** Combining the sub-results is a constant amount of work. So to find the work we have $W(n) = 2W(n/2) + \Theta(1) = \Theta(n)$ by the Master Theorem.

   (d) Analyse the span of the resulting algorithm

   > **Solution:** Combining the sub-results is a constant amount of work, and the sub-computations are independent so $S(n) = S(n/2) + \Theta(1) = \Theta(\log n)$ by the Master Theorem.

   (e) Is the parallel algorithm efficient and optimal?

   > **Solution:** The span is logarithmic and the work is linear—the same as the sequential calculation—so the overhead is constant. This makes the algorithm both efficient and optimal.

8. Ada has developed a parallel algorithm with work $\Theta\left(n^{1.5}\right)$ and span $\Theta\left(\log^3 n\right)$. Gabriëlle thinks that her own algorithm for the task is better, since the span is only $\Theta\left(\log^2 n\right)$, although the work is $\Theta\left(n^2\right)$.

   (a) Which algorithm will perform (asymptotically) better with a linear $\Theta(n)$ number of processors? Motivate your response.

   > **Solution:** With a low number of processors $P < work/span$, both algorithms are work bound so you can estimate the time as $T_P = work/P$. Gabriëlle runs in a time $n^2/n = n$ (linear time), while Ada runs quicker in $n^{1.5}/n = \sqrt{n}$ (polynomial time).

   (b) Which algorithm will perform (asymptotically) better with a quadratic $\Theta\left(n^2\right)$ number of processors? Motivate your response.

   > **Solution:** With a large number of processors both algorithms are span bound, so you can estimate the running time as $T_P = span$. Ada will use $\log^3 n$ time while Gabriëlle uses only $\log^2 n$ time.

(c) For what number of processors is Ada's algorithm (asymptotically) faster?

> **Solution:** The break even point is at $P = n^2/\log^3 n$. With that number of processors Ada is span-bound, using $\log^3 n$ time and will not get any faster by adding more processors. At this point Gabriëlle is still work bound, and will continue to increase performance with more processors (up to a factor $\log n$).

9. Mergesort is a divide-and-conquer comparison-based sorting algorithm. To sort an array, the algorithm will recursively sort the first and last halves of the input, and then merge the two sorted sub-arrays. Recall that the complexity of the standard sequential Mergesort algorithm is $O(n \log n)$.

(a) Analyse the span of parallel `mergesort`, in which the two recursive calls are done in parallel, and merging the sub-arrays is performed sequentially in $O(n)$ time.

> **Solution:** Because the recursive calls are done in parallel they only count once, although the merging is still linear. Then:
>
> $$S(n) = S(n/2) + O(n) = O(n)$$
>
> where using the Master theorem $a = 1$, $b = 2$, and $f(n) = n^1$.

(b) Is the algorithm from part (a) efficient and is it optimal? Explain your response.

> **Solution:** The work is:
>
> $$W(n) = 2W(n/2) + \Theta(n) = \Theta(n \log n)$$
>
> using $a = 2$, $b = 2$, and $f(n) = \Theta(n)$ in the Master theorem.
> The work is equal to the sequential time ($\Theta(n \log n)$), which is optimal for comparison-based sort, so the overhead is constant. Unfortunately the span is not poly-logarithmic, so the algorithm is not optimal (poly-logarithmic span and constant overhead) nor even efficient (poly-logarithmic span and poly-logarithmic overhead).

(c) If the merging of the sub-arrays can be done in parallel in $O(\log n)$ time, what does that mean for the work and span of the `mergesort` algorithm? Using this merging technique is the `mergesort` algorithm efficient and is it optimal? Explain your response.

> **Solution:** The span becomes:
>
> $$S(n) = S(n/2) + O(\log n) = O(\log^2 n)$$
>
> The work is (again) equal to the sequential complexity, so there is no overhead, but the span is now poly-logarithmic, so it is therefore both efficient and even optimal.

10. A polynomial p of degree n is an expression in the form:

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

where we can store the polynomial coefficients a in an array of size $n + 1$.

(a) To add two polynomials a and b to a new polynomial c, it suffices to add the coefficients such that $c_i = a_i + b_i$. Give a parallel algorithm that calculates the sum of two polynomials with the lowest possible work and span. Motivate your response.

Solution: The n additions are completely independent. The work is $O(n)$ and the span is $O(1)$.

(b) Melinda discovers that you can multiply two polynomials of degree $n-1$ by making four independent multiplications of degree $\frac{n}{2}-1$ and three polynomial additions of degree n. Analyse the work and span of Mel's algorithm.

Solution: Name the span $S(n)$. Because the sub-multiplications are independent they only count once. The span of the additions is $O(1)$ from part (a). We have $S(n) = S(n/2) + O(1)$, so from the master theorem ($a = 1, b = 2, f(n) = n^0$) it follows that $S(n) = \log n$.

Name the work $W(n)$. We have $W(n) = 4W(n/2) + 3O(n)$, so from the master theorem ($a = 4, b = 2, f(n) = O(n^1)$) we have $W(n) = n^2$

(c) Sara tells Melinda that the product can also be calculated by six additions of degree $n-1$ and three independent multiplications of degree $\frac{n}{2}-1$. Analyse the work and span of Sara's algorithm.

Solution: The span is the same as in (b): $S(n) = \log n$

For the work we have $W(n) = 3W(n/2) + 6O(n)$, so from the master theorem ($a = 3, b = 2, f(n) = O(n^1)$) it follows $W(n) = n^{\log_2 3} = n^{1.58}$.