Department of Information and Computer Science
Utrecht University

INFOB3CC: Work & Span

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?

   (b) What is the span of this algorithm?

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.

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

   (c) Is the algorithm efficient and optimal?

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?

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

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.

   (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?

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

   (b) Why is the length T of this schedule 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?

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?

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`?

   (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?

   (c) Analyse the work of the resulting algorithm

   (d) Analyse the span of the resulting algorithm

   (e) Is the parallel algorithm 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.

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

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

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.

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

   (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.

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.

   (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.

   (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.