CS3101 Due: Tuesday 10-th of March 2015

Problem Set 1

CS3101 *Submission instructions on last page*

PROBBLEM 1. [70 points]

We consider a two-dimensional tableau (or matrix) T containing elements $T[i, j]$ with $0 \leq i \leq n$ and $0 \leq j \leq n$. We assume that the elements $T[i, j]$ are known whenever $i = 0$ or $j = 0$. For simpliicity, we assume that n is a multiple of 2.

For $1 \leq i \leq n$ and $1 \leq j \leq n$ we assume that $T[i, j]$ can be computed from $T[i - 1, j]$ and $T[i, j - 1]$ in $\Theta(1)$ arithmetic operations, that is

$$\mathrm{T[i,j]} \quad = \quad \mathrm{T[i-1,j]} + \mathrm{T[i,j-1]}.$$

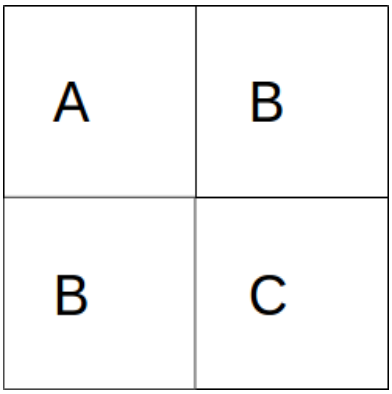


Figure 1: Divide-and-conquer method

We are interested in studying two algorithms for filling the tableau: divide-and-conquer algorithm in Figure 1 and blocking algorithm in Figure 2.

For the divide-and-conquer algorithm, see Figure 1, filling a tableau can be done by:

1. first filling the upper square `A` which has $n/2$ rows and $n/2$ columns,
2. then filling the two squares `B` in parallel, which also have $n/2$ rows and $n/2$ columns,
3. finally filling the lower square `C`, which has $n/2$ rows and $n/2$ columns.

Given a positive integer B dividing n, the blocking algorithm, this is achieved by partitioning the entire tableau into blocks of format $B \times B$, as shown in Figure 2 (where B equals 3 in that case). In practice, the blocking value of `B` should be tuned in order for a block to fit in cache. Then filling a tableau can be done by

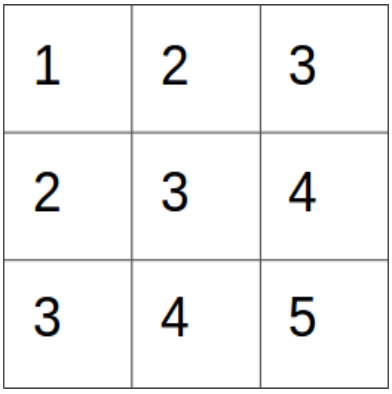


Figure 2: Blocking method

1. first filling the block labeled `1` which has n/B rows and n/B columns,
2. then filling the two blocks labeled `2` in parallel, which also have n/B rows and n/B columns,
3. then filling the three blocks labeled `3` in parallel, which also have n/B rows and n/B columns,
4. then filling the two blocks labeled `4` in parallel, which also have n/B rows and n/B columns,
5. finally filling the block labeled `5`, which has n/B rows and n/B columns.

Recall that we assume that n is a power of 2. We denote by $W_T(n)$ the total number of arithmetic operations (that is, the work) for filling a tableau T with n rows and n columns. We denote by $S_T(n)$ the span for filling the same tableau T with n rows and n columns.

Question 1. [10 points] Write down the recurrence relations satisfied by $W_T(n)$ for the divide-and-conquer algorithm and blocking algorithm, respectively.

Question 2. [10 points] Use the Master Theorem to give asymptotic orders of magnitude for $W_T(n)$. See the link, `http://en.wikipedia.org/wiki/Master_theorem`

Question 3. [10 points] Write down the recurrence relations satisfied by $S_T(n)$ for the divide-and-conquer algorithm and blocking algorithm respectively.

Question 4. [10 points] Use the Master Theorem to give asymptotic orders of magnitude for $S_T(n)$ and deduce the parallelism of your algorithm.

Question 5. [30 points] Following the above rules, write a parallel algorithm using the `CilkPlus` language for filling a tableau (both algorithms).

PROBBLEM 2. [30 points] Given an associative operator $\otimes$, operating on a set T, and an array $x[1..n]$ of elements of T, the ***$\otimes$-prefix computation*** for x, sometimes called a ***$\otimes$-scan***, produces the array $y[1..n]$ given by

$$\begin{aligned} y[1] &= x[1], \\ y[2] &= x[1] \otimes x[2], \\ y[3] &= x[1] \otimes x[2] \otimes x[3], \\ &\vdots \\ y[n] &= x[1] \otimes x[2] \otimes x[3] \otimes \cdots \otimes x[n], \end{aligned}$$

that is, all prefixes of the array x "summed" using the $\otimes$ operator. Using a parallel for loop, we can obtain a multithreaded $\otimes$-prefix computation:

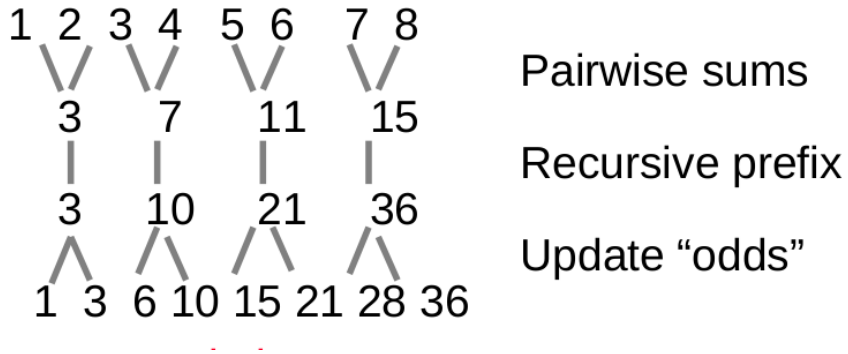
```
P-Scan (x) {
    n = x.length;
    Let y[1..n] be a new array;
    P-Scan-Aux (y, x, 1, n);
    return y;
}
P-Scan-Aux (y, x, i, j) {
    cilk_for (int ℓ = i; ℓ < j; ++ℓ) {
        y[ℓ] = x[1];
        for (int k = 2; k < ℓ; ++k)
            y[ℓ] += x[k];
    }
}
```

Question 1. [10 points] Analyze its work, span and parallelism of the above algorithm.

Question 2. [10 points] Explain why the work of the previous multithreaded algorithm not satisfactory. **Hint:** You can compare it against the work of a serial code (where $\otimes$ is denoted by $+$) computing the vector y as follows:

```
y [1] = x[1]
for  (i=1 ; i < n; i++) {
   y[i+1] = y[i] + x[i+1]
}
```

Question 3. [10 points] To deal with this inefficiency issue, we consider now a divide-and-conquer method suggested by the picture below, where n is assumed to be a power of 2.



1. Under this assumption, give a multithreaded pseudo-code (consider to use `cilk_for` and/or `cilk_spawn`) of the algorithm suggested by the picture
2. Analyze the work, span, and parallelism of this latter algorithm.

Submission instructions.

Format: Problems 1 and 2 involve programming with CilkPlus: they must be submitted as two input files to be called `Pb1.cpp` and `Pb2.cpp`, respectively. Each of these two files must be a valid input file for CilkPlus. In addition, each user defined function must be **documented.**

Submission: The assignment should be returned to the instructor by email.

Collaboration. You are expected to do this assignment *on your own* without assistance from anyone else in the class. However, you can use literature and if you do so, briefly list your references in the assignment. Be careful! You might find on the web solutions to our problems that are not appropriate. For instance, because the cache memory model is different. So please, avoid those traps and work out the solutions by yourself. You should not hesitate to contact the instructor if you have any question regarding this assignment. I will be more than happy to help.

Marking. This assignment will be marked out of 100. A 10 % bonus will be given if your answers are clearly organized, precise and concise. Messy assignments (unclear statements, lack of correctness in the reasoning, many typographical or language mistakes) may give rise to a 10 % malus.