CS3101 Due: Tuesday 12th of March

Problem Set 2

CS3101 *Submission instructions on last page*

PROBBLEM 1. [30 points]

The objective of this problem is to design a parallel algorithm for the *Maximum Subarray Problem* (MSP). In computer science, the MSP is the task of finding the contiguous subarray within a one-dimensional array of numbers (containing at least one positive number) which has the largest sum. We shall consider two algorithms. The first one is called *Kadane's algorithm* and is described at:

`http://en.wikipedia.org/wiki/Maximum_subarray_problem`

The second one is called *Bentley's algorithm* and is described at:

`http://penguin.ewu.edu/ bojianxu/courses/cscd320/slides_dc_2.pdf`

In our Analise's, we rely on the ideal-cache model (see the lecture notes) and assume that our ideal cache memory has Z words in total, with L words per cache-line. We assume that the input of our implementation of each algorithm is a one-dimensional array A consisting of n contiguous words. For parallelism analysis, we rely on the fork-join multithreaded parallelism (see the lecture notes). Note that both Kadane's algorithm and Bentley's algorithm are not stated as parallel algorithms. So a first task is to state fork-join versions of those algorithms, when possible. Once this is done, one can consider the work and the span of those algorithms. We denote respectively by $W_K(n)$, $S_K(n)$ and $Q_K(n)$:

- the total number of arithmetic operations (including comparisons)
- the span,
- the number of cache misses.

during the execution of Kadane's algorithm when applied to an array A of size n. Similarly, we denote by $W_B(n)$, $S_B(n)$ and $Q_B(n)$ the work, the span and the number of cache misses during the execution of Bentley's algorithm when applied to an array A of size n.

Question 1. [10 points] Explain why Kadane's algorithm cannot be parallelized in the fork-join multithreaded parallelism model. Deduce $W_K(n)$ and $S_K(n)$,

Question 2. [10 points] Propose a parallel version of Bentley's algorithm. Determine $W_B(n)$ and $S_B(n)$.

Question 3. [10 points] On multicore architectures, which of the two algorithms should give the best performance? Justify your answer. You are welcome to realize a `Cilk++` implementation of Bentley's algorithm and a `C/C++` implementation of Kadane's algorithm. If successful, those would give 10 bonus points.

PROBBLEM 2. [40 points] The square of a matrix A is its product with itself, namely AA.

Question 1. [5 points] Show that five multiplications are sufficient to compute the square of a 2×2 matrix.

Question 2. [15 points] In this part, we show that if $n \times n$ matrices can be squared in time $W_S(n) = O(n^c)$, for some real number $c \geq 2$, then any matrices can be multiplied in time $O(n^c)$ as well. In other words, squaring matrices is no easier (in terms of work) than matrix multiplication.

1. Given two $n \times n$ matrices A and B, show that the matrix $AB + BA$ can be computed in time $3W_S(n) + O(n^2)$.
2. Given two $n \times n$ matrices X and Y, define the $2n \times 2n$ matrices A and B as follows
$$A = \begin{pmatrix} X & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & Y \\ 0 & 0 \end{pmatrix}$$
What is $AB + BA$ in terms of X and Y?
3. Using the answers of the previous two questions, argue that XY can be computed in time $3W_S(n) + O(n^2)$.
4. Conclude that matrix multiplication can be done in time $W_S(n) = O(n^c)$.

Question 3. [20 points] In this part, we consider squaring matrices in a parallel manner.

1. Using the `Cilk++` language as pseudo-code, propose a parallel divide-and-conquer algorithm for squaring matrices. This algorithm should not allocate extra storage during the recursive calls.
2. Analyze the span of your algorithm.
3. Compare your algorithm with the in-place parallel divide-and-conquer algorithm studied in class for matrix multiplication.

PROBBLEM 3. [30 points]

In this problem, we develop a divide-and-conquer algorithm for the following geometric task, called the *CLOSEST PAIR PROBLEM* (CSP):

Input: A set of n points in the plane

$$\{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$$

Output: The closet pair of points, that is, the pair $p_i \neq p_j$ for which the distance between p_i and p_j, that is,

$$\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

is minimized.

For simplicity, we assume that n is a power of 2 and that all the x-coordinates x_i are pairwise distinct, as well are the y-coordinates y_i. Here's a high-level overview of the proposed algorithm:

1. Find a value x for which exactly half the points have $x_i < x$ and half have $x_i > x$, thus splitting into groups L and R.
2. Recursively find the closest pair in L and R. Let us call these pairs $p_L, q_L \in L$ and $p_R, q_R \in R$, with distances d_L and d_R. Let d be the smaller of these two distances.
3. It remains to be seen whether there is a point in L and a point in R that are less than distance d apart from each other. To this end, discard all points with $x_i < x - d$ or $x_i > x + d$. Then, sort the remaining points by y-coordinate.
4. Now, go through this shorted list, and for each point, compute its distance to the *seven subsequent points* in the list. Let p_M, q_M be the closest pair found in that way.
5. The answer is $\{p_L, q_L\}$, $\{p_R, q_R\}$ and $\{p_M, q_M\}$, whichever is closest.

Question 1. [5 points] In order to prove the correctness of this algorithm, start by showing the following property: any square of size $d \times d$ (where d is as defined in the above overview of the proposed algorithm) in the plane contains at most *four points* of L.

Question 2. [5 points] Now show that the algorithm is correct. The only case which needs careful consideration is when the closest pair is split between L and R.

Question 3. [5 points] Write down the pseudo-code for the algorithm, and show that its work is given by the recurrence:

$$W(n) = 2W(n/2) + O(n\log(n))$$

Deduce that $W(n) \in O(n\log^2(n))$.

Question 4. [5 points] Propose a parallel version of this algorithm and show that its parallelism is limited to $\log(n)$.

Question 5. [10 points] Propose an improved parallel algorithm with a parallelism of $n/\log(n)$.

Submission instructions.

Format: The answers to the problem questions should be typed.

- If these are programs, input test files and a `Makefile` (for compiling and running) are required.
- If these are algorithms or complexity analyzes, LaTeX is highly recommended; in any case a PDF file should gather all these answers.

All the files should be archived using the UNIX `tar` command.

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 parallelism model is different. So please, avoid those traps and work out the solutions by yourself. You should not hesitate to contact me if you have any questions 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 paper is clearly organized, the answers are precise and concise, the typography and the language are in good order. Messy assignments (unclear statements, lack of correctness in the reasoning, many typographical and language mistakes) may give rise to a 10 % malus.