Domain decomposition strategy

By using the domain decomposition strategy, give a parallel solution for the following problems.

Matrix multiplication (1)

The matrix multiplication problem can be solved by applying a divide-and-conquer strategy. It works as follows: Let A and B be two n×n matrices and let C be the result of A×B. We can **decompose** the three matrices in (n/2)×(n/2) submatrices

$$A=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B=\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

and redefine the multiplication as follows:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\times\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}=\begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{pmatrix}\times\begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}$$

Thus, to multiply two n×n matrices we perform eight multiplications of (n/2)×(n/2) matrices and one addition of two n×n matrices.

Implement this matrix multiplication strategy using Cilk Plus and compare it with the direct parallelization of the three-loop matrix multiplication algorithm.

Matrix multiplication (2)

An asymptotically better alternative to perform matrix multiplication is the Strassen's matrix multiplication algorithm. Its complexity is $O(n^{\log_2(7)})$ for n×n matrices. The Strassen's algorithm also divide the matrices in (n/2)×(n/2) submatrices and perform independent operations in the submatrices, as follows:

$$A=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B=\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\times\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} P_6 + P_5 + P_4 - P_2 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}) \\ P_2 &= B_{22}(A_{11} + A_{12}) \\ P_3 &= B_{11}(A_{21} + A_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}) \end{aligned}$$

Implement this algorithm in parallel.

Note: Review the implementation provided in: Parallel algorithm implementing Strassen's Algorithm for matrix-matrix multiplication (Bradley Kuszmaul). URL: https://software.intel.com/en-us/courseware/256196

String matching

Let $\sum$ be an finite alphabet. The string counting problem is defined as follows: Let T and P two sequences over $\sum$; find the number of occurrences of P in T. In general, T is called the *text* and P is called the *pattern*.

Implement a parallel algorithm to solve the string counting problem. Compare it with the naïve parallelization studied in the map pattern section.

Wavelet tree

A wavelet tree is a data structure that maintains a sequence of n symbols $S = s_1, s_2, \ldots, s_n$ over an alphabet $\Sigma = [1..\sigma]$ under the following operations: $\mathrm{access}(S, i)$, which returns the symbol at position i in S; $\mathrm{rank}_c(S, i)$, which counts the times symbol c appears up to position i in S; and $\mathrm{select}_c(S, j)$, which returns the position in S of the j-th appearance of symbol c.

The wavelet tree is a balanced binary tree. We identify the two children of a node as left and right. Each node represents a range $R \subseteq [1, \sigma]$ of the alphabet Σ, its left child represents a subset R_l, which corresponds with the first half of R, and its right child a subset R_r, which corresponds with the second half. Every node virtually represents a subsequence S' of S composed of symbols whose value lies in R. This subsequence is stored as a bitmap in which a 0 bit means that position i belongs to R_l and a 1 bit means that it belongs to R_r.

A wavelet tree requires $n \lg \sigma + o(n \lg \sigma)$ bits of space.

Check the file `wavelet_tree.c` to see a sequential implementation to construct a wavelet tree. Design parallel algorithm to construct a wavelet tree. For each algorithm, give its parallel complexity.