Assignment 2: Longest Common Subsequence (Dynamic Programming)

Lars Karlsson (assistant)
Bo Kågström
Mikael Rännar

Due date: 2009-04-23 (17.00)

1 Introduction and Theory

Dynamic programming (DP) is a commonly used technique for discrete optimization problems. It is similar to divide-and-conquer (DoC) in that it uses solutions to subproblems to construct a solution to a larger problem but in contrast to pure DoC there may be dependencies between the subproblems themselves (e.g., the subproblems may have to be solved in a certain order). In this assignment, you will design and evaluate a parallel, MPI-based, implementation of a DP algorithm to solve the Longest Common Subsequence (LCS) problem. The LCS problem is to find the longest subsequence which is common to both sequences $A = \langle a_1, a_2, \ldots, a_m \rangle$ and $B = \langle b_1, b_2, \ldots, b_n \rangle$. For example, if $A = \langle x, y, z, r, t \rangle$ and $B = \langle y, r, z, t \rangle$ their LCS is $\langle y, r, t \rangle$. The related problem Longest Common Substring imposes the additional condition that the subsequences must be contiguous. The LCS problem has found applications in bioinformatics where it has been used as an integral part of the Smith-Waterman algorithm that matches sequences of amino-acids and nucleotides.

The function $F(i, j)$ defined below, gives the length of the LCS of the leading subsequences $A_i = \langle a_1, \ldots, a_i \rangle$ and $B_i = \langle b_1, \ldots, b_j \rangle$:

$$F(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ F(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } a_i = b_j, \\ \max\{F(i, j-1), F(i-1, j)\} & \text{if } i, j > 0 \text{ and } a_i \neq b_j. \end{cases}$$

The length of the LCS of A and B is hence given by $F(m, n)$. We can conceptualize F as an $(m+1) \times (n+1)$ matrix. The first row and column of F are zero. The dependencies are local in the sense that the value of element $F(i, j)$ depends only on a few elements in its neighbourhood (i.e., $F(i-1, j-1)$, $F(i-1, j)$, and $F(i, j-1)$). A sequential algorithm would compute F row-by-row or column-by-column, but a parallel algorithm must exploit the parallelism that is exposed along the antidiagonals of F.

2 Assignment

The course literature proposes a column distribution of F. However, to get any speedup at all you must use at least a block column distribution, otherwise the granularity would be too fine. Your task is to implement a parallel, MPI-based algorithm to find at least the length of the LCS of two sequences. Analyze your algorithm by finding T_p, S_p, E_p, and the asymptotic iso-efficiency function. Also do an *experimental* analysis of the iso-efficiency.

Below is a list of questions to get you started with your analysis.

- Is your algorithm scalable?

- Is it cost optimal?
- What is the algorithm's memory constrained speedup?

3 Extensions

Here is a few suggestions on how you can extend your implementation.

- Define a simple SPMD interface to your function and internally distribute the sequences and compute the length of the LCS before returning the result to all processes.
- Use a block-cyclic column distribution to improve the performance.
- Recover the LCS from F.
- Solve the longest common substring problem.
- Implement the Smith-Waterman algorithm and test it on real data.