Compulsory Assignment no.1
One-dimensional stencil application

Parallel and Distributed Programming
Division of Scientific Computing,
Department of Information Technology,
Uppsala University

Spring 2021

1 Problem setting

In applications such as computer simulations and digital image processing, an operation that is commonly needed is a *stencil application*. This operation is often performed a large number of times, and the performance of the whole program is highly dependent on it. A *stencil* can be seen as an operator that can be applied on the elements of a vector, matrix or tensor (a "multi-dimensional matrix"). When applying the stencil on an element, you compute a new value using the current value of the element in question, and its neighbors. As an example, consider the two-dimensional 9 point stencil in Figure 1. Here, each square represents an element in a matrix/a two-dimensional array, while the red dots represent the stencil. When applying the stencil on the element covered by the middlemost dot, we compute a weighted sum of all elements covered by a dot.

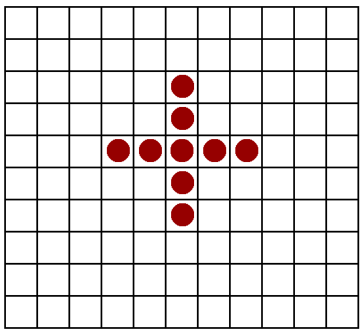

Figure 1: Two-dimensional stencil

In this assignment, we are going to apply a one-dimensional stencil on an array of elements representing function values $f(x)$ for a finite set of N values $x_0, x_1, x_2, ...x_{N-1}$, residing on the interval $0 <= x < 2 \cdot \pi$. Here, each value $x_i = i \cdot h, h = \frac{2 \cdot \pi}{N}$. Applying the stencil on an element v_0 representing the function value $f(x_i)$ simply means computing the sum:

$$\frac{1}{12h} \cdot v_{-2} - \frac{8}{12h} \cdot v_{-1} + 0 \cdot v_0 + \frac{8}{12h} \cdot v_{+1} - \frac{1}{12h} \cdot v_{+2}$$

where v_{-j} is the element j steps to the left of v_0 (that is, $f(x_{i-j})$. Likewise, v_{+j} is the element j steps to the right of v_0 ($f(x_{i+j})$). (Here, the middlemost stencil weight is 0, which is not always the case!) This sum approximates the first derivative $f'(x)$ for $x = x_i$. Your task is to parallelize a short program that applies this stencil on a set of values.

2 Implementation

2.1 The serial code

In Studium, along with these instructions, you can find a file named `stencil.c` which is the serial code to be parallelized. The program provided takes two arguments, the number of function values and the number of times the stencil will be applied. The program applies periodic boundary values. This means that the last element in the array is considered being the closest left neighbor of the first element in the array. Likewise, the first element in the array is used as the closest right neighbor of the last element.

2.2 Requirements for the parallelized code

The generation of the function values should be handled by the process with rank 0. This process is supposed to distribute the values evenly among all processes before the first stencil application and collect the result when the last application is done. You may assume that the number of values is divisible by the number of processes. After the initial distribution, each process should apply the stencil on its received values and store the result in another array. Thereafter, each process repeatedly applies the stencil on the result of its last application until the specified number of applications are made. You can assume that the number of function values per process is larger than the stencil width. Note that in order to apply the stencil, each process will need data that is stored on its neighboring nodes. How will you handle that?
Figure 2 illustrates the procedure when 4 processes are used to apply a stencil twice on an array of length 28. Initially (in state SI), all values are stored in process 0. Next, the values are distributed among the four processes and in state S0 each process keeps its own part (7 elements) of the array. Thereafter, the stencil is applied twice, bringing us to state S2 via state S1. The second stencil application is made on the result of the first one. In state S2, the final result is computed, but it is still distributed among the four processes. Hence, to reach the final state (SF) and being able to report the result, process 0 must collect all the resulting values. The array elements that are marked with gray are needed for the stencil application on a neighboring node.

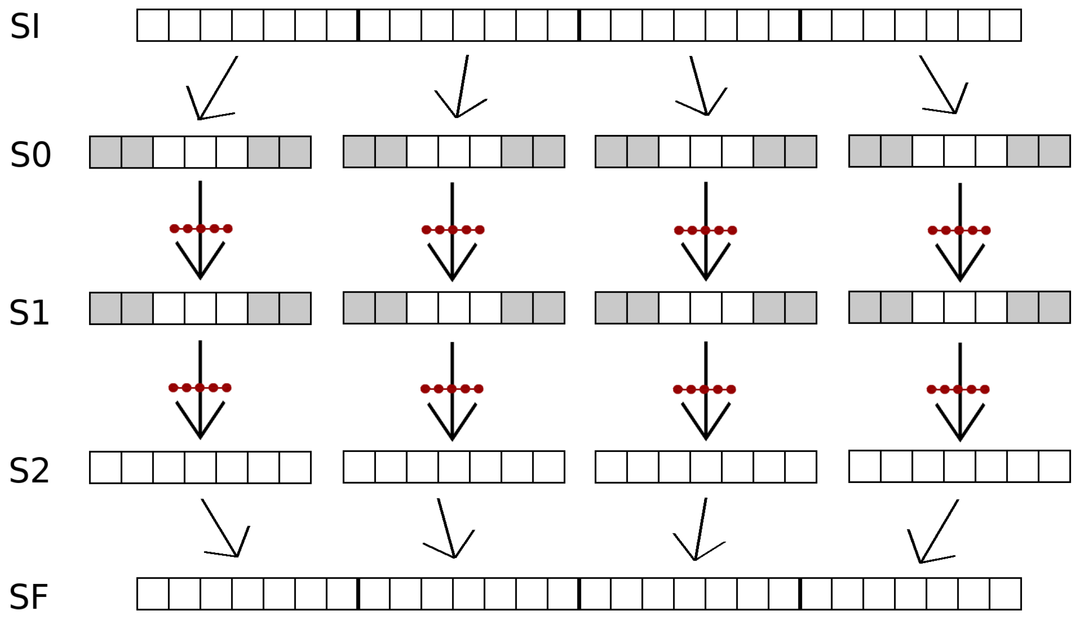

Figure 2: Data distribution and stencil applications

Each process is supposed to measure the stencil application time, but only the largest value should be printed. Measure the time for the application of the stencil and the inter-process communication that needs to be done in each step. Then measure the total parallel runtime including also the time for the initial distribution and the final collection of data. Your parallelization is supposed to speed the program up, but you must not change its functionality! To check the correctness, generate first a reference file with the provided (serial) program and then produce a file with your parallel program. Compare these files with the `diff`-command. You may also plot the input and the output values (eg in R or MATLAB). Does the output look like the $d : th$ derivative of the input after d stencil applications?

3 Performance experiments

When the program is parallelized, it is time for evaluation of its parallel performance. It is recommended to run the experiments on the IT department's Linux servers, but you may use another computer if you like. However, it must be possible to compile your code and run it on the Scientific Linux system (`gullviva, vitsippa, tussilago`). On the Linux servers, use the arguments `--bind-to none` for `mpirun` in order to allow for optimal scheduling of the processes. You are supposed to evaluate the strong and weak scaling of your parallel program.

4 Report

You are supposed to write a report on your results. The report can be written in Swedish or English, and should contain:

1. A brief description of your parallelization. Which kind of communication did you use? Why?
2. Verification of correctness.
3. A description of your performance experiments.
4. The results from the performance experiments. Execution times for different problem sizes and different number of processing elements should be presented in tables. Speedup values should be presented both in a table and a plot. Also plot the ideal speedup in the same figure. The plots can be made using MATLAB, or any other tool that you are familiar with.
5. A discussion of the results. Does the performance of the parallel program follow your expectations? Why/why not?

5 Files to be submitted

Upload the following files to Studium:

- A PDF file named A1_Report.pdf containing you report.
- Your code `stencil.c`, following the specifications listed in Section 2. Note that the code must compile without warnings on the Linux system!
- A makefile for compiling the code.

The assignment must be submitted no later than April 20. Assignments that do not meet the requirements are returned. The final version of the assignment must be submitted no later than June 10.

Happy hacking!

Maya & Jarmo

Any comments on the assignment will be highly appreciated and will be considered for further improvements. Thank you!