Compulsory Assignment no.1
One-dimensional stencil application

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

Spring 2020

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.
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.

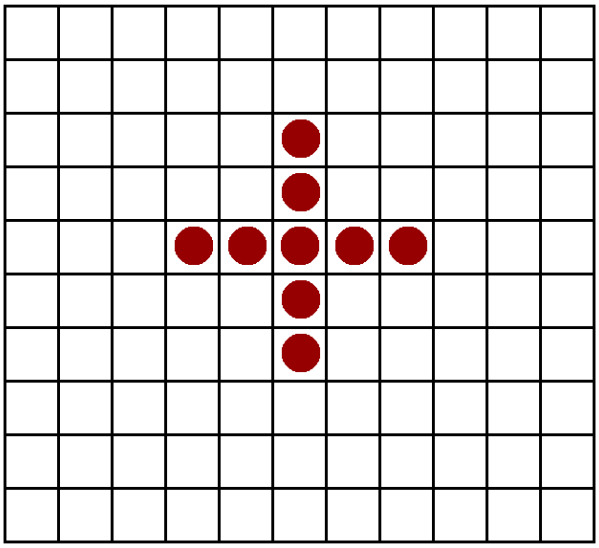

Figure 1: Two-dimensional stencil

2 Implementation

In the Student Portal, along with these instructions, you can find a file named `assignment1.tar.gz` containing the (serial) code to be parallelized, a makefile to build the program and a script which is further described in Section 5.

2.1 The serial code

The program provided takes three arguments. The first argument is the name of a file containing the function values on which the stencil will be applied (the input file). The second argument is the name of the file to which the program will write the resulting function values (the output file). The third argument is an integer specifying how many times the stencil will be applied.
The input file must contain $N + 1$ numbers. The first number is N (that is, the number of function values) and the N subsequent numbers are the actual function values. The output file is created by the program and will contain the result of the stencil application. (Note that the output file is overwritten if it exists already!)
The program provided reads the input file, applies the stencil the specified number of times and writes the result to the output file. Each stencil application is made on the result of the previous stencil application. The program also prints the number of seconds needed for the stencil application (I/O excluded). File I/O is handled in the functions `read_input` and `write_output` respectively. There is no reason for you to change these functions. 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.

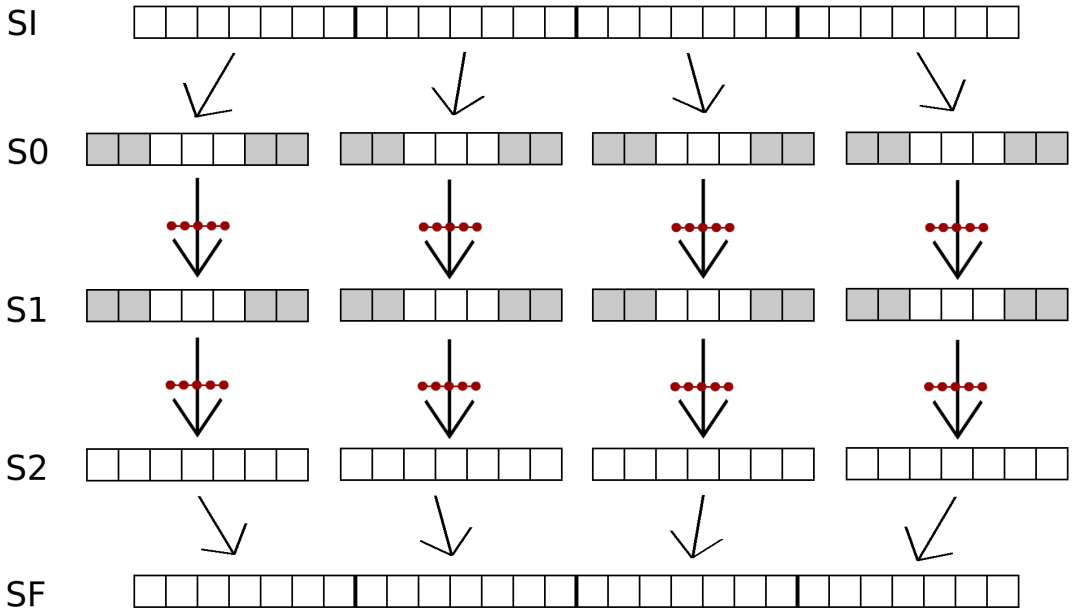

Figure 2: Data distribution and stencil applications

2.2 Requirements for the parallelized code

The I/O of the program 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.

Each process is supposed to measure the stencil application time, but only the largest value

should be printed. The time measurement shall include the application of the stencil and the inter-process communication that needs to be done in each step. It shall not include the time needed for reading the input file and writing of the output file. Neither shall it include the time needed for the initial distribution or the final collection of data.
Your parallelization is supposed to speed the program up, but you must not change its functionality! The assignment is graded automatically and any difference in the output (except from the value (but not the format!) of the stencil application time), compared to the original program, will lead to immediate rejection. To check the correctness of your program, you may use the smallest files stored in `/home/malka730/public/PDP/` on the Linux/Solaris system. Files whose name start with `input` contain input data for the program, on the format specified above. The number in the name of the input files specify the number of function values stored in the file. The files named `outputX_Y_ref.txt` can be compared to the output of your program in order to verify the results. The numbers in the file names (`X` and `Y`) specify the number of function values in the file, and the number of times that the stencil has been applied in order to create the output file. 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 Linux system. There must be no warnings in the compilation step, when the provided makefile is used.
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. You may use the large input files stored in `/home/malka730/public/PDP/` on the Linux/Solaris system for your evaluation. For strong scalability, it is enough to experiment with one file size. For the weak scalability experiments you may vary either the input size or the number of stencil applications.
Note that you have a limited disc quota on the Linux servers and you will not be able to store the input and output files from all runs. Therefore, do not copy the large input files to your home directory. Also remove all large output files immediately! You may also deactivate writing the output file by commenting out the line

```
#define PRODUCE_OUTPUT_FILE
```

from `stencil.h` while performing the performance experiments. However, it is important that you restore it before submitting the assignment. Otherwise, we will not be able to check your program and your submission will be rejected.

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. A description of your performance experiments.
3. 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.
4. A discussion of the results. Does the performance of the parallel program follow your expectations? Why/why not?

5 Files to be submitted

The file that you upload in the Student Portal must be a compressed tar file named `A1.tar.gz`. It shall contain a directory named A1, with the following files inside:

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

Before submitting, please check your file using the script `check_A1.sh` included in `assignment1.tar.gz` (which you downloaded from the Student Portal):

1. Put `check_A1.sh` and `A1.tar.gz` in the same directory.
2. Create a directory named `test_data` in that directory. Without changing their names, copy the following files from `/home/malka730/public/PDP/` on the Linux/-Solaris system to `test_data`:
   - `input96.txt`
   - `output96_1_ref.txt`
   - `output96_4_ref.txt`
3. Make sure that you have execution permission of the script: `chmod u+x check_A1.sh`

4. Run the script: `./check_A1.sh`.

5. If you get the output "`Your file is ready for submission. Well done!`", you may submit the file. Otherwise, fix it according to the output you get.

If you don't get the message "`Your file is ready for submission. Well done!`" your file is **not** ready for submission, and will not be graded until you have fixed the issue(s). However, note that the message is no guarantee for passing the assignment. We will perform additional tests of you code, which we will take into consideration together with your report when we grade the assignment.

6 Deadline

The file `A1.tar.gz` must be submitted no later than April 17. Reports and code submitted after that date will not be graded. Assignments that do not meet the requirements are returned. The final version of the assignment must be submitted no later than June 7. Assignments submitted after this date will not be approved.

Happy hacking!

Malin & Maya

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