Assignment no.1
Clarification for task 2 and submission

Roman Iakymchuk, TDB, IT Dept
roman.iakymchuk@it.uu.se

March 21, 2024

1 Assigned reading

We suggest to

- Refresh your knowledge of (and skill in using) the C programming language.
- Refresh your skills in using the Linux command line interface, including secure remote terminal access and secure file transfer. This is needed to work on UPPMAX
- Read Chapters 1 and 2 (possibly 3 for MPI) in the book.

2 Assigned exercises

2.1 Formulas for block partitioning

This exercise is adapted from 1.1 in the book. Suppose that you are to compute the sum of an array of length n using p processing units. You partition the array into p blocks of near uniform size. Construct formulas for the *first* and *last* indices of block number $k = 0, 1, \ldots, p-1$. Hint: first consider the special case when $p \mid n$ (n is divisible by p).

2.2 Tree-structured global sum

This exercise is adapted from 1.3 in the book. Suppose that you have p processing units, each of which has a number. The goal is to compute their global sum using the tree-structured algorithm sketched in Figure 1.1 in the textbook. The final result should end up on processing unit 0. Construct an algorithm for processing unit $k = 0, 1, \ldots, p-1$ using the communication primitives `send` and `receive`. Note: your solution must work for *any* number of processing units (not only powers of 2, but start with powers of 2).

Tasks:

- Construct an algorithm and explain it
- Implement your solution
- Test on random numbers and different array sizes on different number of processes

2.2.1 Performance experiments

When the program is parallelized, it is time for evaluation of its parallel performance. Your parallelization is supposed to speed up the execution of the program! However, it should 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.

Inputs: generate numbers randomly from the interval. We provide you with a random number generator in the `sum.c` file

You are supposed to evaluate the strong & weak scaling of your parallel program **on Snowy**:

- For strong scalability fix the problem size to $n = 2^{22}$ (higher is also fine, eg $n = 2^{26}$) and vary the number of processes used: $1, 2, 4, 8$. Compute efficiency with respect to sequential run and make a plot
- For weak scalability fix the problem size per process to $n = 2^{19}$ and change the number of processes: $1, 2, 4, 8$. Compute efficiency with respect to sequential run and make a plot

2.3 Cost analysis of tree-structured global sum algorithm

This exercise is adapted from 1.6 in the book. Count the number of *receives* and *additions* that processing unit 0 performs in the algorithm constructed in the previous exercise. Express your solution on the form

$$T(p) = ?r + ?a,$$

where r and a are machine-specific parameters for the time it takes to do one receive and one addition, respectively. Your job is to determine what should go in the places marked with '?'.

2.4 (optional) Hardware multithreading and caches

This exercise is adapted from 2.8 in the book. Caches can improve the effective memory bandwidth and latency by serving some memory accesses from fast cache memory. Hardware multithreading can reduce the negative impact of memory latency by switching to

another (hardware) thread while the current thread waits for a memory access. But running many threads on a hardware multithreaded core might degrade the effectiveness of the cache. Figure out why. Hint: what happens with the cache while a thread is waiting for a memory access?

2.5 (optional) False sharing

This exercise is adapted from 4.17 in the book. Let A be a matrix of size $m \times n$ that is stored column-wise in an array `A` with a configurable column stride $s \geq m$. More precisely, the matrix element in (zero-based) row i and column j is stored in array index $i + js$. Suppose that we have p threads and wish to partition the matrix into p row blocks (one for each thread) with near uniform size. However, we are willing to compromise a little bit on the uniformity of the sizes if we can eliminate the possibility of false sharing. Describe how to choose the stride s and compute the sizes of the p row blocks such that false sharing is completely eliminated.

2.6 Speedup and efficiency

This exercise is adapted from 2.16 in the book. Suppose that the execution times of a sequential and parallel program (solving the same problem) are given by the functions

$$T_s(n) = n^2, \qquad T_p(n,p) = \frac{n^2}{p} + \log_2(p).$$

Produce two tables: one for speedup and one for efficiency. Label the columns by $n = 10, 20, 40, 80, 160, 320$ and the rows by $p = 1, 2, 4, 8, 16, 32, 64, 128$. Look down any particular column (i.e., keep n fixed). What happens to the speedup and the efficiency? Do you think this makes sense given that the problem size is fixed and the number of processing units increases? Now look across any particular row (i.e., keep p fixed). What happens now to the speedup and efficiency? Do you think this makes sense given that the number of processing units is fixed and the problem size increases?

2.7 Scalability

This exercise is adapted from 2.19 in the book. Suppose that the sequential and parallel execution times are given by the functions

$$T_s(n) = n, \qquad T_p(n,p) = \frac{n}{p} + \log_2(p).$$

If we increase p by a factor of $k > 1$, then by which factor do we have to increase n to maintain the same efficiency?

3 Report

You are supposed to write a report on your results. The report should contain:

1. Answers to **all tasks in assignment 1 in the Student's guide** and the task 2 clarification above
2. A brief description of your parallelization.
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 gnuplot or R, 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?

4 Files to be submitted

The file that you upload in Studium must be a compressed tar file named `A1.tar.gz`[1]. It shall contain a directory named A1, with the following files inside:

- A PDF file named A1_Report.pdf containing you report.
- Your code. 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 have downloaded from Studium):

1. Put `check_A1.sh` and `A1.tar.gz` in the same directory.
2. Make sure that you have execution permission of the script: `chmod u+x check_A1.sh`
3. Run the script: `./check_A1.sh`.
4. 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).

[1] Command to create a tar file on Linux: `tar -czvf A1.tar.gz A1`

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.

Important note: When you upload a first revised version of your work, change the name of the file as `A1-R1.tar.gz` and if a second revision is allowed - `A1-R2.tar.gz`.

Best wishes,
Roman Iakymchuk

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