Individual Project

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

May 10, 2024

1 Introduction

The purpose of this individual project is to show that you can use what you have learned to tackle a complete (albeit rather small) parallel computing problem.

In this mandatory task, each student is supposed to work individually on a parallelization of an algorithm chosen from the list, given in Section 6. Pick one of them as your own personal project. The individual project is graded ('3', '4' or '5'). The deadline for submission is given in Studium. Reports submitted within the deadline may fail due to not working or erroneously working code, or due to insufficiencies in the written report. You have the possibility to submit a supplemented version no later than **June 07, 2024**. Revised reports/ supplements will be corrected as time allows, aiming to report the grades before the end of the semester. If you miss the initial deadline and ask us to accept a belated report, we will allow you to submit it, however the final grade could not be '5'.

Each of you should present the project in brief (up to 5 minutes plus 5 minutes for questions) to one of us. This may not be the final version but a good working version. Such presentation preferably should occur before the deadline or, alternatively, within 10 days after the deadline.

We encourage you to do *peer reviewing* of both code and report but under the condition that each member of the reviewing team implements a different algorithm. This can also be added to the report and will achieve an additional bonus point, meaning if one of you fall between grades the mark will be rounded up.

2 What to do

Your task is to implement and experimentally evaluate a parallel algorithm for one particular application/ algorithm using MPI on the distributed memory cluster `snowy`.

Choose one of the possible projects, listed in Section 6 and implement it in a distributed memory setting. Write and submit a report following the instructions in Section 3.

While implementing the corresponding algorithm you should pay a particular attention to communication and load balancing. The project work includes a verification of the correctness of your solution as well as a performance evaluation using the appropriate metrics.

3 The Report

The report should be written as a scientific paper of academic quality, written using either LaTeX (preferred) in English. Plots can be created using e.g. Matlab or Python Matplotlib. Note that the previous three assignments served as preparatory steps for this individual project. The report should include the following:

1. *Introduction*, providing a background and motivation.
2. *Algorithms*
    - Describe the sequential algorithm that underlies the parallel algorithm.
    - Describe the parallel algorithm and your implementation using MPI.
3. *Experiments and results*
    - presenting how you evaluate the performance of your solution along with your results, observations and comments.
    - presenting the results in the form of parallel speedup and efficiency
4. *Discussion*, with explanations of the results and with ideas for possible optimizations or improvements.
5. *References*, listing relevant literature that was consulted in the project.

The report has to be submitted as a PDF file.

4 The Code

The code should be submitted along with the report as a compressed `zip` archive. The `zip` archive should also contain:

- A Makefile (cmake is also possible) to build the code. This Makefile should work on UPPMAX `snowy`.
- A file named README containing instructions detailed enough for the teachers to be able to build and run the code. Also describe the format on any input needed by the program.
- Input or other files that the teachers need to execute the program according to the instructions in README.

5 Grading

The project is graded with one of the usual three pass levels, namely 3,4, or 5, or a fail. When grading your work, we take into account the following aspects:

- *Solution:* Choice of approach and parallel algorithm, serial and parallel efficiency, and use of MPI functions.
- *Methodology:* Demonstration of correctness, and performance evaluation.
- *Code:* Design, robustness, documentation, and general quality.
- *Report:* Disposition, presentation of results, and language quality.

While we consider the quality of your solution, the absolute performance of your implementation will not affect the grade, as long as you present satisfactory evaluations of the correctness and performance, and an analysis of your results.

For completeness, we include the grading criteria for the report, as given in the instructions in Studium.

- Grade 3: You must demonstrate an understanding of the fundamentals of the course. Your code must work correctly and you should have made a reasonable attempt to optimize and parallelize important parts of the code. The report should obey the minimal requirements to contain the description of the problem, the solution algorithm and performance results, presented clearly in tables and graphs.

- Grade 4: You should have tried to optimize the whole code, and you should be able to reason about the performance of your code at a high level and be able to explain why certain optimizations worked/ did not work. The report should be well structured with appropriate references to theory and related work. In addition to the work for grade 3 you should also reflect on and discuss your results. The report should be clearly written and be proof-read so that there are very few typos.
- Grade 5: In addition to very good optimizations and understanding of performance issues, you must also show a higher level of understanding of the parallelized code. You should have optimized the code with respect to synchronization, data dependencies, load balance and parallel work. In your report, in addition to the work for grade 4, you should also argue why you have chosen the specific method/technique, is your solution optimal for the target problem and why. Are there other approaches, how they compare to your choice.

6 Project descriptions

- The power method
- Parallel implementation of the Conjugate Gradient method with stencil-based matrix-vector multiplication
- Monte Carlo computations, combined with the Stochastic Simulation Algorithm to simulate malaria epidemic
- Shear sort

6.1 Power method (max grade 3)

A so-called *dominant eigenvector* of a square matrix A of size $n \times n$ can be computed by the *power method*. There are numerous applications of this method. For example, Google uses the so-called *PageRank* as one factor to determine its search rankings. The PageRank scores can be computed by finding the dominant eigenvector of a particular matrix.

The power method is a simple iterative procedure. Start with a random non-zero initial vector x of length n. Then repeatedly perform the following two steps (for simplicity, let's say a fixed number of times):

1. Compute $x \leftarrow Ax$, i.e., overwrite x with the *matrix-vector product* Ax

2. Normalize x by scaling its components by

$$x_i \leftarrow \frac{x_i}{\sqrt{\sum_{k=1}^{n} x_k^2}}$$

Recall the definition of matrix-vector multiplication. Let $a_{i,j}$ denote the element of A in row i and column j and let x_j denote the j'th component of x. Then the n components y_i, for $i = 1, 2, ..., n$, of the product $y = Ax$ are defined by the equation

$$y_i \leftarrow \sum_{j=1}^{n} a_{i,j} x_j,$$

i.e., the i'th component of the product y is equal to the sum of the products of the n corresponding components of the i'th row of A and the vector x.

The input is a *sparse matrix* A of size $n \times n$ with elements of type `double` (stored in some sparse matrix storage format), and the output is a non-zero vector x of length n that approximates the dominant eigenvector of A. The square test matrices can be randomly generated or sampled from collections such as the Matrix Market (`https://math.nist.gov/MatrixMarket/`) or The Suite Sparse Matrix Collection (`https://sparse.tamu.edu/`).

6.2 Parallel implementation of the Conjugate Gradient method with stencil-based matrix-vector multiplication

Consider the solution of a linear system of equations $A\boldsymbol{u} = \boldsymbol{b}$, where A is a square nonsingular (symmetric positive definite) matrix of size $N \times N$ and $\boldsymbol{u}, \boldsymbol{b}$ are column vectors of length N,

The standard Conjugate gradient (CG) algorithm for computing $\boldsymbol{u}$ reads as follows:
In Algorithm 1, $\boldsymbol{d}, \boldsymbol{q}, \boldsymbol{g}$ are also column vectors of length N.

As we see, apart from the initialization step, each iteration consists of three vector updates, which are perfectly parallelizable (steps 6,7,10), two scalar products (steps 5,8), which require global communications and one matrix-vector operation (step 4).

The background setting is that we have a two-dimensional mesh, as shown in Figure 1, with coordinates $x_i = ih$, $y_j = jh$, $i, j = 1, 2, \cdots n$, $h = \frac{1}{n+1}$, and, thus, $N = (n)^2$. The entries of the vectors $\boldsymbol{u}, \boldsymbol{b}$ are associated with the mesh points.

The requirement is to distribute the mesh points between the processes (with no overlap). The components of the vector $\boldsymbol{b} = \{b_{ij}\}$ should be computed as $b_{ij} = 2h^2(x_i(1 -$

Algorithm 1 The CG algorithm

Initialize $\boldsymbol{u} = \mathbf{0}$, $\boldsymbol{g} = -\boldsymbol{b}$, $\boldsymbol{d} = \boldsymbol{b}$
$q_0 = \boldsymbol{g}^T \boldsymbol{g}$
for $it = 1, 2, \cdots$ until convergence **do**
    $\boldsymbol{q} = A\boldsymbol{d}$
    $\tau = q_0/(\boldsymbol{d}^T \boldsymbol{q})$
    $\boldsymbol{u} = \boldsymbol{u} + \tau \boldsymbol{d}$
    $\boldsymbol{g} = \boldsymbol{g} + \tau \boldsymbol{q}$
    $q_1 = \boldsymbol{g}^T \boldsymbol{g}$
    $\beta = q_1/q_0$
    $\boldsymbol{d} = -\boldsymbol{g} + \beta \boldsymbol{d}$
    $q_0 = q_1$
end for

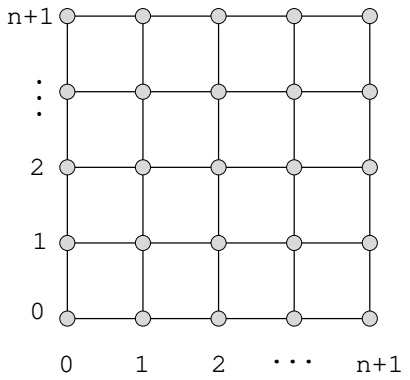

Figure 1: Two-dimensional mesh.

$x_i) + y_j(1 - y_j)$, $i, j = 1, \cdots, n$. Note that we only compute in the interior points. The particular vector $\boldsymbol{b}$ corresponds to a solution, which is zero on the boundary $x = 0, y = 0, x = 1, y = 1$.

The matrix A is defined by the following five-point stencil

$(j+1)$		-1	
(j)	-1	4	-1
$(j-1)$		-1	
	$(i-1)$	(i)	$(i+1)$

The stencil is not periodic, thus, the weights of the missing entries in the stencil along the boundary are equal to zero. Note, that the matrix A in this case is sparse, with not more that five nonzero elements per row. Thus, the computational complexity of one CG iteration is of the order of $10N$.

Implement Algorithm 1. There should be only one input parameter, n, the number of intervals along each coordinate axis. Implement the matrix-vector multiplication step in a stencil-based manner, analogously to the one-dimensional case from Assignment 2.

To simplify the setting, you may assume that your logical architecture is a mesh of size $\sqrt{p} \times \sqrt{p}$, where p is the number of processes used and the process operate on equally (or almost equally) sized parts of the mesh. For the performance study n must be chosen sufficiently large. It is suggested to start with $n = 256$, thus, $N = 65536$, and then increase n as $512, 1024, \cdots$ These sizes correspond to halving h and increasing the number of degrees of freedom N by a factor of 4. For each problem size perform 200 iterations. Think about a good load balance. Present both *fixed-size* and *weak* scalability tests, thus you should increase n and p in a way that allows you to draw relevant conclusions regarding both scalabilities.

Output of the code: the norm of the vector $\boldsymbol{g}$, thus, $\sqrt{\boldsymbol{g}^T\boldsymbol{g}}$ at the last iteration for all problem sizes you have tested, to be included as a table in the report.

6.3 Monte Carlo computations, combined with the Stochastic Simulation Algorithm to simulate malaria epidemic

The general scheme of the Monte Carlo (MC) method is the following.

Algorithm 2 The MC algorithm

1: Choose the number of MC experiments N
2: **for** $i = 1, 2, \cdots, N$ **do**
3: Perform one MC experiment and store the result in a suitable vector
4: **end for**
5: Compute some mean value or another quantity, summarizing the results.

In this case the MC experiment will be one simulation of a stochastic model of the development of an epidemic, in this case, malaria. The vector $\boldsymbol{x}$, introduced below, is the so-called *state vector* and its components correspond to the various quantities, included in the model, such as number of susceptible, exposed, infected, recovered, etc. The simulation of the model is performed via the so-called Stochastic Simulation Algorithm (SSA) and the algorithm reads as follows:

Algorithm 3 Gillespie's direct method (SSA)

1: Set a final simulation time T, current time $t = 0$, initial state $\boldsymbol{x} = \boldsymbol{x}_0$
2: **while** $t < T$ **do**
3: Compute $\boldsymbol{w} = prob(\boldsymbol{x})$
4: Compute $a_0 = \sum_{i=1}^{R} \boldsymbol{w}(i)$
5: Generate two uniform random numbers u_1, u_2 between 0 and 1
6: Set $\tau = -ln(u_1)/a_0$
7: Find r such that $\sum_{k=1}^{r-1} \boldsymbol{w}(k) < a_0 u_2 \leq \sum_{k=1}^{r} \boldsymbol{w}(k)$
8: Update the state vector $\boldsymbol{x} = \boldsymbol{x} + P(r, :)$
9: Update time $t = t + \tau$
10: **end while**

Here P is the following matrix

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

Note, that the matrix P is of size $(15, 7)$, R in step 4 in Algorithm 3 is equal to 15. The length of the vectors $\boldsymbol{x}$ and $\boldsymbol{w}$ are 7 and 15, correspondingly, $\boldsymbol{x}_0 = [900; 900; 30; 330; 50; 270; 20]$, $T = 100$. The function `prop` is given and can be downloaded.

Your task is the following.
For grade max 4:

Choose number of processes p and number of experiments $N = n * p$ for some n, thus, assume that N is divisible by p. Each process shall execute the algorithm SSA n times. Note that the number of steps in each individual SSA simulation depends on a random variable and is not known in advance. The result from each local group of n SSA runs at the final time would form a matrix $X(7, n)$, where each column corresponds to the final

vector $\boldsymbol{x}$ at time T. So far, the algorithm is perfectly parallelizable. After all processes have finished their simulation, the following should be done. We are interested in the distribution of the first component of $\boldsymbol{x}$ (susceptible humans), thus we want to plot a histogram of $\boldsymbol{x}(1, 1 : N)$ with b bins, where $b = 20$. The intervals, related to the bins are to be determined, based on the minimal and the maximal values of $\boldsymbol{x}(1, 1 : N)$.

Perform the necessary collection of the results (within the MPI program) and plot a histogram. The plotting itself can be done, for instance, in Matlab. For large enough N the histogram should resemble some normal distribution. When preparing the results for the histogram use as much parallelism as possible.

Study the performance of your implementation as a function of N and p. Design a good way to relate N and p in a way that allows you to draw relevant conclusions regarding both *fixed-size* and *weak* scalability.

Output: for three values of N, larger than 10^6 provide the bounds of the intervals in the histogram and a plot of the histogram itself.

For grade 5:
Upgrade the code with the following functionality:
All processes work fully in parallel, however, as the time steps are chosen randomly, the processes proceed to reach the final time asynchronously. Record the wall clock time after passing time $25, 50, 75, 100$ (per process) so that at the end you can output the averige time per processor for each time sub-interval. You are encouraged to use the MPI one-sided put/get functionality.

6.4 Shear sort

Assume that you have to sort N numbers, which are given in a matrix form, thus, $N = n^2$ and the numbers reside in a matrix A of size (n, n). Implement the *shear sort* algorithm, as discussed during the lecture. The description can be found in the notes and also in online sources. For completeness, the algorithm is included below, using Matlab notation.

After the end of the algorithm, the data resides 'snake-wise' in the matrix A, as in the following illustration:

$$\text{Before: } \begin{bmatrix} 15 & 9 & 6 & 16 & 17 \\ 1 & 8 & 14 & 6 & 6 \\ 6 & 16 & 14 & 11 & 17 \\ 1 & 16 & 4 & 14 & 5 \\ 2 & 4 & 3 & 18 & 19 \end{bmatrix} \quad \text{After: } \begin{bmatrix} 1 & 1 & 2 & 3 & 4 \\ 6 & 6 & 6 & 5 & 4 \\ 6 & 8 & 9 & 11 & 14 \\ 16 & 16 & 15 & 14 & 14 \\ 16 & 17 & 17 & 18 & 19 \end{bmatrix}$$

Algorithm 4 Shear sort

1: Initialize: the data to be sorted resides in a square matrix $A(n, n)$ and is distributed among the processes
2: Compute `d=ceil(log2(n))` (the number of steps is $d + 1$)
3: **for** $l = 1 : d + 1$ **do**
4: **for** $k = 1 : 2 : n$ **do**
5: $A(k, :) = sort(A(k, :), 'ascend')$ Sort odd rows in ascending order
6: **end for**
7: **for** $k = 2 : 2 : n$ **do**
8: $A(k, :) = sort(A(k, :), 'descend')$ Sort even rows in descending order
9: **end for**
10: **if** $l <= d$ **then**
11: **for** $k = 1 : n$ **do**
12: $A(:, k) = sort(A(:, k), 'ascend')$ Sort all columns in ascending order
13: **end for**
14: **end if**
15: **end for**

After you have insured the correct functioning of the implementation, you must design a performance study. Present both *fixed-size* and *weak* scalability tests, thus you should increase n and p in a way that allows you to draw relevant conclusions regarding both scalabilities.

Input/ output: provide a possibility to check the correctness of your code for a small-sized matrix, say of order 15 and 16.

7 File to be submitted

The file that you upload in Studium must be a compressed tar file named `Proj.zip`. It shall contain a directory named `Proj`, with the following files inside:

- A PDF file named `Proj_Report.pdf` containing you report.
- Your code, following the same specifications as for the assignments. The code shall compile on UPPMAX `snowy` without warnings.
- A Makefile that does the following:

    1. Builds a binary named correspondingly to which project you work on from your code when the command `make` or `cmake` are invoked. This must work

on UPPMAX `snowy`!

2. Removes all binary files and object files (if any) when the command `make clean` is invoked.

- A file `README` with instructions how to run the code, regarding the input parameters, structure of the input file, etc.

Note: When you upload a first revised version of your work, change the name of the file to `Proj-R1.zip` and if a second revision is allowed – `Proj-R2.zip`, similarly, for the directory.

8 Tips

- Writing takes time and any text must undergo numerous revisions before its quality becomes acceptable. Therefore, start writing *from day one* and write new text and *revise old text every day*.
- The latency of the batch system on `snowy` can be long and unpredictable. Therefore, *have your software ready for preliminary testing* at least one week before the deadline. Plan ahead and *choose the inputs to your experiments based on what you need for your report* to meet the requirements rather than the other way around, i.e., writing the report based on the data you have available.
- Regularly use the assessment criteria as a tool. This will help ensure that what you submit will pass without any problems. None of the criteria are negotiable and high standards will be enforced, particularly when it comes to the readability of the source code and the report.
- Ask friends and/ or foes for feedback. Several times and early in the process, if possible. Once again, we encourage peer reviewing.
- Plan your work ahead: break the big task into small; set up deadlines; use task/ time management tools like notion etc.