Tools of high performance computing 2018

Final project

In the final project you are supposed to write a parallel program based either on *your own problem* or on one of the problems given on the following pages. In the case of your own problems you should first discuss with the lecturer to see if its is suitable as a final project.

The deadline for submitting the report to Moodle is 25.5.2018.

You write a report of about 10 pages on your work. The report should consists of

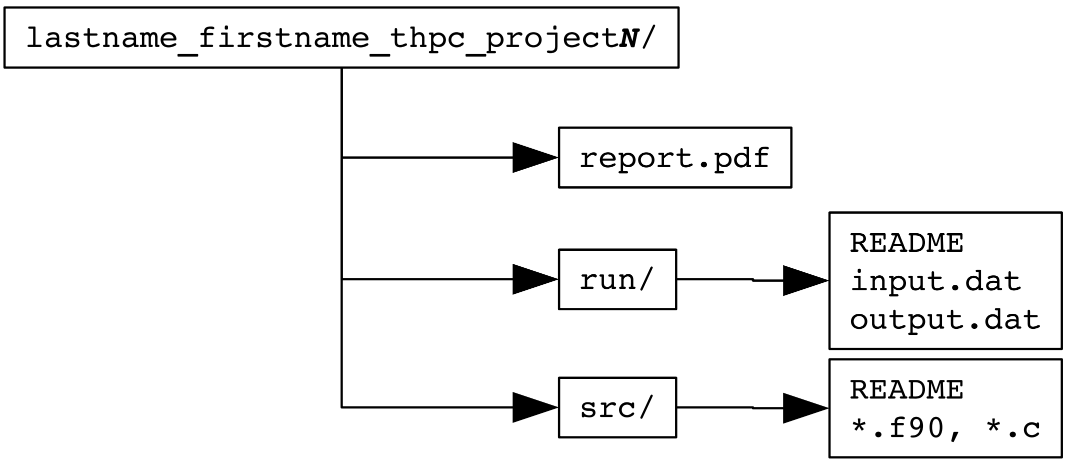
1. short introduction to the problem
2. description of algorithms used
3. principles of parallelization
4. presentation of the code
5. instructions for using the code
6. principles and results of benchmarking (scaling behavior)
7. conclusions

Send the report in PDF format. As in the case of exercises, you should also send the full source code, possible Makefile and sample input and output files. However, source code listing may not be included in the report.

Submit your report and code to Moodle in one file as a `tar` of `zip` archive named as

`lastname_firstname_thpc_project`*`N`*`.tgz` or
`lastname_firstname_thpc_project`*`N`*`.zip`

where *`N`* is the project number (see below; if you have your own title the number is 3) and *containing a folder with the same name where all the files can be found*. This folder in turn, must contain the report (file `report.pdf`) and subfolders `src/` [source code and file `README` containing short compilation instructions] and `run/` [input file(s) and example output and file `README` containing short run instructions].



You may write the program either in Fortran or C/C++. For parallelization you may use either MPI, OpenMP or threads[1]. *Remember to ensure that the parallel code solves the problem faster than the serial one.*

1. Optimization using a parallel genetic algorithm: traveling salesman problem

The optimization problem is either the 2D traveling salesman problem (TSP) or one of your own interest. In genetic algorithms (GAs) the optimization is achieved by maintaining populations of possible solutions and letting only the fittest individuals of these populations survive and reproduce. Material on GAs can be found in pdf file `md_course_lectures.pdf`[2].

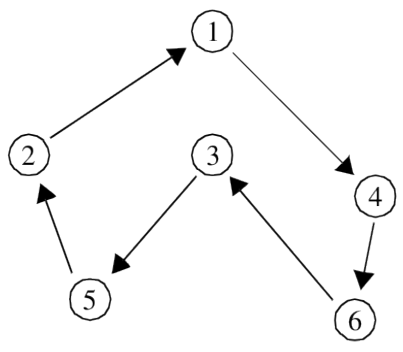
In TSP one has to find a route through a given set of cities with the shortest possible length.

If we denote with $\{x_i, y_i\}$, $i=1\ldots N$ positions of the cities we have to minimize the length L :

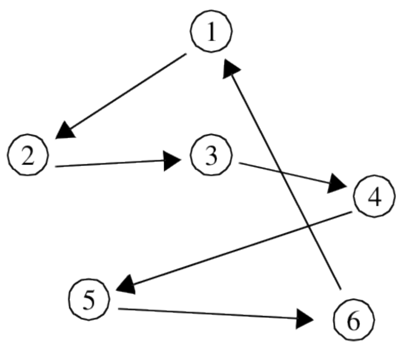
$$L^2=\sum_{i=2}^{N}\left[\left(x_{R(i+1)}-x_{R(i)}\right)^2+\left(y_{R(i+1)}-y_{R(i)}\right)^2\right]$$

where $R(i)$, $i=1\ldots N$ is the order in which the cities are traversed[3] . When applying GA to TSP we need a way to encode the route as a string. The most obvious way is to list the cities in the order they are visited in the route; i.e. the array $R(i)$. Below are shown two examples of individuals of a certain set of cities.

$R_1=(1,4,6,3,5,2)$



$R_2=(1,2,3,4,5,6)$



The problem in the TSP is that it is not straightforward to introduce breeding. A simple way would be to choose randomly a part of the route and swap corresponding parts in the parents to create two children. However, the children may have duplicate cities so that they are not necessarily valid routes (there are cycles in the route). Instead of randomly creating the children one can use heuristics. One possible

1 Or GPU programming if you want a challenging project.

2 In addition you may Google "TSP-kbryant-2001-thesis.pdf" and, of course take a look at the Wikipedia article at https://en.wikipedia.org/wiki/Genetic_algorithm.

3 Well, to be precise we need only to specify $N-1$ elements in the route because we must return to the place where we started.

algorithm to create one child from two parents is the following:

1. Take as the first city of the child the first one from either of the parents.
2. Choose as the second city of the child the one of the corresponding cities in the parents that is closer to the first city in the child.
3. If the new city is already included in the child choose the second city of the other parent. If this is also included in the child then choose the next city from either of the parents randomly (and in such a way that it does not introduce a cycle).
4. Go in a similar fashion through all the cities until the child has all cities.

Lets take a simple example [`d(i,j)` is the distance between cities `i` and `j`]:

```
R1=123456
R2=413265
Choose to start from R2(1) → R3=4XXXXX
Next one is either 5 or 1. Assume d(4,1)<d(4,5) → R3=41XXXX
Next one is either 2 ot 3. Assume d(1,2)<d(1,3) → R3=412XXX
Next one is either 3 or 6. Assume d(2,6)<d(2,3) → R3=4126XX
Next one is either 1 or 5. 1 is already chosen  → R3=41265X
The last one is 3                               → R3=412653
```

Note that periodic boundary conditions are used. That is obvious because the starting point is arbitrary: the length of the route does not depend on where it is started. So individual s

`R1=123456` and `R2=561234`

are equivalent.

Mutation of the child is performed by exchanging two cities.

GA may be parallelized either by distributing one population to processors or by letting each processor to compute its own population. In the latter case mixing of genes between the separate populations is achieved by migration[1] . In this project we choose the latter way of parallelization. In the stepping-stone model each processor sends a few of its best individuals to one of its nearest neighbors (as defined by the processor array topology) at certain intervals. In this way the amount of communication is small and the scaling behavior of the method should be good. Of course, some global communication is necessary for obtain ing global results at certain intervals and at the end of the run.

Your task: Implement the parallel TSP-GA algorithm and test its performance and scaling as a function of the number of processors. The IO and collecting of the data is handled by the root processor. In the beginning the positions of the cities are read in and in the end of the run the best route and its length is printed. In order to monitor the run it is advisable to print the so far best solution at regular intervals.

1 J. Nang, K. Matsuo: A Survey of the Parallel Genetic Algorithms, Research report IIAS-RR-93-7E, Fujitsu Laboratories Ltd. (1993). (You can find the report in the Kumpula Science Library.)

2. Parallel algorithms for solving partial differential eqations: Poisson's equation in two dimensions

Assume we have the Poisson's equation in the unit square[1]:

$$\frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y) = g(x,y)\,, \quad (x,y) \in [0,1]^2 \,.$$

The function $g(x,y)$ is known. In addition to the boundary conditions dictate the function values at the unit square boundaries. The problem is discretized so that the unit square is divided to N parts in both directions:

$$f_{i,j} = f\left(\frac{i}{N}, \frac{j}{N}\right), \quad 0 \le i, j \le N$$

$$g_{i,j} = g\left(\frac{i}{N}, \frac{j}{N}\right), \quad 0 \le i, j \le N \,.$$

The second derivative can be computed by the central difference approximation:

$$\frac{\partial^2}{\partial x^2} f(x,y) \approx \frac{1}{\Delta^2}\left[f(x+\Delta,y) - 2f(x,y) + f(x-\Delta,y)\right], \quad \Delta = \frac{1}{N} \,.$$

A similar equation holds for y derivative and we get from the equation for the interior points

$$f_{i,j} = \frac{1}{4}\left[f_{i+1,j} + f_{i-1,} + f_{i,j+1} + f_{i,j-1}\right] - \frac{1}{4N^2} g_{i,j} \,.$$

This is a system of $(N-1)^2$ equations and $(N-1)^2$ unknowns (boundary conditions give the function values on boundaries). This means that iterative methods developed for linear systems can be used to solve it. In the Jacobi over relaxation (JOR) method the iteration goes as

$$f_{i,j}(t+1) = (1-\gamma) f_{i,j}(t) + \frac{\gamma}{4}\left[f_{i+1,j}(t) + f_{i-1,j}(t) + f_{i,j+1}(t) + f_{i,j-1}(t)\right] - \frac{\gamma}{4N^2} g_{i,j}$$

$$0 < i, j < N\,, \quad 0 < \gamma < 1 \,.$$

Here γ is so called over relaxation parameter and its optimal value is best found by experimenting. Note that 'time' t is here only an iteration index.

Successive over relaxation (SOR) uses the Gauss-Seidel iteration algorithm:

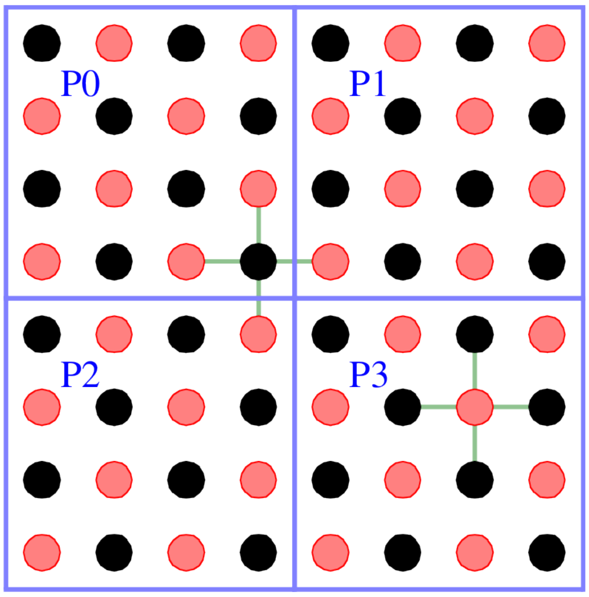
1 See e.g. paragraph 2.5 in the on-line book at http://web.mit.edu/dimitrib/www/pdc.html or chapter 19 of Numerical Recipes.

$$f_{i,j}(t+1)=(1-\gamma)f_{i,j}(t)+\frac{\gamma}{4}\left[f_{i+1,j}(t)+f_{i-1,}(t+1)+f_{i,j+1}(t)+f_{i,j-1}(t+1)\right]-\frac{\gamma}{4N^2}g_{i,j}\ .$$

Note that this equation is meant to express the fact that in each iteration the newest available information is used (i.e. if there is data for $t+1$ then it is used). This means that the order in which the lattice points are traversed is important. The equation above describes a serial algorithm and can not be parallelized as such. The SOR algorithm converges fast but as in the case of JOR the optimum value of parameter γ is best found by experimenting.

Parallelization of JOR is straightforward: just do the domain decomposition of the unit square by giving each processor a small rectangle for computation. Communication between processors is needed when computing the new values of f at the boundaries of each processor's domain.

On the other hand, because of the dependencies parallelization of SOR is not so easy. One way to do it is so called red-black algorithm. In it, the lattice is divided into two sublattices by coloring every second point with red and black (see below). The algorithm first updates the red sublattice. This can be done in parallel by using the domain decomposition. Then the black points are updated using the recently calculated red points. Thus, the updating is done an alternating fashion for each sublattice at a time so that all neighbors of a lattice point have their ' $t+1$ ' values (hmm...in a way).



Convergence criterion for the iteration may be based on the difference between two successive iterations.

Your task: Implement the parallel Poisson's equation solver by using the SOR algorithm and test its scalability. Do the scaling tests for many system sizes. Note that you must first find out the optimum value of the over relaxation parameter and that its value depends on the grid size.