PAP laboratory assignments
Lab 4: Heat equation using MPI

Lluc Álvarez and Eduard Ayguadé

Spring 2019-20

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

1

Sequential heat difussion program

In this session you will work on the parallelization of a sequential code (`heat.c`[1]) that simulates heat diffusion in a solid body using the Jacobi solver for the heat equation. The picture below shows the resulting heat distribution when two sources of heat source are placed in the surface (one on the upper left corner and the other on the middle of the lower line). The program is executed with a configuration file (see `test.dat` and `eval.dat` files) as parameter that specifies the maximum number of simulation steps (`iterations`), the size of the body (`resolution`) and the heat sources, their position, size and temperature. The program generates performance measurements and a file `heat.ppm` providing the solution as image (as portable pixmap file format), a gradient from red (hot) to dark blue (cold).

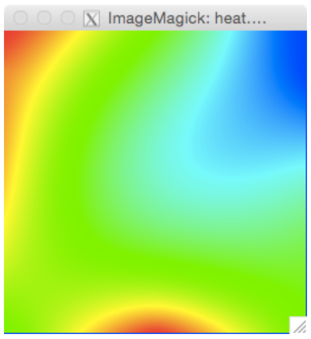

Figure 1.1: Image representing the temperature in each point of the 2D solid body

1. Compile the sequential version of the program using `"make heat"` and execute the binary generated (`"./heat test.dat"`). The execution reports the execution time (in seconds), the number of floating point operations (Flop) performed, the average number of floating point operations performed per second (Flop/s), the residual and the number of simulation steps performed to reach that residual. Visualize the image file generated with an image viewer (e.g. `"display heat.ppm"`) and copy `heat.ppm` to a different name, e.g. `heat-sequential.ppm` for validation purposes.

[1]Copy the file in `/scratch/boada-1/pap0/sessions/lab4.tar.gz`.

2 Message-passing parallelization with `MPI`

Next you will parallelize the heat diffusion sequential code using a distributed-memory (message-passing) paradigm: `MPI`. We suggest that you proceed through the following versions in order to get to the final code:

1. We provide you with an initial parallel code in `heat-mpi.c` and `solver-mpi.c`. Open the files and understand the skeleton for the MPI parallelization. On one side, the master first reads the configuration file, allocates memory and provides the necessary information to the workers. Then it computes the heat equation on the whole 2D space, reports the performance metrics and generates the output file. On the other side, each worker receives from the master the information required to solve the heat equation, allocates memory, performs the computation on the whole 2D space and finishes. Observe that this version doesn't benefit from the parallel execution since workers replicate the work done in the master. Compile this version using `make heat-mpi` and submit a MPI job (`"qsub -pe mpich 1 -l execution2 submit-mpi.sh 1 4"`, being 1 the number of nodes and 4 the total number of processes to be used in the parallel execution). Compared to the sequential code, we have commented the line that checks convergence of the solution, in other words, the line that decides how many iterations of the while loop have to be performed; therefore all processes will do the same number of iterations (`maxiter`, defined in the configuration file).

2. Modify the parallel code so that the master and each worker solve the equation for a subset of consecutive rows (i.e. *resolution/numprocs*, as shown in Figure 2.1 for an image of 256x256 plus 1 row/column halo, distributed on 4 processes). Now workers should return the part of the matrix they have computed to the master in order to have the complete 2D data space. Compile and execute the binary generated as done before. This version should not generate a correct result (check it by visualizing the `.ppm` file generated and by comparing it with the original sequential `.ppm` file, using the `diff` command) since communication during the execution of the solver is not performed in each iteration of the `iter` while loop.

3. Add the necessary communication so that at each iteration of the `iter` while loop the boundaries are exchanged between consecutive processors. Use the most appropriate communication routine(s) for that. Compile and execute the binary generated as done before. This version should generate a correct solution, although not necessarily the same as the one generated by the sequential execution because the total number of iterations done in each process is controlled by the `maxiter` parameter and its local `residual`. Check it again by visualizing the `.ppm` file generated. If you compare the output `ppm` file that is obtained with the MPI version against the sequential one (using the `diff` command) you will realize that the solutions obtained are different; this is due to the fact that in the MPI version the convergence test has been commented, so a different number of iterations is performed. Let's fix this!

4. Uncomment the line that checks convergence, compile and execute again. Why the program is not working now?

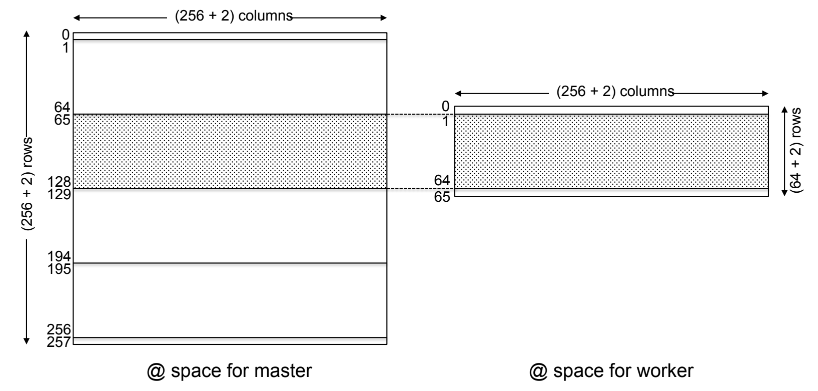

Figure 2.1: Global and local storage for matrix `u`.

5. Add the necessary communication with the master so that the total number of iterations done is also controlled by the value of the **global** `residual` variable in all workers as it is in the sequential code. Use the most appropriate communication routine(s) for that. Compile and execute the binary generated as done before. This version should generate the same result as the one generated by the sequential code.

Now that you have a correct parallel version of the code, you should proceed with its performance evaluation. The input file `eval.dat` has to be used for the evaluation, either changing the execution scripts or copying `eval.dat` on `test.dat`. We suggest the following steps for the evaluation:

1. Execute the `MPI` binary for two different configurations and compare the execution time that is reported: a) 4 processes all in a single node; and b) 4 processes distributed in two nodes. Use the `submit-mpi.sh` script to do the execution; as you can see in the script, the two arguments are the number of nodes and the total number of processes to be used in the parallel execution. Do you observe any difference in the execution time of both runs? Reason about the difference in execution time observed for the two different configurations.

2. Produce a scalability plot showing execution time and speed–up (with respect to the sequential execution time) for executions using 1 and 2 nodes and 1, 2, 4 and 8 processes en each node. Reason about the results obtained and if necessary support your explanations with some *Paraver* analysis.

3

Let's go hybrid!

We propose to use a hybrid parallelization scheme with both `MPI` and `OpenMP`. We provide you with the appropriate entry in the Makefile and scripts to support the execution of hybrid programs (`submit-hybrid.sh`). We recommend you to use your already working `MPI` version as a starting point for the implementation of the hybrid version. To do that, copy the files `heat-mpi.c` and `solver-mpi.c` on `heat-hybrid.c` and `solver-hybrid.c` and then add the `OpenMP` annotations. The programming part should not represent too much effort :). Then proceed with the same performance evaluation you have done for the pure `MPI` version, reasoning about the results obtained. Is it better to use `MPI` processes inside a node or `OpenMP` threads? Compare results with those reported in the previous section. Instrument with *E*xtrae and visualize with *Paraver* whenever you consider necessary to understand what is happening.

4

What to deliver?

For the evaluation of the **technical development** in this laboratory assignment you will have to deliver a compressed `zip` or `tar.gz` file that includes the different final `MPI` and hybrid `MPI/OpenMP` versions of the original program that computes the heat equation and a *PDF* document which should contain:

1. An explanation of the parallelization strategy used and the communication inserted in the code, explaining why did you use a point–to–point or a collective communication routine in each case.
2. The performance analysis of the different `MPI` and hybrid `MPI/OpenMP` parallel codes, including all you consider necessary (performance tables or plots, information returned by the `time` command, *Paraver* captures, ...) to support your conclusions in the most illustrative and accurate way.

For the evaluation of the **transversal competence** we will consider the following aspects:

- *Quality and accuracy* of the document delivered, with especial emphasis in the experimental evaluation part.
- Any optional aspect that you decided to explore along the development of this laboratory assignment.