Prof. Dr. Christoph Pflaum
Dominik Thönnes

Winter Term
2022/2023

Simulation and Scientific Computing
Assignment 3

The Conjugate Gradient Method and MPI Parallelization

1. Compute the solution of the elliptic partial differential equation (PDE):

$$-\Delta u(x,y) + k^2 \cdot u(x,y) = f(x,y), \quad (x,y) \in \Omega, \tag{1}$$

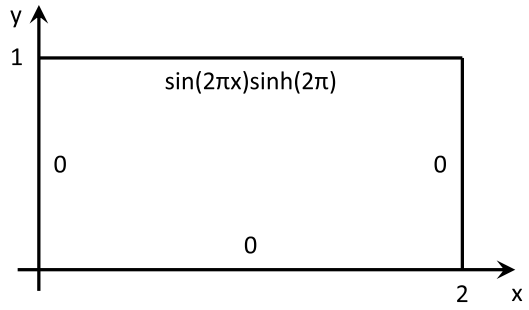
with $\Omega = [0,2] \times [0,1]$. In this assignment, $k = 2\pi$ and the right-hand side is defined as

$$f(x,y) = 4\pi^2 \cdot \sin(2\pi x) \cdot \sinh(2\pi y).$$

On the boundary $\partial\Omega$, use

$$\begin{aligned} u(x,1) &= \sin(2\pi x) \cdot \sinh(2\pi), \\ u(x,0) = u(0,y) &= u(2,y) = 0, \end{aligned}$$

which results in the following setup:



Use a finite difference approximation to discretize Eq. (1). Choose the mesh sizes h_x and h_y of the grid according to the parameters n_x and n_y (number of grid intervals in x- and y-direction, respectively) supplied on the command line:

$$h_x = \frac{2}{n_x}, \qquad h_y = \frac{1}{n_y}.$$

In contrast to assignment 2, the LSE derived from the finite difference scheme, $\boldsymbol{A} \cdot \boldsymbol{u} = \boldsymbol{f}$, is to be solved with the conjugate gradient method (CG), as specified in Algorithm 1, with double precision. Print the computation time (with `Timer.h`), number of CG iterations and the final

residual to the screen. Finally, write the computed solution to a file named `solution.txt`. Choose a file format that can be visualized by `gnuplot`[1].

Algorithm 1 Conjugate Gradient Algorithm

$\boldsymbol{r} \leftarrow \boldsymbol{f} - \boldsymbol{A} \cdot \boldsymbol{u}$
$\delta_0 \leftarrow \boldsymbol{r}^T \boldsymbol{r}$
if $\|\boldsymbol{r}\|_2 \leq \varepsilon$ **then**
  STOP
end if
$\boldsymbol{d} \leftarrow \boldsymbol{r}$
for number of iterations **do**
  $\boldsymbol{z} \leftarrow \boldsymbol{A} \cdot \boldsymbol{d}$
  $\alpha \leftarrow \delta_0 / (\boldsymbol{d}^T \boldsymbol{z})$
  $\boldsymbol{u} \leftarrow \boldsymbol{u} + \alpha \cdot \boldsymbol{d}$
  $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha \cdot \boldsymbol{z}$
  $\delta_1 \leftarrow \boldsymbol{r}^T \boldsymbol{r}$
  if $\|\boldsymbol{r}\|_2 \leq \varepsilon$ **then**
    STOP
  end if
  $\beta \leftarrow \delta_1 / \delta_0$
  $\boldsymbol{d} \leftarrow \boldsymbol{r} + \beta \cdot \boldsymbol{d}$
  $\delta_0 \leftarrow \delta_1$
end for

$\|\boldsymbol{r}\|_2$ is the discrete L2-norm of the residual also used in the previous assignment. Use zero values as initial solution. Also apply standard optimization techniques you have already used in the first assignment to improve the performance of your code. For an efficient implementation, use suitable data structures as in the previous assignment.

2. Parallelize your implementation using the Intel MPI Library version 2021 and the `mpicxx` compiler with g++ 8.5.0 (`module load intelmpi`). Make sure that the domain partitioning works reliably for different numbers of processes and different grid sizes. Test it on the Meggie cluster [2] by supplying the following job script to the job queue via `sbatch <jobScriptName>` on the cluster:

```
#!/bin/bash -l

#SBATCH --job-name=<YourJobName>
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=20
#SBATCH --time=00:30:00
#SBATCH --mail-user=<YourEmailAddress>
#SBATCH --mail-type=END,FAIL
#SBATCH --output=<JobName>
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV
```

```
module load intelmpi

cd <PathToCodeDir>
mpirun -np <N> ./cg <nx> <ny> <c> <eps>
```

3. Your program must be callable in the following form:

   `mpirun -np N ./cg nx ny c eps,`

   where `N` is the number of used processes, `cg` is the name of your executable and `c` is the maximal number of CG iterations. The primary stopping criterion for the CG method is the residual threshold specified by `eps`. Using a negative value for `eps` should thus always lead to `c` CG iterations without early stopping.

4. Performance evaluation

   Provide a PDF file with two *well–explained* performance graphs illustrating the parallel performance of your parallelization. Assign 1) the parallel efficiency and 2) the speed-up to the ordinate and the number of CPUs to the abscissa. Test your program with 1, 2, 5, 10, 20, 40, 60 and 80 processes on the Meggie cluster, i.e. with up to four nodes. Use the parameters `nx` = `ny` = 4096, `c` = 100 and `eps` = -1. Of course, make sure that your program computes the correct results!

5. Theoretical part

   When parallelizing, 1 or 2-dimensional fields of elements are often divided into multiple parts of equal size. They are then distributed over multiple processes that iteratively compute new values of the elements. This, of course, requires values of neighboring elements from the last iteration and, thus, the exchange of data between the processes. Figure 1 illustrates this for the one-dimensional case.

   In Figure 1 (a), only one element is sent from a process to its neighboring process (the dashed boxes). This enables the computation of the new values of the region (the gray boxes). Then the next data transfer is necessary. One problem that arises in this context, however, is that the time needed for the transfers, the parallel overhead, is often dominated by the transfer latency. In this case, it is possible to decrease this overhead by transferring more than one edge element at a time. Then, multiple time steps can be computed without further data transfers at the cost of updating some of the received values as well.

   Figure 1 (b) illustrates this approach for 2 exchanged elements per neighbor (the dashed white boxes): In the first step after the transfer, the process can update the points of its region and additionally 2 of the received points (the dashed grey boxes). These extra updates are necessary for the second step in which all points of the region are updated. In consequence, the parallel overhead is given by the sum of the time required for the transfers and the time spent on the additional calculations.

   Complete the following tasks for the 1D case

   (a) Give a formula for the parallel overhead per process per iteration with respect to

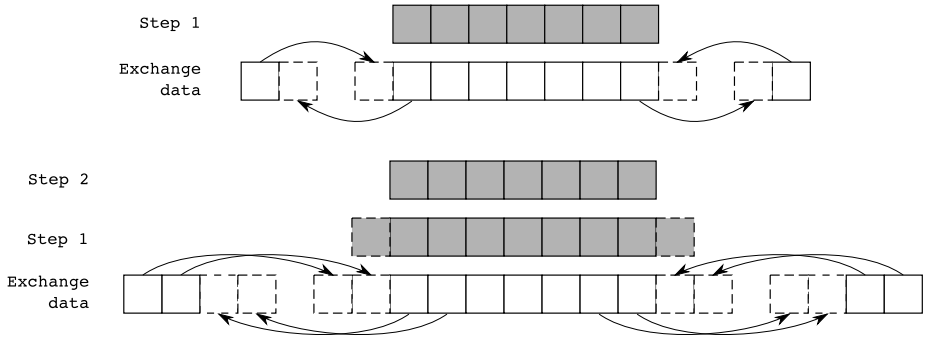


Figure 1: Communication time can be reduced by replicating computations. (a) Passing a single edge element allows the algorithm to proceed only a single time step for each communication. (b) If two edge elements are passed, the algorithm can proceed two time-steps for each communication, at the additional cost of two extra computations.

- k - the number of elements exchanged between two neighbor processes in one transfer
- α - the latency for one transfer
- β - the bandwidth for transfers between two neighbor processes
- χ - the compute time for the update of one element

(b) Determine the optimal integer for k which minimizes the parallel overhead and the resulting overhead for the following (fictional) setup:

- $\alpha = 2$ ms
- $\beta = 30 \frac{\text{elements}}{\text{ms}}$
- $\chi = 0.2$ ms

Hand in your solution on StudOn before the deadline. Make sure the following requirements are met:

- The program must be compilable with a `Makefile` or a `CMakeLists.txt`. Your program must compile with neither errors nor warnings on the Meggie cluster with the following g++ compiler flags:

  ```
  -std=c++17 -Wall -Wextra -Wshadow -Werror -O3 -DNDEBUG
  ```

  You may add additional compiler flags.
  The reference compiler is GCC version 8.5.0 on the Meggie cluster. You can check the version by typing `g++ --version`.

- The program must be callable as specified in 3.

- Include a specialized job script that can be used to run your code on the Meggie cluster.

- Check your solution against the reference implementation (binary on StudOn).

- Your code must be well-commented source files and instructions on how to use your program in a **README file**. The performance evaluation and the theoretical part must be submitted in PDF format. Upload your solution to StudOn as a team submission.

Performance challenge

Every code is compiled with mpicxx and the compiler flags specified above. The program is run several times on the Meggie cluster using 4 nodes. We will use the parameters `nx` = `ny` = 10000, `c` = 100 and `eps` = -1. The fastest team will be awarded with an extra credit point.

Credits

In this assignment, points are awarded in the following way:

1. Up to four points can be obtained if your program correctly performs the above tasks and fulfills all of the above requirements. Submissions with compile errors will lead to zero points! The cluster nodes on the Meggie cluster act as reference environments.
2. One point is given for the performance evaluation and the correct answer of the theoretical task.
3. One bonus point is awarded to the winner of the performance challenge.

References

[1] `http://www.gnuplot.info/docs/gnuplot.pdf`

[2] `https://hpc.fau.de/systems-services/documentation-instructions/clusters/meggie-cluster/`