Prof. Dr. Christoph Pflaum
Florian Schornbaum
Christian Kuschel

Winter Term
2015/2016

# Simulation and Scientific Computing
# Assignment 3

Exercise 3 (*The Conjugate Gradient Method and MPI Parallelization* )

## Tasks

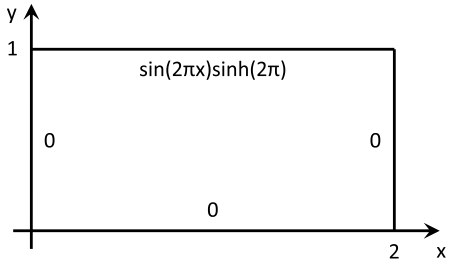
1. Compute the discrete solution of the following elliptic partial differential equation (PDE) from the last assignment:

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

where $k = 2\pi$ and the right-hand side is defined as $f(x,y) = 4\pi^2 \sin(2\pi x)\sinh(2\pi y)$. The domain is defined as $\Omega = [0,2] \times [0,1]$. On the boundary $\partial\Omega$ use

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

which results in the following setup:




To solve the PDE (1), apply a finite difference discretization. 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:

$$h_x = \frac{2}{n_x}, \qquad h_y = \frac{1}{n_y}. \tag{3}$$

In contrast to Assignment 2, you are to solve the LSE arising from the finite difference scheme with the Conjugate Gradient method (CG) and parallelize it using the Message Passing Interface (MPI).

2. To compute the solution of the PDE (1), implement a Conjugate Gradient (CG) solver. Use zero values as initial solution.

   Algorithm 1 gives an overview of the CG algorithm where $(a, b) := a^T b$ denotes the standard scalar product for vectors $a$ and $b$ and $\|r\|_2$ is the discrete L2-norm of the residual also used in the previous assignment.

**Algorithm 1** Conjugate Gradient Algorithm

$r = f - Au$
$\delta_0 = (r, r)$
if $\|r\|_2 \leq \epsilon$ stop
$d = r$
**for** number of iterations **do**
  $z = Ad$
  $\alpha = \frac{\delta_0}{(d,z)}$
  $u := u + \alpha d$
  $r := r - \alpha z$
  $\delta_1 = (r, r)$
  if $\|r\|_2 \leq \epsilon$ stop
  $\beta = \frac{\delta_1}{\delta_0}$
  $d = r + \beta d$
  $\delta_0 := \delta_1$
**end for**

   Consider the standard programming techniques you already used in the first assignment in order to improve the performance of your code. For an efficient implementation, you should also keep in mind the data structures used in the previous assignment.

3. Parallelize your implementation using the Message Passing Interface (MPI) and test it on the `LSS cluster` [2]. Running a MPI program on the cluster is done by the following steps:

   (a) Log on to the cluster machine.

   (b) Start your program on the cluster with a PBS script in the following form:

```
#!/bin/bash -l
#PBS -N <YourJobName>
#PBS -l nodes=<Nodes>:ppn=32
#PBS -q siwir
#PBS -l walltime=0:05:00
#PBS -M <YourEmailAddress> -m abe
#PBS -o $PBS_JOBNAME.out -e $PBS_JOBNAME.err

. /etc/profile.d/modules.sh
module load openmpi/1.6.5-ib
module load gcc/4.9.2

cd <PathToBinaryDir>
make clean
make
mpirun -np <N> ./cg <nx> <ny> <c> <eps>
```

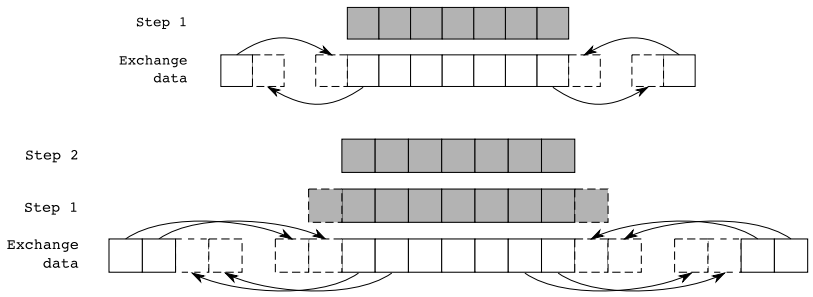



Figure 1: Communication time can be reduced by replicating computations. (a) Passing a single edge element allows the algorithm to proceed only a single time step for each communication. (b) If two edge elements are passed, the algorithm can proceed two time steps for each communication, at the additional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

4. Your program must be callable in the following form:
   *mpirun -np N cg nx ny c eps*, where *N* is the number of processes used, *cg* is the name of your executable and *c* is the number of iterations the CG method is supposed to perform in the worst case. The primary stopping criterion for the CG method is the residual threshold specified by *eps*.

5. At the end of the computation, your program should write on the standard output in one line the number of needed CG iteration steps and the final residual, together with the required runtime (use Timer.h from previous assignments). Finally, write the computed solution to a file named solution.txt. Choose a file format that can be visualized by `gnuplot` [1].

6. After loading modules `openmpi/1.6.5-ib` and `gcc/4.9.2`, use `mpic++` as compiler. Be sure to compile your program on cluster nodes (that's why there is a `make` and `make clean` in the PBS script) and not on the login node.

7. Use double precision for your computations.

8. Also answer the following theoretical part:
   When parallelizing, 1 or 2 dimensional fields of elements are often divided into multiple parts of the same 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 1 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 grey 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 depending on:
- $k$ - the number of elements exchanged between two neighbor processes in one transfer
- $\alpha$ - the latency for one transfer
- $\beta$ - the bandwidth for transfers between two neighbor processes
- $\chi$ - the compute time for the update of one element

(b) Determine the best value for $k$ (i.e. the value that 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

9. Show that the eigenvectors $\boldsymbol{v}_{\nu,\mu}$ and corresponding eigenvalues $\lambda_{\nu,\mu}$ of the finite difference discretized Laplace operator $\boldsymbol{A}$ (i.e. $-\Delta \boldsymbol{u} \approx \boldsymbol{A}\boldsymbol{u}$) satisfy

$$\boldsymbol{A} \cdot \boldsymbol{v}_{\nu,\mu} = \lambda_{\nu,\mu} \cdot \boldsymbol{v}_{\nu,\mu}$$

with

$$\begin{aligned} \lambda_{\nu,\mu} &= \frac{4}{h^2}\left(\sin^2\left(\frac{\pi\nu h}{2}\right) + \sin^2\left(\frac{\pi\mu h}{2}\right)\right) \\ \boldsymbol{v}_{\nu,\mu} &= \Big(\sin(\pi\nu x_i)\cdot\sin(\pi\mu y_j)\Big)_{(x_i,y_j)\in\Omega_h}. \end{aligned}$$

Hint: Trigonometric identities.

10. Please hand in your solution to this exercise until Monday, January 11, 2016, 8:00 am. Make sure the following requirements are met:

(a) The program must be compilable with a Makefile. Your program must compile successfully when calling "make".

(b) The program must be callable as specified above.

(c) The program must compile without errors or warnings on `LSS cluster` nodes with the following mpic++ compiler flags (you can make use of C++11 features):

```
-std=c++11 -Wall -Wextra -Wshadow -Werror -O3 -DNDEBUG
```

If you want to, you can add additional compiler flags.

(d) Please include a specialized `pbs` script.

(e) The solution should contain well commented source files and instructions how to use your program. When submitting the solution, remove all temporary files from your project directory with the name `ex03_groupXX/` and pack it using the following command:

```
tar -cjf ex03_groupXX.tar.bz2 ex03_groupXX/
```

where `XX` stands for your group number and `ex03_groupXX/` contains your solution. Then upload your solution in StudOn.

### Performance challenge

Every code is compiled with mpic++ and the compiler flags specified above. The program is run several times on the `LSS cluster` using 4 nodes. We will use the parameters $n_x = n_y = 10000, c = 100$ and $eps = -1$. The winning team will be awarded an extra credit point.

### Beat Your Tutors!

The ultimate performance challenge: If you are the winning team of the performance challenge and on top of that can beat the time of "/software/sles/siwir/cg", you will earn an additional reward.

### Credits

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

1. You receive at least 1 point 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 LSS cluster acts as reference environment.

2. Two points are awarded if you additionally answer the theoretical questions in Task 2 correctly.

## References

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

[2] `https://www10.cs.fau.de/en/research/hpc-cluster/`