$(C++)_{10}H_{15}N$

PrOgramming Bad

Parallized Jacobi Iteration

Jörg Cosfeld

Jan Hansen

Thphy I
Numerical Simulations II

Table of contents

Jacobi Iteration

Definition

Jacobi Iteration is a method to solve a given sytem of linear equations.

- Guess a solution for the system.
- Solve for diagonal elements.
- Update solution guess.
- Iterate until it converges.

$$a_{11} \cdot x_1 + ... + a_{1n} \cdot x_n = b_1$$
$$a_{21} \cdot x_1 + ... + a_{2n} \cdot x_n = b_2$$
$$...$$
$$a_{n1} \cdot x_1 + ... + a_{nn} \cdot x_n = b_n$$

$$x_i^{(m+1)} = \frac{1}{a_{ii}}(b_i - \sum_{i \neq j} a_{ij} \cdot x_j^{(m)})$$

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Convergence

Method is guaranteed to converge for a diagonally dominant Matrix.

Definition

diagonally dominant $\equiv |a_{ii}| > \sum_{j \neq i} |a_{ji}|$

- Jacobi Iteration Method holds a linear rate of convergence.

$$||x_{k+1} - x|| \leq c||x_k - x||^p \quad ; \quad \text{with} \quad k = 0, 1, ...$$

- p represents the order
- Faster convergence usually leads to instability
- Jacobi Iteration Method has an order of $p = 1$

Usage of MPI

Definition

The components of the next approximation are independent.

$\rightarrow$ Use MPI to spread calculation of components among multiple tasks

```
    //Choose an initial guess x0 to the solution
    k = 0
    while (convergence not reached)
        //Split i-loop among tasks
        for (i := 1+rank*(n/tasks) repeat n/tasks times){
            h = 0
            for (j := 1 step until n){
                if (j != i)
                    h = h + a(i,j) x[i]
            }//end of (j-loop)
            x[i]_k+1 = (b[i]-h)/a(i,i)
        }//end of (i-loop)
        //check if convergence is reached
        k = k + 1
}
```

- when dividing $(n/tasks)$ the remainder needs to be treated separately

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

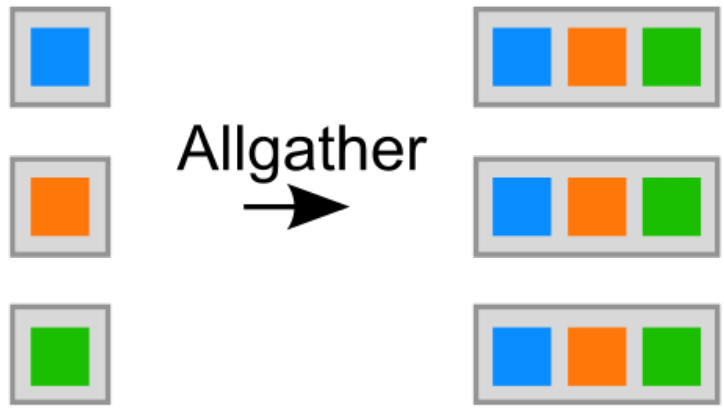
Remaining Rows

- when dividing integers $(n/tasks)$ solution is floored
- this results rows which still need to be calculated
- the remaining rows equal $(n\%tasks)$ and have to be treated separately
- in order to use "allgatherv" comfortably the task with the highest id will do the remaining calculations

```
JacobiStep(id*(n/tasks),(n/tasks));
if(rank==(tasks-1) && (n%tasks) != 0)
JacobiStep((id+1)*(n/tasks),n%tasks);
```

MPI Commands

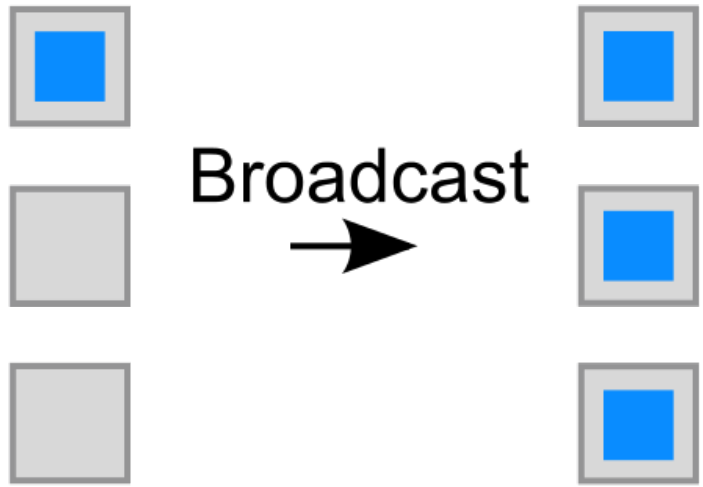
- Distribute solution by "Allgatherv" to all tasks
- Do this for every Jacobi step
- "Allgatherv" more flexible than "Allgather" since the tasks can send and receive different sized arrays




```
int MPI_Allgatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
                   void *recvbuf, int *recvcounts, int *displs,
                   MPI_Datatype recvtype, MPI_Comm comm)
```

Distribution of matrices among tasks

- All tasks need to have the same variable
- Bcast distributes content of a variable from one task to every task




```
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
               MPI_Comm comm )
```

It seems that there is a maximum buffer-size for Bcast, since tasks do not respond when broadcasting large arrays to many tasks.

Hostfile

Schedule processes across nodes by asking two questions:

- How many processes should be launched?
- Where should those processes be launched?

Save commands in hostfile

- Control default number of cores used on machine by "slots"
- Control maximum number of cores used on machine by "max-slots"

```
pc01 slots=4 max-slots=4
pc02 slots=4 max-slots=4
pc03 slots=4 max-slots=4
.
.
.
pc011 slots=4 max-slots=4
```

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Testing performance

Test performance by using each core out of the PC-Network. Write shell-script to schedule programm calls and avoid wicked console mashup.

- Direct operation of system by command line interpreter
- Grab console commands in ".sh" file
- Start ".sh" file to execute commands

```
#start jacobi_mpi_07_07.cxx
#-----------------------------
#shell script
#-----------------------------

for c in ${seq 32}
do
        echo ${c}
        time mpirun --hostfile my_hostfile -np ${c} ./jacobi 5kM Bx
done
```

Create random diagonally dominant matrices

- Create random matrix and add $\sum_{i=j}$ of row to diagonal element a_{ii}
- Solve equations system with C++ program
- Compare Matlab and C++ solution-vector to check accurate behaviour

```
clc
clear all

N = 5000;
a = 20;

M=a*randn(N,N);
M=M+diag(sum(abs(M),2));

x=a*randn(N,1);

b=M*x;

A = [M b];

dlmwrite('At', A,'delimiter', '\t','newline','unix');
dlmwrite('xt',x,'delimiter', '\t','newline','unix');
```

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Solution-vectors

Comparison of solution-vectors:

Matlab-solution (exact)

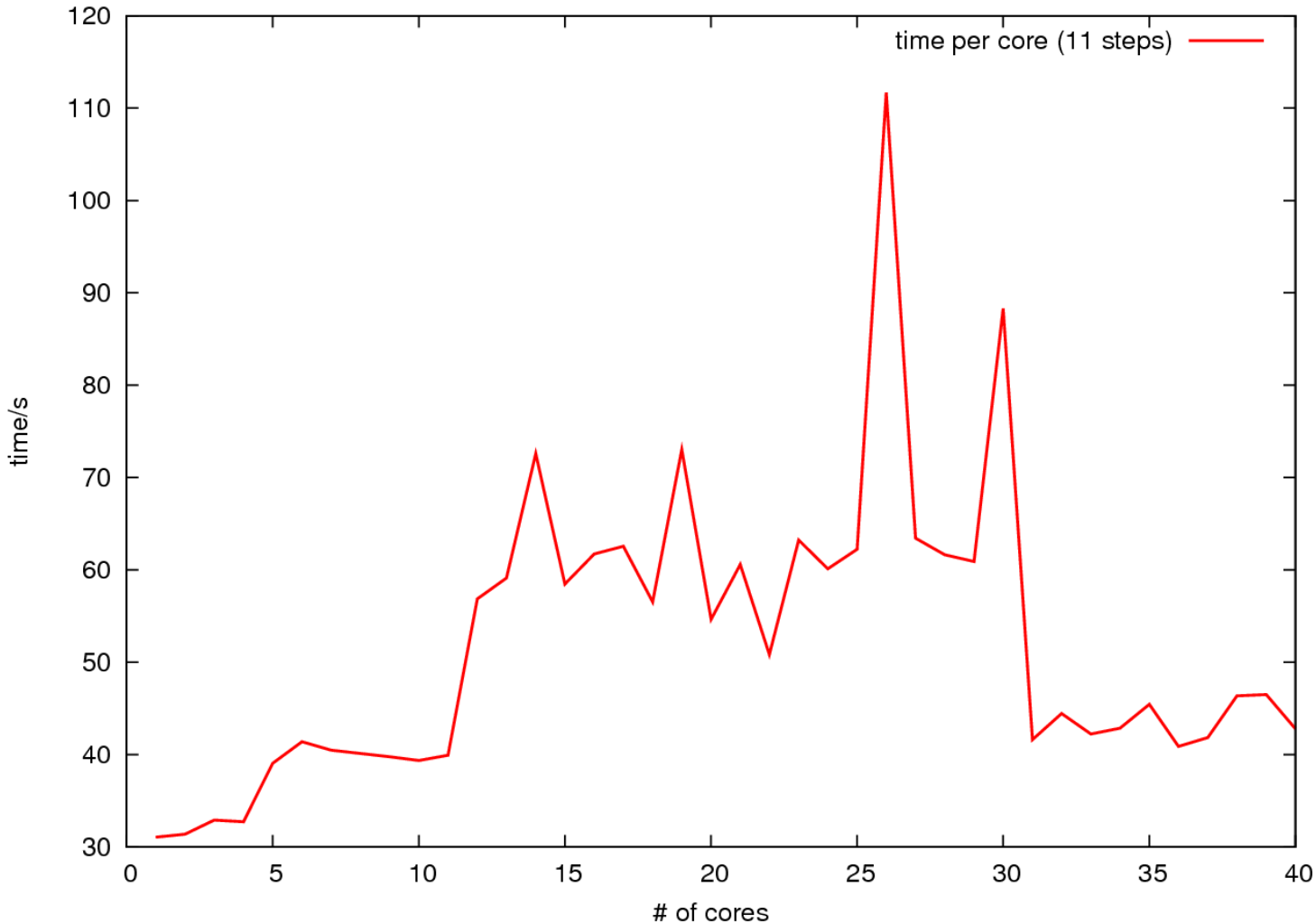
```
18
−15
−22
−8
15
10
−1
−17
24
−4
5
13
−17
−2
2
−6
4
−21
−37
20
```

C++-solution

```
18.0002
−14.9993
−21.9998
−8.00005
15.0002
9.9998
−0.999998
−17.0002
23.9994
−4.00005
5.00004
13.0004
−16.9997
−2
2.00004
−6
3.99992
−20.9996
−36.9986
20.0006
```

Is there any speedup ?

Checking for any speedup by using upper shellscript. Write CPU-time and number of cores to output-file.



At this point, moral disaster. Usage of MPI and no speedup.

Runtime measurement

- Previously runtime of whole program was measured
- Now measure runtime of parallelized part
  $\rightarrow$ runtime of whole program significantly higher than runtime of parallelized part

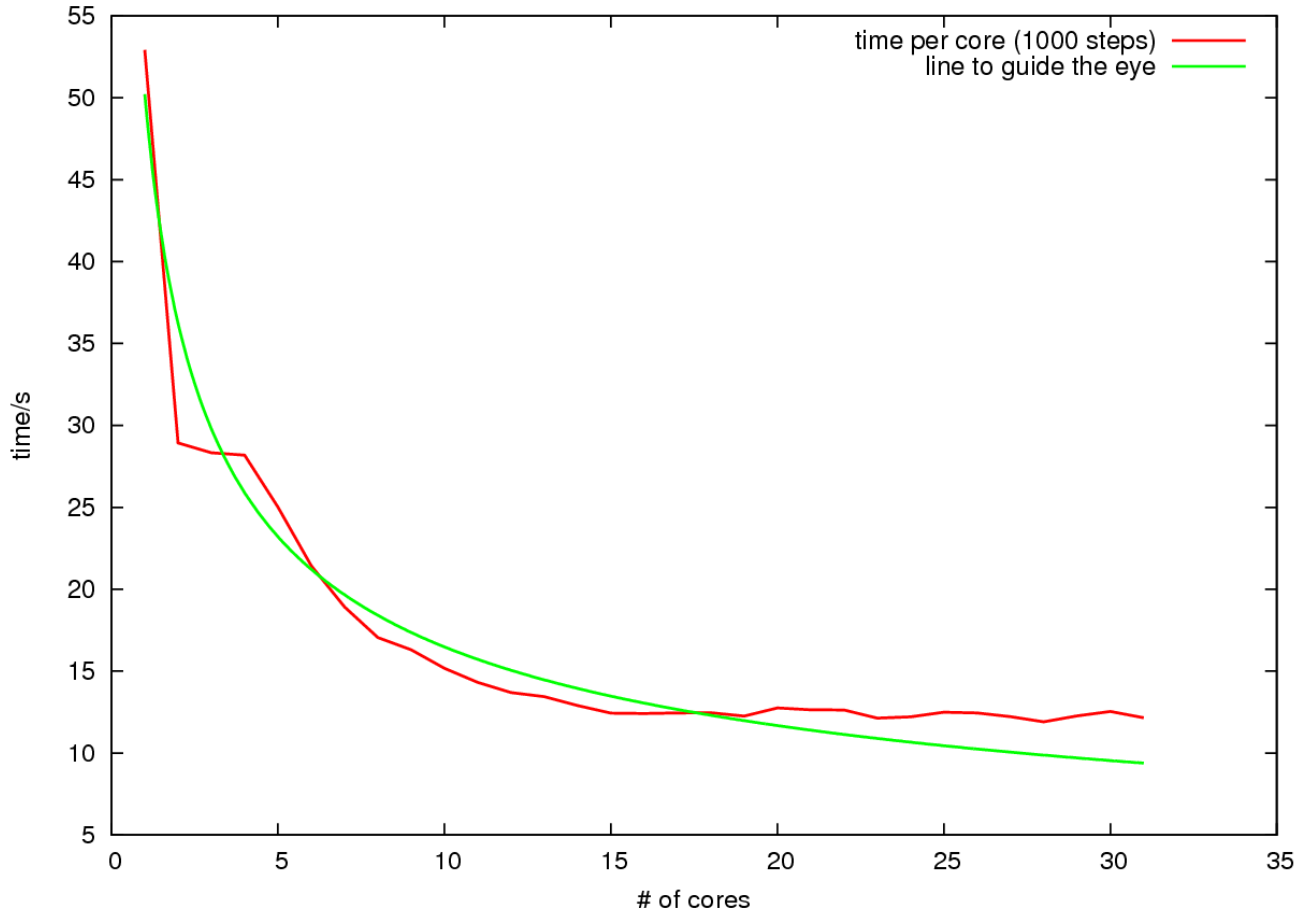
```
#include <time.h>

clock_t t;
t = clock();

if(id==0){
  initialize...
}

while(notdone){
  JacobiSteps...
}
t = (clock()-t) -t;
```

```
#include <time.h>

clock_t t;

if(id==0){
  initialize...
}

t = clock();

while(notdone){
  JacobiSteps...
}
t = (clock()-t) -t;
```

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Yes there is a speedup

Artificially increased number of iterations, since systems are solved after few iterations (15 steps).
Start program with forced number of interaion (1000 steps) on multiple cores.



Does the code work for sparse matrices?

Since the previous plot shows speedup for random dence matrices
$\rightarrow$ try sparse matrix

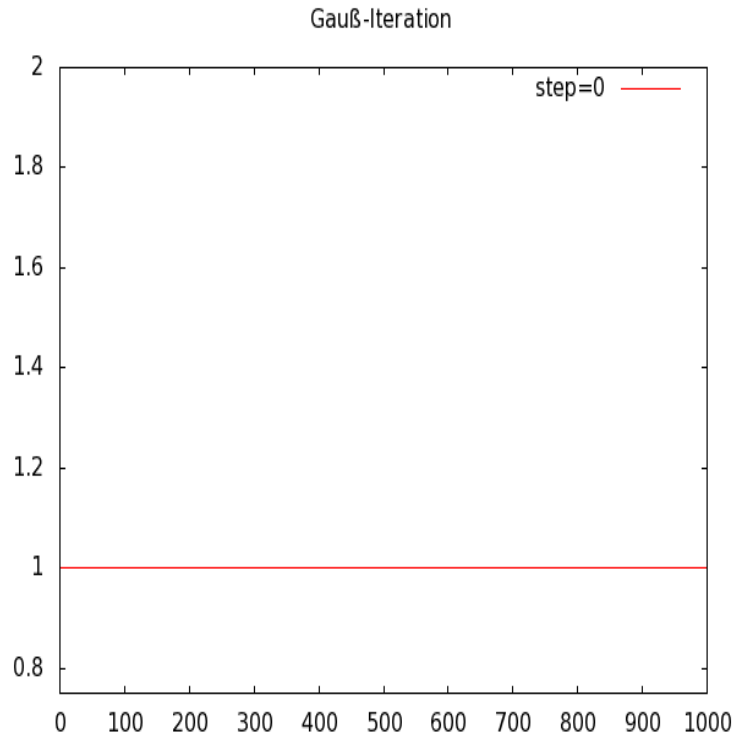
$$\underline{A} = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}$$

$$\vec{b} = (x^2 - x)\Phi(x)$$

Test with 2nd derivative matrix $\underline{A}$ on Vektor $\vec{b}$, the 2nd derivative of a gaussian distribution. This should yield a gaussian distribution.

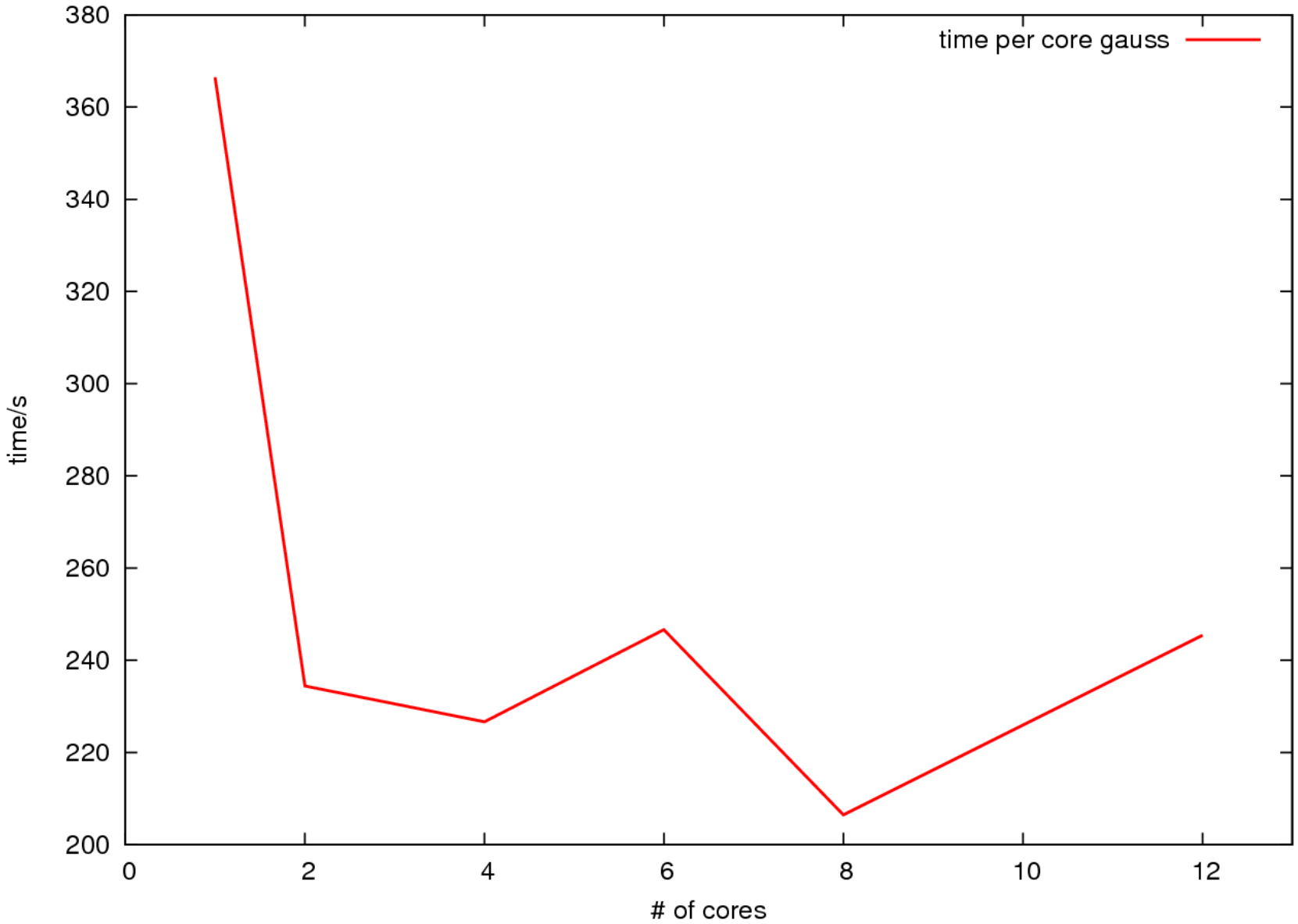
Evolution of Solution

- Solution-vector gets closer to correct solution with every iteration step
- Produces the expected gaussian distribution
- Needs a lot more Iterations compared to a dense matrix



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

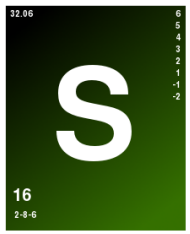
Runtime for sparse matrix



Thank you for your attention!
Any Questions ?

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Have a nice meal @



cotti's

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF