,

,

Poisson equation

Laue, Cedrik

TUM

June 24, 2021

General Problem

- Determine potential distribution in 2D system
- System consists of multiple objects with predefined potentials
- Charge distribution may be applied

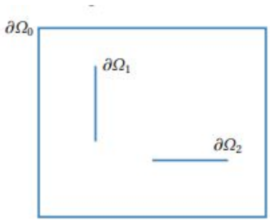

Figure 1: Exemplary System

Poisson equation

- Solve possion equation to determine potential distribution
- Possion equation can be derived from the maxwells equations

Possion equation

$$\Delta u(\boldsymbol{r}) = -\sigma(\boldsymbol{r}) \tag{1}$$

Finite Difference method 1

- Discretize System
- Approximate Laplacian using finite differences

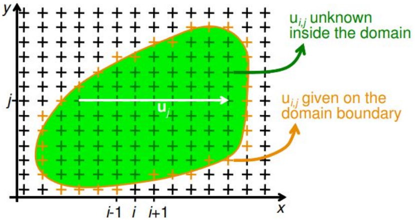

Figure 2: Discretized System

Finite Difference method 2

- Approximate derivatives with finite differences

Finite Difference for one component

$$\partial_x^2 u = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta_x^2} \tag{2}$$

- Combine both derivatives

Finite Difference 2D

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta_y^2} = -\sigma_{i,j} \tag{3}$$

- Write down equation system for all grid points

$$
\underbrace{\begin{bmatrix}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
& -1 & 4 & -1 & & & -1 & & & & & & & & & \\
& & -1 & 4 & & & & -1 & & & & & & & & \\
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
& -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
& & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
& & & -1 & & & -1 & 4 & & & & -1 & & & & \\
& & & & -1 & & & & 4 & -1 & & & -1 & & & \\
& & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
& & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
& & & & & & & -1 & & & -1 & 4 & & & & -1 \\
& & & & & & & & -1 & & & & 4 & -1 & & \\
& & & & & & & & & -1 & & & -1 & 4 & -1 & \\
& & & & & & & & & & -1 & & & -1 & 4 & -1 \\
& & & & & & & & & & & -1 & & & -1 & 4
\end{bmatrix}}_{L}
*
\underbrace{\begin{bmatrix}
U(1,1) \\ U(2,1) \\ U(3,1) \\ U(4,1) \\ U(1,2) \\ U(2,2) \\ U(3,2) \\ U(4,2) \\ U(1,3) \\ U(2,3) \\ U(3,3) \\ U(4,3) \\ U(1,4) \\ U(2,4) \\ U(3,4) \\ U(4,4)
\end{bmatrix}}_{U}
=
\underbrace{\begin{bmatrix}
b(1,1) \\ b(2,1) \\ b(3,1) \\ b(4,1) \\ b(1,2) \\ b(2,2) \\ b(3,2) \\ b(4,2) \\ b(1,3) \\ b(2,3) \\ b(3,3) \\ b(4,3) \\ b(1,4) \\ b(2,4) \\ b(3,4) \\ b(4,4)
\end{bmatrix}}_{-\sigma}
$$

Figure 3: Laplacian for a 4x4 Grid

Finite Difference method 4

- Sort columns and rows to accomodate for boundary conditions

Split equation system

$$\begin{pmatrix} L_{post} & L_{b1} \\ L_{b2} & L_{b3} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{u}_{in} \\ \boldsymbol{u}_b \end{pmatrix} = -\begin{pmatrix} \boldsymbol{\sigma}_{in} \\ \boldsymbol{\sigma}_b \end{pmatrix} \tag{4}$$

Remove boundary components and multiply partially

$$L_{post} \cdot \boldsymbol{u}_{in} = -\boldsymbol{\sigma}_{in} - L_{b1} \cdot \boldsymbol{u}_b \tag{5}$$

Jacobi method

- special splitting method like Gauß-Seidel- or SOR-method
- easily parallizable in comparison to other splitting methods

Complete step

$$x^{(m+1)} = D^{-1}\left(b - (L+U)x^{(m)}\right), \quad A = L + D + U \tag{6}$$

Elementwise Jacobi-method

$$x_i^{(m+1)} := \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}\cdot x_j^{(m)}\right), i = 1, \dots, n \tag{7}$$

LU Decompostion

- Decomposit matrix into upper and lower triangular matrix

LU Decompostion with partial pivot

$$PA = LU \tag{8}$$

LU Decompostion with partial pivot

$$P \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \tag{9}$$

Necessary installations under Ubuntu 20.04

Install g++

sudo apt install g++

Install Cuda

sudo apt install nvidia-cuda-toolkit

Install cmake

sudo apt install cmake

Install Eigen library

sudo apt install libeigen3-dev

Install mathgl library

sudo apt-get install mathgl

Run code under Ubuntu 20.04

create build folder

```
mkdir build
```

Change directory to build

```
cd build
```

create Makefile

```
cmake ..
```

Compile application

```
make
```

Run code

```
./fdm
```

Results

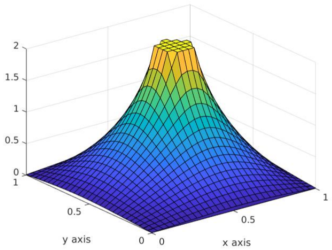

Figure 4: Solution of Laplace equation with full circle boundary with low resolution.

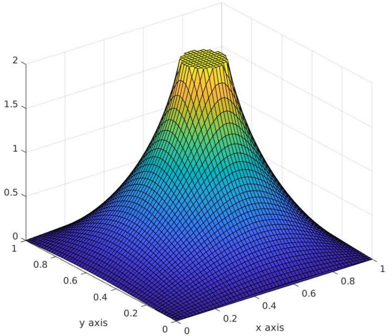

Figure 5: Solution of Laplace equation with full circle boundary with high resolution.

Results

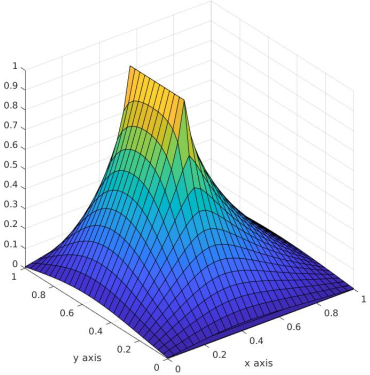

Figure 6: Solution of Laplace equation with line boundary with low resolution.

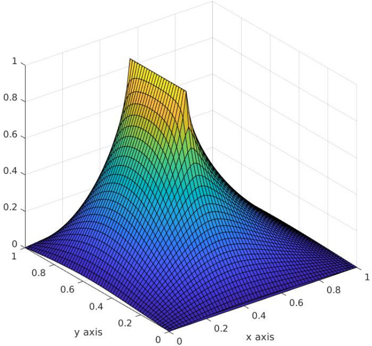

Figure 7: Solution of Laplace equation with line boundary with high resolution.

Results

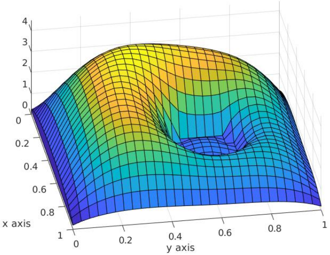

Figure 8: Solution of Poission equation with full circle boundary with low resolution.

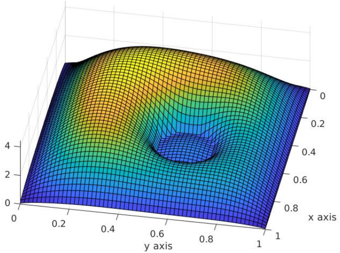

Figure 9: Solution of Poission equation with circle boundary with high resolution.

Benchmarking 1

- Cuda1: GTX 1050. Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
- Cuda2: GTX 960. Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
- Cuda4: GTX 1650. AMD Ryzen 7 2700 Eight-Core Processor

Benchmarking 2 - TPB CUDA4

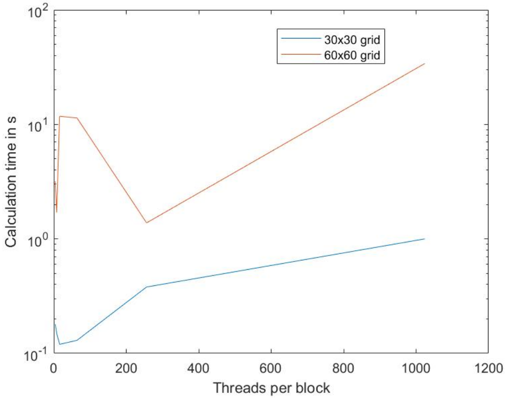

Figure 10: Calculation time depending on Threads per block and grid size on Cuda4.

Benchmarking 3 - CPU vs. GPU

Device	Cuda1	Cuda2	Cuda3
CPU	18.8	19.1	18.4
GPU	0.2	0.2	0.13

Table 1: Calculation times for 30x30 grid and 16 TPB depending on used device.

Sources of figures

- Figure 1: Josef Knapp. Computational and Analytical Methods in Electromagnetics
- Figure 2: Prof. Dr.-Ing. Christian Jirauschek.Partial Differential Equations in Electrical Engineering - Lecture Notes
- Figure 3: https://people.eecs.berkeley.edu/ demmel/cs267/lecture24/lecture24.html