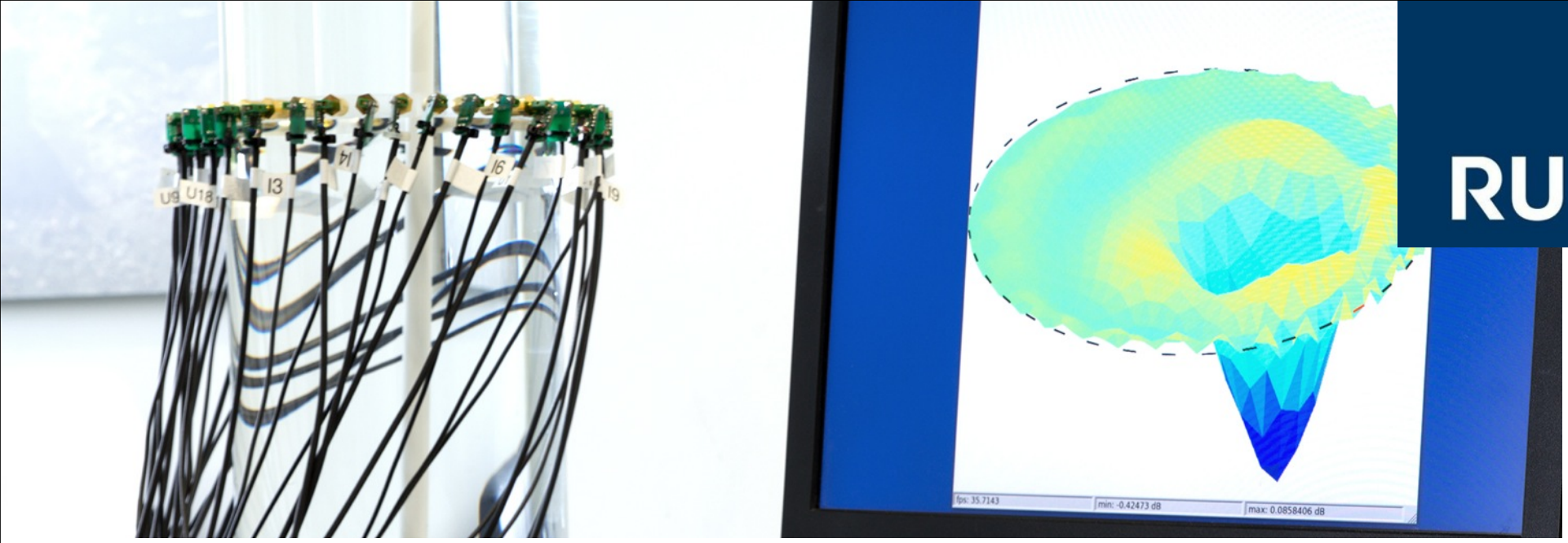


RUHR-UNIVERSITÄT BOCHUM

Hardware and Software Design for a 1000 FPS Real-Time Soft-Field Tomography System

M. Sc. Patrik Gebhardt

GPU Technology Conference, San Jose, 03/26/2014

LEHRSTUHL FÜR
Elektronische Schaltungstechnik

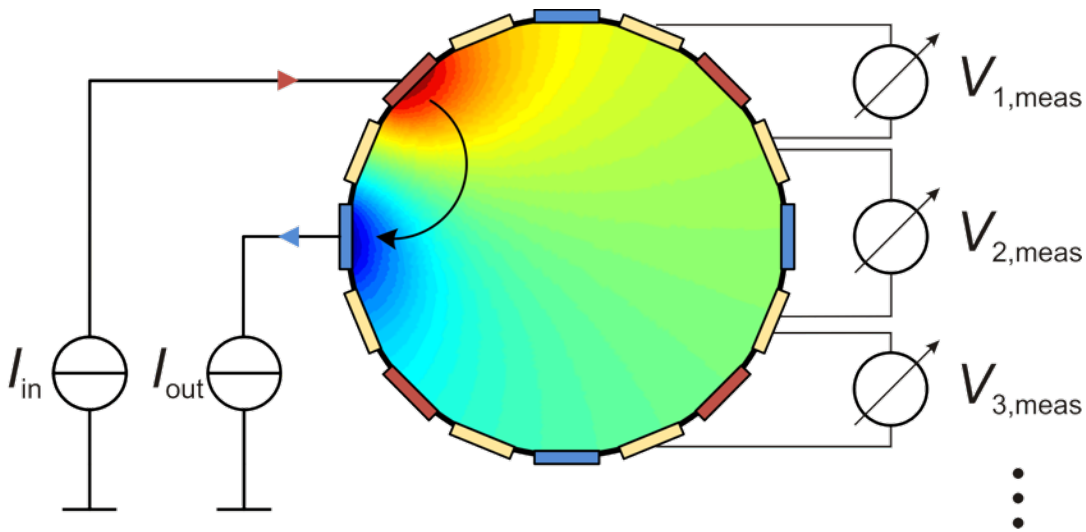
Content

What is Soft-field Tomography?

Basics

- Method for generating cross-sectional images
- Use of non ionizing radiation, unlike hard-field tomography

- Example: Electrical Impedance Tomography (EIT)
  - Attach surface with electrodes
  - Inject electrical currents
  - Measure corresponding voltages



What is Soft-field Tomography?

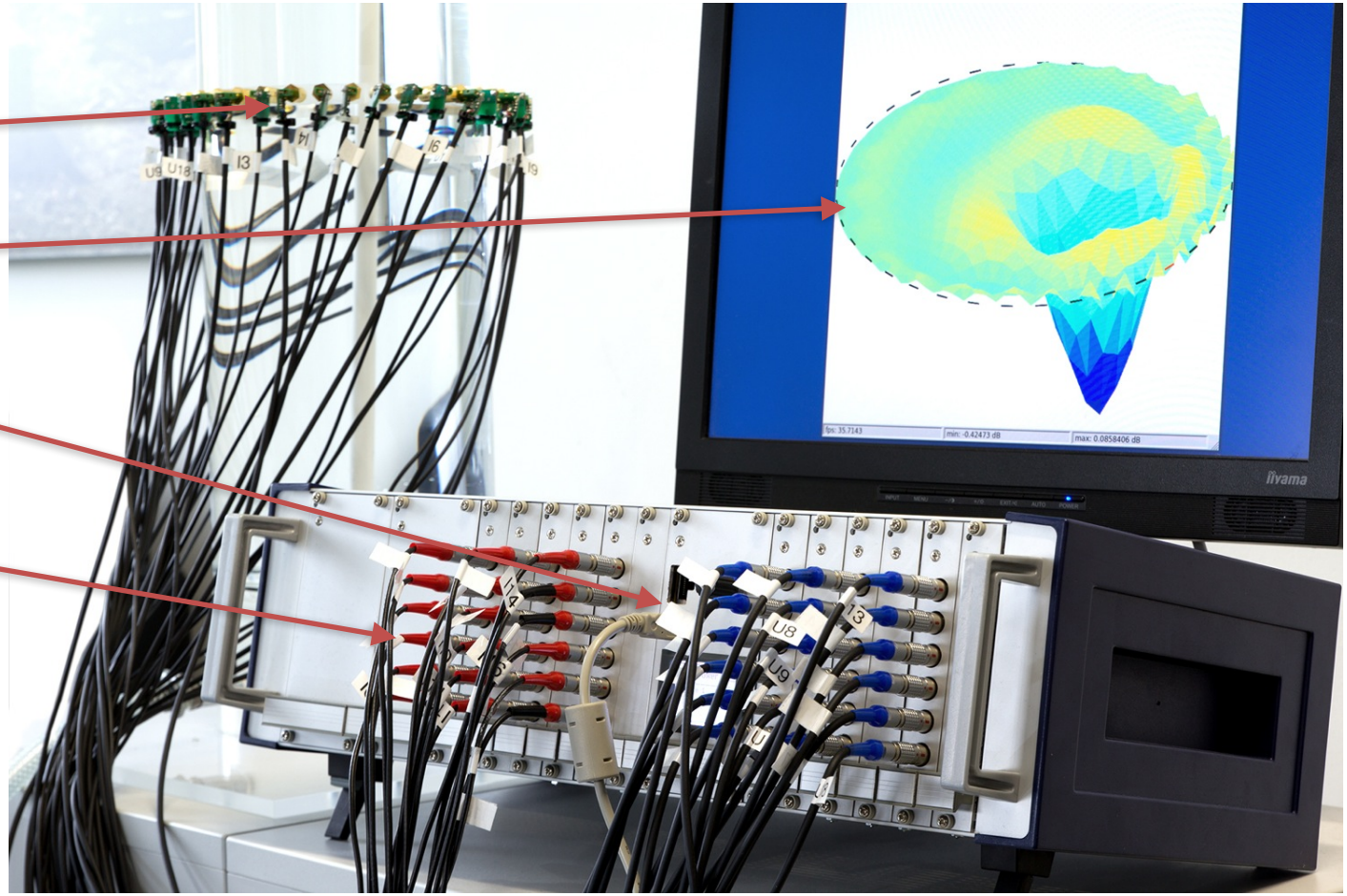
EIT-System

Electrodes

Reconstruction

Voltage Measurement

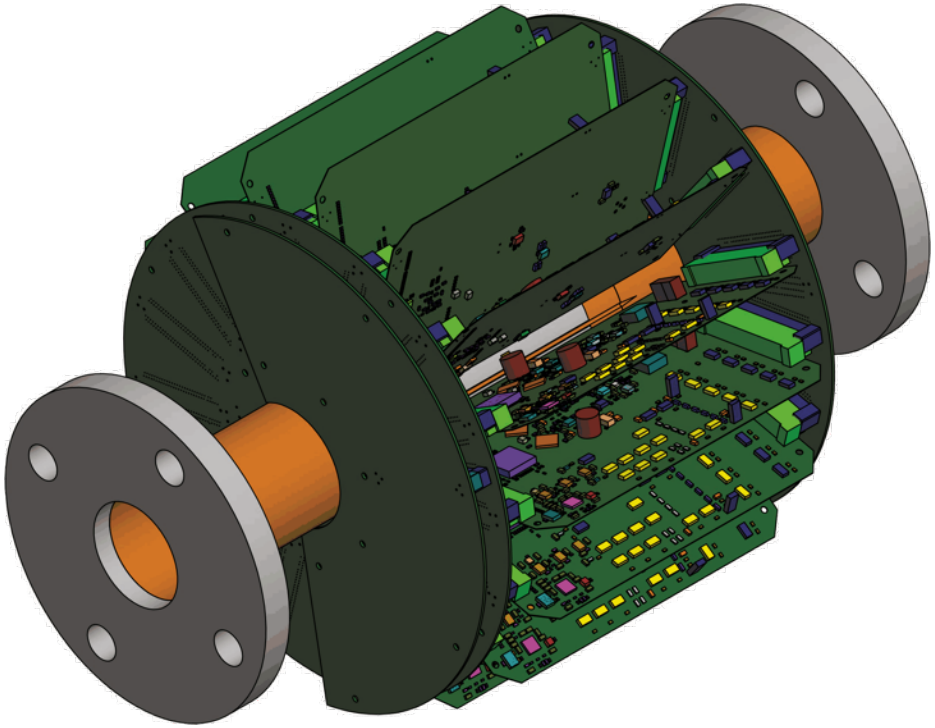
Current Sources



What is Soft-field Tomography?

Use Cases

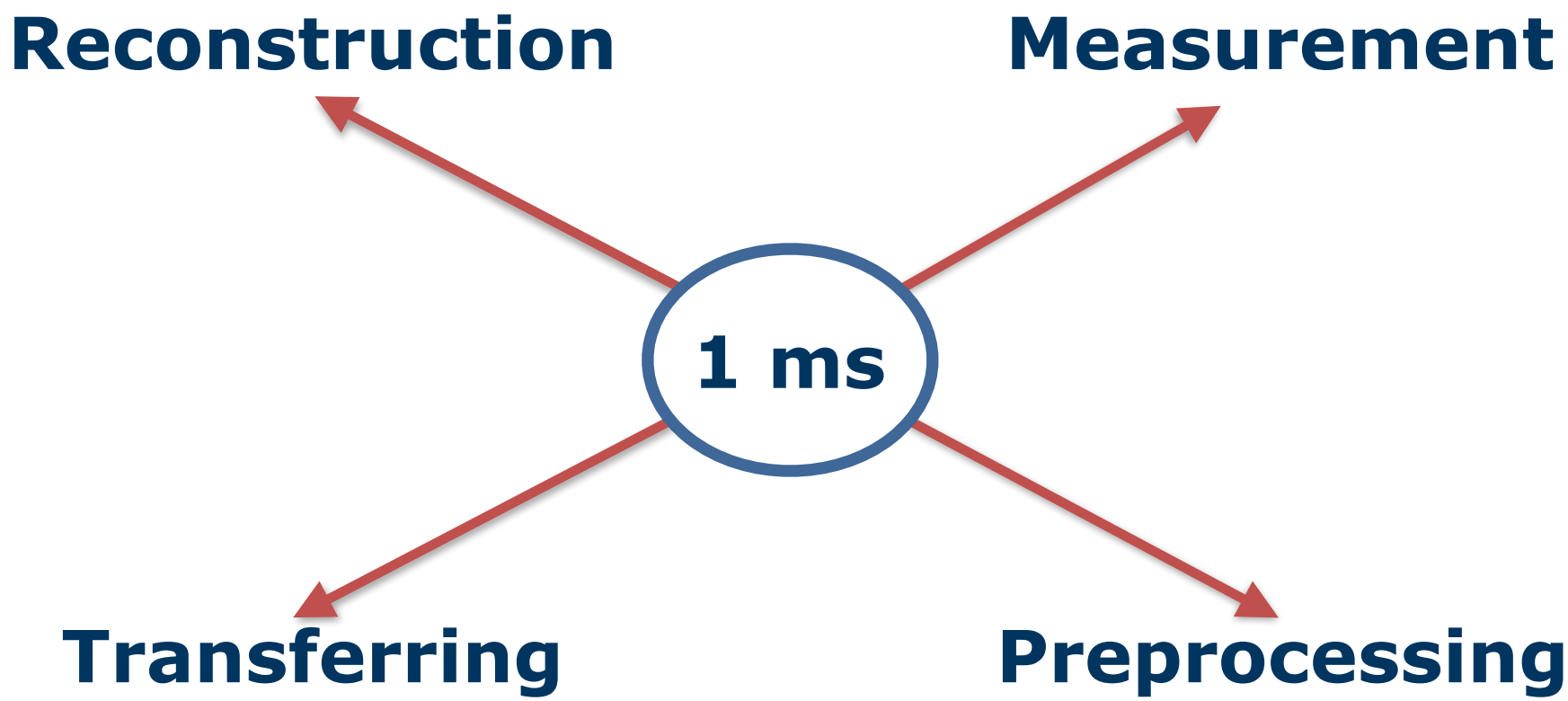
http://www.draeger.com/



Content

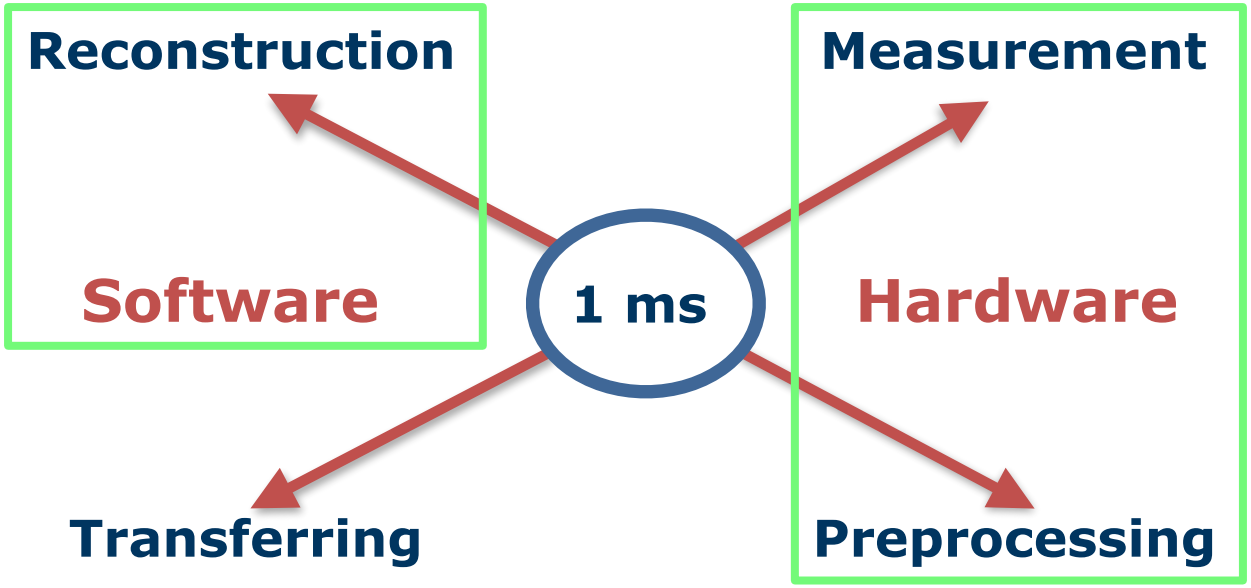
What it needs to be that fast?

Situation



What it needs to be that fast?

Consequences

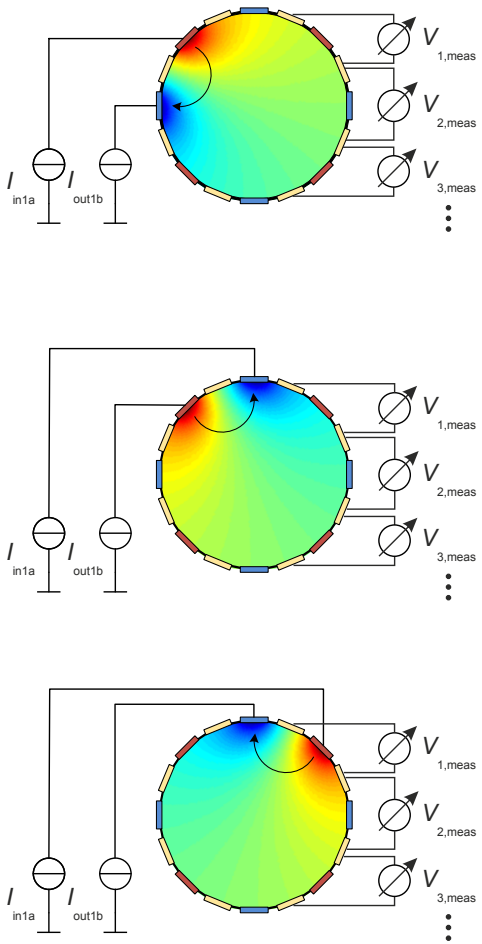


- Parallel hardware design
- Hardware based preprocessing
- Pipelining
- Reconstruction on GPUs

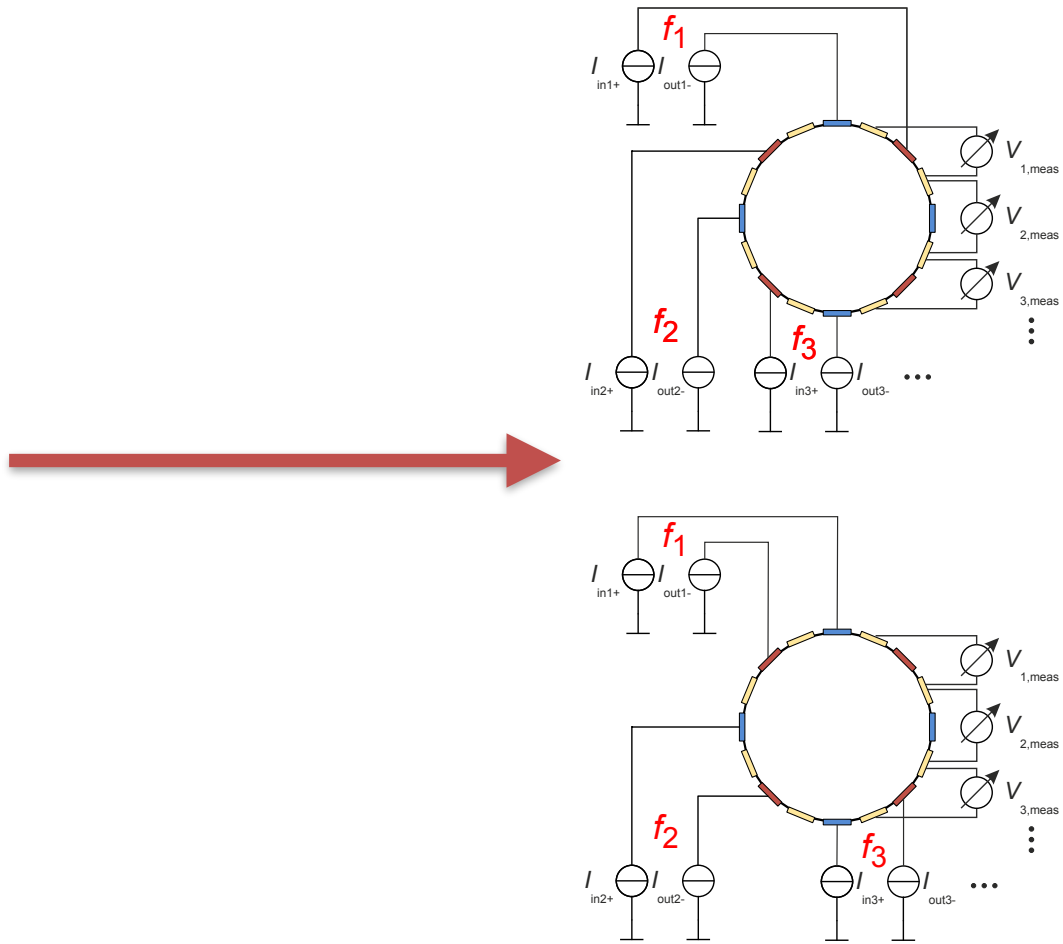
Content

Hardware Design and Data Acquisition

Parallelizing Measurement Procedure



TDMA

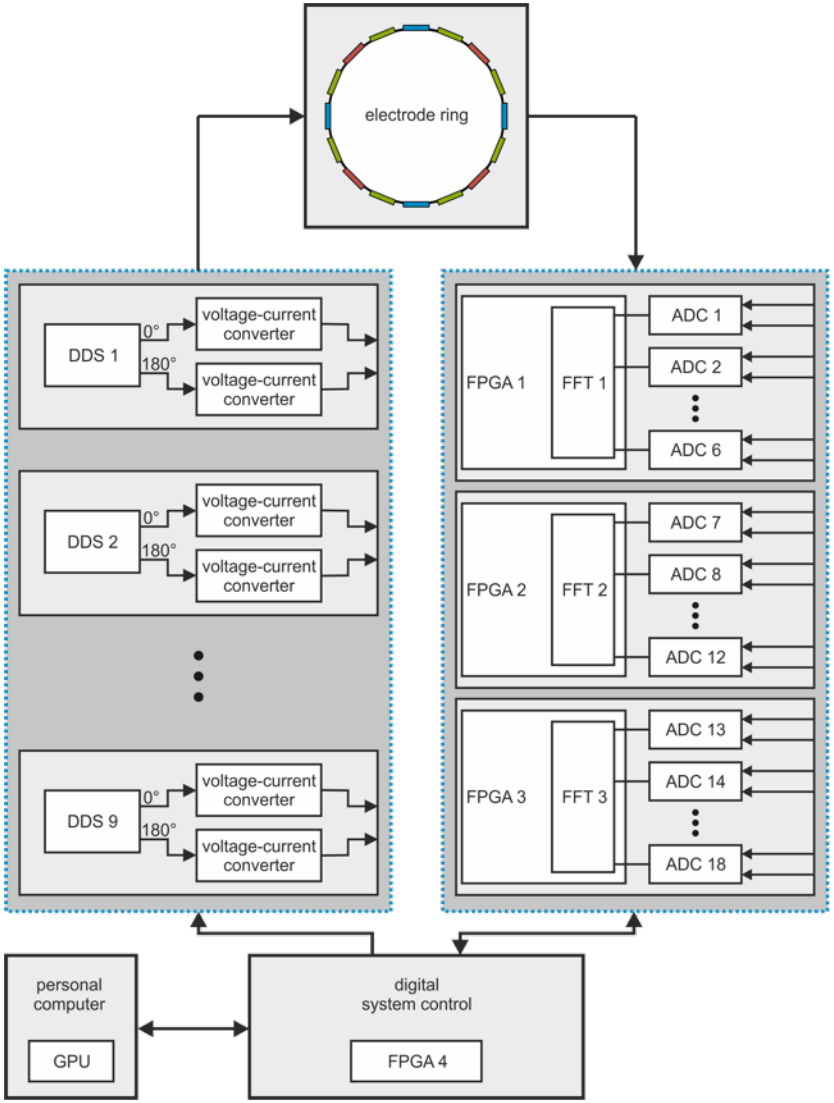


FDMA

Hardware Design and Data Acquisition

Parallel Hardware Architecture

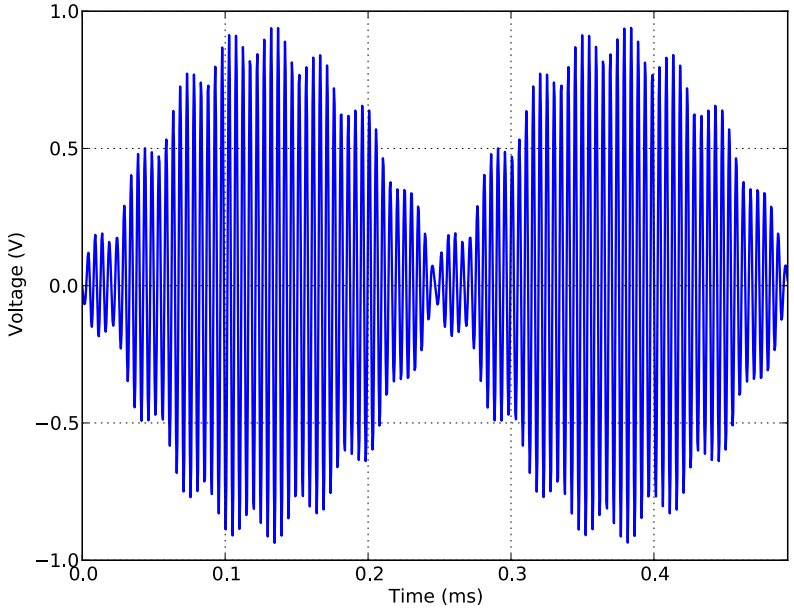
- 36 gold-plated electrodes
- 9 parallel floating current sources
  - Direct digital synthesis
  - Controlled floating current sources
- 18 parallel measurement channels
  - 18 analog-to-digital converters
  - 3 field programmable gate arrays for fast Fourier transform
- Digital system control unit
  - Ethernet and USB interface
  - Central clock generation and distribution

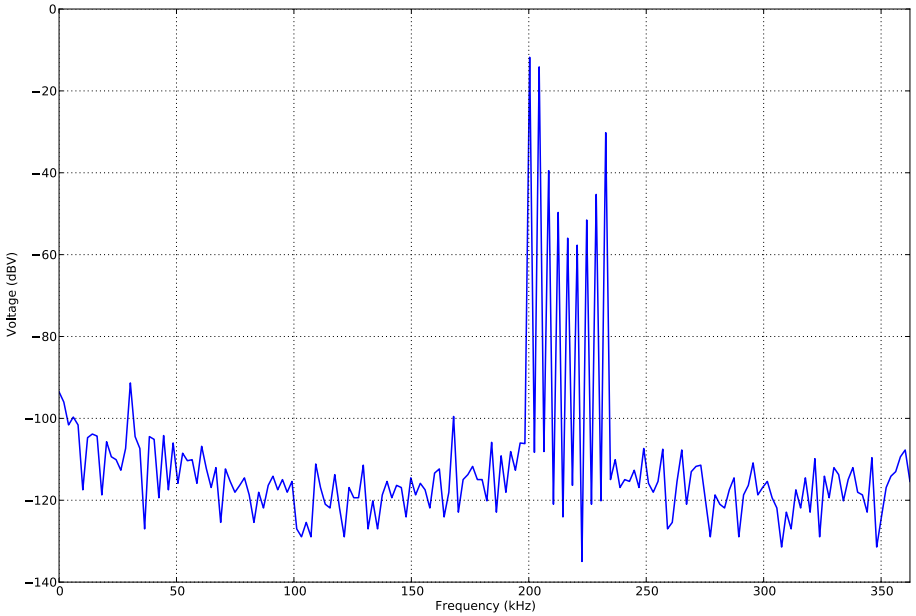


Hardware Design and Data Acquisition

Preprocessing

- FPGA based FFT calculation
- In parallel with data acquisition
- Synchronized with signal generation
  —> no leakage-effect of FFT



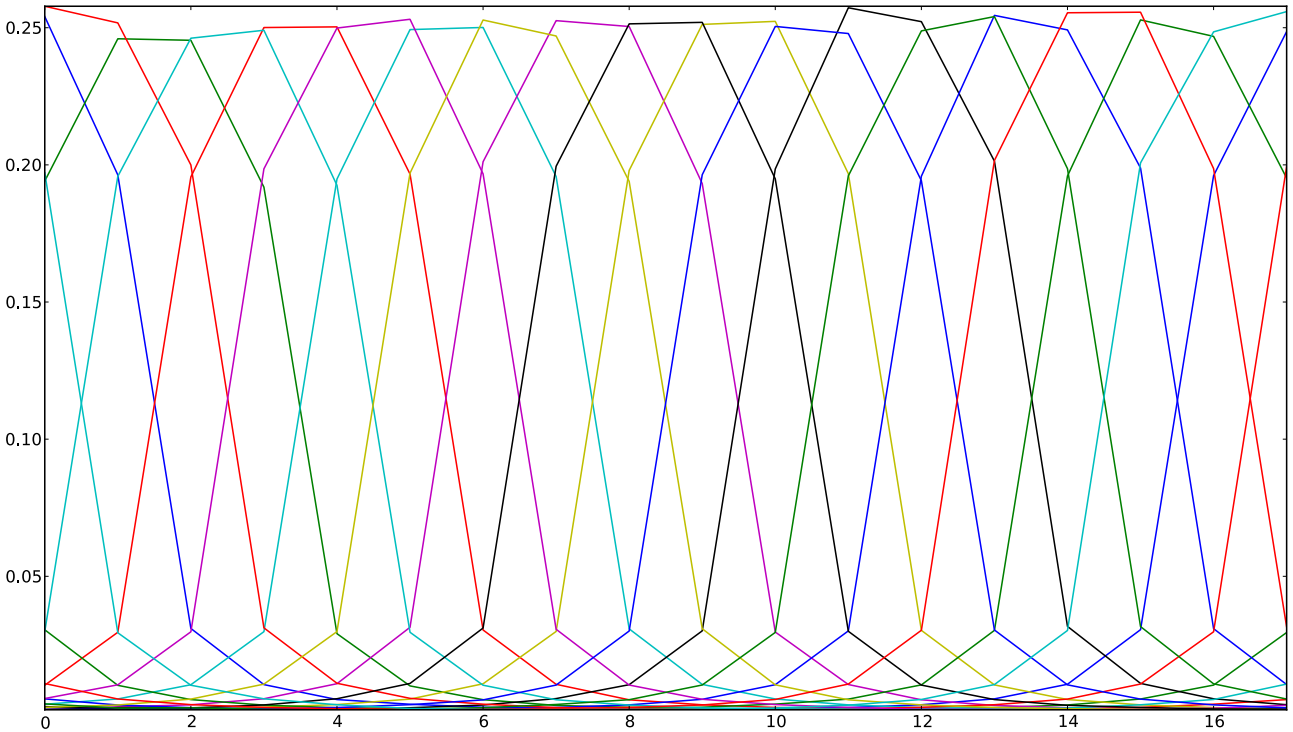


- 1024 Samples in 0.5 ms for 2 multiplexing steps

Hardware Design and Data Acquisition

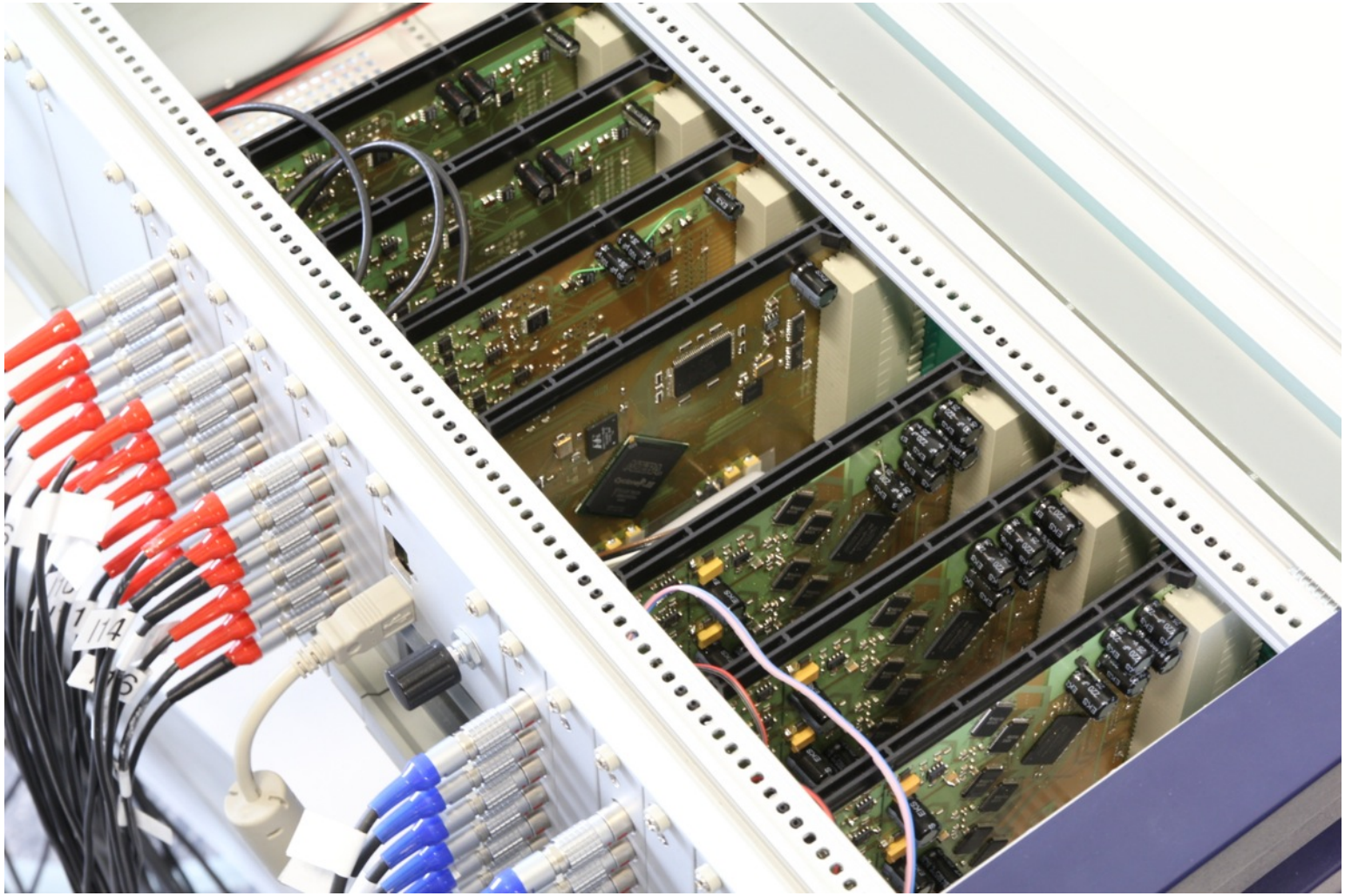
Package generation

- Main-FPGA collects all data
- Only relevant frequency points are transmitted
- Total data transfer rate: 9.89 MBit/s



Hardware Design and Data Acquisition

Complete System



Content

GPU Based Image Reconstruction

Algorithm

- Underlying PDE, derived from Maxwells Equations

$$\nabla \cdot \left(\sigma\left(\vec{r}\right) \nabla \Phi\left(\vec{r}\right)\right) = 0$$

- Only solvable with ideal boundary conditions
- Better suited is solving an optimization problem

$$\sigma\left(V_m\right) = \arg\min_{\sigma} \parallel V_m - V_c(\sigma) \parallel^2$$

- Algorithm split up into inverse and forward problem

GPU Based Image Reconstruction

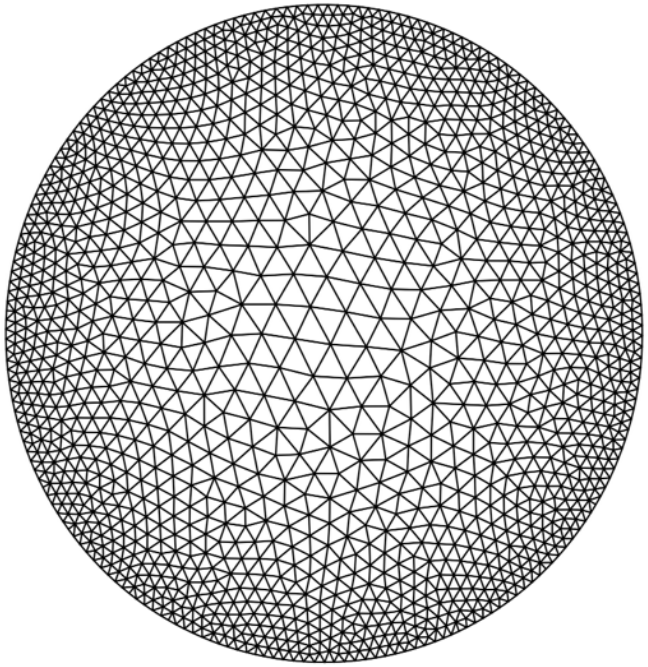
Solving Forward Problem

- PDE solved with Finite Element Method
- Domain split up into elements
  - Piecewise constant conductivity
  - Electrical potential interpolated from nodes
- Equation transformed into linear system

$$\left(S(\sigma) + \left(\frac{n\pi}{z_m}\right)^2 R(\sigma)\right)\vec{V}_n = B\vec{I}_n$$

$$S_{i,j}(\sigma) = \sum_{k=1}^{K} \sigma_k \int_{\Omega_k} \nabla u_i \cdot \nabla u_j dV$$

$$R_{i,j}(\sigma) = \sum_{k=1}^{K} \sigma_k \int_{\Omega_k} u_i u_j dV$$

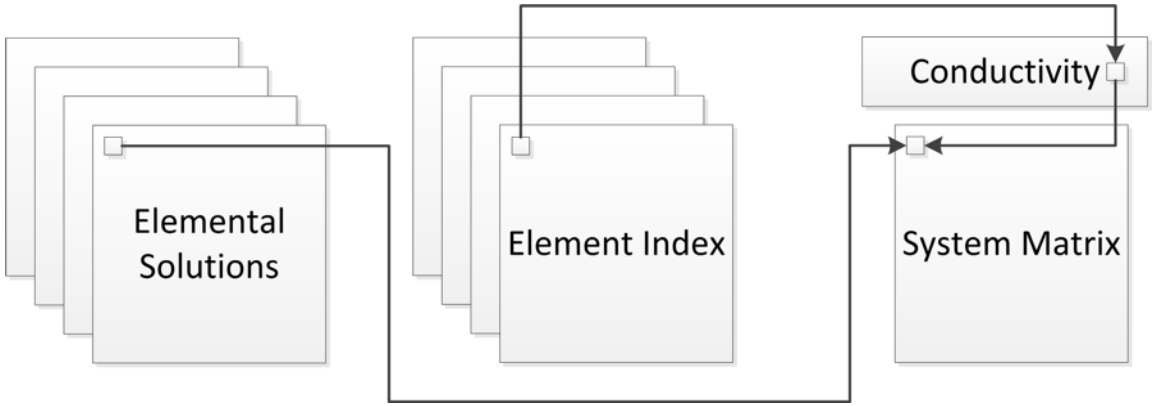


GPU Based Image Reconstruction

Matrix assembly on GPU using Metaprogramming

- FEM model matrices needs to be updated often
- Matrices needs to be as accurate as possible
- Solution: Analytic integration using Metaprogramming

- Matrix assembly algorithm on GPU



GPU Based Image Reconstruction

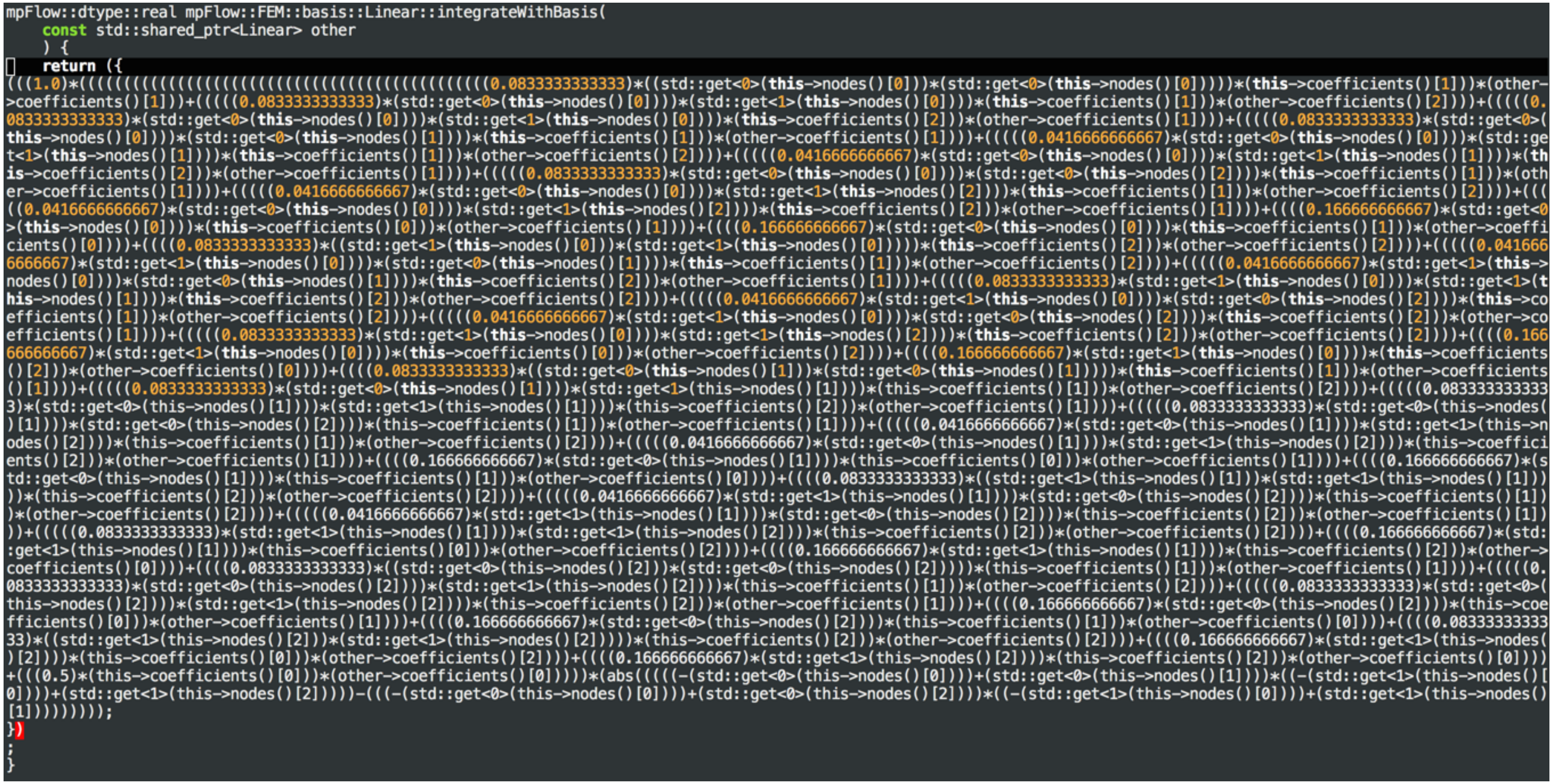
Matrix assembly on GPU using Metaprogramming

$$R_{i,j}(\sigma) = \sum_{k=1}^{K} \sigma_k \int_{\Omega_k} u_i u_j dV$$

```
@kernel
def integrateWithBasis(self, points, ci, cj):
    # create coordinats
    x, y = symbols('x, y')

    # basis function
    ui = self.basis_function([x, y], ci)
    uj = self.basis_function([x, y], cj)

    # equation
    equation = ui * uj

    # integrate on triangle
    return integrateOnTriangle(equation, x, y, points)
```

GPU Based Image Reconstruction

Matrix assembly on GPU using Metaprogramming



```
mpFlow::dtype::real mpFlow::FEM::basis::Linear::integrateWithBasis(
    const std::shared_ptr<Linear> other
    ) {
    return ({
(((1.0)*((((((((((((((((((((((((((((((((((((((((((((((0.0833333333333)*((std::get<0>(this->nodes()[0]))*(std::get<0>(this->nodes()[0]))))*(this->coefficients()[1]))*(other-
>coefficients()[1]))+(((((0.0833333333333)*(std::get<0>(this->nodes()[0])))*(std::get<1>(this->nodes()[0])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((((0.
0833333333333)*(std::get<0>(this->nodes()[0])))*(std::get<1>(this->nodes()[0])))*(this->coefficients()[2]))*(other->coefficients()[1])))+(((((0.0833333333333)*(std::get<0>(
this->nodes()[0])))*(std::get<0>(this->nodes()[1])))*(this->coefficients()[1]))*(other->coefficients()[1])))+(((((0.0416666666667)*(std::get<0>(this->nodes()[0])))*(std::ge
t<1>(this->nodes()[1])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<0>(this->nodes()[0])))*(std::get<1>(this->nodes()[1])))*(th
is->coefficients()[2]))*(other->coefficients()[1])))+(((((0.0833333333333)*(std::get<0>(this->nodes()[0])))*(std::get<0>(this->nodes()[2])))*(this->coefficients()[1]))*(oth
er->coefficients()[1])))+(((((0.0416666666667)*(std::get<0>(this->nodes()[0])))*(std::get<1>(this->nodes()[2])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((
((0.0416666666667)*(std::get<0>(this->nodes()[0])))*(std::get<1>(this->nodes()[2])))*(this->coefficients()[2]))*(other->coefficients()[1])))+((((0.166666666667)*(std::get<0
>(this->nodes()[0])))*(this->coefficients()[0]))*(other->coefficients()[1])))+((((0.166666666667)*(std::get<0>(this->nodes()[0])))*(this->coefficients()[1]))*(other->coeffi
cients()[0])))+((((0.0833333333333)*((std::get<1>(this->nodes()[0]))*(std::get<1>(this->nodes()[0]))))*(this->coefficients()[2]))*(other->coefficients()[2])))+(((((0.041666
6666667)*(std::get<1>(this->nodes()[0])))*(std::get<0>(this->nodes()[1])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<1>(this->
nodes()[0])))*(std::get<0>(this->nodes()[1])))*(this->coefficients()[2]))*(other->coefficients()[1])))+(((((0.0833333333333)*(std::get<1>(this->nodes()[0])))*(std::get<1>(t
his->nodes()[1])))*(this->coefficients()[2]))*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<1>(this->nodes()[0])))*(std::get<0>(this->nodes()[2])))*(this->co
efficients()[1]))*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<1>(this->nodes()[0])))*(std::get<0>(this->nodes()[2])))*(this->coefficients()[2]))*(other->co
efficients()[1])))+(((((0.0833333333333)*(std::get<1>(this->nodes()[0])))*(std::get<1>(this->nodes()[2])))*(this->coefficients()[2]))*(other->coefficients()[2])))+((((0.166
666666667)*(std::get<1>(this->nodes()[0])))*(this->coefficients()[0]))*(other->coefficients()[2])))+((((0.166666666667)*(std::get<1>(this->nodes()[0])))*(this->coefficients
()[2]))*(other->coefficients()[0])))+((((0.0833333333333)*((std::get<0>(this->nodes()[1]))*(std::get<0>(this->nodes()[1]))))*(this->coefficients()[1]))*(other->coefficients
()[1])))+(((((0.0833333333333)*(std::get<0>(this->nodes()[1])))*(std::get<1>(this->nodes()[1])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((((0.083333333333
3)*(std::get<0>(this->nodes()[1])))*(std::get<1>(this->nodes()[1])))*(this->coefficients()[2]))*(other->coefficients()[1])))+(((((0.0833333333333)*(std::get<0>(this->nodes(
)[1])))*(std::get<0>(this->nodes()[2])))*(this->coefficients()[1]))*(other->coefficients()[1])))+(((((0.0416666666667)*(std::get<0>(this->nodes()[1])))*(std::get<1>(this->n
odes()[2])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<0>(this->nodes()[1])))*(std::get<1>(this->nodes()[2])))*(this->coeffici
ents()[2]))*(other->coefficients()[1])))+((((0.166666666667)*(std::get<0>(this->nodes()[1])))*(this->coefficients()[0]))*(other->coefficients()[1])))+((((0.166666666667)*(s
td::get<0>(this->nodes()[1])))*(this->coefficients()[1]))*(other->coefficients()[0])))+((((0.0833333333333)*((std::get<1>(this->nodes()[1]))*(std::get<1>(this->nodes()[1]))
))*(this->coefficients()[2]))*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<1>(this->nodes()[1])))*(std::get<0>(this->nodes()[2])))*(this->coefficients()[1])
)*(other->coefficients()[2])))+(((((0.0416666666667)*(std::get<1>(this->nodes()[1])))*(std::get<0>(this->nodes()[2])))*(this->coefficients()[2]))*(other->coefficients()[1])
))+(((((0.0833333333333)*(std::get<1>(this->nodes()[1])))*(std::get<1>(this->nodes()[2])))*(this->coefficients()[2]))*(other->coefficients()[2])))+((((0.166666666667)*(std:
:get<1>(this->nodes()[1])))*(this->coefficients()[0]))*(other->coefficients()[2])))+((((0.166666666667)*(std::get<1>(this->nodes()[1])))*(this->coefficients()[2]))*(other->
coefficients()[0])))+((((0.0833333333333)*((std::get<0>(this->nodes()[2]))*(std::get<0>(this->nodes()[2]))))*(this->coefficients()[1]))*(other->coefficients()[1])))+(((((0.
0833333333333)*(std::get<0>(this->nodes()[2])))*(std::get<1>(this->nodes()[2])))*(this->coefficients()[1]))*(other->coefficients()[2])))+(((((0.0833333333333)*(std::get<0>(
this->nodes()[2])))*(std::get<1>(this->nodes()[2])))*(this->coefficients()[2]))*(other->coefficients()[1])))+((((0.166666666667)*(std::get<0>(this->nodes()[2])))*(this->coe
fficients()[0]))*(other->coefficients()[1])))+((((0.166666666667)*(std::get<0>(this->nodes()[2])))*(this->coefficients()[1]))*(other->coefficients()[0])))+((((0.08333333333
33)*((std::get<1>(this->nodes()[2]))*(std::get<1>(this->nodes()[2]))))*(this->coefficients()[2]))*(other->coefficients()[2])))+((((0.166666666667)*(std::get<1>(this->nodes(
)[2])))*(this->coefficients()[0]))*(other->coefficients()[2])))+((((0.166666666667)*(std::get<1>(this->nodes()[2])))*(this->coefficients()[2]))*(other->coefficients()[0])))
+(((0.5)*(this->coefficients()[0]))*(other->coefficients()[0]))))*(abs(((((-(std::get<0>(this->nodes()[0])))+(std::get<0>(this->nodes()[1])))*((-(std::get<1>(this->nodes()[
0])))+(std::get<1>(this->nodes()[2]))))-(((-(std::get<0>(this->nodes()[0])))+(std::get<0>(this->nodes()[2])))*((-(std::get<1>(this->nodes()[0])))+(std::get<1>(this->nodes()
[1]))))))));
})
;
}
```

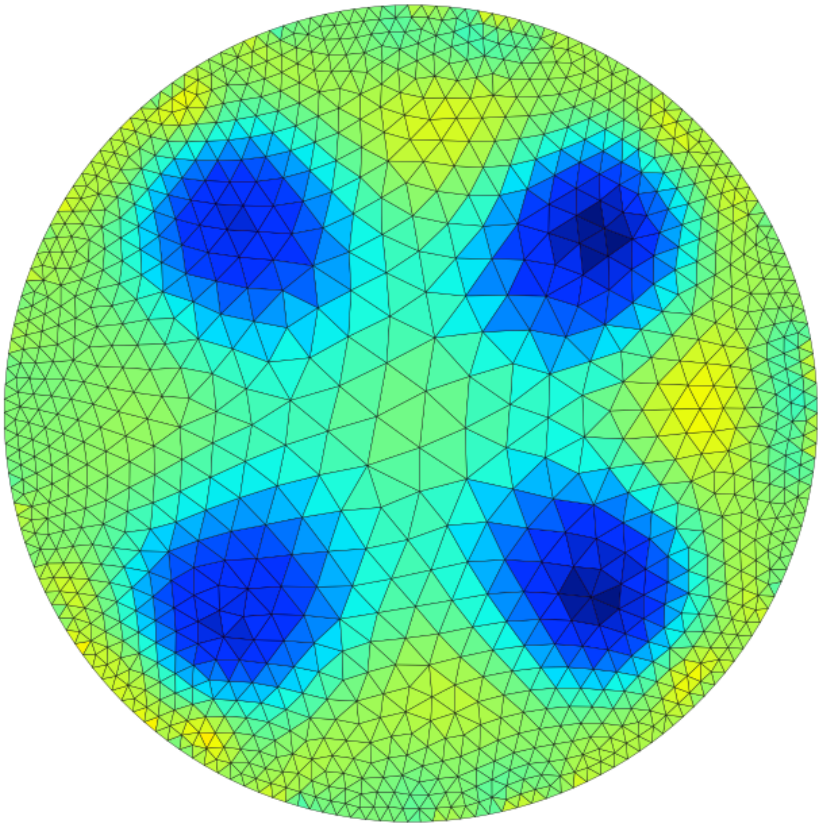
GPU Based Image Reconstruction

Solving Inverse Problem

- Newton's method used for solving inverse problem

$$\vec{\sigma}_{n+1} = \vec{\sigma}_n + \Delta\vec{\sigma}$$

$$J\left(\vec{\sigma}_n\right)\Delta\vec{\sigma} = \left[V_m - V_c\left(\vec{\sigma}_n\right)\right]$$



- Very ill-posed
  —> Regularization necessary

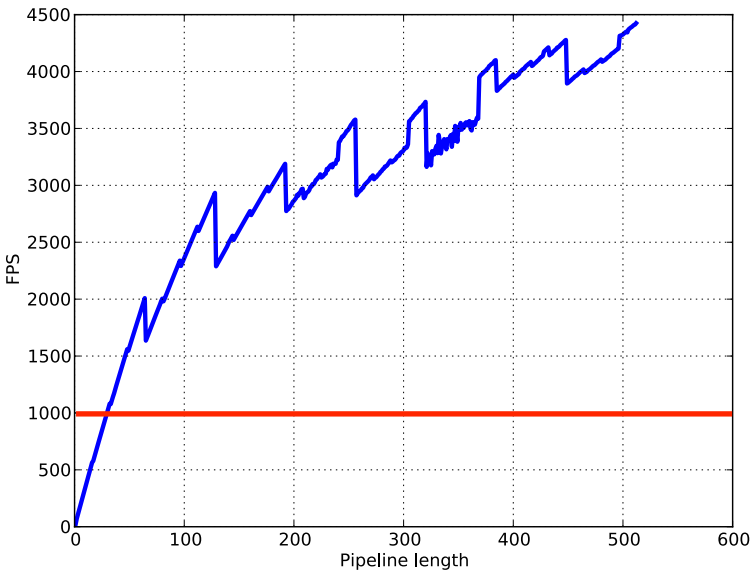
$$\Delta\vec{\sigma} = \left(J^T J + \lambda^2 L^T L\right)^{-1} J^T \Delta\vec{V}$$

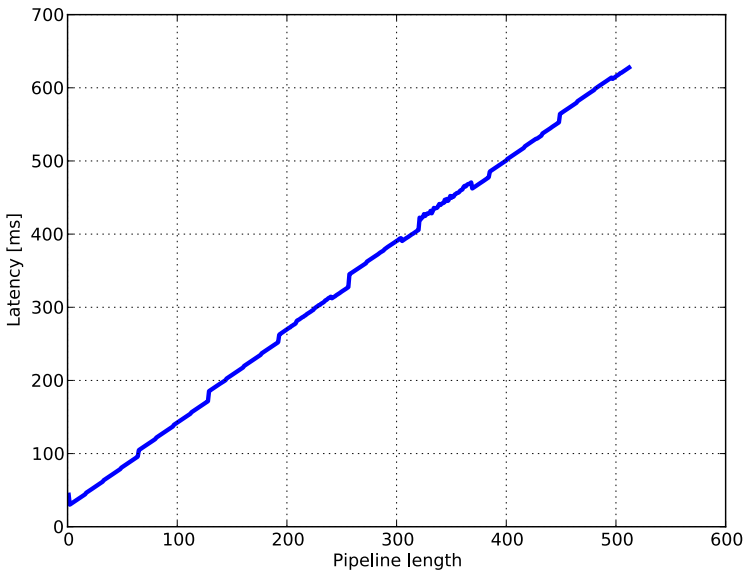
- Usually, only one step is performed
  —> Differential EIT

GPU Based Image Reconstruction

Pipelining for 1000 FPS

- Data set too small for maximum performance
- Solution: Pipelining
  —> reconstruction of multiple images simultaneously
- Tradeoff between Latency and FPS



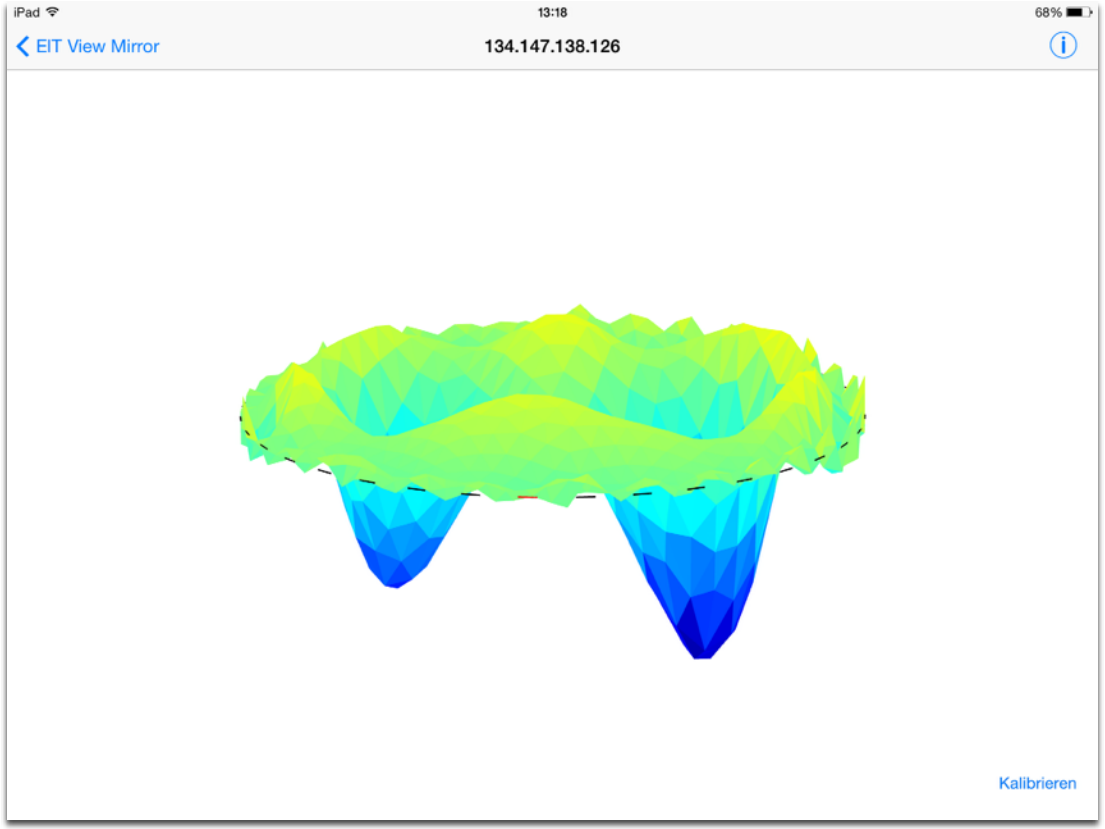


- Values measured on one Tesla K20

GPU Based Image Reconstruction

Visualization in Real-Time

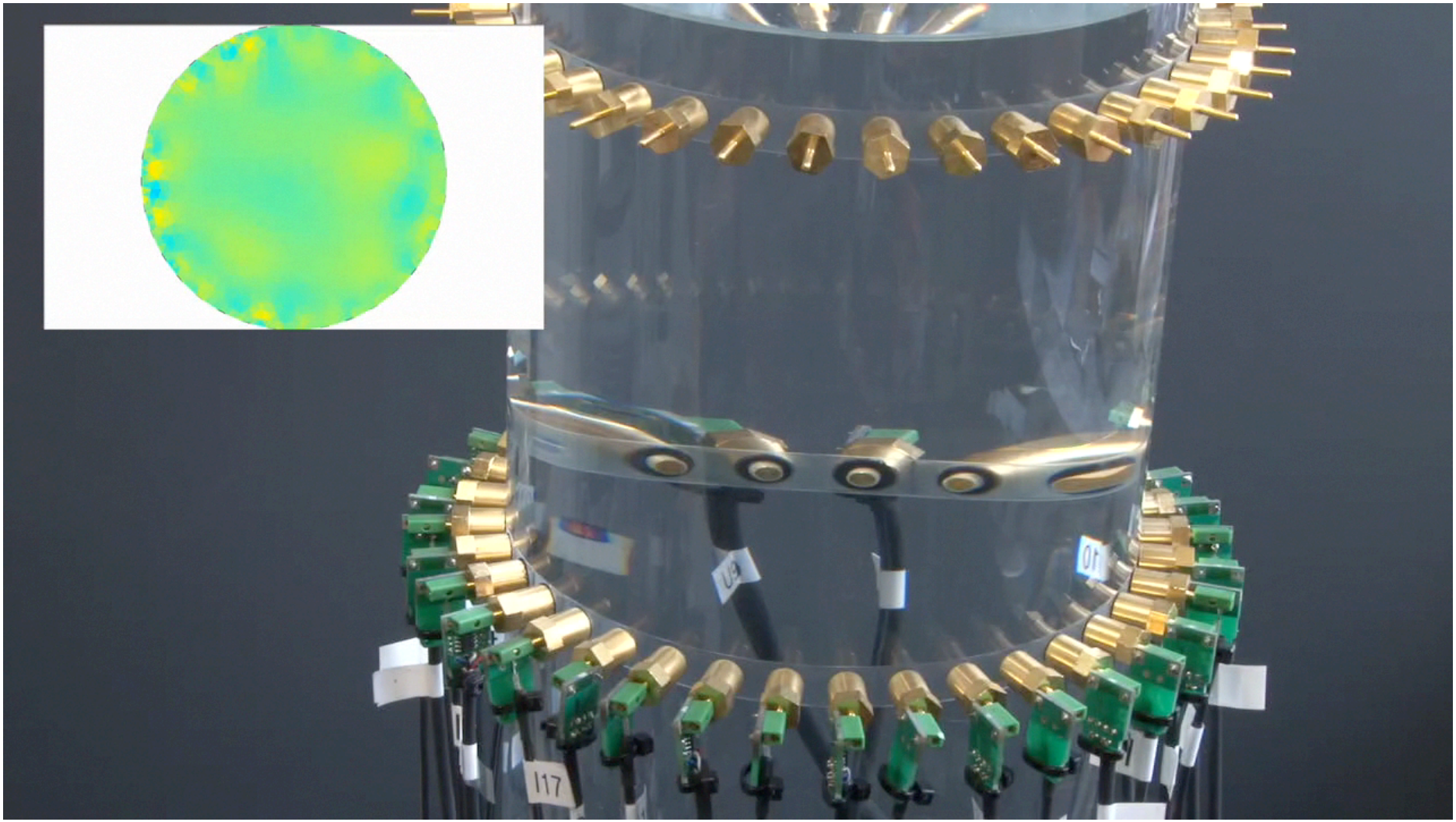
- Streaming reconstructed data to imaging server (i.e. iPad)
- Only subset of images viewable in realtime



Content

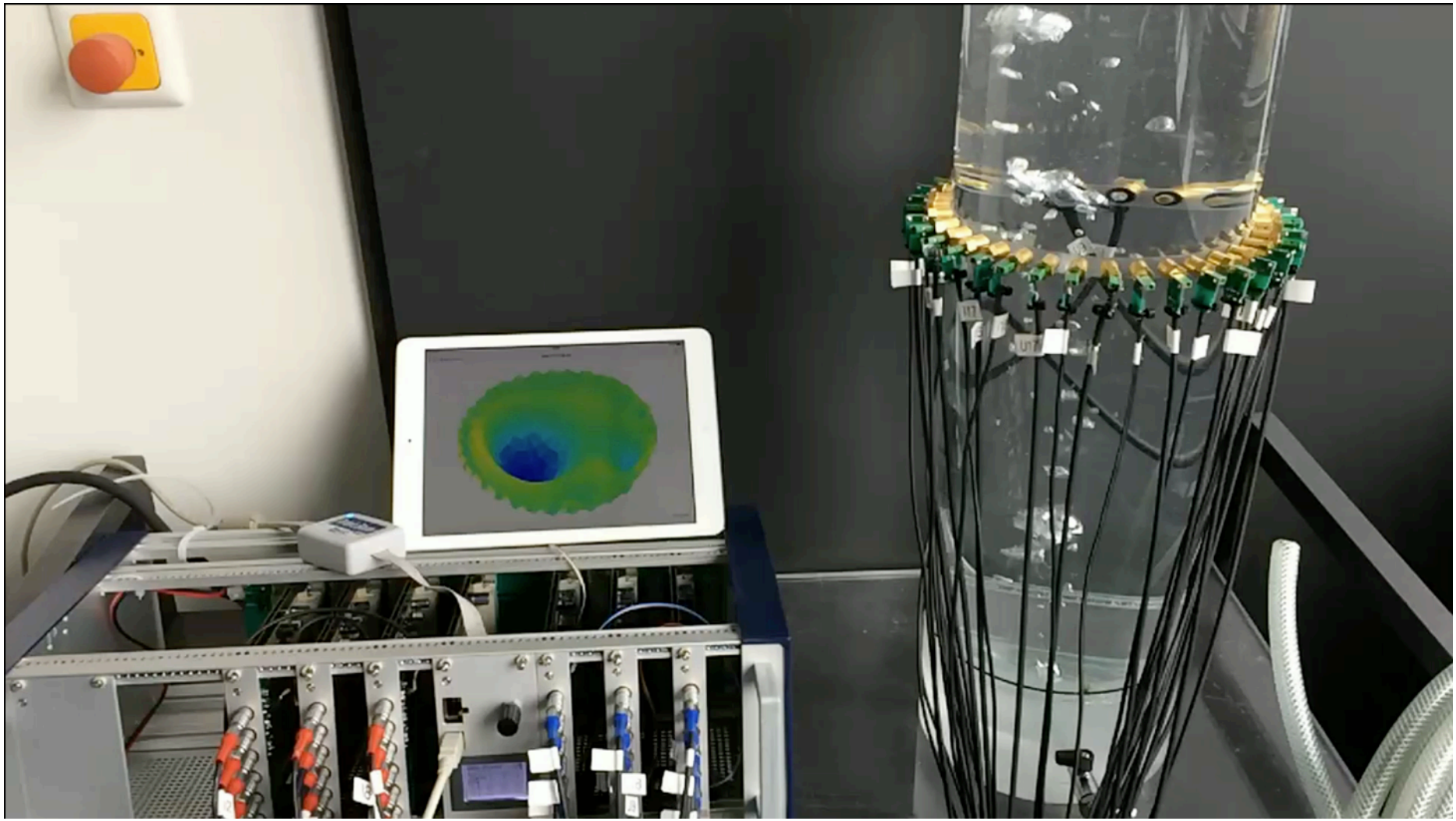
System in Action

Video Presentation



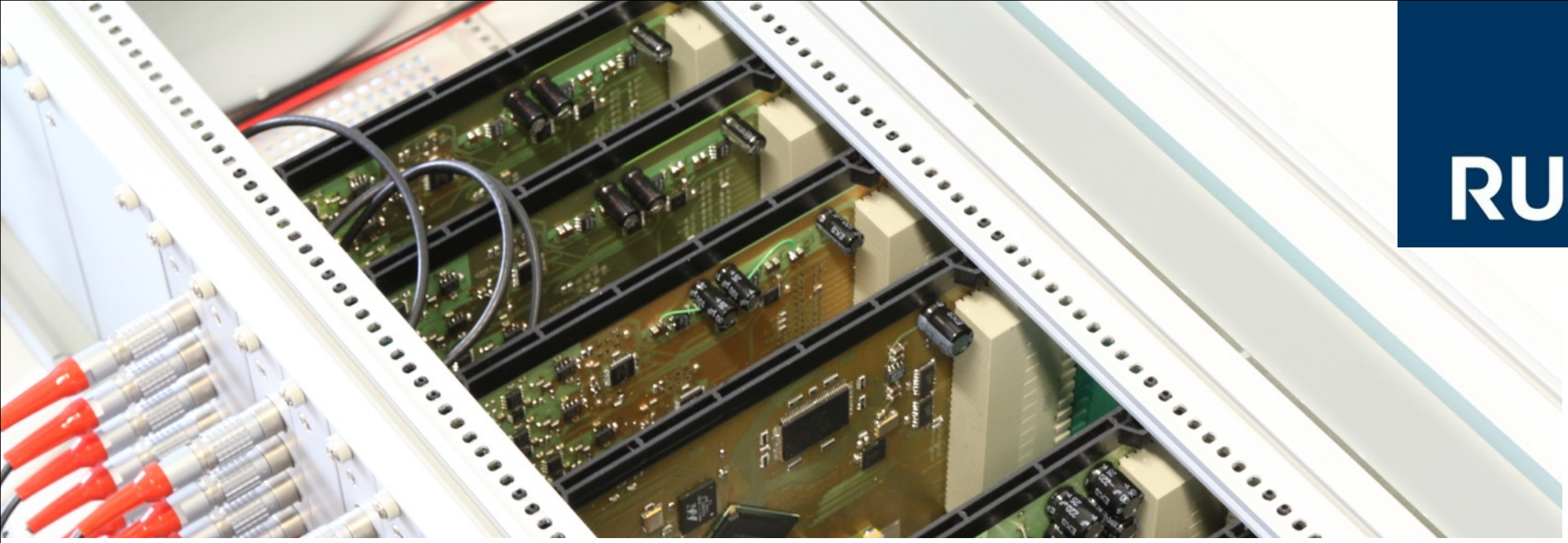
System in Action

Video Presentation



Conclusion

- Soft-field tomography using non-ionizing radiation
- Pipelining and parallel hardware design necessary
- GPU for solving the linear systems
- Metaprogramming for analytical integration
- Pipelining to better utilize GPU recourses



RUB

RUHR-UNIVERSITÄT BOCHUM

Thank You for your being here ;)

LEHRSTUHL FÜR
Elektronische Schaltungstechnik