AutoProg
Project Overview

TECHNISCHE UNIVERSITÄT DARMSTADT

Christian Bischof
FG Scientific Computing
Hochschulrechenzentrum
TU Darmstadt
christian.bischof@sc.tu-darmstadt.de

Scientific Computing Staff

Michael Burger	michael.burger@sc.tu-darmstadt.de
Christian Iwainsky	christian.iwainsky@sc.tu-darmstadt.de
Artur Mariano	artur.mariano@sc.tu-darmstadt.de
Johannes Willkomm	johannes.willkomm@sc.tu-darmstadt.de

Automatic Programming and Code Generation

TECHNISCHE UNIVERSITÄT DARMSTADT

Welcome to our practical course

„Automatic Programming and Code Generation“

- Organisational details
  - To be held roughly every second summer term
  - Workload 4 SWS, resulting in 6 CP
  - Dates 13:30-15:10 on Tuesdays and Thursdays
  - Building S4|14, Room 3.0.01
  - Initial meeting and lectures in the first two weeks
  - After that once a month (see Moodle for details)
  - Weekly meetings of groups with their advisors

Automatic Programming and Code Generation

TECHNISCHE UNIVERSITÄT DARMSTADT

Contents

- Language: Subset of C++ or MATLAB
- Example code
  - 2D Heatflow simulation, provided in C++ or MATLAB version
- Automated source code transformation
  - Automatic differentiation (Group A)
  - Parallelization (Group B)
- Tools to use (either one of)
  - ROSE, a parsing, analysation and AST manipulation infrastructure for C and C++
  - XML and XSLT, for general purpose tree manipulation

Example code: 2D Heatflow simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

The heat equation describes the diffusion of heat in a homogeneous medium with thermal diffusivity a

$$\frac{\partial T}{\partial t} = a\Delta T$$

That is, the change (first derivative) in time of the temperature T is equal to the second derivative in space

- This is called a *partial differential equation* (PDE)

In two dimensions:

$$\frac{\partial T}{\partial t} = a\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

Example code: 2D Heatflow simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

The heat PDE is solved by *discretization*, that is, we consider only a finite amount of spatial grid points

- The domain is rectangular, $\Omega \equiv [0, 0.01] \times [0,1]$,
- discretized into a rectangular grid with $n_x \times n_y$points
- with distance h_x in the x-dimension and h_y in the y-dimension
- The temperature at the grid points is stored in a 2D array
  - $T_{i,j}$ is the temperature at point (ih_x, jh_y)
- The second derivatives in space are approximated by *finite differences*

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T_{i+1,j} + T_{i-1,j} - 2T_{i,j}}{{h_x}^2}$$

Example code: 2D Heatflow simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

The discretization allows us to compute the right-hand side of the heat PDE and so we also know the left-hand side

This directly leads to the explicit Euler method

$$T_{i,j}(t+\delta t) = T_{i+1,j}(t) + \alpha\left(\frac{T_{i+1,j}(t)+T_{i-1,j}(t)-2T_{i,j}(t)}{{h_x}^2}+\frac{T_{i,j+1}(t)+T_{i,j-1}(t)-2T_{i,j}(t)}{{h_y}^2}\right)\delta t$$

At the boundaries of the domain we have to to something else

- Either we prescribe T there: Dirichlet boundaries
- Or we prescribe the change (derivative) of T, that is the flow of heat in or out of the domain: Neumann boundaries

Example code: 2D Heatflow simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

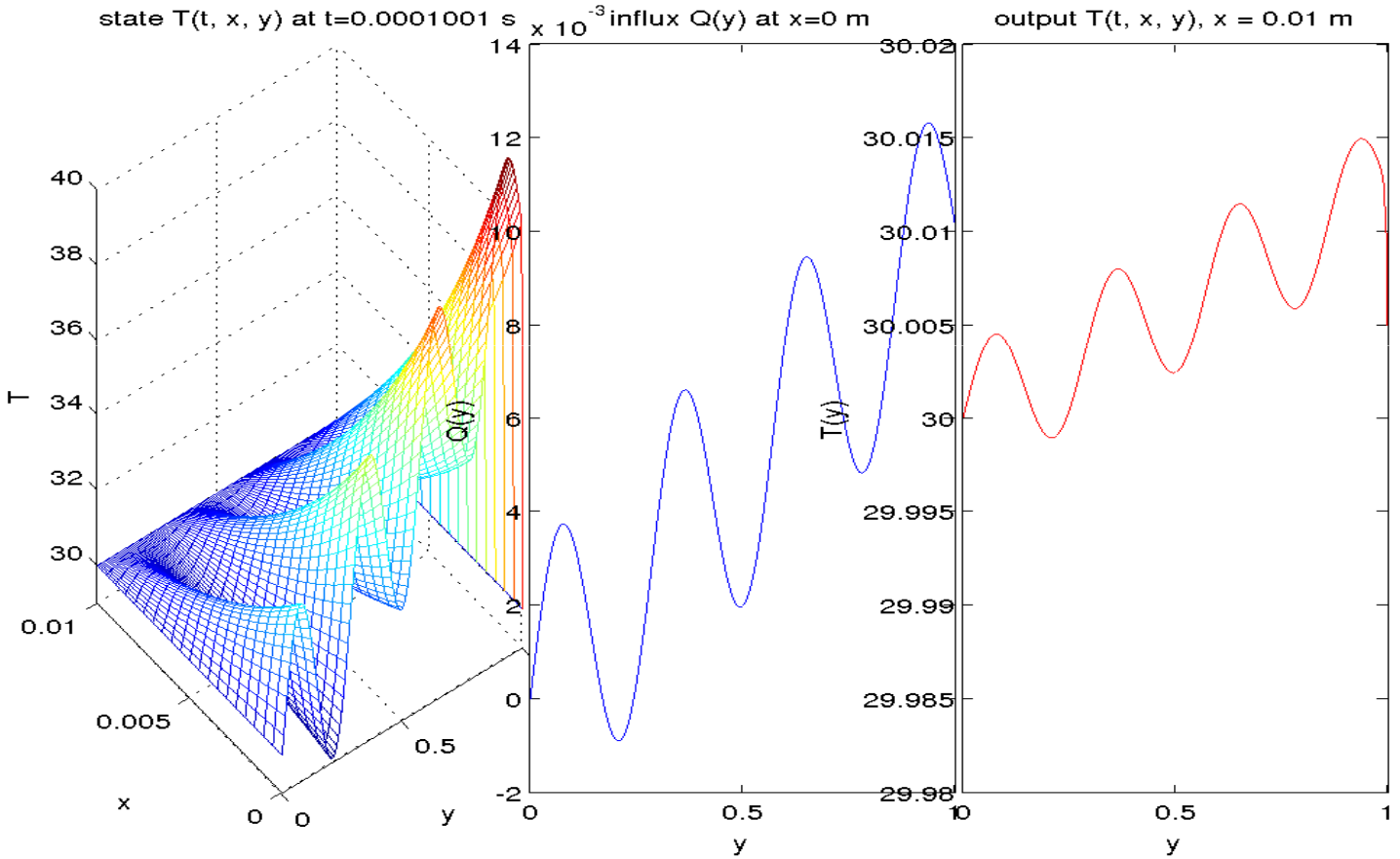
The method we described can now be used to simulate the temperature on our domain from time $t = 0s$ until some final time t_f

- Initial condition:
  - at $t = 0s$, T is given, e.g. $T = 30°C$ everywhere
- Boundary conditions:
  - On the left ($x = 0m$), a certain heat flux Q flows in
  - On the right ($x = 0.01m$), the flux is zero
  - On the upper and lower border T is fixed
- We are interested in $T_{out} = T(t_f)$ on the right border

This simulation is implemented in the function

```
Tout = poissonSimulation2D(Qin, Tinitial)
```

Example code: 2D Heatflow simulation



Inverse Heat Conduction Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

- Let us put the simulation into a wider context: We assume that we have measuments M of the temperature on the right border, but we don‘t know what the influx Q was
- This is called the Inverse Heat Conduction Problem (IHCP), which is an example of *parameter estimation*
- It can be solved by optimization, for example using the function **fsolve** in Matlab‘s Optimization Toolbox
- For that we compute the *residual* $R = T_{\text{out}} - M$,
  - this is done by the function $R = F(Q)$
- **fsolve** will try to find Q such that R is the zero vector
- F is implemented in Matlab as the function heatflow:

```
[Residual] = heatflow(Qin, Tinitial, Meas)
```

Inverse Heat Conduction Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

The function **fsolve** searches for a local minimum of Q by computing the derivative of R with respect to Q

- This is the *Jacobian matrix* $\mathrm{J} \in \mathbb{R}^{n_x \times n_x}$ matrix
  - The entry $J_{i,j}$ has the derivative of R_i with respect to Q_j
  - This shows how R_i depends on Q_j
- This information tells **fsolve** how to change Q such that R becomes smaller
- We have to provide a starting point (guess) for Q

Derivatives in optimization

TECHNISCHE UNIVERSITÄT DARMSTADT

When we just give the function F to **fsolve**, it computes the derivative by *finite differences* (FD)

$$\frac{\mathrm{d}\mathbf{R}}{\mathrm{d}\mathbf{Q}_i} = \frac{F(\mathbf{Q} + h\mathbf{e}_i) - F(\mathbf{Q})}{h}$$

That is, in order to compute the derivative of $\mathbf{R}$ with respect to the i-th component of $\mathbf{Q}$, it distorts the i-th component by the step size h ($\mathbf{e}_i$ is the i-th unit vector) and evaluates F, that is, it runs the simulation

Evaluating J by FD costs n_x function evaluations

Derivatives in optimization

TECHNISCHE UNIVERSITÄT DARMSTADT

Derivatives by FD have two disadvantages

- They are imprecise
- They are slow

What are the alternatives?

- **fsolve** can be given the true derivative by the user
  - Write a second program that computes the derivative
  - Or use *automatic differentiation*, which exactly does that, but automatically

Parallelization in Scientific Computing

TECHNISCHE UNIVERSITÄT DARMSTADT

PDEs are the most important tool for describing natural phenomena, and they have to be solved numerically by discretizing

Discretization leads to huge amounts of data that have to be computed and stored, in our case the grid points

- For more detailed results, finer resolutions are needed and that means even more grid points
- And yet these are still relatively coarse approximations
  - Compare the number of grid points in our 2D simulation domain to the number of atoms in a piece of tin of same size

These problems can only be solved by combining the power of many computers and/or processing cores

Parallelization in Scientific Computing

TECHNISCHE UNIVERSITÄT DARMSTADT

For parallelization there are several main paradigmas

- Distributed memory parallelization
  - Several computers exchange messages over the network, usually using MPI
- Shared memory computing
  - Several cores in a multi-core processor work together on the same data
  - Work has to be given to the cores, often using the OpenMP language extensions available in Fortran and C/C++
- Accelerators (GPUs) are also staging a revival
  - But they are just as awkward to program as then

Code generation in Scientific Computing

Combination of these different ways to organize compute power leads to complex hierarchical machines that still lack the adequate software

- Software that allows programmers to spend more time working on a higher level, on their actual goals, and not on problems that are of the bookkeeping and housekeeping type
- Code generation is one way to help in this regard
  - AD gives programmer code that computes the derivative of the computations that he wrote
  - OpenMP directives are a very concise way to express parallelism, the compiler takes care of the details of expanding this to plain C or Fortran
  - Domain Specific Languages (DSLs) allow the user to express their problems concisely and take care of solving them, often using AD and parallelization under the hood

Code Transformation Tasks in AutoProg

TECHNISCHE UNIVERSITÄT DARMSTADT

We have two main tasks to do in our example project

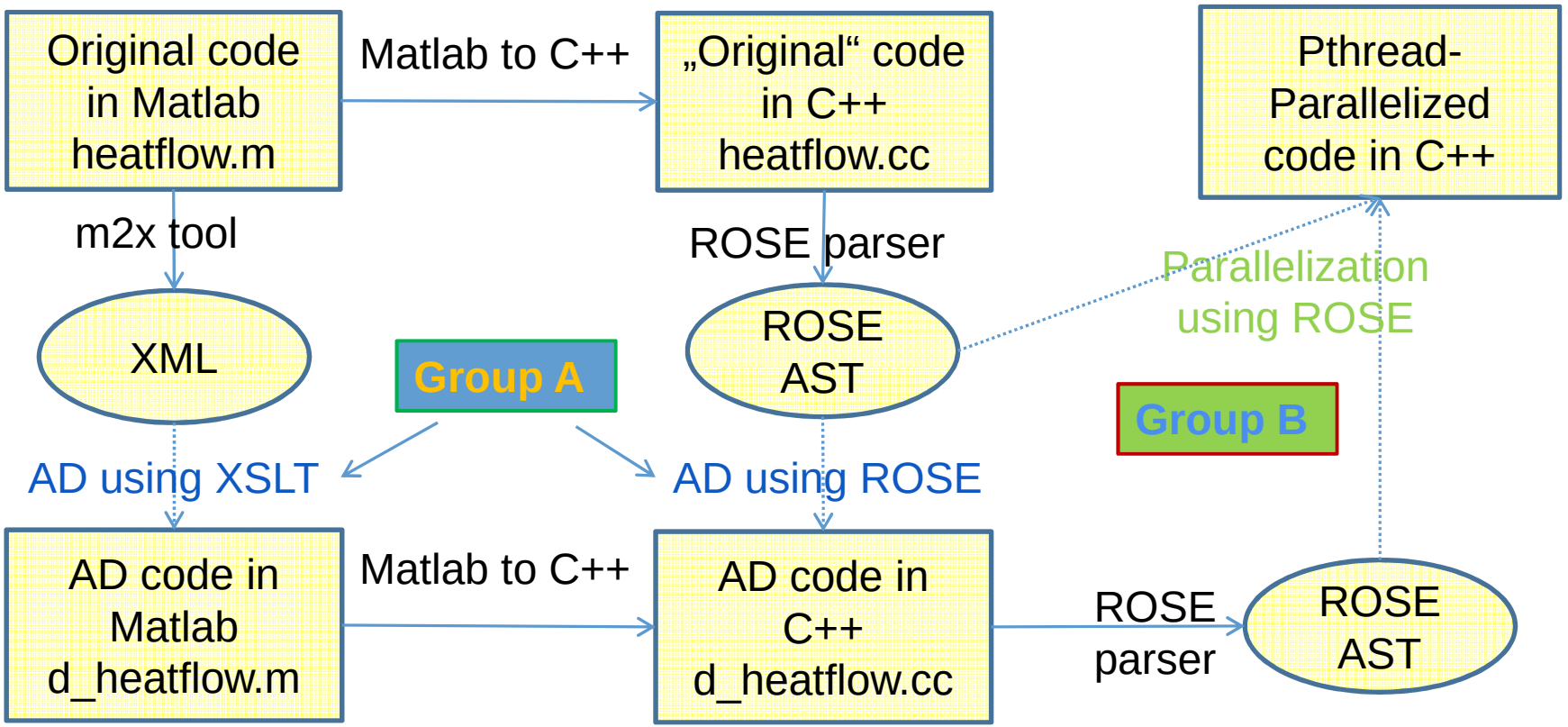
- Need the true derivative of our simulation code **heatflow**, so that **fsolve** works more efficiently
  - We shall do this by Automatic Differentiation

Group A

- The simulation code, and also its AD version, can be parallelized so that it runs faster on multi-core processors
  - We shall parallelize the code as mandated by the OpenMP directives in the code

Group B

Code Transformation Flow Chart



There are several ways to the goal or several things that you can do.

File d_heatflow.cc is of course provided from the start to Group B!

Tasks for Group A

TECHNISCHE UNIVERSITÄT DARMSTADT

Design and implement a tool that can do AD in vector forward mode of the function heatflow

- The result should be C++ code similar to our reference solution, so that Group B can use it as input

There are two options to do the AD transformation

- With XML and XSLT, starting from the Matlab code
- With ROSE, starting from the C++ version of heatflow

Group A: The XML/XSLT way

TECHNISCHE UNIVERSITÄT DARMSTADT

- Use our little parser m2x to produce a XML representation of the Matlab code
- Write XSLT stylesheets to transform the code step by step to the desired result
- A stylesheet to output C++ again (the last step) is provided
- You can use tcproc.sh, which allows you to specify the series of XSLT stylesheets in an XML document and apply these in a single command
- The transformation of the Matlab code to C++ is implemented in this manner and serves as an example
- Pro: Easy debugging, by looking inside the XML at each stage, and the solution will be rather short
- Con: you will need time to learn XSLT

Group A: The ROSE way

TECHNISCHE UNIVERSITÄT DARMSTADT

- Use ROSE to parse the C++ version of heatflow
- Use the code analysis and transformation tools that ROSE provides to implement the solution
- Rose also has a Pretty-Printer to output C++ again
- Pro: ROSE is a professional tool, and is programmed in C++, which you hopefully know
- Con: you will need time to learn ROSE, and probably also have to debug some zero-pointer dereferences

Tasks for Group B

TECHNISCHE UNIVERSITÄT DARMSTADT

Design and write a tool that implements a subset of OpenMP

- the example C++ codes contain OpenMP directives, so that the computations run in parallel when compiled with an OpenMP-enabled compiler (e.g. g++ -fopenmp)
- The goal now is to transform the code in such a way that the parallelization is implemented with simple multi-threating (Pthreads) so that a regular compiler suffices (that is, g++ invoked without the –fopenmp flag)

Task for both groups

TECHNISCHE UNIVERSITÄT DARMSTADT

Work in a team to achieve the goal

Use the adequate and recognized methods for project management and software construction

Give three presentations

- After planning and research phase (End of May, 15-20 min)
- Mid term (End of June, 15-20 min)
- Final project presentation (End of semester, 30-45 min)

Hand in the source code of your solution

Write documentation for your software (10-20 pages)

Write a project report (3-10 pages)

Organization of groups

TECHNISCHE UNIVERSITÄT DARMSTADT

Please find yourself together in groups of 3-4 people

When there are more than two groups, about half of the groups should do the Group A project (AD) and the other half Group B (Parallelization)

A few more Groups B are welcome since we have more staff to help with Parallelization that with AD

Project environment

TECHNISCHE UNIVERSITÄT DARMSTADT

- You can download the initial sources from Moodle
- Read the file README to get you started
- Building, compiling, generation of files is done via make
- The results are run using Matlab
  - In particular the optimization to solve the IHCP
  - C++ code is run using MEX interface functions (provided)
  - There are also Matlab scripts to just run the model function and/or the derivative AD function and to check the results against finite differences and AD code generated by ADiMat