Lecture 9

Code Generation

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Content

- HPC and its Challenges
- The 3P’s
- Code Generation
- Demonstration

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

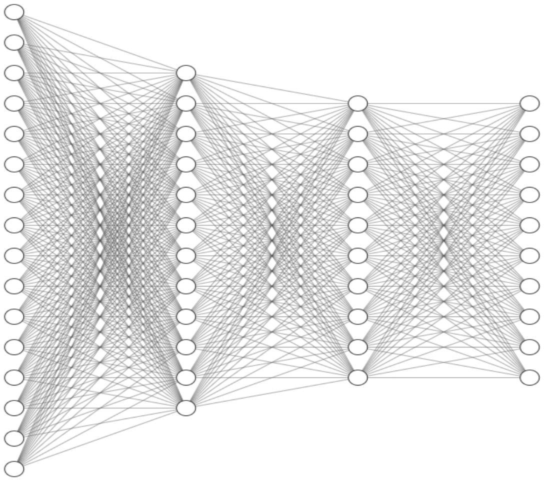
TECHNISCHE FAKULTÄT

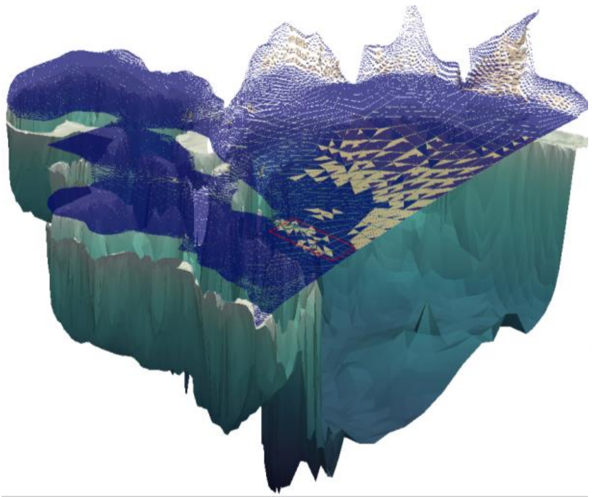
HPC and its Challenges

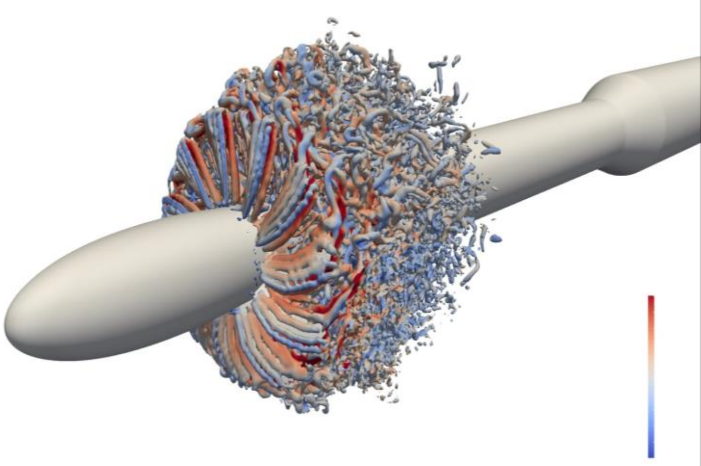
HPC in General

Essential topic for scientific research and computational science

- Simulation of natural phenomena
  - Weather/climate modeling, fluid- and aero-dynamics, molecular dynamics, nuclear fusion, astrophysics, etc.
- ML/AI
  - Natural language/speech processing
  - Prediction of natural phenomena
  - Robotics, computer vision, etc.
- Insurance, Finance, Healthcare, etc.

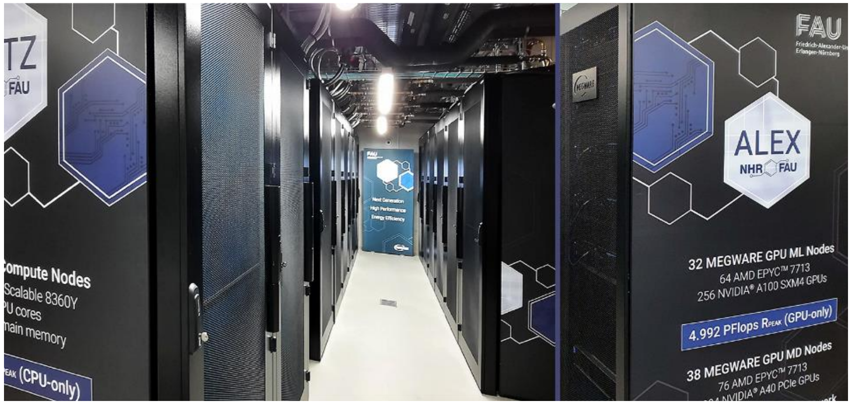






HPC Platforms

- High resolutions required ➔ Multiple millions of data points
- Increasing demand for computational resources
- Regular workstations may have insufficient compute power



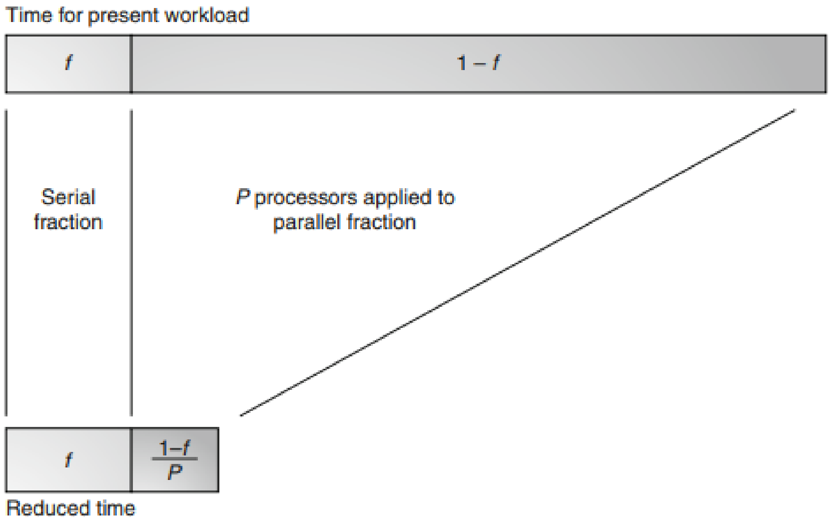
HPC Requirements

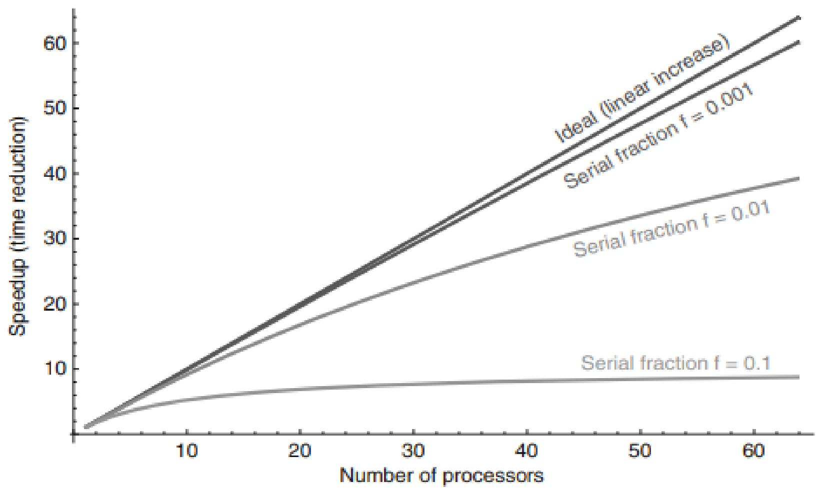
- Important: **efficient utilization** of expensive hardware
- Example Frontier:
  - ~600 million USD acquisition cost
  - 21 MW power envelope (~250 USD/MWh per hour in Germany)
  - ➢ Requires high attention to **energy footprint**
- ➢ Requirements for software running on such systems
  - ➢ Implementations must be **optimized** for target system
  - ➢ Software must be **scalable**

Meaning of Scalability: Amdahl's Law

- Law for speed-up of fixed-size problems
- Often referred as **strong scaling**

$$\text{Speedup} = \frac{1}{f + \frac{1-f}{P}}.$$





cf https://doi.org/10.1007/978-0-387-09766-4_77

The 3P‘s

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

HPC Diversity

- Optimal choice of optimizations is dependent on hardware
- Problem: **very large search space**
- Systems have different
  - Vendors (Intel, AMD, NVIDIA, …)
  - Architectures (cf. Flynn‘s Taxonomy: SISD, SIMD, ...)
  - Chips (CPU, GPU, FPGA, …)
  - Memory interfaces (Registers, Caches, main memory with DDR/Optane/HBM/…)
  - Notions of parallelism (Shared- and distributed-memory parallelism)
  - Parallelization libraries (OpenMP, MPI, CUDA, HIP, …)
  - Compilers
  - Network topologies
  - …

Other HPC Requirements

- Hardware evolves quickly (2006: first petaFLOPS, 2022: first exaFLOPS)

➢ Frequent upgrade of cluster systems

- Changing clusters may impose changes to the code:
  - Extensions to support new hardware and its features
  - Revisit applied optimizations or complete reimplementation
- Next requirement: (performance) **portability**

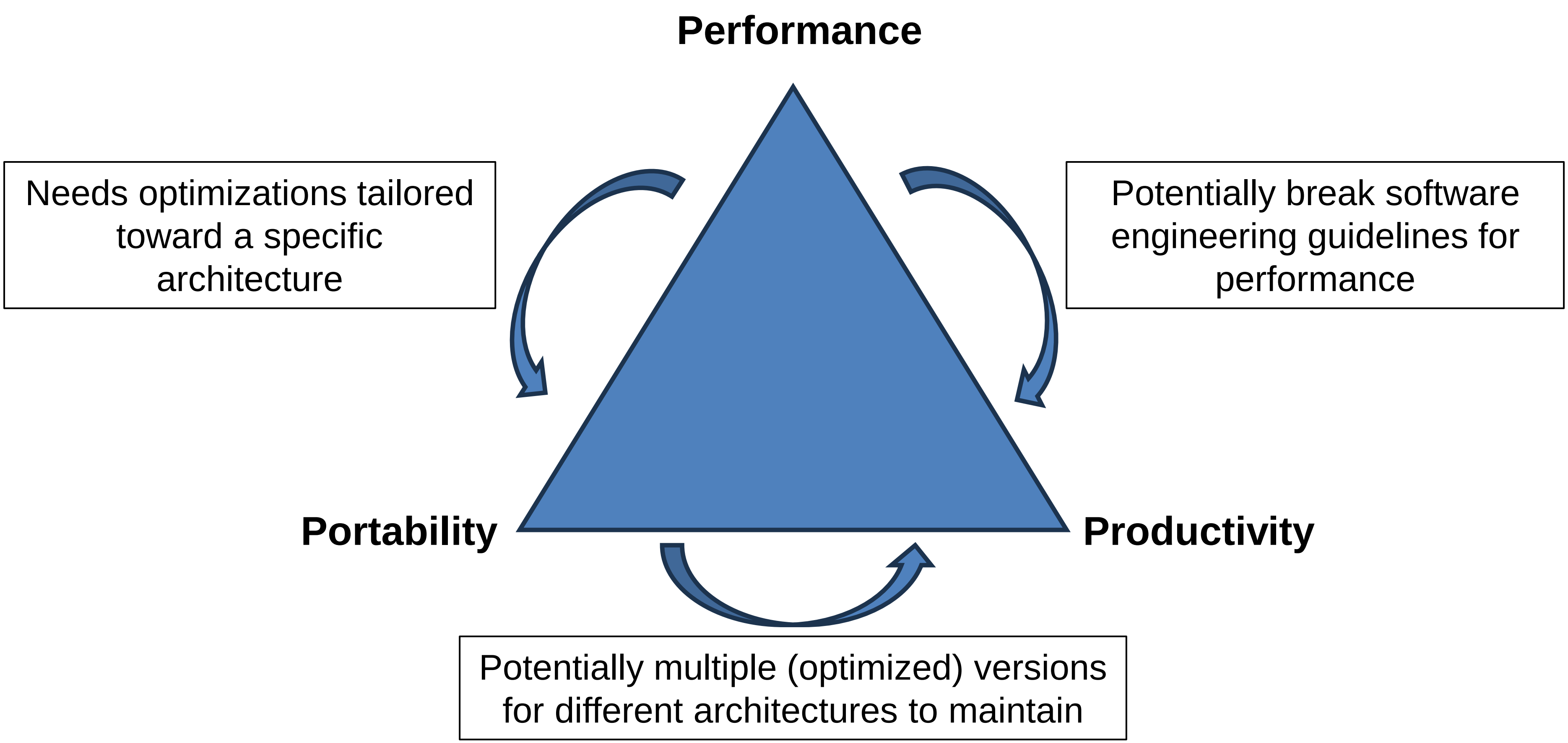
- A software engineering goal: minimize development time
  - Maintain code correctness and readability
  - Enable collaboration between experts from different domains

  ➢ Separation-of-concerns: Domain scientists (e.g. mathematicians) may not be performance-engineers and vice-versa

  ➢ Third requirement: **productivity**

The 3P’s

- All requirements summarized: The 3P‘s
- Are the requirements conflict-free?
- Common trade-offs:



Tackling the 3P’s - Solutions

- Scientific libraries/frameworks (e.g. Kokkos)
  - Provide an performance-portable ecosystem
  - Productivity granted by abstractions of the framework
  - Libraries are commonly highly generic
  - Lack of domain-specific optimizations
- Code Generation with
  - Compiler technology
  - String interpolation (e.g. Jinja templates)

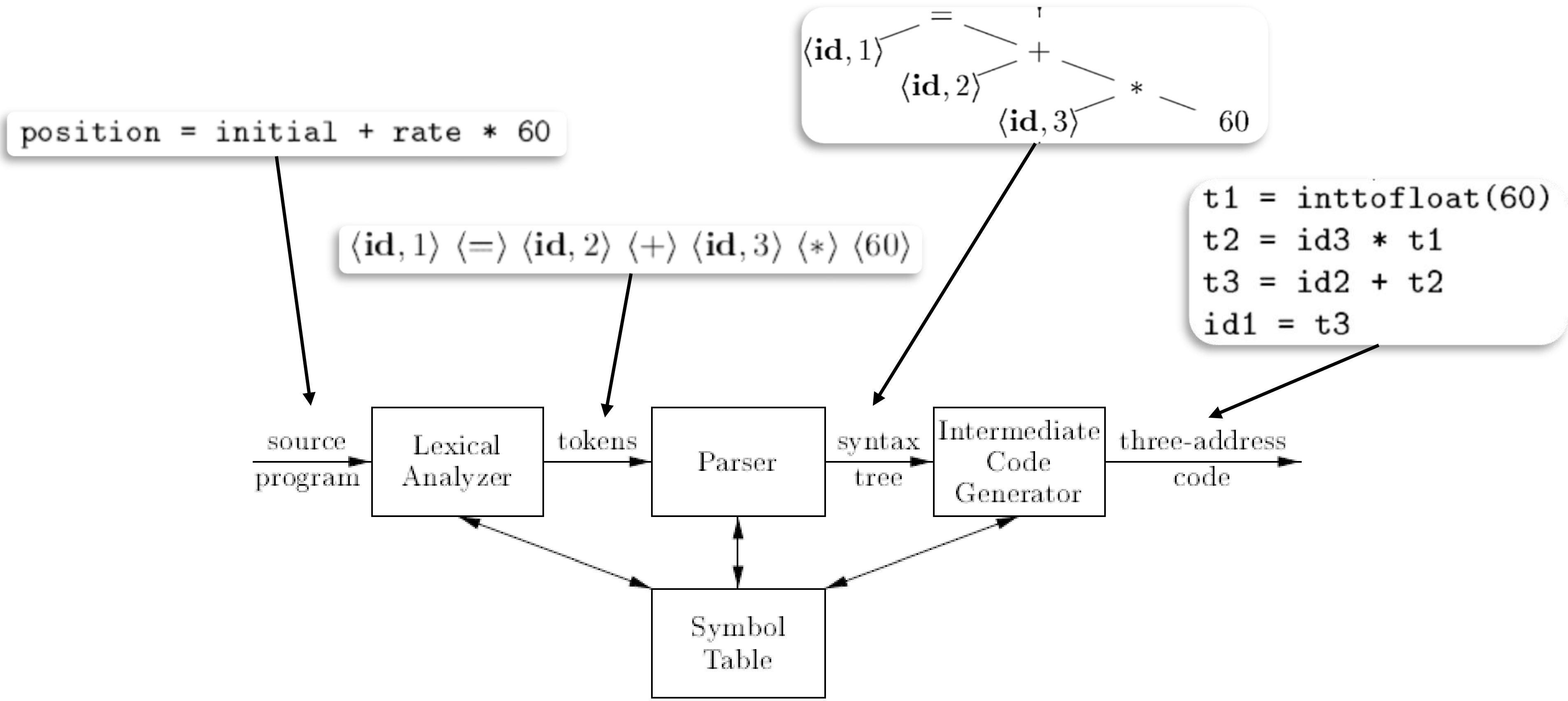
Code Generation

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Source-to-source compilers

- Recap from lecture 2: compiler **front-ends**
- Machine-independent steps



Source-to-source compilers

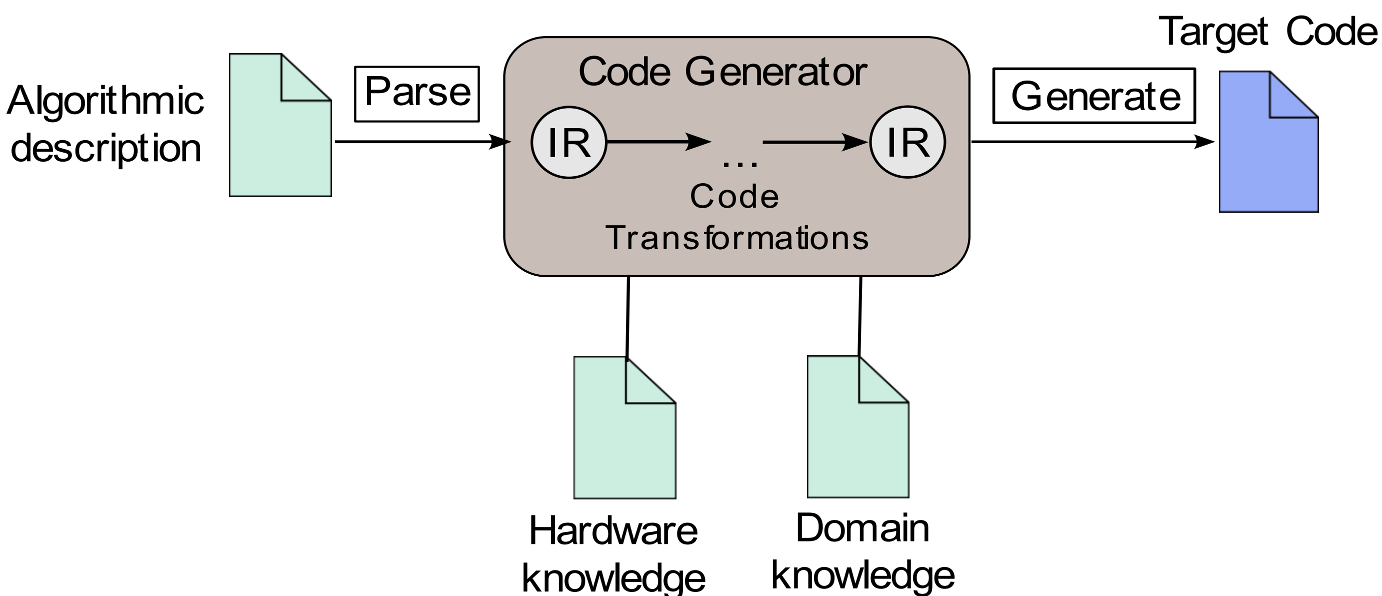
```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1
```

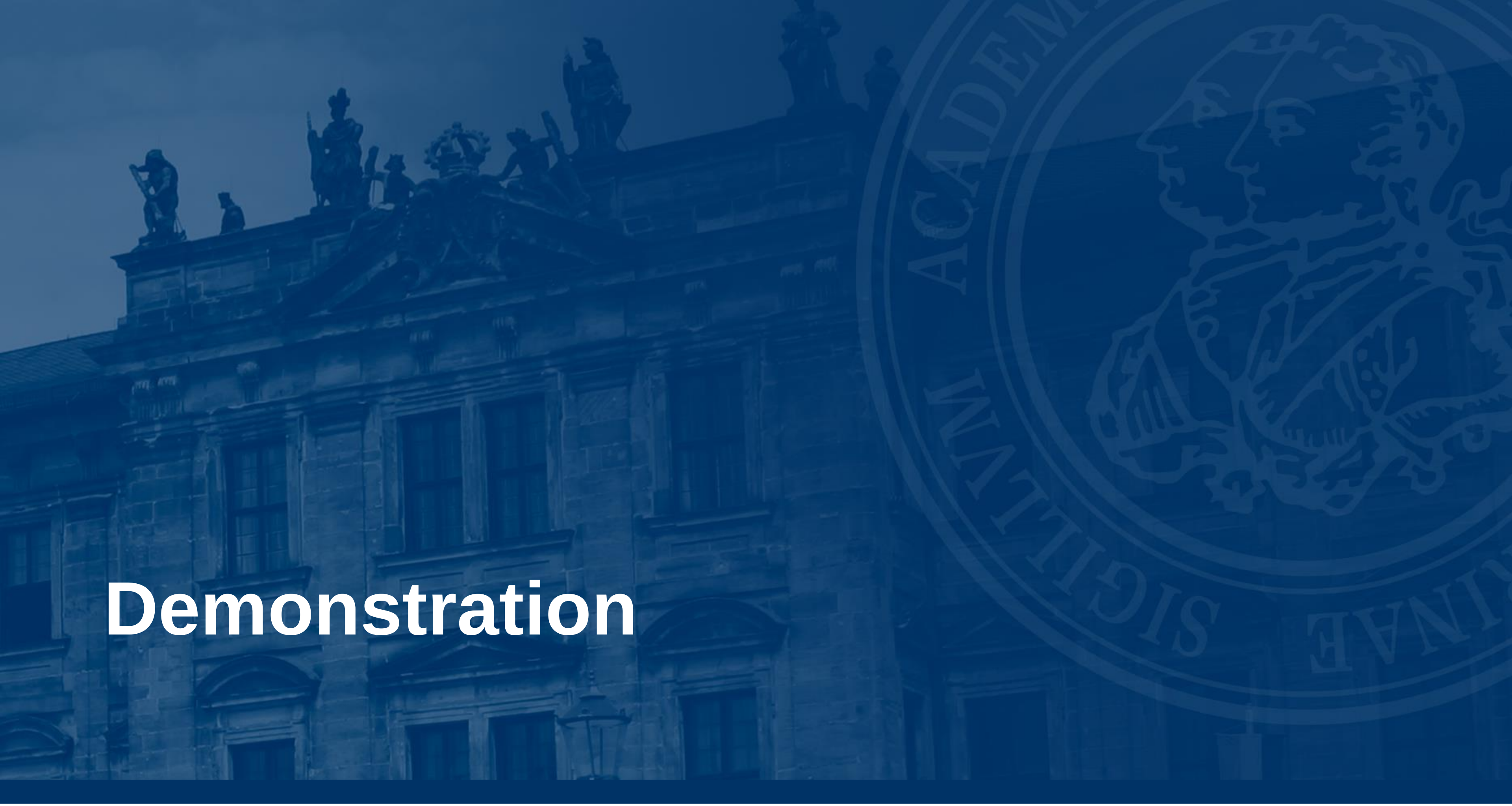
- Recap from lecture 2: compiler **back-ends**
- Machine-dependent steps

- Source-to-source idea:
  - Generate source code (e.g. in a GPL) instead of machine code
  - Transform code in **language A** to code in **language B**
  - Code in language A: typically high-level and domain-specific
  - Code in language B: typically low-level and optimized

Tackling the 3P's with Code Generation

- Description of the algorithm for an abstract machine
- Code generator parses input and produces internal program representation
- Optimizations via code transformations within compiler ➔ **Performance**
- Provide multiple parallelization back-ends ➔ **Portability**
- Separation-of-concerns ➔ **Productivity**
  - Computer scientists augment compiler with HPC knowledge
  - Domain scientists can write algorithms on abstract input layer





Demonstration

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Example: Jacobi 5-Point Stencil

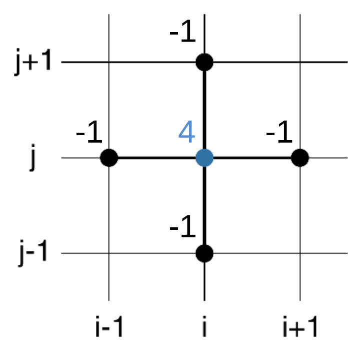
- Very simple and well-known 2D benchmark
- Governing equation

$$-\Delta u = b$$

- Laplace operator $-\Delta$ can be approximated with stencil

$$\begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix}$$

- And simplifies with $\Delta x = \Delta y = 1$ to



Example: Jacobi 5-Point Stencil

- Results in linear system of equations $Au = b$ with matrix

$$A = \begin{bmatrix} D & -I & 0 & 0 & 0 & \cdots & 0 \\ -I & D & -I & 0 & 0 & \cdots & 0 \\ 0 & -I & D & -I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -I & D & -I & 0 \\ 0 & \cdots & \cdots & 0 & -I & D & -I \\ 0 & \cdots & \cdots & \cdots & 0 & -I & D \end{bmatrix},$$

$$D = 4$$
$$I = 1$$

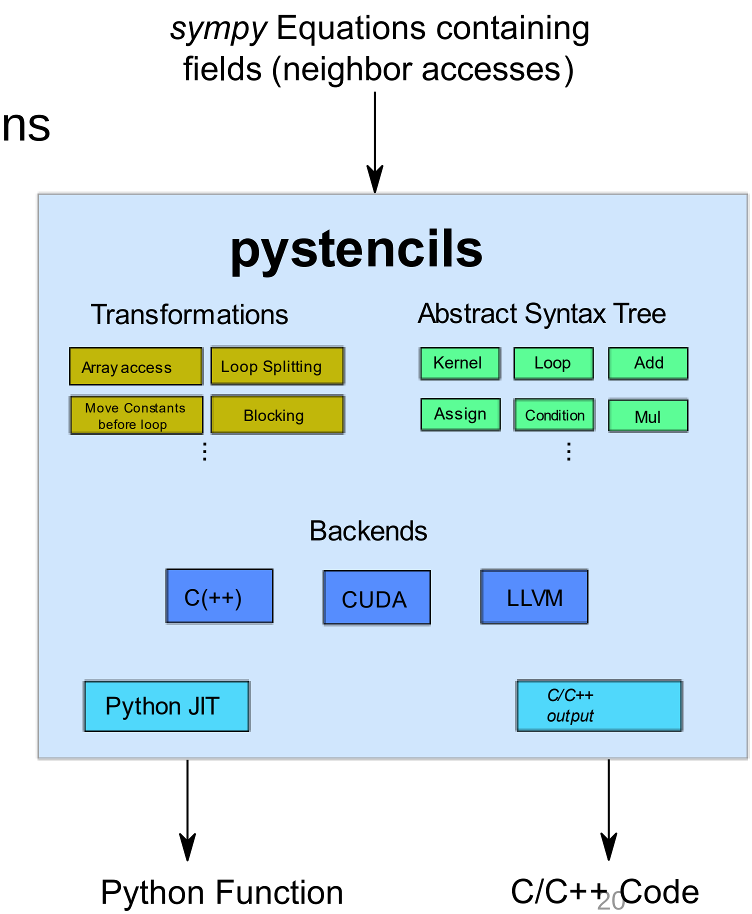
- Approximate solution for system can be obtained with iterative Jacobi method

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

pystencils

- Open-source Python module: https://pypi.org/project/pystencils/
- Kernel code generation for **stencil codes**
- Built on top of computer algebra module **Sympy**
- Can also speed up Numpy computations
- Main applications are numerical **CFD** simulations
- Idea:
  - Define algebraic problem on symbolic input layer
  - Generate fast and **portable C/C++ kernels** from it

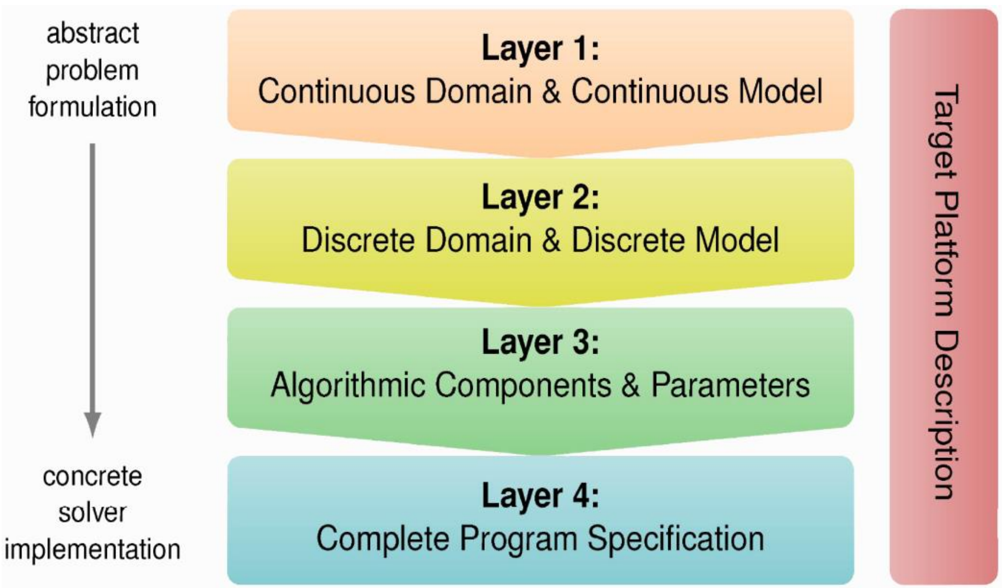
➢ Clear separation of concerns

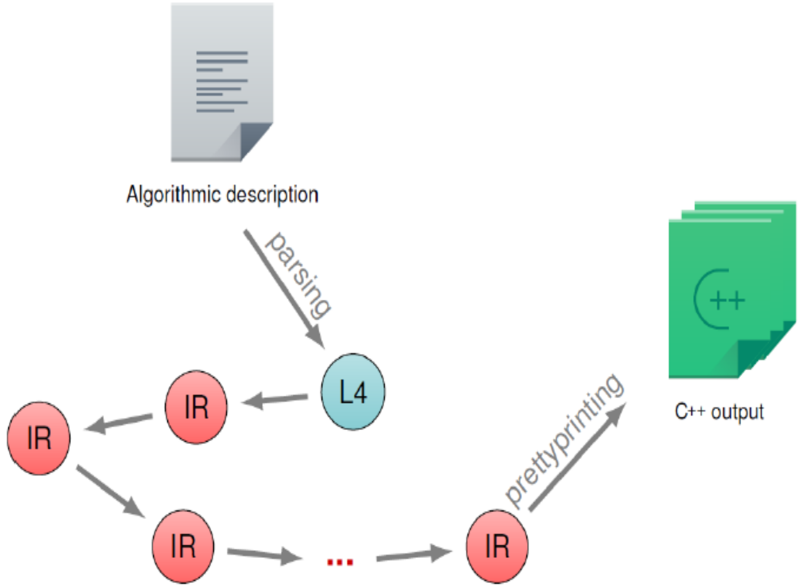


Cf Demonstration Materials

ExaStencils

- Open-source project: https://www.exastencils.fau.de/
- Own multi-layered, domain-specific language (DSL): **ExaSlang**
- Source-to-source compiler written in Scala
- Generation of whole programs **output in C++**
- Main applications are **CFD** simulations
- **Automatic parallelization** with OpenMP, MPI and CUDA





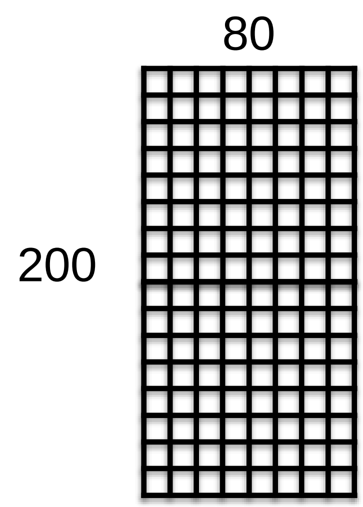
ExaStencils – The 5pt Example

- Declare domain and fields

```
Domain global < [0, 0] to [80, 200] >

Layout FieldLayout < Real, Cell > {
  innerPoints = [80, 200]
  /*...*/
}

Field u < global, FieldLayout, 0.0 >[2]
```

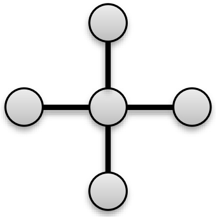


Cf Demonstration Materials

ExaStencils – The 5pt Example

- Declare operators and Jacobi kernel

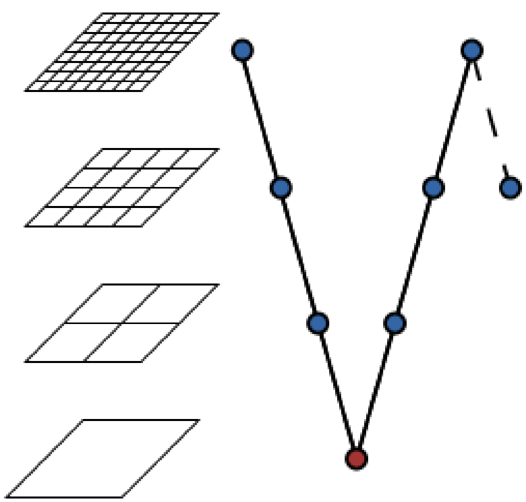
```
Stencil Laplace {
  [ 0,  0] =>  4.0
  [-1,  0] => -1.0
  [ 1,  0] => -1.0
  [ 0, -1] => -1.0
  [ 0,  1] => -1.0
}
```

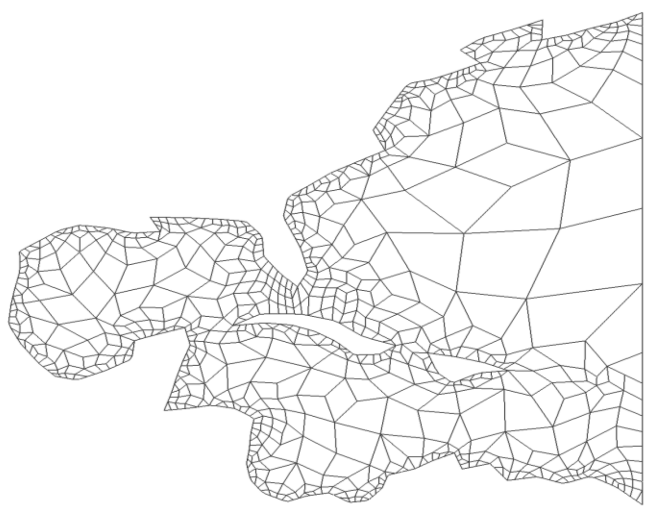


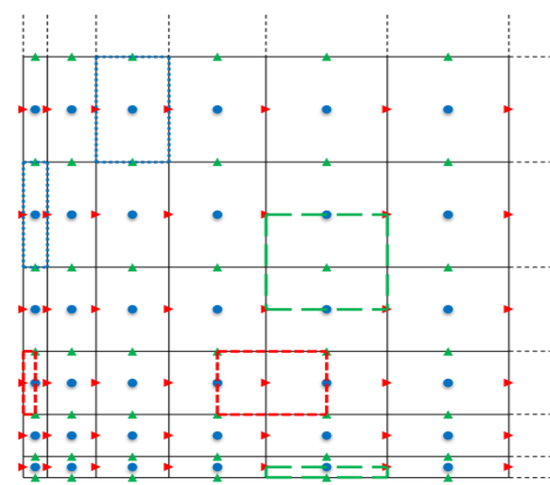
```
Function jacobiStep ( ) {
  communicate u
  loop over u {
    u<next> = u + ( 1.0 / diag ( Laplace ) ) * ( b – Laplace * u )
  }
  advance u
}
```

ExaStencils

- Differences to pystencils?
- Whole program is generated with automatic
  - Build script
  - Synchronization/Communication for parallel programs
  - Automatic memory management
  - I/O routines: Visualization
- ➢ Domain: multi-grid method
- ➢ Support for different grids







Thank you for your attention!
Questions?