Karlsruher Institut für Technologie

Accelerating COBAYA3 on multi-core CPU and GPU systems using PARALUTION

N. Trost, J. Jimenez, D. Lukarski, V. Sanchez

Institute for Neutron Physics and Reactor Technology (INR)
Reactor Physics and Dynamic Group (RPD)

Content

- Motivation
- COBAYA3 ANDES solver
- PARALUTION
- Implementation
- Performance results
- Conclusion

Motivation

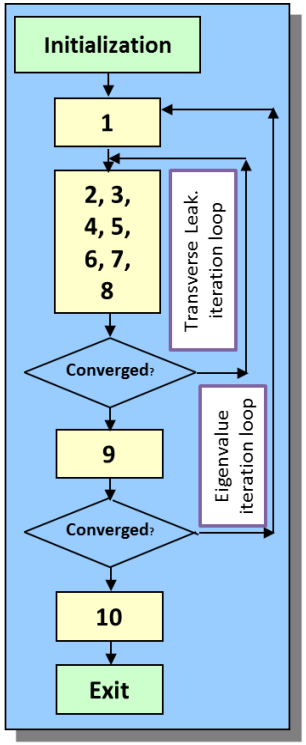
- Multi/many-core systems are here
  - CPUs, GPUs, Accelerators
  - Trend: more cores CPU(16), MIC(60), GPU(2496)
  - High performance, better performance/watt ratio

- Scientific computing is expected to be
  - Fast – bleeding edge technology
  - Scalable – exploit the available hardware resources
  - Sustainable – over years without the necessity of development work

- But: How to adapt well established simulation tools, such as COBAYA3?

COBAYA3 ANDES solver (UPM)

ANDES CODE STRUCTURE

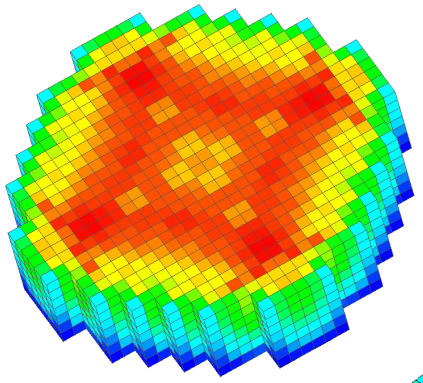


0. **Input data** and initialization
1. **Multigroup eigenvalues** and analytic matrices
2. **Interpolation** of group interface **currents** at arista's
3. Directional **transverse leakage** and ACMFD NGxNG matrices per interface and node
4. **Heterogeneity factors** (if local solution and first pass)
5. Setup of the **linear system** matrix in nodal fluxes
6. *Build and apply* ***pre-conditioner*** ← PARALUTION
7. ***Resolution*** *of the resulting linear system* →
8. Update of nodal **interface currents**

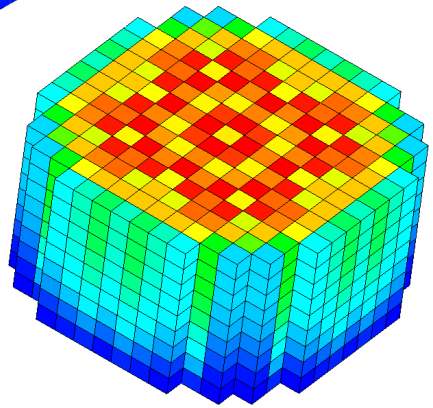
   Test for convergence of transverse leakage
9. Update of **fission source** [Keff] and nodal powers

   Test convergence (if steady-state eigenvalue)
10. **Interface average fluxes** (for local reconstruction)

    Exit from nodal solver



NODAL 3D Power distribution



- Nodal solver for 3D neutron multigroup diffusion (ACMFD method)
- Available sequential solver:
  - GMRES preconditioned with incomplete LU factorization based on threshold with drop off tolerance for additional fill ins

PARALUTION

UPPSALA UNIVERSITET

Karlsruher Institut für Technologie

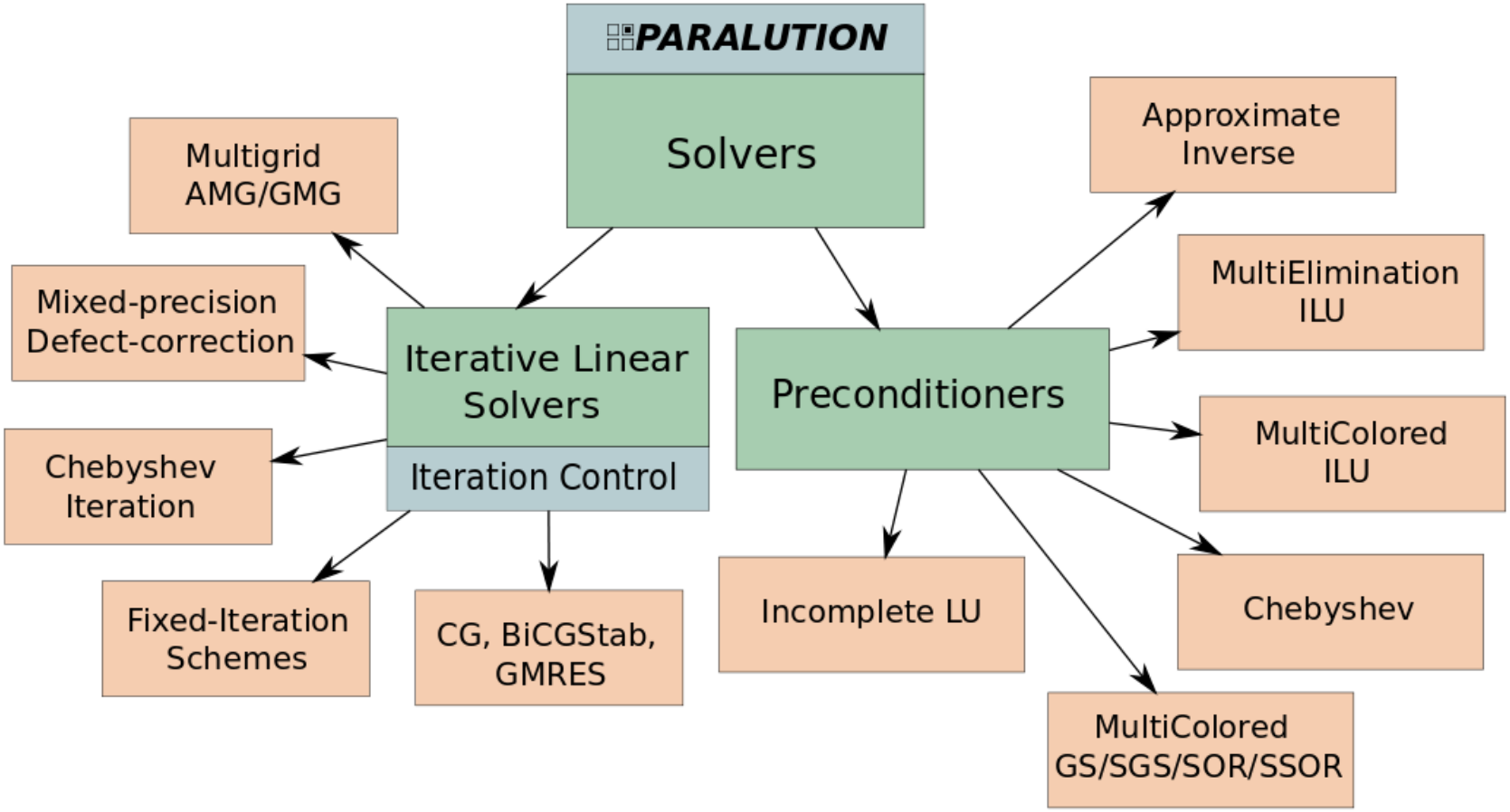
- C++ library for iterative sparse methods
- Developed at Uppsala University (Sweden)
- Hardware abstraction, no knowledge in CUDA, OpenCL, etc. required
- Support for many/multi-core CPU and GPU devices
  - High portability by design, same code can be compiled on CPU and GPU architectures
  - Binary can be executed on any platform, obtaining same results
- Hardware abstraction, no knowledge in CUDA, OpenCL, etc. required

- Variety of different solvers and preconditioners
- Many different matrix formats
  - CSR, MCSR, HYB, ELL, DIA, COO

PARALUTION

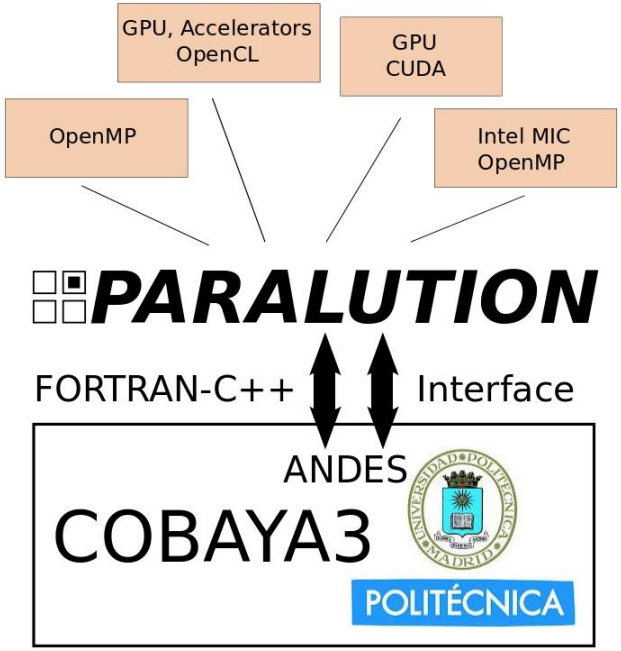
UPPSALA UNIVERSITET

Karlsruher Institut für Technologie



Implementation aspects

- Extension of ANDES
  - Implementation of PARALUTION as an external library with support for all available solvers, preconditioners and matrix formats
- Investigation of communication between FORTRAN and C++ with respect to
  - communication overhead; index shift, separate address space (CUDA, OpenCL)
  - object passing



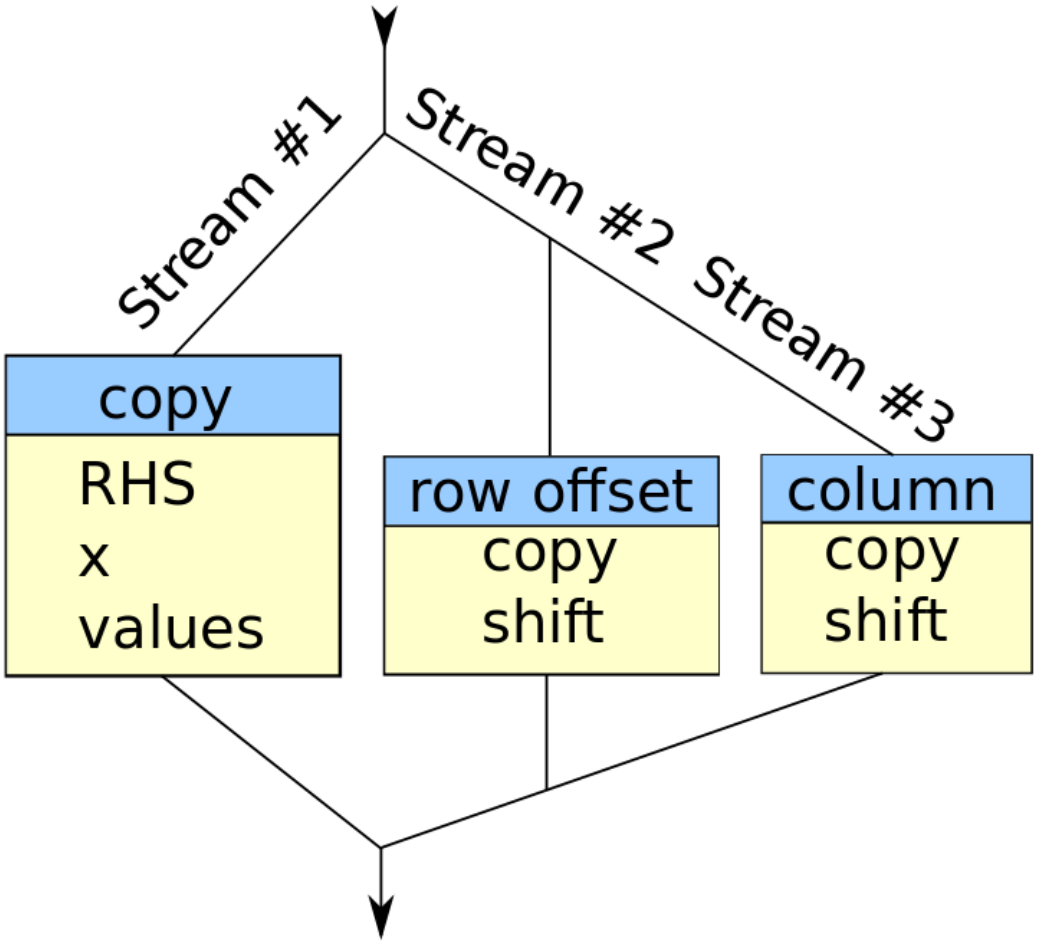
Interaction FORTRAN ←→ C++

- ISO_C_BINDING intrinsic allows
  - C types in FORTRAN: C_INT, C_DOUBLE, C_PTR, C_CHAR
  - Passing of targetable FORTRAN data structures: C_LOC, C_FUNLOC
  - Passing of **ANY** C++ data structures / objects
- Minimization of data transfers by passing pointers
  - Index shift still necessary
- Possibility to ‘keep’ allocated C++ objects in FORTRAN beyond C++ function execution time for e.g. re-use of preconditioners
  - This also applies for device pointers in foreign memory regions (CUDA)

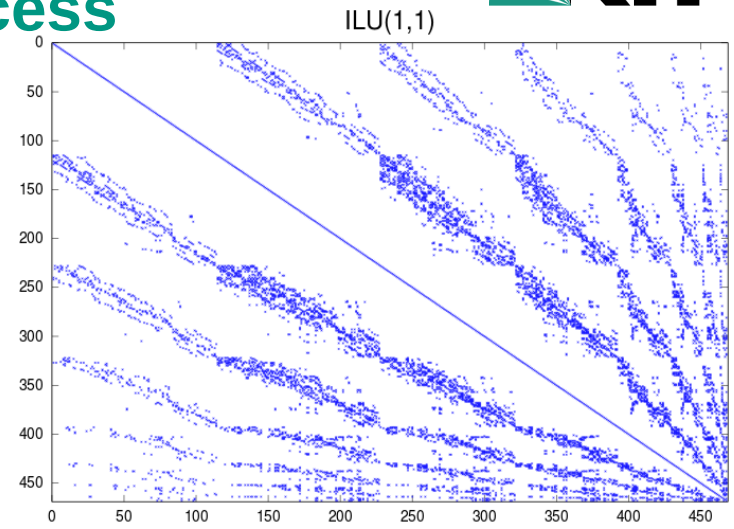
GPU FORTRAN ←→ PARALUTION interface

- Issues:
  - Separate address space – explicit data transfer necessary
  - Bottleneck PCI-E bus

- Solution:
  - Asynchronous data transfer (CC >= 2.0 support asynchronous kernel call and data transfer) while shifting FORTRAN arrays to 0-based indexing



Reusage of the preconditioner system – to further accelerate the solving process



- Constant mesh and neighboring relations
  - → systems sparsity pattern will not change
  - → preconditioning structure can be kept
- Only minor changes in the system matrix each solver call
  - → preconditioning operator can be applied for a certain solver call interval

→ Not only the solving process but also the building of the solver structure can be accelerated.

- Implementation:
  - Fixed preconditioner rebuild interval and
  - Error fall back logic (max iter reach, diverged, ...)

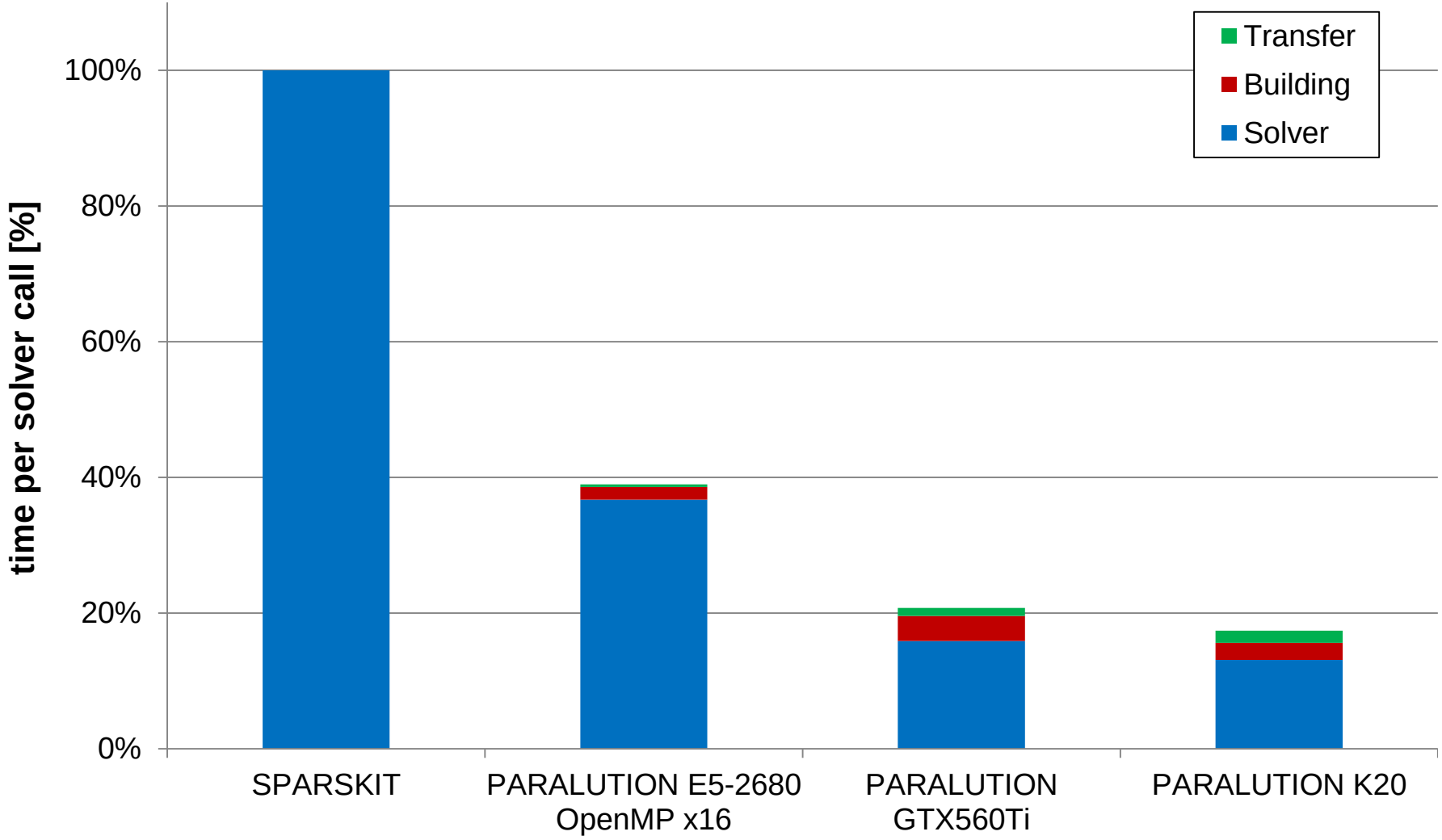
Performance results

- Test case - MOX-3D-HFP.D:
  - 8 energy groups
  - 4x4 nodes per assembly (Cartesian geometry)
  - 20 axial layers
  - Matrix dimensions: n = m = 723,712
  - nnz = 39,618,560
  - Application of GMRES solver
    - with multi-colored ILU-0 – a priori known sparsity pattern (PARALUTION)
    - with ILU-T (SPARSKIT, University of Minnesota)
  - ELL format (GPU), CSR format (CPU) for both, system and preconditioning operator

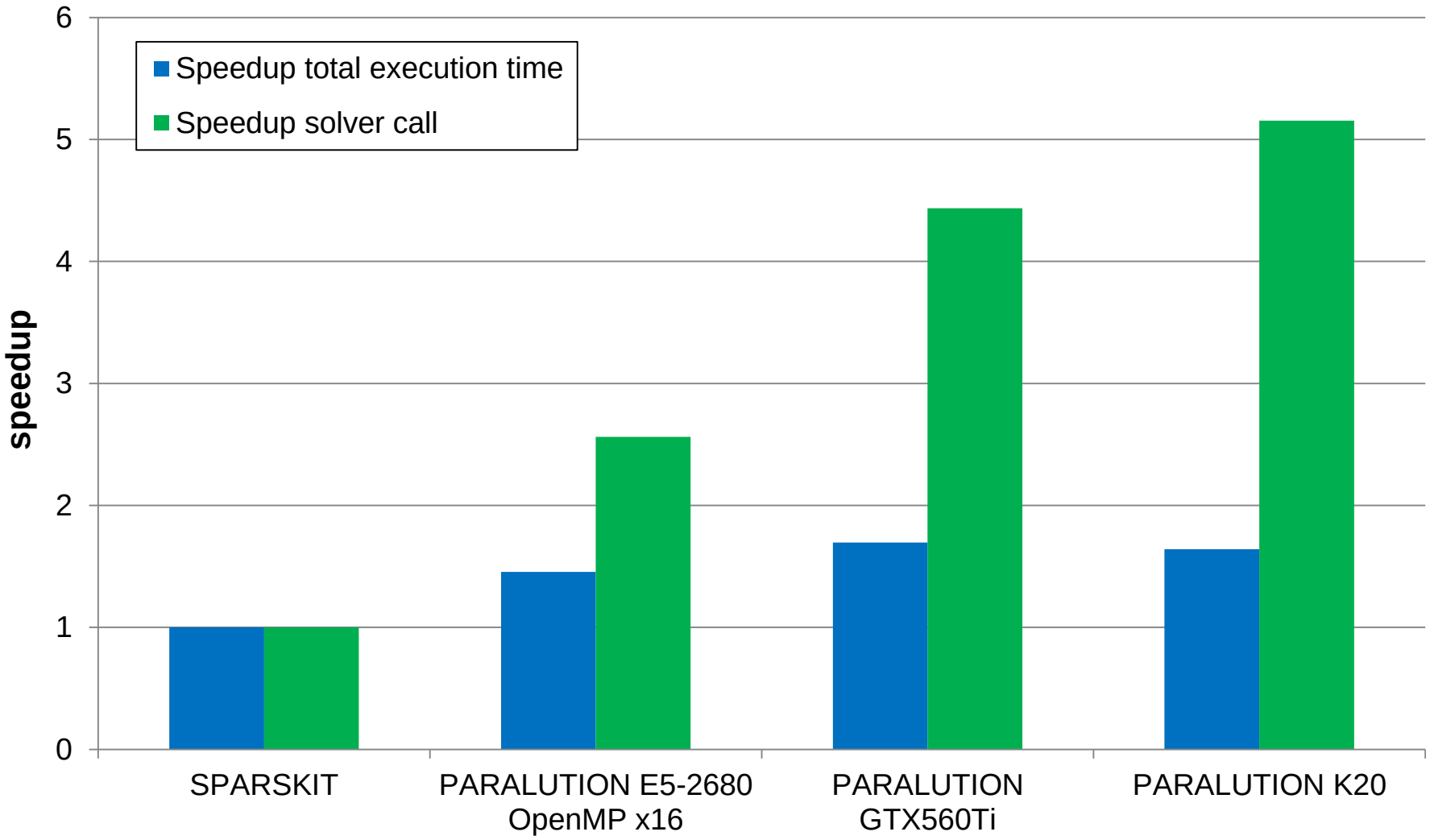
Performance results

- Test system:
  - Dual-socket Intel Xeon E5-2680, **2.7GHz** (Sandy Bridge EP), **16 physical cores** (8 per socket), approx. **3200€** (CPUs only)
  - 64GB ECC DDR3-1600 (quad), **51.2GB/s (peak)**
- GPUs:
  - NVIDIA Tesla K20 (Kepler), 5GB ECC GDDR5, approx. **3200€**
  - NVIDIA GTX560Ti, 2GB GDDR5, approx. **200€** (released in 2010)
- Backends:
  - PARALUTION CPU: OpenMP using up to 16 threads
  - PARALUTION GPU: CUDA
  - SPARSKIT: sequential using 1 core

Performance results – MOX-3D-HFP.D

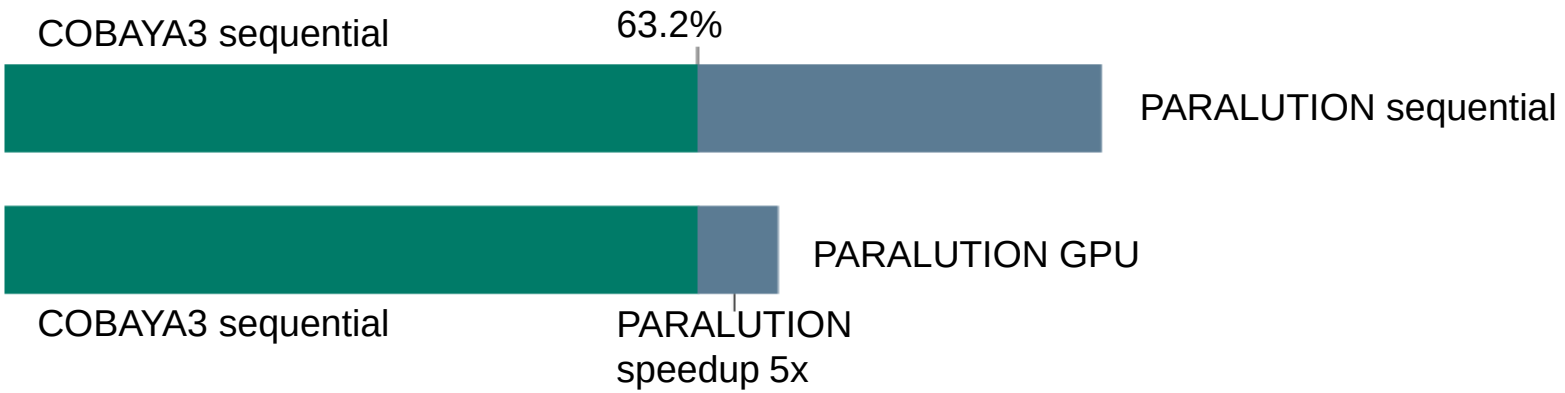


Performance results – MOX-3D-HFP.D



COBAYA3 – some profiling results

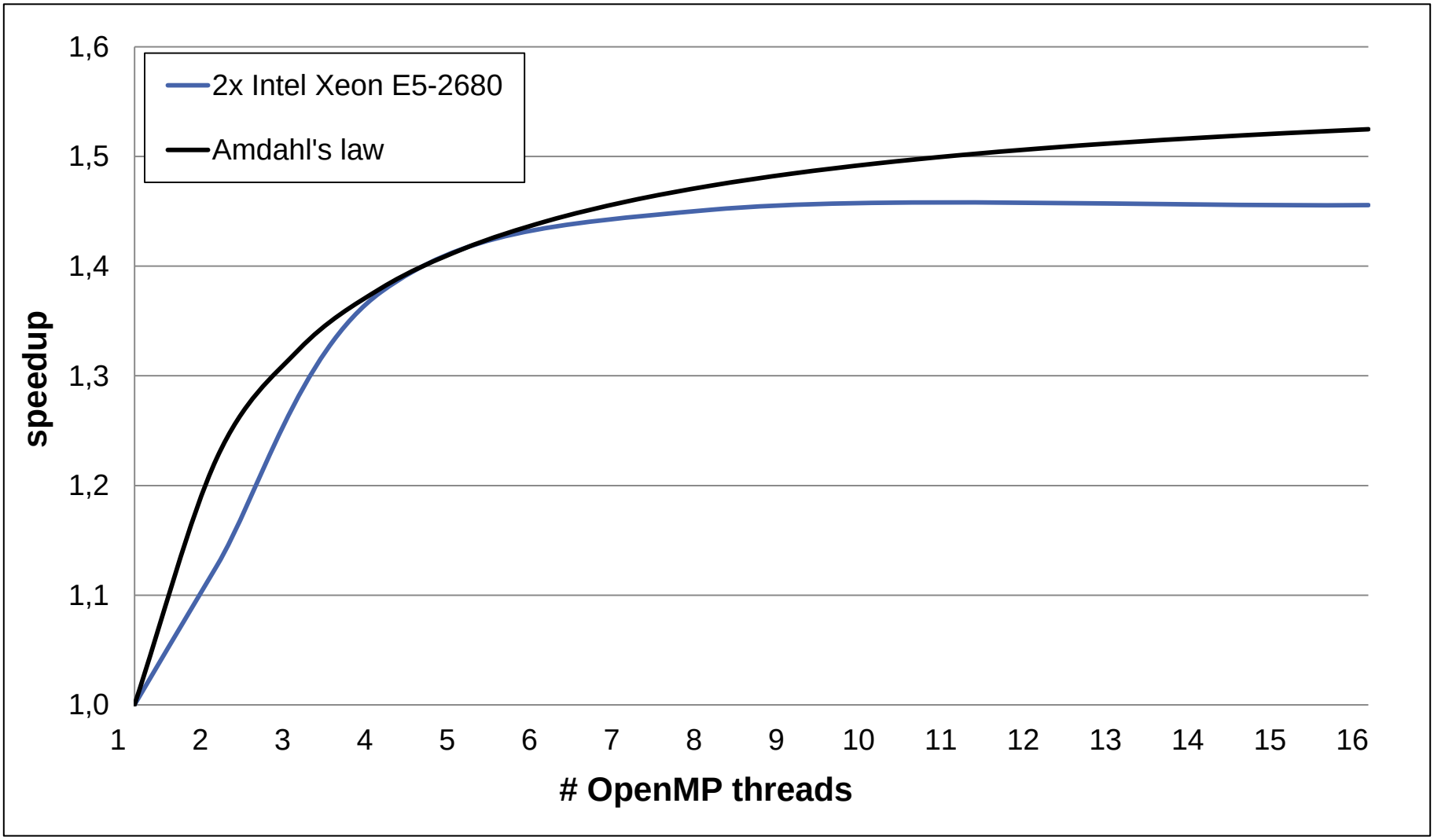
- Profiling results for COBAYA3 with ANDES solver and MOX case for a full simulation run:
  - Sequential program parts **63.2%**



Speedup by Amdahl‘s law:

$$S = \frac{1}{(1-P) + \frac{P}{N}} \leq \frac{1}{1-P} (N \to \infty) = \mathbf{1.582}\ (P = 1 - 0.632)$$

Performance results – MOX-3D-HFP.D
OpenMP scaling



Conclusion

- PARALUTION is a very modular library, well structured
- **Easy integration**
  - Variety of preconditioners and solvers that can easily be exchanged with the current implementation
- **High portability**
  - Possibility to move to GPUs or upcoming new architectures without modifying existing code, e.g. Intel Xeon Phi (MIC)
- Almost no time spent by Fortran ←→ C++ interaction
- Open source software

Outlook and future work

- ILU-T preconditioner GPU implementation
  - Not trivial to achieve fine-grained parallelism on accelerator devices, sparsity pattern is not known a-priori
- Approximate Inverse preconditioners
  - SPAI, FSAI
- Coarse level parallelization using MPI
  - Block Jacobi preconditioners (ILUT, …)
- Suitability of Algebraic Multigrid
  - Linear complexity!
- Suitability of PARALUTION for COBAYA3 pin by pin solver
- OpenMP implementation of the remaining sequential COBAYA3 program parts (63.2%)

Karlsruher Institut für Technologie

THANKS FOR YOUR ATTENTION