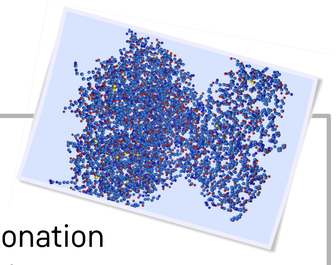
Make HPC Software Great Again

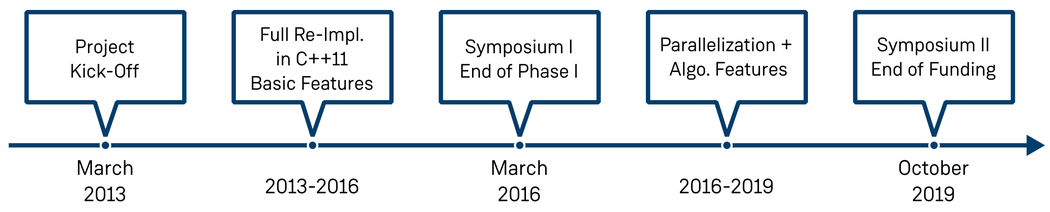
Developing A Fast Multipole Toolbox For Modern Applications On Modern Hardware

A. Beckmann, L. Morgenstern, I. Kabadshow | Jülich Supercomputing Centre, Research Centre Jülich, Germany
C. Kutzner, Th. Ullmann, B. Kohnke | Max Planck Institute for Biophysical Chemistry, Göttingen, Germany



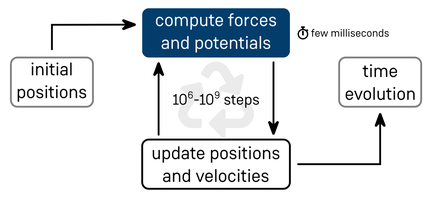
⓪ Scientific Background

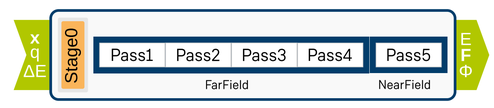
- **DFG-funded Project:** GROMEX (2013-2019)
  Unified long-range electrostatics and dynamic protonation for realistic biomolecular simulations on the Exascale
- **Application:** Molecular Dynamics Code GROMACS
- **Library:** Fast Multipole Method (FMM) with O(N) complexity



① Starting Point

- ✔ GROMACS is written in C++11
- ❗ GROMACS is tightly coupled with PME as long range solver
- ❗ No external libraries are used for long range interactions until now





- ❗ FMM available in highly optimized Fortran90 version only
- ❗ 150k+ lines of code, optimized for minimal FLOPs
- ❗ Supports only a few platforms via hand-written C-intrinsics
- ❗ Code duplications, non-generic codebase, no threading support

② Challenges

Hardware flexibility

- ILP, SIMD, OoOE
- Cache levels & sizes
- NUMA
- Threading
- Message passing

Algorithmic configurability

- Different implementations of compute kernels
- Different critical paths

Application customization

- Physical model
- Accuracy range
- System size

Lifecycle of hardware

- Changes every year

Lifecycle of staff

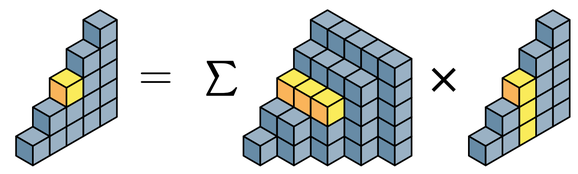
- PhD every 3-5 years
- Intern every 6-12 months

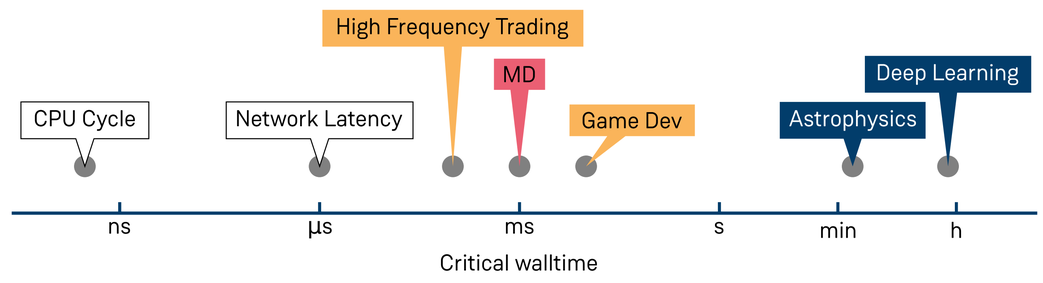
Lifecycle of code

- Minutes to lifetime

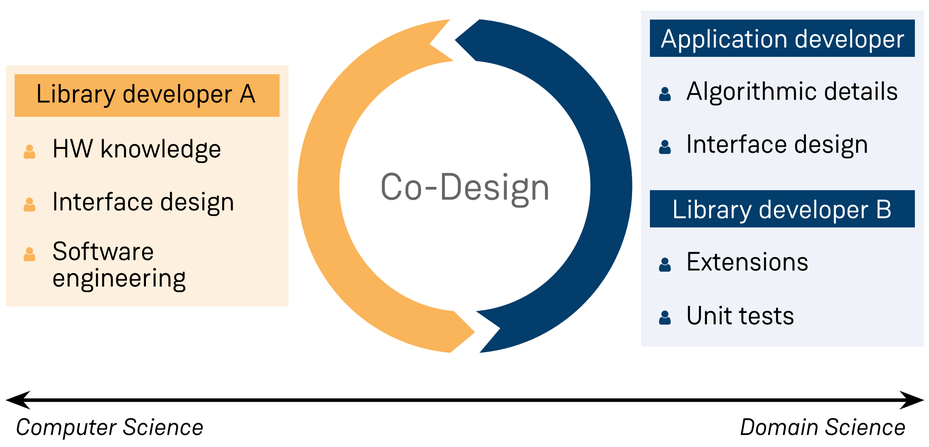
③ HPC≠HPC

- Non-trivial data structures
- Latency-critical walltime
- Non-uniform parallelism
- ❗ Parallelism on all levels required

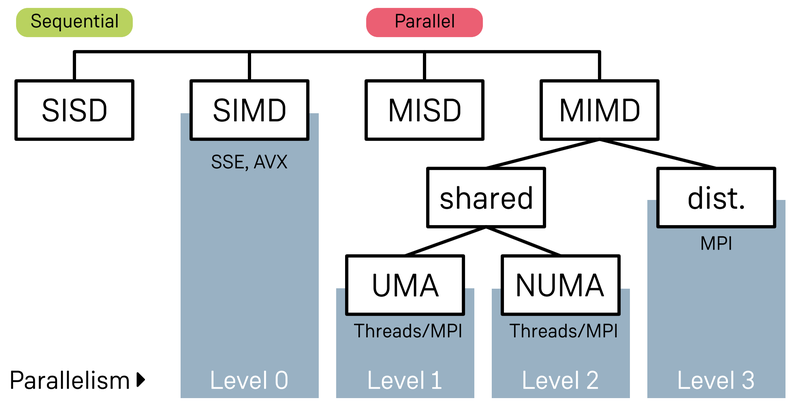




④ Co-Design Approach



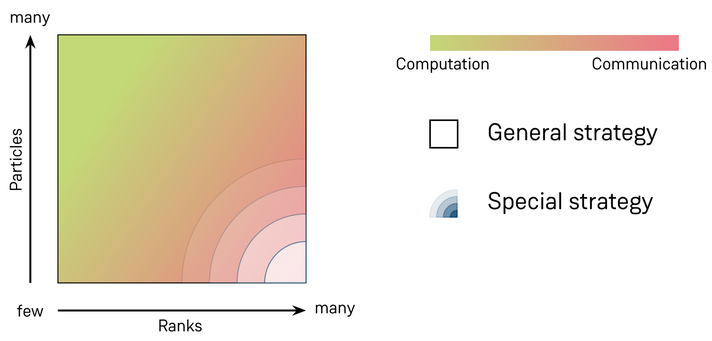
⑤ Modularity of Parallelism



- Decouple parallelization strategies from algorithmic workflow
- Different parallelization levels designed as independent sub-libraries, e.g. vectorization, tasking

⑥ Algorithm-Aware Specialization

- Parallelization strategy depends on the critical path of the algorithm
- Strategy must be interchangeable without 'large' changes to the user code
- Algorithm knowledge allows for better performance



⑦ Conclusion

- Mature programming language like C++11 mandatory
- Template metaprogramming helps to write readable and performant code
- Encapsulation of features helps maintainability
- Lines of code reduced to 25k+ due to deduplication
- No performance drawback with respect to C++ classes
- Zero-overhead abstraction layers via templates

- Domain scientist deals with domain science not the hardware
- Computer scientist deals with software design & hardware only
- Co-design cycle of interfaces defines overlap of domain and computer scientist

- Reuse of individual components possible through library design
- Independent projects can benefit by using such libraries