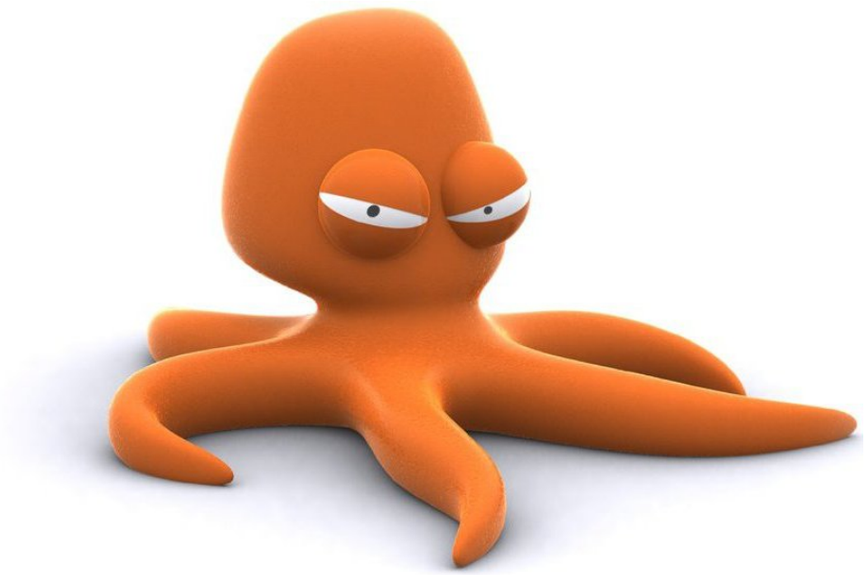
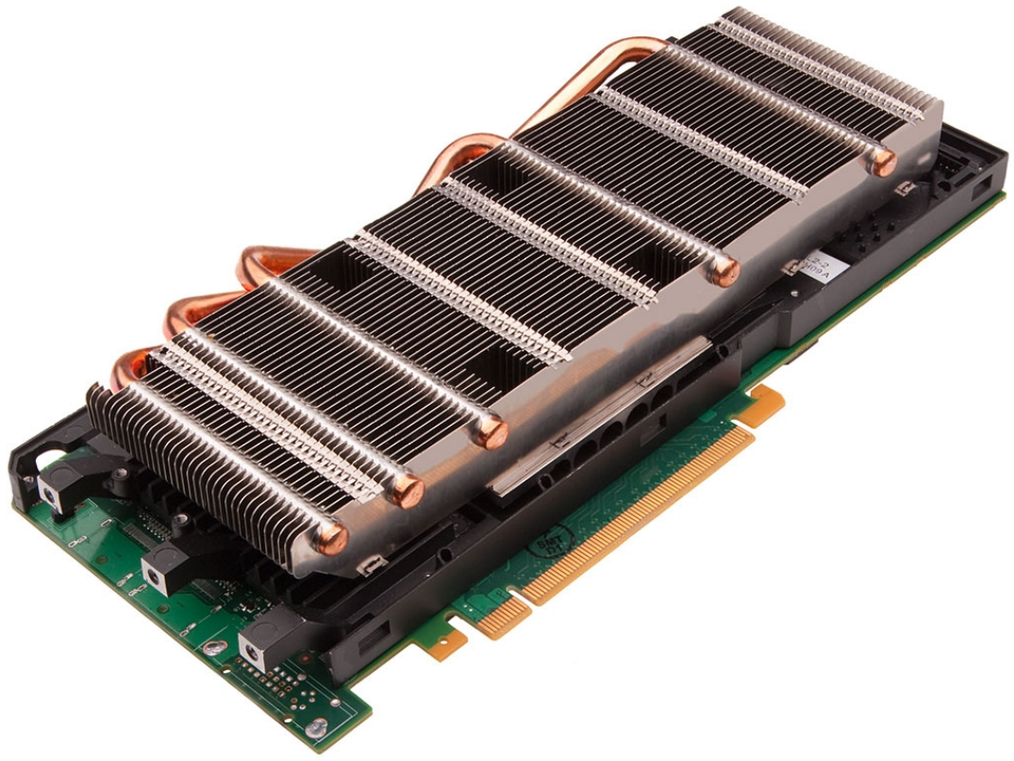


Octopus on graphical processing units

Xavier Andrade <xavier@tddft.org>
Department of Chemistry and Chemical Biology
Harvard University, United States

Introduction

- Processor development is limited by power consumption
- The future: specialized processors for each task
- CPUs are not focused on numerical calculations
- GPUs are processors developed for real-time 3D graphics
- How good are GPUs for electronic structure calculations?



Why GPUs?

An analogy:

CPU

GPU

Transport one briefcase

Transport a pile of boxes

GPUs are "slow"

	CPU Phenom II X4	GPU Radeon 5670	Ratio
Frequency	2800 MHz	750 MHz	4x
Int add latency	0.4 ns	10.7 ns	30x
FP add latency	1.4 ns	10.7 ns	7x
Memory latency	50 ns	300 ns	6x

GPUs are massively parallel

	CPU AMD FX 8150	GPU Radeon 7970	Ratio
Execution units	32	2048	64x
FP throughput	115 GFlops	947 GFlops	8x
Memory bandwidth	30 GB/s	264 GB/s	9x
Power consumption	125 Watts	230 Watts	2x

GPUs are good for numerical calculations

Large numbers of independent operations
Throughput over latency

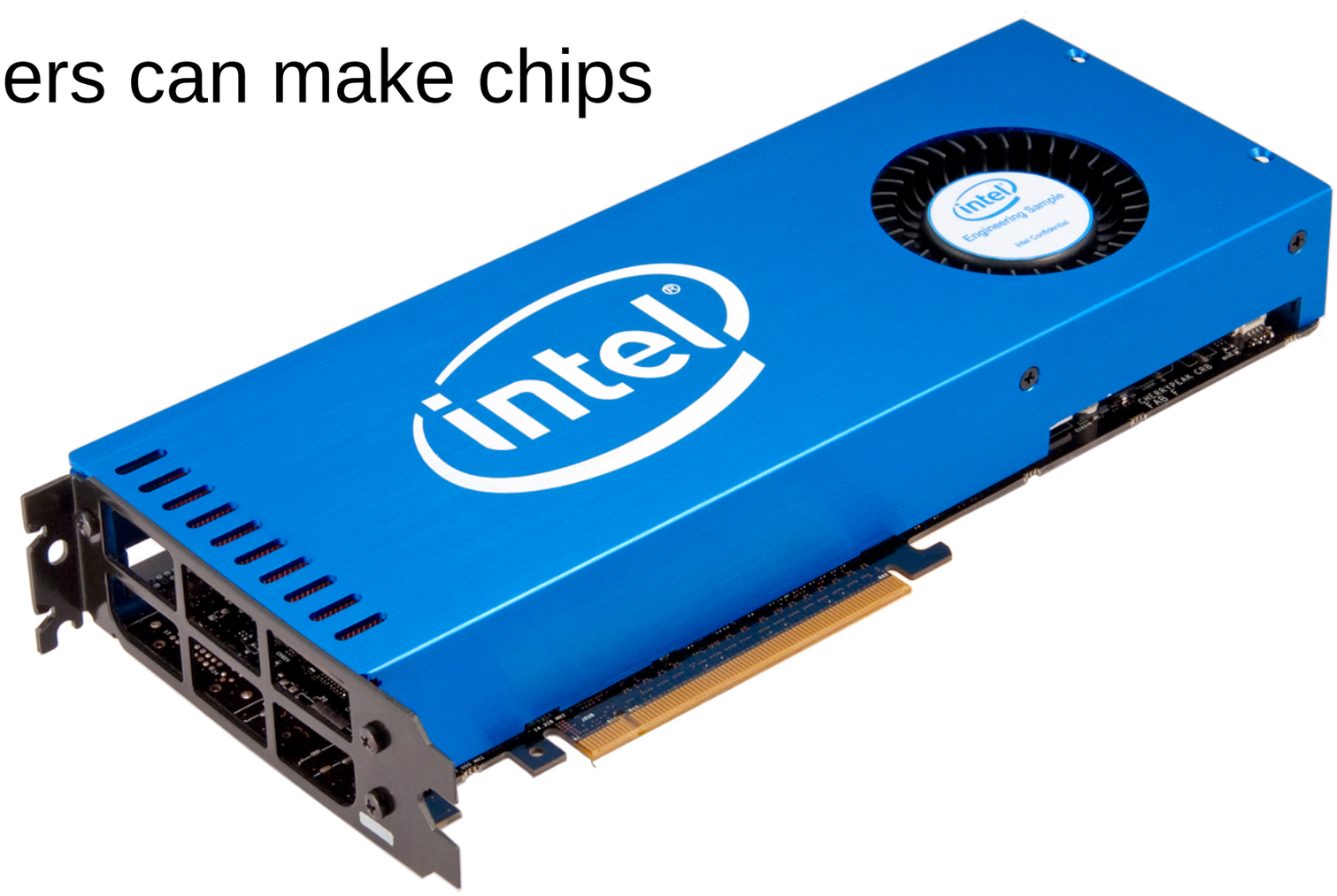
Myth:
A GPU code can be 100x-300x faster than a CPU one

GPU throughput is at most 20x the one of a CPU

Comparisons with unoptimized serial CPU code

True paradigm change: in-processor parallel programming

- Programmers can profit from parallelism
- Processor designers can make chips more parallel



GPUs are challenging for code developers

- Parallel programming is hard
- Expose parallelism to the routines
- Independent memory: copy data to the GPU
- Involves code planning
- GPUs are very sensitive to optimization

GPUs for electronic structure applications

- Less developed than other areas (for example classical molecular dynamics)
- Large amounts of data: requires adequate code planning for efficiency
- Specific routines rather than full code is executed on the GPU

Octopus GPU implementation

OpenCL

- Based on OpenCL
- Runs on AMD/ATI and NVidia GPUs
- OpenCL from Fortran

FortranCL*: an OpenCL interface for Fortran 90

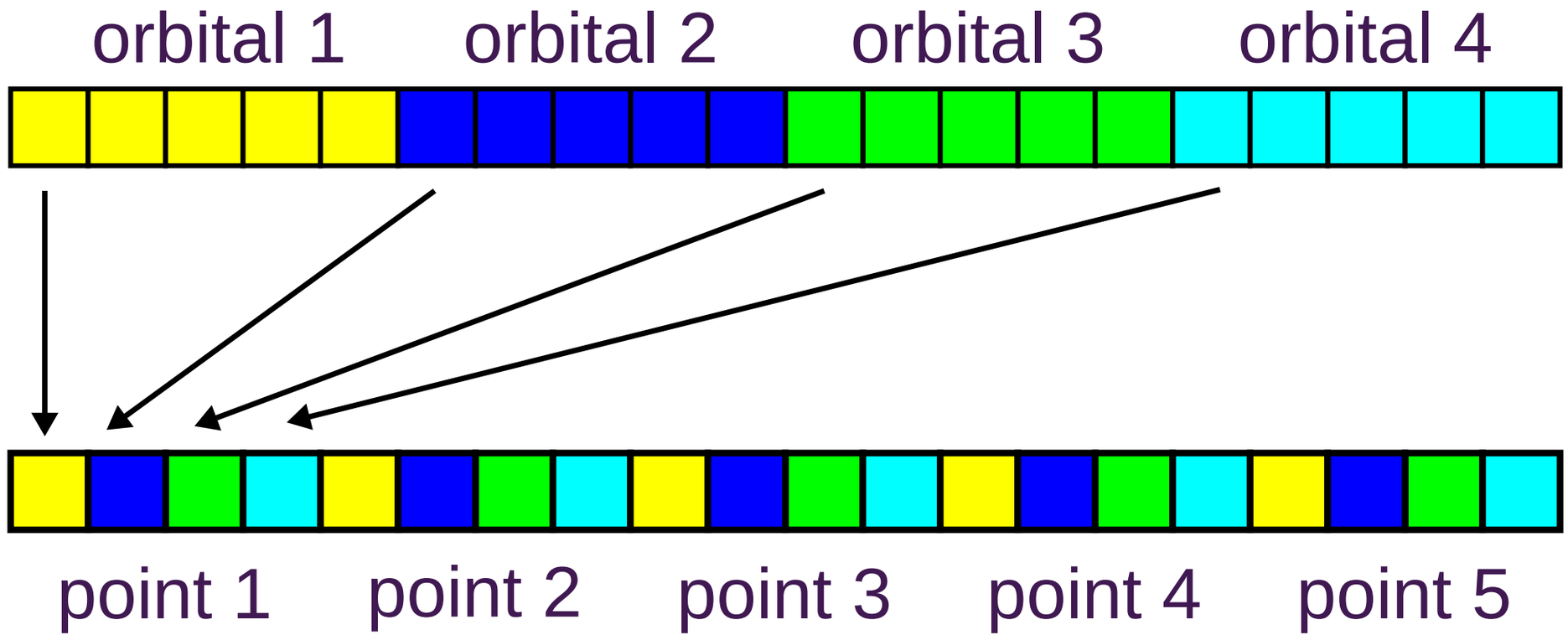
- Initially a part of Octopus, now a standalone library
- Close to the C OpenCL interface
  - Fortran 90 features: explicit intefaces, abstract types, polymorphism
  - LGPL license
  - Allows Fortran scientific codes to use OpenCL

*http://fortrancl.googlecode.com/

GPU strategy

- Blocks of Kohn-Sham orbitals as the basic data object
- Provide enough parallelism for the GPU
- Trade-off: large blocks are good for parallelism, bad for caching
- Block-size is controlled at execution time

For optimal memory access, store coefficients contiguously in memory

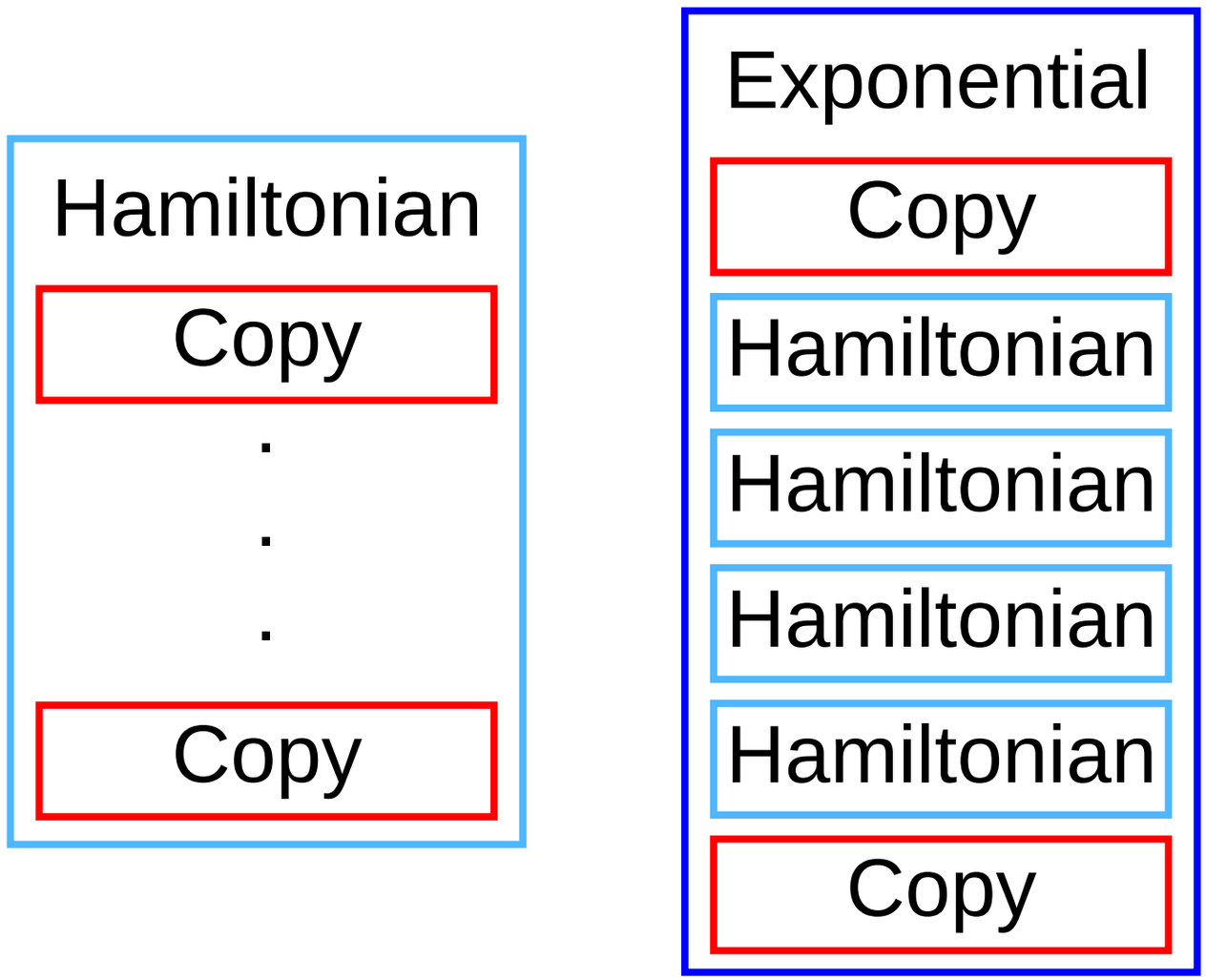


GPU memory management

- GPUs have their own banks of memory
- Limited in size, not enough for all orbitals
- Blocks are copied on demand to the GPU
- Copies to/from GPU memory are slow, should be avoided

Incremental approach for memory copies

- GPU-aware routines requests a block to be copied to GPU memory
- The copy could have been done by the parent routine
- Only specific parts of the code know about GPUs



Octopus components executed on the GPU

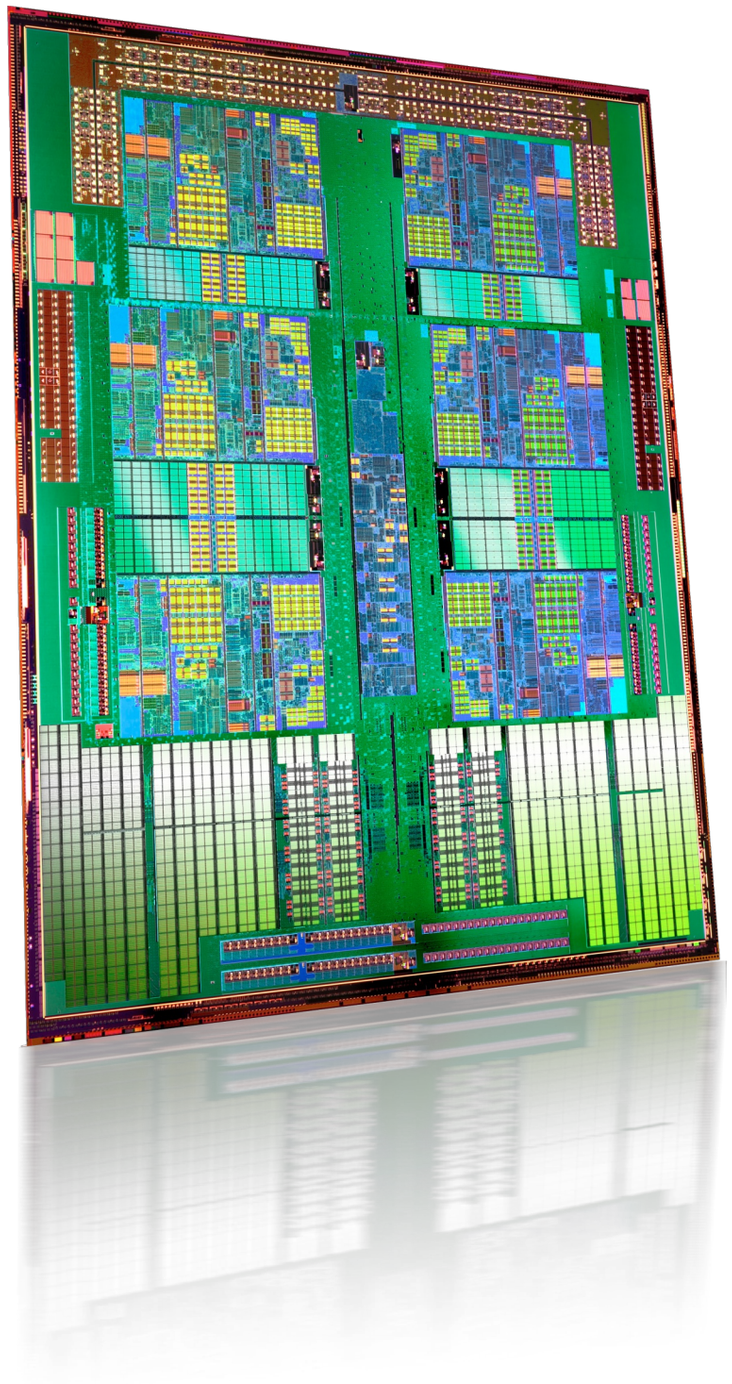
First phase: time-propagation

- Hamiltonian
- Exponential

Second phase: ground-state (work in progress)

- RMM-DIIS
- Orthogonalization and subspace diagonalization (BLAS)
- Poisson solver (FFT)

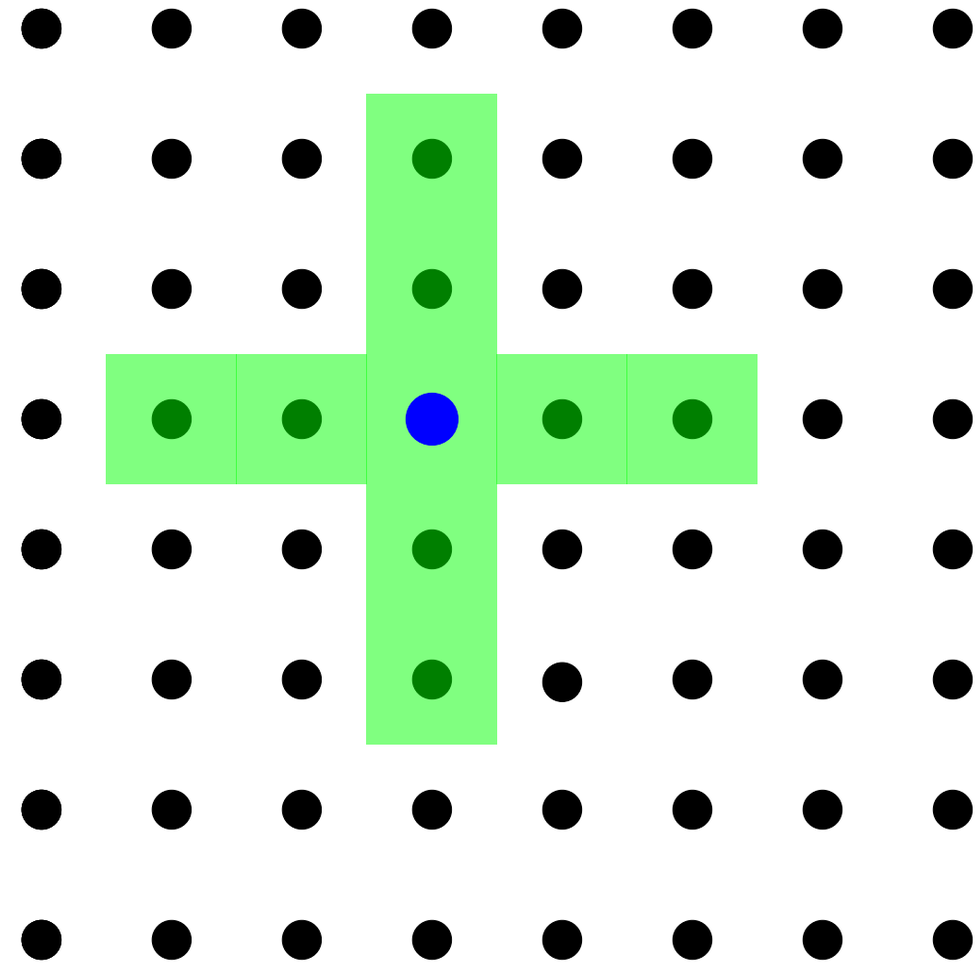
Many GPU optimization strategies are also valid for CPUs



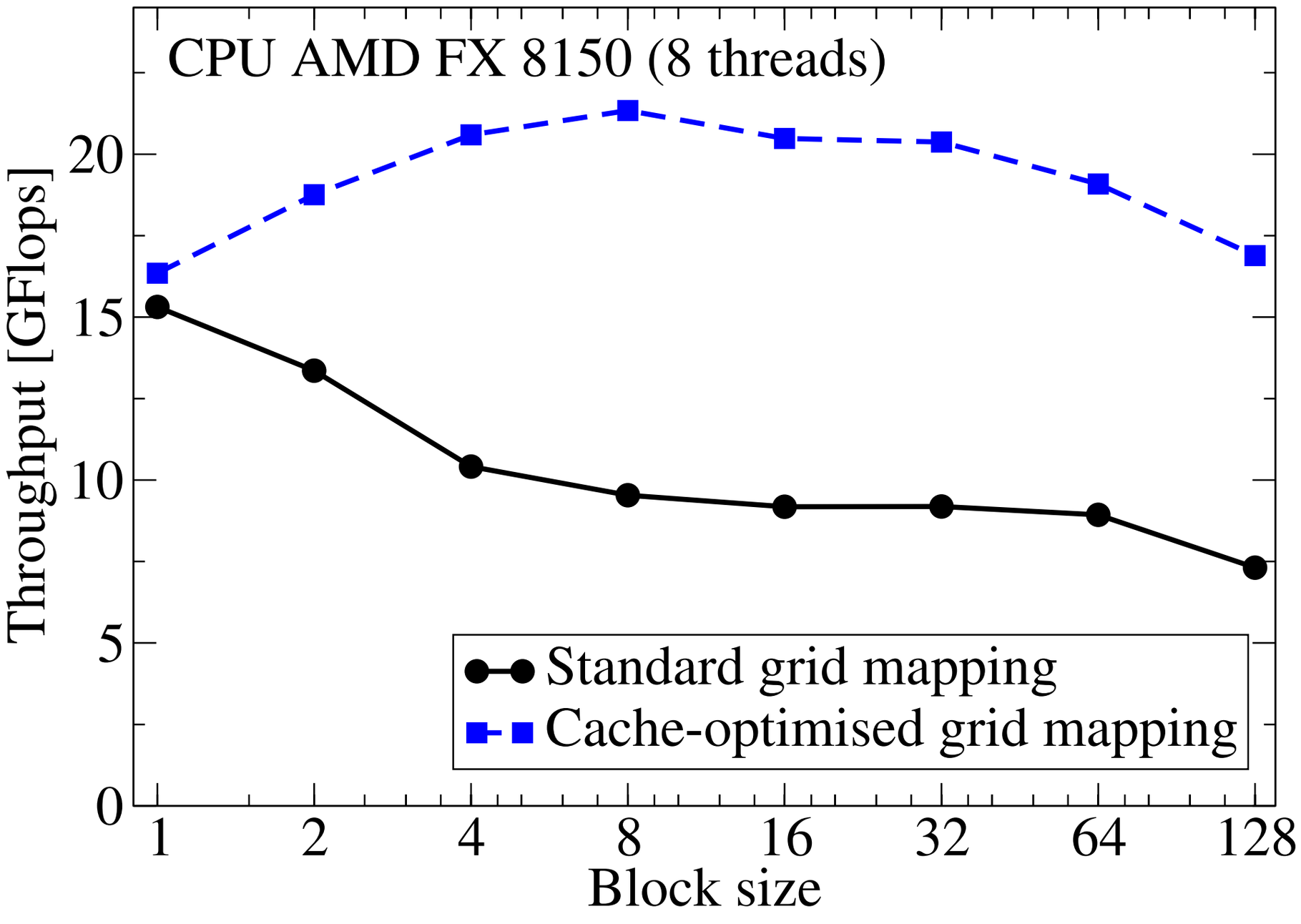
- Vectorial instructions: SSE2, AVX, Blue Gene
- Accessed through compiler directives
- OpenMP threads for multiple cores

The Laplacian operator

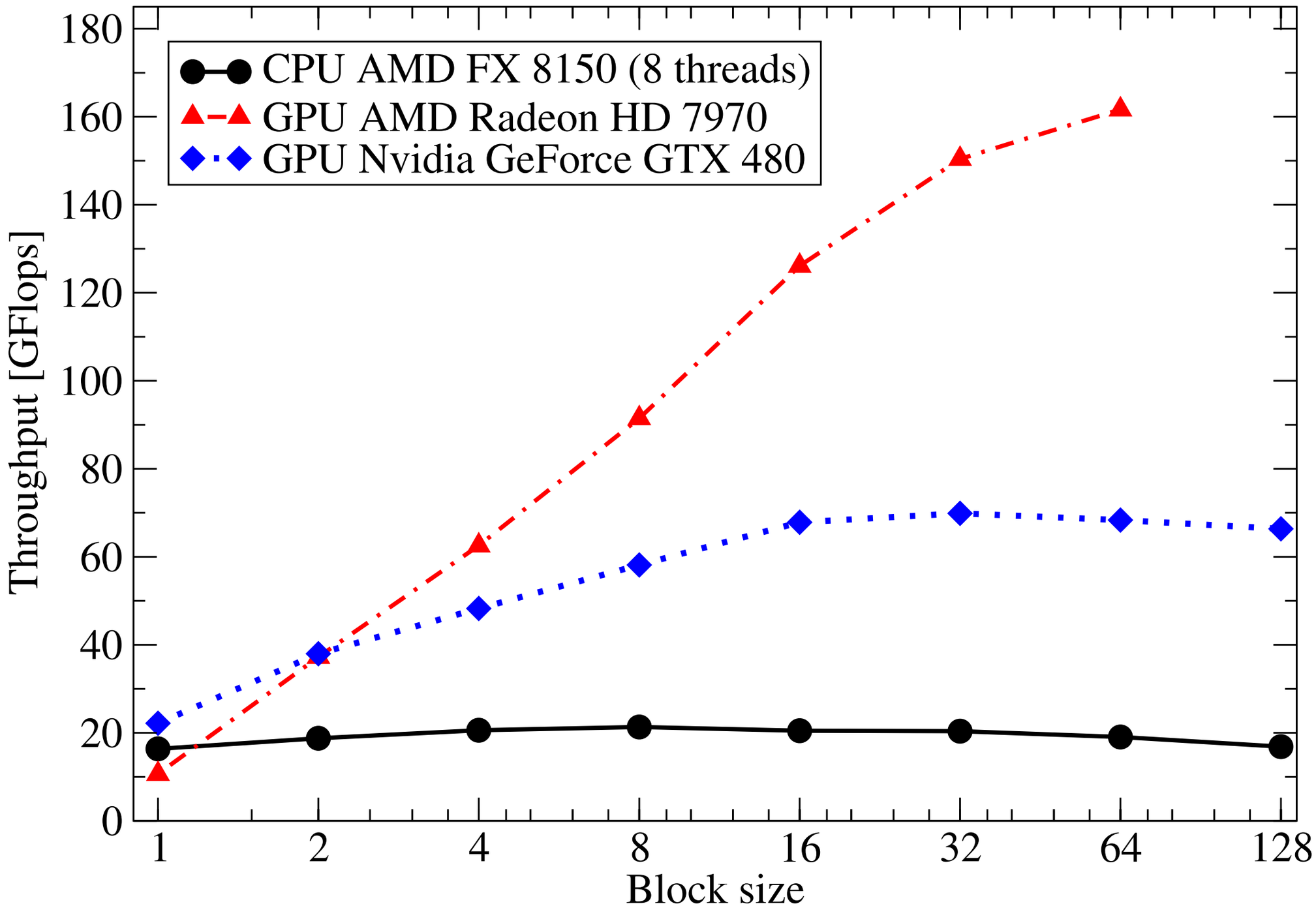
- Finite-difference calculation: sum over neighboring points
- Most time-consuming part of Octopus
- Spent considerable time optimizing it
- Arbitrary grid-shape: table of neighbors



Enumeration of the grid can be optimized for caching

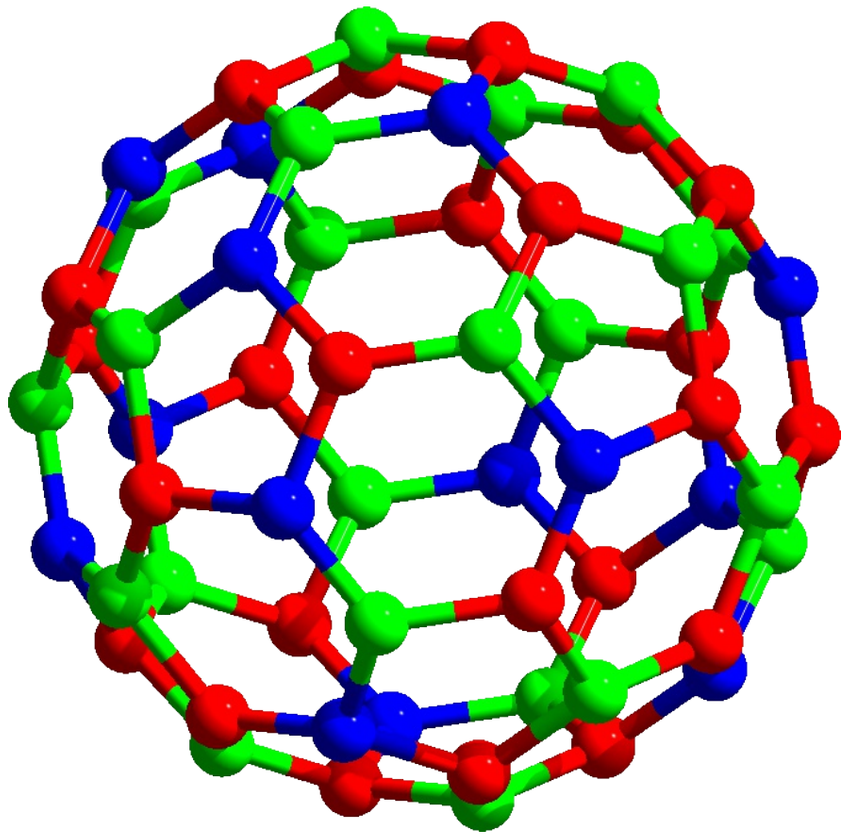


Laplacian operator performance



Pseudo-potential projectors

$$V_{\text{pseudo}} = \sum_{i}^{\text{atoms}} \sum_{lm} |p^i_{lm}\rangle\langle p^i_{lm}|$$

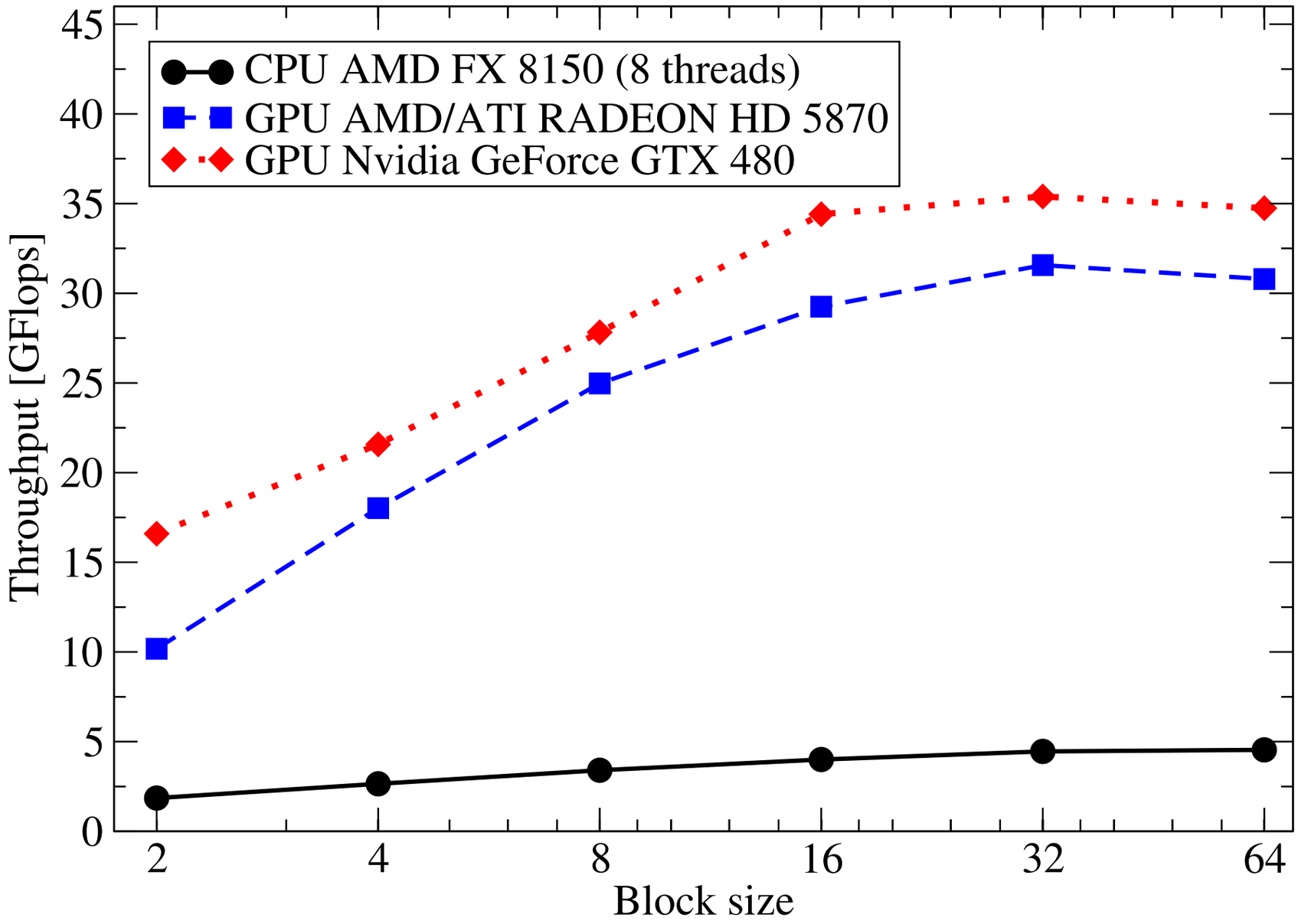


Two parts: bra and ket
Apply concurrently for all atoms
For the bra part, apply in parallel over atoms that do not overlap
Graph coloring

Other GPU operations

- Local potential
- Sum and copy of arrays
- Calculation of the electronic density

Performance of the propagation of the Kohn-Sham orbitals



Final speed-up of the GPU calculation (Fullerene molecule)

Processor	Time per propagation step [s]
CPU AMD FX 8150	9.45
GPU AMD Radeon 5870	1.86
GPU Nvidia GeForce GTX 480	1.72

Ground state calculations

- RMM-DIIS
- AMD APPML implementation of Blas
- Orthogonalization and subspace diagonalization
- Limitation: store all states in GPU memory
- Non-GPU accelerated routines become an issue

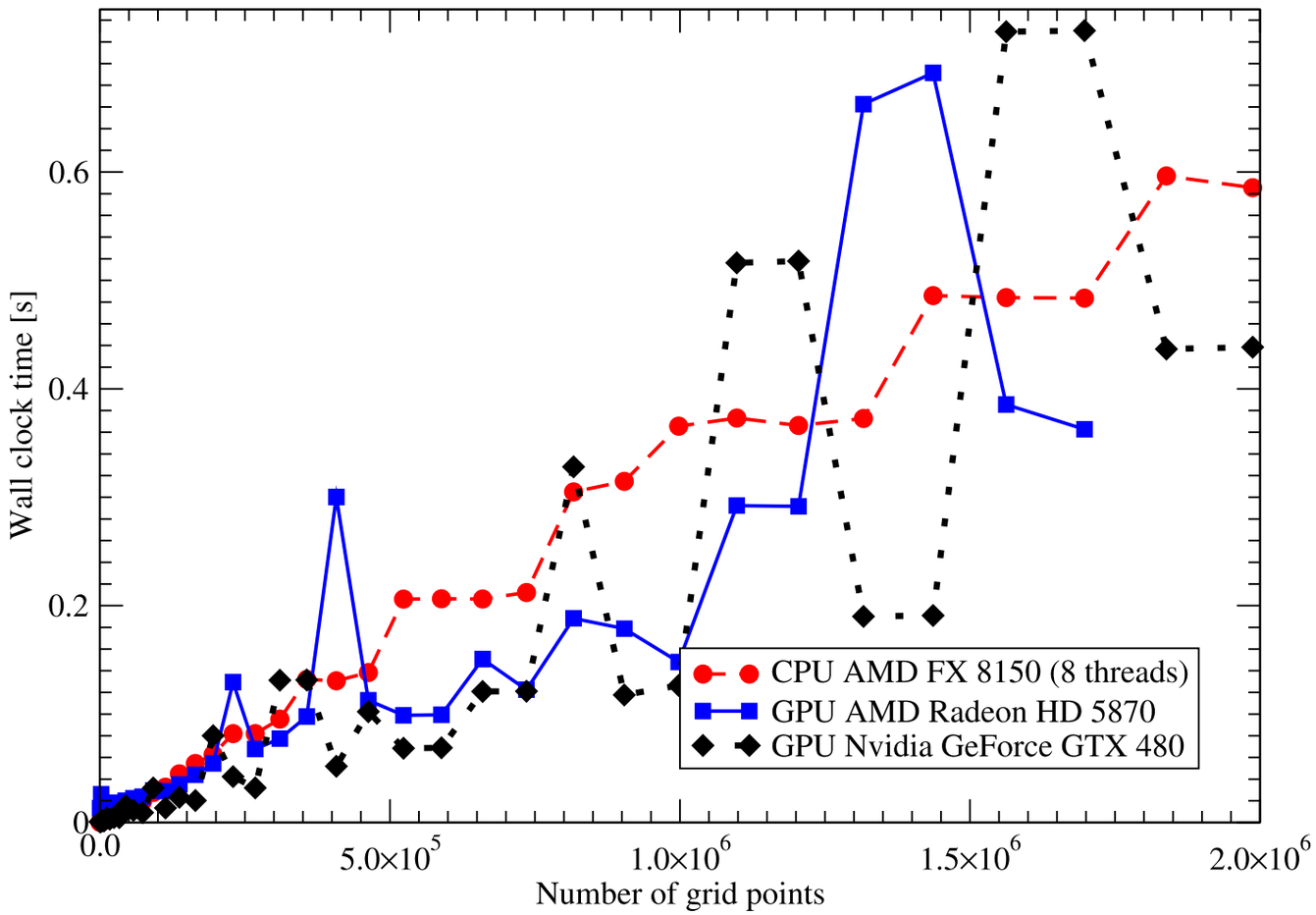
Comparison with Terachem

Gaussian-based DFT on GPUs

Number of atoms	Octopus Tesla M2090	Octopus Radeon 7970	Terachem Tesla C2050*
Valynomicin	168	358	388	299
Taxol	110	-	227	262

*Results for 8 GPUs normalized to one

Poisson solver on the GPU



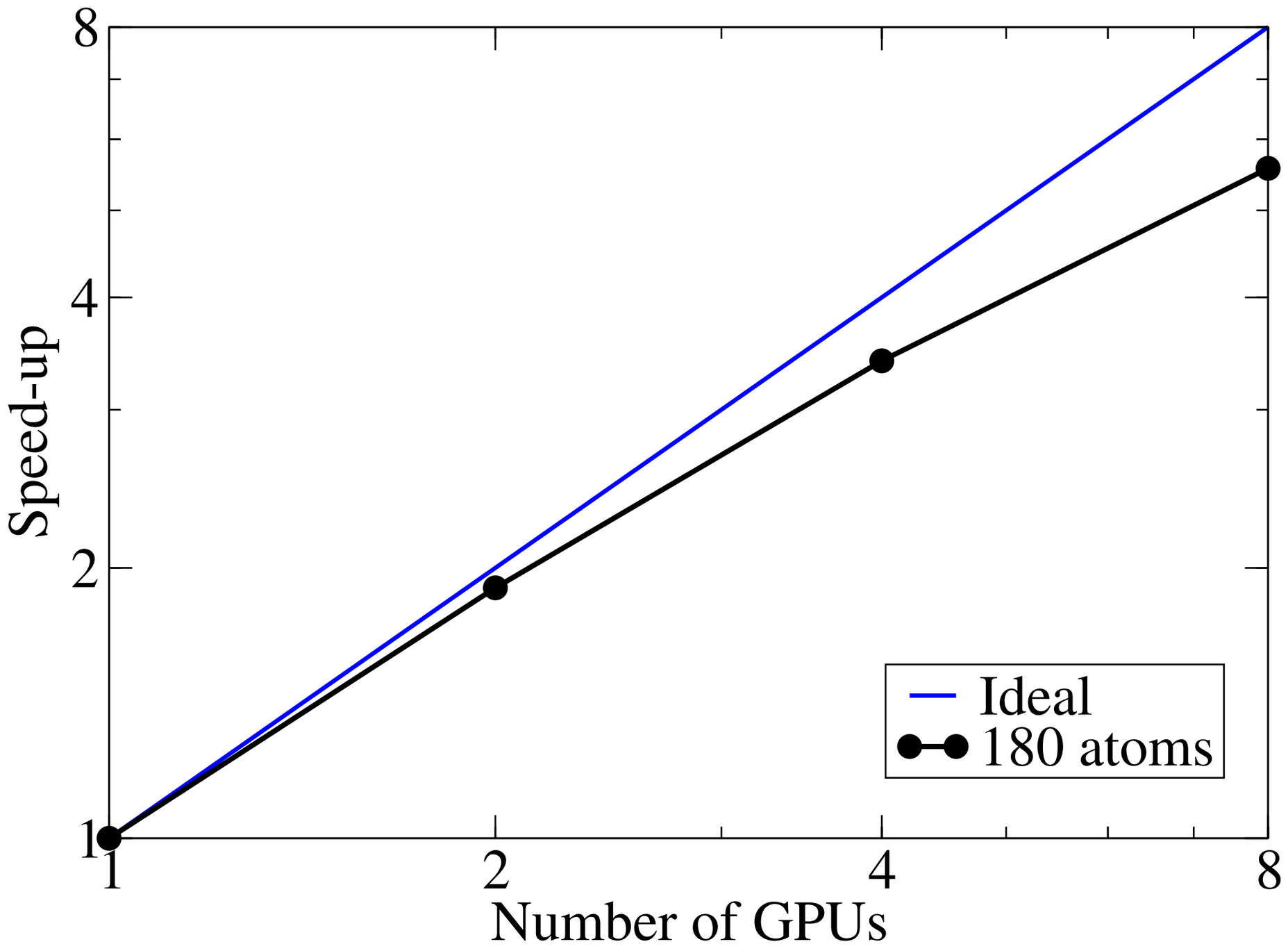
- AMD OpenCL FFT library
- Not a significant gain over CPU (multi-threaded FFTW)
- Data needs to be copied to the GPU
- Inmature library: complex transform, uneven performance
- Other options: FMM

Combined MPI+OpenCL parallelization

- GPU-based clusters and supercomputer
- Combined parallelization: MPI and OpenCL
- Copies between GPU memory are a 3-step process
- Communication costs increases relative to computation

Octopus MPI-GPU scaling

Nvidia Tesla M2090



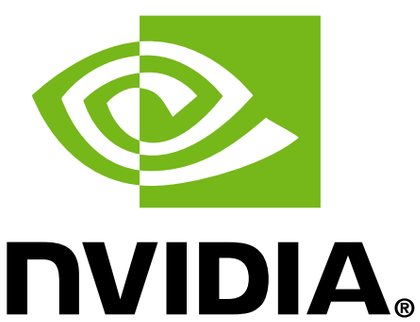
Current status of GPU support

- Useful for systems with ~100 atoms
- Limited support for non-usual features (correct but slow)
- Correct results
- Tutorial soon
- MPI works but not optimized

Conclusions

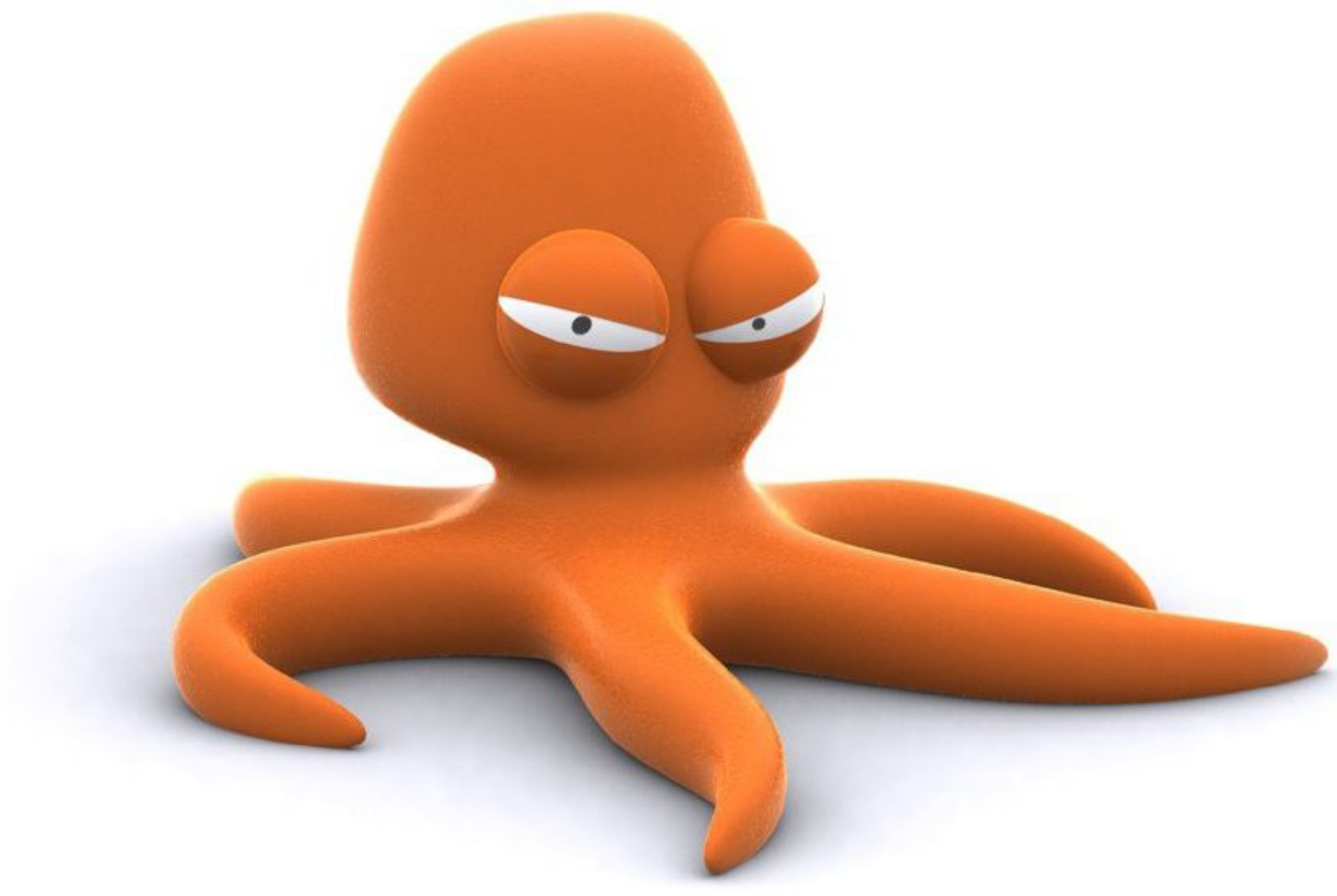
- Significant improvement in speed of electronic structure calculations
- Through careful code-planning, it is possible to include GPU support into an existing code
- MPI + GPU: Essential for current generation of supercomputers

Thanks:



Octopus

Open source real-space DFT and TDDFT code



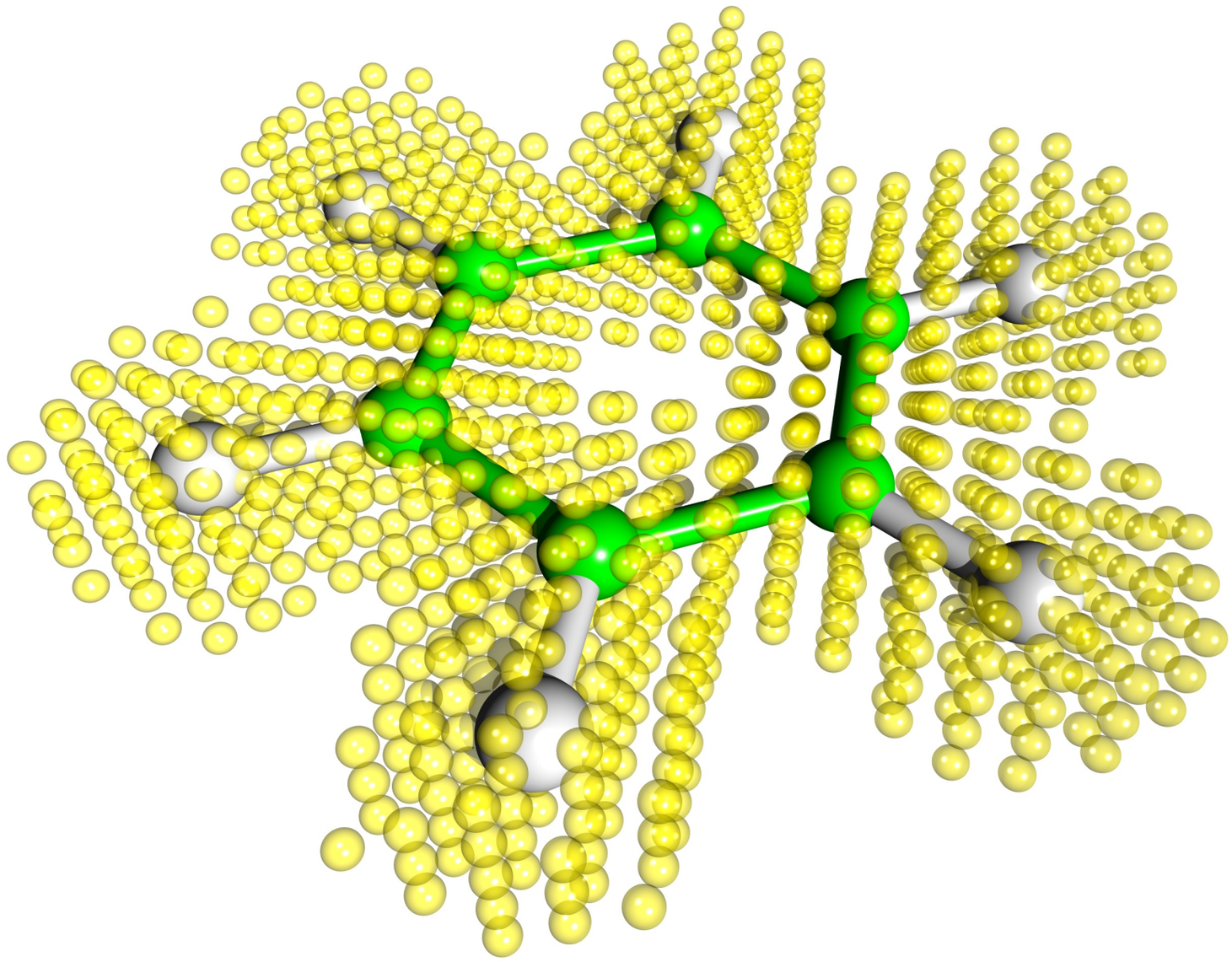
http://www.tddft.org/programs/octopus/

Octopus features

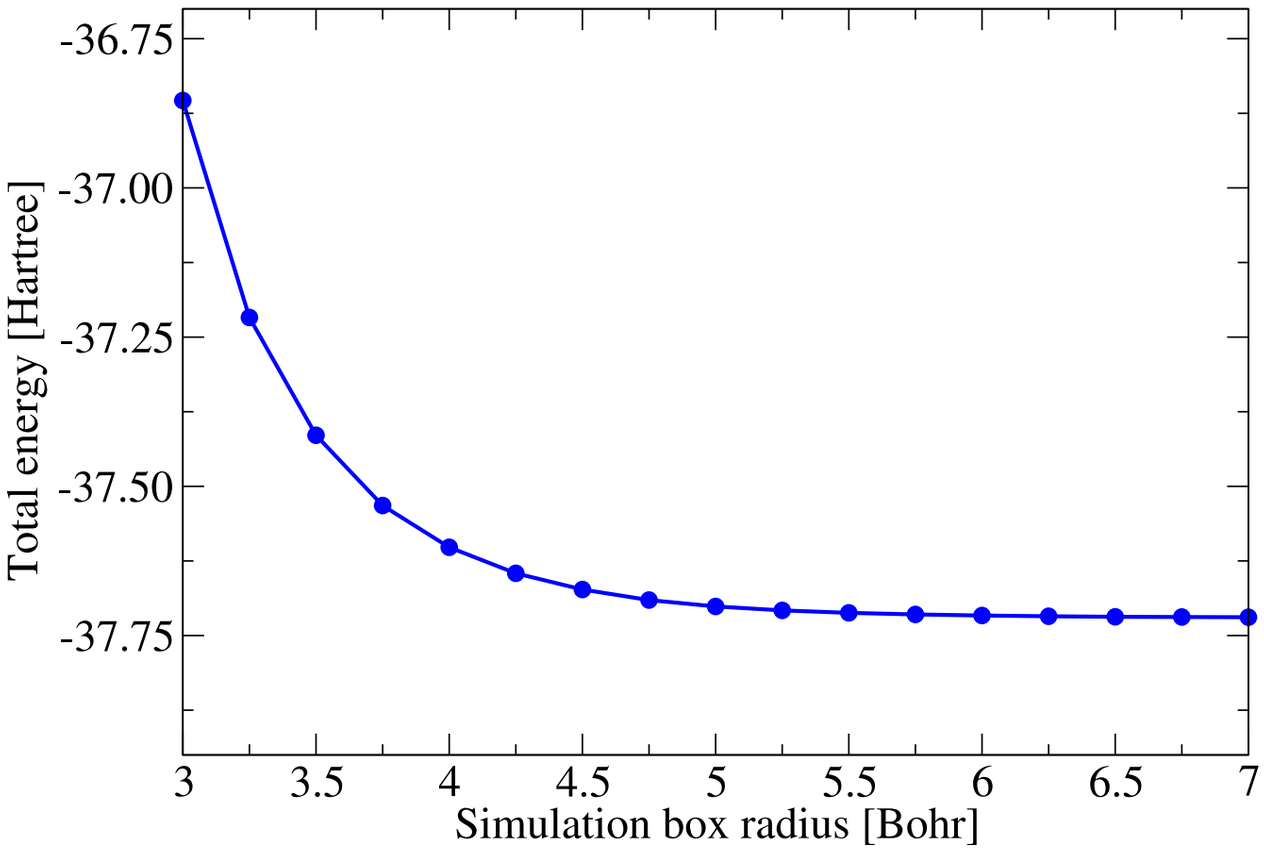
- LDA, GGA, mGGA, EXX/OEP and Hybrids
- Finite and periodic systems
- Norm-conserving pseudo-potentials (PAW in the future)
- Real-time electron dynamics
- Ground state, linear response, molecular dynamics
- Written in Fortran and some C

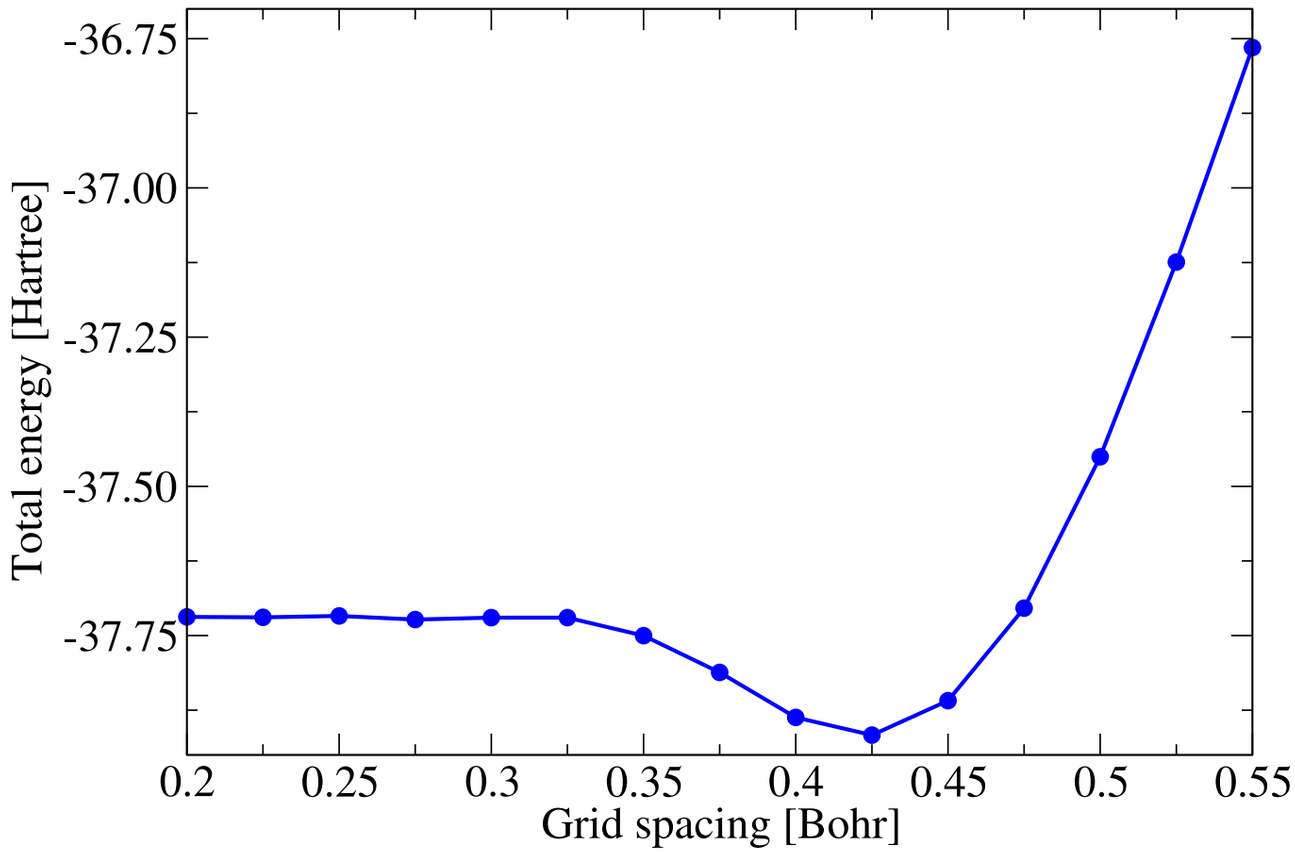
Real-space DFT

- The Kohn-Sham orbitals and the density are discretized over a grid of points
- Large number of coefficients (10k-10M)



Systematic and continuous control of the discretization error





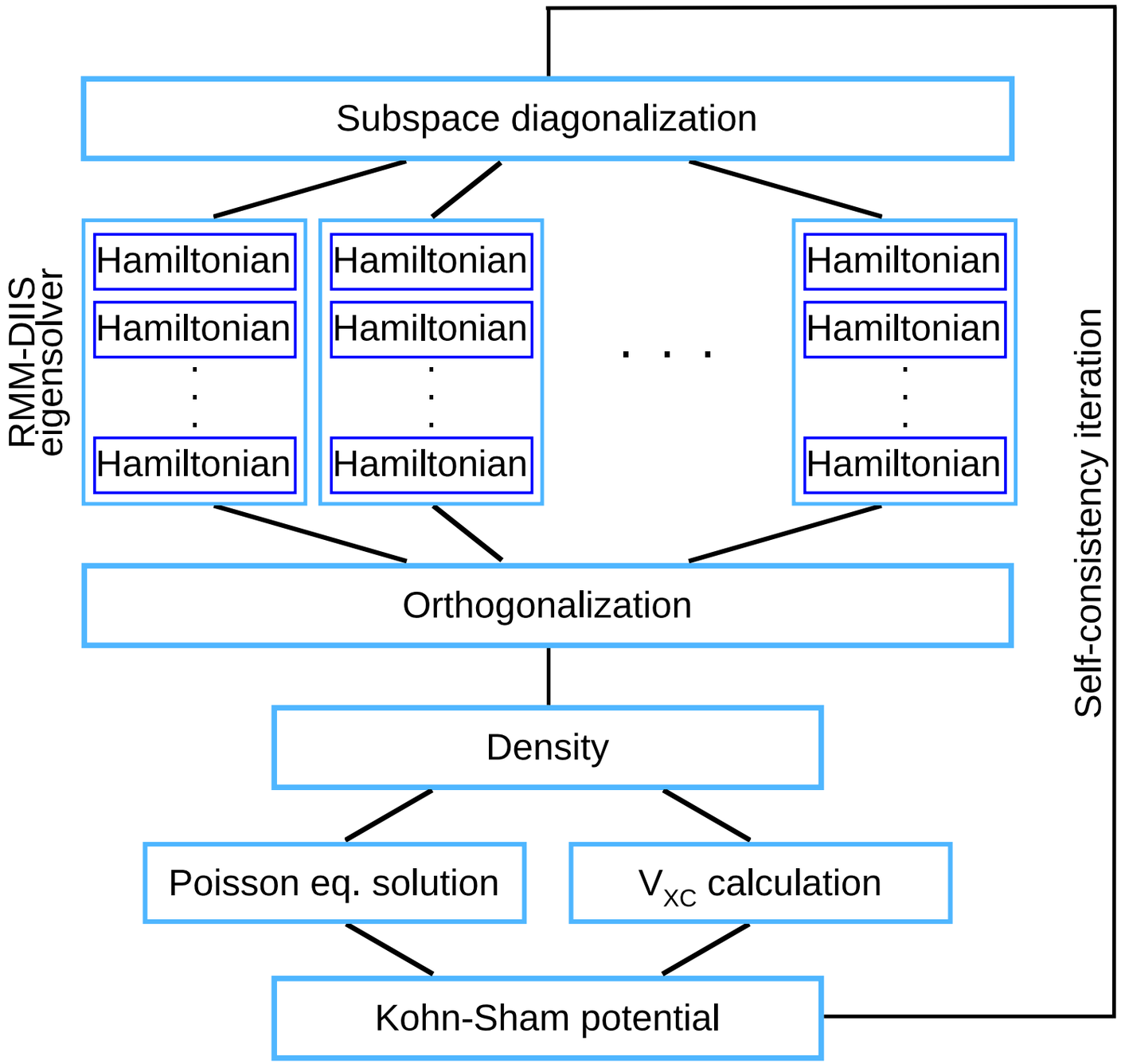
The Hamiltonian operator is sparse

- Number of components is linear with the number of grid coefficients
- Matrix is never built explicitly, it is only applied
- Kohn-Sham orbitals are the main object
- Kinetic energy operaton: finite-differences Laplacian
- Local potential term is diagonal
- Non-local pseudopotential: integration in spheres around each atom

Some properties of real-space grids

- Simple and flexible discretization
- High cost for small systems compared to atomic orbitals
- Not practical for two-point objects
- Exact exchange is expensive
- Egg-box effect: artificial energy dependence of the position of the atoms with respect to the grid

Ground state DFT calculation in real-space



Real-time DFT calculation in real-space