Overlapping communication and computation using the Intel MPI library's asynchronous progress control

Sebastian Ohlmann (Max Planck Computing and Data Facility, Garching)
Fabio Baruffa (Intel)
Markus Rampp (Max Planck Computing and Data Facility, Garching)

IXPUG Meeting, 13.-16.10.2020

Motivation

- Communication time can become bottleneck when scaling
- Overlap with computation: hide communication time
- Non-blocking MPI calls: communication **is not** progressing asynchronously
- Implementation in user code: possible, but lots of changes needed (e.g., calls to MPI_Test in computational loops)
- Promise of Intel MPI's progress control:
  - Progress handled internally
  - No change to user code needed

Intel MPI's asynchronous progress control

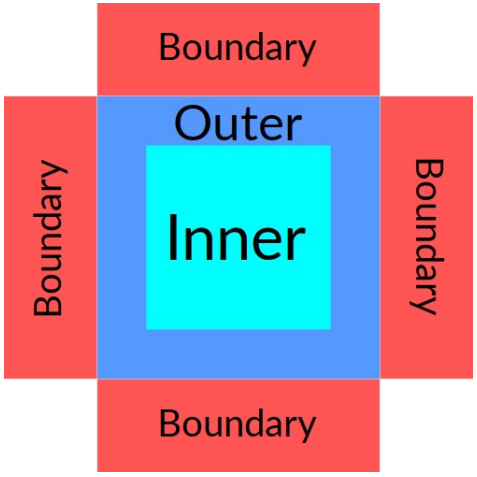
- Support for
  - Point-to-point operations
  - Blocking collectives
  - Non-blocking collectives (only Ibcast, Ireduce, Iallreduce)
- Only in release_mt and debug_mt variants
- Progress threads are spawned driving the communication
- Enable with I_MPI_ASYNC_PROGRESS=1
- Control pinning with I_MPI_ASYNC_PROGRESS_PIN
- Number of progress threads per MPI rank: I_MPI_ASYNC_PROGRESS_THREADS

Goals

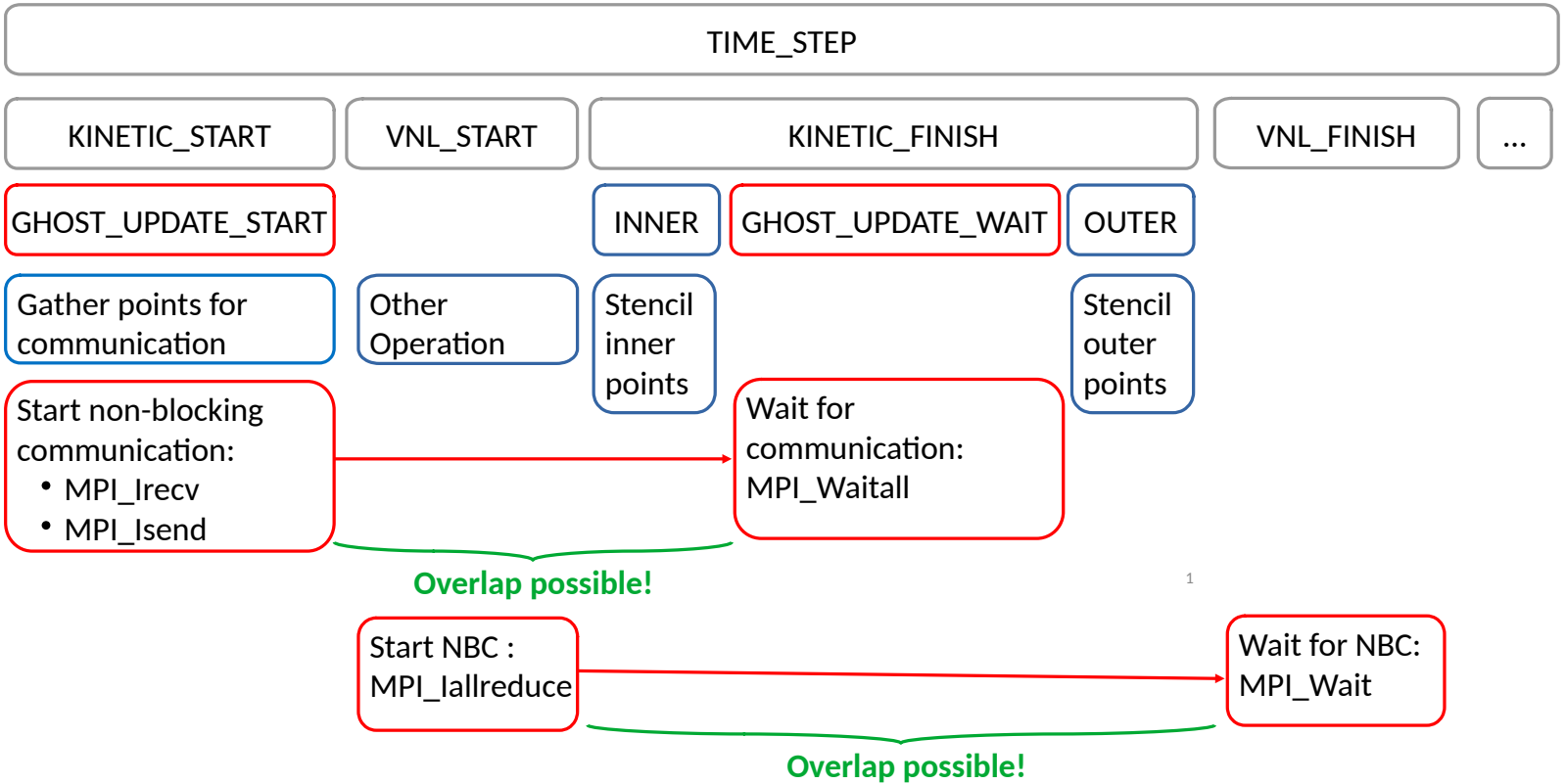
- Evaluate Intel MPI's asynchronous progress control for a real application
- Find best configuration
  - Fastest time to solution → compare full-node runs
  - Which combination MPI ranks & OpenMP threads fastest?
  - Spare cores for progress threads necessary?
  - Pinning necessary?
- Generalization to other codes

Application: Octopus

- Density functional theory code with pseudopotentials
- Real-space grid + finite differences
- Real-time time-dependent calculations
- Hybrid parallelization (MPI + OpenMP)
- Mainly Fortran, plus some C, plus some CUDA
- Open source: octopus-code.org
- Overlap of computation & communication:
  - Communication of boundary/ghost cells
  - Computation of inner part of stencil



Octopus: one time step



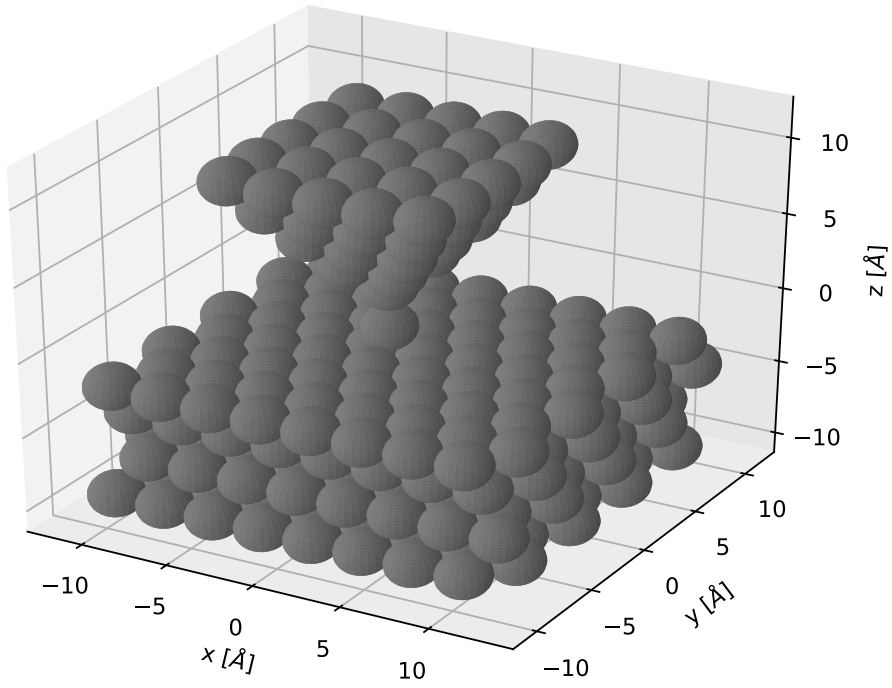
Benchmark clusters

MPCDF Max Planck Computing & Data Facility

- **Cobra** @ MPCDF
  - CPU: Intel Xeon 6148 Gold (Skylake)
  - 2x 20-core sockets/node = 40 cores/node
  - Interconnect: Omnipath (100 Gbit/s)
- **Raven** (interim system) @ MPCDF
  - CPU: Intel Xeon 9242 Platinum (Cascade lake AP)
  - 2x 48-core sockets/node = 96 cores/node
  - Interconnect: Infiniband HDR (100 Gbit/s)

Example system

- Silver tip over crystal
- Periodic in x and y
- 312 Ag atoms
- 3200 orbitals
- 2.4 M grid points

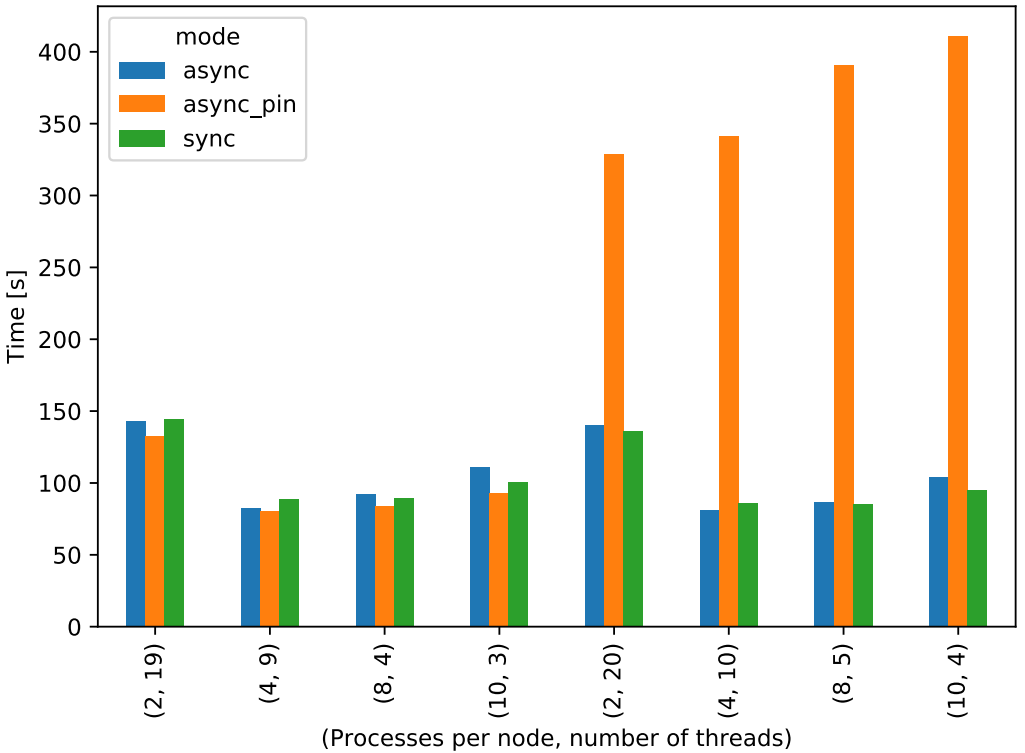


Results

Explanations

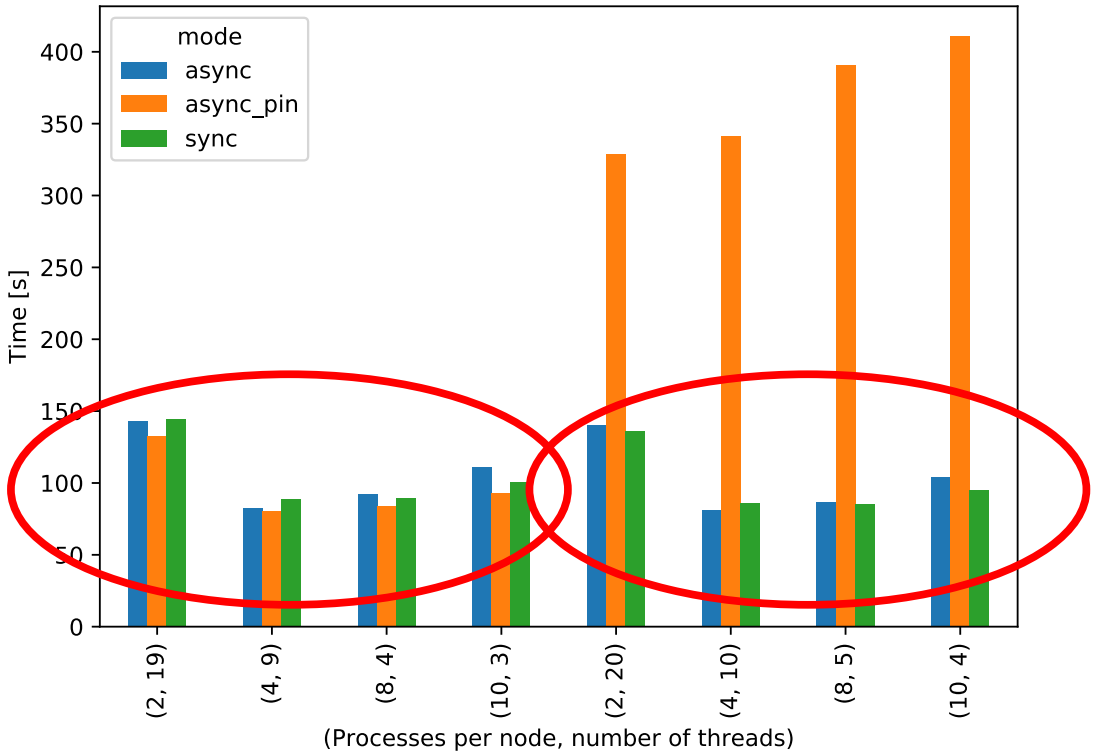
- 3 modes
  - sync: no asynchronous progress
  - async: asynchronous progress threads without pinning
  - async_pin: asynchronous progress threads with pinning
- Different combinations of MPI ranks x OpenMP threads for application
  - Full node: e.g. 1x40, 2x20, 4x10, 8x5, 10x4
  - Dedicated cores: e.g. 1x39, 2x19, 4x9, 8x4, 10x3
- Pinning using srun's cpu masks

Best combinations [cobra 40 cores/node]



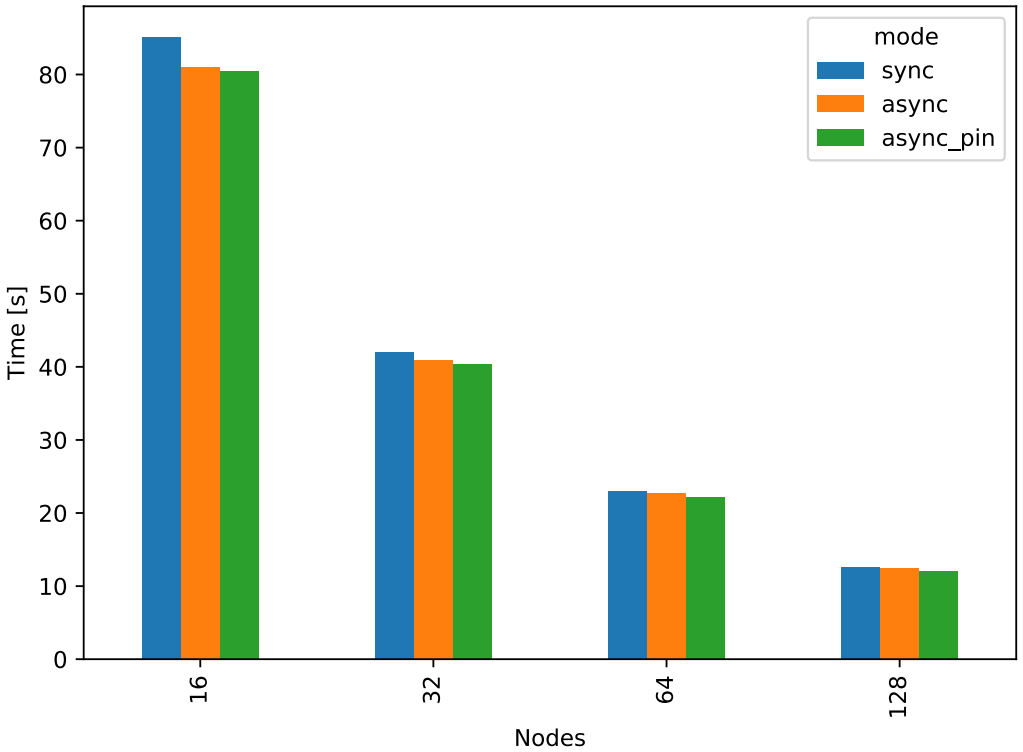
Best combinations [cobra 40 cores/node]

Dedicated cores
→ async_pin fastest



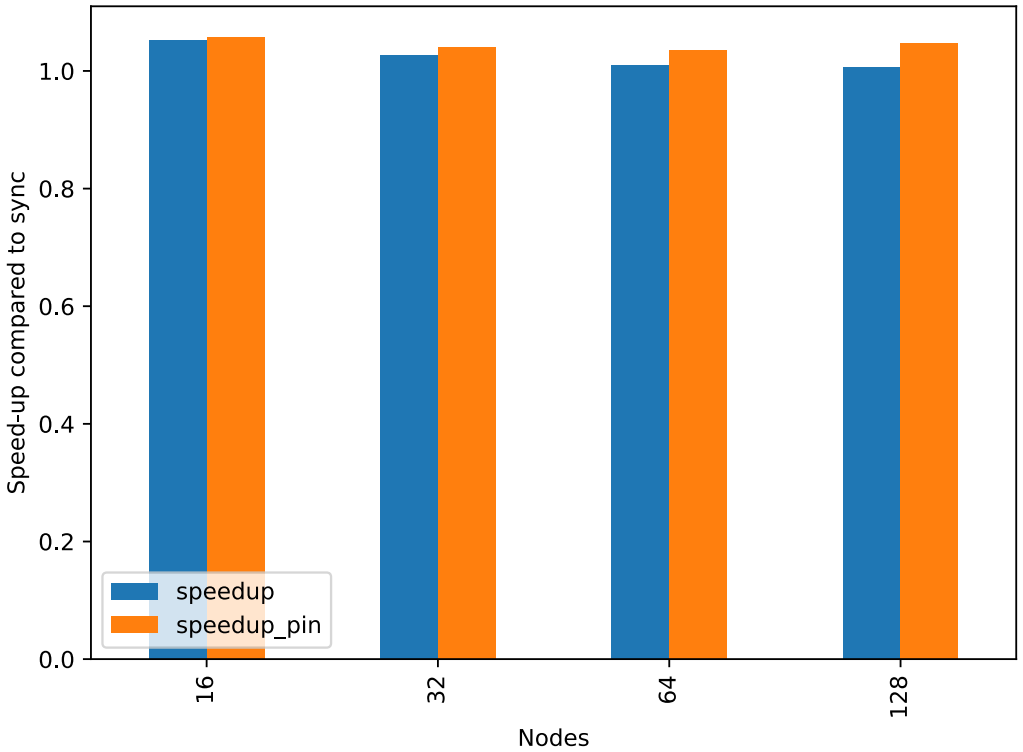
No dedicated cores
→ resource contention
if pinning enabled

Dedicated cores needed [cobra]



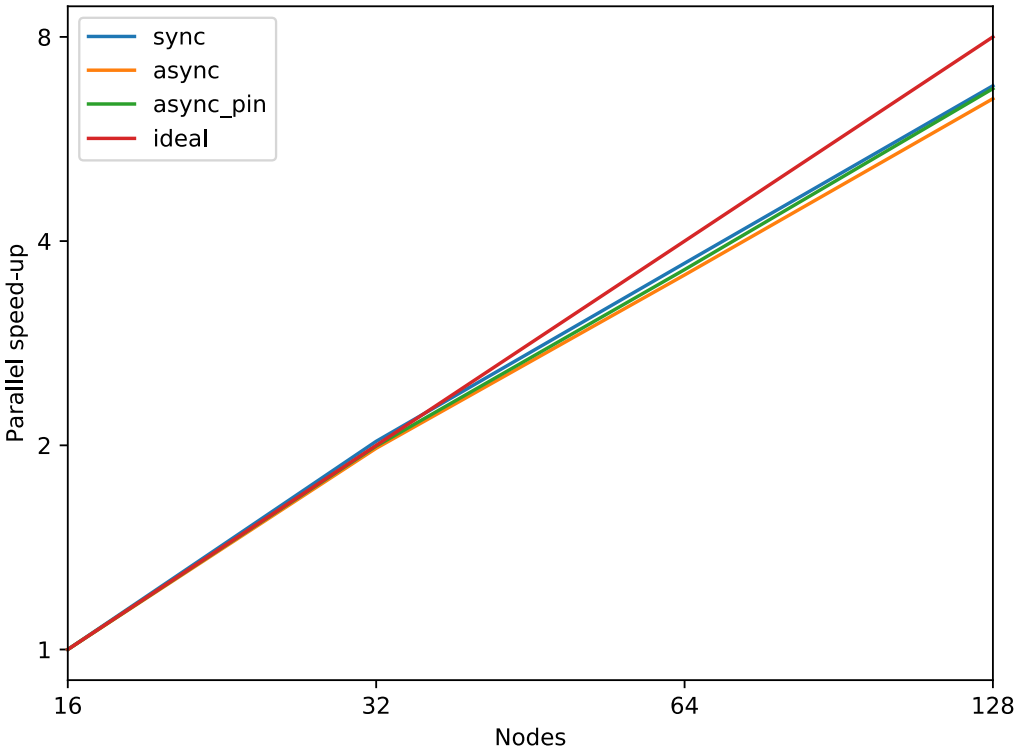
Fastest runs with pinning and dedicated cores!

Speed-up [cobra]



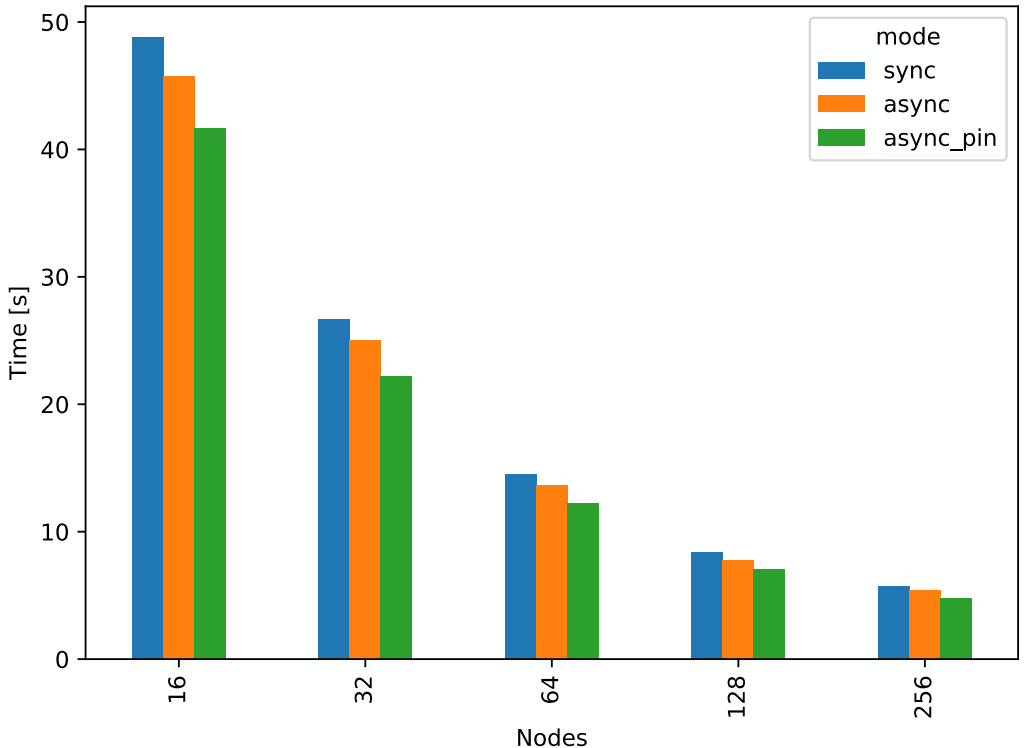
With pinning: speed-up of about 1.03x – 1.06x

Scaling of fastest runs [cobra]



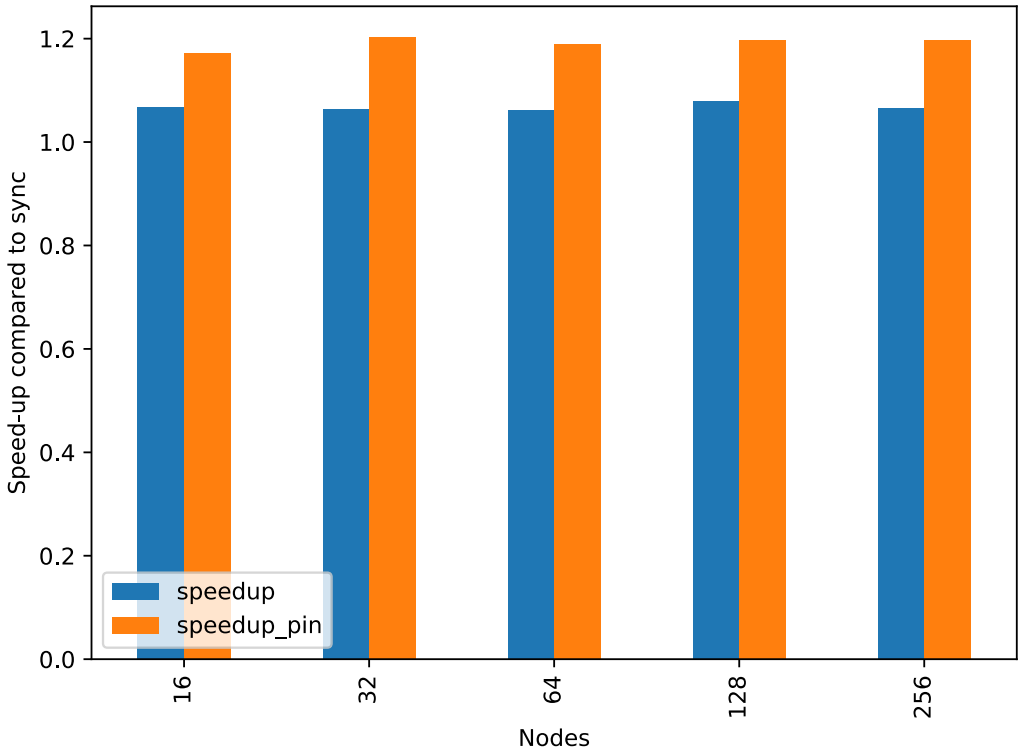
Good scaling: efficiency still 84% at 128 nodes (5120 cores)

Fastest runs [raven]



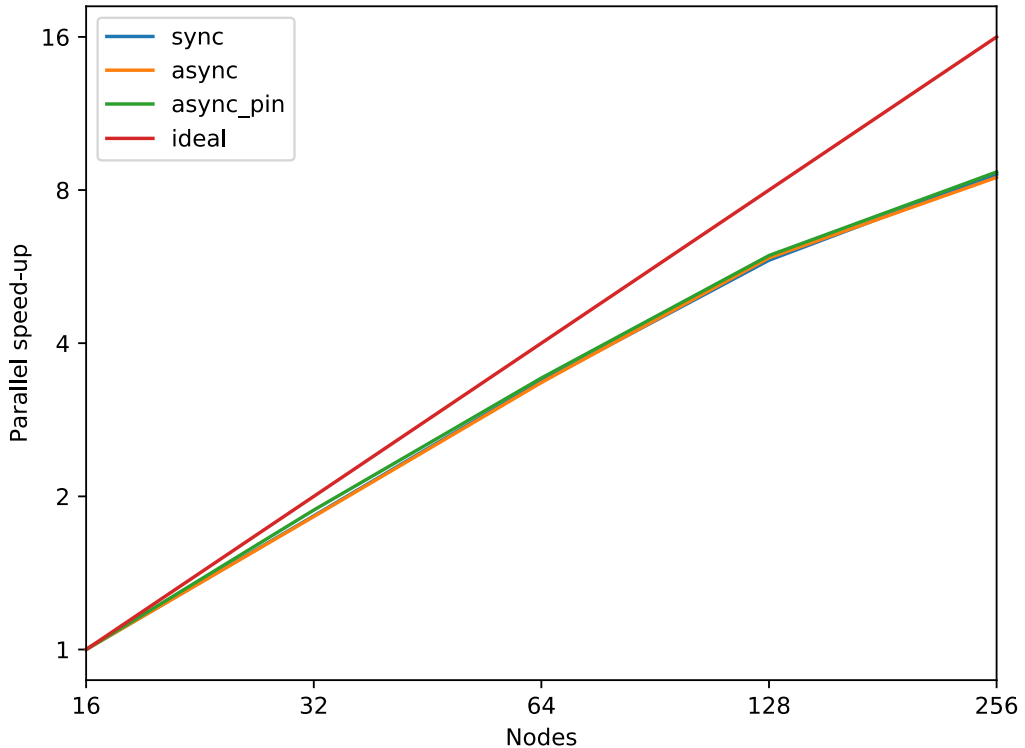
Fastest runs with pinning and dedicated cores!

Speed-up [raven]



With pinning: speed-up of about 1.17x – 1.20x

Scaling of fastest runs [raven]



Scaling efficiency:
74% at 128 nodes (12288 cores),
54% at 256 nodes (24576 cores)

Quantify overlap

- Difficult due to strong imbalance in communication volume (strange geometry)
- Compare average time in MPI functions
- On cobra, 16 nodes:
  - sync: 40.8 s of 89.7 s
  - async_pin: 33.1 s of 86.8 s

  → reduction of MPI time due to overlap

Generalization

- Stencil:
  - Split in inner & outer part needed
  - Benefit for other codes may be similar
  - Depends on stencil size (Octopus: 25 points)
- Hybrid codes probably benefit more
  - Less cores needed for progress threads
  - On cobra: on most node numbers, 8x4 best combination
    → 32 cores for compute, 8 for progress threads
  - Sacrificing a few cores probably ok for codes that are bound by memory bandwidth

Conclusions

- Intel MPI's asynchronous progress control allows overlap of computation & communication
- No change in user code needed for asynchronous progress
- Speed-up for octopus
  - Up to 1.05x on cobra (40-core Skylake nodes, Omnipath)
  - Up to 1.20x on raven (96-core Cascade Lake AP nodes, Infiniband)
- Lessons learned
  - Use release_mt/debug_mt
  - Dedicated cores needed
  - Do pinning right
  - Best potential for hybrid codes

Backup slides

Parallelization in Octopus

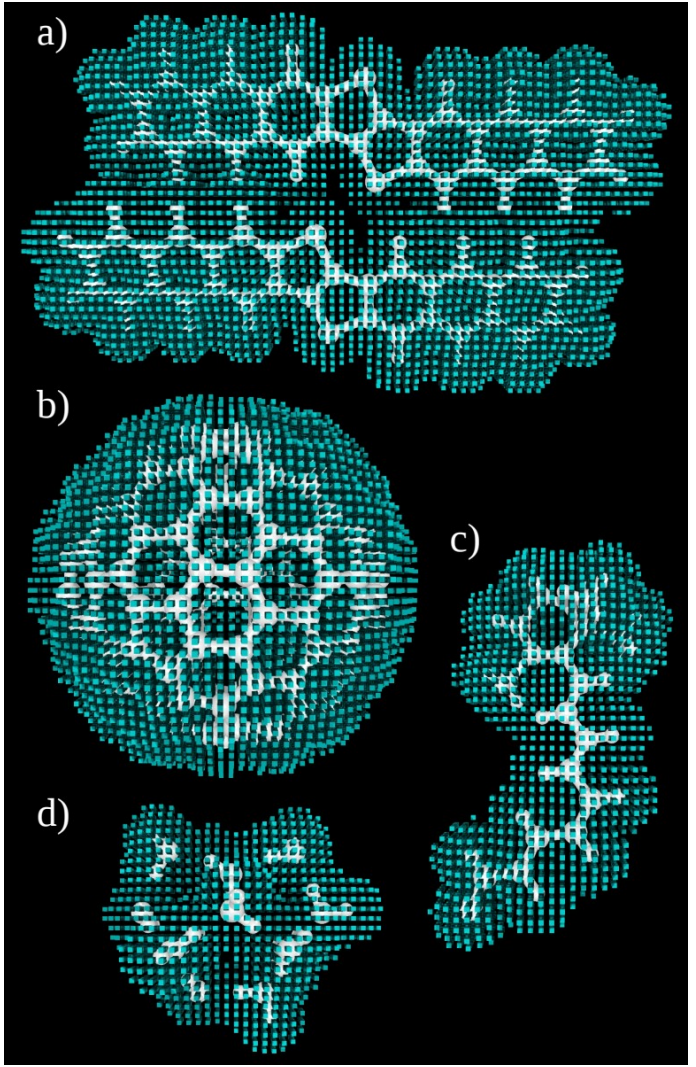
- Several dimensions
  - k points for periodic systems
  - States
  - Domain
- OpenMP parallelization also over domain loops
- Cobra: 2 nodes for domain (MPI + OpenMP)
  → 80 cores
- Raven: 1 node for domain (MPI + OpenMP)
  → 96 cores

Pinning: details

- Pinning needed for best results
  - Pin MPI rank + OpenMP threads next to progress thread
- Example: 4 MPI ranks, 9 OpenMP threads
  - Pin first rank to cores 0-8, second to 10-18, third to 20-28, fourth to 30-38
  - srun --cpu-bind=mask_cpu:0x1ff,0x7fc00,0x1ff00000,0x7fc0000000
  - Pin progress threads to dedicated cores 9, 19, 29, 39
  - export I_MPI_ASYNC_PROGRESS_PIN=9,19,29,39
- Slurm CPU mask:
  - Hexadecimal number, binary representation → enabled cores
  - 0x7fc00 = 0b11111111110000000000 (cores 10 to 18)

Data layout

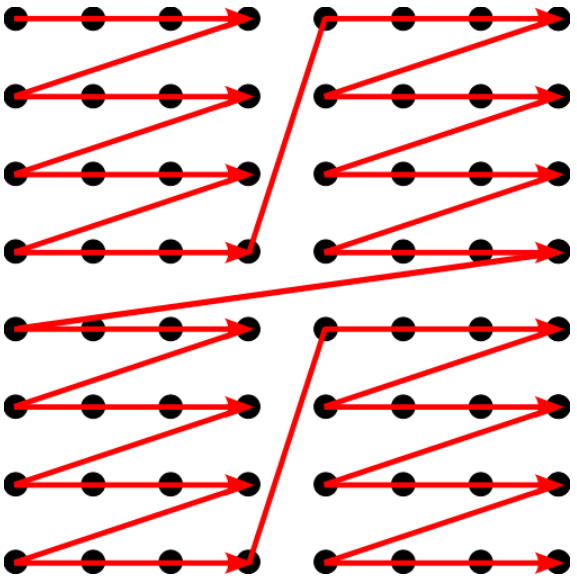
- Real-space grid for FD
- Complicated shape possible, e.g. molecules



X. Andrade & A. Aspuru-Guzik, J. Chem. Theory Comput. (2013), 9, 10, 4360-4373

Data layout

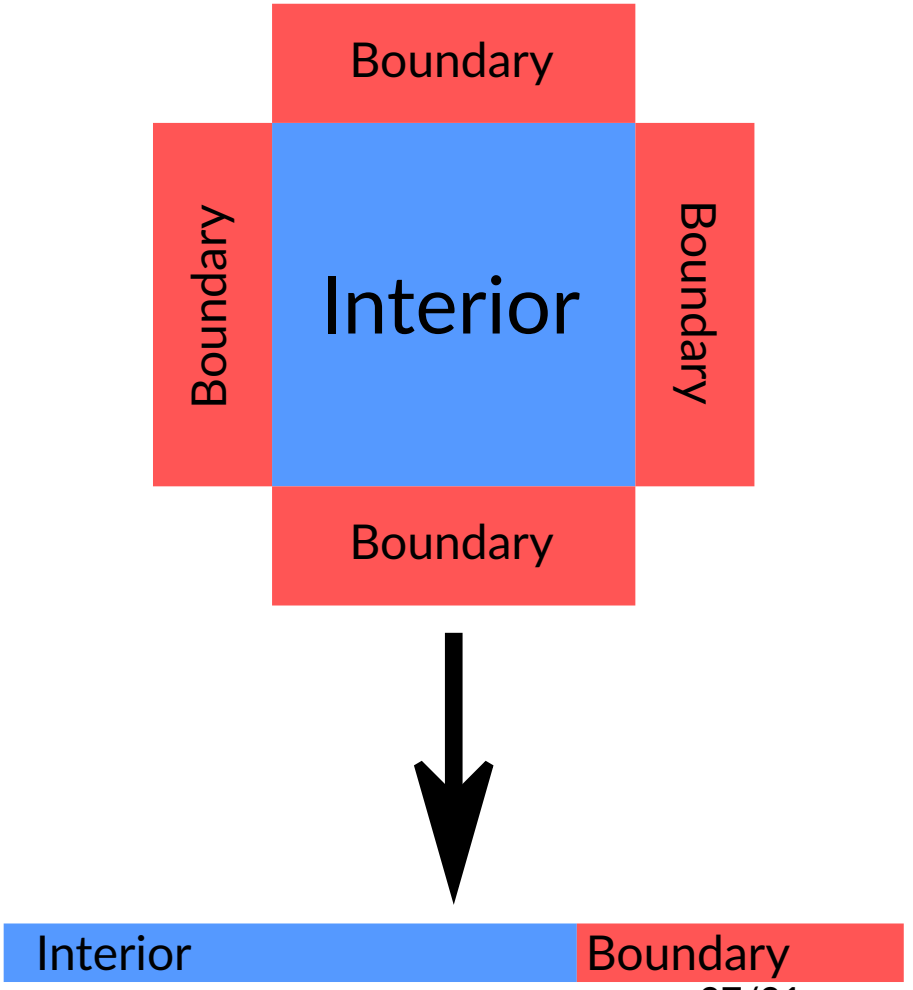
- Real-space grid for FD
- Complicated shape possible, e.g. molecules
- Cache-aware mapping to 1D array



X. Andrade & A. Aspuru-Guzik, J. Chem. Theory Comput. (2013), 9, 10, 4360-4373

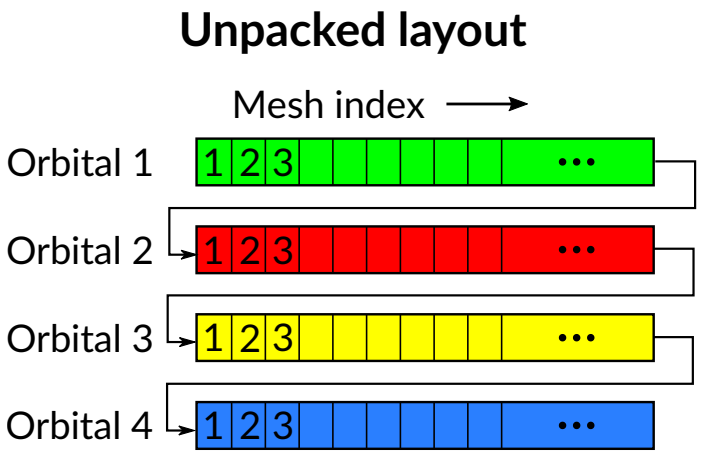
Data layout

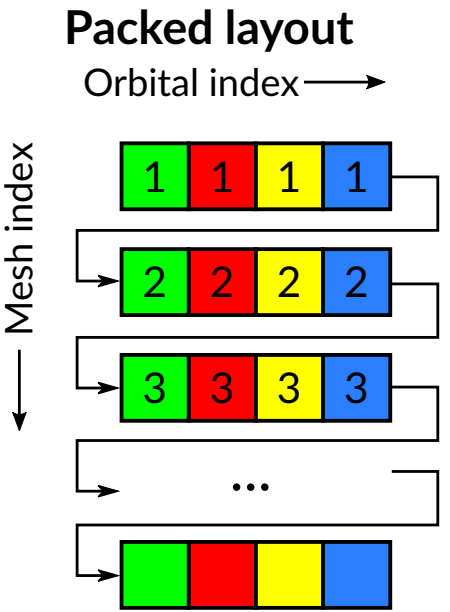
- Real-space grid for FD
- Complicated shape possible, e.g. molecules
- Cache-aware mapping to 1D array
- 1D data layout: 2 blocks
  - Interior points
  - Boundary/ghost points



Data layout II: batches

- Aggregate several orbitals into one batch
- Operations done over batches
- 2 layouts:
  - Unpacked
  - Packed → vectorization, GPUs

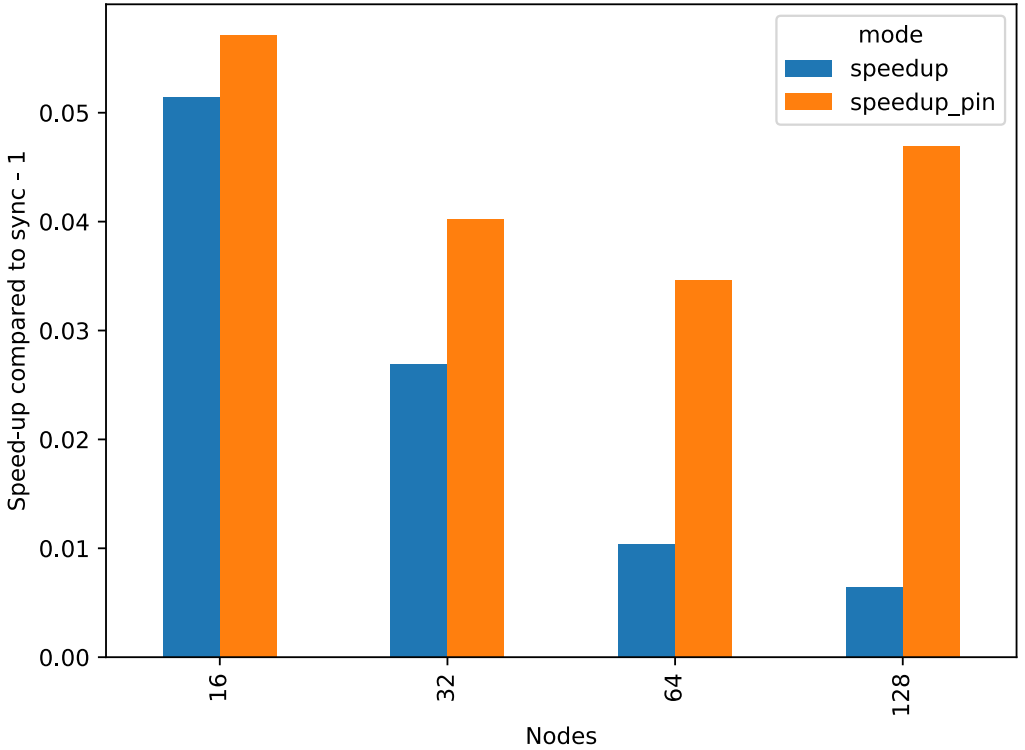




Best combinations [cobra]

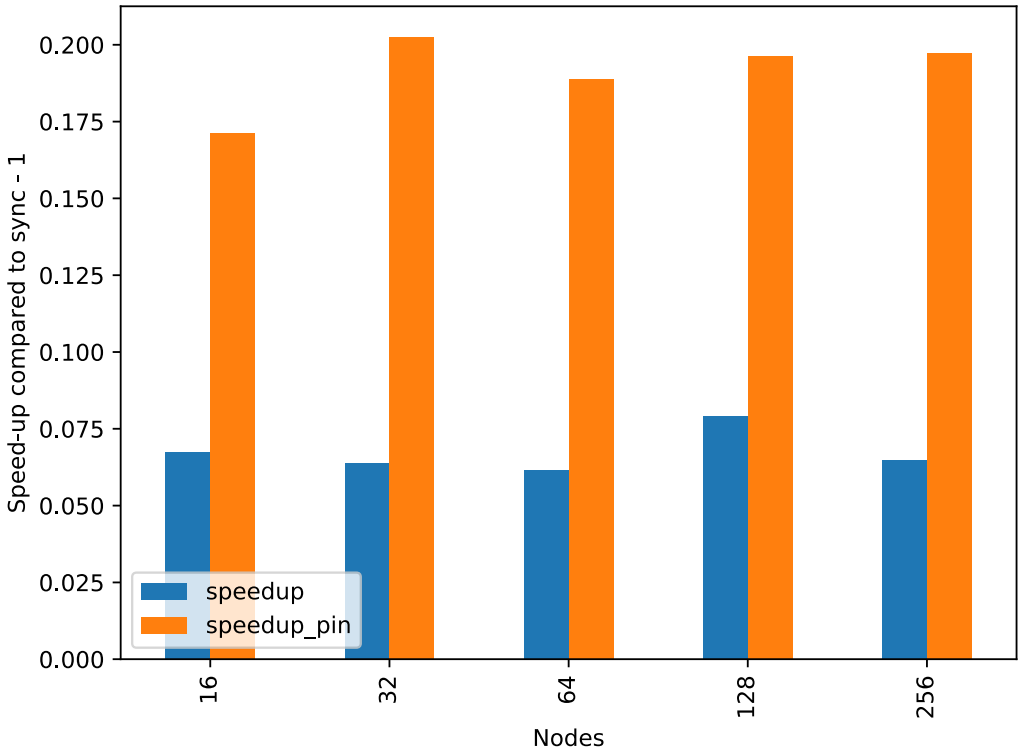
Nodes	sync	async	async_pin
16	8x5	4x10	4x9
32	8x5	4x10	8x4
64	8x5	4x10	8x4
128	8x5	4x10	8x4

Speed-up: zoom in [cobra]



With pinning: speed-up of about 1.03x – 1.06x

Speed-up: zoom in [raven]



With pinning: speed-up of about 1.17x – 1.20x