NEMORB's Fourier filter and distributed matrix transposition on Petaflop systems

Tiago Ribeiro
Matthieu Haefele

Max-Planck-Institut für Plasmaphysik

September 13, 2013

Motivation

NEMORB: Gyrokinetic PIC code (5D)

- 3D (un-aligned) spatial grid (s, χ, ϕ)
- Fourier filter in the angles
- Domain decomposition (pure MPI)
- Large datasets transposed across tasks

- Bottleneck: ITER-sized simulations impaired

Part I

- NEMORB domain decomposition
- Hermitian symmetry
- NEMORB original transposes
- XOR algorithm
- Partial transposition

Part II

- Experimental setup
- Results for the “flat MPI” implementation
- Alternative implementations
- Best execution time

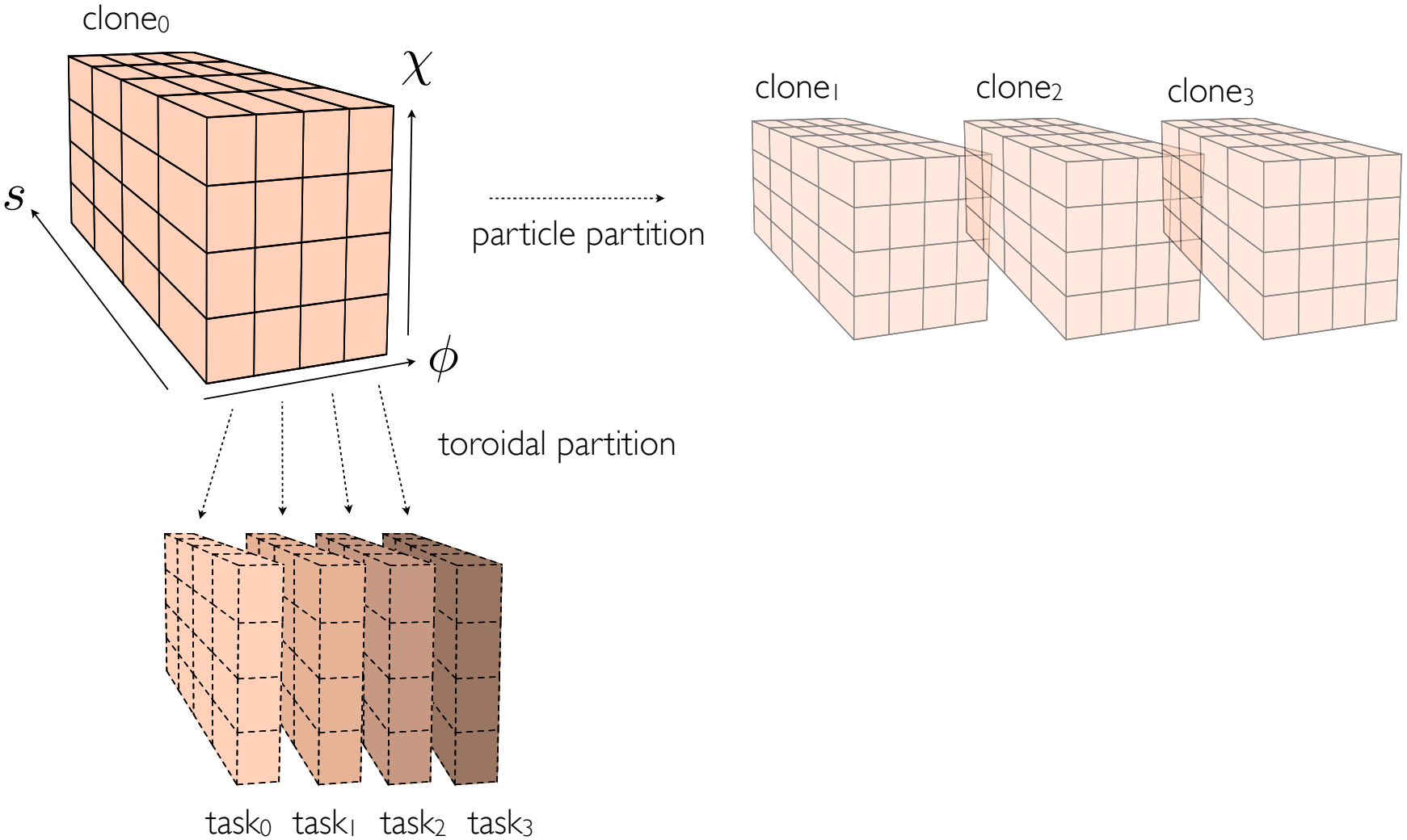
Part I

- NEMORB domain decomposition
- Hermitian symmetry
- NEMORB original transposes
- XOR algorithm
- Partial transposition

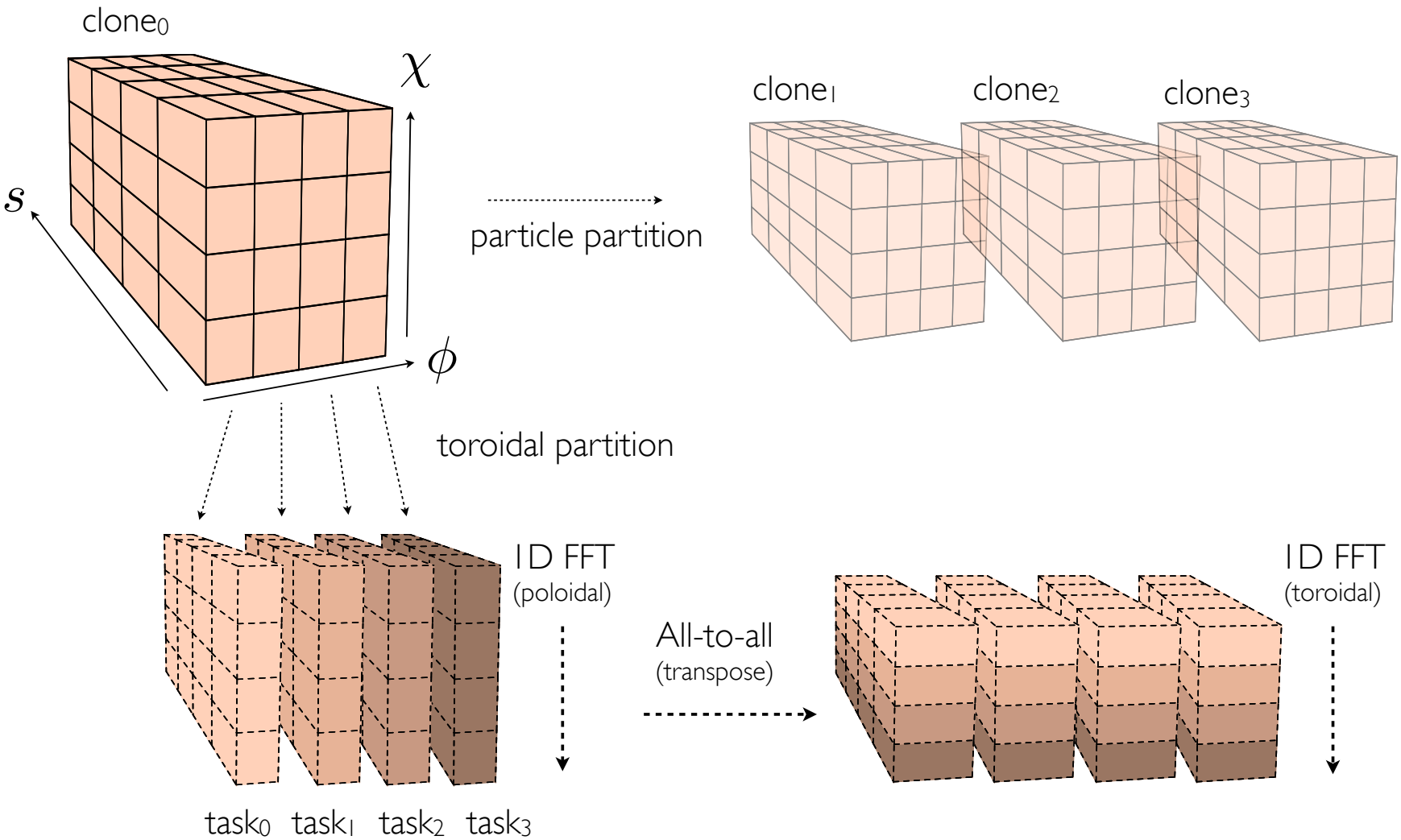
Part II

- Experimental setup
- Results for the “flat MPI” implementation
- Alternative implementations
- Best execution time

Domain decomposition (spatial and particle)



Bottleneck, why?

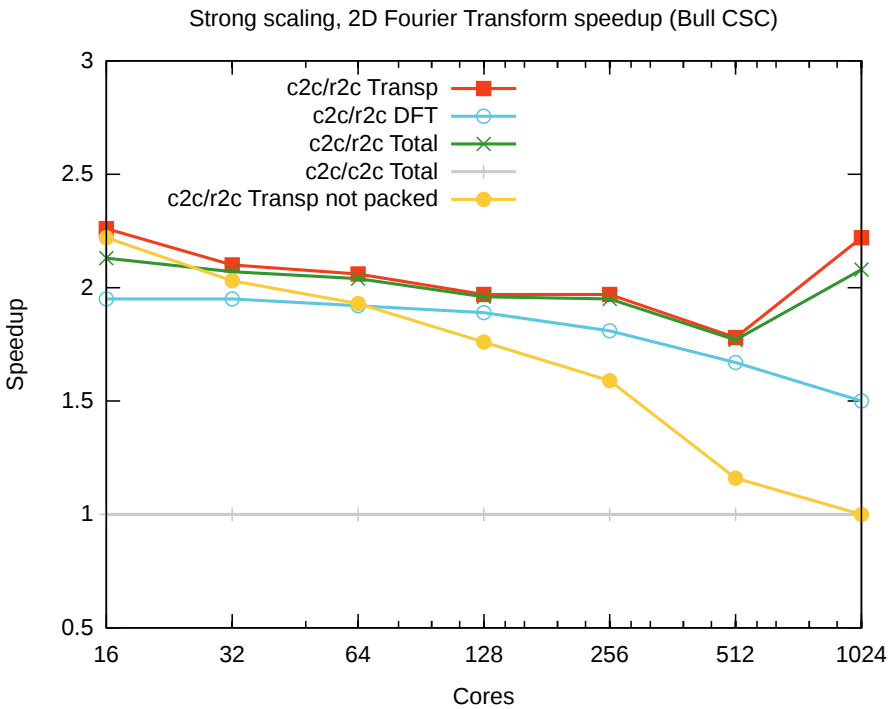


Optimization strategy

- Fourier transforms
  - Real-valued input data: Hermitian symmetry ($N \rightarrow N/2+1$)
  - Half-complex packed format: avoid zero padding ($N/2+1 \rightarrow N/2$)
  - Reduce both cost of computation and communication

- Transpose methods
  - Test different alternative transposes
  - Use the filter's shape to do partial transposes

Hermitian symmetry



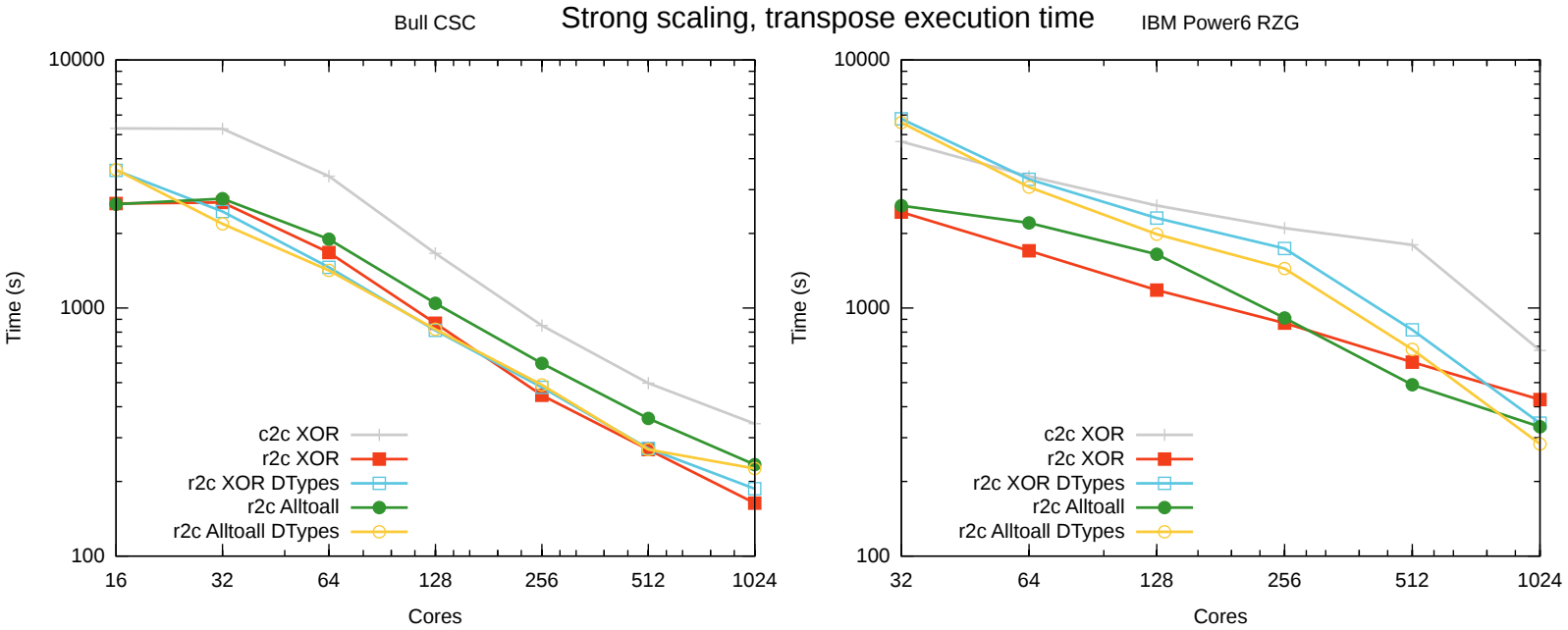
- ITER grid-count
  - $N_s = 512$
  - $N_\chi = 2048$
  - $N_\phi = 1024$
- Speedup by a factor of two
- Communication bound

- Architecture
  - **Bull** B510 blade system (70 560 cores): Computational Simulation Centre of the International Fusion Energy Research Centre (IFERC-**CSC**), Japan

NEMORB original transposes

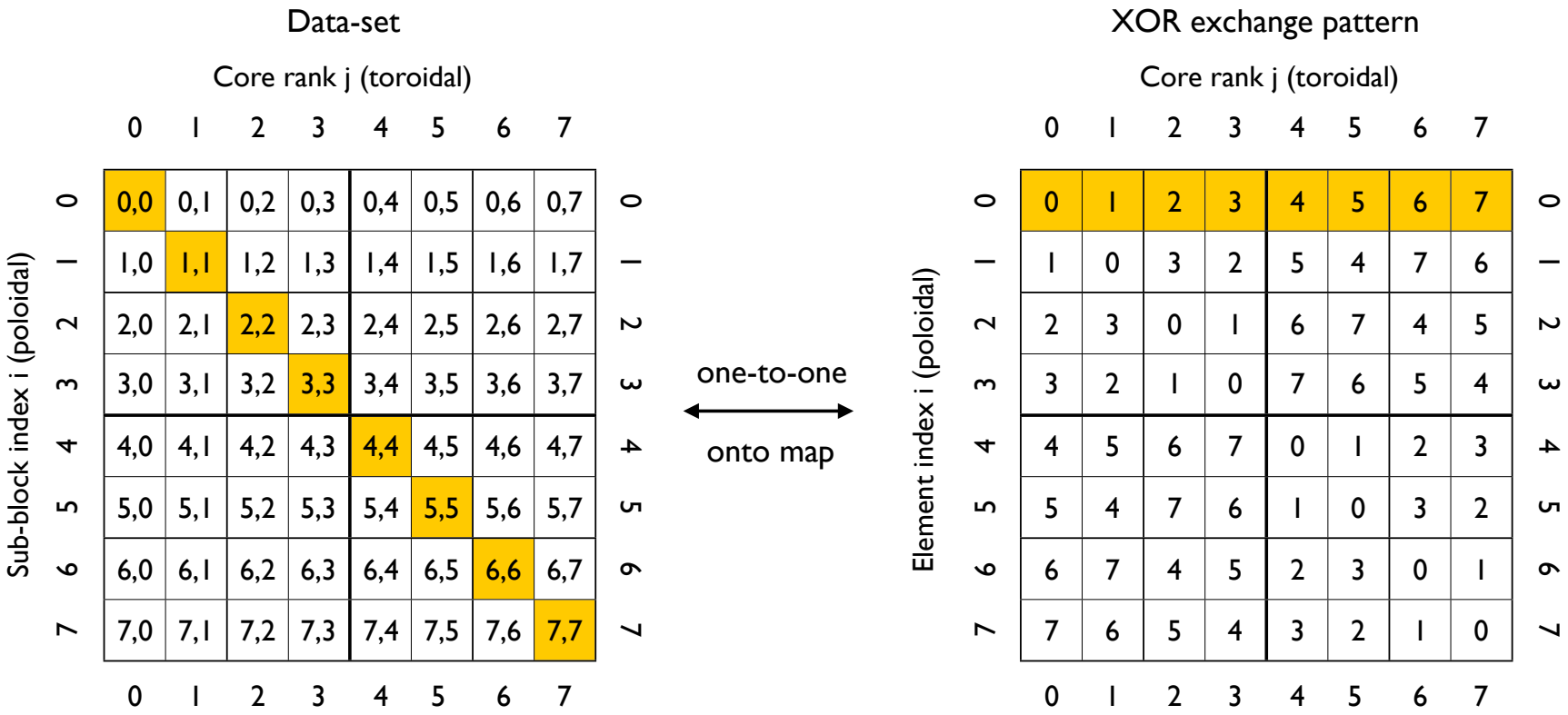
- Transpose methods
  - `MPI_Alltoall`
  - XOR/`MPI_Sendrecv`
- Memory access
  - Local buffers for data contiguity
  - MPI derived data types
- Architectures
  - **Bull** B510 blade system (70 560 cores): Computational Simulation Centre of the International Fusion Energy Research Centre (IFERC-**CSC**), Japan
  - **Bull** NovaScale R422-E2 system (8 640 cores): Jülich Supercomputing Centre (**JSC**), Germany
  - **IBM Power6** 575 (6 560 cores): Rechenzentrum Garching (**RZG**), Germany

NEMORB original transposes

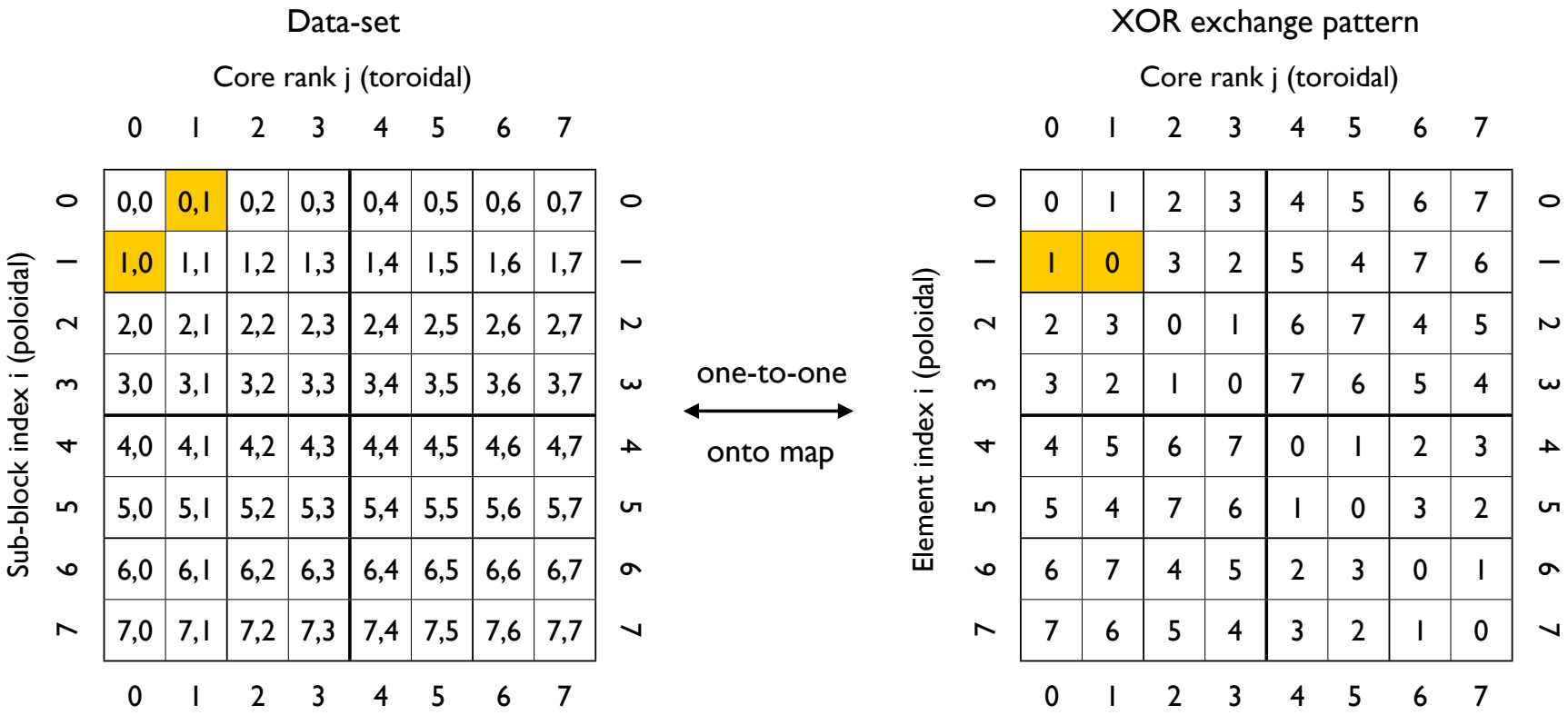


- Best method is machine/MPI implementation dependent
  - `MPI_Alltoall`: best on IBM Power6 (RZG) with derived types
  - XOR method: best on dedicated fusion HPCs (Bull CSC and JSC)
  - XOR method: suited for finer tuning

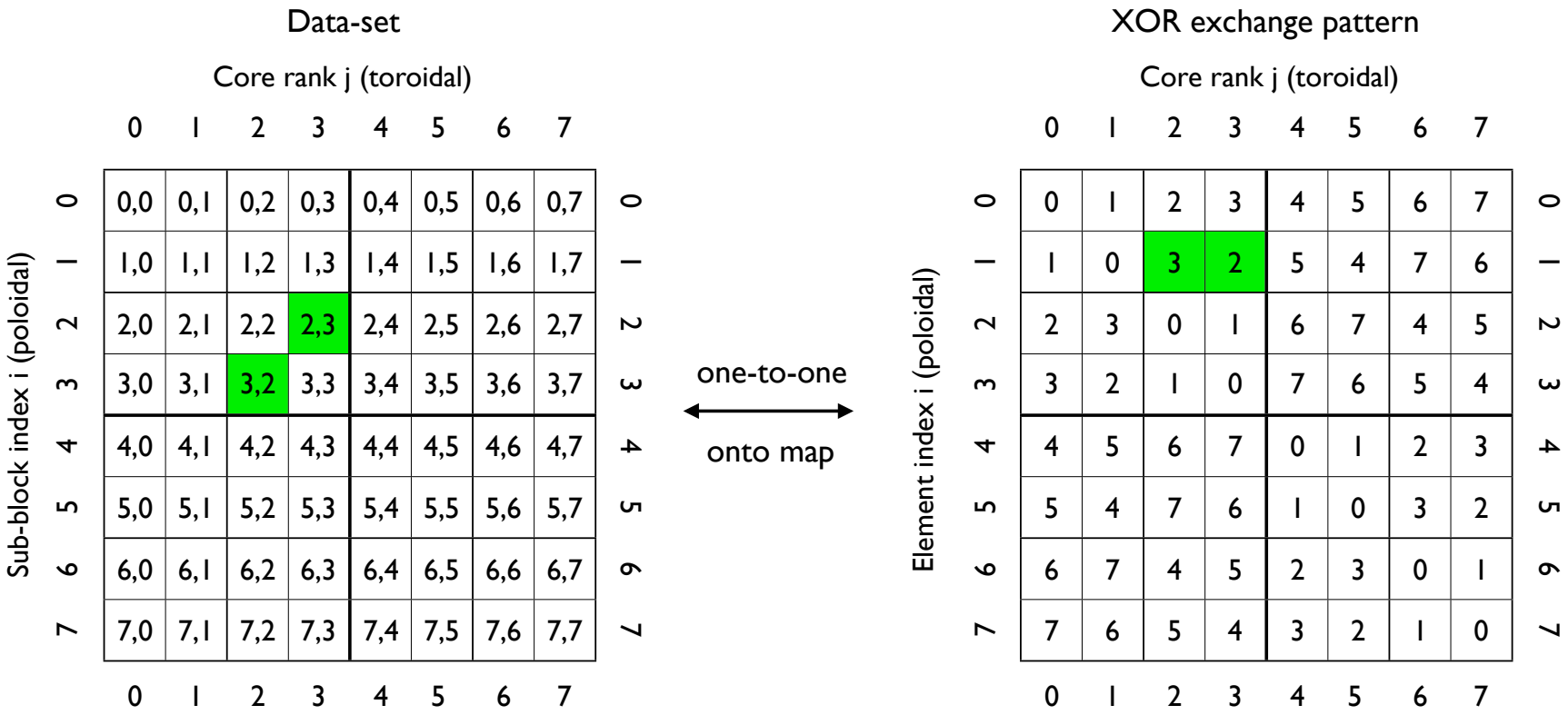
XOR algorithm



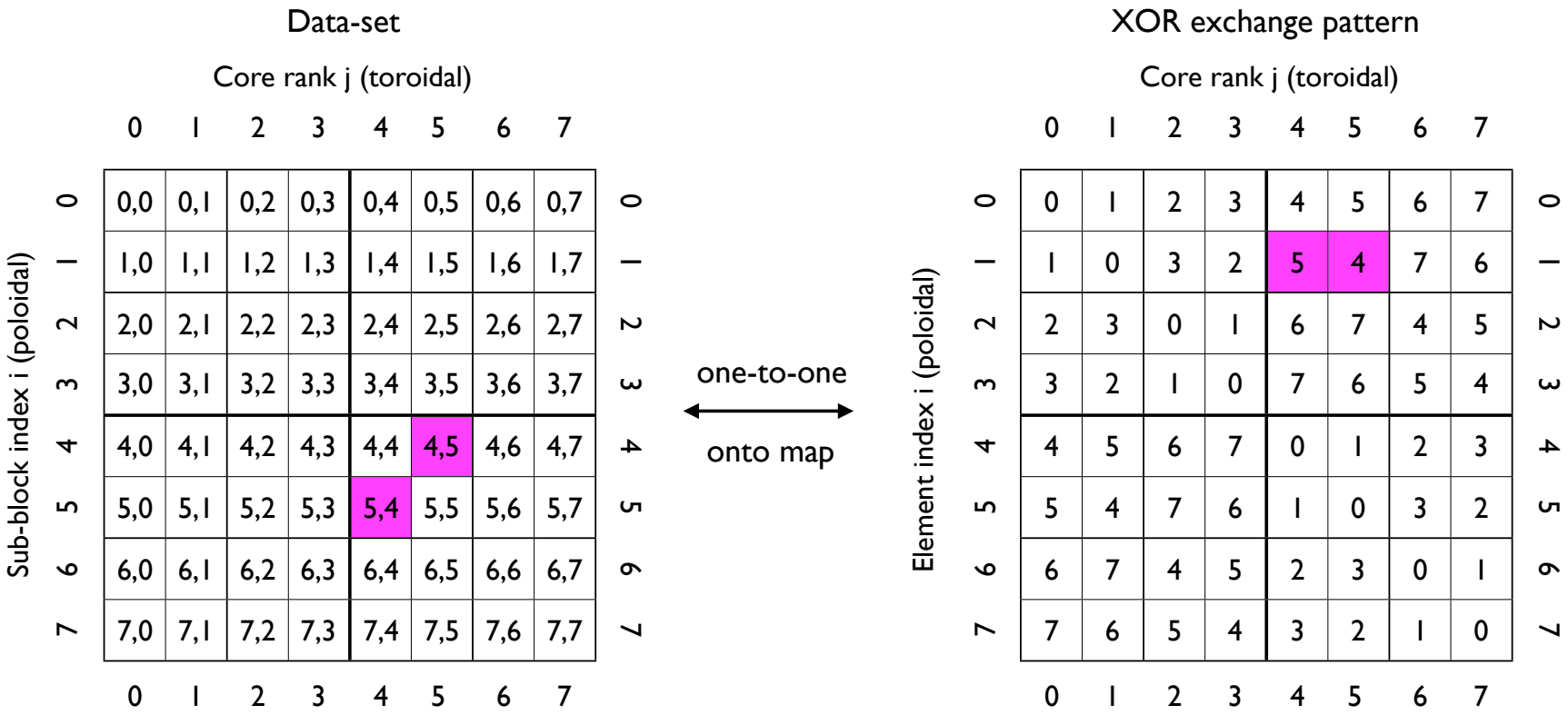
XOR algorithm



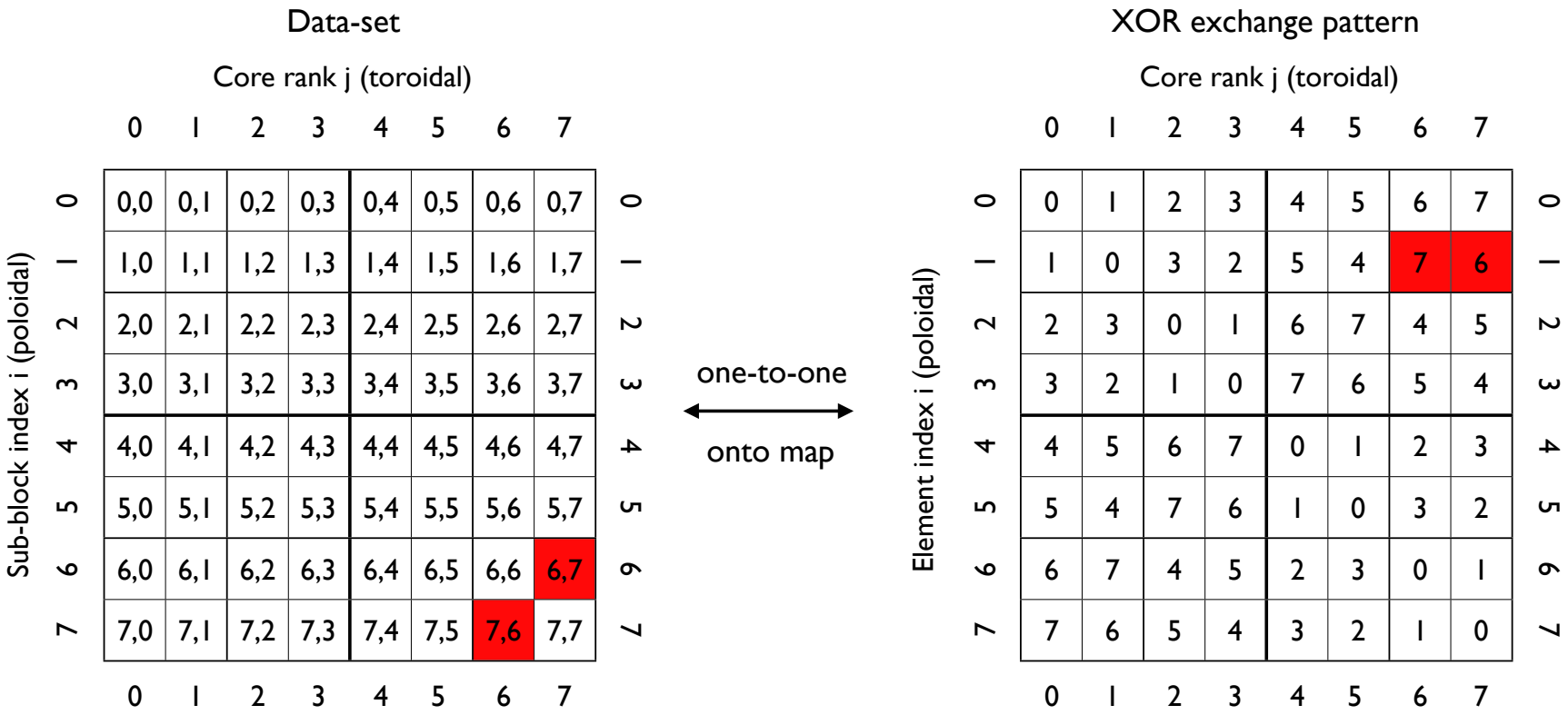
XOR algorithm



XOR algorithm

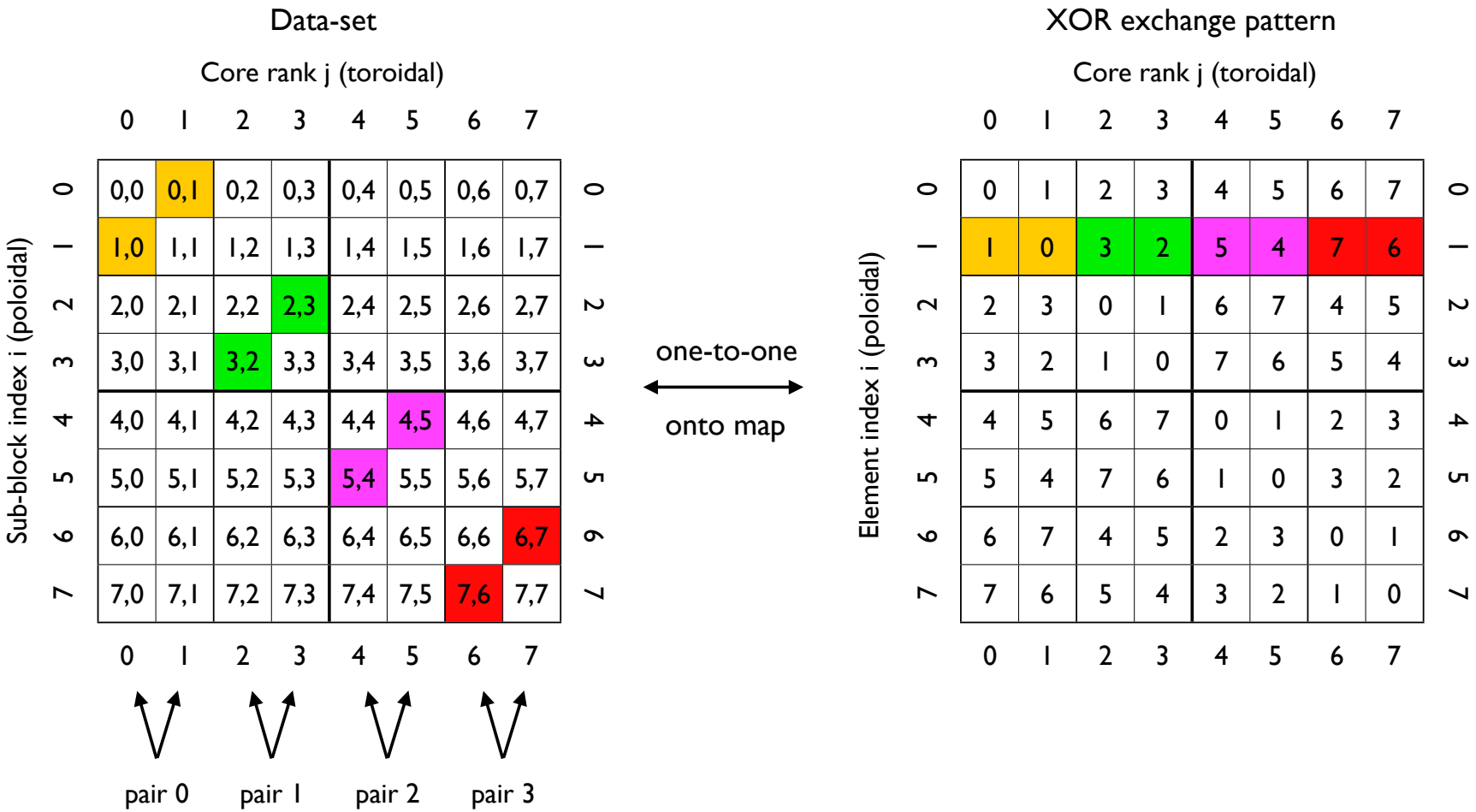


XOR algorithm

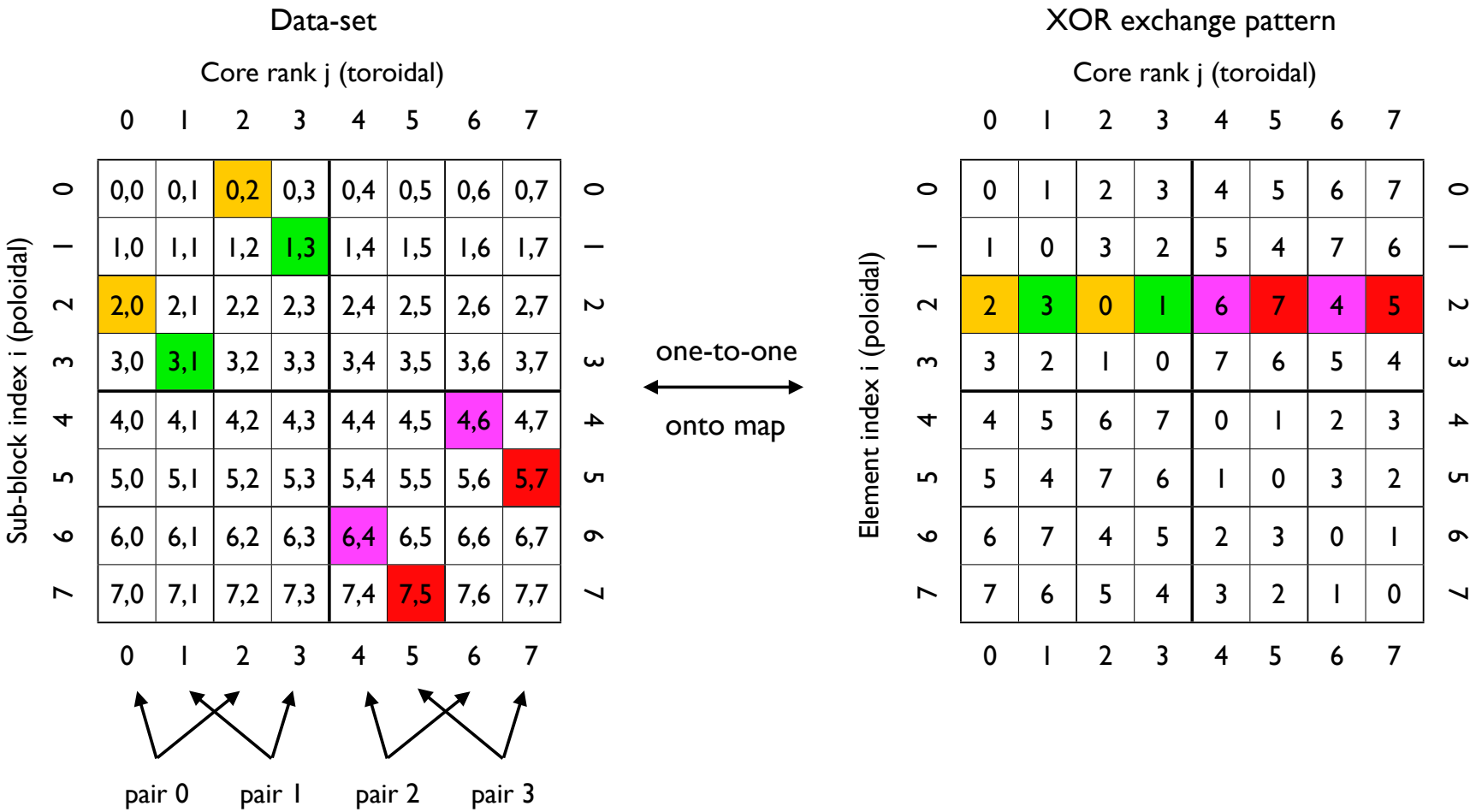


All pairs within row executed in parallel

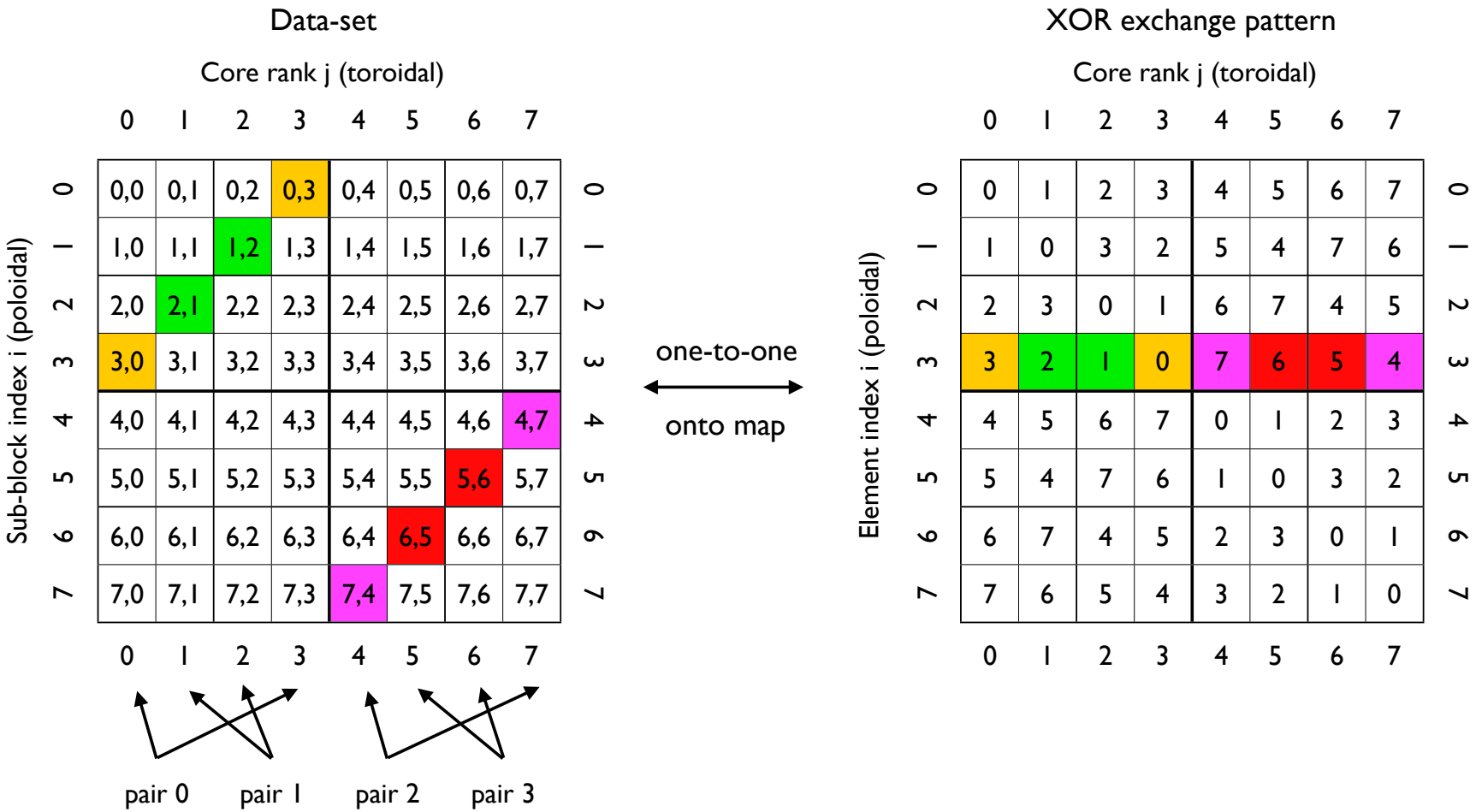
XOR algorithm



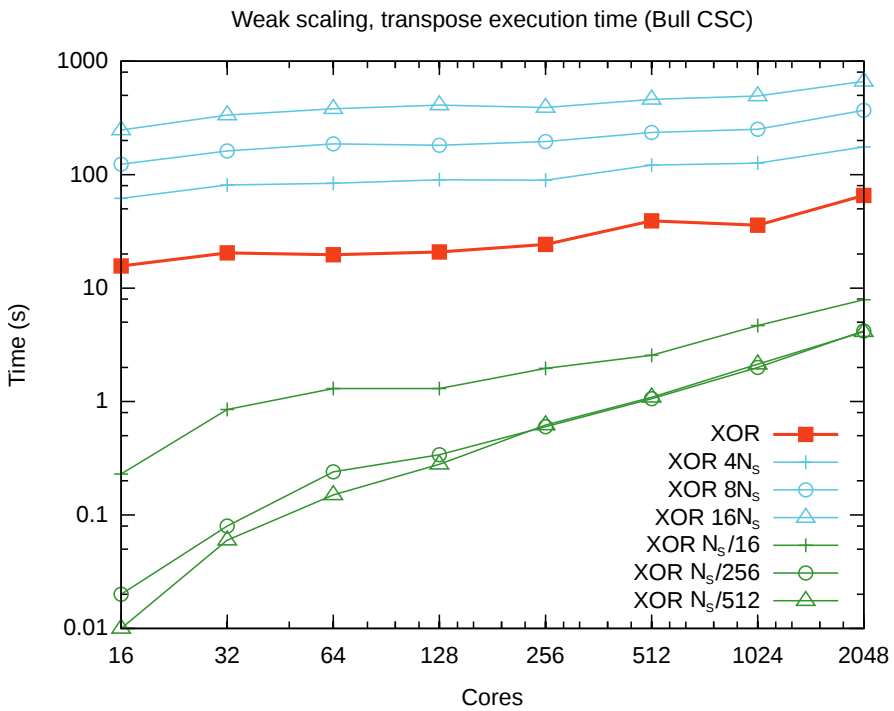
XOR algorithm



XOR algorithm



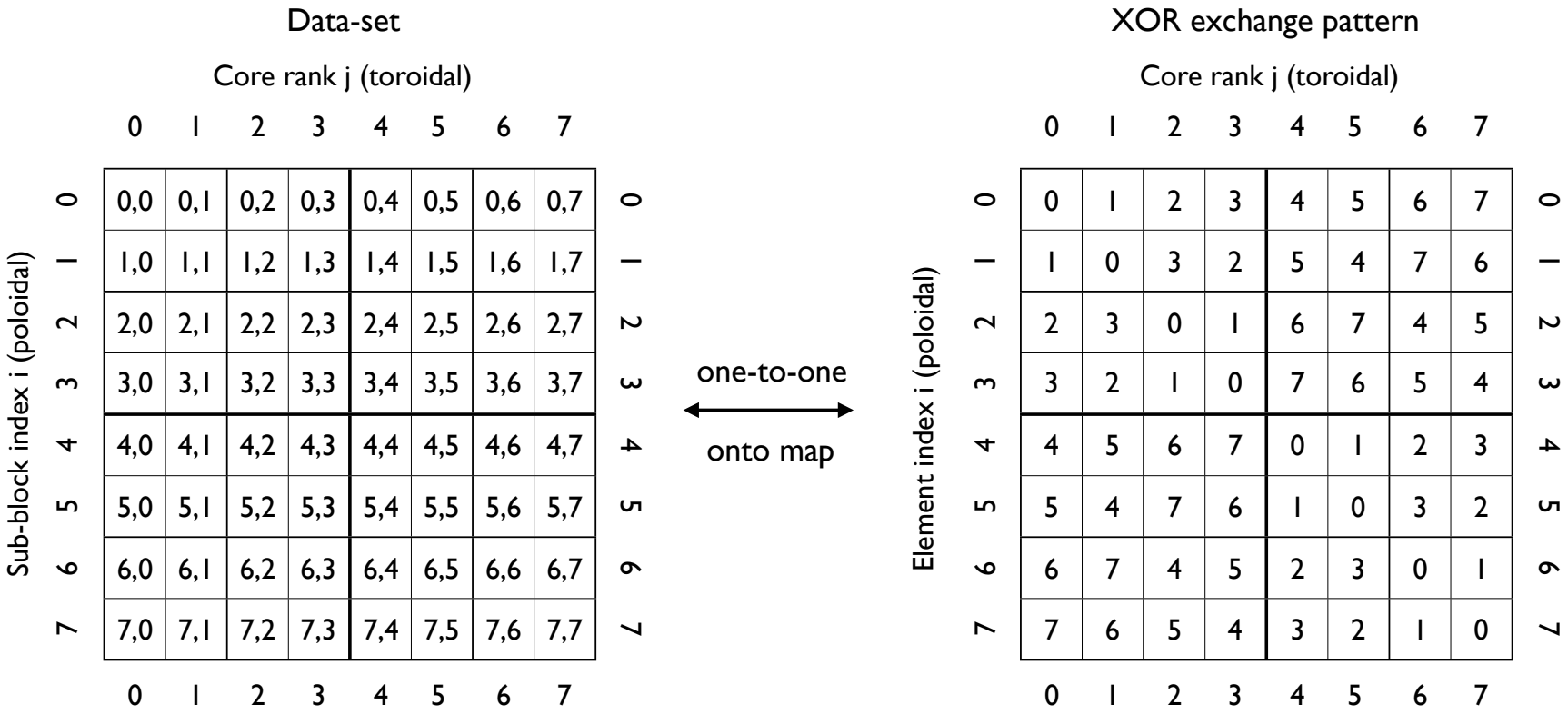
XOR performance: message size



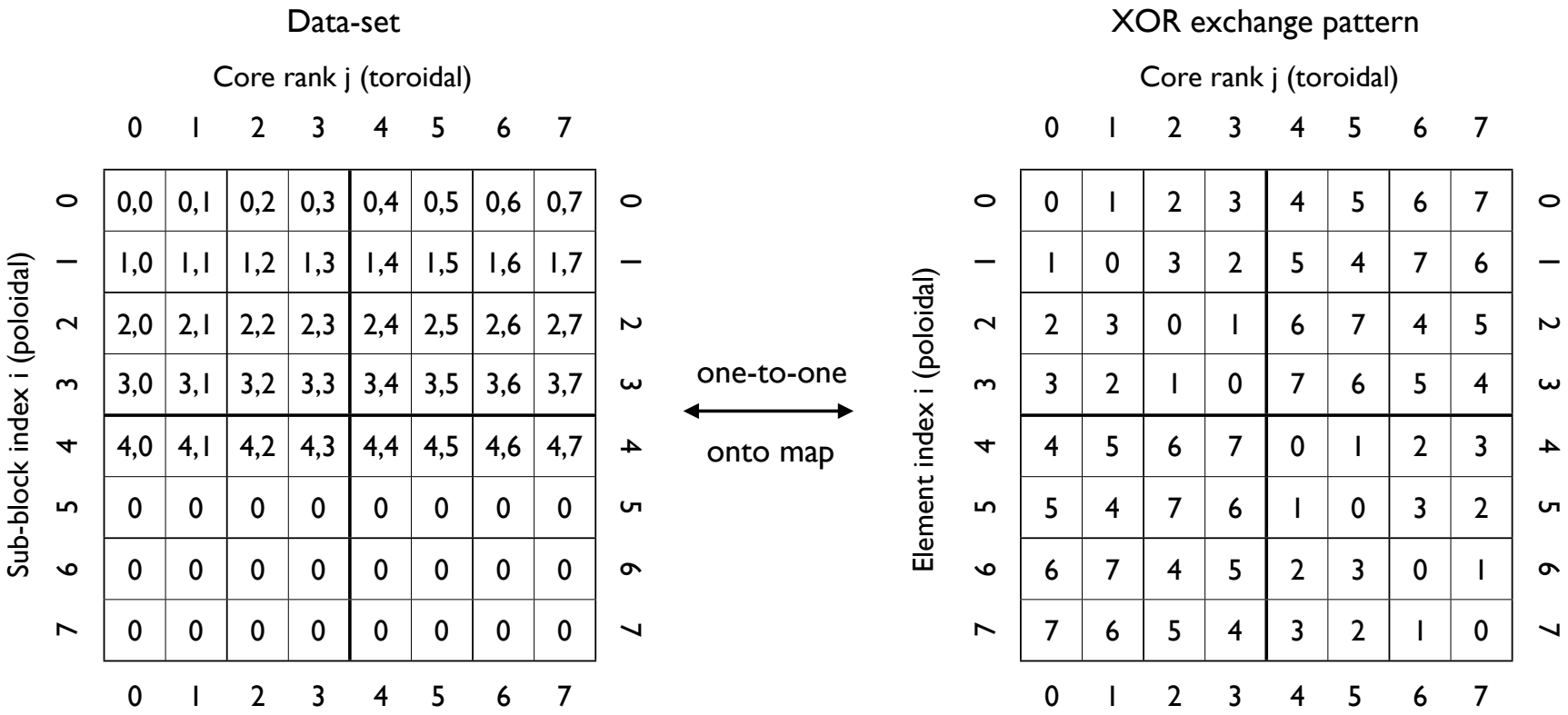
- Weak scaling:
  $512 \times 2048 \times N_{\text{cores}}$
- Constant data amount per task
  - # messages $\mathcal{O}(N_{\text{cores}}^2)$
  - # exchange pairs $\mathcal{O}(N_{\text{cores}})$
  - Messages sizes $\mathcal{O}(N_{\text{cores}}^{-1})$
  - Cancellation: flat curve
- Degradation
  - Network limitations
  - Smaller messages: latency
  - Bigger messages: congestion

Can we improve burden on the network?

XOR partial transposition

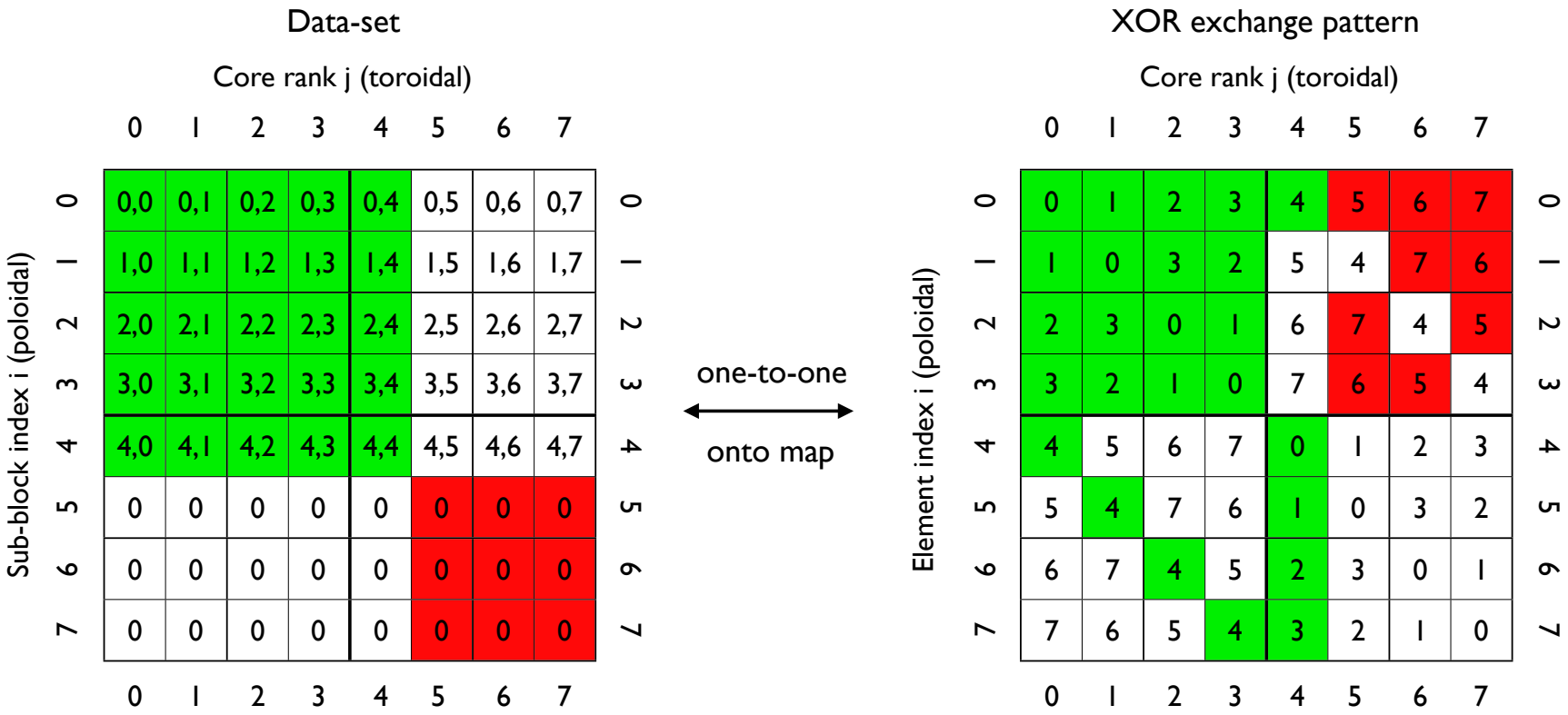


XOR partial transposition



- Low pass square filter: poloidal component

XOR partial transposition



- Untouched and inhibited parts of the XOR exchange pattern
- Additional 5 – 10% speedup gain (ITER-size)

Part I

- NEMORB domain decomposition
- Hermitian symmetry
- NEMORB original transposes
- XOR algorithm
- Partial transposition

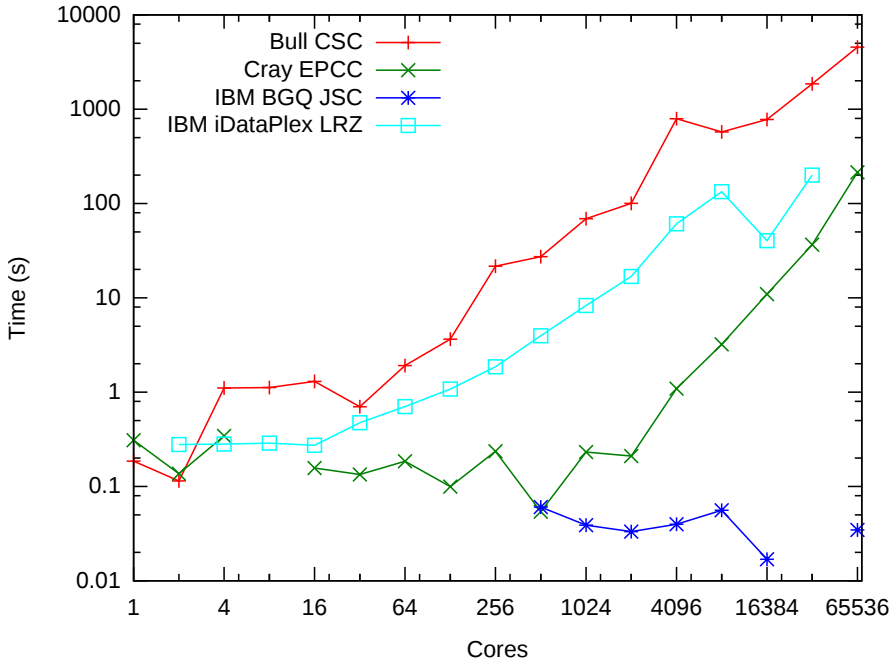
Part II

- Experimental setup
- Results for the “flat MPI” implementation
- Alternative implementations
- Best execution time

Experimental setup

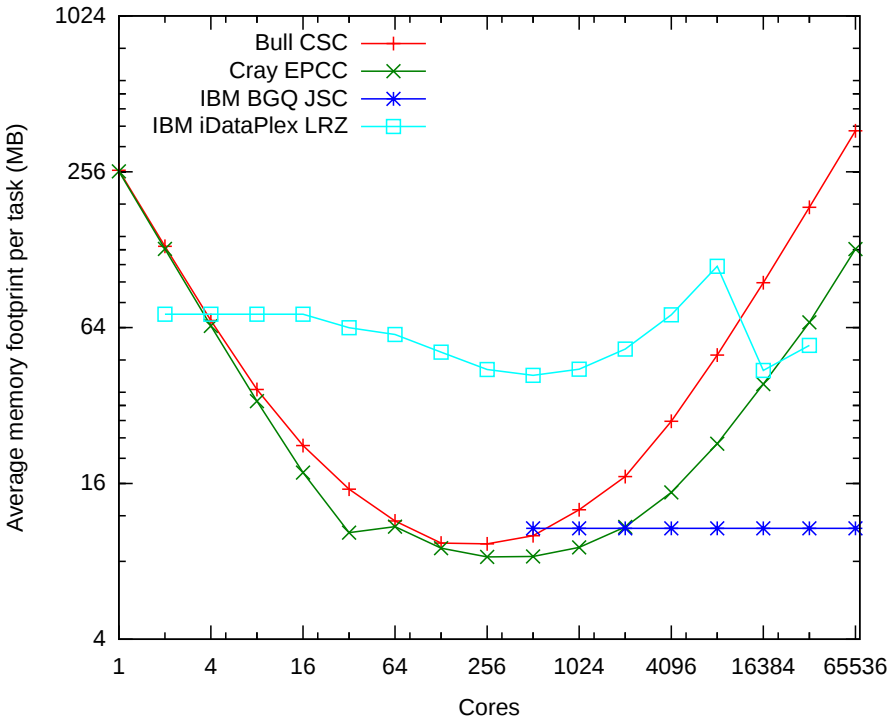
- Weak scaling with 128 MB sub-matrix per task
- Ten successive 2D matrix transposes
- Results out of one run using the XOR method
- Four architectures
  - **Bull** B510 blade system (70 560 cores): Computational Simulation Centre of the International Fusion Energy Research Centre (IFERC-**CSC**), Japan
  - **Cray** XE6 (90 112 cores): Edinburgh Parallel Comp. Centre (**EPCC**), U.K.
  - **IBM Blugene Q** (458 752 cores): Jülich Supercomputing Center (**JSC**), Germany
  - **IBM** System x **iDataPlex** (147 456 cores): Leibniz-Rechenzentrum (**LRZ**), Germany
- Three quantities measured for each transpose on each task
  - Execution time: $t_{\mathrm{exec}} = \langle t_{5-10} \rangle = \sum_{i=1}^{5} t_i/5$
  - MPI initialization time: $t_{\mathrm{init}} = t_{\texttt{MPI\_Init}} + t_1 - t_{\mathrm{exec}}$
  - MPI memory usage: measured around `MPI_Init` and `MPI_Sendrecv`

MPI Initialization time



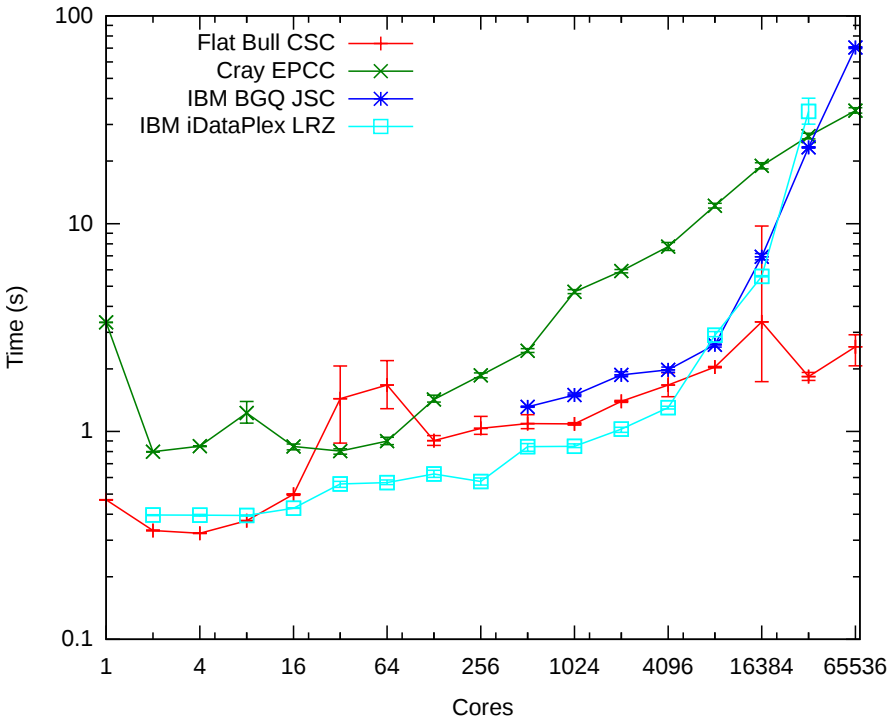
- Prohibitive cost for Bull system hopefully fixed in the next Bull MPI release
- Cost negligible for IBM BGQ
- Affordable cost of 200 s for both Cray (64k cores) and IBM iDataPlex (32k cores) systems

MPI memory usage



- MPI memory footprint negligible for IBM BGQ
- Oscillations between 40 and 110 MB for IBM iDataPlex
- Stronger variations for Bull and Cray systems (up to 370 and 256 MB resp.)

Execution time



- Stays below 2.5 s up to 64k cores for Bull
- Stays below 3 s for IBM iDataPlex up to 8k cores and degrades hereafter
- Flat MPI implementations perform poorly for Cray and IBM BGQ

Alternative implementations

- **Hybrid** MPI/OpenMP
- Pure MPI, one communicating task per node
  - **Shared memory segments**
  - **Two communication levels**

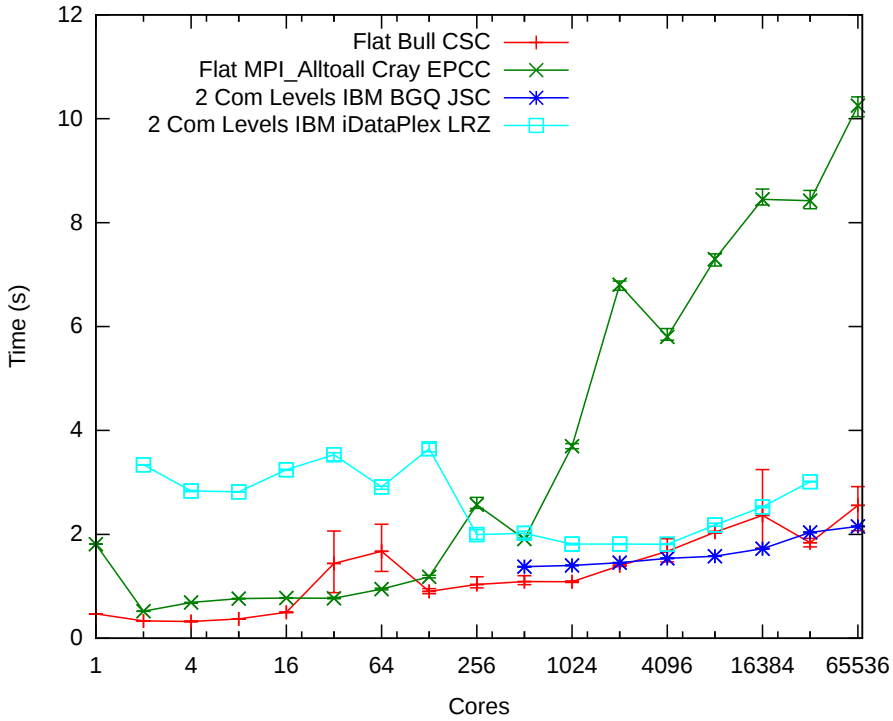
Alternative implementations

- **Hybrid** MPI/OpenMP
- Pure MPI, one communicating task per node
  - **Shared memory segments**
  - **Two communication levels**

- MPI initialization time: an order of magnitude lower (more on Bull)
- MPI memory usage
  - Reduced by an order of magnitude for the **hybrid** approach
  - **Shared memory segment** similar to **flat**, still under investigation
  - **Two communication levels** similar to **flat** due to additional buffer

Execution time (best)

- Bull: **MPI flat** is the fastest
- Cray: performance drops for large numbers of cores. Flat `MPI_Alltoall` is the best
- IBM BGQ: **Two comm. levels** is the fastest
- IBM iDataPlex: much less degradation on more islands, **two comm. levels** is the best



Conclusion

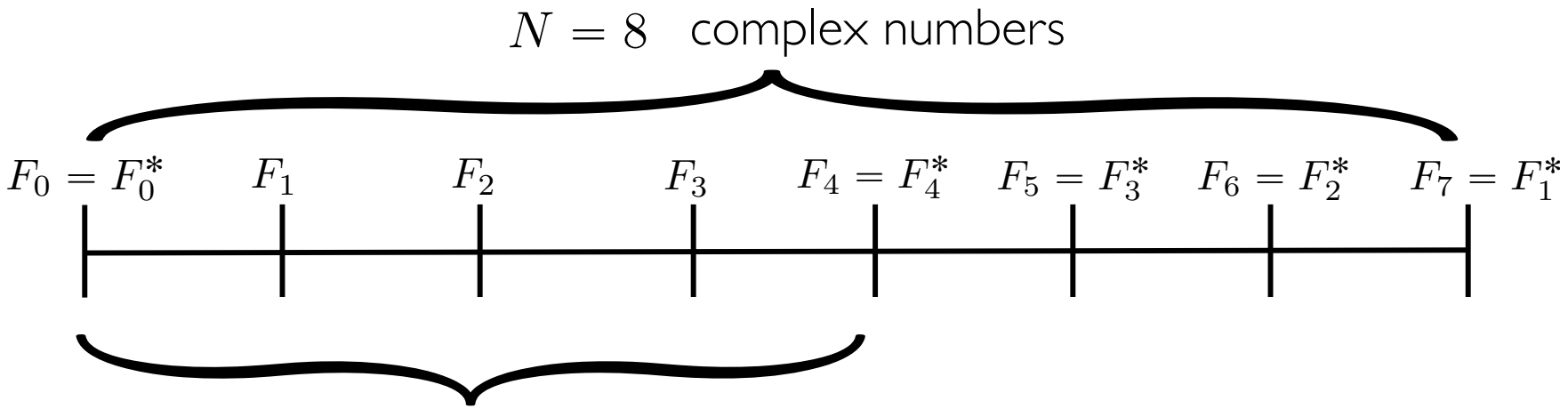
- Fourier filtering procedures in NEMORB successfully improved
- Speedup $\gtrsim$ two obtained: Hermitian symmetry + partial transpose
- Gather/scatter, BLACS, FFTW alternatives attempted: not better
- MPI 3 implementations: non blocking collective directives (future)

- Four matrix transpose implementations on four petaflop systems
- Flat MPI implementation not yet prohibitive on nowadays systems
- Hybrid MPI/OpenMP implementation
  - Substantial improvements in MPI init time and memory footprint
  - Never exhibited the best performance in execution time

Backup slides

- Hermitian symmetry
- Alternative “canned” transposes
- Gather/scatter, yet another alternative
- Gather vs. XOR transposes
- Initial implementation (MPI flat)
- Alternative implementations (Hybrid/SMS/Two level)

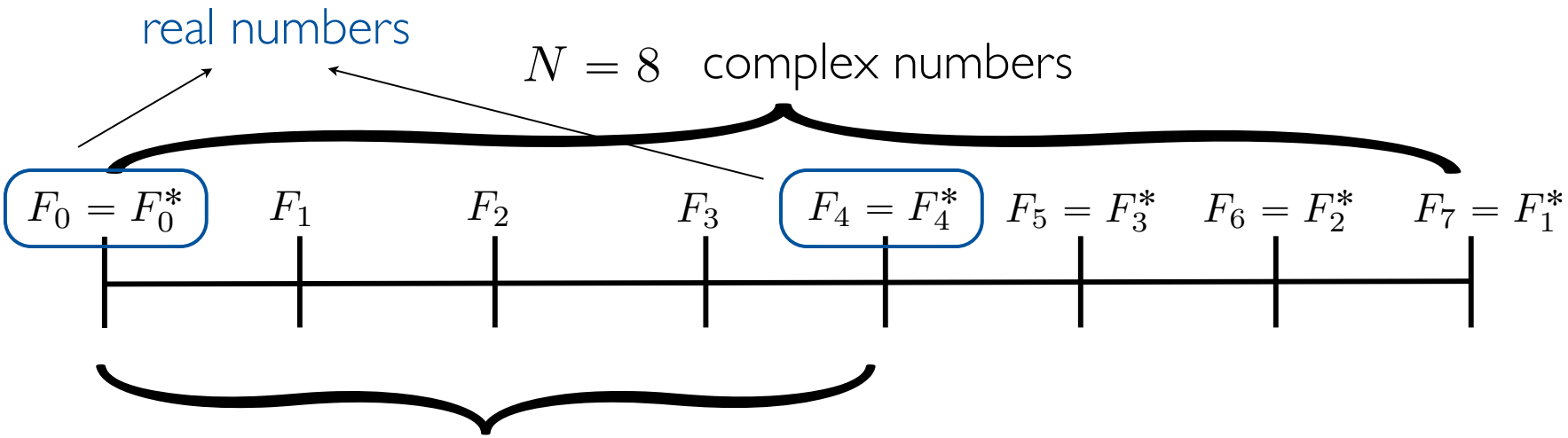
Hermitian symmetry



$N/2 + 1 = 5$ independent complex numbers

- Redundant spectra of real-valued data

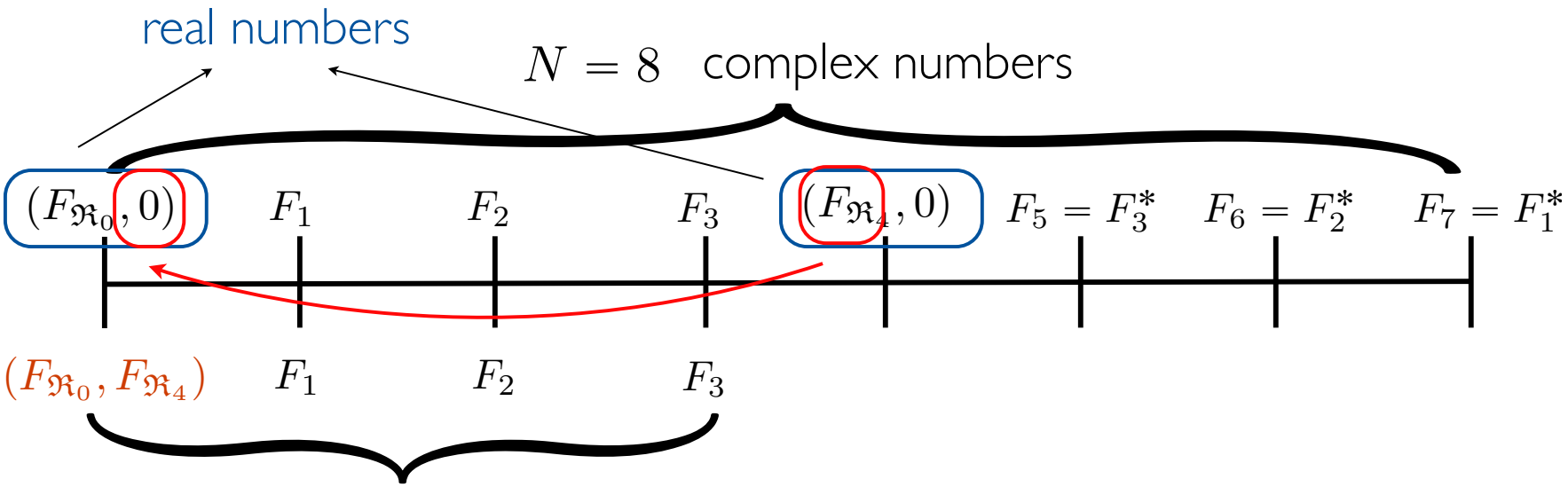
Hermitian symmetry



$N/2 + 1 = 5$ independent complex numbers

- Redundant spectra of real-valued data
- Real-valued zero and Nyquist modes

Hermitian symmetry



$N/2 = 4$ complex numbers (no information loss)

- Redundant spectra of real-valued data
- Real-valued zero and Nyquist modes
- Half-complex packed format: binary exchange

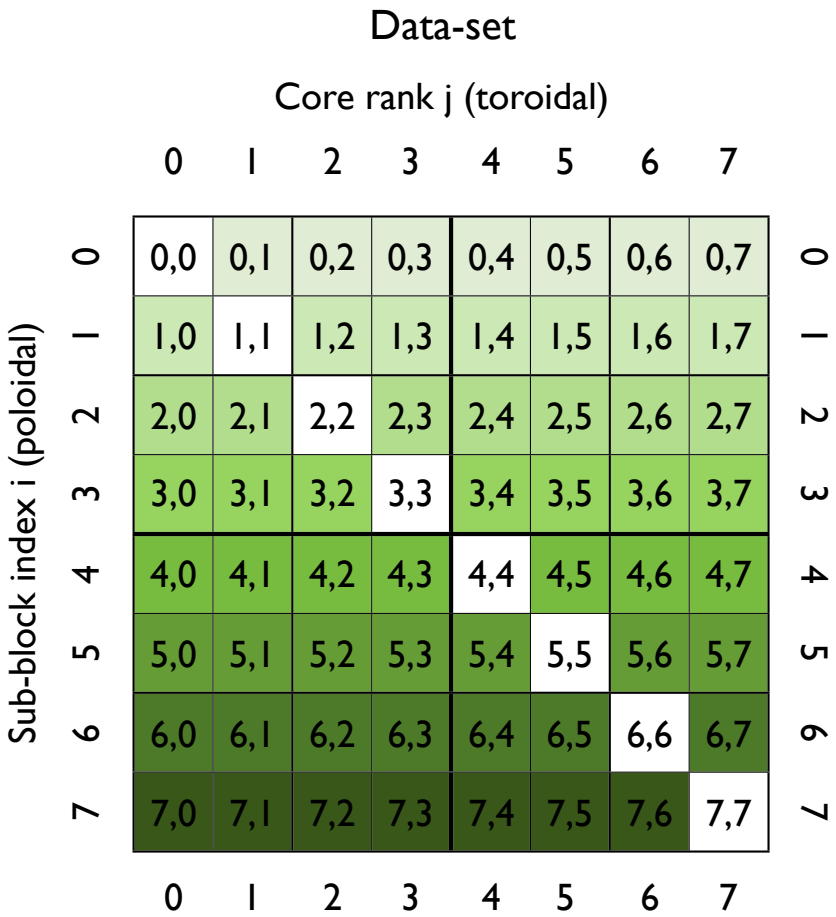
Alternative "canned" transposes

- FFTW3.3 transpose (Fortran ISO C bindings)
- BLACS via Intel MKL library
- 2D spatial domain
  - $N_\chi = 32\,768$, $N_\phi = 32\,768$ on 1024 cores
  - Same message sizes as ITER 3D grid-count
  - Elapsed times for 100 transposes

	XOR	Alltoall	FFTW3.3	BLACS
Bull CSC	14.4 s	25.2 s	46.1 s	839.6 s
Bull JSC	12.9 s	41.2 s	25.6 s	1166.9 s

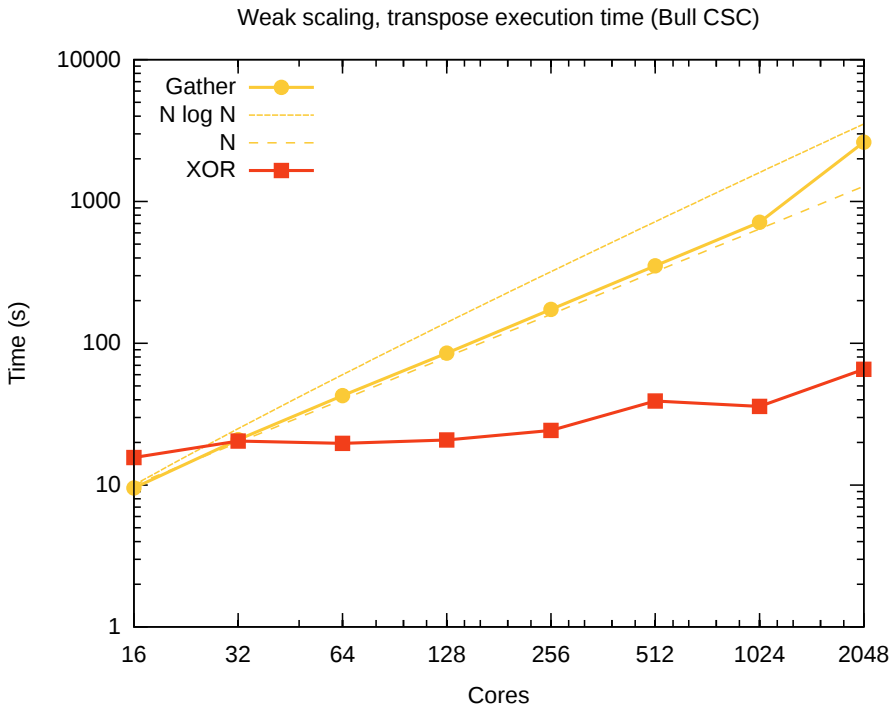
These alternatives are NOT competitive for such problem sizes

Gather/scatter, yet another alternative



- Gather method
  - Each task gathers its row
  - Next task starts once previous finishes
  - Blocking nature of `MPI_Gather`
  - Scatter method equivalent
  - How does it scale?

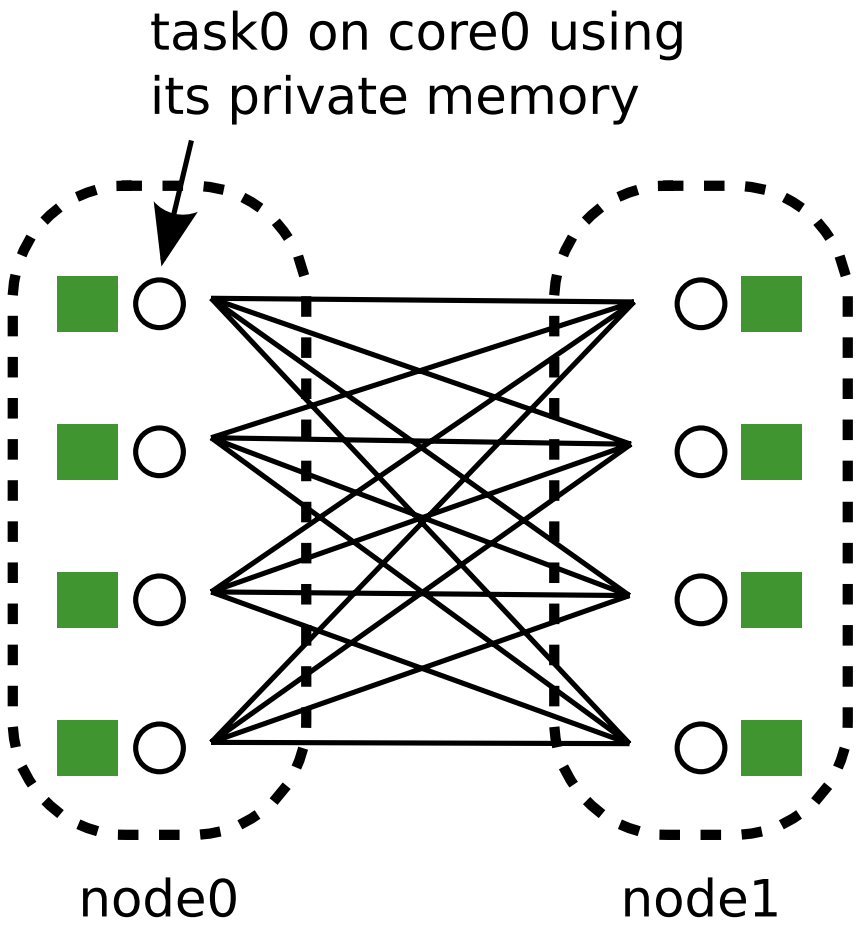
Gather vs. XOR transposes



- Gather method
  - Each gather perfectly scales
  - N_χ such blocking steps
  - Bad linear weak scaling
- XOR method
  - Much better weak scaling
  - Flat up to 256 cores
  - Degradation afterwards

XOR performs better

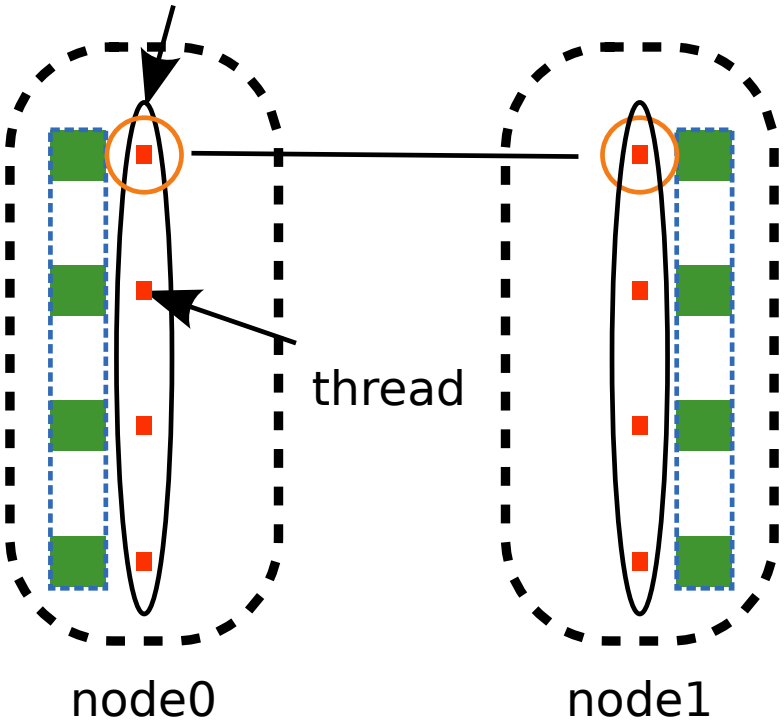
Initial implementation (MPI flat)



- Full connected net of tasks
- n^2 problem
- Initialization time and memory footprint limitation

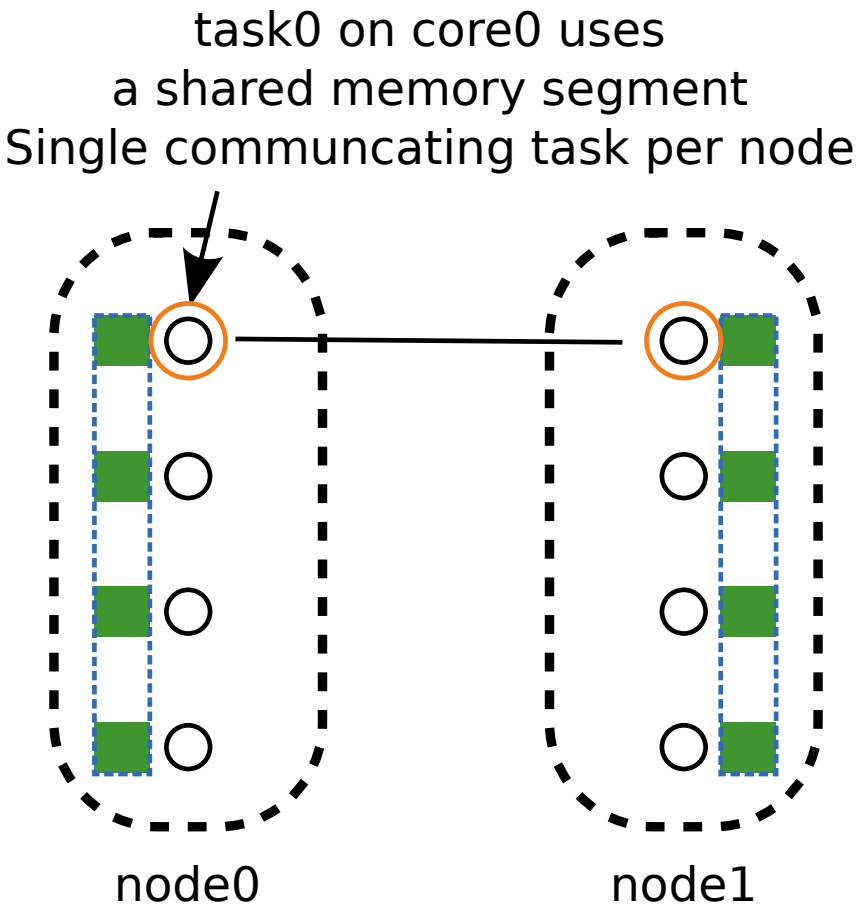
MPI OpenMP

task0 on node0 uses
the shared memory over the node
Single communcating task per node
Computations done by threads



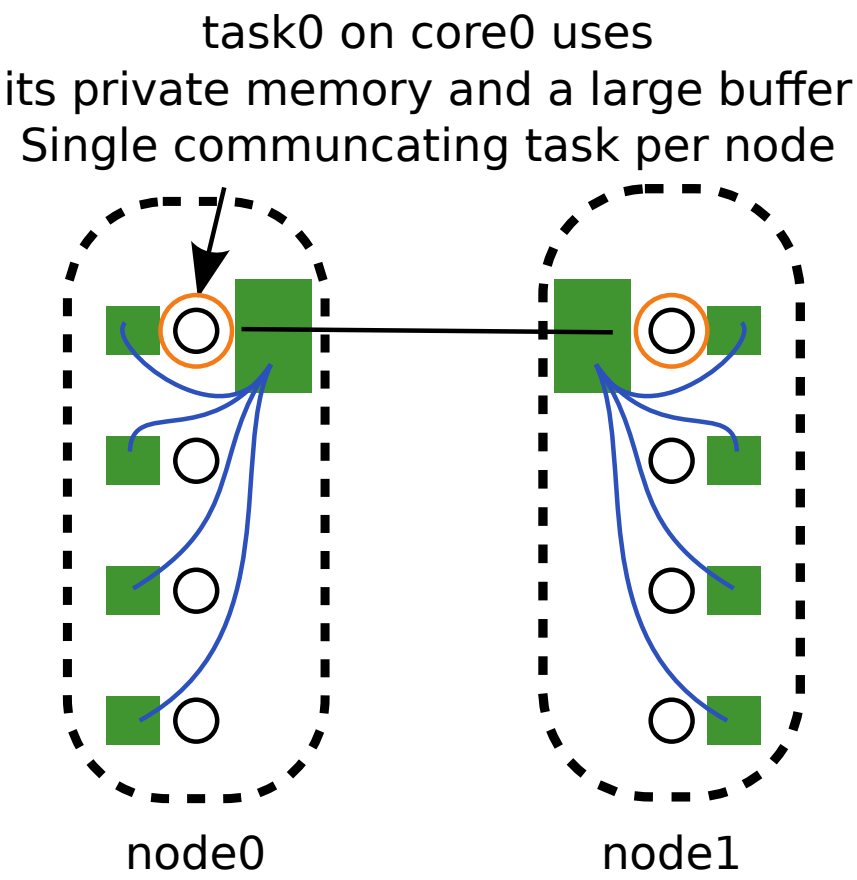
- $(n/nbTaskPerNode)^2$ problem
  - OpenMP implementation

MPI shared memory segment



- Usage of a shared memory segment instead of private memory
- $(n/nbTaskPerNode)^2$ problem
- Shared memory segment (de)allocation and race conditions management

MPI two levels



- Usage of an additional buffer to communicate with other nodes
- $(n/nbTaskPerNode)^2$ problem
- Memory overhead, execution time