Molecular Dynamics Practical

Domain decomposition

Hybrid parallel execution – process & thread placement

Partners

Funding

Horizon 2020
European Union Funding
for Research & Innovation

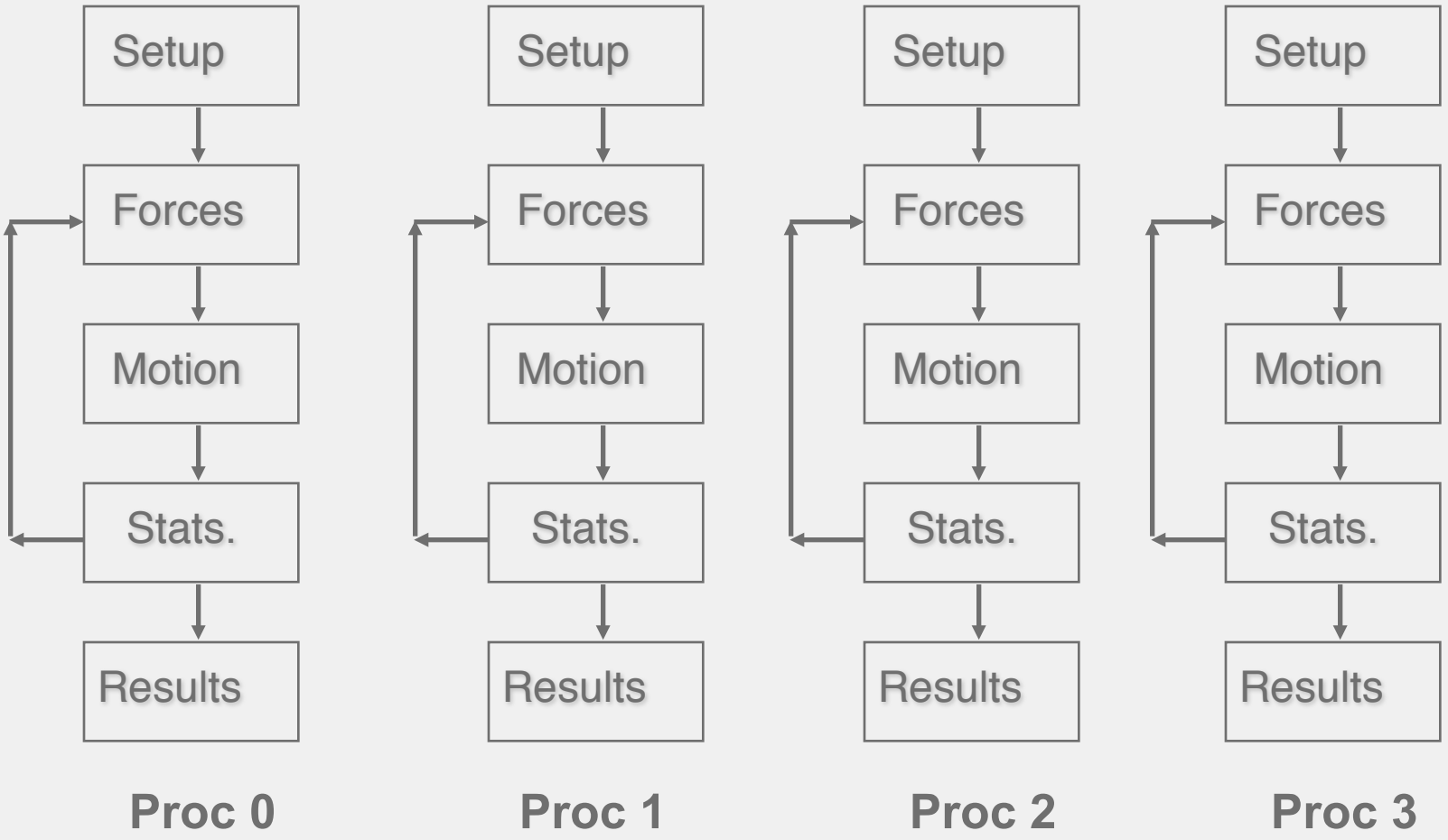
Reusing this material

Parallel Decompositions for MD

- Given P processes, how could the work be split up?

- Goals:
  - Achieve good load balance
    - Each processor takes an equal share of the work / time
    - Poor load balance limits scaling (similar to Amdahl's Law)
  - Reduce communication
    - Especially global communication e.g. Broadcast, gather
  - Asynchronous communication
    - If possible, do communication while other work is going on

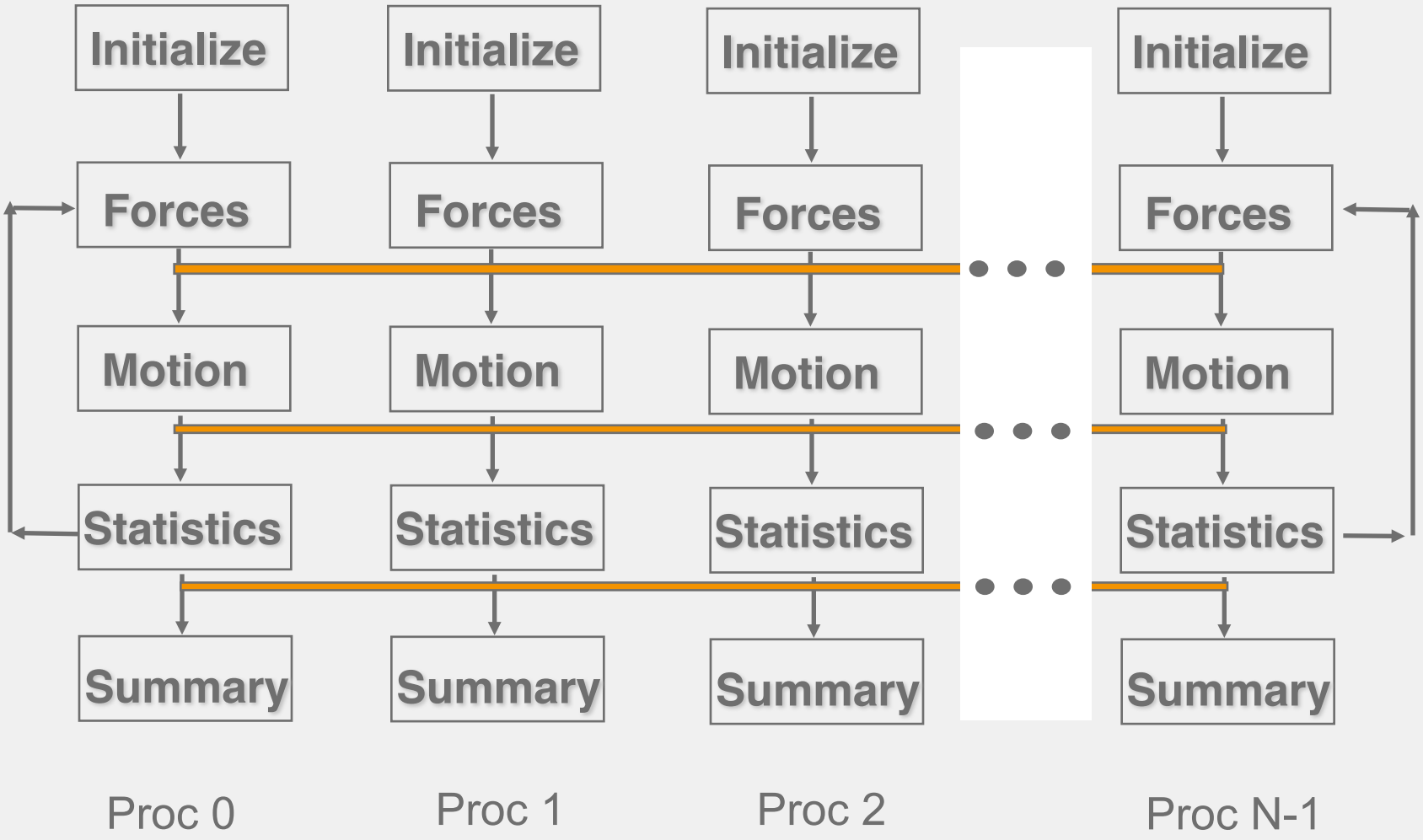
Parallel MD - Task farm?



Parallel MD - Task farm

- Advantages:
  - Simple to implement – no communications
  - Excellent load balance (assuming all systems are the same size)
  - 'Embarassingly parallel' – perfect scaling
- Disadvantages:
  - Only for replica / multiple walker sampling
  - Cannot reduce runtime per MD step – limit to short MD timescales

Parallel MD – Replicated Data?



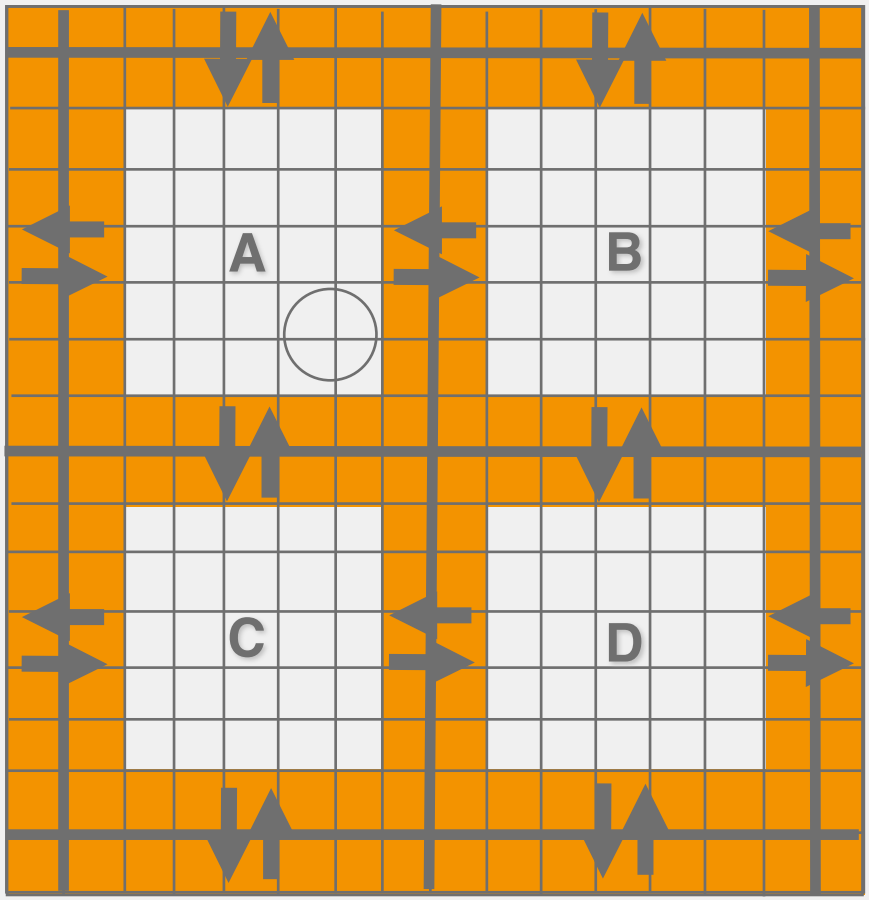
Parallel MD – Replicated Data

- Advantages:
  - Relatively simple to implement
  - Possible to achieve good load balance
    - Can decompose over particles, terms in the force field …
  - Works well with complex force-fields

- Disadvantages:
  - Global communication overhead
  - Leads to limited scalability
  - Requires large amount of memory in total
    - Every process stores all the particles

Parallel MD – Domain Decomposition

2D Example



- Short range potential cut off ($r_{cut} << L_{cell}$)
- Spatial decomposition of atoms into domains
- Map domains onto processors
- Use *link cells* in each domain
- Pass border link cells to adjacent processors
- Calculate forces, solve equations of motion
- Re-allocate atoms leaving domains

Parallel MD – Domain Decomposition

- Advantages:
  - Communication is mainly local (between neighbouring processes)
  - Possible to achieve good load balance
    - If system is isotropic
  - Memory is distributed over all processes
    - Allows large scaling
  - Enables bigger systems than can be handled by a single CPU (millions of atoms)
- Disadvantages:
  - Larger cut-offs lead to more communication
  - Implementation is more complex
- Examples:
  - GROMACS
  - DL_POLY 4
  - LAMMPS

Parallelisation in NAMD

- Workload is modelled as follows:
  - Local force computation ~ Na_p^2
    - All pairs of local atoms
  - Force computations between neighbouring patches ~ $w * Na_a * Na_b$
    - Weighting *w* depends on if patches share a corner, edge or face
  - Forces between patches are assigned to 'compute objects'
    - May be migrated freely between processors later
- Then at runtime, use dynamic load balancing to optimise the domain decomposition
  - Accounts for costs not covered by the model
  - Cope with changing system geometry during MD

Parallelisation in NAMD

- Workload metrics are recorded as follows:
  - Background load (non migratable work)
  - Idle time
  - Migratable compute objects and their associated compute load
  - The patches that compute objects depend upon
  - The home processor of each patch
  - The proxy patches required by each processor
- Load balancing heuristic
  - Move most expensive migratable object (compute objects) to least loaded processor, taking into account possible communication increases
  - Details in Kalé *et al*, LNCS 1457, 1998

MD Performance

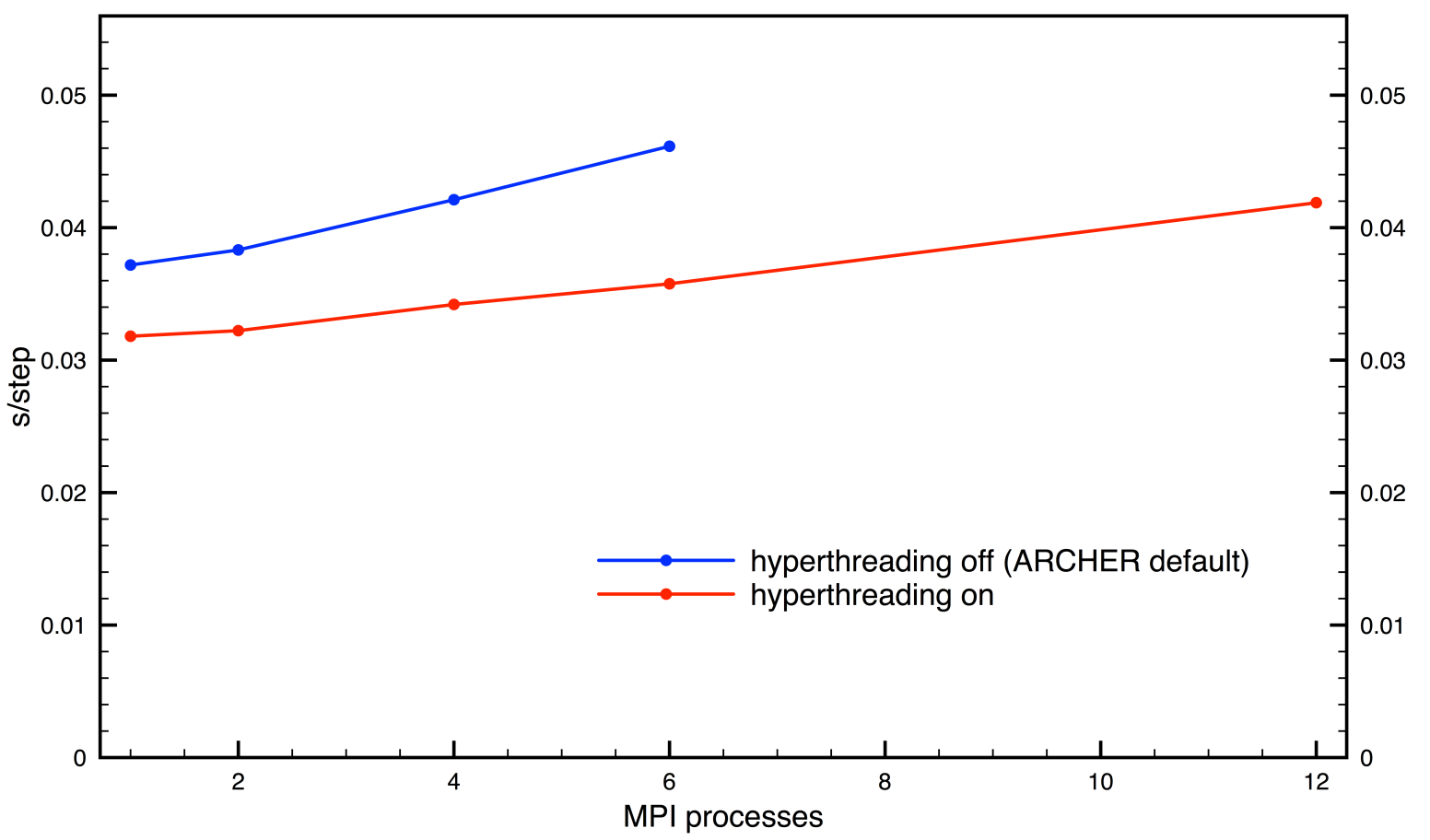
- Basic measure – wallclock time
  - Lower is better
- Application-specific measures
  - For MD, simulation time per wallclock time
    - ns / day
  - Higher is better
- Many ways to parallelise MD
  - All are a compromise between complexity and performance
  - 'Best' method depends on the system
    - e.g. in vacuo, solvated, solid state?
- Always run scaling tests before spending large amounts of compute time on long MD runs

MD Practical

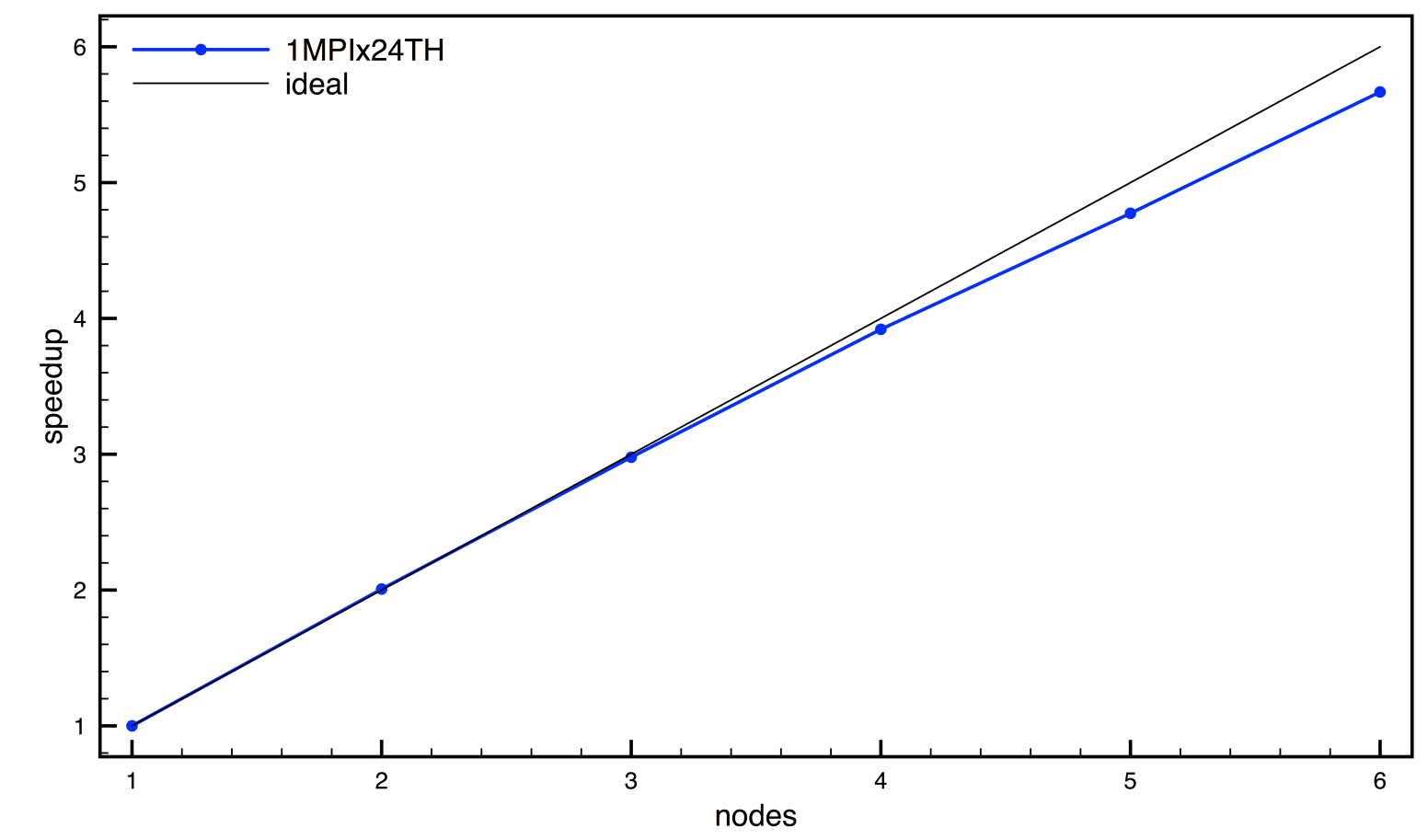
- Run NAMD simulation of Apolipoprotein A1
  - No need to compile, NAMD is already installed on ARCHER
- Investigate best way to run NAMD on a single node by experimenting with different numbers of MPI processes & OpenMP threads, e.g.:
  - 1 MPI x 24 OpenMP
  - 2 MPI x 12 OpenMP
  - 6 MPI x 4 OpenMP
  - 12 MPI x 2 OpenMP
  - 24 MPI x 1 OpenMP
- Find out whether hyperthreads help performance
- Using the fastest single-node option, investigate parallel scaling of NAMD to more nodes

Outcomes

NAMD single node walltime per step



NAMD multinode speedup



Process & threading placement no hyperthreading (-j 1 = default)

n	d	ppn	pemap	commap
1	24	23	1-23	0
2	12	11	1-11, 13-23	0,12
4	6	5	1-5, 7-11, 13-17, 19-23	0,6,12,18
6	4	3	1-3, 5-7, 9-11, 13-15, 17-19, 21-23	0,4,8,12,16,20

aprun –n $n –d $d namd2 +ppn $ppn +pemap $pemap +commap $commap

Process & threading placement with hyperthreading (-j 2)

n	d	ppn	pemap	commap
1	48	47	1-47	0
2	24	23	1-23, 25-47	0,24
4	12	11	1-11, 13-23, 25-35, 37-47	0,12,24,36
6	8	7	1-7, 9-15, 17-23, 25-31, 33-39, 41-47	0,8,16,24,32,40

aprun –n $n –d $d namd2 +ppn $ppn +pemap $pemap +commap $commap