GROMACS simulation optimisation

Olivier Fisette *olivier.fisette@usask.ca*
Advanced Research Computing, ICT
University of Saskatchewan
https://wiki.usask.ca/display/ARC/

WestGrid 2020 Summer School
https://wgschool.netlify.app/
2020-06-15

Presentation

- What is this session about?
  - Maximising the performance and throughput of MD simulations performed with GROMACS
  - Understanding how GROMACS accelerates and parallelises simulations
- Intended audience
  - You have already performed MD simulations with GROMACS.
  - You do not have a deep knowledge of GROMACS' architecture.
- The topics will be mostly technical rather than scientific, but the two cannot be separated entirely.
- The slides and a pre-recorded presentation are available online.
- An interactive Zoom session will be held at 11:00-13:00 PDT to allow attendees to ask their questions.

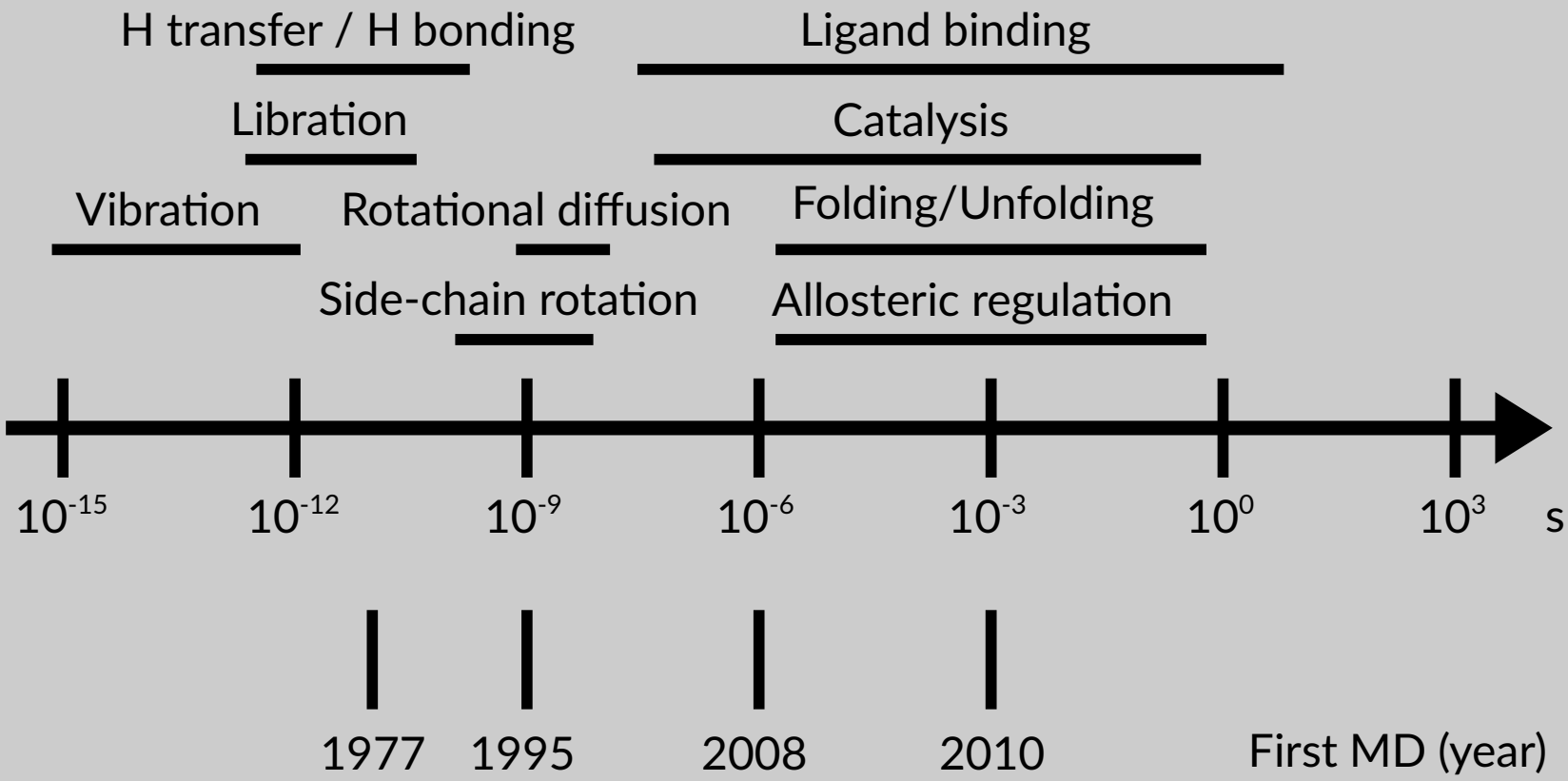
Contents

Motivation

- Why do we care about the performance of our MD simulations?
  - More simulation time means better sampling of biological events.

Motivation

- Why do we care about the performance of our MD simulations?
  - More simulation time means better sampling of biological events.



Biological event timescales

Motivation

- Why do we care about the performance of our MD simulations?
  - More simulation time means better sampling of biological events.
- How do make GROMACS faster?
  - We use several CPUs in parallel.
  - We use GPUs.
- When using CPUs in parallel, there is a loss of efficiency (e.g. doubling the number of CPUs does not always double the performance).
  1. How do we measure efficiency?
  2. Why does efficiency decrease?
  3. How do we avoid or limit loss of efficiency?
  4. How can we best configure our simulations to use multiple CPUs?

Speedup and efficiency

- Speedup (S) is the ratio of serial over parallel execution time (t)

$$S = \frac{t_{serial}}{t_{\text{parallel}}}$$

  - Example: running a program on a single CPU core takes 10 minutes to complete, but only 6 minutes when run on 2 cores; the speedup is 1.67.

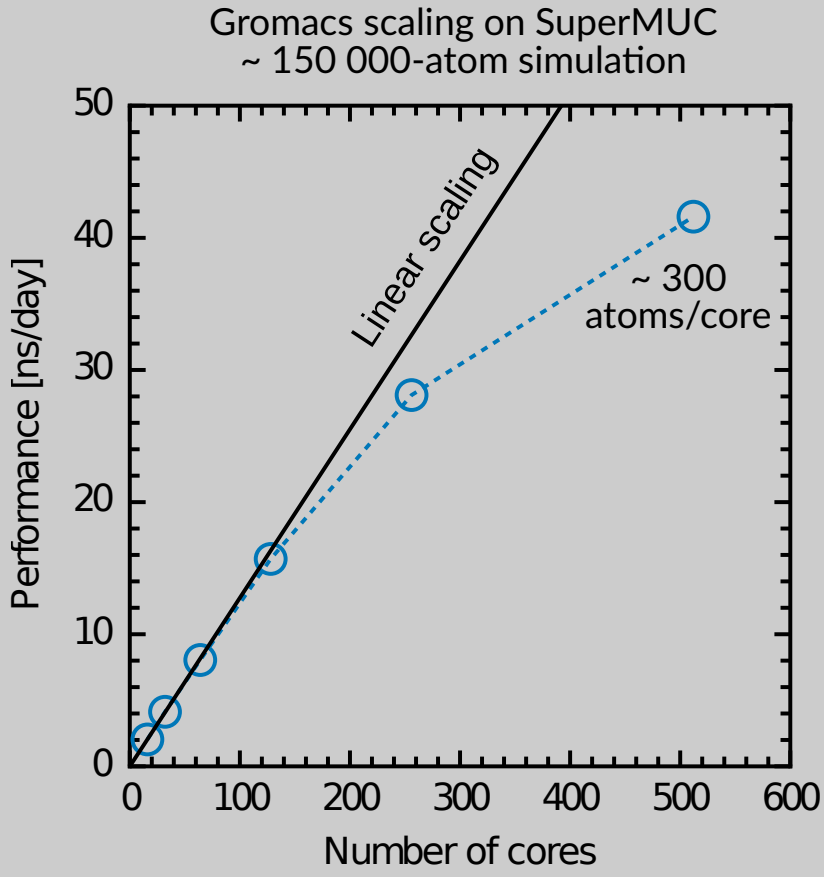
- Efficiency (η) is the ratio of speedup over number of parallel tasks (s)

$$\eta = \frac{S}{s} = \frac{t_{serial}}{t_{\text{parallel}}\, s}$$

  - Example: A 1.67 speedup on 2 cores yields an efficiency of 0.835, or 83.5 %.
  - When the speedup is equal to the number of parallel tasks (S = s), the efficiency is said to be linear (η = 1.0).

How well does GROMACS scale?

- Rule of the thumb: the scaling limit is ~100 atoms / CPU core.
  - At that point, adding more CPUs will not make your simulation go any faster.
  - Efficiency decreases long before that!
- Efficiency depends on system size, composition, and simulation parameters.
- To avoid wasting resources, you should measure scaling for each new molecular system and parameter set.



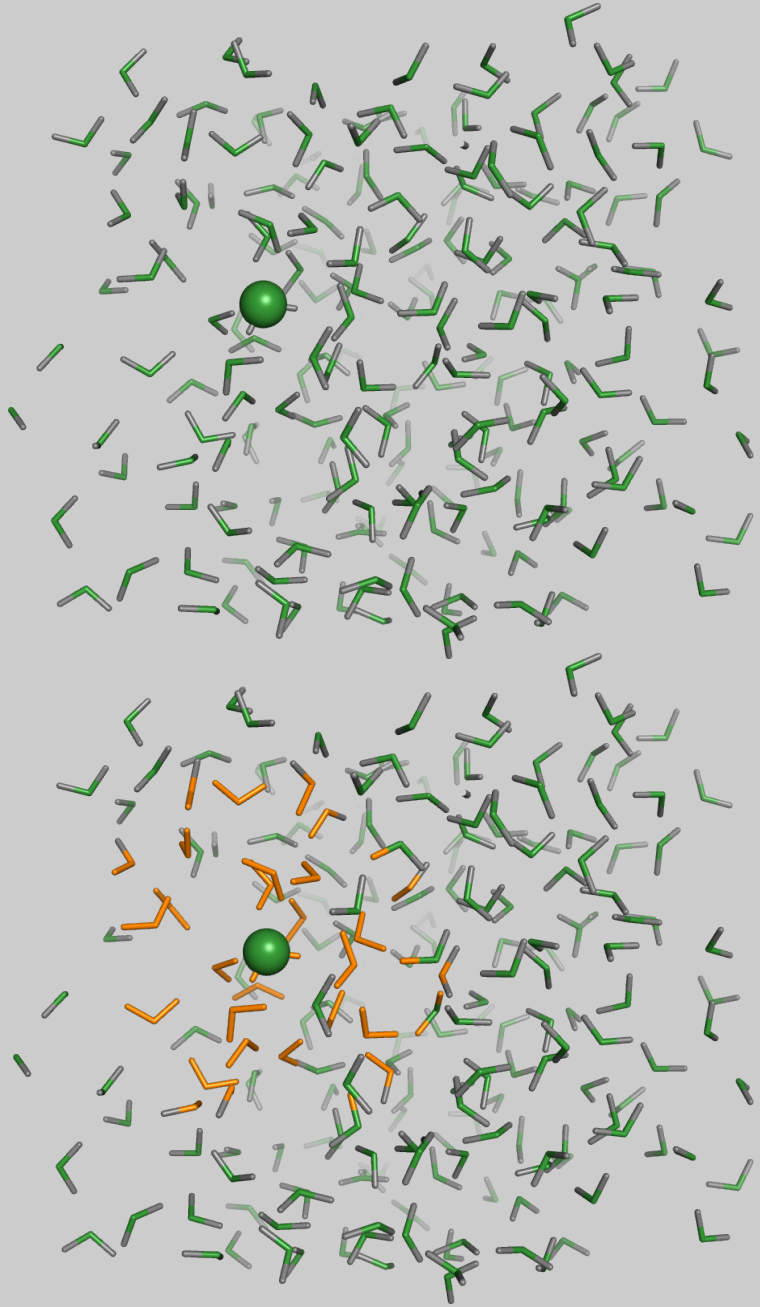
Why are MD simulations so computationally expensive?

- Most time in MD simulations is spent computing interatomic potentials from the force field.
- Non-bonded interactions are the bulk of the work.
  - Adding one atom to a 1000-atom system adds 0 to 3 new bonds.
  - Adding one atom to a 1000-atom system adds 1000 new non-bonded pairs!
  - Complexity grows quadratically with the number of atoms: $O(n^2)$
  - Clearly, this is not sustainable!

$$
\begin{aligned}
V = & \sum_{bonds} k_b (b - b_0)^2 \\
& + \sum_{angles} k_\theta (\theta - \theta_0)^2 \\
& + \sum_{dihedrals} k_\phi [1 + \cos(n\phi - \delta)] \\
& + \sum_{impropers} k_\omega (\omega - \omega_0)^2 \\
& + \sum_{VdW} \varepsilon \left[\left(\frac{r_{min}}{r} \right)^{12} - \left(\frac{r_{min}}{r} \right)^{6} \right] \\
& + \sum_{Coulomb} \frac{q_i q_j}{k_e r}
\end{aligned}
$$

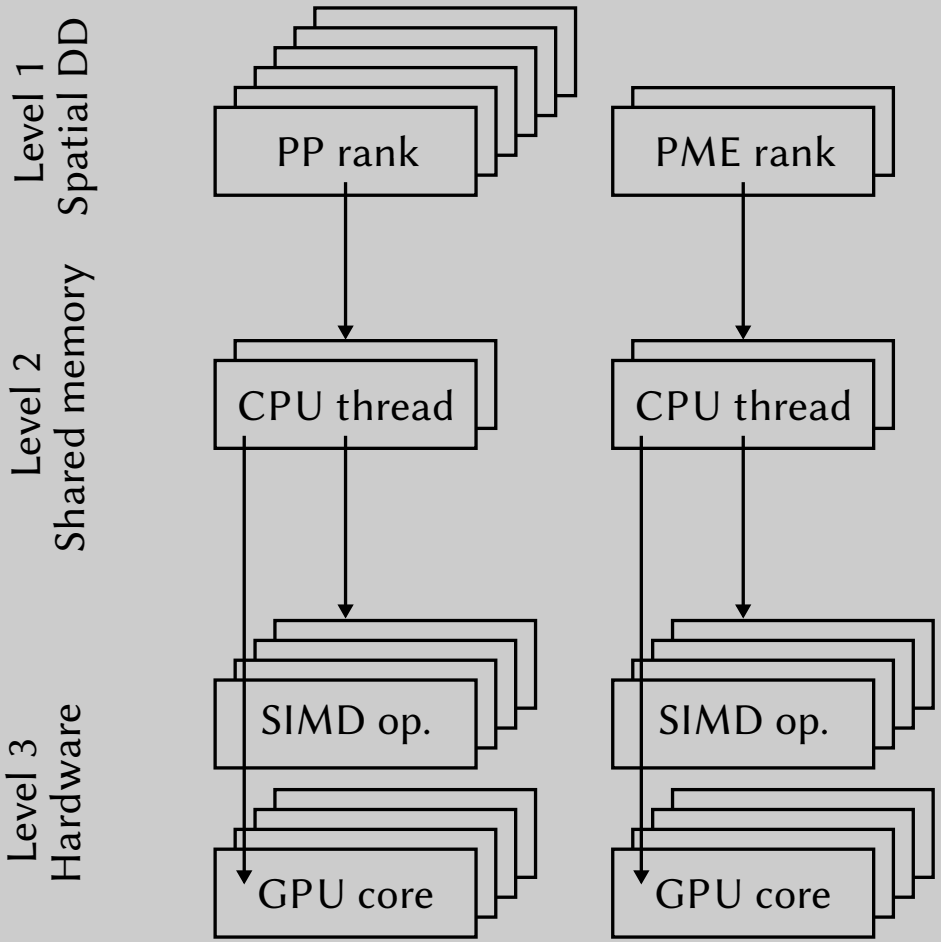
Neighbour lists make large simulations possible

- Only non-bonded interactions between atoms that are close are considered.
  - Potentials between atoms farther apart than a cut-off (e.g. 10 Å) are not computed.
- Long-range electrostatics are computed with Particle Mesh Ewald (PME).
- Neighbour lists are used to keep track of atoms in proximity.
  - These lists are updated as the simulation progresses.
  - GROMACS uses Verlet lists.
- Complexity becomes $O(n \log(n))$



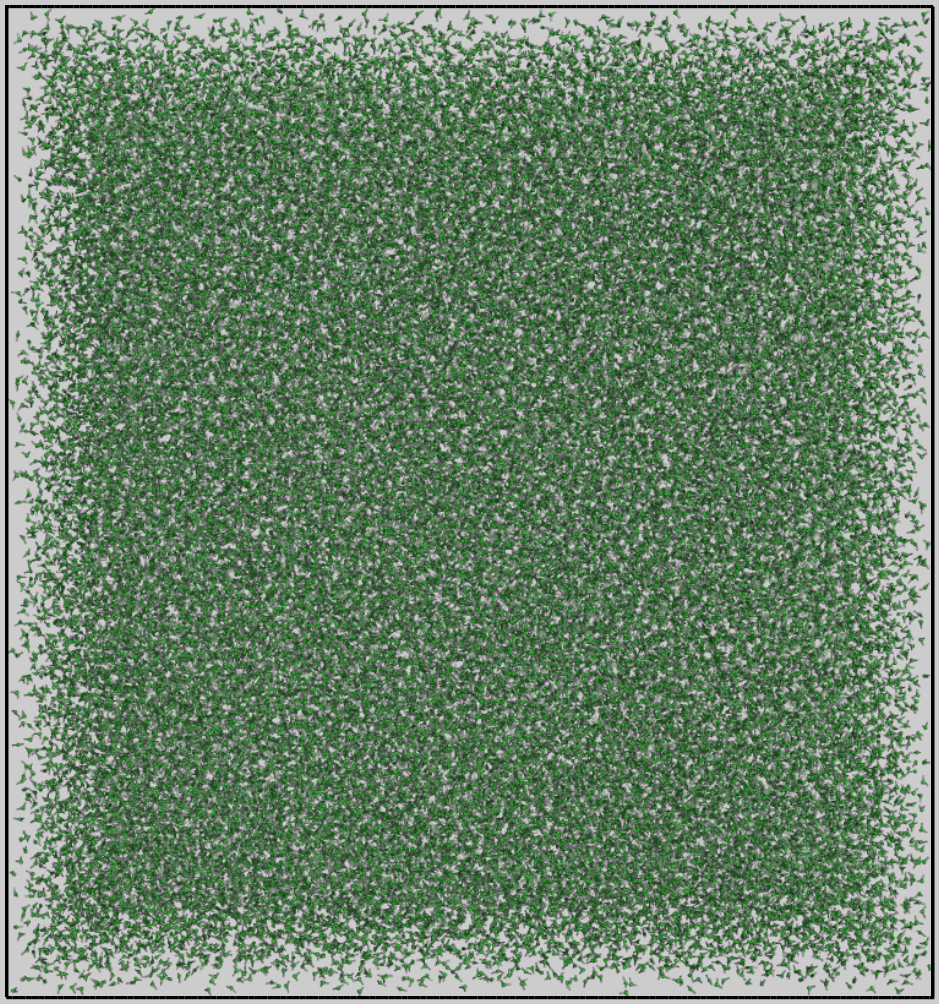
Overview of GROMACS parallelism

- GROMACS uses a three-level hybrid parallel approach.
  - All levels are independent.
  - All levels can be used together.
- This allows GROMACS to take full advantage of modern supercomputers and be very flexible at the same time.
- It requires the user to understand how the program works and to pay attention.



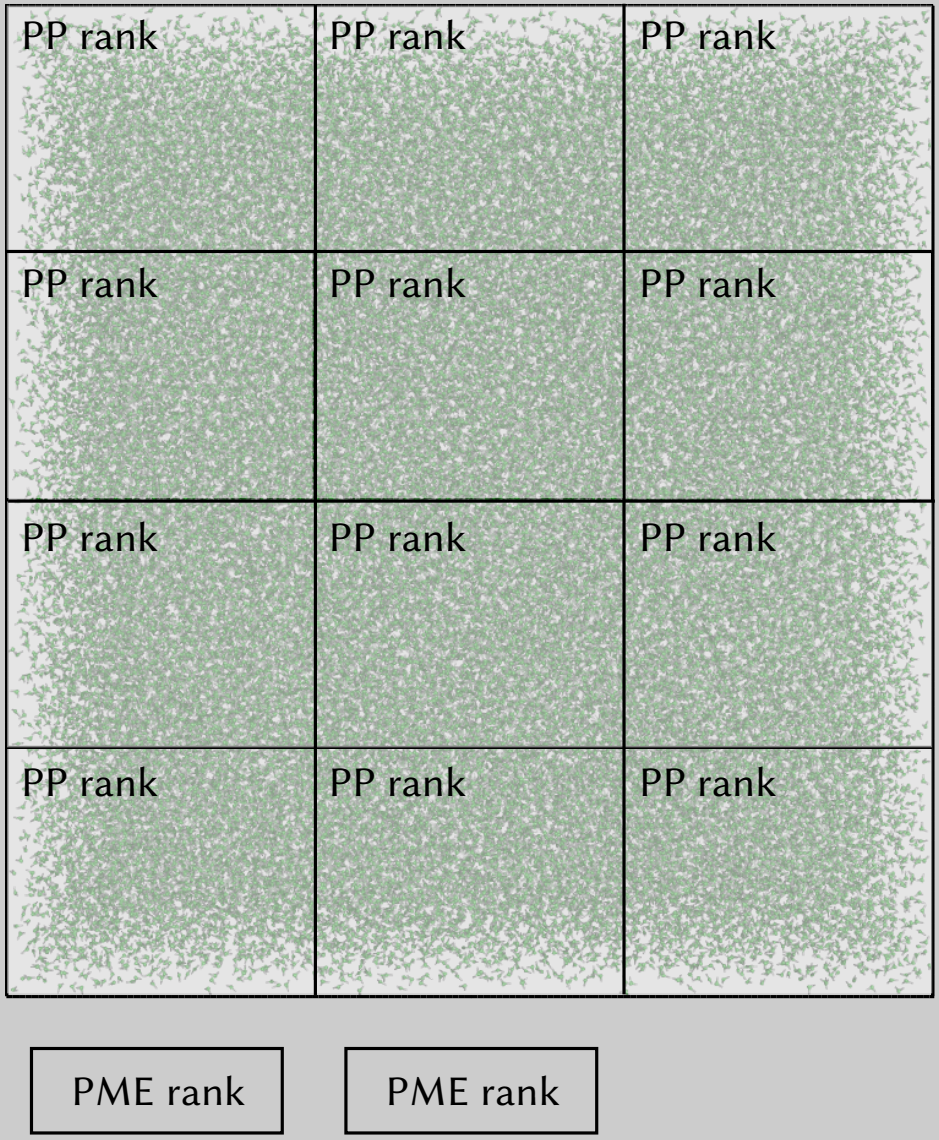
Spatial domain decomposition

- Let us consider a water box as our MD system.
- When performing an MD simulation on single CPU core, that core is responsible for all non-bonded potentials
  - Short-range interactions (using cut-offs and neighbour lists)
  - Long-range interactions (using PME)



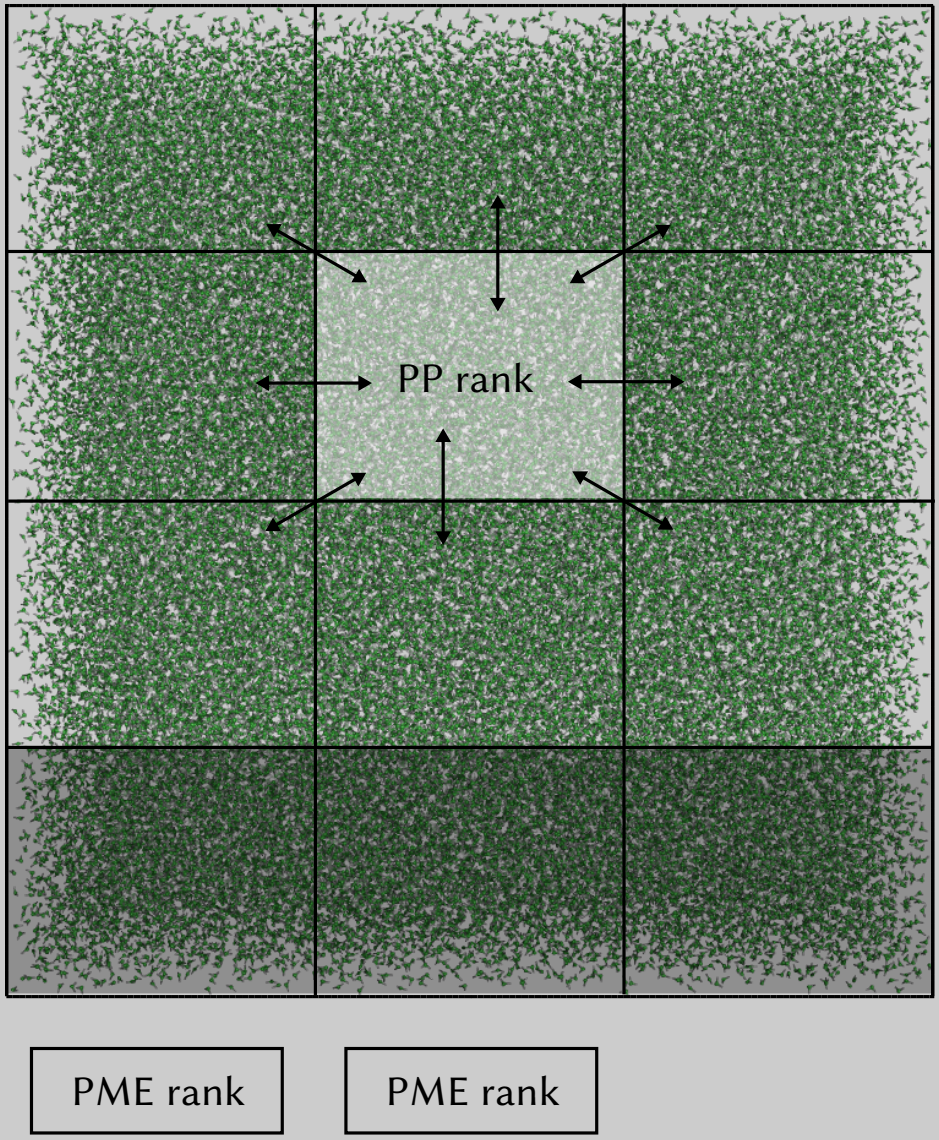
Spatial domain decomposition

- One strategy to use several CPU cores is to break up the system into smaller cells.
- GROMACS performs this domain decomposition (DD) using MPI.
- Some MPI ranks compute short-range particle-particle potentials (PP ranks).
- Other MPI ranks compute long-range electrostatics using PME (PME ranks).
- Domain decomposition can be performed in all three dimensions (2D case shown).



Spatial domain decomposition

- Each PP rank is responsible for a subset of atoms.
- Adjacent PP ranks need to exchange information
  - Potential between nearby atoms
  - Atoms that move from one cell to another
- Non-adjacent PP ranks do not exchange information
  - Communication is minimised
- GROMACS optimises the way cells are organised and the ranks between PP and PME automatically.



Spatial domain decomposition

Advantages

- Can distribute a simulation on many compute nodes
  - It is the only way to run a GROMACS simulation on several nodes.
- Performs very well for large systems (~1000 atoms per domain or more)
- Minimises the necessary memory per CPU
  - Better use of CPU cache.

Disadvantages

- Adds a significant overhead
  - Sometimes not worth it for single-node simulations
- Performs poorly for small systems
  - There is a limit to how small DD cells can be…
- Requires a fast network interconnect
  - InfiniBand and OmniPath are appropriate.
  - Ethernet is too slow.

Spatial domain decomposition

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32

# Using one full 32-core node

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

srun gmx_mpi mdrun
```

Spatial domain decomposition

```
#!/usr/bin/env bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=32

# Using two full 32-core nodes

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

srun gmx_mpi mdrun
```

Spatial domain decomposition

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8

# Using only 8 cores on a single node (very small
# systems may not scale well to a full node)

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

srun gmx_mpi mdrun
```

Spatial domain decomposition

```
#!/usr/bin/env bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8

# BAD: Using 2 nodes and 16 cores, 8 cores on each
# node. This will be slower than 16 cores on a
# single node. Always use full nodes in multi-node
# jobs.

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

srun gmx_mpi mdrun
```

Spatial domain decomposition

```
#!/usr/bin/env bash

#SBATCH --ntasks=32

# BAD: Using 32 CPU cores that could be spread on
# many nodes. Always specify the number of nodes
# explicitly.

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

srun gmx_mpi mdrun
```

Spatial domain decomposition

```
$ cat md.log
...
MPI library:        MPI
...
Running on 2 nodes with total 80 cores, 80 logical cores
  Cores per node:           40
...
Initializing Domain Decomposition on 80 ranks
Will use 64 particle-particle and 16 PME only ranks
Using 16 separate PME ranks, as guessed by mdrun
...
Using 80 MPI processes
...
NOTE: 11.1 % of the available CPU time was lost due to
load imbalance
...
NOTE: 16.0 % performance was lost because the PME ranks
had more work to do than the PP ranks.
...
```

Shared memory multiprocessing

- Let us consider a molecular system or a DD cell inside that system.
- Several CPU cores can work simultaneously to compute the inter-atomic potentials in that system or sub-system.
- All involved CPU cores need access to the same atom positions, i.e. the cores share access to the memory where positions are stored.
- GROMACS uses OpenMP threads for this shared memory parallelism.
- OpenMP threads can compute PP interactions, PME, or both.
- The system (or sub-system) is not split like it is with DD.

Shared memory multiprocessing

Advantages

- Small computational overhead
- Usually works well on a single node
- Performs better than MPI DD for small systems (less than 1000 atoms per core)

Disadvantages

- Uses more memory per core compared to MPI DD
  - Less efficient use of CPU cache
- Cannot distribute the workload over several compute nodes
  - But it can be used in conjunction with MPI DD across many nodes

Shared memory multiprocessing

```
#!/usr/bin/env bash

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=32

# Using one full 32-core node

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx mdrun
```

Shared memory multiprocessing

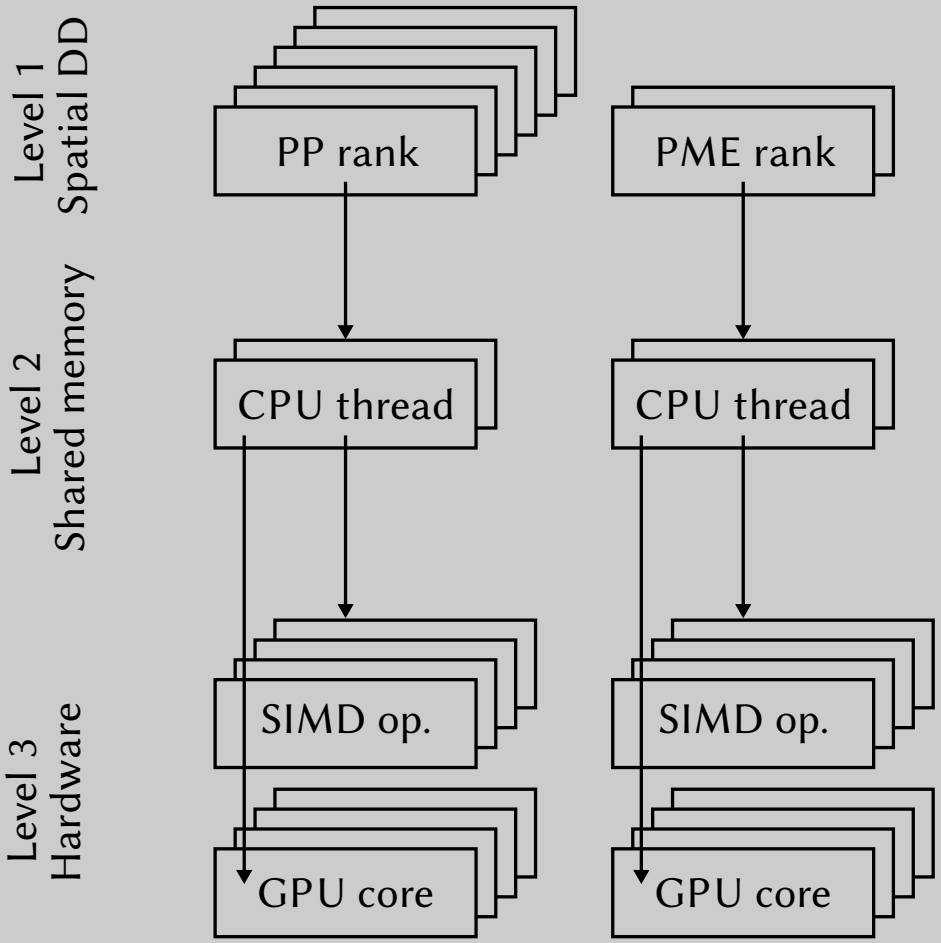
```
#!/usr/bin/env bash

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8

# Using only 8 cores on a single node (very small
# systems may not scale well to a full node)

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx mdrun
```

Shared memory multiprocessing

```
$ cat md.log
...
OpenMP support:    enabled (GMX_OPENMP_MAX_THREADS = 64)
...
Running on 1 node with total 40 cores, 40 logical cores
...
Using 40 OpenMP threads
...
```

Using both spatial DD and shared memory

- Spatial domain decomposition (MPI ranks) can be used in combination with shared memory multiprocessing (OpenMP threads).
- MPI is the “1st level” of parallelism, and “sits atop” OpenMP.
- Each MPI rank “controls” the same number of OpenMP threads.



Using both spatial DD and shared memory

Advantages

- Can use more CPU cores in parallel without performing more DD
  - Sometimes provides increased performance
  - DD cannot be done indefinitely

Disadvantages

- Slightly more complex to set up
  - The number of threads per rank has to be fine-tuned.
- Has the overhead of both methods
  - Sometimes not as fast as pure MPI or OpenMP

Using both spatial DD and shared memory

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --cpus-per-task=2

# Using one full 32-core node with 16 MPI ranks
# and 2 OpenMP threads per MPI rank

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx_mpi mdrun
```

Using both spatial DD and shared memory

```
#!/usr/bin/env bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
#SBATCH --cpus-per-task=2

# Using two full 32-core nodes with 32 MPI ranks
# and 2 OpenMP threads per MPI rank

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx_mpi mdrun
```

Using both spatial DD and shared memory

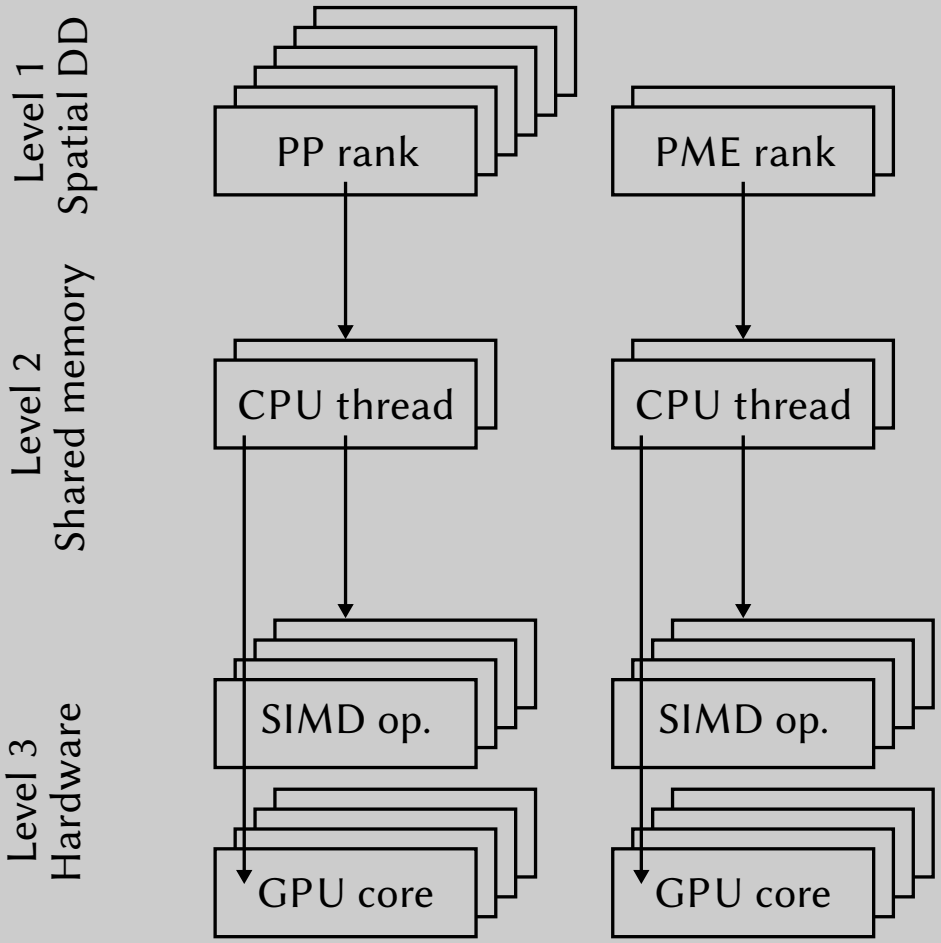
```
#!/usr/bin/env bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=32

# BAD: Using two full 32-core nodes with 2 MPI
# ranks and 32 OpenMP threads per MPI rank. The
# optimal number of threads per rank is usually
# between 2 and 6.

module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx_mpi mdrun
```

Using both spatial DD and shared memory

```
$ cat md.log
...
MPI library:        MPI
OpenMP support:     enabled (GMX_OPENMP_MAX_THREADS = 64)
...
Running on 2 nodes with total 80 cores, 80 logical cores
  Cores per node:           40
...
The number of OpenMP threads was set by environment variable
OMP_NUM_THREADS to 2
...
Initializing Domain Decomposition on 40 ranks
Will use 32 particle-particle and 8 PME only ranks
Using 8 separate PME ranks, as guessed by mdrun
...
Using 40 MPI processes
Using 2 OpenMP threads per MPI process
...
NOTE: 5.5 % of the available CPU time was lost due to load
imbalance
...
NOTE: 34.1 % performance was lost because the PME ranks
had more work to do than the PP ranks.
...
```

Hardware-based acceleration

- Modern CPUs are able to apply the same operations to multiple data points simultaneously, using specialised hardware.
- This is known as “single instruction, multiple data” (SIMD).
- Intel CPUs support the AVX, AVX2 and AVX512 instruction sets that allow programmers to achieve this hardware-level parallelism.
- GROMACS has code to compute short-range potentials using AVX, AVX2, AVX512 and similar technologies.



Hardware-based acceleration

- GROMACS routines for hardware acceleration must be chosen at compilation time.
- On CC clusters, GROMACS is already optimised for you.
- There is nothing special for you to do at run-time.

```
$ cat md.log
SIMD instructions:  AVX_512
...
Number of AVX-512 FMA units: 2
...
Highest SIMD level requested by all nodes in run:
AVX_512
SIMD instructions selected at compile time:
AVX_256
This program was compiled for different hardware than
you are running on,
...
```

Hardware-based acceleration

- AVX support on the CC clusters
  - Béluga
    - All compute nodes support AVX512
    - The default software stack is built for AVX512
  - Cedar and Graham
    - All compute nodes support AVX2 (Broadwell)
    - Some nodes also support AVX512 (Skylake, Cascade Lake)
    - The default software stack is built for AVX2
    - The AVX512 software stack can be loaded manually
- Tests performed with GROMACS on Skylake and Cascade Lake nodes show a 20–30 % performance increase when using the AVX512 software stack.
- You can ask for AVX512-capable nodes, but your wait time in the queue will likely be longer.

Hardware-based acceleration

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --constraint=skylake|cascade

module load arch/avx512
module load gcc/7.3.0
module load openmpi/3.1.2
module load gromacs/2020.2

srun gmx_mpi mdrun
```

Optimising a GROMACS simulation in practice

- Make short tests (~10 ps, adjust to get at least 5 minutes of runtime).

```
dt       =  0.002  ;   2 fs
nsteps   =  5000   ;  10 ps
```

- Deactivate output to avoid I/O skewing the results.

```
nstxout-compressed  =  0
nstlog              =  0
nstenergy           =  0
```

- Get the performance from the log file (in ns/day).
- Repeat all tests at least three times.
  - Use the average performance.
  - Verify that the deviation between the runs is small.

Optimising a GROMACS simulation in practice

1. Start with a serial run (single CPU core).

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
```

2. Increase the number of cores progressively (2, 4, 8, 16…) until you occupy all cores on the node.

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
```

3. Compute the speed-up for each configuration.

$$S = \frac{t_{serial}}{t_{\text{parallel}}} = \frac{p_{\text{parallel}}}{p_{serial}}$$

4. Compute the efficiency for each configuration.

$$\eta = \frac{S}{s} = t_{serial}$$

Optimising a GROMACS simulation in practice

- If efficiency is still acceptable on a full node:
  1. Increase the number of nodes progressively (2, 4, 8, 16) until efficiency becomes low.

     ```
     #SBATCH --nodes=2
     #SBATCH --ntasks-per-node=32
     ```

  2. Once you have chosen an optimal number of nodes, try using OpenMP threads in combination with MPI.

     ```
     #SBATCH --nodes=4
     #SBATCH --ntasks-per-node=16
     #SBATCH --cpus-per-task=2
     ```

- If efficiency is not acceptable on a full node, repeat your tests using OpenMP instead of MPI.

Using GPUs with GROMACS

- GROMACS can use GPUs to accelerate certains operations such as evaluating short-range non-bonded interactions.
  - Much like SIMD on CPUs
- Because GPUs are massively parallel, they are well suited to MD simulations and can be faster than CPUs, especially on single-node jobs.
- However, GROMACS has excellent CPU performance and using multiple GPUs, especially on several nodes, is often not faster than using only CPUs.
- GPUs offer massively increased *throughput*, i.e. the total amount of simulation time you can perform, but they do not always increase the speed of a single simulation, i.e. *performance*.
- GROMACS ties each GPU to an MPI rank, i.e. there should be as many MPI ranks as GPUs. Each MPI rank can still make use of OpenMP threads.

Using GPUs with GROMACS

```
#!/usr/bin/env bash

#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:v100l:1

# Using 1/4 of the cores and one of the 4 GPUs
# on a single node (Cedar V100L GPUs).

module load gcc/7.3.0
module load cuda/10.0.130
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx mdrun
```

Using GPUs with GROMACS

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:v100l:4

# Using all cores and 4 GPUs on a single node
# (Cedar V100L GPUs).

module load gcc/7.3.0
module load cuda/10.0.130
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx_mpi mdrun
```

Using GPUs with GROMACS

```
#!/usr/bin/env bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:v100l:4

# Using all cores and 8 GPUs on two nodes
# (Cedar V100L GPUs).

module load gcc/7.3.0
module load cuda/10.0.130
module load openmpi/3.1.2
module load gromacs/2020.2

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun gmx_mpi mdrun
```

Using GPUs with GROMACS

```
$ cat md.log
...
GPU support:          CUDA
...
Running on 2 nodes with total 64 cores, 64 logical
cores, 8 compatible GPUs
...
On host cdr2546.int.cedar.computecanada.ca 4 GPUs
selected for this run.
...
Using 8 MPI processes
...
Using 8 OpenMP threads per MPI process
...
```

Optimising a GROMACS simulation on GPUs in practice

1. Start with a single GPU and the corresponding number of cores. (Get the number of cores by dividing the total number of cores by the total number of GPUs on a node.)
2. Increase the number GPUs and cores until you use all GPUs on a node.
3. Try using multiple nodes.

Tuning non-bonded interactions

- In modern MD simulations, non-bonded interactions are typically split into two parts:
  - VdW and short-range electrostatics, computed in real space (GROMACS uses optimised CPU SIMD or GPU routines)
  - Long-range electrostatics, computed in reciprocal space using Particle Mesh Eward (GROMACS uses an optimised FFT library)
- By changing cut-offs and grid spacing, the balance between these two can be tuned. Longer cut-offs and a larger grid spacing mean more short-range work, while shorter cut-offs and a smaller grid spacing mean more long-range work.
- GROMACS balances short- and long-range interactions automatically. It is not necessary or useful to define cut-offs or the grid spacing.
- Verlet lists do not need to be updated often (`nstlist` parameter). GROMACS ensures their accuracy dynamically. A large `nstlist` is important with GPUs.

Tuning non-bonded interactions

```
$ cat grompp.mdp
...
; Non-bonded parameters
PBC              =  XYZ
cutoff-scheme    =  Verlet
nstlist          =  50
Coulombtype      =  PME
VdWtype          =  cut-off
VdW-modifier     =  potential-shift-Verlet
dispcorr         =  enerpres
...
```

Integrator frequency

- The fastest motions in an MD simulation are usually X–H vibrations.
  - 10-fs timescale
  - Require a 1-fs integrator
- Usually, we constrain X–H bonds to remove these motions.
  - Allows for a 2-fs integrator
  - We use rigid water models (TIP, SPC) anyway.
- The integrator can even be increased to 4 fs. In such schemes, all bonds are rigid, and there is a trade-off between speed and accuracy.
  - Virtual sites for hydrogens
  - Mass repartitioning
  - United-atom force fields
  - RESPA integrators

Concluding remarks

- Always read your log files: GROMACS is very informative…
- Optimise every time you make a new system, unless the system is very similar to another you already optimised.
- Multiple smaller trajectories are easier and faster to acquire than a single long one.
  - What motions are you interested in?
  - A single call to `gmx mdrun` can run multiple simulations on nearly arbitrary resources, including several simulations on a single GPU (see the `-multidir` option).
  - Replica exchange MD (REMD/REST) can accelerate sampling using multiple “replica” simulations instead of a single longer simulation.
- Read the release notes when changing GROMACS version. Despite being mature software, GROMACS is still in active development.

References

- CC Doc: GROMACS

  https://docs.computecanada.ca/wiki/GROMACS

- GROMACS documentation

  http://manual.gromacs.org/

- Annex: MDP file for recent GROMACS versions

- S. Páll M. Abraham, C. Kutzner, B. Hess, E. Lindahl. Tackling Exascale Software Challenges in Molecular Dynamics Simulations with GROMACS. *Solving Software Challenges for Exascale*, Springer International Publishing, 2015, 8759, 3–27.

- C. Kutzner, S. Páll, M. Fechner, A Esztermann, B.L. de Groot, H. Grubmüller. Best bang for your buck: GPU nodes for GROMACS biomolecular simulations. *J. Comput. Chem.*, 2015, 36(26): 1990–2008.

Annex: Example `grompp.mdp` for recent GROMACS

```
; Output control
nstxout-compressed  =  5000  ; 10 ps
nstlog              =  5000
nstenergy           =  5000

; Integrator settings
integrator  =  md
tinit       =  0
dt          =  0.002       ; 2 fs
nsteps      =  500000000   ; 1 us

; Bonded parameters
constraints  =  h-bonds
LINCS-iter   =  1
LINCS-order  =  4
```

Annex: Example `grompp.mdp` for recent GROMACS

```
; Non-bonded parameters
PBC              =  XYZ
cutoff-scheme    =  Verlet
nstlist          =  50
Coulombtype      =  PME
VdWtype          =  cut-off
VdW-modifier     =  potential-shift-Verlet
dispcorr         =  enerpres

; Temperature coupling
tcoupl   =  v-rescale
tc-grps  =  Protein Water_and_ions
tau-t    =  0.1 0.1
ref-t    =  300 300
```

Annex: Example `grompp.mdp` for recent GROMACS

```
; Pressure coupling
pcoupl              =  Berendsen
tau-p               =  2.0
pcoupltype          =  isotropic
compressibility     =  4.5e-5
ref-p               =  1.0

; (Re)Generate Velocities
gen-vel   =  yes
gen-seed  =  1
gen-temp  =  300
```