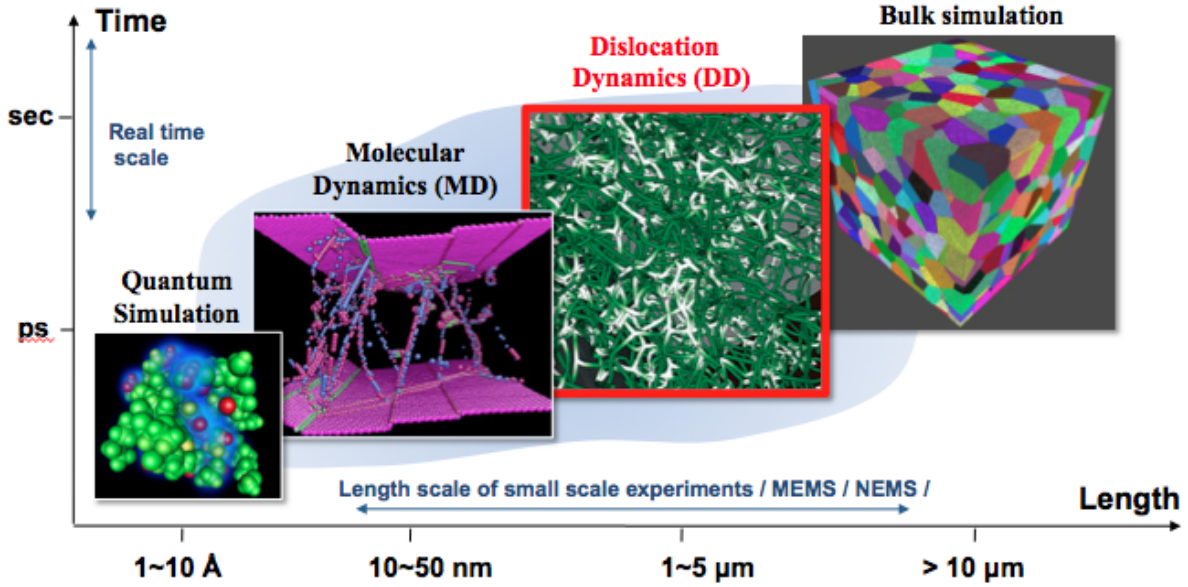

Algorithms Optimization in GROMACS
Co Quach
SC3260

Molecular Dynamics Simulation

- Molecular Dynamics (MD) Simulation is an important tool for different research fields.
- These simulations can be used to:
  - Gain more understanding about processes at molecular level
  - Predict properties
  - Validate theories

MD Simulation is expensive

- However, these simulations are very computationally expensive, and is the main restriction to computational scientists.
- The scale of MD simulation is much smaller comparing to real system.
  - System size around ten of thousands of atoms
  - Time scale around a few nanosecond

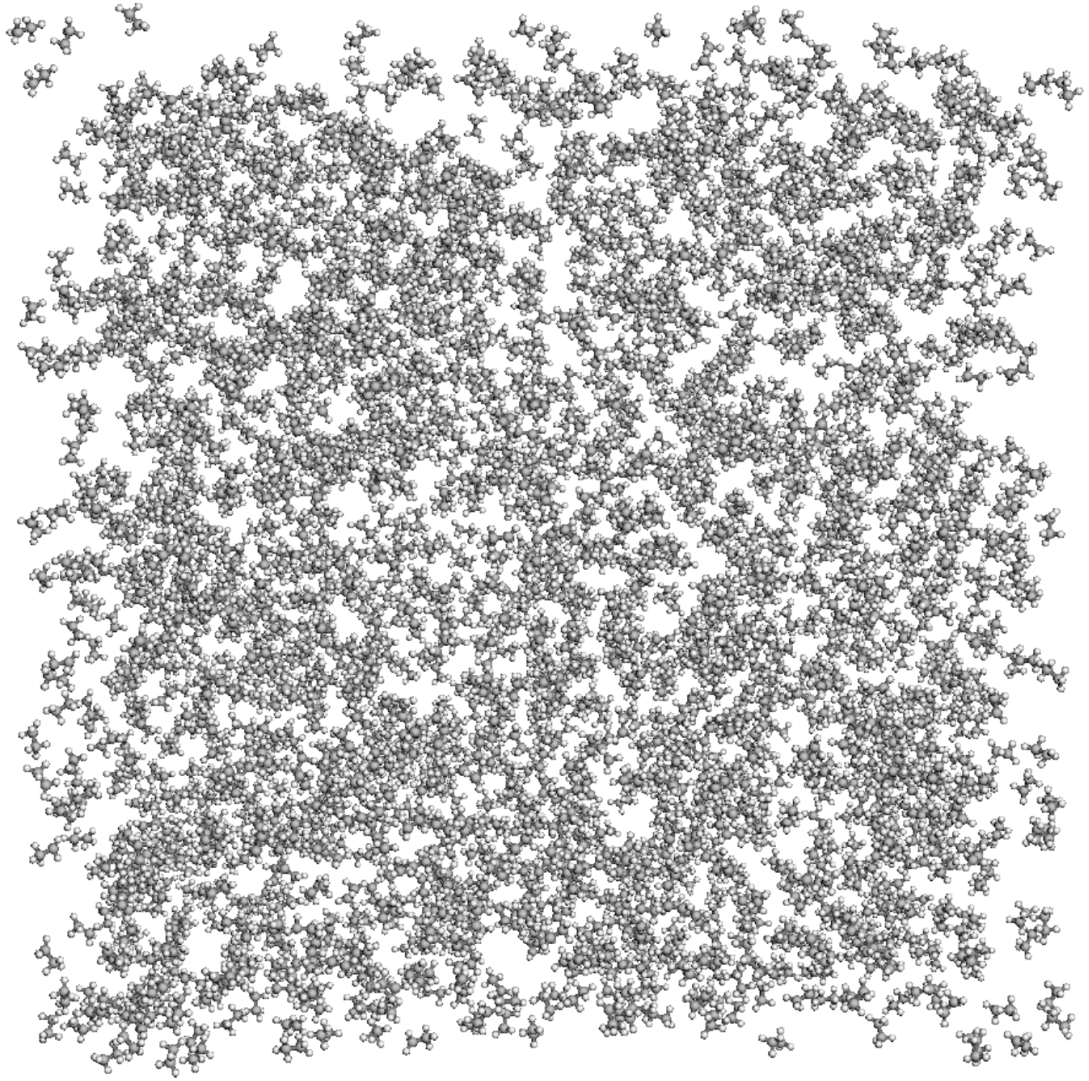


Time scale vs Length scale of multi-scale computational techniques

Figure obtained from: https://sites.google.com/site/seokwoolab/research/dislocation-dynamics

MD Simulation is expensive

- For example, a box of 2,500 Ethane molecules have (4.15e-21 mol) :
  - 20,000 atoms
  - 17,500 bonds
  - 30,000 angles
  - 22,500 torsions
- (And this is very very small comparing to a real system)
- Visualization of a box of ethane



A box of 2,500 Ethane

MD Simulation is expensive

- To simulate an equilibration of such system for 10ns, with a 2fs time step, we will need to run 5 million steps. Each step, we will need to perform:
  - Potential energy:
    - Nonbonded potential: base on pairwise interaction:
      - Lennard Jones or Buckingham potential equation
      - Coulomb electrostatic potential
    - Bond potentials
    - Angle potentials
    - Dihedrals potentials
  - Dynamic energy:
    - Atom velocities
- Repeat that 5 million times
- Hence, having an optimized, both parallelly and linearly is highly important for the success of the field.

GROMACS

GROMACS
fast, flexible & free

- GROningen MAchine for Chemical Simulations (GROMACS) is a MD simulation code developed mainly to simulate proteins, lipid, and fatty acid.
- The project was started in 1991, with first paper published in 1995.
- Six more publications have been published marking the development of the project.
- Since, the code has been widely adopted and used across the field.

State of the community

- Before the first GROMACS toolkits were first published in 1991 and later widely adopted:
  - Ad Hoc codes which are usually written internally in a lab for specific purpose:
    - Hardly extendable
    - Usually not optimized (linearly and parallelly)
    - Might or might not have parallel portions
  - Besides the performant issues:
    - Not always reproducible:
      - Code is not always readable + lack of documentation
      - Bugs might be buried deep in the code base
    - Always prone to bugs due to lag of external review

GROMACS

- The development of GROMACS provide the community with:
  - Access to reliable and performant code
  - The option to run simulations linearly or parallelly, on a laptop or on a supercomputer
  - Tools to analyze simulation data
  - The ease to report details of simulation in publication (since most code-specific details can be referred to GROMACS documentation)

References on last page

GROMACS

- Summary of a GROMACS simulation step:
  - Compute accelerations from all the potential forces information
  - Update and scale velocities
  - Compute new coordinates
  - Attempt to move atoms to new coordinates
  - Correct velocities for constraints (due to atoms overlaps, etc.)
  - Scale coordinates and box

Berendsen, et al. (1995) *Comp. Phys. Comm.* 91: 43-56. (DOI, Citations of this paper)

Earliest version

GROMACS
fast, flexible & free

- First published version (in 1995 paper):
  - Utilize parallel message passing method instead of data-parallel (since this is still in early development stage)
  - Specifically designed for ring architecture
  - Particle decomposition scheme: each processors handle a set of particles, particles' coordinates and partial forces are shared around half the ring once per time step
- Main design choice motivation:
  - Straightforwardness and simplicity

Berendsen, et al. (1995) *Comp. Phys. Comm.* 91: 43-56. (DOI, Citations of this paper)

Earliest version

GROMACS
fast, flexible & free

- Details of the algorithm, taking from the 1995 paper
  1. Starting from its home processor, the position of each particle i is distributed anticlockwise over half the ring
  2. If required (e.g. once every 20 time-steps), a neighbor list is constructed and ‘shell forces’ from particles between R_short and R_long is computed
  3. All forces are are computed and partial sums of forces accumulated in each processor.
  4. Partial sums of forces on every particle i are communicated, going clockwise direction over half the ring. The forces are summed to net forces in each home processor
  5. Using the net forces, velocities and positions are updated for every particle on its home processor.

Berendsen, et al. (1995) *Comp. Phys. Comm.* 91: 43-56. (DOI, Citations of this paper)

Earliest version

GROMACS

- GROMACS used a primitive version of message passing interface routines, include six steps and were demonstrated in the 1995 paper

`void network_tx(int chan,void *buf,int bufsize)`
This routine asynchronously sends `bufsize` bytes from the buffer pointed to by the pointer `buf` over the communication channel. The channel is identified by `chan`.
`void network_tx_wait(int chan)`
This routine implements a wait function until the asynchronous send operation associated with the communications channel designated by `chan` has ended.
`void network_txs(int chan,void *buf,int bufsize)`
This routine synchronously sends `bufsize` bytes from the buffer pointed to by the pointer `buf` to the processor or the process identified by `chan`.
`void network_rx(int chan,void *buf,int bufsize)`
This routine asynchronously receives bufsize bytes in the buffer pointed to by te pointer `buf` from an communication channel identified by `chan`.
`void network_rx_wait(int chan)`
This routine implements a wait function until the asynchronous receive operation, associated with the communications channel designated by `chan`.
`void network_rxs(int chan,void *buf,int bufsize)`
This routine asynchronously receives `bufsize` bytes from the buffer pointed to by the pointer `buf` over the communication channel identified by `chan`.
`void network_init(int pid,int nprocs)`
This call initialises the network. On various machines one needs to for example define the network communications and open the channels for use. This is being done in this routine. It sets the variable `pid` to a unique id for the processor it is executed on, and sets `nprocs` to the number of processors allocated for this particular run.
`void get_left_right(int nprocs,int pid,int *left,int *right)`
This routine maps the internal representation of the variables `right` and `left` to the processor numbers of the processors that are virtually located on the 'left' and on the 'right' of the calling processor. The GROMACS program assumes a **ring** communication topology. Therefore it will only call the communication routines with a `chan` value of either `left` or `right`.

The primitives message passing interface routines in GROMACS, extracted from the 1995 paper

Berendsen, et al. (1995) *Comp. Phys. Comm.* 91: 43-56. (DOI, Citations of this paper)

Earliest version

GROMACS
fast, flexible & free

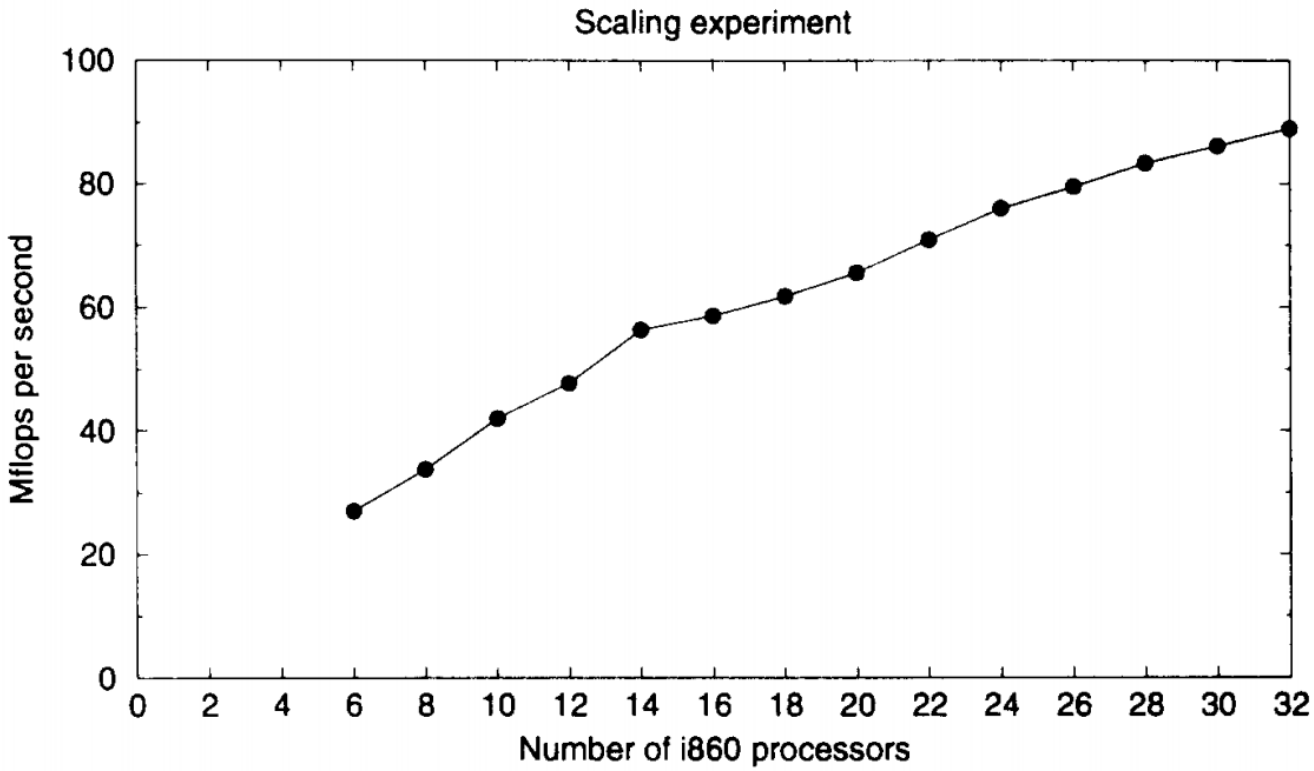


Fig. 2. Scaling performance for peptide in water.

Berendsen, et al. (1995) *Comp. Phys. Comm.* 91: 43-56. (DOI, Citations of this paper)

Progression

- Over the years, GROMACS's developers and maintainers have been updating the package to integrate more efficient algorithms, as well as to cope with the hardware development.
- Six core papers have been published to report about the development progress of this tool-kit. Each features new method addition to toolkit as well as more codes optimization.
- In the next page, there is a list of all the optimizations that have been mentioned in their core papers.
- Among those, I will go into details how a few of those are implemented.

References on last page

Algorithm Optimization

- Optimization of the virial summation: turn an inner loop to a single sum over particles (2001)
- Introduce new routine to calculate r^-1 from r^2 without having to do the expensive square root calculation (2001)
- Started integrating SIMD instructions instead of the original message passing parallel (2001)
- Introduce shift concept for nonbonded interaction to reused and reduced number of calculation need to be done (2005)
- Specialized kernels to perform nonbonded interaction to avoid making unnecessary calculation (2005)
- Domain decomposition and dynamics load balancing, improve parallel scaling (2008)
- Optimize memory access by introduce new sorting algorithm (2008)
- Optionally support MPMD parallelization (PME processors) (2008)
- Replace ring communication topology with more sophisticated one to improve both performance and memory utilization (2008) (PME stands for Particle Mesh Ewald)
- Better PME decomposition (2013) (one of the main bottle neck of parallel performance issue)
- GROMACS thread_MPI scaling behavior is near-identical to OpenMPI (2013)
- GPU accelerator implementation (2015)
- Heterogenous parallelization using both CPU and GPU (2015)
- Multi-level Parallelism scheme (2015)

References on last page

Serialize Virial Sum

GROMACS
fast, flexible & free

- Virial equation: $$\Xi = -\frac{1}{2}\sum_{i<j}^{N} \boldsymbol{r}_{ij}^{n} \otimes \boldsymbol{F}_{ij}$$

  Where r_ij denotes the distance vector of the nearest images of atom i from atom j.

- This has been traditionally done by iterating over i and j to calculate the sum.
- The GROMACS developed a new algorithm to simplify the inner loop (so only need to iterate over i), by replacing it with a single sum. The new implementation increases their performance by roughly 40%.
- This implementation is non-trivial, so I do not plan to go into greater details. More information for this transform can be found at in their paper: https://www.rug.nl/research/portal/files/3251563/c2.pdf

Lindahl, et al. (2001) *J. Mol. Model.* 7: 306-317. (DOI, Citations of this paper)

Square root work around

GROMACS
fast, flexible & free

- In potential force calculation, we need to obtain the value for 1/r, starting from r^2. This is originally done by taking the square root of r^2 and then doing the invert function. This, back in the time, can be quite expensive (mainly due to taking the square root). Hence, they found a walk around (an approximation):

$$\frac{1}{r} = a(3 - ar^2)/2$$

- The value of a is stored in a table in the memory space.
- This implementation doubles performance on processors without hardware square-root instruction

Lindahl, et al. (2001) *J. Mol. Model.* 7: 306-317. (DOI, Citations of this paper)

SIMD Implementation

- This is the earliest attempt of GROMACS to implement SIMD algorithm.
- The implementation, although still quite primitive, make use of the new processor architecture and be able to significantly speed up the simulation.
- At this point, what GROMACS did was parallelize the inner loop when calculating different pairwise interaction.
  - In the usual double for loop iterating over i and j, (with i and j marking atom i and j), the inside loop was parallelized. This implementation was able to double the speed of the calculation on processor available at the time like Pentium III and IV.

Lindahl, et al. (2001) *J. Mol. Model.* 7: 306-317. (DOI, Citations of this paper)

Domain decomposition

- Domain decomposition refer to a way problem is broken down and is crucial step to parallel a computational problem.
- GROMACS developers have studied different domain decomposition options to minimize the communication overhead.
- The developers concluded that the eighth shell method and the midpoint method always yield the lowest communication time.
- The eighth shell method is used as GROMACS domain decomposition method.

Hess, et al. (2008) *J. Chem. Theory Comput.* 4: 435-447. (DOI, Citations of this paper)

Domain decomposition

- In short, the eight-shell decomposition improved upon the half-shell method.
  - In the half shell method, interactions between particles i and j are calculated in the cell where i and j resides. The minimum communication for this method is half of the volume of a boundary of a thickness equal to cut off radius.
  - The eighth shell method, adopt the above method, and in addition, also calculate the interactions between particles that reside in different communicated zones. Thus, its minimum communication is even less than the original algorithm.
- Comparing to the original ring communication pattern, this new method can save significant on communication overhead and memory requirements.
- More details on the eighth shell method can be found at: https://doi.org/10.1016/0010-4655(91)90021-C

Hess, et al. (2008) *J. Chem. Theory Comput.* 4: 435-447. (DOI, Citations of this paper)

Domain decomposition

GROMACS
fast, flexible & free

- For example, consider a charge group in a MD system (from Hess et al):
  - The charge group's home cell is cell 0, which will be responsible for performing the integration of the equation of motion.
  - Cell 0 receives the coordinates of the particles in the dashed zones from 1-7, communication in only one direction.
  - Each processor calculates the interactions between charge groups of zone 0 with zones 0-7, of zone 1 with zones 3-6, zone 2 with zone 5, and zone 3 with zones 5-6
  - If this procedure is applied for all processors, all pair interactions within the cutoff radius will be calculated.

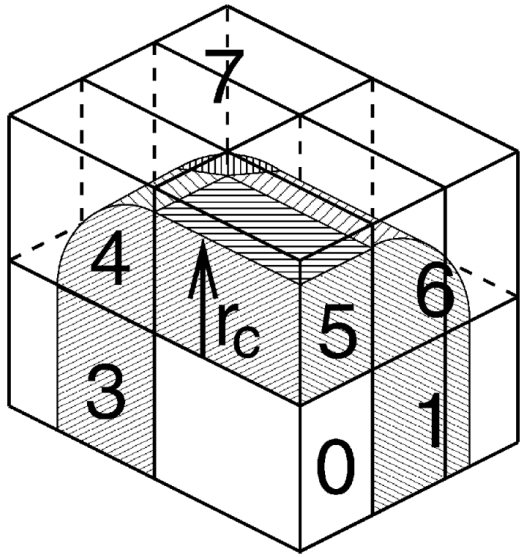
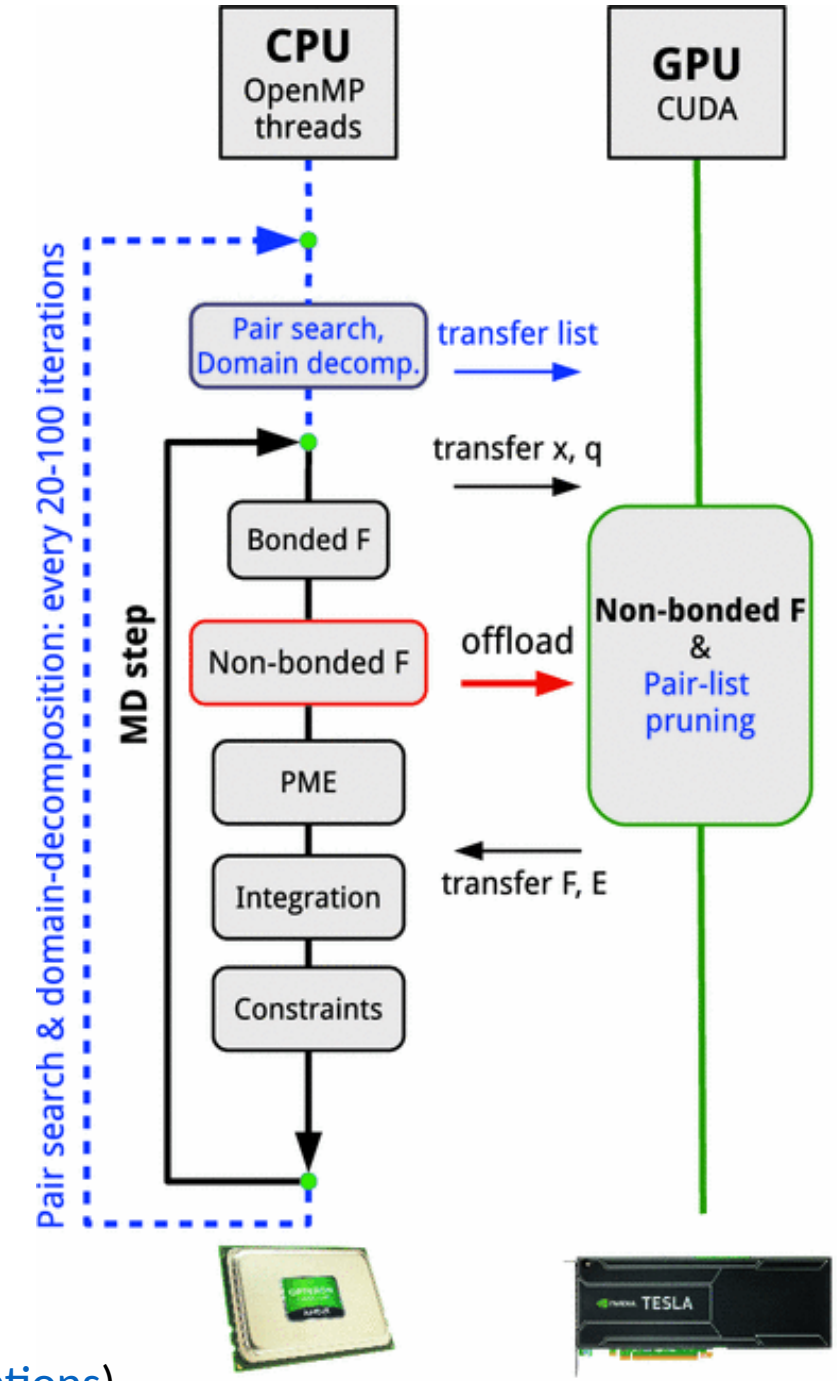


Figure 1. A nonstaggered domain decomposition grid of 3 × 2 × 2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. r_c is the cutoff radius.

Hess, et al. (2008) *J. Chem. Theory Comput.* 4: 435-447. (DOI, Citations of this paper)

Heterogenous Parallelism

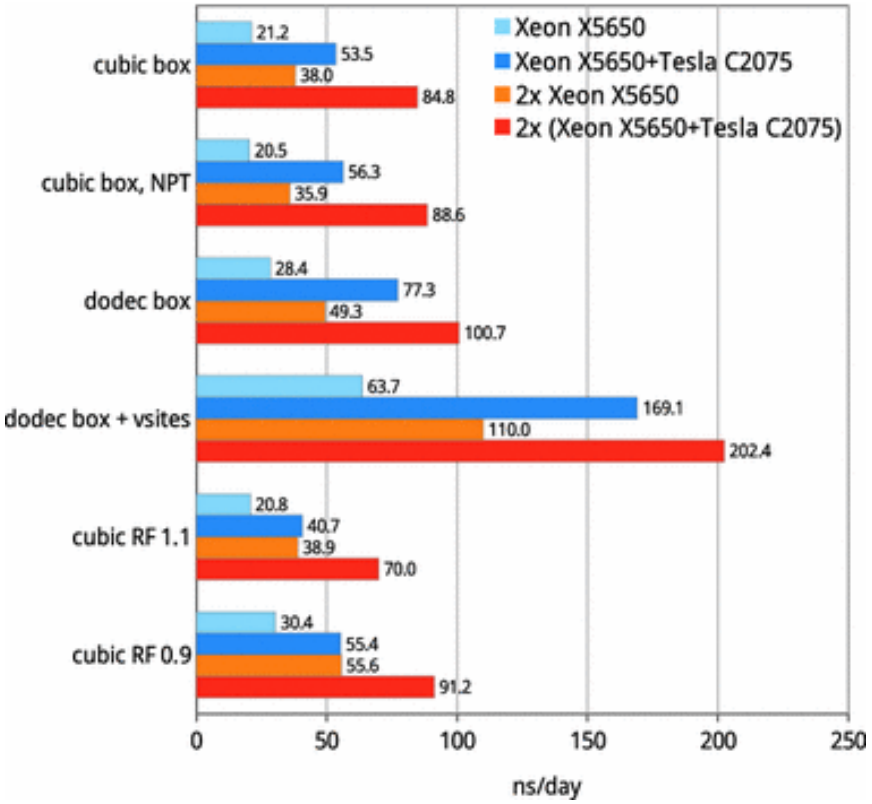
- GROMACS implement a heterogenous parallelism scheme utilize both CPU and GPU to handle different section of the simulations.
- This scheme used the CPU to calculate the more sophisticate portion (searching, domain decomposition) and pass the processed data to the GPU.
- In turn GPUs, implemented through OpenMM library, are used to calculate the embarrassingly parallel portion (like iteration over all atoms pair and calculate potential forces)
- This implementation requires more code to manage the concurrent execution on both the CPU and GPU, but it offers 3-4x speedup comparing CPU only.

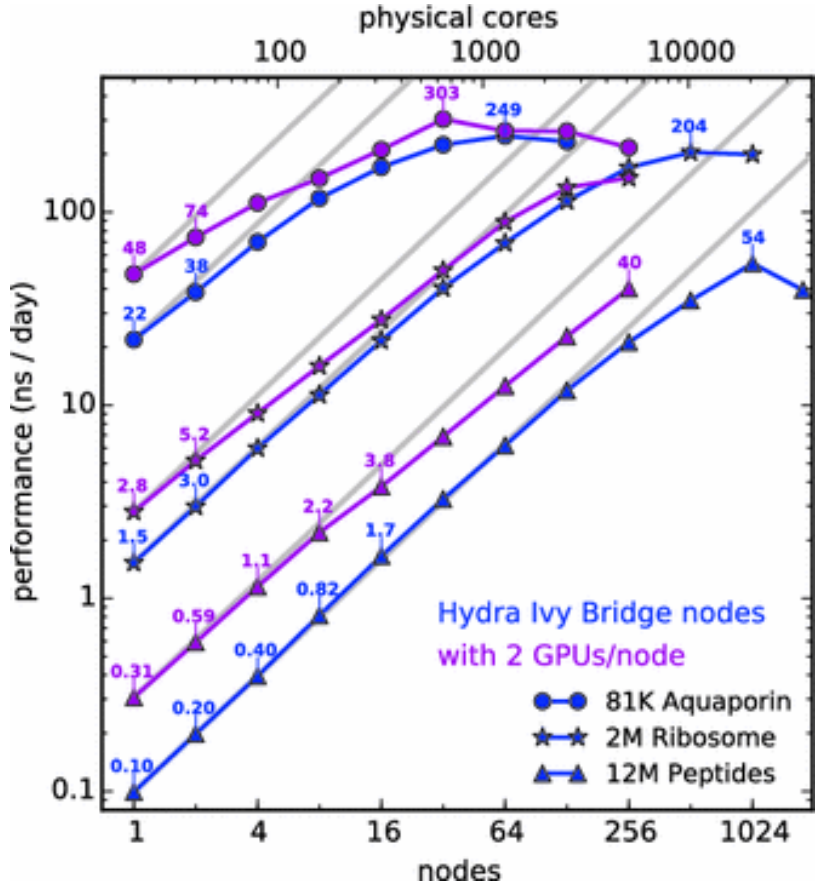


Abraham, et al. (2015) *SoftwareX* 1-2 19-25 (DOI, Citations)

Heterogeneous Parallelism

GROMACS
fast, flexible & free





- The heterogeneous parallelism shows impressive performance as well as strong scaling on different level of hardware – from a single processor to multi-nodes machines

Abraham, et al. (2015) *SoftwareX* 1-2 19-25 (DOI, Citations)

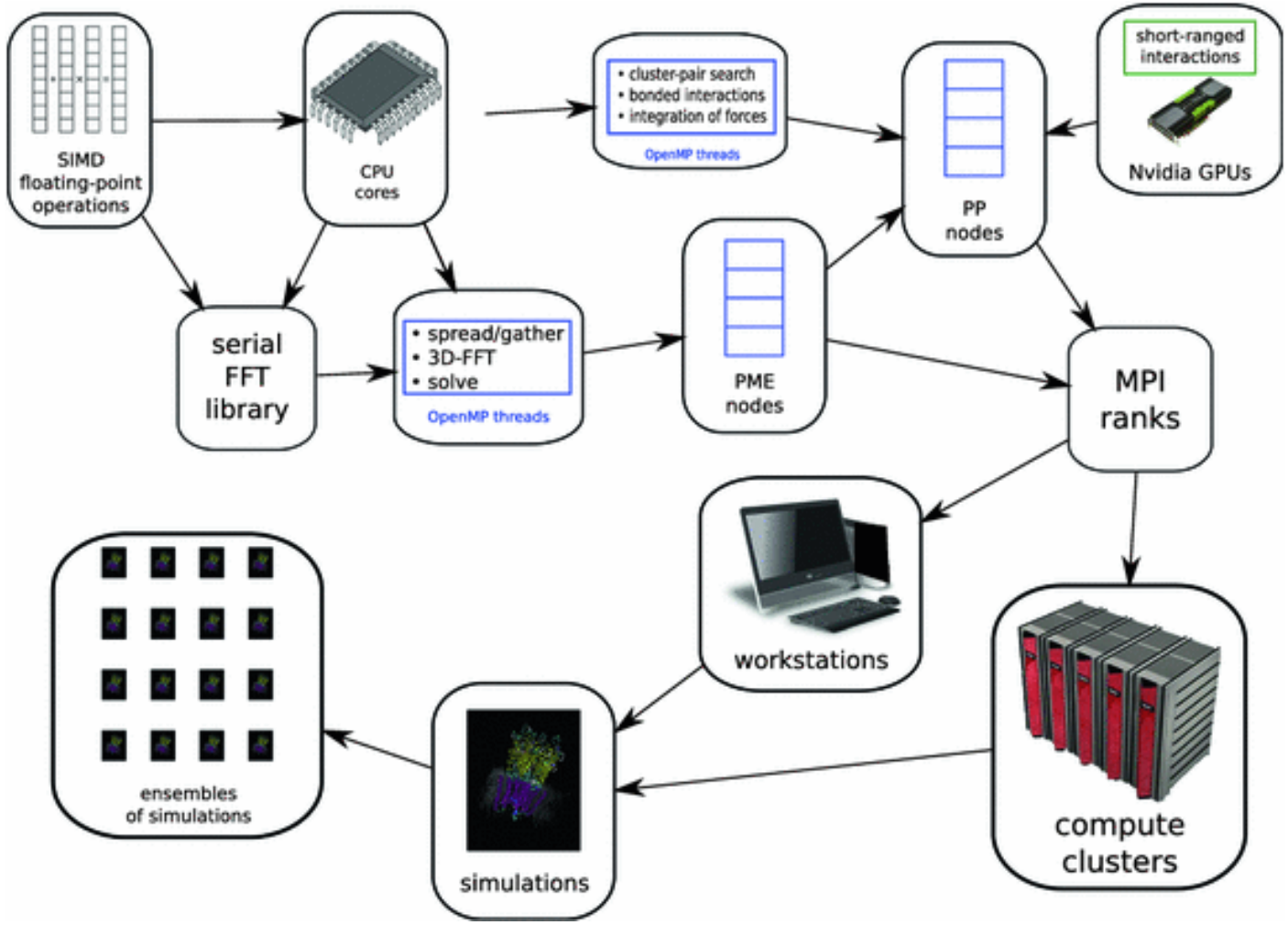
Multi-level Parallelism

- All the optimizations above turn GROMACS into a highly flexible and effective simulation package. This allow the engine to utilizes different level of hardware, from a personal computer to grand scale supercomputer clusters.

Abraham, et al. (2015) *SoftwareX* 1-2 19-25 (DOI, Citations)

Multi-level Parallelism

GROMACS
fast, flexible & free



Abraham, et al. (2015) *SoftwareX* 1-2 19-25 (DOI, Citations)

Results

- Algorithms optimization, from as simple as circumvent the square root calculation to as elaborate as implementing heterogenous parallelism, improve the efficiency and speed of the package significantly.
- Until today, GROMACS is still a widely used toolkit knowing for its efficient and fast simulation speed.
- Allow simulation of more complicated system at longer time.

Opinions

- It is very interesting for me to see how GROMACS develop over years. Starting from having just a working version, to implement different optimization, from simple to complicated, from linearly optimized version to optimized parallel version, the progression of the GROMACS project is very inspirational to me.
- I used GROMACS a lot for my research, and hence, knowing how they implement different algorithms optimization, and whether that sacrifices accuracy in the process, is very important.

GROMACS Principal Papers

1. Berendsen, et al. (1995) *Comp. Phys. Comm.* 91: 43-56. (DOI, Citations of this paper)
2. Lindahl, et al. (2001) *J. Mol. Model.* 7: 306-317. (DOI, Citations of this paper)
3. van der Spoel, et al. (2005) *J. Comput. Chem.* 26: 1701-1718. (DOI, Citations of this paper)
4. Hess, et al. (2008) *J. Chem. Theory Comput.* 4: 435-447. (DOI, Citations of this paper)
5. Pronk, et al. (2013) *Bioinformatics* 29 845-854. (DOI, Citations of this paper)
6. Páll, et al. (2015) *Proc. of EASC 2015 LNCS*, 8759 3-27. (DOI, arxiv, Citations)
7. Abraham, et al. (2015) *SoftwareX* 1-2 19-25 (DOI, Citations)