LATEST PERFORMANCE ENHANCEMENTS TO GROMACS ON SINGLE AND MULTIPLE NVIDIA GPUS

Alan Gray, NVIDIA.
University of Warwick Seminar, 3rd February 2020
ACKNOWLEDGEMENTS

• We are very grateful to the core Gromacs development team in Stockholm for the ongoing collaboration, in particular:
  • Erik Lindahl, Stockholm University/SciLifeLab/KTH
  • Mark Abraham, formerly SciLifeLab/KTH
  • Szilard Pall, KTH/PDC
  • Berk Hess, SciLifeLab/KTH
  • Artem Zhmurov, KTH/PDC
  • Paul Bauer, SciLifeLab/KTH
• The EU BioExcel Center of Excellence for Biomolecular Research supports this collaboration.
AGENDA

• Introduction
• A high-level overview of developments
• Performance results
• Development details
• How to
INTRODUCTION
INTRODUCTION

• Gromacs, a simulation package for biomolecular systems, is one of the most highly used HPC applications globally.

• It evolves systems of particles using the Newtonian equations of motion:
  - **Forces** between particles dictate their movement (e.g. two positively charged ions will repel).

• Calculating forces is most expensive part of simulation - all pairs of particles in the simulation can potentially interact. Forces get weaker with distance, but long-range forces still must be accounted for.
INTRODUCTION

• Force calcs typically fall into three classes in Gromacs:
  
  • **Non-bonded forces:** (short range) - particles within a certain cutoff range interact directly
  
  • **PME:** long-range forces accounted for through a “Particle Mesh Ewald” scheme, where Fourier transforms are used to perform calculations in Fourier space, which is much cheaper than calculating all interactions directly in real space
  
  • **Bonded forces:** required due to specific behaviour of bonds between particles, e.g. the harmonic potential when two covalently bonded atoms are stretched

• These are all now accelerated, most recently the addition of GPU bonded forces in Gromacs 2019 (evolved through prototype work by NVIDIA). But we still have a problem....

  • ...force calcs are now so fast on modern GPUs that other parts are now very significant, especially when we wish to utilize multiple GPUs.

• I will describe work to port all significant remaining computational kernels to the GPU, and to perform the required Inter-GPU communications using peer-to-peer memory copies, such that the GPU is exploited throughout and repeated PCIe transfers are avoided.
A HIGH LEVEL OVERVIEW OF DEVELOPMENTS
GROMACS ON OLD KEPLER ARCHITECTURE

- On old architectures such as Kepler, force calculations are very dominant and other overheads are dwarfed.
- ~400K atom “Cellulose” case.
- : GPU Idle time
But on new architectures such as Volta, force kernels are so fast that other overheads are very significant.

The timescales are aligned in the above profiles.
THE PROBLEM

Single GPU

BO | H2D | NB | D2H | BO | Update&Constrants

Bonded
PME

BO = Buffer Ops

CPU
GPU
PCIe
THE SOLUTION

Single GPU

BO = Buffer Ops
SINGLE GPU: NEW DEVELOPMENT

- Aligned timescales
THE PROBLEM

Multi (4X) GPU

As above

As above

As above

PME

MPI | H2D | PME | D2H | MPI

PP

MPI | BO | H2D | NB | D2H | MPI | BO | Update&Constraints

Bonded

PP

PP

PP

CPU

GPU

PCIe
THE SOLUTION

Multi (4X) GPU

PME

PP

PP

PP

As above

As above

As above

P2P

PME

P2P

PP

NB

P2P

BO

Bonded

BO

U&C

P2P

P2P

GPU

NVLink
• For our multi-GPU experiments we use 4 x V100 SXM2 GPUs fully-connected with NVLink, plus 2xCPU.
• Aligned timescales. STMV (~1M atom) case.
PERFORMANCE RESULTS
Performance results are dependent on system size. We strive to aim our benchmarking and optimization to cover the range of typical sizes in use. We welcome any feedback on further cases to include.
MULTI-GPU: GMX 2020 VS GMX 2019.5

2020 vs 2019.5, 4 x V100 SXM2 16GB

- ADHD: 196 ns/day (2019.5), 441 ns/day (2020), Speedup: 2.26x
- Cellulose: 54 ns/day (2019.5), 147 ns/day (2020), Speedup: 2.74x
- STMV: 15.0 ns/day (2019.5), 43.1 ns/day (2020), Speedup: 2.87x
SINGLE-GPU: GMX 2020 VS GMX 2019.5

2020 vs 2019.5, 1xV100 SXM2 16GB

<table>
<thead>
<tr>
<th></th>
<th>2019.5</th>
<th>2020</th>
<th>Speedup vs 2019.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADHD</td>
<td>163 ns/day</td>
<td>225 ns/day</td>
<td>1.38</td>
</tr>
<tr>
<td>Cellulose</td>
<td>45 ns/day</td>
<td>63 ns/day</td>
<td>1.38</td>
</tr>
<tr>
<td>STMV</td>
<td>8.4 ns/day</td>
<td>12.9 ns/day</td>
<td>1.55</td>
</tr>
</tbody>
</table>
GROMACS 2020: GPU VS CPU

NVIDIA V100-SXM2 16GB GPU vs Dual-socket Intel Gold 6240 CPU server (36 cores total)

GPU Speedup vs 2xCPU

- ADHD
  - 2XCPU: 48 ns/day
  - 1XGPU: 225 ns/day
  - 4XGPU: 441 ns/day

- Cellulose
  - 2XCPU: 14 ns/day
  - 1XGPU: 63 ns/day
  - 4XGPU: 147 ns/day

- STMV
  - 2XCPU: 3.5 ns/day
  - 1XGPU: 12.9 ns/day
  - 4XGPU: 43.1 ns/day
DEVELOPMENT DETAILS

• Reminder: All developments in collaboration with core Gromacs developers.

• GPU Bonded Force calculations:
  • 2019 release: 8 new kernels corresponding to bonded force types
  • 2020 release: kernels merged together for better performance

• GPU Buffer Ops: transformations between different data formats used in gromacs, and force reduction operations. New kernels and major restructuring/refactoring.
DEVELOPMENT DETAILS

• GPU Update and Constraints
  • New kernels related to the “update”, “lincs” and “settle” operations to update and constrain atom positions from forces. Major restructuring and refactoring.

• GPU PME/PP Communications
  • Use of CUDA Peer-to-peer (P2P) memory copies to exchange data directly between GPUs
  • More details coming up

• Device MPI: PP halo exchanges
  • New functionality to pack device-buffers and exchange directly between GPUs using Use of CUDA Peer-to-peer (P2P) memory copies.
  • More details coming up
PP TO PME COMMUNICATION

Original GROMACS

New development

PP task
GPU
Data D2H
Data MPI
CPU

PME task
CPU
Data MPI
Data H2D
GPU

GPU
Data CUDA Direct

GPU
Data CUDA Direct

CPU
**PP TO PP HALO EXCHANGE COMMUNICATION**

**Original Gromacs**

**PP task**
- Data D2H
- Buffer Packing
- Data MPI
- Data H2D
- CPU

**PP task**
- Data D2H
- Buffer Packing
- Data MPI
- Data H2D
- CPU

**New development**

**Small & infrequent**
- Build index map
- Index map D2H
- CPU

**Small & infrequent**
- Build index map
- Index map D2H
- CPU

**GPU**
- Data CUDA direct
- Buffer Packing
- Data CUDA direct
- GPU

**CPU**
- CPU
COORDINATE CPU HALO EXCHANGE

Original CPU MPI Path

Sending MPI Rank

- GPU Local Stream
- GPU Non-Local Stream

CPU Thread

- Pack CPU halo buffer
- MPI Send CPU halo buffer

Receiving MPI Rank

- CPU Thread
- GPU Non-Local Stream
- GPU Local Stream

- MPI Recv data into non-local part of CPU X buffer
- Launch cudaMemcpyAsync
- Launch non-local NB kernel
- cudaMemcpyAsync
- Non-local NB kernel
COORDINATE GPU HALO EXCHANGE

Sending Thread-MPI Rank

- GPU Local Stream (Data previously copied to GPU in local stream)
  - Event record
  - Stream Wait event
  - Pack GPU halo buffer
  - cudaMemcpyAsync push of halo buffer

CPU Thread
- Launch event record
- Launch stream wait event
- Launch packing kernel
- Launch cudaMemcpyAsync
- Launch event record
- MPI Send event pointer

Receiving Thread-MPI Rank

CPU Thread

- MPI Recv remote event pointer
- Stream wait event
- Non-local NB kernel

GPU Non-Local Stream
- Non-local NB kernel

GPU Local Stream
- Data pushed to non-local part of remote GPU X buffer
- Provides required sync on event record

Event record
FORCE GPU HALO EXCHANGE

Sending Thread-MPI Rank

<table>
<thead>
<tr>
<th>CPU Thread</th>
<th>GPU Non-Local Stream</th>
<th>GPU Local Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch cudaMemcpyAsync or cudaMemcpyAsync(0)</td>
<td>Event record</td>
<td>cudaMemcpyAsync</td>
</tr>
<tr>
<td>Launch event record</td>
<td>Stream Wait event</td>
<td></td>
</tr>
<tr>
<td>Launch cudaMemcpyAsync</td>
<td>Event record</td>
<td></td>
</tr>
<tr>
<td>Push of non-local part of F buffer</td>
<td>MPI Send event pointer</td>
<td></td>
</tr>
</tbody>
</table>

Receiving Thread-MPI Rank

<table>
<thead>
<tr>
<th>CPU Thread</th>
<th>GPU Non-Local Stream</th>
<th>GPU Local Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recv remote event pointer</td>
<td>Stream wait event</td>
<td></td>
</tr>
<tr>
<td>Launch unpacking kernel</td>
<td>Unpack GPU halo buffer into local part of F buffer</td>
<td></td>
</tr>
<tr>
<td>Launch event record</td>
<td>Event record</td>
<td></td>
</tr>
<tr>
<td>Launch stream wait event</td>
<td>Stream Wait event</td>
<td></td>
</tr>
<tr>
<td>Launch local F Buffer Ops</td>
<td>Local F buffer ops</td>
<td></td>
</tr>
</tbody>
</table>
HOW TO
ENABLING NEW OPTIMIZATIONS

• The new optimizations are not yet enabled by default: the new code-paths have been verified by the standard GROMACS regression tests, but still lack substantial “real-world” testing.

• We stress that users should carefully verify results against the default path, and any reported issues will be gratefully received by the developer team to help us mature the software.
ENABLING NEW OPTIMIZATIONS

Gromacs should be built using its internal threadMPI library instead of any external MPI library (see above), and at run time the optimizations can be fully enabled by setting the following three environment variables to any non-NULL value in your shell (as shown for bash shell here).

For halo exchange communications between PP tasks:

```bash
export GMX_GPU_DD_COMMS=true
```

For communications between PME and PP tasks:

```bash
export GMX_GPU_PME_PP_COMMS=true
```

To multi-GPU enable the update and constraints part of the timestep:

```bash
export GMX_FORCE_UPDATE_DEFAULT_GPU=true
```

The combination of these will trigger all optimizations, including dependencies such as GPU-acceleration of buffer operations.
ENABLING NEW OPTIMIZATIONS

When running on a single GPU, only `GMX_FORCE_UPDATE_DEFAULT_GPU` is required (or, for single-GPU only, this can equivalently be enabled by adding the `-update gpu` flag to the `mdrun` command).

Of course, in both single and multi GPU cases it is also necessary to assign the three classes of force calculations to the GPU through the

```
-nb gpu -bonded gpu -pme gpu
```

options to `mdrun` (where, on multi-GPU, the `-npme 1` option will also be required to limit PME to a single GPU).
USE H-BONDS CONSTRAINTS

• Note also that the new GPU update and constraints code-path is only supported in combination with domain decomposition of the PP tasks across multiple GPUs when `update groups` are used.

• This means constraining all bonds is not supported, except for small molecules. To facilitate this, it is recommended to convert bonds with hydrogen bonds to constraints, rather than all bonds.

• To do this, in the `.mdp` input file to the Gromacs preprocessor `grompp`, change the line

```
constraints = all-bonds
```

to

```
constraints = h-bonds
```

For more information see the `mdp options section of the user guide`. 
SPECIFIC COMMAND USED FOR RESULTS

The full set of mdrun options used when running the above 4XGPU performance comparisons are as follows:

```bash
gmx mdrun -v -nsteps 100000 -resetstep 90000 -noconfout \
-ntmpi 4 -ntomp 10 \ 
-nb gpu -bonded gpu -pme gpu -npme 1 \ 
-nstlist 400
```
SPECIFIC COMMAND USED FOR RESULTS

• The first line instructs Gromacs to run 100,000 steps for this relatively short benchmark test, with the timing counters reset at step 90,000 to avoid initialization costs being included in the timing since these will not be important for typical long production runs.

• These specific values have been chosen to allow Gromacs long enough to automatically perform PME tuning, where the size of the PME grid is tuned to give an optimal load balance between PP and PME tasks.

• Similarly, the -noconfout option instructs Gromacs not to write output files at the end of this short benchmarking run to avoid artificially high I/O overheads.
SPECIFIC COMMAND USED FOR RESULTS

- The second line specifies that 4 thread-MPI tasks should be used (i.e. one per GPU), with 10 OpenMP threads per thread-MPI task such that a total of 40 OpenMP threads are in use to match the number of physical CPU cores in our server.

- The third line offloads all force calculations to the GPU.

- And the final `-nstlist 400` option instructs Gromacs to update the neighbor list with frequency of 400 steps - this can be adapted without any loss of accuracy (when using the Verlet scheme), and we found this value to give the best performance through experimentation.
DEFAULT: ONLY NB FORCE ON GPU

gmx mdrun -v -s cellulose-hbond.tpr -nsteps 2000 \
   -ntmpi 4 \ #run on 4 GPUs (1 tMPI task per GPU)...
   -ntomp 10 \ # ... with 10 OpenMP tasks per tMPI task (40-core server)
   -notunepme \ #Don’t perform PME tuning (more later)
   -resetstep 1000 \ #Benchmarking: reset timers to rm init overhead
   -noconfout \ #Benchmarking: don’t write output configuration

... (ns/day) (hour/ns)
Performance: 6.822 3.518
ACTIVATE BONDED AND PME FORCE ON GPU

gmx mdrun -v -s cellulose-hbond.tpr -nsteps 2000 -ntmpi 4 -ntomp 10 \ 
-notunepme -resetstep 1000 -noconfout \ 
-bonded gpu -pme gpu -npme 1 #put bonded and PME force calcs on GPU
...

(ns/day)    (hour/ns)

Performance:    42.785       0.561
ACTIVATE NEW GROMACS 2020 FEATURES

#Activate new 2020 features
export GMX_GPU_DD_COMMS=1
export GMX_GPU_PME_PP_COMMS=1
export GMX_FORCE_UPDATE_DEFAULT_GPU=1

gmx mdrun -v -s cellulose-hbond.tpr -nsteps 2000 -ntmpi 4 -ntomp 10 \ -notunepme -resetstep 1000 -noconfout \ -bonded gpu -pme gpu -npme 1
...

(nS/day)  (hour/ns)
Performance:  87.661  0.274
ALLOW TUNING OF PME GRID SIZE AND DECREASE NEIGHBOUR SEARCH FREQUENCY

#Activate new 2020 features
export GMX_GPU_DD_COMMS=1
export GMX_GPU_PME_PP_COMMS=1
export GMX_FORCE_UPDATE_DEFAULT_GPU=1

gmx mdrun -v -s cellulose-hbond.tpr -nsteps 100000 -ntmpi 4 -ntomp 10 \ 
  -resetstep 90000 \ 
  -noconfout \ 
  -bonded gpu -pme gpu -npme 1 \ 
  -nstlist 400

#removed -notunepme and increased number of steps to allow PME tuning to complete
...

(ns/day)   (hour/ns)
Performance: 147.440 0.163
SUMMARY AND FUTURE WORK

• Modern GPUs are so fast in performing Gromacs force calculations that the other parts of the simulation timestep are becoming a bottleneck.

• In Gromacs 2020 we have accelerated all other main computational parts and enabled peer-to-peer communication directly between GPUs.

• Large performance improvements available on multi-GPU. Less dramatic but significant performance improvements available on single-GPU.

• Future work will include
  • Stabilization and further refactoring.
  • PME decomposition: enablement of multi-GPU for PME could improve load balance, and also potentially allow scaling to higher numbers of GPUs.
  • Remove single-node limitation for new GPU communication performance features.
  • Further performance improvements, especially for smaller cases, through more overlapping