## GROMACS - Feature #1420

**Fast force reduction for large number of threads**

01/13/2014 09:11 PM - Roland Schulz

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Roland Schulz</td>
</tr>
<tr>
<td>Category:</td>
<td>mdrun</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
<tr>
<td>Difficulty:</td>
<td>uncategorized</td>
</tr>
</tbody>
</table>

### Description

Each thread has its own force buffer during computation. The current force reduction partitions the final force buffer across the threads and reads from all force buffers. This doesn't utilize any locality and isn't fast for large number of threads especially on Intel Phi.

### Related issues:

- Related to GROMACS - Feature #1187: Optimized Verlet SIMD kernels for native ... Closed
- Related to GROMACS - Feature #1181: Implementing asymmetric offload to MIC (Xeon Phi co-processor) Closed

### Associated revisions

Revision 0acdcca5 - 01/15/2014 11:25 PM - Roland Schulz

Add nbnxn tree force reduction

Alternative implementation of the nbnx force reduction. The previous method is implemented as a loop over all thread buffers. Its advantage is that it doesn't require any synchronizations during reduction and only requires a single write per element. The new method utilizes a binary tree for the reduction. Its advantage is that the data locality of the force buffers is used during reduction.

The current implementation isn't faster on the CPU (for E526070 with icc). On the MIC it is faster (36% reduction of buffer-f time for 32 threads).

Part of #1420

Change-Id: l2d84345e9a19d9030a837d324671f2ab0ba9b91b

### History

#### #1 - 01/13/2014 09:12 PM - Roland Schulz

- Related to Feature #1187: Optimized Verlet SIMD kernels for native MIC (Xeon Phi co-processor) added

#### #2 - 01/13/2014 09:12 PM - Roland Schulz

- Related to Feature #1181: Implementing asymmetric offload to MIC (Xeon Phi Co-processor) added

#### #3 - 01/13/2014 11:48 PM - Roland Schulz

An implementation which is faster on the MIC: [https://gerrit.gromacs.org/#/c/2825](https://gerrit.gromacs.org/#/c/2825)

This implementation still has room for future improvements:

- Test whether the exponential backoff implemented in tbb::internal::atomic_backoff is better than constant pause currently used
- do the work first which is read by different thread in next step (could help with caching and allows to report ready after 1/2 work is done)
- add prefetch
- dont zero in first level if not necessary (instead update flags)
- dont do tree for first level(s)
- balance the tree more evenly (e.g. by doing non-power of 2 in first step). Might help with e.g. 2x6 cores because currently tree is spanning CPUs.
- nbnxn_atomdata_add_nbat_f_to_f_part should also utilize locality
- we assume that we have fast access to the data written in the previous step. That is only true if it fits into the cache. Otherwise it might not even be local memory (for NUMA) because we work on thread buffers.
- Test whether the tree implementation is faster on certain CPUs and whether it should be used by default

#### #4 - 01/14/2014 09:49 PM - Szilárd Páll

Roland Schulz wrote:
Each thread has its own force buffer during computation. The current force reduction partitions the final force buffer across the threads and reads from all force buffers. This doesn't utilize any locality and isn't fast for large number of threads especially on Intel Phi.

That's why there was a masters project in our group investigating reduction techniques; the work was rather good and the results pretty interesting - although quite messy and inconclusive, but unfortunately the different reduction algorithms did not get plugged into GROMACS. Still, the algorithms and findings may help with, among others, answering some of the questions regarding future direction/improvements.

The thesis is [here](#).

- Test whether the exponential backoff implemented in `tbb::internal::atomic_backoff` is better than constant pause currently used

Any pause/backoff only makes sense if SMT is used (and maybe later with taking + interleaved tasks), so this will depend on the architecture.

**#5 - 01/14/2014 11:02 PM - Roland Schulz**

Szilárd Páll wrote:

Any pause/backoff only makes sense if SMT is used (and maybe later with taking + interleaved tasks), so this will depend on the architecture.

Actually it could help even without SMT, because the extra memory accesses in the spin-loop could slow down the memory access of the working thread(s). Of course it still depends on the architecture.

**#6 - 01/15/2014 05:27 PM - Szilárd Páll**

Roland Schulz wrote:

Szilárd Páll wrote:

Any pause/backoff only makes sense if SMT is used (and maybe later with taking + interleaved tasks), so this will depend on the architecture.

Actually it could help even without SMT, because the extra memory accesses in the spin-loop could slow down the memory access of the working thread(s). Of course it still depends on the architecture.

Indeed - although that depends quite a bit on how much cache traffic does the loop conditional generate as well as on the cache behavior. I can imagine that this is an issue with the KNC's coherent L2.

**#7 - 06/20/2014 10:44 AM - Erik Lindahl**

- Target version changed from 5.0 to 5.x

**#8 - 06/23/2016 11:31 AM - Mark Abraham**

- Target version deleted (5.x)

**#9 - 03/06/2018 11:33 PM - Mark Abraham**

- Status changed from In Progress to Closed

No activity, so I figure we're done here.