### Gromacs - Bug #1171

**Unsupported heterogeneity of PME ranks**

03/01/2013 02:07 PM - Mikhail Plotnikov

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>High</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Berk Hess</td>
</tr>
<tr>
<td>Category:</td>
<td>mdrun</td>
</tr>
<tr>
<td>Target version:</td>
<td>4.6.1</td>
</tr>
<tr>
<td>Affected version - extra info:</td>
<td>Affected version:</td>
</tr>
</tbody>
</table>

**Description**

If some of PME-related ranks are executed with one number of OpenMP threads (OMP_NUM_THREADS=nt1) and other PME-ranks executed with a different number of OpenMP threads (OMP_NUM_THREADS=nt2), then application fails in MPI_Sendrecv (pme.c:3870). Root cause is that the function is called with different size of send-receive buffer for different MPI-ranks while it must be the same for all.

To reproduce the issue please use attached test case. It uses 2 nodes defined in nodes.txt file. Mpirun executes wrapper, which sets OMP_NUM_THREADS=16 on one node and =4 on another, and then executes binary mdrun_mpi.SNB, compiled from gromacs-4.6.tar.gz tarball for SNB architecture. Application fails with error:

Fatal error in MPI_Sendrecv: Message truncated, error stack:

```none
MPI_Sendrecv(svbuf=0x2ab04ee57010, scount=150528, MPI_FLOAT, dest=0, stag=0, recvbuf=0x2ab04ef22010, rcount=200704, MPI_FLOAT, src=3, rtag=0, comm=0x84000000, status=0x7fffeb04394c) failed
MPIDI_CH3U_Receive_data_found(129): Message from rank 3 and tag 0 truncated; 824464 bytes received but buffer size is 802816
```

Diagnostics from ITAC:

- [19] ERROR: See the MPI_ERROR field in MPI_Status for the error code.
- [19] ERROR: Error occurred at:
- [19] ERROR: MPI_Sendrecv(*sendbuf=0x2442500, sendcount=150528, sendtype=MPI_FLOAT, dest=0, sendtag=0, *recvbuf=0x250c7d0, recvcount=200704, recvtype=MPI_FLOAT, source=3, recvtag=0, comm=0xffffffff84000002 SPLIT COMM_WORLD [3:19:4], *status=0x7fff1cf2894c)

In case there is a heterogeneous cluster, e.g. with SNB + WSM nodes or SNB + MIC, optimal number of OpenMP threads can vary for every MPI-rank. By default, PME nodes are evenly interleaved with PP nodes and some of them assigned to SNB nodes, some to WSM or MIC. In this case MPI will fail in Sendrecv function. MPI-decomposition should be designed in the way it is not sensible to the number of OpenMP threads inside MPI ranks. PP-ranks don't have such problem – they work fine with different number of OpenMP threads in different PP-ranks.

**Associated revisions**

Revision cddf15cb - 03/21/2013 10:54 AM - Berk Hess

made PME work with a mix of 1 and more threads

Using a mix of 1 and more OpenMP threads on different MPI ranks

would make mdrun terminate with an MPI error.

Fixes #1171

Change-Id: lfa16e18baf074be826b59503208dca01d1ec14

---

Revision 1eff6a81 - 06/13/2013 09:45 AM - Berk Hess

made PME work with a mix of 1 and more threads

Using a mix of 1 and more OpenMP threads on different MPI ranks
would make mdrun terminate with an MPI error.

Fixes #1171

Change-Id: iffa16e18bal0f74be826b59503208dca01d1ec14

--- a/src/mdlib/pme.c  
+++ b/src/mdlib/pme.c  
@@ -3867,6 +3867,8 @@ static void sum_fftgrid_dd(gmx_pme_t pme, real *fftgrid)
 }

#ifdef GMX_MPI
+        printf("rank %d send %4d recv %4d data %4d
",
+               pme->nodeid, send_nindex, recv_nindex, datasize);
MPI_Sendrecv(sendptr, send_nindex*datasize, GMX_MPI_REAL, send_id, ipulse,
recvptr, recv_nindex*datasize, GMX_MPI_REAL, src=3, rtag=0, comm=0x84000000, status=0x7fffde5184c4) failed
MPIDI_CH3U_Receive_data_found(129): Message from rank 3 and tag 0 truncated; 824464 bytes received but buffer size is 802816

Do you use exactly the configuration I've set up in reproducer (16x1+4x4)?
Ah, you have 1 and multiple threads mixed. I thought it was 16 and 4. Indeed mixing no OpenMP with OpenMP on other processes in currently not supported (but there is no check). As these two cases use partially different code paths, supporting that takes some effort. As I don't see many use cases where this could lead to optimal performance, I'd rather avoid implementing this. I will add a check for 4.6.2 on mixing 1 and >1 OpenMP threads.

I use the same binary, which have OpenMP support, on both nodes. Why cases with nthreads=1 and nthreads>1 should have different paths? Does it give any advantage in case OpenMP is already enabled in compiled binary? BTW, cases with any other number of OpenMP threads on the first node (16x2+4x4 or 16x3+4x4) work fine.

I use the same binary, which have OpenMP support, on both nodes. Why cases with nthreads=1 and nthreads>1 should have different paths? Does it give any advantage in case OpenMP is already enabled in compiled binary? BTW, cases with any other number of OpenMP threads on the first node (16x2+4x4 or 16x3+4x4) work fine.

In PME we have a compact charge-grid and we copy that to a padded FFT grid layout. With multiple threads we need to reduce thread-local charge contributions to the charge grid, which are then directly stored in the padded FFT grid and then the overlap is reduced over MPI. Without threads we use the "old" code path, where we reduce the charge-grid over MPI and then copy. Without threads we could use the threaded code-path, I realize now, but then we increase the communication cost somewhat.

I guess using threaded code-path is better than application failure anyway, even if this requires communication overhead. Probably it worth to implement switch to non-thread path only in case all PME ranks have single thread. It shouldn't be difficult to check.

While implementing the suggestion to switch to the threaded code path when at least of the the PME MPI ranks uses OpenMP, I realized that also the 3D FFT code has two different code paths. It will take some effort to also implementing the code path switch there. So the question is if all this is worth the effort, or if we should simply put in a fatal error in case some ranks use 1 and others >1 threads?

Are there realistic use cases where you would want to mix 1 and 2 OpenMP threads?
On Intel MIC we would probably want to use MIC only for particle-particles interactions.

Are there realistic use cases where you would want to mix 1 and 2 OpenMP threads?
On Intel MIC we would probably want to use MIC only for particle-particles interactions.

This is a future plan. Currently I have already problems with adjusting optimal decomposition because of this issue and this affects performance. For me this is a high priority issue. Certainly, you can demote priority from "urgent" to "normal" if you have more urgent bugs to solve, but I hope this one will be properly fixed (not just by generating fatal error).
It turned out that I overlooked one code path difference and no changes were needed in the FFT code at all. As it's a simple fix now, and I wanted to check for nthread>1 on any rank anyhow, I fixed it and uploaded the change to gerrit. Note that I still think we would rather avoid heterogeneous thread counts for PME as this will generally lead to load imbalance in different parts of PME. For MIC the proper solution to this is already in another redmine issue.

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>pme_test.tbz2</td>
<td>6.88 MB</td>
<td>03/01/2013</td>
<td>Mikhail Plotnikov</td>
</tr>
</tbody>
</table>