From MPI to openSHMEM: porting LAMMPS

ICL Friday Lunch Talk May 8th, 2015

CHUNYAN TANG
Outline

• Background
• Methodology
  - Design and implementations
• Evaluation
  - Performance analysis
• Summary and future work
What is OpenSHMEM?

- SHMEM – SHared MEMory
- 1-sided Communication, Partitioned Global Address Space (PGAS)
- Symmetric objects have same address on all Processing Element (PEs)

* Memory Model from OpenSHMEM specification 1.2
What can OpenSHMEM do?

- Point to point (put, get)
- Synchronization mechanism (barrier, quiet, fence, wait)
- Collective operations (broadcast, collection, reduction)
- atomic operations (swap, add, increment)

Why OpenSHMEM?
- Different model:
  - Direct access vs. message passing
- Low overhead
- Symmetric heap
# OpenSHMEM vs. MPI APIs

<table>
<thead>
<tr>
<th>MPI</th>
<th>OpenSHMEM</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Init()</td>
<td>Start_pes()</td>
<td>Initialize the library</td>
</tr>
<tr>
<td>MPI_Comm_rank()</td>
<td>Shmem_my_pe()</td>
<td>Return the rank of PE</td>
</tr>
<tr>
<td>MPI_Comm_size()</td>
<td>Shmem_n_pes()</td>
<td>Return number of PEs</td>
</tr>
<tr>
<td>MPI_Barrier()</td>
<td>Shmem_barrier_all()</td>
<td>Synchronization</td>
</tr>
<tr>
<td>MPI_Send, MPI_Irecv</td>
<td>Shmem_put()</td>
<td>Put data to a remote PE</td>
</tr>
<tr>
<td></td>
<td>Shmem_fence()</td>
<td>Ensure ordering operations</td>
</tr>
<tr>
<td></td>
<td>Shmem_int_p()</td>
<td>Put an integer to remote PE</td>
</tr>
</tbody>
</table>
Why LAMMPS?

- Large-scale Atomic/Molecular Massively Parallel Simulator
- Solid-state materials (metals, semiconductors)
- Soft matters (biomolecules, polymers)
- MPI based application
LAMMPS Profiling

- Vampir: 512 processes, 8*8*8*32K atoms
- Communication time: 11.5%
- 167 MPI call sites.
- Top aggregate sent message site
  - MPI_Send in Remap_3d(): > 35% data
2-D Decomposition of Parallel 3-D FFTs

1d-fft fast axis
1d-fft mid axis
1d-fft slow axis

Remap1
Remap2

Process Matrix 8*8
3D-FFT in LAMMPS – $P(x,y,z,e)$

Step 0:
FFT$_e$ → 1d-fft fast axis → Remap0 → 1d-fft mid axis → Remap1 → 1d-fft slow axis → Remap2 → Remap3

Step 1:
FFT$_x$ → 1d-fft fast axis → Remap4 → 1d-fft mid axis → Remap5 → 1d-fft slow axis → Remap6

Step 2:
FFT$_y$ → 1d-fft fast axis → Remap4 → 1d-fft mid axis → Remap5 → 1d-fft slow axis → Remap6

Step 3:
FFT$_z$ → 1d-fft fast axis → Remap4 → 1d-fft mid axis → Remap5 → 1d-fft slow axis → Remap6
MPI Point-Point Comm.

Step 1

PE0 - Sender

in

Pack

MPI_send

Step 1

PE1 - Receiver

MPI_Irecv

Scratch

unPack

out
OpenSHMEM Point-Point Comm.

PE0 - Sender

- Pack
- Shmem_put()
- Shmem_fence()
- Shmem_int_P()

PE1 - Receiver

- Scratch
- unPack
- out

Step 1

Time
OpenSHMEM Point-Point Comm.

**PE0 - Sender**
- Pack
- Shmem_put()
- Shmem_fence()
- Shmem_int_P()

**PE1 - Receiver**
- Scratch
- unPack
- out

- **Step 1**
  - Wrong data!

- **Step 2**
  - Shmem_put()
First Implementation
– Hybrid1_Barrier

Step 1:
- PEO - Sender
  - Pack
  - Shmem_put()
  - Shmem_fence()
  - Shmem_int_P()

Step 2:
- MPI_Barrier()

Step 1:
- PE1 - Receiver
  - Scratch
  - unPack
  - out

Step 2:
- MPI_Barrier()
Performance of LAMMPS - Hybrid1_Barrier vs. MPI

Strong Scaling

Problem size: 8*8*8*32k

Performance improvement: 2% (512 cores) - 7% (2048 cores)

Weak Scaling

32k/core

The performance is measured on TITAN at ORNL
Second Implementation - Hybrid2_ChkBuf

• Barrier synchronize all the processes in the same group
• Can we reduce the wait time?
  • Do not use Barrier
  • Check peer’s receive buffer status
Second Implementation - Hybrid2_ChkBuf

**PE0 - Sender**

- Pack
- Shmem_put()
- Shmem_fence()
- Shmem_int_P()

**PE1 - Receiver**

- Scratch
- unPack
- out
- Shmem_int_p()

**Step 1**

- Time
- I am ready!

**Step 2**

- Shmem_put()
Performance of LAMMPS - Hybrid2_ChkBuf vs. MPI

Strong Scaling
Problem size: 8*8*8*32k

Weak Scaling
32k/core

Performance improvement: 1% (512 cores) - 3% (2048 cores)

The performance is measured on TITAN at ORNL
Steps Between Processes - Hybrid2_ChkBuf

PE0-fast

Step 1

Step 2

Step 3

PE1-slow

Step 1

Step 1

Step 2

Step 2

Wait for others

Time

One processes can only be one step further than it’s peer

Wait for others

I am ready!

I am ready!
Third Implementation
- Hybrid4_2Buffer

• One process can only be one step further than the other.
• Can we reduce the wait time?
  • Do not check peer’s receive buffer status
  • Use double buffer
OpenSHMEM Point-Point Comm.

Step 1:
- Pack
- Shmem_put()
- Shmem_fence()
- Shmem_int_P()

Step 2:
- Shmem_put()

PE0 - Sender

PE1 - Receiver

Scratch1
unPack
Scratch2
Performance of LAMMPS - Hybrid4_2buf vs. MPI

The performance is measured on TITAN at ORNL

Strong Scaling
Problem size: 8*8*8*32k

Weak Scaling
32k/core

Performance improvement: 0% (512 cores) - 1% (2048 cores)
Future work

• Why Hybrid2_ChkBuf and Hybrid4_2Buf is slower than Hybrid2_Barrier?
• All the efforts are focused on reducing the wait time.
• Add more memory hits? More expensive?
Summary

• Ported OpenSHMEM to a large scale application.
• Proposed three different semantics of OpenSHMEM point to point communications.
• Hybrid MPI and OpenSHMEM implementations indicate better performance than the highly optimized MPI implementation when working with large number of processes.
• OpenSHMEM has great potential to improve performance.
Acknowledgements

• ILC
  • George Bosilca
  • Thomas Herault
  • Aurelien Boutleiller

• ORNL
  • Manjunath Gorentla Venkata
Thank you!
Profiling of 3D-FFT

Profiling of 3DFFT in LAMMPS on Titan (nprocs = 512)

Profiling of 3D FFT in LAMMPS on Titan (nprocs = 2048)

nprocs = 512

nprocs = 2048
Hybrid_chkbuf2 – Check Buffer Status

Sender

Scratch

Symmetric Heap Memory

out

in

Ready?

Receiver

Scratch

in

out

Arrived?

status

Shmem_int_p
... 
for (i = 0; i < nrecv; i++)
    MPI_Irecv( ... )

for (i = 0; i < nsend; i++){
    plan->pack( ... );
    MPI_Send( ... );
}

for (i = 0; i < nrecv; i++) {
    MPI_Waitany ( ... );
    plan->unpack( ... );
}
...
shmem_double_put (send continuous data on fast)

for (slow = 0; slow < nslow; slow++) {
  plane = slow*nstride_plane;
  for (mid = 0; mid < nmid; mid++) {
    nout = plane + mid*nstride_line;
    buf_offset = mid*nfast;
    shmem_double_put(&scratch[...buf_offset],
                     &data[nout],nfast,plan->send_proc[isend]);
  }
}

shmem_double_iput (send scattered data on slow)

for (mid = 0; mid < nmid; mid++) {
  tst = sst = plane = nstride_plane;
  for (fast = 0; fast < nfast; fast++) {
    nout = mid*nstride_line;
    buf_offset = mid*nfast;
    shmem_double_iput(&scratch[...],
                      &data[nout], tst, sst, nslow, plan->send_proc[isend]);
  }
}
Heat Map for Remap – x,y,z gradients
Heat Map for Remap - Charge

Remap0

Remap1

Remap2

Remap3
Steps Between Processes

- **P2**
  - Step 1
  - Step 2
  - Step 2
  - Step 3

- **P0**
  - Step 1
  - Step 2
  - Step 2
  - Step 3

- **P1**
  - Step 1
  - Step 1
  - Step 2

Background