MPI Optimization

HPC Saudi, March 15th 2016
Useful Variables

- **MPI Variables to get more insights**
  - `MPICH_VERSION_DISPLAY=1`
  - `MPICH_ENV_DISPLAY=1`
  - `MPICH_CPUMASK_DISPLAY=1`
  - `MPICH_RANK_REORDER_DISPLAY=1`

- **When using MPI-IO**
  - `MPICH_MPIIO_STATS=1` # or 2 for gnuplot and cvs output
  - `MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1`
  - `MPICH_MPIIO_HINTS_DISPLAY=1`

- **Check MPI man page for more details.**
  - Also DMAPP for collective communications.
Day in the Life of an MPI Message

- Four Main Pathways through the MPICH2 GNI NetMod
  - Two EAGER paths (E0 and E1)
    - For a message that can fit in a GNI SMSG mailbox (E0)
    - For a message that can't fit into a mailbox but is less than MPICH_GNI_MAX_EAGER_MSG_SIZE in length (E1)
  - Two RENDEZVOUS (aka LMT) paths: R0 (RDMA get) and R1 (RDMA put)

- Selected Pathway is based on Message Size

```
<table>
<thead>
<tr>
<th></th>
<th>E0</th>
<th>E1</th>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>512</td>
<td>1K</td>
<td>2K</td>
<td>4K</td>
</tr>
<tr>
<td>8K</td>
<td>16K</td>
<td>32K</td>
<td>64K</td>
<td>128K</td>
</tr>
<tr>
<td>256K</td>
<td>512K</td>
<td>1MB</td>
<td>4MB</td>
<td></td>
</tr>
</tbody>
</table>
```

- MPICH_GNI_MAX_VSHORT_MSG_SIZE (64-8144 bytes)
- MPICH_GNI_MAX_EAGER_MSG_SIZE
- MPICH_GNI_NDREG_MAXSIZE
Day in the Life of Message type E0
EAGER messages that fit in the GNI SMSG Mailbox

- GNI SMSG Mailbox size changes with number of ranks in job
- If user data is 16 bytes or less, it is copied into the MPI header
Day in the Life of Message type E1

EAGER messages that don't fit in the GNI SMSG Mailbox

- User data is copied into internal MPI buffers on both send and receive side

 Sender

1. Mempcpy data to pre-allocated MPI buffers

2. GNI SMSG Send (MPI header)

3. RDMA GET

4. GNI SMSG Send (Recv done)

Receiver

SMSG Mailboxes

5. Mempcpy

MPICH_GNI_NUM_BUFS
default 64 buffers, each 32K
Day in the Life of Message type R0
Rendezvous messages using RDMA Get

- No extra data copies
- Best chance of overlapping communication with computation
Day in the Life of Message type R1
Rendezvous messages using RDMA Put

- Repeat steps 2-6 until all sender data is transferred
- Chunksize is MPI_GNI_MAX_NDREG_SIZE (default of 4MB)
Scenario and Application Background
Optimization Scenario

- You have been given a benchmark that is not meeting required performance
- You are allowed to make changes to the source
- The code is spending much of the time in MPI
- What can be done?
Brief Gauss-Seidel Overview

● Code solves this equation (Poisson’s Eqn.)
  ● U is gravitational potential, \( \rho \) is mass, \( \beta \) is a constant that includes the gravitational constant

\[
\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = \beta \rho
\]

● Which can be approximately discretized into

\[
\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{\Delta x^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{\Delta y^2} = \beta \rho_{i,j}
\]

● There is a stencil one zone deep in X and Y
Brief Gauss-Seidel Overview

- Gauss-Seidel is also called red-black

- In 1D the process is
  - Loop until residual is at desired convergence
    1. Red sweep: Update $U_i$ (even i) using fixed latest values for $U_{i+1}$ and $U_{i-1}$
    2. Black sweep: Update $U_i$ (odd i) using fixed latest values for $U_{i+1}$ and $U_{i-1}$
    3. Measure residual

- 2D pattern on the right

http://people.engr.ncsu.edu/efg/506/s01/lectures/notes/lec7.html
Brief Gauss-Seidel Overview

- For multiple MPI ranks the communication needs are
  - Boundary values from nearest neighbors
    - In the normal implementation boundaries must be communicated before each red or black sweep
    - It is possible to do red and black sweeps with only a single set of boundary exchanges
      - This technique, extensions of it and performance implications will be discussed at the end of the tutorial
  - A reduction across ranks for the maximum residual error on the grid
    - In practice this does NOT need to be the latest and greatest value
    - It is just fine to do an extra few sweeps based off an old residual as it does not change the result in any meaningful way
Gauss-Seidel Profile

- Load perftools-lite, recompile and rerun

- Do a ‘pat_report *ap2 > output’ and read the part about Rank reordering

- Change the submit script to use the rank reorder file
  - Did the performance improve? (101 sec)
  - Easy and “free” optimization possibility

---

Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Group</th>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb. %</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>100.0%</td>
<td>6464.8</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>71.2%</td>
<td>4602.2</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>MPI_SENDRECV</td>
<td>40.6%</td>
<td>2625.1</td>
<td>324.9</td>
<td>11.1%</td>
<td></td>
</tr>
<tr>
<td>MPI_ALLREDUCE</td>
<td>24.6%</td>
<td>1588.3</td>
<td>443.7</td>
<td>21.9%</td>
<td></td>
</tr>
<tr>
<td>MPI_CART_CREATE</td>
<td>6.0%</td>
<td>388.7</td>
<td>372.3</td>
<td>49.1%</td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td>27.2%</td>
<td>1758.0</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>gaussseidel$gs_</td>
<td>17.9%</td>
<td>1154.2</td>
<td>73.8</td>
<td>6.0%</td>
<td>gaussseidel$gs_</td>
</tr>
<tr>
<td>localmaxresidual$gs_</td>
<td>5.2%</td>
<td>335.2</td>
<td>52.8</td>
<td>13.7%</td>
<td>localmaxresidual$gs_</td>
</tr>
<tr>
<td>exchangeboundaries$gs_</td>
<td>1.4%</td>
<td>92.2</td>
<td>28.8</td>
<td>23.9%</td>
<td>exchangeboundaries$gs_</td>
</tr>
<tr>
<td>packarray2d$gs_</td>
<td>1.2%</td>
<td>75.4</td>
<td>24.6</td>
<td>24.7%</td>
<td>packarray2d$gs_</td>
</tr>
<tr>
<td>unpackarray2d$gs_</td>
<td>1.1%</td>
<td>73.4</td>
<td>27.6</td>
<td>27.4%</td>
<td>unpackarray2d$gs_</td>
</tr>
<tr>
<td>ETC</td>
<td>1.6%</td>
<td>104.6</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>CPU_TIME_8</td>
<td>1.6%</td>
<td>104.3</td>
<td>43.7</td>
<td>29.6%</td>
<td>CPU_TIME_8</td>
</tr>
</tbody>
</table>
Analyzing the Application
MPI buffer allocation

Boundary exchanges

MPI_Allreduce

!## establish the Cartesian domain decomposition across MPI ranks ##
CALL SetupMPIDomainDecomp()
CALL AllocateMPIBuffers()

!## allocate space to our grids and initialize the array ##
CALL AllocateInitArrays()

!## exchange all of our boundaries of the mass source term ##
CALL ExchangeBoundaries(1)

!## start the iterations loop ##
converged = .FALSE.

nIters = 0

myCPUT = 0.0_R4

myWallT = MPI_WTIME()

DO WHILE (.NOT. converged .AND. nIters .LT. nsweepsMax)

!## run a set of Gauss-Seidel sweeps ##
CALL GaussSeidel()

!## exchange all of our boundaries in the potential solution ##
CALL ExchangeBoundaries(2)

!## check our local residual ##
CALL LocalMaxResidual()

!## get the global residual ##
CALL MPI_ALL REDUCE(myResidual, residual, 1, MPI_REAL8, MPI_MAX, gridComm, ierr)

!## update our flags and counters ##
IF (residual .LE. convCriteria) converged = .TRUE.
nIters = nIters + 1 + (mb-NB_CRIT)/2

END DO

!## this time cannot end until the global residual has been calculated ##

myWallT = MPI_WTIME() - myWallT

!## cleanup everything ##
CALL CleanupAllocations()
CALL Shutdown()
8 boundaries are exchanged (4 total for X and Y, 4 more for diagonals)

Loop has 4 trips exchanging with lower and upper ranks on the 4 rotations (X, Y, +45°, -45°)

MPI_Sendrecv used to
- Send left and recv right
- Recv right and send left

Messages are packed/unpacked using
- packArray2d()
- unpackArray2d()
What should be done?

- Pulling in work from other subroutines is more surgery than we want to start with

- Can boundary exchange be made more efficient with less synchronization?
  - Perhaps we can get some form of overlap with message packing and/or unpacking?

- Something needs to be done to `MPI_Sendrecv` calls, but we’ll restrict changes to within `ExchangeBoundaries()`

- `MPI_Allreduce` will be dealt with later
On your own time…

- The 4th argument to both packArray2d() and unpackArray2d() is the ID of the boundary. It is ALWAYS from the perspective of the caller not the remote end
  - 1,2 = left and right X
  - 3,4 = bottom and top Y
  - 5,6 = bottom left corner, top right corner
  - 7,8 = top left corner, bottom right corner

- To put more messages in flight the MPI buffer must be made bigger

```c
MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
  <type>  BUF(*)
  INTEGER  COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_Irecv(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)
  <type>  BUF(*)
  INTEGER  COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR

MPI_Wait(REQUEST, STATUS, IERROR)
  INTEGER  REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

MPI_Waitall(COUNT, ARRAY_OF_REQUESTS, ARRAY_OF_STATUSES, IERROR)
  INTEGER  COUNT, ARRAY_OF_REQUESTS(*), ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

MPI_Waitany(COUNT, ARRAY_OF_REQUESTS, INDEX, STATUS, IERROR)
  INTEGER  COUNT, ARRAY_OF_REQUESTS(*), INDEX , STATUS(MPI_STATUS_SIZE), IERROR
```
Checking In

- MPI buffer size in AllocateMPIBuffers() must be made bigger to allow more messages in flight.

- The single loop in ExchangeBoundaries() must be broken into at least two loops:
  - Loop 1 posts some non-blocking operations
  - Loop 2 posts remain operations blocking or non-blocking

- Message completion can be done one at a time with MPI_WAIT/MPI_WAITANY or all at once with MPI_WAITALL
Possible Modification You Could Make

● Make MPI buffer this size:
  ● bufLength = 9 * bb * nb * MAX(nx, ny)

● In ExchangeBoundaries(), have two loops
  ● Loop over 8 bounds
    ● Pack message
    ● Post Isend
  ● Loop over 8 bounds
    ● BlockingRecv
    ● Unpack message
  ● Finish with a MPI_WAITALL on outstanding Isends
Possible Paths of Optimization

- MPI_Sendrecv can be converted into
  - Non-blocking MPI_Irecv() followed by blocking MPI_Send and some form of wait
  - Non-blocking MPI_Isend() followed by blocking MPI_Recv and some form of wait
  - Non-blocking MPI_Isend() and MPI_Irecv in either order with some form of wait

- In any path, the single loop over boundaries must be broken into several loops

- Increase in the MPI buffer size from 2 “slots” to at least 9
Non-blocking Send

1. Pack message and post `Isend` for each boundary

```fortran
DO n = 1, 8
    CALL packArray2d(…,n)
    CALL MPI_ISEND(…,handles(n),…)
END DO
```

2. Do blocking `Recvs` unpacking message right after completion

```fortran
DO n = 1, 8
    CALL MPI_RECV(…)
    CALL unpackArray2d(…,nc)
END DO
```

3. Wait for `Isends` to complete

```fortran
CALL MPI_WAITALL(8, handles, …)
```

Code is `gauss_seidel_async2.f90`

- Run time = 67 s
Non-blocking Send then Recv

1. Pack message and post Isend for each boundary
   
   ```fortran
   DO n = 1, 8
       CALL packArray2d(...,n)
       CALL MPI_ISEND(..., handles(n+8), ...)
   END DO
   ```

2. Post Irecvs
   
   ```fortran
   DO n = 1, 8
       CALL MPI_IRECV(..., handles(n), ...)
   END DO
   ```

3. Wait on messages and unpack recvs as they come
   
   ```fortran
   DO n = 1, 16
       CALL MPI_WAITANY(16, handles, nc,...)
       IF (nc .LE. 8) THEN
           CALL unpackArray2d(..., nc)
       END IF
   END DO
   ```

Code is `gauss_seidel_async3.f90`

- Run time = 72 s
Conclusions and Discussion of 1st Work Time

- Of versions I presented the best was async2
  - Non-blocking sends followed by loop of blocking recvs

- Why is that?
  - Think about the pathway messages in this range take
  - ~ 3 KB messages will go through the E0 (eager) pathway for a 256 rank job
  - Transfers are done with SMSG Sends
    - Transfer starts as soon as send is encountered
    -Recv side just copies data out of mailbox

- What would happen if the message size was much bigger?
  - Recall default R0/R1 (rendezvous) cutoff is 4 MB on Aries
  - R0 uses RDMA Gets
  - R1 uses RDMA Puts
What to do with MPI_Allreduce

- There is still significant time spent in MPI_Allreduce

- Since Gauss-Seidel is an iterative solver it does not necessarily need the latest and greatest value of convergence
  - Other example where one can get often away with this are timestep calculations in hydrodynamic solvers

- It should be possible to convert MPI_Allreduce to MPI_Iallreduce
  - Cost will simply be a extra iteration(s)
On your own time (10 minutes…)

- Modify MPI_Allreduce to MPI_Iallreduce

- Use either the best of my versions (async2) or your version if it is faster as a starting point

- Recall that the send buffer must be untouched until the non-blocking allreduce is completed
  - You may need another variable to store allreduce input into

```c
MPI_IALLREDUCE(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, REQUEST, IERROR)
```
- `<type>` SENDBUF(*), RECVBUF(*)
- `INTEGER` COUNT, DATATYPE, OP, COMM, REQUEST, IERROR
Conversion to MPI_Iallreduce

- Link with DMAPP
  - `-Wl,--whole-archive,ldmapp,--no-whole-archive`

- export
  - `MPICH_USE_DMA PP_COLL=MPI_Iallreduce`

- Walltime = 64
What We Accomplished

- Initial walltime before rank reordering and code modification
  - 121 s

- Walltime with just rank reordering
  - 72 s

- Walltime with rank reordering and final code modifications
  - 64 s

- That is almost a 50% improvement!
  - we can always do better
What else could we have done?

- I see two possible options (could be combined)

  1. Merge ExchangeBoundaries() and GaussSeidel() into a single routine in an attempt to overlap more work
     - Would require a 2nd level of decomposition (explicitly or by loop bounds), which would allow partial updates and partial communication to overlap

  2. Reduce the frequency of MPI calls

- Both require quite a bit more effort and understanding of the algorithm
Getting More Overlap

- Create some sub-rank decomposition

- Generate a pipeline of sending some data while updating other

1. Pack and Isend dark blue bounds
2. Wait on dark red bounds
3. Post lrecvs for dark blue bounds
4. Update light red grid
5. Pack and Isend dark red bounds
6. Wait on dark blue bounds
7. Post lrecvs for dark red bounds
8. Update light blue grid
Communication Avoidance

● Why does this code communicate along diagonals?
  ● Algorithm only includes terms along primary X/Y axis

● How is it able to perform red and black sweeps without communication between them?

● Hint, just make the number of boundary zones bigger
  ● In fact, make it completely tunable!
Communication Avoidance

- It is possible to perform red and black sweeps without communication if there are at least 3 boundary zones.

- Base algorithm requires just 1 boundary zone.
Communication Avoidance

- What happens if we increase the number of boundary zones?
  - Memory usage increases, but maybe we don’t care because it’s so much faster

- At \( nb = 13 \)
  - walltime = \( 27.8 \) s
  - Best performance overall
  - 4.3x speedup from original baseline
Communication Avoidance

- What happens if we increase the number of boundary zones?
  - Memory usage increases, but maybe we don’t care because it’s so much faster

- At nb > 15
  - walltime = 28 s
  - Best performance overall
  - ~ 37% speedup from original baseline
Conclusions

- Often reasonable analysis can find MPI bottlenecks especially from blocking MPI calls

- There are a number of pathways for improving blocking point-to-point communication
  - Which one is best often depends on message size

- Sometimes a blocking collective is used when the algorithm might permit non-blocking

- Techniques are available to improve overlap and reduce communication time
  - Example here with sub-rank decomposition

- Sometimes the best optimization is to delay communication/synchronization for as long as possible
  - Redundant work between MPI ranks is not necessarily bad!