Possible Memory Leak in the Octave MPI Package (Formerly Openmpi_Ext Package)

Sukanta Basu
North Carolina State University
Date: 1/1/2014

Over the past few weeks, several parallel simulations utilizing an in-house, Octave/Matlab-based computational fluid dynamics code (named MATLES) were performed to better understand the memory leak problem associated with the MPI package (http://octave.sourceforge.net/mpi/index.html). The MATLES code can use either the MPI package or the MatlabMPI package developed by MIT Lincoln Lab (http://www.ll.mit.edu/mission/cybersec/softwaretools/matlabmpi/matlabmpi.html).

From any user standpoint, the difference in coding syntax between MPI and MatlabMPI is very small. For example:

MPI package uses: MPI_Send(dest,mpitag,comm,var);
MatlabMPI uses: MPI_Send(var,dest,mpitag,comm);

The syntax for MPI_Recv is identical.

The MatlabMPI suffers from significant amount of latency; so, from a computational efficiency point-of-view, it is more desirable to use the MPI package. Unfortunately, this package is causing severe memory leaks. In this brief report, we used the MatlabMPI package to demonstrate that the memory leak is not inherent to our code (i.e., MATLES) or other factors; most likely it is rooted in the MPI package.

To rule out any platform dependency, several computational platforms were utilized:

- Cray CX1 (at NCSU): dual-socket, quad-core nodes - intel X5570 (Nehalem), 2.93 GHz (turboboost); each node has 24 GB RAM; 8192 KB cache; Ubuntu 13.04; openmpi.
- Stampede (at UT Austin): Xeon (E5) nodes with 16 cores and 32 GB RAM; CentOS; mvapich2; for more details please refer to: https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide
- DELL workstation (at NCSU): one node with 8 cores and 12 GB RAM; Ubuntu 13.04; openmpi.
- HP workstation (home computer): one node with 8 cores and 24 GB RAM; Ubuntu 12.10-LTS; openmpi.

To avoid the negative impact of inter-node networking (e.g., Cray CX1 has slow gigabit connection; Stampede uses infiniband), in this report, we mostly used one computational node; in two of the simulations, we used two nodes. Based on all the simulations, it is clear that the memory leak is not dependent on the network.

Instead of using Valgrind, a simple strategy was used to quantify the memory leak. Every fourth iteration of the simulation, the following system command was invoked (inside MATLES) to measure and output available RAM in the system.

```
system('free -m | grep "buffers/cache" | cut -c 36-40 >> Memory_Usage.out');
```

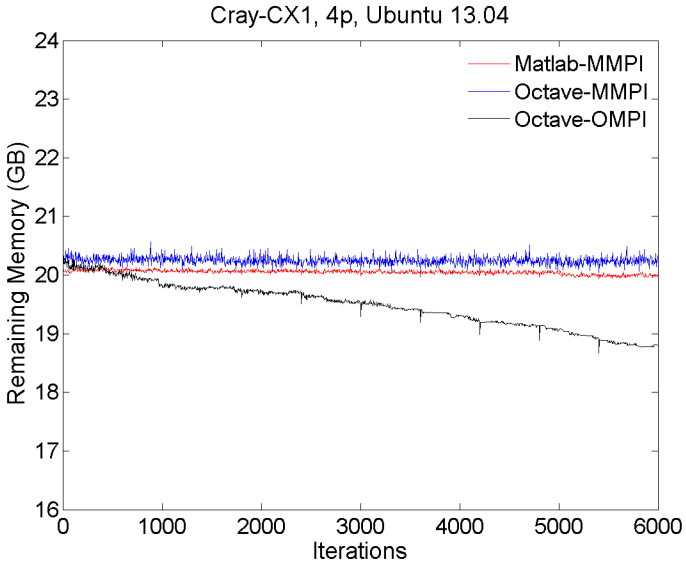
Note:

- this command accounts for buffers/cache.
- this command is only executed by one of the processors (rank = 0).

Case 1:

In the figure below, the memory leaks from 3 simulations using fixed number of cores (4 cores to be specific) are compared. The simulations which use MatlabMPI (MMPI) do not show any sign of memory leak. In contrast, the simulation using the MPI (or openmpi_ext) package (OMPI) shows a loss of 1 GB of RAM within 6,000 iterations.

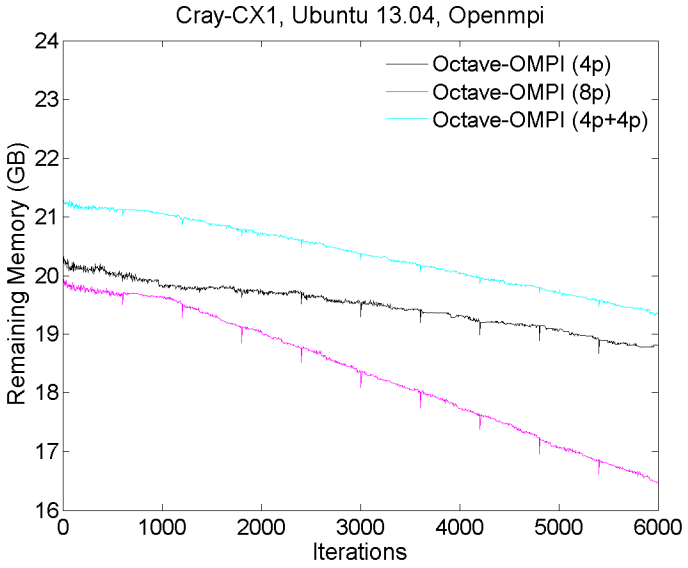
From this figure, it can be concluded that the memory leak is not inherent to: (i) Octave/Matlab or (ii) the in-house MATLES code.



Case 2:

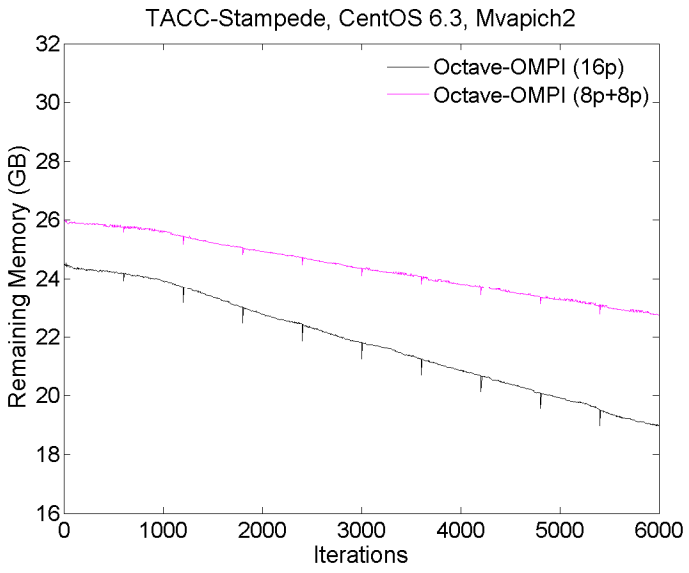
Next, the effects of the number of cores on the memory leak are studied. The aforementioned simulation is run with 4 and 8 cores, respectively. From the figure below, it is clear that with increasing number of cores (i.e., increasing communications), the memory leak increases proportionally.

Note: the slower decay of 4p+4p run (i.e., 2 nodes each having 4 cores) in comparison with the 8p run (i.e., using all the 8 cores on a node) is deceiving. As noted before, only the rank = 0 processor outputs memory loss information. So, in the case of the 4p+4p run, only the loss of RAM on node #1 is quantified; one would expect similar level of memory loss on node #2.



Case 3:

So far, only Ubuntu was used in conjunction with openmpi. One might argue that the memory leak could be due to OS and/or native mpi implementation. For this reason, the MATLES code was tested on TACC's Stampede system running CentOS. In this case, mvapich2 was used. Clearly, the memory leak problem persists.

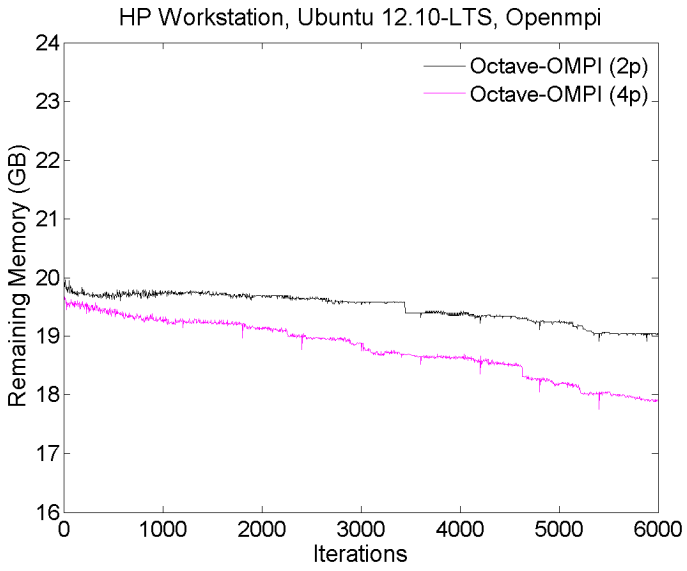


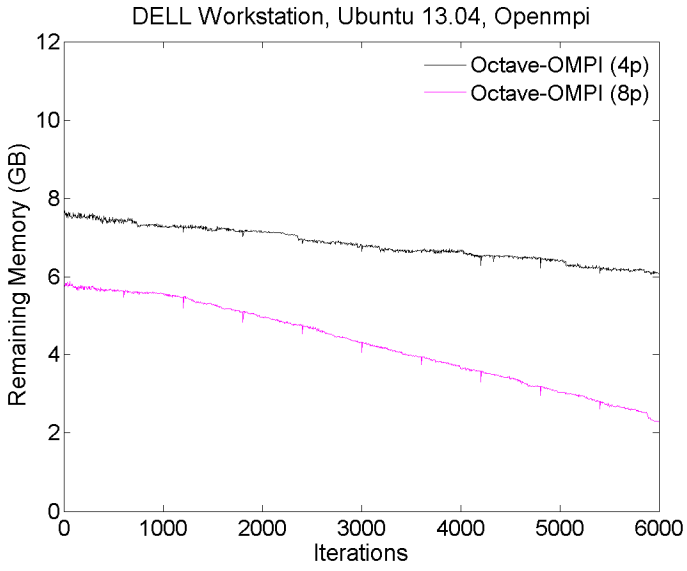
Note: we have not been successful in running our code with mpich2. The installtion of MPI package with mpich2 was successful; however, the code blows up after several hours of simulation. The error message was not clear to us:

Fatal error in PMPI_Type_contiguous: Other MPI error, error stack:
PMPI_Type_contiguous(144): MPI_Type_contiguous(count=4096, MPI_DOUBLE,
new_type_p=0x7fffecab1ffc) failed
PMPI_Type_contiguous(130):
MPID_Type_contiguous(46).: Out of memory
Fatal error in PMPI_Type_contiguous: Other MPI error, error stack:
PMPI_Type_contiguous(144): MPI_Type_contiguous(count=4096, MPI_DOUBLE,
new_type_p=0x7fff3cf2381c) failed
PMPI_Type_contiguous(130):
MPID_Type_contiguous(46).: Out of memory
panic: Segmentation fault -- stopping myself...
panic: Segmentation fault -- stopping myself...
attempting to save variables to 'octave-core'...
attempting to save variables to 'octave-core'...
error: octave_base_value::save_binary(): wrong type argument 'simple'
save to 'octave-core' complete
error: octave_base_value::save_binary(): wrong type argument 'simple'
save to 'octave-core' complete

Case 4:

The MATLES code was further tested on two workstations with different RAM availability and OS. The results corroborate previous findings (see below).





In summary, we are more or less convinced that the memory leak is coming from the MPI package. Other factors might play minor roles in this problem.