5. Use the Berendsen Thermostat (ntt=1) or Anderson Thermostat (ntt=2) instead of the Langevin Thermostat (ntt=3). Langevin simulations require very large numbers of random numbers which slows performance slightly.

6. Do not assume that for small systems the GPU will always be faster. Typically for GB simulations of less than 150 atoms and PME simulations of less than 9,000 atoms it is not uncommon for the CPU version of the code to outperform the GPU version on a single node. Typically the performance differential between GPU and CPU runs will increase as atom count increases. Additionally the larger the non-bond cutoff used the better the GPU to CPU performance gain will be.

7. When running in parallel across multiple GPUs you should NOT attempt to share nodes and thus interconnects. For example you should avoid running 2 separate MPI jobs on individual nodes (unless both are using peer to peer). For example if you have 2 nodes, each with a QDR IB card in, 1 M2090 and 1 C2050 you will likely get very poor performance if you attempt to run a dual GPU job on the 2 M2090's and a second dual GPU job on the 2 C2050's. It is also not advisable to mix GPU models when running in parallel. In this situation you are advised to physically place both M2090's in one node and both C2050's in the other. You could of course run a dual M2090 job across the two nodes and then 2 single GPU jobs on each of the C2050's.

8. When running in parallel you should restrict jobs to a single node and select GPUs that are on the same IOH controller and thus can communicate via peer to peer. For most 4 GPU nodes gpus 0 and 1 can typically communicate via peer to peer and gpus 2 and 3 can communicate via peer to peer. The mdout file contains a section that indicates if peer to peer support is enabled.

9. Avoid using CUDA Toolkit v5.5 or v6.0. At the time of writing the recommended CUDA Toolkit and compiler is v5.0. This provides maximum performance. A performance regression bug in v5.5 means that it is about 5 to 8% slower than v5.0.

10. Turn off ECC (Tesla models C2050 and later). ECC can cost you up to 10% in performance. You should verify that your GPUs are working correctly, and not giving ECC errors for example before attempting this. You can turn this off on Fermi based cards and later by running the following command for each GPU ID as root, followed by a reboot:

    `nvidia-smi -g 0 --ecc-config=0` (repeat with -g x for each GPU ID)

    Extensive testing of AMBER on a wide range of hardware has established that ECC has little to no benefit on the reliability of AMBER simulations. This is part of the reason it is acceptable (see recommended hardware) to use the GeForce gaming cards for AMBER simulations.

11. Turn on boost clocks if supported. Newer GPUs from NVIDIA, such as the K40, support boost clocks which allow the clock speed to be increased if there is power and temperature headroom. This must be turned on to obtain optimum performance with AMBER. If you have a K40 or newer GPU supporting boost clocks then run the following:

12. `sudo nvidia-smi -i 0 -ac 3004,875` which puts device 0 into the highest boost state.

13. To return to normal do: `sudo nvidia-smi -rac`

14. To enable this setting without being root do: `nvidia-smi -acp 0`

18.7 Intel® Many Integrated Core Architecture

One of the newest features of PMEMD is support for Intel MIC Products, such as Intel® Xeon Phi™ coprocessors. Intel MIC Architecture support in PMEMD includes both native and offload mode in Amber 14. These versions of the PMEMD engine are considered experimental and there is no guarantee of any performance improvement. However, this support should produce results directly comparable to the CPU implementation due to the floating-point consistency of Intel's processors. Any differences in tests will be a consequence of floating-point

rounding differences in hardware. Support for Intel® MIC Architecture in PMEMD is an ongoing project and therefore frequent updates are likely. Improved performance can be expected in upcoming patches.

Support for the Intel MIC Architecture in Amber 14 offers two modes:

- **MIC native mode:** allows all the functionality of the PMEMD engine to be run on Intel® Xeon Phi™ coprocessors in native mode in either serial or parallel (MPI).

- **MIC offload mode:** offloads a portion of the direct sum calculation in a standard PME simulation to the Intel® Xeon Phi™ coprocessor for improved performance. We are anticipating in later releases to extend support to most functionality of PMEMD, which would include offloading to the Intel® Xeon Phi™ coprocessor for Thermodynamic Integration (TI), Isotropic Periodic Sum (IPS) and pmemd.amoeba simulations. Recommended system size to be explored using the MIC offload version of PMEMD is >200,000 atoms in order to benefit from performance improvements. The MIC offload version of PMEMD requires Intel®'s MPI library which is included in the Intel® Cluster Studio XE compiler package.

The supported Intel® Xeon Phi™ product family includes the Intel® Xeon Phi™ coprocessor 3100/5100/7100 series. (*Intel, Xeon, and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.*)

Before attempting to build these versions you should have built and tested the serial and parallel CPU versions of Amber (*pmemd* and *pmemd.MPI*) with the Intel compiler suite and Intel MPI. This will help to ensure that basic issues relating to standard compilation on you hardware and operating system do not lead to confusion with coprocessor related compilation problems. It is recommended that you are familiar with building and running simple code in native/offload mode on an Intel® Xeon Phi™ Coprocessor, which is described in *https://software.intel.com/mic-developer*. You should also be familiar with Amber's compilation procedures.

18.7.1 MIC Native Model

The MIC native version supporting Intel® Xeon Phi™ coprocessors is called *pmemd.mic_native* (or *pmemd.mic_native.MPI* for running simulations in parallel on the coprocessor using MPI) and must be built separately from the standard serial and parallel installations.

Building PMEMD serial for Intel® Xeon Phi™ Coprocessors (native mode)

Assuming you have installed Intel® Parallel Studio XE version 2013 or later, you can build *pmemd.mic_native* as follows:

```
cd $AMBERHOME
./configure -mic_native intel
make install
```

Building PMEMD parallel for Intel® Xeon PhiTM Coprocessors (native mode)

PMEMD parallel for Intel® Xeon Phi™ coprocessors can only be built using the MPI (mpiicc/mpiifort) from the Intel® Cluster Studio XE product, which is supported in Amber 14 through the use of a new MPI flag (-intelmpi). Build *pmemd.mic_native.MPI* as follows:

```
cd $AMBERHOME
./configure -mic_native -intelmpi intel
make install
```

It is possible to run across multiple Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors, even on a cluster, with this implementation. However, it is a functional implementation and not performance optimized at this time. Detailing how to run this way is beyond the scope of the current manual, however to see how to do this with MPI applications in general, see

http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems .

Running simulations on Intel® Xeon Phi™ Coprocessors (native mode)

In order to run a simulation on the Intel® Xeon Phi™ coprocessor, it is advised that you read the Intel® Xeon Phi™ Coprocessor Developer's Quick Start Guide (*http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide*) in addition to the instructions given here. This guide includes a description of the Intel® Manycore Platform Software Stack (Intel® MPSS), which enables the wide range of usage models that Intel® Xeon Phi™ coprocessors support. Running a simulation on the coprocessor in native mode requires that all files and binaries be visible to the coprocessor. Either mount your file system on the coprocessor (requires root access) or explicitly transfer the binaries, libraries and input files to the */tmp* directory on the coprocessor.

Note: Mounting requires the Amber directory plus any additional libraries used in an Amber simulation to be visible to the coprocessor

Running simulations with a mounted filesystem:

1. To mount a filesystem please follow instructions available in the Intel MPSS readme file. You can also follow the instructions in *http://software.intel.com/sites/default/files/article/373934/system-administration-for-the-intel-xeon-phi-coprocessor.pdf*

2. Mount your AMBERHOME directory, working directory, Intel compiler directory, Intel MPI_HOME directory (for parallel run), and MKL_HOME directory (if MKL is used) on the coprocessor.

3. Add the following environment variables to a file (source_knc.sh in this example), which will be sourced on execution of *pmemd.mic_native*:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$INTEL_COMPILER_HOME/ lib/mic/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: $MKL_HOME/lib/mic/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: $MPI_HOME/mic/lib/
export PATH=$PATH:$MPI_HOME/mic/bin
```

4. Run a simulation from your working directory on the host (CPU) with a mounted filesystem:

```
ssh mic0 "source source_knc.sh; \
$AMBERHOME/bin/pmemd.mic_native -O -i mdin -o mdout -p prmtop -c inpcrd"
```

Running simulations without a mounted filesystem:

1. Upload Intel® Xeon Phi™ coprocessor version of the Intel compiler library to the coprocessor:

```
scp -r $INTEL_COMPILER_HOME/lib/mic/* mic0:/tmp/
```

2. Similarly, upload coprocessor versions of the MPI and MKL libraries:

```
scp -r $INTEL_MPI_HOME/mic/lib/* mic0:/tmp/
scp -r $INTEL_MPI_HOME/lib/mic/* mic0:/tmp/
```

3. Upload the coprocessor version of PMEMD (*pmemd.mic_native* or *pmemd.mic_native.MPI*) and your working directory (containing the input fies for simulation) to mic0:

```
scp -r $AMBERHOME/bin/pmemd.mic_native mic0:/tmp/
scp -r working_directory/* mic0:/tmp/
```

4. Change the permission of the libraries and binaries so that they are executable on the coprocessor:

```
chmod 777 -R /tmp/*
```

5. Finally run the simulation from host:

```
ssh mic0 "Export LD_LIBRARY_PATH=/tmp/; cd /tmp; \
./pmemd.mic_native -O -i  mdin -o mdout -p prmtop -c inpcrd"
```

18.7.2 MIC Offload Mode

The MIC offload version supporting Intel® Xeon Phi™ coprocessors is called *pmemd.mic_offload.MPI* and must be built separately from the standard parallel installation. MIC offload is not available in serial.

Building PMEMD for Intel® Xeon Phi™ Coprocessors (offload mode)

Assuming you have installed Intel® Parallel Studio XE version 2013 or later, you can build *pmemd.mic_offload.MPI* as follows:

```
cd $AMBERHOME
./configure -mic_offload intel
make install
```

There is no need to specify the –intelmpi flag as this is the default behavior of the configure script.

Testing PMEMD for Intel® Xeon PhiTM Coprocessors (offload mode)

You can run the test suite using the MIC coprocessor with:

```
make test.mic_offload
```

The majority of these tests should pass. However, given the parallel nature of the MIC coprocessor, meaning the order of operation is not well defined, it is not uncommon for there to be several "possible FAILURES". You should inspect the *.diff* file created in the *$AMBERHOME/logs/test_amber_mic_offload/* directory to manually verify any "possible FAILURES". Differences that occur on only a few lines and are minor in nature can be safely ignored. Any large differences, or if you are unsure, should be posted to the Amber mailing list for comment.

Running simulations on Intel® Xeon Phi™ Coprocessors (offload mode)

Unlike MIC native mode, once PMEMD has been configured with the –mic_offload flag and compiled, no additional steps are required to run *pmemd.mic_offload.MPI*. Work is automatically offloaded to the Intel MIC Architecture.

Execute the following command on the host to run a simulation in MIC offload mode:

```
mpirun -np 8 $AMBERHOME/bin/pmemd.mic_offload.MPI -O
```

Note: Choose the number of MPI processes to suit the specifications of the host CPU, i.e. 8 MPI processes for an Intel Xeon E5-2680 8 core processor, in order to achieve optimum performance. In version 2 support for MIC offload in PMEMD, the amount of offloaded work to the coprocessor increases and settles to a stable value after running multiple time steps governed by the Amber load balancer. Thus, it is recommended that the simulation should run for at least 200 time steps to benefit from the coprocessor.

Compiler switches for performance:

PMEMD MIC offload version 2 includes optional compiler switches that can be used to improve the performance of simulations of systems with greater than 200,000 atoms. These are disabled by default but can be enabled by compiling with:

```
make AMBERBUILDFLAGS=' [ options ] '
```

Options

-DOffload_excludes_recip This switch ensures that the offloading MPI ranks are not assigned reciprocal force calculations allowing for more efficient concurrent execution of the direct sum between the host and the MIC.

–Doffload_excludes_bond This switch ensures that the offloading MPI ranks are not assigned bonded (bond, angles and dihedrals) force calculations allowing for more efficient concurrent execution of the direct sum between the host and the MIC, similar to the switch described above.

-DFaster_erfc This switch enables a custom lower precision implementation of the erfc function instead of the standard math "erfc" function call that is used in calculating the direct space force contributions.

The compiler switches described above do not guarantee any performance improvement. It is advised that the user tests the switches one by one before using them in a production environment.

Adjusting the stack size

For larger simulations (>100,000 atoms) more OMP threads are spawned on a MIC coprocessor to provide better performance. However, for more OMP threads we need a larger stack size per thread and a larger total stack size on a MIC coprocessor. The default stack size is 8 KB.

- "-env MIC_OMP_STACKSIZE 4M" increases the OMP thread stack size to 4 MB.

```
mpirun -env MIC_OMP_STACKSIZE 4M -np 8 $AMBERHOME/bin/pmemd.mic_offload.MPI -O
```

- For MPSS version 3.2.3 and later the total stack size of a MIC coprocessor is increased by the following steps:

1. On the host, as root, create the directories *etc/* and *etc/security* in */var/mpss/common*:

```
cd /vars/mpss/common
su
mkdir -p etc/security
```

2. Next create the file *limits.conf* (in */var/mpss/common/etc/security*) containing the following line of text (with tab separated values):

```
"*      soft    stack           unlimited"
```

3. Create the file */var/mpss/common.filelist* containing the following (with space separated values) lines:

```
dir /etc/security 755 0 0
file /etc/security/limits.conf etc/security/limits.conf 644 0 0
```

4. Finally, cycle the MPSS daemon to reboot the cards using:

```
service mpss restart
```

18.8 pmemd.amoeba

The Amoeba force field is a recently developed polarizable force field with parameters for water, univalent ions, small organic molecules and proteins.[313, 314, 330–332] Differences from the current amber force fields include more complex valence terms including anharmonic bond and angle corrections and bond angle and bond dihedral cross terms, and a two dimensional spline fit for the phi-psi bitorsional energy. The differences in the nonbond treatment include the use of atomic multipoles up to quadrupole order, induced dipoles using a Tholé screening model, and the use of the Halgren buffered 7-14 functional form for van der Waals interactions. The PME implementation used here, as well as a multigrid approach for atomic multipoles, is described in Ref. [320].

Right now, setting up the system is a bit complex: you need to set up the system in Tinker, then run the *tinker-to-amber* program to convert to Amber prmtop and coordinate files. Some examples are in *$AMBERHOME/src/pmemd.amoeba/build_amoeba*. But keep checking the Amber web page, since we hope to provide a simpler path soon.