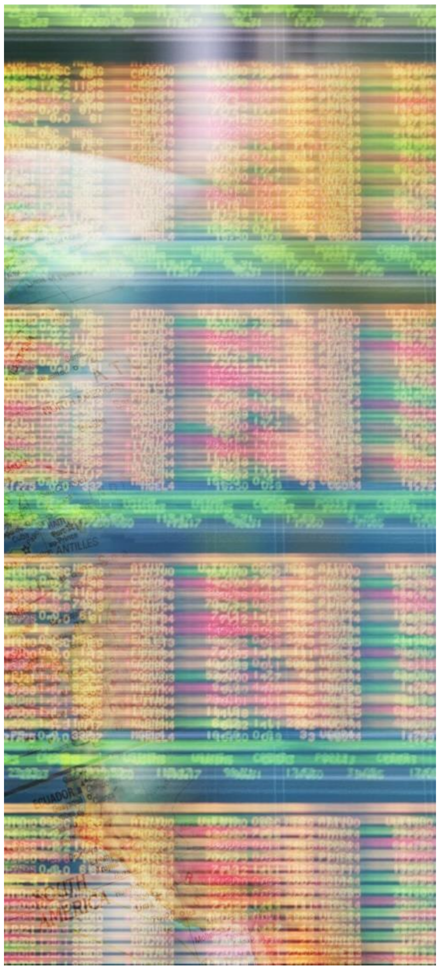
Optimization Guide

Intel® Xeon® Scalable Processor

Recipe: Build and Run NAMD on Intel® Xeon® Processors



Revision Record

Date	**Revision**	**Description**
01/20/2022	1.0	Initial draft

Purpose

This recipe describes a step-by-step process for building and running the NAMD* molecular dynamics application on Intel® Xeon® Scalable Processors.

Introduction

NAMD is a parallel molecular dynamics application designed for high-performance simulation of large biomolecule systems. Based on Charm++* parallel objects, NAMD scales to hundreds of cores for typical simulations and beyond 500,000 cores for the largest simulations. NAMD uses the popular molecular graphics program VMD* for simulation setup and trajectory analysis, but is also file-compatible with AMBER*, CHARMM*, and X-PLOR*.

NAMD has been optimized for Intel® Advanced Vector Extensions 512 (Intel® AVX-512). These optimizations significantly improve performance on supported processors. They will be included in NAMD version 2.15 but early use is available by building NAMD from its git repository. These builds will use the Intel AVX-512 optimizations by default whenever supported by the simulation, as indicated by the following line in the initial NAMD run output:

```
MIXED PRECISION AVX-512 TILES OPTIMIZATIONS: ENABLED
```

The optimizations may also be disabled for any simulation at run-time by either adding the "+notiles" option to the commandline or adding "useAVXTiles no" to the input simulation script.

In general, the build and run procedure is unchanged from typical NAMD procedures. However, it is important to use the latest NAMD code that includes the optimizations. Additionally, an appropriate NAMD "architecture" file should be chosen to enable the optimizations with the required compiler flags.

Downloading the benchmarks

Here, we use the popular APOA1 and STMV benchmarks available from the NAMD site. The simulation parameters for these benchmarks are often modified in a manner that can significantly affect performance. We use the standard simulation parameters, with modifications to increase the duration of the simulation and reduce energy output as recommend for benchmarking by the NAMD developers. To download the benchmarks and perform these modifications:

1. Create a directory for the work described in this recipe.

```
$ mkdir namd_global

$ cd namd_global
```

2. Download the benchmarks from the NAMD site and modify them for longer runs with reduced energy output.

```
$ wget http://www.ks.uiuc.edu/Research/namd/utilities/apoa1.tar.gz

$ wget http://www.ks.uiuc.edu/Research/namd/utilities/stmv.tar.gz

$ tar -zxvf apoa1.tar.gz; tar -zxvf stmv.tar.gz

$ sed -i -e '/numsteps/s/500/1000/' apoa1/apoa1.namd stmv/stmv.namd
$ sed -i -e "/outputtiming/a\\outputenergies 600" apoa1/apoa1.namd

$ sed -i -e '/outputEnergies/s/20/600/' stmv/stmv.namd
```

Downloading the code

Charm++ is required by NAMD, and the TCL* library is an optional component for full scripting support.

3. (Optional) Download TCL. This step can be skipped if TCL support is not desired, or if TCL is already installed on the system (on some systems, one can check for existing TCL with "locate libtcl8.5").

```
$ git clone https://github.com/tcltk/tcl.git -b core-8-5-branch --depth 1

$ NAMD_TCL_BASE="`pwd`/tcl"
```

4. Download Charm++ and NAMD:

```
$ git clone https://charm.cs.illinois.edu/gerrit/namd.git -b release-2-15-alpha-1 --depth 1

$ cd namd; git clone https://github.com/UIUC-PPL/charm.git -b v6.10.2 --depth 1

$ cd ..
```

Build NAMD for single-process benchmarking

The build and run procedure for single-process executables is simpler, however runs are restricted to a single node and a single NAMD communication thread. When the number of atoms per core is small, multiple processes with multiple communication threads can improve performance, even when running on a single node. We describe the build process for both, starting with single-process executables:

5. Setup the compiler and library environment. Here, we use the Intel® MPI Library, Intel® Math Kernel Library (Intel® MKL), and the Intel® C++ Compiler. For Intel® Parallel Studio XE, the environment can be setup for the Bash shell with a single command. The example below is for Intel® Parallel Studio XE 2020:

```
$ source /opt/intel/parallel_studio_xe_2020.2.108/psxevars.sh
```

6. (Optional) Build the TCL library

```
$ cd tcl/unix; ./configure --disable-shared --prefix=$NAMD_TCL_BASE

$ make install -j; cd ../../
```

7. Build the single-process Charm++ library

```
$ export CC=icc; export CXX=icpc; export F90=ifort; export F77=ifort

$ cd ./namd/charm

$ ./build charm++ multicore-linux64 iccstatic --with-production "-O3 -ip -xCORE-AVX512 -qopt-zmm-
usage=high"

$ cd ../
```

8. Build the single-process NAMD executable. If building without TCL, change "--tcl-prefix $NAMD_TCL_BASE" to "--without-tcl"

```
$ ./config Linux-AVX512-icc --with-mkl --tcl-prefix $NAMD_TCL_BASE

$ cd Linux-AVX512-icc; make -j; cd ../../
```

Run the benchmarks

For smaller workloads or high node counts, better performance might be achieved without using all hyper-threads. NAMD performance can be measured by the simulation rate in nanoseconds per day (ns/day). Higher is better.

9. (Optional) A small performance improvement can be achieved by using Intel® Threading Building Blocks (Intel® TBB)memory allocation

```
$ export LD_PRELOAD="`ls $TBBROOT/lib/intel64/gcc*/libtbbmalloc_proxy.so` $LD_PRELOAD"
```

10. Set the total number of cores to use on the node for NAMD runs and the variable with the command to extract performance

```
$ N_CORES=`grep processor /proc/cpuinfo | wc -l`

$ H_CORES=`expr $N_CORES / 2`

$ GET_PERF="\$2==\"Benchmark\"{n++; s+=log(\$8); print \"Instant:\",1/\$8,\" ns/day\"}END{print
\"Final:\",1/exp(s/n)}"
```

11. Run the APOA1 and STMV benchmarks:

```
$ ./namd/Linux-AVX512-icc/namd2 +p $N_CORES +setcpuaffinity ./apoa1/apoa1.namd | awk "$GET_PERF"

$ ./namd/Linux-AVX512-icc/namd2 +p $H_CORES +setcpuaffinity ./apoa1/apoa1.namd | awk "$GET_PERF"

$ ./namd/Linux-AVX512-icc/namd2 +p $N_CORES +setcpuaffinity ./stmv/stmv.namd | awk "GET_PERF"
```

Build and run NAMD for multi-process benchmarking

For multi-node benchmarks, optimal build and run options can depend on the cluster. Here, we use the MPI backend for Charm++ (OFI* can also be used) and we use the Intel® MPI launch commands. Please consult the documentation for NAMD and your cluster for details.

12. Build Charm++ using the Intel® MPI Library:

```
$ CC=icc; CXX=icpc; F90=ifort; F77=ifort; MPICXX=mpiicpc; MPI_CXX=mpiicpc

$ I_MPI_CC=icc;I_MPI_CXX=icpc;I_MPI_F90=ifort;I_MPI_F77=ifort

$ export I_MPI_CC I_MPI_CXX I_MPI_F90 I_MPI_F77 CC CXX F90 F77 MPICXX MPI_CXX

$ cd ./namd/charm

$ ./build charm++ mpi-linux-x86_64 smp mpicxx --with-production "-O3 -ip -xCORE-AVX512" -
DCMK_OPTIMIZE -DMPICH_IGNORE_CXX_SEEK

$ cd ../
```

13. Build NAMD with Intel® MPI Library support. If building without TCL, change "--tcl-prefix $NAMD_TCL_BASE" to "--without-tcl"

```
$ cp arch/Linux-AVX512-icc.arch arch/Linux-AVX512MPI.arch

$ ./config Linux-AVX512MPI --charm-base ./charm --charm-arch mpi-linux-x86_64-smp-mpicxx --with-mkl --tcl-prefix $NAMD_TCL_BASE

$ cd Linux-AVX512MPI

$ make -j

$ cd ../../
```

14. (Optional) Specify that memory allocation should use the Intel® Threading Building Blocks (Intel® TBB) library

```
$ export LD_PRELOAD="`ls $TBBROOT/lib/intel64/gcc*/libtbbmalloc_proxy.so` $LD_PRELOAD"
```

15. Set the number of cores to use on each node

```
$ N_CORES=`grep processor /proc/cpuinfo | wc -l`
```

16. (Optional) Adjust core count to avoid use of Intel® Hyper-Threading Technology when the number of atoms per core is small

```
$ N_CORES=`expr $N_CORES / 2`
```

17. Choose the number of NAMD processes to use on each node. The best choice depends on the number of atoms per core and the system configuration. The number should be chosen such that the N_CORES setting is evenly divisible. In this example, we use 4.

```
$ NPPN=4
```

18. Set the total number of MPI processes and the affinity flags for NAMD communication and worker threads (assumes that $NODES is set to the number of nodes

```
$ NMPI=`expr $NPPN \* $NODES`

$ NAFFIN=`echo $N_CORES $NPPN | awk '{p=($1-$2)/$2; c=$1-1; f=p+1; print "+ppn",p,"+commap ",0"-"c":"f,"+pemap 1-"c":"f"."p}'`
```

19. Perform the run. Performance can be parsed from the output in an identical manner to single-process benchmarks (not shown below).

```
$ mpirun -ppn $NPPN -f $HOSTFILE -np $NMPI ./namd/Linux-AVX512MPI/namd2 $NAFFIN ./stmv/stmv.namd
```

Expected Performance

Expected performance numbers for single-node benchmarks with Intel® Hyper-Threading Technology and Intel® Turbo Boost Technology enabled (ns/day; higher is better) are given below:

	Intel® Xeon® Scalable Processor			
Workload	Platinum 8268	Platinum 9221	Platinum 9242	Platinum 9282
STMV	1.56	1.86	2.62	3.07
APOA1	18.1	20.7	27.3	30.8

Performance Measurement Details

Expected performance numbers were measured by Intel® corporation and collected using NAMD version 2.14-beta-1 with a patch to include the code for the optimizations now available from the download procedure described above. Intel® Parallel Studio 2020 update 1 was used for the build following the recipe described in this document. Performance numbers were measured on systems running CentOS* Linux 7.7.1908 with Intel® Hyper-Threading Technology and Intel® Turbo Boost Technology 2.0 enabled. The Intel® Xeon® Platinum 8268 (2 sockets, 24 cores, 2,9GHz base frequency) system was configured with 12x16GB 2933MHz DDR4 memory. The Intel® Xeon® Platinum 9221 (2 sockets, 32 cores, 2.3GHz base frequency), Intel® Xeon® Platinum 9242 (2 sockets, 48 cores, 2.3GHz base frequency), and Intel® Xeon® Platinum 9282 (2 sockets, 56 cores, 2.6GHz base frequency) systems were configured with 24*16GB 2933MT/s DDR memory.

Conclusion

Use this section as a brief wrap-up for the guide.

Example: We understand every application is unique. We shared many of our experiences with MySQL and PostgreSQL hoping that some of our learnings could be applied to your specific application. Both Open-Source relational database management systems have been well tested on Intel platforms. With 3rd Generation Intel® Xeon® Scalable processor, Intel takes it even further by optimizing the platform as a whole – CPU, memory, storage, and networking working together for the best user experience.

Feedback

We value your feedback. If you have comments (positive or negative) on this guide or are seeking something that is not part of this guide, please reach out and let us know what you think.