Some practical aspects in using LAMMPS

LAMMPS can be run in

1. different pallallel schemes - MPI or openMP

2. a single PC or computer cluster

Running LAMMPS in sequential mode in a single machine
===

In a LAMMPS, the executable lmp_machine is located in /src, where machine is a name used to differentiate the many ways lammps is compiled. For convenience we usualy place lmp_machine in the path of a user, e.g., in /home/user/.bashrc,

```
alias lmp_machine="/home/user/mylammps_5March_12/src/lmp_machine"
```

To run a code with no parallalisation on a single machine,

```
mpiexec lmp_machine -c off < in.melt
```

The option "-c off" means shuting off the cuda capability. By default all lammps in a machine that support cuda will be installed with cuda capability. But whether "-c on" can be evoked depends whether a lammps input script is cuda supported. To swith on cuda, use "-c on". For machine in which lammps is not compiled using cuda, the switch "-c off" can be omitted.

mpiexec could be the intel version of mpiexe (as found in /intel/impi/.../bin64) or gnu version (as found in /mpich2/bin). One could search for mpiexec by issuing

```
which mpiexec
```

to find out which mpiexec is being used by the computer's default (there could be more than one mpiexec in a single machine). If you are not sure which mpiexec is the default, simply specific the exact path for mpiexec. e.g.,

```
/usr/local/mpich2/bin/mpiexec lmp_machine -c off < in.melt
```

If impi's mpiexec is used, make sure to launch "mpdboot", an executale found in intel/impi folder, such as in /opt/intel/impi/4.1.0.024/bin64/mpdboot. One only needs to mpdboot on the first time only.

Running LAMMPS in parallel mode with MPI in a single machine
==

When installing lammps, it must be compiled with either gnu compilesr {mpich2/bin/mpicc, g++}, or intel compilers {impi/bin64/mpiicc, ifort, icc}. It has been benchmarked that intel parallalisation scheme run slightly faster than the gnu parallisation scheme (faster by 1.5 seconds for each 52 seconds). But the problem is that lammps compiled using intel has problem hanlding openmp at times, presumably due to incompatibality between impi library with libstdc++.

In a single computer, you may find more than one versions of lmp_machine. For example, you will find folder Obj_gnu, Obj_intel which contain respectively executable lmp_intel and lmp_gnu. These are lammps compiled using different Makefiles. For example, if Makefile.intel is used to compile the lammps, we got Obj_intel and lmp_intel; If Makefile.gnu is used, we got Obj_gnu and lmp_gnu instead. Makefile.machine found in /mylammps/MAKE/ contains the info of compilers and MPI scheme used to compile lmp_machine.

You should first determine in the present computer how many lmp_machines are there. You may find for example two versions of lammps exectables have been compiled, such as lmp_intel and lmp_gnu. Decide which lmp_machine you want to use.

To run a lammps input script in.melt using two cores sitting in the same cpu, issue the command

```
mpiexec -np 2 lmp_machine -c off < in.melt
```

It is advised to use mpiexec that is compatible with lmp_machine. For example, for lmp_intel, suggest to use impi/../bin64/mpiexec. However, this is not mandatory. It is possible to use mpich2/bin/mpiexec in place of impi/../bin64/mpiexec, despite lmp_intel is compiled using intel mpi.

Running LAMMPS in parallel mode with OPENMP in a single machine
===

Alternatively, one can also run lammps in parallel using multi-thread instead of MPI. This is known as OPENMP. Both lmp_intel and lmp_gnu support openmp.

To run openmp with lmp_intel, one must place the following in his /her .bashrc:

```
export I_MPI_PIN_DOMAIN=omp
```

For example, to run a lmp_intel using 2 threads and np = 2, one first set the value of the environment variable $OMP_NUM_THREADS to 2. To do so, one may either type

```
export OMP_NUM_THREADS=2
```

To run lmp_intel in openmp mode, add the option "-sf omp" into the command line

```
mpiexec -np 2 lmp_intel -c off -sf omp < in.melt
```

Alternatively, one can do the same thing in a single line of command:

```
export OMP_NUM_THREADS=2; mpiexec -np 2 lmp_machine  -c off -sf omp < in.melt
```

To check out the current thread number $OMP_NUM_THREADS, one type

```
echo $OMP_NUM_THREADS
```

Alternatively, one may set the value of the environment variable OMP_NUM_THREADS in ./bashrc

```
export OMP_NUM_THREADS=4
```

By default, OMP_NUM_THREADS=1 is set in .bashrc.

We also note that due to unknown techical reasons, "-sf omp" may not work in lammps compiled using intel compilers.

On the other hand, "-sf omp" works fine for lmp_gnu.

What combination of parallelisation is the best for running LAMMPS in a single PC?
===

It is found that, after some benchmarking, in a single machines with 4 cores, such as asus u36s series with a intel Core I5, the following combinations run approximately equally fast

```
export OMP_NUM_THREADS=4; mpirun -np 1 lmp_machine -c off -sf omp < in.melt

export OMP_NUM_THREADS=1; mpirun -np 4 lmp_machine -c off -sf omp < in.melt
export OMP_NUM_THREADS=2; mpirun -np 2 lmp_machine -c off -sf omp < in.melt
```

Since the total core number in asus U36S is Ncore = 4,

(omp + np) = Ncore

seems to be the optimal combination to use if one wishes to complete a lammps run in short wall time in a single PC.

FFTW3
=====

One also has to choose whether to use gnu fftw3 or intel's fftw. Experience shows that intel fftw2 or fftw3 often clashes with lammps. So by default we abondon intel fftw. We shall use gnu fftw3 throughtout.

Running LAMMPS in a cluster environment

Preparing cluster to run MPI
===================

To run lammps in parallel with MPI which involves distinct computer nodes, one needs to ask the present machine (e.g. Comsics or Anicca) to be 'mpi-ready' before the command line such as

```
export OMP_NUM_THREADS=$OMP; mpiexec -np $np lmp_gnu -c off -sf omp < in.melt_test > $np.$OMP.out &
```

can be correctly executed from within the frontend. The above command line involves cross-node calculation if either $NP or $OMP or the sum of them is larger than 4.

1. First check which nodes are readily available in the cluster by issuing

```
rocks run host hostname
```

in the frontend terminal. This will allow you know which are the nodes that are up and down. Prepare a file ~/mpd.host that contains the full list of all compute node that is up.

```
compute-0-1
compute-0-2
...
compute-0-20
```

2. Issue the command

```
mpdtrace
```

to see whether you got all compute-nodes as listed in ~/mpd.host is ready to participate in a cross-node MPI calculation. Say for example, after step 1, you realise that there is a total of 18 nodes (excluding the frontend) is up. If you have not seen all of these 18 nodes in step 1, issue the command, for example,

```
mpdboot -n 19 -f ~/mpd.hosts
```

The number count 19 includes the frontend (18 nodes + 1 frontend). If any of the compute-nodes listed in ~/mpd.host is not up, mpdboot -n 19 -f ~/mpd.hosts will return an error. You have to fix the compute-node list in mpd.hosts (e.g., by commenting out the nodes that are down). If the number count (19 here) is larger than the number of nodes available (including frontend), you must reduce the number count until no error is returne by mpdboot -n XX -f ~/mpd.hosts, where XX is the number equal or less than the total number of available nodes. Then check again by issing

```
mpdtrace
```

You should find that the total number of nodes returned is equal to XX.

3. Now the computer cluster should be able to execute

```
export OMP_NUM_THREADS=$OMP; mpiexec -np $np lmp_gnu -c off -sf omp < in.melt_test > $np.$OMP.out &
```

The optimal combination of $OMP and $np for jaws, comsics and anicca, as was benchmarked in these machines, is

{$np, $OMP} = {4,4}.

If you use any combination other that the optimal ones, the total completion time of your LAMMPS run will be longer than that run with optimal values of $OMP and $np. In particular one may tempted to use larger values for $np and $OMP$, such as {$np, $OMP} = {8,8}, {1,64}, {64,1},{32,2} etc, but this is likely to be counter productive. Not only you will not gain any speed-up, the computing resources otherwise available for others will be wasted without anyone gaining any benefit.

As a conclusion, the best practice while sending a LAMMPS script to run in our computer cluster is by issuing the following command line:

```
export OMP_NUM_THREADS=4; nohup mpiexec -np 4 lmp_gnu -c off -sf omp < in.filename > filename.out &
```

The output of mpiexec can be checked by

```
cat  filename.out
```

Appendix: Results of benchmarking LAMMPS in various machines
==

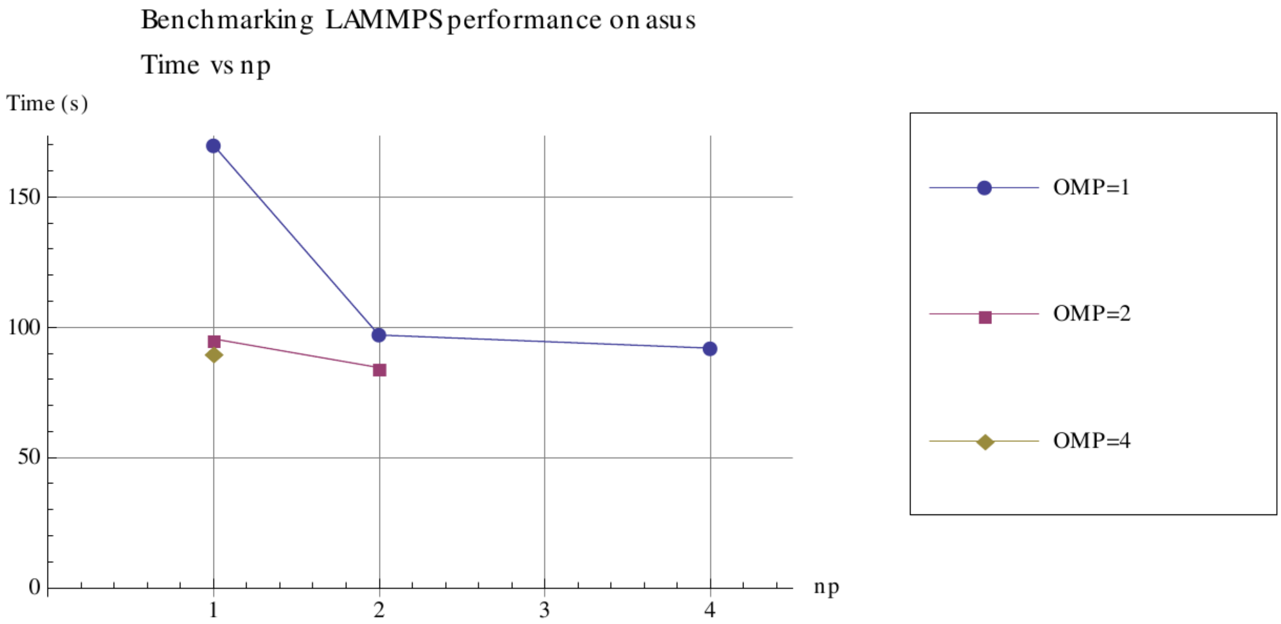
The benchmarked was carried out only to LAMMPS compiled with gnu packages (mpich2, g++, libblas, liblapack, libfftw3). No CUDA / GPU was involved. All machines run a test LAMMPS script using different values of $np and $OMP. The benchmark is divided into two groups: (1) Single node machines, and (2) cluster.

Single node machines

	Jaws, AMD Interlagos, (4 cpu x 16 cores) = 64 cores	Comsics node, Intel I5, (1 cpu x 4 cores)	Anicca, Intel Core 2, Duo, (1 cpu x 4 cores)	Asus, Intel Core I5, (1 cpu x 4 cores)
Performance using a single core (s)	255	205	255	175
Best performance in seconds (the shorter the better is the performance)	20	65	70	80 (90)
{$np,$OMP} used in best performance	{4,4}	{1,4}={4,1}	{1,4}={4,1}	{2,2} ({1,2}={2,1})

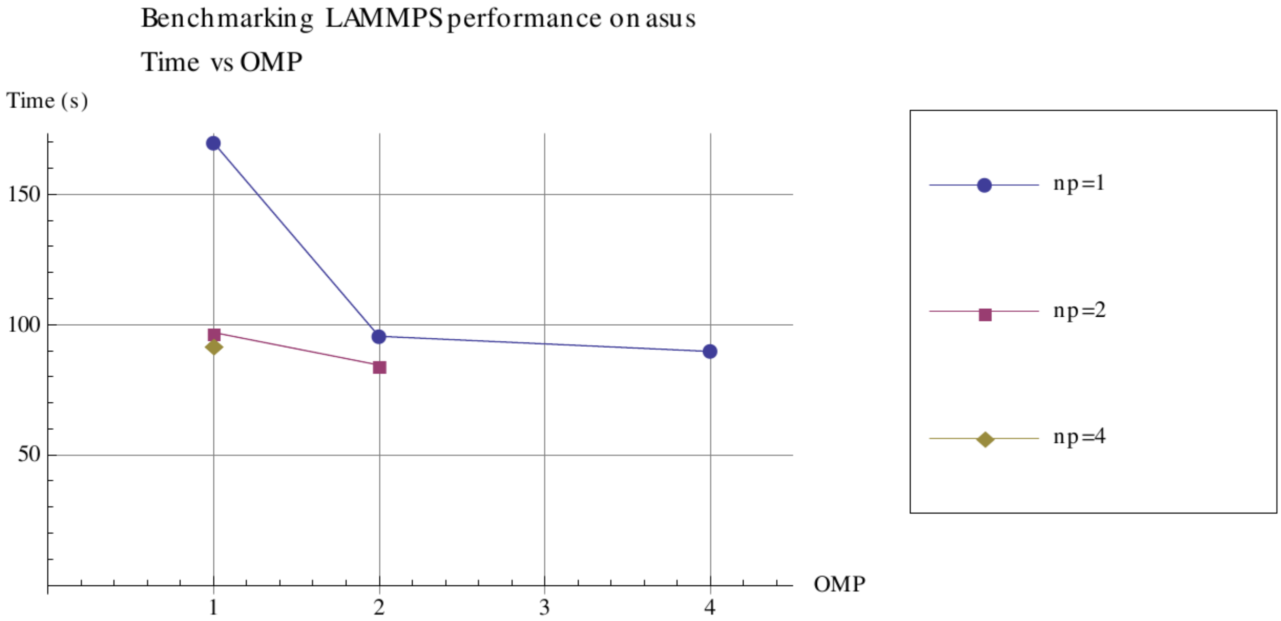
Cluster

	Comsics, Intel I5, (1 cpu x 4 cores x 20 nodes)	Anicca, Intel Core 2 Duo, (1 cpu x 4 cores x 20 nodes)
Best performance in seconds (the shorter the better is the performance)	30	50
{$np,$OMP} used in best performance	{4,4}	{4,4}



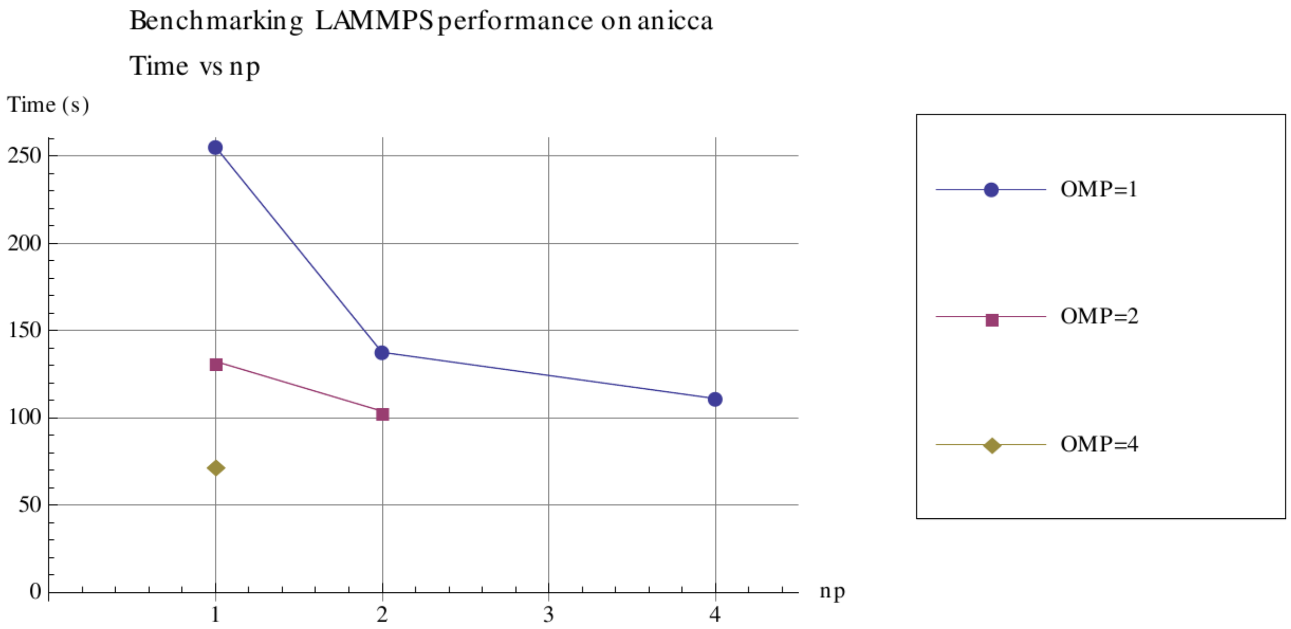
Benchmarking LAMMPS performance on asus

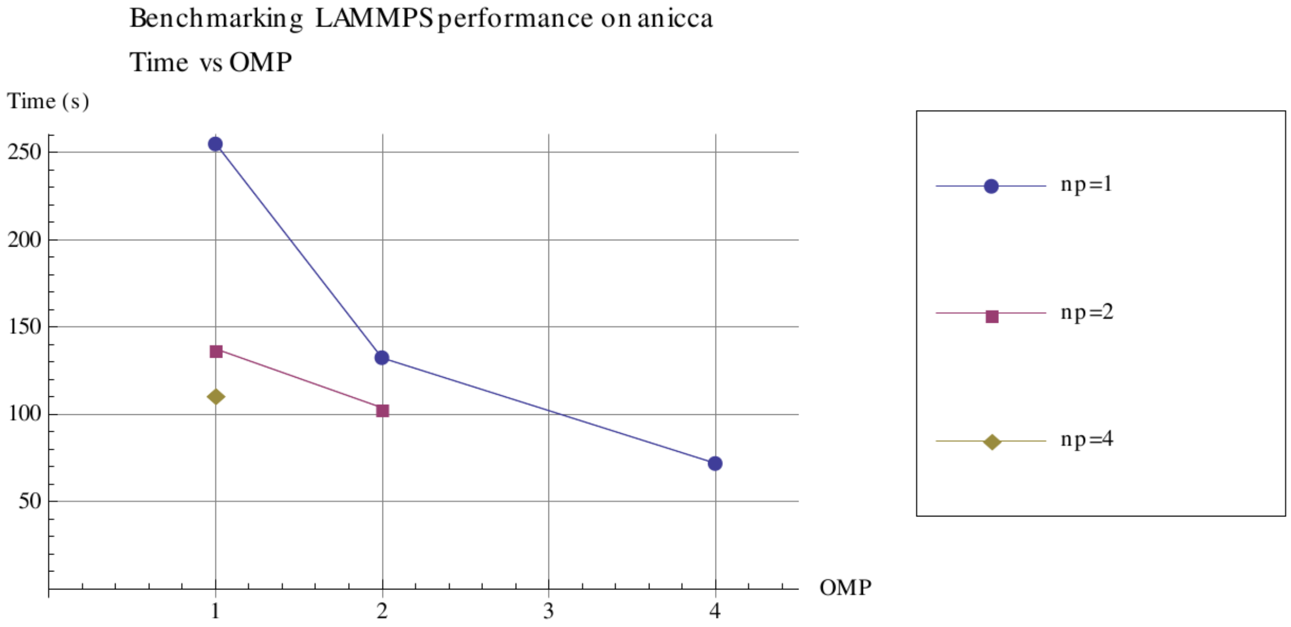
Time vs np



Benchmarking LAMMPS performance on asus

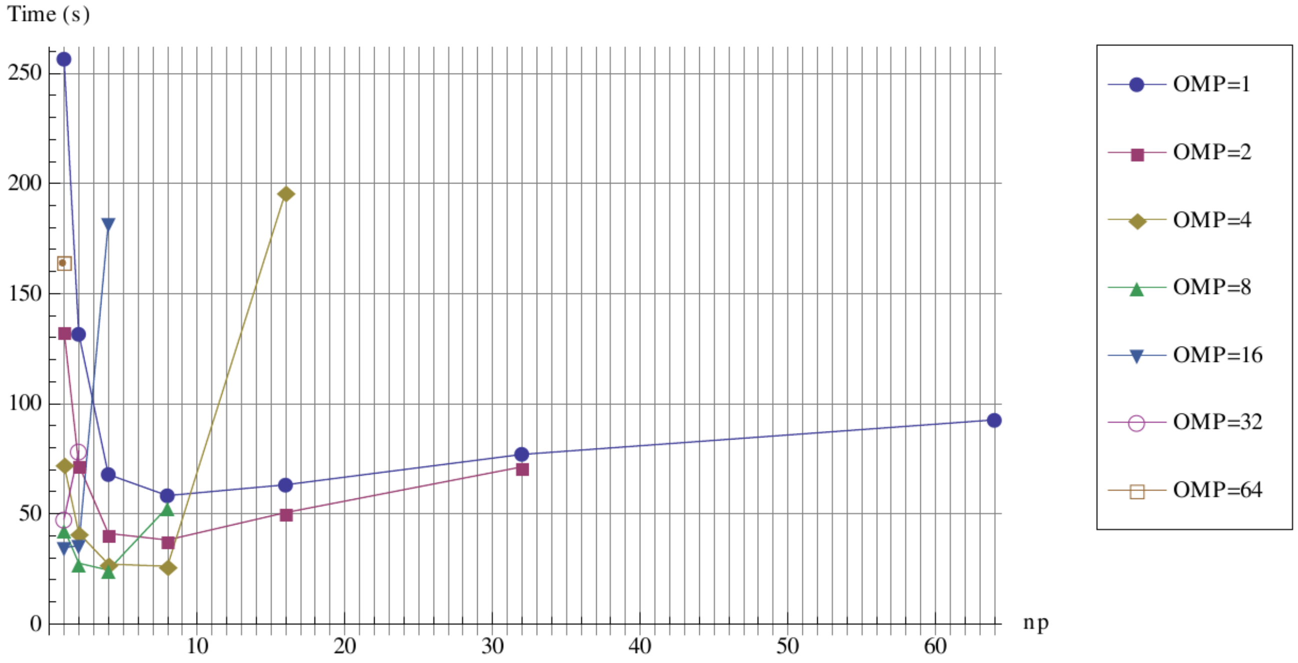
Time vs OMP


Benchmarking LAMMPS performance on anicca
Time vs np
Time (s)
250
200
150
100
50
0
1
2
3
4
np
OMP=1
OMP=2
OMP=4


Benchmarking LAMMPS performance on anicca
Time vs OMP
Time (s)
250
200
150
100
50
1
2
3
4
OMP
np=1
np=2
np=4

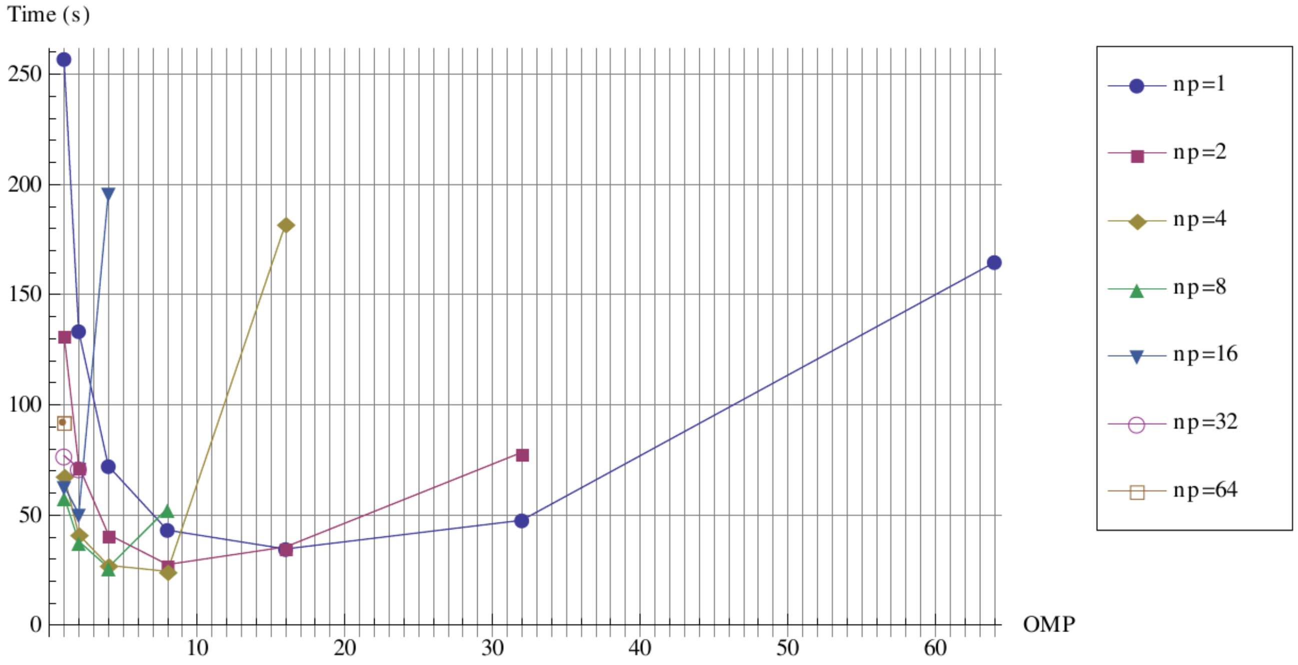
Benchmarking LAMMPS performance on jaws

Time vs np



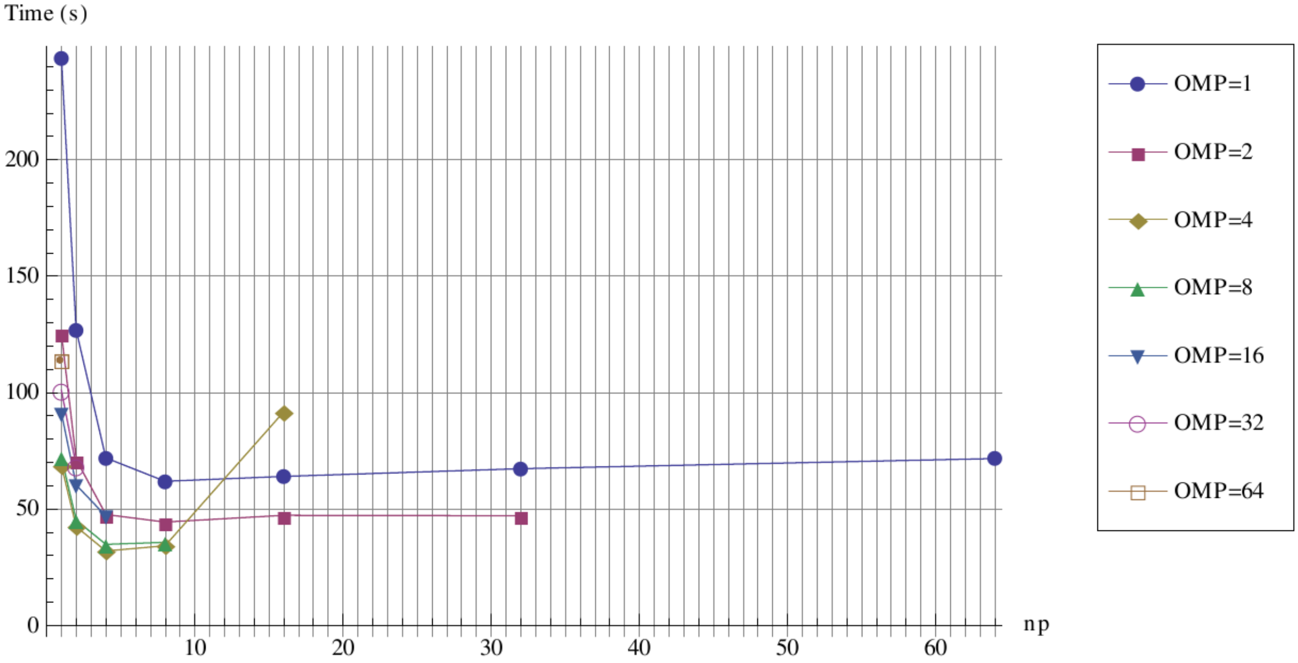
Benchmarking LAMMPS performance on jaws

Time vs OMP



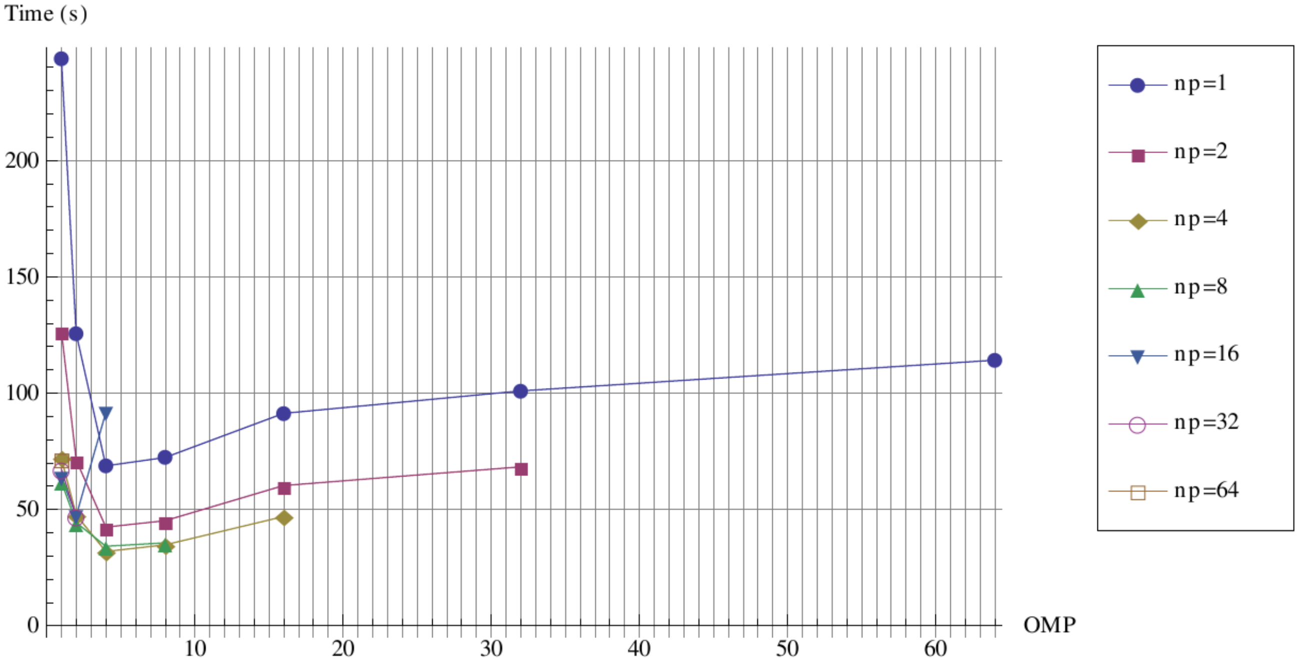
Benchmarking LAMMPS performance on comsics

Time vs np



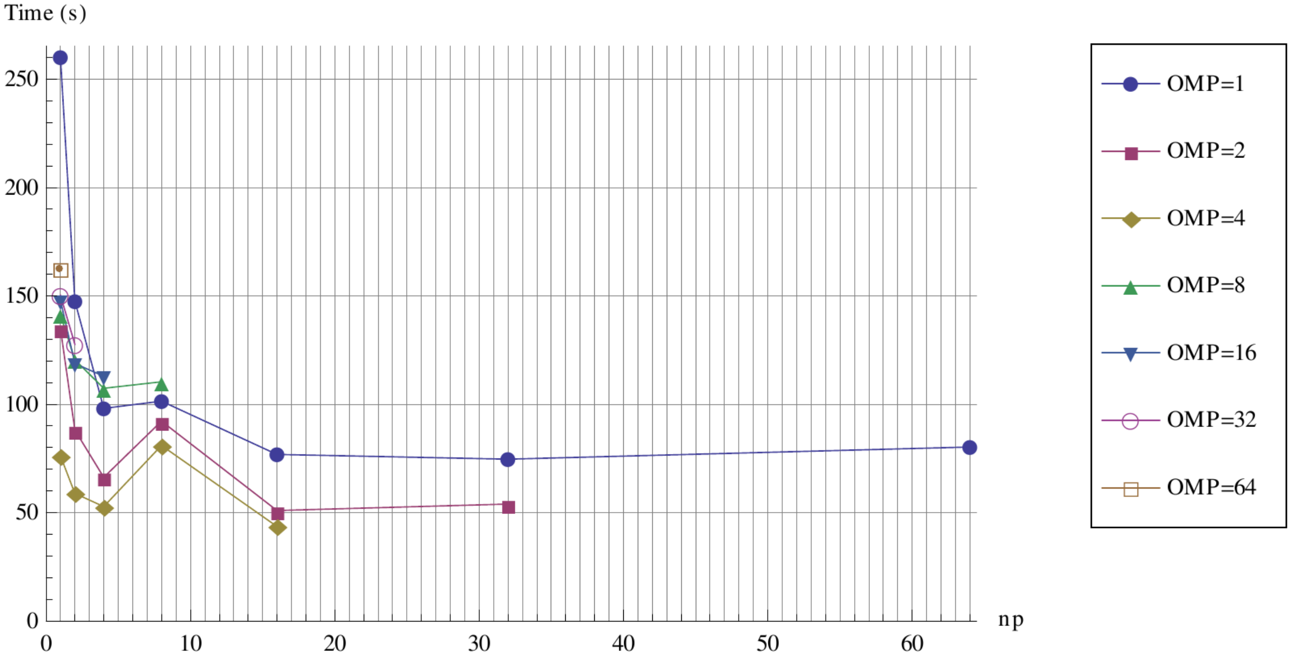
Benchmarking LAMMPS performance on comsics

Time vs OMP



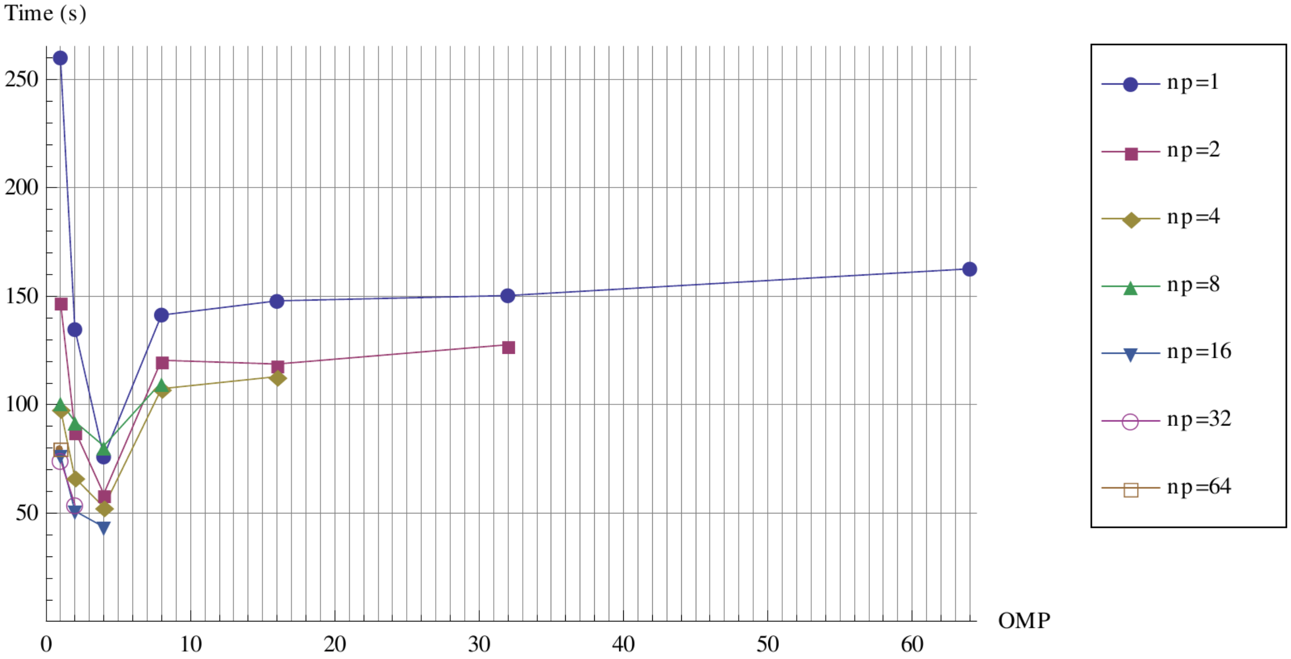
Benchmarking LAMMPS performance on Anicca

Time vs np



Benchmarking LAMMPS performance on Anicca

Time vs OMP


Time (s)
OMP
np=1
np=2
np=4
np=8
np=16
np=32
np=64