Hands-on : Case Studies in Effective Parallelization of MD Simulations

Instructor: L.V. Kale

Section 0: Compile Charm++ and NAMD (Optional)

Charm++ and NAMD can be compiled on almost all popular parallel supercomputers and clusters. Here you can learn how to compile Charm++ and NAMD from source code. You will also need to install and compile Charm++ for the other sections.

```
   1) To compile Charm++

      (1) Copy charm.tgz from
        sumschool03/tutorials/08-effective-parallelization/charm.tgz
          to your home directory ($HOME).
          cp /cdrom/sumschool03/tutorials/08-effective-
          parallelization/charm.tgz $HOME

      (2) unpack the source code:
          cd $HOME;
          gzip -dc charm.tgz | tar xvf -
          You will get a directory called "charm"
      (3) compile it:
          cd charm
          ./build charm++ net-linux   (if you on a Linux Box),
          or
          ./build charm++ net-sol cc  (if you have a Sun Desktop)

   2) To compile NAMD
      (1) copy namd.tgz to your home directory:
          >cp  sumschool03/tutorials/08-effective-
           parallelization/namd2.tgz $HOME
      (2) unpack it:
          > gzip -dc namd2.tgz | tar xvf -
          and you will get a directory "namd2"
      (3) modify the namd2/Make.charm, changing CHARMBASE
          to point to the charm directory you are going to
          use:
          CHARMBASE = $HOME/charm
          Here $HOME/charm is the directory where you
          installed charm (from step 1).

      (4) run "config"
          > ./config Linux-i686-g++    (On Linux)
          or,
               > ./config Solaris-Sparc-CC  (On Sun Sparc)
      (5) compile under the directory you just created
          > cd Linux-i686-g++
          (or  cd Solaris-Sparc-CC, on a Sun box)
          > make
          This will create namd2 executable after compilation finishes.
          To run NAMD locally
          ./charmrun ++local +p1 ./namd2 <config files>
```

Section 1: Study Parallel Performance of NAMD

The purpose of this session is to show you how to run NAMD and obtain performance results.

Login to platinum, and then run the following commands to copy the files you will need for this Hands-On Session:

```
  tbss
  mkdir kaletut
  cd kaletut
  mkdir mscl
  cp ~sameer/SS03/mscl/*     mscl/
  mkdir apoa1
  cp ~sameer/SS03/apoa1/*    apoa1/
```

To run the MSCL system on two CPUs, type the following commands:

```
  cd mscl
  ~sameer/SS03/runbatch2 test.conf test2.out
```

Once the job has completed, you can find out the performance by looking at the last Benchmark line in the output file. A command to show you the last such line is:

```
  grep Benchmark test2.out | tail -1
```

(Note: the | symbol is on the keyboard above the \ symbol.)
For example, this may be your result:

```
Info: Benchmark time: 2 CPUs 0.30369 s/step 3.51493 days/ns 55320 kB
memory
```

This means that the performance was 0.30369 seconds of computer time using 2 CPUs to calculate one step of the simulation.

Start filling out the following chart with the performance numbers:

TIME/STEP

Processors	2	4	8	16(Optional)
mscl				
apoa1				

How much faster did it run when using more CPU's? Fill up the following table by dividing the first column of the above table by the other columns.

SPEEDUP OVER 2 CPU'S

Processors	2	4	8	16(Optional)
mscl	1			
apoa1	1			

As you can see, running on twice as many processors does not give you fully twice as much performance.

Similarly, to run on 4 or 8 CPUs, type:

```
  ~sameer/SS03/runbatch4 test.conf test4.out
  ~sameer/SS03/runbatch8 test.conf test8.out
```

Again, you can determine the performance by running the above "grep" command on the appropriate output file.

Running on 16 CPUs is optional:

```
  ~sameer/SS03/runbatch16 test.conf test16.out
```

Similarly, to run on the larger apoa1 system:

```
  cd kaletut/apoa1
  ~sameer/SS03/runbatch2  apoa1.namd test2.out
  ~sameer/SS03/runbatch4  apoa1.namd test4.out
  ~sameer/SS03/runbatch8  apoa1.namd test8.out
```

Again, running on 16 CPUs is optional:

```
  ~sameer/SS03/runbatch16 apoa1.namd test16.out
```

Section 2: Analyze NAMD Parallel Performance using Projections

Projections is a tool for High Performance Visualization. The motivation of this experiment is to use the performance analysis tool Projections to understand the performance of parallel programs. All Charm++ programs like NAMD can be profiled using projections.

To build NAMD with projections do

make projections

instead of just

make, in step 2(5) of Section 0.

Now Running NAMD will produce log files that look like namd2.*.log. We have also put some sample log files in

```
     sumschool03/tutorials/08-effective-parallelization/logs.tgz
```

Copy this archive to your home directory and untar the archive.

```
cp sumschool03/tutorials/08-effective-parallelization/logs.tgz $HOME
cd $HOME/logs
```

The following projections views should show you the power of projections. The projections logs are for NAMD on the Apo-lipo protein A1.

```
   ../charm/tools/projections/projections &
```

A window will popup, do the following

1) "Click File" and then click "Open File(s)"
2) Click on namd2.pro.sts and then click "Open"

3) Now Click the "Tools" menu and then click on "Graphs"
4) On the window that pops up fill the following parameters
   a) Processors: 0-15
   b) Interval Size: 100ms
   c) Start time: 19.39s
   d) End time: 37.5s
   d) Once the new window pops with the graph click on "Line Graph"

   This plot shows the performance of NAMD before the loadbalancing operation. As you can see the performance is quite bad. To view the performance after the loadbalancing,

   DON'T Forget to put the 's' for seconds and 'ms' for milliseconds for the various time values.

   a) Close the Graphs window.
   b) Now click the "Tools" menu
   c) On the window that pops up fill the following parameters

   d) Processors: 0-15
   e) Interval Size: 100ms
   f) Start time: 38.51s
   g) End time: 45.93s
   h) Once the new window pops with the graph click on "Line Graph"

   You will notice that the performance is much better.

5) Close the Graphs Window
6) Click on Tools->"Usage Profile" and fill in the following parameters
   a) Processors: 0-15
   b) Phase: 0
   c) Begin Time: 38.51s
   d) End Time: 45.93s
   e) Then Click "OK"
   f) Ignore the "Length" option
   g) Enlarge the window and wait for 1 min.

   The "Usage Profile" view shows the performance of each entry method in NAMD.

6) Please observe the utilization of the *Average Bar* in the *histogram. What is the average utilization of the system?* The utilization is the height of the Avg. bar till the *white* region!
   Also which processor has the Maximum Utilization?

Section 3: Write a Parallel Program in Charm++ (Optional)

```
NAMD is written in Charm++, which is a Parallel programming language
based on C++. In this section we will learn to write a simple parallel
program to calculate the integral of the function
        f(x)=1.0/x

over the range [1.0, 2.0].
We have written the parallel sections of program. You need to fill in
the missing code for the acutal computation.

Refer Section 0 to get the source code for Charm++ and compiling it on
your local machine:

1. set the environment up for your convenience:
   for csh:
   > setenv CHARM_HOME  $HOME/charm
   if you are using ksh or bash:
   > export CHARM_HOME=$HOME/charm
   setting CHARM_HOME to the charm you just built in Section 0.

2. copy the incomplete source code to your local home directory:
   > cp -r $CHARM_HOME/pgms/charm++/examples/integrateArray \
integrateArray

3. edit the Makefile, make sure the CHARMC points to CHARM_HOME:
   (1) If you are using Linux, change CHARMC to:
       CHARMC=$(CHARM_HOME)/net-linux/bin/charmc
   (2) If you are using Sun, change CHARMC to:
       CHARMC=$(CHARM_HOME)/net-sol-cc/bin/charmc

4. compile the program by simply typing "make"
   If nothing goes wrong, you should get an executable called "pgm" and
   a utility program "charmrun".

5. you should now read the code, and try to fill in the missing part in
   file pgm.C

6. re-compile the code, and run it, using command for example:
   > ./charmrun +p4 ./pgm 10000 100 ++local
   the first parameter 10000 is the number of slices to divide the
[1.0, 2.0],
   and the second parameter is the number of Chares to work. That is,
it
   creates 100 parallel objects, each works on 10000/100=100 slices.
   "+p4" will fire 4 copies of this program in parallel;
   "++local" will run this job locally, not using a parallel machine.

7. if you are interested in parallel performance analysis, you can also
   make runs with Projections.
   (1) type "make pgm_prof", this will compile another executable
called
   "pgm_prof" which has the performance trace module linked in.
   (2) Run this executable like this:
       ./charmrun +p4 ./pgm 10000 100 ++local
```

```
        will produce 4 log files: pgm.0.log - pgm.3.log.
    (3) Run Projections (installed at $CHARM_HOME/tools/projections) to
view
    these logs, command line:
        $CHARM_HOME/tools/projections/bin/projections
```