This is a step-by-step tutorial to validate SKT-HPL.

1 Requirements

1.1 Hardware Requirements

We need a cluster with at least eight nodes. Intel Xeon processor is also required. Memory space for each process should be larger than 2GB.

1.2 Software Requirements

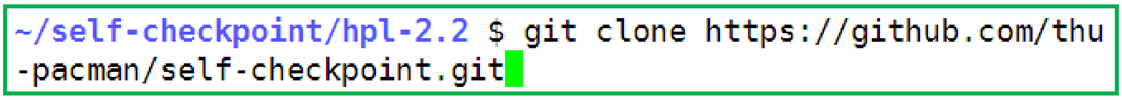
Linux system, BLAS library, MPI compiler and MPI runtime. SysV* shared memory must be supported. The reviewers should be able to make a node failure or simulate a node failure, e.g. power-off, remove it from resource pool, or clean its memory data.

It would be helpful to have SLURM installed, or you may need to modify same running scripts.

We use High-Performance Linpack(HPL) to validate our idea. The version we used for our work is HPL-2.2. We provide both original version and our modified version. Also, one can get HPL from www.netlib.org/benchmark/hpl.

2 Installation

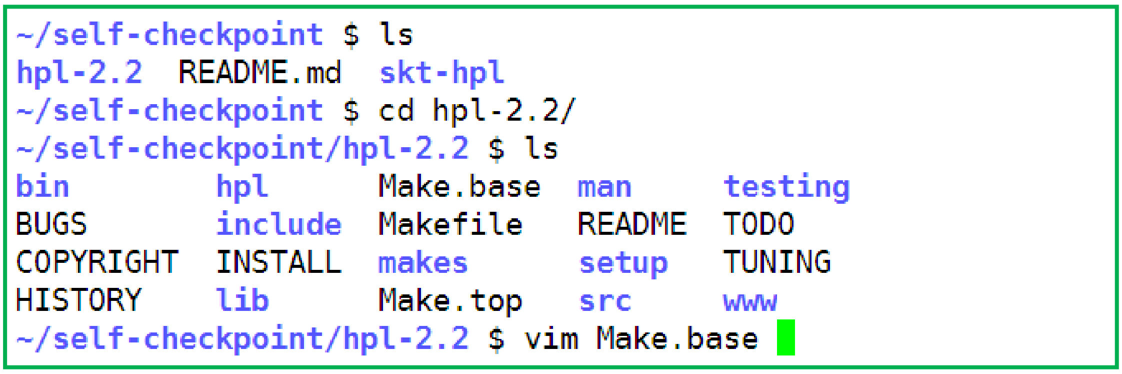
First of all, download self-checkpoint.



```
~/self-checkpoint/hpl-2.2 $ git clone https://github.com/thu
-pacman/self-checkpoint.git
```

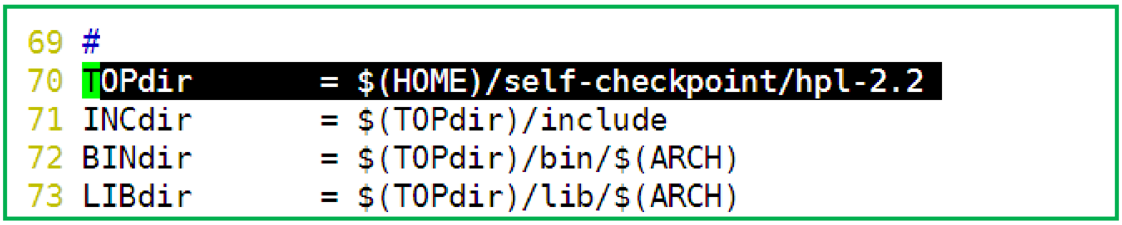
2.1 Original HPL

Enter the directory of original HPL, i.e. **hpl-2.2**, and open **Make.base.**

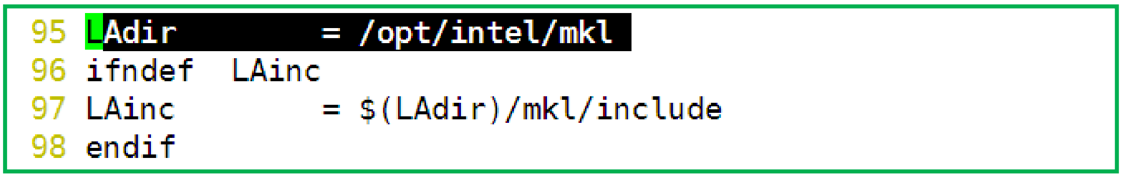


```
~/self-checkpoint $ ls
hpl-2.2  README.md  skt-hpl
~/self-checkpoint $ cd hpl-2.2/
~/self-checkpoint/hpl-2.2 $ ls
bin        hpl      Make.base  man     testing
BUGS       include  Makefile   README  TODO
COPYRIGHT  INSTALL  makes      setup   TUNING
HISTORY    lib      Make.top   src     www
~/self-checkpoint/hpl-2.2 $ vim Make.base
```

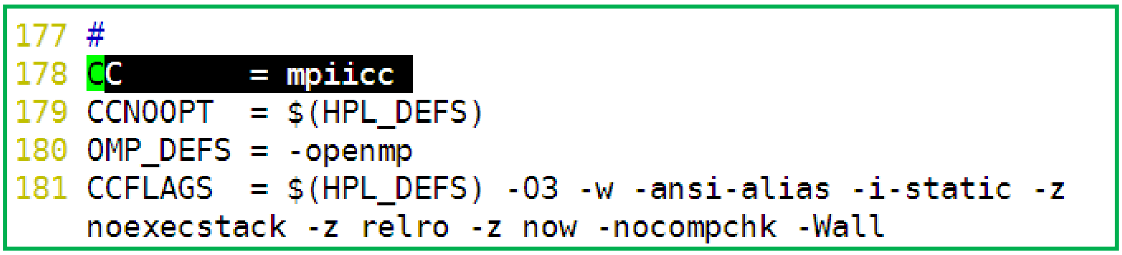
Modify the path in Line-70, Line-95 and Line-178 according to your system configuration.



```
69 #
70 TOPdir        = $(HOME)/self-checkpoint/hpl-2.2
71 INCdir        = $(TOPdir)/include
72 BINdir        = $(TOPdir)/bin/$(ARCH)
73 LIBdir        = $(TOPdir)/lib/$(ARCH)
```

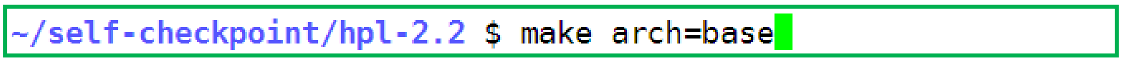


```
95 LAdir         = /opt/intel/mkl
96 ifndef  LAinc
97 LAinc         = $(LAdir)/mkl/include
98 endif
```



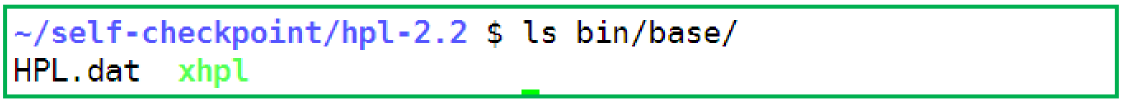
```
177 #
178 CC         = mpiicc
179 CCNOOPT    = $(HPL_DEFS)
180 OMP_DEFS = -openmp
181 CCFLAGS    = $(HPL_DEFS) -O3 -w -ansi-alias -i-static -z
    noexecstack -z relro -z now -nocompchk -Wall
```

Build the original HPL, it may take several minutes.



```
~/self-checkpoint/hpl-2.2 $ make arch=base
```

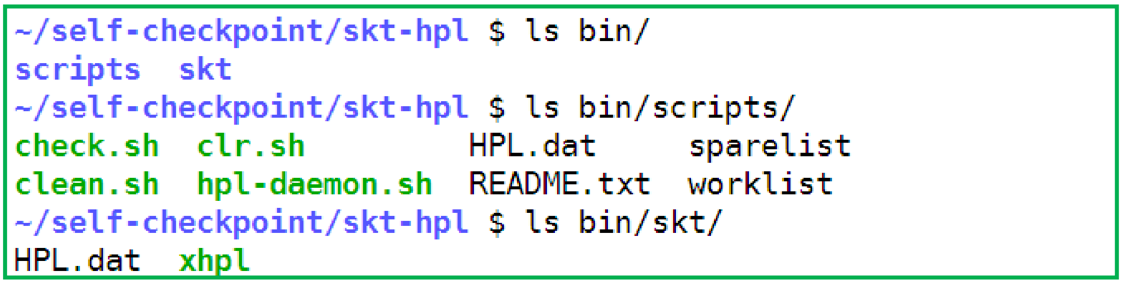
After compilation, there should be an executable file **xhpl** and configuration file **HPL.dat** in **bin/base**



```
~/self-checkpoint/hpl-2.2 $ ls bin/base/
HPL.dat  xhpl
```

2.2 SKT-HPL

Similarly, we enter the directory **skt-hpl**, and modify **Make.skt** according to the configuration of our system, then type **make arch=skt**. Besides the files generated by make, in skt-hpl/bin, there is a directory named **scripts**. The running scripts are in this directory.



```
~/self-checkpoint/skt-hpl $ ls bin/
scripts  skt
~/self-checkpoint/skt-hpl $ ls bin/scripts/
check.sh  clr.sh         HPL.dat     sparelist
clean.sh  hpl-daemon.sh  README.txt  worklist
~/self-checkpoint/skt-hpl $ ls bin/skt/
HPL.dat  xhpl
```

3 A Simple Run

3.1 Problem Sizes

Empirically, if we have M GB memory in total, then the max problem size of HPL will be `N = sqrt(M)*10000` (A matrix with N dimensions and each element is a double, so it occupies 80% memory space, the rest memory is for other data structures and operation system).

For example, in our tutorial we will use a cluster with 512 GB memory in total, so the max problem size for original HPL is `N_max=226274`.

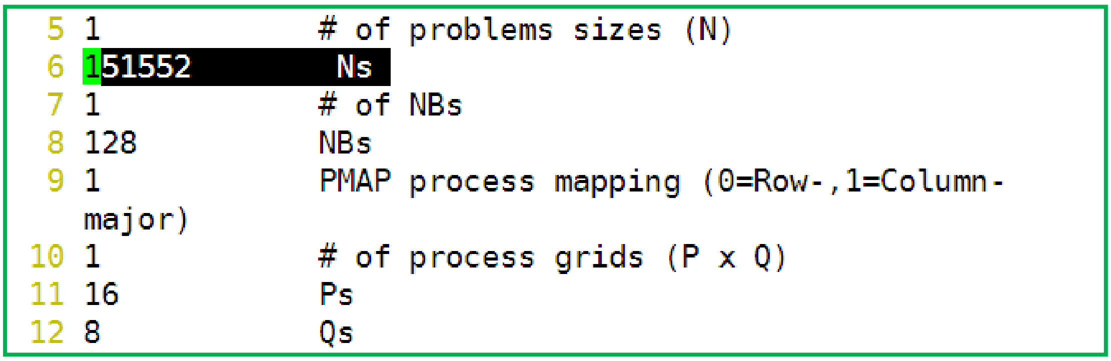
SKT-HPL take half memory space for fault tolerance, and approximately half is left for applications. And there are some constraints for SKT-HPL problem size. One can get a reasonable number by:

1. `N_skt = N_max * 0.67` (about 45% memory is available for application).
2. Adjust `N_skt` to satisfy `N_skt%(NB*GCD(P,Q))==0`. Here GCD means Greatest common divisor.

The below figure shows a part of HPL configuration file. Line-6 is the problem size, Line-8 is a parameter `NB`, Line-11 and Line-12 specify that the program will be run by `P * Q` processes. SKT-HPL requires that

`N_skt % (NB * P) == N_skt % (NB * Q) == 0`

In our tutorial, the best problem size for SKT-HPL is `N_skt=151522.`

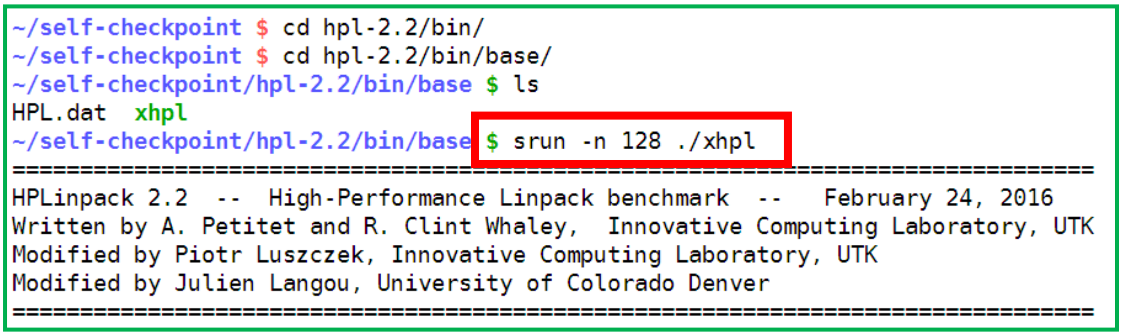


```
1            # of problems sizes (N)
151552        Ns
1            # of NBs
128          NBs
1            PMAP process mapping (0=Row-,1=Column-
major)
1            # of process grids (P x Q)
16           Ps
8            Qs
```

A complete run of max problem size takes a while (~30 minutes).

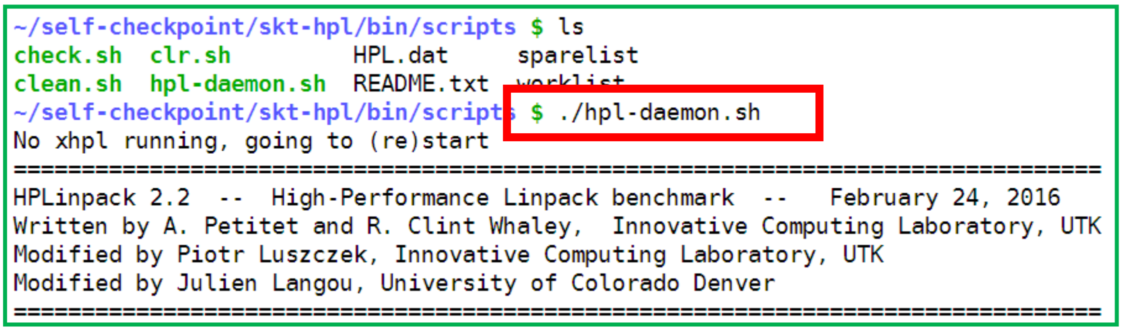
For original HPL, submit **bin/base/xhpl** to job management system. For SKT-HPL, submit **bin/skt/scripts/hpl-daemon.sh**. The daemon is responsible to restart SKT-HPL after failures, and to clean the environment after completion. Examples are show as below figures

To submit original HPL, type [srun –n “number of processes” ./xhpl].



```
~/self-checkpoint $ cd hpl-2.2/bin/
~/self-checkpoint $ cd hpl-2.2/bin/base/
~/self-checkpoint/hpl-2.2/bin/base $ ls
HPL.dat  xhpl
~/self-checkpoint/hpl-2.2/bin/base $ srun -n 128 ./xhpl
================================================================================
HPLinpack 2.2  --  High-Performance Linpack benchmark  --   February 24, 2016
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================
```

To submit SKT-HPL, use the script [./hpl-daemon.sh]. The srun command is inside that script.

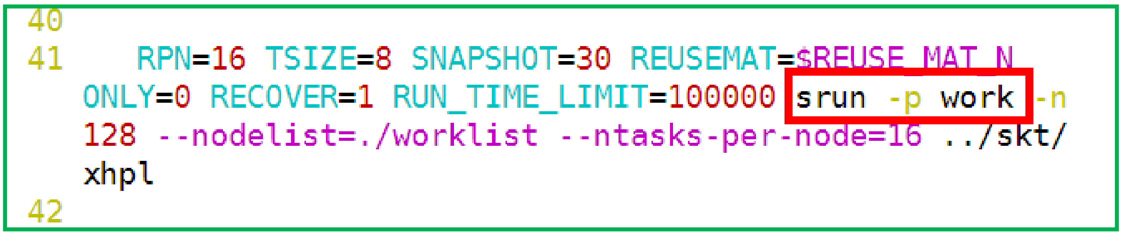


```
~/self-checkpoint/skt-hpl/bin/scripts $ ls
check.sh  clr.sh          HPL.dat     sparelist
clean.sh  hpl-daemon.sh   README.txt  worklist
~/self-checkpoint/skt-hpl/bin/scripts $ ./hpl-daemon.sh
No xhpl running, going to (re)start
================================================================================
HPLinpack 2.2  --  High-Performance Linpack benchmark  --   February 24, 2016
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================
```

If you don’t use SLURM, modify the submission command according to your system. Usually, you may use mpirun/mpiexec/bsub/qsub.

3.2 Setup Environment for SKT-HPL

Before running SKT-HPL, we need to setup several environment variables in Line-41 of **bin/scripts/hpl-daemon.sh**.

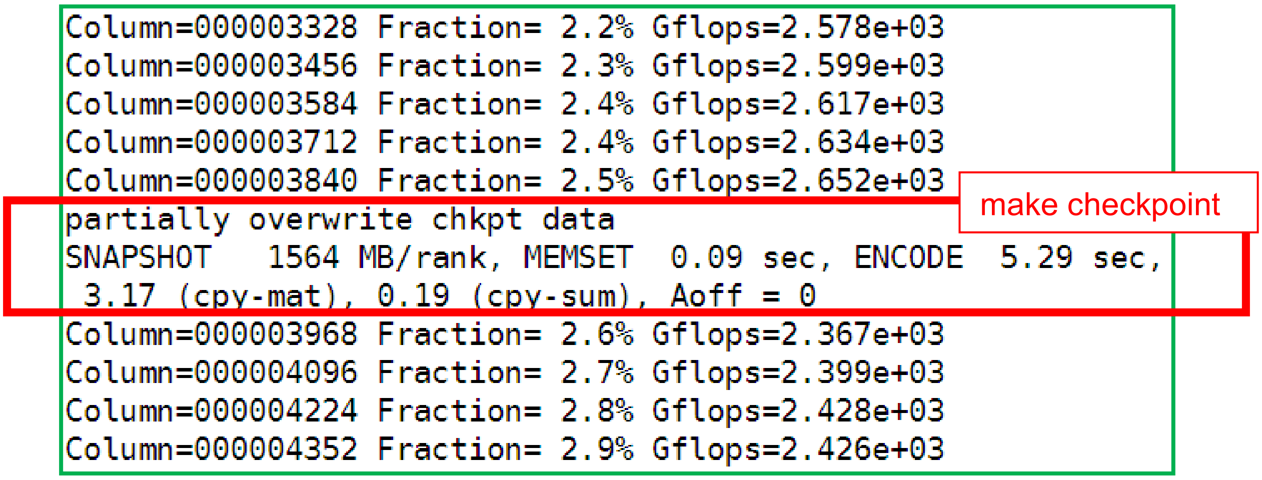


```
40
41    RPN=16 TSIZE=8 SNAPSHOT=30 REUSEMAT=$REUSE_MAT_N
   ONLY=0 RECOVER=1 RUN_TIME_LIMIT=100000 srun -p work -n
   128 --nodelist=./worklist --ntasks-per-node=16 ../skt/
   xhpl
42
```

Below is a list to describe these variables. In this tutorial, there is no need to change these values.

- **RPN**: How many MPI processes on a node.
- **TSIZE**: Group size described in the paper, 8 and 16 are good choices.
- **SNAPSHOT**: Checkpoint interval. It means *how many iterations*, NOT *how many seconds*.
- **REUSEMAT**: If set to 0, generate a new problem, otherwise, read from checkpoint. Should be 0 for a fresh run and set to 1 for recovery.
- **ONLY**: For debugging purpose. If set to 1, only one checkpoint will be made. If set to 0, will do checkpoint periodically.
- **RECOVER**: For debugging purpose. If set to 0, checkpoint will not be read after restart.
- **RUN_TIME_LIMIT**: For debugging purpose. A complete SKT-HPL could last for hours, but it will exit after RUN_TIME_LIMIT seconds.

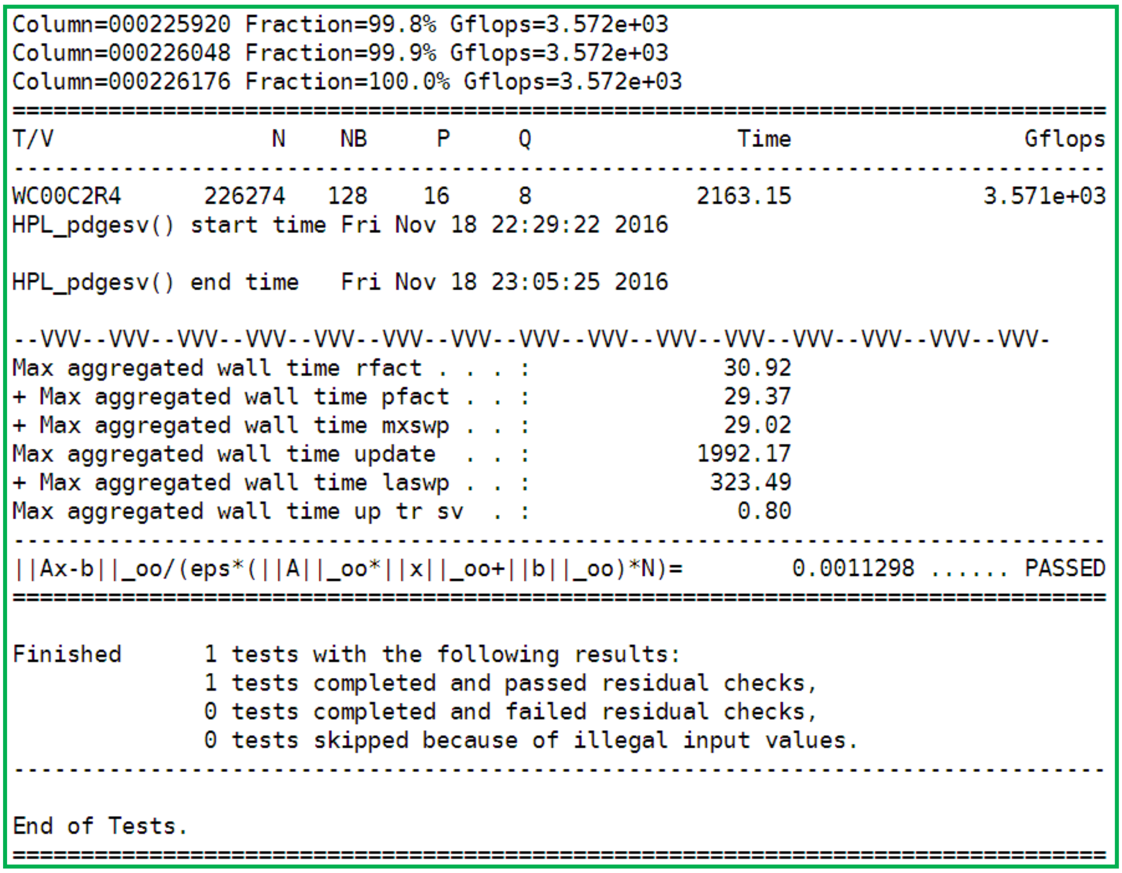
3.3 SKT-HPL Output



```
Column=000003328 Fraction= 2.2% Gflops=2.578e+03
Column=000003456 Fraction= 2.3% Gflops=2.599e+03
Column=000003584 Fraction= 2.4% Gflops=2.617e+03
Column=000003712 Fraction= 2.4% Gflops=2.634e+03
Column=000003840 Fraction= 2.5% Gflops=2.652e+03
partially overwrite chkpt data
SNAPSHOT   1564 MB/rank, MEMSET  0.09 sec, ENCODE  5.29 sec,
 3.17 (cpy-mat), 0.19 (cpy-sum), Aoff = 0
Column=000003968 Fraction= 2.6% Gflops=2.367e+03
Column=000004096 Fraction= 2.7% Gflops=2.399e+03
Column=000004224 Fraction= 2.8% Gflops=2.428e+03
Column=000004352 Fraction= 2.9% Gflops=2.426e+03
```

make checkpoint

When SKT-HPL is computing, it prints current progress like “Column=xxxx Fraction=xxxx”. When it is making a checkpoint, it prints the time cost by checkpoint, including network operation and local operation. The overhead of a single checkpoint can be calculated by summing up the numbers in a line starts with “SNAPSHOT”



```
Column=000225920 Fraction=99.8% Gflops=3.572e+03
Column=000226048 Fraction=99.9% Gflops=3.572e+03
Column=000226176 Fraction=100.0% Gflops=3.572e+03
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00C2R4      226274   128    16     8            2163.15              3.571e+03
HPL_pdgesv() start time Fri Nov 18 22:29:22 2016

HPL_pdgesv() end time   Fri Nov 18 23:05:25 2016

--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV-
Max aggregated wall time rfact . . . :              30.92
+ Max aggregated wall time pfact . . :              29.37
+ Max aggregated wall time mxswp . . :              29.02
Max aggregated wall time update  . . :            1992.17
+ Max aggregated wall time laswp . . :             323.49
Max aggregated wall time up tr sv  . :               0.80
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0011298 ...... PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

Both SKT-HPL and original HPL print out the performance (3.571e+03 Gflops in above figure) after completion. The last several lines show if the result is correct or wrong.

4 Fault Injection

It is not practical to wait for a node failure in a small cluster, so we inject failures.

4.1 Inject a Single Node Failure

The simplest way to inject a single node failure is to turn it off, or power it off. However, one may cannot do that or do not want to reboot nodes. It is also possible to simulate a node failure. To do so, following the steps:

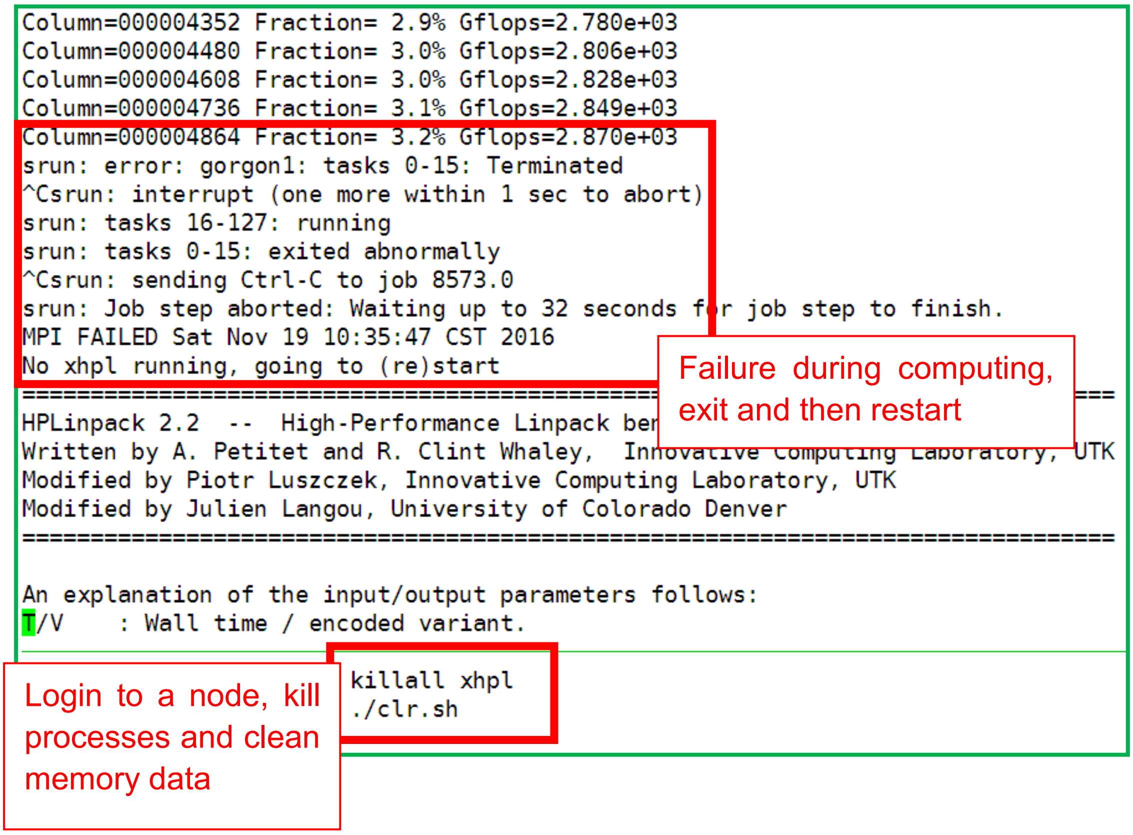
1. Login a node that SKT-HPL is currently running on

2. Type **killall xhpl**, to kill all SKT-HPL processes on this node
3. Run script **skt-hpl/bin/scripts/clr.sh**

After a failure injected, the job management system may take a while to detect it then exit. Alternatively, we can manually kill the job for quick. SKT-HPL is run by a script **bin/skt/scripts/hpl-daemon.sh**, which will restart SKT-HPL.

4.2 Failure During Computing

We can inject a single node failure during computing, shown as below



```
Column=000004352 Fraction= 2.9% Gflops=2.780e+03
Column=000004480 Fraction= 3.0% Gflops=2.806e+03
Column=000004608 Fraction= 3.0% Gflops=2.828e+03
Column=000004736 Fraction= 3.1% Gflops=2.849e+03
Column=000004864 Fraction= 3.2% Gflops=2.870e+03
srun: error: gorgon1: tasks 0-15: Terminated
^Csrun: interrupt (one more within 1 sec to abort)
srun: tasks 16-127: running
srun: tasks 0-15: exited abnormally
^Csrun: sending Ctrl-C to job 8573.0
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
MPI FAILED Sat Nov 19 10:35:47 CST 2016
No xhpl running, going to (re)start
================================================                                  ===
HPLinpack 2.2  --  High-Performance Linpack ben
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
```

```
killall xhpl
./clr.sh
```

Failure during computing, exit and then restart

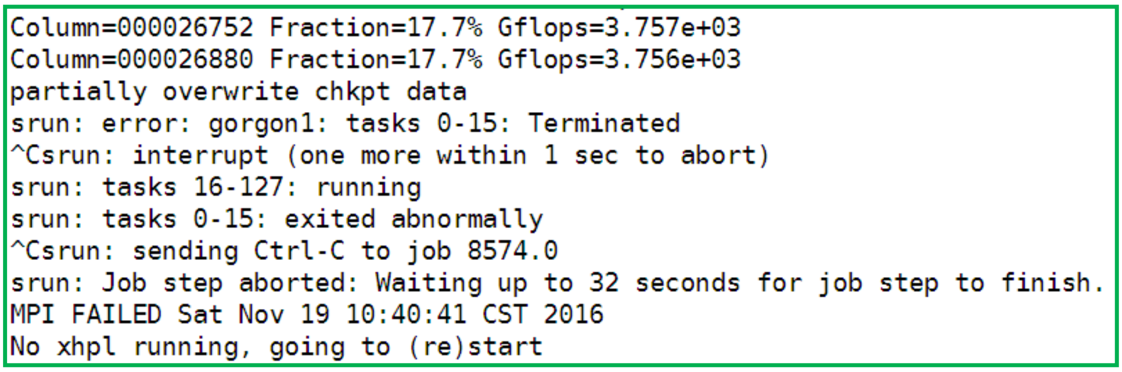
Login to a node, kill processes and clean memory data

4.3 Failure During Checkpoint Updating

Similarly, failure can also be injected during checkpoint updating

`

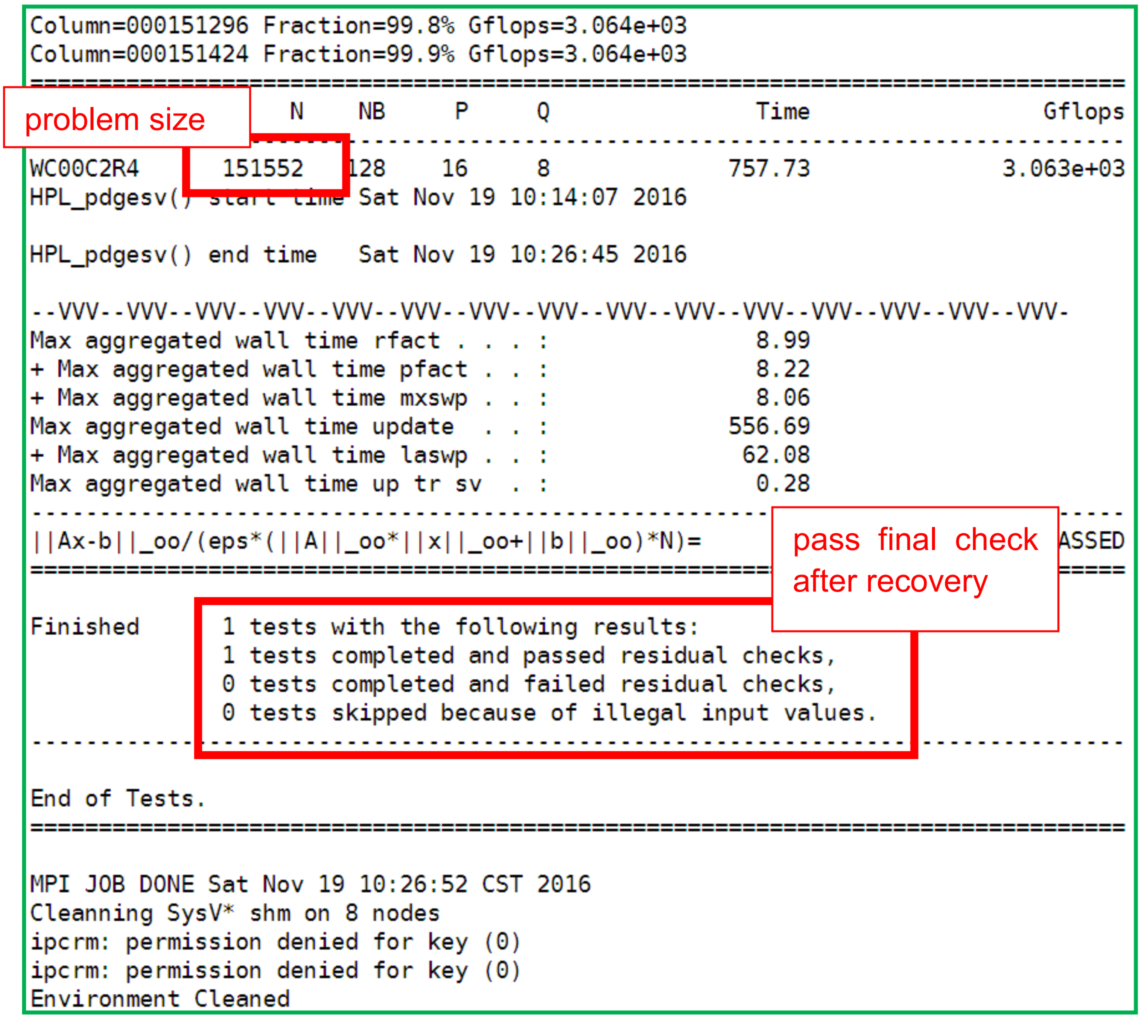
Failure during checkpoint updating



```
Column=000026752 Fraction=17.7% Gflops=3.757e+03
Column=000026880 Fraction=17.7% Gflops=3.756e+03
partially overwrite chkpt data
srun: error: gorgon1: tasks 0-15: Terminated
^Csrun: interrupt (one more within 1 sec to abort)
srun: tasks 16-127: running
srun: tasks 0-15: exited abnormally
^Csrun: sending Ctrl-C to job 8574.0
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
MPI FAILED Sat Nov 19 10:40:41 CST 2016
No xhpl running, going to (re)start
```

4.4 Check Correctness and Performance

We can inject node failures more than once. But ensure that only a single node is down for each injection. And a second injection should be done after SKT-HPL restart. Finally, when SKT-HPL completes, we can check if it passes the result check integrated in HPL.



```
Column=000151296 Fraction=99.8% Gflops=3.064e+03
Column=000151424 Fraction=99.9% Gflops=3.064e+03
================================================================================
                   N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00C2R4      151552   128    16     8             757.73              3.063e+03
HPL_pdgesv() start time Sat Nov 19 10:14:07 2016

HPL_pdgesv() end time   Sat Nov 19 10:26:45 2016

--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV-
Max aggregated wall time rfact . . . :               8.99
+ Max aggregated wall time pfact . . :               8.22
+ Max aggregated wall time mxswp . . :               8.06
Max aggregated wall time update  . . :             556.69
+ Max aggregated wall time laswp . . :              62.08
Max aggregated wall time up tr sv  . :               0.28
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=                          ASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================

MPI JOB DONE Sat Nov 19 10:26:52 CST 2016
Cleanning SysV* shm on 8 nodes
ipcrm: permission denied for key (0)
ipcrm: permission denied for key (0)
Environment Cleaned
```

Current implementation does not save the runtime in checkpoint. Therefore, to measure the performance of SKT-HPL, please do not inject any failure. Also, checkpoint number has a big impact on SKT-HPL performance, so the variable **ONLY** should be set to **1** for performance measurement.