APPENDIX

A. Abstract

This artifact contains the code to build the emulation system and the information needed to launch some experiments in the paper "ELS Debugging and Tuning Large-scale Parallel Programs with Emulation Execution on Small Clusters". We explain how to compile and run the benchmarks used in Section IV

B. Description

1) Check-list (artifact meta information):

- **Program:** C and MPI code, C library
- **Compilation:** mpicc
- **Binary:** NPB executables, HPL,SHOC
- **Data Set:** default input of the three benchmarks
- **OS environment:** Linux, MPICH3.1
- **Hardware:** Each node has two Intel Xeon E5-2692 v2 CPU with 24 cores and 64GB memory. We use up to 64 nodes to complete the overhead evaluation in Section IV.
- **Output:** execution time, the benchmark output
- **Experiment workflow:** We build emulation system libraries, pre-link the libraries when compile benchmarks. In actual-execution phase, we submit a job script to the PBS batch system, wait for the job to be scheduled, and then collect the log. In emulation-execution phase, we emulation the program on a local cluster.
- **Publicly available?:** Yes.

2) *How delivered:* NPB, HPL and HPCC are open-source benchmarks, you can get them in the following URLs separately:

http://www.netlib.org/benchmark/hpl/.

https://www.nas.nasa.gov/publications/npb.html

https://github.com/vetter/shoc

Besides, our emulation system code can be cloned from GitHub using the following URL:

https://github.com/answerfly/MPIEmulation.git

3) Hardware dependencies: We used 1000-nodes HPC system for performance evaluation and functional verification.

4) Software dependencies: HPL depends on BLAS or Intel MKL. We used the existing HPL2.2 for our experiments in Section IV, which uses Intel MKL as its math library.

5) Datasets: The performance evaluation requires running the application on the target HPC system and the local small cluster. It's necessary to adjust the corresponding parameters in the input files including problem size and running scales, as well as the content in submit script.

C. Installation

Build two static library librecord.a and libmpiemulation.a.

```
$ cd MPIEmulation
$ ./make.sh
```

D. Experiment workflow

1) Actual-execution phase

Taking the experiments of IS(a program in NPB) as an example, the first step is to add librecord.a to NPB. Modify NPB-MPI/config/make.def. Compiling NPB code to generate an executable of NPB (with recording).

```
vim NPB-MPI/config/make.def
# CMPI_LIB    =    -lmpich
CMPI_LIB    = -L../common -lmyrecord -lmpich
$ cp /lib/librecord.a      NPB-MPI/common/
$ cd NPB-MPI/
$ make IS CLASS=C NPROCS=128
```

Then submit bin/is.C.128 to target HPC system to execute.

2) Emulation-execution phase

After recording the necessary information, we copy the recorded files to the share folder and add libmpiemulation.a to NPB. Modify NPB-MPI/config/make.def. Compiling NPB code to generate an executable of NPB (emulation-execution).

```
vim NPB-MPI/config/make.def
# CMPI_LIB    =    -lmpich
CMPI_LIB    = -L../common -lmpiemulation -lmpich -
lrdmacm -libverbs -lpthread
$ cp /lib/libmpiemulation.a      NPB-MPI/common/
$ cd NPB-MPI/
$ make IS CLASS=C NPROCS=128
```

Then use mpi_emulation_run.sh to start the emulation. If the users want to emulation process 0, 10, 20, 40 on node1 and node2 of the local cluster, they can do as follows. (If the users do not support the parameter target_mapping_file, it means that the process is distributed in order at each node in the actual- execution phase.)

```
vim emulation_mapping_file
node1
0,10
node2
20,40
mpi_emulation_run –n 128 –l 0,10,20,40 –p
bin/is.C.128 –m emulation_mapping_file
```

E. Evaluation and expected result

After the emulation-execution of a program, it output the result of the execution. Taking the program IS (1024 processes and the input data is class C) as an example. The result is shown as follows:

```
NAS Parallel Benchmarks 3.3 -- IS Benchmark

Size:  134217728  (class C)
Iterations:   10
Number of processes:     1024

IS Benchmark Completed
Class           =                        C
Size            =                134217728
Iterations      =                       10
Time in seconds =                     0.22
Total processes =                     1024
Compiled procs  =                     1024
Mop/s total     =                  6100.57
Mop/s/process   =                     5.99
Operation type  =              keys ranked
Verification    =                SUCCESSFUL
Version         =                     3.3.1
Compile date    =              16 Aug 2018
```