

APPENDIX A

ARTIFACT DESCRIPTION APPENDIX: WORKFLOW FOR PARALLEL PROCESSING OF SEQUENTIAL MESH DATABASES

A. Abstract

This appendix describes steps to re-run benchmarks that were used in the poster. We explain installation of the ESPRESO library that contains the implementation of the algorithm. Tests were performed on the Salomon supercomputer at IT4Innovations National Supercomputing Center. Thus we also describe its environment settings.

B. Description

1) Check-list (artifact meta information):

- **Algorithm:** the parallel loader of sequential mesh databases
- **Program:** C++ with MPI and OpenMP
- **Compilation:** mpiicpc version 17.0.1 (gcc version 6.3.0 compatibility) with `-O3` flags
- **Binary:** MPI/C++ executable
- **Data set:** ANSYS Workbench mesh datasets
- **Run-time environment:** CentOS release 6.9 (Final)
- **Hardware:** 2 x Intel Xeon E5-2680v3, 2.5 GHz, 12 cores; 128GB DDR4@2133 MHz interconnected by InfiniBand FDR56 / 7D hypercube
- **Output:** running times of algorithm steps
- **Experiment workflow:** clone and install ESPRESO, get mesh databases with configuration files, generate and run tests, observe the results
- **Experiment customization:** number of nodes, MPI processes, threads
- **Publicly available?:** the algorithm is publicly available, test databases are in a private storage due to their size (~200GB in total)

2) *How software can be obtained:* The ESPRESO library with the algorithm can be cloned from GitHub at <https://github.com/It4innovations/espresso>. The documentation of the installation process is available at espresso.it4i.cz.

3) *Hardware dependencies:* There should be no dependencies. The presented parallel loader was ran on the Salomon supercomputer at IT4Innovations National Supercomputing Center. Salomon consists of 1008 compute nodes (2 x Intel Xeon E5-2680v3, 2.5 GHz, 12 cores; 128GB DDR4@2133 MHz) interconnected by InfiniBand FDR56 / 7D hypercube. The rest of the ESPRESO library was successfully tested on the wide range of supercomputers including: Cray XK7 - ORNL Titan; Cray XC30 - CSCS Piz Daint, EPCC Archer; Cray XC40 - CSC SisU, HLRS Hornet. Since the newly implemented loader contains only MPI functions, it should be possible to run the loader at any architecture.

4) *Software dependencies:* The current version of ESPRESO is dependent on Intel-MKL, METIS, and ParMETIS. Even the parallel loader does not utilize any function from Intel-MKL, the library is mandatory for installation of ESPRESO (we are working on the version that avoids this dependency).

The ESPRESO library is tested with Intel compiler 17 and above (with the highest supported g++), g++ version 6.1 and above together with OpenMPI version 2 and above. Intel-MKL library should be at least version 2017 and above. We required C++11 compatibility and supporting of OpenMP. Versions of METIS and ParMETIS are 5.1.0 and 4.0.3, respectively.

5) *Datasets:* The algorithm was tested with ANSYS mesh databases. They are not publicly available due to their size (datasets have about 200GB in total). We store them at CES-NET (a long-term storage that is available at our department). We are able to provide files on demand. On the other hand, it is possible to test the loader with arbitrary database files compatible with ANSYS.

Each run of ESPRESO is controlled by ESPRESO configuration files (*.ecf). Used benchmarks configurations files are stored together with benchmarks data. A wide set of configuration files can be also found in `ESPRESO_ROOT/benchmarks` directory. In the case of testing the algorithm with another input data, these files can be easily used as a source for their configuration.

C. Installation

The latest version of ESPRESO library can be obtained from the git repository hosted at GitHub:

```
$ git clone https://github.com/It4innovations/espresso.git
```

The library contains configuration scripts for installation with Intel compiler or g++/OpenMPI in `$ESPRESO_ROOT/install` directory (in fact, scripts contain only compiler names and compilation flags that can be changed to fit the environment where the library is installed). An appropriate script has to be copied into the ESPRESO root directory. For installation with the Intel compiler use the following:

```
$ cp install/build.config.icpc build.config
```

The installation process considers that compilers and Intel-MKL is available in the environment. On Salomon (the cluster where benchmarks were tested), it is achieved by loading the following modules:

```
$ ml load intel/2017a
```

These modules can also be set via an environment configuration script `env/modules.salomon`. This script always contains the last set of modules that should be used for the correct installation. When the environment is set, it is possible to install ESPRESO using the following commands:

```
$ ./waf configure
$ ./waf install
```

Installation puts the executable into directory `$ESPRESO_ROOT/bin` and creates libraries in `$ESPRESO_ROOT/lib`. These paths have to be added to the environment path to run ESPRESO. In addition, the number of threads has to be set. The correct settings can be

done by the following commands (for usage of four threads per MPI process):

```
$ . env/paths.default
$ . env/threading.default 4
```

It finish the installation and the configuration process. Then the following command starts the computation (with `#n` MPI processes and the configuration in file `configuration.ecf`):

```
$ mpirun -n #n espresso -c configuration.ecf
```

D. Experiment workflow

To reproduce results from the poster one can use benchmarks and run them with the ESPRESO. Each benchmark has its own configuration file. The configuration file can be used for direct run of ESPRESO or it can be also used for generation of scalability tests using the ESPRESO internal test generator. Let files `JET_ENGINE_822M.dat` and `jet822M.ecf` are in the ESPRESO root directory. If the module with Python in version 2 is loaded, then the scalability test can be generated with the following command:

```
$ python tests/generator.py jet822M.ecf
```

The above creates the directory structure with appropriately configured benchmarks in directory `generatedtests/JET822M`. Besides benchmarks (*.ecf files) the directory contains a script that can be used for submitting the job on Salomon and a script for evaluate results.

Remark: since the scalability tests need 128 nodes on Salomon, a user has to set a project (in the submitting script) with enough free core hours before submitting the job.

```
$ cd generatedtest/JET822M
$ ./execute.sh
```

The above submit the job. When the job is finished, directory `generatedtests/JET822M/log` contains log files of all ran tests that contain data from a given run – application progress, statistical data, and run times of measured functions. Run times can be averaged and gathered into the table using the following script (in `generatedtests/JET822`):

```
$ python evaluator.py
```

It create tables with the results in directory `generatedtests/JET822M/stats`. Tables are text files at the csv format. Hence, it can be easily post-processed using an arbitrary spreadsheet editor. Tables correspond with steps that are presented in the poster.

E. Evaluation and expected result

The results of the benchmarks are running times (in seconds) of measured functions. Log files contain a more precise time evaluation that is presented in the poster. In the poster, measured functions are grouped into the three main blocks – reading and parsing the input mesh database, clusterization of the mesh using the space filling curve, and preprocessing of the mesh for the parallel solver (including calling of ParMETIS).

The running times of the blocks are expected to be similar as are presented in the poster with exception of the first block. Since the first block includes loading of the mesh database, the running times can be significantly influenced by other running jobs.

F. Experiment customization

The described workflow reproduces the same tests that are in the poster. Since all tests are generated using the ESPRESO test generator, it is simple to modify input settings for various customizations. The generator is a python script that gets a list of parameters and evaluates their available values. Then, the generator compute the Cartesian product of all parameters. Each element of the produced set defines a test case. Since tests are organized in the recursive directory structure, parameters are called *levels* in the generator. Python list generators are used to specify available values for each level:

```
LEVELS {
  1 [(x, "N_" + str(x).zfill(3)) for x in [128, 64, 32]];
  2 [(x, "MPI_" + str(x).zfill(3)) for x in [24, 12]];
}
```

The above code in the particular part in the configuration file prepares tests for all combination of two parameters with values `N_128`, `N_064`, `N_032` and `MPI_24`, `MPI_12`. The first parameter (level1) is used for the number of used nodes, the second parameter (level2) is used for the number of MPI processes per a node. The levels can be referenced in the configuration file by variables `L1` and `L2`. It allows to adjust tests for a given combination of parameters (e.g. the number of domains).

We hope that the rest of the generator description in the configuration file is self-explained. Hence, modifying available configuration files one can achieve desired behavior.

G. Notes

Please visit espresso.it4i.cz for the latest installation and configuration instructions.