

APPENDIX A

ARTIFACT DESCRIPTION APPENDIX: MPI/OPENMP
PARALLELIZATION OF THE FRAGMENT MOLECULAR
ORBITALS METHOD IN GAMESS

A. Abstract

This description contains the information needed to run some simulations for the SC18 poster “MPI/OpenMP Parallelization of the Fragment Molecular Orbital Method in GAMESS”.

B. Description

1) Check-list (artifact meta information):

- **Algorithm:** Fragment Molecular Orbital (FMO) method
- **Program:** GAMESS binary, Fortran and C libraries
- **Compilation:** Intel® Parallel Studio XE 2018 v2
- **Data set:** spherical water droplet, 30 Å radius, cc-pVTZ basis set
- **Run-time environment:** when running on Cray XC40 (Theta) the following modules were loaded:
 - craype/2.5.9
 - PrgEnv-intel/6.0.3
 - craype-mic-knl

when running on Stampede2 the following modules were loaded:

- git/2.9.0
- xalt/2.1.2
- intel/18.0.2
- impi/18.0.2
- autotools/1.1
- TACC
- libfabric/1.6.1
- **Hardware:** FMO1 benchmarks were run on Stampede2 supercomputer with up to 128 Intel® Xeon Phi™ 7250 processors; FMO2 benchmarks were run on Cray XC40 with up to 2,048 Intel® Xeon Phi™ 7230 processors
- **Experiment customization:** varied number of MPI ranks from 1 to 524,288, switching MPI and hybrid MPI+OpenMP codes
- **Publicly available?:** yes (partially)

2) *How software can be obtained (if available):* In this work, we used latest development snapshot of GAMESS software. The original GAMESS code can be downloaded freely from the official GAMESS website (registration required): <http://www.msg.ameslab.gov/gamesss/download.html>

Patches for GAMESS developed in this work can be currently obtained from the authors by request and later from the main GAMESS repository.

3) *Hardware dependencies:* All measurements in this work were performed on the 2nd generation of Intel® Xeon Phi™ CPU. However, the same benchmarks can be run on any other CPU architectures.

4) *Software dependencies:* C (C99 standard) compiler, Fortran 95 compatible compiler with OpenMP 3.0 support, MPI library. The code has been extensively tested only for Intel® Parallel Studio XE 2018 update 2 compilers and libraries. MKL BLAS library was used to link GAMESS.

5) *Datasets:* Structure of chemical system and corresponding input files used for benchmarking code can be downloaded from <https://github.com/gms-bbg/gamesss-papers> repository. It is a 30 Å spherical water droplet containing 3,932 water molecules (11,796 atoms). The basis set used in all calculations is cc-pVTZ.

C. Installation

We followed the standard installation procedure outlined in `PROG.DOC` in GAMESS root directory. GAMESS uses a custom build configuration system. The first step is configuration of the `install.info` file. To perform the basic configuration one need to run `${GMS_HOME}/config` script and specify compilers and libraries for the compilation. When the script asks whether to compile GAMESS with LIBC-CHEM one need to refuse. After that the `install.info` and `Makefile` files can be edited directly to get the desired configuration. The relevant variable to compile GAMESS with OpenMP support is `GMS_OPENMP=true` in both files. After setup is finished GAMESS compilation can be done with `make` command. If it succeeds the file `games.${(VERNO)}.x` will appear in GAMESS home directory, where `$(VERNO)` is a variable specified at basic configuration step with `${GMS_HOME}/config` script. `VERNO="00"` by default.

D. Experiment workflow

We used the standard workflow of the experiment: code was compiled and ran on different Intel® Xeon Phi™ systems. We varied number of nodes from 1 to 2,048 depending on the target supercomputer while keeping number of threads/MPI ranks per node fixed. We used 256 MPI processes per node for benchmarking MPI code. In this setup, 128 working processes and 128 GAMESS data servers were ran on each node. We used 16 MPI ranks per node for benchmarking hybrid MPI/OpenMP code. Thus, 8 working processes and 8 GAMESS data servers were ran on each node. The number of threads per working process was 16, resulting in 128 threads per node.

E. Evaluation and expected result

We ran GAMESS on Stampede 2 using the following submission line for OpenMP code:

```
mpirun -n $2 \
  -env OMP_NUM_THREADS $3 \
  -env I_MPI_SHM_LMT shm \
  -env KMP_STACKSIZE 20m \
  -env I_MPI_PIN_DOMAIN <MASK> \
  -env KMP_AFFINITY scatter \
  -env DDI_LOGICAL_NODE_SIZE 1 \
  -env <GAMESS options> games.00.x
```

On XC40 system, we used the following submission line using Cobalt queuing system for OpenMP code:

```
rpn=Number of MPI Ranks per node
allranks=$((COBALT_JOB_SIZE*rpn))
threads=Number of OMP threads per MPI rank
aprun -n $allranks \
  -N $rpn -d $threads -cc depth -j 4 \
  --env OMP_NUM_THREADS=$threads \
  --env I_MPI_PIN_DOMAIN=<MASK> \
  --env KMP_AFFINITY=scatter \
  --env KMP_STACKSIZE=2000m \
  --env DDI_LOGICAL_NODE_SIZE=1 \
  --env <GAMESS options> games.00.x
```

For MPI code we used this submission line:

```

rpn=Number of MPI Ranks per node
allranks=$((COBALT_JOBSIZE*rpn))
aprun -n $allranks \
    -N $rpn -d 1 -cc depth -j 4 \
    --env I_MPI_PIN_DOMAIN=auto \
    --env DDI_LOGICAL_NODE_SIZE=16 \
    --env <GAMESS options> gamess.00.x

```

For the hybrid code it is extremely important to correctly setup `I_MPI_PIN_DOMAIN` variable to the specific mask, which is dependent on the value of `OMP_NUM_THREADS` and `DDI_LOGICAL_NODE_SIZE` variables. In this work we used 8 MPI workers per node + 8 data servers per node. Each worker uses 16 threads. Logical size of DDI node was set to 1. For this setup the correct mask will be as follows:

```

I_MPI_PIN_DOMAIN=[\
'ff00000000000000000000000000ff',\
'ff000000000000000000000000ff0000000000000',\
'ff000000000000000000000000ff00',\
'ff000000000000000000000000ff0000000000000',\
'ff000000000000000000000000ff0000',\
'ff000000000000000000000000ff0000000000000',\
'ff000000000000000000000000ff000000',\
'ff000000000000000000000000ff0000000',\
'ff000000000000000000000000ff00000000',\
'ff000000000000000000000000ff000000000',\
'ff000000000000000000000000ff0000000000',\
'ff000000000000000000000000ff00000000000',\
'ff000000000000000000000000ff000000000000',\
'ff000000000000000000000000ff0000000000000',\
'ff000000000000000000000000ff00000000000000',\
'ff000000000000000000000000ff000000000000000']

```

The results of the computation are printed to `STDOUT` or redirected to the file specified in submission/run script. Last timing can be obtained with the following command:

```

$ grep 'WALL CLOCK' <logfile> \
  | tail -1 \
  | awk '{print $5}'

```

It will return the time in seconds of the calculation.

F. Experiment customization

During the experiment we changed the following runtime parameters:

- Total number of MPI processes with batch script parameters
- Optimal balance between MPI processes and OpenMP threads per MPI process by adjusting `OMP_NUM_THREADS` environment variable and number of MPI tasks per node
- Parameters of Intel MPI library (`I_MPI_PIN_DOMAIN`)