NEMO

Cevdet Aykanat and Ozan Karsavuran – 26/02/2020

Summary Version

v0.1

Purpose of Benchmark

NEMO (Nucleus for European Modelling of the Ocean) is a mathematical modelling framework for research activities and prediction services in ocean and climate sciences developed by a European consortium. It is intended to be a tool for studying the ocean and its interaction with the other components of the earth climate system over a large number of space and time scales. It comprises of the core engines namely OPA (ocean dynamics and thermodynamics), SI3 (sea ice dynamics and thermodynamics), TOP (oceanic tracers) and PISCES (biogeochemical process).

Prognostic variables in NEMO are the three-dimensional velocity field, a linear or non-linear sea surface height, the temperature and the salinity. In the horizontal direction, the model uses a curvilinear orthogonal grid and in the vertical direction, a full or partial step z-coordinate, or s-coordinate, or a mixture of the two. The distribution of variables is a three-dimensional Arakawa C-type grid for most of the cases.

Characteristics of Benchmark

The model is implemented in Fortran 90, with pre-processing (C-pre-processor). It is optimized for vector computers and parallelized by domain decomposition with MPI. It supports modern C/C++ and FORTRAN compilers. All input and output is done with third party software called XIOS with a dependency on NetCDF (Network Common Data Format) and HDF5. It is highly scalable and a perfect application for measuring supercomputing performances in terms of compute capacity, memory subsystem, I/O and interconnect performance.

Mechanics of Building Benchmark

Building XIOS

1. Download the XIOS source code

   ```
   svn co https://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-2.5
   ```

2. There are available known architectures which can be seen with the following command:
   `./make_xios –avail`
   If target architecture is a known one, it can be built by the following command
   `./make_xios --arch X64_CURIE`
   Otherwise `arch-local.env`, `arch-local.fcm`, `arch-local.path` files should be placed according to target architecture. Then build by:
   `./make_xios --arch local`

Note that XIOS requires Netcdf4. Please load the appropriate HDF5 and NetCDF4 modules. You might have to change the path in the configuration file.

Building NEMO

1. Download the XIOS source code

   ```
   svn co https://forge.ipsl.jussieu.fr/nemo/svn/NEMO/releases/release-4.0
   ```

2. Copy and setup the appropriate architecture file in the arch folder. The following changes are recommended:
   a. add the `-lnetcdff` and `-lstdc++` flags to NetCDF flags
   b. using `mpif90` which is a MPI binding of `gfortran-4.9`
   c. add `-cpp` and `-ffree-line-length-none` to Fortran flags
   d. swap out `gmake` with `make`
3. Then build the executable with the following command

   ```
   ./makenemo -m MY_CONFIG -r GYRE_XIOS -n MY_GYRE add_key "key_nosignedzero"
   ```

4. Apply the patch as described here to measure step time :
   `https://software.intel.com/en-us/articles/building-and-running-nemo-on-xeon-processors`

Mechanics of Running Benchmark

Prepare input files

```
cd MY_GYRE/EXP00
sed -i '/using_server/s/false/true/' iodef.xml
sed -i '/&nameos/a ln_useCT = .false.' namelist_cfg
sed -i '/&namctl/a nn_bench = 1' namelist_cfg
```

Run the experiment interactively

```
mpirun -n 4 ../BLD/bin/nemo.exe -n 2 $PATH_TO_XIOS/bin/xios_server.exe
```

GYRE configuration with higher resolution

Modify configuration (for example for the test case A):

```
rm -f time.step solver.stat output.namelist.dyn ocean.output  slurm-*  GYRE_* mesh_mask_00*
jp_cfg=4
sed -i -r \
    -e 's/^( *nn_itend *=).*/\1 21600/' \
    -e 's/^( *nn_stock *=).*/\1 21600/' \
    -e 's/^( *nn_write *=).*/\1 1000/' \
    -e 's/^( *jp_cfg *=).*/\1 '"$jp_cfg"'/' \
    -e 's/^( *jpidta *=).*/\1 '"$(( 30 * jp_cfg +2))"'/' \
    -e 's/^( *jpjdta *=).*/\1 '"$(( 20 * jp_cfg +2))"'/' \
    -e 's/^( *jpiglo *=).*/\1 '"$(( 30 * jp_cfg +2))"'/' \
    -e 's/^( *jpjglo *=).*/\1 '"$(( 20 * jp_cfg +2))"'/' \
    namelist_cfg
```

Verification of Results

The GYRE configuration is set through the `namelist_cfg` file. The horizontal resolution is determined by setting `jp_cfg` as follows:

- `Jpiglo = 30 × jp_cfg + 2`
- `Jpjglo = 20 × jp_cfg + 2`

In this configuration, we use a default value of 30 ocean levels, depicted by `jpk=31`. The GYRE configuration is an ideal case for benchmark tests as it is very simple to increase the resolution and perform both weak and strong scalability experiment using the same input files. We use two configurations as follows:

Test Case A:

- `jp_cfg = 128` suitable up to 1000 cores
- Number of Days: 20
- Number of Time steps: 1440
- Time step size: 20 mins
- Number of seconds per time step: 1200

Test Case B:

- `jp_cfg = 256` suitable up to 20,000 cores.
- Number of Days (real): 80
- Number of time step: 4320
- Time step size(real): 20 mins
- Number of seconds per time step: 1200

We performed scalability test on 512 cores and 1024 cores for test case A. We performed scalability test for 4096 cores, 8192 cores and 16384 cores for test case B.

Both these test cases can give us quite good understanding of node performance and interconnect behavior. We switch off the generation of mesh files by setting the flag `nn_mesh = 0` in the `namelist_ref` file. Also `using_server = false` is defined in `io_server` file.

We report the performance in step time which is the total computational time averaged over the number of time steps for different test cases. This helps us to compare systems in a standard manner across all combinations of system architectures. The other main reason for reporting time per computational time step is to make sure that results are more reproducible and comparable.

Since NEMO supports both weak and strong scalability, test case A and test case B both can be scaled down to run on smaller number of processors while keeping the memory per processor constant achieving similar results for step time. To measure the step time, we inserted a patch which includes the `MPI_wtime()` functional call in `nemogcn.f90` file for each step which also cumulatively adds the step time until the second last step. We then divide the total cumulative time by the number of time steps to average out any overhead.

Sources

`https://forge.ipsl.jussieu.fr/nemo/chrome/site/doc/NEMO/guide/html/install.html`

`https://forge.ipsl.jussieu.fr/ioserver/wiki/documentation`

`https://nemo-related.readthedocs.io/en/latest/compilation_notes/nemo37.html`