LFRic and Singularity Hackerthon

This document details the containerisation of LFRic in Singularity, and instructions on how to run on Piz Daint. The general requirements for LFRIC containerisation discussed, followed by the Singularity containerisation workflow and a description of how to install on Piz Daint.

LFRic containerisation requirements

Requirements

1. Put the LFRic build environment into a portable package that can be deployed on different x86 target machines to build the LFRic executable.

2. Produce an executable that is able to make use of local MPI libraries and therefore local fast interconnects.

3. Produce an executable that be run on the native system with no runtime dependencies on the build environment to minimise the conflict between the packaged libraries and local libraries used for MPI and job control.

The LFRic build environment at
https://code.metoffice.gov.uk/trac/lfric/wiki/LFRicTechnical/LFRicBuildEnvironment
was used. As the initial purpose of this project was to simplify building with the Intel compiler, gfortran was replaced by Intel fortran throughout.

The build environment was containerised and Singularity was chosen container system due to its widespread use on HPCs and Tier II systems.

Pre-requisites

- Intel Fortran 17 or 19 on build and run system.
- Singularity on build and run system.
- Ability to run Singularity in sudo on build system.
- MPICH based MPI on run machine. This includes MPICH, Intel MPI, Cray MPT and MVAPICH.

Theory

Locally installed software can be used from inside a containerised shell. The local Intel compiler can be accessed via either using bind points when the container shell is invoked, or by including environment-modules in the container, bind mounting the compiler's location and then use the 'module load' command to set up the compiler.

MPI ABI https://www.mpich.org/abi This provides compatibility between MPICH used to build the a execrable and the local MPICH derivative used by the executable at run-time. Therefore it is possible to build the executable using one MPICH derivative, and run it using a different MPICH derivative.

Any statically compiled library will be included in the executable at link time. Therefore there are no run-time dependency on these libraries outside the container.

Build environment containerisation workflow

To facilitate requirements 2) and 3) above, the following changes were made to the standard LFRic build environment.

1. gfortran replaced by Intel fortran throughout, and any necessary changes made to compile flags and configure option.
2. All packages configured to produce static libraries.
3. The exception to 2) are the MPI libraries, which were built with shared libraries to use MPI ABI.

LFRic software stack workflow

See https://github.com/NCAS-CMS/LFRic_Container for a full description of the workflows.

1. Build a base build container comprising of gcc and build tools only.
2. Start a shell inside this container, mounting a bind point for the location of the local Intel compiler. Modules can also be used for this step. The top-level locations of the module configuration and Intel compiler (if different) need to be mounted as bind points. The compiler then can

be used after the usual "module load" command. Then build all the LFRic software dependencies using a common installation directory.

3. Build the LFRic dependency software stack. Tar this directory.

4. Build a second, final container with the tarball generated by 3) plus all build python dependencies and also environment-modules.

LFRic build workflow

1. Copy container to target machine.

2. Run a shell inside the container. Configure the Intel compiler as described above, again suitable bind points are required. Run the setup tool.

3. Build LFRic as usual. The environment is pre-configured. Change the Makefile thus:

```
export EXTERNAL_DYNAMIC_LIBRARIES =
EXTERNAL_STATIC_LIBRARIES = yaxt yaxt_c xios netcdff
   netcdf hdf5_hl hdf5 z :libstdc++.a
```

LFRic run workflow

1. Outside the container, set up MPICH based MPI

2. Run LFRic as usual with mpiexec.

Example of the dependencies of the built container inside and outside the container

With no Intel nor MPICH2 libraries in PATH:

```
$ ldd gungho
linux-vdso.so.1 (0x00007fff5cad1000)
libz.so.1 => /usr/lib/libz.so.1 (0x00001514fec3a000)
libmpifort.so.12 => not found
libmpi.so.12 => not found
libm.so.6 => /usr/lib/libm.so.6 (0x00001514feaf4000)
libiomp5.so => not found
```

```
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x00001514fead
   3000)
libc.so.6 => /usr/lib/libc.so.6 (0x00001514fe90e000)
/lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
   (0x00001514fee82000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x00001514fe8f4000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x00001514fe8ef000)
```

Inside container:

```
$ ldd gungho
linux-vdso.so.1 => (0x00007ffe77af6000)
libz.so.1 => /lib64/libz.so.1 (0x00007fbfcc793000)
libmpifort.so.12 => /container/usr/lib/libmpifort.so.12 (0x
   00007fbfcc556000)
libmpi.so.12 => /container/usr/lib/libmpi.so.12 (0x00007fbfcc0c
   0000)
libm.so.6 => /lib64/libm.so.6 (0x00007fbfcbdbe000)
libiomp5.so => /opt/intel/compilers_and_libraries_2017.4.196/
   linux/compiler/lib/intel64/libiomp5.so (0x00007fbfcba1a000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fbfcb7fe000)
libc.so.6 => /lib64/libc.so.6 (0x00007fbfcb430000)
/lib64/ld-linux-x86-64.so.2 (0x00007fbfcc9a9000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007fbfcb21a000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007fbfcb016000)
librt.so.1 => /lib64/librt.so.1 (0x00007fbfcae0e000)
libifport.so.5 => /opt/intel/compilers_and_libraries
   _2017.4.196/linux/compiler/lib/intel64/libifport.so.5 (0x
   00007fbfcabdf000)
libifcoremt.so.5 => /opt/intel/compilers_and_libraries
   _2017.4.196/linux/compiler/lib/intel64/libifcoremt.so.5 (0x
   00007fbfca84f000)
libimf.so => /opt/intel/compilers_and_libraries_2017.4.196/
   linux/compiler/lib/intel64/libimf.so (0x00007fbfca362000)
libintlc.so.5 => /opt/intel/compilers_and_libraries_2017.4.196/
   linux/compiler/lib/intel64/libintlc.so.5 (0x00007fbfca0f
   7000)
libsvml.so => /opt/intel/compilers_and_libraries_2017.4.196/
   linux/compiler/lib/intel64/libsvml.so (0x00007fbfc91de000)
```

On target machine using Intel MPI:

```
$ ldd gungho
linux-vdso.so.1 (0x00007ffd0e98e000)
libz.so.1 => /usr/lib/libz.so.1 (0x000014e33d8e0000)
libmpifort.so.12 => /opt/intel/compilers_and_libraries
   _2017.4.196/linux/mpi/intel64/lib/libmpifort.so.12 (0x
   000014e33d537000)
libmpi.so.12 => /opt/intel/compilers_and_libraries_2017.4.196/
   linux/mpi/intel64/lib/libmpi.so.12 (0x000014e33c80f000)
libm.so.6 => /usr/lib/libm.so.6 (0x000014e33c6c9000)
libiomp5.so => /opt/intel/compilers_and_libraries_2017.4.196/
   linux/compiler/lib/intel64/libiomp5.so (0x000014e33c325000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x000014e33c
   302000)
libc.so.6 => /usr/lib/libc.so.6 (0x000014e33c13f000)
/lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
   (0x000014e33db28000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x000014e33c125000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x000014e33c120000)
librt.so.1 => /usr/lib/librt.so.1 (0x000014e33c115000)
```

Note that there are no dependencies of the LFRic software stack, and that inside the container libmpifort.so.12 and libmpi.so.12 under /container/usr are listed as dependencies, while outside they are under /opt/intel/compilers_and_libraries_2017.4.196/linux/mpi/intel64/lib as part of Intel MPI.

Installing on Piz Daint

Build container

Build the Singularity LFRic build system as described in step 1-4 in https://github.com/NCAS-CMS/LFRic_Container then copy to Piz Daint.

On Piz Daint

```
unset LD_PRELOAD # There is a general LD_PRELOAD with
   interferes with Singularity, but isn't required for it.
module swap PrgEnv-cray PrgEnv-intel
```

```
module load singularity
svn checkout --username <username> https://code.metoffice.gov.
   uk/svn/lfric/LFRic/trunk
```

For Gungho the Makefile needs to be edited. Change the lines:

```
export EXTERNAL_DYNAMIC_LIBRARIES = yaxt yaxt_c netcdff netcdf
   hdf5 \
                                    $(CXX_RUNTIME_LIBRARY)
export EXTERNAL_STATIC_LIBRARIES = xios
```

to

```
export EXTERNAL_DYNAMIC_LIBRARIES =
export EXTERNAL_STATIC_LIBRARIES = yaxt yaxt_c xios netcdff
   netcdf hdf5_hl hdf5 z :libstdc++.a
```

Start the container

```
singularity shell -B /opt/intel:/opt/intel lfric_usr.sif #Start
    singularity with bind points for the local Intel compilers
   .
```

Inside the container

```
. /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/
   ifortvars.sh intel64 #Change compilers_and_libraries
   _2017.4.196 to match the same major number of the Intel
   compiler used to build the container. Ignore the "WARNING:
   'gcc' was not found" message.
. /container/setup #Set up LFRic compile environment
cd trunk/gungho # or to the downloaded location.
make build #Build the main exec.
```

Log out of the container. The executable built inside the container can be run outside the container using the standard Slurm submission system.

Submission

The following submission script will set up the Intel compiler version used to build the executable, and ensure that the Cray MPICH-ABI libraries are used at run time.

```
#!/bin/bash -l
```

```
#SBATCH --job-name="gungho"
#SBATCH --time=00:05:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-core=1
#SBATCH --ntasks-per-node=6
#SBATCH --cpus-per-task=1
#SBATCH --partition=normal
#SBATCH --constraint=gpu
#SBATCH --hint=nomultithread
#SBATCH --reservation=esiwace_1
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module swap PrgEnv-cray PrgEnv-intel
module swap intel/19.0.1.144 intel/17.0.4.196
module unload cray-mpich
module load cray-mpich-abi
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
# The following line should be customised to match the location
    of wlm_detect on the system.
export LD_LIBRARY_PATH=/opt/cray/wlm_detect/1.3.3-7.0.1.1_4.6__
   g7109084.ari/lib64:$LD_LIBRARY_PATH
export CRAY_ROOTFS=UDI
cd $HOME/trunk/gungho/example
echo $LD_LIBRARY_PATH
ldd ../bin/gungho
srun ../bin/gungho gungho_configuration.nml
```

Conclusion

The LFRic build infrastructure singularity container was built on a laptop during the hackathon and then deployed and run on Piz Daint using the local MPT libraries. As the executable was run natively using the MPT on the system there was no slow down with respect to an executable built without the use of the container. The same container has been used to build LFRic on a laptop and Cirrus, a UK Tier 2 HPC.