GRChombo: building, basic profiling and debugging

Miren Radia, DAMTP, University of Cambridge

Conventions

- `shell command`
- `/path/to/directory` or `filename.ext`
- `Makefile configuration`
- `C++ code`

Chombo Prerequisites & Dependencies

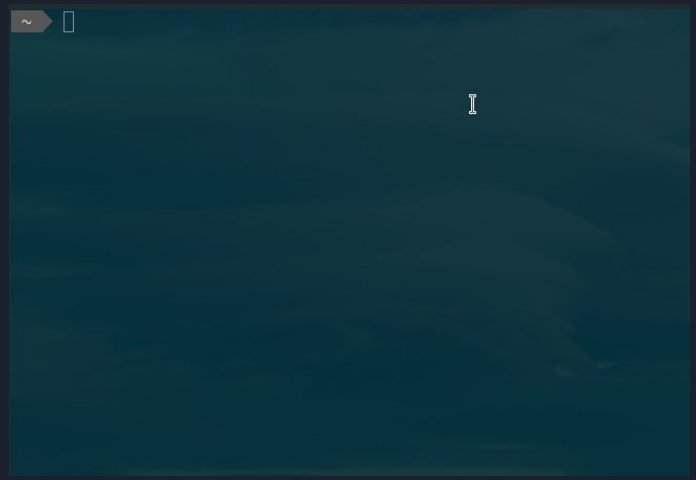
intel®

L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K

- Libraries and other dependencies:
  - GNU Make (`make`)
  - C shell (`csh`)
  - Perl
  - Compilers:
    - C++14: Intel Classic v ≥ 17, GCC v ≥ 5 or LLVM Clang (and derivatives) v ≥ 3.4
    - Corresponding Fortran compiler
  - MPI: compatible with C++ compiler, preferably MPICH based
  - HDF5
    - Serial and parallel?
  - BLAS (Basic Linear Algebra Subprograms) & LAPACK (Linear Algebra Package)
- Clusters: `modules`
- Own PC: Use package managers e.g.
  - Ubuntu/Debian: `apt`
  - macOS: Homebrew

Using modules on clusters

- Many clusters use some kind of `modules` command to control the build environment
- Useful commands:
  - `module avail [<name>]`
    - What modules are available?
  - `module [un]load <module name>`
    - Load/unload a module
  - `module swap <old module> <new module>`
    - Swap one module for another
  - `module list`
    - What modules do I have loaded
  - `module display <module name>`
    - Tell me more about `<module name>`
- Some clusters use `spack` to manage the modules
  - Once you have `source`d the spack `setup-env.sh` script, you can query dependencies with:
    `spack find -lvd <name>`



Make.defs.local

- This is the list of Makefile variables that Chombo uses to build
- Place in `Chombo/lib/mk/Make.defs.local`
- Main Variables:
  - `DIM [= 3]`
  - `DEBUG [= TRUE/FALSE]`
  - `OPT [= FALSE/TRUE/HIGH]`
  - `PRECISION [= FLOAT/DOUBLE]`
  - `CXX [= <C++ compiler>]`
  - `FC [= <Fortran compiler>]`
  - `OPENMPCC [= TRUE/FALSE]`
  - `MPI [= TRUE/FALSE]`
  - `MPICXX [= <MPI C++ compiler>]`
- HDF5:
  - Use `module display <hdf5 module>` to get paths!
  - `USE_HDF [= TRUE/FALSE]`
  - `HDFINCFLAGS [= -I<hdf5 serial include path>]`
  - `HDFLIBFLAGS [= -L<hdf5 serial lib path> -lhdf5 -lz]`
  - `HDFMPIINCFLAGS [= -I<hdf5 parallel include path>]`
  - `HDFMPILIBFLAGS [= -L<hdf5 parallel lib path> -lhdf5 -lz]`

```
DIM             = 3
DEBUG           = TRUE
PRECISION       = DOUBLE
OPT             = TRUE
PROFILE         = FALSE
CXX             = g++
FC              = gfortran
MPI             = TRUE
OPENMPCC        = TRUE
MPICXX          = mpicxx
USE_64          = TRUE
USE_HDF         = TRUE
HDFINCFLAGS     = -I/usr/lib/x86_64-linux-gnu/hdf5/serial/include
HDFLIBFLAGS     = -L/usr/lib/x86_64-linux-gnu/hdf5/serial/lib -lhdf5 -lz
HDFMPIINCFLAGS  = -I/usr/lib/x86_64-linux-gnu/hdf5/openmpi/include
HDFMPILIBFLAGS  = -L/usr/lib/x86_64-linux-gnu/hdf5/openmpi/lib -lhdf5 -lz
USE_MT          = FALSE
cxxoptflags     = -march=native -O3
foptflags       = -march=native -O3
syslibflags     = -lblas -llapack
```

Example Make.defs.local for Ubuntu 18.04

Make.defs.local (continued)

- Other useful variables:
  - `[cxx/f/cpp][dbg/opt]flags` (e.g. `cxxoptflags`): C++/F/Cpp specific flags depending on OPT/DEBUG settings
  - CXXFLAGS, CPPFLAGS, FFLAGS, LDFLAGS: Flags common to all relevant compilations
  - `syslibflags` e.g. `[= -lblas -llapack]`
  - XTRACONFIG: Useful variable which appends to all filenames - allows you to keep multiple builds compiled with different settings
  - See `Chombo/lib/mk/Make.defs.local.template` for a more comprehensive list of other variables
- Example `Make.defs.locals` are in `GRChombo/InstallNotes/MakeDefsLocalExamples/`

```
DIM             = 3
DEBUG           = TRUE
PRECISION       = DOUBLE
OPT             = TRUE
PROFILE         = FALSE
CXX             = g++
FC              = gfortran
MPI             = TRUE
OPENMPCC        = TRUE
MPICXX          = mpicxx
USE_64          = TRUE
USE_HDF         = TRUE
HDFINCFLAGS     = -I/usr/lib/x86_64-linux-gnu/hdf5/serial/include
HDFLIBFLAGS     = -L/usr/lib/x86_64-linux-gnu/hdf5/serial/lib -lhdf5 -lz
HDFMPIINCFLAGS  = -I/usr/lib/x86_64-linux-gnu/hdf5/openmpi/include
HDFMPILIBFLAGS  = -L/usr/lib/x86_64-linux-gnu/hdf5/openmpi/lib -lhdf5 -lz
USE_MT          = FALSE
cxxoptflags     = -march=native -O3
foptflags       = -march=native -O3
syslibflags     = -lblas -llapack
```

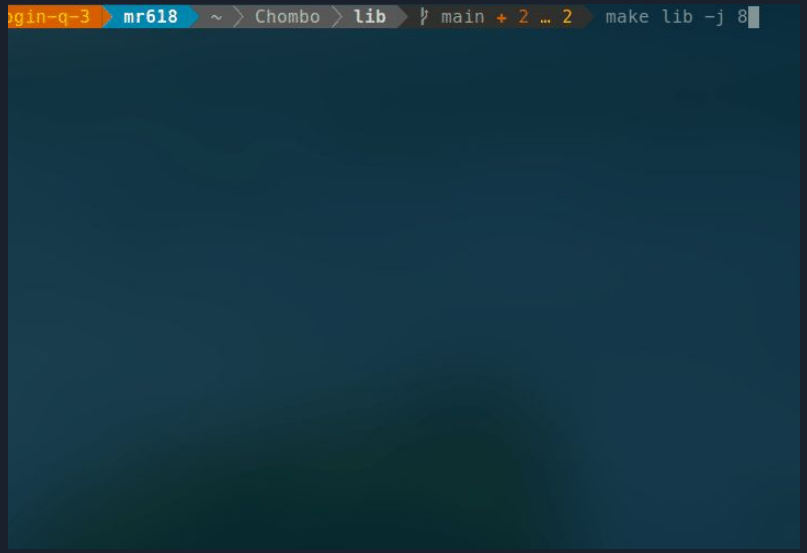
Example Make.defs.local for Ubuntu 18.04

Optimizing: speeding up your code

- Optimization Level
  - `-On` where `n` is 1, 2 or 3
  - Differences between compilers
    - Intel is generally more aggressive than GCC (Intel O2 ≈ GCC O3)
- Vectorization/architecture
  - Target current architecture:
    - GCC: `-march=native`
    - Intel: `-xHost`
  - Target specific architecture
    - Look up in cluster documentation (examples below for Skylakes)
    - GCC: `-march=skylake-avx512`
    - Intel: `-xCORE-AVX512 -qopt-zmm-usage=high`
- Some of these done automatically (see `Make.defs.<compiler>`)

Building (Finally!)

1. Build Chombo
   a. `cd` into `Chombo/lib`
   b. `make all -j <n>` where `n` is the number of parallel processes to build libraries and tests or `make lib -j <n>` just to build the libraries
2. Build GRChombo
   a. `export CHOMBO_HOME=/path/to/Chombo/lib`
   b. `cd` into `GRChombo` base directory
   c. `make all -j <n>` to build and run tests, and build examples
3. Profit 💪



Building Chombo on CSD3

[Very] basic profiling: Using Chombo timers

- To use Chombo timers
  - Add at start of functions:
    `CH_TIME("<function_name>");` e.g.

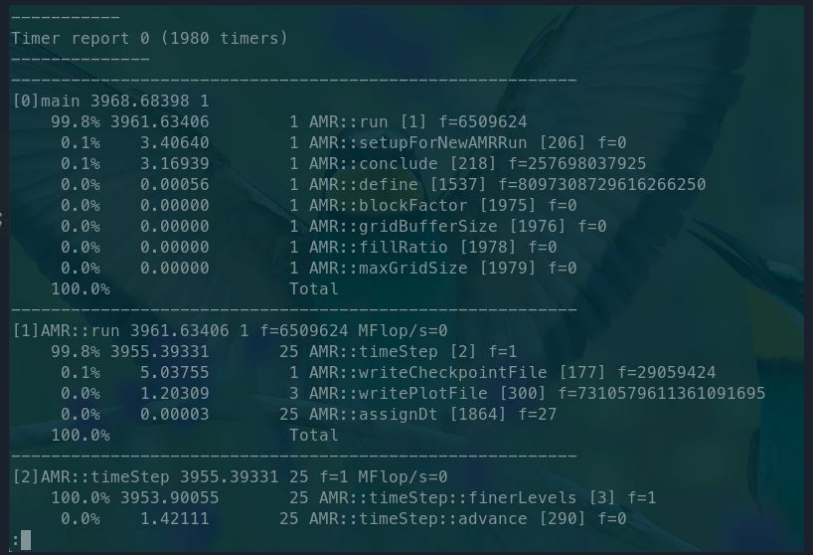
```
void BinaryBHLevel::specificPostTimeStep()
{
    CH_TIME("BinaryBHLevel::specificPostTimeStep");
```

  - Add at end of `runGRChombo()`:
    `CH_TIMER_REPORT();` e.g.

```
    CH_TIMER_REPORT();

    return 0;
}
```

  - `export CH_TIMER=<something>`
  - Look at `time.table.<n>`



```
-----------
Timer report 0 (1980 timers)
-------------
------------------------------------------------------------
[0]main 3968.68398 1
    99.8% 3961.63406        1 AMR::run [1] f=6509624
     0.1%    3.40640        1 AMR::setupForNewAMRRun [206] f=0
     0.1%    3.16939        1 AMR::conclude [218] f=257698037925
     0.0%    0.00056        1 AMR::define [1537] f=8097308729616266250
     0.0%    0.00000        1 AMR::blockFactor [1975] f=0
     0.0%    0.00000        1 AMR::gridBufferSize [1976] f=0
     0.0%    0.00000        1 AMR::fillRatio [1978] f=0
     0.0%    0.00000        1 AMR::maxGridSize [1979] f=0
    100.0%                  Total
------------------------------------------------------------
[1]AMR::run 3961.63406 1 f=6509624 MFlop/s=0
    99.8% 3955.39331       25 AMR::timeStep [2] f=1
     0.1%    5.03755        1 AMR::writeCheckpointFile [177] f=29059424
     0.0%    1.20309        3 AMR::writePlotFile [300] f=7310579611361091695
     0.0%    0.00003       25 AMR::assignDt [1864] f=27
    100.0%                  Total
------------------------------------------------------------
[2]AMR::timeStep 3955.39331 25 f=1 MFlop/s=0
    100.0% 3953.90055       25 AMR::timeStep::finerLevels [3] f=1
     0.0%    1.42111       25 AMR::timeStep::advance [290] f=0
:
```

Example `time.table.0` output

[Very] basic debugging

- Print debugging using `pout()`
  - Easy to use
  - Good first step
  - Can be slow to converge?
- GNU Project Debugger: `gdb`
  - Compile with debugging flags `-g` or `DEBUG=TRUE` and `MPI=FALSE`
  - A bit of a learning curve
  - Can be quicker on more complicated bugs?
  - Widely available on clusters
- Other debuggers?
  - Many exist
  - Some not free but might be available on some clusters
  - I don't have much experience

Messing around with `gdb`

Any questions?