What SWIFT can do

SWIFT can solve a variety of problems aimed at cosmological and astrophysical applications. SWIFT's features include:

- Hydrodynamics, using a variety of particle methods
- Planetary science, with e.g. multiple equations of state
- Dark Matter
- Neutrinos
- Gravity: self-gravity and external potentials
- Cosmology
- Radiative cooling
- Radiative transfer
- On-the-fly analysis: halo finding (FOF), power spectra
- And more!

To enable and use these features, SWIFT needs to be compiled accordingly and corresponding flags need to be passed at runtime. Please consult the instructions provided in the documentation for full details.

Getting The Code

The code is available from our GitLab (core developers) and GitHub (public mirror) repositories. You can download it over https from the following locations:

- https://github.com/swiftsim/swiftsim.git
- https://gitlab.cosma.dur.ac.uk/swift/swiftsim.git

Getting Help

Feel free to contact us on Gitter (gitter.im/swiftsim) or on our GitHub (github.com/swiftsim/swiftsim) by creating an issue.

The code documentation is available on swiftsim.com/docs, and is also shipped along with the code in the `docs/RTD` directory. This onboarding guide is available online as well on swiftsim.com/onboarding.pdf

Initial Setup

We use autotools for setup. To get a basic running version of the code (the executable binaries are found in the top directory), use:

```
./autogen.sh
./configure
make
```

MacOS Specific Oddities

To build on MacOS you will need to enable compiler warnings due to an incomplete implementation of pthread barriers. DOXYGEN also has some issues on MacOS, so it is best to leave it out. To configure:

```
./configure --enable-compiler-warnings \
    --disable-doxygen-doc
```

When using the `clang` compiler, the hand-written vectorized routines have to be disabled. This is done at configuration time by adding the flag `--disable-hand-vec`.

Dependencies

To compile SWIFT, you will need the following libraries:

HDF5

Version 1.10.x or higher is required. Input and output files are stored as HDF5 and are compatible with the GADGET-2 specification. A parallel-HDF5 build and HDF5 >=1.12.x is recommended when running over MPI.

MPI

A recent implementation of MPI, such as Open MPI (v3.x or higher), is required, or any library that implements at least the MPI 3 standard.

Libtool

The build system depends on libtool.

FFTW

Version 3.3.x or higher is required for periodic gravity.

ParMETIS or METIS

One is required for domain decomposition and load balancing.

GSL

The GSL 2.x is required for cosmological integration.

In most cases the configuration script will be able to detect the libraries installed on the system. If that is not the case, the script can be pointed towards the libraries' location using the following parameters

```
./configure --with-gsl=<PATH-TO-GSL>
```

and similar for the other libaries.

Optional Dependencies

There are also the following *optional* dependencies:

libNUMA

libNUMA is used to pin threads.

TCmalloc/Jemalloc/TBBmalloc

TCmalloc/Jemalloc/TBBmalloc are used for faster memory allocations when available.

Python

To run the examples, you will need python 3 and some of the standard scientific libraries (numpy, matplotlib). Some examples make use of the swiftsimio library, which is a dedicated and maintained visualisation and analysis library for SWIFT.

GRACKLE

GRACKLE cooling is implemented in SWIFT. If you wish to take advantage of it, you will need it installed.

HEALPix C library

This is required for making light cone HEALPix maps.

CFITSIO

This may be required as a dependency of HEALPix.

Useful Configuration Flags

A description of the available options for all flags including the examples below can be found by using `./configure --help`.

`--with-hydro=sphenix`

There are several hydrodynamical schemes available in SWIFT. You can choose between them at compile-time with this option.

`--with-riemann-solver=none`

Some hydrodynamical schemes, for example GIZMO, require a Riemann solver.

`--with-kernel=cubic-spline`

Several kernels are made available for use with the hydrodynamical schemes. Choose between them with this compile-time flag.

`--with-hydro-dimension=3`

Run problems in 1, 2, and 3 (default) dimensions.

`--with-equation-of-state=ideal-gas`

Several equations of state are made available with this flag.

`--with-cooling=none`

Several cooling implementations (including GRACKLE) are available.

`--with-ext-potential=none`

Many external potentials are available for use with SWIFT.

Runtime Options and Parameter Files

SWIFT requires a number of runtime options to run and get any sensible output. For instance, just running the `swift` binary will not use any SPH or gravity; the particles will just sit still!

A list of command line options can be found by running the compiled binary with the `-h` or `--help` flag:

```
./swift --help
```

You will also need to specify a number of runtime parameters that are dependent on your compile-time configuration in a parameter file. A list of all of these parameters can be found in `examples/parameter_example.yml`, and you can check out examples in the `examples/` directory.

Running an Example

SWIFT provides a number of examples that you can run in the `examples/` directory. Many are detailed in their respective `README` files, and contain python scripts (files with the suffix `.py`) to both generate initial conditions and plot results. The python scripts usually contain their respective documentation at the top of the script file itself.

Sod Shock

In this example, we will run the 3D SodShock test. You will need to configure and compile the code as follows:

```
./configure
make
```

Then to run the code, we first download and build the initial conditions:

```
cd examples/HydroTests/SodShock_3D
./getGlass.sh
python3 makeIC.py
../../../swift --hydro --threads=4 sodShock.yml
```

We can plot the solution with the included python script as follows:

```
python3 plotSolution.py 1
```

The argument `1` tells the python plotting script to use the snapshot with number 1 for the plot.

Small Cosmological Volume

As a second example, we run a small cosmolgical volume containing dark matter only starting at redshift $z = 50$. Like for the Sod Shock example, it suffices to configure (`./configure`) and compile (`make`) the code without any extra flags.

After downloading the initial conditions, we run the code with cosmology and self-gravity:

```
cd examples/SmallCosmoVolume/SmallCosmoVolume_DM
./getIC.sh
../../../swift --cosmology --self-gravity \
    --threads=8 small_cosmo_volume_dm.yml
```

We can plot the solution with the included python script as follows:

```
python3 plotProjection.py 31
```

The `plotProjection.py` script requires the swiftsimio library.

An example containing both baryonic and dark matter is `examples/SmallCosmoVolume/SmallCosmoVolume_hydro`. To run with hydrodynamics, the `--hydro` flag needs to be provided as well:

```
../../../swift --cosmology --self-gravity \
    --hydro --threads=8 small_cosmo_volume.yml
```

Submission Script

Below is an example submission script for the SLURM batch system. This runs SWIFT with MPI, thread pinning, hydrodynamics, and self-gravity.

```
#SBATCH --partition=<queue>
#SBATCH --account-name=<groupName>
#SBATCH --job-name=<jobName>
#SBATCH --nodes=<nNodes>
#SBATCH --ntasks-per-node=<nMPIRank>
#SBATCH --cpus-per-task=<nThreadsPerMPIRank>
#SBATCH --time=<hh>:<mm>:<ss>

srun -n $SLURM_NTASKS ./swift_mpi \
    --threads=$SLURM_CPUS_PER_TASK --pin \
    --hydro --self-gravity parameter_file.yml
```