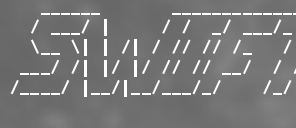
Getting Started 

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, or on our GitHub by creating an issue.

```
https://gitter.im/swiftsim
```

```
https://github.com/swiftsim/swiftsim
```

Initial Setup

We use autotools for setup. To get a basic running version of the code (the binary is created in swiftsim/examples) on most platforms, run

```
./autogen.sh
```

```
./configure
```

```
make
```

MacOS Specific Oddities

To build on MacOS you will need to disable compiler warnings due to an incomplete implementation of pthread barriers. To configure, just disable the compiler warnings:

```
./configure --disable-compiler-warnings
```

Runtime Options

A list of runtime options (*which you will need to run*) is available one compiled from the binary through

```
./swift -h
```

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
```

Dependencies

HDF5

Version 1.8.x or higher is required. Input and output files are stored as HDF5 and are compatible with the existing GADGET-2 specification. Please consider using a build of parallel-HDF5, as SWIFT can leverage this when writing and reading snapshots.

MPI

A recent implementation of MPI, such as Open MPI (v2.x or higher), is required.

Libtool

The build system depends on libtool.

FFTW

Version 3.3.x or higher is required.

METIS

METIS is used for domain decomposition and load balancing.

libNUMA

libNUMA is used to pin threads

GSL

The GSL is required for cosmological integration.

Optional Dependencies

TCmalloc/Jemalloc

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

DOXYGEN

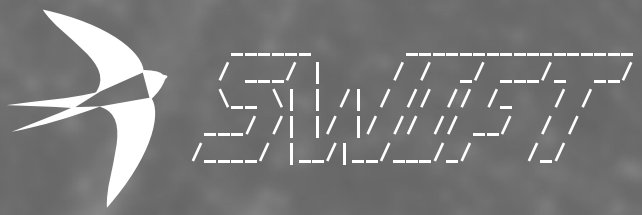
You can build documentation for SWIFT with DOXYGEN.

Python

To run the examples, you will need python and some of the standard scientific libraries (numpy, matplotlib). Some examples use Python 2 scripts, but the more recent ones use Python 3 (this is specified in individual READMEs).

GRACKLE

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



Useful Configuration Flags

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

`--with-hydro=gadget2`
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. Also consider `--with-adiabatic-index`.

`--with-cooling=none`
Several cooling implementations (including GRACKLE) are available.

`--with-ext-potential=none`
Many external potentials are available for use with SWIFT. You can choose between them at compile time. Some examples include a central potential, a softened central potential, and a sinusoidal potential. You will need to configure, for example, the mass in your parameter file at runtime.

Running the Code

After compilation, you will be left with *two* binaries. One is called `swift`, and the other `swift_mpi`. Current wisdom is to run `swift` if you are only using one NUMA region (e.g. a single chip), and one MPI rank per NUMA region using `swift_mpi` for anything larger. You will need some `GADGET-2` HDF5 initial conditions to run SWIFT, as well as a compatible yaml parameter file.

Running an Example

SWIFT provides a number of examples that you can run in the examples directory. In this example, we will run the 3D SodShock test, with GIZMO hydrodynamics.

You will need to configure the code as follows:

```
./configure \
--with-hydro=gizmo-mfm \
--with-riemann-solver=hllc \
--disable-vec
make
```

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

```
cd examples/SodShock_3D
./getGlass.sh
python makeIC.py
../swift -s -t 4 sodShock.yml
```

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

```
python plotSolution.py 1
```

Submission Script

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

```
#SBATCH -J <jobName>
#SBATCH -n <nCoresTotal>
#SBATCH -o outFile.out
#SBATCH -o errFile.err
#SBATCH -p <queue>
#SBATCH -A <groupName>
#SBATCH --exclusive
## Expected Runtime
#SBATCH -t <hh>:<mm>:<ss>
./swift -asG -t <nThreadsPerMPIRank>
```

The development of SWIFT has been made possible due to funding from Intel to support the Intel Parallel Computing Centre at Durham. Additional physics modules for SWIFT are being developed with funding from DIRAC and STFC.