psyclone Documentation

Release 1.0.0

Rupert Ford

January 14, 2015

CONTENTS

PSyclone, the PSy code generator, is being developed for use in finite element, finite volume and finite difference codes. PSyclone is being developed to support the emerging API in the GungHo project for a finite element dynamical core.

The GungHo project is designing and building the heart of the Met Office's next generation software (known as the dynamical core) using algorithms that will scale to millions of cores. The project is a collaboration between the Met Office, NERC (via NERC funded academics) and STFC, and the resultant software is expected to be operational in 2022.

The associated GungHo software infrastructure is being developed to support multiple meshes and element types thus allowing for future model development. GungHo is also proposing a novel separation of concerns for the software implementation of the dynamical core. This approach distinguishes between three layers: the Algorithm layer, the Kernel layer and the Parallelisation System (PSy) layer. Together this separation is termed PSyKAl.

The Algorithm layer specifies the algorithm that the scientist would like to run (in terms of calls to kernel and infrastructure routines) and logically operates on full fields.

The Kernel layer provides the implementation of the code kernels as subroutines. These subroutines operate on local fields (a set of elements, a vertical column, or a set of vertical columns, depending on the kernel).

The PSy layer sits in-between the algorithm and kernel layers and its primary role is to provide node-based parallel performance for the target architecture. The PSy layer can be optimised for a particular hardware architecture, such as multi-core, many-core, GPGPUs, or some combination thereof with no change to the algorithm or kernel layer code. This approach therefore offers the potential for portable performance.

Rather than writing the PSy layer manually, the GungHo project is developing the PSyclone code generation system which can help a user to optimise the code for a particular architecture (by providing optimisations such as blocking, loop merging, inlining etc), or alternatively, generate the PSy layer automatically.

PSyclone is also being extended to support an API being developed in the GOcean project for two finite difference ocean model benchmarks, one of which is based on the NEMO ocean model.

Contents:

CHAPTER

ONE

GETTING GOING

1.1 Download

PSyclone is available for download from the GungHo repository.

```
svn co https://puma.nerc.ac.uk/svn/GungHo_svn/PSyclone/trunk PSyclone
```

Hereon the location where you download PSyclone (including the PSyclone directory itself) will be refered to as <PSYCLONEHOME>

1.2 Dependencies

PSyclone is written in python so needs python to be installed on the target machine. PSyclone has been tested under python 2.6.5 and 2.7.3.

PSyclone immediately relies on two external libraries, f2py and pyparsing.

1.2.1 f2py quick setup

The source code of f2py (revision 88) is provided with PSyclone in the sub-directory `f2py_88`.

To use f2py provided with PSyclone you can simply set up your PYTHONPATH variable to include this directory.

```
> export PYTHONPATH=<PSYCLONEHOME>/f2py_88:${PYTHONPATH}
```

You can now skip the f2py installation section.

1.2.2 f2py installation

PSyclone requires version 3 of f2py, a library designed to allow fortran to be called from python (see http://code.google.com/p/f2py/wiki/F2PYDevelopment for more information). PSyclone makes use of the fortran parser (fparser) contained within.

The source code of f2py (revision 88) is provided with PSyclone in the sub-directory `f2py_88`. If you would prefer to install f2py rather than simply use it as is (see the previous section) then the rest of this section explains how to do this.

f2py uses the numpy distutils package to install. In version 1.6.1 of distutils (currently the default in Ubuntu) distutils supports interactive setup. In this case to install f2py using gfortran and gcc (for example) you can perform the following (where cgcc, fgfortran, 1 and 2 are interactive commands to setup.py)

```
> cd f2py_88
> sudo ./setup.py
cgcc
fgfortran
1
> sudo ./setup.py
cgcc
fgfortran
2
```

For later versions of distutils (1.8.0 has been tested) where the interactive setup has been disabled you can perform the following (using g95 and gcc in this case)

```
> cd f2py_88
> sudo ./setup.py build -fcompiler=g95 -ccompiler=gcc
> sudo ./setup.py install
```

For more information about possible build options you can use the available help

```
> ./setup.py --help
> ./setup.py build --help
> ./setup.py build --help-fcompiler
```

In particular, if you do not have root access then the python modules can be installed in your user account by specifying –user to the install command

```
> ./setup.py install --user
```

This causes the software to be installed under ${HOME}/.local/

1.2.3 pyparsing

PSyclone requires pyparsing, a library designed to allow parsers to be be built in Python. PSyclone uses pyparsing to parse fortran regular expressions as f2py does not fully parse these, (see http://pyparsing.wikispaces.com for more information).

PSyclone has been tested with pyparsing version 1.5.2 which is a relatively old version but is currently the version available in the Ubuntu software center.

You can test if pyparsing is already installed on your machine by typing `import pyparsing` from the python command line. If pyparsing is installed, this command will complete succesfully. If pyparsing is installed you can check its version by typing `pyparsing.__version__` after succesfully importing it. Versions higher than 1.5.2 should work but have not been tested.

If pyparsing is not installed on your system you can install it from within Ubuntu using the software center (search for the "python-pyparsing" module in the software center and install). If you do not run Ubuntu you could follow the instructions here http://pyparsing.wikispaces.com/Download+and+Installation.

1.3 Run

The generator.py script can be used to generate the required PSy code as well as the modified algorithm code.

```
> cd <PSYCLONEHOME>/src
> python ./generator.py
usage: generator.py [-h] [-oalg OALG] [-opsy OPSY]  [-api API] filename
generator.py: error: too few arguments
```

Examples are provided in the example directory. There are 3 subdirectories in the examples directory corresponding to different API's that are supported by PSyclone. In this case we are going to use one of the dynamo examples

```
> cd <PSYCLONEHOME>/example/dynamo/eg1
> python ../../../src/generator.py -oalg dynamo_alg.f90 -opsy dynamo_psy.f90 dynamo.F90
```

You should see two new files created called dynamo_alg.f90 and dynamo_psy.f90

You can also run the runme.py example to see the interactive API in action

```
> cd <PSYCLONEHOME>/example/dynamo/eg1
> python runme.py
```

CHAPTER

TWO

API

generator.py

-oalg `<filename>`

-opsy `<filename>`

`<filename>`

Command line version of the generator. If -oalg or -opsy or not provided then the generated code is printed to stdout. Uses the `generator.generate()` function to generate the code. Please see the run documentation for more details.

For example:

```
> python generator.py algspec.f90
```

`generator.`**`generate`**(*filename*, *api=''*)

Takes a GungHo algorithm specification as input and outputs the associated generated algorithm and psy codes suitable for compiling with the specified kernel(s) and GungHo infrastructure. Uses the `parse.parse()` function to parse the algorithm specification, the `psyGen.PSy` class to generate the PSy code and the `algGen.Alg` class to generate the modified algorithm code.

Parameters **filename** (*str*) – The file containing the algorithm specification.

Returns The algorithm code and the psy code.

Return type ast

Raises IOError if the filename does not exist

For example:

```
>>> from generator import generate
>>> psy,alg=generate("algspec.f90")
```

`parse.`**`parse`**(*filename*, *api=''*, *invoke_name='invoke'*, *inf_name='inf'*)

Takes a GungHo algorithm specification as input and outputs an AST of this specification and an object containing information about the invocation calls in the algorithm specification and any associated kernel implementations.

Parameters

- **filename** (*str*) – The file containing the algorithm specification.
- **invoke_name** (*str*) – The expected name of the invocation calls in the algorithm specification
- **inf_name** (*str*) – The expected module name of any required infrastructure routines.

Return type ast,invoke_info

Raises

- **IOError** – if the filename does not exist
- **ParseError** – if there is an error in the parsing
- **RuntimeError** – if there is an error in the parsing

For example:

```
>>> from parse import parse
>>> ast,info=parse("argspec.F90")
```

class `algGen.`**`Alg`**(*ast*, *psy*)

Generate a modified algorithm code for a single algorithm specification. Takes the ast of the algorithm specification output from the function `parse.parse()` and an instance of the `psyGen.PSy` class as input.

Parameters

- **ast** (*ast*) – An object containing an ast of the algorithm specification which was produced by the function `parse.parse()`.
- **psy** (*PSy*) – An object (`psyGen.PSy`) containing information about the PSy layer.

For example:

```
>>> from parse import parse
>>> ast,info=parse("argspec.F90")
>>> from psyGen import PSy
>>> psy=PSy(info)
>>> from algGen import Alg
>>> alg=Alg(ast,psy)
>>> print(alg.gen)
```

`gen`

Generate modified algorithm code

Return type ast

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX