Bones and A-Darwin

This README [p. 1] covers both the source-to-source compiler Bones [p. 33] and the species extraction tool A-Darwin. Please refer to the corresponding sections for documentation.

Bones

Introduction

Recent advances in multi-core and many-core processors requires programmers to exploit an increasing amount of parallelism from their applications. Data parallel languages such as CUDA and OpenCL make it possible to take advantage of such processors, but still require a large amount of effort from programmers. To address the challenge of parallel programming, we introduce Bones [p. 33].

Bones [p. 33] is a source-to-source compiler based on algorithmic skeletons and a new algorithm classification (named 'algorithmic species'). The compiler takes C-code annotated with class information as input and generates parallelized target code. Targets include NVIDIA GPUs (through CUDA), AMD GPUs (through OpenCL) and CPUs (through OpenCL and OpenMP). Bones [p. 33] is open-source, written in the Ruby programming language, and is available through our website. The compiler is based on the C-parser CAST, which is used to parse the input code into an abstract syntax tree (AST) and to generate the target code from a transformed AST.

Usage

The usage is as follows:

```
bones --application <input> --target <target> [OPTIONS]
```

With the following flags:

```
--application, -a <s>:   Input application file
--target, -t <s>:   Target processor (choose from: CPU-C, CPU-OPENCL-AMD,
CPU-OPENCL-INTEL, CPU-OPENMP,GPU-CUDA, GPU-OPENCL-AMD)
--measurements, -m:   Enable/disable timers
--verify, -c:   Verify correctness of the generated code
--only-alg-number, -o <i>:   Only generate code for the x-th species (99 -> all)
--merge-factor, -f <i>:   Thread merge factor, default is 1 (==disabled)
--register-caching, -r <i>:   Enable register caching: 1:enabled (default),
0:disabled
--zero-copy, -z <i>:   Enable OpenCL zero-copy: 1:enabled (default), 0:disabled
--skeletons, -s <i>:   Enable non-default skeletons: 1:enabled (default),
0:disabled
--version, -v:   Print version and exit
--help, -h:   Show this message
```

Bones [p. 33] can be invoked from the command-line. Two arguments (-a and -t) are mandatory, others are optional. This is an example of the usage of Bones [p. 33] assuming the file '`example.c`' to be present:

```
bones -a example.c -t GPU-CUDA -c
```

Examples

The best place to start experimenting with Bones [p. 33] is the '`examples`' directory. A large number of examples are available in this folder, grouped by algorithmic species (either element, neighbourhood, shared or chunk). The examples illustrate different kinds of coding styles and give a large number of different classes to work with. The folder '`benchmarks`' gives more examples, taken from the PolyBench/C benchmark set. Additionally, a folder '`applications`' is included, containing example complete applications.

All examples can be run through Bones [p. 33] for a specific target using an automated Rake task. Executing '`rake examples:generate`' or simply '`rake`' will execute Bones [p. 33] for all examples for a given target. The target can be changed in the '`Rakefile`' found in the root directory of Bones [p. 33].

Limitations

Bones [p. 33] takes C99 source code as input. However, several coding styles are unsupported as of now or might yield worse performance compared to others. The numerous examples provided should give the user an idea of the possibilities and limitations of the tool. A complete list of coding guidelines and limitations will follow in the future. Currently, an initial list of major limitations and guidelines is given below. In this list, we use '`algorithm`' to denote an algorithm captured by an algorithmic species.

- If the algorithm works on a N-dimensional data structure, use N-dimensional arrays (don't flatten it yourself, e.g. use '`example[i][j]`' instead of '`example[i+j*A]`') and specify an N-dimensional algorithmic species.
- Write your while-loops as for-loops if possible. For-loops should have a unit increment, other loops (e.g. decrementing loops) must be re-written.
- Loops can have affine bounds containing constants, defines and variables. Variables should not include loop variables of loops that are part of the '`algorithm`'.
- Function calls are not allowed within the '`algorithm`'. Some mathematical functions are allowed.
- Variables are allowed in the definition of an algorithmic species. If they are used, the should also be used somewhere in the body of the '`algorithm`'.
- Bones [p. 33] is designed to work on a single input file with at least a function called 'main'. If your (to-be-accelerated) code spawns over multiple C-files, Bones [p. 33] could either be applied multiple times, or the code could be merged into a single file.

A-Darwin

Introduction

The original algorithmic species theory included ASET, a polyhedral based algorithmic species extraction tool. Along with a new non-polyhedral theory, we present a new automatic extraction tool named A-Darwin (short for `automatic Darwin').

The new tool is largely equal to ASET in terms of functionality, but is different internally. The tool is based on CAST, a C99 parser which allows analysis on an abstract syntax tree (AST). From the AST, the tool extracts the array references and constructs a 5 or 6-tuple for each loop nest. Following, merging is applied and the species are extracted. Finally, the species are inserted as pragma's in the original source code. To perform the dependence tests in A-Darwin, we make use of a combination of the GCD and Banerjee tests.

Together, these tests are conservative, i.e. we might not find all species.

Usage

The usage is as follows:
```
adarwin --application <input> [OPTIONS]
```

With the following flags:
```
--application, -a <s>:   Input application file
--no-memory-annotations, -m:   Disable the printing of memory annotations
--mem-remove-spurious, -r:   Memcopy optimisation: remove spurious copies
--mem-copyin-to-front, -f:   Memcopy optimisation: move copyins to front
--mem-copyout-to-back, -b:   Memcopy optimisation: move copyouts to back
--mem-to-outer-loop, -l:   Memcopy optimisation: move copies to outer loops
--fusion, -k <i>:   Type of kernel fusion to perform (0 -> disable)
--print-arc, -c:   Print array reference characterisations (ARC) instead of
species
--silent, -s:   Become silent (no message printing)
--only-alg-number, -o <i>:   Only generate code for the x-th species (99 -> all)
--version, -v:   Print version and exit
--help, -h:   Show this message
```

A-Darwin can be invoked from the command-line. One arguments (-a) is mandatory, others are optional. This is an example of the usage of A-Darwin assuming the file '`example.c`' to be present:
```
adarwin -a example.c -m -s
```

For now, it is recommended to use the '-m' flag. The memory optimisation flags ('-rfbl') are not fully tested yet. For a more fine-grained classification, A-Darwin is able to print the internal array reference characterisations (ARC) instead (use the '-c' flag).

Known limitations

- The dependence test is not reliable yet
- Code similar to the failing examples are not supported yet
- Multi-line comments with pre-processor directives inside will not be considered commented out.
- Custom defined types are not supported. Apart from the default C99 types, FILE and size_t are supported.

Installation procedure

Installation of Bones [p. 33] and A-Darin is a simple matter of extracting the Bones/A-Darwin package to a directory of your choice. Bones [p. 33] can also be installed as a gem ('`gem install bones-compiler`'). However, there are a number of prerequisites before doing this.

Prerequisites

Bones/A-Darwin requires the installation of Ruby, the Rubygems gem package manager and several gems:

1. Any version of **Ruby 1.8** or **1.9**. Information on Ruby is found at `http://www.ruby-lang.org`
   - [OS X]: Ruby is pre-installed on any OS X system since Tiger (10.4).
   -

[Linux]: Ruby is pre-installed on some Linux based systems. Most Linux package managers (yum, apt-get) will be able to provide a Ruby installation. Make sure that the ruby development package ('`ruby-devel`') is also installed, as it is required by one of the gems.
   - [Windows]: Ruby for Windows can be obtained from `http://rubyinstaller.org`
2. The **Rubygems** gem package manager. Information on Rubygems can be found at `http://rubygems.org`
   - [OS X]: Rubygems is pre-installed on any OS X system since Tiger (10.4).
   - [Linux]: Most Linux package managers will be able to provide a Rubygems installation by installing the package '`rubygems`'.
   - [Windows]: Rubygems for Windows is obtained automatically when installing from `http://rubyinstaller.org`
3. Bones/A-Darwin require the gems, **trollop**, **cast**, and **symbolic**. These gems can be installed by calling Rubygems from the command line, i.e.: '`gem install trollop cast symbolic`'.

For example, all prerequisites can be installed as follows on a Fedora, Red-Hat or CentOS system:
```
yum install ruby ruby-devel rubygems
gem install trollop cast symbolic
```

For an Ubuntu, Debian or Mint system, the equivalent commands are:
```
apt-get install ruby ruby-devel rubygems
gem install trollop cast symbolic
```

Installing Bones/A-Darwin manually

To install the tools manually, simply extract the '`bones_x.x.tar.gz`' or '`adarwin_x.x.tar.gz`' package into a directory of your choice. The Bones/A-Darwin executables are found in the '`bin`' subdirectory. Including the path to the '`bin`' directory to your environmental variable '`PATH`' will make Bones/A-Darwin available from any directory on your machine. Starting at version 1.1, Bones [p. 33] and A-Darwin are also available as a gem ('`gem install bones-compiler`').

Documentation

There are two ways to go to obtain more information regarding Bones/A-Darwin. To obtain more information about the tools themselves, the ideas behind it and the algorithm classification, it is a good idea to read scientific publications. To get more information about the code structure, HTML documentation can be generated automatically using RDoc.

Code documentation

Code documentation can be generated automatically using RDoc. Navigate to the installation root of Bones/A-Darwin and use Rake to generate documentation: '`rake rdoc`'. More information on using Rake is provided later in this document. Next, open '`rdoc/index.html`' to navigate through the documentation. The same documentation is also available on the web at `http://parse.ele.tue.nl/tools/bones/rdoc/index`.

Scientific publications

Scientific publications related to Bones/A-Darwin can be obtained from

http://www.cedricnugteren.nl/publications.php. Several publications are relevant:

1. **Bones: An Automatic Skeleton-Based C-to-CUDA Compiler for GPUs**, which provides details on the Bones [p. 33] source-to-source compiler, including optimizations in host-accelerator transfer and loop fusion in kernel code. When referring to GPU code generation using Bones [p. 33], loop fusion or optimizations in host-accelerator transfer in scientific work, you are kindly asked to include the following citation:

```
@INPROCEEDINGS{Nugteren2015a,
author = {Cedric Nugteren and and Henk Corporaal},
title = {Bones: An Automatic Skeleton-Based C-to-CUDA Compiler for GPUs},
journal = {ACM Trans. Archit. Code Optim.},
volume = {11},
number = {4},
year = {2015},
}
```

2. **Algorithmic Species Revisited: A Program Code Classification Based on Array References**, which provides details on the algorithm classification (the species) and A-Darwin (the tool). When referring to the algorithm classification in scientific work, you are kindly asked to include the following citation:

```
@INPROCEEDINGS{Nugteren2013a,
author = {Cedric Nugteren and Rosilde Corvino and Henk Corporaal},
title = {Algorithmic Species Revisited: A Program Code Classification Based
on Array References},
booktitle = {MuCoCoS '13: International Workshop on Multi-/Many-core
Computing Systems},
year = {2013},
}
```

3. **Automatic Skeleton-Based Compilation through Integration with an Algorithm Classification**, which discusses the Bones [p. 33] source-to-source compiler. When referring to Bones [p. 33] in scientific work, you are kindly asked to include the following citation:

```
@INPROCEEDINGS{Nugteren2013b,
author    = {Cedric Nugteren and Pieter Custers and Henk Corporaal},
title     = {Automatic Skeleton-Based Compilation through Integration with
an Algorithm Classification},
booktitle = {APPT '13: Advanced Parallel Processing Technology},
year      = {2013},
}
```

Rake

Rake is Ruby's make and can be used to automate tasks. By invoking '`rake -T`', a list of commands will become available. For example, for A-Darwin, the following rake commands are available:

```
rake adarwin[file]  # Extract species descriptions using A-Darwin
rake adarwin_test   # Test A-Darwin`s output against golden samples
rake clean          # Remove any temporary products.
rake clobber        # Remove any generated file.
rake clobber_rdoc   # Remove RDoc HTML files
rake rdoc           # Build RDoc HTML files
```

```
rake rerdoc          # Rebuild RDoc HTML files
```

With rake, A-Darwin can be tested on a set of examples '`rake adarwin_test`'. Pre-created golden samples are available in the '`test`' folder.

Questions

Questions can be directed by email. You can find contact details on the personal page of the author at `http://www.cedricnugteren.nl` or on the project page at GitHub.

Module

Adarwin

#RDoc::Comment:0x000000033c5bb0>
#RDoc::Comment:0x000000036f6190>
#RDoc::Comment:0x000000049ca690>
#RDoc::Comment:0x000000044214a0>
#RDoc::Comment:0x0000000362d740>
#RDoc::Comment:0x000000040c4f50>
#RDoc::Comment:0x00000002e0dd30>

Constants

Name	Value	Description
MESSAGE	'[A-Darwin] ### Info …	A string given as a start of an informative message.
WARNING	'[A-Darwin] ### Warni…	A string given as a start of an warning message.
ERROR	'[A-Darwin] ### Error…	A string given as a start of an error message.
SCOP_START	'#pragma scop'	Start of the scop
SCOP_END	'#pragma endscop'	Enf of the scop
PRAGMA_SPECIES	'#pragma species'	Species pragma
PRAGMA_ARC	'#pragma ARC'	Array reference characterisation (ARC) pragma
PRAGMA_DELIMITER_START	'"PRAGMA '	Create a string from a pragma because pragma's are unsupported by CAST.
PRAGMA_DELIMITER_END	' PRAGMA"'	

Class

Adarwin::Common

This class is created to be a parent class of all classes.

Class

Adarwin::Dependence

This class represents the dependence tests. The dependence tests are not objects as such, the use of a class might therefore be a bit out of place. Instead, the class rather holds all methods related to dependence tests.

For an M-dimensional access, the problem of dependence testing is reduced to that of determining whether a system of M linear equations of the form >>> a_1*I_1 + a_2*I_2 + ... + a_n*I_n = a_0 has a simultaneous integer solution satisfying the loop/if bounds given as >>> min_k = I_k = max_k

Currently, the following conservative tests are implemented:

- The GCD (greatest common divisor) test
- The Banerjee test

In case the accesses are multi-dimensional, we perform a subscript-by- subscript checking. In other words, we test each dimension separately using the two tests. If we find a possible dependence in one dimension, we conclude that there is a dependence.

Attributes

result [RW]

Public Class methods

new new(access1,access2,verbose)

Method to initialise the dependence tests. This method actually already computes all the dependence tests and stores the result in a class variable. It takes as input the pair of accesses it needs to check for dependences.

Public Instance methods

ban_test ban_test(all_vars,equation,conditions)

This method implements the Banerjee test. This test takes loop bounds into consideration. The test is based on a linear equation in the form of >>> a_1*I_1 + a_2*I_2 + ... + a_n*I_n = a_0 and loop bounds in the form of >>> min_k = I_k = max_k

The test proceeds as follows. First, the values a_k+ and a_k- are computed. Also, the bounds min_k and max_k are calculated from the loop conditions. Following, the test computes the extreme values 'low' and 'high'. Finally, the test computes whether the following holds: >>> low = a_0 = high If this holds, there might be a dependence (method returns true). If this does not hold, there is definitely no dependence (method returns false).

gcd gcd(args)

Implementation of a GCD method with any number of arguments. Relies on Ruby's default GCD method. In contrast to the normal gcd method, this method does not act on a number, but instead takes an array of numbers as an input.

gcd_test gcd_test(all_vars,equation)

This method implements the GCD test. The test is based on the computation of the greatest common divisor, giving it its name. The GCD test is based on the fact that a linear equation in the form of >>> a_1*I_1 + a_2*I_2 + ... + a_n*I_n = a_0 has an integer solution if and only if the greatest common divisor of a_1, a_2,...,a_n is a divisor of a_0. The GCD test checks for this divisability by performing the division and checking if the result is integer.

This method returns true if there is an integer solution, not necessarily within the loop bounds. Thus, if the method returns true, there might be a dependence. If the method returns false, there is definitely no dependence.

TODO: If the result (`division`) is symbolic, can we conclude anything?

---

get_linear_equation get_linear_equation(access1,access2,bounds,all_loop_vars)

This method retrieves a linear equation from a pair of access. Accesses are transformed into a linear equation of the form >>> a_1*I_1 + a_2*I_2 + ... + a_n*I_n = a_0 Additionally, this method returns a list of all variables and a list of loop bounds corresponding to the linear equation's variables.

---

get_loop_vars get_loop_vars(vars,all_loop_vars)

Method to obtain all variables in an array reference that are also loop variables.

Class

Adarwin::Engine

This is the main 'engine' for the A-darwin algorithmic species extraction tool. It contains methods to parse the command-line arguments, to run the pre-processor, to insert the annotations, and to pretty print the final output. TODO: Add a syntax check by a normal compiler first (e.g. gcc)

Public Class methods

new new()

Initializes the engine and processes the command line arguments. This method uses the 'trollop' gem to parse the arguments and to create a nicely formatted help menu. This method additionally initializes a result- hash and reads the contents of the source file from disk.

Command-line usage:

```
adarwin --application <input> [OPTIONS]
```

Options:

```
--application, -a <s>:   Input application file
--no-memory-annotations, -m:   Disable the printing of memory annotations
--mem-remove-spurious, -s:   Memcopy optimisation: remove spurious copies
--mem-copyin-to-front, -f:   Memcopy optimisation: move copyins to front
--mem-copyout-to-back, -b:   Memcopy optimisation: move copyouts to back
--mem-to-outer-loop, -l:   Memcopy optimisation: move copies to outer
loops
--only-alg-number, -o <i>:   Only generate code for the x-th species (99
-> all) (default: 99)
--version, -v:   Print version and exit
--help, -h:   Show this message
```

Public Instance methods

get_children get_children(parent)

Method to obtain the children of a nest

insert_copies insert_copies(scop_ast)

Iterate over the loop nests and insert the memory copy annotations into the original AST.

insert_species insert_species(scop_ast)

This method iterates over the loop nests and inserts the species into the original AST. It also inserts the synchronisation barries when needed, and only if the user is interested in the memory copy annotations.

populate_nests populate_nests(ast,level=[])

This method populates the Nest [p. 27] datastructure (recursively). It is the main method to

process the loop nests and fine the species information. It is called recursively.

process process()

Method to process a file and to output target code. This method calls all the other methods, it is the main engine.

Tasks:

- Run the preprocessor to obtain algorithm information.
- Use the 'CAST' gem to parse the source into an AST.
- Call the code generator to perform the real work and produce output.

process_scop process_scop(scop_code)

remove_inner_species remove_inner_species(nests)

This method removes all species in the current loop nest (called recursively). It assumes these species should be removed.

write_output write_output()

This method writes the output code to a file.

Class

Adarwin::Interval

This class represents an interval [a..b] including a and b. The class has the following methods:

- Initialise the interval (`initialize`)
- Print the interval (`to_s`)
- Merge an interval with another interval (`merge`)
- Return the length of the interval (`length`)

Attributes

a [RW]
b [RW]

Public Class methods

new new(a,b,loops)

Initialise the interval. This method performs a comparison to see whether a or b is the upper-bound. This comparison is based on guesses made by the `compare` method. This method uses loop information if needed. FIXME: Uses the `compare` method which might be based on a guess

Public Instance methods

length length()

Method to compute the length of the interval. For example, the length of

is equal to (b-a+1).

merge merge(other_interval)

Merge this interval with another interval. This is based on a comparison made by the `compare` method, which is an approximation based on loop information. FIXME: Uses the `compare` method which might be based on a guess

to_s to_s()

Print the interval as a string (e.g. [4..9]).

Class

Adarwin::Nest

This class represents a loop nest. The end goal is to annotate the loop nest with the corresponding species information. If the loop nest cannot be parallelised (if there are dependences), the species information is not printed.

This class contains methods to perform among others the following:

- Find all array references in the loop nest
- Merge found array references into another array reference
- Translate array references into species
- Perform dependence tests to check for parallelism

Attributes

code [RW]
copyins [RW]
copyouts [RW]
depth [RW]
fused [RW]
id [RW]
level [RW]
name [RW]
outer_loops [RW]
reads [RW]
removed [RW]
species [RW]
verbose [RW]
writes [RW]

Public Class methods

new new(level, code, id, name, verbose, fused=0)

Method to initialise the loop nest. The loop nest is initialised with the following variables:

- An identifier for the order/depth in which the nest appears (`level`)
- The loop nest body in AST form (`code`)
- A unique identifier for this loop nest (`id`)
- A human readable name for this loop nest (`name`)
- Whether or not verbose information should be printed (`verbose`)

Public Instance methods

has_copyins? has_copyins?()

Method to check if the loop nest has copyins.

has_copyouts? has_copyouts?()

Method to check if the loop nest has copyouts.

has_dependences?

Perform the dependence test for the current loop nest. This method gathers all pairs of array references to test and calls the actual dependence tests. Currently, the dependence tests are a combination of the GCD test and the Banerjee test.

has_species? has_species?()

Perform a check to see if the loop nest has species that are not just formed from shared or full patterns. If so, there is no parallelism.

merge_references merge_references()

Perform the algorithm to merge array reference characterisations into merged array references. This method is a copy of the merging algorithm as found in the scientific paper. TODO: Complete this algorithm to match the scientific paper version.

print_arc_end print_arc_end()

Method to print the end of an array reference characterisation (ARC).

print_arc_start print_arc_start()

Method to print the start of an array reference characterisation (ARC).

print_copyins print_copyins()

Method to print the copyin pragma.

print_copyouts print_copyouts()

Method to print the copyout pragma.

print_species_end print_species_end()

Method to print the end pragma of a species.

print_species_start print_species_start()

Method to print the start pragma of a species.

translate_into_arc translate_into_arc()

Method to translate the array reference characterisations into a string.

translate_into_species translate_into_species()

Method to translate the array reference characterisations into species. The actual logic is performed within the Reference [p. 30] class. In this method, only the combining of the separate parts is performed.

Class

Adarwin::Preprocessor

This is the C99 pre-processor for Adarwin [p. 15]. It has the following tasks:

- Extract the SCoP part from the code (the region of interest)
- Extract the header code (defines, includes, etc.)
- Output the original code without pre-processor directives
- Output the original code minus the SCoP (SCoP to be filled in later)

Constants

Name	Value	Description
`WHITESPACE`	`'\s*'`	Regular expression to identify whitespaces (tabs, spaces).

Attributes

header_code [R]
parsed_code [R]
scop_code [R]
source_code [R]
target_code [R]

Public Class methods

new new(source_code)

This is the method which initializes the preprocessor. Initialization requires the target source code to process, which is then set as the class variable +@source_code+.

Public Instance methods

process process()

This is the method to perform the actual preprocessing. This method takes care of all the pre-processor tasks. The output is stored in the two attributes `header_code`, and `scop`. FIXME: What about multi-line statements? For example, a multi-line comment could have a commented-out SCoP or define or include.

Class

Adarwin::Reference

This class represents an array reference characterisation. This reference is constructed as a 5-tuple (tN,tA,tD,tE,tS) with the following information:

- tN: The name of the reference.
- tA: The access direction (read or write).
- tD: The full domain accessed.
- tE: The number of elements accessed each iteration (the size).
- tS: The step of a accesses among iterations.

To be able to compute the 5-tuple, the reference also stores information about the loops and conditional statements to which the original array reference is subjected.

This class contains methods to perform among others the following:

- Initialise the class and sets the 5-tuple (N,A,D,E,S)
- Retrieve information on array indices
- Print in different forms (species, ARC, copy/sync pragma's)

Attributes

all_loops [RW]
bounds [RW]
id [RW]
indices [RW]
pattern [RW]
tA [RW]
tD [RW]
tE [RW]
tN [RW]
tS [RW]

Public Class methods

new new(reference,id,inner_loops,outer_loops,var_declarations,verbose)

This method initialises the array reference class. It takes details of the reference itself and details of the loop nest it belongs to. The method performs among others the following:

- It initialises the 5-tuple (N,A,D,E,S)
- It constructs the sets of loops (all,inner,outer) for this reference
- It computes the bounds based on loop data and on if-statements
- It computes the domain (D), number of elements (E), and step (S)

Public Instance methods

array_includes_local_vars array_includes_local_vars(array, loop_vars)

Method to find if local variables are included

dependent? dependent?(index,loops)

Method to check whether the an index is dependent on a given set of loops. For example, A is independent of j, but dependent on i.

depends_on? depends_on?(var)

Method to find out if the reference is dependent on a variable. It is used by the copy optimisations.

find_extreme find_extreme(position,index,loops)

Substitute loop data with the upper-bound or lower-bound of a loop to find the minimum/maximum of an array reference. The body is executed twice, because a loop bound can be based on another loop variable.

get_base_offset get_base_offset(index)

This method replaces loop variables for a given set of loops with 0. This basically gives us the offset of array references with respect to the loop variable. For example, A and A will give us [4,j+3] with repsect to an i-loop.

get_references get_references()

Method to print out a human readable form of the array references (e.g. [4*i+6]). This is basically what the `puts` method also does.

get_step get_step(index,loops)

Method to retrieve the step for a given array index and loops. The method returns the difference between two subsequent iterations: one with the loop variable at 0 and one after the first increment.

get_sync_id get_sync_id()

Method to print the unique identifier of the loop nest in terms of synchronisation statements to be printed. This is a per-reference id instead of a per-loop id, because it depends on the type of access (read or write).

index_to_interval index_to_interval(index,loops)

Method to fill in the ranges for an array reference. This is based on information of the loop nests. The output is an interval.

step_smaller_than_num_elements? step_smaller_than_num_elements?()

Helper method for the `to_species` method. This method compares the step with the number of elements accessed to determine which one is smaller. FIXME: This is based on the `compare` method which might take a guess.

string_includes_local_vars string_includes_local_vars(string)

to_arc to_arc()

Method to output the result as an array reference characterisation (ARC).

to_copy to_copy(id)

Method to output a copyin/copyout statement. This indicates the name (N), the domain (D), and a unique identifier.

to_species to_species()

Method to output the result as algorithmic species. This reflects the algorithm as presented in the scientific paper.

Module

Bones

#RDoc::Comment:0x00000004791a18>
#RDoc::Comment:0x0000000417ee00>
#RDoc::Comment:0x0000000376ee88>
#RDoc::Comment:0x000000045fc9f0>
#RDoc::Comment:0x00000002e3d210>
#RDoc::Comment:0x00000004526440>
#RDoc::Comment:0x000000032c7df8>
#RDoc::Comment:0x000000035de7a8>
#RDoc::Comment:0x000000043ffa58>

Constants

Name	Value	Description
NL	"\n"	Set the newline character
INDENT	' '	Set the tab size (currently: 2 spaces)
MESSAGE	'[Bones] ### Info :…	A string given as a start of an informative message. See also ERROR [p. 33] and WARNING [p. 33].
WARNING	'[Bones] ### Warning:…	A string given as a start of an warning message. See also ERROR [p. 33] and MESSAGE [p. 28].
ERROR	'[Bones] ### Error :…	A string given as a start of an error message. See also MESSAGE [p. 28] and WARNING [p. 28].
INPUT	'in'	Gives a string representing an read-only variable. See also OUTPUT [p. 33], INOUT [p. 33] and DIRECTIONS [p. 33].
OUTPUT	'out'	Gives a string representing an write-only variable. See also INPUT [p. 28], INOUT [p. 33] and DIRECTIONS [p. 33].
INOUT	'inout'	Gives a string representing an read/write variable. See also INPUT [p. 28], OUTPUT [p. 28] and DIRECTIONS [p. 33].

Name	Value	Description
DIRECTIONS	[INPUT,OUTPUT]	Gives a list of all directions considered. Makes use of the INPUT [p. 28] and OUTPUT [p. 28] constants.
WEDGE	'^'	A string representing the combination character ('^') of a species. See also ARROW [p. 33] and PIPE [p. 33].
ARROW	'->'	A string representing the production character ('->') of a species. See also WEDGE [p. 28] and PIPE [p. 33].
PIPE	'|'	A string representing the pipe character ('|') of a species. See also WEDGE [p. 28] and ARROW [p. 28].
RANGE_SEP	':'	A string representing the colon character (':') to separate ranges in dimensions.
DIM_SEP	','	A string representing the comma character (',') to separate different ranges.
VARIABLE_PREFIX	'bones_'	Sets the prefix used by variables in the skeleton library. This is used in LOCAL_MEMORY [p. 33], GLOBAL_ID [p. 33], LOCAL_ID [p. 33], GLOBAL_SIZE [p. 33] and LOCAL_SIZE [p. 33].
LOCAL_MEMORY	VARIABLE_PREFIX+'loca…	Sets the variable name for the local memory variable in the skeleton library.
PRIVATE_MEMORY	VARIABLE_PREFIX+'priv…	Sets the variable name for the thread private (i.e. register) memory variable in the skeleton library.
GLOBAL_ID	VARIABLE_PREFIX+'glob…	Sets the variable name for the global memory thread index in the skeleton library.
LOCAL_ID	VARIABLE_PREFIX+'loca…	Sets the variable name for the local memory thread index in the skeleton library.

Name	Value	Description
GLOBAL_SIZE	VARIABLE_PREFIX+'glob…	Sets the variable name for the global memory size as used in the skeleton library.
LOCAL_SIZE	VARIABLE_PREFIX+'loca…	Sets the variable name for the local memory size as used in the skeleton library.
INITIALIZATION_DEFINITION	'void '+VARIABLE_PREF…	Provide a function definition for the initialization C-code (if present). See als INITIALIZATION_CODE [p. 33].
INITIALIZATION_CODE	VARIABLE_PREFIX+'init…	Provide a function call to the initialization C-code (if present). See als INITIALIZATION_DEFINITION [p. 28].
GOLDEN	VARIABLE_PREFIX+'gold…	Sets the name for the 'golden' output, required for verification purposes.
LOOP	VARIABLE_PREFIX+'loop…	Sets the loop variable name for the 'golden' output, required for verification purposes.
DEVICE	'device_'	Constant to set the device variable name
SAR_MARKER1	'<'	Provides the starting marker for a search-and-replace variable. See also SAR_MARKER2 [p. 33].
SAR_MARKER2	'>'	Provides the ending marker for a search-and-replace variable. See also SAR_MARKER1 [p. 28].
START_DEFINITION	'\/* STARTDEF'	Set the start of a function definition, used in the skeleton library files. See also END_DEFINITION [p. 33].
END_DEFINITION	'ENDDEF *\/'	Set the end of a function definition, used in the skeleton library files. See also START_DEFINITION [p. 28].

Class

Bones::Algorithm

This class holds one algorithm, which includes a species, a name, and the source C-code.

The algorithm class holds all sorts of information on var- iables. This information is only available after calling the 'populate' method, which populates a lists of varia- bles of all sorts: a regular list, a specialized hash, and lists of input/output array variables.

Constants

Name	Value	Description
ACCELERATED	'_accelerated'	Constant to set the name of the algorithm's accelerated version
ORIGINAL	'_original'	Constant to set the name of the algorithm's original version

Attributes

arrays [R]
code [R]
function_name [R]
hash [RW]
id [R]
lists [R]
merge_factor [RW]
name [R]
register_caching_enabled [RW]
species [R]

Public Class methods

new new(name, filename, id, species, code)

This method initializes the class. It gives the new algorithm a name, species and source code. At initiali- zation, this method checks if the name starts with a digit. This is not allowed, so an underscore is added prior to the digit.

Public Instance methods

generate_replacement_code generate_replacement_code(options, skeleton, verify_code, prefix, timer_start, timer_stop)

This method is used to generate verification code. This verification code contains a copy of the original code. It also provides a verification which compares the output of the original code with the output of the generated code. The verification code prints warnings if the outputs are not equal, else it prints a success message.

opencl_arguments opencl_arguments(list,kernel_id)

Helper function to create a the special code which is required for OpenCL function calls to be able to use kernel arguments.

perform_transformations perform_transformations(transformation_settings)

This method performs the code transformations according to the transformation settings as provided as an argument to the function. It calls the various code transformation functions as implemented for the CAST class. The resulting modified code is finally stored in the search-and-replace hash. This method assumes that the populate method has already been called, such that the hash contains the dimensions needed to create the global ID definitions.

performance_model_code performance_model_code(model_dir)

Method to generate performance modeling code. This method is still under construction and will not be called yet. TODO: Complete this method

populate_hash populate_hash()

This method creates the search-and-replace hash based on information provided by the algorithm. It is called from the 'populate' method of this class.

List of possible hash keys:

algorithm_id
_name
_basename
_filename
_code*

(in*|out*)_type
_name
_devicename
_devicepointer
_dimensions
_dimension*_to
_from
_sum
_to
_from
_parameters
_parameter*_to
_from
_sum
_ids
_localids
_flatindex

(in|out)_names
_devicenames
_devicedefinitions
_devicedefinitionsopencl

names devicenames devicedefinitions devicedefinitionsopencl

parallelism factors ids verifyids

argument_name argument_definition kernel_argument_list

populate_lists populate_lists()

Method to populate 5 lists with variable information. Below are listed the names of the four lists with an example value:

host_name
Example: 'array'
host_definition
Example: 'int array[10]'
Example: 'threshold'
Example: 'float threshold'
golden_name
Example: 'golden_array'

populate_variables populate_variables(original_code,defines)

Method to create a list of variables for the current algorithm. These variables should hold two conditions: 1) they are not local to the algorithm's code, and 2), they are used in the algorithm's code.

The method gets a lists of undefined variables in the algorithm's code and subsequently searches the original code for the definition of this variable.

set_function set_function(full_code)

This method sets the code and name for the function in which the algorithm is found. This is done based on the original code, which is given as input to this method. The method does not return any value, instead, it sets two class variables (@function_code and @function_name).

update_hash update_hash(loop_variable)

This method updates the hash after loops are removed from the code. It takes as an argument a loop variable, which it removes from both the ':argument_name' and ':argument_ definition' hash entries.

Class

Bones::Common

This class is created to be a parent class of the Bones [p. 28] engine, the Bones [p. 28] species and the Bones [p. 28] algorithm class.

Public Instance methods

flatten_hash flatten_hash(hash,flat_hash={},prefix='')

This method flattens a multidimensional hash into a one dimensional hash. This method is called recursively.

from from(range)

Helper to obtain the 'from' part from a range. For example, the method will yield '1' if applied to '1:N-1'.

replace_defines replace_defines(original_code,defines)

Method to process the defines in a piece of code. The code must be formatted as a string. It returns a copy of the code with all defines replaced.

search_and_replace search_and_replace(hash,code)

This method calls search_and_replace [p. 34]! to replaces markers in code with a hash of search-and-replace values. Before, it clones the existing code so that the original copy is maintained.

search_and_replace! search_and_replace!(hash,code)

This method performs a search-and-replace. It searches for the index> of the input hash and replaces it with the corresponding key. It searches for to-be-replaced variables of the form 'name>'. If such patterns still occur after searching and replacing, the method raises an error.

sum sum(range)

Helper to obtain the sum of a range. For example, the method will yield 'N-2' if applied to '1:N-1'. There is a check to ensure that the range is correct.

sum_and_from sum_and_from(range)

Helper to obtain the sum and 'from' part of a range. For example, the method will yield '((N-2)+1)' if applied to '1:N-1'.

to to(range)

Helper to obtain the 'to' part from a range. For example, the method will yield 'N-1' if applied to '1:N-1'.

Class

Bones::Copy

Class copyin/out

Attributes

deadline [RW]
direction [RW]
domain [RW]
id [RW]
name [RW]
scop [RW]

Public Class methods

new new(scop,name,domain,deadline,direction,id)

Public Instance methods

get_definition get_definition(array_definition,type)

get_function_call get_function_call(type)

Class

Bones::Engine

This class holds the main functionality: the Bones [p. 28] source- to-source compilation engine based on algorithmic skeletons. This class processes command line arguments, makes calls to the Bones [p. 28] preprocessor and the CAST gem, analyzes the source code, performs source transformations, instantiates the skeletons, and finally writes output code to file.

Constants

Name	Value	Description
BONES_DIR_SKELETONS	File.join(BONES_DIR,'…	Locate the skeletons directory.
SKELETON_FILE	'skeletons.txt'	Set the name of the transformations file as found in the skeleton library.
TIMER_FILES	['timer_1_start','tim…	A list of timer files to be found in the skeleton library.
COMMON_FILES	['prologue','epilogue…	A list of files to be found in the common directory of the skeleton library (excluding timer files).
COMMON_GLOBALS	'globals'	The name of the file containing the globals as found in the skeleton library
COMMON_HEADER	'header'	The name of the file containing the header file for the original C [p. 54] code as found in the skeleton library
COMMON_GLOBALS_KERNEL	'globals_kernel'	The name of the file containing the globals for the kernel files as found in the skeleton library
COMMON_SCHEDULER	'scheduler'	The name of the file containing the scheduler code
GLOBAL_TIMERS	'timer_globals'	Global timers
SKELETON_HOST	'.host'	The extension of a host file in the skeleton library. See also SKELETON_DEVICE [p. 41].
SKELETON_DEVICE	'.kernel'	The extension of a device file in the skeleton library. See also SKELETON_HOST [p. 36].

Name	Value	Description
OUTPUT_HOST	'_host'	The suffix added to the generated output file for the host file. See also OUTPUT_DEVICE [p. 41].
OUTPUT_DEVICE	'_device'	The suffix added to the generated output file for the device file. See also OUTPUT_HOST [p. 36].
OUTPUT_VERIFICATION	'_verification'	The suffix added to the generated verification file. See also OUTPUT_DEVICE [p. 36] and OUTPUT_HOST [p. 36].

Public Class methods

new new()

Initializes the engine and processes the command line arguments. This method uses the 'trollop' gem to parse the arguments and to create a nicely formatted help menu. This method additionally initializes a result-hash and reads the contents of the source file from disk.

Command-line usage:

```
bones --application <input> --target <target> [OPTIONS]
```

Options:

```
--application, -a <s>:   Input application file
--target, -t <s>:   Target processor (choose from: 'GPU-CUDA','GPU-
OPENCL-AMD','CPU-OPENCL-INTEL','CPU-OPENCL-AMD','CPU-OPENMP','CPU-C')
--measurements, -m:   Enable/disable timers
--version, -v:   Print version and exit
--help, -h:   Show this message
```

Public Instance methods

process process()

Method to process a file and to output target code. This method calls all relevant private methods.

Tasks:

- Run the preprocessor to obtain algorithm information.
- Use the 'CAST' gem to parse the source into an AST.
- Call the code generator to perform the real work and produce output.

write_output write_output()

This method writes the output code to files. It creates a new directory formatted as 'name_target' and produces three files.

Output files:

- `main` - a file containing the original code with function calls substituting the original algorithms.
- `target` - a file containing the host code for the target.
- `kernel` - a file containing the kernel code for the target.

Class

Bones::Preprocessor

This is the C99 pre-processor for Bones [p. 28]. It has two tasks:

- To remove all lines starting with '#' (pre-processor directives).
- To detect all pragma's forming algorithm classes from the source.

Attributes:

- `header_code` - All the code that was removed by the pre-processor but was not relevant to Bones [p. 28]. This contains for example includes and defines.
- `algorithms` - An array of identified algorithms, each of class Bones::Algorithm [p. 31].
- `target_code` - The processed code containing no Bones [p. 28] directives nor other pre-processor directives (such as includes and defines).

Constants

Name	Value	Description
`IDENTIFIER`	`'#pragma species'`	Denotes the start of an algorithmic species.
`WHITESPACE`	`'\s*'`	Regular expression to identify whitespaces (tabs, spaces).
`SPECIES_START`	`IDENTIFIER+' kernel'`	This directive denotes the start of a algorithm. It is based on the IDENTIFIER [p. 39] constant.
`SPECIES_END`	`IDENTIFIER+' endkerne…`	This directive denotes the end of a algorithm. It is based on the IDENTIFIER [p. 39] constant.
`SCOP_START`	`'#pragma scop'`	Start of the scop
`SCOP_END`	`'#pragma endscop'`	Enf of the scop
`SYNC`	`IDENTIFIER+' sync'`	Synchronise directive.
`COPYIN`	`IDENTIFIER+ ' copyin'…`	Copy [p. 35] in directive.
`COPYOUT`	`IDENTIFIER+ ' copyout…`	Copy [p. 35] out directive.
`REGEXP_PREFIX`	`/^[a-z]+ /`	A regular expression captures a prefix in a algorithm (e.g. unordered/multiple).
`DEFAULT_NAME`	`'algorithm'`	Providing a default name in case a algorithm is not named.

Attributes

algorithms [R]
copies [R]
defines [R]
device_header [R]
header_code [R]
scop [R]
target_code [R]

Public Class methods

new new(source_code,directory,filename,scheduler)

This is the method which initializes the preprocessor. Initialization requires the target source code to process, which is then set as the class variable +@source_code+.

Public Instance methods

process process()

This is the method to perform the actual preprocessing. This method takes care of all the preprocessor tasks. The output is stored in the three attributes `header_code`, `algorithms`, and `target_code`.

process_header process_header(filename)

This is the method to preprocess a header file. Currently, it only searches for defines and adds those to a list. In the meanwhile, it also handles ifdef's.

Class

Bones::Species

The species class contains 'algorithm classes', or 'species'. Individual species contain a number of input and output structures and possibly a prefix.

Examples of species are found below:

```
0:9|element -> 0:0|shared
0:31,0:31|neighbourhood(-1:1,-1:1) -> 0:31,0:31|element
unordered 0:99,0:9|chunk(0:0,0:9) -> 0:99|element
```

Naming conventions

The species class uses several naming conventions within functions. They are as follows for the example species '0:31,0:15|neighbourhood(-1:1,-1:1) ^ 0:31,0:15|element -> 0:31,0:15|element':

input
'0:31,0:15|neighbourhood(-1:1,-1:1) ^ 0:31,0:15|element'
output
'0:31,0:15|element'
structures
\['0:31,0:15|neighbourhood(-1:1,-1:1)', '0:31,0:15|element', '0:31,0:15|element']
structure
'0:31,0:15|neighbourhood(-1:1,-1:1)' or '0:31,0:15|element' (twice)
pattern
'neighbourhood' or 'element'
ranges
\['0:31', '0:15'] or ['-1:1', '-1:1']
range
'0:31' or '0:15' or '-1:1'
from
'0' or '-1'
to
'31' or '15' or '1'
sum
'32' or '16' or '3'

Attributes

inputs [R]
name [R]

outputs [R]
prefix [R]
settings [RW]
skeleton_name [RW]

Public Class methods

new new(prefix, input, output)

Initializes the species with a prefix, inputs and out- puts. It additionally verifies the correctness of the provided raw data.

Public Instance methods

all_structures all_structures()

Method to return an array of structures for both direc- tions.

match? match?(file,search)

This method implements the match between a species' structure and a structure found in the mapping file. It is called from the `match_species` method. It first checks for a pattern match, followed by a match of the dimensions. The method returns either true or false. TODO: Complete the matching (N-dimensional).

match_species? match_species?(file_data)

This method is called by the `set_skeleton` method. It performs a match between the current species and the species found in the mapping file. The matching is based on a fixed order of patterns. The method returns either true or false.

ordered ordered(direction,order)

Method to return an ordered array of structures in a given direction. The order is specified by an argument which contains a list of pattern names.

set_skeleton set_skeleton(mapping_file)

This method maps an algorithm species to a skeleton. This is done based on a mapping file provided as part of the skeleton library. If multiple skeletons match the current species, the first found match is taken. This method does not return any values, but instead sets the class variables `skeleton_name` and `settings`.

set_structures set_structures(raw_data)

This method splits the raw data (a string) into seperate structures. The method returns an array of structures.

shared? shared?()

Method to verify whether this species is shared-based or not. The method either returns true (if 'shared' is included in the input or output) or false (if not).

structures structures(direction)

Method to return an array of structures in a given di- rection.

verify_species

This method verifies if the species dimensions match with its parameters in terms of number of dimensions.

Class

Bones::Structure

This class represents a single structure in a species. Such a structure is a single input or output 'structure'. Stuctures can be set as part of array variables. Examples of structures are given below:

```
0:9|element
0:0|shared
0:31,0:31|neighbourhood(-1:1,-1:1)
0:99,0:9|chunk(0:0,0:9)
```

Attributes

dimensions [R]
name [RW]
parameters [R]
pattern [R]

Public Class methods

new new(raw_data)

The structure is initialized by the full name given as a string. It is then analyzed and stored accordingly in a number of class variables.

Public Instance methods

chunk? chunk?()

Method to check whether a structure is chunk-based.

element? element?()

Method to check whether a structure is element-based.

empty? empty?()

Method to verify if a structure is empty or not (e.g. if it is based on the 'void' pattern.

from_at from_at(n)

Method to get the start of a range given a dimension 'n'. The method returns the proper simplified result, taking chunk/neighbourhood-sizes into account.

full? full?()

Method to check whether a structure is full-based.

has_arrayname? has_arrayname?()

Method to find out if the structure has a arrayname defined. This is optional for a structure.

has_parameter? has_parameter?()

Method to find out if the structure has a parameter. This is only the case if it is neighbourhood or chunk based.

neighbourhood? neighbourhood?()

Method to check whether a structure is neighbourhood-based.

reverse? reverse?()

TODO: Implement the reverse function

shared? shared?()

Method to check whether a structure is shared-based.

to_at to_at(n)

Method to get the end of a range given a dimension 'n'. The method returns the proper simplified result, taking chunk/neighbourhood-sizes into account.

Class

Bones::Variable

This class represents individual variables. Variables have a name, a type, a direction and an id. They might also have an algorithmic species coupled if they represent an array. The variable type is in AST form and holds information which can be extracted through methods implemented for the Type class.

The provided methods are able to obtain information from the variables, such as the number of dimensions and the definition. Other methods query variables for properties, such as whether a variable is an array or not. Several methods should only be executed if the variable is an array - this is not checked within the methods itself.

Attributes

direction [R]
factors [R]
guess [RW]
name [R]
size [RW]
species [RW]
type [R]

Public Class methods

new new(name,type,size,direction,id,shared)

Method to initilize the variable class with a name, type and a direction.

Public Instance methods

definition definition()

Method to obtain the full defintion of a variable. This includes the variable type, the variable name, and the dimensions and/or pointers. Example return values are:

```
int example[10]
char **variable
unsigned int* example[N]
float array[][]
```

device_definition device_definition()

Method to return the device version of a variable's definition. This includes the variable's type, zero or more stars and the variable's name, all returned as a single string.

device_name device_name()

This method returns the device name of the variable.

device_pointer device_pointer()

This method returns a star ('*') if the variable is an array or an empty string if it is not. It will not

be able to return more than a single star, as device varia- bles are assumed to be flattened.

dimensions dimensions()

Method to obtain the number of dimensions of a variable. This method returns a positive integer. The functionality is implemented as a recursive search in the Type class.

dynamic? dynamic?()

This method detects whether the variable is dynamically allocated or not. It returns either true or false.

flatindex flatindex()

Method to return the full flattened address.

flatten flatten()

Method to flatten an array into a one dimensional array. If an array has multiple dimensions, the method will return a '[0]' for each additional dimension. This method assumes that the variable is an array.

golden_name golden_name()

This method returns the 'golden' name of a variable. This is used to generate verification code.

initialization initialization()

This method returns the initialization code for a varia- ble, formatted as a string. This is used to generate the verification code.

input? input?()

Method to find out whether a variable is used as an in- put. If the current algorithm uses the 'shared' pattern, then the variable must be input only, otherwise it can be both input or inout to be considered input. The return value is boolean.

output? output?()

Method to find out whether a variable is used as an out- put. The variable can be both output or inout to be con- sidered input. The return value is boolean.

set_factors set_factors()

Method to return an array of multiplication factors for multi-dimensional arrays that need to be flattened. For every dimension, one factor is generated.

type_name type_name()

Method to obtain the type of a variable, omitting any information about arrays or pointers. Examples of out- puts are: `int`, `float` or `char`. This functionality is implemented as a recursive search in the Type class, since the type is in AST form.

Class

Bones::Variablelist

This class is based on the standard Array class. It is meant to contain a list of elements of the Variable [p. 46] class. In that sense, using the Array class will suffice. However, this class extends the list with a small number of addi- tional methods. These methods involve selecting a subset of the list or sorting the list.

Attributes

representative [RW]

Public Instance methods

inputs inputs()

This method is a short-hand version to select a list of input variables. It calls the `select` method internally.

inputs_only inputs_only()

This method is a short-hand version to select a list of input only variables. It calls the `select` method internally.

outputs outputs()

This method is a short-hand version to select a list of output variables. It calls the `select` method internally.

outputs_only outputs_only()

This method is a short-hand version to select a list of input only variables. It calls the `select` method internally.

select select(direction)

This method returns a subset of the list, based on the argument `direction` given. It either returns a list of input variables or a list of output variables.

set_representative set_representative(ids)

Method to set a representative variable for this variable- list. It is set based on the variable's species-name, e.g. 'in0' or 'out2'.

sort_by sort_by(alphabet)

This method sorts the list of variables based on its species' pattern (e.g. element or chunk). An alphabet is based as an argument to this method to specify the prefered order. This alphabet must be an array of strings.

Module

C

#RDoc::Comment:0x00000004292058>

#RDoc::Comment:0x000000032cd2d0>

Class

C::FloatLiteral

update the FloatLiteral [p. 50] and IntLiteral classes in CAST to print suffixes

Public Instance methods

to_s to_s()

Class

C::Index

This class provides an extention to the CAST index class. The class contains a number of functions applicable to array accesses of the form 'array[y]' or 'vector'.

The provided methods are helpers to extend the CAST functionality and to clean-up the Bones [p. 28] classes.

Public Instance methods

dimension dimension(count=1)

This method returns the dimension of an index expression. It starts at dimension 1, but if it can find a new dimension it will increment the count and call itself again.

index_at_dimension index_at_dimension(dimension)

This method returns the index itself at a given dimension. It uses recursion to iterate through the dimensions, but will eventually return a new index node.

variable_name variable_name()

This method is a recursive method which **gets** the name of a variable from the index definition. Depending on the number of dimensions, it will go deeper into the structure and eventually return the name.

variable_name= variable_name=(name)

This method is a recursive method which **sets** the name of a variable from the index definition. Depending on the number of dimensions, it will go deeper into the structure and eventually set the name.

Class

C::IntLiteral

Public Instance methods

to_s to_s()

Class

C::Node

#RDoc::Comment:0x00000004596cb8>
#RDoc::Comment:0x000000036a7a68>
#RDoc::Comment:0x00000002d4f088>
#RDoc::Comment:0x00000004347ae8>

Public Instance methods

add? add?()

This method returns 'true' if the node is of the 'Add' class. Otherwise, it returns 'false'.

addassign? addassign?()

This method returns 'true' if the node is of the 'AddAssign' class. Otherwise, it returns 'false'.

alu? alu?()

This method returns 'true' if the node is performing an ALU operation. Otherwise, it returns 'false'.

and? and?()

This method returns 'true' if the node is of the 'And' class. Otherwise, it returns 'false'.

array? array?()

This method returns 'true' if the node is of the 'Array' class. Otherwise, it returns 'false'.

assign? assign?()

This method returns 'true' if the node is of the 'Assign' class. Otherwise, it returns 'false'.

assignment_expression? assignment_expression?()

This method returns 'true' if the node's class inherits from the 'AssignmentExpression' class. Otherwise, it returns 'false'.

binary_expression? binary_expression?()

This method returns 'true' if the node's class inherits from the 'BinaryExpression' class. Otherwise, it returns 'false'.

block? block?()

This method returns 'true' if the node is of the 'Block' class. Otherwise, it returns 'false'.

call? call?()

This method returns 'true' if the node is of the 'Call' class. Otherwise, it returns 'false'.

declaration? declaration?()

This method returns 'true' if the node is of the 'Declaration' class. Otherwise, it returns 'false'.

declarator? declarator?()

This method returns 'true' if the node is of the 'Declarator' class. Otherwise, it returns 'false'.

direction direction(variable_name)

This method finds the direction of a variable based on the node information. The direction of a variable can be either:

- `in:` - The variable is accessed read-only.
- `out:` - The variable is accessed write-only.

The method takes the name of a variable and walks through the node to search for expressions (assignments and binary expressions). For each expression it takes the first and second part of the expression and stores it in a list. Afterwards, the expressions in the list are analysed for occurrences of the variable.

The method raises an error if the variable does not appear at all: it is neither input nor output.

equality? equality?()

This method returns 'true' if the node is of the 'Equal' class. Otherwise, it returns 'false'.

for_statement? for_statement?()

This method returns 'true' if the node is of the 'For' class. Otherwise, it returns 'false'.

function_declaration? function_declaration?()

This method returns 'true' if the node is of the 'Declarator' class with its 'indirect_type' equal to 'Function' . Otherwise, it returns 'false'.

function_definition? function_definition?()

This method returns 'true' if the node is of the 'FunctionDef' class. Otherwise, it returns 'false'.

get_accesses get_accesses(accesses = [],loop_data = [],if_statements = [])

This method retrieves all array references in the current node. It retrieves information on loops and on if-statements as well. This method is destructive on the current node. It is furthermore called recursively.

get_all_loops get_all_loops()

This method retrieves all loops from a loop nest. The method is based on the `get_single_loop` method to extract the actual loop information.

get_array_name get_array_name()

This method retrieves the name of the array reference.

get_complexity get_complexity()

This method returns the complexity of a piece of code in terms of the amount of ALU operations (multiplications, additions, etc.).

get_conditions get_conditions(results=[])

This method retrieves the bounds for an if-statement. The method is called recursively if there are multiple conditions. TODO: What about '||' (or) conditions? They are currently handles as '&&'. TODO: Are these all the possibilities (&&,||,>=,>,=,) for conditions?

get_direct_loops get_direct_loops(loop_data = [])

This method retrieves all directly following loops from a node, i.e. the loops belonging to a perfectly nested loop. It is a recursive method: it retrieves a first loop and calls the method again on the body of the loop. It collects all the data in the `loop_data` array.

get_functions get_functions()

Method to obtain a list of all functions in the code. If no functions can be found, an empty array is returned. For every function found, the function itself is pushed to the list. This method also makes sure that there is at least one function with the name 'main'. If this is not the case, an error is raised.

get_index_nodes get_index_nodes()

This method retrieves all nodes from the current node that are index node. Such nodes represent array references, e.g. in A, [i+3] is the index node.

get_indices get_indices()

This method retrieves all indices of index nodes from the current node.

get_single_loop get_single_loop()

This method retrieves a single loop from the current node and collects its data: 1) the loop variable, 2) the lower-bound, 3) the upper-bound, and 4) the loop step. FIXME: For decrementing loops, should the min/max be swapped?

get_value get_value()

This method retrieves the value from the current node. The value can be an integer (in case of a constant) or a string (in case of a variable).

get_var_declarations get_var_declarations()

This method retrieves all variable declarations

has_conditional_statements? has_conditional_statements?()

This method checks whether the given code has any conditional statements (if-statements)

if_statement? if_statement?()

This method returns 'true' if the node is of the 'If' class. Otherwise, it returns 'false'.

index? index?()

This method returns 'true' if the node is of the 'Index' class. Otherwise, it returns 'false'.

intliteral? intliteral?()

This method returns 'true' if the node is of the 'IntLiteral' class. Otherwise, it returns 'false'.

lengths lengths(result = [])

This is a helper method which calls itself recursively, depending on the dimensions of the variable. It stores the resulting array sizes in an array 'result'.

less? less?()

This method returns 'true' if the node is of the 'Less' class. Otherwise, it returns 'false'.

less_or_equal? less_or_equal?()

This method returns 'true' if the node is of the 'LessOrEqual' class. Otherwise, it returns 'false'.

more? more?()

This method returns 'true' if the node is of the 'More' class. Otherwise, it returns 'false'.

more_or_equal? more_or_equal?()

This method returns 'true' if the node is of the 'MoreOrEqual' class. Otherwise, it returns 'false'.

node_exists? node_exists?(target)

This method searches for a target node and checks whether it exists. The input to the method is the target node. The method walks through the node and checks whether:

- The node's class is the same as the target class (+node.class == target.class+)
- The node has a parent (+node.parent != nil+)
- The node is equal to the target node (+node.match?(target)+)

If all checks are successful, the method will return the value 'true' immediately. If the target node cannot be found, the method returns 'false'.

or? or?()

This method returns 'true' if the node is of the 'Or' class. Otherwise, it returns 'false'.

parameter? parameter?()

This method returns 'true' if the node is of the 'Parameter' class. Otherwise, it returns 'false'.

pointer? pointer?()

This method returns 'true' if the node is of the 'Pointer' class. Otherwise, it returns 'false'.

postdec? postdec?()

This method returns 'true' if the node is of the 'PostDec' class. Otherwise, it returns 'false'.

postinc? postinc?()

This method returns 'true' if the node is of the 'PostInc' class. Otherwise, it returns 'false'.

predec? predec?()

This method returns 'true' if the node is of the 'PreDec' class. Otherwise, it returns 'false'.

preinc? preinc?()

This method returns 'true' if the node is of the 'PreInc' class. Otherwise, it returns 'false'.

preprocess preprocess(conditional=true)

Pre-process method. It currently pre-processes a piece of code (typically the kernel code) to replace particular code structures with others, which can be handled (better) by Bones [p. 28]. For now, the pre-process method performs the following transformations:

- Replaces all incrementors (i++) outside for loops with an assignment (i=i+1).
- Replaces all decrementors (i--) outside for loops with an assignment (i=i-1).

remove_index remove_index()

This method is a small helper function to remove index nodes from a node. It first clones to original node in order to not overwrite it, then walks the node and removes index nodes. Finally, it returns a new node.

remove_loop remove_loop(from,to)

This method walks through the node and finds the first for-loop. If it is found, it returns the contents of the for-loop and the name of the loop variable. Obtaining the loop variable is conditional because it can either be an assignment ('k=0') or a variable definition ('int k=0').

The method raises an error when no for-loop can be found. It also raises an error if the loop is not in canonical form.

remove_once remove_once(target)

This method is a small helper function to remove a node once.

rename_variables rename_variables(suffix,excludes)

replace_variable replace_variable(variable_name,replacement)

This method searches for a variable name in the node and replaces it with the method's argument, which is given as a string. The node itself is modified. The method checks whether:

- The node is a variable (`node.variable?`)
- The variable has the correct name (+node.name == variable_name+)
- The variable is not in a function call (+!node.parent.call?+)

search_and_replace_function_call search_and_replace_function_call(target,replacements)

This method searches for a target function call and replaces it with another. Both the target and the replacement function call are given as arguments to the method. The method walks through the node and checks whether:

- The node's class is the same as the target class (+node.class == target.class+)
- The node has a parent which is a function call (`node.parent.call?`)
- The node is equal to the target node (+node.match?(target)+)

If all checks are successful, the node will be replaced with the replacement node. The method will continue searching for other occurrences of the function call.

The method returns itself.

search_and_replace_function_definition search_and_replace_function_definition(old_name,new_name)

This method searches for a target function name and replaces it with another name. Both the target and the replacement name are given as arguments to the method. The method walks through the node and checks whether:

- The node's class is a function definition or declaration
- The node's name is equal to the target node's name

If the checks are successful, the node's name will be replaced The method will continue searching for other occurrences of functions with the same name.

The method returns itself.

search_and_replace_node search_and_replace_node(target,replacements)

This method searches for a target node and replaces it with a replacement node. Both the target

node and the replacement node are given as arguments to the method. The method walks through the node and checks whether:

- The node's class is the same as the target class (+node.class == target.class+)
- The node has a parent (+node.parent != nil+)
- The node is equal to the target node (+node.match?(target)+)

If all checks are successful, the node will be replaced with the replacement node and the method will return immediately.

The method returns itself if the target node cannot be found.

---

size size(variable_name)

This method returns the sizes of a variable as defined at the initialization of the array. There are multiple possibilities:

- Static arrays (e.g. int array)
- Static arrays with defines (e.g. int input[N2])
- Variable length arrays (e.g. float temp[m])
- Dynamically allocated arrays (e.g. int *a = (int *)malloc(size*4))

---

statement? statement?()

This method returns 'true' if the node is of the 'ExpressionStatement' class. Otherwise, it returns 'false'.

---

string? string?()

This method returns 'true' if the node is of the 'StringLiteral' class. Otherwise, it returns 'false'.

---

strip_brackets strip_brackets()

This method is a small helper function which simply strips any outer brackets from a node. If no outer brackets are found, then nothing happens and the node itself is returned.

---

subtract? subtract?()

This method returns 'true' if the node is of the 'Subtract' class. Otherwise, it returns 'false'.

---

transform_flatten transform_flatten(array)

This method transforms multi-dimensional arrays into 1D arrays. The target array variable list is given as argument. The method walks through the node. First, it checks whether:

- The node represents an array access (`node.index?`)
- The node has a parent node (`node.parent`)

Then, the method loops over all array variables. It then checks two more things:

- Whether the given name is the same as the name found in the array access node (+node.variable_name == array.name+)
- Whether the dimensions of the given array are the same as the dimensions of the node (+node.dimension == array.dimension+)

Then, the method is ready to perform the flattening. It first gets the index for the first dimension and then iterates over all remaining dimensions. For those dimensions, the index is multiplied by the size of the previous dimension.

The method performs the transformation on the node itself. Any old data is thus lost.

---

transform_merge_threads transform_merge_threads(amount,excludes)

Method to merge the computations of multiple threads.

transform_reduction transform_reduction(input_variable,shared_variable,id)

This method provides the transformations necessary to perform reduction type of operations. The transformations involved in this function are on variable names and index locations. The argument `id` specifies which transformation to be performed.

Accepted inputs at this point: 2, 3 and 4 (CUDA/OPENCL) Also accepted input: 8 (CUDA), 9 (OPENCL) (to create an atomic version of the code) TODO: Complete the atomic support, e.g. add support for multiplications and ternary operators

transform_shuffle transform_shuffle(arrays)

Method to shuffle a 2D array access (e.g. transform from A[j] into A[i]).

transform_substitution transform_substitution(array,inout)

Method to transform array accesses into register accesses. This is only valid for the local loop body and could have been done by the actual compiler in a number of cases.

transform_use_local_memory transform_use_local_memory(arrays)

Method to enable the use of local memory for a list of array variables which is given as an argument to the method. The method walks through the node. First, it checks whether:

- The node represents an array access (`node.index?`)
- The node has a parent node (`node.parent`)

Then, the method loops over all array variables. It checks one more thing: whether the array variable's name is the same as the name found in the array access node.

If all conditions are met, the method performs two replacements:

- The variable name is changed to correspond to local memory
- The index names are changed to correspond to local indices

The method performs the transformation on the node itself. Any old data is thus lost.

undefined_variables undefined_variables()

This method returns a list of undefined variables in the node. It walks the node tree until it finds a node that full-fills the following:

- The node is a variable (`node.variable?`)
- The variable is not in a function call (+!node.parent.call?+)
- The variable is not defined in the code (+!self.variable_type(node.name)+)

unit_increment? unit_increment?()

This method returns 'true' if the node is of the 'PostInc', 'PreInc' class or if it is of the 'Assign' class and adds with a value of 1. Otherwise, it returns 'false'.

variable? variable?()

This method returns 'true' if the node is of the 'Variable' class. Otherwise, it returns 'false'.

variable_type variable_type(variable_name)

This method returns the type of a variable (e.g. int, float). The method requires the name of a variable as an argument. It first tries to find a declaration for the variable in by walking through the node. If it cannot find it, it will search for a parameter definition from a function call. If that

cannot be found either, the method will return 'nil', meaning that the variable is not defined at all in the current node.

Class

C::NodeList

Modify the NodeArray and NodeChain lists to output correct code when printed to a file.

Public Instance methods

to_s to_s()

Modify the 'to_s' method to output correct code when printed to a file. Originally, it would separate instances of the list with a ','. Instead, a newline command is added.

Class

C::Type

This class provides an extention to the CAST type class. It contains a number of functions applicable to types such as pointers, arrays, structures, floats, integers, etc.

The provided methods are just helpers to extend the CAST functionality and to clean-up the Bones [p. 28] classes.

Public Instance methods

array_or_pointer? array_or_pointer?()

This method is used to determine whether the variable is an array and/or a pointer. Returns either true or false.

dimensions dimensions(count=0)

This method returns the variable's dimension as an integer. it uses recursion in case the type is an array or a pointer. Types that are neither arrays nor pointers have a dimension of zero. For arrays and pointers, each '*' or '[]' contributes to one additional dimension.

type_name type_name()

This method recursively searches for the type of a variable. Recursion is needed when a type is an array or a pointer. The method eventually returns one of the CAST algorithm types being either: void, int, float, char, bool, complex or imaginary.

Class

Object

Constants

Name	Value	Description
NL	"\n"	Set the newline character
INDENT	"\t"	Set the tab size (currently: 2 spaces)
WEDGE	'^'	A string representing the combination character ('^') of a species.
ARROW	'->'	A string representing the production character ('->') of a species.
PIPE	'|'	A string representing the pipe character ('|') of a species.
RANGE_SEP	':'	A string representing the colon character (':') to separate ranges in dimensions.
DIM_SEP	','	A string representing the comma character (',') to separate different ranges.
ASSUME_VAL	'1000'	Value to assume a variable to be

Public Instance methods

abs abs(expr)

Return the absolute value (if possible)

code_from_loops code_from_loops(loops,statements)

Generate code from a combination of loops and statements (the body)

compare compare(expr1,expr2,loop_data,assumptions=[])

Compare two expressions

create_if create_if(loop_var,reference_bound,loop_bound,code,condition)

Create an if-statement in front of a statement

exact_max

Find the exact maximum value of 2 expressions

exact_min exact_min(expr1,expr2)

Find the exact minimum value of 2 expressions (based on the exact_max [p. 63] method)

fusion_is_legal? fusion_is_legal?(a, b)

Determine whether kernel fusion is legal (see algorithm in paper/thesis)

get_body get_body(num_loops,code)

Return the body of a loop nest

get_vars get_vars(expr)

Get the variables in an expression

kernel_fusion kernel_fusion(nests, settings)

Perform the kernel fusion transformations

max max(expr1,expr2,assumptions=[])

Find the maximum value of 2 expressions

min min(expr1,expr2)

Find the minimum value of 2 expressions (based on the max method)

perform_copy_optimisations1 perform_copy_optimisations1(nests,options)

First set of copyin/copyout optimisations (recursive)

perform_copy_optimisations2 perform_copy_optimisations2(nests,options)

Second set of copyin/copyout optimisations (non-recursive)

perform_copy_optimisations3 perform_copy_optimisations3(nests,options)

Third set of copyin/copyout optimisations (inter-level)

recursive_copy_optimisations recursive_copy_optimisations(nests,options)

Recursive copy optimisations

simplify simplify(expr)

Helper method to evaluate mathematical expressions, possibly containing symbols. This method is only used for readability, without it the code is functionally correct, but expressions might be larger than needed.

solve solve(equality,variable,forbidden_vars)

Solve a linear equality (work in progress)