Tadashi

Release 0.1.0-23-g7a9ac8a

Anonymous authors

Sep 21, 2024

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

TADASHI

1.1 tadashi package

1.1.1 Submodules

1.1.2 tadashi.apps module

class tadashi.apps.**App**

Bases: object

property include_paths: list[Path]

compile() → bool

property compile_cmd: list[str]

property source_path: Path

property output_binary: Path

property run_cmd: list

measure(**args*, ***kwargs*) → float

static extract_runtime(*stdout*) → float

class tadashi.apps.**Simple**(*source: str*, *alt_source: str = ''*)

Bases: *App*

source: Path

tmpdir: TemporaryDirectory

alt_source: Path

property compile_cmd: list[str]

property source_path: Path

property alt_source_path: Path

property output_binary: Path

static extract_runtime(*stdout*)

class tadashi.apps.**Polybench**(*benchmark: str*, *base: str*, *compiler_options=[]*)

Bases: *App*

A single benchmark in of the Polybench suite.

benchmark: Path

base: Path

property source_path: Path

property output_binary: Path

property utilities_path: Path

property include_paths: list[Path]

property alt_source_path: Path

property compile_cmd: list[str]

static extract_runtime(*stdout*) → float

1.1.3 tadashi.simple module

tadashi.simple.**main**()

1.1.4 tadashi.tadashilib module

Main Tadashi package.

class tadashi.tadashilib.**AstLoopType**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Possible values for SET_LOOP_OPT.

UNROLL should be avoided unless the requirements in the *ISL Docs* are satisfied.

ISL Docs

ISL online user manual (AST generation options).

DEFAULT = 0

ATOMIC = 1

UNROLL = 2

SEPARATE = 3

class tadashi.tadashilib.**NodeType**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Type of the schedule tree node.

Details: ISL online user manual (Schedule Trees).

BAND = 0

CONTEXT = 1

DOMAIN = 2

EXPANSION = 3

EXTENSION = 4

FILTER = 5

LEAF = 6

GUARD = 7

MARK = 8

SEQUENCE = 9

SET = 10

class tadashi.tadashilib.**TrEnum**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

Enums of implemented transformations.

One of these enums needs to be passed to *Node.transform()* (with args) to perform the transformation.

TILE = 'tile'

INTERCHANGE = 'interchange'

FUSE = 'fuse'

FULL_FUSE = 'full_fuse'

PARTIAL_SHIFT_VAR = 'partial_shift_var'

PARTIAL_SHIFT_VAL = 'partial_shift_val'

FULL_SHIFT_VAR = 'full_shift_var'

FULL_SHIFT_VAL = 'full_shift_val'

FULL_SHIFT_PARAM = 'full_shift_param'

PARTIAL_SHIFT_PARAM = 'partial_shift_param'

SET_PARALLEL = 'set_parallel'

SET_LOOP_OPT = 'set_loop_opt'

PRINT_SCHEDULE_NODE = 'print_schedule_node'

class tadashi.tadashilib.**Node**(*scop: Scop*, *node_type: NodeType*, *num_children: int*, *parent_idx: int*, *location: list[int]*, *loop_signature: list[dict]*, *expr: str*, *children_idx: list[str]*)

Bases: object

Schedule node (Python representation).

scop: *Scop*

Pointer to the *Scop* object the node belongs to.

node_type: *NodeType*

Type of the node in the schedule tree.

num_children: int

Number of children of the node in the schedule tree.

parent_idx: int

The index of the parent of the node in the schedule tree according to *Scop.schedule_tree*.

location: list[int]

List of child indexes which determine the location of the node starting from the root. See *Scop.locate*.

loop_signature: list[dict]

Description of the band nodes (see *Scop.get_loop_signature*).

expr: str

The ISL expression of the schedule node.

children_idx: list[str]

Index of the children in *Scop.schedule_tree*.

property parent

The node which is the parent of the current node.

property children

List of nodes which are the children of the current node.

locate()

Set the current_node to point to self.

transform(*trkey:* TrEnum, **args*)

Execute the selected transformation.

Parameters

- **trkey** (TrEnum) – Transformation Enum.
- **args** – Arguments passed to the transformation corresponding toe trkey.

rollback() → None

Roll back (revert) the last transformation.

property valid_transformation: bool

Check the validity of the transformation.

property available_transformations: list[*TrEnum*]

List transformations available at the *Node*.

valid_args(*tr:* TrEnum, **args*) → bool

Check the validity of args.

available_args(*tr:* TrEnum) → list

Describe available args.

class tadashi.tadashilib.**LowerUpperBound**(*lower*, *upper*)

Bases: tuple

Integer interval description.

Lower and upper bounds for describing (integer) intervals of valid arguments for transformations. None indicates no upper/lower bound.

lower

Alias for field number 0

upper

Alias for field number 1

class tadashi.tadashilib.**TransformInfo**

Bases: object

Abstract base class used to describe transformations.

func_name: str

The name of the C/C++ function in the so file.

argtypes: list[type] = []

Types of arguments as required by ctypes.

arg_help: list[str] = []

Help string describing the arg.

restype

Type of the result as required by ctypes.

alias of c_bool

static valid(*node:* Node) → bool

Check that the transformation is valid on the node.

static valid_args(*node:* Node, **arg*, ***kwargs*) → bool

Check that args of the transformation is valid on node.

static available_args(*node:* Node) → list

Return a list describing each of the args.

class tadashi.tadashilib.**TileInfo**

Bases: *TransformInfo*

func_name: str = 'tile'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_ulong'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Tile size']

Help string describing the arg.

restype

alias of c_bool

static valid_args(*node*, *arg*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**InterchangeInfo**

Bases: *TransformInfo*

func_name: str = 'interchange'

The name of the C/C++ function in the so file.

static valid(*node:* Node)

Check that the transformation is valid on the node.

class tadashi.tadashilib.**FuseInfo**

Bases: *TransformInfo*

func_name: str = 'fuse'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_int'>, <class 'ctypes.c_int'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Index of first loop to fuse', 'Index of second loop to fuse']

Help string describing the arg.

static valid(*node:* Node)

Check that the transformation is valid on the node.

static valid_args(*node:* Node, *loop_idx1: int*, *loop_idx2: int*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**FullFuseInfo**

Bases: *TransformInfo*

func_name: str = 'full_fuse'

The name of the C/C++ function in the so file.

static valid(*node:* Node)

Check that the transformation is valid on the node.

class tadashi.tadashilib.**FullShiftValInfo**

Bases: *TransformInfo*

func_name: str = 'full_shift_val'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_long'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Value']

Help string describing the arg.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**PartialShiftValInfo**

Bases: *TransformInfo*

func_name: str = 'partial_shift_val'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_int'>, <class 'ctypes.c_long'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Statement index', 'Value']

Help string describing the arg.

static valid_args(*node:* Node, *stmt_idx: int*, *value: int*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**FullShiftVarInfo**

Bases: *TransformInfo*

func_name: str = 'full_shift_var'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_long'>, <class 'ctypes.c_long'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Coefficient', 'Variable index']

Help string describing the arg.

static valid_args(*node:* Node, *_coeff: int*, *var_idx: int*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**PartialShiftVarInfo**

Bases: *TransformInfo*

func_name: str = 'partial_shift_var'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_int'>, <class 'ctypes.c_long'>, <class 'ctypes.c_long'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Statement index', 'Coefficient', 'Variable index']

Help string describing the arg.

static valid_args(*node:* Node, *stmt_idx: int*, *coeff: int*, *var_idx: int*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**FullShiftParamInfo**

Bases: *TransformInfo*

func_name: str = 'full_shift_param'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_long'>, <class 'ctypes.c_long'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Coefficient', 'Parameter index']

Help string describing the arg.

static valid_args(*node:* Node, *coeff:* *int*, *param_idx:* *int*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**PartialShiftParamInfo**

Bases: *TransformInfo*

func_name: str = 'partial_shift_param'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_int'>, <class 'ctypes.c_long'>, <class 'ctypes.c_long'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Statement index', 'Coefficient', 'Parameter index']

Help string describing the arg.

static valid_args(*node:* Node, *stmt_idx:* *int*, *coeff:* *int*, *param_idx:* *int*)

Check that args of the transformation is valid on node.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**SetParallelInfo**

Bases: *TransformInfo*

func_name: str = 'set_parallel'

The name of the C/C++ function in the so file.

class tadashi.tadashilib.**SetLoopOptInfo**

Bases: *TransformInfo*

func_name: str = 'set_loop_opt'

The name of the C/C++ function in the so file.

argtypes: list[type] = [<class 'ctypes.c_int'>, <class 'ctypes.c_int'>]

Types of arguments as required by ctypes.

arg_help: list[str] = ['Iterator index', 'Option']

Help string describing the arg.

static available_args(*node:* Node)

Return a list describing each of the args.

class tadashi.tadashilib.**PrintScheduleNodeInfo**

Bases: *TransformInfo*

func_name: `str = 'print_schedule_node'`

The name of the C/C++ function in the `so` file.

static valid(*node:* Node) → bool

Check that the transformation is valid on the node.

class tadashi.tadashilib.**Scop**(*idx*, *ctadashi*)

Bases: `object`

One SCoP in *Scops*, a loop nest (to use a rough analogy).

In the .so file, there is a global `std::vector` of `isl_scop` objects. Objects of *Scop* (in python) represents a the `isl_scop` object by storing its index in the `std::vecto`.

get_loop_signature()

Extract the value for *Node.loop_signature*.

A "loop signature", contains the information which is relevant for the shift transformations. Loop signature is a list. The entries in this list describes the parameters and iteration variables of each statement covered by the loop/band node.

property schedule_tree: `list[Node]`

locate(*location: list[int]*)

Update the current node on the C/C++ side.

class tadashi.tadashilib.**Scops**(*app:* App)

Bases: `object`

All SCoPs which belong to a given file.

The object of type *Scops* is similar to a list.

get_input_path_bytes_and_backup_source()

Get the 'input' to *generate_code()* which is a copy of the current 'source'.

generate_code(*input_path=''*, *output_path=''*)

Generate the source code.

The transformations happen on the SCoPs (polyhedral representations), and to put that into code, this method needs to be called.

1.1.5 Module contents

Tadashi package root.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

t

INDEX

E

F

G

I

L

M

N

O

P

R

S

T

U

V