PSI Compiler
User's Guide

May 29, 2018

1 Introduction

The PSI Compiler is an array compiler that performs algebraic reductions on the expressions. The reductions eliminate unnecessary computation and temporary storage. Although the input language is not sufficient for general purpose programming, the goal of the project is to incorporate the compiler techniques into existing compilers. Currently, work is under way to translate subsets of FORTRAN90, High Performance FORTRAN, and SETL into our input grammar for input to our compiler. The mathematical techniques used by the compiler are described in the technical report "A Reduction Semantics for Array Expressions: The PSI Compiler", which is also in this distribution.

The compiler is currently available for the Sun, and SGI architectures. The target language of the compiler is ANSI C, so you will need to have an ANSI C compiler available to you. The files required for installation are contained in the *bin*, *include*, and *lib* directories in this archive. The files in each of these directories should be copied the desired location. In order for everything to work the directory containing the files from the bin directory must be in the PATH variable. The include and lib files need to be in the search path of your C compiler.

2 The Language

The compiler's input grammar is given by the following BNF description.

```
Moa Compiler 0.3 input grammar specification.

Uppercase words indicate a type of an object.

program := { procedure_definition }
procedure_definition := PROCEDURE_name "(" formal_parameter_list ")" block_body
formal_parameter_list := parameter_definition { "," parameter_definition }
parameter_definition := "int" PARAMETER_name |
"array" PARAMETER_name array_definition
array_definition := "^" INTEGER_number "<" { INTEGER_number |
INTEGER_PARAMETER_name } ">"
block_body := "{" definition_part statement_list "}"
definition_part := { constant_definition_part | variable_definition_part |
global_definition }
constant_definition_part := constant_definition ";" |
```

```
constant_scalar_definition ";"
constant_definition := "const" "array" VARIABLE_name array_definition "="
vector_constant
constant_scalar_definition := "const" "array" VARIABLE_name "^0 <>=" number
vector_constant := "<" { number } ">"
variable_definition_part := variable_definition ";"
variable_definition := array VARIABLE_name array_definition
global_definition := "global" VARIABLE_name ";"
statement_list := { statement ";" }
statement := assignment_statement | for_statement | allocate_statement |
procedure_call
procedure_call := PROCEDURE_name "(" actual_parameter_list ");"
actual_parameter_list := variable_access { "," variable_access }
allocate_statement := "allocate" identifier variable_access
for_statement := for "(" term "<=" variable_access "<" term ")" "{"
statement_list "}"
assignment_statement := variable_access "=" expression
variable_access := VARIABLE_name | PARAMETER_name
expression := factor { operator expression }
operator := "+" | "-" | "*" | "/" | "psi" | "take" | "drop" | "cat" | "pdrop" |
"ptake" | operator "omega" constant_vector
unary_operator := "iota" | "dim" | "shp" | "+ red" | "- red" | "* red" |
"/ red" | "tau" | "rav"
factor := term  | "(" expression ")" | unary_operator factor
term := variable_access | constant_vector
variable_access := identifier;
identifier := letter { letter | digit | '_' }
constant_vector := "<" { number } ">"
```

2.1 Procedure Definitions

An input file consists of one or more procedure definitions. The following is an example of a procedure definition with an empty statement body.

```
test(int n, array A^2 <n n>)

{
}
```

Each procedure definition consists of the procedure name and argument list and a procedure body. The argument list is a list of declarations separated by commas. Argument declarations may be either a integer or an array declaration. The procedure body contains a definition part that contains object declarations and a statement part.

Integer arguments are declared with the keyword “int” followed by the name of the argument. Integer arguments may only be used in inline vectors and are provided for the purpose of indicating the size of an array or looping parameters. An integer arguments are not intended for computation. The array argument declaration consists of the keyword “array” followed by the argument name, the dimension, and the shape (size). The dimension is specified by a “^” symbol followed by an integer. The shape is an inline vector that may include integers and previous integer arguments. The inline vector is a whitespace separated list of elements included in angle brackets.

2.2 The Definition Part

The definition part of a procedure contains declarations of objects and object property definitions. The only type of object that can be declared is an array object. The array object definition declares the name of the object and defines the structure of the array by it's dimension and shape (size). An example array declaration is

```
array A^2 <4 3>;
```

This declaration declares the array A that has 2 dimensions with 4 rows and 3 columns. An array can be declared with a dynamic shape in two ways, with an integer parameter or a delayed binding. If n is an integer parameter to the enclosing procedure than it can be used in the shape of a declaration giving the array a dynamic shape. The second method can be accomplished be leaving the shape part of the declaration empty. For example

```
array A^3;
```

declares and array with a delayed shape binding. The shape of the array must be bound to the array in the statement part of the procedure before it is used. The "allocate" statement described in statement section is used for delayed binding.

A constant array may be declared and initialized by preceding the "array" keyword with the "const" keyword and including initial data after the shape part. In this case the declaration must contain a shape part and may not reference integer arguments. The initial data is defined by an "=" symbol followed by an inline vector containing the components of the array in row major order. For example

```
const array A^2 <2 2>=<3 1 2 4>;
```

declares the constant array

$$A \equiv \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

If the target architecture is a multi-processor architecture (indicated by the "arch" directive) then the definition part may contain property commands. Currently there is only one, the global property. This property indicates that an array is to be entirely stored on each processor (i.e. not distributed). By default an array will be distributed if the target architecture is a multi-processor architecture. The global property is indicated by the keyword "global" followed by the array name and a ";". For example

```
global A;
```

In the future property statements will allow the specification of an arrays distribution (e.g. by rows or columns, block or cyclic).

2.3 The Statement Part

There are three types of statements, assignments, control flow, and allocation statements.

There is one allocation statement that is used to specify the delayed shape binding of and array. The statements consists of the keyword "allocate" followed by the name of the array to bind the shape to, the shape, and a ";". The shape can be specified by an inline vector or an array variable of dimension one.

The two control flow statements are the procedure call statement and the for statement. The procedure call statement allows a call to procedure defined in the current file or any other C file that you will later link together. The heat equation implementation in the examples directory show how this is used to call a procedure written in C to print out the results of the computation. The syntax is the name of the procedure followed by the list of actual parameters in parentheses and ";". The actual parameters is restricted to named variables. For example

```
printout(A);
```

The for statement allows the use of iteration. The syntax of this command is

```
for ( vector1 <= array_name < vector2 ) {
statement list
}
```

where vector1 and vector2 are either an inline array or a one dimensional array variable, array_name is a one dimensional array variable, and statement list is a list of statements. This construct operates by executing the statement body repeatively. The vector denoted by array_name is initially equal to vector1 and is incremented at each iteration until the vector is no longer bounded by vector2. For example

```
for (<0 0> <= t < <3 2>) {
  A=B;
}
```

would execute $A = B$ 6 times with t successively equal to $< 00 >, < 01 >, < 10 >, < 11 >, < 20 >$, and $< 21 >$.

The assignment statements consist of array expressions written in MOA notion (see technical report mention in introduction). This is discussed in the Array Expressions section.

2.4 Array Expressions

3 Using the Compiler