FASTER – “Need for speed”

Language Reference Manual

Anand Rajan asr2171@columbia.edu

July 2nd 2014

Introduction:

The document describes the FASTER, which is an approach that simplifies parallel programming that is based on OpenMP frameworks. Its main forte is simplicity which the programmer of the language can utilize these features in the application.

2. Lexical conventions

1.1 Identifiers

An identifier consists of a letter followed by alphanumeric characters, the non alphanumeric characters are allowed as identifier are underscores

2.2

Keywords:

- int
- Boolean
- char
- string
- true
- false
- pmap
- pdistmap
- null

2.3

Comments:

The comments will be similar to C#/Java/C++ in

```
/* this is a multiline comment
  * */
  // this is a single line comment

  /* this is a invalid comment
        * */
                      * */
```

3.

4. Types

3.1 Fundamental Types

In FASTER the following fundamental data types are supported:

- Int : integer variable
- Double : floating point variable
- Boolean: Boolean variable
- Char: single ASCII character
- String: multiple ASCII characters
- Pmap<T>: A parallel map which symbolizes that the function that will be used with it will be used using OpenMP (i.e. in multiple cores).
- Void – Void is a fundamental type that symbolizes that function does not return anything

In addition FASTER uses void to symbolize the function does not have a return `value.`

3.2 User Defined Types

The user can create their user defined types, types like an Employee, which are like java/ c++/c# classes in that they contain fundamental types in them. They do not however contain methods.

```
class Employee
    {
        int age;
        int pay;
    }
```

One major difference is there is no public/private identifiers everything is assumed to be public

a. Storage Classes:

The scope of the fundamental type can be local, that is restricted to the function or global that is restricted to the stack of the entire program. The scope is derived from where it has been defined. If it is defined inside the function the scope is local otherwise the fundamental type is global.

5. Expressions

4.1 Operators

The table lists the following operators that are supported by FASTER.

<table>
<tr><th>Operator</th><th>Description</th></tr>
<tr><td>+</td><td>Addition</td></tr>
<tr><td>-</td><td>Subtraction</td></tr>
<tr><td>*</td><td>Multiply</td></tr>
<tr><td>/</td><td>Divide</td></tr>
<tr><td><</td><td>Less than comparison</td></tr>
<tr><td>></td><td>Greater than comparison</td></tr>
<tr><td><=</td><td>Less than equal to comparison</td></tr>
<tr><td>>=</td><td>Greater than equal to comparison</td></tr>
<tr><td>==</td><td>Equality comparison</td></tr>
<tr><td>!=</td><td>Inequality comparison</td></tr>
<tr><td>%</td><td>Modulus operator</td></tr>
<tr><td>&&</td><td>And operator</td></tr>
<tr><td>||</td><td>Or operator</td></tr>
<tr><td>=</td><td>Assignment operator</td></tr>
<tr><td>,</td><td>Argument separator</td></tr>
<tr><td>;</td><td>Statement separator</td></tr>
<tr><td>A[i]</td><td>Element of the list operator</td></tr>
</table>

4.2 Precedence

It is ordered from highest to lowest.

a. Function calls, array subscript operations
b. Negation operator
c. Multiplication, Division, Modulus
d. Addition, Subtraction
e. Greater than, Less than, Greater than equal to, Less than equal to
f. Equality and inequality operators
g. And operator
h. Or operator
i. Assignment
j. Argument operator

7. Syntax

5.1 Statements

A statement can have different formats:

a. Expression
b. {Statement list}
c. If (Expression) Statement else statement
d. For (expression; expression; expression)
e. Return expression;

5.2 Variable Declaration

The variable declaration syntax is as follows:

Type identifier;

Type identifier = value; // Assigning to a default value.

Data Type	Default Value
Int	0
Double	0.0
Boolean	False
Char	'\0'
String	empty

5.3 Array Declaration

The array declaration in FASTER has the format of this

Type[size] identifier

The indexing of the array starts at zero

5.4 Function Declaration

The syntax for the function has the following format.

Return-type identifier(arg-type arg1, arg-type arg2,…………, arg-type argN)

{

Declaration list

Statement list

}

The return-type of type of function can be any fundamental type it can also be any user defined type including void. The arg-type can be any fundamental or user defined type except void.

6 Parallel Constructs

6.1 PMap<T>

This is one of the parallel constructs that will be created in the FASTER language. It will use a similar syntax to generics and you can define a fundamental or user defined type in its syntax so you could have PMap<Employee> or PMap<String>. One would invoke it by creating a method that has the enclosed type as its signature.

So an example declaration would be:

```
PMap<Employee> employees;

employees.ProcessEmployee


ProcessEmployee(Employee e)

{

    e.Age = e.Age * 10 + 12 /13 * 6

}
```

ProcessEmployee in this case would be executed in parallel loop. The number of threads that would be used in OpenMP could depend on the runtime or could be equal to the number of processors in the users runtime environment.

7. Miscellaneous

- See if OpenMPI is also possible given time constraints this will be implemented.