FASTER Project Proposal

Anand Rajan (asr2171)

1. Description:

   This is the decade of parallelism. FASTER is a programming language the focuses on the two different aspects of parallelism, parallel cores and parallel distributed systems. The main focus in terms of parallelism would be accepting a collection argument i.e. split distribute the number of elements between cores and/or then distribute the number of elements into different slave machines. The focus of FASTER will be on "embarrassing parallel programs" (brute force crypto attacks or prime factorization for example). To parallelize cores the language will spawn threads internally and to parallelize between different machines, there will be a master slave model, where the master would complete the "map" step, and then divide stuff into different nodes. FASTER would make it dead simple and cleaner for the users to use parallelism and distribution.

   Another purpose of mine with this is to reacquaint myself with C++ by potentially using the OpenMP (for parallel cores) and OpenMPI (used for distributed computing) frameworks (if allowed). If time permits (time never permits as Professor says!!) there will also be exploration of splitting these "embarrassing parallel" tasks between the CPU and GPU (using NVidia's CUDA). In terms of languages I am excited about the functional use of Ocaml for the compiler and the runtime use of C++ and with playing around with the above mentioned frameworks.

2. Syntax:
   a. **Comments:** Use the standard java/c#/c++ comments

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

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

   b. **Strings**: String will be enclosed by the double quote character and backslash will be used for escaping any double quotes for example.

   c. **Data types**: There will be only three data types that are defined, ints chars, bool and strings. A char would be a byte long, A bool would be a byte long, Int's would be 8 bytes long and Strings maximum length would

   c. Classes/Objects: Classes will be defined by the traditional class structure. Pretty similar to Java/C++ except that no public and private keywords and also like mentioned only ints and strings as data types. Variables in objects are accessed by the "." Notation.

```
class Employee{

int age;
```

```
string name;

int pay;

}
```

d. **Collections**: there will be no arrays, the only collection set that would be available would be ClassName[] or primitivetype[]. And access to the n the variable list[n] similar to Java etc.

e. **Methods**: Methods are supported with all parameters passed by value (except classes where the actual pointer value is copied). Methods do not return a type instead the type is inferred by the compiler/runtime system.

f. **Control blocks**

```
if (true)
       {
       }
else if
 {
 }
 else
 {
 }
```

g. **Loops**

```
// Determine the number of cores and create that many threads automatically
// work will be a method that takes one element of list and then one can process
this.
PCoreFor(list,work)
```

// Set up slave machines.

RegSlave("ipAddress")

RegSlave("ipAddress")

```
// Divide loop up linearly and send to slaves and the work will take in one
element of the list and the one can process this.

PDistribututeFor(list, work)
```

h. **Memory management** – memory is assumed to be unlimited

i. **Operators** - +, *, /, >, <, =, %, == are supported.
j. **Utility** – Utility methods that are available are print()

Sample code:

```
 Main()
 {
     Test("Go");
 }

 class Employee
 {
```

```
    int age;
    int pay;
}

ProcessEmp(Employee emp)
{
    return emp.age * 9181232 / 232342 * 12 % 982332;
}
Test(String t)
{
    List<Employee> employees;
    results = PCoreFor(employees,ProcessEmp);
    print(results);

    results = PDistFor(employees,ProcessEmp);
    print(results);

    results = PSplitGPUCPU(employees,ProcessEmp);
    print(results);

}
```