A brief introduction to Chapel

20 June 2016

Introduction

”Why doesn’t HPC programming have an equivalent to Python / Matlab / Java ?”

Chapel (Cascade High Productivity Language) wants to be that language!

Chapel is a programming language developed by Cray for being ’a Productive Parallel Programming Language’

`http://chapel.cray.com`

Chapel is:

- Created for parallel programming
  - Hides the implementation details
  - Runs on from multicore desktops and laptops to high-end supercomputers
- Portable
  - Any platform with a C/C++ compiler, pthread and *NIX environment
- Free and open source
  - Apache 2.0, hosted on Github
    - `https://github.com/chapel-lang/chapel`
- Native
  - The application is compiled into binary code


```
sync begin {
        writeln("Hello");
}
```

```
class Hello extends Thread{
public void run(){
  System.out.println("Hello");
}
}

public class MainClass{
public static void main
(String args[]){
try{
  Hello ht = new Hello();
  ht.start();
  ht.join();
}
catch (Exception e){
  e.printStackTrace();
}}}
```

A little example

The language presents a syntax based on C and Java:

```
proc fact(i : int): int{
  if(i == 1) then 1;
  else return i*fact(i-1);
}

proc main(){
  var x: int = 10;
  var y = fact(5);
  writeln(x + y);
}
```

We'll concentrate on how Chapel handles parallel programming

Task

A task is an unit of execution
It's not important knowing how a task is implemented. It could be:

- A physical processor
- A thread
- A process

The developer only has to say which code should be executed on how many tasks

begin

begin: Creates a new task. Tasks are asynchronous by default

```
begin {writeln("Hello");}
begin {writeln("World");}
```

sync: Makes the task(s) synchronous

```
sync begin {writeln("Hello");}
begin {writeln("World");}

sync {
  begin {writeln("Hello");}
  begin {writeln("World");}
}
writeln("again");
```


Domains

Chapel defines a **domain** as a set of indexes

```
var Z5 = {0 .. 4};      //Z5
var N = 0..;            //Range with all numbers > 0
var intersect = Z5[N]; //Intersection between Z5 and N

//Domains can be multidimensional
var Z = {0 .. 10, 0 .. 10};
var sindex = {"a", "b", "c"};

//This is a map from sindex to integer
var M: [sindex] int;
M("a") = 10;
//This is a function from sindex to Z5
proc F(a : sindex) : Z5 { ... }
```

coforall

coforall: Starts a new task for each element of a domain

```
var Z5 = {0 .. 4};

coforall i in Z5 {
  writeln("Sono ",i);
}
```

The coforall tasks are all synced with their creator

Example: summation

```
proc summation(M){
  var n = M.numElements;

  for j in 1 .. log2(n) do {
    coforall k in 1 .. n/(2**j) {
      var ind = (2**j) * k;
      M[ind] = M[ind] + M[ind - (2**(j-1))];
    }
  }
  return M[n];
}
```


Scan and Reduce

Since a lot of parallel algorithms can be implemented with binary associative and prefix operators, Chapel provides both as key concepts in the language

```
//M = [1 1 1 1 1]
//Summation
var tot = + reduce M;
//tot = 5
//Prefix sum
var prefix = + scan M;
//prefix = [1 2 3 4 5]
```


Locales

A **Locale** is a set of tasks that possess uniform access to some resources (like a memory area)
Some examples of locales:

- CPU
- GPU
- ASIC

Using Locales we can implement a parallel architecture with distributed memory. Our system will be a network of Locales

Computation distribution

The runtime handles the communication between Locales. For the developer a Locale is an object of type Locales

on: Execute a block of instructions on a specific Locale(s)

```
var loc = Locales[2];
on loc {
    writeln("Hello from ", here.id);
}
```


Data distribution

We can distribute subsets of a domain into several Locales

```
use CyclicDist;
var dom = {0..numLocales-1} dmapped Cyclic(startIdx=0);
var M:[dom] int;

on Locales[1]{
  M[2] = 10;
}

for k in dom do
  writeln("M[",k,"] = ",M[k]," in locale ",here.id);
```

Chapel provides some data distribution policy by default but the developer can write their own

Example: multilocal sorting

```
var dom = {1..numLocales} dmapped Cyclic(startIdx=1);
var M: [dom] int;

proc MINMAX(M,i,j){
  if(M[i] > M[j]) then M[i] <=> M[j];
}

for i in dom {
  for k in 1..numLocales-1 by 2 do
    on Locales[k] { MINMAX(M, k, k+1); }
  for k in 2..numLocales-1 by 2 do
    on Locales[k] { MINMAX(M, k, k+1); }
}
```


Conclusions

Chapel is a young language and it has some little problems:

- Currently available only from source code
- Object orientation is supported but presents some problems
- No exception handling
- Performances are good, but not yet optimal
- It can be complex to setup the runtime

However, the language is currently in an usable state

- Principally used by early adopters and researchers at the moment

Conclusions

`http://chapel.cray.com/learning.html`

- There are a lot of guides and exercises for learning the language
- Current version 1.13 (April 2016)
- Each new main release (every 6 months) brings big improvements in both performances and supported features
- Chapel is free and open source, every contribution is welcomed!