

# A Brief History of Project Fortress

Eric Allen

Two Sigma Investments, LLC

*eric.allen@twosigma.com*

April 22, 2015

# The DARPA HPCS Project

- In 2003, The United States determined to retake the lead in high performance computing
- HPCS: High Performance/Productivity Computer Systems
- Participants charged with rethinking computing from the ground up at the ‘peta scale’ :
  - quadrillions of operations per second
  - quadrillions of bytes in memory
- Rethinking chip design, communication, operating systems, languages and programming models at this scale
- Three participants: IBM, Cray, Sun

# Language Design at the Petascale

- Pervasive parallelism must be at the heart of computation at this scale
- Programmer productivity critical
  - Software development at the national labs has been dominated by the time to design and implement a solution
- Participants were charged with aiming for a 10x improvement in productivity

# Project Fortress: Sun's Approach to a Scientific Programming Language

- Fortress Design Philosophy:
  - Start with a fresh design and first see what productivity improvements we might achieve in that context
    - Integration with legacy languages could be dealt with later
  - Make the code look as much as possible like the specification (seriously)
    - Mathematical notation as concrete syntax
  - Make things parallel by default
  - 'Growing a Language' (Steele, OOPSLA 1998)
- Many, many participants (Sun employees, interns, academic researchers, and community members) contributed significantly to Fortress, over nearly a decade

Example Map/Reduce in Fortress:

$$\pi = 4 \left( \sum_{1 \leftarrow 1: \text{trials}} \text{if } \text{random}()^2 + \text{random}()^2 \leq 1 \text{ then } 1 \text{ else } 0 \right) / \text{trials}$$

# Mathematical Notation as Concrete Syntax

Example Map/Reduce in Fortress:

$$\pi = 4 \left( \sum_{1 \leftarrow 1: \text{trials}} \text{if } \text{random}()^2 + \text{random}()^2 \leq 1 \text{ then } 1 \text{ else } 0 \right) / \text{trials}$$

Equivalent to the following code in Apache Spark:

```
val count = sc.parallelize(1 to NUM_TRIALS).map{i =>
  val x = java.util.concurrent.ThreadLocalRandom.nextDouble(1)
  val y = java.util.concurrent.ThreadLocalRandom.nextDouble(1)
  if (x*x + y*y <= 1) 1 else 0
}.reduce(_ + _)
val result = (4 * count) / NUM_TRIALS
```

# Mathematical Notation as Concrete Syntax

Example Map/Reduce in Fortress:

$$\pi = 4 \left( \sum_{1 \leftarrow 1: \text{trials}} \text{if } \text{random}()^2 + \text{random}()^2 \leq 1 \text{ then } 1 \text{ else } 0 \right) / \text{trials}$$

How is this entered at a keyboard?

```
pi = 4 (SUM[1 <- 1 : trials]
        if random()^2 + random()^2 <= 1 then 1 else 0)
/ trials
```

In fact, that is what was typed on this slide to produce the rendered version of the code (using standard Fortress tools for preprocessing  $\text{\LaTeX}$ )

# Static Checking of Physical Units and Dimensions

*dimension* Velocity = Distance/Time

*dimension* Acceleration = Velocity/Time

*dimension* Force = Mass Acceleration

$$g = 9.81 \frac{\text{m}}{\text{s}^2}$$

$v(t: \mathbb{R}^{64} \text{ Time}, v_0: \mathbb{R}^{64} \text{ Velocity}): \mathbb{R}^{64} \text{ Velocity} = -(g t) + v_0$

$y(t: \mathbb{R}^{64} \text{ Time}, v_0: \mathbb{R}^{64} \text{ Velocity}, y_0: \mathbb{R}^{64} \text{ Distance}): \mathbb{R}^{64} \text{ Distance} =$   
 $-\frac{1}{2} g t^2 + v_0 t + y_0$

$$y \left( 3.14 \text{ s}, 2.718 \frac{\text{km}}{\text{s}}, .57721 \text{ km} \right)$$



# Operator Overloading

Operators can be overloaded in libraries, including prefix, postfix, infix, and ‘enclosing operators’ (various kinds of brackets)

$$\text{opr } (n: \mathbb{Z}_{64})! = \prod_{i \leftarrow 1:n} i$$

# User-Extensible Concrete Syntax

grammar ForLoop extends { Expression, Identifier }

Expr ::=

for {  $i$ : Id  $\leftarrow$  e: Expr, ?Space}\* do *block*: Expr end  $\Rightarrow$   
 $\langle$  for<sub>2</sub> i \* \*; e \* \*; do *block*; end  $\rangle$

| for<sub>2</sub> i: Id\*; e: Expr\*; do *block*: Expr; end  $\Rightarrow$

case *i* of

Empty  $\Rightarrow$

$\langle$  *block*  $\rangle$

Cons(*ia*, *ib*)  $\Rightarrow$

case *e* of

Empty  $\Rightarrow$   $\langle$  throw Unreachable  $\rangle$

Cons(*ea*, *eb*)  $\Rightarrow$

$\langle$  ((*ea*).loop(fn *ia*  $\Rightarrow$  (for<sub>2</sub> *ib* \* \*; *ed* \* \*;  
do *block*; end))))  $\rangle$

end

# Symmetric Multiple Dynamic Dispatch

*isContinuous*( $f$ : Function[[ $\mathbb{R}^{64}$ ,  $\mathbb{R}^{64}$ ]])

*isContinuous*( $f$ : Function[[ $\mathbb{R}^{64}$ ,  $\mathbb{R}^{64}$ ]],  $g$ : Function[[ $\mathbb{R}^{64}$ ,  $\mathbb{R}^{64}$ ]])

*isContinuous*( $f$ : Function[[( $\mathbb{R}^{64}$ ,  $\mathbb{R}^{64}$ ),  $\mathbb{R}^{64}$ ]])

*isContinuous*( $f$ : Function[[Graph, Graph]])

*isContinuous*( $f$ : Function[[Graph, Graph]],  $g$ : Function[[Graph, Graph]])

*isContinuous*( $f$ : Function[[ $\mathbb{R}^{64}$ ,  $\mathbb{R}^{64}$ ]],  $g$ : Function[[Graph, Graph]])

*isContinuous*[[ $S$ : MetricSpace]]( $f$ : Function[[ $S$ ,  $S$ ]])

*isContinuous*[[ $T$ : TopologicalSpace]]( $f$ : Function[[ $T$ ,  $T$ ]])

*isContinuous*[[ $S$ : TopologicalSpace,  $T$ : TopologicalSpace]]  
( $f$ : Function[[ $S$ ,  $T$ ]])

# Parallel by Default

- Make it difficult for programmers to avoid parallelism.
- A tuple expression (including the arguments to a function) is equivalent to an HJ finish with asyncs:

$(e_1, e_2, e_3, e_4)$  is equivalent to:

```
finish {  
  async e1;  
  async e2;  
  async e3;  
  async e4;  
}
```

By making parallelism pervasive, programmers are subtly encouraged to avoid side effects in code whenever possible, to prevent race conditions.

# Parallel by Default

- For loops are parallel by default
- Maps and reductions are parallel by default
- Variables written to but not read within for loops are implicit accumulators

$\{x^2 \mid x \leftarrow 1 : trials\}$

$$\sum_{i \leftarrow 1 : trials} i^2 + 1$$

```
for  $i \leftarrow 1 : trials$  do
   $result \ += i^2 + 1$ 
end
```

All of the above are *desugared* into calls to *generators* and *reductions*: objects defined in libraries that act somewhat like map and reduce operations.

# Atomic Blocks

An atomic block in Fortress is equivalent to an unqualified isolated block in HJ:

```
atomic do
   $f(x)$ 
end
```

- Atomic blocks can be aborted explicitly with the *abort()* command
- There is also `tryatomic`

# Spawn and Regions

`spawn` is a lot like HJ's *async*

```
spawn do
  f(x)
end
```

Tasks can be spawned at particular *regions*:

```
spawn at a.region(d) do
  f(x)
end
```

This should look quite familiar!

# Evolution of HPC During Fortress

- When HPCS started, the focus was on scientific computing at national labs
- The advent of multicore architectures made parallelism pervasive
- The advent of big data dramatically increased the user base for cluster computing



# Where is Fortress Now?

- A research compiler was implemented for multicore computing using an early version of the Java workstealing library
- The specification and all code is available under a BSD license
- Sun/Oracle wrapped up work on Fortress in 2012
- Many open research problems were solved as part of the project

*'And finally, when the project is at its end, carefully reassess it, recognize that many aspects could be improved, and do it all over again.'*

Nicholas Wirth, On The Design of Programming Languages. 1974.