# A Brief History of Project Fortress

Eric Allen

Two Sigma Investments, LLC

*eric.allen@twosigma.com*

May 8, 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
  - Why Sun? Proximity Interchip Communication (Drost and Sutherland)

# 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 signficantly to Fortress, over nearly a decade

# Mathematical Notation as Concrete Syntax

Example Map/Reduce in Fortress:

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

# Mathematical Notation as Concrete Syntax

Example Map/Reduce in Fortress:

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

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:

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

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 LaTeX)

# Static Checking of Physical Units and Dimensions

$$\textit{dimension}\,\mathrm{Velocity} = \mathrm{Distance}/\mathrm{Time}$$

$$\textit{dimension}\,\mathrm{Acceleration} = \mathrm{Velocity}/\mathrm{Time}$$

$$\textit{dimension}\,\mathrm{Force} = \mathrm{Mass}\,\mathrm{Acceleration}$$

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

$$v(t\colon \mathbb{R}64\,\mathrm{Time}, v_0\colon \mathbb{R}64\,\mathrm{Velocity})\colon \mathbb{R}64\,\mathrm{Velocity} = -(g\,t) + v_0$$

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

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

# Operator Overloading

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

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

# User-Extensible Concrete Syntax

```
grammar ForLoop extends { Expression, Identifier }
Expr |:=
  for {i: Id ← e: Expr, ?Space}* do block: Expr end ⇒
  ⟨ for2 i * *; e * *; do block; end ⟩
| for2 i: Id*; e: Expr*; do block: Expr; end ⇒
 case i of
   Empty ⇒
     ⟨block⟩
   Cons(ia, ib) ⇒
     case e of
       Empty ⇒ ⟨throw Unreachable⟩
       Cons(ea, eb) ⇒
         ⟨((ea).loop(fn ia ⇒ (for2 ib * *; ed * *;
                              do block; end)))⟩
   end
```

# Modular Symmetric Multiple Dynamic Dispatch

$\texttt{opr}\ +(m\colon \mathbb{Z}64, n\colon \mathbb{Z}64)$

$\texttt{opr}\ +(q\colon \mathbb{Q}64, v\colon \mathbb{Q}64)$

$\texttt{opr}\ +(x\colon \mathbb{R}64, y\colon \mathbb{R}64)$

$\texttt{opr}\ +[\![\texttt{nat}\ n]\!](v\colon \mathbb{R}64^n, w\colon \mathbb{R}64^n)$

$\texttt{opr}\ +[\![\texttt{nat}\ m, \texttt{nat}\ n]\!](M\colon \mathbb{R}64^{m\times n}, N\colon \mathbb{R}64^{m\times n})$

$\texttt{opr}\ +[\![\texttt{nat}\ m]\!](x\colon \mathbb{R}64, m\colon \mathbb{R}64^n)$

*And then there is addition on measurements, tensors, vector spaces, algebras, graphs, topological spaces, etc., etc.*

# 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 ← 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
```

# do and also do

```
do
    v := a_i
also at a.region(j) do
    w := a_j
end
```

# Generators

$$\sum_{x \leftarrow 1 \# 100} (3x + 2)$$

```
object SumZZ64 extends Reduction[ℤ64]
    empty(): ℤ64 = 0
    join(a: ℤ64, b: ℤ64) = a + b
end
```

$$z = (1 \# 100).generate[\mathbb{Z}64](\mathrm{SumZZ64}, \texttt{fn}\ (x) \Rightarrow 3x + 2)$$

# 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

# Fortress and Future Language Design

*'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.