

What's Cool about Fortress

Guy Steele

Sun Fellow

Sun Microsystems Laboratories

May 2008

Project Fortress

- Programming language for scientific applications
- Begun at Sun Labs in 2003 as part of US DARPA program for High Productivity Computing Systems (HPCS)
 - > Goal: improve programmer productivity for 2010 and beyond
- Now an operational open-source project with international participation
 - > <http://projectfortress.sun.com>
- Freely available language specification
 - > <http://research.sun.com/projects/plrg>

“Do for Fortran What Java™ Did for C”

- Catch “stupid mistakes” (like array bounds errors)
- Extensive libraries (e.g., for network environment)
- Security model (including type safety)
- Dynamic compilation
- Platform independence
- Multithreading
- Make programmers more productive

Key Ideas

- Ease use of parallelism
- Don't build the language—grow it
- Make programming notation closer to math

A Parallel Language

High productivity for multicore, SMP, and cluster computing

- Hard to write a program that isn't potentially parallel
- Support for parallelism at several levels
 - > Expressions
 - > Loops, reductions, and comprehensions
 - > Parallel code regions
 - > Explicit multithreading
- Shared global address space model with shared data
- Thread synchronization through atomic blocks and transactional memory

These Are All Potentially Parallel

$f(a) + g(b)$

$$s = \sum_{k \leftarrow 1:n} c_k x^k$$

do

$f(a)$

also do

$g(b)$

end

$L = \langle find(k, x) \mid k \leftarrow 1:n, x \leftarrow A \rangle$

for $k \leftarrow 1:n$ do

$a_k := b_k$

$sum += c_k x^k$

end

do

$T_1 = \text{spawn } f(a)$

$T_2 = \text{spawn } g(b)$

$T_1.wait(); T_2.wait()$

end

Growing a Language

- Languages have gotten much bigger.
- You can't build one all at once.
- Therefore it must grow over time.
- What happens if you design it to grow?
- How does the need to grow affect the design?
- Need to grow a user community, too.
- Can technical features assist community growth?

See Steele, “Growing a Language” keynote talk, ACM OOPSLA 1998;
also in *Higher-Order and Symbolic Computation* **12**, 221–236 (1999)

Fortress Language Design Strategy

Wherever possible,
consider whether a proposed language feature
can be provided by a library
rather than having it built into the compiler.

To make this work, library designers need
substantial control over syntax and semantics—
not just the ability to code new functions or methods
invoked by a single method call syntax `foo(a,b,c)`.

Minimalist Approach

- As few primitive types as possible (cf. Bacon's Kava)
 - > Objects with methods and fields
 - > Binary words of many different sizes
 - > Linear sequences (fixed length known at compile time)
 - > Heap sequences (fixed length known at allocation time)
- User-defined parameterized types
- User-defined polymorphic (overloaded) operators
- Aggressive type inference to reduce clutter
 - > Many variables require no type declarations
- Aggressive static and dynamic optimization

Designed to Grow

Technical design supports growth by an open-source community.

- Emphasis on replaceable components with multiple versions
- Language extensibility
 - > Parametric polymorphism with multiple inheritance
 - > Overloading of functions, methods, and operators
 - > User-defined syntactic extensions
- Plenty of room for experimentation
- Language encourages unit testing and explicit descriptions of code invariants and properties
 - > Design-by-contract
 - > Test-driven development

Mathematical Syntax 1

Integrated mathematical and object-oriented notation

- Supports a stylistic spectrum that runs from Fortran to Java™—and sticks out at both ends!
 - > More mathematical than Fortran
 - Compare $a*x**2+b*x+c$ and $a x^2 + b x + c$
 - > More object-oriented than Java
 - Multiple inheritance
 - Numbers, booleans, and characters are objects
 - > To find the size of a set S : either $|S|$ or $S.size$

Mathematical Syntax 2

- Full Unicode character set available for use, including mathematical operators and Greek letters:

\times	\div	\oplus	\ominus	\otimes	\oslash	\odot	\approx	α	β	γ	δ
\boxplus	\boxminus	\boxtimes	\leftrightarrow	\wedge	\vee	\equiv	\neq	ϵ	ζ	η	θ
\leq	\geq	Σ	Π	\rceil	\lrcorner	\rceil	\rceil	ι	κ	λ	μ
\cap	\cup	\oplus	\subset	\subseteq	\supseteq	\supset	\in	ξ	π	ρ	σ
\sqcap	\sqcup	\sqsubset	\sqsubseteq	\sqsupseteq	\sqsupset	\neg	\notin	ϕ	χ	ψ	ω
\lfloor	\rfloor	\lceil	\rceil	\langle	\rangle	λ	γ	Γ	Θ	and so on	

- Use of “funny characters” is under the control of libraries (and therefore users)

Conventional Mathematical Notation

- The language of mathematics is centuries old, concise, convenient, and widely taught
- Early programming languages were constrained by keyboards and printers
- Experiments in the 1960s were not portable
- Now we have bitmap displays and Unicode
- Still, parsing mathematical notation is a challenge
 - > Subtle reliance on whitespace: $\{ |x| \mid x \leftarrow S, 3 \mid x \}$
 - > Semantic conventions: $y = 3x \sin x \cos 2x \log \log x$
- Programming language tradition has contributions
- Type theory, block structure, variable scope

What Expression Syntax Is Built In?

- Parentheses () for grouping and tuples
- White brackets [] delimit static type information
- Comma , to separate expressions in tuples (a, b, c)
- Semicolon ; to separate statements on a line
- Dot . for field and method selection
- Conservative, traditional rules of precedence
 - > Partial order, not total order; not always transitive
 - > Example: $A + B > C$ is okay, as is $B > C \vee D > E$;
but $A + B \vee C$ needs parentheses

What Syntax Is Programmable?

- Juxtaposition $A B$ is a binary operator
 - > If first item is a function, it's function application
 - > All other cases are programmable by libraries
 - Numeric multiplication
 - Scalar multiplication of vectors and matrices
 - Matrix-vector and matrix-matrix products
 - String concatenation? Group/monoid multiplication?
- Any other operator can be infix, prefix, and/or postfix
 - > Libraries provide operator definitions
- Many sets of brackets $[] \langle \rangle \{ \} \lceil \rceil \lfloor \rfloor \lceil \rceil \dots$
 - > Libraries provide bracket definitions

Libraries Define ...

- Which operators have definitions and what types they accept

opr $-(m: \mathbb{Z}, n: \mathbb{Z}) = m.subtract(n)$

opr $-(m: \mathbb{Z}) = m.negate()$

opr $(n: \mathbb{Z})! = \text{if } n = 0 \text{ then } 1 \text{ else } n(n - 1)! \text{ end}$

- Whether a juxtaposition is meaningful

opr juxtaposition $(m: \mathbb{Z}, n: \mathbb{Z}) = m.times(n)$

opr juxtaposition $(a: \text{String}, b: \text{String}) = a.concat(b)$

- What bracketing operators actually mean

opr $\lceil x: \mathbb{R} \rceil : \mathbb{Z} = ceiling(x)$

opr $|x: \mathbb{R}| : \mathbb{R} = \text{if } x < 0 \text{ then } -x \text{ else } x \text{ end}$

opr $|\llbracket T \rrbracket s: \text{Set} \llbracket T \rrbracket| : \mathbb{N} = s.size$

But Wasn't Operator Overloading a **Disaster** in C++?

- Yes, it was
 - > Not enough operators to go around
 - > Failure to stick to traditional meanings
- We have also been tempted and had to resist
- We believe “Unicode + discipline” can avert disaster
- We see benefits in using notations for programming that are also used for specification

NAS CG Benchmark: NPB 1 Spec

$$z = 0$$

$$r = x$$

$$\rho = r^T r$$

$$p = r$$

DO $i = 1, 25$

$$q = A p$$

$$\alpha = \rho / (p^T q)$$

$$z = z + \alpha p$$

$$\rho_0 = \rho$$

$$r = r - \alpha q$$

$$\rho = r^T r$$

$$\beta = \rho / \rho_0$$

$$p = r + \beta p$$

ENDDO

compute residual norm explicitly: $\|r\| = \|x - A z\|$

$$z: \text{Vector}[\mathbb{R}] := 0$$

$$r: \text{Vector}[\mathbb{R}] := x$$

$$p: \text{Vector}[\mathbb{R}] := r$$

$$\rho: \mathbb{R} := r^T r$$

for $j \leftarrow \text{seq}(1:25)$ **do**

$$q = A p$$

$$\alpha = \rho / p^T q$$

$$z := z + \alpha p$$

$$r := r - \alpha q$$

$$\rho_0 = \rho$$

$$\rho := r^T r$$

$$\beta = \rho / \rho_0$$

$$p := r + \beta p$$

end

$(z, \|x - A z\|)$

Types Defined by Libraries

- Lists, vectors, sets, multisets, and maps
- Like C++ Standard Template Library, but better notation

$\langle 1, 2, 4, 3, 4 \rangle$

$A \cup \{1, 2, 3, 4\}$

$[3 \ 4 \ 5] \times [1 \ 0 \ 0]$

- Matrices and multidimensional arrays
- Integers, floats, rationals, with physical units

$m: \mathbb{R} \text{ Mass} = 3.7 \text{ kg}$

$\mathbf{v}: \mathbb{R}^3 \text{ Velocity} = [3.5 \ 0 \ 1] \text{ m/s}$

$\mathbf{p}: \mathbb{R}^3 \text{ Momentum} = m \mathbf{v}$

- Data structures may be local or distributed

ASCII “Wiki-like Markup” Notation

- Lists, vectors, sets, multisets, and maps
- Like C++ Standard Template Library, but better notation

`<| 1,2,4,3,4 |>`

`A UNION {1,2,3,4}`

`[3 4 5] TIMES [1 0 0]`

- Matrices and multidimensional arrays
- Integers, floats, rationals, with physical units

`m: RR Mass = 3.7 kg_`

`_v: RR^3 Velocity = [3.5 0 1] m_/s_`

`_p: RR^3 Momentum = m _v`

- Data structures may be local or distributed

Replaceable Components

- Avoid a monolithic “Standard Library”
- Replaceable components with version control
 - > Components implement APIs
 - > Components import APIs, not other components
 - > An API may have multiple implementations
- Encourage alternate implementations
 - > Performance choices
 - > Test them against each other
- Encourage experimentation
 - > Framework for alternate language designs

Type System: Objects and Traits

- Traits: like interfaces, but may contain code
Based on work by Schärli, Ducasse, Nierstrasz, Black, et al.
- Multiple inheritance of code (but not fields)
Objects with fields are the leaves of the hierarchy
- Multiple inheritance of contracts and tests
Automated unit testing
- Traits and methods may be parameterized
Parameters may be types or compile-time constants
- Primitive types are first-class
Booleans, integers, floats, characters are all objects

Sample Code: Algebraic Constraints

```
trait BinaryPredicate[[T extends BinaryPredicate[[T, ~]], opr ~]]
  opr ~(self, other: T): Boolean
end

trait Symmetric[[T extends Symmetric[[T, ~]], opr ~]]
  extends { BinaryPredicate[[T, ~]] }
  property  $\forall(a: T, b: T) (a \sim b) \leftrightarrow (b \sim a)$ 
end

trait EquivalenceRelation[[T extends EquivalenceRelation[[T, ~]], opr ~]]
  extends { Reflexive[[T, ~]], Symmetric[[T, ~]], Transitive[[T, ~]] }
end

trait  $\mathbb{Z}$  extends { EquivalenceRelation[[ $\mathbb{Z}$ , =]], CommutativeRing[[ $\mathbb{Z}$ , +, -, ·]],
  TotalOrderOperators[[ $\mathbb{Z}$ , <, ≤, ≥, <, CMP], ... ]
  ...
end
```

Data and Control Models

- Data model: shared global address space
- Control model: multithreaded
 - > Basic primitive is spawn
 - > We hope application code seldom uses it
 - Implicit parallelism is actually more flexible
- Declared distribution of data and threads
 - > Management of aggregates integrated into type system
 - > Policies programmed as libraries, not wired in
- Transactional access to shared variables
 - > Atomic blocks (implicit or explicit retry)
 - > Lock-free (no blocking, no deadlock)

Why Transactional Memory?

- Locks can manage parallelism but have problems
 - > Overprotective (don't allow overlapping reads)
 - > Susceptible to deadlock and priority inversion
- Just mark a block of code `atomic` (indivisible)
- Let compiler and runtime manage the details
 - > With locks, which locks protect which data?
- Many hardware, software, or hybrid approaches
 - > Most of them are “optimistic”: better throughput
- Supports overlapping reads
- No deadlock (though “livelock” is still an issue)

Visit <http://projectfortress.sun.com>

An open-source project with international participation

- Open source since January 2007
- University participation includes:
 - > University of Tokyo: matrix algorithms
 - > Rice University: code optimization
 - > Aarhus University: syntactic abstraction
 - > University of Texas at Austin: static type checking
- Also participation by many individuals

A Growing Library

The Fortress library now includes over 10,000 lines of code.

- Integer, floating-point, and string operations
- Collections (lists, sets, maps, heaps, etc.)
- Multidimensional arrays
- Sparse vectors and matrices
- Generators and reducers
 - > Implement loops, comprehensions, and reductions
 - > Support implicit parallelism
- Fortress abstract syntax trees
- Sorting

Sample Library Code 1

This is a portion of an implementation of Skip Lists by Michael Spiegel of the University of Virginia:

```
object SkipList[[Key, Val, nat pInverse]](root: Node[[Key, Val]])
  (* add a (key, value) pair *)
  add(k: Key, v: Val): SkipList[[Key, Val, pInverse]] = do
    level:  $\mathbb{Z}32$  = randomLevel()
    values: Array[[Val,  $\mathbb{Z}32$ ]] = array[[Val]](1)
    values0 := v
    leaf: LeafNode[[Key, Val]] = LeafNode[[Key, Val]](k, values)
    SkipList[[Key, Val, pInverse]](root.add(leaf, level))
end
```

Sample Library Code 2

This is a portion of an implementation of Skip Lists by Michael Spiegel of the University of Virginia:

```
(* remove one (key, value) pair *)  
remove(k: Key): SkipList[[Key, Val, pInverse]] = do  
  (root', maybe) = root.remove(k)  
  SkipList[[Key, Val, pInverse]](root')  
end
```

Tools: 'Fortify' Code Formatter

- Emacs-based tool
- Fortress programs can be typed on ASCII keyboards
- Code automatically formatted for processing by \LaTeX

```
sum: RR64 := 0
for k<-1:n do
  a[k] := (1-alpha)b[k]
  sum += c[k] x^k
end
```

```
sum:  $\mathbb{R}64$  := 0
for  $k \leftarrow 1:n$  do
   $a_k := (1 - \alpha)b_k$ 
   $sum += c_k x^k$ 
end
```

All code on these slides was formatted by this tool.

Tools: Editing Environments

- Fortress mode for Emacs
 - > Provides syntax coloring
 - > Some automatic formatting and font conversion
- Fortress NetBeans™ plug-in
 - > Syntax highlighting
 - > Mark occurrences
 - > Instant rename
- These tools were contributed by people outside Sun

Fortress 1.0

- With the Fortress 1.0 release (March, 2008), we have synchronized the specification and implementation
- Implementation expanded and made more reliable since Fortress 1.0 β
- Many features in the 1.0 β specification have been removed for 1.0

Fortress 1.0

- With the Fortress 1.0 release (March, 2008), we have synchronized the specification and implementation
- Implementation expanded and made more reliable since Fortress 1.0 β
- Many features in the 1.0 β specification have been removed for 1.0
 - > *But with every intention of adding them back as the language grows*

Automated Testing During Spec Build

- Consistent with our emphasis on unit testing, all code examples in the Fortress specification are:
 - > Automatically tested
 - > Automatically formatted as part of our build process
 - > Included in our open source distribution
- All examples that follow are working code taken from the Fortress 1.0 distribution and tested on every build

This slide . . .

which contains the Fortress code

$$\{ x^2 \mapsto x^3 \mid x \leftarrow \{0, 1, 2, 3, 4, 5\}, x \text{ MOD } 2 = 0 \}$$

... is auto-rendered from this LaTeX

```
\begin{slide}{This slide \ldots}
```

```
\bigskip
```

which contains the Fortress code

```
{
```

```
  {  $x^2 \mapsto x^3 \mid x \leftarrow \{0, 1, 2, 3, 4, 5\}, x \bmod 2 = 0$  }
```

```
{
```

```
\end{slide}
```

This example in the Fortress spec . . .

$$A: \mathbb{Z}_{32}[2, 2] = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

... is auto-extracted from this test file

```
component Expr.Array.b
```

```
export Executable
```

```
f() = do
```

```
  (** EXAMPLE **)
```

```
    A: ZZ32[2,2] = [3 4  
                   5 6]
```

```
  (** END EXAMPLE **)
```

```
    A[1,0]
```

```
end
```

```
run(args: String...) = println f()
```

```
end
```

What works NOW

- Parallelism in loops, reductions, comprehensions, tuples
- Automatic load balancing via work-stealing

```
for  $i \leftarrow 0 \# |children'|$  do  
   $children'_i := generate\_tail[[Key, Val]](children_{i+lsize+1}, 1)$   
end
```

What works NOW

- Parallelism in loops, reductions, comprehensions, tuples
- Automatic load balancing via work-stealing

```
object SumZZ32 extends Reduction[[Z32]]
```

```
  empty(): Z32 = 0
```

```
  join(a: Z32, b: Z32): Z32 = a + b
```

```
end
```

```
z = (1 # 100).generate[[Z32]](SumZZ32, fn (x) => 3x + 2)
```

What works NOW

- Parallelism in loops, reductions, comprehensions, tuples
- Automatic load balancing via work-stealing

$$\mathit{factorial}(n: \mathbb{Z}32) = \prod_{i \leftarrow 1:n} i$$

$$\mathit{opr} (n: \mathbb{Z}32)! = \prod_{i \leftarrow 1:n} i$$

What works NOW

- Parallelism in loops, reductions, comprehensions, tuples
- Automatic load balancing via work-stealing

$$\langle x^2 \mid x \leftarrow \{0, 1, 2, 3, 4, 5\}, x \text{ MOD } 2 = 0 \rangle$$

What works NOW

- Spawn

```
spawn do
```

```
     $s := \text{Done}[[T]](\text{old.val}())$ 
```

```
end
```

What works NOW

- Atomic blocks

```

attempt(): (State[[T]], Boolean) = atomic do
  old = s
  computed := old.isDone()
  if ¬old.isDone() then
    if old.isPending() then abort() end
    s := Pending[[T]]
    (old, true)
  else
    (old, false)
  end
end
end

```

What works NOW

- Object-oriented type system with multiple inheritance

```
trait Comparison
  extends { StandardPartialOrder[[Comparison]] }
  comprises { Unordered, TotalComparison }
  getter toString(): String
  opr =(self, other: Comparison): Boolean = false
  opr LEXICO(self, other: Comparison): Comparison =
    Unordered
  opr INVERSE(self): Comparison
end
```

What works NOW

- Object-oriented type system with multiple inheritance
- Overloaded methods and operators with dynamic multimethod dispatch

```
opr ◦[[A, B, C]](f: B → C, g: A → B): A → C = fn (a: A): C ⇒ f(g(a))
```

```
opr |x:ℤ32| : ℤ32 = if x ≥ 0 then x else -x end
```

```
opr |x:ℤ64| : ℤ64 = if x ≥ 0 then x else -x end
```

```
opr juxtaposition [[T extends Number, nat n, nat m, nat p]]
  (other: T, me: Vector[[T, n]]): Vector[[T, n]] = me.scale(other)
```

```
opr juxtaposition [[T extends Number, nat n, nat m, nat p]]
  (me: Vector[[T, n], other: T): Vector[[T, n]] = me.scale(other)
```

```
opr juxtaposition (a:ℤ32, b:ℤ32):ℤ32 = builtinPrimitive(
  “com.sun.fortress.interpreter.glue.prim.Int$Mul”)
```

What works NOW

- Sets, arrays, lists, maps
- Pure queues, dequeues, priority queues
- Integers, floating-point, strings, booleans

$$v = \{x \mid x \leftarrow t, x \geq 0\}$$

$$c: \mathbb{Z}^{32}[2, 2, 3] = [1\ 2$$

$$3\ 4; ; 5\ 6$$

$$7\ 8; ; 9\ 10$$

$$11\ 12]$$

$$l = \langle 2x \mid x \leftarrow v \rangle$$

Next steps

- Full static type checker
- Static type inference to reduce “visual clutter”
- Syntactic abstraction
- Parallel nested transactions
- *Code generation*

Future Directions

- Compiler initially targeted to Java Virtual Machine
 - > Full multithreaded platform independence
- Fortress-specific optimizations

Libraries: What's Next

- Efficient concurrent mutable data structures
- Better I/O
- Parsing and string formatting
- Bindings to external BLAS, FFTW, etc., for linear algebra
- Bindings to talk to databases and use them as data sources/sinks
- Additional basic collections (such as bags and graphs)

It is an exciting time for the project

- External contributions and feedback are increasing
- Many implementation tasks are being done outside Sun
- The language is growing
- A community of developers is participating in its evolution

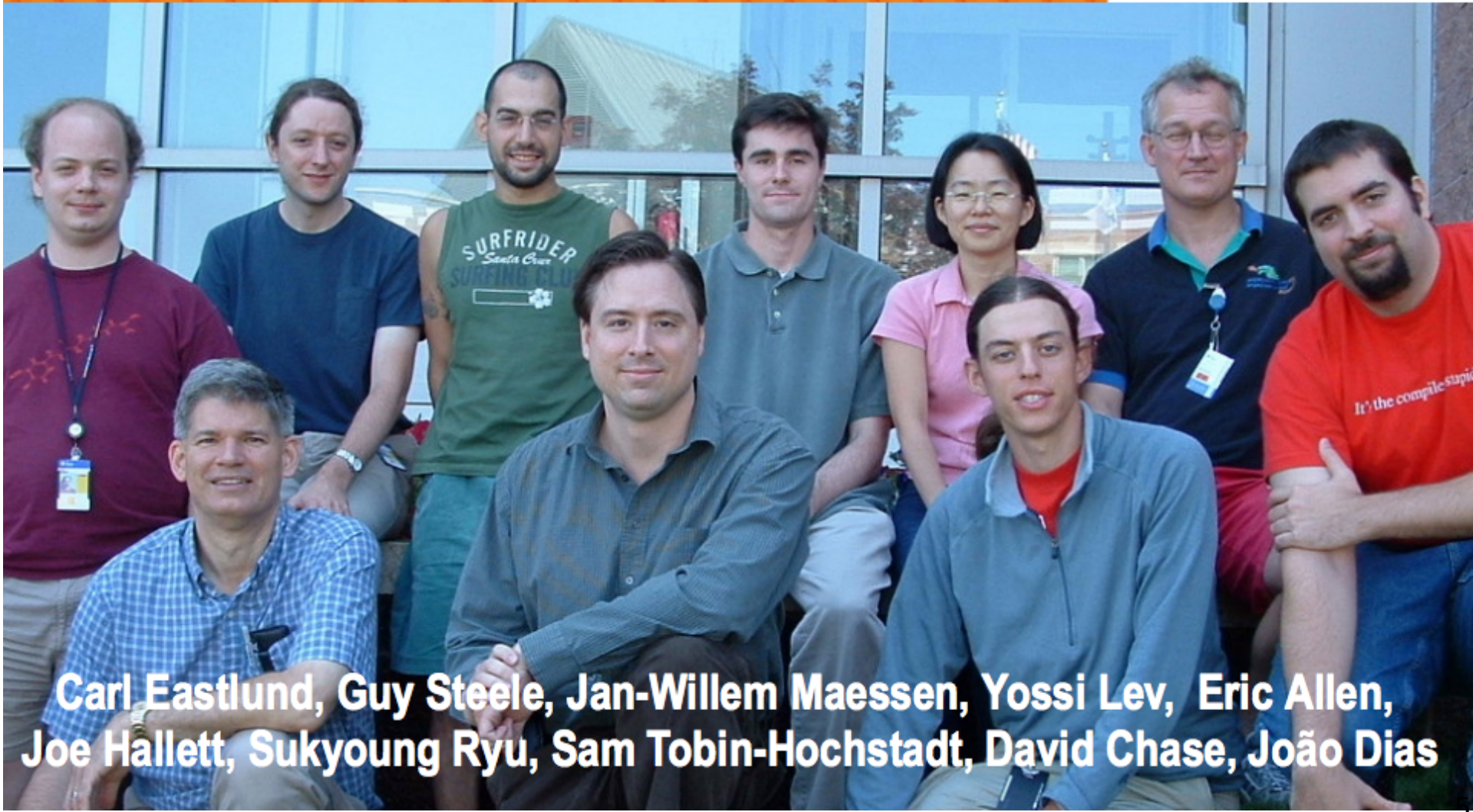
Summary

- Fortress is a growable, mathematically oriented, parallel programming language
- Fortress is an open-source project with international participation
- The Fortress 1.0 release has synchronized the specification and implementation
- Moving forward, we expect to grow the language and libraries and develop a compiler

guy.steele@sun.com

<http://research.sun.com/projects/plrg>

<http://projectfortress.sun.com>



Carl Eastlund, Guy Steele, Jan-Willem Maessen, Yossi Lev, Eric Allen, Joe Hallett, Sukyoung Ryu, Sam Tobin-Hochstadt, David Chase, João Dias