Fortress
Programming Language Project Status

Guy Steele

Sun Fellow

Sun Microsystems Laboratories

September 2008

Fortress Status Report

- Fortress is a growable, mathematically oriented, parallel programming language
- Started under Sun/DARPA HPCS program, 2003–2006
- Fortress is now an open-source project with international participation
- The Fortress 1.0 release (March 2008) synchronized the specification and implementation
- Moving forward, we are growing the language and libraries and developing a compiler

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
```

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

```
for k ← 1:n do
   a_k := b_k
   sum += c_k x^k
end
```

```
do
   T1 = spawn f(a)
   T2 = spawn g(b)
   T1.wait(); T2.wait()
end
```

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

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 conventionally 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 $S.size$
    - If you prefer $\#\,S$, defining it is a one-liner.

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

$\boxplus$ $\boxminus$ $\boxtimes$ $\leftrightarrow$ $\wedge$ $\vee$ $\equiv$ $\not\equiv$ ϵ ζ η θ

$\leq$ $\geq$ $\sum$ $\prod$ $\prec$ $\preccurlyeq$ $\succcurlyeq$ $\succ$ ι κ λ μ

$\cap$ $\cup$ $\uplus$ $\subset$ $\subseteq$ $\supseteq$ $\supset$ $\in$ ξ π ρ σ

$\sqcap$ $\sqcup$ $\sqsubset$ $\sqsubseteq$ $\sqsupseteq$ $\sqsupset$ $\neg$ $\notin$ ϕ χ ψ ω

$\lfloor$ $\rfloor$ $\lceil$ $\rceil$ $\langle$ $\rangle$ $\curlywedge$ $\curlyvee$ Γ Θ and so on

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

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
- Big integers, rational numbers, intervals
- 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

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
```

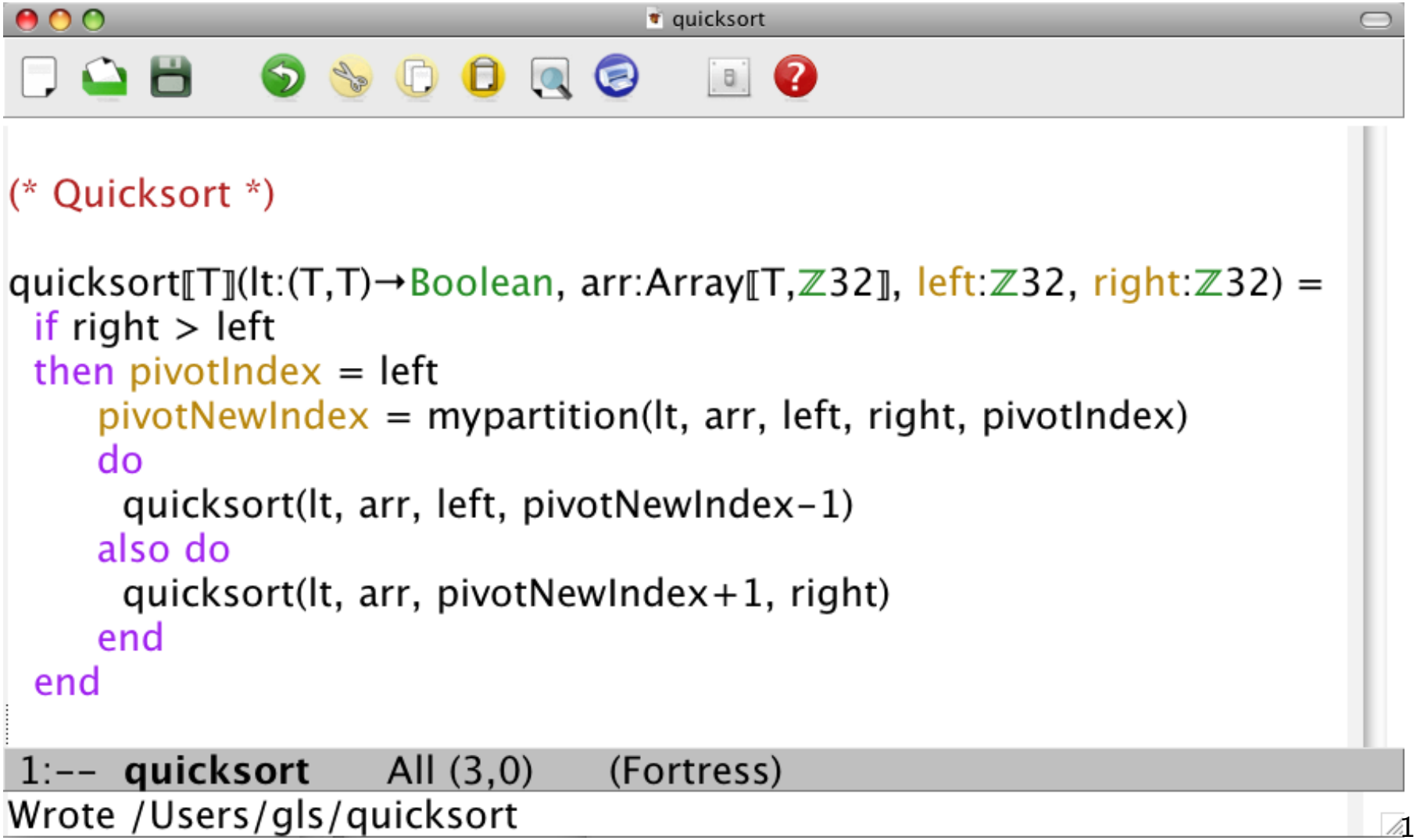
$$
\begin{array}{l}
sum\colon \mathbb{R}64 := 0 \\
\texttt{for } k \leftarrow 1\colon n \texttt{ do} \\
\quad a_k := (1-\alpha) b_k \\
\quad sum \mathrel{+}= c_k\, x^k \\
\texttt{end}
\end{array}
$$

All code on these slides was formatted by this tool.

Tools: Editing Environments

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

Syntax Coloring Screen Shot



Fortress 1.0

- With the Fortress 1.0 release in March 2008, we synchronized the specification and implementation
- Implementation expanded and made more reliable since Fortress 1.0β
- Many features in the 1.0β specification were removed for 1.0
  - *But with every intention of adding them back as the language grows*
  - And we have done so over the last six months

Automated Testing During Spec Build

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

This slide...

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

...is auto-rendered from this LaTeX

```
\begin{slide}{This slide...}
\begin{center}
‘
{ x^2 |-> x^3 | x <- {0, 1, 2, 3, 4, 5}, x MOD 2 = 0}
‘
\end{center}
\end{slide}
```

This example in the Fortress spec...

$$A\colon \mathbb{Z}32[2,2] = [\,3\ 4$$
$$5\ 6\,]$$

...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

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

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

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

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

What works NOW

- Spawn

```
spawn do
        s := Done⟦T⟧(old.val())
end
```

What works NOW

- Atomic blocks with transactional memory

$attempt()$: (State$[\![T]\!]$, Boolean) = `atomic do`

    $old = s$

    $computed := old.isDone()$

    `if` $\neg old.isDone()$ `then`

        `if` $old.isPending()$ `then` $abort()$ `end`

        $s :=$ Pending$[\![T]\!]$

        $(old, true)$

    `else`

        $(old, false)$

    `end`

`end`

What works NOW

- Object-oriented type system with multiple inheritance
- Overloaded methods and operators with dynamic multimethod dispatch
- Sets, arrays, lists, maps, skip lists
- Pure queues, deques, priority queues
- Integers, floating-point, strings, booleans
- Big integers, rational numbers, interval arithmetic
- Syntactic abstraction (just barely)

Next steps:

- Full static type checker (almost there!)
- Static type inference to reduce "visual clutter"
- Parallel nested transactions
- Compiler
  - Initially targeted to JVM for full multithreaded platform independence
  - After that, VM customization for Fortress-specific optimizations

It is an exciting time for the project

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

guy.steele@sun.com
http://research.sun.com/projects/plrg
http://projectfortress.sun.com