



# What's Cool about Fortress

**Guy L. Steele Jr.**

Sun Fellow

Sun Microsystems Laboratories



**2007  
Sun Labs  
Open House**



# Project Fortress

- Programming language for scientific applications
- Begun at Sun Labs as part of DARPA HPCS program for High Productivity Computing Systems
- Now an operational open-source project with international participation
  - > <http://fortress.sunsource.net>
- Freely available language specification
  - > <http://research.sun.com/projects/plrg>

# Mathematical Notation

- A unique yet familiar look

$$A = \{ p \mid p \leftarrow 1:100, \text{prime } p \}$$

$$n = |A|$$

$$B = \{ |x| \mid x \leftarrow \text{myStrings} \}$$

$$m_{total} = \sum B$$

$$C = (A \cap B) \cup \{0, 1\}$$

**for**  $i \leftarrow 1:n, j \leftarrow 1:n$  **do**

$$a_{i,j} = \sum_{k \leftarrow 1:n} b_{i,k} c_{k,j}$$

**end**

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

$$q = A p$$

$$\alpha = \frac{\rho}{p^T q}$$

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

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

$$\rho_0 = \rho$$

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

$$\beta = \frac{\rho}{\rho_0}$$

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

**end**

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

# Implicit Parallelism (1 of 2)

- Most parallel languages start with a serial language and add features or annotations
- Fortress is implicitly parallel wherever possible

$$A = \{ \text{crunch}(f(p), g(p)) \mid p \leftarrow 1:100, \text{prime } p \}$$

$$C_{total} = \sum A$$

**for**  $(i, j) \leftarrow a.indices$  **do**

$$a_{i,j} = \sum_{k \leftarrow 1:n} b_{i,k} c_{k,j}$$

**end**

**for**  $(i, j) \leftarrow a.indices.rowMajorOrder$  **do**

*print*  $a_{i,j}$

**end**

# Implicit Parallelism (2 of 2)

- It does require some care
- Parallelism is crucial now for high-end applications
- Crucial soon for multicore desktops and servers

[As multicore processor chips become ubiquitous,]  
“what we are seeing is not a gradual shift but a cataclysmic shift from the sequential world to one in which every processor is parallel. In a small number of years, if your language does not support parallelism, that language will just wither and die.”

—John Mellor-Crummey, Rice University  
(*Computerworld*, March 12, 2007)

# Transactional Memory/Atomic Blocks

- 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 (“livelock” is still an issue)

# Units and Dimensions

- Explicit declarations of physical dimensions
- Compiler does automatic dimensional analysis
- Make sure physical dimensions make sense
- Catches coding errors that ordinary types cannot (confusing mass with weight, or meters with feet)

$$a = 9.8 \text{ m}^2/\text{s}$$

$$m = 30 \text{ kg}$$

$$f = ma$$

$$z = \cancel{f} + a$$

# A Focus on Libraries

- We've tried to keep Fortress “small”
- Wherever possible, add features as libraries
  - > Example: units are enforced by library code
  - > Example: loops implemented by library code (see later)
- Distinction between application programming and library programming
- Many language features support library coding
  - > Support for rich syntax, not just method calls
- Try to avoid application-specific language features

# Expressive Type System

- This topic is for the real language wonks: professional-strength tools for library coders
- Generic types done right
  - > Powerful “where” clauses express type constraints
    - Covariance and contravariance fall out as a special case
  - > No type erasure, no loopholes
- Trying to capture as type information what is usually computed by flow analysis
  - > What can you know without actually executing the code?
- Library coders use complicated types so application coders won't have to

# Best of O-O and Functional Styles

- Fortress is at heart an object-oriented language
- But lots of support for a mostly functional style
- Array-oriented style is good for many applications
  - > “Fortran style”
  - > Arrays are one kind of object supplied by Fortress libraries
- We may find that array-oriented side effects just don't scale well to petascale machines
  - > Supporting functional style keeps options open
  - > Some find functional style more productive for some codes

# Generators and Reducers

- A mixed O-O/functional style of library coding implements the implicit parallelism in loops!

**for**  $k \leftarrow 1:100$  **do**  $a_k := 2^k$  **end**

$A = \{ 2^k \mid k \leftarrow 1:100 \}$

$n = \sum_{k \leftarrow 1:100} 2^k$

$(1:100).generate(forReducer, \mathbf{fn}(k) \rightarrow (a_k := 2^k))$

$(1:100).generate(setReducer, \mathbf{fn}(k) \rightarrow 2^k)$

$(1:100).generate(sumReducer, \mathbf{fn}(k) \rightarrow 2^k)$

- Reducers combine two things and provide defaults

# Programming in the Large

- Components and APIs
- Version control
- Contracts
- Automated unit testing

# Fortress Is an Open-Source Project

- Interpreter coded in Java
  - > Released January 2007; multiple updates since then
  - > BSD license (mostly—see web site)
  - > Platform-independent
  - > Uses Java multithreading to implement Fortress threads
    - But not one-to-one!
    - Work-stealing strategy automatically balances the load
- EMACS-based code formatting tool
  - > Rewrites Fortress code into LaTeX source
- Rudimentary libraries and some sample code

# Community Participation

- People outside Sun in multiple countries are:
  - > Engaged in active discussions
  - > Reporting bugs
  - > Writing Fortress source code
- Today's demo includes Fortress code contributed by the open-source community
- Fortress is catching on!

# What's Next?

- More libraries!
  - > Fortress is designed to grow that way
  - > Technical design facilitates open-source community
- Interpreter, compiler, and run-time work
  - > Let programmer say where parallelism is *possible*
  - > Let implementation decide where parallelism is *effective*
- Good prospects not just for high-end systems but also multicore processors (singly or in clusters)



<http://fortress.sunsource.net>

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

**Guy L. Steele Jr.**

[guy.steele@sun.com](mailto:guy.steele@sun.com)



**2007  
Sun Labs  
Open House**

