Project Fortress:
A Growable Language for Scientists and Engineers

Sukyoung Ryu
Department of Computer Science
Korea Advanced Institute of Science and Technology
October 10, 2010
Project Fortress

- A multicore language for scientists and engineers
Project Fortress

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!
Project Fortress

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!

\[ v_{\text{norm}} = \frac{v}{\|v\|} \]

\[ \sum_{k=1}^{n} a_k x^k \]

\[ C = A \cup B \]

\[ y = 3x \sin x \cos 2x \log \log x \]
Project Fortress

- A **multicore language** for scientists and engineers
- Run your whiteboard **in parallel**!

\[ v_{\text{norm}} = \frac{v}{\|v\|} \]
\[ \sum_{k \leftarrow 1:n} a_k x^k \]

\[ C = A \cup B \]

\[ y = 3x \sin x \cos 2x \log \log x \]
Project Fortress

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!

\[ v_{\text{norm}} = \frac{v}{\|v\|} \]
\[ \sum_{k=1}^{n} a_k x^k \]
\[ C = A \cup B \]
\[ y = 3x \sin x \cos 2x \log \log x \]

- “Growing a Language”

Guy L. Steele Jr., keynote talk, OOPSLA 1998

*Higher-Order and Symbolic Computation* 12, 221-236 (1999)
Project Fortress

- Fortress is a growable, mathematically oriented, parallel programming language for scientific applications.
- Fortress is an open-source project with international participation.
- The Fortress 1.0 release (March 2008) synchronized the specification and implementation.
- Now at Sun Labs, Oracle, we are growing the language and libraries and developing a compiler.
Mathematical Syntax
Mathematical Syntax

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 \( a \times x^2 + b \times x + c \) and \( ax^2 + bx + c \)
  > 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 Using Unicode

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

  \times \div \oplus \ominus \otimes \oslash \odot \approx \alpha \beta \gamma \delta

  \boxplus \boxminus \leftrightarrow \land \lor \equiv \not\equiv \epsilon \zeta \eta \theta

  \leq \geq \sum \prod \prec \preceq \succeq \succ \iota \kappa \lambda \mu

  \cap \cup \uplus \subset \subseteq \supset \supseteq \in \xi \pi \rho \sigma

  \bigcap \bigcup \subseteq \supseteq \ni \not\in \phi \chi \psi \omega

  \llbracket \rrbracket \langle \rangle \wedge \vee \Gamma \Theta \text{ and so on}

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

NAS NPB 1 Specification

\[ z = 0 \]
\[ r = x \]
\[ \rho = r^T r \]
\[ p = r \]
\[ \textbf{DO } i = 1,25 \]
\[ \quad q = A p \]
\[ \quad \alpha = \rho l(p^T q) \]
\[ \quad z = z + \alpha p \]
\[ \quad \rho_0 = \rho \]
\[ \quad r = r - \alpha q \]
\[ \quad \rho = r^T r \]
\[ \quad \beta = \rho l\rho_0 \]
\[ \quad p = r + \beta p \]
\[ \textbf{ENDDO} \]

compute residual norm explicitly: \[ \|r\| = \|x - Az\| \]
Mathematical Syntax Example

NAS NPB 2.3 Serial Code in Fortran

do j=1,naa+1
    q(j) = 0.0d0
    z(j) = 0.0d0
    r(j) = x(j)
    p(j) = r(j)
    w(j) = 0.0d0
endo

sum = 0.0d0
do j=1,lastcol-firstcol+1
    sum = sum + r(j)*r(j)
endo

rho = sum
do cgit = 1,cgitmax
    do j=1,lastrow-firstrow+1
        sum = 0.d0
        do k=rowstr(j),rowstr(j+1)-1
            sum = sum + a(k)*p(colidx(k))
        enddo
        w(j) = sum
    enddo
    do j=1,lastcol-firstcol+1
        q(j) = w(j)
    enddo
endo

do j=1,lastrow-firstrow+1
    w(j) = 0.0d0
endo

rho = sum
beta = rho / rho0

sum = 0.0d0
do j=1,lastcol-firstcol+1
    sum = sum + r(j)*r(j)
endo

rho = sum
beta = rho / rho0

d = x(j) - r(j)
sum = sum + d*d
endo

d = sum
rnorm = sqrt( d )
Mathematical Syntax Example

NAS NPB 1 Specification

\[ z = 0 \]
\[ r = x \]
\[ \rho = r^T r \]
\[ p = r \]
\[ \text{DO } i = 1, 25 \]
\[ q = A \rho \]
\[ \alpha = \rho l(p^T q) \]
\[ z = z + \alpha p \]
\[ \rho_0 = \rho \]
\[ r = r - \alpha q \]
\[ \rho = r^T r \]
\[ \beta = \rho l \rho_0 \]
\[ p = r + \beta \rho \]
\[ \text{ENDDO} \]

compute residual norm explicitly: \[ \|r\| = \|x - Az\| \]

\[
\begin{align*}
z & : \text{Vec} := 0 \\
r & : \text{Vec} := x \\
p & : \text{Vec} := r \\
\rho & : \text{Elt} := r^T r \\
\text{for } j \leftarrow \text{seq}(1 : cgit_{\text{max}}) \text{ do} \\
\quad q & = A \rho \\
\quad \alpha & = \frac{\rho}{p^T q} \\
\quad z & := z + \alpha p \\
\quad r & := r - \alpha q \\
\quad \rho_0 & = \rho \\
\quad \rho & := r^T r \\
\quad \beta & = \frac{\rho}{\rho_0} \\
\quad p & := r + \beta \rho \\
\text{end} \\
(z, \|x - Az\|)
\end{align*}
\]
Mathematical Syntax by ‘Fortify’

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

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

All code on these slides was formatted by this tool.
Mathematical Syntax by ‘Fortify’

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

\begin{align*}
  \text{sum: RR}64 & := 0 \\
  \text{for } k \leftarrow 1:n \text{ do} \\
  & a[k] := (1 - \alpha)b[k] \\
  & \text{sum } += c[k] x^k \\
  \text{end}
\end{align*}

All code on these slides was formatted by this tool.
Mathematical Syntax Using Editors

- Fortress mode for Emacs
  - Provides syntax coloring
  - Some automatic formatting
  - Unicode font conversion
- Fortress NetBeans™ plug-in
  - Syntax highlighting
  - Mark occurrences
  - Instant rename
- Fortress Eclipse plug-in (work in progress)
- These tools were contributed by people outside Sun.
Parallelism by Default
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
Implicit Parallelism

- Tuples

\[(a, b, c) = (f(x), g(y), h(z))\]

- Functions, operators, method call recipients, and their arguments

\[e_1 e_2, \quad e_1(e_2), \quad e_1 + e_2, \quad e_1.method(e_2)\]

- Expressions with generators

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

\[\{ x^2 \mid x \leftarrow xs, x > 43 \}\]
Recursive Subdivision and Work Stealing

\[
\sum_{k=1}^{1024} c_k x^k
\]

Thread 1

Thread 2

Thread 3

Thread 4
Should Parallelism Be the Default?

- “Loop” is a misleading term in the Fortress world
  - A set of executions of a parameterized block of code.
  - Whether and how they run in parallel is a separate concern.
- In Fortress loops are parallel by default
  - This is a convention of library code, as we will see.

In order to get programmers to write parallel code, we must change their (and our) mind set. The ubiquitous parallelism of Fortress is our attempt.
Growable Language
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.
Growing a Language

“So I think the sole way to win is to plan for growth with help from users.”

Guy L. Steele Jr.
keynote talk, OOPSLA 1998;
Higher-Order and Symbolic Computation 12, 221-236 (1999)
Design Strategy

Consider how a proposed language feature might be provided by a library rather than building features directly into the compiler.

This requires control over both syntax and semantics, not just the ability to add new functions and methods.
for Loops in Fortress

```fortress
for generators do
    body
end
```

- **Generator list** produces data items (usually in parallel).
- **Body computation** is executed for each data item.
- Whether/how they run in parallel is defined in the libraries.
for Loops in Fortress

```
for generators do
  body
end
```

```
for i ← {1, 2}, j ← {3, 4} do
  println "(" i "", " j "")"
end
```

- **Generator list** produces data items (usually in parallel).
- **Body computation** is executed for each data item.
- Whether/how they run in parallel is defined in the libraries.
- In Fortress, `for` loops are parallel by default.

(2, 3)
(1, 3)
(2, 4)
(1, 4)
Desugaring for Loops

\[ g_1 = \{1, 2\} \]
\[ g_2 = \{3, 4\} \]

for \(i \leftarrow g_1, j \leftarrow g_2\) do
\hspace{1cm} println “(” i “, ” j “)”
end
Desugaring for Loops

\[ g_1 = \{1, 2\} \]
\[ g_2 = \{3, 4\} \]

for \( i \leftarrow g_1, j \leftarrow g_2 \) do
  println "(" i " , " j " )"
end

desugars to

\[ g_1 . loop(fn \ i \Rightarrow \]
\[ g_2 . loop(fn \ j \Rightarrow \]
  println "(" i " , " j " )" ))
Desugaring for Loops

- Design strategy

Consider how a proposed language feature might be provided by a library rather than building features directly into the compiler.
Desugaring for Loops

• Design strategy
  Consider how a proposed language feature might be provided by a library rather than building features directly into the compiler.

• for loops desugaring in the current implementation is built directly into the interpreter.
Desugaring for Loops

- Design strategy
  Consider how a proposed language feature might be provided by a library rather than building features directly into the compiler.

- for loops desugaring in the current implementation is built directly into the interpreter.

- for loops desugaring by syntactic abstraction is provided by a library.
Syntactic Abstraction Goals\textsuperscript{a}

- New syntax indistinguishable from the core syntax
- Similar syntax for definition/use of a language extension
- Composition of independent language extensions
- Expansion into other language extensions
- Mutually recursive definition of a language extension

\textsuperscript{a}Growing a Syntax, Eric Allen, Ryan Culpepper, Janus Dam Nielsen, Jon Rafkind, and Sukyoung Ryu. The Foundations of Object-Oriented Languages 2009
Grammar of Simplified *for* Loops

grammar ForLoop extends { Expression, Identifier }

Expr ::= 
  for { i : Id ← e : Expr , ? Space }* do block : Expr end ⇒ 
  \[ for_2 i * * ; e * * ; do block ; end \]
  | for_2 i : Id* ; e : Expr* ; do block : Expr ; end ⇒ 
  case i of 
    Empty ⇒ \[ block \]
    Cons(ia, ib) ⇒
      case e of
        Cons(ea, eb) ⇒
          \[ ((ea).loop(fn ia ⇒ (for_2 ib * * ; eb * * ;
          do block ; end))))\]
          end
          end
    end
end
Syntactic Abstraction in Fortress

- New syntax indistinguishable from the core syntax

```plaintext
for i ← g₁, j ← g₂ do
  println "(" i " , " j " )"
end
```

- Similar syntax for definition/use of a language extension

```plaintext
for { i:Id ← e:Expr , ? Space } * do block:Expr end ⇒ ...
```
Syntactic Abstraction in Fortress

• Composition of independent macros

  grammar ForLoop extends \{Expression, Identifier\}

• Expansion into other language extensions

  \texttt{Expr ::=}
  \begin{align*}
  \text{for } \{ i : \text{Id} \leftarrow e : \text{Expr} \, , \, ? \text{Space} \}^* \text{ do } & \text{block} : \text{Expr} \text{ end } \Rightarrow \\
  \langle \text{for}_2 i \ast \star; e \ast \star; \text{do } \text{block} ; \text{end} \rangle \\
  | \text{for}_2 i : \text{Id}^*; e : \text{Expr}^*; \text{do } \text{block} : \text{Expr}; \text{end } \Rightarrow \\
  \ldots
  \end{align*}
Syntactic Abstraction in Fortress

- Mutually recursive definition of a language extension

\[
\text{for}^2 i : \text{Id}^*; e : \text{Expr}^*; \text{do } \text{block} : \text{Expr} ; \text{end} \Rightarrow \\
\text{case } i \text{ of} \\
\quad \text{Empty} \Rightarrow \langle \text{block} \rangle \\
\quad \text{Cons}(ia, ib) \Rightarrow \\
\quad \quad \text{case } e \text{ of} \\
\quad \quad \quad \text{Cons}(ea, eb) \Rightarrow \\
\quad \quad \quad \quad \langle ((ea).\text{loop}(\text{fn} \ ia \Rightarrow (\text{for}^2 ib \ast\ast; eb \ast\ast; \\
\quad \quad \quad \quad \quad \text{do } \text{block}; \text{end}))), \rangle \rangle \\
\quad \quad \text{end} \\
\text{end}
\]
Desugaring for Loops Recursively

\[ \text{for } i \leftarrow g_1, j \leftarrow g_2 \text{ do } \text{println } "\text{ (" i "", " j ")" } \text{ end} \]
Desugaring for Loops Recursively

for \( i \leftarrow g_1, j \leftarrow g_2 \) do \( \text{println "(" i "", "" j "")"} \) end

desugars to

\[
g_1.\text{loop}(\text{fn } i \mapsto \\
\quad \text{for } j \leftarrow g_2 \text{ do} \\
\quad \quad \text{println "(" i "", "" j "")"} \\
\quad \text{end})
\]
Desugaring for Loops Recursively

for \( i \leftarrow g_1, j \leftarrow g_2 \) do \( \text{println "" ("" i "", "" j "")"" end} \)

desugars to

\[
g_1\.loop(fn \ i \mapsto \\
\hspace{1cm} \text{for } j \leftarrow g_2 \text{ do} \\
\hspace{1.5cm} \text{println "" ("" i "", "" j "")""}
\hspace{1.5cm} \text{end})
\]

desugars to

\[
g_1\.loop(fn \ i \mapsto \\
\hspace{1.5cm} g_2\.loop(fn \ j \mapsto \\
\hspace{2cm} \text{println "" ("" i "", "" j "")"" ))
\]
Syntax Normalization

- Parsing stage
  transforms a source program (in string) into a parsed program (in node expression)

- Transformation stage
  transforms the parsed program into a program in core Fortress AST
Parsing: Source Program $\Rightarrow$ Parsed Program

- First step
  parses the macro definition into an intermediate form to generate a parser that recognizes the new syntax.

- Second step
  uses the generated parser to parse a source program using the new syntax.
Node Expressions

\[
    \text{NodeExpr} ::= \quad \text{PatternVar} \\
    \quad | \quad \text{Transformer} (\text{NodeExpr}) \\
    \quad | \quad \text{NodeConstructor}(\text{NodeExpr}) \\
    \quad | \quad \text{Ellipses}(\text{NodeExpr}, \text{NodeExpr}) \\
    \quad | \quad \text{case} \ \text{PatternVar} \ \text{of} \\
    \quad | \quad \text{Empty} \Rightarrow \text{NodeExpr} \\
    \quad | \quad \text{Cons}(\text{PatternVar}, \text{PatternVar}) \Rightarrow \text{NodeExpr} \\
    \text{end}
\]
Transformation: Parsed Program $\Rightarrow$ AST

- Pattern variables are substituted by the corresponding inputs.
- Transformers are replaced with their bodies, substituting pattern variables along the way.
- Core Fortress AST nodes transform their arguments.
- Ellipses nodes transform multiple occurrences of patterns.
- Case expressions match input to a constructor and invoke the corresponding transformer.
Translation of Node Expressions

[Pattern Variable]

\[ \text{Γ}(x) = v \]

\[ \vdash \text{Γ} \vdash x \rightarrow \text{Υ}, \text{Γ} \vdash v \]

[Node Constructor]

\[ \vdash \text{Υ}, \text{Γ} \vdash \overline{n} \rightarrow \text{Υ}, \text{Γ} \vdash \overline{n}' \]

\[ \vdash \text{Υ}, \text{Γ} \vdash c(\overline{n}) \rightarrow \text{Υ}, \text{Γ} \vdash c(\overline{n}') \]

[Macro Invocation Arguments]

\[ \vdash \text{Υ}, \text{Γ} \vdash \overline{n} \rightarrow \text{Υ}, \text{Γ} \vdash \overline{n}' \]

\[ \vdash \text{Υ}, \text{Γ} \vdash t(\overline{n}) \rightarrow \text{Υ}, \text{Γ} \vdash t(\overline{n}') \]

[Macro Invocation]

\[ \text{Υ}(t) = t \overline{x}.n \]

\[ \vdash \text{Υ}, \text{Γ} \vdash t(\overline{v}) \rightarrow \text{Υ}, \text{Γ} [\overline{x} \mapsto \overline{v}] \vdash n \]
Typed Macros (Work in Progress)

- Soudness
  If a macro definition is well typed then there are no type errors at the use sites of the macro unless the user of the macro provides an input of a wrong type.
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.
Summary

- Fortress is a growable language.
- Syntactic abstraction supports the language growth.
- Implementation is available from the website: http://projectfortress.sun.com/
Sukyoung Ryu
sryu@cs.kaist.ac.kr
http://plrg.kaist.ac.kr