Fortress: a growable language for HPC

Victor Luchangco
Programming Language Research Group
Sun Microsystems Laboratories

EPCC Workshop on Novel Parallel Programming Languages for HPC
18 June 2008
Context

- Improve programmer productivity for scientific and engineering applications
- Funded in part by DARPA HPCS program
- Goal: economically viable technologies for government and industrial applications by 2010
High-Performance Computing Today

- High-performance computing in Fortran and C++
  - languages have ill-defined semantics, are unsafe
  - extrinsic concurrency, must be managed explicitly
  - difficult to abstract functionality
  - error-prone and difficult to debug
  - not portable (especially for performance)


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

compute residual norm: \[ ||r|| = ||x - Az|| \]
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
  enddo
sum = 0.0d0
do j=1,firstcol+1
    sum = sum + r(j)*r(j)
  enddo
rho = sum
dcgit = 1,cgitmax
do j=1,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
    w(j) = 0.0d0
  enddo
enddo
do j=1,lastrow-firstrow+1
  sum = 0.d0
do k=rowstr(j),rowstr(j+1)-1
    sum = sum + a(k)*z(colidx(k))
  enddo
w(j) = sum
enddo
do j=1,lastcol-firstcol+1
    r(j) = w(j)
  enddo
enddo
sum = 0.0d0
do j=1,lastcol-firstcol+1
    z(j) = z(j) + alpha*p(j)
    r(j) = r(j) - alpha*q(j)
  enddo
sum = sum + d*d
enddo
d = sum
rnorm = sqrt( d )
enddo
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,cgltmax
  do j=1,lastrow-firstrow+1
    sum = 0.0d0
    do k=rowstr(j),rowstr(j+1)-1
      sum = sum + a(k)*p(colidx(k))
    enddo
    w(j) = sum
  enddo
do j=lastcol-firstcol+1
  q(j) = w(j)
endo

reset temporary variable for reuse

do j=1, lastrow-firstrow+1
  w(j) = 0.0d0
endo
sum = 0.0d0
endo
do k=rowstr(j),rowstr(j+1)-1
  sum = sum + a(k)*z(colidx(k))
endo
endo
w(j) = sum
do j=1, lastcol-firstcol+1
  sum = sum + p(j)*q(j)
endo
d = sum
alpha = rho / d
rho0 = rho
endo
do j=1, lastcol-firstcol+1
  z(j) = z(j) + alpha*p(j)
  r(j) = r(j) - alpha*q(j)
endo
endo
sum = 0.0d0
endo
do j=1, lastcol-firstcol+1
  sum = sum + r(j)*r(j)
endo
rho = sum
beta = rho / rho0
endo
do j=1, lastcol-firstcol+1
  p(j) = r(j) + beta*p(j)
endo
do j=lastcol-firstcol+1
endo
d = x(j) - r(j)
endo
sum = sum + d*d
endo
d = sum
rnorm = sqrt( d )

Change How HPC is Done

- Allow scientists to program closer to the problem domain
- Allow for abstraction and reuse without overhead
- Make errors easy to detect and correct
- Make parallelism and data distribution easier to get right
Do for Fortran what Java™ did for C

Some great ideas from the Java™ programming language

- Catch “stupid mistakes”
  - array bounds and null pointer checking
  - automatic storage management
- Platform independence
- Platform-independent multithreading
- Dynamic compilation
- Make programmers more productive
Key Ideas

• Don’t build the language—grow it

• Make programming notation closer to math

• Make parallelism easier to use
Key Ideas

- Don’t build the language—grow it
  > enable incremental development, improvement
- Make programming notation closer to math
  > allow scientists to program closer to problem domain
- Make parallelism easier to use
\begin{align*}
 z &= 0 \\
 r &= x \\
 \rho &= r^T r \\
 p &= r \\
 \textbf{DO} & \ i = 1, 25 \\
 & \quad q = Ap \\
 & \quad \alpha = \frac{\rho}{(p^T q)} \\
 & \quad z = z + \alpha p \\
 & \quad \rho_0 = \rho \\
 & \quad r = r - \alpha q \\
 & \quad \rho = r^T r \\
 & \quad \beta = \frac{\rho}{\rho_0} \\
 & \quad p = r + \beta p \\
 \textbf{ENDDO} \\
 \text{compute residual norm: } & \|r\| = \|x - Az\| 
\end{align*}
NAS Parallel Benchmark in Fortress

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

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

\[ cg it = 25 \]
\[ z: E_n := \text{Vector}[E, n](0) \]
\[ r: E_n := x \]
\[ \rho: E := r \cdot r \]
\[ p: E_n := r \]
\[ \text{for } j \leftarrow \text{seq}(1 \# cg it) \text{ do} \]
\[ \quad q = A p \]
\[ \quad \alpha = \rho / (p \cdot q) \]
\[ \quad z += \alpha p \]
\[ \quad \rho_0 = \rho \]
\[ \quad r -= \alpha q \]
\[ \quad \rho := r \cdot r \]
\[ \quad \beta = \rho / \rho_0 \]
\[ \quad p := r + \beta p \]
\[ \text{end} \]

\( (z, ||x - A z||) \)
Mathematical Syntax

- Uses standard mathematical operators
  - including bracketing operators
- Juxtaposition is an operator
- Operators (and functions) can be overloaded
- Special rendering rules (e.g., subscripts)
  \[ \sum_{i=1}^{20} a_i x^i \] vs \[ \text{SUM}[i<1\#20] a[i] x^i \]
- No semicolon
Parallel Constructs in Fortress

- do-also blocks
- tuples (including function/method arguments)
- for “loops”
  > for loops are really just trivial reductions
- reductions
  \[ \sum_{i=1}^{20} a_i x^i \]
- comprehensions
  \[ B = \{ a^2 | a \leftarrow A, \text{prime } A \} \]
Data Distributions

- Tree of abstract regions
- Data and computation associated with a region
- Distributions map indexed data structures to regions
- do–also blocks can specify region
NAS Parallel Benchmark in Fortress

cgit = 25

\[ z : E_n := \text{Vector}[E, n](0) \]

\[ r : E_n := x \]

\[ \rho : E := r \cdot r \]

\[ p : E_n := r \]

\text{for} \ j \leftarrow \text{seq}(1 \# \text{cg}it) \text{ do}

\[ q = A p \]

\[ \alpha = \rho / (p \cdot q) \]

\[ z += \alpha p \]

\[ \rho_0 = \rho \]

\[ r -= \alpha q \]

\[ \rho := r \cdot r \]

\[ \beta = \rho / \rho_0 \]

\[ p := r + \beta p \]

\text{end}

\( (z, \|x - A z\|) \)
Sparse Matrices

(* Compressed sparse row representation *)

object Csr[T extends Number, nat n, nat m]
    (rows: Array1[SparseVector[T, m], 0, n])
    extends Matrix[T, n, m]
opr juxtaposition (self, vec: Vector[T, m]): Vector[T, n] = do
    (* Layout of result should result layout of columns *)
    result = rows.replica[T]()
    result.init(i, r · vec), (i, r) ← rows.indexValuePairs
    result
end
...
end
Sparse Vectors

```plaintext
object SparseVector[J T extends Number, nat n ]
   (mem: Array[(Z32, T), Z32])
   extends Vector[J T, n]
opr · (self, other: Vector[J T, n]) = \sum[(i, v) \leftarrow mem] v other_i
opr · (self, other: SparseVector[J T, n] = do
   f(i: Z32, x: T, y: T, r: T) = r + x y
   self.foldMatches(f, other, 0)
end
   ...
end
```
Sparse Vectors

\[
\text{foldMatches}(f: (\mathbb{Z}^{32}, T, T, T) \rightarrow T, \text{other}: \text{SparseVector}[T, n], r: T): T = \text{do}
\]

\[
\text{var } i_1: \mathbb{Z}^{32} := \text{other}.\text{mem}.\text{bounds}.\text{lower}
\]

\[
\text{var } \text{res}: T := r
\]

\[
\text{var } (j_1: \mathbb{Z}^{32}, v_1: T) := (-2, 0)
\]

\[
\text{for } (j_0, v_0) \leftarrow \text{seq}(\text{mem}) \text{ do}
\]

\[
\text{while } j_1 < j_0 \land i_1 < |\text{other}.\text{mem}| \text{ do}
\]

\[
(j_1, v_1) := \text{other}.\text{mem}_{i_1}
\]

\[
i_1 += 1
\]

\[
\text{end}
\]

\[
\text{if } j_0 = j_1 \text{ then } \text{res} := f(j_0, v_0, v_1, \text{res}) \text{ end}
\]

\[
\text{end}
\]

\[
\text{res}
\]

\[
\text{end}
\]
Generators and Reductions

trait Generator[E] excludes Number
  
  abstract generate[R](r: Reduction[R], body: E → R): R

  ...

end

trait Reduction[R]
  
  abstract empty(): R

  abstract join(a: R, b: R): R

end

trait CommutativeReduction[R] extends Reduction[R] end
Project Fortress

- Open-source project ([http://projectfortress.sun.com](http://projectfortress.sun.com))
  - written in Java™, targets JVM™
  - BSD license
  - includes libraries
  - uses third-party tools
  - active community participation

- Language specification
  - v1.0 simplified to synchronize with implementation
  - continuing to evolve