Vectoral: only by trying new things will we ever get computer languages right

Alan Wray
NASA Ames
What it tries to be . . .

" a language for expressing scientific and engineering computational tasks

" a concise notation for such algorithms, as consistent as possible with normal mathematical notation

" syntactically orthogonal

" a self-documenting balance between terse and verbose

" minimally pre-conceived and thus hopefully novel
A Brief History of the Compiler

ca. 1978: First implementation for Illiac IV
Written in PL/I, generated Illiac binary
Cross-compiler running on IBM 360/67

1979: Illiac cross-compiler ported to VAX-11

1980: Generator for Cray 1 binary
Compiler rewritten in Vectoral to run on Cray

1982: Cyber 205 binary, first cross-compiled on Cray
then ported to run natively on 205

1980's-now: Series of ports to XMP, YMP, Cray 2, C90, J90, T90, SV1

1989: Port to Intel i860-based parallel computers:
Gamma = 32 to 128 processors, Delta = 528 processors

1994: Rewrite of backend to generate C;
A Brief History of the Language

Illiac days: Explicit vector notation, asynchronous I/O, user control of both stack and heap storage

Cray days: Vector "loops" replace explicit vector notation, array operations introduced

C-days: Object-oriented features, multitasking added
Inspirations

Algol, Algol 68, Fortran, PL/I, Pascal, APL, CFD, C++, even Basic ...
Overall compiler structure

- Single pass parser with occasional local re-parsing and parse-ahead
- Parser generates n-tuple intermediate code
- Optimizer makes several passes over the i-code
- Code generator forms binary or C from optimized i-code
- For binary, a scheduler reorders/modifies the instruction stream
Major Contrasts with C

" True multidimensional, variable bounded arrays in static, stack, or heap

" Complex type is native

" Exponentiation is native ( ^ )

" Type precisions can be explicit

" No reserved words

" Fewer delimiters, more agglutinators: (, [], {})

" Richer control, data structures

" Flexible, mnemonic call syntax

" Vectoring, tasking built in

" Many more scientific intrinsics
Earliest (Illiac) Vector Syntax

(* # spans the vector (64 or 128 elements) *)

Static real $A[#, 10, 10], B[#, 10, 10]$

index $i, j, k, m[#]$

...

$A[#, i, j] = B[# + i, j, k]^2 + 2*B[#, m[#], k]$

$A[m[#] > 3, i, j] = 0.$
Revised Vector Syntax

(* n is some variable or constant expression giving the vector length for this part of the code *)

Static real $A[n,10,10]$, $B[n,10,10]$, index $v, i, j, k, m[n]$

...  
For $1 \leq v \leq n$:

$$A[v, i, j] = B[v+i, j, k]^2 + 2 \times B[v, m[v], k]$$

if $m[n] > 3$: $A[v, i, j] = 0$. !
Control Structures 1: if

```plaintext
if boolean : statement_list !

boolean : statement_list ?

boolean : statement_list ;

boolean : statement_list !

boolean : statement_list ?
```
QUIZ: How do you do a plain "else"?
QUIZ: How do you do a plain "else"?

A: Use an empty boolean:

```python
if boolean: then_list!
    else_list!
```
QUIZ: How about "while" or "do ... until"?
QUIZ: How about "while" or "do ... until"?

A: Use looping and empty booleans:

```python
if boolean : while_list ?

if : do_list ;
not(boolean) :
    ?
```
QUIZ: Go to ???
A: Sorry.
Control Structures 2: do case

do case index_expression of :

  case_list1: statement_list1 !

  case_list2: statement_list2 ?

  case_list3: statement_list3 ;

  case_list4: statement_list4 !

  case_list5: statement_list5 ?
Control Structures 3: iteration

For $i = 1$ to $n$ and $-5$ to $-10$ by $-2$ and $17$ :
...
?

For $m \leq j \leq n$ :
...
!

For $n+1 > j > m-1$: -- equivalent to the last for
...
!
Control Structures 3, continued

Conditional vectorization:

For \( m \leq j \leq n \), vectorize if \( n-m > 10 \):

\[
\ldots
\]

\[
!
\]
Control Structures 4: defining procedures

Global index a, b, string s, real r

Local DOIT( real X, Y, Z ) =
:    X = a*Y + b*Z !

If a=b:  DOIT = : X = a*(Y + Z) !!

Global DOIT( string X, Y, Z ) = : X = (a+b)*Y + Z !

DOIT( s, "q", ".") -- calls the 2nd DOIT

DOIT( r, 1., 3.) -- calls the 1st DOIT
Control Structures 4: calling procedures

Local DOIT( real X, Y, Z )

raw call: DOIT( A, B, C)

keyword call: DOIT( Y=B, X=A, Z=C )

mixed call: DOIT( Y=B, C, X=A )
Control Structures 4: calling procedures with default arguments

Local DOIT( real X=A , Y , Z=C )

raw call: DOIT( , B )

keyword call: DOIT( Y=B )
Control Structures 4: calling procedures with prepositional separators

Local DOIT( real X with Y from Z )

raw call: DOIT( A with B from C )
or DOIT( A from C with B )
mixed call: DOIT( A with B, Z=C )
Control Structures 4: disambiguating calls

Local DOIT( real X with Y from Z ),
    DOIT( real x into y and z )

DOIT(1., 2., 3.)               -- compile-time error: ambiguous

DOIT(X=1., 2., 3.)            -- call 1st based on argument name

DOIT(a from c with b)            -- call 1st based on separator(s)

DOIT(1., y=a, b)                -- call 2nd based on argument name
Control Structures 5: Tasking

Dynamic task index $i$

For $1 \leq i \leq n$:
-- a process is spawned for each $i$

For $1 \leq i \leq n$, groups of $m$:
-- a process is spawned for each $m\ i$'s
Data Structures 1: Storage Classes

Memory classes
- **Global**: name known to all compilations in a link
- **Local**: name known throughout a compilation
- **Static**: memory allocated once per run
- **Dynamic**: memory allocated on entering scope

Definitional Classes
- **Symbolic**: compile-time parameterized macro
- **Valoric**: run-time temporary
- **Declarator**: parameterized abstract type declaration
Arrays may be multidimensional, variable bounded:

Global index m, n
Static real f[m, n, n+1],
    complex g[-1 to n, m to 2×m, n+1]
Dynamic integer(16) R[10, n]
Areas are basically instantiated **structs**, but unambiguous references to contained identifiers need not be qualified.

Global dims{index m, n}
Static data{real f[m, n],
    complex g[-1 to n, m to 2×m]}[n+1]

Dynamic {integer(16) R[10, n], f[2,2,3]}

**g[1,2,3]** is the same as **data[3]{g[1,2]}** or **data{g[1,2]}.{3}**
or **data{g[1,2,3]}**

**f[1,2,3]** is however distinct from **data{f[1,2,3]}**.
Data Structures 3: Symbolic

Symbolic \( \pi = 3.14159265358979, \)

\[ \text{CUBEROOT}(x) = \]
\[
\text{if}( x>0: x^{[1./3.]}, x=0: 0 ! : -|x|^{[1./3.]}!), \]

\[ \Delta(A, I) = A[I+1] - A[I-1], \]

\[ \text{Diag[array=D, position]} = \text{array[position, position]} \]

\[ \text{Diag}[A,i] \text{ is the same as } A[i,i] \]

and

\[ \text{Diag[position}=j+k-1] \text{ is the same as } D[j+k-1, j+k-1]. \]
Symbolics can be assigned-to and used as compile-time loop variables, tested, etc., to expand the source code at compile-time:

Symbolic i, j, k=3

$For i=1 to 10 : a[i] = b[i] \times c[k]$
$\text{if mod}(i,2)=0: a[i] = a[i] + 1 !$
$j = i + 1$
$d[j] = a[i]^2$
$\text{if } i<5: k = k+1 !$
$\therefore k = j !$
Data Structures 4: Declarator

Declarator character c = bits(8) c,
Array[n] q = q[n],
real_matrix[m,n] M = real M[m,n],
data_area(x,y) D = D{real x,y},
field(type, variable, dimension) F
    = F{type variable}[dimension]

Local character c_a, c_b[20]
is equivalent to

Local bits(8) c_a, c_b[20]
Declarator character $c = \text{bits}(8) \ c,$

Array[$n$] $q = q[n],$

real_matrix[$m,n$] $M = \text{real} \ M[m,n],$

data_area($x,y$) $D = D\{\text{real} \ x,y\},$

field(type, variable, dimension) $F$

$= F\{\text{type variable}\}[\text{dimension}]

\text{Dynamic complex Array}[12] \ u, v, w,$

real Array[`2, 4`] $U1, U2$

is equivalent to

\text{Dynamic complex} \ u[12], \ v[12], \ w[12],$

real $U1[2, 4], U2[2, 4]$
Data Structures 4: Declarator

Declarator character c = bits(8) c,
Array[n] q = q[n],
real_matrix[m,n] M = real M[m,n],
data_area(x,y) D = D{real x,y},
field(type, variable, dimension) F
    = F{type variable}[dimension]

Static real_matrix[5,10] a, b, c[2]
is equivalent to

Static real a[5,10], b[5,10], c[5,10,2]
Declarator character c = bits(8) c,
Array[n] q = q[n],
real_matrix[m,n] M = real M[m,n],
data_area(x, y) D = D{real x, y},
field(type, variable, dimension) F
    = F{type variable}[dimension]

Static data_area(a, b) A1,
data_area(p[10], q[5,4]) A2{real r[3]}
is equivalent to
Static A1{real a, b},
A2{real p[10], q[5,4], real r[3]}
Data Structures 4: Declarator

Declarator  character c = bits(8) c,
  Array[n] q = q[n],
  real_matrix[m,n] M = real M[m,n],
  data_area(x,y) D = D{real x,y},
  field(type, variable, dimension) F
    = F{type variable}[dimension]

Dynamic  field(type=real  variable=`U,V`  dimension=`nx,ny`) velocity[2]

is equivalent to

Dynamic  velocity{real U,V}[nx, ny, 2]
Declarator $H(m)$ $a = \texttt{hoard} \ a\{\text{index } n, \text{ real } x[m]\} :$

\[
a \rightarrow \texttt{new}(H(m)) \quad a\{n\} = m
\]

Dynamic hoard $h\{\text{real } x, y, z\}$

Dynamic $H(100) \ A$

\[
h \rightarrow \texttt{new}(h) \\
\texttt{subr1}(h, A) \\
A \rightarrow \texttt{renew}(A \text{ as } H(1000)) \\
\texttt{subr2}(h, A)
\]

\[
\texttt{free}(h) \quad \texttt{free}(A)
\]
Declarator

velocities \( V = V\{\text{real } u, v, w \} \),
coordinates \( X = X\{\text{real } x, y, z \} \),
flow_data \( Q = Q\{\text{coordinates } X, \text{velocities } V, \)
flow_data hoard neighbor\[6\] \}

Dynamic flow_data data[nx, ny, nz]

For i=1 to nx: for j=1 to ny: for k=1 to nz:

data\{neighbor[1]\}[i,j,k] == if(i=1 : % ! : data[i-1, j, k] !)
data\{neighbor[2]\}[i,j,k] == if(i=nx: % ! : data[i+1, j, k] !)
data\{neighbor[3]\}[i,j,k] == if(j=1 : % ! : data[i, j-1, k] !)
data\{neighbor[4]\}[i,j,k] == if(j=ny: % ! : data[i, j+1, k] !)
data\{neighbor[5]\}[i,j,k] == if(k=1 : % ! : data[i, j, k-1] !)
data\{neighbor[6]\}[i,j,k] == if(k=nz: % ! : data[i, j, k+1] !)
Array Operations

Global dimensions \{ \text{index m, n, p, q} \}

Declarator \text{mat}[a,b] \text{ M = M}[a,b], \\
\text{thing}(a,b) \text{ T = T}\{ \text{real b}[a]\} [b]

Dynamic \text{mat}[m,n] a,b, \text{ thing}(p,q) \text{ Q, R}

\text{a} = 0.

... 

\text{a} = b + \cos(a/2) + 1

\text{Q} = \text{R}

\text{Q}\{b\}[i] = \text{Q}\{b\}[i] + 1 \\
\text{Q}\{b[j]\} = \text{Q}\{b[j]\} -1
Global dimensions{ index m, n, p, q }, subr(real a[3])

Declarator mat[n1,n2] M = M[n1,n2],

thing(n1,n2) T = T{ real b[n1] }[n2]

Dynamic mat[m,n] a,b, thing(p,q) Q, R, index i,j

\[
a = \begin{bmatrix}
    i+j & \text{for } i=1 \text{ to } 10, & i-j & \text{for } i=11 \text{ to } m \\
\end{bmatrix}
\]

\[
b[2,3] = \begin{bmatrix}
    1,2,3,4, & -i^2+1 & \text{for } i=1 \text{ to } 10 \\
\end{bmatrix}
\]

\[
Q = \begin{bmatrix}
    \{ a[i,j]+b[i,j] & \text{for } i=1 \text{ to } p \} & \text{for } j=1 \text{ to } q \\
\end{bmatrix}
\]

subr( [1., 2., 4.] )