Overall Outline

- Fortran 90 specific topics
- General 7.2 optimizations
- Parallelization
  - Explicit
  - Origin 2000 techniques
- Automatic parallelization

Outline

F90 Specific

- Language differences and similarities
- Known problems
- Debugging
- Performance
  - Comparisons
  - How to get it
- Directives and Array Syntax

7.2 vs. 7.1

Language Differences

- 7.1
  - Minimal extensions to F90
- 7.2
  - Cray pointers supported
  - Many new intrinsics (Cray, MIPSPro F77)
  - Uses new back–end technology
  - Support for Cray Flexible File I/O
- Incompatibilities
  - 7.1 and 7.2 not object–code compatible
  - 7.1 (.kmo) and 7.2 (.mod) F90 modules not compatible

7.2 vs. 7.1

Run–time behavior differences

- List–directed output has different format
  - Trailing zeros are suppressed
  - lists are separated by commas
  - repeat counts used
- Unit * and units 5/6 are not the same
- NAMELIST I/O behaves differently by default
  - NAMELIST expects file to be positioned at the start of the NAMELIST record
  - in 7.2 MR there will be several options to control this
- You'll get the right answer more often

MIPSPro 7 F90 vs. Cray F90

Main differences

- Sizes of objects
  - INTEGER, LOGICAL, REAL 32 bits by default
  - INTEGER(1), INTEGER(2) storage sizes
  - Pointer sizes –n32/–64
- "Typeless" intrinsics slightly different
  - On MIPS, return INTEGER type
  - On Cray, return typeless type
- REAL(16) has different representation
- Cray CHARACTER pointers not yet implemented

Known Problems

To be fixed for 7.2 MR

- ERR= or END= and INTEGER(8) IOSTAT variable gets wrong result
- –pfa fails with ALLOCATABLE arrays and F90 pointers
- Incorrect directive syntax often not marked
- Problem with multiple definitions of generic procedures
  - fix came a day too late for Beta
- Some array syntax dependence analysis done incorrectly
  - case is pretty obscure
  - fix came 2 days after Beta
- –IPA can run into problems with internal procedures

Debugging

What works in 7.2 dbx

- Fortran 77 expression evaluation
- All data types supported
- Module data, F90 pointers, assumed–shape arrays
- Host associated data
- Derived types
  - Can use either . or % to separate components
  - Output looks VAX–like

Debugging

What's missing from 7.2 Beta

- Non–contiguous arrays of objects of size 1,2 incorrect
- Non–contiguous arrays of types containing only CHARACTER data incorrect
- Fully qualifying hosted data or module procedure name
  - If modules A and B each contain ZAP, stop in ZAP stops in ZAP in both A and B
- Pointers and arrays in derived types
  - Need somewhat strange syntax to get at data
- Array syntax

Debugging

Arrays in derived types

```
TYPE ZAP
  REAL,POINTER :: A(:,:)
  REAL         :: B(3,5)
END TYPE
TYPE (ZAP) :: z
```

- Whole array
  - print z%b (or z.b)
  - print *z.a
- Element (2,4)
  - print z.b[2,4]
  - print z.a[0][4][2] (we will try to fix this for 7.2 MR)

Debugging

Putting on my user hat, I'd want...

- Array slices and triplet notation
- All data types to be correct
- Some additional intrinsics
  - Global SUM, PRODUCT, MAXVAL, MINVAL
  - Global ANY, ALL
- Setting breakpoints on generic functions
- Fully–qualified module names

F90 Performance Improvements

(Quetzal Tests)

Test	7.1	7.2	Ratio (higher is better)	
dynamic	1.82	1.27	1.43	
static	0.79	0.81	0.98	
test_big	1.10	0.97	1.13	
channel	320	277	1.16	
cortesa_f90	6.65	10.6	0.63	
cortesa_f90	6.16	4.84	1.27	with –OPT:IEEE_arithmetic=3
gen_romans	1.35	0.85	1.59	
where	35.2	34.1	1.03	
wo_where	35.5	37.3	0.95	
integral	17.1	16.5	1.04	dominated by math library
symmetric	3.20	3.81	0.84	(0.97 w/ IEEE_arithmetic=3)
testlu	78.0	74.0	1.05	
casar_f90	2.10	1.34	1.57	
rnflow	89.9	65.2	1.38	
denali	63.5	16.0	3.97	better code for complex power

Geometric mean: 1.28
monte_carlo and reshape work with 7.2, but not with 7.1

F90 Performance Improvements

Basic Operations

	7.1	7.2	ratio	7.1pfa	7.2pfa	ratio	7.2 scale
Explicit Shape	4.45	2.28	2.0	1.13	0.57	2.0	4.0
Automatic	4.51	2.28	2.0	1.17	0.58	2.0	3.9
Allocatable	7.01	2.31	3.0	14.9	0.60	25	3.9
Assumed−shape	9.07	7.82	1.2	2.20	2.00	1.1	3.9
F90 Pointer	11.8	9.66	1.2	24.0	3.20	7.5	3.0

Test case:

```
REAL,DIMENSION (100,100) :: a,b,c,d
a = b + c * d
d = b * c + a + 12.
```

F90 Performance Improvements

Miscellaneous Codes

- 2D Finite Element CFD code
  - 7.1 time: 21.8
  - 7.2 time: 17.2 (factor of 1.27 better)

- Matrix multiply (Megaflops, 200x200)

	7.1	7.2	ratio	4 threads
explicit shape loops	226	296	1.30	1051
MATMUL intrinsic	258	267	1.03	738

(note: parallel MATMUL is not in 7.2 Beta)

- Anecdotal evidence suggests 5–200% improvement

Getting Performance, I

General Rules

- Most techniques in F77/C apply
  - Optimization level (–O2, –O3)
  - Other switches (–OPT:roundoff=3)
- Where possible, use
  - Explicit–shape arrays
  - Automatic arrays
- Performance on F77 codes is virtually identical

Getting Performance, II

Avoid Fortran 90 Pointers

- Aliasing problems

```
A = B + C
```

$\Rightarrow$

```
TEMP = alloca(..)
TEMP = B + C
A = TEMP
free(TEMP)
```

- Unavoidable consequence of language semantics
  - Sometimes optimizer can track aliasing
  - Global pointers are worst

Getting Performance, III

Use assumed–shape arrays judiciously

- This:

```
SUBROUTINE DOIT(A,B,C)
REAL, DIMENSION(:,:) :: A
REAL, DIMENSION(SIZE(A,1),SIZE(A,2)) :: B,C
```

- Not this:

```
SUBROUTINE DOIT(A,B,C)
REAL, DIMENSION(:,:) :: A,B,C
```

Getting Performance, IV

Why method 1?

- Compiler knows A, B and C are same size ⇒ Enables loop fusion, which increases re−use

```
REAL,DIMENSION (:,:) :: a
REAL,DIMENSION(size(a,1),
                size(a,2)) ::b,c,d
a = b + c * d; d = b * c + a + 12.
        ⇒
DO i = ...; DO j = ...
    a(j,i) = b(j,i) + c(j,i)*d(j,i)
    d(j,i) = b(j,i)*c(j,i) + a(j,i) + 12.0
ENDDO ; ENDDO
```

```
REAL,DIMENSION (:,:) :: a,b,c,d
        ⇒
DO i = ...; DO j = ...
    a(j,i) = b(j,i) + c(j,i)*d(j,i)
ENDDO; ENDDO
DO ii= ...; DO jj= ...
    d(jj,ii) = b(jj,ii)*c(jj,ii) + a(jj,ii) + 12.0
ENDDO ; ENDDO
```

Directives and Array Syntax

What I think is desirable

- Directive indicating next statement has no dependencies
  - Useful for F90 pointers
  - Useful for vector–valued subscripts
- Directive indicating next statement should not be parallelized
- Directive indicating that a particular F90 pointer will never be pointer–assigned.
  - Allows compiler to treat pointer as ALLOCATABLE
  - Useful since derived types cannot contain allocatables

Directives and Array Syntax

What I think is not desirable

- Anything which tries to indicate something about how to parallelize/vectorize a statement.

Why so restricted?

- Purpose of a directive is to exercise "fine control"
- Array syntax abstracts far too much to allow this
- Many different transformations for given expression
  - No simple rules to follow
  - Consider `A(:,:) = B(:,:) + C(:,:)`

  what does it become? At least 4 different ways this may be transformed.

Why no Directives?

A more complex example

```
A(1:m,2:m+1) = A(1:m,1:m)
```

- One loop running from 1 to m, one from m to 1
- Copy into a temp, then copy temp

In 7.2 compiler, no temp needed, but now modify slightly

```
A(1:m,2:m+1) = A(1:m,1:m) +
  SUM(B(1:m,1:m,1:n),DIM=3)
```

Now loop structure quite different.

- A simple array statement may turn into several loop nests, with complicated structure.

Why No Directives?

Grouping

- May try to group things incorrectly

```
A(1:m,2:m+1)  =  A(1:m,1:m)
B(1:m,1:m)    =  B(1:m,2:m+1)
```

- Presence of function calls and some intrinsics makes grouping incorrect or meaningless.

A "Surprising" Example from F95

(which arises in practice)

Consider

FORALL(I=1:N) A(I) = SUM(B(1:I))

Is it this?

```
DO I = 1,N
   T = 0
   DO J = 1,I
      T = T + B(J)
   ENDDO
   A(I) = T
ENDDO
```

Or this?

```
T = 0
DO I = 1,N
  T = T + B(I)
  A(I) = T
ENDDO
```

Why no Directives?

A Better Way

- Make the axis/axes over which parallelism is desired explicit
  - Clearly expresses to compiler the user's intention
  - Forces the user to think through what he really wants
  - Avoids "surprises"
- In most cases, compiler can automatically parallelize
  - Simple cases need no help
  - Complicated cases user would likely guess wrong