

IN:
F source

```
subroutine daerfj(x,fvec)          Fortran90
  double precision x(4), fvec(11)
  ! declare and initialize local variables

  do i = 1, 11
    ! more computations
    fvec(i) = y(i) - x(1)*temp1/temp2
  end do

  ! print some results (PASSIVE statements)
end subroutine daerfj
```

1 Open64/sl

- Focuses on source-to-source transformations
- Derived from SGI's Pro64™ optimizing compiler for IA64
- F90/95 front-end generates WHIRL intermediate language
- Includes back-end that unparses WHIRL into Fortran

```
FUNC_ENTRY <1,20,daerfj_>           WHIRL
...
F8ISTORE 0 T<33,anon_ptr.,8>
F8SUB
  F8F8ILOAD 0 T<11,.predef_F8,8>...
  U8ARRAY 1 8
  U8LDA 0 <2,7,Y> T<32,anon_ptr.,8>
  I4INTCONST 11 (0xb)
  I4I4LDID 0 <2,3,I> T<4,.predef_I4,4>
F8DIV
F8MPY
  F8F8ILOAD 0 T<11,.predef_F8,8>...
  U8ARRAY 1 8
  U8U8LDID 0 <2,1,X>...
  I4INTCONST 4 (0x4)
  I4INTCONST 1 (0x1)
  F8F8LDID 0 <2,4,TEMP1>...
  F8F8LDID 0 <2,5,TEMP2>...
# left-hand-side (elided)
...
```

2 whirl2xaif

- Performs canonicalization, e.g. subroutines to functions (in development)
- Translates control flow and 'active' statements into XAIF; performs scalarization
- Inserts placeholders for 'passive' statements

```
<xaif:CallGraph ...>
  <!-- Scope and symbol tables -->           XAIF
  ...
<xaif:ControlFlowGraph symbol_id="daerfj_"...>
  <!-- BBs for 1) CFG Entry, 2) init statements, 3) DO loop... -->

  <!-- statements inside DO loop -->
  <xaif:BasicBlock vertex_id="4">
    ...
    <xaif:Assignment statement_id="3">
      <xaif:AssignmentLHS ... />

      <xaif:AssignmentRHS>
        <xaif:Intrinsic vertex_id="1" name="sub_scal_scal"/>
        <xaif:VariableReference vertex_id="2">
          <xaif:SymbolReference vertex_id="1" symbol_id="Y"/>
          <xaif:ArrayElementReference vertex_id="2">
            <xaif:Index>
              <xaif:VariableReference vertex_id="1" ... "I" />
            </xaif:Index>
          </xaif:ArrayElementReference>
        <xaif:VariableReferenceEdge source="1" target="2"/>
      </xaif:AssignmentRHS>
      <xaif:Intrinsic vertex_id="3" name="div_scal_scal"/>
      <-- x(1)*temp1/temp2 -->
      <xaif:ExpressionEdge source="2" target="1" position="1"/>
      <xaif:ExpressionEdge source="3" target="1" position="2"/>
    </xaif:AssignmentRHS>
    ...
    <!-- PASSIVE statements after DO loop to print results -->
    <xaif:BasicBlock vertex_id="6">
      <xaif:Marker statement_id="1" annotation="{WHIRL_Id#201}" />
    </xaif:BasicBlock>
  <!-- ControlFlowGraph edges -->
```

- Drives the AD pipeline
- Translates appropriate parts of a Fortran90 program into XAIF, the language of the AD engine
- Generates a transformed Fortran program by translating derivative code from XAIF into WHIRL/Fortran and composing with the original program

OUT:
F' source

```
subroutine daerfj(X,FVEC)          Fortran, adjoint
  use OpenAD_tape
  use active_module
  ...
  type(active) :: FVEC(1 : 11)
  INTEGER(w2f__i8) OpenAD_Symbol_55
  REAL(w2f__8) OpenAD_Symbol_56
  type(active) :: OpenAD_Symbol_39
  ...

  if (our_rev_mode%tape) then
    ! REPLACEMENT 1: original function
  end if
  !
  if (our_rev_mode%adjoint) then
    ! REPLACEMENT 3: adjoint
    DO WHILE(OpenAD_Symbol_47 .LE. OpenAD_Symbol_46)
      !
      integer_tape_pointer = integer_tape_pointer-1
      OpenAD_Symbol_55 = integer_tape(integer_tape_pointer)
      double_tape_pointer = double_tape_pointer-1
      OpenAD_Symbol_56 = double_tape(double_tape_pointer)
      OpenAD_Symbol_39%d = OpenAD_Symbol_39%d+FVEC(&
        &INT(OpenAD_Symbol_55))%d*OpenAD_Symbol_56
      FVEC(INT(OpenAD_Symbol_55))%d = 0.0d0
      !
    END DO
    !
  end if
end subroutine daerfj
```

From template
Use active type

From template

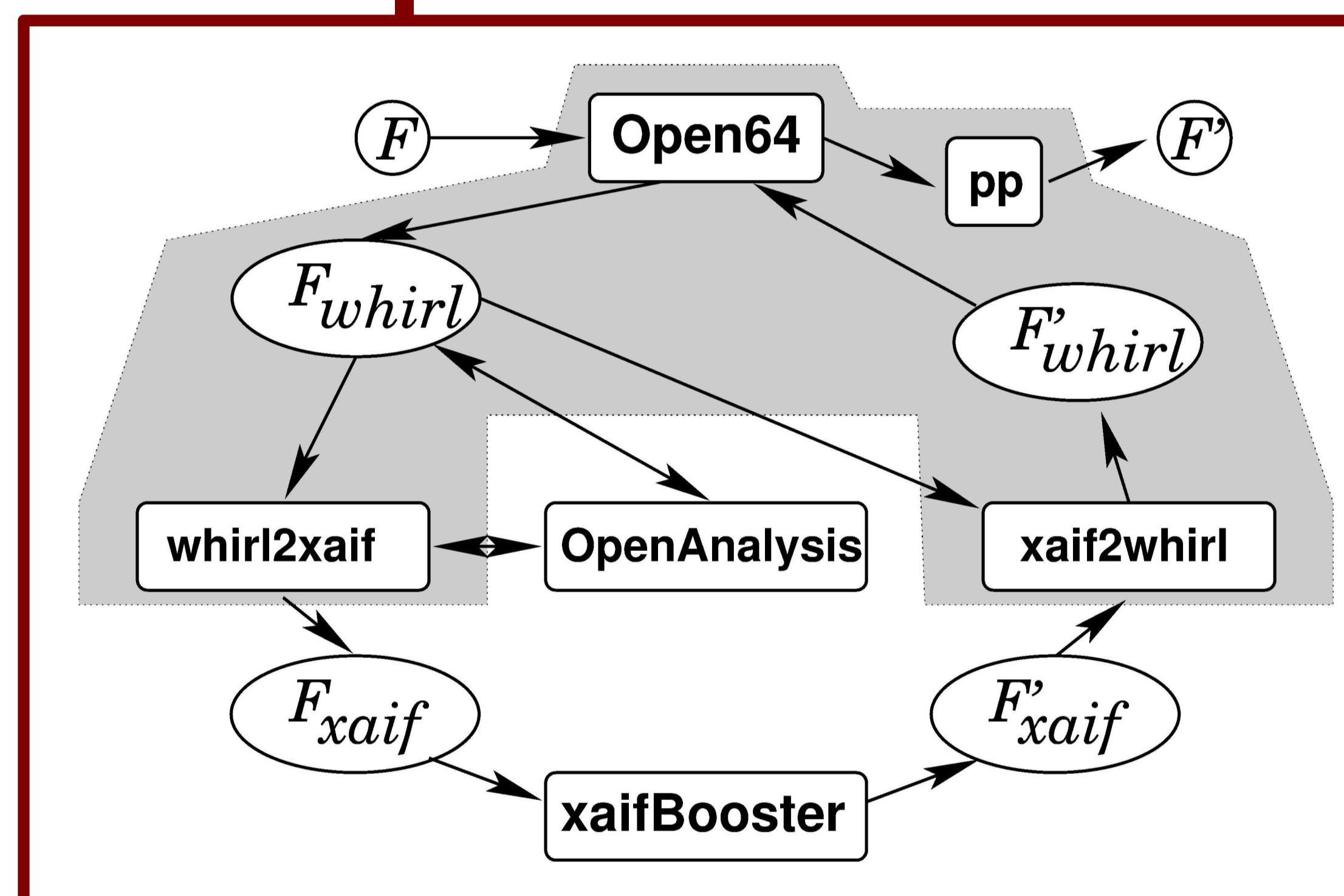
Inline of pop_i
Inline of Pop
Inline of Saxpy
Inline of ZeroDeriv

Fully transformed
source code

6 postprocessor

Benefits of Templating and Inlining:

- Shortens AD algorithm development cycle by hiding details of AD active type, tape, and dynamic call tree from front-end.
- Permits experimentation with AD details without changing either front-end or xaifBooster.
- Results in happy developers and cheap contractors.



3 xaifbooster

4 xaif2whirl

- Composes new derivative code with any original code that could not be represented in XAIF (e.g. passive statements, structured types)
- Creates new control flow structures for adjoint code
- Inserts special directives that drive postprocessing (e.g. active type placeholder; templating and inlining directives)

```
SUBROUTINE daerfj(X, FVEC)
use w2f__types
TYPE (OpenADTy_active) FVEC(1 : 11)
INTEGER(w2f__i8) OpenAD_Symbol_55
REAL(w2f__8) OpenAD_Symbol_56
TYPE (OpenADTy_active) OpenAD_Symbol_39
...
C $OpenAD$ BEGIN REPLACEMENT 1
C $OpenAD$ END REPLACEMENT
C $OpenAD$ BEGIN REPLACEMENT 3
C ...
DO WHILE(OpenAD_Symbol_47 .LE. OpenAD_Symbol_46)
  ...
  C $OpenAD$ INLINE pop_i(subst)
  C CALL pop_i(OpenAD_Symbol_55)
  C $OpenAD$ INLINE Pop(subst)
  C CALL Pop(OpenAD_Symbol_56)
  C $OpenAD$ INLINE Saxpy(subst,subst,subst)
  C CALL Saxpy(OpenAD_Symbol_56, __deriv__(FVEC(INT(
  C $OpenAD_Symbol_55))), __deriv__(OpenAD_Symbol_39))
  C ...
  C ...
END DO
C ...
END SUBROUTINE
```

Fortran fragments and directives
Fortran template (AD algorithm-specific)
Inlinable routines (AD algorithm-specific)

5 Open64/sl whirl2f

- Unparses WHIRL into Fortran