The Fortran-P Translator

1. Basic idea / goal
2. Sample translations
3. Intermediate Representation
4. Structure of translator
5. SORCERER Fortran-P Template / Example

1. Basic idea / goal

- FORTRAN-P Translator translates a restricted subset of FORTRAN-77 into efficient FORTRAN for parallel execution on massively parallel computers.

- Current target: CM-5

- More info: Profs. Paul Woodward and Matt O'Keefe, U of MN; `paul@s1.arc.umn.edu` and `okeefe@everest.ee.umn.edu`

2. Sample translations:

"Tripletization":

```
      do 200 i =  - nbdy + 1, 0
      do 200 k =  - nbdy + 1, ny + nbdy
      rho( i, k, 1, :) = rho( i + nx, k, NODE_X, :)
      p( i, k, 1, :) = p( i + nx, k, NODE_X, :)
      ux( i, k, 1, :) = ux( i + nx, k, NODE_X, :)
      uy( i, k, 1, :) = uy( i + nx, k, NODE_X, :)
200   continue
```

```
C DO 200 i=...
C DO 200 k=...
      rho( -nbdy+1:0, -nbdy+1:ny+nbdy, 1, :) =
     &      rho( nx-nbdy+1:nx, -nbdy+1:ny+nbdy, NODE_X, :)
      p( -nbdy+1:0, -nbdy+1:ny+nbdy, 1, :) =
     &      p( nx-nbdy+1:nx, -nbdy+1:ny+nbdy, NODE_X, :)
      ux( -nbdy+1:0, -nbdy+1:ny+nbdy, 1, :) =
     &      ux( nx-nbdy+1:nx, -nbdy+1:ny+nbdy, NODE_X, :)
      uy( -nbdy+1:0, -nbdy+1:ny+nbdy, 1, :) =
     &      uy( nx-nbdy+1:nx, -nbdy+1:ny+nbdy, NODE_X, :)
200   continue
```

Data layout:

```
      dimension   xxl(-nbdy+1:nx+nbdy)
      dimension   ddd(5, -nbdy+1:nx+nbdy,-nbdy+1:ny+nbdy)
```

```
      real xxl(-nbdy+1:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT xxl(:SERIAL,:NEWS,:NEWS)
      real ddd(5,-nbdy+1:nx+nbdy,-nbdy+1:ny+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT xxl(:SERIAL,:SERIAL,:SERIAL,:NEWS,:NEWS)
```

All non-conformant dimensions are considered local to a processor. The conformant dimensions such as `-nbdy+1:nx+nbdy` are chopped up across the processor grid.

```fortran
      subroutine monslp (a,da,dal,dalfac,darfac,nx,ks,nbdy,noff)
      parameter(iqs=8)
      dimension    a(iqs,1-nbdy:nx+nbdy), da(iqs,1-nbdy:nx+nbdy),
     1           dal(iqs,1-nbdy:nx+nbdy), dalfac(iqs,1-nbdy:nx+nbdy),
     2                                    darfac(iqs,1-nbdy:nx+nbdy)
      mbdy = nbdy - noff
      do 1000   i = 2-mbdy,nx+mbdy
      do 1000   k = 1,ks
 1000 dal(k,i) = a(k,i) - a(k,i-1)
      do 2000   i = 2-mbdy,nx+mbdy-1
      do 2000   k = 1,ks
      dda = dalfac(k,i) * dal(k,i)  +  darfac(k,i) * dal(k,i+1)
      s = sign (1., dda)
      thyng = 2.  *  amin1 (s * dal(k,i),  s * dal(k,i+1))
      da(k,i) = s  *  amax1 (0.,  amin1 (s * dda, thyng))
 2000 continue
      return
      end
```

```
      subroutine monslp(a,da,dal,dalfac,darfac,nx,ks,nbdy,noff)
      include 'nodes.inc'
      parameter(iqs=8)
      real a(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT a(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real da(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT da(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real dal(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT dal(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real dalfac(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT dalfac(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real darfac(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT darfac(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real thyng(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT thyng(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real s(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT s(:SERIAL,:SERIAL,:NEWS,:NEWS)
      real dda(iqs,1-nbdy:nx+nbdy,NODE_X,NODE_Y)
CMF$LAYOUT dda(:SERIAL,:SERIAL,:NEWS,:NEWS)
```

```
      mbdy=nbdy-noff
C DO 1000 i=...
C DO 1000 k=...
1000  dal(1:ks,2-mbdy:nx+mbdy,:,:)=a(1:ks,2-mbdy:nx+mbdy,:,:)-
     &      a(1:ks,2-mbdy-1:nx+mbdy-1,:,:)
C DO 2000 i=...
C DO 2000 k=...
      dda(1:ks,2-mbdy:nx+mbdy-1,:,:)=dalfac(1:ks,2-mbdy:nx+mbdy-1,:,:)*
     &      dal(1:ks,2-mbdy:nx+mbdy-1,:,:)+darfac(1:ks,2
     &      -mbdy:nx+mbdy-1,:,:)*dal(1:ks,2-mbdy+1:nx+mbdy-1+1,:,:)

      s(1:ks,2-mbdy:nx+mbdy-1,:,:)=sign(1.,dda(1:ks,2-mbdy:nx+mbdy-1,:,:))

      thyng(1:ks,2-mbdy:nx+mbdy-1,:,:)=2.*
     &      amin1(s(1:ks,2-mbdy:nx+mbdy-1,:,:)*
     &      dal(1:ks,2-mbdy:nx+mbdy-1,:,:),s(1:ks,2
     &      -mbdy:nx+mbdy-1,:,:)*dal(1:ks,2-mbdy+1:nx+mbdy-1+1,:,:))

      da(1:ks,2-mbdy:nx+mbdy-1,:,:)=s(1:ks,2-mbdy:nx+mbdy-1,:,:)*
     &      amax1(0.,amin1(s(1:ks,2-mbdy:nx+mbdy-1,:,:)*dda(1
     &      :ks,2-mbdy:nx+mbdy-1,:,:),thyng(1:ks,2-mbdy:nx+mbdy-1,:,:)))
2000  continue
      call dummy
      return
      end
```

3. Intermediate Representation

Subroutines:

subroutine *name* (arg_1 arg_2 ... arg_n)

declarations

$stat_1$

$stat_2$

...

$stat_n$

end

where the arguments are optional, has a corresponding tree in the IR of the form:

#(DefSub *name* *arg1 arg2* ... *argn slist*)

Graphically,

$$\begin{array}{l}\texttt{DefSub}\\ \downarrow \\ \textit{name} \longrightarrow \textit{arg1} \longrightarrow \textit{arg2} \longrightarrow \ldots \longrightarrow \textit{argn} \longrightarrow \textit{slist}\end{array}$$

where *slist* is a node representing the root of a list of statements.

Do loops

do *label loop_var* = *begin_expr*, *end_expr* {, *step_expr*}

  slist

label: **continue**

is represented by:

#(Do *label loop_var*

  begin_expr end_expr step_expr slist)

If statements

```
if ( expr ) then
  slist_1
elseif ( expr ) then
  slist_2
elseif ( expr ) then
  slist_3
 ...
else
  else_clause_slist
endif
```

is represented by

```
#( If expr slist_1
  #( If expr slist_2
    #( If expr slist_3 else_clause_slist )
  )
)
```

Structure of the translator

- ANTLR front generates IR, in ASCII file

- Multiple SORCERER phases make passes over IR and write modified IR back to a file with last phase generating output CMF: e.g.,
  `fort t.f | phase1 | phase2 | cmf`

- Easy to test/develop each phase independently, but a bit slower because of file read/write.

- Phases can be merged after independent testing.

Example Fortran-P to text IR:

```
      subroutine f(a,n)
      dimension a(100)
      integer n

      do 100 i=1,n
        if ( i .eq. 7 ) then
          a(i) = 0
        else
          a(i) = 1
        endif
        j = i
100   continue
      return
```

```
#symbols=7
6:[j, Var, 2, ( [Type,Int] [IConst,1] )]
5:[i, Var, 2, ( [Type,Int] [IConst,1] )]
4:[100, Label, 2,]
3:[n, Var, 2, ( [Type,Int] [IConst,1] )]
1:[a, Var, 1, ( [Type,Real] [IConst,100] )]
0:[f, Subroutine, 0,]
#subgraphs=0
#graphs=1
 ( [DefSub,] [Subroutine,0] [Var,1] [Var,2] ( [SLIST,]
 ( [DefDim,] [Var,1] [IConst,100] )
 ( [DefVar,] [Var,3] ( [Type,Int] [IConst,1] ) )
 ( [Do,] [Label,4] [Var,5] [IConst,1] [Var,3] ( [SLIST,]
 ( [If,] ( [EQ,] [Var,5] [IConst,7] ) ( [SLIST,]
 ( [Assign,] ( [ArrayRef,] [Var,1] [Var,5] ) [IConst,0]
) )
 ( [SLIST,]
 ( [Assign,] ( [ArrayRef,] [Var,1] [Var,5] ) [IConst,1]
) ) )
 ( [Assign,] [Var,6] [Var,5] ) ) )
   [Return,] ) )
```

5. SORCERER Fortran-P Template / Example

```
unit:   #( DefSub ( Subroutine | Function | Program )
            (Var)* slist )
    ;

slist:  #( SLIST (stat)+ )
    ;

stat:   #( Assign expr expr )
    |   #( Goto Label )
    |   #( Call Subroutine (expr)* )
    |   Label
    |   Return
    |   Continue
    |   EndDo
    |   #( Include CConst )
    |   Stop
    |   #( Do Label Var expr expr expr slist )
    |   #( If expr slist slist )
    |   #( DefVar Var #( Type (expr)+ ) )
    |   #( DefCom Label Var #( Type (expr)+ ) )
    |   #( Param Var expr )
    |   #( Comment CConst )
    |   #( Cdir CConst )
    ;
```

```
expr:   e_atom
    |   (#( OPSTART..OPEND . . ))?
        #( OPSTART..OPEND expr expr )
    |   #( OPSTART..OPEND expr )
    |   RangeOp
    |   ComSub
    ;

e_atom: Var
    |   ind
    |   IConst
    |   FConst
    |   LConst
    |   #( FuncCall Function (expr)+ )
    |   #( ArrayRef Var (expr)+ )
    ;

ind:    #( Ind ( Var | #(ArrayRef Var (expr)+))) ;
```

Fortran-P LAYOUT Phase

```
stat:   ...
    |   <<int ndim=0, nconformant=0; IRnode *x,*y,*z; int xyz;>>
        #( def:DefVar v:Var
            #(  Type
                (   dim:expr
                    <<
                    ndim++;
                    if (array_bounds_conform(_parser, dim, &xyz))
                        nconformant++;
                    >>
                )+
            )
        )
    ...
    ;

e_atom:    Var
    |   ind
    |   IConst
    |   FConst
    |   LConst
    |   #( FuncCall Function (expr)+ )
    |   #( ArrayRef v:Var (dim:expr)+ )
        <<
        if ( syms[SYM_INDEX(v)].is_distributed )
            ast_append(dim, #(NULL, #[RangeOp], #[RangeOp]));
        >>
    ;
```

Future: Reduction of Array Temporaries

```
      ...
#define rhonu paul1
#define unu paul2
#define utnu paul3
#define dmnu paul6
#define enu paul9
#define dxnu paul11
      Do 8000 i=-j+5,n+j-4,1
      dxnu(i)=xlnu(i+1)-xlnu(i)
#undef xlnu
      dmnu(i)=dm(i)+dmassl(i)-dmassl(i+1)
#undef dmassl
      rhonu(i)=dmnu(i)/dxnu(i)
      dmnui=1.0D+00/dmnu(i)
      enu(i)=(e(i)*dm(i)+denl(i)-denl(i+1))*dmnui
#undef denl
#undef e
      unu(i)=(u(i)*dm(i)+dmoml(i)-dmoml(i+1))*dmnui
#undef dmoml
#undef u
      utnu(i)=(ut(i)*dm(i)+dmomtl(i)-dmomtl(i+1))*dmnui
#undef dmomtl
#undef dm
#undef ut
      ...
```

Conclusions

- Fortran-P translator translates serial programs, written in subset of Fortran 77, to CMF for parallel execution.

- Multiple phases of translator communicate via C file streams, allowing independent phase development and testing.

- ANTLR and SORCERER used for IR generation and manipulation. Latest tool set allowed rapid construction of translator.