A Fortran Code Transformer

Paul Madden

paul.a.madden@noaa.gov
The Scalable Modeling System

- SMS developed mainly 1990-2000
  - Directive-based parallelization of Fortran codes
  - Front-end translator (PPP)
    - Interprets directives and outputs new, parallel code
    - Fix distributed-array loop bounds and array indices, serialize IO statements, allow serial serial regions in otherwise parallel code
  - Back-end library
    - API routines, MPI interface, decomposition information
SMS Examples: sms$distribute

!sms$distribute begin
  real :: u(:)
!sms$distribute end

allocate (u(1:n))

do i=1,n
  u(i)=i
enddo

allocate (u(sms__local_lo:sms__local_hi))

do i=sms__local_lo,sms__local_hi
  u(i)=i
enddo
read (lun) u

if (sms__i_am_root()) then
  allocate (sms__global_u(sms__global_size))
  read (lun) sms__global_u
endif

sms__scatter(sms__global_u,u)
if (sms__i_am_root()) then
  deallocate (sms__global_u)
endif
SMS Examples: sms$serial

sum=0.0
!sms$serial begin
do i=1,n
   sum=sum+u(i)
enddo
!sms$serial end

 Avoid floating-point order-of operation differences

sum=0.0
sms__gather(sms__global_u,u)
if (sms__i_am_root()) then
   do i=1,n
      sum=sum+sms__global_u(i)
   enddo
endif
sms__bcast(sum)
SMS Weaknesses

• Based on Eli compiler toolkit
  – Powerful, but complicated and unfamiliar
  – F77 parser with incomplete F90 extensions

• Issues with
  – F90 features like modules (scoping, \texttt{use, only}), array syntax (array-index correction), kinds
  – Delineation of declaration, executable sections
  – Parallel builds

• Many workaround hacks in model code
Parsing Basics: AST

Code

```plaintext
if (p)
  write (lun) u
return
endif
```

Abstract Syntax Tree

```
if
  condition [p]
  body
  write-stmt
  return-stmt
  write-target [lun]
  variable [u]
```
Parsing Basics: Parse Tree

Code

if (p)
    write (lun) u
return
endif

Parse Tree
aka Concrete Syntax Tree
AST vs Parse Tree

• Simplified AST ideal for translation between different languages
  – e.g. Fortran to C, or C to assembly

• Parse Tree has benefits for translation within one language
  – e.g. Fortran to SMS-Fortran

• Simple to output an *identity* translation from parse tree...
if (p)
    write (lun) u
return
endif
Emit Text via Depth-First Walk

if (p)
    write (lun) u
    return
endif
if (p)
    write (lun) u
return
endif
if (p)
    write (lun) u
    return
endif
if (p)
  write (lun) u
return
endif

Emit Text via Depth-First Walk

if (p)
  write
endif
Parser Technologies

• Recursive Descent parsers
  – Often handwritten in a single language
  – Mutually-recursive procedures to consume and recognize language
  – Easy to write and understand, hard to maintain

• LR parsers
  – Generated from specifications (lex, yacc)
  – Must learn tool languages, understand their error messages
  – Harder to write and understand, easier to maintain
Parsing Expression Grammars

• Introduced by Bryan Ford in 2004
• Best of both worlds: A recursive-descent parser, generated by tools, from specifications
  – Ambiguities, ordering dealt with in grammar
  – Easy to understand and maintain
• Many implementations in various languages
PPP Implementation

• Treetop, a Ruby-based PEG parser generator
  – Developed by Boulder's own Pivotal Labs

• Codebase
  – 6550 loc, core Fortran 90 support
  – 2475 loc, SMS extensions (incl. translation logic)
  – 884 loc, support utilities

• 3374 grammar loc → 38k generated parser loc

• 706 language-recognition tests
  – Including much cruel and unusual Fortran
Treetop PEG: Basic

rule if_stmt
  'if' condition 'then' stmt 'else' stmt /
  'if' condition 'then' stmt
end

rule stmt
  assign_stmt /
  do_stmt /
  if_stmt /
  ... 
end

rule condition
  ...
end
Treetop PEG: Classes and Repetition

rule if_stmt
  if_t condition then_t stmt ( else_t stmt )? <IfStmt>
end

rule if_t
  'if' <T>
end

rule block
  stmt+ <Block>
end

Ruby tree-node
classes
Treetop PEG: Lookahead Assertions and Semantic Predicates

Negative lookahead assertion

rule assign_stmt
  var ']=' val !comma_t
end

Positive semantic predicate

rule array_assignment
  var ']=' var &{ |e| both_arrays?(e[0],e[2]) } end

Semantic predicate only for side-effect

rule hollerith_with_report
  hollerith &{ |e| puts "Found one!"; true } end
Translation via Tree Manipulation

\[ \text{do } i = 1, n \]

\[ \text{do } i = \text{sms\_local\_lo}, \text{sms\_local\_hi} \]
Tree Manipulation Code

do i=1,n
0 12345

do i=sms__local_lo,sms__local_hi
0 123 45

In Ruby:

new_code="do #{e[1]}=sms__local_lo,sms__local_hi"
new_tree=parse(new_code)
replace_node(this,new_tree)

Or, reuse even more elements from the parse tree:

new_code="#{e[0]} #{e[1]}#{e[2]}sms__local_lo#{e[4]}sms__local_hi"
Translation

• First, obtain the parse tree.
• Second, a depth-first walk over the parse tree is performed for translation.
  – Some nodes have `translate` methods, for replacing themselves, recording, and/or error handling
  – Children are translated first, and can record information for later use by ancestor nodes.
• Then, a second depth-first walk over the transformed tree is done to emit translated source, ready for compilation.
During translation walk, we note that distributed array $u$ occurs in the serial region, triggering roll-out of gather code. Similarly for scalar $\sum$ and broadcast code.
Analysis 2

Branch into serial region could permit some tasks from bypassing the collective gather, leading to a hang. So, branches into or out of serial regions are detected and rejected.

if (p) goto 88
!sms$serial begin
doi=1, n
  if (u(i).gt.max) max=u(i)
enddo
88 print *, 'hello'
!sms$serial end

if (p) goto 88
sms__gather(sms__global_u, u)
if (sms__i_am_root()) then
do i=1, n
  if (sms__global_u(i).gt.max) max=sms__global_u(i)
enddo
88 print *, 'hello'
endif
Status

• Outcomes
  - FIM and NIM models updated to use new PPP
  - Many workaround hacks eliminated, better implicit translation of IO statements, further development is now tractable (lower bar to entry with Ruby)

• Issues
  - Some parses technically wrong, need derived-type and F95+ support, better support for other directive families, need unit tests
  - Can't broadcast pointer assignments with MPI
Future Work, Etc.

- Modular for other translator plug-ins
- Produce AST for easier re-use
- Open Fortran Parser / Rose
- Code at https://github.com/maddenp/ppp
Thanks!