HERCULES/PL: The Pattern Language of HERCULES

Christos Kartsaklis
Oscar Hernandez

28 July 2014
Workshop on Programming Language Evolution 2014 (PLE14)
HERCULES Framework

• DoE application challenges in accelerator & exascale era
  – Large production-grade simulation codes need porting
  – Reengineering challenges going beyond parameterization
  – Little or no expectation for compiler readiness

• HERCULES: A user-extensible compiler infrastructure
  – Compiler-supported analysis/transformation tools
  – Help computational scientists with average optimization experience achieve reuse in this context

• Typical scenario: users search for codes similar to what they have optimized in the past
  – Good candidates for similar optimizations
  – How to do this?
User-Level Analysis

• Impractical for users to construct custom analysis by diving into a compiler’s implementation

• Possibly easier to formulate analysis in terms of high-level primitives
  – User analysis as a set of properties that different parts of a program must have

• Mixed-type Compiler Intermediate Representation (IR) CPS
  – A constraint programming system (CPS)
  – Informally referred to as a “pattern”
  – Type of unknowns is drawn from the IR domain (symbols, cfg, …)

• The CPS solution tells you which components of a given program satisfy certain constraints
HERCULES/PL

• Language-level front-end for this compiler IR CPS
  – C/Fortran + Directives

• Incremental
  – User writes a sample program or uses an existing program that resembles what they are looking for
  – User uses HERCULES/PL directives to generalize/specialize or “steer” the pattern derivation process

• Directives
  – `#pragma NAMESPACE` in C, `!$NAMESPACE` in Fortran
  – HPC programmers accustomed to this level of programming (HPF, OpenMP, HMPP, OpenACC)
  – Decorate code with hints, transformation requests, etc.
subroutine driver(J,N,A)
   integer :: I,J,N,A(1:N)
!$hercules pattern declare loop_accesses(symbol A, statement L), statement
!$hercules symbol I,J promote(expression)

!$hercules statement bind L
   do I=1,N
!$hercules statement insert ...
! -- array write here or deeper
!$hercules statement bind ACCESS +nested() +affine(I)
     A(I)=J
!$hercules statement insert ...
   end do
!$hercules pattern declare end
end subroutine

• Find a DO LOOP that contains a nested array write with an index expression that is an affine expression of I and which may be assigned anything.

• HERCULES will tell you in a matching code which loop L (by line number) and which symbol A (by name).
Overview: Declarations

• Signifies which portion/region of the program will be used for constraint derivation (scoping)

• \texttt{!$hercules pattern declare NAME(TYPE ARG,...) [, MODE]}
  – block (default): all the non-declaring statements
  – statement: a single statement
  – expression: the right hand side of an assignment statement
  – \{statement, expression\}_{\text{nested}}: as before, but at any depth

• Declared patterns can be reused via the property mechanism (later)
Overview: Binding

• For the CPS everything you write generates unknowns

• Statement: `for (i=0 ; i<n ; i++) s[i]=0;`
  – 3 symbol unknowns: `i`, `n` & `s`
  – 4 statement unknowns: `for (..)`, `i=0`, `i++` & `s[i]=0`
  – Multiple expression unknowns: `i & 0` in `i=0`, etc.

• Binding assigns ids/labels to these unknowns
  – Symbols default to their actual name
  – Statements can be tagged
  – Statement components can be “reached”: e.g. index of `s[i]`

• `#pragma hercules symbol|statement|type bind`
Overview: Modifying (I)

• Often need to manipulate the syntax tree

• Symbol promotion to expression interprets all symbol “use sites” as expressions
  – A(I) = J in the example with both I & J promoted
  – Using a promoted symbol at multiple sites requires all sites to have identical ASTs
    – #pragma hercules symbol NAME promote

• Statement insertion allows additional statements to be present
  – Inserting “…” to give a floating effect to successors
    – #pragma hercules statement insert EXPR
Overview: Modifying (II)

• Often need to relate or limit elements

• Properties
  – Constraining: what must or must not hold
  – Inspective: find something that exhibits certain behavior
    • E.g. find an array access as opposed to “writing” one
  – \texttt{+PROPERTY(ARG, \ldots)} mechanism & variants (aggregation, negation, etc.)
  – HERCULES/PL does not define who “proves” the property

• Combining with other modifiers for a “foreach” effect
  – To require all sites where a promoted symbol appears to have a property
  – To require all inserted statements to a have a property
Implementation

• Implemented in the Open64 compiler for both C & Fortran using PROLOG as the CPS back-end
  – At VHO time, but deferrable to later stages (e.g. LNO) also
  – Source + Directives compiled into a PROLOG predicate, standard patterns library (SPL) incorporated
  – Target program’s IR “lowered” to PROLOG facts, extra analyses run and/or merged with the db
  – SWIPL invoked on db + predicate, solutions passed to hfe

• HERCULES front-end (hfe) gives choice of
  – Return-to-user: report to shell, e.g. hscan
  – Return-to-compiler: solution → real IR refs + callback
Experiences

• Amount of directives used subject to workflow
  – Plenty if starting from scratch, a lot less if building on existing sources, hardly any if targeting clone detection

• Implementation reveals many compiler challenges
  – Intercepting expression trees with directives
  – HERCULES/PL processing needs to happen before the compiler “lowers” the code
  – CPS back-end choice critical, HERCULES/PL performance

• Application
  – DoE CAM/SE & Sweep3D auto-opt (HIPS)
  – Predictive modeling & custom feature vectors (ICPP)
Current Work

• Insertion & ordering directives for types
  
  ```c
  struct list {
      #pragma hercules type unordered
      struct list* next;
      #pragma hercules type +extrafield(F1)
  };
  ```

• statement ignore keyword

• operator OP ungrouped (no left/right associativity)
  - a+b+c: ((a+b)+c) if L/A; we may not care about order
Questions
Example

subroutine driver()
    integer :: I,J,N
!$hercules pattern declare test(statement FOR1, statement FOR2, list:symbol MSYMS)
!$hercules symbol E1,E2 promote(expression)
!$hercules statement insert ...
!$hercules statement bind FOR1 +body(B1) +B1:reads_only_all_of(MSYMS)
    do I=E1,E2
!$hercules statement insert ...
end do

!$hercules statement insert [ANY] +[]:!writes(MSYMS)

!$hercules statement bind FOR2 +body(B2) +B2:writes_only_all_of(MSYMS)
    do I=E1,E2
!$hercules statement insert ...
end do

!$hercules statement insert ...
!$hercules pattern declare end
end subroutine

do I=2,n-1
    b(i) = a(i-1) + a(i) + a(i+1)
end do

do I=2,n-1
    a(i) = b(i)
end do