TeMa: An Efficient Tool to find High-Performance Library Patterns in Source Code

Christophe Alias
PRiSM, University of Versailles, France
Christophe.Alias@prism.uvsq.fr
http://www.prism.uvsq.fr/~ca
Motivations

• High-Performance Computing requires
  – High Performance Hardware (//, grids, EPIC)
  – Optimized code (loop transf., soft. pipelining ...)

• A solution: High-Performance Libraries!
  – e.g. ATLAS, FFTW, PhiPAC, ...
  – 😊 Increase portability, and lisibility
  – 😞 Tedious task
  – ==> Decide where to put the calls
An example

Given an obfuscated code:

```fortran
DO j = 1,n
  K(0,j) = x(i)
ENDDO
DO i = 1,p
  DO i' = 1,n
    K(i,i') = 0
  END DO
  DO j' = 1,n
    K(i,i') = K(i,i') + A(i',j')*K(i-1,j')
  END DO
END DO
END DO
```

Recover the library calls:

```fortran
DO i = 1,n
  K(i,0) = x(i)
ENDDO
DO i = 1,p
  K(i) = matmul(A,K(i-1))
ENDDO
```
Related Work

- Graph parsing [Wills, 96] [Kim, 98]
  - Program = dependence graph
  - Pattern = grammar rules
  - ==> Recognition = parsing
  - 😊 Parsing tree recovers design decisions
  - 😞 Expensive method
  - 😞 Pattern database maintenance
Related Work

- Specification-driven slicing [Cimetile, 96]
  - Find program slices verifying given Pre- and Post- conditions
  - Based on symbolic execution & theorem proving
  - 😊 Detect a large amount of pattern variations
  - 😞 Finding invariants is undecidable
    - => User interaction needed
  - 😞 Expensive method
Related Work

- AST normalization [Metzger, 2001]
  - Extract «computationnaly confluent» slices
  - Normalize pattern's and slice's ASTs
  - Then compare their ASTs
  - 🥰 Quick method
  - 😞 Few variations hurt the matching
Our approach ...

- Pattern = program with «wild-cards»:
  - \( s = a(0) \)
    - \( \text{DO } i = 1, n \)
      - \( s = \square ( s, a(i) ) \)
    - \( \text{ENDDO} \)
    - \( \text{RETURN } s \)
  - Patterns are 
    - \textit{naive versions} of library functions

- Finds all pattern \textit{instances} in the program
  - And yields the values of \( \square \)
Example of matching

- **Template**
  
  ```
  s = 0
  DO i = 1,n
      s = □ ( s , a(i) )
  ENDDO
  RETURN s
  ```

  $$\Rightarrow \square(x,y) = x + y^5$$

- **Program**
  
  ```
  s = 0
  DO i = 1,n
      p = 1
      DO j = 1,5
          p = p * a(i)
      ENDDO
      s = s + p
  ENDDO
  RETURN x + y
  ```

  $$\Rightarrow \sum a(i)^5$$
Applications

- **Program optimization**
  - *Replace a naive implementation by an optimized version*

- **Hard/soft partitioning**
  - *Detection of treatments which could be performed by hardware*

- **Compiler verification**
  - *Equivalence before and after transformation*

- **Re-engineering & Software maintenance**
Overview of TeMa

Pattern

Program

SSA-graph generation

Pattern detection

Candidate slices

Exact Instantiation Test

Equivalent slices & wild-card values
Pattern detection

- Step the pattern and the program along flow-dependences:

  \[ s = 0 \]
  \[ \text{DO } i = 1, n \]
  \[ s = \square ( s, I ) \]
  \[ \text{ENDDO} \]
  \[ \text{RETURN } s \]

- Pattern wild-cards absorb program parts.
  - Approximate data-flow
  - Exact data-flow
Principle (1/2)

- Dependences are captured in tree-automata:

\[
\begin{align*}
  &s = 0 \\
  &\textbf{DO } i = 1, n \\
  &\quad s = \square(s, I) \\
  &\textbf{ENDDO} \\
  &\textbf{RETURN } s
\end{align*}
\]

<table>
<thead>
<tr>
<th>param</th>
<th>0</th>
<th>(\rightarrow)</th>
<th>(Q\square)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td>(\rightarrow)</td>
<td>(Q\square)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>def.</th>
<th>1</th>
<th>(\rightarrow)</th>
<th>(Q\square)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\ast) ((Q\square, Q\square))</td>
<td>(\rightarrow)</td>
<td>(Q\square)</td>
<td></td>
</tr>
<tr>
<td>(\ast) ((Q\square, Q\square))</td>
<td>(\rightarrow)</td>
<td>(Q\square)</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
  &s = 0 \\
  &\textbf{DO } i = 1, n \\
  &\quad p = 1 \\
  &\textbf{DO } j = 1, 5 \\
  &\quad p = p \ast l \\
  &\textbf{ENDDO} \\
  &\textbf{ENDDO} \\
  &\textbf{RETURN } s
\end{align*}
\]

\[
\begin{align*}
  &S1 \quad \ast (1, I) \rightarrow QP \\
  &S1 \quad \ast (QP, I) \rightarrow QP \\
  &S2 \quad + (0, QP) \rightarrow QS \\
  &S2 \quad + (QS, QP) \rightarrow QS
\end{align*}
\]
Principle (2/2)

- Then dependences are stepped by using cartesian product:

\[
\begin{align*}
    s &= 0 \\
    \text{DO } i &= 1, n \\
    s &= \Box (s, I) \\
    \text{ENDDO} \\
    \text{RETURN } s
\end{align*}
\]

- Candidates are program states with \( \text{Qend} \)
Slices detected

- All slices computing the **same formula**, syntactically (Herbrand-equivalence).

\[
\begin{align*}
\text{DO } & i = 1, 10 \\
s & = s + a(i) \\
\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
\text{DO } & i = 1, 5 \\
s & = s + a(i) \\
\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
\text{DO } & i = 6, 10 \\
s & = s + a(i) \\
\text{ENDDO}
\end{align*}
\]

\[
( \ldots (0 + a(1)) + \ldots ) + a(10)
\]
Organization variations are detected

Temporaries, Legal permutations, Garbage code

```
s = a(0)  
c = 0
DO i = 1, n
  s = s + a(i)
  c = c + 1
ENDDO
RETURN s + c
```

```
s = a(0)  
c = 0  
garbage = 0
DO i = 1, n
  c = c + 1
  temp = a(i)
  DO j = 1, p
    garbage = garbage + a(j)
  ENDDO
  s = s + temp
ENDDO
OUTPUT = s + c
```
Data-structure variations are detected

Array, Non-recursive structures

s(0) = a(0)
DO i = 1, 2*n
  s(i) = s(i-1) + a(i)
ENDDO
RETURN s(2*n)

v.s1 = a(0)
DO i = 1, n
  IF i<n/2 THEN
    v.s1 = v.s1 + a(i)
  ELSE
    v.s2 = v.s2 + a(i)
  ENDIF
ENDDO
ENDDO
OUTPUT = v.s1 + v.s2
Control variations are detected

peeling, splitting / fusion, tiling ...

\[ s = a(0) \]
\[ \text{DO } i = 1, n \]
\[ s = s + a(i) \]
\[ \text{ENDDO} \]
\[ \text{RETURN } s \]

\[ s = a(0) \]
\[ \text{DO } i = 1, n-1 \]
\[ s = s + a(i) \]
\[ \text{ENDDO} \]
\[ s = s + a(n) \]
\[ \text{OUTPUT } = s \]
Control variations are detected

peeling, splitting / fusion, tiling ...

\[
s = a(0) \\
\text{DO } i = 1,n \\
\quad sa = sa + a(i) \\
\quad sb = sb + b(i) \\
\text{ENDDO} \\
\text{RETURN } sa + sb
\]

\[
s = a(0) \\
\text{DO } i = 1,n \\
\quad sa = sa + a(i) \\
\text{ENDDO} \\
\text{DO } i = 1, n \\
\quad sb = sb + b(i) \\
\text{ENDDO} \\
\text{OUTPUT } = sa + sb
\]
Control variations are detected
peeling, splitting / fusion, **tiling** ...

```
s = a(0)
DO i = 1,n
  s = s + a(i)
ENDDO
RETURN s
```

```
s = a(0)
DO i = 1,n , tile
  DO it = i,min(n,i+tile-1)
    s = s + a(it)
  ENDDO
ENDDO
ENDDO
OUTPUT = s
```
Control variations are detected

IF extraction

\[
s = 0
\]

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

\[
\text{IF } a_{\text{sum}} \text{ THEN}
\]

\[
s = s + a(i)
\]

\[
\text{ELSE}
\]

\[
s = s + b(i)
\]

\[
\text{ENDIF}
\]

\[
\text{ENDDO}
\]

\[
\text{IF } a_{\text{sum}} \text{ THEN}
\]

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

\[
s = s + a(i)
\]

\[
\text{ENDDO}
\]

\[
\text{ELSE}
\]

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

\[
s = s + b(i)
\]

\[
\text{ENDDO}
\]

\[
\text{ENDIF}
\]
Control variations are detected

IF-conversion

\[
s = 0 \\
\text{DO } i = 1, n \\
\quad \text{IF } a_{\text{sum}} \text{ THEN} \\
\quad \quad s = s + a(i) \\
\quad \text{ELSE} \\
\quad \quad s = s + b(i) \\
\quad \text{ENDIF} \\
\text{ENDIF} \\
\text{ENDDO}
\]
Experimental results

- Matching **BLAS 1 & 2** on the **SPEC FP**
Analysis of the results (1/2)

- 50% of wrong detections
  - Due to approximate dataflow (reaching def.):

\[
\begin{align*}
    &s = 0 \\
    &\text{DO } i = 1,10 \\
    &\quad s = s + a(i) \\
    &\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
    &s = 0 \\
    &\text{DO } i = 1,10 \\
    &\quad \text{DO } j = 1,i \\
    &\quad s = f [s,a(i,j)] \\
    &\text{ENDDO} \\
    &\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
    &s = 0 \\
    &\text{DO } i = 1,5 \\
    &\quad s = s + a(i) \\
    &\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
    &s = 0 \\
    &\text{DO } i = 1,5 \\
    &\quad \text{DO } j = 1,5 \\
    &\quad s = f [s,a(i,j)] \\
    &\text{ENDDO} \\
    &\text{ENDDO}
\end{align*}
\]
Analysis of the results (2/2)

- 25% of trivial detections

\[
\begin{align*}
\text{DO } i &= 1, n \\
y(i) &= a \cdot x(i) + y(i) \\
s &= 2 \cdot a + 1 \\
\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
dot &= 0 \\
\text{DO } i &= 1, n \\
s &= 0 \\
dot &= \text{dot} + a(i) \cdot b(i) \\
\text{ENDDO}
\end{align*}
\]

- 25% of right detections, leading to a substitution
Re-engineering SPIRAL

- Domain-specific language for Signal Processing applications
- Multistage compilation

```
DO i = 1, n
... ENDDO
```

![Diagram showing the flow from high-level formula to Fortran program through optimization stages.](image-url)
Conclusions & Future Work

- Automatic detection of high-performance library functions
- Tolerates many program variations
  - Organization, data-structure & control variations
- Validated on BLAS/SPEC FP
- TODO: Substitute detected portions by a call
  - Source-to-source optimization
  - Re-engineering & Maintenance (SPIRAL)
Dataflow information?

- **Approximate**
  - \( s = S1 \) or \( S2 \)

- **Exact**
  - \( s = \left< S1, \right>_{i=1} \)
  - \( \left< S2, i-1 \right>_{i=2..n} \)

S1  \( s = 0 \)
DO  \( i = 1,n \)
S2  \( s = s + a(i) \)
ENDDO