

# 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](mailto: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:

```

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

```

Recover the library calls:

```

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»:
  - $s = a(0)$   
**DO**  $i = 1, n$   
   $s = \square ( s , a(i) )$   
**ENDDO**  
**RETURN**  $s$
  - *Patterns are **naive versions** of library functions*
- Finds all pattern **instances** in the program
  - And yields the values of  $\square$

# Example of matching

- Template

$s = 0$

**DO**  $i = 1, n$

$s = \square (s, a(i))$

**ENDDO**

**RETURN**  $s$

$\implies \square(x, y) =$

```

p = 1
DO j = 1,5
  p = p * y
ENDDO
RETURN 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**

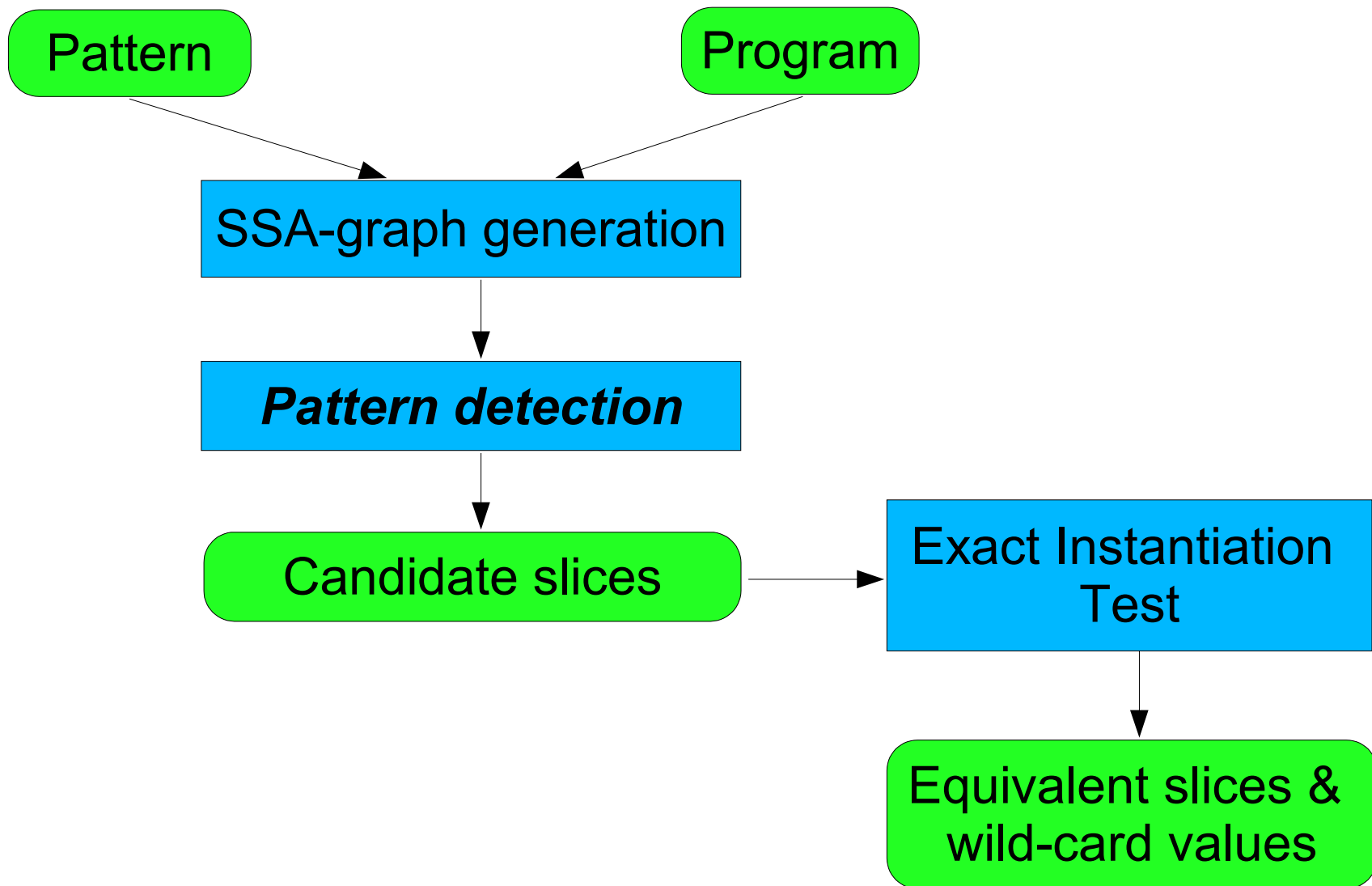
...

$= \sum a(i)^5$

# Applications

- **Program optimization**
  - *Replace a naive implementation by an optimized version*
- **Hard/soft partitionning**
  - *Detection of treatments which could be performed by hardware*
- **Compiler verification**
  - *Equivalence before and after transformation*
- **Re-engineering & Software maintenance**

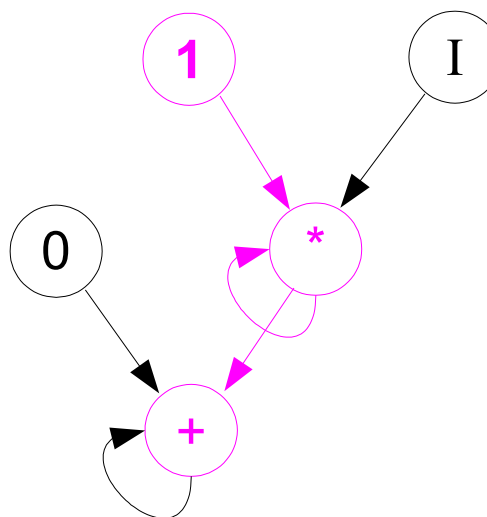
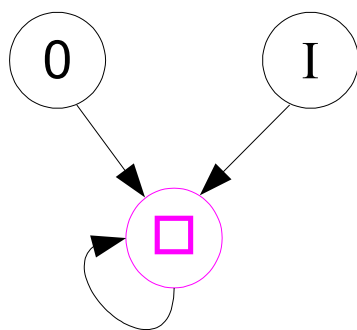
# Overview of TeMa



# Pattern detection

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

```
s = 0
DO i = 1,n
  s = □ (s, I)
ENDDO
RETURN s
```

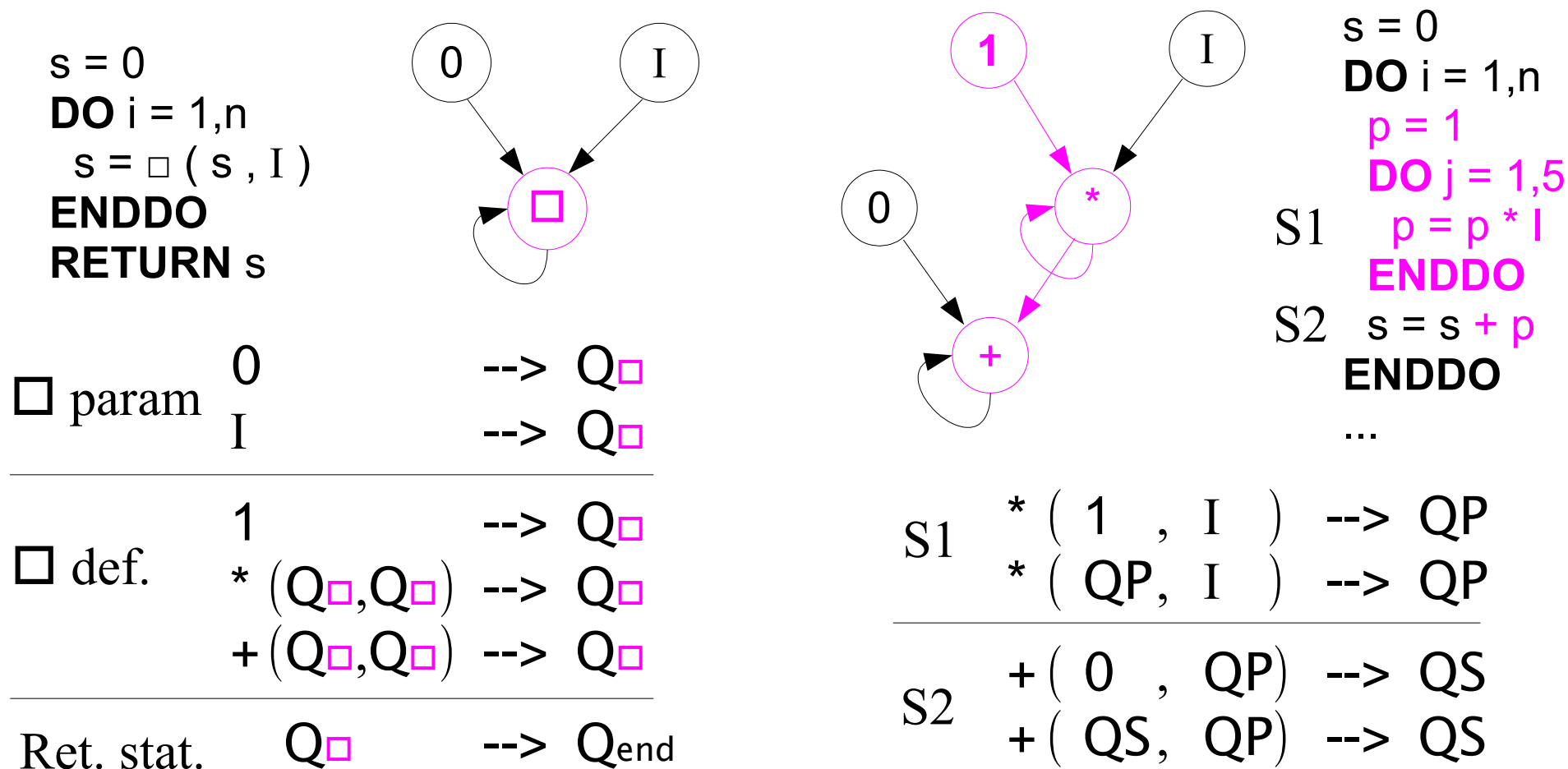


```
s = 0
DO i = 1,n
  p = 1
  DO j = 1,5
    p = p * I
  ENDDO
  s = s + p
ENDDO
...
```

- Pattern wild-cards absorb program parts.
  - *Approximate data-flow*      Pattern detection
  - *Exact data-flow*              Instantiation Test

# Principle (1/2)

- Dependences are captured in tree-automata:



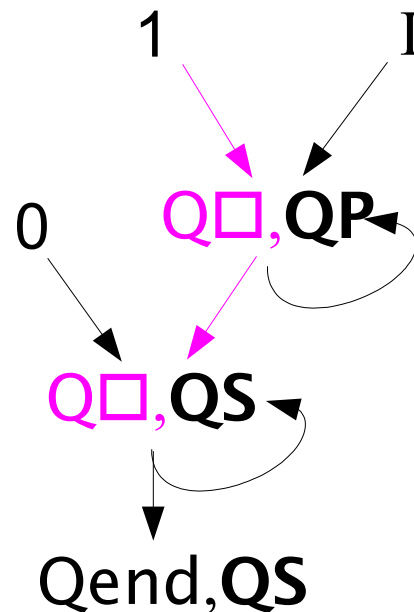
# Principle (2/2)

- Then dependences are stepped by using **cartesian product**:

```

s = 0
DO i = 1,n
  s = □ (s, I)
ENDDO
RETURN s

```



```

s = 0
DO i = 1,n
  p = 1
  DO j = 1,5
    p = p * I
  ENDDO
  s = s + p
ENDDO
...

```

- Candidates are program states with Qend

# Slices detected

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

```
s = 0
DO i = 1, 10
  s = s + a(i)
ENDDO
```

```
s = 0
DO i = 1, 5
  s = s + a(i)
ENDDO
DO i = 6, 10
  s = s + a(i)
ENDDO
```

( ... (0 + a(1)) + ... ) + 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
OUTPUT = v.s1 + v.s2
```

# 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-1
  s = s + a(i)
ENDDO
s = s + a(n)
OUTPUT = s
```

# Control variations are detected

peeling, **splitting / fusion**, tiling ...

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

```
s = a(0)
DO i = 1,n
  sa = sa + a(i)
ENDDO
DO i = 1, n
  sb = sb + b(i)
ENDDO
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
OUTPUT = s
```

# Control variations are detected

## IF extraction

```
s = 0
DO i = 1,n
  IF a_sum THEN
    s = s + a(i)
  ELSE
    s = s + b(i)
  ENDIF
ENDDO
```

```
s = 0
IF a_sum THEN
  DO i = 1,n
    s = s + a(i)
  ENDDO
ELSE
  DO i = 1,n
    s = s + b(i)
  ENDDO
ENDIF
```

# Control variations are detected

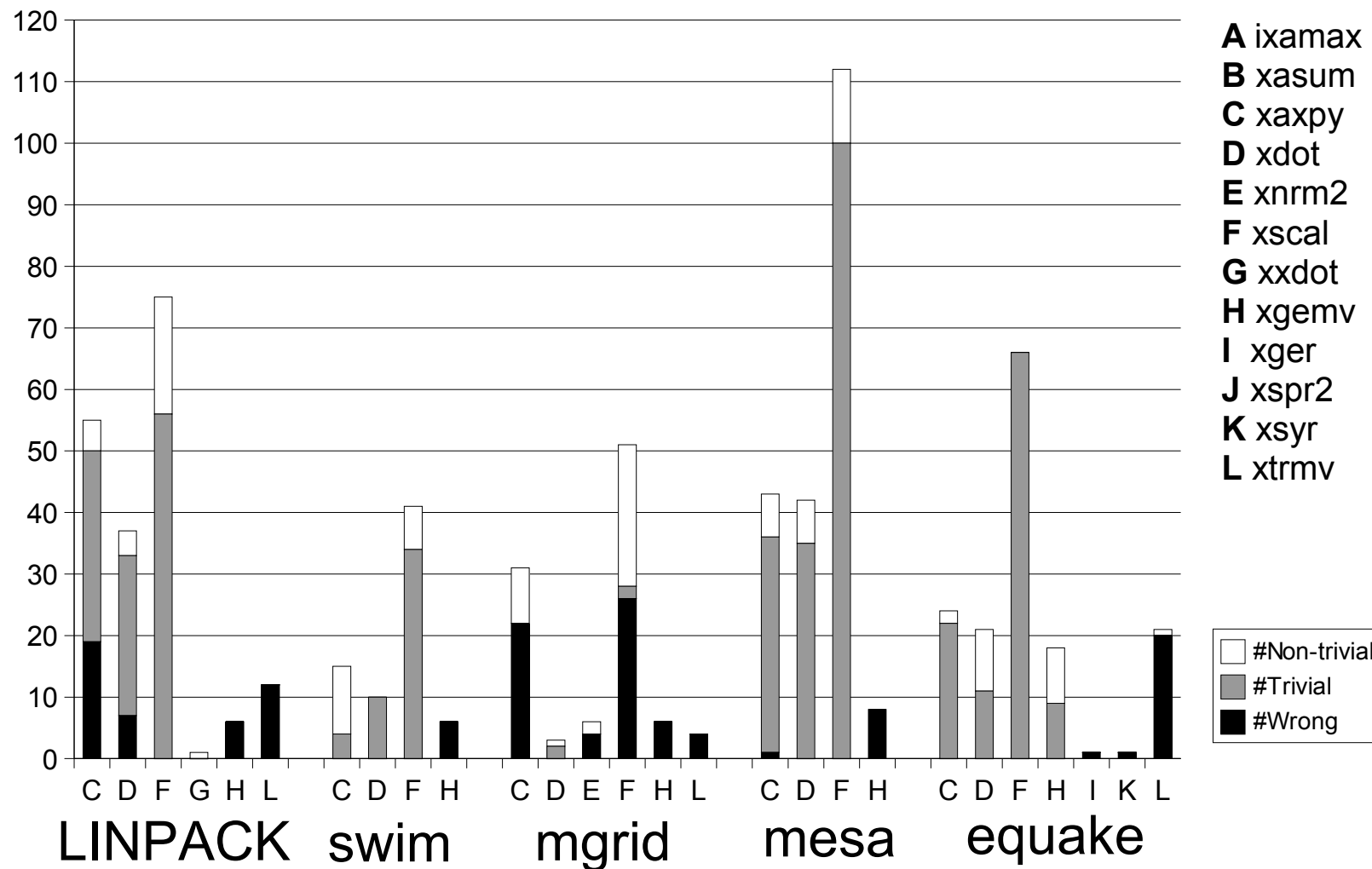
## IF-conversion

```
s = 0
DO i = 1,n
  IF a_sum THEN
    s = s + a(i)
  ELSE
    s = s + b(i)
  ENDIF
ENDDO
```

```
s = 0
DO i = 1,n
  IF a_sum THEN
    s = s + a(i)
  ENDIF
  IF NOT(a_sum) THEN
    s = s + b(i)
  ENDIF
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.):

```
s = 0
DO i = 1, 10
  s = s + a(i)
ENDDO
```

```
s = 0
DO i = 1, 5
  s = s + a(i)
ENDDO
```

```
s = 0
DO i = 1, 10
  DO j = 1, i
    s = f [s, a(i, j)]
  ENDDO
ENDDO
```

```
s = 0
DO i = 1, 10
  DO j = 1, 5
    s = f [s, a(i, j)]
  ENDDO
ENDDO
```

# Analysis of the results (2/2)

- 25% of trivial detections

```

DO i = 1,n
  y(i) = a*x(i) + y(i)      s = 2*a + 1
ENDDO

```

```

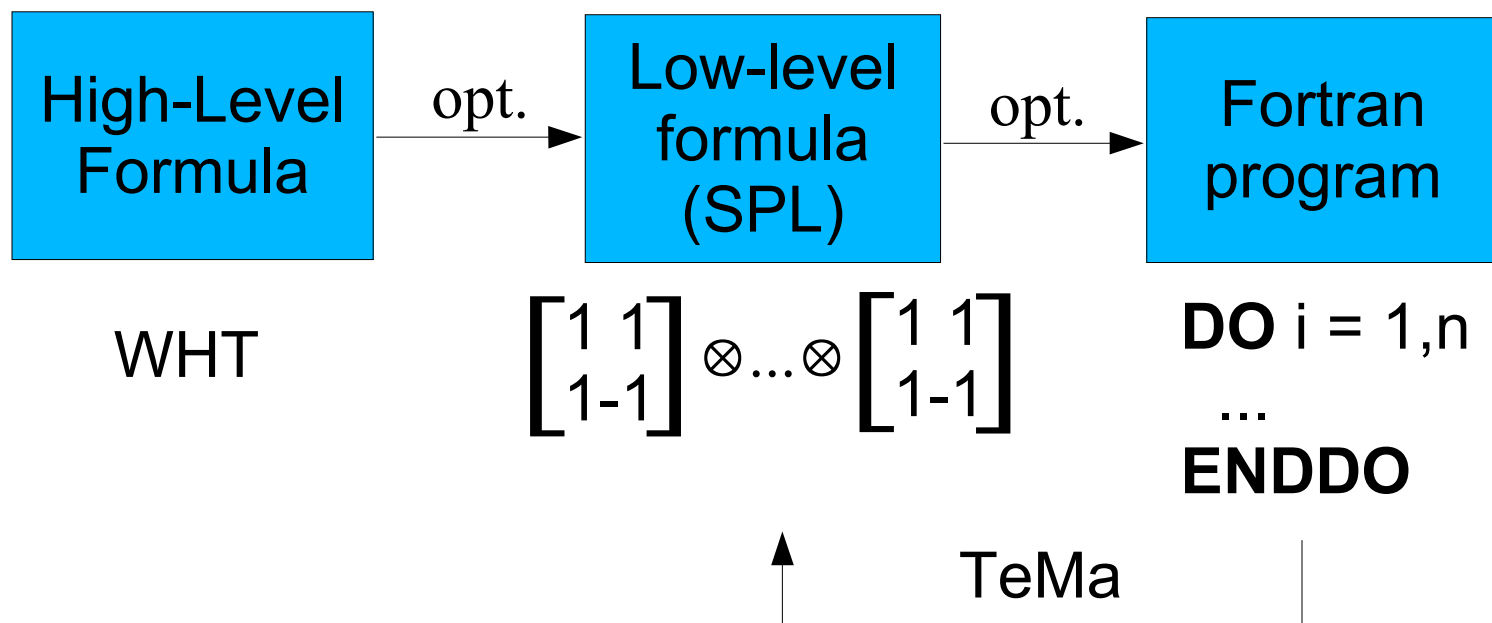
dot = 0
DO i = 1,n                s = 0
  dot = dot + a(i)*b(i)
ENDDO

```

- 25% of right detections, leading to a substitution

# Re-engineering SPIRAL

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



# 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 ?

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

- Approximate

- $s = S1$  or  $S2$

- Exact

- $s = \langle S1, \rangle \quad i=1$   
 $\langle S2, i-1 \rangle \quad i=2..n$