GIANT
INNOVATION CAMPUS

Profiling Guided Hybrid Compilation

Diogo Sampaio

Inria - UJF/CAPES

September 24, 2015

Profiling Guided Hybrid Compilation

Introduction Goal: Generate faster code

Profiling Guided Hybrid Compilation

Introduction Goal: Generate faster code

How: Use of compiler and mathematical tools

- Static analyses
- Profiling
- Polytope model

Profiling Guided Hybrid Compilation

Compiler Optimization and Runtime SystEms

Introduction Goal: Generate faster code

How: Use of compiler and mathematical tools

- Static analyses
- Profiling
- Polytope model

Methodology: Use of run-time information to guide static optimizer

- Code versioning
- Fast run-time tests

Profiling Guided Hybrid Compilation

Presentation outline

Profiling Guided Hybrid Compilation

Outline

Profiling Guided Hybrid Compilation

Compiling process

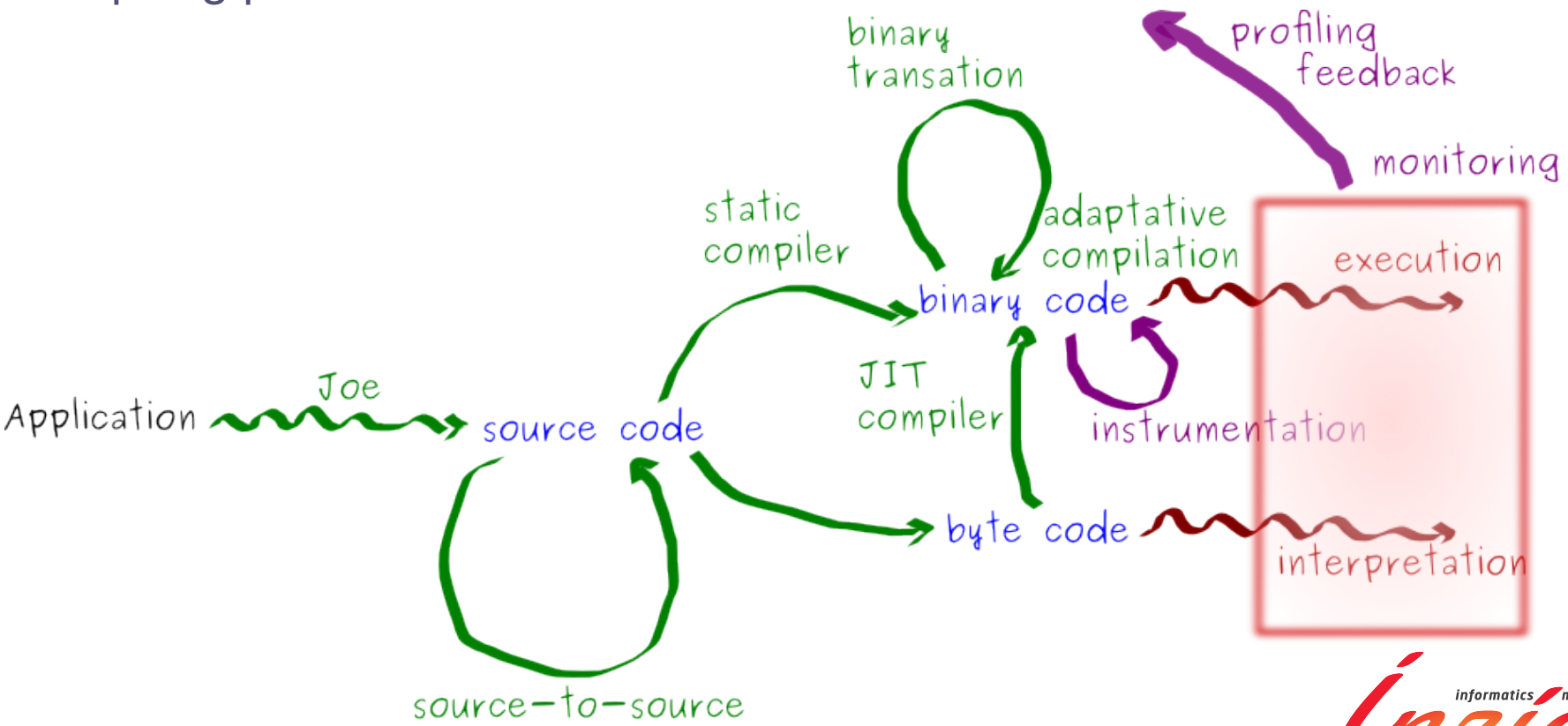
Compiler (Wikipedia):"A compiler is a computer program that transforms source code written in a programming language into another computer language."

"A compiler is likely to perform many or all of the following operations: lexical analysis, preprocessing, parsing, semantic analysis, code generation, and code optimization."

Profiling Guided Hybrid Compilation

Compiling process

Compiling process overview:



Profiling Guided Hybrid Compilation

Compiler static analyses and optimizations

- Based on facts observed on the code
- Ahead time - Free in resource usage
- Have access to abstract code structures (loops, basic blocks...)
- Limited knowledge of application behavior and input

Profiling Guided Hybrid Compilation

Dynamic analyses and optimizations

- Based on facts observed at run-time
- Access to values of variables
- Can do profiling of the program
- Limited resources usage
- Limited knowledge of the application structure

Profiling Guided Hybrid Compilation

Hybrid Compilation

- Multi-step compilation process
- Join knowledge of static and dynamic analyses
- Program execution traces provide run-time information

Profiling Guided Hybrid Compilation

Hybrid Compilation

- Multi-step compilation process
- Join knowledge of static and dynamic analyses
- Program execution traces provide run-time information
- Observed behavior is input dependent
- Instrumented code executes much slower
- The transformation space is huge

Profiling Guided Hybrid Compilation

Outline

Profiling Guided Hybrid Compilation

Profiling Guided Hybrid Compilation

- Focus on loop transformations
- Transformations that improve locality
- Profiling to prune optimization space
- Sampling to limit amount instrumented execution time
- Run-time data guide profitable transformations
- Code versioning to tackle with different situations
- Fast run-time checks guarding each optimization
- Run-time data guide checks if tests are meaningful

Profiling Guided Hybrid Compilation

Outline

Profiling Guided Hybrid Compilation

Loops

- Most execution time of a scientific program is spent on loops
- Many loops nests define a SCoP (static control part)
- And have instructions based on the loop indexes

Stencil

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

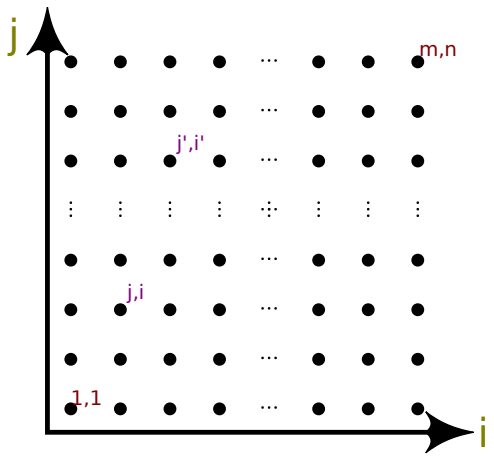
Polytope model

- The Polytope model is the most accurate representation of loops
- Is a mathematical framework for loop nest optimization
- Treats each loop iteration within nested loops as lattice points inside polyhedrons

- Iteration domain constraints
- Order constraints
- Access constraints

Polytope model
Iteration domain constraints

- Number of dimensions depends on loop nest depth
- Inequalities that limit the polyhedrons

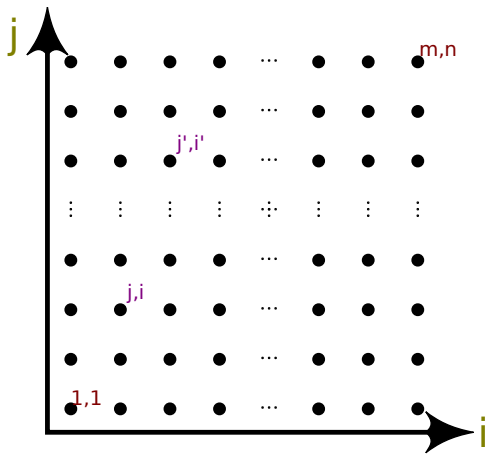


Stencil

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

Polytope model
Iteration domain constraints

- Number of dimensions depends on loop nest depth
- Inequalities that limit the polyhedrons



The iterations (j, i) and (j', i') are in the iteration space if:

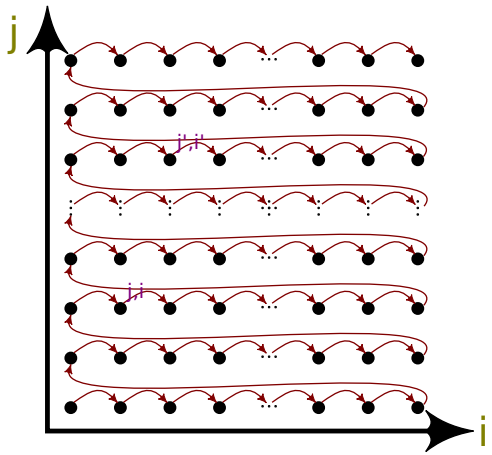
Domain constraints

$$1 \leq j, j' \leq m$$
$$1 \leq i, i' \leq n$$

Profiling Guided Hybrid Compilation

Polytope model
Order constraints

- Lexicographic ordering of the iterations
- Gives the original (identity) execution order to the scheduler



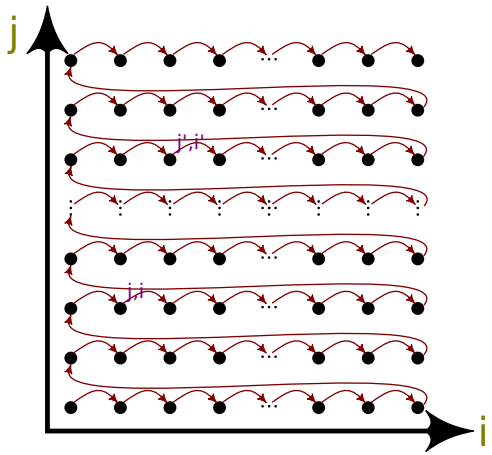
(j, i) is executed before (j', i') then:

Stencil

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

Polytope model
Order constraints

- Lexicographic ordering of the iterations
- Gives the original (identity) execution order to the scheduler



(j, i) is executed before (j', i') then:

Ordering constraints

$$j, i < j', i' \rightarrow j < j' \mid j = j' \ \& \ i < i'$$

Profiling Guided Hybrid Compilation

Polytope model
Access constraints

- Describe the memory accesses
- Depend on the position of the iteration on space
- Two kinds of access: Read and write

The iteration (j, i):

Stencil

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

Profiling Guided Hybrid Compilation

Polytope model
Access constraints

- Describe the memory accesses
- Depend on the position of the iteration on space
- Two kinds of access: Read and write

The iteration (j, i):

Memory access constraints

- Read: $t(i-1, j)$
- Read: $t(i, j-i)$
- Write: $t(i, j)$

Profiling Guided Hybrid Compilation

Polytope model
Dependency

- Write instruction might dependencies
- Dependencies described as distance vector

Distance vector representing dependency:

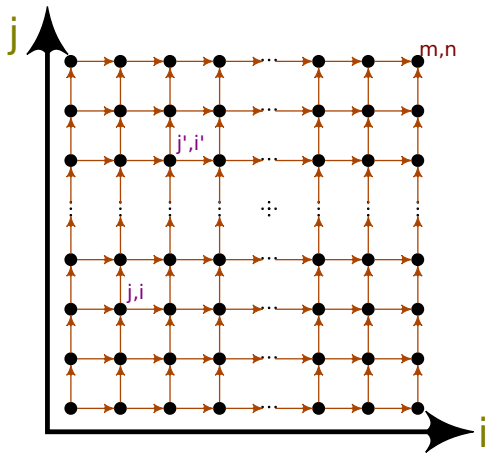
Stencil

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

Profiling Guided Hybrid Compilation

Polytope model
Dependency

- Write instruction might dependencies
- Dependencies described as distance vector



Distance vector representing dependency:

Distance vectors

- $(1, 0)$
- $(0, 1)$

Profiling Guided Hybrid Compilation

Polytope model
Transformations

- Transformation are done by changing the iterations ordering
- Transformation validity: no negative dependencies
- Try to improve locality

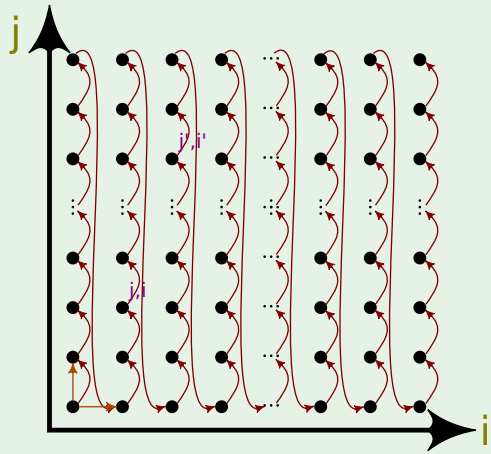
Original Stencil code

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

Polytope model
Transformations

- Interchange example:
- $T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
- $D_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- $D_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
- $T \times D_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
- $T \times D_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

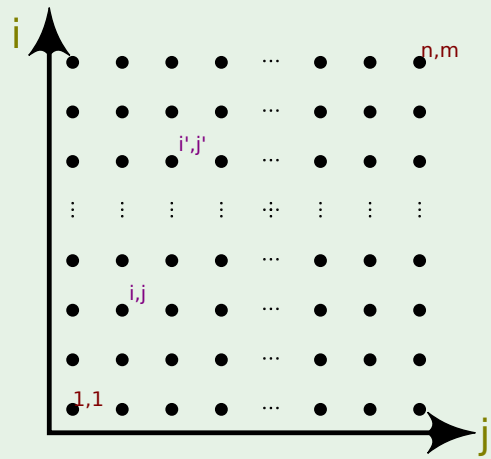
Loop interchange



Polytope model
Transformations

- Interchange example:
- $T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
- $(j, i) = \begin{bmatrix} j \\ i \end{bmatrix}$
- $T \times (j, i) = \begin{bmatrix} i \\ j \end{bmatrix}$

Loop interchange



Profiling Guided Hybrid Compilation

Polytope model
Transformations

- Interchange example:
- $T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
- $(j, i) = \begin{bmatrix} j \\ i \end{bmatrix}$
- $T \times (j, i) = \begin{bmatrix} i \\ j \end{bmatrix}$

Transformed Stencil code

```
for(i=1; i<n; i++)
 for(j=1; j<m; j++)
  t[i][j] = t[i-1][j]
          + t[i][j-1]
```

Profiling Guided Hybrid Compilation

Limitations

Static approach fail on one of these:

- Handle while loops
- Handle pointers
- Non statically known linear memory accesses
- Avoid to create optimizations that can't be used

Profiling Guided Hybrid Compilation

Limitations

Dynamic and hybrid approach fail on one of these:

- Low cost profiling
- Low code size overhead
- Low overhead of run-time tests

Profiling Guided Hybrid Compilation

Outline

Profiling Guided Hybrid Compilation

Instrumentation

Change the code to observe run-time behavior

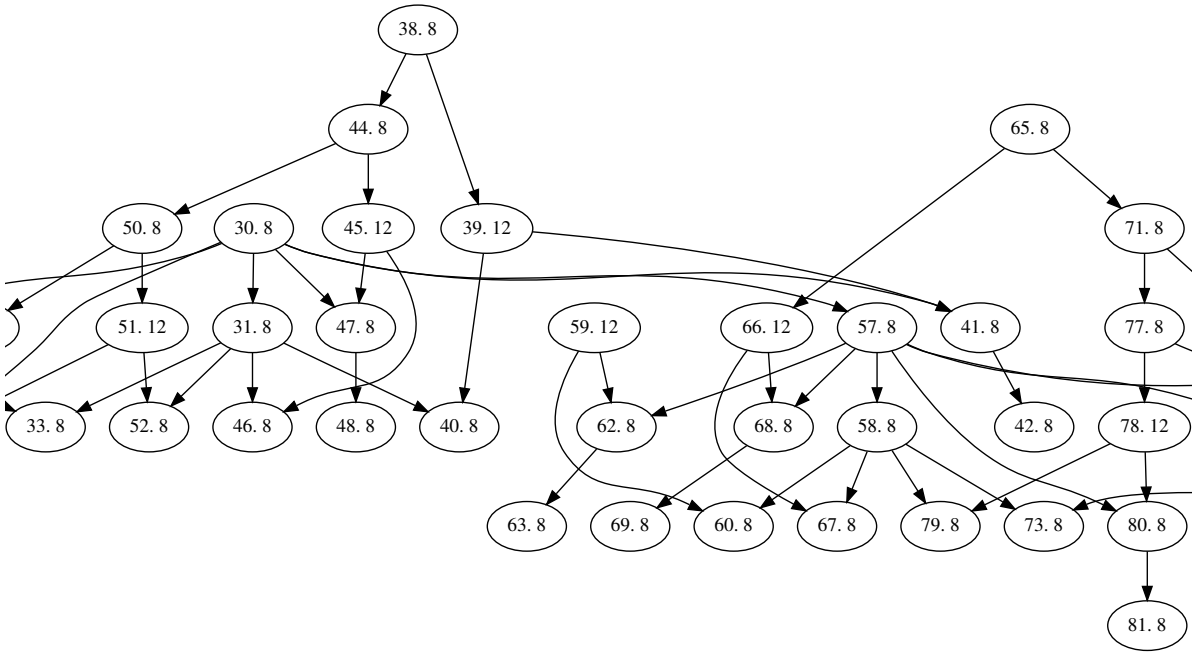
Write to output file:

- Basic block execution
- Loop indexes (create one if required)
- Function calls and returns (if accessible source)
- Memory access addresses
- Limit number of iterations of loops

Linearized Stencil

```
for(j=1; j<m; j++)
 for(i=1; i<n; i++)
  t[i*H+j] = t[H*i-H+j]
           + t[i*H+j-1]
```

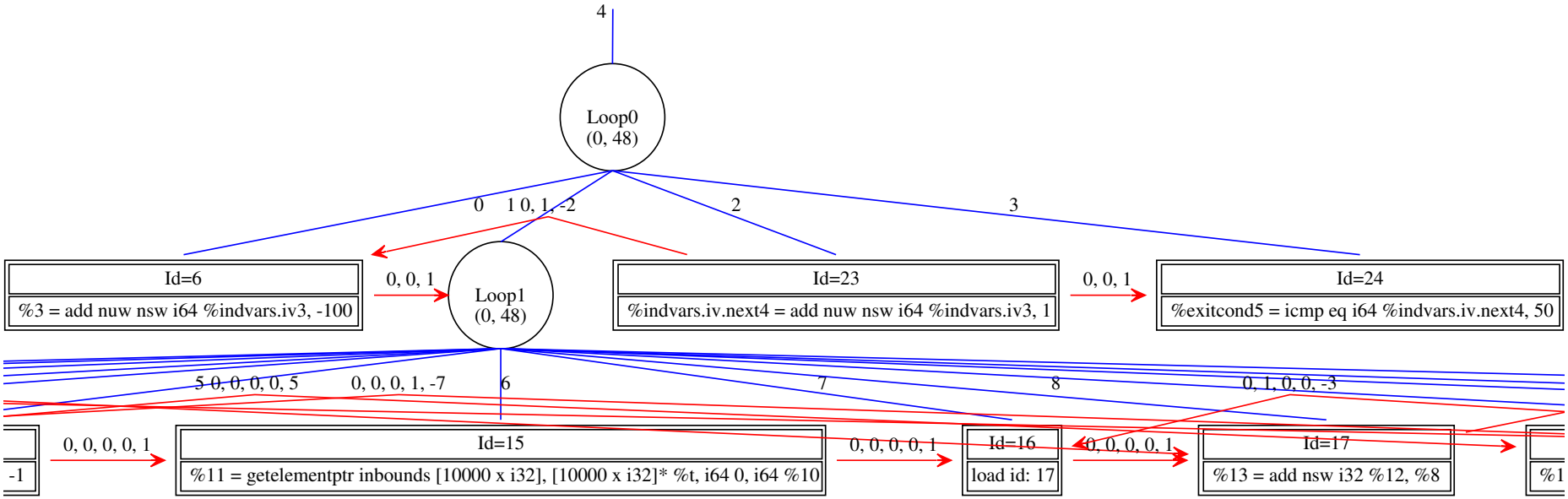
Dynamic dependency graph



- Nodes associated to instructions
- Nodes associated with executing loop indexes
- Easy mapping to polytope model
- Huge number of nodes

Profiling Guided Hybrid Compilation

Dynamic syntax tree



- Nodes into instructions
- Compressed edges into distance vectors
- Easy to validate dependencies
- No existing tools for manipulation

Profiling Guided Hybrid Compilation

Outline

Profiling Guided Hybrid Compilation

Lightweigh run-time tests

```
for(j=1; j<m; j++)
    for(i=1; i<n; i++)
       t[H*i+j] = t[H*i-H+j)] + t[-1+H*i+j]
```

W_3 R_1 R_2

How to test create a test if the transformation is valid?

Profiling Guided Hybrid Compilation

Lightweigh run-time tests

- We must preserve dependent iterations execution order
- The transformation is valid if we test:

$$D_{all} = D_{R_1 \to W_3} \cup D_{W_3 \to R_1} \cup D_{W_3 \to R_2} \cup D_{R_2 \to W_3} \cup D_{W_3 \to W_3} = \emptyset$$

Where

$$D_{R_1 \to W_3} = (i,j) \prec (i',j') \;\&\; W_3(i,j) = R_1(i',j') \& T \times (i',j') \prec T \times (i,j)$$
$$D_{W_3 \to R_1} = (i,j) \prec (i',j') \;\&\; R_1(i,j) = W_3(i',j') \& T \times (i',j') \prec T \times (i,j)$$
$$D_{R_2 \to W_3} = (i,j) \prec (i',j') \;\&\; W_3(i,j) = R_2(i',j') \& T \times (i',j') \prec T \times (i,j)$$
$$D_{W_3 \to R_2} = (i,j) \prec (i',j') \;\&\; R_2(i,j) = W_3(i',j') \& T \times (i',j') \prec T \times (i,j)$$
$$D_{W_3 \to W_3} = (i,j) \prec (i',j') \;\&\; W_3(i,j) = W_3(i',j') \& T \times (i',j') \prec T \times (i,j)$$

Affine systems

- Suppose that always H = 100
- Simple case; affine system

http://www2.cs.kuleuven.be/cgi-bin/dtai/barvinok.cgi

```
Dom:={[n,m,i,j,i',j']:1<=i,i'<n and 1<=j,j'<=m-1 };
E31:={[n,m,i,j,i',j']:100i +j = - 100 + 100i' + j' };
E13:={[n,m,i,j,i',j']:100i' + j' = - 100 + 100i + j };
E32:={[n,m,i,j,i',j']:100i + j = -1 + 100i' + j' };
E23:={[n,m,i,j,i',j']:100i' + j' = -1 + 100i + j };
E33:={[n,m,i,j,i',j']:100i + j = 100i' + j' };
Dlex := { [n,m,i,j,i',j']: j<j' and i'<i };
D13 := E13*Dlex*Dom;
D31 := E31*Dlex*Dom;
D23 := E23*Dlex*Dom;
D32 := E32*Dlex*Dom;
D33 := E33*Dlex*Dom;
Dall := D13 + D31 + D23 + D33 + D32;
Dproj := { [n,m,i,j,i',j'] -> [n,m] }(Dall);
Dcond := {[n,m]: } - Dproj;
print coalesce(Dcond);

{ [n, m] : m <= 100 or (n <= 2 and m >= 101) }
```

Profiling Guided Hybrid Compilation

Non-affine systems

Loop-interchange is valid if and only if

$$D_{R_1 \to W_3} = \emptyset \ \wedge \ D_{W_3 \to R_1} = \emptyset \ \wedge \ D_{W_3 \to R_2} = \emptyset \ \wedge \ D_{R_2 \to W_3} = \emptyset \ \wedge \ D_{W_3 \to W_3} = \emptyset$$

$$\begin{aligned}
D_{R_1 \to W_3} = \{ \quad & [H, n, m, i, j, i', j'] : \\
\wedge \quad & 1 \leq i < n && (1) \\
\wedge \quad & 1 \leq i' < n && (2) \\
\wedge \quad & 1 \leq j < m && (3) \\
\wedge \quad & 1 \leq j' < m && (4) \\
\wedge \quad & (j, i) \prec_{lex} (j', i') && (5) \\
\wedge \quad & (i, j) \not\prec_{lex} (i', j') && (6) \\
\wedge \quad & -H + Hi + j = Hi' + j' \} && (7)
\end{aligned}$$

Profiling Guided Hybrid Compilation

Non-affine systems

Loop-interchange is valid if and only if

$$D_{R_1 \to W_3} = \emptyset \ \wedge \ D_{W_3 \to R_1} = \emptyset \ \wedge \ D_{W_3 \to R_2} = \emptyset \ \wedge \ D_{R_2 \to W_3} = \emptyset \ \wedge \ D_{W_3 \to W_3} = \emptyset$$

$$\begin{aligned}
D_{R_1 \to W_3} = \{ \quad & [H, n, m, i, j, i', j'] : \\
\wedge \quad & 1 \leq i \leq n-1 && (1) \\
\wedge \quad & 1 \leq i' \leq n-1 && (2) \\
\wedge \quad & 1 \leq j \leq m-1 && (3) \\
\wedge \quad & 1 \leq j' \leq m-1 && (4) \\
\wedge \quad & j \leq j'-1 && (5) \\
\wedge \quad & i' \leq i-1 && (6) \\
\wedge \quad & -H + Hi + j = Hi' + j' \} && (7)
\end{aligned}$$

Non-affine systems - Hand solving

Solving dependence $D_{W_3 \to R_1} : H \cdot i + j + H - H \cdot i' - j' = 0$

Start equations:

$$
\begin{array}{lrcccl}
0) & 0 & \leq & H & & \\
1) & 1 & \leq & i & \leq & n-1 \\
2) & 1 & \leq & i' & \leq & n-1 \\
3) & 1 & \leq & j & \leq & m-1 \\
4) & 1 & \leq & j' & \leq & m-1 \\
5) & j - j' & \leq & -1 & & \\
6) & i' - i & \leq & -1 & & \\
7) & H \cdot i + j + H - H \cdot i' - j' = 0 & & & &
\end{array}
$$

Profiling Guided Hybrid Compilation

Non-affine systems - Hand solving

Solving dependence $D_{W_3 \to R_1} : H \cdot i + j + H - H \cdot i' - j' = 0$

Writting expressions in function of $i - i'$ and $j' - j$ (for simplicity: result is unchanged)

$$
\begin{array}{l|rcccl}
0) & 0 & \leq & H & & \\
1) & -(n-2) & \leq & i - i' & \leq & n-2 \\
2) & -(m-2) & \leq & j' - j & \leq & m-2 \\
3) & 1 & \leq & j' - j & & \\
4) & 1 & \leq & i - i' & & \\
5) & & H(i - i') + H & = & j' - j &
\end{array}
$$

Profiling Guided Hybrid Compilation

Non-affine systems - Hand solving

Solving dependence $D_{W_3 \to R_1} : H \cdot i + j + H - H \cdot i' - j' = 0$

Eliminating $j' - j$

0)		0	$\leq$	H		
1)		$-(n-2)$	$\leq$	$i - i'$	$\leq$	$n-2$
5&2)		$-(m-2)$	$\leq$	$H(i-i') + H$	$\leq$	$m-2$
5&3)		1	$\leq$	$H(i-i') + H$		
4)		1	$\leq$	$i - i'$		

Profiling Guided Hybrid Compilation

Non-affine systems - Hand solving

Solving dependence $D_{W_3 \to R_1} : H \cdot i + j + H - H \cdot i' - j' = 0$

Prepare for eliminating $H(i - i')$ (suppose $H \neq 0$. Note that $H = 0 \Rightarrow$ false)

0)	1	$\leq$	H		
1)	$-H(n-2)$	$\leq$	$H(i-i')$	$\leq$	$H(n-2)$
2)	$-(m-2)-H$	$\leq$	$H(i-i')$	$\leq$	$m-2-H$
3)	$1-H$	$\leq$	$H(i-i')$		
4)	H	$\leq$	$H(i-i')$		

Non-affine systems - Hand solving

Solving dependence $D_{W_3 \to R_1} : H \cdot i + j + H - H \cdot i' - j' = 0$

Eliminating $H(i - i')$

0)	1	$\leq$	H
1&1)	$-H(n-2)$	$\leq$	$H(n-2)$
1&2)	$-H(n-2)$	$\leq$	$m-2-H$
2&1)	$-(m-2)-H$	$\leq$	$H(n-2)$
2&2)	$-(m-2)-H$	$\leq$	$m-2-H$
3&1)	$1-H$	$\leq$	$H(n-2)$
3&2)	$1-H$	$\leq$	$m-2-H$
4&1)	H	$\leq$	$H(n-2)$
4&2)	H	$\leq$	$m-2-H$

Profiling Guided Hybrid Compilation

Non-affine systems - Hand solving

Solving dependence $D_{W_3 \to R_1} : H \cdot i + j + H - H \cdot i' - j' = 0$

0)	1	$\leq$	H	
1)	2	$\leq$	n	implied by (7)
2)	$-H(n-3)$	$\leq$	$m-2$	implied by $(0) \wedge (7) \wedge (8)$
3)	$-(m-2)$	$\leq$	$H(n-1)$	implied by $(0) \wedge (7) \wedge (8)$
4)	2	$\leq$	m	implied by $(0) \wedge (8)$
5)	1	$\leq$	$H(n-1)$	implied by $(0) \wedge (7)$
6)	3	$\leq$	m	implied by $(0) \wedge (8)$
7)	3	$\leq$	n	
8)	$2H+2$	$\leq$	m	

That simplifies into $3 \leq n \wedge 2H + 2 \leq m$ (forget condition on $H \neq 0$)

Leading to a condition for validity: $n \leq 2 \vee m \leq 2H + 1$

Profiling Guided Hybrid Compilation

Non-affine systems - Hand solving

Do it for all pairs of memory accesses

$$
\begin{array}{lcll}
 & m \leq H+1 \vee n \leq 3 & (D_{R_1 \to W_3}) \\
\wedge & m \leq 2H+1 \vee n \leq 2 & (D_{W_3 \to R_1}) \\
\wedge & m \leq H \vee n \leq 2 & (D_{R_2 \to W_3}) \\
\wedge & m \leq H+1 \vee n \leq 2 & (D_{W_3 \to W_3}) \\
\wedge & m \leq H+2 \vee n \leq 2 & (D_{W_3 \to W_3})
\end{array}
$$

That should simplify into: $m \leq H \vee n \leq 2$

Profiling Guided Hybrid Compilation

Non-affine systems - Automated solving Fourier–Motzkin elimination → huge growth of number of constraints

- Schwighofer algorithm: Remove redundant constraints and prune search space
- Factorization: Can benefit from factorized form to reduce search space

For the example test obtained: $m \leq H + 2 \vee n \leq 2$
Manual case: $m \leq H \vee n \leq 2$

Profiling Guided Hybrid Compilation

Under progress

- Use trace information to guide run-time test builder
- Memory model to calculate locality
- Identify profitable loop transformations using trace file
- Generate loop transformation using existing tools (polly, pocc ...)