GRAPHITE: Polyhedral Analyses and Optimizations for GCC

Sebastian Pop ¹, Albert Cohen ², Cédric Bastoul ², Sylvain Girbal ², Georges-André Silber ¹, Nicolas Vasilache ²

¹CRI/ENSMP
²Alchemy/INRIA, LRI/Paris Sud 11 University

June, 2006
Architecture of GCC and Loop Nest Optimizer

- C
- C++
- Java
- F95
- Ada

**GENERIC**

**GIMPLE**

Analyses
- aliasing
- data dependences
- number of iterations

**GIMPLE + CFG + SSA + Loops**

**LNO**

**Machine description**

**RTL**

- x86
- ppc
- arm
- ...
Problems with Classical LNO Transforms

- “source to source” modifies the compiled program
- difficult to undo
- order of transforms fixed once for all
- invalidated data deps: ad-hoc correction or rebuild
- difficult to compose
Problems with Classical LNO Transforms

- “source to source” modifies the compiled program
- difficult to undo
- order of transforms fixed once for all
- invalidated data deps: ad-hoc correction or rebuild
- difficult to compose

solved in **WRaP-IT** (from 2002 at INRIA on ORC/Open64)

**GRAPHITE** = **WRaP-IT** for GCC
Statements + parametric affine inequalities

1. a **domain** = bounds of enclosing loops
2. a list of access functions
3. a **schedule** = execution time

```plaintext
for (i=0; i<m; i++)
  for (j=5; j<n; j++)
    A[2*i][j+1] = ...
```

```
\[
\begin{bmatrix}
  i & j & m & n & cst \\
  1 & 0 & 0 & 0 & 0 \\
 -1 & 0 & 1 & 0 & -1 \\
  0 & 1 & 0 & 0 & 5 \\
  0 & -1 & 0 & 1 & -1 \\
\end{bmatrix}
\]

\begin{align*}
i & \geq 0 \\
-i + m & \geq -1 \\
j & \geq 5 \\
-j + n & \geq -1
\end{align*}
```
Statements + parametric affine inequalities

1. a **domain** = bounds of enclosing loops
2. a list of **access functions**
3. a **schedule** = execution time

for (i=0; i<m; i++)
   for (j=5; j<n; j++)
      $A[2*i][j+1] = \ldots$

\[
\begin{bmatrix}
  i & j & m & n & cst \\
  2 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
2 \times i \quad j + 1
\]
Statements + parametric affine inequalities

1. a domain = bounds of enclosing loops
2. a list of access functions
3. a schedule = execution time

GRAPHITE (1, 2, 3) extends LAMBDA (1, 2)

GRAPHITE: Gimple Represented As Polyhedra
(with interchangeable envelopes)
GRAPHITE versus LAMBDA

- common part: unimodular transform data and iteration order
- transform regions: extended from loops to SCoP
  - “static control parts”: sequences, affine conditions and loops
- GRAPHITE knows about the sequence!
  - enables more loop transforms: fusion, fission, tiling, software pipelining, scheduling
GRAPHITE versus LAMBDA

- **common part:** unimodular transform data and iteration order
- **transform regions:** extended from loops to SCoP
  - "static control parts": sequences, affine conditions and loops
- **GRAPHITE knows about the sequence!**
  - enables more loop transforms: fusion, fission, tiling, software pipelining, scheduling
common part: unimodular transform data and iteration order

transform regions: extended from loops to SCoP
  “static control parts”: sequences, affine conditions and loops

GRAPHITE knows about the sequence!

enables more loop transforms: fusion, fission, tiling, software pipelining, scheduling
build a scheduling function $S[stmt] \to time$

- sequence $[s_1; s_2]$: trivial
  
  
  $S[s_1] = t$
  $S[s_2] = t + 1$

- loop $[loop_1 s end_1]$: add new dimensions
  
  
  $S[loop_1] = t$
  $S[s] = (t, i_1, 0)$

  $i_1$ indexes $loop_1$ iterations: dynamic time
build a scheduling function $S[stmt] \rightarrow time$

- sequence $[s_1; s_2]$: trivial
  
  $S[s_1] = t$
  $S[s_2] = t + 1$

- loop $[loop_1 s end_1]$: add new dimensions
  
  $S[loop_1] = t$
  $S[s] = (t, i_1, 0)$

  $i_1$ indexes $loop_1$ iterations: dynamic time
S0;
S1;
for (i=0; i<m; i++) {
    S2;
    for (j=5; j<n; j++)
        S3;
}
S4;

\[ S[S0] = \begin{bmatrix}
    i & j & m & n & cst \\
    0 & 0 & 0 & 0 & 0
\end{bmatrix} \]
S0;
S1;
for (i=0; i<m; i++) {
    S2;
    for (j=5; j<n; j++)
        S3;
}
S4;

\[
S[S0] = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
S[S1] = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]
S0;
S1;
for (i=0; i<m; i++) {
    S2;
    for (j=5; j<n; j++)
        S3;
}
S4;

\[
S[S0] = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
S[S1] = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
S[S2] = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 2 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
Schedule: Example

\[ S[S0] = \begin{bmatrix}
  i & j & m & n & cst \\
  0 & 0 & 0 & 0 & 0
\end{bmatrix} \]

\[ S[S1] = \begin{bmatrix}
  i & j & m & n & cst \\
  0 & 0 & 0 & 0 & 1
\end{bmatrix} \]

\[ S[S2] = \begin{bmatrix}
  i & j & m & n & cst \\
  0 & 0 & 0 & 0 & 2 \\
  1 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & 0
\end{bmatrix} \]

\[ S[S3] = \begin{bmatrix}
  i & j & m & n & cst \\
  0 & 0 & 0 & 0 & 2 \\
  1 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & 1 \\
  0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & 0
\end{bmatrix} \]
Schedule: Example

S0;
S1;
for (i=0; i<m; i++) {
    S2;
    for (j=5; j<n; j++)
        S3;
}
S4;

\[
S_{[S0]} = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
S_{[S1]} = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
S_{[S2]} = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 2 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
S_{[S3]} = \begin{bmatrix}
i & j & m & n & cst \\
0 & 0 & 0 & 0 & 2 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
Schedule: Separation Example

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

scheduling matrix $S[S3]$
### Schedule: Separation Example

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

separate static / dynamic schedules
Schedule: Separation Example

static scheduling vector

- fusion, fission, code motion
### Schedule: Separation Example

#### Parameter Scheduling Matrix

- **Shifting**

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Graph showing the parameter scheduling matrix with arrows indicating the shifting process.
### Schedule: Separation Example

#### Iteration scheduling matrix

- interchange, skewing, reversal

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>cst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Small set of primitives (basic operations on matrices)

1. motion
2. interchange
3. strip-mine
4. insert, delete
5. shift
6. skew, reversal, reindexing
7. privatize

- fission/fusion (1)
- tiling (2 + 3)
Find sequences of transforms based on
- size of loops
- cache misses
- simulation

Automatic selection of transforms
- amounts to choosing a point in a vector space
- hard part (open questions)
- WRaP-IT uses directives
- some transforms yield cool speedups . . .
Results From \textbf{WRaP-IT} on Top of PathScale EKOPath

swim from SPEC CPU2000

- 32\% speedup on AthlonXP \textit{wrt.} peak EKOPath (V2.1)
- 38\% speedup for Athlon64 \textit{wrt.} peak EKOPath (V2.1)
- principal SCoP: 421 lines of code
- apply 30 transforms to principal SCoP
  - fusion, tiling, peeling, unrolling, interchange, strip-mining
- result 2267 LOC
- 39 sec source to assembly on AthlonXP 2.08GHz
- 22 sec in the backend
- 12 sec polyhedral data deps
- 4 sec polyhedral code gen
GRAPHITE: Road Map

1. select SCoPs filter out difficult codes (Alexandru Plesco)
2. extend LAMBDA build schedule functions, GLooG
3. cost models more static analyzers, and transform selection
4. array regions improve data deps in interproc mode
5. lib integration PolyLib, PiPLib, Omega, lib-APRON
lib-APRON: interchange envelopes

limit computation complexity = restrict expressivity
use coarser representations

Polyhedra (n constraints)  Octagons (8 constraints)  Boxes (4 constraints)
proposed libs:

- PolyLib, PiPLib, Omega, Octagon, lib-APRON
- public domain, or GPL,
- about 20 kLOC
- in GCC, or GCC depend on?
Questions?