GRAPHITE: Polyhedral Analyses and Optimizations for GCC

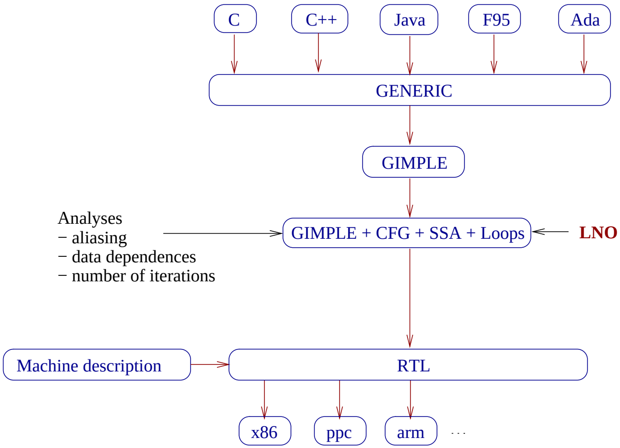
Sebastian Pop [1], Albert Cohen [2], Cédric Bastoul [2], Sylvain Girbal [2], Georges-André Silber [1], Nicolas Vasilache [2]

[1]CRI/ENSMP

[2]Alchemy/INRIA, LRI/Paris Sud 11 University

June, 2006

Architecture of GCC and Loop Nest Optimizer



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

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

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

$i \geq 0$
$-i + m \geq -1$
$j \geq 5$
$-j + n \geq -1$

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] = ...
```

$$\left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right] \quad \begin{array}{c} 2*i \\ j+1 \end{array}$$

GRAPHITE : Representation on Top of Gimple-SSA

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

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

Schedule: Operational Semantics (How Program Works)

build a scheduling function $\mathcal{S}[\![stmt]\!] \rightarrow time$

- sequence $[\![s_1; s_2]\!]$: trivial

$$\begin{aligned} \mathcal{S}[\![s_1]\!] &= t \\ \mathcal{S}[\![s_2]\!] &= t+1 \end{aligned}$$

- loop $[\![loop_1 \;\; s \;\; end_1]\!]$: add new dimensions

$$\begin{aligned} \mathcal{S}[\![loop_1]\!] &= t \\ \mathcal{S}[\![s]\!] &= (t, i_1, 0) \end{aligned}$$

i_1 indexes $loop_1$ iterations: dynamic time

Schedule: Operational Semantics (How Program Works)

build a scheduling function $\mathcal{S}[\![stmt]\!] \rightarrow time$

- sequence $[\![s_1; s_2]\!]$: trivial

$$\mathcal{S}[\![s_1]\!] = t$$
$$\mathcal{S}[\![s_2]\!] = t+1$$

- loop $[\![loop_1 \;\; s \;\; end_1]\!]$: add new dimensions

$$\mathcal{S}[\![loop_1]\!] = t$$
$$\mathcal{S}[\![s]\!] = (t, i_1, 0)$$

i_1 indexes $loop_1$ iterations: dynamic time

Schedule: Example

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

$$\mathcal{S}[\![S0]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Schedule: Example

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

$$\mathcal{S}[\![S0]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathcal{S}[\![S1]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Schedule: Example

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

$$\mathcal{S}[\![S0]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathcal{S}[\![S1]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\mathcal{S}[\![S2]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Schedule: Example

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

$$\mathcal{S}[\![S0]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathcal{S}[\![S1]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\mathcal{S}[\![S2]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathcal{S}[\![S3]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Schedule: Example

$$\mathcal{S}[\![S0]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

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

$$\mathcal{S}[\![S1]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\mathcal{S}[\![S2]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathcal{S}[\![S4]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 3 \end{array}\right]$$

$$\mathcal{S}[\![S3]\!] = \left[\begin{array}{ccccc} i & j & m & n & cst \\ \hline 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Schedule: Separation Example

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

scheduling matrix $\mathcal{S}[\![S3]\!]$

Schedule: Separation Example

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

i	j	m	n	cst
0	0	0	0	2
0	0	0	0	1
0	0	0	0	0

i	j	m	n	cst
1	0	0	0	0
0	1	0	0	0

separate static / dynamic schedules

Schedule: Separation Example

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

i	j	m	n	cst
0	0	0	0	2
0	0	0	0	1
0	0	0	0	0

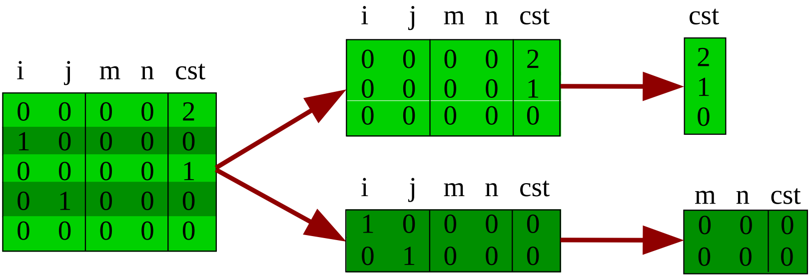
cst
2
1
0

i	j	m	n	cst
1	0	0	0	0
0	1	0	0	0

static scheduling vector

- fusion, fission, code motion

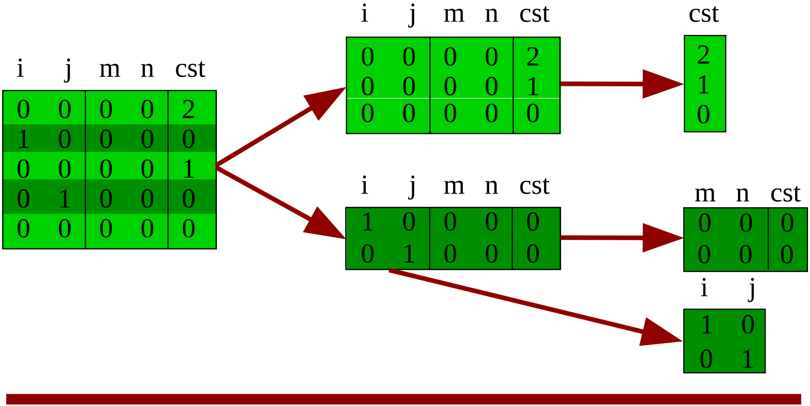
Schedule: Separation Example



Parameter scheduling matrix

- shifting

Schedule: Separation Example



Iteration scheduling matrix

- interchange, skewing, reversal

Compose Transforms

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 $(2 + 3)$

Optimal Transform?

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 ...

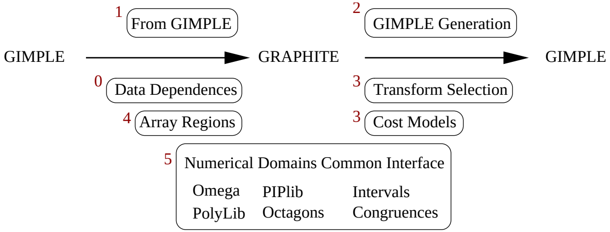
Results From WRaP-IT on Top of PathScale EKOPath

swim from SPEC CPU2000

- 32% speedup on AthlonXP wrt. peak EKOPath (V2.1)
- 38% speedup for Athlon64 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

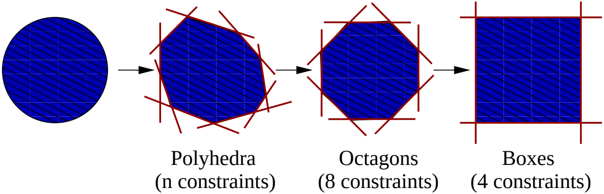
1. select SCoPs filter out difficult codes (Alexandru Plesco)
2. extend LAMBDA build schedule functions, GLooG
3. cost models more static analyzers, and transform selection
4. array regions improve data deps in interproc mode
5. lib integration PolyLib, PiPLib, Omega, lib-APRON



lib-APRON: interchange envelopes

limit computation complexity = restrict expressivity

use coarser representations



Library integrations

proposed libs:

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

Questions?