GRAPHITE Two Years After

First Lessons Learned From Real-World Polyhedral Compilation

Konrad Trifunovic[2] Albert Cohen[2] David Edelsohn[3] Li Feng[6]
Tobias Grosser[5] Harsha Jagasia[1] Razya Ladelsky[4] Sebastian
Pop[1] Jan Sjödin[1] Ramakrishna Upadrasta[2]

[1]Open Source Compiler Engineering, AMD, Austin, Texas, USA

[2]INRIA Saclay – Île-de-France and LRI, Paris-Sud 11 University, Orsay, France

[3]IBM T. J. Watson Research, Yorktown Heights, USA

[4]IBM Haifa Research, Haifa, Israel

[5]University of Passau, Passau, Germany

[6]Xi'an Jiaotong University, Xi'an, China

January 30, 2010

Keeping sustained performance increase

Keeping sustained performance increase

Multi-level parallelism

- (ILP) Instruction-Level-Parallelism (instruction scheduling)
- Data-level parallelism (vectorization)
- Thread-level parallelism (automatic parallelization)

Keeping sustained performance increase

Multi-level parallelism

- (ILP) Instruction-Level-Parallelism (instruction scheduling)
- Data-level parallelism (vectorization)
- Thread-level parallelism (automatic parallelization)

Memory hierarchy

- Caches
- Registers
- Scratchpad memories

Keeping sustained performance increase

Multi-level parallelism

- (ILP) Instruction-Level-Parallelism (instruction scheduling)
- Data-level parallelism (vectorization)
- Thread-level parallelism (automatic parallelization)

Memory hierarchy

- Caches
- Registers
- Scratchpad memories

Need for complex program (loop) optimizations

Why polyhedral model in GCC?

Why polyhedral model in GCC?

Source to source compilers

- Syntax based
- Output source code might lose semantical information
- Need for source code normalization

Why polyhedral model in GCC?

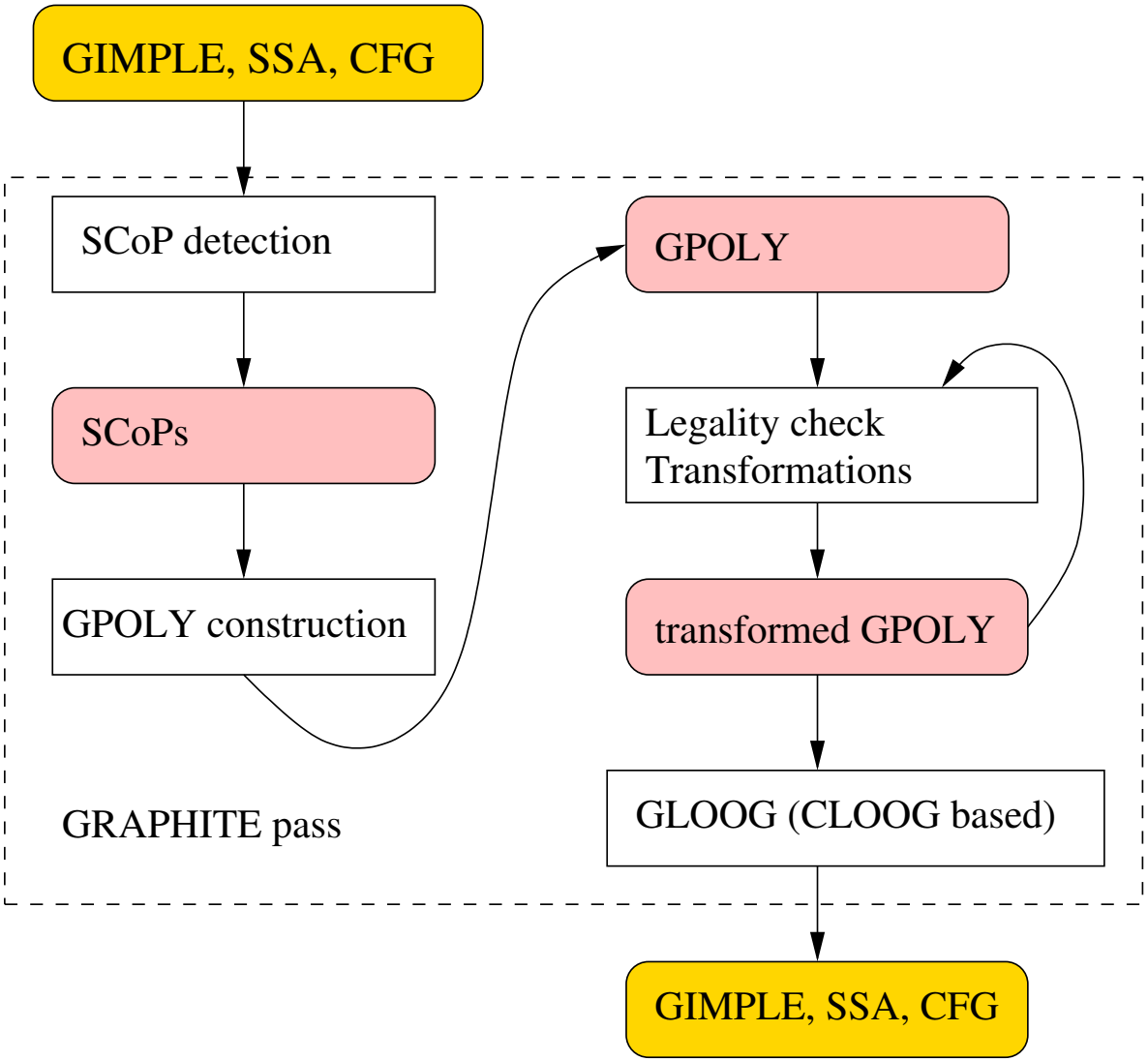
Source to source compilers

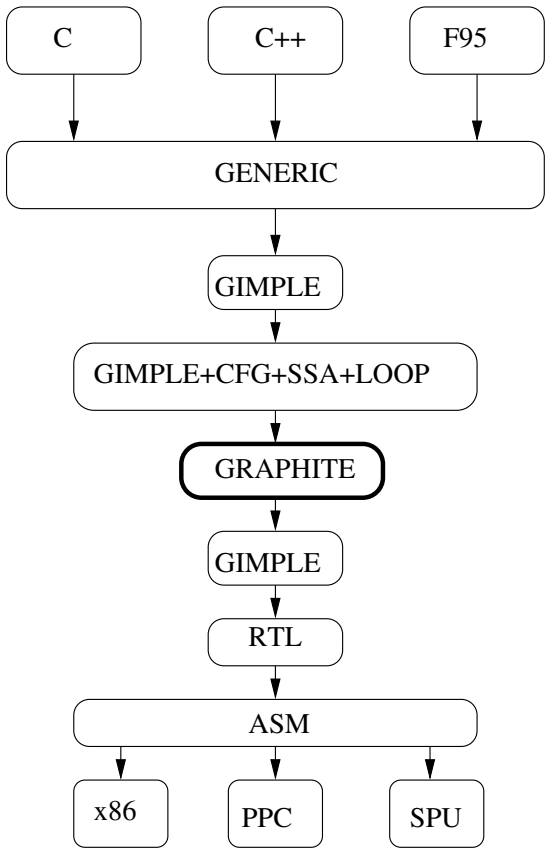
- Syntax based
- Output source code might lose semantical information
- Need for source code normalization

Low level internal polyhedral representation

- Semantics based
- SSA GIMPLE form
- Scalar evolution analysis (inductions, reductions)
- Leveraging $>$ 100 optimization passes of GCC
- Tight interaction with vectorizer, parallelizer and memory layout optimizations

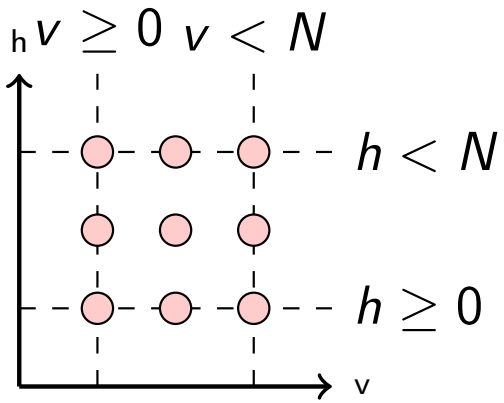
Compilation workflow





GPOLY - Iteration domains

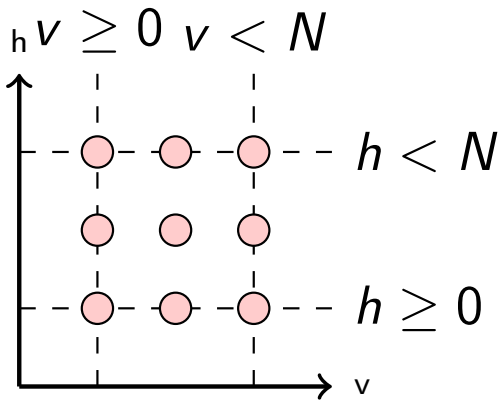
$$\mathcal{D}^S = \{(v, h) \mid 0 \leq v, h \leq N - 1\}$$




```
for (v=0; v<N; v++)
  for (h=0; h<N; h++)
    out[v][h] = 0;
```

GPOLY - Iteration domains

$$\mathcal{D}^S = \{(v,h) \mid 0 \leq v,h \leq N-1\}$$

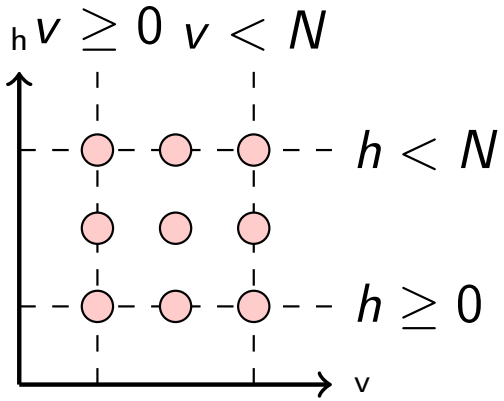


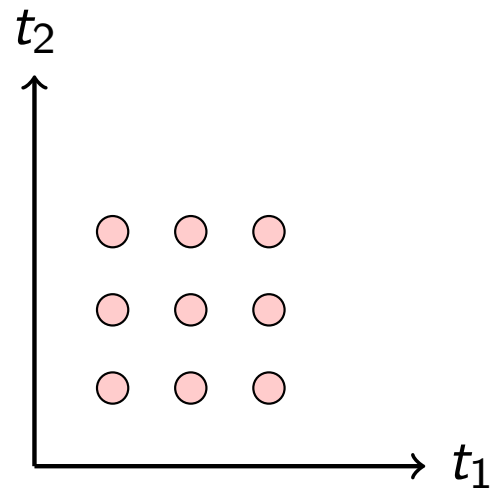

```
for (v=0; v<N; v++)
  for (h=0; h<N; h++)
    out[v][h] = 0;
```

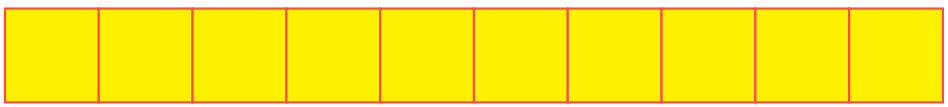
$$\left[\begin{array}{cc|c|c} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 \end{array}\right] \begin{pmatrix} v \\ h \\ N \\ 1 \end{pmatrix} \geq \mathbf{0} \quad \left| \begin{array}{l} v \geq 0 \\ v \leq N-1 \\ h \geq 0 \\ h \leq N-1 \end{array} \right.$$

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





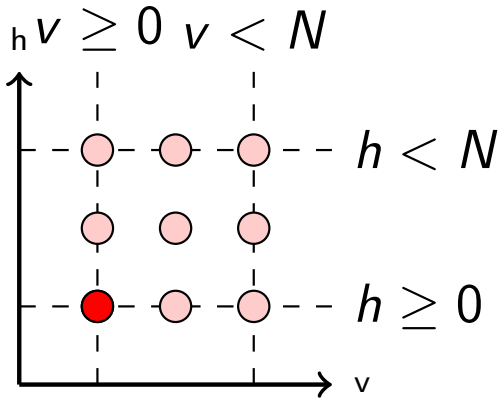


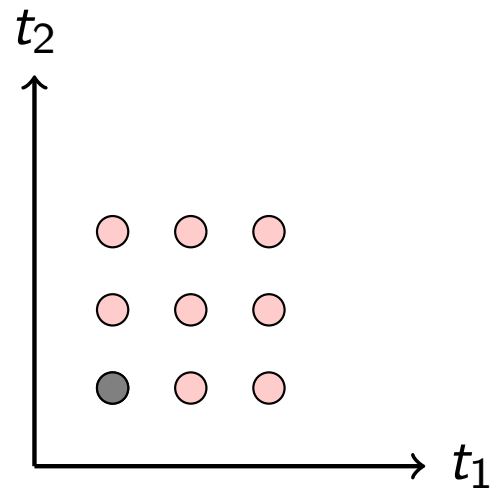
Linearized memory layout

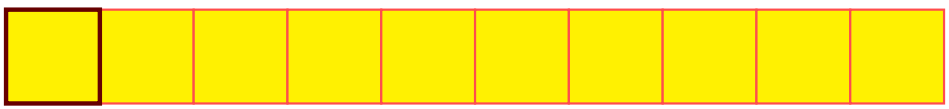
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





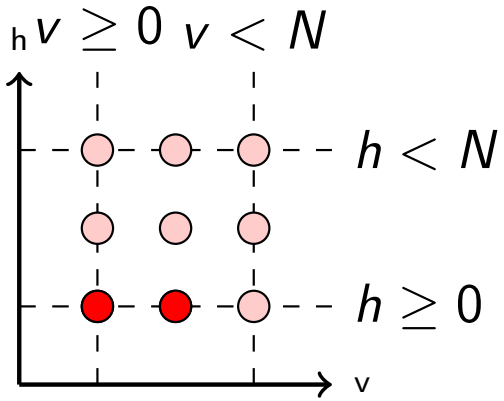


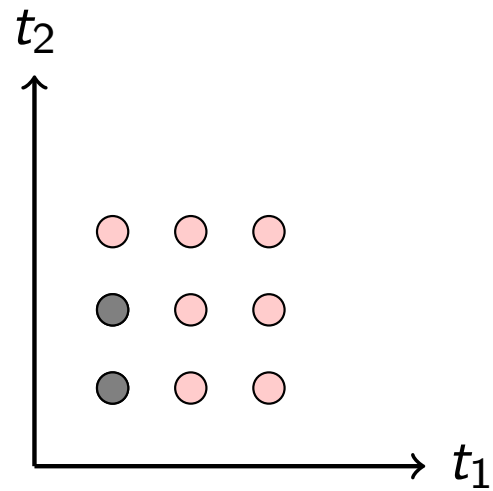
Linearized memory layout

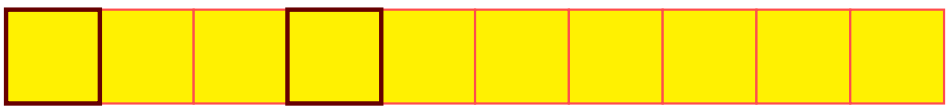
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





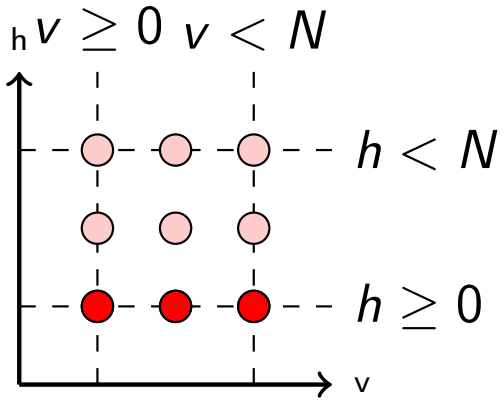


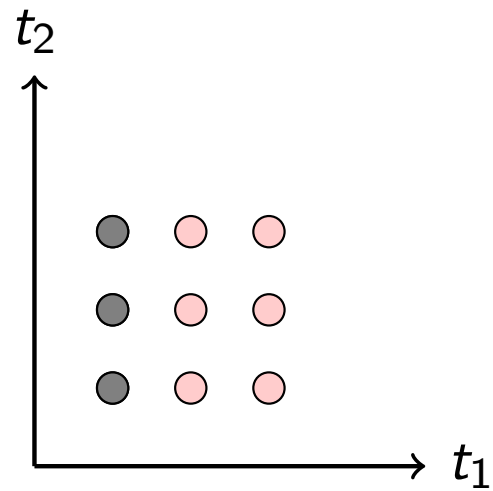
Linearized memory layout

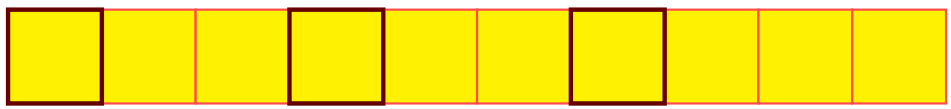
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





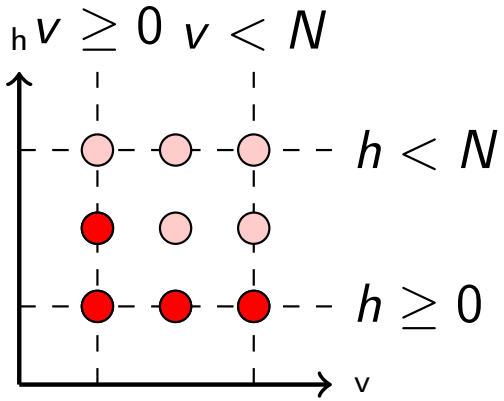


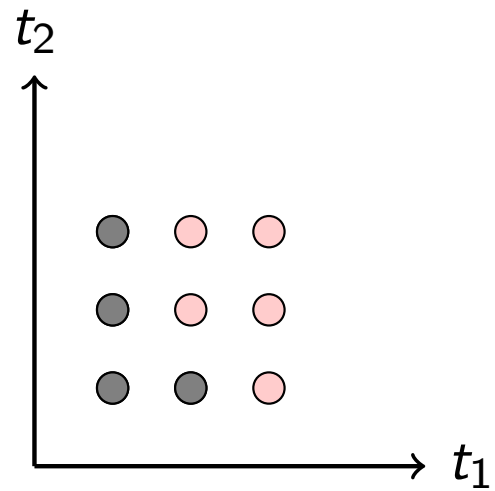
Linearized memory layout

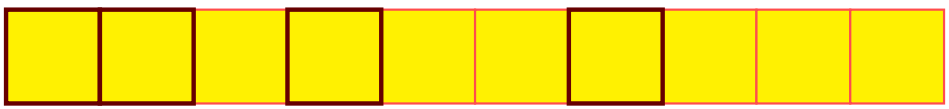
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





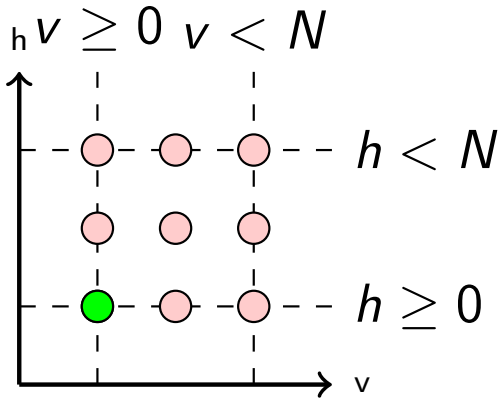


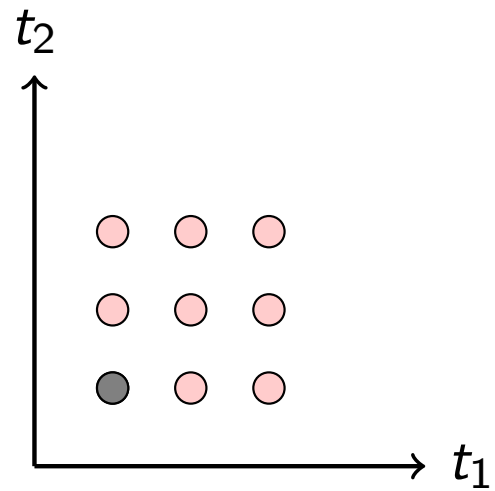
Linearized memory layout

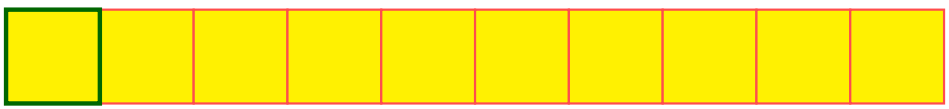
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





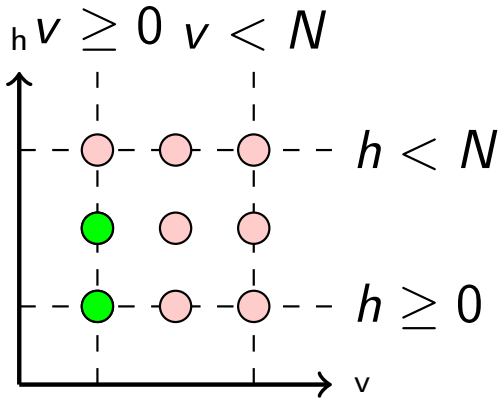


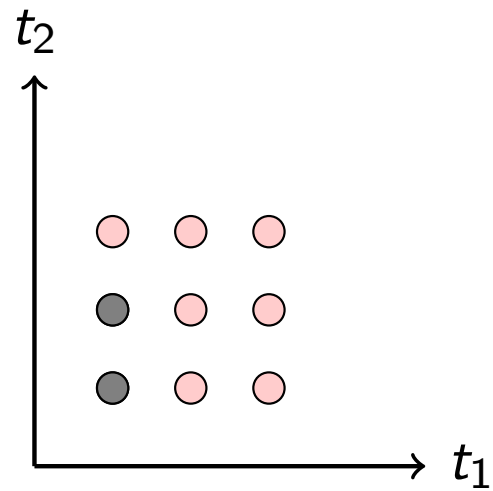
Linearized memory layout

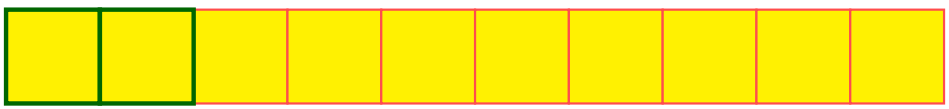
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





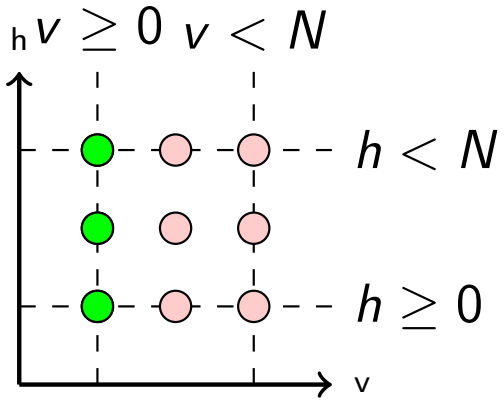


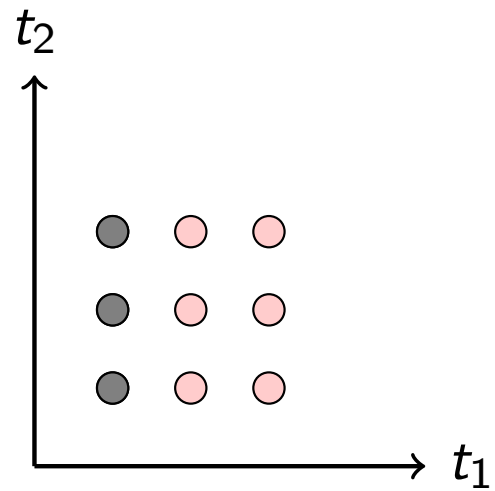
Linearized memory layout

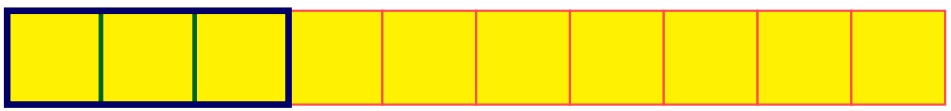
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





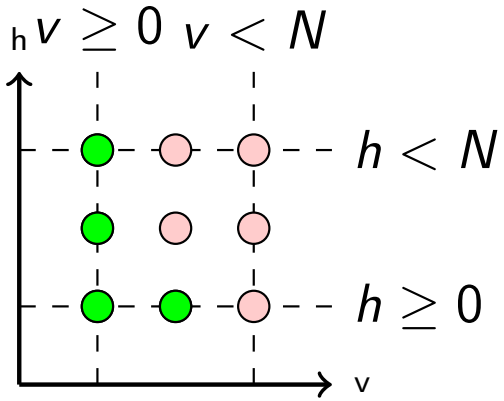


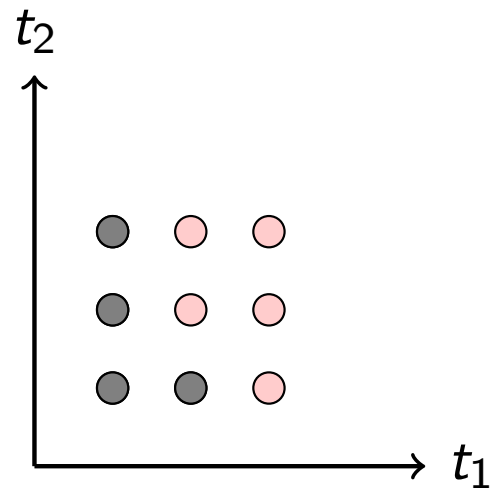
Linearized memory layout

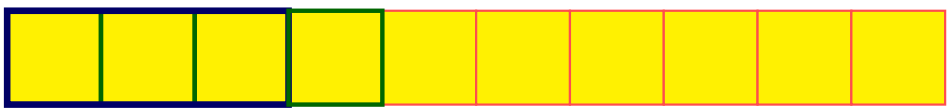
out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

Data accesses - mapping iterations to memory

$$f(\mathbf{i}, \mathbf{g}) = \mathcal{F} \times (\mathbf{i}, \mathbf{g}, 1)^T$$





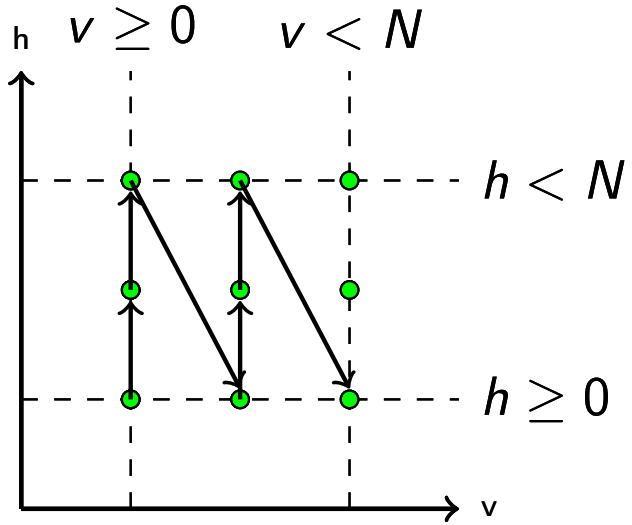


Linearized memory layout

out[1][1]	out[1][2]	out[1][3]
out[2][1]	out[2][2]	out[2][3]
out[3][1]	out[3][2]	out[3][3]

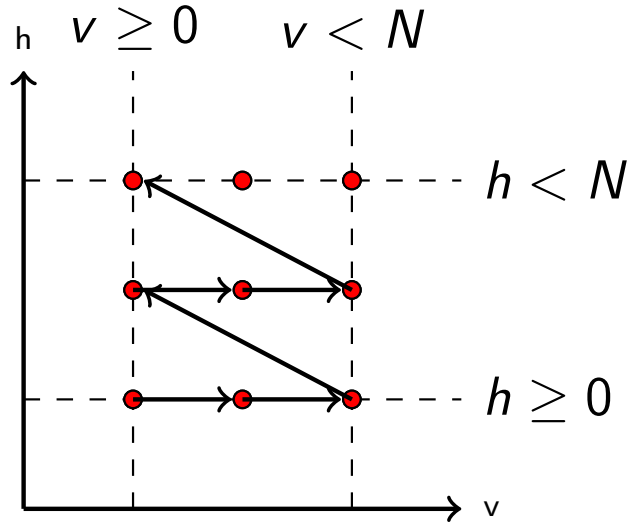
Scheduling - execution order

$$\mathbf{t} = \theta^S(\mathbf{i}) = \Theta^S \times (\mathbf{i}, \mathbf{g}, 1)^T$$



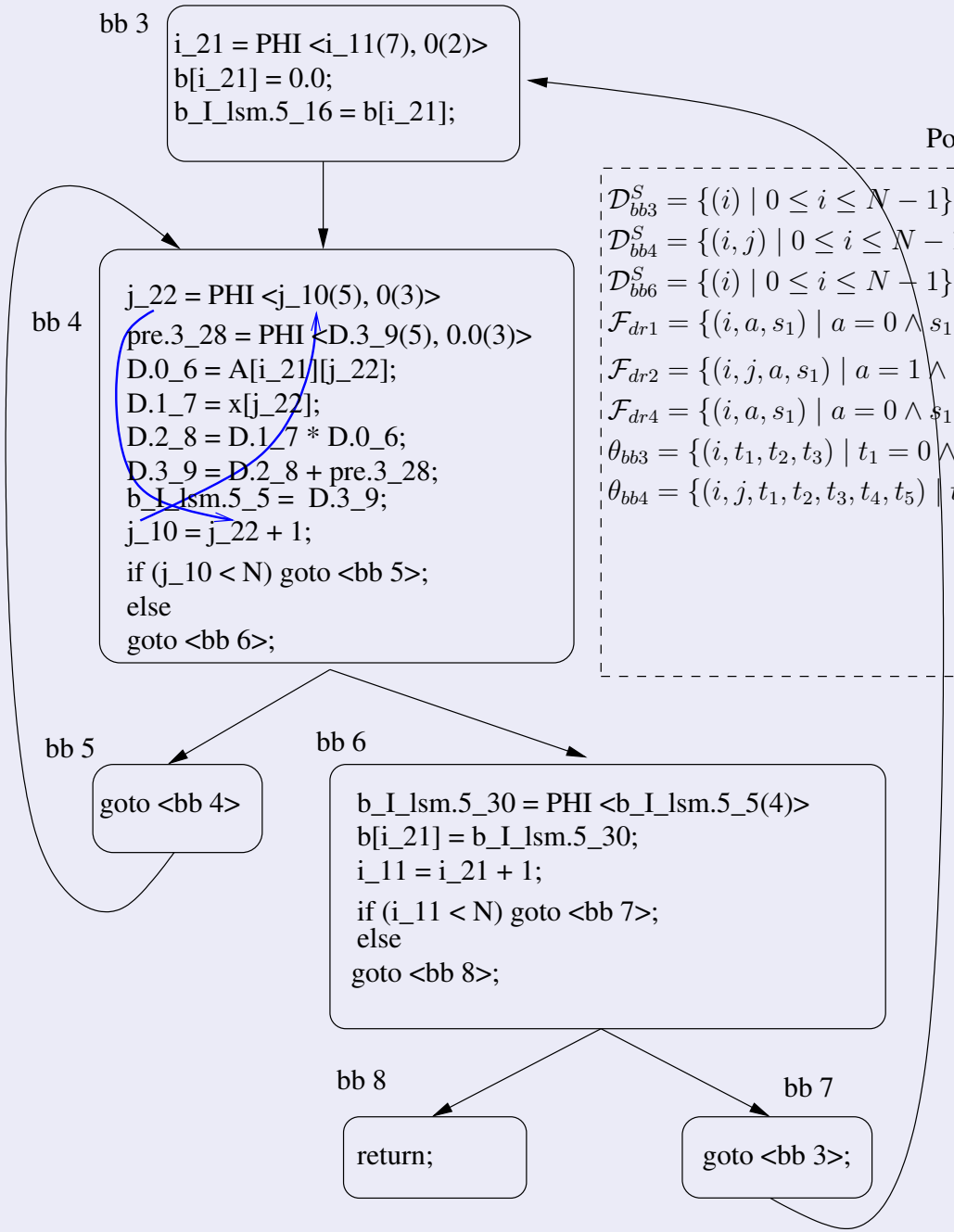

```
for (v=0; v<N; v++)
  for (h=0; h<N; h++)
    out[v][h] = 0;
```

$$\Theta^S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$




```
for (t1=0; t1<N; t1++)
  for (t2=0; t2<N; t2++)
    out[t2][t1] = 0;
```

$$\Theta'^S = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



Polyhedral representation

$\mathcal{D}^S_{bb3} = \{(i) \mid 0 \leq i \leq N-1\}$
$\mathcal{D}^S_{bb4} = \{(i,j) \mid 0 \leq i \leq N-1 \wedge 0 \leq j \leq N-1\}$
$\mathcal{D}^S_{bb6} = \{(i) \mid 0 \leq i \leq N-1\}$
$\mathcal{F}_{dr1} = \{(i,a,s_1) \mid a = 0 \wedge s_1 = i \wedge 0 \leq s_1 \leq N-1\}$
$\mathcal{F}_{dr2} = \{(i,j,a,s_1) \mid a = 1 \wedge s_1 = j \wedge 0 \leq s_1 \leq N-1\}$
$\mathcal{F}_{dr4} = \{(i,a,s_1) \mid a = 0 \wedge s_1 = i \wedge 0 \leq s_1 \leq N-1\}$
$\theta_{bb3} = \{(i,t_1,t_2,t_3) \mid t_1 = 0 \wedge t_2 = i \wedge t_3 = 0\}$
$\theta_{bb4} = \{(i,j,t_1,t_2,t_3,t_4,t_5) \mid t_1 = 0 \wedge t_2 = i \wedge t_3 = 1 \wedge t_4 = j \wedge t_5 = 0\}$

MVT kernel

```
for (i = 0; i < N; i++) {
  b[i] = 0;
  for (j = 0; j < N; j++)
    b[i] += A[i][j] * x[j];
}
```

Cost-modelling for vectorization

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++)
   for (j=0; j<K; j++)
    s+=img[v+i][h+j]
       * filter[i][j];
  out[v][h]= s;
 }
```

[Trifunovic et al. 2009]

Cost-modelling for vectorization

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++)
   for (j=0; j<K; j++)
    s+=img[v+i][h+j]
       * filter[i][j];
  out[v][h]= s;
 }
```

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++) {
   vs[0:3]={0,0,0,0};
   for (j=0; j<K; j+=4) {
    vs[0:3]+=img[v+i][h+j:h+j+3]
             *filter[i][j:j+3]
   }
   s+=sum(vs[0:3]);
  }
  out[v][h] = s;
 }
}
```

[Trifunovic et al. 2009]

Cost-modelling for vectorization

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++)
   for (j=0; j<K; j++)
    s+=img[v+i][h+j]
       * filter[i][j];
  out[v][h]= s;
 }
```

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++) {
   vs[0:3]={0,0,0,0};
   for (j=0; j<K; j+=4) {
    vs[0:3]+=img[v+i][h+j:h+j+3]
             *filter[i][j:j+3]
   }
   s+=sum(vs[0:3]);
  }
  out[v][h] = s;
 }
}
```

- Reduction costs: `sum` operation – vector `vs` is reduced into scalar `s`: $N^2 \cdot K$ number of times

[Trifunovic et al. 2009]

Cost-modelling for vectorization

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++)
   for (j=0; j<K; j++)
    s+=img[v+i][h+j]
       * filter[i][j];
  out[v][h]= s;
 }
```

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++) {
   vs[0:3]={0,0,0,0};
   for (j=0; j<K; j+=4) {
    vs[0:3]+=img[v+i][h+j:h+j+3]
             *filter[i][j:j+3]
   }
   s+=sum(vs[0:3]);
  }
  out[v][h] = s;
 }
}
```

- Reduction costs: `sum` operation – vector `vs` is reduced into scalar `s`: $N^2 \cdot K$ number of times
- Benefits: $VF = 4$ scalar ops are replaced by 1 vector operation

[Trifunovic et al. 2009]

Cost-modelling for vectorization

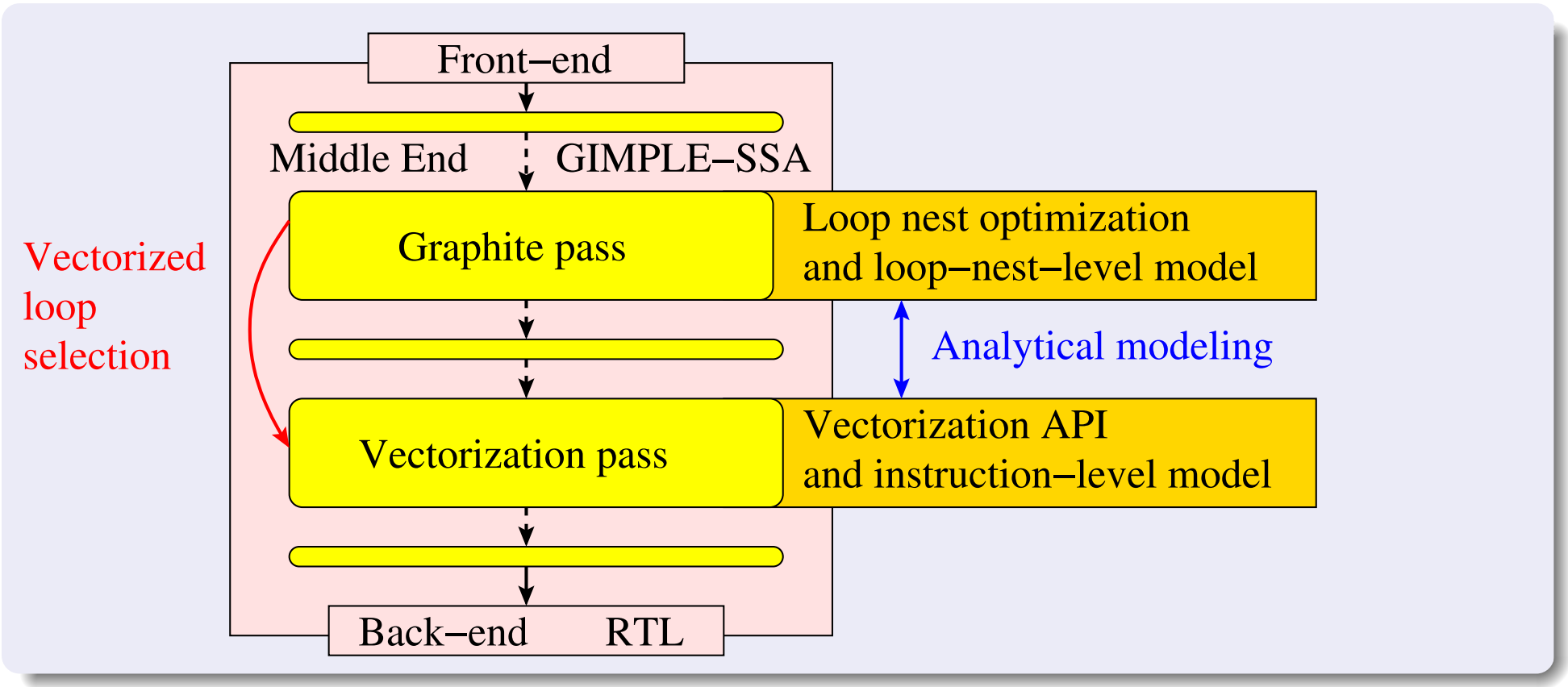
```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++)
   for (j=0; j<K; j++)
    s+=img[v+i][h+j]
       * filter[i][j];
  out[v][h]= s;
 }
```

```
for (v=0; v<N; v++)
 for (h=0; h<N; h++) {
  s=0;
  for (i=0; i<K; i++) {
   vs[0:3]={0,0,0,0};
   for (j=0; j<K; j+=4) {
    vs[0:3]+=img[v+i][h+j:h+j+3]
             *filter[i][j:j+3]
   }
   s+=sum(vs[0:3]);
  }
  out[v][h] = s;
 }
}
```

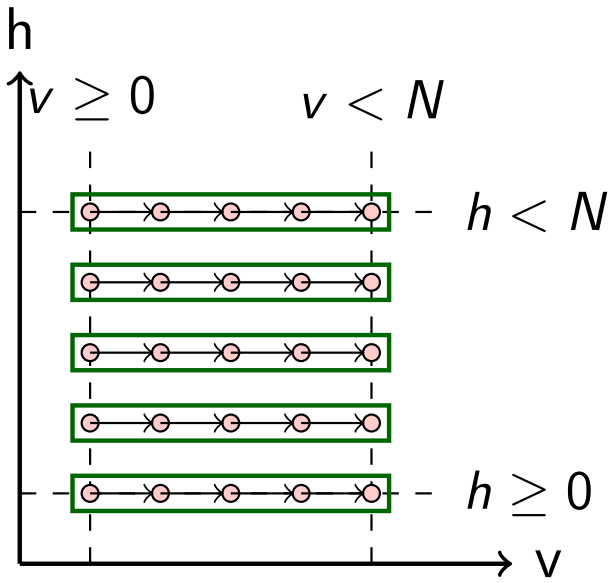
- Reduction costs: `sum` operation – vector `vs` is reduced into scalar `s`: $N^2 \cdot K$ number of times
- Benefits: $VF = 4$ scalar ops are replaced by 1 vector operation

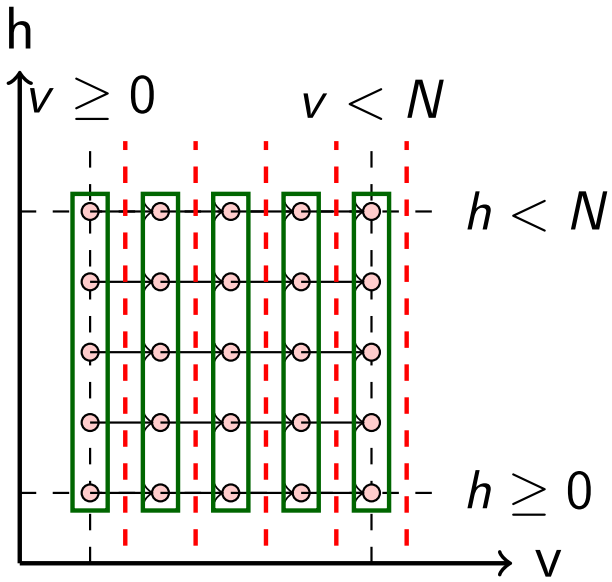
[Trifunovic et al. 2009]

Cost-modelling for vectorization



Autopar






```
parloop ()
{
  for (h = 0; h < N; h++)
    for (v = 1; v < N; v++)
      x[h][v] = x[h][v-1] + 1;
}
```

(a)

```
parloop ()
{
  .paral_data.x = &x;
  __builtin_GOMP_parallel_start (parloop._loopfn,
                                 &.paral_data, 4);
  parloop._loopfn (&.paral_data);

  __builtin_GOMP_parallel_end ();
}

parloop._loopfn (.paral_data)
{
  for (h = start; h < end; h++)
    for (v = 1; v < N; v++)
      (*.paral_data->x)[h][v] = x[h][v-1] + 1;
}
```

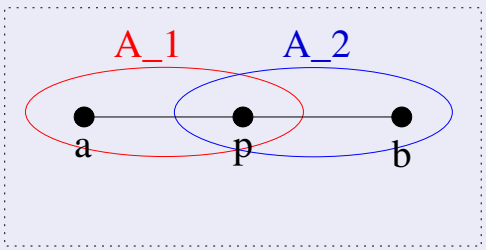
(b)

Encoding aliasing information

Representation of Alias-sets in GRAPHITE

- Dependence Analysis requires Alias information
- Alias sets encoded as an extra dimension of access functions

```
int a[10], b[10];
void foo (int *p);
```

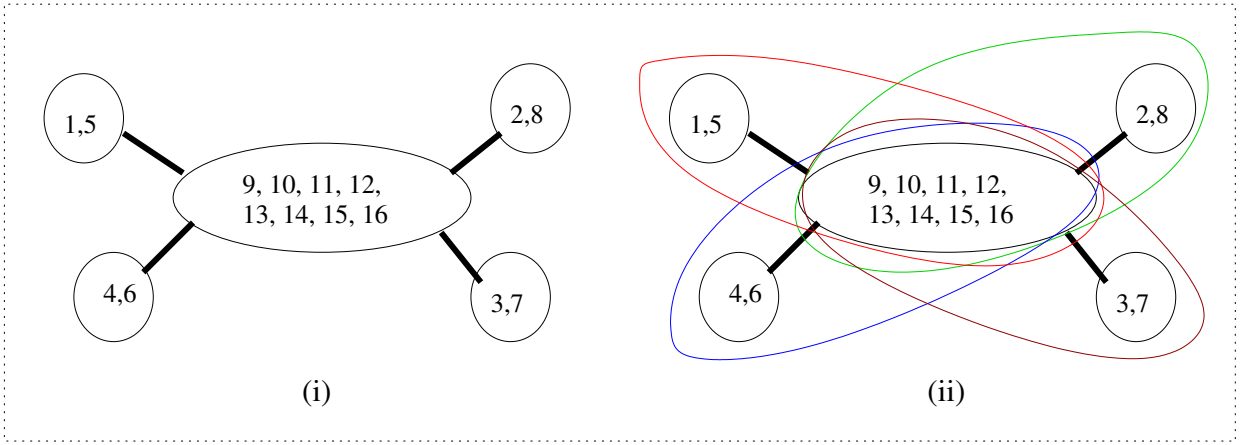


Points-to mapping: $a \longrightarrow \{A_1\}$, $p \longrightarrow \{A_1, A_2\}$, $b \longrightarrow \{A_2\}$

- Equivalent to solving Minimum Edge Clique Cover (ECC)
- NP-Complete problem

Empirical Analysis on Alias Graphs (4481 graphs)

- Only 11 graphs are interesting, up to 90 vertices
- In others, every connected component is a clique!



Alias Graph from H.264

Future Work

- A faster algorithm using modular decomposition properties
- Currently, the fastest is a $O(|V||E|)$ algorithm ([Gramm et al. 2009], in Haskell, using Patricia trees, does not seem simple to implement)

Development

Libraries used

PPL - The Parma Polyhedra Library `http://www.cs.unipr.it/ppl/`
CLooG - the Chunky Loop Generator `http://www.cloog.org`

Year	Commits
2009	497
2008	216
2007	36
2006	10

Weekly phonecalls

Every Wednesday 15.00 Pisa time `sip:00077723146@iptel.org`

`http://gcc.gnu.org/wiki/Graphite`

Bibliography

[Gramm et al. 2009] J. Gramm, J. Guo, F. Hüffner and R. Niedermeier. Data reduction and exact algorithms for clique cover. *J. Exp. Algorithmics, 14:2.2-2.15, 1999.*

[Trifunovic et al. 2009] K. Trifunovic, D. Nuzman, A. Cohen, A. Zaks and I. Rosen. Polyhedral-Model Guided Loop-Nest Auto-Vectorization. *In Parallel Architectures and Compilation Techniques (PACT'09), Raleigh, North Carolina, Sept. 2009*

Thank You for Your attention

Questions?