Fast Cache Modeling using Polyhedral Compilation

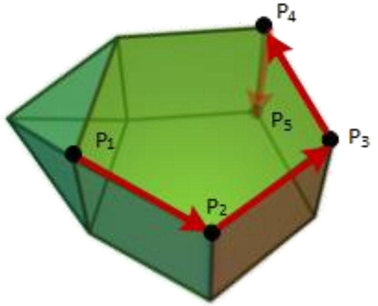
TOBIAS GYSI, **TOBIAS GROSSER,** LAURIN BRANDNER, TORSTEN HOEFLER

Historic Overview of Polyhedral Compilation

Polly Labs

Tensor Comprehensions

Mathematical Foundations



1960 - 1990

Basic Research

```
    int sum = 0;
    for(int i=0; i<4; ++i)
S0:   M[i] = i;
    for(int j=0; j<4; ++j)
S1:   sum += M[3-j];
```

1990 - 2000

Production Compilers GCC/Graphite, LLVM/Polly

GCC

Open64

2000 - 2015

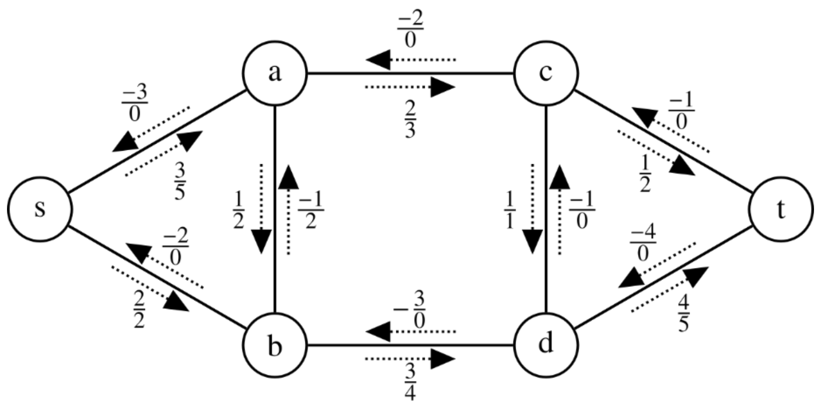
MLIR

2015 - 2019

Polyhedral Compilation: Analyses to build your own Magic

Memory Footprints

Data-Transfer Volumes



Data Dependences

The Cost of Data Movement Depend on Global State and Do Not Compose

```
int N = 1000;
for(int i = 0; i < N; i++) {
  for(int j = 0; j < i; j++) {
    for(int k = 0; k < j; k++) {
      A[i][j] -= A[i][k] * A[j][k];
    }
    A[i][j] /= A[j][j];
  }
  for(int k = 0; k < i; k++) {
    A[i][i] -= A[i][k] * A[i][k];
  }
  A[i][i] = sqrt(A[i][i]);
}
```

Cholesky kernel from https://sourceforge.net/projects/polybench/

- cache sizes (32k and 512k)
- cacheline size (64B)

percentage of cache misses?

L1 cache **1.6%**
L2 cache **1.4%**

most expensive memory access?

A[j][k]

amount of compulsory and capacity misses?

# compulsory misses	**31,752**
# capacity misses	**10,630,620**

HayStack Output for Cholesky Factorization

parameters:

- cache sizes (32k and 512k)
- cacheline size (64B)

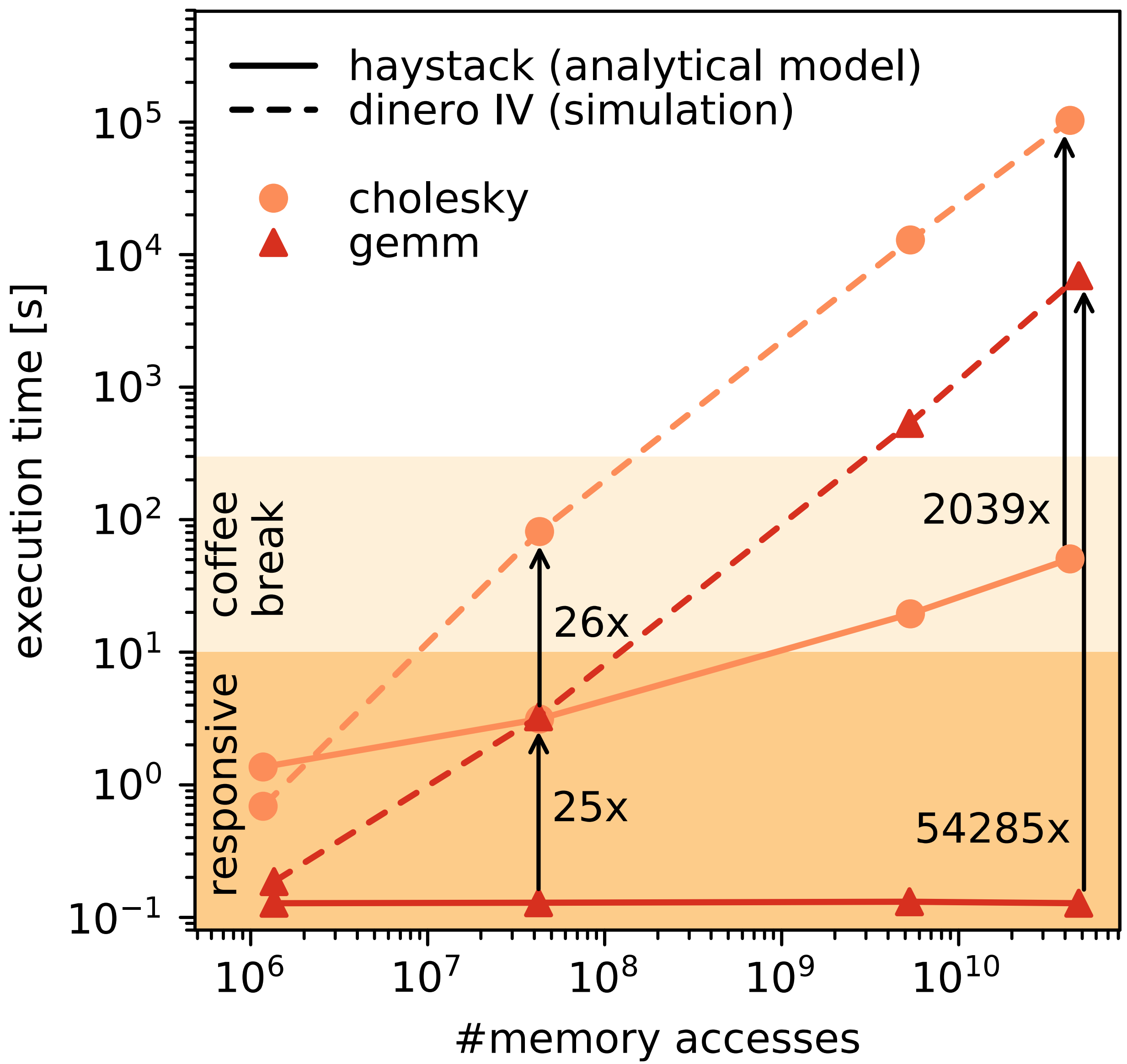
relative number of cache misses (statement)

```
5   for (int i = 0; i < N; i++) {
6     for (int j = 0; j < i; j++) {
7       for (int k = 0; k < j; k++) {
8         A[i][j] -= A[i][k] * A[j][k];
   ------------------------------------------------------------------------------
           ref   type   comp[%]    L1[%]      L2[%]      tot[%]     reuse[ln]
       A[i][j]   rd     0.00459    0.00000    0.00000    24.86910   8,10
       A[i][k]   rd     0.00000    0.00000    0.00000    24.86910   8,10
       A[j][k]   rd     0.00000    1.58635    1.38213    24.86910   8,10,13,15
       A[i][j]   wr     0.00000    0.00000    0.00000    24.86910   8
   ------------------------------------------------------------------------------
```

absolute number of cache misses (program)

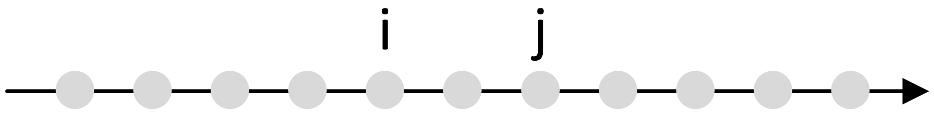
```
compulsory:                        31'752
capacity (L1):                 10'630'620
capacity (L2):                  9'258'460
total:                        668'166'500
```

Comparison to Simulation



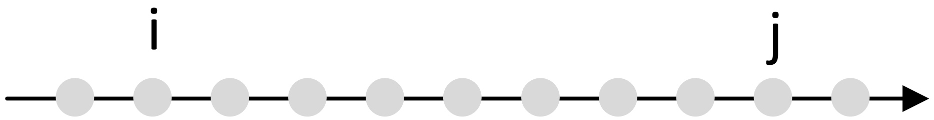
Symbolic Counting Avoids the Explicit Enumeration

1d illustration



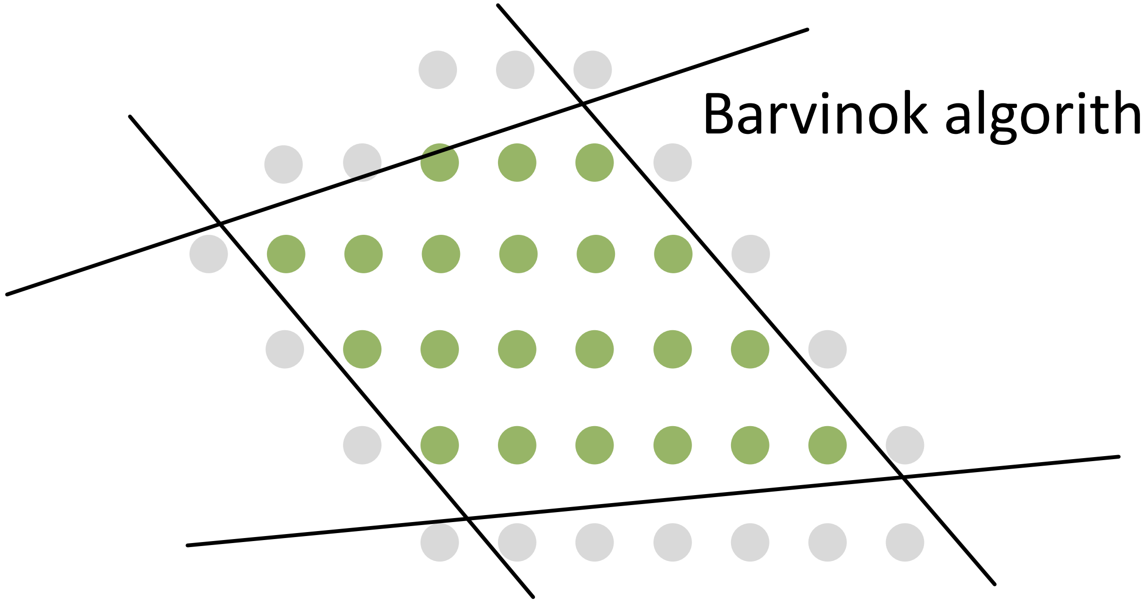
enumeration #points = 3

symbolic #points = j-i+1 = 3



enumeration #points = 9

symbolic #points = j-i+1 = 9



Barvinok algorithm

Alexander I. Barvinok, *A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension is Fixed*. 1994.

The LRU Stack Distance Allows Us to Model Fully Associative Caches

example

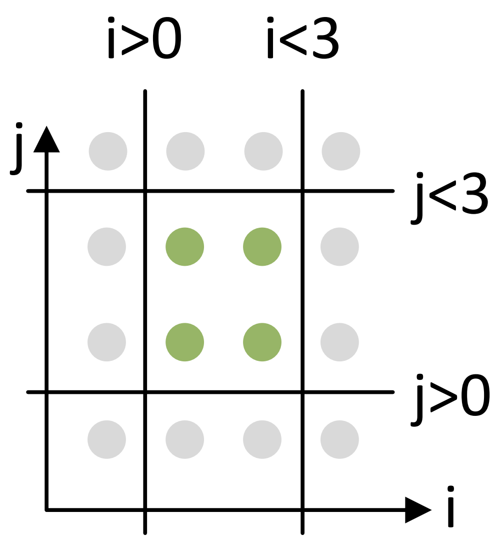
```
    int sum = 0;
    for(int i=0; i<4; ++i)
S0:   M[i] = i;
    for(int j=0; j<4; ++j)
S1:   sum += M[3-j];
```

deliberately **generic model**

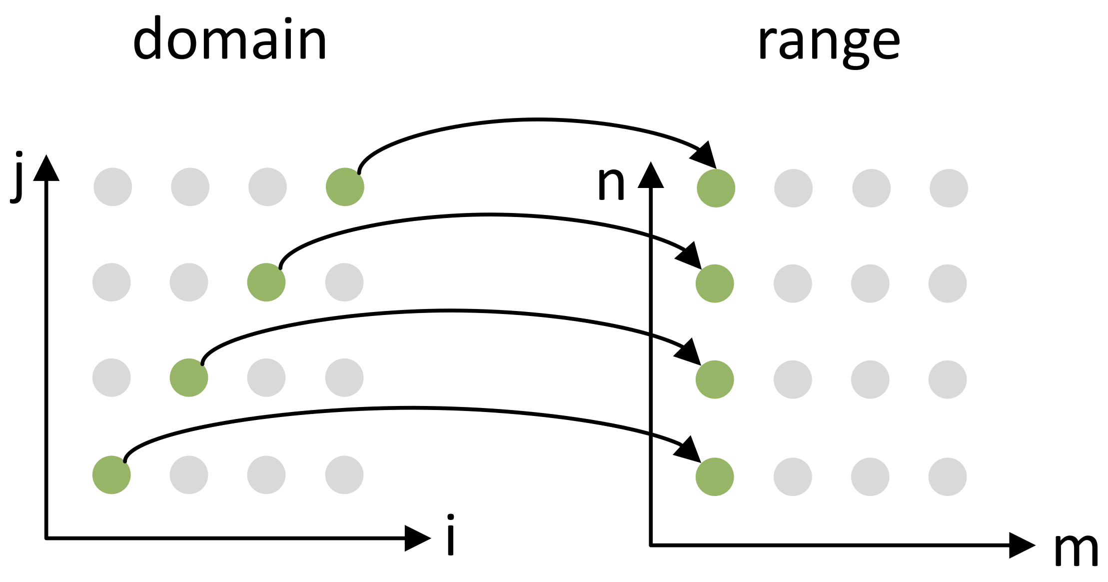
Richard L Mattson, Jan Gecsei, Donald R Slutz, and Irving L Traiger, *Evaluation techniques for storage hierarchies.* 1970.

Integer Sets and Relations

affine set



affine relation

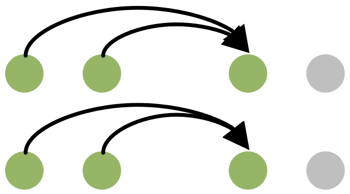


$\mathbf{S} = \{(i,j) : 0 < i < 3 \wedge 0 < j < 3\}$ $\quad \mathbf{R} = \{(i,j) \rightarrow (m,n) : 0 \leq i < 4 \wedge i = j \wedge n = j \wedge m = 0\}$

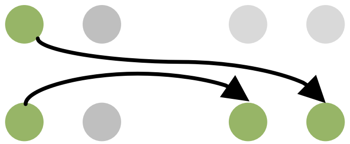
Sven Verdoolaege, *isl: An integer set library for the polyhedral model.* 2010.

Computation with Affine Sets and Relations

$\boldsymbol{A}$

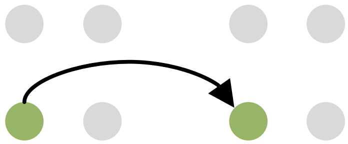


$\boldsymbol{B}$



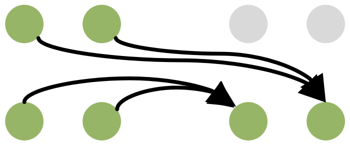
intersection

$\boldsymbol{A} \cap \boldsymbol{B}$



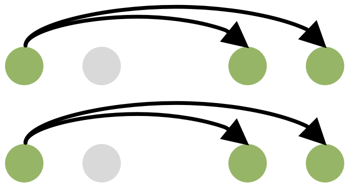
composition

$\boldsymbol{B} \circ \boldsymbol{A}$

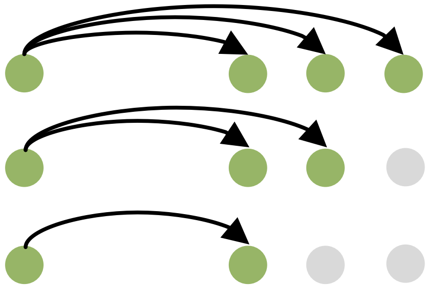


inverse

$\boldsymbol{A}^{-1}$

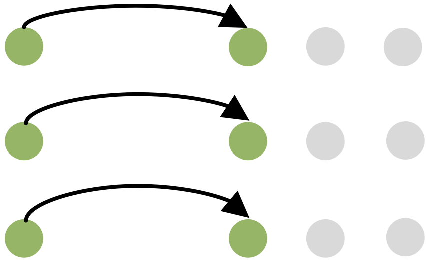


$\boldsymbol{C}$



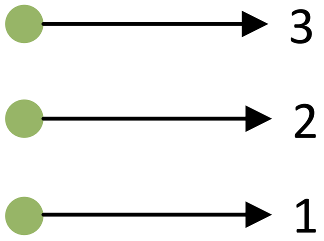
lexmin

$lexmin(\boldsymbol{C})$



cardinality

$|\boldsymbol{C}|$

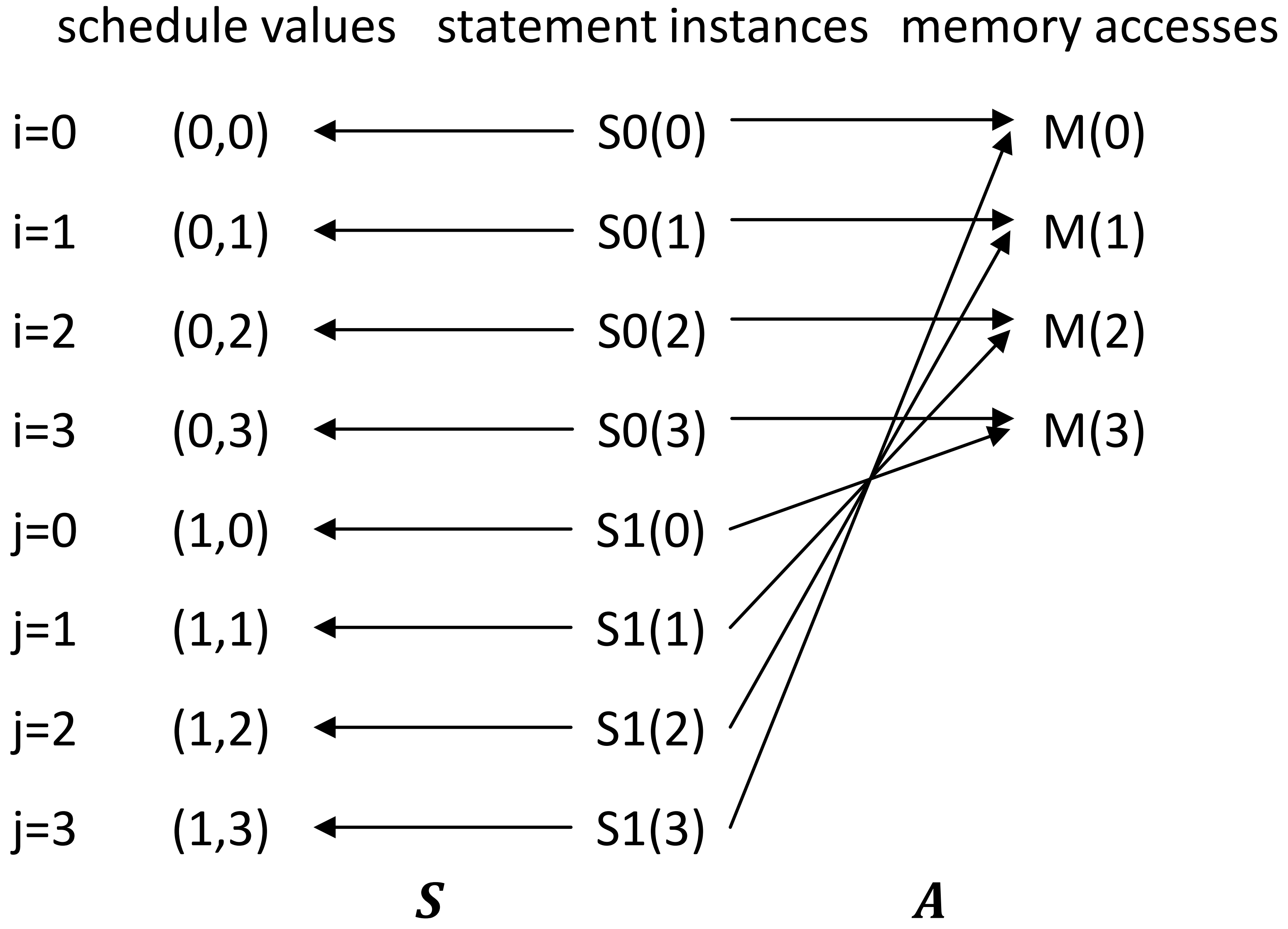


Sven Verdoolaege, *isl: An integer set library for the polyhedral model.* 2010.

Express Programs by Integer Sets and Relations

example

```
    int sum = 0;
    for(int i=0; i<4; ++i)
S0:   M[i] = i;
    for(int j=0; j<4; ++j)
S1:   sum += M[3-j];
```



Express Programs by Integer Sets and Relations

schedule values

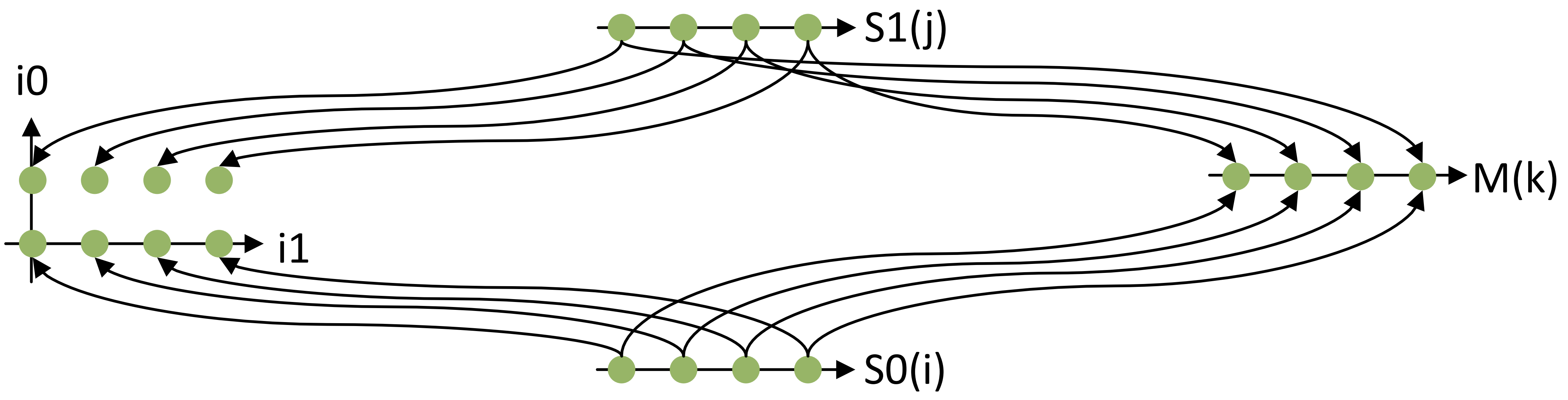
$\{(i0, i1) : 0 \leq i0 < 2 \wedge$
$0 \leq i1 < 4\}$

statement instances

$\{S0(i) : 0 \leq i < 4;$
$S1(j) : 0 \leq j < 4\}$

memory accesses

$\{M(k) : 0 \leq k < 4\}$



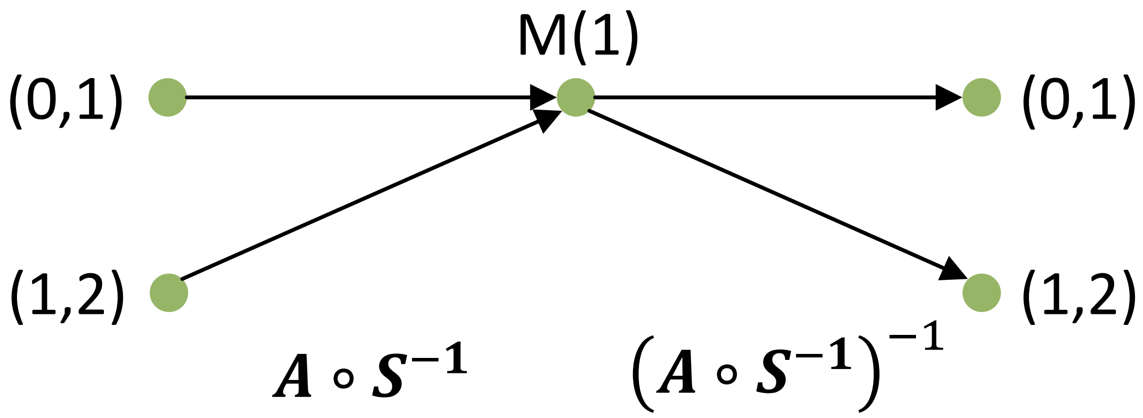
$\boldsymbol{S} = \{S0(i) \rightarrow (0, i) : 0 \leq i < 4;$
$S1(j) \rightarrow (1, j) : 0 \leq j < 4\}$

$\boldsymbol{A} = \{S0(i) \rightarrow M(k) : k = i;$
$S1(j) \rightarrow M(k) : k = 3 - j\}$

Identify Next Access of the Same Memory Location

example

```
    int sum = 0;
    for(int i=0; i<4; ++i)
S0:   M[i] = i;
    for(int j=0; j<4; ++j)
S1:   sum += M[3-j];
```



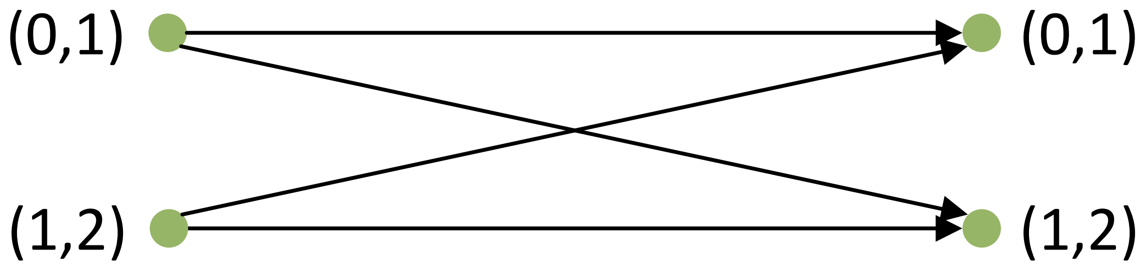
Identify Next Access of the Same Memory Location

example

```
     int sum = 0;
     for(int i=0; i<4; ++i)
S0:    M[i] = i;
     for(int j=0; j<4; ++j)
S1:    sum += M[3-j];
```

$$\boldsymbol{E} = (\boldsymbol{A} \circ \boldsymbol{S}^{-1})^{-\mathbf{1}} \circ \boldsymbol{A} \circ \boldsymbol{S}^{-1}$$

$$\boldsymbol{N} = lexmin(\boldsymbol{E} \cap \boldsymbol{L}_{\prec})$$



$$lexmin(\boldsymbol{E} \cap \boldsymbol{L}_{\prec})$$

$$\boldsymbol{L}_{\prec} = \{(\mathrm{i0}, \mathrm{i1}) \rightarrow (\mathrm{j0}, \mathrm{j1}) : (\mathrm{i0} < \mathrm{j0}) \vee (i0 = j0 \wedge i1 < j1)\}$$

Compute the LRU Stack Distance

example

```
     int sum = 0;
     for(int i=0; i<4; ++i)
S0:    M[i] = i;
     for(int j=0; j<4; ++j)
S1:    sum += M[3-j];
```

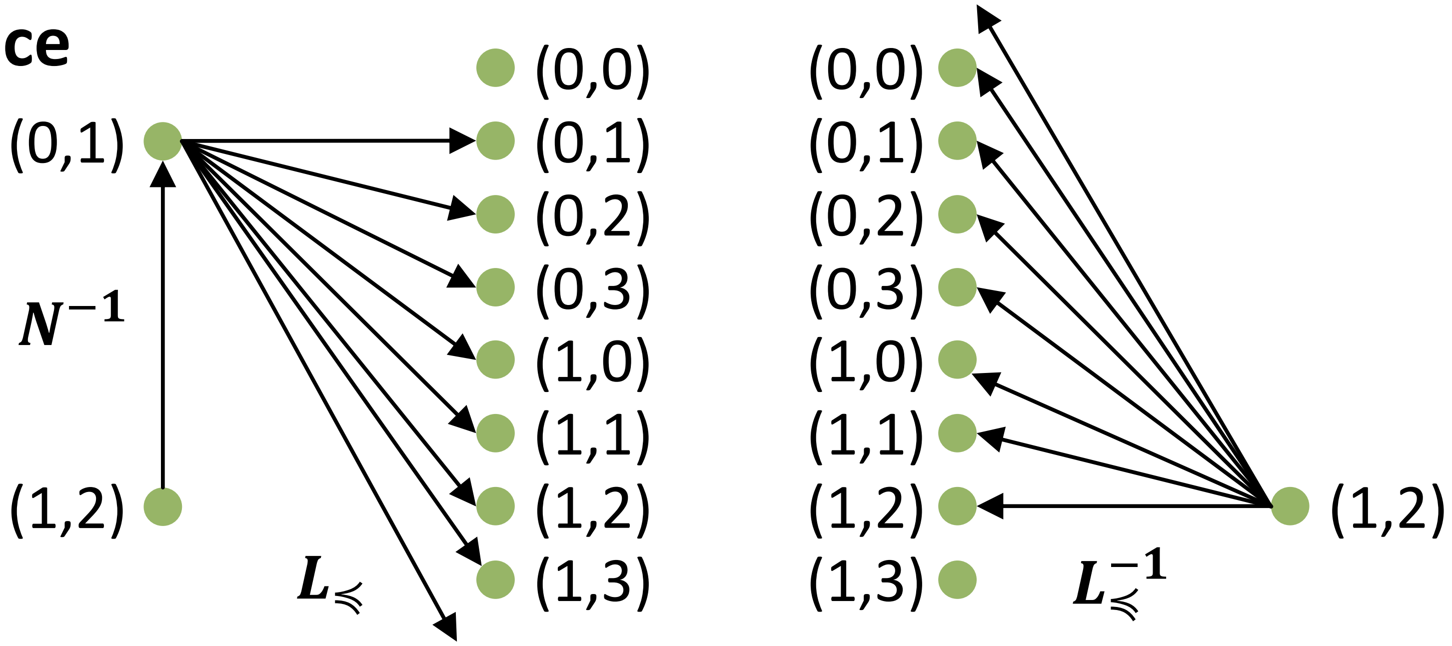
$$E = (A \circ S^{-1})^{-1} \circ A \circ S^{-1}$$

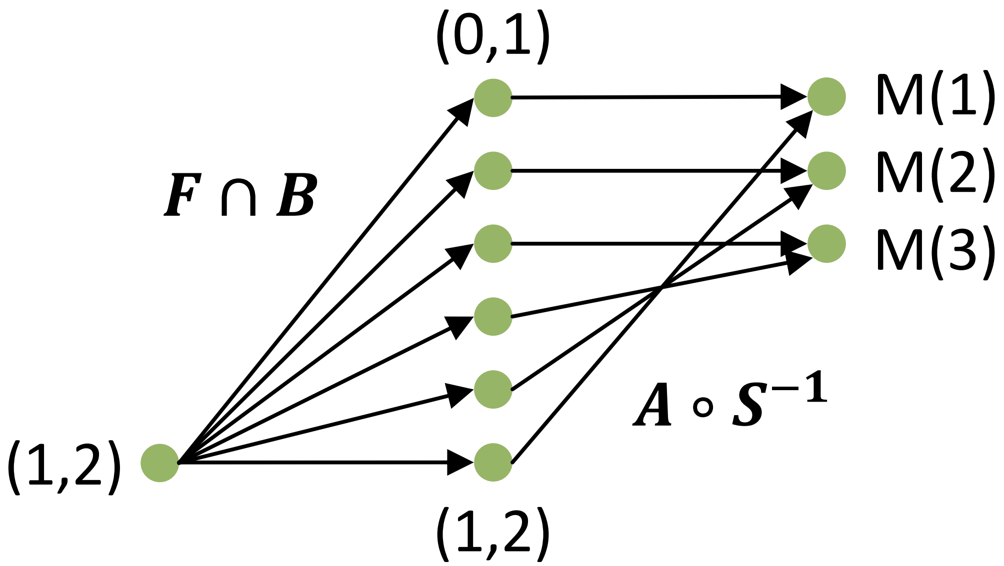
$$N = lexmin(E \cap L_{\prec})$$

$$F = L_{\preccurlyeq} \circ N^{-1}$$

$$B = L_{\preccurlyeq}^{-1}$$

$$D = \{\left|A \circ S^{-1} \circ (F \cap B) \circ S\right|\}$$





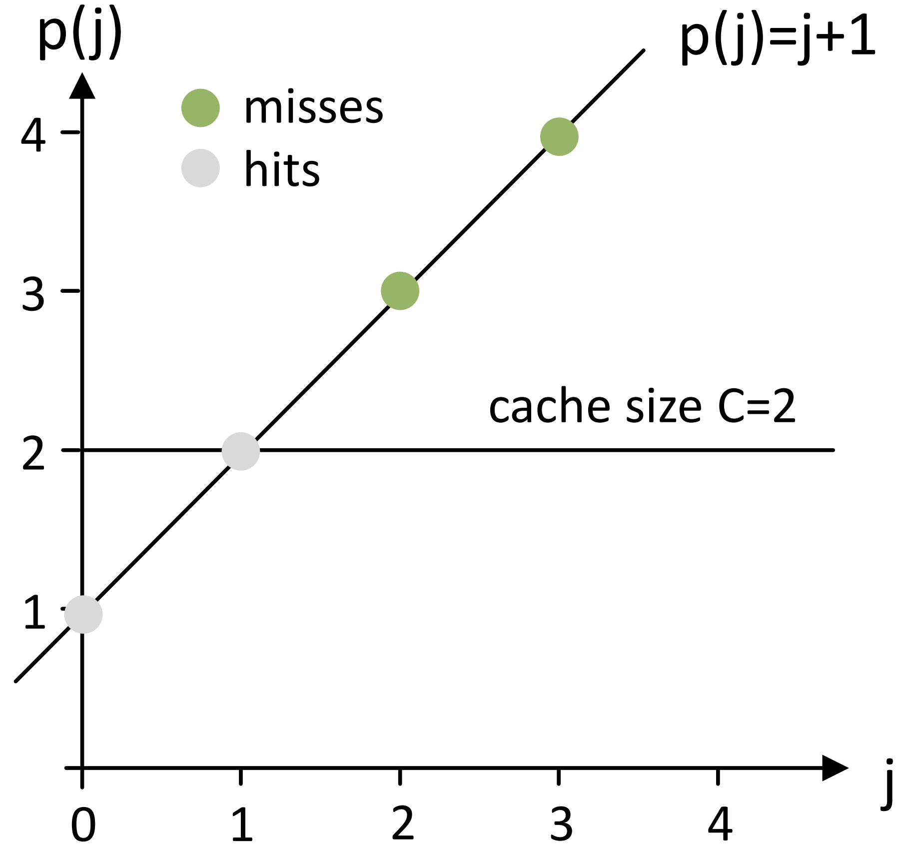
Count the Cache Misses Given the Backward LRU Stack Distance

example

```
     int sum = 0;
     for(int i=0; i<4; ++i)
S0:    M[i] = i;
     for(int j=0; j<4; ++j)
S1:    sum += M[3-j];
```

$$\mathbf{D} = \{|\boldsymbol{A} \circ \boldsymbol{S}^{-1} \circ (\boldsymbol{F} \cap \boldsymbol{B}) \circ \boldsymbol{S}|\}$$

$$= \{S1(j) \to j+1 :\ 0 \leq j < 4\}$$



$$|\mathbf{M}| = |\{S1(j) :\ p(j) > C \wedge 0 \leq j < 4\}|$$
$$= |\{S1(2), S1(3)\}| = 2$$

Kristof Beyls and Erik H D'Hollander, *Generating cache hints for improved program efficiency*. 2005.

Enumerate the Non-Affine Dimensions of the Stack Distance Polynomials

$\mathbf{P} = \{S0(i,j) \rightarrow i + \boldsymbol{j^2} : 0 \leq i,j < 3\}$

(0,0) (1,0) (2,0)

(0,1) (1,1) (2,1)

(0,2) (1,2) (2,2)

π_i

$\mathbf{E} = \{j : 0 \leq j < 3\}$

0

1

2

$\mathbf{P_{j=0}} = \{S0(i) \rightarrow i + \mathbf{0} : 0 \leq i < 3\}$

0 1 2

$\mathbf{P_{j=1}} = \{S0(i) \rightarrow i + \mathbf{1} : 0 \leq i < 3\}$

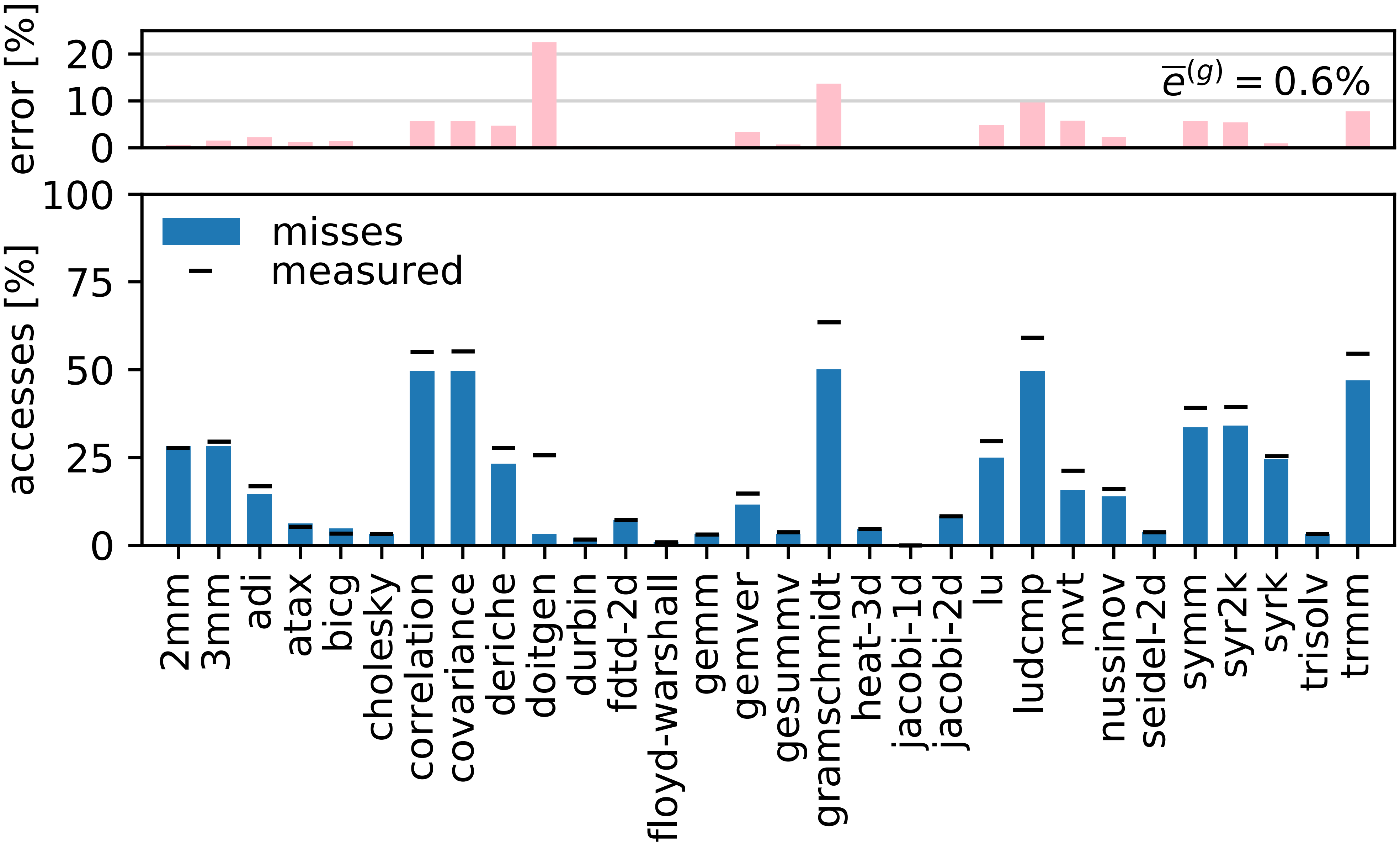
0 1 2

$\mathbf{P_{j=2}} = \{S0(i) \rightarrow i + \mathbf{4} : 0 \leq i < 3\}$

0 1 2

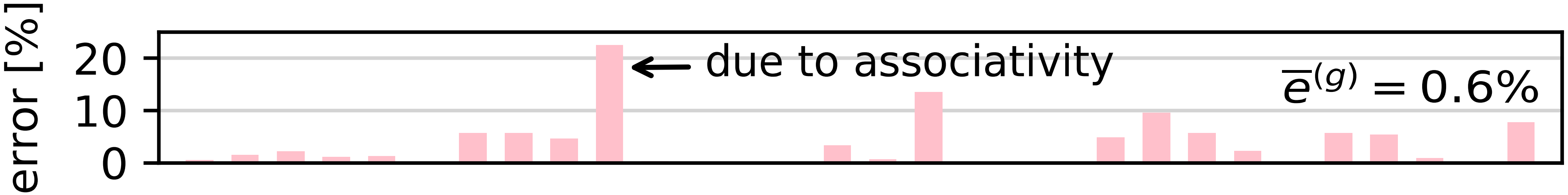
12.4x speedup
due to **partial enumeration**

Accuracy of HayStack for the L1 Cache of Our Test System

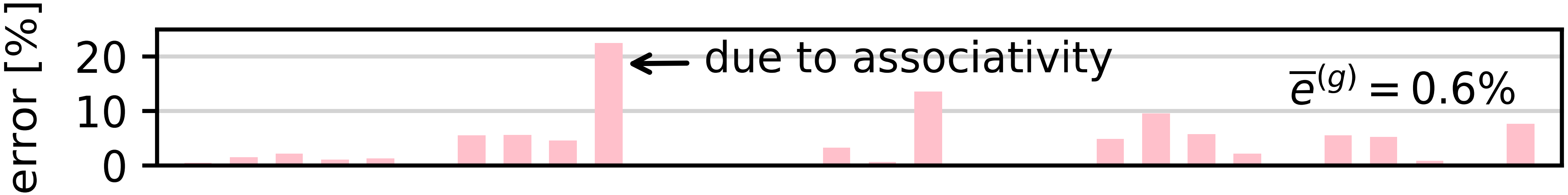


Error of HayStack Compared to Simulation (Dinero IV)

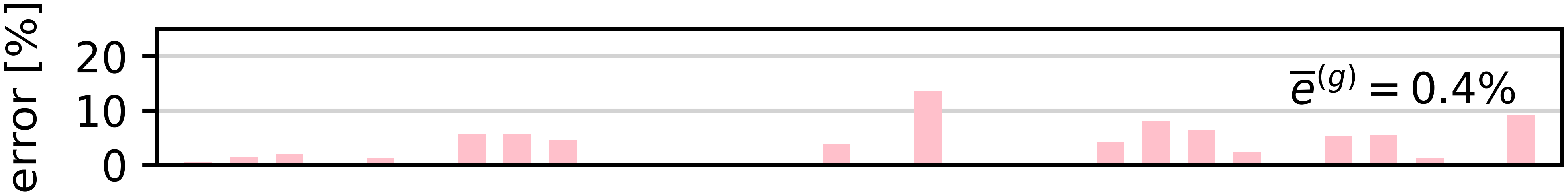
HayStack
(fully associative)



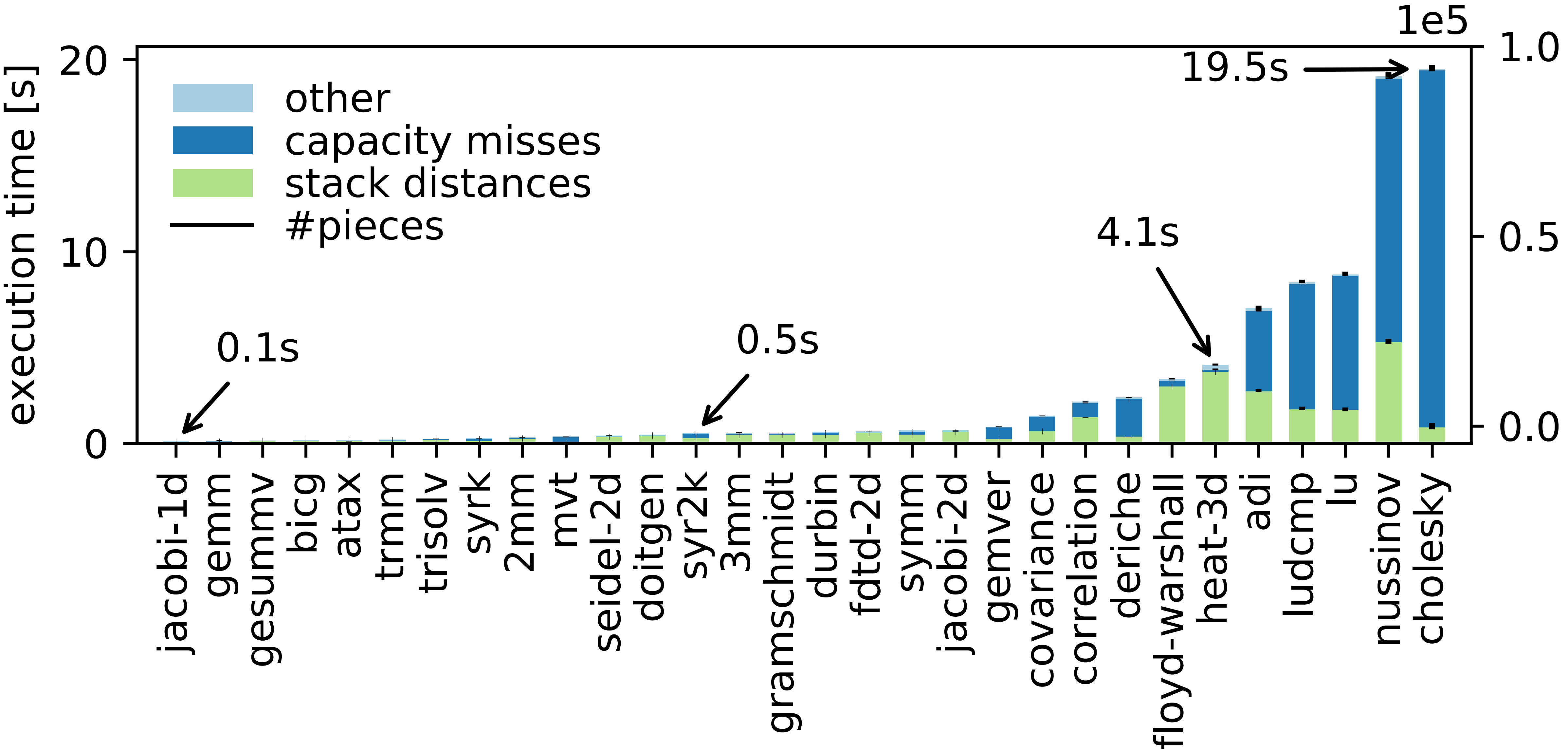
Dinero IV
(fully associative)



Dinero IV
(8-way associative)



Performance of HayStack for the Large Problem Size of PolyBench

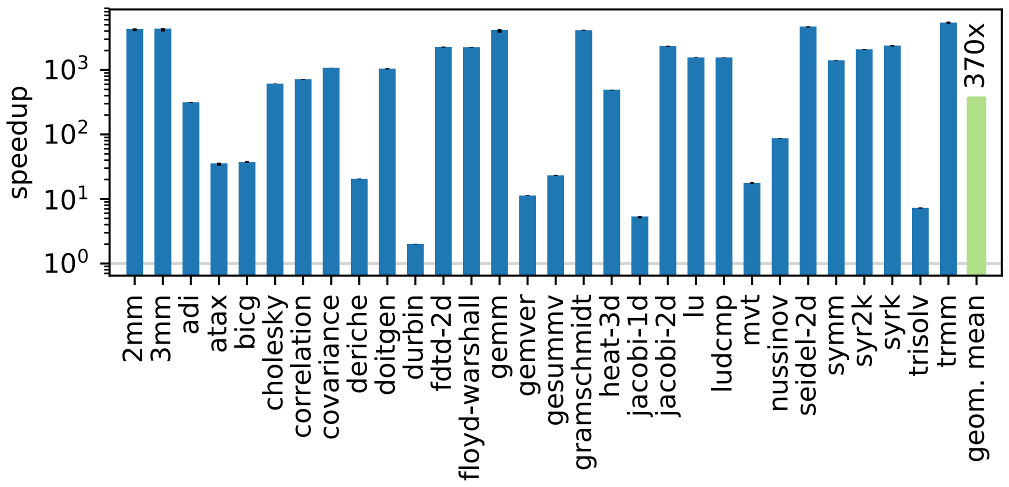


Performance of HayStack Compared to PolyCache and Dinero

Dinero IV

- simulator
- setup to simulate full associativity
- problem size dependent performance

370x speedup

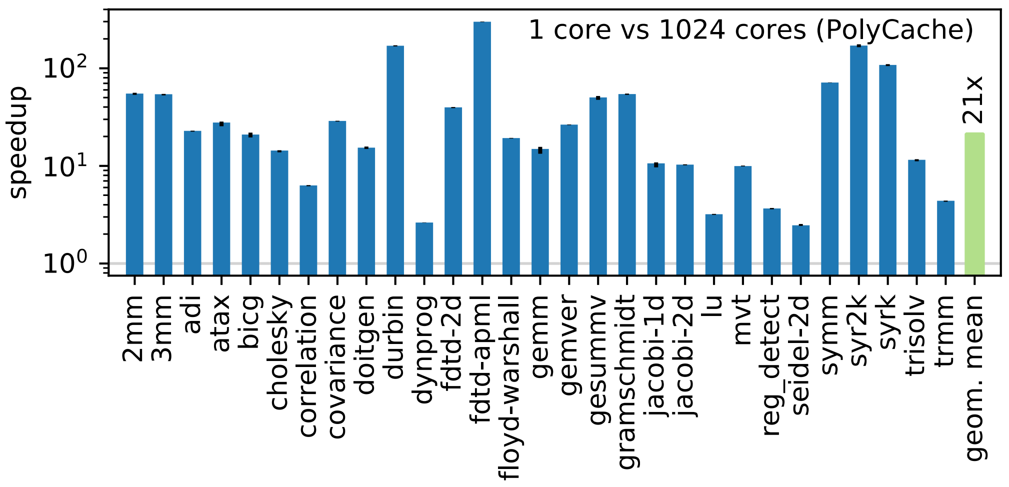


Jan Elder and Mark D. Hill, *Dinero IV Trace-Driven Uniprocessor Cache Simulator*. 2003.

PolyCache

- analytical cache model
- models set associativity
- one core per cache set

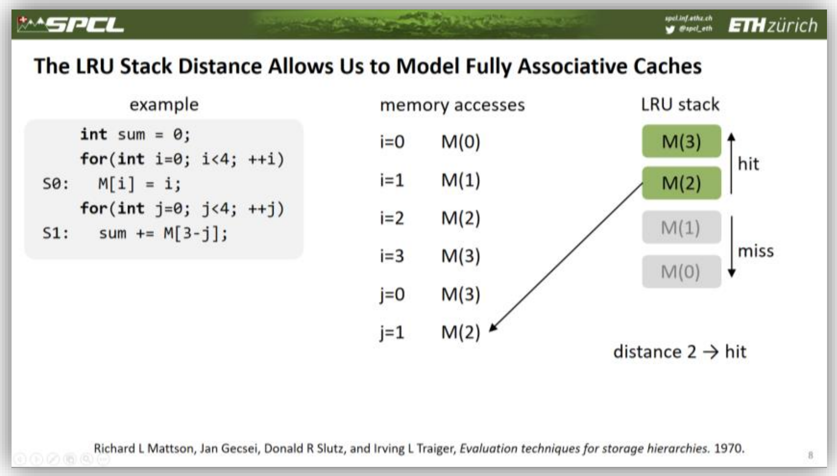
21x speedup



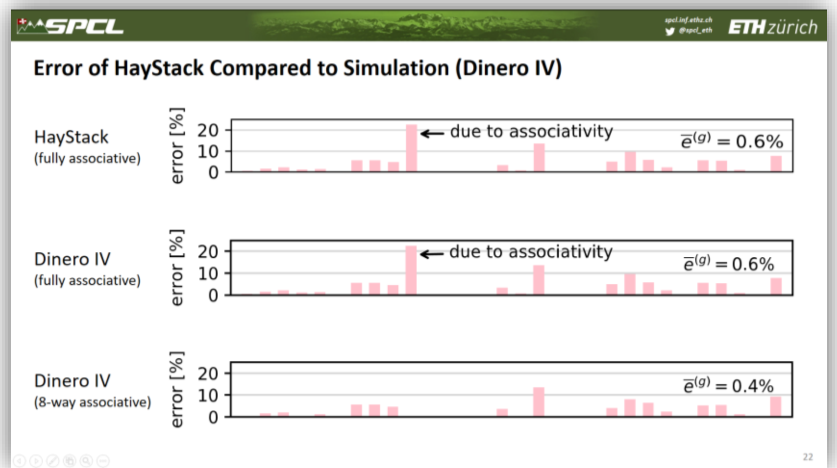
Wenlei Bao, Sriram Krishnamoorthy, Louis-Noel Pouchet, and P Sadayappan, *Analytical modeling of cache behavior for affine programs.* 2017.

Conclusion

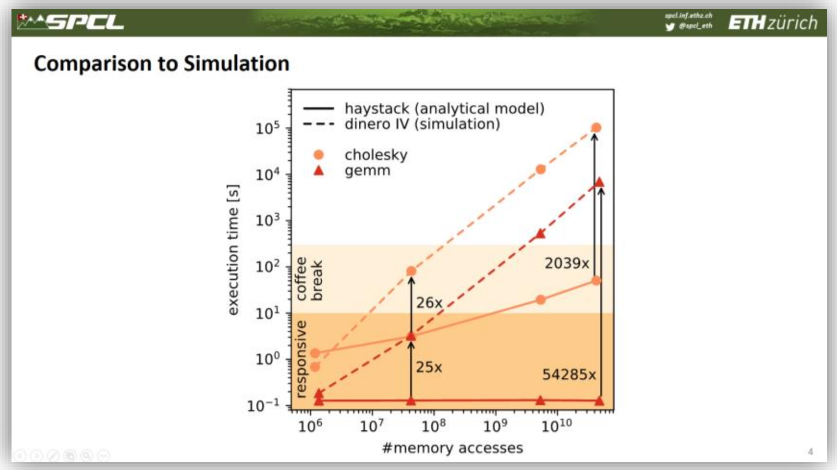
generic model of **fully associative caches**



accurate results compared to measurements



fast enough to provide **interactive feedback**



symbolic counting avoids explicit enumeration