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

A Fast Analytical Model of Fully Associative Caches
The Cost of Data Movement Depends on Global State and Does Not Compose

```c
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[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/

- 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

relative number of cache misses (statement)

```c
for (int i = 0; i < N; i++) {
    for (int j = 0; j < i; j++) {
        for (int k = 0; k < j; k++) {
        }
    }
}
```

<table>
<thead>
<tr>
<th>ref</th>
<th>type</th>
<th>comp[%]</th>
<th>L1[%]</th>
<th>L2[%]</th>
<th>tot[%]</th>
<th>reuse[ln]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[i][j]</td>
<td>rd</td>
<td>0.00459</td>
<td>0.00000</td>
<td>0.00000</td>
<td>24.86910</td>
<td>8,10</td>
</tr>
<tr>
<td>A[i][k]</td>
<td>rd</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>24.86910</td>
<td>8,10</td>
</tr>
<tr>
<td>A[j][k]</td>
<td>rd</td>
<td>0.00000</td>
<td>1.58635</td>
<td>1.38213</td>
<td>24.86910</td>
<td>8,10,13,15</td>
</tr>
<tr>
<td>A[i][j]</td>
<td>wr</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>24.86910</td>
<td>8</td>
</tr>
</tbody>
</table>

absolute number of cache misses (program)

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

parameters:
- cache sizes (32k and 512k)
- cacheline size (64B)
Comparison to Simulation

- haystack (analytical model)
- dinero IV (simulation)

- cholesky
- gemm

**Execution Time vs. Memory Accesses**

- 1 day
- 1 hour
- 1 minute
- 1 second
Symbolic Counting Avoids the Explicit Enumeration

1d illustration

![Diagram showing symbolic and enumeration counting methods with points (i, j) and #points = j - i + 1 calculation examples.

The LRU Stack Distance Allows Us to Model Fully Associative Caches

example

```c
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

Compute the LRU Stack Distance

example

```c
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];
```

apply **symbolic counting** once

$p(j) = j + 1$
Count the Cache Misses Given the LRU Stack Distance

example

```c
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];
```

apply **symbolic counting** twice

many different pieces and sometimes **non-affine** polynomials

\[ p(j) = j + 1 \]

\[ |\{ j : p(j) > C \land 0 \leq j < 4 \}| = 2 \]
Some Access Patterns Result in Non-Linearities

example

```c
int sum = 0;
for(int t=0; t<4; ++t) {
    for(int i=0; i<4; ++i)
        S0: M[i] = i;
    for(int m=0; m<t; ++m)
        for(int n=0; n<t; ++n)
            N[m][n] = t;
    for(int j=0; j<4; ++j)
        S1: sum += M[3-j];
}
```

original

```
\begin{align*}
    p(j) &= j + 1 \\
    p(j, t) &= t^2 + j + 1
\end{align*}
```

additional time loop

partial enumeration
Enumerate the Non-Affine Dimensions

\[ p(j, t) = t^2 + j + 1 \]

\[
\begin{align*}
(0,0) & \quad (1,0) & \quad (2,0) \\
(0,1) & \quad (1,1) & \quad (2,1) \\
(0,2) & \quad (1,2) & \quad (2,2)
\end{align*}
\]

\[ 0 \leq (j, t) < 3 \]

\[
\begin{align*}
p_{t=0}(j) &= 0 + j + 1 \\
p_{t=1}(j) &= 1 + j + 1 \\
p_{t=2}(j) &= 4 + j + 1
\end{align*}
\]

12.4x speedup due to partial enumeration
Modelling Cache Lines Introduces Floor Terms

```
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];
```

```
original

\[ p(j) = j + 1 \]
```

```
modelling cache lines

\[ p(j) = \frac{j}{2} \left( \left\lfloor \frac{j}{2} \right\rfloor - \left\lfloor \frac{j-1}{2} \right\rfloor \right) + 1 \]
```

equalization and rasterization
Split the Domain to Eliminate Floor Terms

\[ p(j) = \frac{j}{2} \left( \left\lfloor \frac{j}{2} \right\rfloor - \left\lfloor \frac{j-1}{2} \right\rfloor \right) + 1 \]

\[ 0 \leq j < 4 \]

\[ p(j)_{j \% 2 = 0} = \frac{j}{2} 1 + 1 \]

\[ \begin{array}{c|c|c|c}
0 & 1 & 2 & 3 \\
\end{array} \]

\[ p(j)_{j \% 2 > 0} = \frac{j}{2} 0 + 1 \]

\[ \begin{array}{c|c|c|c}
1 & 2 & 3 & \end{array} \]

1.9x speedup due to equalization
Accuracy of HayStack for the L1 Cache of Our Test System

\[ \bar{e} = 0.6\% \]

- **misses**
- **measured**
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

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

21x speedup


Conclusion

generic model of **fully associative caches**

**accurate results** compared to measurements

fast enough to provide **interactive feedback**

excellent performance compared to **alternatives**