# A tool for reconstructing codes from memory traces

José M. Andión, Gabriel Rodríguez and Juan Touriño

UNIVERSIDADE DA CORUÑA

Mahmut T. Kandemir

PennState

## Goal

**Rebuilding affine loop nests from a trace of memory accesses**

- ✦ without user intervention
- ✦ without usage of source or binary code

## Memory Trace

```
 1 | 0x1e2d140
 2 | 0x1e2d140
...
30 | 0x1e2d140
31 | 0x1e2d240
32 | 0x1e2d248
33 | 0x1e2d240
34 | 0x1e2d248
...
88 | 0x1e2d248
89 | 0x1e2d340
90 | 0x1e2d348
91 | 0x1e2d350
92 | 0x1e2d340
93 | 0x1e2d348
94 | 0x1e2d350
...
```

## Problem Formulation & Reconstruction Method

### Affine loop

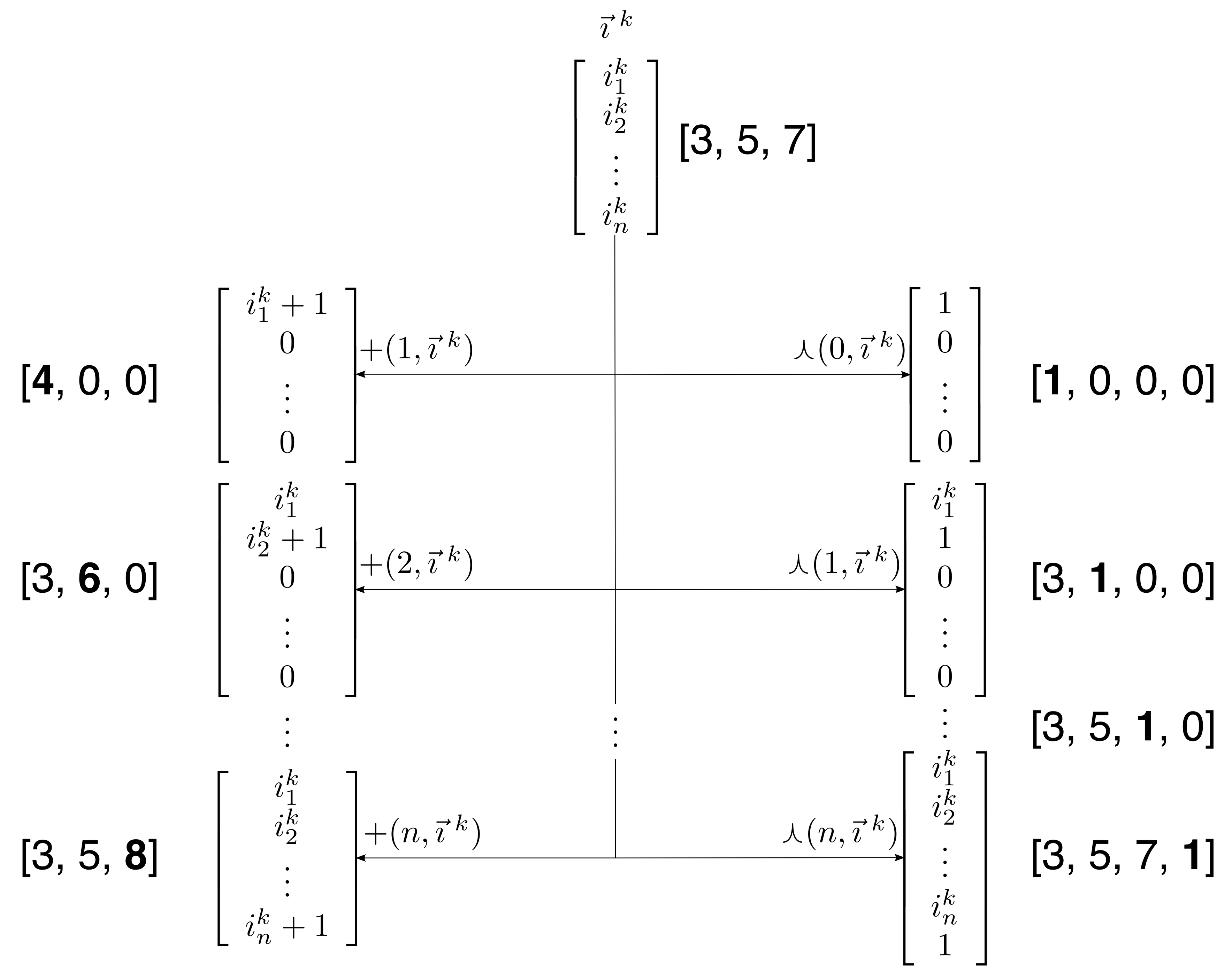
$$
\begin{array}{l}
\texttt{DO}\ i_1 = 0,\ u_1(\vec{\imath}) \\
\quad \ldots \\
\quad \texttt{DO}\ i_n = 0,\ u_n(\vec{\imath}) \\
\qquad V[f_1(\vec{\imath})]\ldots[f_m(\vec{\imath})]
\end{array}
$$

### The access can be rewritten as a linear combination of the loop indices

$$V[f_1(\vec{\imath})]\ldots[f_m(\vec{\imath})] = V[c_0 + i_1c_1 + \ldots + i_nc_n]$$

$$\sigma^k = V(\vec{\imath}^{\,k+1}) - V(\vec{\imath}^{\,k})$$

### Traversal of a tree-like space to generate the observed strides




## Reconstructed Code

```
#define N 32
double p[N], A[N][N];
for(i = 0; i < N; ++i) {
  x = A[i][i];
  for(j = 0; j <= i-1; ++j)
    x = x - A[i][j] * A[i][j];
  p[i] = 1.0 / sqrt(x);
  for(j = i+1; j < N; ++j) {
    x = A[i][j];
    for(k = 0; k <= i-1; ++k)
      x = x - A[j][k] * A[i][k];
    A[j][i] = x * p[i];
  }
}
```

## Applications

- ✦ Hardware and software prefetching
- ✦ Data placement
- ✦ Dependence analysis
- ✦ Design of embedded memories
- ✦ Trace compression

## Web

Free Download!

## Experimental Evaluation