Far Fetched Prefetching?

Tomofumi Yuki — INRIA Rennes

Antoine Morvan — ENS Cachan Bretagne

Steven Derrien — University of Rennes 1

Memory Optimizations

- Memory Wall
  - Memory improves slower than processor
  - Very little improvement in latency
- Making sure processors have data to consume
  - Software: tiling, prefetching, array contraction
  - Hardware: cache, prefetching
- Important for both speed and power
  - Further memory takes more power

Prefetching

- Anticipate future memory accesses and start data transfer in advance
  - Both HW and SW versions exist
- Hides latency of memory accesses
  - Cannot help if bandwidth is the issue
- Inaccurate prefetch is harmful
  - Consumes unnecessary bandwidth/energy
  - Pressure on caches

This talk

- A *failure experience* based on our attempt to improve prefetching
- We target cases where trip count is small
  - Prefetch instructions must be placed in previous iterations of the **outer** loop
- Use polyhedral techniques to find where to insert prefetch

Outline

- Introduction
- Software Prefetching
  - When it doesn't work
  - Improving prefetch placement
- Code Generation
- Simple Example
- Summary and Next Steps

Software Prefetching [Mowry 96]

- Shift the iterations by *prefetching distance*
- Simple yet effective method for regular computations

```
for (i=0; i<N; i++)
   … = foo(A[i], …)
```

```
for (i=-4; i<0; i++)
   prefetch(A[i+4]);
for (i=0; i<N-4; i++)
   … = foo(A[i], …)
   prefetch(A[i+4]);
for (i=N-4; i<N; i++)
   … = foo(A[i], …)
```

Prologue

Epilogue

[Lee, Kim, and Vuduc HiPEAC 2013]

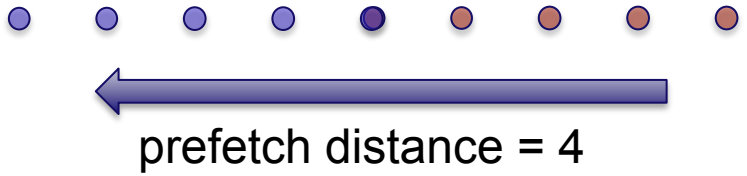
When Prefetching Works, When It Doesn't, and Why

- Difficult to statically determine prefetching distance
  - They suggest use of tuning
  - Not the scope of our work
- Interference with HW prefetchers
- Cannot handle short stream of accesses
  - Limitation for both software and hardware
  - We try to handle short streams

Problem with Short Streams

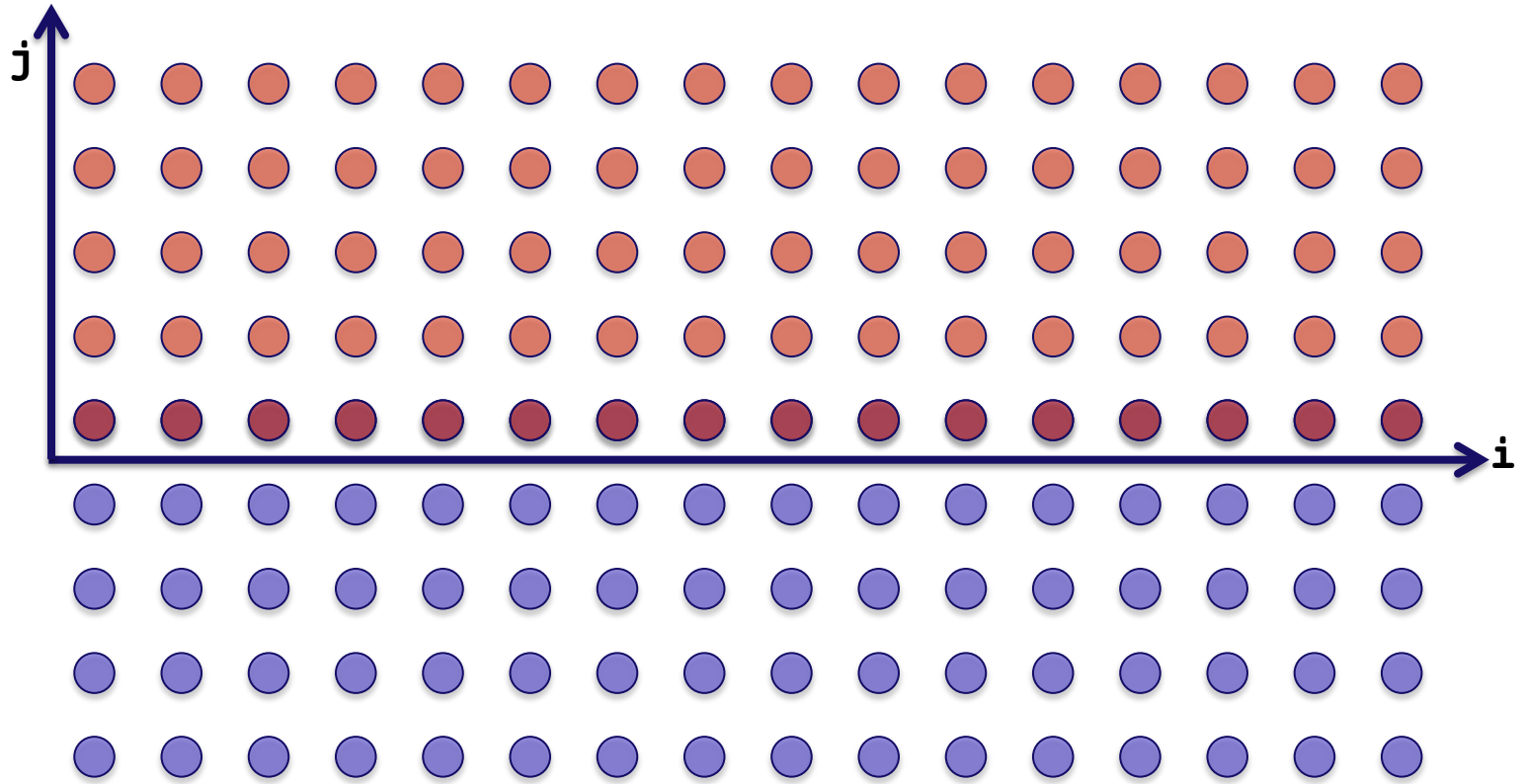
- When N=5
  - Most prefetches are issued “too early”
  - Not enough computation to hide the latency
- Simply translating iterations in the innermost loop is not sufficient

```
for (i=-4; i<0; i++)
   prefetch(A[i+4]);
for (i=0; i<1; i++)
   … = foo(A[i], …)
   prefetch(A[i+4]);
for (i=1; i<5; i++)
   … = foo(A[i], …)
```



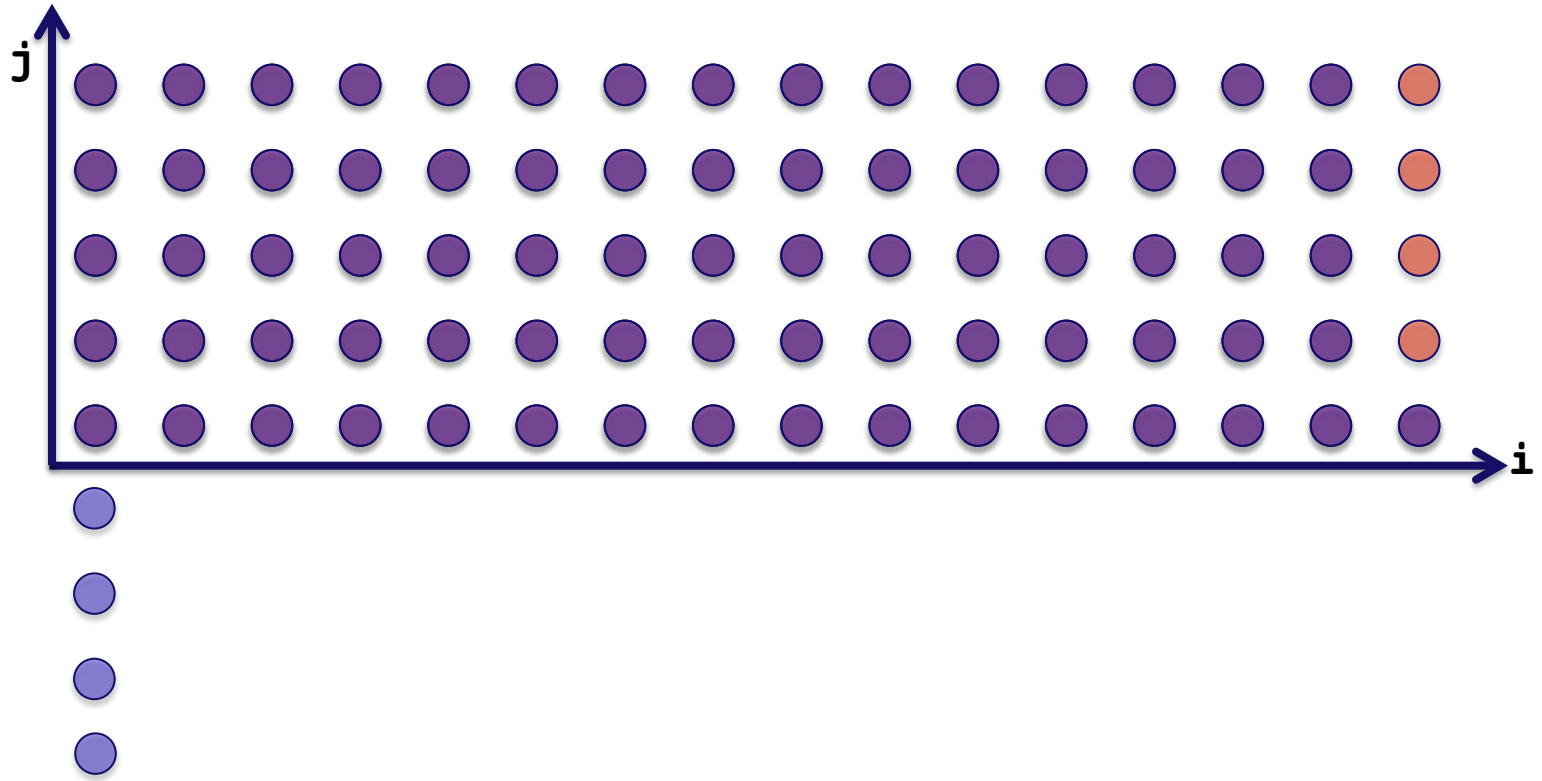
2D Illustration

- Large number of useless prefetches



Lexicographical Shift

- The main idea to improve prefetch placement



Outline

- Introduction
- Software Prefetching
- Code Generation
  - Polyhedral representation
  - Avoiding redundant prefetch
- Simple Example
- Summary and Next Steps

Prefetch Code Generation

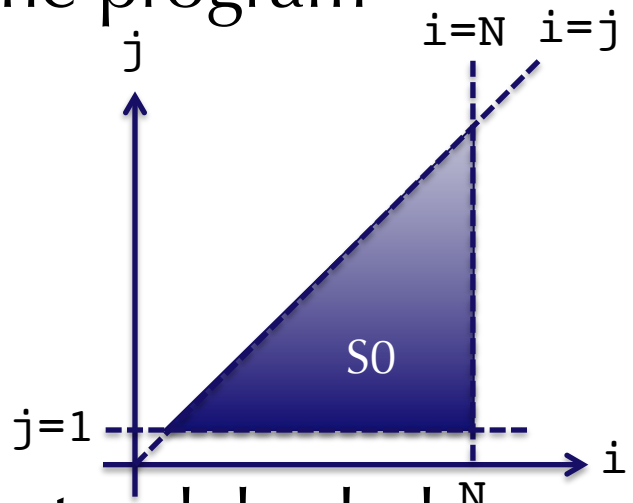
- Main Problem: Placement
  - Prefetching distance
    - Manually given parameter in our work
  - Lexicographically shifting the iteration spaces
  - Avoid reading the same array element twice
  - Avoid prefetching the same line multiple times
- We use the polyhedral representation to handle these difficulties

Polyhedral Representation

- Represent iteration space (of polyhedral programs) as a set of points (polyhedra)
- Simply a different *view* of the program

```
for (i=1; i<=N; i++)
  for (j=1; j<=i; j++)
    S0
```

Domain(S0) = [i,j] : 1≤j≤i≤N



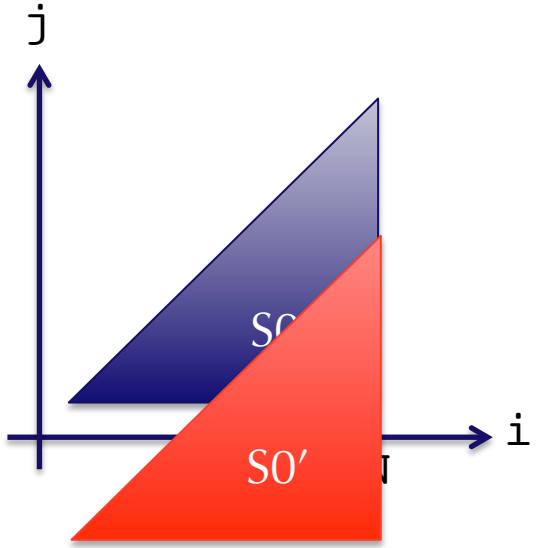
- Manipulate using convenient polyhedral representation, and then generate loops

Transforming Iteration Spaces

- Expressed as affine transformations
  - Shifting by constant is even simpler
- Example: Shift the iteration by 4 along `j`
  - `(i,j→i,j-4)`

```
Domain(S0)  = [i,j] : 1≤j≤i≤N
```

```
Domain(S0') = [i,j] : 1≤i≤N &&
                      -3≤j≤i-4
```



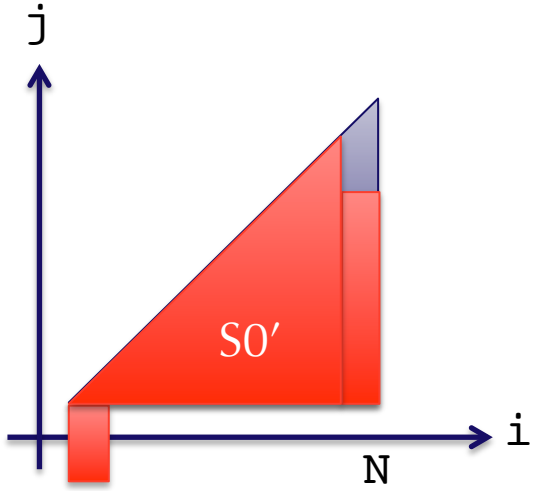
Lex. Shift as Affine Transform

- *Piece-wise* affine transformation
  - `(i,j→i,j-1)` if `j>1` or `i=1`
  - `(i,j→i-1,i-1)` if `j=1` and `i>1`
- Apply `n` times for prefetch distance `n`

`Domain(S0) = [i,j] : 1≤j≤i≤N`

n

`Domain(S0') = [i,j] : <complicated>`



Avoiding Redundant Prefetch: Same Element

- Given:
  - Target array: `A`
  - Set of statement instances that read from `A`
  - Array access functions for each read access
- Find the set of first readers:
  - Statement instances that first read from each element in `A`
  - Find the lex. min among the set of points accessing the same array

Avoiding Redundant Prefetch: Same Cache Line

- Let an element of array **A** be ¼ of a line.
  - The following prefetches the same line **4** times

```
for i …
    prefetch(A[i+4]);
    … = foo(A[i], …);
```

- We apply unrolling to avoid redundancy

```
for i …
    prefetch(A[i+4]);
    … = foo(A[i], …);
    … = foo(A[i+1], …);
    … = foo(A[i+2], …);
    … = foo(A[i+3], …);
```

Outline

- Introduction
- Software Prefetching
- Code Generation
- Simple Example
  - Simulation results
- Summary and Next Steps

Simple Example

- Contrived example that *better* work

```
for(i=0; i<N; i+=1)
   for(j=0; j<M; j+=1)
      sum = sum + A[i][j];
```

- Expectation: when `M` is small and `N` is large, lexicographical shift should work better
- Compare between
  - unroll only
  - Mowery prefetching (shift in innermost)
  - Proposed (lexicographic shift)

Search for Simulators

- Need simulators to experiment with
  - Memory latency
  - Number of outstanding prefetches
  - Line size, and so on
- Tried on many simulators
  - XEEMU: Intel Xscale
  - SoCLib: SoC
  - gem5: Alpha/ARM/SPARC/x86
  - VEX: VLIW

Simulation with VEX (HP Labs)

- Miss penalty: 72 cycles

	cycles	misses	prefetches	effective	speedup
original	658k	2015	-	-	-
unroll	610k	2015	-	-	1.08
mowry	551k	1020	2000	992	1.19
lex. shift	480k	25	2001	1985	1.37

- We see what we expect
  - Effective: number of useful prefetches
- But, for more "complex" examples, the benefit diminishes (<3%)

Lex. Shift was Overkill

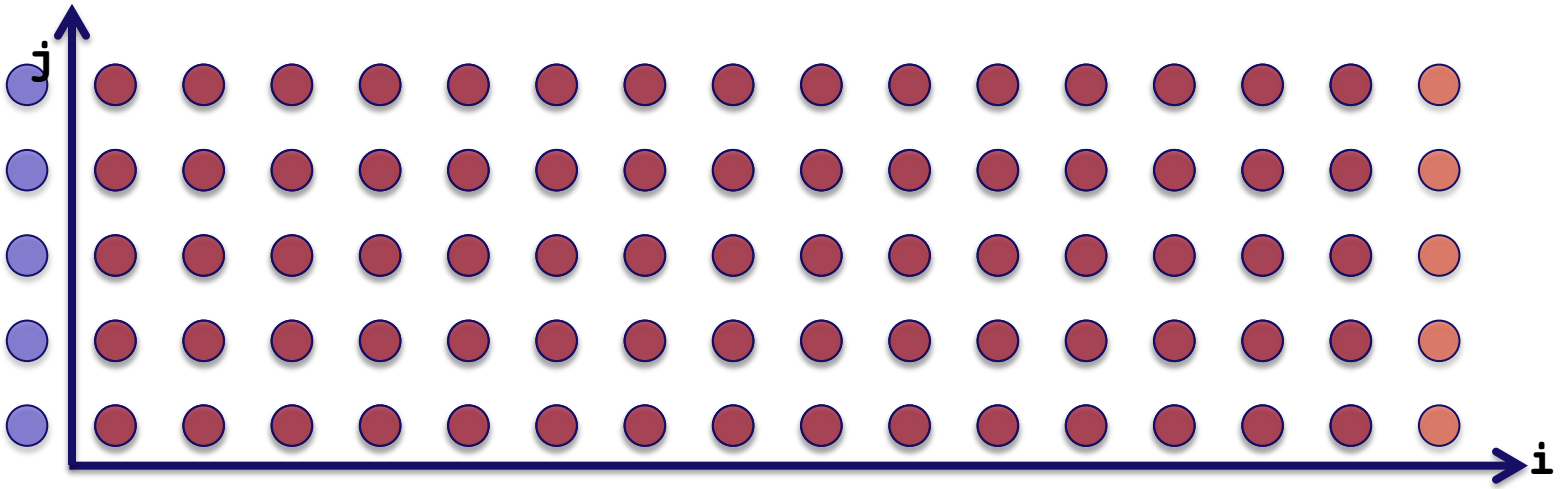
- Precision in placement does not necessary translate to benefit
- Computationally very expensive
  - Power of piecewise affine function by the prefetch distance
    - Takes a lot of memory and time
- High control overhead
  - Code size more than doubles compared to translation in the innermost loop

Summary

- Prefetching doesn't work with short streams
  - Epilogue dominates
  - Can we refine prefetch placement?
- Lexicographic Shift
  - Increase overlap of prefetch and computation
  - Can be done with polyhedral representation
- But, it didn't work
  - Overkill

Another Possibility

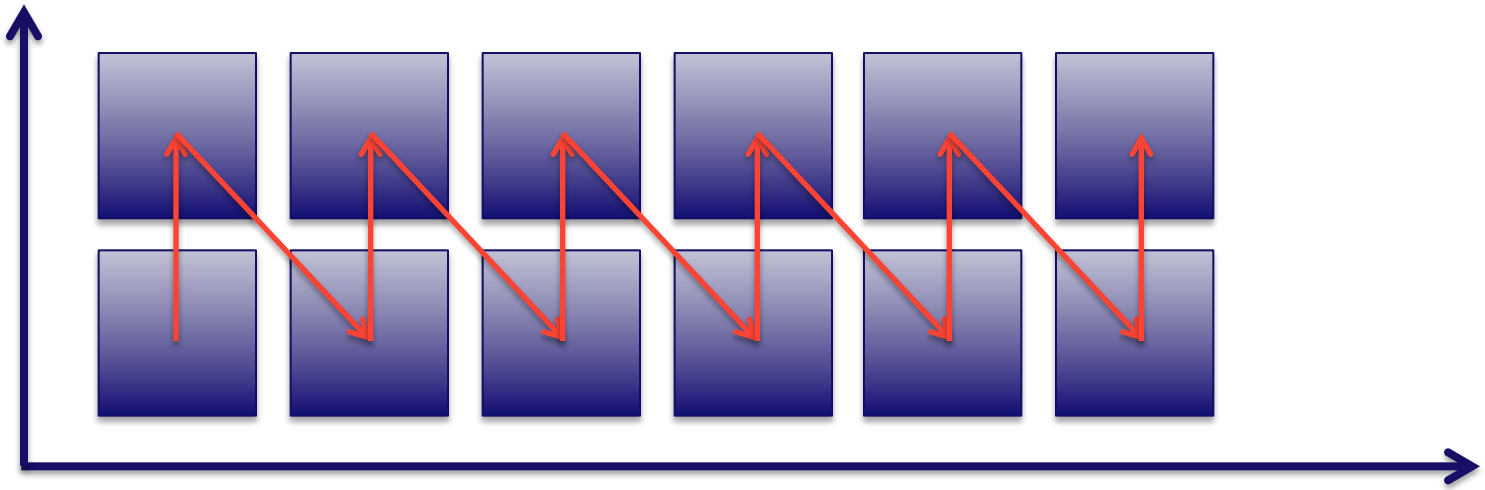
- Translation in outer dimensions



- Keeps things simple by selecting one appropriate dimension to shift
- Works well for rectangular spaces

Coarser Granularity

- Lexicographic shift at statement instance level seems too fine grained
- Can it work with tiles?
  - Prefetch for next tile as you execute one



Questions?

INVENTEURS DU MONDE NUMÉRIQUE