

Tiling: A Data Locality Optimizing Algorithm

Previously

- Kelly & Pugh transformation framework
- Affine space partitions for parallelism

Today

- “Unroll and Jam” and Tiling
- Specifying tiling in the Kelly and Pugh transformation framework
- Status of code generation for tiling

Loop Unrolling

Motivation

- Reduces loop overhead
- Improves effectiveness of other transformations
 - Code scheduling
 - CSE

The Transformation

- Make n copies of the loop: n is the **unrolling factor**
- Adjust loop bounds accordingly

Loop Unrolling (cont)

Example

```
do i=1,n
  A(i) = B(i) + C(i)
enddo
```

➔

```
do i=1,n-1 by 2
  A(i) = B(i) + C(i)
  A(i+1) = B(i+1) + C(i+1)
enddo
if (i=n)
  A(i) = B(i) + C(i)
```

Details

- When is loop unrolling legal?
- Handle end cases with a cloned copy of the loop
 - Enter this special case if the remaining number of iteration is less than the unrolling factor

Loop Balance

Problem

- We'd like to produce loops with the right balance of memory operations and floating point operations
- The ideal balance is machine-dependent
 - e.g. How many load-store units are connected to the L1 cache?
 - e.g. How many functional units are provided?

Example

```
do j = 1,2*n
  do i = 1,m
    A(j) = A(j) + B(i)
  enddo
enddo
```

- The inner loop has 1 memory operation per iteration and 1 floating point operation per iteration
- If our target machine can only support 1 memory operation for every two floating point operations, this loop will be memory bound

What can we do?

Unroll and Jam

Idea

- Restructure loops so that loaded values are used many times per iteration

Unroll and Jam

- Unroll the outer loop some number of times
- Fuse (Jam) the resulting inner loops

Example

```
do j = 1,2*n
  do i = 1,m
    A(j) = A(j) + B(i)
  enddo
enddo
```



Unroll the Outer Loop

```
do j = 1,2*n by 2
  do i = 1,m
    A(j) = A(j) + B(i)
  enddo
  do i = 1,m
    A(j+1) = A(j+1) + B(i)
  enddo
enddo
```

Unroll and Jam Example (cont)

Unroll the Outer Loop

```
do j = 1,2*n by 2
  do i = 1,m
    A(j) = A(j) + B(i)
  enddo
  do i = 1,m
    A(j+1) = A(j+1) + B(i)
  enddo
enddo
```



Jam the inner loops

- The inner loop has 1 load per iteration and 2 floating point operations per iteration
- We reuse the loaded value of **B(i)**
- The Loop Balance matches the machine balance

```
do j = 1,2*n by 2
  do i = 1,m
    A(j) = A(j) + B(i)
    A(j+1) = A(j+1) + B(i)
  enddo
enddo
```

Unroll and Jam (cont)

Legality

- When is Unroll and Jam legal?

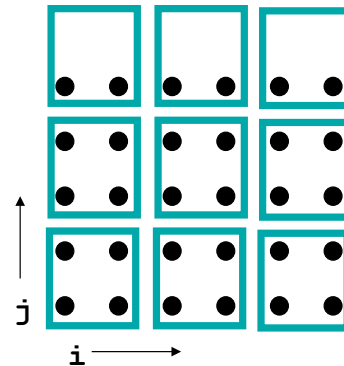
Disadvantages

- What limits the degree of unrolling?

Tiling

A non-unimodular transformation that ...

- groups iteration points into tiles that are executed atomically
- can improve spatial and temporal data locality
- can expose larger granularities of parallelism



Implementing tiling

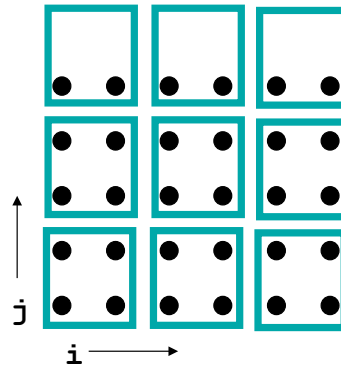
- how can we specify tiling?
- when is tiling legal?
- how do we generate tiled code?

```
do ii = 1, 6, by 2
  do jj = 1, 5, by 2
    do i = ii, ii+2-1
      do j = jj, min(jj+2-1, 5)
        A(i, j) = ...
```

Specifying Tiling

Rectangular tiling

- tile size vector (s_1, s_2, \dots, s_d)
- tile offset, (o_1, o_2, \dots, o_d)



Possible Transformation Mappings

- creating a tile space

$$\{[i, j] \rightarrow [ti, tj, i, j] \mid ti = \text{floor}((i - o_1)/s_1) \wedge tj = \text{floor}((j - o_2)/s_2)\}$$

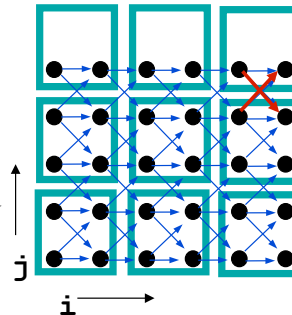
- keeping tile iterators in original iteration space

$$\{[i, j] \rightarrow [ii, jj, i, j] \mid ii = s_1 \text{floor}((i - o_1)/s_1) + o_1 \wedge jj = s_2 \text{floor}((j - o_2)/s_2) + o_2\}$$

Legality of Tiling

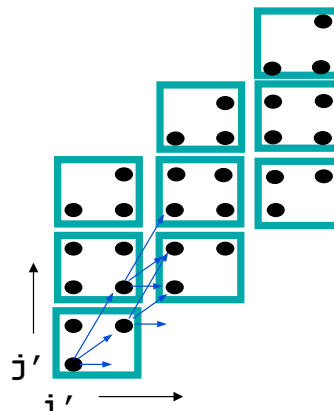
A legal rectangular tiling

- each tile executed atomically
- no dependence cycles between tiles
- Check legality by verifying that transformed data dependences are lexicographically positive



Fully permutable loops

- rectangular tiling is legal on fully permutable loops



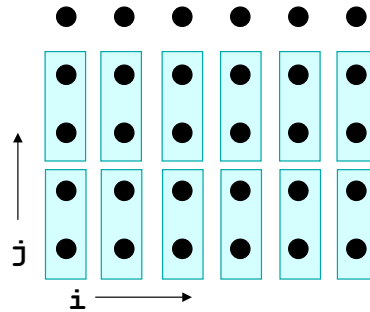
“A Data Locality Optimizing Algorithm” by Michael E. Wolf and Monica S. Lam, 1991.

How can we apply loop interchange, skewing, and reversal to generate

- a loop that is legally tilable (ie. fully permutable)
- a loop that when tiled will result in improved data locality

Original Loop

```
do j = 1, 2*n by 2
  do i = 1, m
    A(j) = A(j) + B(i)
  enddo
enddo
```



Their heuristic for solving data locality optimization problem

Perform reuse analysis to determine innermost tile (ie. localized vector space)

- only consider elementary vectors as reuse vectors

For the localized vector space, break problem into all possible tiling combinations

Apply SRP algorithm in an attempt to make loops fully permutable

- (S)kew transformations, (R)eversal transformation, and (P)ermutation
- Definitely works when dependences are lexicographically positive distance vectors
- $O(n^2 \cdot d)$ where n is the loop nest depth and d is the number of dependence vectors

Code Generation for Tiling

Fixed-size Tiles

- Omega library
- Cloog
- for rectangular space and tiles, straight-forward

```
do ii = 1,6, by 2
do jj = 1, 5, by 2
do i = ii, ii+2-1
do j = jj, min(jj+2-1,5)
A(i,j) = ...
```

Parameterized tile sizes

- Parameterized tiled loops for free, PLDI 2007
- TLOG - A Tiled Loop Generator, <http://www.cs.colostate.edu/~ln/TLOG/>

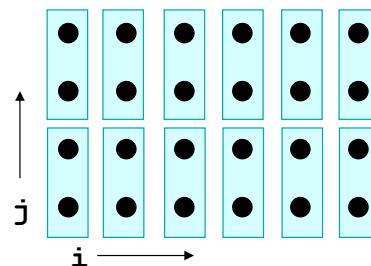
Overview of decoupled approach

- find polyhedron that may contain any loop origins
- generate code that traverses that polyhedron
- post process the code to start a tile origins and step by tile size
- generate loops over points in tile to stay within original iteration space and within tile

Unroll and Jam IS Tiling (followed by inner loop unrolling)

Original Loop

```
do j = 1,2*n
do i = 1,m
A(j) = A(j) + B(i)
enddo
enddo
```



After Tiling

```
do jj = 1,2*n by 2
do i = 1,m
do j = jj, jj+2-1
A(j) = A(j)+B(i)
enddo
enddo
enddo
```



After Unroll and Jam

```
do jj = 1,2*n by 2
do i = 1,m
A(j) = A(j)+B(i)
A(j+1) = A(j+1)+B(i)
enddo
enddo
```

Concepts

Unroll and Jam is the same as Tiling with the inner loop unrolled

Tiling can improve ...

- loop balance
- spatial locality
- data locality
- computation to communication ratio

Implementing tiling

- specification
- checking legality
- code generation

Next Time

Lecture

- Run-time reordering transformations

Suggested Exercises

- after array expansion of the scalar T , is it legal to tile the three loops in Figure 11.23? write the tiled code for a block size of your choice.