Tiling: A Data Locality Optimizing Algorithm

Previously

- Unroll and Jam

Homework

- PA3 is due Monday November 2nd

Today

- Unroll and Jam is tiling
- Code generation for fixed-sized tiles
- Paper writing and critique with Lam and Wolfe, “A Data Locality Optimizing Algorithm” as an example

Unroll and Jam

Idea

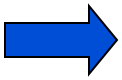
- Restructure loops so that loaded values are used many times per iteration

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
```

- The inner loop has 1 load per iteration and 2 floating point operations per iteration
- We reuse the loaded value of **B(i)**
- Aim to match the machine balance, how many floating point ops per load?

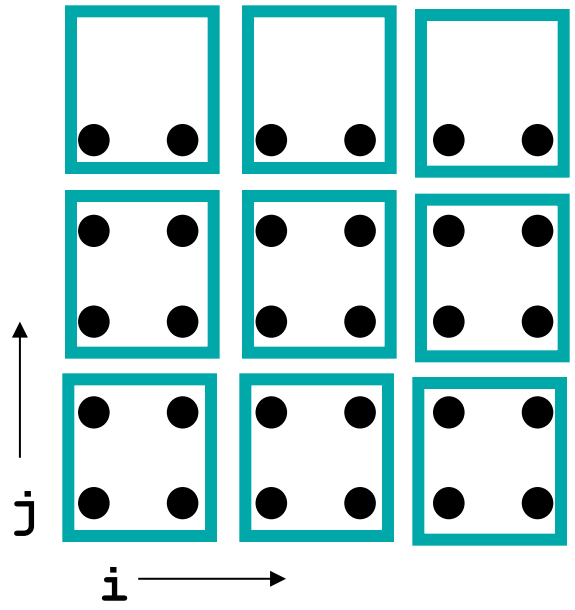
Jam the inner loops

```
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
```

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

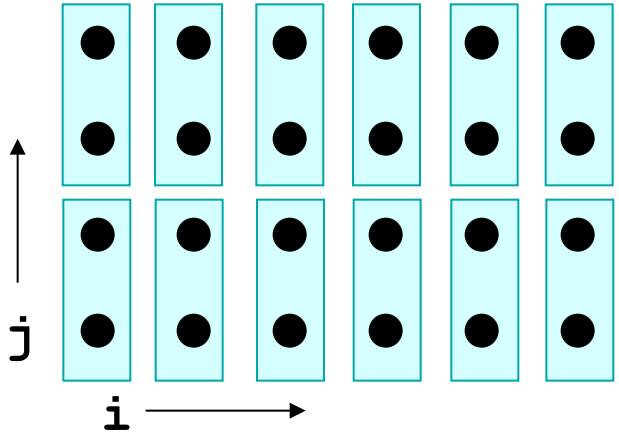
- how can we specify tiling?
- when is tiling legal?
- how do we generate tiled code?

```
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) = ...
```

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

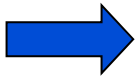
Original Loop

```
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
```

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
- HiTLOG - A Tiled Loop Generator that is part of AlphaZ

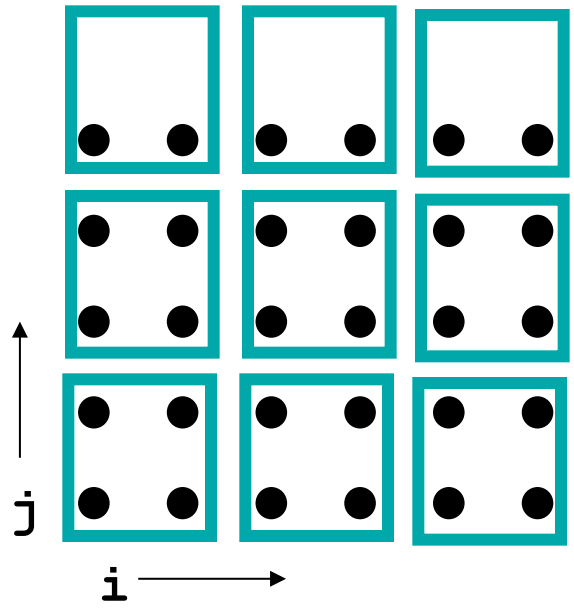
Overview of decoupled approach

- 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

Specifying Tiling

Rectangular tiling

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



Possible Transformation Mappings

- creating a tile space

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

- keeping tile iterators in original iteration space

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

Using iscc to do code generation for tiling

Iteration space: `S := { s[i,j] : 1 <=i<=6 && 1<=j<=5 };`

Tiling specification

```
T :={s[i,j]->[ti,tj,i,j]: ti=(i-1)/2&&tj=(j-1)/2};
```

Gives output but not correct.

Getting rid of integer divison

```
ti=(i-1)/2 becomes
  0<=ri<2 && (i-1)=2*ti+ri
tj=(j-1)/2 becomes
  0<=rj<2 && (j-1)=2*tj+rj
T :={s[i,j]->[ti,tj,i,j]: exists ri,rj:
  0<=ri<2 && i-1=ti*2+ri
  && 0<=rj<2 && j-1=tj*2+ rj};
```

Let's try other specification that doesn't create new tile space.

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

Paper Writing and Critique in General

Papers should answer the following questions

- What problem did the paper address?
- Is the problem important/interesting?
- What is the approach used to solve the problem?
- How does the paper support or justify the conclusions it reaches?
- What problems are explicitly or implicitly left as future research questions?

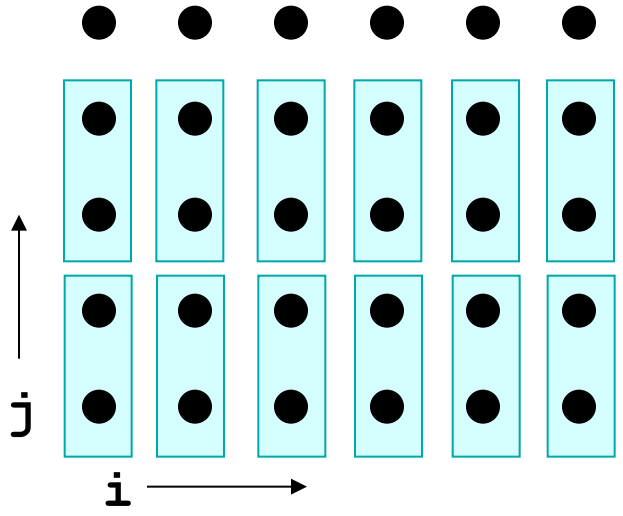
"A Data Locality Optimizing Algorithm" by Michael E. Wolf and Monica S. Lam, 1991.

Specific Problem: 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
```



Is the problem important/interesting?

Performance improvements due to tiling can be significant

- For matrix multiply, 2.75 speedup on a single processor
- Enables better scaling on parallel processors

Tiling Loops More Complex than MM

- requires making loops permutable
- goal is to make loops exhibiting reuse permutable

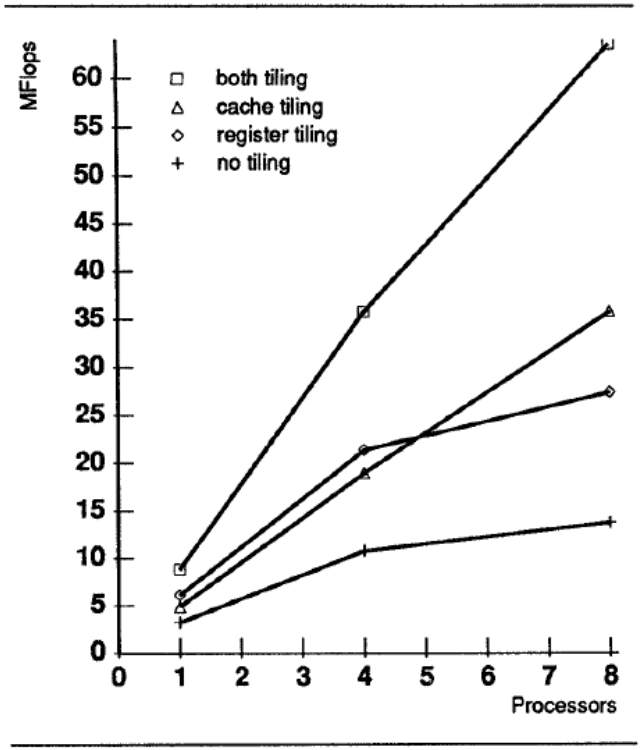


Figure 1: Performance of 500 × 500 double precision matrix multiplication on the SGI 4D/380. Cache tiles are 64 × 64 iterations and register tiles are 4 × 2.

What is the approach used to solve the problem?

Create a unimodular transformation that results in loops experiencing reuse becoming fully permutable and therefore tilable

Formulation of the data locality optimization problem (the specific problem their approach solves)

- For a given iteration space with
  - a set of dependence vectors, and
  - uniformly generate reference sets

  the data locality optimization problem is to find the unimodular and/or tiling transform, subject to data dependences, that minimizes the number of memory accesses per iteration.

The problem is hard

- Just finding a legal unimodular transformation is exponential in the number of loops.

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
- $O(n^2*d)$ where n is the loop nest depth and d is the number of dependence vectors

How does the paper support the conclusion it reaches?

"The algorithm ... is successful in optimizing codes such as matrix multiplication, successive over-relaxation (SOR), LU decomposition without pivoting, and Givens QR factorization".

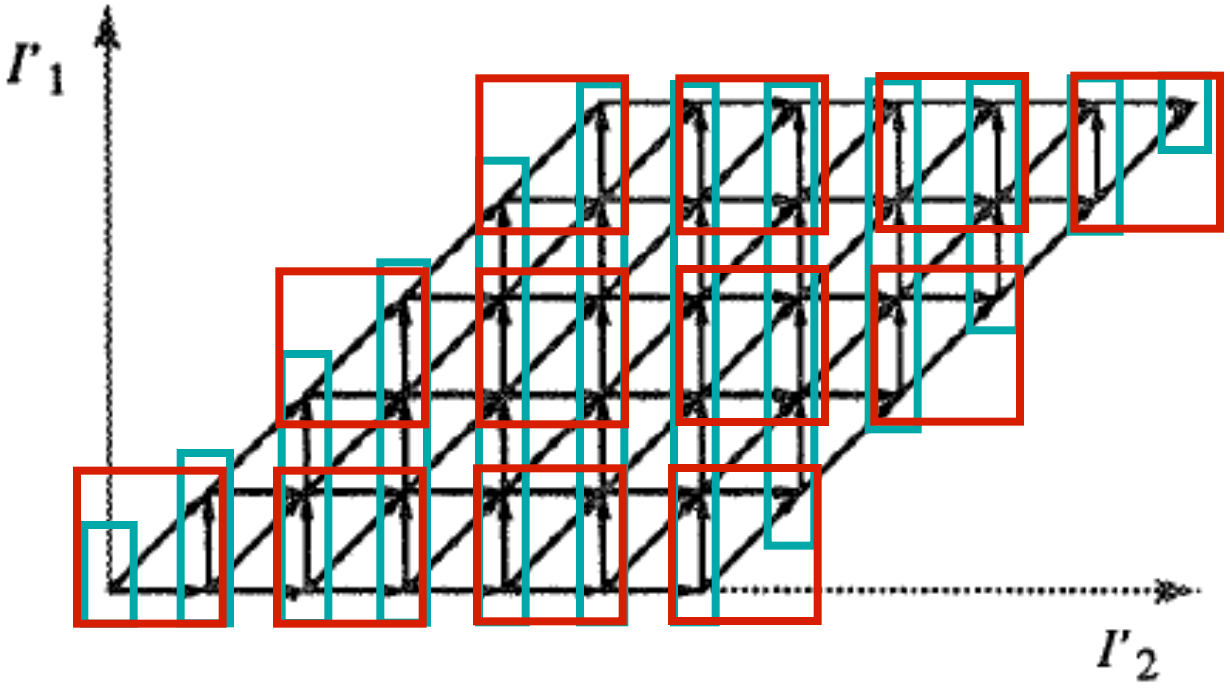
- They implement their algorithm in the SUIF compiler
- They have the compiler generate serial and parallel code for the SGI 4D/380
- They perform some optimization by hand
  - register allocation of array elements
  - loop invariant code motion
  - unrolling the innermost loop
- Benchmarks and parameters
  - LU kernel on 1, 4, and 8 processors using a matrix of size 500x500 and tile sizes of 32x32 and 64x64
  - SOR kernel on 500x500 matrix, 30 time steps

SOR Transformations

Variations of 2D (data) SOR

- wavefront version, theoretically great parallelism, but bad locality
- "2D tile", better than wavefront, doesn't exploit temporal reuse
- "3D tile" version, best performance

Picture for 1D (data) SOR



What problems are left as future research?

Explicitly stated future work

- The authors suggest that their SRP algorithm may have its performance improved with a branch and bound formulation.

Questions left unanswered in the paper

- How should the tile sizes be selected?
- After performing tiling, what algorithm should be used to determine further transformations for improved performance?
  - They perform inner loop unrolling and other, but do not perform a model for which transformations should be performed and what their parameters should be.
- What is the relationship between storage reuse, data locality, and parallelism?