Tiling: A Data Locality Optimizing Algorithm

Announcements
- Monday November 28th, Dr. Sanjay Rajopadhye is talking at BMAC
- Friday December 2nd, Dr. Sanjay Rajopadhye will be leading CS553

Last Monday
- Kelly & Pugh transformation framework
- Loop fusion and fission
- Brief intro to scheduling Alpha programs

Today
- “Unroll and Jam” and Tiling
- Review of the paper “A Data Locality Optimizing Algorithm” by Michael E. Wolf and Monica S. Lam

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

\[
\begin{align*}
\text{do } & \text{i=1,n} & \text{do } & \text{i=1,n by 2} \\
A(i) &= B(i) + C(i) & A(i) &= B(i) + C(i) \\
\text{endo} & & A(i+1) &= B(i+1) + C(i+1) \\
& & \text{endo}
\end{align*}
\]

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

\[
\begin{align*}
\text{do } & \text{j = 1,2*n} & \text{do } & \text{i = 1,m} \\
& & A(j) &= A(j) + B(i) \\
\text{endo} & & \text{endo}
\end{align*}
\]

- 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**

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

**Unroll the Outer Loop**

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

**Unroll and Jam Example (cont)**

Unroll the Outer Loop

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

Jam the inner loops

```
Jam the inner loops
    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)
- The Loop Balance matches the machine balance
Unroll and Jam (cont)

Legality
– When is Unroll and Jam legal?

Disadvantages
– What limits the degree of unrolling?

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

Original Loop
\[
\begin{align*}
do & \ j = 1,2^*n \text{ by } 2 \\
do & \ i = 1,m \\
A(j) & = A(j) + B(i) \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

After Tiling
\[
\begin{align*}
do & \ jj = 1,2^*n \text{ by } 2 \\
do & \ i = 1,m \\
do & \ j = jj, jj+2-1 \\
A(j) & = A(j)+B(i) \\
\text{enddo} \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

After Unroll and Jam
\[
\begin{align*}
do & \ jj = 1,2^*n \text{ by } 2 \\
do & \ i = 1,m \\
A(j) & = A(j)+B(i) \\
A(j+1) & = A(j+1)+B(i) \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]
**Paper Critique and Presentation Format**

Critique: 1-2 pages with a paragraph answering each of 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?

Presentation: 10-15 slides that present the answers to the critique questions (example for the paper “A Data Locality Optimizing Algorithm” by Michael E. Wolf and Monica S. Lam follows)

---

**What is the problem the paper addresses?**

How can we apply loop interchange, skewing, and reversal to generate
- a loop that is legally tiling (i.e., fully permutable)
- a loop that when tiled will result in improved data locality

**Original Loop**

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

---

**Tiling Diagram**
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

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.
**Terminology**

Dependence vector - a generalization of distance and direction vectors

Reuse versus Locality

Localized vector space

Uniformly generated reference sets

---

**Heuristic for solving data locality optimization problem**

Perform reuse analysis to determine innermost tile (i.e., 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)everal transformation, and (P)ermutation
- Definitely works when dependences are lexicographically positive distance vectors
- $O(n^2d)$ 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 tiling, 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?

Concepts

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

Tiling can improve ...
- loop balance
- spatial locality
- data locality
Next Time (November 28th)

Student Surveys

Lecture
  – Compiling object-oriented languages