Tiling: A Data Locality Optimizing Algorithm

Previously
- Data dependence analysis for detecting parallelism
- Specifying transformations using frameworks

Homework
- PA3 is due Monday November 2nd

Today
- Iscc
  - Scattering function for two statements in one loop
  - Applying transformations to data dependencies
- Usefulness of the polyhedral model
- “Unroll and Jam” and Tiling
- Specifying tiling in the Kelly and Pugh transformation framework
- Status of code generation for tiling
Usefulness of the Polyhedral Model

Loop transformations and parallelization can have a significant impact on performance

– Other than new algorithms, data locality and parallelization is necessary for future improvements
– Loop permutation in smithWaterman.c resulted in an order of magnitude execution time difference

Execution time for varying problem sizes
Usefulness of the Polyhedral Model, cont…

Existing tools in industry that use the polyhedral model
- IBM XL/Poly
- Reservoir Labs R-Stream
- Apolent

Polyhedral model in open source and research
- Graphite GCC, au auto parallelization pass
- Pluto and Orio
- PoCC (Clan-Candl-LetSee-Pluto-Cloog-Polylib-PIPLib-ISL-FM)
- AlphaZ, POET, Chill, URUK, Omega, Loopo, iscc

Programming language abstractions that are related
- Domains in Chapel
- Regions in X10
- …
Loop Unrolling leading to tiling

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

The Transformation
- Make \( n \) copies of the loop: \( n \) is the \textit{unrolling factor}
- Adjust loop bounds accordingly
Loop Unrolling (cont)

Example

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

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

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

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

do i = 1, m
    A(j+1) = A(j+1) + B(i)
enddo
```
Unroll and Jam Example (cont)

Unroll the Outer Loop

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

- 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

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

\[
\begin{align*}
doi &= 1, 6, \text{ by 2} \\
doj &= 1, 5, \text{ by 2} \\
deoi &= ii, ii+2-1 \\
deoj &= jj, \min(jj+2-1, 5) \\
A(i,j) &= \ldots
\end{align*}
\]
Specifying Tiling

Rectangular tiling
- tile size vector \((s_1, s_2, ..., s_d)\)
- tile offset, \((o_1, o_2, ..., o_d)\)

Possible Transformation Mappings
- creating a tile space
\[
\{[i, j] \rightarrow [ti, tj, i, j] \mid ti = \text{floor}((i - o_1)/s_1) \land tj = \text{floor}((j - o_2)/s_2)\}
\]

- keeping tile iterators in original iteration space
\[
\{[i, j] \rightarrow [ii, jj, i, j] \mid ii = s_1 \text{floor}((i - o_1)/s_1) + o_1 \land jj = s_2 \text{floor}((j - o_2)/s_2) + o_2\}
\]
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
Code Generation for Tiling

**Fixed-size Tiles**
- Omega library
- Cloog
- for rectangular space and tiles, straight-forward

**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

```plaintext
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) = ...
  end do
end do
```
Unroll and Jam IS Tiling (followed by inner loop unrolling)

Original Loop
\[
do\ j = 1, 2n \\
\quad do\ i = 1, m \\
\quad\quad A(j) = A(j) + B(i) \\
\quad enddo \\
enddo
\]

After Tiling
\[
do\ jj = 1, 2n by 2 \\
\quad do\ i = 1, m \\
\quad\quad do\ j = jj, jj+2-1 \\
\quad\quad\quad A(j) = A(j) + B(i) \\
\quad\quad enddo \\
\quad enddo \\
enddo
\]

After Unroll and Jam
\[
do\ jj = 1, 2n by 2 \\
\quad do\ i = 1, m \\
\quad\quad do\ j = jj, jj+2-1 \\
\quad\quad\quad A(j) = A(j) + B(i) \\
\quad\quad enddo \\
\quad enddo \\
enddo
\]
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

– More Tiling