Tiling: A Data Locality Optimizing Algorithm

Previously
- Performance analysis of existing codes
- Data dependence analysis for detecting parallelism
- Specifying transformations using frameworks

Today
- Usefulness of the polyhedral model
- Using Pluto to automatically transform code
- “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
- Recall that loop permutation in smithWaterman.c resulted in an order of magnitude execution time difference

![Graph](image_url)
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

Programming language abstractions that are related
– Domains in Chapel
– Regions in X10
– …

Getting Started with Pluto

Pluto has been installed on the CS linux machines

Try out some examples
– git clone git://repo.or.cz/pluto.git will create a directory called pluto
– Test and examples directory contains example input files
– polycc test/seidel.c --tile --parallel

Read some documentation
– more pluto/doc/DOC.txt
– http://pluto-compiler.sourceforge.net/

Using it on your own code
– Put in #pragma scop and #pragma endscop around the code to be transformed
– <show using polycc with the smithWaterman.c example>
Static Control Programs

Static Control Programs Definition
- Symbolic constants are variables that are not modified in the loop
- Loop bounds are affine combinations of loop variables and symbolic constants
- Array accesses are affine combinations of loop variables and symbolic constants
- If conditions are affine functions of loop variables and symbolic constants
- All function calls are pure

References
- Pluto_general_notes.txt (will post with this lecture)
- Pluto/doc/DOC.txt
- Section 2.2 of the Feautrier paper

Control over Pluto transformations (see doc/DOC.txt)

polycc infile.c
--parallel, Parallelize code with OpenMP
--tile, Tile code. Can control tile sizes with an input file in a specific format.
--maxfuse, Do the maximum amount of loop fusion
--unroll, does unroll and jam up to two loops
--prevector, make code amenable to vectorization
--debug, Intermediate files are kept so can see dependences, etc.

What is tiling?
- “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
\[
\begin{align*}
\text{do } i=1, n & \\
A(i) &= B(i) + C(i) \\
\text{enddo} \\
\text{do } i=1, n-1 \text{ by 2} & \\
A(i) &= B(i) + C(i) \\
A(i+1) &= B(i+1) + C(i+1) \\
\text{enddo} \\
\text{if } (i=n) & \\
A(i) &= B(i) + C(i)
\end{align*}
\]

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**

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

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**

```plaintext
Unroll the Outer Loop
```

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

```plaintext
Unroll the Outer Loop
```

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

```plaintext
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

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

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.

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

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?

```plaintext
doi = 1, 6, by 2

doj = 1, 5, by 2

doi = ii, ii+2-1

doj = jj, min(jj+2-1, 5)

A(i, j) = ...
```

---

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


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

**Original Loop**

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

**After Tiling**

```latex
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**

```latex
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**
- Review of transformations covered so far
- Run-time reordering transformations

**Schedule**
- Project intermediate report due March 28th
- April 3rd will be a lab day during class, Manaf will help people with AlphaZ and Pluto. Distance students can email questions or use the discussion board.
- HW6 and HW7 will BOTH be due April 4th