Parameterized Tiling

Previously

- Code generation with parameterized tile sizes

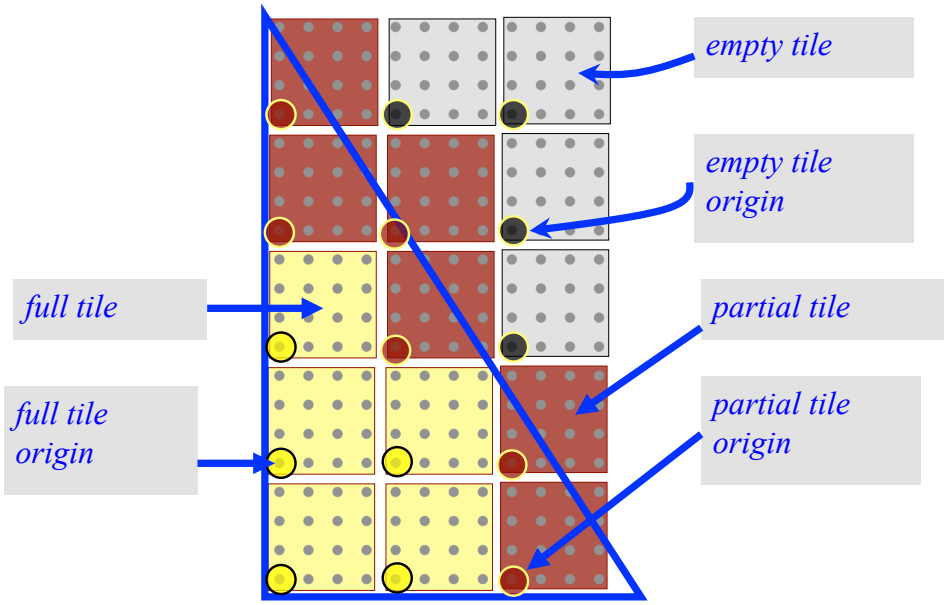
Today

- Finish code generation with parameterized tile sizes
- FM and Farkas questions
- Final report and poster expectations

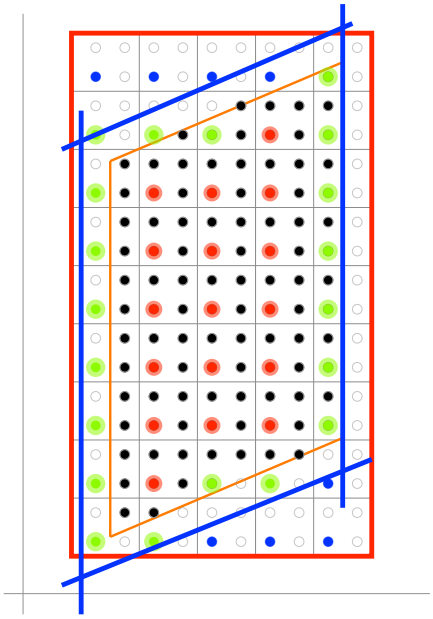
Logistics

- HW10 due Friday
- Reading assignment for next week is posted
- Final report for project due next Friday, 5/4/12
- Quiz 4 due next Monday, 5/7/12
- Poster session Thursday May 10 from 2-4pm

Tile Origins



Outset



Set of non-empty tile origins (red & green dots)

Bounding box contains lots of empty tile (blue) origins

Outset is a tight approximation of the set of non-empty tile origins

Outset Formal Definition

Original Iteration Space

$$P_{iter} = \{\vec{x} \mid Q\vec{x} \geq (\vec{q} + B\vec{p})\}$$

Outset

$$P_{out} = \{\vec{x} \mid Q\vec{x} \geq (\vec{q} + B\vec{p}) - \boxed{Q^+}\vec{s'}\}$$

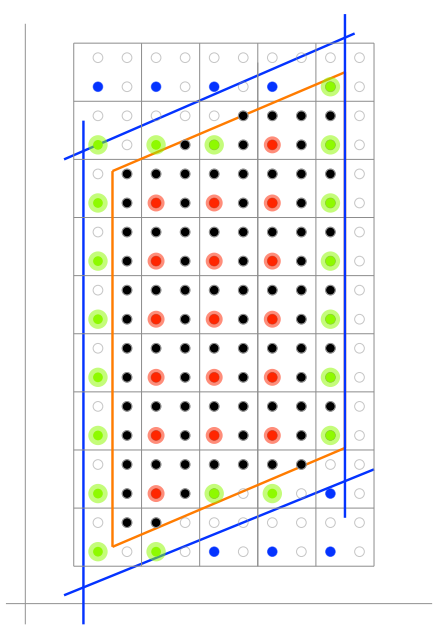
$$Q^+_{ij} = \begin{cases} Q_{ij}, & \text{if } Q_{ij} \geq 0 \\ 0, & \text{if } Q_{ij} < 0 \end{cases}$$

Outset is a parameterized polyhedra

Note that the tile sizes are parameters

Can be constructed in linear time/space

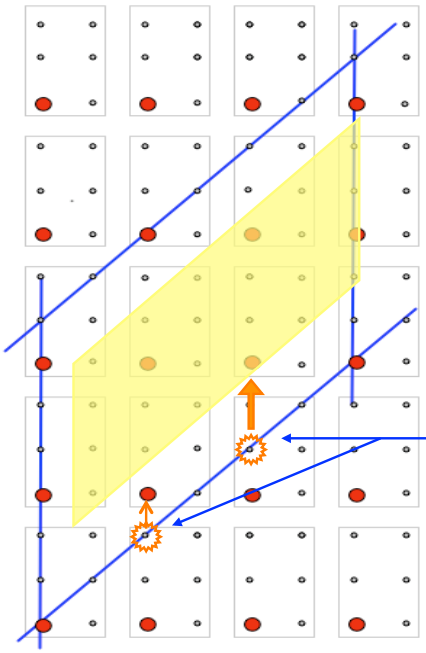
Outset Properties



Outset properties

- *Tight approximation of the set of non-empty tile origins*
- *Linear time/space construction*
- *Tile sizes are parameters*
- *Parameterized polyhedron*
  - *enables use of standard loop generation tools*

Generating tile-loops with Outset



Algorithm

Construct outset

Generate loops that scan all points in the outset

Skip non-tile origins

- Shift lower bounds
- Stride iterations

Outset based Tiled Loop Generation Method

tile-loops

```
SHIFT_LB(t1)
for t1 = …; …; t1 += S1
  …  …
  SHIFT_LB(tn)
   for tn = …;…;tn += Sn
```

```
for i1 = … ; …
        … …
        for in = … ; …

              S()
```

point-loops

Method outline

- *Gen. tile-loops using outset*
  - *using **CLOOG***
- *Post-process to add*
  - *lower bound shifts*
  - *strides (step sizes)*
- *Generate point-loops*
  - *our generator + **CLOOG***
- *Insert point-loops inside tile-loops.*

The TLoG Tool

Tiled Loop Generator

Implements parameterized tiling

- Outset based (parameterized / fixed /mixed)
- Classic method
- Inset based separation of full and partial tiles
- Internally uses CLOOG (can also use Omega)

Used (beta tested ☺) in graduate level course at Colorado State University

Evaluation

Goal is to compare

- generation efficiency
- quality of generated code

Benchmarks

- stencils, LUD, SSYRK, STRMM

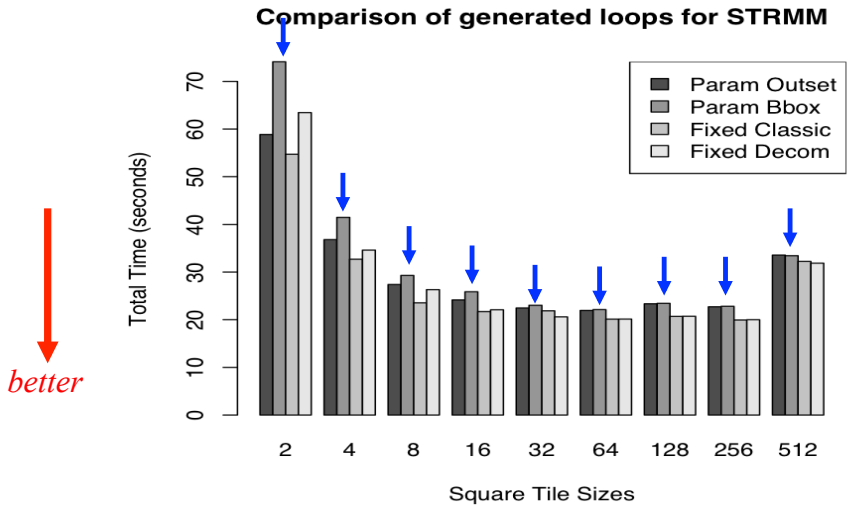
Methods compared

- Outset and Bounding box for parameterized
- Outset and classic for fixed tile sizes

Cost of Parameterization (1)

Experiments run on Intel coreDuo @ 1.86 GHz with 2 MB L2 cache

Benchmark: Triangular matrix product from BLAS 3 - tetrahedral iteration space



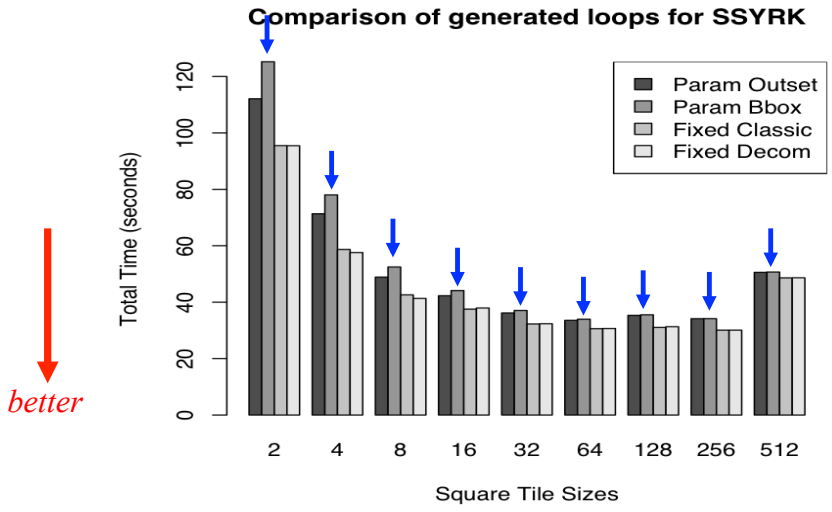
Bbox inefficient on small tile sizes

Parameterization overhead is insignificant

Cost of Parameterization (2)

Experiments run on Intel coreDuo @ 1.86 GHz with 2 MB L2 cache

Benchmark: Symmetric rank k update from BLAS 3 - tetrahedral iteration space



Bbox inefficient on small tile sizes

Parameterization overhead is insignificant

Code Generation Time

Tiled loop generation times in milliseconds

	LUD	SSYRK	STRMM	3D Stencil
Fixed Classic	32.4	28.6	29.0	26.0
Fixed Decom	55.2	51.0	50.4	45.0
Param Outset	52.0	53.8	52.1	54.1
Param Bbox*	53.5	53.2	51.2	54.0

Fixed classic method does not scale

Parameterized tiled code can be generated as fast as fixed tiled code

**Bounding box computation time not included*

Summary of Results

Two fundamental polyhedra: Outset and Inset

– Useful for generating a variety of tiled loops

Parameterization for free

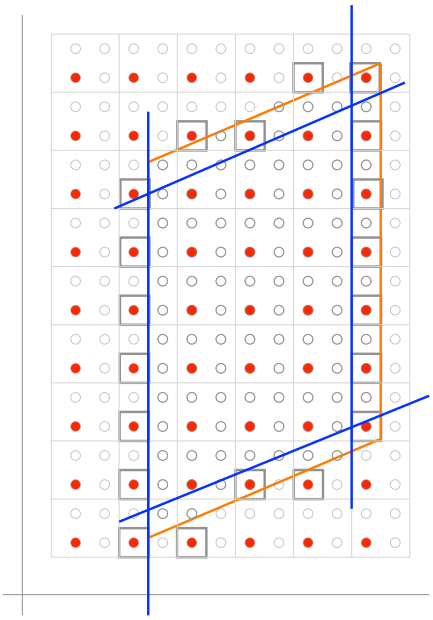
Single unified technique

– fixed, parameterized or mixed tiled loop generation

Ideal for compilers

Open source tool available

Inset



Inset properties

- *(Exact) Set of full-tile origins*
- *Linear time/space construction*
- *Parameterized tile sizes*
- *Higher dimensional polyhedron*
  - *standard tools can be used to scan*

Separating Partial and Full Tiles

Full tiles have simpler loop bounds

– increase applicability of optimizations

tile-loops:
(enumerate tile origins)

if (tile origin is in inset)

point-loops: tile-box-bounds

else

point-loops: tile-box and IS bounds

Fourier-Motzkin and Farkas Questions (HW10)

1D scheduling

- Specify the data dependence relation with all inequality constraints. The result is a data dependence polyhedron.
- Want the function $\theta_S(\vec{j}) - \theta_R(\vec{i}) - 1$ to be non-negative over the dependence polyhedron
- Use Farkas lemma to set up a new set of constraints

$$\theta_R(\vec{i}) = \vec{v}^T\vec{i} + \vec{b} \qquad \theta_S(\vec{j}) - \theta_R(\vec{i}) - 1 = \lambda_0 + \vec{\lambda}^T(A \begin{bmatrix} \vec{i} \\ \vec{j} \\ \vec{p} \\ 1 \end{bmatrix})$$

$$\theta_S(\vec{j}) = \vec{w}^T\vec{j} + \vec{c} \qquad \lambda_0 \geq 0 \text{ and } \vec{\lambda} \geq \vec{0}$$

- Use Fourier-Motzkin to project out all variables except for v, w, b, and c

Code Generation with Fourier-Motzkin

- Write down bounds for innermost loop, ALL bounds, use max and min to aggregate
- Project out the innermost loop iterator with FM
- Loop this process until the loop iterates over all bounds

Final Report

Logistics

- 15% of total grade
- Form of a 6-10 page conference paper

Content

- describe and motivate the problem (why is the application important, what are the performance problems in the application)
- present the approach (your transformation and automation strategy)
- evaluate the approach (evaluate the performance of the transformation strategy and the usability and performance of the automation tool in relation to doing the transformation by hand and also in comparison to the other tools covered in class)

Help

- See “Writing a Paper” links on Resource page
- Have a fellow student read through your report and provide feedback
- Review feedback from the project proposals and intermediate report

Poster

Logistics

- Thursday May 10, 2-4pm, in 325 (same room)
- 5% of total grade
- 36in width by 24in height (custom slide size in page setup of powerpoint)
- Printing the poster
  - Go to Kim Judith on 2nd floor to get an account number
  - Go to library (http://lib.colostate.edu/services/computers/plotting)
  - Print the day BEFORE the poster session
- Distance students
  - Do not need to print the poster
  - Email pdf or ppt of poster to cs560@cs.colostate.edu
  - Set up a skype session with Michelle BEFORE the poster session on campus. Send email to suggest times or call 970-491-4193.

Poster Content

Graded On

- How clear the message is presented
- Illustrations that support the message
- Succinct and "on message" text
- Organization and flow

Help

- Michelle will provide feedback on drafts sent by Tuesday May 8th, 8pm
- See the "Developing a Poster" links on the resource page

Concepts

Code generation for fixed tiling

- Using omega to generate tiled code
- Tile loops
- Point loops
- Tile origin

Code generation for parameterized tiling

- outset
- inset
- Generating code using the outset

Course Outline I

Review

- Timing benchmarks and graphing results
- Parallelization with OpenMP
- Big-O, functions, set theoretic notation
- Linear algebra concepts such as matrix multiply

Working with Applications

- Touchstone problems for this class
- Categories of applications based on The Berkeley View

Computer Architectures and Parallelism

- Multiple levels of parallelism and memory
- What is the best performance possible for a particular application?

Performance Analysis, Metrics, and Models (Roofline model)

Loop transformations and parallelization

- Applying them by hand
- When are they useful

Course Outline II

Frameworks for Automating Loop Transformations

- Frameworks: unimodular, polyhedral, presburger, and sparse polyhedral
- Represent computations, represent loop transformations, test the legality of transformations, and compose transformations
- Generate efficient code

Tiling: Aggregate computation to reduce overhead

Run-time Reordering Transformations: Inspector/Executor strategies

Full Automation

- Set and relation manipulation (composition and apply/image)
- Scheduling and storage mapping algorithms (1D scheduling with Farkas)
- Code generation algorithms (Fourier Motzkin and outset)

Current State of the Tools

- Tools: omega, AlphaZ, and Pluto
- What are the implementation gaps?
- What are the limitations and future research questions?

Concepts

Parallelism is critical to improving performance in current and future machines.

Compilation

- Stages of compilation
- Automating parallelization and performance optimization in compiler is the ideal.

Example Program Optimizations

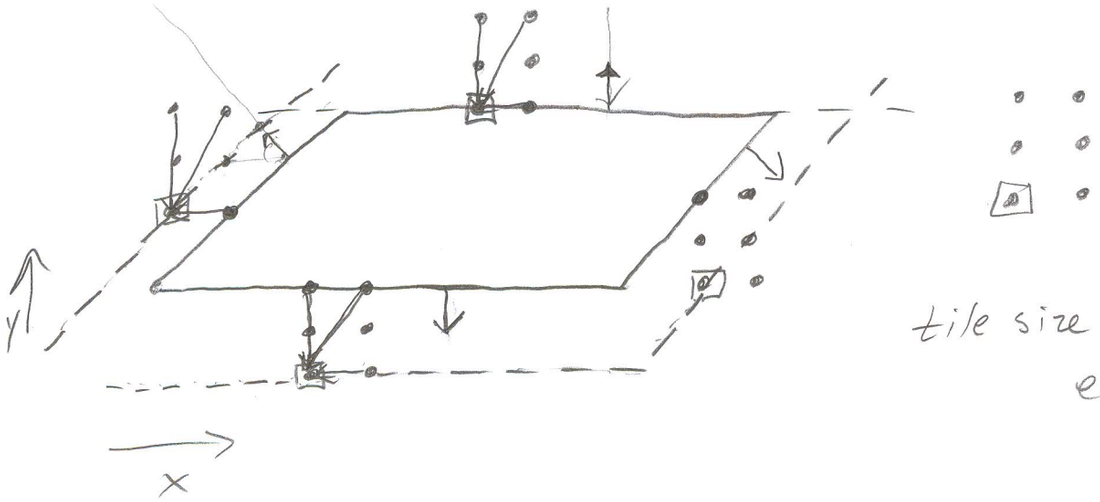
- Loop-invariant code motion
- Induction variable elimination
- Loop unrolling, permutation, and parallelization

Approach for Automating Program Parallelization and Optimization

- Represent computations with iteration spaces
- Schedule the iteration space

main idea

- shift all lower bounds along their normal
- normal that points out of iteration space
- shift by largest negative projection of tile vectors on normal



tile size $\vec{s} = (s_1, s_2)$

ex. $(\underline{2}, \underline{3})$

tile vectors

ex. $(0,0)$, $(-1,-2)$, $(0,-2)$ $(-1,0)$

gen 2D $(0,0)$, $(-(s_1-1), -(s_2-1))$, $(0, -(s_2-1))$, $(-(s_1-1), 0)$

diag

Recall vec A project on B

$$C = |A| \cos\theta \left(B / |B| \right)$$

use identities

$$C = \frac{(A \cdot B)}{B \cdot B} B$$

$$r_x x + r_y y \geq b + r_x \underline{x_1} + r_y \underline{y_1}$$

$$(b + B\vec{p}) - Q^{+} \vec{s}^{-1}$$

$$(x_1, y_1) = \frac{(tv) \cdot (r_x, r_y)}{(r_x, r_y) \cdot (r_x, r_y)} (r_x, r_y)$$

tv	(x_1, y_1)	$r_x x_1 + r_y y_1$
$(0,0)$		0
$(0, -(s_y - 1))$		$-(s_y - 1) r_y$
$(-(s_x - 1), -(s_y - 1))$		$-(s_x - 1) r_x - (s_y - 1) r_y$
		$-(s_x - 1) r_x$