Parametric Multi-Level Tiling of Imperfectly Nested Loops*

Albert Hartono\(^1\)
Cédric Bastoul\(^2,3\)
Sriram Krishnamoorthy\(^4\)
J. Ramanujam\(^6\)
Muthu Baskaran\(^1\)
Albert Cohen\(^2\)
Boyana Norris\(^5\)
P. Sadayappan\(^1\)

\(^1\)Ohio State University
\(^2\)INRIA Saclay
\(^3\)Paris-Sud 11 University
\(^4\)Pacific Northwest National Laboratory
\(^5\)Argonne National Laboratory
\(^6\)Louisiana State University

*Funded by NSF
Imperfectly nested loops are common in practice. Parametric tiled loop generator can provide valuable compiler support for auto-tuning. Current general solutions for tiled code generation:
- Parametric tiling of perfect loop nests
- Non-parametric tiling of imperfect loop nests
Both use polyhedral model and ILP machinery:
- *(Constraint)* Inequalities of the loop bounds must be linear in terms of loop iterators and problem sizes => problem with parametric tile sizes
We have recently developed a “hybrid” solution for parametric tiling of imperfect loop nests.
**Loop Tiling**

- Key loop transformation for both:
  - Efficient coarse-grained parallel execution
  - Data locality optimization

```
for (i=1; i<=7; i++)
  for (j=1; j<=6; j++)
    S(i,j);
```

- **Inter-tile loops**
  - for (it=1; it<=7; it+=Ti)
    - for (jt=1; jt<=6; jt+=Tj)
      - for (i=it; i<min(7,it+Ti-1); i++)
        - for (j=jt; j<min(6,jt+Tj-1); j++)
          - S(i,j);

- **Intra-tile loops**
  - for (it=1; it<=7; it+=Ti)
    - for (jt=1; jt<=6; jt+=Tj)
      - for (i=it; i<min(7,it+Ti-1); i++)
        - for (j=jt; j<min(6,jt+Tj-1); j++)
          - S(i,j);
for (i=1; i<n; i++)
for (j=2; j<n; j++)
    S1: a[i][j] = a[j][i] + a[i][j-1];

Stmt instances $\Leftrightarrow$ integer points in polyhedra $\Leftrightarrow$ systems of linear inequalities
for (i=0; i<N; i++)
{
for (j=0; j<N; j++)
for(k=0; k<N; k++) S1;
for (p=0; p<M; p++)S2;
}
Polyhedral Compiler Transformation

Input Program

- Loops -> Polyhedra
- Data Dependence Analysis

Transforms (Affine Functions)

Output Program

- Code Generation:
  - Polyhedra -> Loops

Loops -> Polyhedra

Data Dependence Analysis

Transforms (Affine Functions)

Code Generation:
  - Polyhedra -> Loops
Parametric Tiled Code Generation

- Tiled code generation is straightforward for rectangular, perfectly nested loops
- But tiled code generation is more challenging if
  - Inner loop bounds depend on outer loops
  - Data dependences make rectangular tiling illegal
  - Loops are imperfectly nested
- Polyhedral compilation model enables tiled code generation for arbitrary affine codes with imperfectly nested loops
“Code generation” in polyhedral compiler framework:
- The process of converting a polyhedral representation of computations back into loop structures

CLooG
- State-of-the-art polyhedral code generator
- Takes statement domains and affine schedules to generate transformed code
- Uses efficient “polyhedral scanning” algorithm to generate imperfectly nested loops that scan a union of polyhedra (corresponding to statement domains)

→ http://www.cloog.org
for (i=1; i<=N; i++)
  for (j=i; j<=N; j++)
    S1(i,j);

for (i=1; i<=M; i++) /* M<N */
  for (j=1; j<=N; j++)
    S2(i,j);

for (j=1; j<=N; j++) {
  S1(1,j);
  S2(1,j);
}

for (i=2; i<=N; i++) {
  for (j=1; j<=i-1; j++)
    S2(i,j);
  for (j=i; j<=N; j++) {
    S1(i,j);
    S2(i,j);
  }
}

for (i=M+1; i<=N; i++)
  for (j=i; j<=N; j++)
    S1(i,j);
Tiled Code Generation in Polyhedral Model

Original loop:
for (i=1; i<=N; i++)
for (j=1; j<=N; j++)
S(i,j);

Tiled loop:
for (it=0; it<=floor(N/32); it++)
for (jt=0; jt<=floor(N/32); jt++)
for (i=max(1,32*it); i<=min(N,32*it+31); i++)
for (j=max(1,32*jt); j<=min(N,32*jt+31); j++)
S(i,j);

Statement domain:

Affine schedule:

Constraint of polyhedral model and ILP machinery:
Inequalities of the loop bounds must be linear in terms of loop iterators and symbolic parameters

Tile sizes = 32 x 32
Parametric Tiling: Perfectly Nested Loop

```
for (i=lbi; i<=ubi; i++)
for (j=lbj(i); j<=ubj(i); j++)
S(i,j);
```

**Output pseudocode:**

```
for it {
    [compute lbv]
    [compute ubv]
    if (lbv<ubv) {
        [prolog j]
        [full tiles j]
        [epilog j]
    } else {
        [untiled j]
    }
}
[epilog i]
```

```
No full tiles
Full tiles
Partial tile (loop i)
Full tiles (loop i)
```
Parametric Tiling: Imperfectly Nested Loops

Output pseudocode:

for it {
    [compute lbv1,ubv1,lbv2,ubv2]
    if (lbv1<ubv1) {
        [prolog j1]
        [full tiles j1]
        if (lbv2<ubv2) {
            [epilog j1 + prolog j2]
            [full tiles j2]
            [epilog j2]
        } else {
            [epilog j1 + untiled j2]
        }
    } else {
        /* ... omitted ... */
    }
    [epilog i]
}
Multi-Level Tiling

- **Essential for:**
  - Exploiting data locality in deep multi-level memory hierarchies

- **Approach:**
  - Boundary tiles can be recursively tiled using smaller tile sizes
Implementation: PrimeTile

- A **Parametric Multi-Level Tiler for Imperfect Loop Nests**
Experiments

- **Xeon workstation** (dual quad-core E5462 Xeon processors (8 cores total) running at 2.8 GHz (1600 MHz FSB) with 32 KB L1 cache, 12 MB of L2 cache (6 MB shared per core pair), and 16 GB of DDR2 FBDIMM RAM, running Linux kernel version 2.6.25 (x86-64))

- **GCC version 4.2.4**
  - Options: -O3

- **Comparisons with other tiled-code generators**

<table>
<thead>
<tr>
<th>Tiled code generator</th>
<th>Tile sizes</th>
<th>Loop nest structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiTLOG</td>
<td>Parametric</td>
<td>Perfect</td>
</tr>
<tr>
<td>Pluto</td>
<td>Fixed</td>
<td>Imperfect</td>
</tr>
<tr>
<td>PrimeTile</td>
<td>Parametric</td>
<td>Imperfect</td>
</tr>
</tbody>
</table>
## Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Imperfect nest</th>
<th>Require skewing</th>
<th>Input problem size</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>LU factorization</td>
<td>Yes</td>
<td>No</td>
<td>N=2500</td>
</tr>
<tr>
<td>2D FDTD</td>
<td>2D Finite Difference Time Domain method</td>
<td>Yes</td>
<td>Yes</td>
<td>T=2000, N=2000</td>
</tr>
<tr>
<td>1D Jacobi</td>
<td>1D Jacobi method</td>
<td>Yes</td>
<td>Yes</td>
<td>T=2000, N=6x10^6</td>
</tr>
<tr>
<td>Cholesky</td>
<td>Cholesky factorization</td>
<td>Yes</td>
<td>No</td>
<td>N=5000</td>
</tr>
<tr>
<td>TriSolver</td>
<td>Triangular solver</td>
<td>Yes</td>
<td>No</td>
<td>N=3000</td>
</tr>
<tr>
<td>DSYRK</td>
<td>Symmetric rank k update</td>
<td>No</td>
<td>No</td>
<td>N=3000</td>
</tr>
<tr>
<td>DTRMM</td>
<td>Triangular matrix multiplication</td>
<td>No</td>
<td>No</td>
<td>N=3000</td>
</tr>
</tbody>
</table>
Efficiency of Code Generation

LU

- Pluto
- PrimeTile (full)
- PrimeTile (no boundary tiling)

Generation time (seconds) vs. Levels of tiling

Cholesky

- Pluto
- PrimeTile (full)
- PrimeTile (no boundary tiling)

Generation time (seconds) vs. Levels of tiling
• Fully polyhedral fixed tiled code generation does not scale
• Double benefit of PrimeTile: better scalability and parametric tiling
Performance of Generated Tiled Code

Parametric tiled code efficiency is comparable to or better than fixed tiled code
Impact of Separation of Partial and Full Tiles

- Pluto
- PrimeTile
- Pluto (unroll/jam)
- PrimeTile (unroll)
- PrimeTile (regtile)

Execution time (seconds)
Impact of Separation of Partial and Full Tiles

Execution time (seconds)

Seidel

DSYRK

DTRMM

Pluto

PrimeTile

HiTLOG

Pluto(unroll/jam)

PrimeTile(unroll)

PrimeTile(regtile)

HiTLOG(unroll)

HiTLOG(regtile)

Identification of full-tile loops enables downstream optimization (e.g., register tiling)
Summary

- Developed an effective general approach to parametric multi-level tiling of imperfectly nested affine loops
- Achieved separation of partial tiles from full tiles, thereby enabling optimizations such as register tiling
- Ongoing/follow-up work targets parallel parametric tiling of affine imperfect loop nests

Software download:
1. A beta release of PrimeTile
2. A modified version of CLooG
   http://primetile.sourceforge.net
Thank You!