Tessellating Stencils

Liang Yuan, Yunquan Zhang, Peng Guo, Shan Huang
SKL of Computer Architecture, ICT, CAS
Outline

• Introduction
• Related work
• Tessellating Stencils
Stencil Overview

• Stencil
  – update each point in a $d$-dimensional space ($data space$) with a pre-defined pattern of neighbor points
  – time-iterated updates ($iteration space$)
Stencil Overview

• Classification
  – grid dimensions (1D, 2D, ...)
  – number of neighbors (3-point, 5-point, ...)
  – shapes (box, star, ...)
  – dependence types (Gauss-Seidel, Jacobi)
  – boundary conditions (constant, periodic, ...)
Related Work

• Overlapped tiling
  – Hyper-rectangle [PLDI’07]
  – 3.5d blocking [SC’10]

• Time skewing
  – eliminates the redundant computations [SC’01]
  – pipelined startup and limited concurrency [SPAA’01]
Related Work

- **Diamond tiling**
  - classic block [PLDI’08]
  - higher dimensional [SC’12]
  - hexagon in 2D and truncated octahedron in 3D [PPL’14]

- **Cache oblivious tiling**
  - serial cache oblivious stencil algorithms [ICS’05]
  - parallel cache oblivious stencil algorithms [SPAA’06]
  - Pochoir [SPAA’11]
Related Work

- **Split tiling**
  - avoids the overhead of the pipelined start-up in wavefront parallelization
  - classic split tiling [IMPACT’13]
  - nested split-tiling [ICS’13]

- **Hybrid tiling**
  - CATS [ICPP’11]
  - MWD [SIAMJOSC’15]
  - Grosser [CGO’14]
  - Hybrid split-tiling [ICS’13]
Reformulating Classic 1D Diamond Tiling

- Iteration space of 1D stencil
- Diamond tiling

- Each diamond is further split by a horizontal line
- 2D *time tile*.
  - The region between two splitting lines
  - every grid point in a *time tile* starts in a same time dimension and is updated by identical steps.
  - interleaved computations of triangle ($B_0$) or inverted triangle ($B_1$)
Extend to 2D Stencil

- Triangles in 1d stencil correspond to pyramids $B_0$
- Inverted triangles correspond to inverted pyramids $B_2$
- Tetrahedrons $B_1$
Implementation

• Coarsening
  – cut in different sizes

• Merge
  – $B_0$ and $B_1$ in 1D stencil.
  – $B_0$ and $B_2$ in 2D stencil.

• m-order stencil
  – combine every m points to a supernode.
  – 1-order one between supernodes.

• Periodic boundary
  – stretch and transform one block
1D code

for(tt = bt; tt < T; tt += bt){
    for(n = 0; n < #B0B1[level]; n++, level = 1 - level) {
        for(t = max(tt, 0) ; t < min(tt + 2 * bt, T); t++){
            xmin = max(XSLOPE, xright[level] - bx + n * ix + abs(t+1, tt+bt) * XSLOPE);
            xmax = min(N + XSLOPE, xright[level] + n * ix - abs(t+1, tt+bt) * XSLOPE);
            for(x = xmin; x < xmax; x++){
                update(t, x);}}}
}

bt: block size in time dimension
bx: block size in data space
bx ≠ bt: support of coarsening
#B0B1 is different in different level
Determine the scope of 1D data space
# B0B2 represents #B0 in level 0 or #B2 in level 1

two kinds of B1: B11 and B12

for 3D stencil we leave the unit-stride dimension uncut.
Evaluation

- two Intel Xeon E5-2670 processors with 2.70 GHz clock speed
- 32KB private L1 cache
- 256KB private L2 cache
- a unified 30MB L3 cache shared by 12 cores
- ICC compiler version 16.0.1, flag ‘-O3 -openmp’.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Problem Size</th>
<th>our blocking</th>
<th>Pluto blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heat-1D</td>
<td>12000000 × 4000</td>
<td>2000 × 1000</td>
<td>2000 × 2000</td>
</tr>
<tr>
<td>1d5p</td>
<td>12000000 × 4000</td>
<td>2000 × 500</td>
<td>2000 × 2000</td>
</tr>
<tr>
<td>Heat-2D</td>
<td>6000² × 2000</td>
<td>128 × 256 × 64</td>
<td>64 × 64 × 64</td>
</tr>
<tr>
<td>2d9p</td>
<td>6000² × 2000</td>
<td>128 × 256 × 64</td>
<td>64 × 64 × 64</td>
</tr>
<tr>
<td>Game of life</td>
<td>6000² × 2000</td>
<td>128 × 256 × 64</td>
<td>128 × 128 × 128</td>
</tr>
<tr>
<td>Heat-3D</td>
<td>256³ × 1000</td>
<td>24 × 24 × 12</td>
<td>12 × 12 × 12</td>
</tr>
<tr>
<td>3d27p</td>
<td>256³ × 1000</td>
<td>24 × 24 × 12</td>
<td>12 × 12 × 12</td>
</tr>
</tbody>
</table>
Experimental Results

- Our scheme and PluTo produce the same diamond tiling codes
- Pochoir generates trapezoidal blocks of different sizes.

same to Pluto  
better than Pochoir
Experimental Results

Pluto outperforms less than 5% with 24 cores

Our code outperforms Pluto and Pochoir by 14% and 20% on average.

- more suitable for box stencil.
Experimental Results

- 3D stencil leaves unit-stride uncut

Our code and Pochoir exhibit better scalability than Pluto. Our code outperforms Pluto and Pochoir by a maximum of 33% and 68%, and by 22% and 31% on average.

Our code outperforms Pluto and Pochoir by a maximum of 74% and 100%, and by 30% and 99% on average.
new?

• extension of 1D diamond
Not that new!

• Existing techniques produce similar blocks
  – Cache oblivious [SPAA’11]
  – Nested split-tiling [ICS’13]
  – divide multiple dimensions simultaneously
Advantage!

- **Diamond tiling**
  - fixed tile size at compile time
  - small size at the apex of a diamond
  - hard to choose the proper tile sizes to ensure the concurrent start

- **Cache oblivious tiling**
  - overhead of recursion
  - artificial dependencies

- **Split tiling**
  - synchronization overhead $2^d$
Formulating the two-level tessellation

Project $B_i$ in iteration space on data space, denoted as $B_i$

Determine the number of updating of each point in $B_i$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$E_i$</th>
<th>step1</th>
<th>step2</th>
<th>step3</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000000 0111110 0122200 0133210 0122210 0111110 0000000</td>
<td>11110 11110 11110</td>
<td>11110</td>
<td>11110</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>010 00210 0123210 0121210 010 0</td>
<td>11110</td>
<td>11110</td>
<td>11110</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>010 0123210 0121210 010 0</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0000000 0111110 0122210 0133210 0122210 0101110 0000000</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
<td>1</td>
</tr>
</tbody>
</table>
Formulating the two-level tessellation

- Consider a point \( a (a_0, \ldots, a_{d-1}) \) in \( B_0(0, \ldots, 0) \)
- updated time in \( B_i \), denoted as \( T_i(a_0, \ldots, a_{d-1}) \).

\[
T_i(a_0, \ldots, a_{d-1}) = \min(b, a_i, \ldots, a_{d-1}) - \max(0, a_0, \ldots, a_{i-1})
\]

\[
\sum_{i=0}^{d} T_i(a_0, \ldots, a_{d-1}) = (b - a_0) + (a_0 - a_1) + \cdots + (a_{d-2} - a_{d-1}) + a_{d-1} = b
\]

\( a_i \geq a_{i+1} \)
Summary

• two-level tessellation scheme
  – with highly concurrent execution
  – executes without redundant computation
  – achieves maximize parallelism
  – without false dependencies.
  – Calculate blocks without relying on the compiler
  – Extend to n-dimensional stencil

• mathematical structure of tessellation
  – Associate coordinates of elements with number of updating steps for each block.