Input Space Splitting for OpenCL

Simon Moll, Johannes Doerfert, Sebastian Hack

Saarbrücken Graduate School of Computer Science
Saarland University
Saarbrücken, Germany

October 29, 2015
OpenCL: Execution Model

NDRange
OpenCL: Parallelized & Vectorized
Vectorization (SIMD)

Perform the same operation for multiple vector lanes simultaneously.
Vectorization (SIMD)

Perform the same operation for multiple vector lanes simultaneously.

Vector Patterns

Consecutive: contiguous entries
Uniform: single entry
Divergent: unrelated entries

\[
\begin{align*}
< i, i+1, i+2, i+3 > & \\
<i,i,i,i> & \rightarrow i \\
<i,j,7,-> & 
\end{align*}
\]

```c
for (i = 0; i < 16; i++)
    O[i] = I[i] + 2;
```

```c
for (i = 0; i < 16; i += 2)
    O[i] = I[i] + 1;
```
Different threads execute different code paths
Different threads execute different code paths

Execute everything, mask out results of inactive threads (using predication, blending)

Control flow to data flow conversion on ASTs [Allen & Kennedy ‘83]

Whole-Function Vectorization on SSA CFGs [K & H ‘11]
Non-Divergent Control Flow

- Idea: optimize cases where threads do not diverge

```
Thread | Trace
--- | ---
1 | a b c e b d e f
2 | a b c e b d e f
3 | a b c e b d e f
4 | a b c e b d e f
```
Non-Divergent Control Flow

- Idea: optimize cases where threads do not diverge

<table>
<thead>
<tr>
<th>Thread</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>2</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>3</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>4</td>
<td>a b c e b d e f</td>
</tr>
</tbody>
</table>

- Option 1: Insert dynamic predicate-tests & branches to skip paths
  - “Branch on superword condition code” (BOSCC) [Shin et al. PACT’07]
  - Additional overhead for dynamic test
  - Does not help against increased register pressure
Non-Divergent Control Flow

- Idea: optimize cases where threads do not diverge

![Thread Trace Diagram]

<table>
<thead>
<tr>
<th>Thread</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>2</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>3</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>4</td>
<td>a b c e b d e f</td>
</tr>
</tbody>
</table>

- Option 2: **Statically prove non-divergence** of certain blocks
  - Non-divergent blocks can be excluded from linearization
  - Less executed code, less register pressure
  - More conservative than dynamic test
Non-Divergent Control Flow

- Idea: optimize cases where threads do **not** diverge

<table>
<thead>
<tr>
<th>Thread</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a b c e f</td>
</tr>
<tr>
<td>2</td>
<td>a b c e f</td>
</tr>
<tr>
<td>3</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>4</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>5</td>
<td>a b c e b d e f</td>
</tr>
<tr>
<td>6</td>
<td>a b c e b d e f</td>
</tr>
</tbody>
</table>

- Option 3: **Statically split non-divergence inputs**
  - Code versions with improved divergence properties
  - Orthogonal to both other options $\implies$ combination possible
2D Convolution
2D Convolution
2D Convolution
2D Convolution
2D Convolution

x

y

Simon Moll, Johannes Doerfert, Sebastian Hack
2D Convolution

```c
int left   = x - 2;
int right  = x + 2;
int top    = y - 2;
int bottom = y + 2;

int sum = 0;
for (int i = left; i <= right; ++i)
    for (int j = top; j <= bottom; ++j)
        sum += input[j][i] * mask[j - top][i - left];
output[y][x] = sum;
```
2D Convolution

```cpp
auto left  = x - 2;
auto right = x + 2;
int top    = y - 2;
int bottom = y + 2;

int sum = 0;
for (auto i = left; i <= right; ++i)
    for (int j = top; j <= bottom; ++j)
        sum += input[j][i] * mask[j - top][i - left];
output[y][x] = sum;
```
2D Convolution

Simon Moll, Johannes Doerfert, Sebastian Hack
2D Convolution

Motivation

October 29, 2015
2D Convolution

```c
int left    = MAX(0, x - 2);
int right   = MIN(width - 1, x + 2);
int top     = MAX(0, y - 2);
int bottom  = MIN(height - 1, y + 2);

int sum = 0;
for (int i = left; i <= right; ++i)
    for (int j = top; j <= bottom; ++j)
        sum += input[j][i] * mask[j - (y - 2)][i - (x - 2)];
output[y][x] = sum;
```
auto left = MAX(0, x - 2);
auto right = MIN(width - 1, x + 2);
int top = MAX(0, y - 2);
int bottom = MIN(height - 1, y + 2);

int sum = 0;
for (auto i = left; i <= right; ++i)
    for (int j = top; j <= bottom; ++j)
        sum += input[j][i] * mask[j - (y - 2)][i - (x - 2)];
output[y][x] = sum;
Input Space Splitting

\[ x \quad y \]

Simon Moll, Johannes Doerfert, Sebastian Hack

October 29, 2015 13 / 25
Input Space Splitting

\[ x \]
\[ y \]

\[ x \]
\[ y \]
Input Space Splitting

vector

scalar
The Polyhedral Model

\[ S: \quad A[i][j] = /* ... */; \]
\[ \text{if} \ (j \leq i) \]
\[ P: \quad A[i][j] += A[j][i]; \]
The Polyhedral Model

```
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j++) {
        S: A[i][j] = /* ... */
        if (j <= i)
            P: A[i][j] += A[j][i];
    }
```
The Polyhedral Model

for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j++) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }

\[ I_S = \{(S, (i, j)) \mid 0 \leq i \leq N \land 0 \leq j \leq N\} \]
The Polyhedral Model

for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j++) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }

\[ I_S = \{(S, (i, j)) | 0 \leq i \leq N \land 0 \leq j \leq N \} \]

\[ I_P = \{(P, (i, j)) | 0 \leq i \leq N \land 0 \leq j \leq i \} \]
The Polyhedral Model

for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j++) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }

\[ F_S = \{(S, (i, j)) \rightarrow (i, j)\} \]
The Polyhedral Model

\[
\text{for (int } i = 0; i \leq N; i++) \\
\quad \text{for (int } j = 0; j \leq N; j++) \\
\quad \text{S: A}[i][j] = /* ... */; \\
\quad \text{if } (j \leq i) \\
\quad \text{P: A}[i][j] += A[j][i]; \\
\}
\]

\[
\mathcal{F}_S = \{(S,(i,j)) \rightarrow (i,j)\} \\
\mathcal{F}_{P_1} = \{(P,(i,j)) \rightarrow (i,j)\} \\
\mathcal{F}_{P_2} = \{(P,(i,j)) \rightarrow (j,i)\}
\]
Splitting Predicates

Full Tile Predicate

for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }
Splitting Predicates

Full Tile Predicate

```java
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }
```

\[
\text{Full}_S = \{(S, (i, j)) \mid (j - (j \mod 8)) + 7 \leq N\}
\]
Splitting Predicates

Full Tile Predicate

\[
\text{for (int } i = 0; i <= N; i++) \\
\quad \text{for (int } j = 0; j <= N; j += 8) \{ \\
\quad \quad S: A[i][j] = /* ... */; \\
\quad \quad \quad \text{if (} j \leq i \text{)} \\
\quad \quad P: A[i][j] += A[j][i]; \\
\quad \}
\]

\[
\text{Full}_S = \{(S, (i,j)) \mid (j - (j \mod 8)) + 7 \leq N\}
\]
\[
\text{Full}_P = \{(P, (i,j)) \mid (j - (j \mod 8)) + 7 \leq \min(i, N)\}
\]
Splitting Predicates

Uniform Access Predicate

```c
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }
```

\[ \text{Uni}_{\mathcal{F}_S} = \{ (S, (i, j)) \mid \mathcal{F}_S (i, j+1) = \mathcal{F}_S (i, j) \} \]
Splitting Predicates

Uniform Access Predicate

\[
\text{for (int } i = 0; i \leq N; i++)
\]
\[
\text{for (int } j = 0; j \leq N; j += 8) \{
\text{S: } A[i][j] = /* ... */;
\text{if (j } \leq i)
\text{P: } A[i][j] += A[j][i];
\}
\]

\[
\text{Uni}_{\mathcal{F}_S} = \{(S, (i, j)) \mid \mathcal{F}_S (i, j + 1) = \mathcal{F}_S (i, j)\}
\]
\[
\text{Uni}_{\mathcal{F}_{P_1}} = \{(P, (i, j)) \mid \mathcal{F}_{P_1} (i, j + 1) = \mathcal{F}_{P_1} (i, j)\}
\]
\[
\text{Uni}_{\mathcal{F}_{P_2}} = \{(P, (i, j)) \mid \mathcal{F}_{P_2} (i, j + 1) = \mathcal{F}_{P_2} (i, j)\}
\]
Splitting Predicates

Uniform Access Predicate

```c
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }
```

\[
\text{Uni}_{F_S} = \{(S, (i, j)) \mid F_S (i, j + 1) = F_S (i, j)\} = \{
\}
\]

\[
\text{Uni}_{F_{P_1}} = \{(P, (i, j)) \mid F_{P_1} (i, j + 1) = F_{P_1} (i, j)\} = \{
\}
\]

\[
\text{Uni}_{F_{P_2}} = \{(P, (i, j)) \mid F_{P_2} (i, j + 1) = F_{P_2} (i, j)\} = \{
\}
\]
Splitting Predicates

Consecutive Access Predicate

for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }

Cons\(_{\mathcal{F}_S}\) = \{(S, (i, j)) | \mathcal{F}_S (i, j + 1) = \mathcal{F}_S (i, j) + 1\}

Cons\(_{\mathcal{F}_{P_1}}\) = \{(P, (i, j)) | \mathcal{F}_{P_1} (i, j + 1) = \mathcal{F}_{P_1} (i, j) + 1\}

Cons\(_{\mathcal{F}_{P_2}}\) = \{(P, (i, j)) | \mathcal{F}_{P_2} (i, j + 1) = \mathcal{F}_{P_2} (i, j) + 1\}
Splitting Predicates

Consecutive Access Predicate

```c
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8) {
        S: A[i][j] = /* ... */;
        if (j <= i)
            P: A[i][j] += A[j][i];
    }
```

Cons \(\mathcal{F}_S\) = \{(S, (i, j)) \mid \mathcal{F}_S (i, j + 1) = \mathcal{F}_S (i, j) + 1\} = \mathcal{I}_S

Cons \(\mathcal{F}_{P_1}\) = \{(P, (i, j)) \mid \mathcal{F}_{P_1} (i, j + 1) = \mathcal{F}_{P_1} (i, j) + 1\} = \mathcal{I}_P

Cons \(\mathcal{F}_{P_2}\) = \{(P, (i, j)) \mid \mathcal{F}_{P_2} (i, j + 1) = \mathcal{F}_{P_2} (i, j) + 1\} = \{\}
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8)
        if (i <= NumParticles) {
            S: ...
        }
    }
CFG simplification
Hoisting conditionals

for (int i = 0; i <= N; i++)
  for (int j = 0; j <= N; j += 8)
    if (i <= NumParticles) {
      S: ... 
    }

for (int i = 0; i <= NumParticles; i++)
  for (int j = 0; j <= i; j += 8)
    S: ...
Hoisting conditionals

```java
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8)
        if (reverse) {
            S: ...
        } else {
            P: ...
        }
```
CFG simplification

Hoisting conditionals

```java
for (int i = 0; i <= N; i++)
    for (int j = 0; j <= N; j += 8)
        if (reverse) {
            S: ...
        } else {
            P: ...
        }
```

```java
if (reverse) {
    for (int i = 0; i <= N; i++)
        for (int j = 0; j <= i; j += 8)
            S: ...
} else {
    for (int i = 0; i <= N; i++)
        for (int j = 0; j <= i; j += 8)
            P: ...
}
Predicate Based Domain Splitting
Predicate Based Domain Splitting

Kernel

Polyhedral Model
Predicate Based Domain Splitting

- Kernel
- Polyhedral Model
- Subkernel
- Subkernel
- Subkernel
Predicate Based Domain Splitting

Kernel

Polyhedral Model

Subkernel
Subkernel
Subkernel
Scalar Kernel
Predicate Based Domain Splitting

- Splitting Predicates
  - Polyhedral Model
    - Subkernel
      - Scalar Codelet
    - Subkernel
      - Scalar Codelet
    - Subkernel
      - Scalar Codelet
  - Scalar Kernel
Predicate Based Domain Splitting

Kernel

- Splitting Predicates

Polyhedral Model

- Subkernel
- Subkernel
- Subkernel

Scalar Kernel

Vector Codelet

Vector Codelet

Scalar Codelet

Vector Codelet
Predicate Based Domain Splitting

Kernel

Polyhedral Model

Splitting Predicates

Subkernel

Vector Codelet

Optimized Kernel

Subkernel

Vector Codelet

Scalar Codelet

Subkernel

Vector Codelet

Scalar Kernel
Evaluation

Pipeline

```
OpenCL driver -- OpenCL API --> App

<table>
<thead>
<tr>
<th>Kernel module (LLVM)</th>
<th>Barrier elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Barrier-free kernels</td>
</tr>
<tr>
<td></td>
<td>Polly</td>
</tr>
<tr>
<td></td>
<td>Polyhedral kernel</td>
</tr>
<tr>
<td></td>
<td>Optimization</td>
</tr>
</tbody>
</table>

- Domain Knowledge

<table>
<thead>
<tr>
<th>Optimization --&gt;</th>
<th>Polyhedral kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Polly</td>
</tr>
<tr>
<td></td>
<td>Barrier-free kernels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization --&gt;</th>
<th>Domain Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Polyhedral kernel</td>
</tr>
<tr>
<td></td>
<td>Barrier-free kernels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Polyhedral kernel</th>
<th>Barrier-free kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Polly</td>
</tr>
<tr>
<td></td>
<td>OpenCL driver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IOC --&gt;</th>
<th>Vector codelets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dispatch code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IOC --&gt;</th>
<th>Vector codelets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dispatch code</td>
</tr>
</tbody>
</table>
```
Evaluation

Performance

![Bar chart showing the performance comparison between different algorithms: scalar, vec only, split, and Intel. The chart displays speed up for each algorithm (BS, DCT, LUD, Myo, Floyd, C2D, BinOpt).](chart.png)
Ongoing Work

- Model synchronization the Polyhedral Model.
- Apply polyhedral optimizations (scheduling).
- Improve the representation of non-affine parts.
Conclusion

[Diagram and graph showing different kernels and codelets with speedup comparison across various benchmarks.]
OpenCL Programming Model
OpenCL Programming Model

work group

work
OpenCL Programming Model

work item

work group

work
Codelet Score

\[
\text{Score}_n(k) := \begin{cases} \\
\sum_{Q \in k, F \in \mathcal{F}_Q} w_{\text{cons}} \| \text{Box} (\text{Cons}_F (d_k)) \| & \text{if } n \geq w \\
+ w_{\text{uni}} \| \text{Box} (\text{Uni}_F (d_k)) \| \\
0 & \text{otw.}
\end{cases}
\]
Access Splitting Predicate

\[ \mathcal{I}_k^C := \bigcap_{Q \in k} \bigcap_{\mathcal{F} \in \mathcal{F}_Q, \text{st} \ Cons_{\mathcal{F}}(d_k) \neq \emptyset} Cons_{\mathcal{F}}(d_k). \]
Full Tile Predicate

\[
\begin{align*}
\text{partial tile} & \quad \begin{array}{c}
0 \\
i_d \\
\cdots \\
i_d - (i_d \mod w) \\
i_d \\
n - 1 \\
n
\end{array} \\
\text{full tile} & \quad \begin{array}{c}
0 \\
i_d \\
\cdots \\
i_d - (i_d \mod w) + (w - 1) \\
n - 1 \\
n
\end{array}
\end{align*}
\]