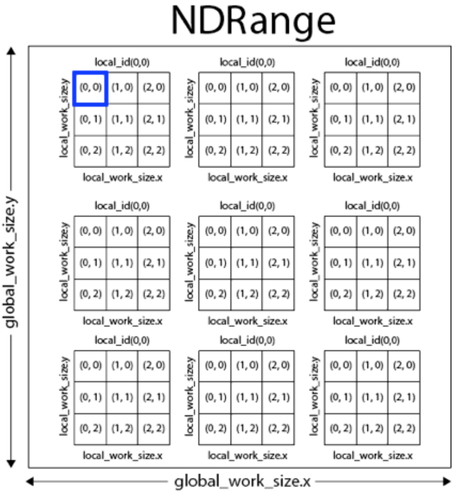
Input Space Splitting for OpenCL

Simon Moll, Johannes Doerfert, Sebastian Hack

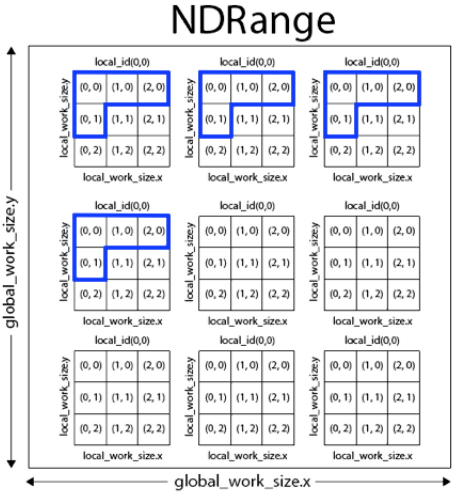
Saarbrücken Graduate School of Computer Science
Saarland University
Saarbrücken, Germany

October 29, 2015

OpenCL: Execution Model



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 $\underline{< \mathtt{i}, \mathtt{i}+1, \mathtt{i}+2, \mathtt{i}+3 >}$

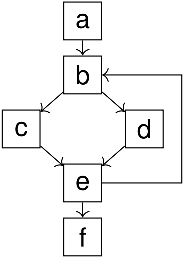
Uniform: single entry $\mathtt{<i,i,i,i>} \rightarrow \mathtt{i}$

Divergent: unrelated entries $\overline{< \mathtt{i}, \mathtt{j}, 7, - >}$

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

```
for (i = 0; i < 16; i += 2)
  O[i] = I[i] + 1;
```

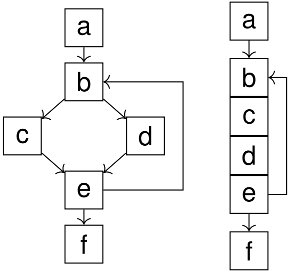
Diverging Control Flow



Thread	Trace
1	a b c e f
2	a b d e f
3	a b c e b c e f
4	a b c e b d e f

- Different threads execute different code paths

Diverging Control Flow

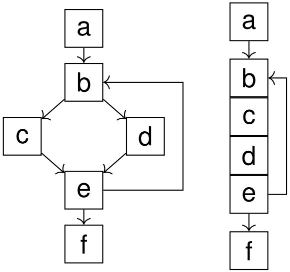


Thread	Trace
1	a b c d e b c d e f
2	a b c d e b c d e f
3	a b c d e b c d e f
4	a b c d e b c d e f

- 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

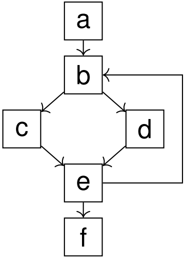
- Idea: optimize cases where threads do not diverge

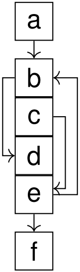


Thread	Trace
1	a b c e b d e f
2	a b c e b d e f
3	a b c e b d e f
4	a b c e b d e f

Non-Divergent Control Flow

- Idea: optimize cases where threads do not diverge



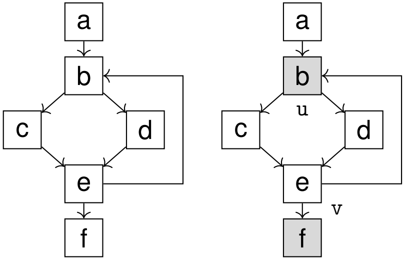


Thread	Trace
1	a b c e b d e f
2	a b c e b d e f
3	a b c e b d e f
4	a b c e b d e f

- 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



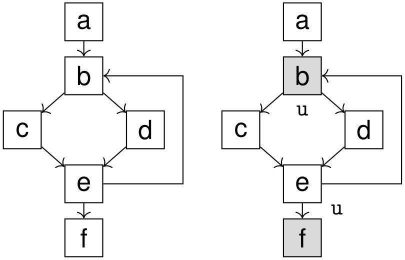
Thread	Trace
1	a b c e b d e f
2	a b c e b d e f
3	a b c e b d e f
4	a b c e b d e f

- 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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Idea: optimize cases where threads do not diverge

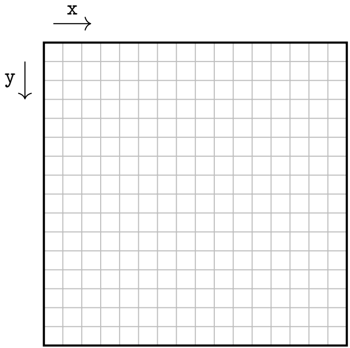


Thread	Trace
1	a b c e f
2	a b c e f
3	a b c e b d e f
4	a b c e b d e f
5	a b c e b d e f
6	a b c e b d e f

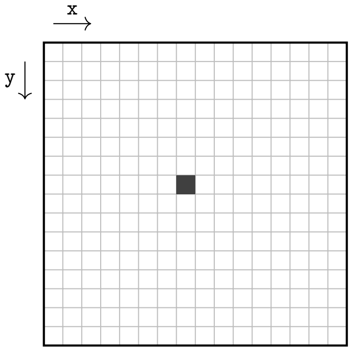
- Option 3: Statically split non-divergence inputs
  - Code versions with improved divergence properties
  - Orthogonal to both other options $\Longrightarrow$ combination possible

2D Convolution

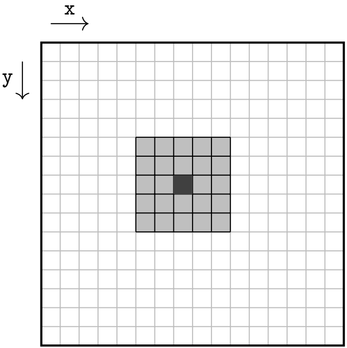
SAARLAND UNIVERSITY
COMPUTER SCIENCE



2D Convolution

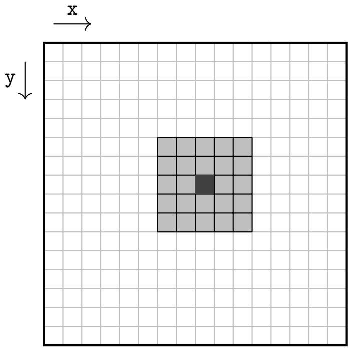


2D Convolution

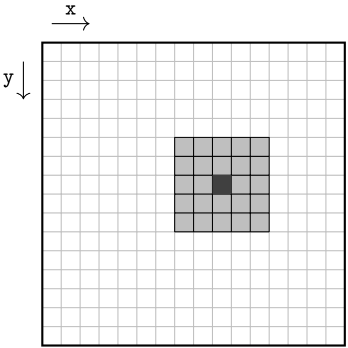


2D Convolution

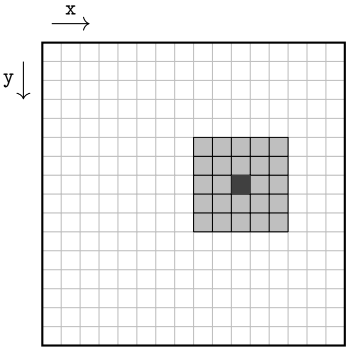
SAARLAND UNIVERSITY
COMPUTER SCIENCE



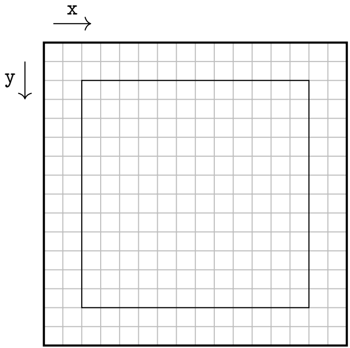
2D Convolution



2D Convolution



2D Convolution



2D Convolution

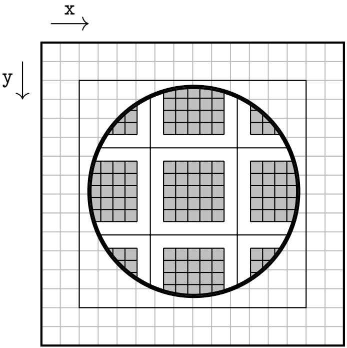
SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
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;
```

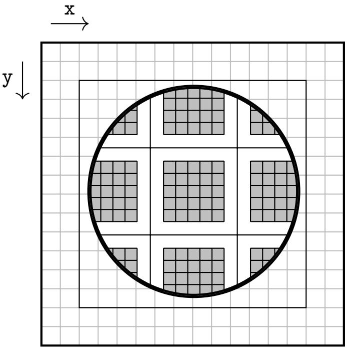
2D Convolution

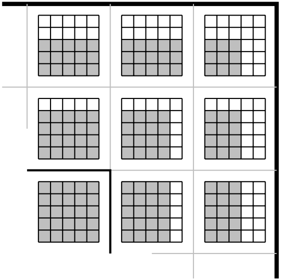

```
auto left   = x - 2;
auto right  = x + 2;
int top     = y - 2;
int bottom  = y + 2;

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



2D Convolution





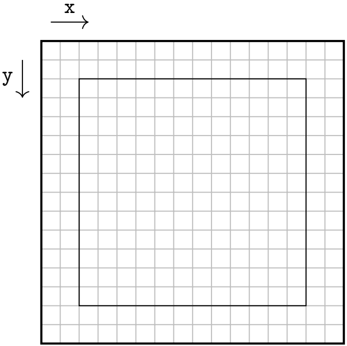
2D Convolution


```
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;
```

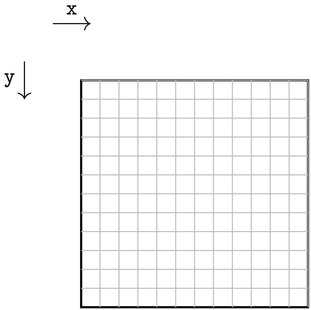
2D Convolution

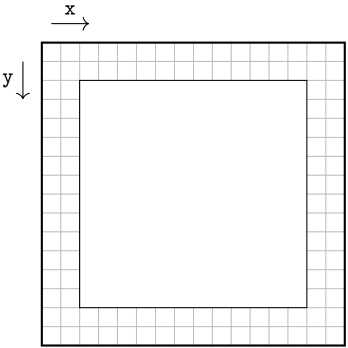
```
auto left   = MAX(0, x - 2);
auto right  = MIN(width - 1, x + 2);
int top     = MAX(0, y - 2);
int bottom  = MIN(height - 1, y + 2);

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

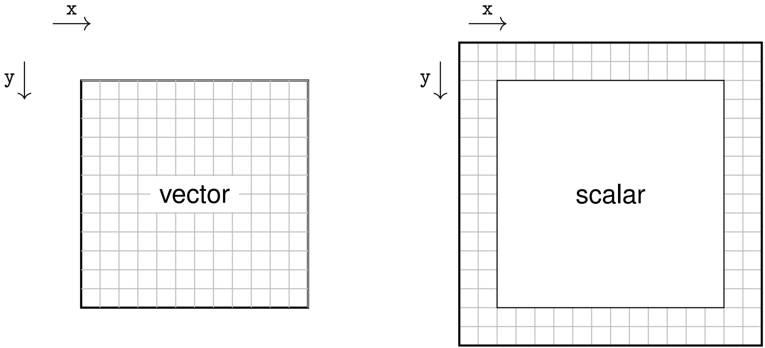


Input Space Splitting





Input Space Splitting



The Polyhedral Model

SAARLAND UNIVERSITY
COMPUTER SCIENCE

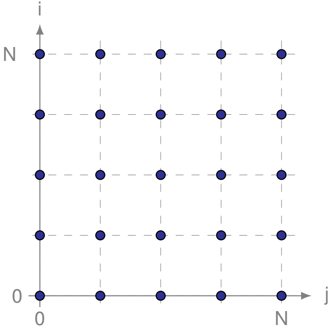
```
S:  A[i][j] = /* ... */;
    if (j <= i)
P:    A[i][j]+= A[j][i];
```

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];
  }
```

The Polyhedral Model

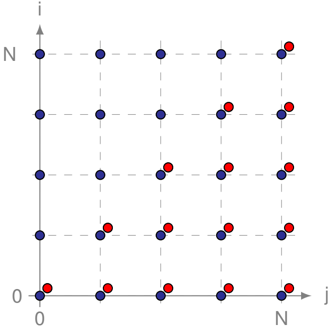

```
for (int i = 0; i <= N; i++)
  for (int j = 0; j <= N; j++) {
S:  A[i][j] = /* ... */;
    if (j <= i)
P:    A[i][j]+= A[j][i];
  }
```



$$\mathcal{I}_S = \{(S, (i, j)) \mid 0 \leq i \leq N \wedge 0 \leq j \leq N\}$$

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];
  }
```

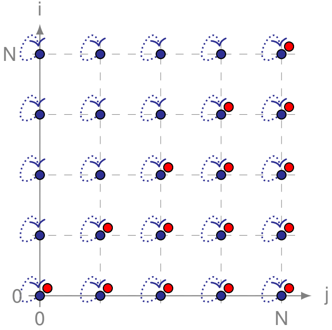


$$\mathcal{I}_S = \{(S, (i, j)) \mid 0 \leq i \leq N \wedge 0 \leq j \leq N\}$$

$$\mathcal{I}_P = \{(P, (i, j)) \mid 0 \leq i \leq N \wedge 0 \leq j \leq i\}$$

The Polyhedral Model

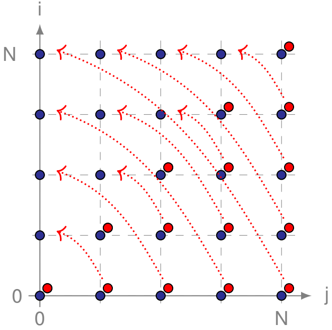

```
for (int i = 0; i <= N; i++)
  for (int j = 0; j <= N; j++) {
S:   A[i][j] = /* ... */;
     if (j <= i)
P:     A[i][j]+= A[j][i];
  }
```



$$\mathcal{F}_{S} = \{(S, (i, j)) \to (i, j)\}$$

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];
  }
```



$$\mathcal{F}_{\mathtt{S}} = \{(\mathtt{S}, (\mathtt{i}, \mathtt{j})) \rightarrow (\mathtt{i}, \mathtt{j})\}$$

$$\mathcal{F}_{\mathtt{P}_1} = \{(\mathtt{P}, (\mathtt{i}, \mathtt{j})) \rightarrow (\mathtt{i}, \mathtt{j})\} \quad \mathcal{F}_{\mathtt{P}_2} = \{(\mathtt{P}, (\mathtt{i}, \mathtt{j})) \rightarrow (\mathtt{j}, \mathtt{i})\}$$

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

```
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];
  }
```

$$\text{Full}_{\text{S}} = \{(\text{S}, (i, j)) \mid (j - (j \bmod 8)) + 7 \leq N\}$$

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];
  }
```

$$\text{Full}_{\texttt{S}} = \{(\texttt{S}, (i, j)) \mid (j - (j \bmod 8)) + 7 \leq N\}$$

$$\text{Full}_{\texttt{P}} = \{(\texttt{P}, (i, j)) \mid (j - (j \bmod 8)) + 7 \leq \texttt{min}(i, N)\}$$

Splitting Predicates

Uniform 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];
  }
```

$$\mathrm{Uni}_{\mathcal{F}_S} = \{(S,(i,j)) \mid \mathcal{F}_S\,(i,j+1) = \mathcal{F}_S\,(i,j)\}$$

Splitting Predicates

Uniform 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];
  }
```

$$\mathrm{Uni}_{\mathcal{F}_S} = \{(S,(i,j)) \mid \mathcal{F}_S\,(i,j+1) = \mathcal{F}_S\,(i,j)\}$$

$$\mathrm{Uni}_{\mathcal{F}_{P_1}} = \{(P,(i,j)) \mid \mathcal{F}_{P_1}(i,j+1) = \mathcal{F}_{P_1}(i,j)\}$$

$$\mathrm{Uni}_{\mathcal{F}_{P_2}} = \{(P,(i,j)) \mid \mathcal{F}_{P_2}(i,j+1) = \mathcal{F}_{P_2}(i,j)\}$$

Splitting Predicates

Uniform Access Predicate


```
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];
  }
```

$$\text{Uni}_{\mathcal{F}_S} = \{(S,(i,j)) \mid \mathcal{F}_S\,(i,j+1) = \mathcal{F}_S\,(i,j)\} = \{\}$$

$$\text{Uni}_{\mathcal{F}_{P_1}} = \{(P,(i,j)) \mid \mathcal{F}_{P_1}(i,j+1) = \mathcal{F}_{P_1}(i,j)\} = \{\}$$

$$\text{Uni}_{\mathcal{F}_{P_2}} = \{(P,(i,j)) \mid \mathcal{F}_{P_2}(i,j+1) = \mathcal{F}_{P_2}(i,j)\} = \{\}$$

Splitting Predicates

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];
  }
```

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

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

$$\text{Cons}_{\mathcal{F}_{P_2}} = \{(P,(i,j)) \mid \mathcal{F}_{P_2}(i,j+1) = \mathcal{F}_{P_2}(i,j)+1\}$$

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];
  }
```

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

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

$$\mathrm{Cons}_{\mathcal{F}_{P_2}} = \{(P, (i,j)) \mid \mathcal{F}_{P_2}(i, j+1) = \mathcal{F}_{P_2}(i,j) + 1\} = \{\}$$

CFG simplification

Hoisting conditionals

```
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:  ...
```

CFG simplification

Hoisting conditionals

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

CFG simplification

Hoisting conditionals

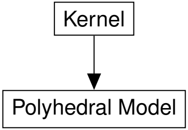
```
 for (int i = 0; i <= N; i++)
   for (int j = 0; j <= N; j += 8)
     if (reverse) {
 S:    ...
     } else {
 P:    ...
     }

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

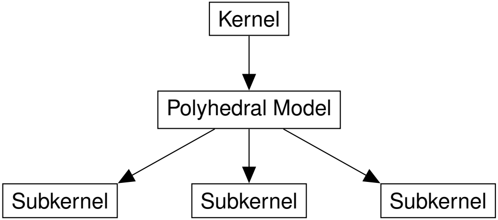
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Kernel

Predicate Based Domain Splitting

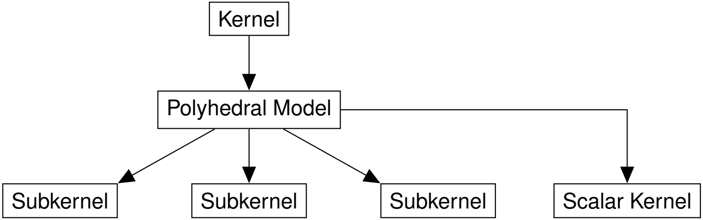


Predicate Based Domain Splitting

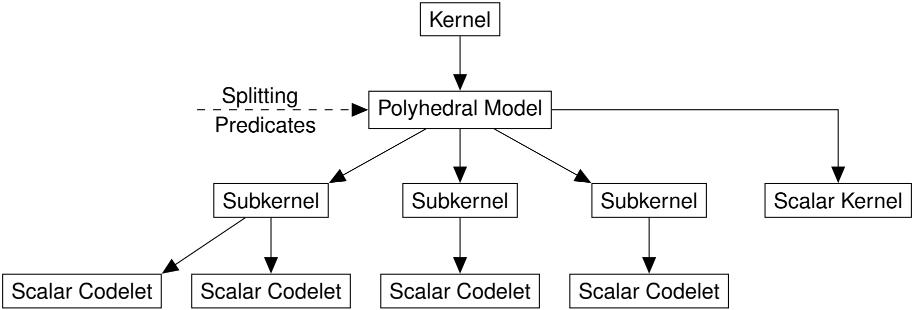


Predicate Based Domain Splitting

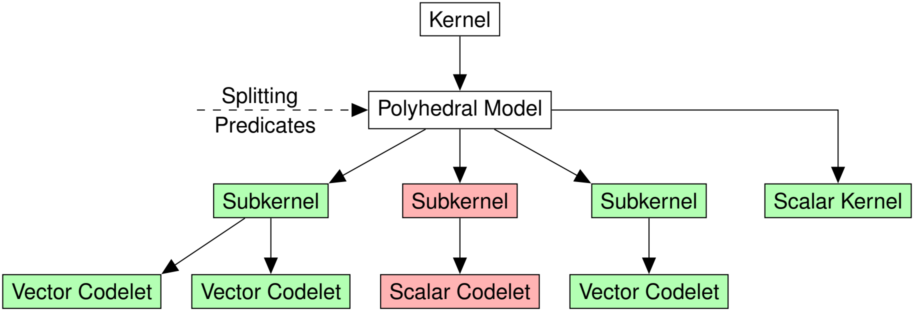
SAARLAND UNIVERSITY
COMPUTER SCIENCE



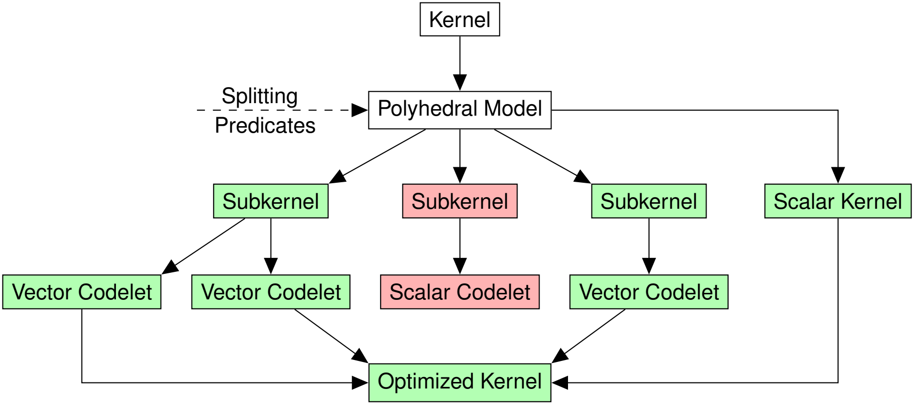
Predicate Based Domain Splitting



Predicate Based Domain Splitting

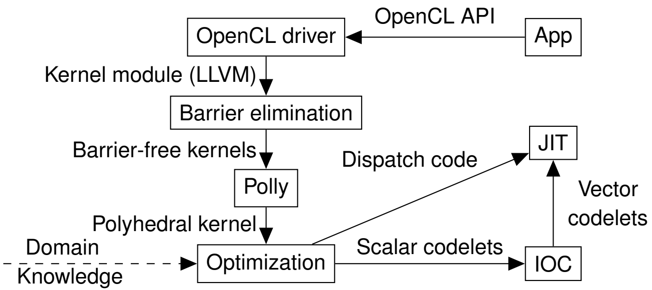


Predicate Based Domain Splitting



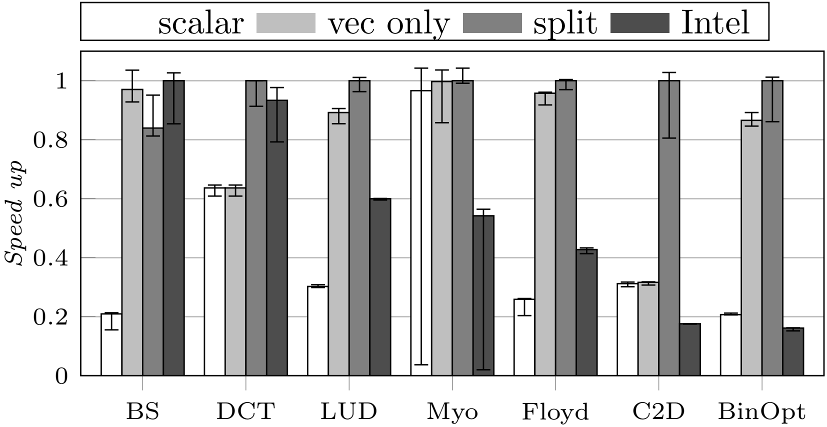
Evaluation

Pipeline



Evaluation

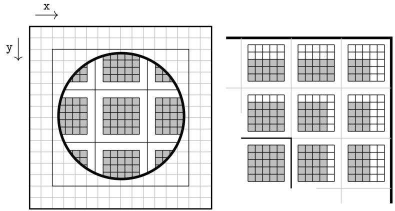
Performance

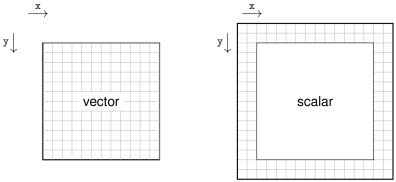


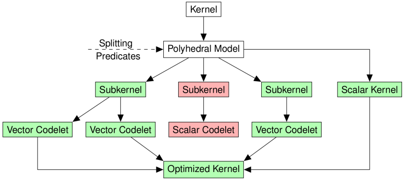
Ongoing Work

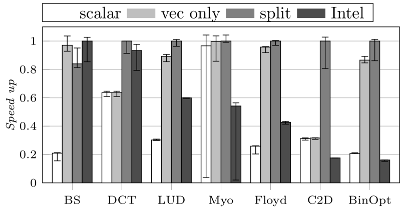
- Model synchronization the Polyhedral Model.
- Apply polyhedral optimizations (scheduling).
- Improve the representation of non-affine parts.

Conclusion

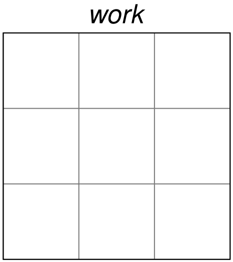

x
y


x
y
vector
x
y
scalar

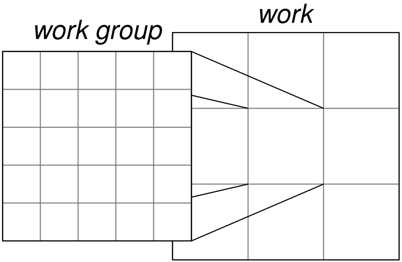

Kernel
Splitting Predicates
Polyhedral Model
Subkernel
Subkernel
Subkernel
Scalar Kernel
Vector Codelet
Vector Codelet
Scalar Codelet
Vector Codelet
Optimized Kernel


scalar
vec only
split
Intel
Speed up
0
0.2
0.4
0.6
0.8
1
BS
DCT
LUD
Myo
Floyd
C2D
BinOpt

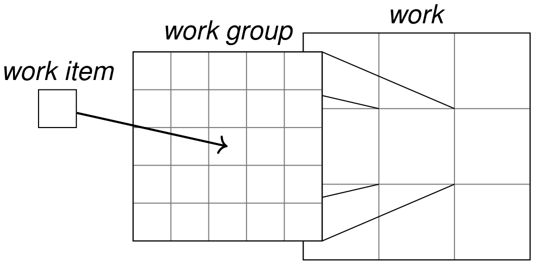
OpenCL Programming Model



OpenCL Programming Model



OpenCL Programming Model



Codelet Score

$$Score_n(k) := \begin{cases} \Sigma_{Q \in k, \mathcal{F} \in \mathbb{F}_Q} & \\ \quad w_{cons} \| Box(Cons_{\mathcal{F}}(d_k)) \| & \text{if } n \geq w \\ \quad + w_{uni} \| Box(Uni_{\mathcal{F}}(d_k)) \| & \\ 0 & otw. \end{cases}$$

Access Splitting Predicate

$$\mathcal{I}_k^C := \bigcap_{Q \in k} \bigcap_{\substack{\mathcal{F} \in \mathbb{F}_Q, st \\ Cons_{\mathcal{F}}(d_k) \neq \varnothing}} Cons_{\mathcal{F}}(d_k) \ .$$

Full Tile Predicate