Optimizing the Memory Hierarchy by Compositing Automatic Transformations on Computations and Data

Jie Zhao
State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, 450001, China

Peng Di
Huawei Technologies Co., Ltd.
Beijing, 100085, China

The 53rd IEEE/ACM International Symposium on Microarchitecture (MICRO-53)
Global Online Event

October 20, 2020
Outline

1. Introduction
2. Constructing Tile Shapes
3. Post-tiling Fusion
4. Code Generation
5. Experimental results
6. Conclusion
While providing the programmer the illusion of unlimited, fastest memories, it also complicates the programming issue. Optimizing compilers use the compositions of loop tiling and fusion to maximize the usage of the memory hierarchy. Loop tiling and fusion interfere with each other due to the oversight of transformations on data in memories. Polyhedral compilation is recognized for its powerful ability to composite loop transformations.
While providing the programmer the illusion of unlimited, fastest memories, it also complicates the programming issue.

Optimizing compilers use the compositions of loop tiling and fusion to maximize the usage of the memory hierarchy.
While providing the programmer the illusion of unlimited, fastest memories, it also complicates the programming issue.

Optimizing compilers use the compositions of loop tiling and fusion to maximize the usage of the memory hierarchy.

Loop tiling and fusion interfere with each other due to the oversight of transformations on data in memories.

Polyhedral compilation is recognized for its powerful ability to composite loop transformations.
The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.
The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

```c
for (h=0; h<H; h++)
    for (w=0; w<W; w++)
        A[h][w] = Quant(A[h][w]); /* S0 */
for (h=0; h<=H-KH; h++)
    for (w=0; w<=W-KW; w++){
        C[h][w] = 0; /* S1 */
        for (kh=0; kh<KH; kh++)
            for (kw=0; kw<KW; kw++)
                C[h][w] += A[h+kh][w+kw]*B[kh][kw]; /* S2 */
    }
for (h=0; h<=H-KH; h++)
    for (w=0; w<=W-KW; w++)
        C[h][w] = ReLU(C[h][w]); /* S3 */
```
The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

```c
for (h = 0; h < H; h++)
  for (w = 0; w < W; w++)
    A[h][w] = Quant(A[h][w]); /* S0 */
for (h = 0; h <= H - KH; h++)
  for (w = 0; w <= W - KW; w++) {
    C[h][w] = 0; /* S1 */
    for (kh = 0; kh < KH; kh++)
      for (kw = 0; kw < KW; kw++)
        C[h][w] += A[h+kh][w+kw] * B[kh][kw]; /* S2 */
  }
for (h = 0; h <= H - KH; h++)
  for (w = 0; w <= W - KW; w++)
    C[h][w] = ReLU(C[h][w]); /* S3 */
```

iteration domain (integer sets):

\{S_0(h, w) : 0 \leq h < H \land 0 \leq w < W; S_1(h, w) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW; S_2(h, w, kh, kw) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \land 0 \leq kh < KH \land 0 \leq kw < KW; S_3(h, w) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW\}
The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

```c
for (h = 0; h < H; h++)
    for (w = 0; w < W; w++)
        A[h][w] = Quant(A[h][w]); /* S0 */
for (h = 0; h <= H - KH; h++)
    for (w = 0; w <= W - KW; w++) {
        C[h][w] = 0; /* S1 */
        for (kh = 0; kh < KH; kh++)
            for (kw = 0; kw < KW; kw++)
                C[h][w] += A[h + kh][w + kw] * B[kh][kw]; /* S2 */
    }
for (h = 0; h <= H - KH; h++)
    for (w = 0; w <= W - KW; w++)
        C[h][w] = ReLU(C[h][w]); /* S3 */
```

write access relations (affine maps):

\[
\{ S_0(h, w) \rightarrow A(h, w) : 0 \leq h < H \land 0 \leq w < W ;
S_1(h, w) \rightarrow C(h, w) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW ;
S_2(h, w, kh, kw) \rightarrow C(h, w) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \land 0 \leq kh < KH \land 0 \leq kw < KW ;
S_3(h, w) \rightarrow C(h, w) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \} \]
The polyhedral model

The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

```plaintext
for (h=0; h<H; h++)
  for (w=0; w<W; w++)
    A[h][w] = Quant(A[h][w]); /* S0 */
for (h=0; h<=H-KH; h++)
  for (w=0; w<=W-KW; w++){
    C[h][w] = 0; /* S1 */
    for (kh=0; kh<KH; kh++)
      for (kw=0; kw<KW; kw++)
        C[h][w] += A[h+kh][w+kw]*B[kh][kw]; /* S2 */
  }
for (h=0; h<=H-KH; h++)
  for (w=0; w<=W-KW; w++)
    C[h][w] = ReLU(C[h][w]); /* S3 */
```

read access relations (affine maps):

\[
\{ S_0(h, w) \rightarrow A(h, w) : 0 \leq h < H \land 0 \leq w < W; S_2(h, w, kh, kw) \rightarrow A(h + kh, w + kw) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \land 0 \leq kh < KH \land 0 \leq kw < KW; S_2(h, w, kh, kw) \rightarrow B(kh, kw) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \land 0 \leq kh < KH \land 0 \leq kw < KW; S_3(h, w) \rightarrow C(h, w) : 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \}\]

The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

```plaintext
for (h=0; h<H; h++)
  for (w=0; w<W; w++)
    A[h][w] = Quant(A[h][w]); /* S0 */
for (h=0; h<=H-KH; h++)
  for (w=0; w=W-KW; w++)
    C[h][w] = 0; /* S1 */
  for (kh=0; kh<KH; kh++)
    for (kw=0; kw<KW; kw++)
      C[h][w] += A[h+kh][w+kw]*B[kh][kw]; /* S2 */
for (h=0; h<=H-KH; h++)
  for (w=0; w=W-KW; w++)
    C[h][w] = ReLU(C[h][w]); /* S3 */
```

dependence relations (affine maps):

\{ S_0(h, w) \rightarrow S_2(h', w', kh = h - h', kw = w - w') : h' \geq 0 \land h - KH < h' \leq h \land h' \leq H - KH \land w' \geq 0 \land w - KW < w' \leq w \land w' \leq W - KW; S_2(h, w, kh = KH - 1, kw = KW - 1) \rightarrow S_3(h' = h, w' = w) : KH > 0 \land KW > 0 \land 0 \leq h \leq H - KH \land 0 \leq w \leq W - KW \}
The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

\[
\begin{align*}
  & \text{for} (h=0; h<H; h++) \\
  & \quad \text{for} (w=0; w<W; w++) \\
  & \quad \quad A[h][w] = \text{Quant}(A[h][w]); /* S_0 */ \\
  & \text{for} (h=0; h<=H-KH; h++) \\
  & \quad \text{for} (w=0; w=W-KW; w++) \\
  & \quad \quad C[h][w] = 0; /* S_1 */ \\
  & \quad \quad \text{for} (kh=0; kh<KH; kh++) \\
  & \quad \quad \quad \text{for} (kw=0; kw<KW; kw++) \\
  & \quad \quad \quad \quad C[h][w] += A[h+kh][w+kw]*B[kh][kw]; /* S_2 */ \\
  & \text{for} (h=0; h<=H-KH; h++) \\
  & \quad \text{for} (w=0; w=W-KW; w++) \\
  & \quad \quad C[h][w] = \text{ReLU}(C[h][w]); /* S_3 */
\end{align*}
\]

original schedule (textual execution order, affine maps):

\[
[S_0(h, w) \rightarrow (0, h, w); S_1(h, w) \rightarrow (1, h, w, 0); S_2(h, w, kh, kw) \rightarrow (1, h, w, 1, kh, kw); S_3(h, w) \rightarrow (2, h, w)]
\]
The polyhedral model [3, 5, 19] represents a program and its semantics using iteration domains, access relations, dependences and schedules.

```c
for (h = 0; h < H; h++)
    for (w = 0; w < W; w++)
        A[h][w] = Quant(A[h][w]); /* S0 */
for (h = 0; h <= H - KH; h++)
    for (w = 0; w <= W - KW; w++) {
        C[h][w] = 0; /* S1 */
        for (kh = 0; kh < KH; kh++)
            for (kw = 0; kw < KW; kw++)
                C[h][w] += A[h + kh][w + kw] * B[kh][kw]; /* S2 */
    }
for (h = 0; h <= H - KH; h++)
    for (w = 0; w <= W - KW; w++)
        C[h][w] = ReLU(C[h][w]); /* S3 */
```

new schedule (new execution order, affine maps):

\[
S_0(h, w) \to (0, h, w); S_1(h, w) \to (1, h, w, 0, 0, 0); S_2(h, w, kh, kw) \to (1, h, w, kh, kw, 1); S_3(h, w) \to (1, h, w, KH - 1, KW - 1, 2)\]
The new schedule implies a fusion strategy \( \{ S_0 \}, \{ S_1, S_2, S_3 \} \), and optimizing compilers can apply tiling on the generated code.

```c
for (h=0; h<H; h++)
    for (w=0; w<W; w++)
        A[h][w] = Quant(A[h][w]); /* S_0 */
for (h=0; h<=H-KH; h++)
    for (w=0; w<=W-KW; w++)
        C[h][w] = 0; /* S_1 */
    for (kh=0; kh<KH; kh++)
        for (kw=0; kw<KW; kw++)
            C[h][w] += A[h+kh][w+kw]*B[kh][kw]; /* S_2 */
for (h=0; h<=H-KH; h++)
    for (w=0; w<=W-KW; w++)
        C[h][w] = ReLU(C[h][w]); /* S_3 */
```

new schedule (new execution order, affine maps):

\[
S_0(h, w) \rightarrow (0, h, w); 
S_1(h, w) \rightarrow (1, h, w, 0, 0, 0); 
S_2(h, w, kh, kw) \rightarrow (1, h, w, kh, kw, 1); 
S_3(h, w) \rightarrow (1, h, w, KH - 1, KW - 1, 2) \]
The polyhedral model

The new schedule implies a fusion strategy ($S_0$, $S_1, S_2, S_3$), and optimizing compilers can apply tiling on the generated code.

```c
for(ht=0;ht<H/T0;ht+=T0)
  for(wt=0;wt<W/T1;wt+=W/T1)
    for(hp=0;hp<T0;hp++)
      for(wp=0;wp<T1;wp++)
        S0(ht+h,wt+wp);

for(ht=0;ht<(H-KH)/T2;ht+=T2)
  for(wt=0;wt<(W-KW)/T3;wt+=W/T3)
    for(hp=0;hp<T2;hp++)
      for(wp=0;wp<T3;wp++){
        S1(ht+h,wt+wp);
        for(kh=0;kh<=H-KH;kh++)
          for(kw=0;kw<=W-KW;kw++)
            S2(ht+h,wt+wp,kh,kw);
        S3(ht+h,wt+wp);
      }
```

new schedule (new execution order, affine maps):

$[S_0(h, w) \rightarrow (0, h, w); S_1(h, w) \rightarrow (1, h, w, 0, 0, 0); S_2(h, w, kh, kw) \rightarrow (1, h, w, kh, kw, 1); S_3(h, w) \rightarrow (1, h, w, KH - 1, KW - 1, 2)]$
The polyhedral model

The new schedule implies a fusion strategy \(\{S_0\}, \{S_1, S_2, S_3\}\), and optimizing compilers can apply tiling on the generated code.

```c
#pragma omp parallel for
for(ht=0;ht<H/T0;ht+=T0)
    for(wt=0;wt<W/T1;wt+=W/T1)
        for(hp=0;hp<T0;hp++)
            for(wp=0;wp<T1;wp++)
                S0(ht+h,wt+wp);

#pragma omp parallel for
for(ht=0;ht<(H-KH)/T2;ht+=T2)
    for(wt=0;wt<(W-KW)/T3;wt+=W/T3)
        for(hp=0;hp<T2;hp++)
            for(wp=0;wp<T3;wp++){
                S1(ht+h,wt+wp);
                for(kh=0;kh<=H-KH;kh++)
                    for(kw=0;kw<=W-KW;kw++)
                        S2(ht+h,wt+wp,kh,kw);
                S3(ht+h,wt+wp);
            }
```

One can generate OpenMP code for CPUs \((T_0, T_1, T_2, T_3\) are tile sizes).
The polyhedral model

The new schedule implies a fusion strategy ($\{S_0\}, \{S_1, S_2, S_3\}$), and optimizing compilers can apply tiling on the generated code.

One can map code to GPU thread blocks and threads.
The new schedule implies a fusion strategy \( \{S_0\}, \{S_1, S_2, S_3\} \), and optimizing compilers can apply tiling on the generated code.

One can map code to GPU thread blocks and threads.
The polyhedral model

The new schedule implies a fusion strategy \( \{ S_0 \}, \{ S_1, S_2, S_3 \} \), and optimizing compilers can apply tiling on the generated code.

One can map code to GPU thread blocks and threads.

Can we fuse all statements into a single kernel? Can we reorder the sequence of loop fusion and tiling?
Conflicts in the data space

Let us first construct the \((h, w)\) computation spaces for each fusion group, quantization for the first and reduction for the second.
Conflicts in the data space

for(ht=0; ht< H/T; ht+=T)
    for(wt=0; wt<W/T; wt+=W/T)
        for(hp=0; hp<T; hp++)
            for(wp=0; wp<T; wp++)
                S0(ht+h, wt+wp);

for(ht=0; ht<(H-KH)/T; ht+=T)
    for(wt=0; wt<(W-KW)/T; wt+=T)
        for(hp=0; hp<T; hp++)
            for(wp=0; wp<T; wp++)
                { S1(ht+h, wt+wp);
                for(kh=0; kh<=H-KH; kh++)
                    for(kw=0; kw<=W-KW; kw++)
                        S2(ht+h, wt+wp, kh, kw);
                        S3(ht+h, wt+wp);
                }

One can construct the data space of tensor A that is written by $S_0$ in the quantization group and read by $S_2$ in the reduction group.
Conflicts in the data space

for (ht = 0; ht < H/T_0; ht += T_0)
  for (wt = 0; wt < W/T_1; wt += W/T_1)
    for (hp = 0; hp < T_0; hp++)
      for (wp = 0; wp < T_1; wp++)
        S_0 (ht+h, wt+wp);

for (ht = 0; ht < (H-KH)/T_2; ht += T_2)
  for (wt = 0; wt < (W-KW)/T_3; wt += W/T_3)
    for (hp = 0; hp < T_2; hp++)
      for (wp = 0; wp < T_3; wp++)
        { S_1 (ht+h, wt+wp);
          for (kh = 0; kh <= H-KH; kh++)
            for (kw = 0; kw <= W-KW; kw++)
              S_2 (ht+h, wt+wp, kh, kw);
        S_3 (ht+h, wt+wp); }

Existing polyhedral compilers tile each computation space individually, with tile sizes $T_0 = T_1 = 4$, $T_2 = T_3 = 2$. 
Conflicts in the data space

for(ht=0;ht<H/T;ht+=T)
  for(wt=0;wt<W/T;wt+=W/T)
    for(hp=0;hp<T;hp++)
      for(wp=0;wp<T;wp++)
        S0(ht+h,wt+wp);

for(ht=0;ht<(H-KH)/T;ht+=T)
  for(wt=0;wt<(W-KW)/T;wt+=W/T)
    for(hp=0;hp<T;hp++)
      for(wp=0;wp<T;wp++)
        { 
          S1(ht+h,wt+wp);
          for(kh=0;kh<=H-KH;kh++)
            for(kw=0;kw<=W-KW;kw++)
              S2(ht+h,wt+wp,kh,kw);
          S3(ht+h,wt+wp);
        }

A tile of the quantization space writes to 4 points into the data space of tensor A.
A tile of the reduction space requires 16 points from the data space of tensor A.
There exists a conflict in the data space of tensor $A$. 
Conflicts in the data space

```c
for(ht=0;ht<H/T0;ht+=T0)
    for(wt=0;wt<W/T1;wt+=W/T1)
        for(hp=0;hp<T0;hp++)
            for(wp=0;wp<T1;wp++)
                S0(ht+h,wt+wp);

for(ht=0;ht<(H-KH)/T2;ht+=T2)
    for(wt=0;wt<(W-KW)/T3;wt+=W/T3)
        for(hp=0;hp<T2;hp++)
            for(wp=0;wp<T3;wp++){
                S1(ht+h,wt+wp);
                for(kh=0;kh<=H-KH;kh++)
                    for(kw=0;kw<=W-KW;kw++)
                        S2(ht+h,wt+wp,kh,kw);
                S3(ht+h,wt+wp);
            }
```

We can first apply tiling only to the reduction space.
And we only tile the reduction space. This can tighten the tile size space and thus reduce compilation time.
Conflicts in the data space

```
for(ht=0;ht<H/T0;ht+=T0)
    for(wt=0;wt<W/T1;wt+=W/T1)
        for(hp=0;hp<T0;hp++)
            for(wp=0;wp<T1;wp++)
                S0(ht+h,wt+wp);

for(ht=0;ht<(H-KH)/T2;ht+=T2)
    for(wt=0;wt<(W-KW)/T3;wt+=W/T3)
        for(hp=0;hp<T2;hp++)
            for(wp=0;wp<T3;wp++)
                {
                    for(kh=0;kh<=H-KH;kh++)
                        for(kw=0;kw<=W-KW;kw++)
                            S2(ht+h,wt+wp,kh,kw);
                    S3(ht+h,wt+wp);
                }
```

Next we construct the data space of tensor $A$. 
Conflicts in the data space

One can compute the data tiles required by each of those computation tile of the reduction space. Let us focus on the blue and red tiles.
Conflicts in the data space

```c
for(ht=0;ht<H/T0;ht+=T0)
    for(wt=0;wt<W/T1;wt+=W/T1)
        for(hp=0;hp<T0;hp++)
            for(wp=0;wp<T1;wp++)
                S0(ht+h,wt+wp);

for(ht=0;ht<(H-KH)/T2;ht+=T2)
    for(wt=0;wt<(W-KW)/T3;wt+=W/T3)
        for(hp=0;hp<T2;hp++)
            for(wp=0;wp<T3;wp++){
                S1(ht+h,wt+wp);
                for(kh=0;kh<=H-KH;kh++)
                    for(kw=0;kw<=W-KW;kw++)
                        S2(ht+h,wt+wp,kh,kw);
                S3(ht+h,wt+wp);
            }
```

Now we can think about the tiling of the quantization space.
Conflicts in the data space

Polyhedral compilers can infer the tile shapes of the quantization space using the reverse of read access relation between $S_0$ and tensor $A$. 

\begin{verbatim}
for(ht=0;ht<H/T_0;ht+=T_0)
  for(wt=0;wt<W/T_1;wt+=W/T_1)
    for(hp=0;hp<T_0;hp++)
      for(wp=0;wp<T_1;wp++)
        S_0(ht+h,wt+wp);

for(ht=0;ht<(H-KH)/T_2;ht+=T_2)
  for(wt=0;wt<(W-KW)/T_3;wt+=W/T_3)
    for(hp=0;hp<T_2;hp++)
      for(wp=0;wp<T_3;wp++)
        \{ 
          S_1(ht+h,wt+wp);
          for(kh=0;kh<=H-KH;kh++)
            for(kw=0;kw<=W-KW;kw++)
              S_2(ht+h,wt+wp,kh,kw);
          S_3(ht+h,wt+wp);
        \}
\end{verbatim}
Conflicts in the data space

Our algorithm determines the tile shapes of intermediate computation spaces using the reverse of access relations, which was impossible in existing work.
Our tiling algorithm can be summarized as:

- Compute tiling schedules for a live-out computation space (e.g., the reduction space) using polyhedral schedulers;
- Compute the data tiles required by each tile of a live-out computation space;
- Determine the tile shapes of an intermediate computation space using the reverse of access relations.
The tiling algorithm

Our tiling algorithm can be summarized as:
- Compute tiling schedules for a live-out computation space (e.g., the reduction space) using polyhedral schedulers;
- Compute the data tiles required by each tile of a live-out computation space;
- Determine the tile shapes of an intermediate computation space using the reverse of access relations.

Our tiling algorithm
- is described in Algorithm 1 in the paper.
- can construct arbitrary tile shapes for intermediate computation spaces.
- can be used to handle more general applications like image processing pipelines, SpMV, linear algbra.
Our tiling algorithm can be summarized as:

- Compute tiling schedules for a live-out computation space (e.g., the reduction space) using polyhedral schedulers;
- Compute the data tiles required by each tile of a live-out computation space;
- Determine the tile shapes of an intermediate computation space using the reverse of access relations.

Our tiling algorithm generates

- A tiling schedule for the reduction space:

  \[
  \begin{align*}
  \{S_1(h, w) &\rightarrow \left( \frac{h}{T_2}, \frac{w}{T_3}, h, w \right); S_2(h, w, kh, kw) &\rightarrow \left( \frac{h}{T_2}, \frac{w}{T_3}, h, w, kh, kw \right); S_3(h, w) &\rightarrow \left( \frac{h}{T_2}, \frac{w}{T_3}, h, w \right)\} \\
  \end{align*}
  \]

  (1)

- An extension schedule for the quantization space:

  \[
  \{ (\alpha_0, \alpha_1) \rightarrow S_0(h, w) : 0 \leq \alpha_0 < \left\lceil (H - KH + 1) / T_2 \right\rceil \land 0 \leq \alpha_1 < \left\lceil (W - KW + 1) / T_3 \right\rceil \\
  \land T_2 \cdot \alpha_0 \leq h < T_2 \cdot \alpha_0 + KH + T_2 - 1 \land T_3 \cdot \alpha_1 \leq w < T_3 \cdot \alpha_1 + KW + T_3 - 1 \}
  \]

  (2)
The polyhedral model can also represent a program and its semantics using schedule trees [6].
The polyhedral model can also represent a program and its semantics using schedule trees [6].

original schedule (textual execution order, affine maps):

\[
S_0(h, w) \rightarrow (0, h, w); S_1(h, w) \rightarrow (1, h, w, 0); S_2(h, w, kh, kw) \rightarrow (1, h, w, 1, kh, kw); S_3(h, w) \rightarrow (2, h, w)
\]
The polyhedral model can also represent a program and its semantics using schedule trees [6].

original schedule (textual execution order, affine maps):

\[
\begin{align*}
S_0(h, w) &\rightarrow (0, h, w); \\
S_1(h, w) &\rightarrow (1, h, w, 0); \\
S_2(h, w, kh, kw) &\rightarrow (1, h, w, 1, kh, kw); \\
S_3(h, w) &\rightarrow (2, h, w)
\end{align*}
\]
The polyhedral model can also represent a program and its semantics using schedule trees [6].

new schedule (new execution order, affine maps):

\[ S_0 (h, w) \rightarrow (0, h, w); S_1 (h, w) \rightarrow (1, h, w, 0, 0); S_2 (h, w, kh, kw) \rightarrow (1, h, w, kh, kw, 1); S_3 (h, w) \rightarrow (1, h, w, KH - 1, KW - 1, 2) \]
The polyhedral model can also represent a program and its semantics using schedule trees \[6\].

new schedule (new execution order, affine maps):
\[
[S_0(h, w) \rightarrow (0, h, w); S_1(h, w) \rightarrow (1, h, w, 0, 0, 0); S_2(h, w, kh, kw) \rightarrow (1, h, w, kh, kw, 1); S_3(h, w) \rightarrow (1, h, w, KH - 1, KW - 1, 2)]
\]
The polyhedral model can also represent a program and its semantics using schedule trees [6].

- **Domain**: set of statement instances to be scheduled
- **Band**: multi-dimensional piecewise quasi-affine partial schedule
- **Filter**: selects statement instances that are executed by descendants
- **Sequence/Set**: children executed in given/arbitrary order
The polyhedral model can also represent a program and its semantics using schedule trees [6].

- **Domain**: set of statement instances to be scheduled
- **Band**: multi-dimensional piecewise quasi-affine partial schedule
- **Filter**: selects statement instances that are executed by descendants
- **Sequence/Set**: children executed in given/arbitrary order

```
\[
\begin{array}{c}
\text{domain} \\
\text{sequence} \\
\{S_0\} \\
\{S_0(h, w) \rightarrow (h, w)\} \\
\{S_1; S_2; S_3\} \\
\{S_1(h, w) \rightarrow (h, w); S_2(h, w, kh, kw) \rightarrow (h, w); S_3(h, w) \rightarrow (h, w)\} \\
\text{sequence} \\
\{S_1\} \\
\{S_2\} \\
\{S_3\} \\
\{S_2(h, w, kh, kw) \rightarrow (kh, kw)\}
\end{array}
\]
```
The polyhedral model can also represent a program and its semantics using schedule trees [6].

- **Domain**: set of statement instances to be scheduled
- **Band**: multi-dimensional piecewise quasi-affine partial schedule
- **Filter**: selects statement instances that are executed by descendants
- **Sequence/Set**: children executed in given/arbitrary order

```
domain
  sequence
    \{S_0\}       \{S_1; S_2; S_3\}
        \{S_0(h, w) \rightarrow (h, w)\}
        \{S_1(h, w) \rightarrow (h, w); S_2(h, w, kh, kw) \rightarrow (h, w); S_3(h, w) \rightarrow (h, w)\}
```

```
  sequence
    \{S_1\}       \{S_2\}       \{S_3\}
      \{S_2(h, w, kh, kw) \rightarrow (kh, kw)\}
```
The polyhedral model can also represent a program and its semantics using schedule trees [6].

- **Domain**: set of statement instances to be scheduled
- **Band**: multi-dimensional piecewise quasi-affine partial schedule
- **Filter**: selects statement instances that are executed by descendants
- **Sequence/Set**: children executed in given/arbitrary order

![Schedule trees diagram]

Post-tiling Fusion
The polyhedral model can also represent a program and its semantics using schedule trees [6].

Schedule trees also provide convenience node types:

- **Mark**: attach additional information to subtrees
- **Extension**: add additional domain elements to facilitate non-polyhedral semantics
Manipulations on schedule trees

We can implement post-tiling fusion by manipulating a schedule tree obtained after applying the Pluto-like schedulers [3].
Manipulations on schedule trees

Classical rectangular/parallelogram tiling can be applied using the tiling schedule (1) on page 6, with $T_2 \times T_3$ the tile sizes along $h \times w$ dimensions.
Manipulations on schedule trees

Split the band node that has implemented rectangular/parallelogram tiling for post-tiling fusion.
Manipulations on schedule trees

The subtree of $S_0$ is introduced with a sequence node indicating the order with its siblings.
Manipulations on schedule trees

An expansion node is mandatory to introduced additional statements, i.e., the subtree of $S_0$. 
Manipulations on schedule trees

\[\{S_0\} \quad \xrightarrow{mark: \text{"skipped"}} \quad \{S_1; S_2; S_3\}\]

\[\{S_0(h, w) \rightarrow (h, w)\}\]

\[\{S_1(h, w) \rightarrow (\frac{h}{2}, \frac{w}{3}); S_2(h, w, kh, kw) \rightarrow (\frac{h}{2}, \frac{w}{3}); S_3(h, w) \rightarrow (\frac{h}{2}, \frac{w}{3})\}\]

\[\text{extension: \"extension schedule (2) on page 6\"}\]

\[\{S_0\} \quad \xrightarrow{sequence} \quad \{S_1; S_2; S_3\}\]

\[\{S_0(h, w) \rightarrow (h, w)\}\]

\[\{S_1(h, w) \rightarrow (h, w); S_2(h, w, kh, kw) \rightarrow (h, w); S_3(h, w) \rightarrow (h, w)\}\]

\[\text{sequence}\]

\[\{S_1\} \quad \xrightarrow{sequence} \quad \{S_2\} \quad \xrightarrow{sequence} \quad \{S_3\}\]

\[\{S_2(h, w, kh, kw) \rightarrow (kh, kw)\}\]

A mark node is used to indicate the absence of the original subtree of \(S_0\).
The post-tiling fusion algorithm

Our post-tiling fusion algorithm can be summarized as:

- Tile a live-out computation space using the tiling schedules obtained by the tiling algorithm;
- Integrate extension schedules obtained by the tiling algorithm into the schedule tree representation;
- Indicate the absence of original subtree of the fused intermediate computation spaces.
Our post-tiling fusion algorithm can be summarized as:

- Tile a live-out computation space using the tiling schedules obtained by the tiling algorithm;
- Integrate extension schedules obtained by the tiling algorithm into the schedule tree representation;
- Indicate the absence of original subtree of the fused intermediate computation spaces.

Our post-tiling fusion algorithm

- is described in Algorithm 2 in the paper.
- does not resort to tedious aggressive fusion heuristics used by existing optimizers [3, 5, 17, 19].
- does not lose the parallelism of the program, guaranteeing the high performance of the generated code.
Our post-tiling fusion algorithm can be summarized as:

- Tile a live-out computation space using the tiling schedules obtained by the tiling algorithm;
- Integrate extension schedules obtained by the tiling algorithm into the schedule tree representation;
- Indicate the absence of original subtree of the fused intermediate computation spaces.

Our post-tiling algorithm returns a fusion strategy of \( \{S_0, S_1, S_2, S_3\} \), fusing all statements into a single loop nest without hampering the parallelism.
Generalizing the approach

We have to handle the scene that the values defined by an intermediate space $op_0$ are used by multiple live-out spaces $op_1$ and $op_2$. 

![Diagram showing the relationship between $op_0$, $op_1$, and $op_2$]
We have to handle the scene that the values defined by an intermediate space $op_0$ are used by multiple live-out spaces $op_1$ and $op_2$.

$op'_0$ represents the subset of $op_0$ that computes the values used by $op_1$, and $op''_0$ the subset that writes to the values read by $op_2$. 
Unlike existing heuristics [8, 11], $op'_0$ and $op''_0$ can be fused with their uses, respectively, without introducing redundant computations.

Otherwise, fusion is prevented due to possible redundancy.

This strategy also implements dead code elimination in some extreme cases that was not considered by existing polyhedral optimizers [3, 5, 18].

See Algorithm 3 in the paper for detailed explanation.
Code generation for CPUs and GPUs is implemented in PPCG [19], a polyhedral compiler using the isl library [18] as the solver.
Code generation for CPUs and GPUs is implemented in PPCG [19], a polyhedral compiler using the *isl* library [18] as the solver.
Code generation for CPUs and GPUs is implemented in PPCG [19], a polyhedral compiler using the *isl* library [18] as the solver.
Code generation for Huawei Ascend910 chips is available in the *akg* project\(^a\), a wrapper of TVM compiler [4].

\(^a\)https://gitee.com/mindspore/akg
Aggressive memory optimizations are fully considered:

- Allow scratchpad allocations on CPUs.
- Software-controlled memory management of private/shared memory on GPUs.
- Automatic memory promotion between different hierarchy buffers on Ascend910 chips.
Setup and methodology

- **Benchmarks:**
  - Resnet-50 workload [7]
  - PolyMage benchmarks [13]
  - *equake* from the SPEC CPU2000 benchmarks [2]
  - PolyBench benchmarks [14]

- **Architectures:**
  - Huawei Ascend910 chip
  - NVIDIA Quadro K4000 GPU
  - dural-socket 32-core Intel Xeon(R) CPU E5-2683 v4 @2.10GHz

- **Methodology:** Run each benchmark 10 times and report the average value.

- Please refer to our paper for more details.
## Performance of the PolyMage benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>stages</th>
<th>Tile size</th>
<th>CPU execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>parameter</td>
<td>naïve (1 core)</td>
</tr>
<tr>
<td>Bilateral Grid</td>
<td>7</td>
<td>8 × 64</td>
<td>66.01</td>
</tr>
<tr>
<td>Camera Pipeline</td>
<td>32</td>
<td>16 × 32</td>
<td>116.32</td>
</tr>
<tr>
<td>Harris Corner Detection</td>
<td>11</td>
<td>16 × 32</td>
<td>246.88</td>
</tr>
<tr>
<td>Local Laplacian Filter</td>
<td>99</td>
<td>8 × 64</td>
<td>480.48</td>
</tr>
<tr>
<td>Multiscale Interpolation</td>
<td>49</td>
<td>32 × 16</td>
<td>209.10</td>
</tr>
<tr>
<td>Unsharp Mask</td>
<td>4</td>
<td>8 × 32 × 3</td>
<td>142.16</td>
</tr>
</tbody>
</table>

- Our work provides 20% and 33% improvements over PolyMage [12] and Halide [15] when targeting CPUs.
Performance of the PolyMage benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>stages</th>
<th>GPU grid</th>
<th>GPU execution time (ms)</th>
<th>Compilation time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>parameter</td>
<td>PPCG (minfuse)</td>
<td>Halide</td>
</tr>
<tr>
<td>Bilateral Grid</td>
<td>7</td>
<td>8×64</td>
<td>5.07</td>
<td>3.79</td>
</tr>
<tr>
<td>Camera Pipeline</td>
<td>32</td>
<td>16×32</td>
<td>3.51</td>
<td>2.47</td>
</tr>
<tr>
<td>Harris Corner Detection</td>
<td>11</td>
<td>16×32</td>
<td>1.79</td>
<td>1.68</td>
</tr>
<tr>
<td>Local Laplacian Filter</td>
<td>99</td>
<td>8×64</td>
<td>16.73</td>
<td>12.53</td>
</tr>
<tr>
<td>Multiscale Interpolation</td>
<td>49</td>
<td>32×16</td>
<td>15.75</td>
<td>25.65</td>
</tr>
<tr>
<td>Unsharp Mask</td>
<td>4</td>
<td>8×32×3</td>
<td>2.03</td>
<td>1.94</td>
</tr>
</tbody>
</table>

- Our work provides 20% and 33% improvements over PolyMage [12] and Halide [15] when targeting CPUs.
- Our approach outperforms different fusion heuristics of PPCG [19] and provides a mean improvement of 17% over Halide [15] when targeting GPUs.
- Our approach also alleviates the compilation time.
The network is trained with the requirement of no less than 76% validation accuracy and the execution time is reported for a single training epoch.

The tile sizes are specified by experts in the DSL and we did not use the auto-tuner of the framework.
Performance of the Resnet-50 workload

<table>
<thead>
<tr>
<th></th>
<th>Execution time (ms)</th>
<th>Compilation time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>smart</td>
<td>Our work</td>
</tr>
<tr>
<td><strong>fwd conv+batchnorm</strong></td>
<td>11.50</td>
<td>6.69</td>
</tr>
<tr>
<td><strong>entire workload</strong></td>
<td>35.03</td>
<td>30.25</td>
</tr>
</tbody>
</table>

- The network is trained with the requirement of no less than 76% validation accuracy and the execution time is reported for a single training epoch.
- The tile sizes are specified by experts in the DSL and we did not use the auto-tuner of the framework.
- Please refer to our paper for more results on Polybench benchmarks and the *equake* benchmark.
Our tiling algorithm can construct arbitrary tile shapes, without refining scheduling algorithms [12] or resorting to complicated constraints [20].
Our tiling algorithm can construct arbitrary tile shapes, without refining scheduling algorithms [12] or resorting to complicated constraints [20].

We model the composition of tiling and fusion in the absence of tradeoffs between parallelism, locality and recomputation.
Our tiling algorithm can construct arbitrary tile shapes, without refining scheduling algorithms [12] or resorting to complicated constraints [20].

We model the composition of tiling and fusion in the absence of tradeoffs between parallelism, locality and recomputation.

Our approach moderates compilation time without restricting to special cases [16] or relaxing scheduling constraints [1].
Our tiling algorithm can construct arbitrary tile shapes, without refining scheduling algorithms [12] or resorting to complicated constraints [20].

We model the composition of tiling and fusion in the absence of tradeoffs between parallelism, locality and recomputation.

Our approach moderates compilation time without restricting to special cases [16] or relaxing scheduling constraints [1].

We show an in-depth performance comparison with the state of the art, with CPU, GPU and an AI accelerator being taken into consideration.
References

Polyhedral auto-transformation with no integer linear programming.

Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers.
Containing papers presented at the Symposium on Advances in Computational Mechanics.

A practical automatic polyhedral parallelizer and locality optimizer.

Tvm: An automated end-to-end optimizing compiler for deep learning.

Polly—performing polyhedral optimizations on a low-level intermediate representation.
Parallel Processing Letters 22, 04 (2012), 1250010.

Polyhedral ast generation is more than scanning polyhedra.

Deep residual learning for image recognition.
References

An effective fusion and tile size model for optimizing image processing pipelines.

Gpu implementation of linear morphological openings with arbitrary angle.

[10] Manegold, S.
Memory Hierarchy.

Revisiting loop fusion in the polyhedral framework.

Polymage: Automatic optimization for image processing pipelines.

Polymage benchmarks.

Polybench: The polyhedral benchmark suite.
Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines.

[16] Upadrashta, R., and Cohen, A.
Sub-polyhedral scheduling using (unit-)two-variable-per-inequality polyhedra.

The next 700 accelerated layers: From mathematical expressions of network computation graphs to accelerated gpu kernels, automatically.

[18] Verdoolaege, S.
Isl: An integer set library for the polyhedral model.

Polyhedral parallel code generation for cuda.

[20] Zhao, J., and Cohen, A.
Flextended tiles: A flexible extension of overlapped tiles for polyhedral compilation.