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

Jie Zhao

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhenghzhou, 450001, China

Peng Di

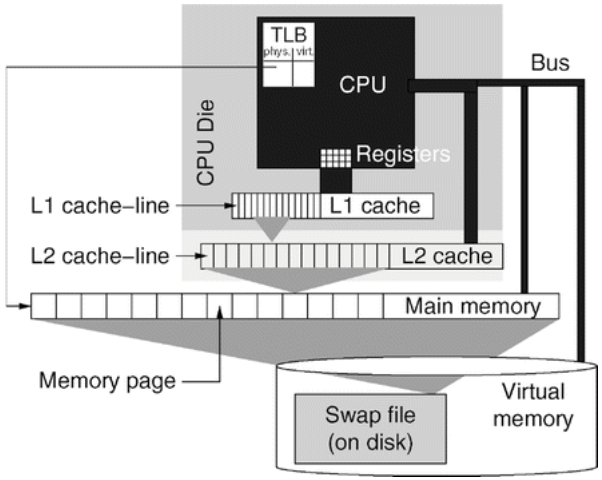
Huawei Technologies Co. ,Ltd. Beijing, 100085, China

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

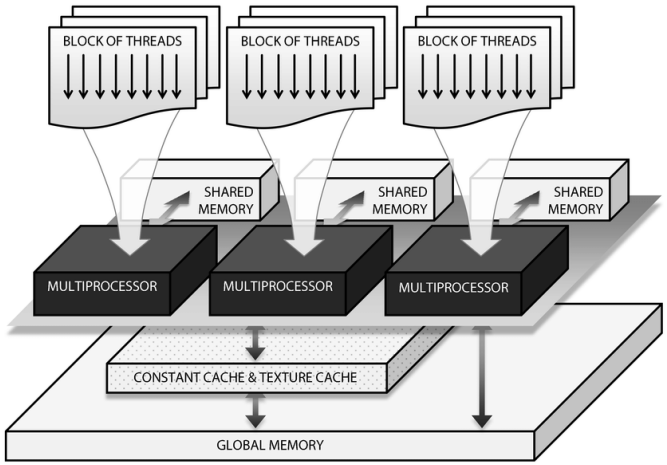
October 20, 2020

Outline

Memory Hierarchy on Modern Architectures

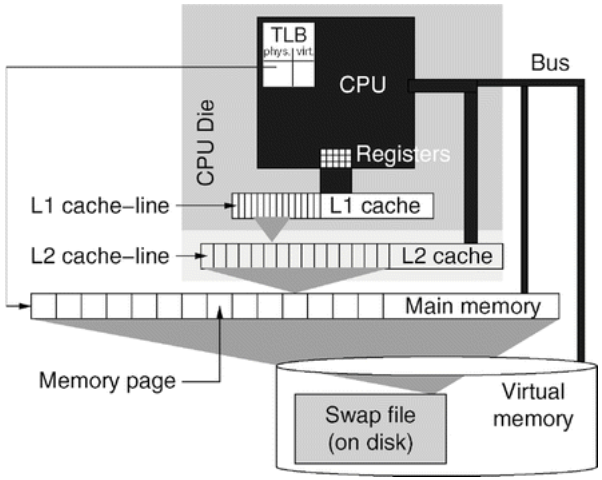


Memory hierarchy on CPUs [10]

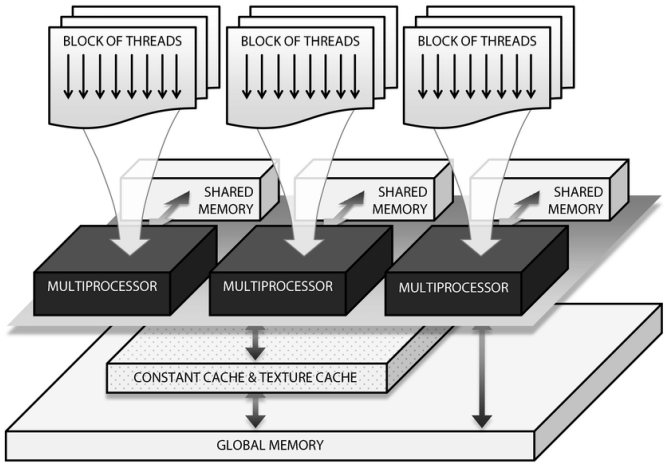


Memory hierarchy on GPUs [9]

Memory Hierarchy on Modern Architectures



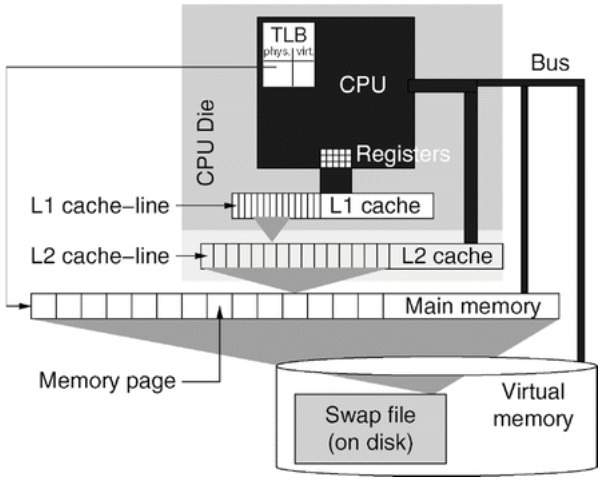
Memory hierarchy on CPUs [10]



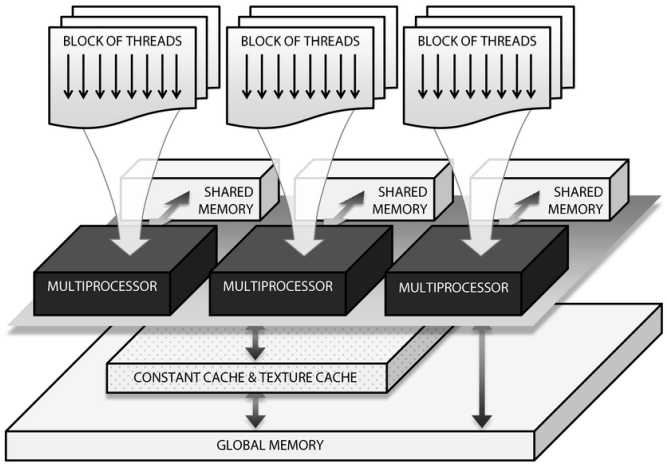
Memory hierarchy on GPUs [9]

- 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.

Memory Hierarchy on Modern Architectures



Memory hierarchy on CPUs [10]



Memory hierarchy on GPUs [9]

- 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

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

The polyhedral model

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

```
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

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

```
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):

$$\{S_0(h,w) : 0 \leq h < H \wedge 0 \leq w < W; S_1(h,w) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW; S_2(h,w,kh,kw) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW \wedge 0 \leq kh < KH \wedge 0 \leq kw < KW; S_3(h,w) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW\}$$

The polyhedral model

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

```
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):

$$\{S_0(h,w) \to A(h,w) : 0 \leq h < H \wedge 0 \leq w < W; S_1(h,w) \to C(h,w) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW; S_2(h,w,kh,kw) \to C(h,w) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW \wedge 0 \leq kh < KH \wedge 0 \leq kw < KW; S_3(h,w) \to C(h,w) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW\}$$

The polyhedral model

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

```
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):

$$\{S_0(h,w) \to A(h,w) : 0 \leq h < H \wedge 0 \leq w < W; S_2(h,w,kh,kw) \to A(h+kh, w+kw) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW \wedge 0 \leq kh < KH \wedge 0 \leq kw < KW; S_2(h,w,kh,kw) \to B(kh,kw) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW \wedge 0 \leq kh < KH \wedge 0 \leq kw < KW; S_3(h,w) \to C(h,w) : 0 \leq h \leq H - KH \wedge 0 \leq w \leq W - KW\}$$

The polyhedral model

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

```
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):

$\{S_0(h,w) \rightarrow S_2(h',w',kh=h-h',kw=w-w') : h' \geq 0 \wedge h-KH < h' \leq h \wedge h' \leq H-KH \wedge w' \geq 0 \wedge w-KW < w' \leq w \wedge w' \leq W-KW; S_2(h,w,kh=KH-1,kw=KW-1) \rightarrow S_3(h'=h,w'=w) : KH > 0 \wedge KW > 0 \wedge 0 \leq h \leq H-KH \wedge 0 \leq w \leq W-KW\}$

The polyhedral model

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

```
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 */
```

original schedule (textual execution order, affine maps):

$[S_0(h, w) \to (0, h, w); S_1(h, w) \to (1, h, w, 0); S_2(h, w, kh, kw) \to (1, h, w, 1, kh, kw); S_3(h, w) \to (2, h, w)]$

The polyhedral model

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

```
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):

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

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.

```
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):

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

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.

```
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.

```
#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);
      }
```

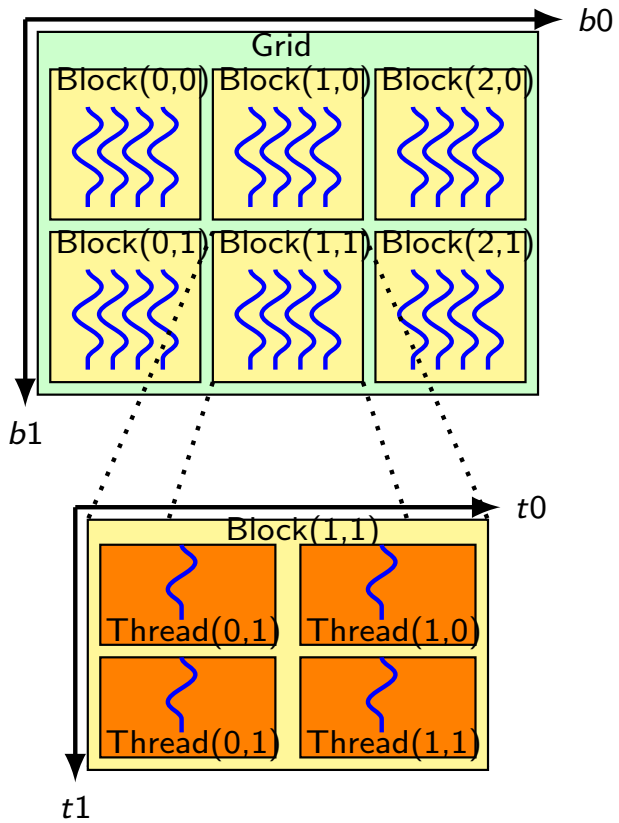
One can generate OpenMP code for CPUs (T_0, T_1, T_2, T_3 are tile sizes).

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.

```
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);
```

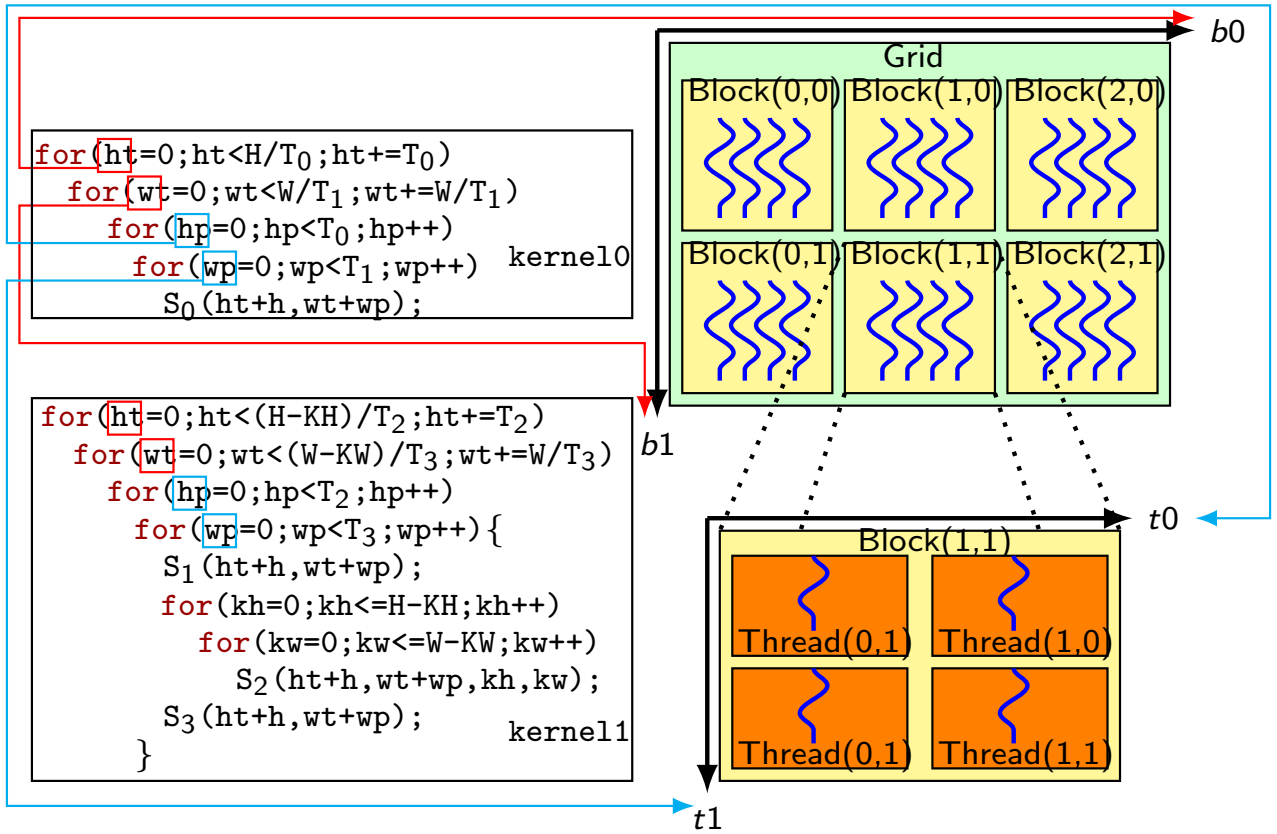
```
for(ht=0;ht<(H-KH)/T2;ht+=T2)
  for(wt=0;wt<(W-KW)/T3;wt+=W/T3)
    for(hp=0;hp<T2;hp++)
      for(wp=0;wp<T3;wp++){
        S1(ht+h,wt+wp);
        for(kh=0;kh<=H-KH;kh++)
          for(kw=0;kw<=W-KW;kw++)
            S2(ht+h,wt+wp,kh,kw);
        S3(ht+h,wt+wp);
      }
```



One can map code to GPU thread blocks and threads.

The polyhedral model

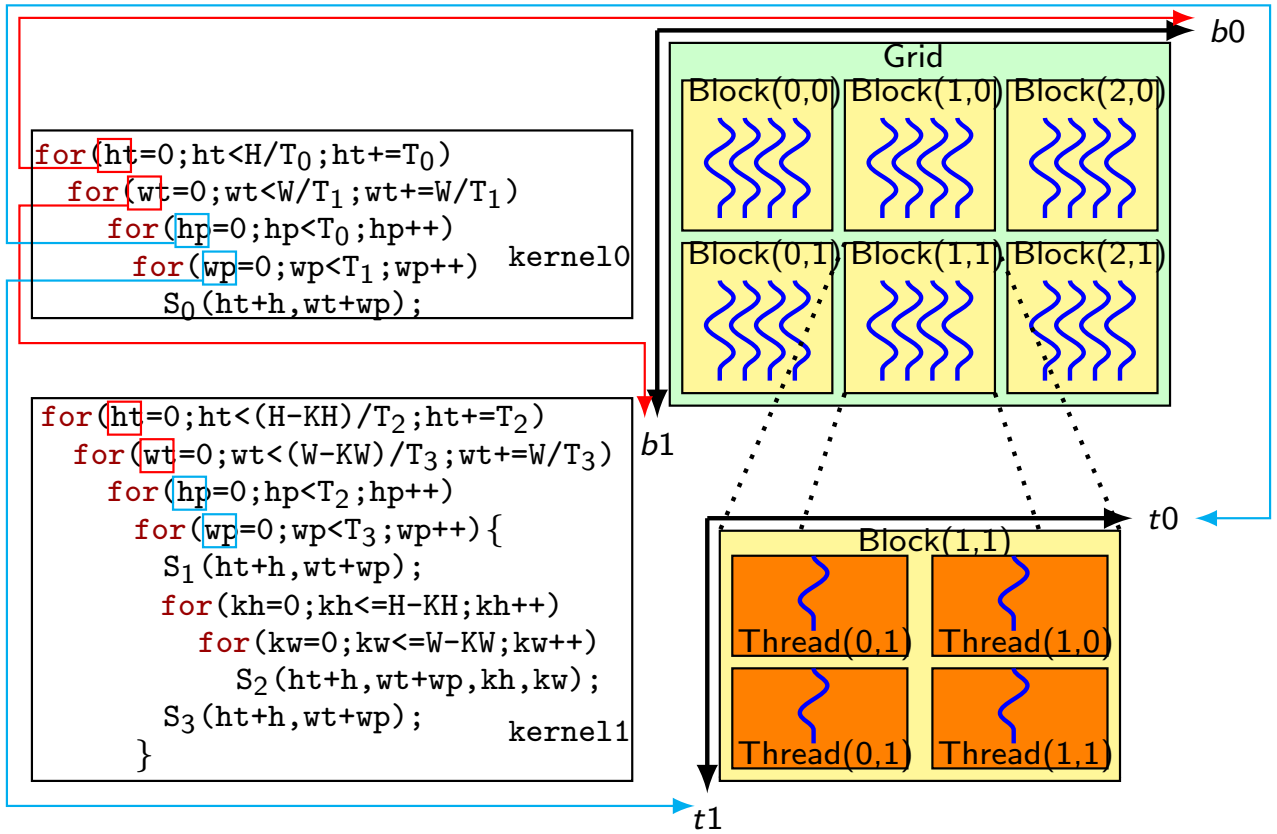
The new schedule implies a fusion strategy $(\{S_0\}, \{S_1, S_2, S_3\})$, and optimizing compilers can apply tiling on the generated code.



One can map code to GPU thread blocks and threads.

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.



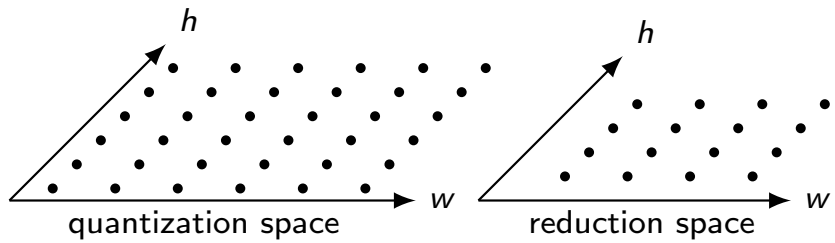
One can map code to GPU thread blocks and threads.

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

Conflicts in the data space

```
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);
      }
```

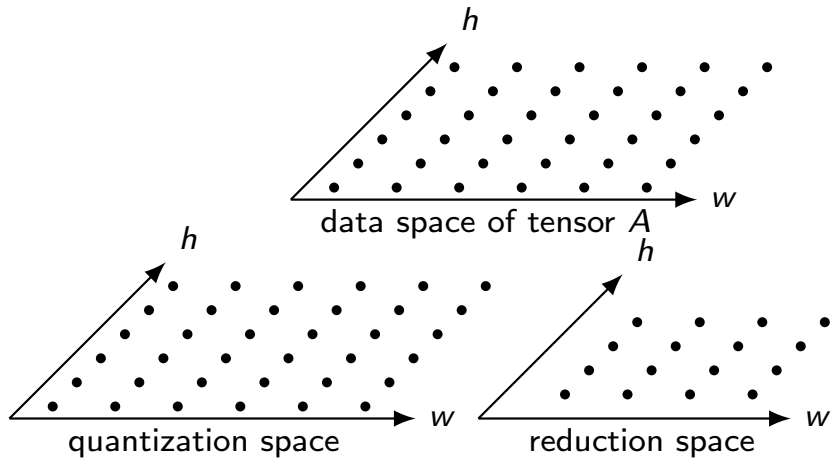


Let us first construct the (h, w) computation spaces for each fusion group, quantization for the first and reduction for the second.

Conflicts in the data space

```
for(ht=0;ht<H/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);
      }
```

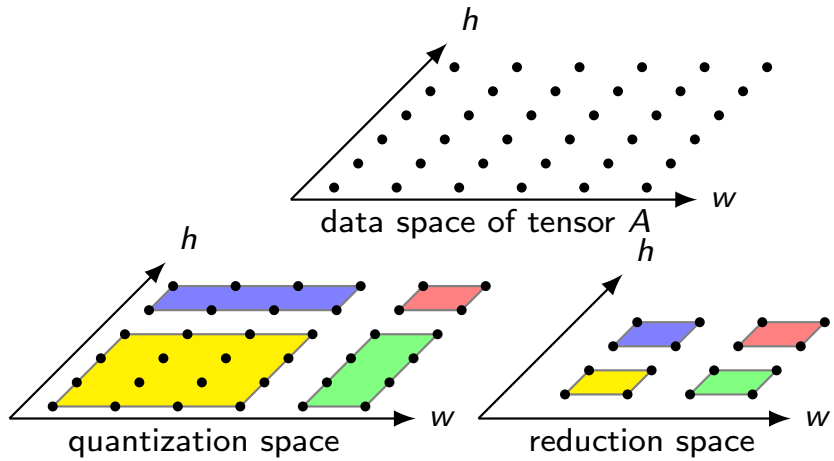


One can construct the data space of tensor A that is written by S_0 in the quantization group and read by S_2 in the reduction group.

Conflicts in the data space

```
for(ht=0;ht<H/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);
      }
```

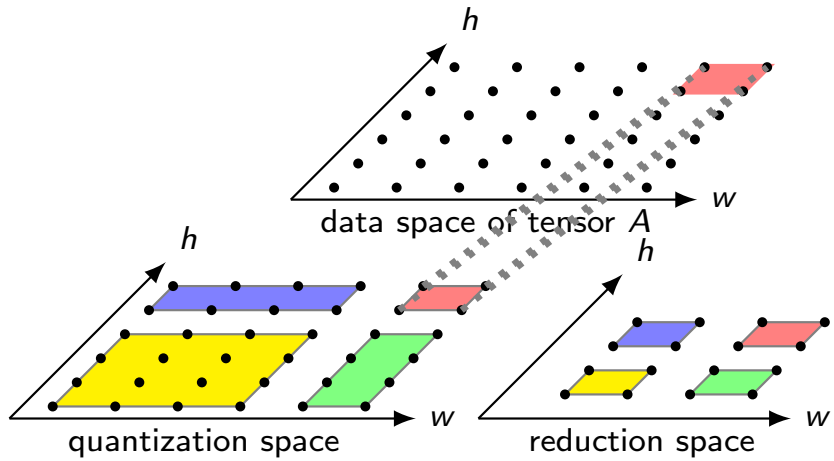


Existing polyhedral compilers tile each computation space individually, with tile sizes $T_0 = T_1 = 4$, $T_2 = T_3 = 2$.

Conflicts in the data space

```
for(ht=0;ht<H/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);
      }
```

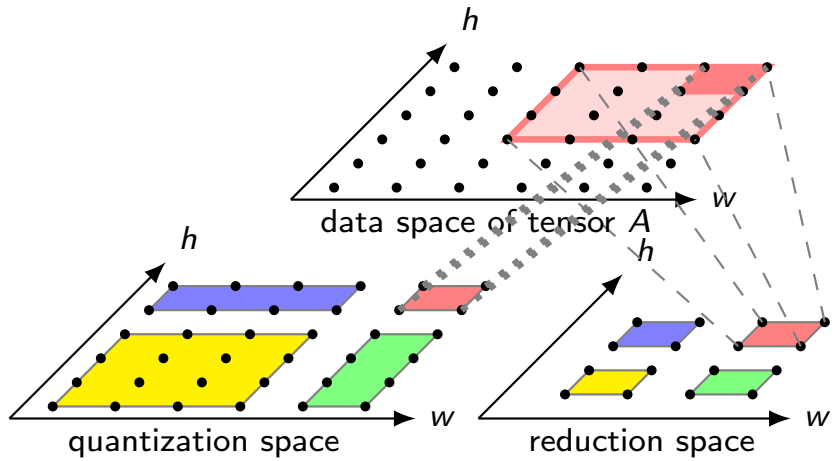


A tile of the quantization space writes to 4 points into the data space of tensor A.

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++){
        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);
      }
```

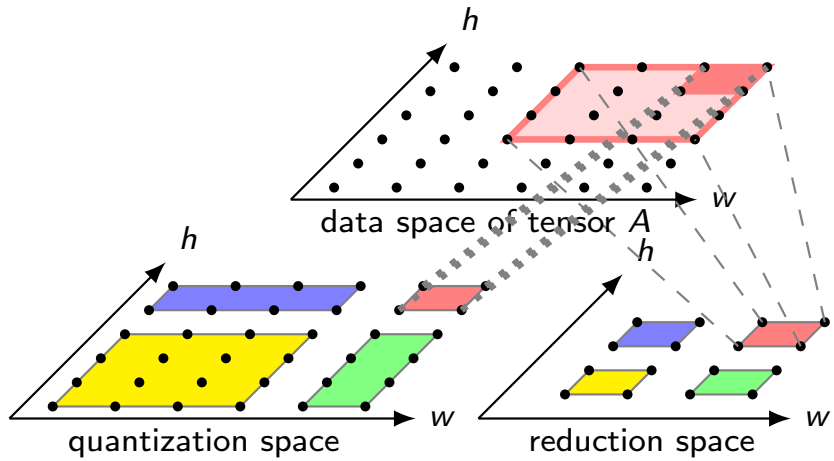


A tile of the reduction space requires 16 points from the data space of tensor A.

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++){
        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);
      }
```

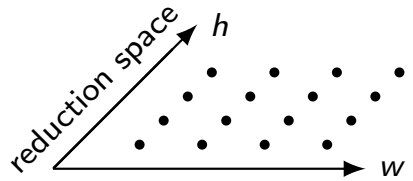


There exists a conflict in the data space of tensor A.

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++){
        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);
      }
```

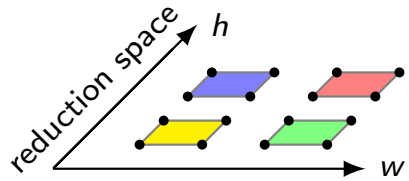


We can first apply tiling only to the reduction space.

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++){
        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);
      }
```

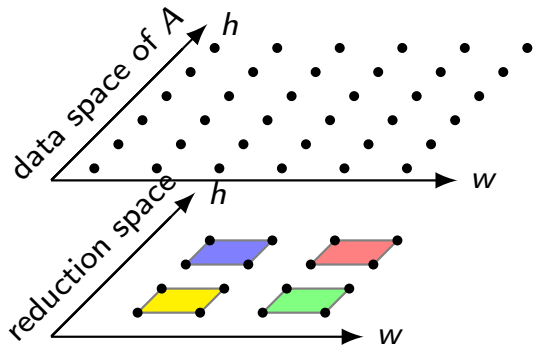


And we only tile the reduction space. This can tighten the tile size space and thus reduce compilation time.

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++){
        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);
      }
```

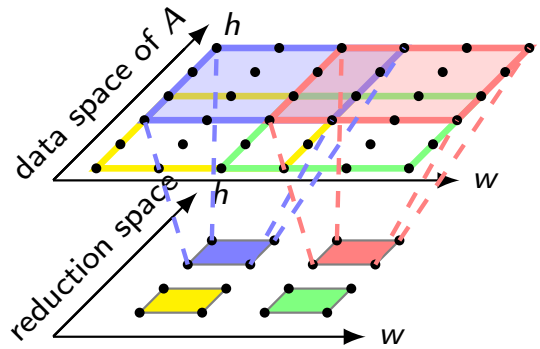


Next we construct the data space of tensor A.

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++){
        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);
      }
```

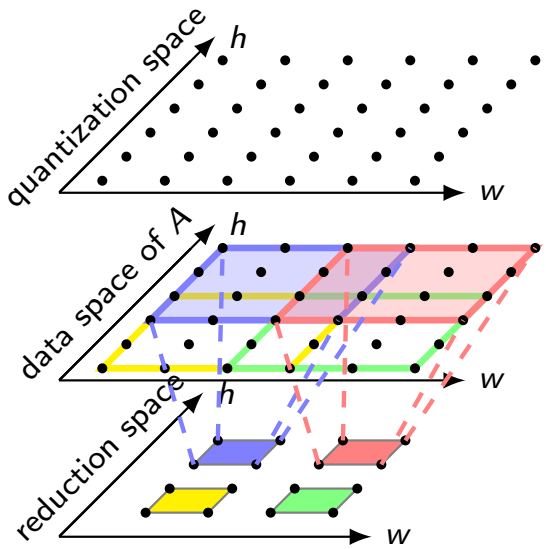


One can compute the data tiles required by each of those computation tile of the reduction space. Let us focus on the blue and red tiles.

Conflicts in the data space

```
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);
      }
```

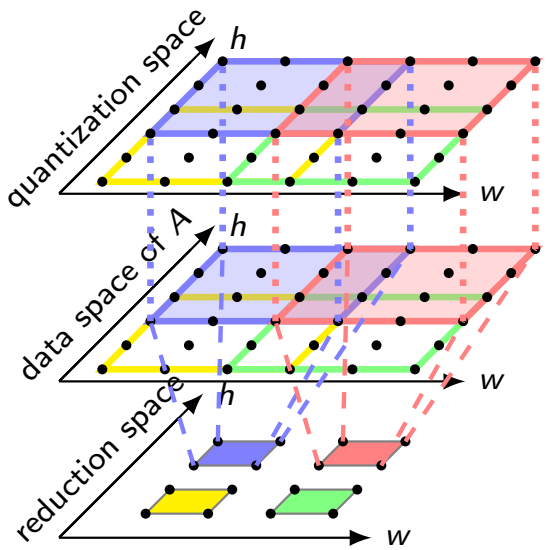


Now we can think about the tiling of the quantization space.

Conflicts in the data space

```
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);
      }
```

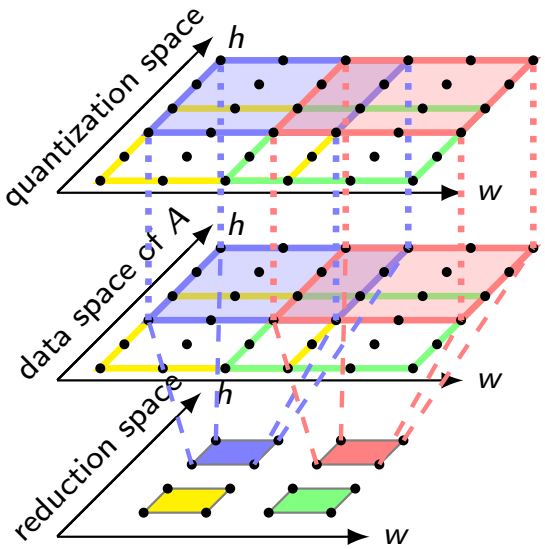


Polyhedral compilers can infer the tile shapes of the quantization space using the reverse of read access relation between S_0 and tensor A.

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++){
        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);
      }
```



Our algorithm determines the tile shapes of intermediate computation spaces using the reverse of access relations, which was impossible in existing work.

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.

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 abitrary tile shapes for intermediate computation spaces.
- can be used to handle more general applications like image processing pipelines, SpMV, linear algbra.

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 generates

- A tiling schedule for the reduction space:

$$\left[\{S_1(h,w) \to (\frac{h}{T_2}, \frac{w}{T_3}, h, w); S_2(h,w,kh,kw) \to (\frac{h}{T_2}, \frac{w}{T_3}, h, w, kh, kw); S_3(h,w) \to (\frac{h}{T_2}, \frac{w}{T_3}, h, w)\}\right] \tag{1}$$

- An extension schedule for the quantization space:

$$\begin{aligned}\{(o_0, o_1) \to S_0(h,w) : 0 \le o_0 < \lceil (H - KH + 1)/T_2 \rceil \wedge 0 \le o_1 < \lceil (W - KW + 1)/T_3 \rceil \\ \wedge T_2 \cdot o_0 \le h < T_2 \cdot o_0 + KH + T_2 - 1 \wedge T_3 \cdot o_1 \le w < T_3 \cdot o_1 + KW + T_3 - 1\}\end{aligned} \tag{2}$$

Schedule trees

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

Schedule trees

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

original schedule (textual execution order, affine maps):

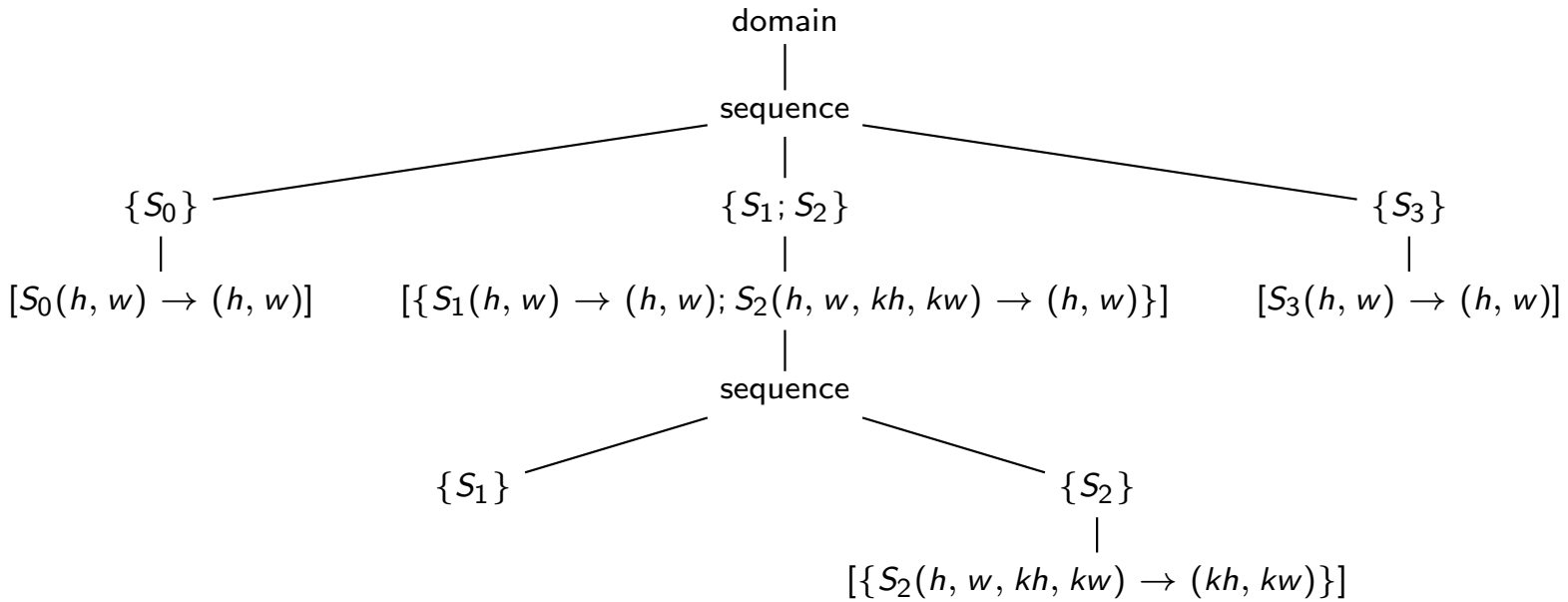
$[S_0(h, w) \to (0, h, w); S_1(h, w) \to (1, h, w, 0); S_2(h, w, kh, kw) \to (1, h, w, 1, kh, kw); S_3(h, w) \to (2, h, w)]$

Schedule trees

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

original schedule (textual execution order, affine maps):

$[S_0(h, w) \to (0, h, w); S_1(h, w) \to (1, h, w, 0); S_2(h, w, kh, kw) \to (1, h, w, 1, kh, kw); S_3(h, w) \to (2, h, w)]$



Schedule trees

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

new schedule (new execution order, affine maps):

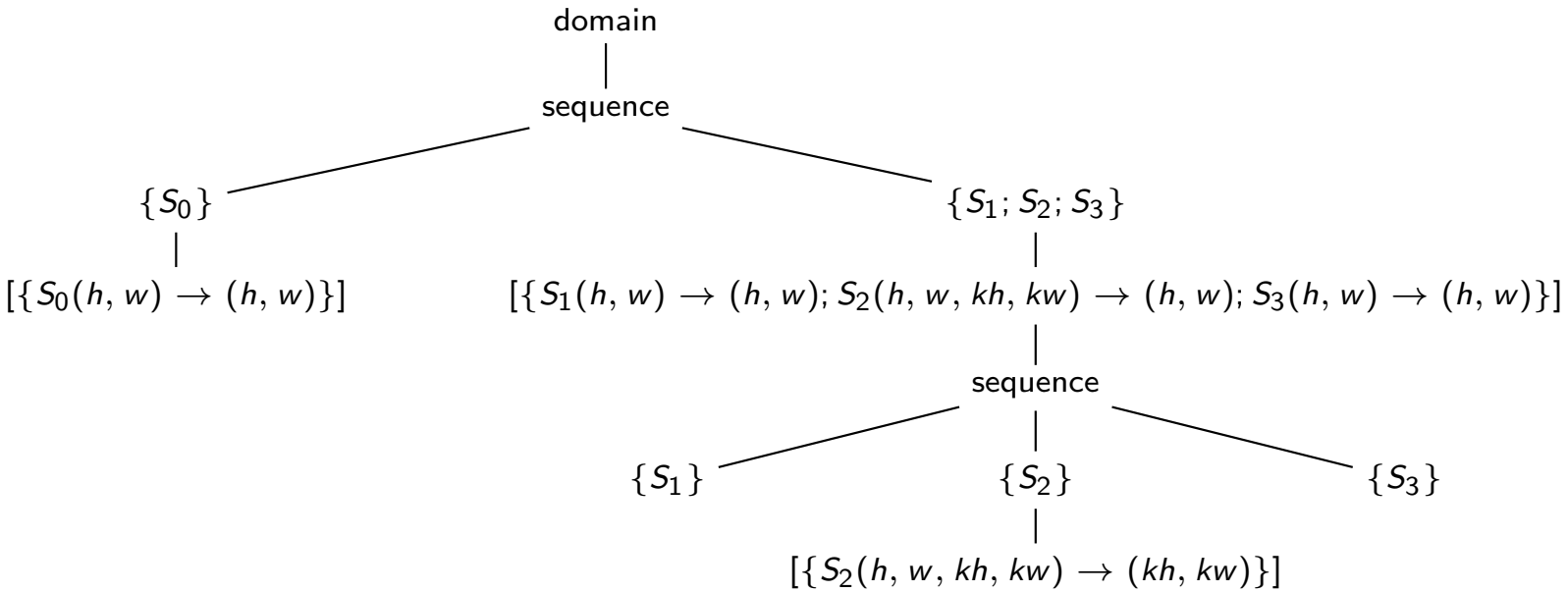
$[S_0(h, w) \rightarrow (0, h, w); S_1(h, w) \rightarrow (1, h, w, 0, 0, 0); S_2(h, w, kh, kw) \rightarrow (1, h, w, kh, kw, 1); S_3(h, w) \rightarrow (1, h, w, KH - 1, KW - 1, 2)]$

Schedule trees

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

new schedule (new execution order, affine maps):

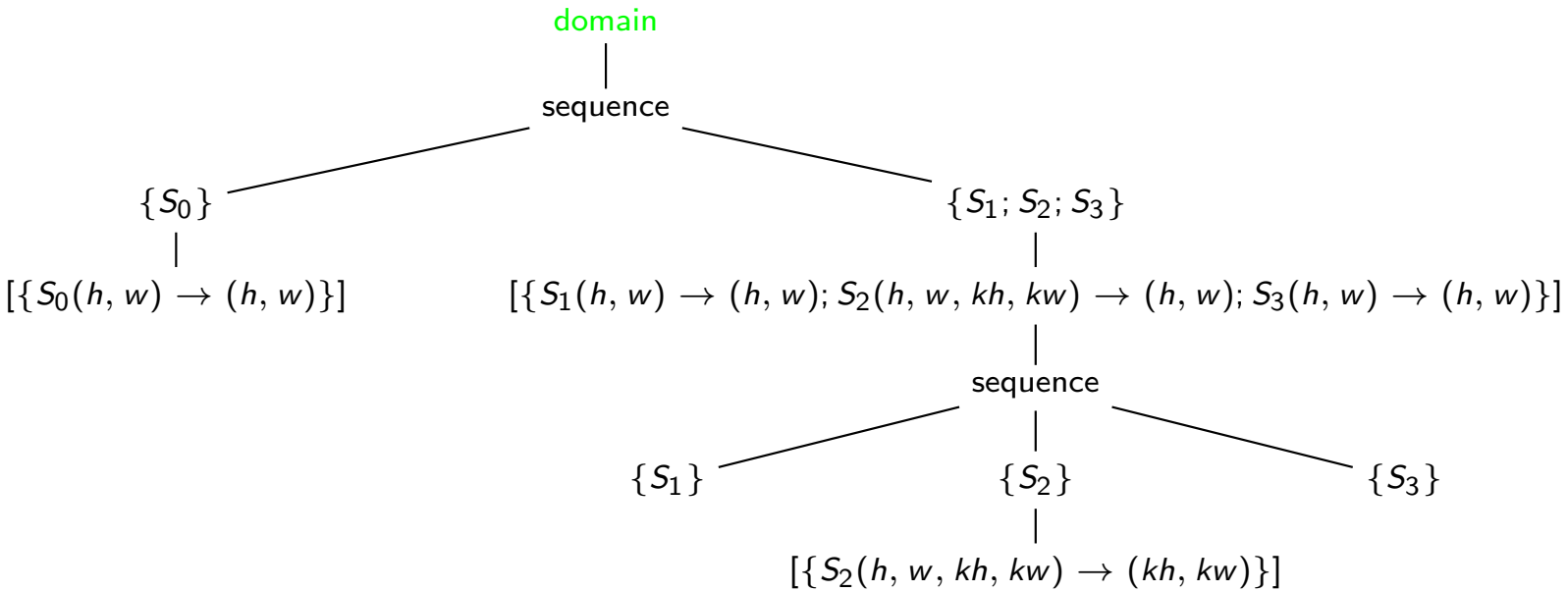
$[S_0(h, w) \rightarrow (0, h, w); S_1(h, w) \rightarrow (1, h, w, 0, 0, 0); S_2(h, w, kh, kw) \rightarrow (1, h, w, kh, kw, 1); S_3(h, w) \rightarrow (1, h, w, KH - 1, KW - 1, 2)]$



Schedule trees

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

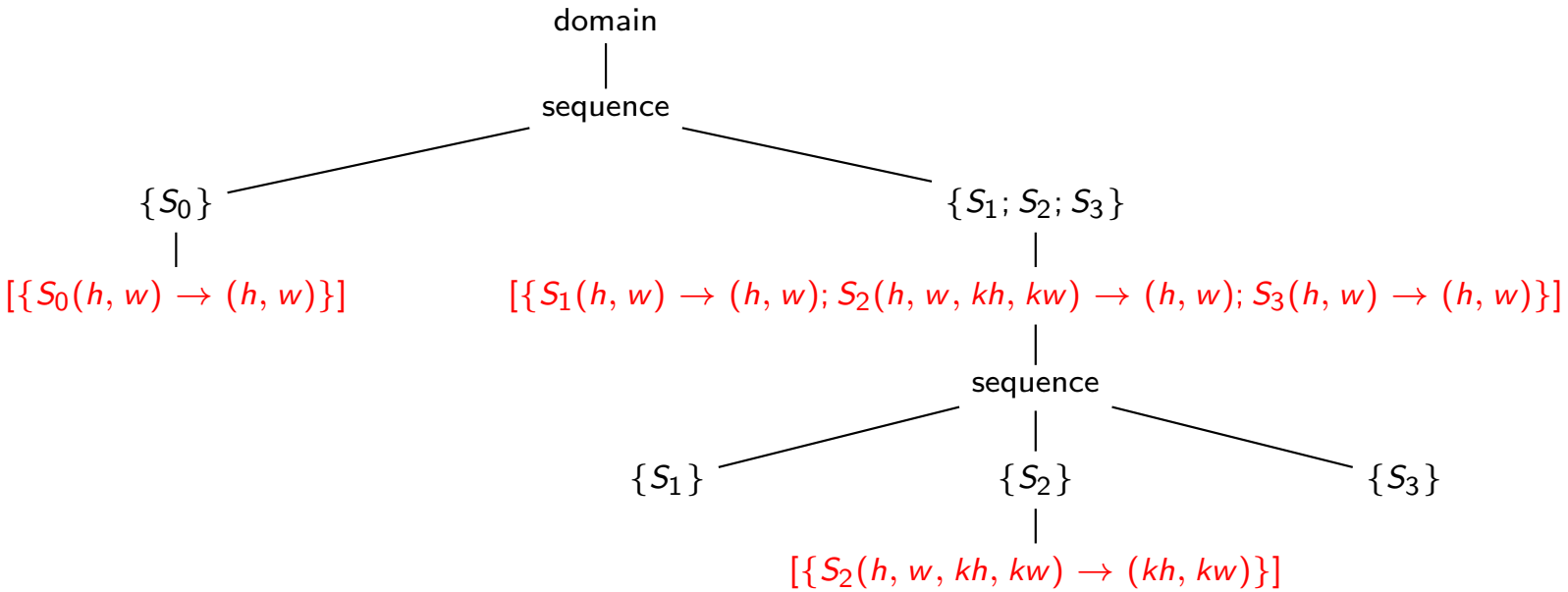
- Domain: set of statement instances to be scheduled
- Band: multi-dimensional piecewise quasi-affine partial schedule
- Filter: selects statement instances that are executed by descendants
- Sequence/Set: children executed in given/arbitrary order



Schedule trees

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

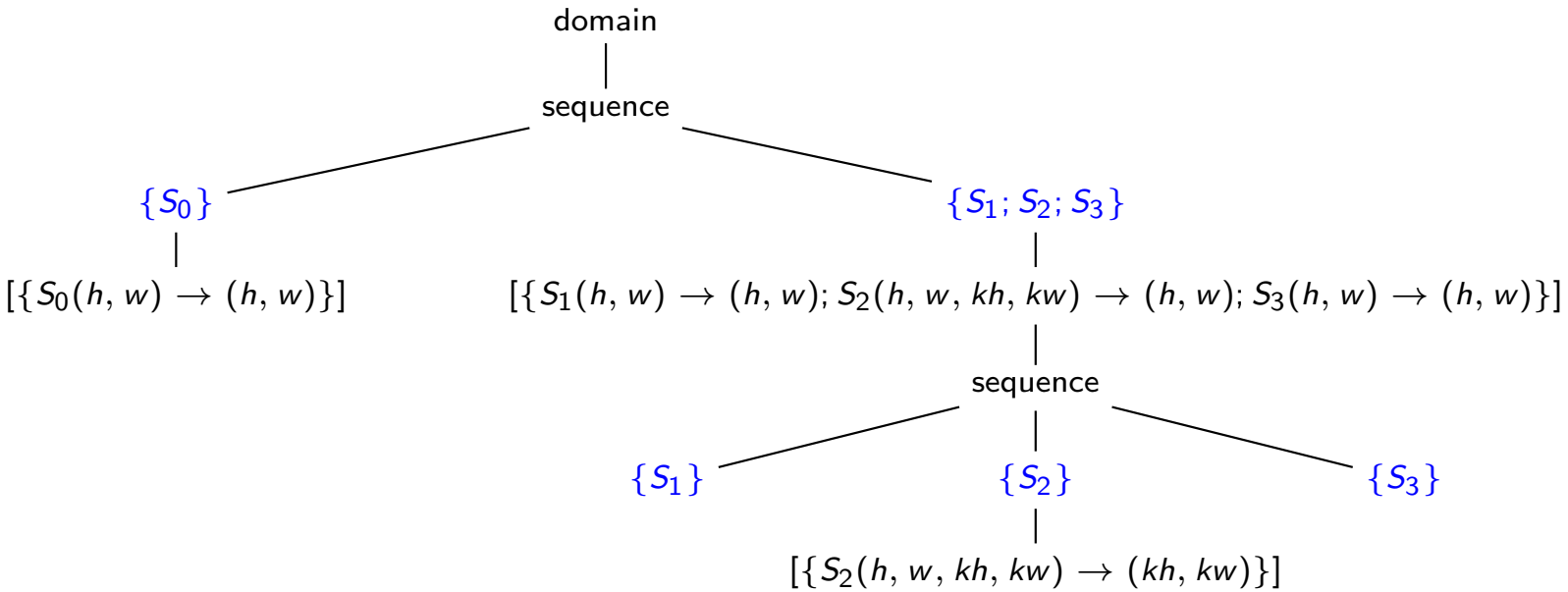
- Domain: set of statement instances to be scheduled
- Band: multi-dimensional piecewise quasi-affine partial schedule
- Filter: selects statement instances that are executed by descendants
- Sequence/Set: children executed in given/arbitrary order



Schedule trees

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

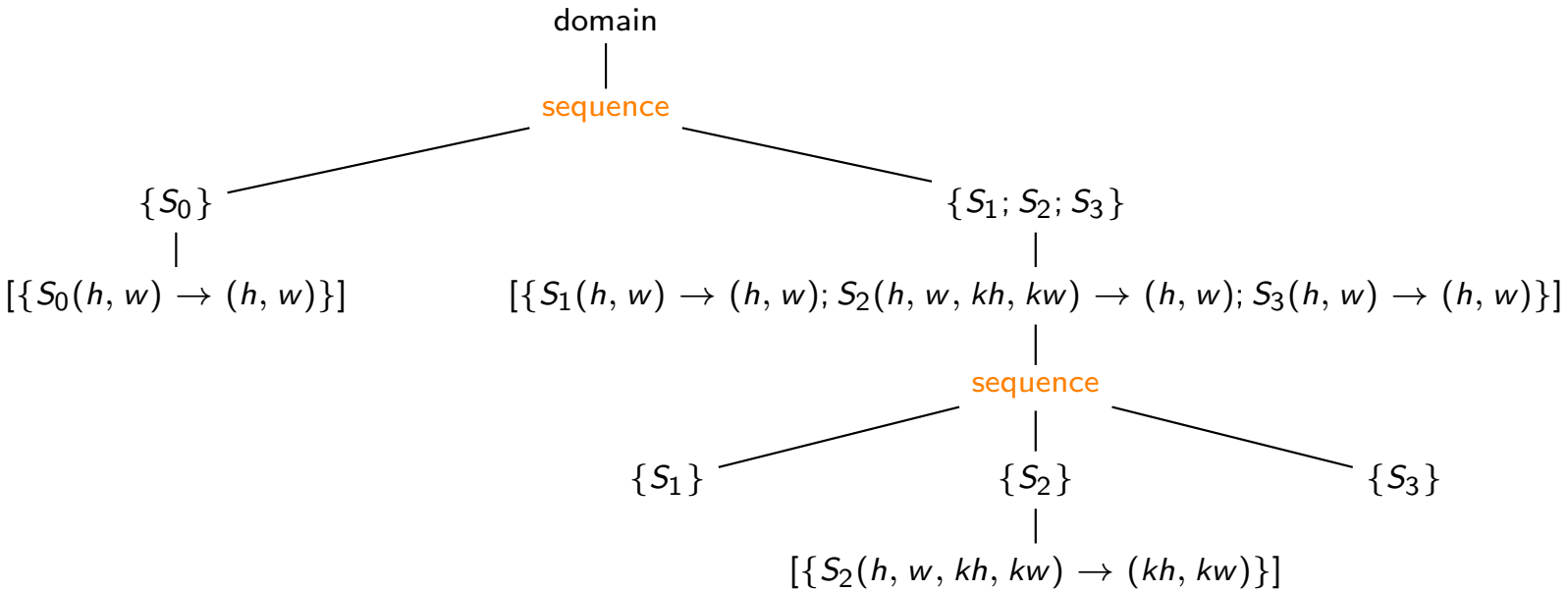
- Domain: set of statement instances to be scheduled
- Band: multi-dimensional piecewise quasi-affine partial schedule
- Filter: selects statement instances that are executed by descendants
- Sequence/Set: children executed in given/arbitrary order



Schedule trees

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

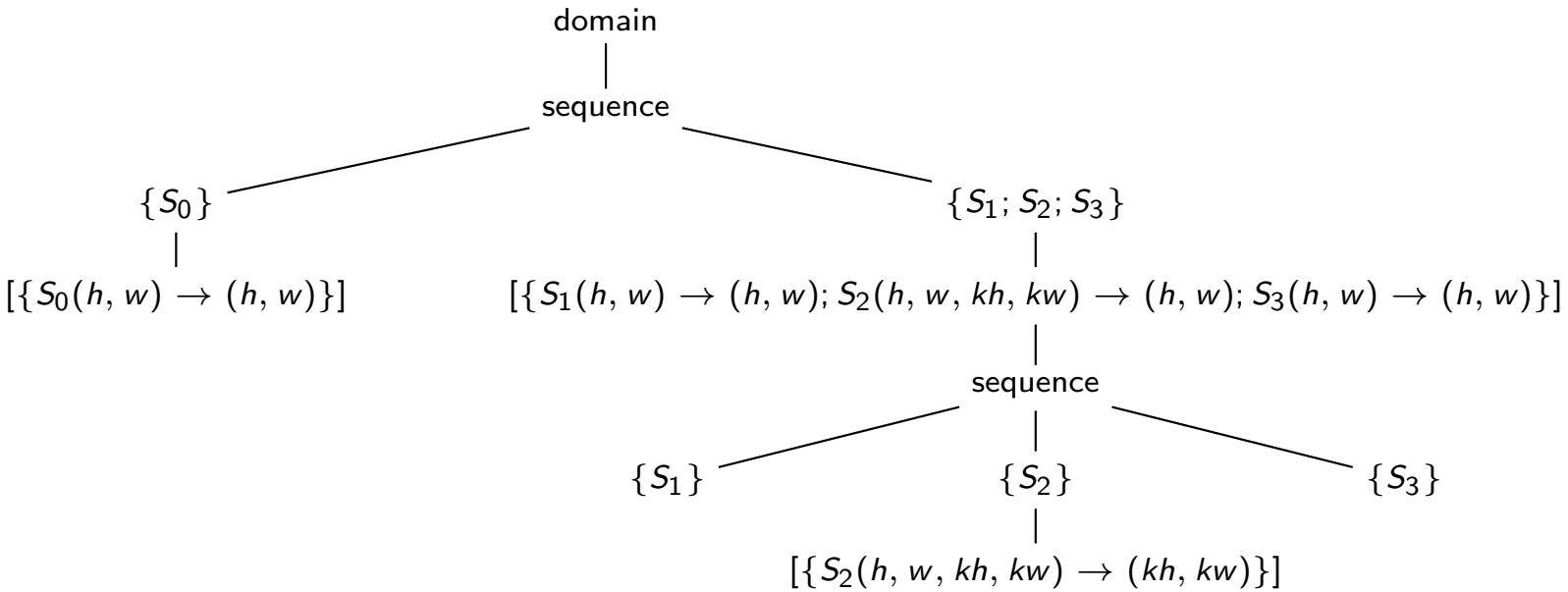


Schedule trees

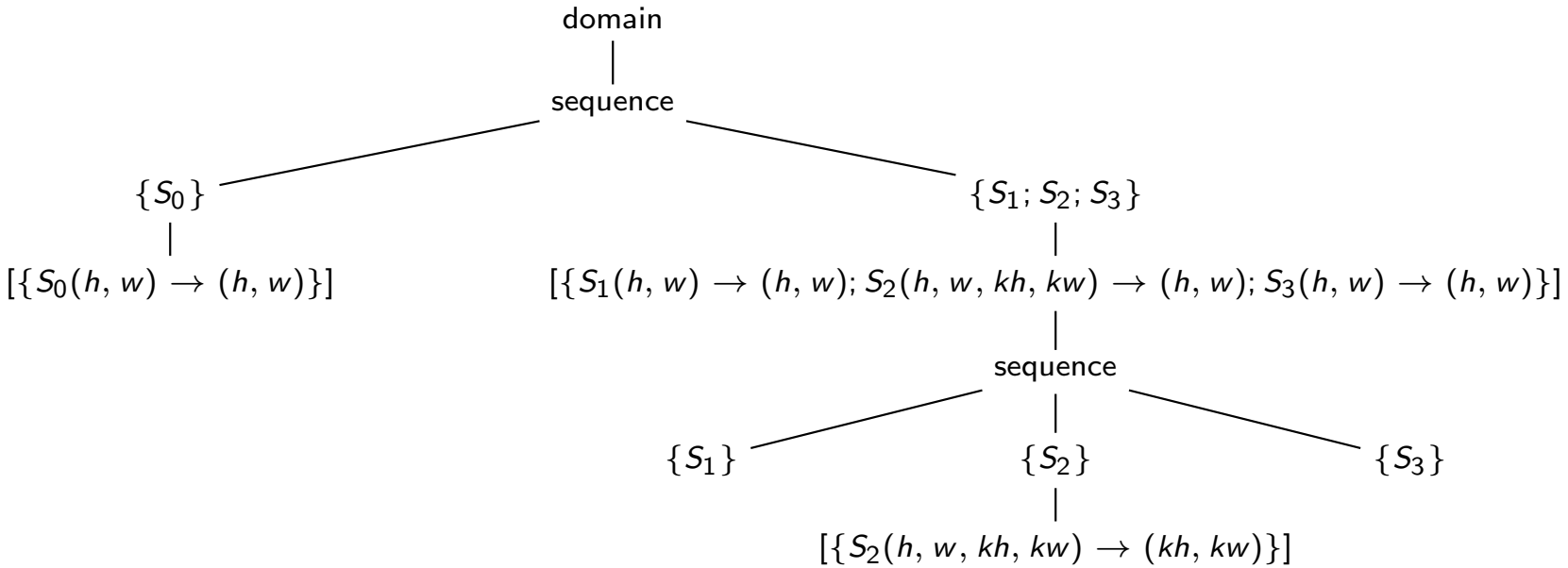
The polyhedral model can also represent a program and its semantics using schedule trees [6].

Schedule trees also provide convenience node types:

- Mark: attach additional information to subtrees
- Extension: add additional domain elements to facilitate non-polyhedral semantics

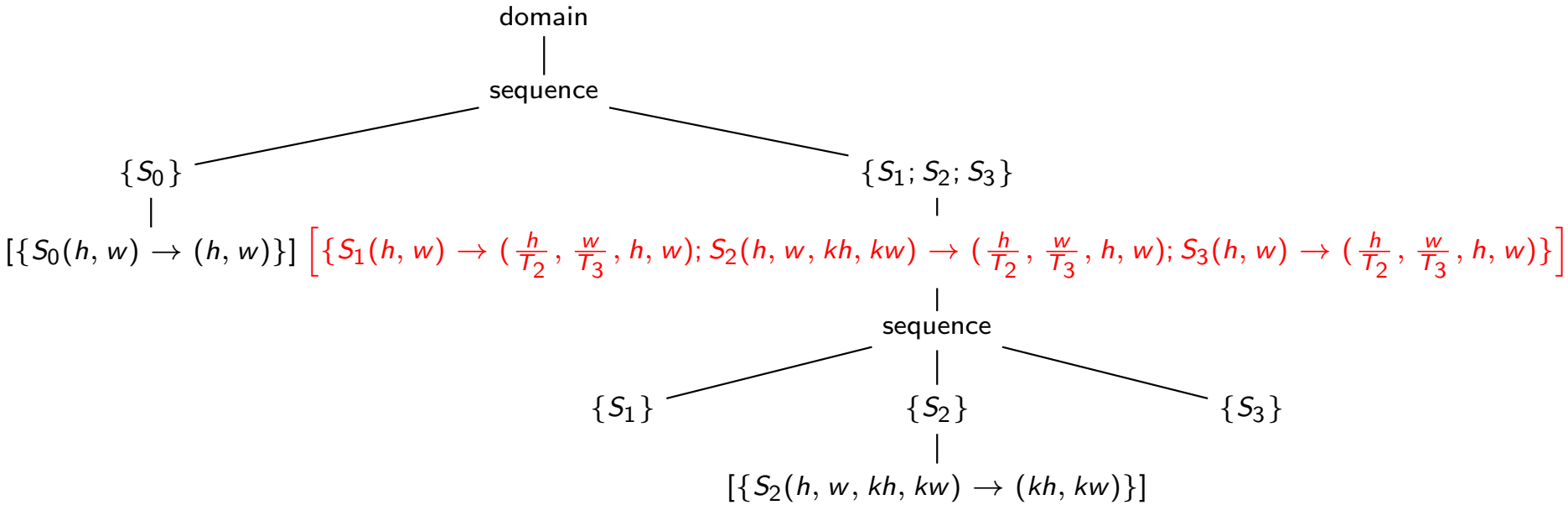


Manipulations on schedule trees



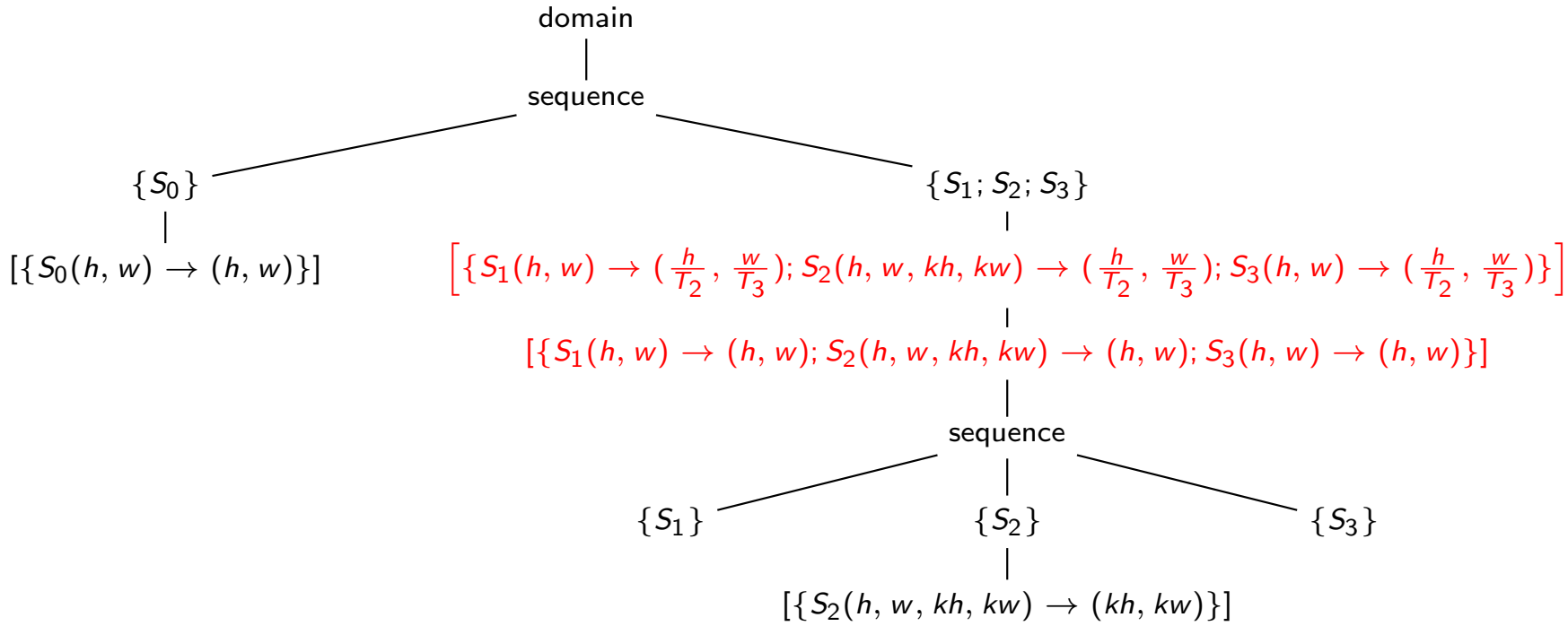
We can implement post-tiling fusion by manipulating a schedule tree obtained after applying the Pluto-like schedulers [3].

Manipulations on schedule trees



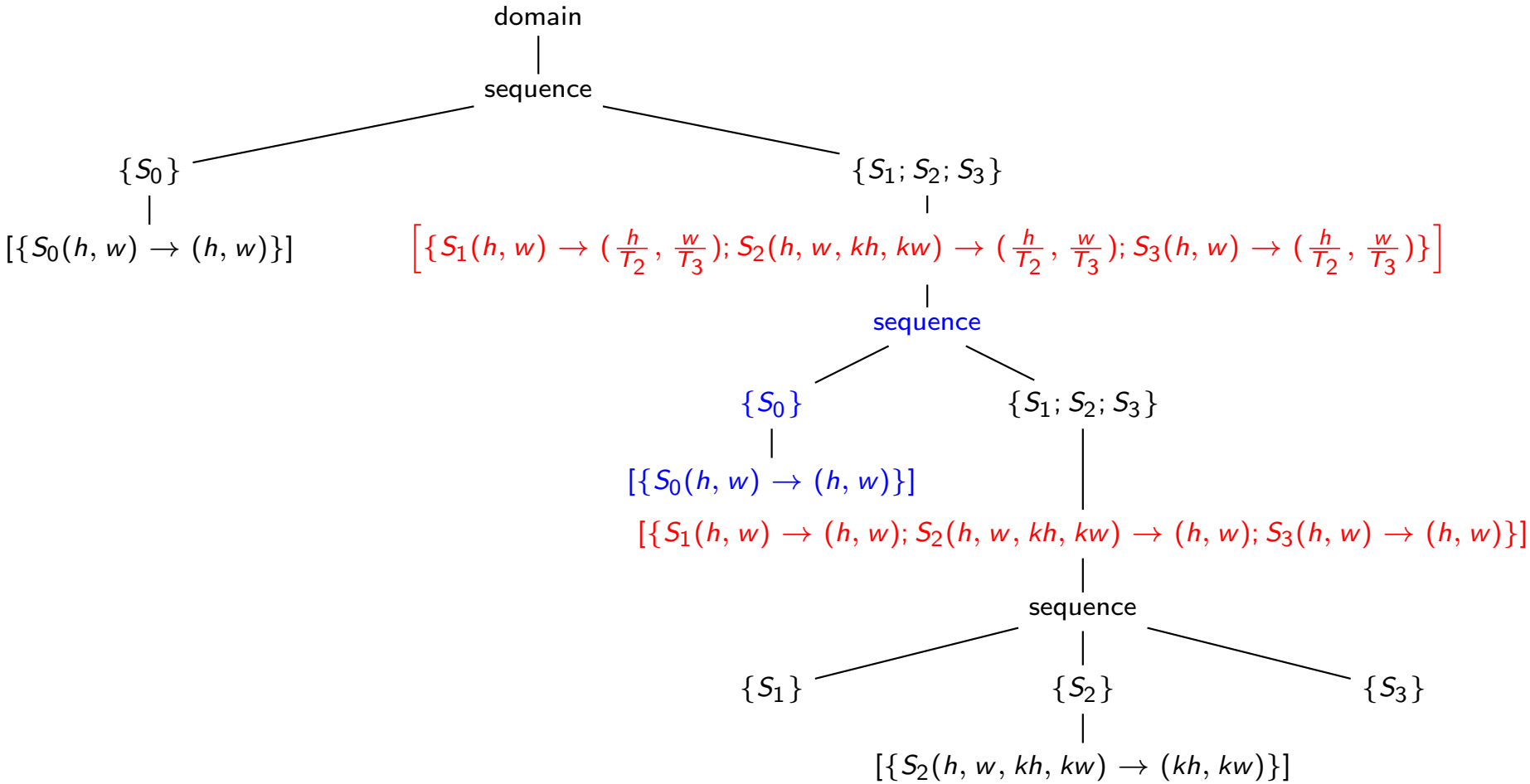
Classical rectangular/parallelogram tiling can be applied using the tiling schedule (1) on page 6, with $T_2 \times T_3$ the tile sizes along $h \times w$ dimensions.

Manipulations on schedule trees



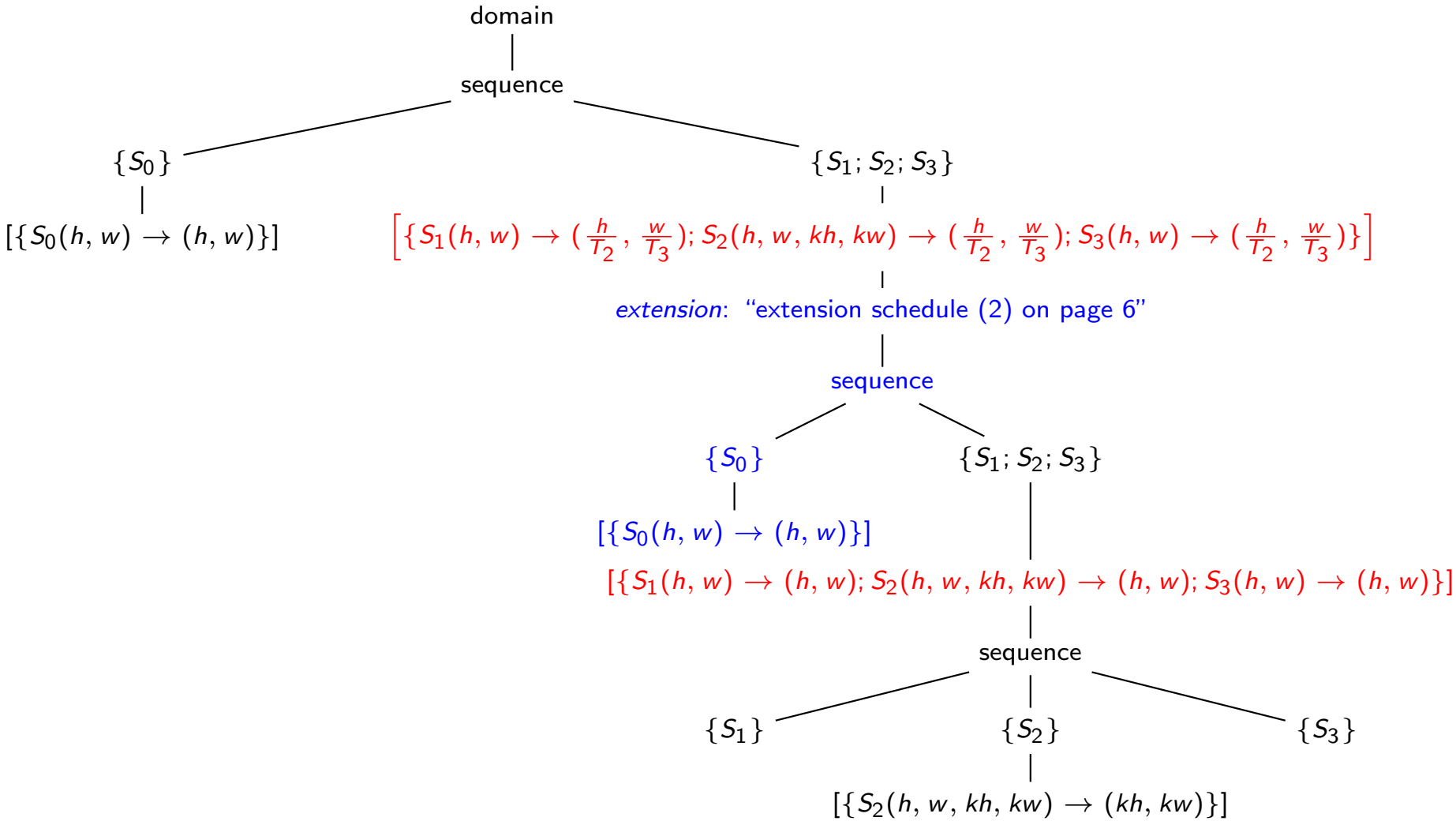
Split the band node that has implemented rectangular/parallelogram tiling for post-tiling fusion.

Manipulations on schedule trees



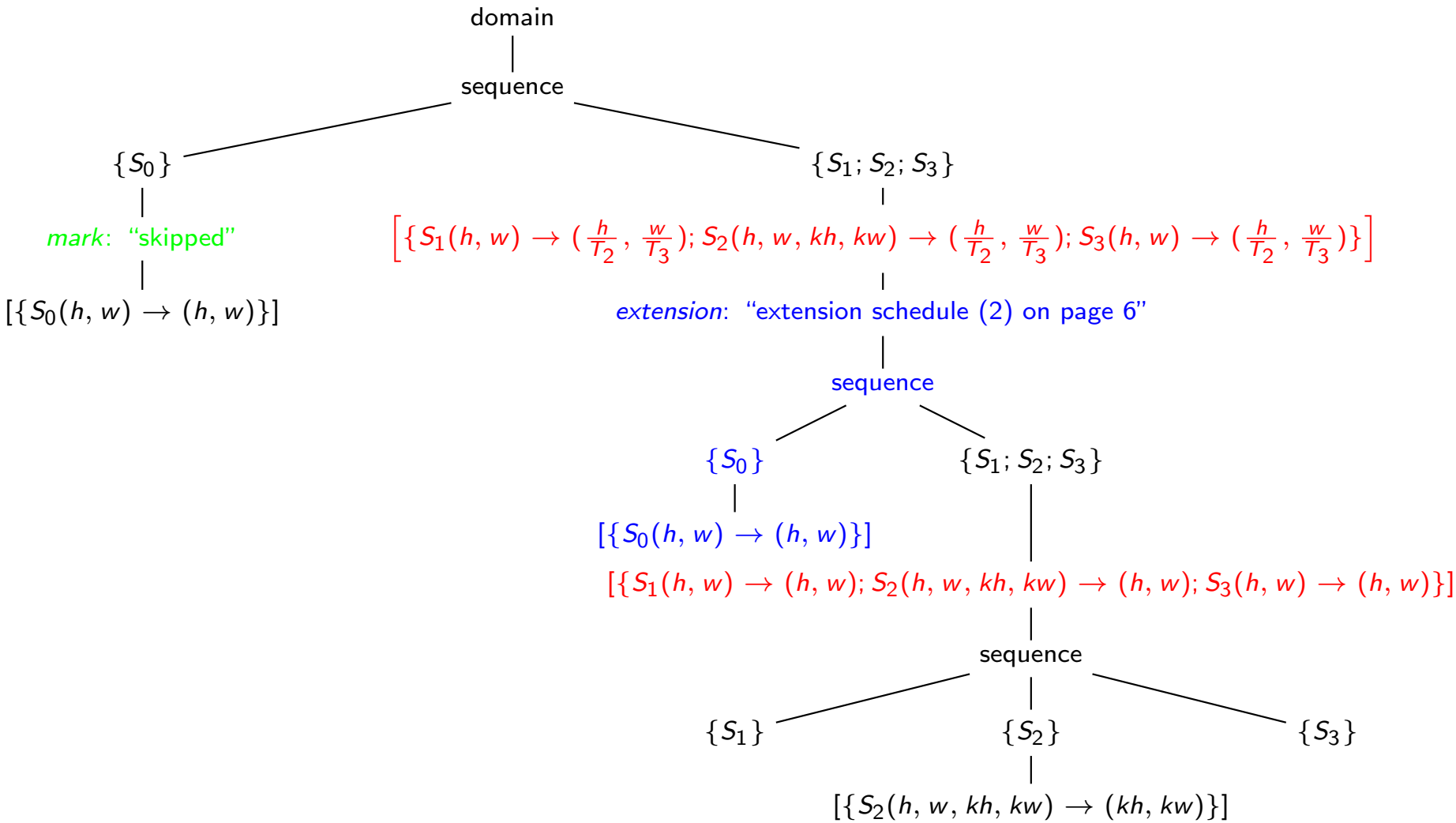
The subtree of S_0 is introduced with a sequence node indicating the order with its siblings.

Manipulations on schedule trees



An expansion node is mandatory to introduced additional statements, i.e., the subtree of S_0.

Manipulations on schedule trees



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.

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

- 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.

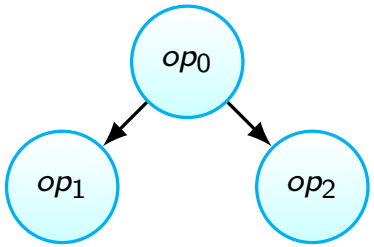
The post-tiling fusion algorithm

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.

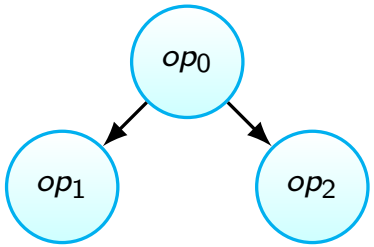
Our post-tiling algorithm returns a fusion strategy of $(\{S_0, S_1, S_2, S_3\})$, fusing all statements into a single loop nest without hampering the parallelism.

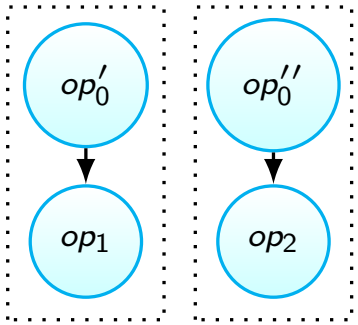
Generalizing the approach



- 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.

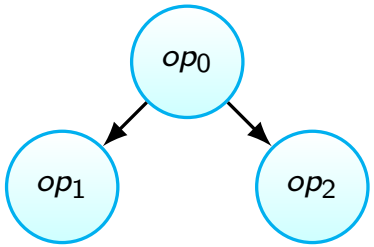
Generalizing the approach

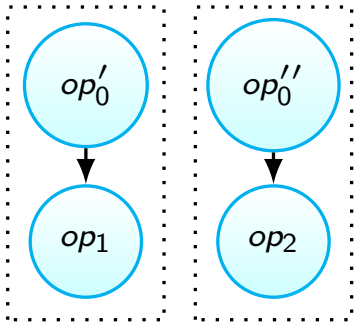




- We have to handle the scene that the values defined by an intermediate space op_0 are used by multiple live-out spaces op_1 and op_2.
- op_0' represents the subset of op_0 that computes the values used by op_1, and op_0'' the subset that writes to the values read by op_2.

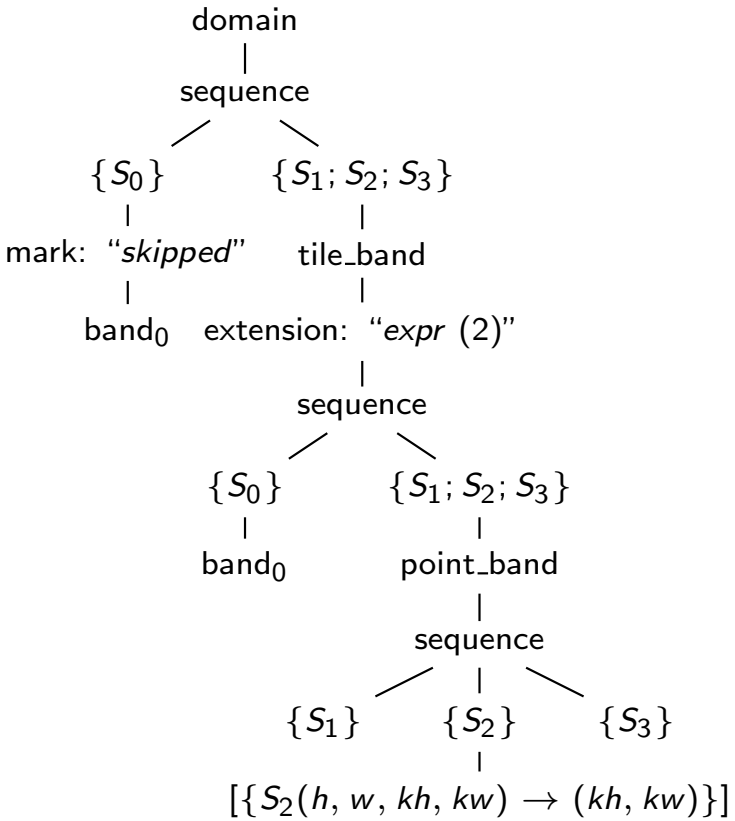
Generalizing the approach





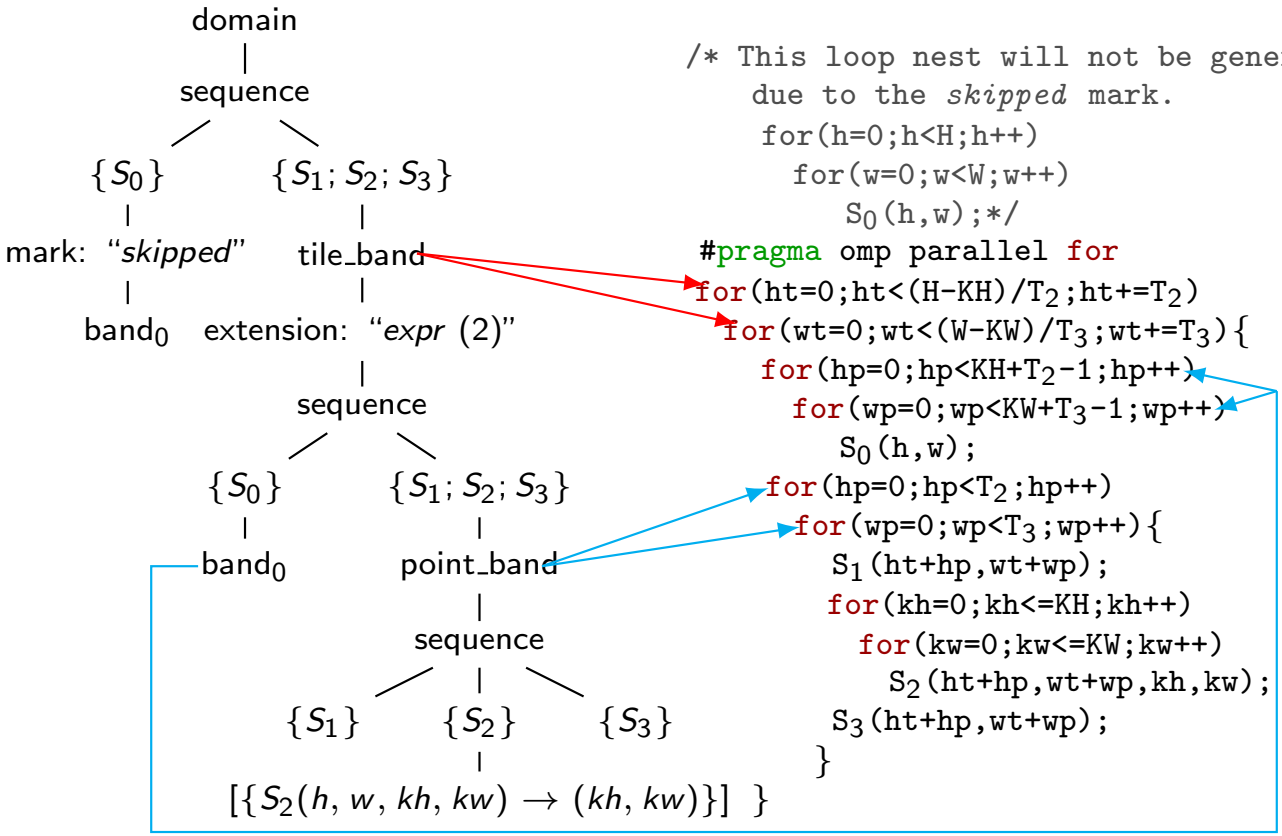
- Unlike existing heuristics [8, 11], op_0' and op_0'' can be fused with their uses, respectively, without introducing redundant computations.
- Otherwise, fusion is prevented due to possible redundancy.
- This strategy also implements dead code elimination in some extreme cases that was not considered by existing polyhedral optimizers [3, 5, 18].
- See Algorithm 3 in the paper for detailed explanation.

Code generation



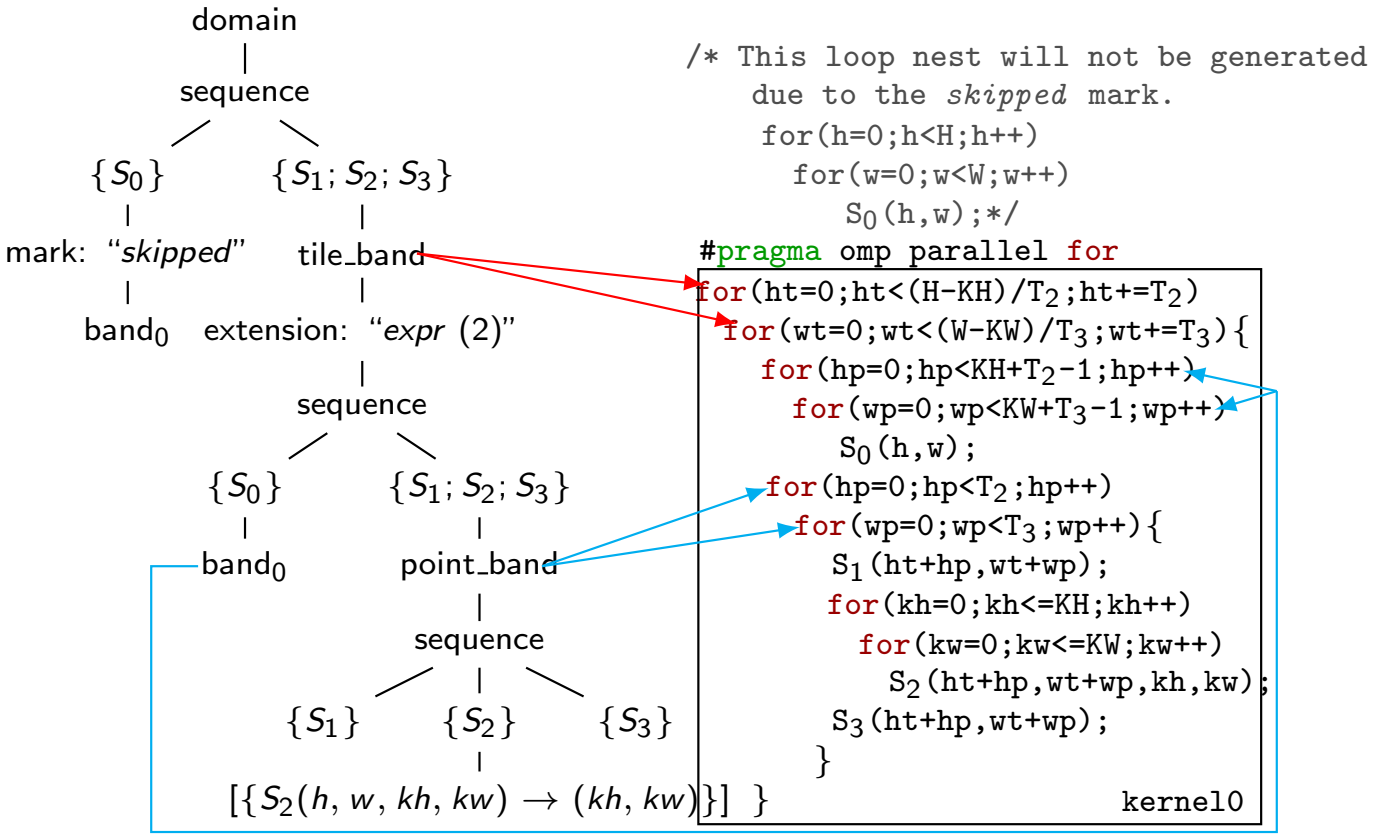
Code generation for CPUs and GPUs is implemented in PPCG [19], a polyhedral compiler using the *isl* library [18] as the solver.

Code generation



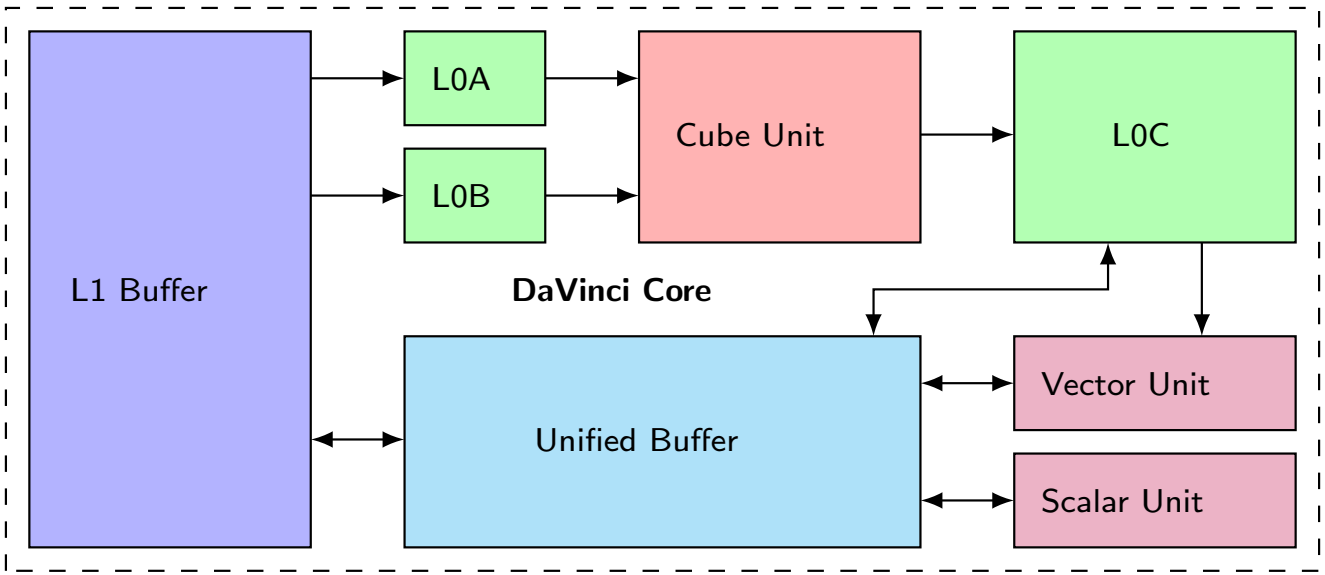
Code generation for CPUs and GPUs is implemented in PPCG [19], a polyhedral compiler using the *isl* library [18] as the solver.

Code generation



Code generation for CPUs and GPUs is implemented in PPCG [19], a polyhedral compiler using the *isl* library [18] as the solver.

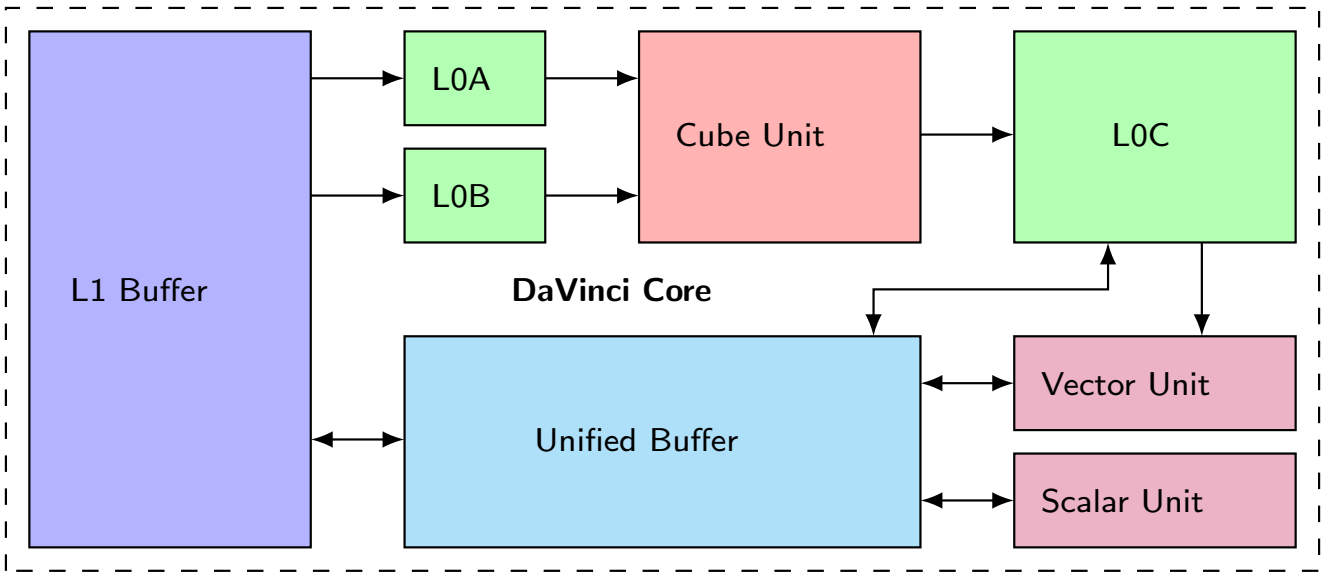
Code generation



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

Code generation



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

Benchmark	stages	Tile size parameter	CPU execution time (ms) naïve (1 core)	PolyMage (32 cores)	Halide (32 cores)	Our work (32 cores)
Bilateral Grid	7	8×64	66.01	5.57	4.23	4.11
Camera Pipeline	32	16×32	116.32	4.68	4.76	4.40
Harris Corner Detection	11	16×32	246.88	5.10	10.71	5.10
Local Laplacian Filter	99	8×64	480.48	35.35	29.12	27.08
Multiscale Interpolation	49	32×16	209.10	16.44	20.07	14.87
Unsharp Mask	4	8×32×3	142.16	5.01	5.02	3.68

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

Performance of the PolyMage benchmarks

Benchmark	stages	GPU grid	GPU execution time (ms)			Compilation time (s)			
		parameter	PPCG (minfuse)	Halide	Our work	minfuse	smartfuse	maxfuse	Our work
Bilateral Grid	7	8×64	5.07	3.79	4.09	0.15	120	$>24h$	0.86
Camera Pipeline	32	16×32	3.51	2.47	2.38	18.3	20.9	$>24h$	4560
Harris Corner Detection	11	16×32	1.79	1.68	1.60	0.03	0.06	0.12	435
Local Laplacian Filter	99	8×64	16.73	12.53	11.12	6.94	90.8	$>24h$	89.3
Multiscale Interpolation	49	32×16	15.75	25.65	13.37	0.68	1.40	$>24h$	3.30
Unsharp Mask	4	$8\times32\times3$	2.03	1.94	2.01	0.06	0.08	0.10	0.05

- Our work provides 20% and 33% improvements over PolyMage [12] and Halide [15] when targeting CPUs.
- Our approach outperfoms 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.

Performance of the Resnet-50 workload

	Execution time (ms)			Compilation time (s)	
	smart	Our work	Speedup	smart	Our work
fwd conv+batchnorm	11.50	6.69	1.72×	-	-
entire workload	35.03	30.25	1.16×	736	487

- 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

	Execution time (ms)			Compilation time (s)	
	smart	Our work	Speedup	smart	Our work
fwd conv+batchnorm	11.50	6.69	1.72×	-	-
entire workload	35.03	30.25	1.16×	736	487

- 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.

Conclusion

- Our tiling algorithm can construct arbitrary tile shapes, without refining scheduling algorithms [12] or resorting to complicated constraints [20].

Conclusion

- 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.

Conclusion

- 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].

Conclusion

- 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

[1] Acharya, A., Bondhugula, U., and Cohen, A.
Polyhedral auto-transformation with no integer linear programming.
In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2018), PLDI 2018, ACM, pp. 529–542.

[2] Bao, H., Bielak, J., Ghattas, O., Kallivokas, L. F., O'Hallaron, D. R., Shewchuk, J. R., and Xu, J.
Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers.
Computer Methods in Applied Mechanics and Engineering 152, 1 (1998), 85 – 102.
Containing papers presented at the Symposium on Advances in Computational Mechanics.

[3] Bondhugula, U., Hartono, A., Ramanujam, J., and Sadayappan, P.
A practical automatic polyhedral parallelizer and locality optimizer.
In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2008), PLDI '08, ACM, pp. 101–113.

[4] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A.
Tvm: An automated end-to-end optimizing compiler for deep learning.
In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2018), OSDI'18, USENIX Association, pp. 579–594.

[5] Grosser, T., Groesslinger, A., and Lengauer, C.
Polly–performing polyhedral optimizations on a low-level intermediate representation.
Parallel Processing Letters 22, 04 (2012), 1250010.

[6] Grosser, T., Verdoolaege, S., and Cohen, A.
Polyhedral ast generation is more than scanning polyhedra.
ACM Trans. Program. Lang. Syst. 37, 4 (July 2015), 12:1–12:50.

[7] He, K., Zhang, X., Ren, S., and Sun, J.
Deep residual learning for image recognition.
In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016), pp. 770–778.

References

[8] Jangda, A., and Bondhugula, U.
An effective fusion and tile size model for optimizing image processing pipelines.
In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2018), PPoPP '18, ACM, pp. 261–275.

[9] Karas, P., Morard, V., Bartovsky, J., Grandpierre, T., Dokladalova, E., Matula, P., and Dokladal, P.
Gpu implementation of linear morphological openings with arbitrary angle.
Journal of Real-Time Image Processing 10 (04 2012).

[10] Manegold, S.
Memory Hierarchy.
Springer New York, New York, NY, 2016, pp. 1–8.

[11] Mehta, S., Lin, P.-H., and Yew, P.-C.
Revisiting loop fusion in the polyhedral framework.
In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2014), PPoPP '14, ACM, pp. 233–246.

[12] Mullapudi, R. T., Vasista, V., and Bondhugula, U.
Polymage: Automatic optimization for image processing pipelines.
In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2015), ASPLOS '15, ACM, pp. 429–443.

[13] Mullapudi, R. T., Vasista, V., and Bondhugula, U.
Polymage benchmarks.
URL: https://github.com/bondhugula/polymage-benchmarks (commit d20264ef) (2017).

[14] Pouchet, L.-N., et al.
Polybench: The polyhedral benchmark suite.
URL: http://www. cs. ucla. edu/pouchet/software/polybench (2012).

References

[15] Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., and Amarasinghe, S.
Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines.
In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2013), PLDI '13, ACM, pp. 519–530.

[16] Upadrasta, R., and Cohen, A.
Sub-polyhedral scheduling using (unit-)two-variable-per-inequality polyhedra.
In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New York, NY, USA, 2013), POPL '13, ACM, pp. 483–496.

[17] Vasilache, N., Zinenko, O., Theodoridis, T., Goyal, P., Devito, Z., Moses, W. S., Verdoolaege, S., Adams, A., and Cohen, A.
The next 700 accelerated layers: From mathematical expressions of network computation graphs to accelerated gpu kernels, automatically.
ACM Trans. Archit. Code Optim. 16, 4 (Oct. 2019).

[18] Verdoolaege, S.
Isl: An integer set library for the polyhedral model.
In *Proceedings of the Third International Congress Conference on Mathematical Software* (Berlin, Heidelberg, 2010), ICMS'10, Springer-Verlag, pp. 299–302.

[19] Verdoolaege, S., Carlos Juega, J., Cohen, A., Ignacio Gómez, J., Tenllado, C., and Catthoor, F.
Polyhedral parallel code generation for cuda.
ACM Trans. Archit. Code Optim. 9, 4 (Jan. 2013), 54:1–54:23.

[20] Zhao, J., and Cohen, A.
Flextended tiles: A flexible extension of overlapped tiles for polyhedral compilation.
ACM Trans. Archit. Code Optim. 16, 4 (Dec. 2019).