PlaidML & Stripe

Model-guided Optimization & Polyhedral IR

Brian Retford

PlaidML: Tile DSL

Tensor DSLs

Compiler	Matrix Multiplication in Native DSL
PlaidML	`C[i, j: I, J] = +(A[i, k] * B[k, j]);`
(taco)	`c(i, j) = a(i,k) * b(k,j)`
TVM	`tvm.sum(a[i, k] * b[j, k], axis=k)`
Tensor Comprehensions	`C(i, j) +=! A(i, k) * B(k, j)`

Tile: Automatic Differentiation

… start with a dilated & strided convolution:

```
function (I[N, H, W, CI], K[KH, KW, CI, CO]) -> (O) {
    O[n, y, x, co: N, H/3, W/3, CO] =
      +(I[n, 3*y + 2*j, 3*x + 2*i, ci] * K[j, i, ci, co]);
  }
```

… DI/DO is obtained by swapping the input I and the output O:

```
function (DO[N, OH, OW, CO], K[KH, KW, CI, CO]) -> (DI) {
    DI[n, 3*y + 2*j, 3*x + 2*i, ci: N, 3*OH, 3*OW, CI] =
      +(DO[n, y, x, co] * K[j, i, ci, co]);
  }
```

PlaidML v0

i.e., the currently available one

PlaidML v0.x: Summary

- https://github.com/plaidml/plaidml
- Open source, Apache 2 (new), supports training & inference
- Reasonable community starting to build on GitHub, 1600 stars
- Supports most popular frameworks (except training via pyTorch) via upcoming nGraph integration
- Performance portable for major GPU architectures
    - Fixed Optimization passes, Minimal hardware config
    - Between .5-1.5x as fast as AutoTVM
- Not well suited for deep learning accelerators or other architectures that benefit from micro-kernels


plaidML

PlaidML v0: Optimization

Fixed passes, locally optimal, config driven

```
"settings": {
   "threads": 256,
   "vec_size": 1,
   "mem_width": 128,
   "max_mem": 32768,
   "max_regs": 16384,
   "goal_groups": 16,
   "goal_flops_per_byte": 50
}
```

Vectorize

- Find a stride-1 dimension such that v = N^2 : v < vec_size , constrain tiling to multiples of v

Tile

- For each index hill climb and use cost model to maximize reuse while fitting in cache & registers

Load

- Create a loading pattern designed to minimize bank conflicts for any number of parallel readers

Loop

- Order loops using a topological ordering to maximize cache reuse

Thread

- Rollup as many inner loops into hardware threads as possible

PlaidML v1: Stripe

Extending PlaidML to encompass the modern accelerator landscape

PlaidML v1 / Stripe

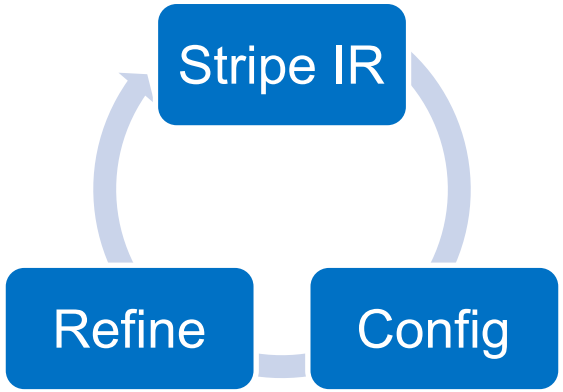
- Stripe enables:
  - Arbitrary tensorization
  - Complex vertical fusion
  - Arbitrarily complex memory hierarchies
  - Heterogenous compute topologies
  - Detailed performance / cost estimates
  - Software / hardware co-design

PlaidML v1 / Stripe: Polyhedral IR

PlaidML v1 introduces **Stripe**: a polyhedral IR that is highly amenable to optimization.

Stripe enables distinct passes that process stripe and emit more stripe

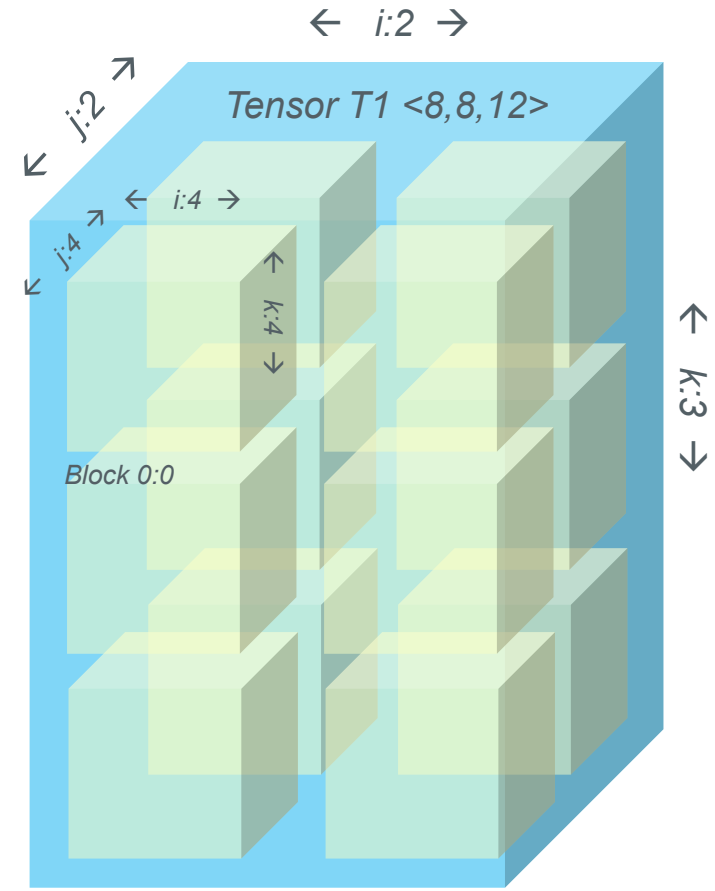
Stripe fundamentally represents operations over a polyhedral tensor space.



Stripe in Depth

Stripe Conceptual Model

- Describes nested and repeated computational **BLOCKS,** each **BLOCK** represents a set of parallelizable computations
- **BLOCKS** are described by **INDEXES** and **CONSTRAINTS** that create polyhedral bounds over views of tensors called **REFINEMENTS**
- Nested **BLOCKS** have their own **INDEXES**
- Nested **BLOCKS** can create polyhedral sub regions of **REFINEMENTS** in the parent block by creating more **REFINEMENTS** which are automatically offset.
- The interior of a **BLOCK** nest contains code that is executed for every valid value of every **INDEX** of every containing **BLOCK**.



Block 0:

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Tags

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Allocations

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Nested Blocks

Refinements

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Indexes

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Constraints

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Nested Blocks

Aggregators

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe IR Explained: Stripe Top (HW Independent)

```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Nested Blocks

Tile Code

SSA IL

Stripe IR Explained: Stripe Top (HW Independent)

Nested Blocks

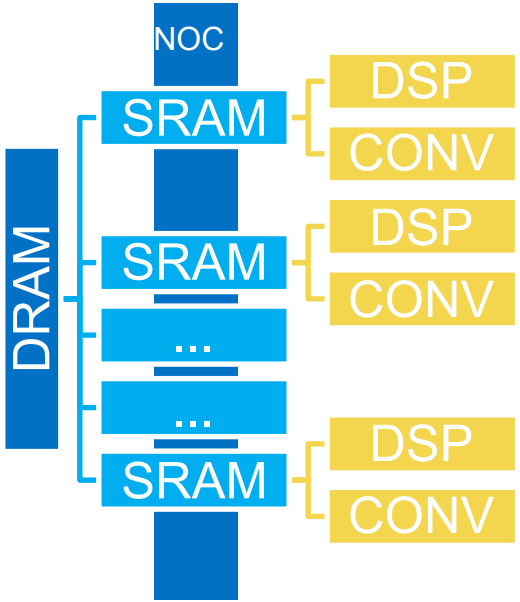
```
0: #program block [] ( // layer_test7
     none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
     none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
     …
     none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
       in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
       in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
       out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
       none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
         // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
         -1 + kx + x >= 0
         1024 - kx - x >= 0
         -1 + ky + y >= 0
         1024 - ky - y >= 0
         out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
         in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
         in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Tile Code

Stripe: Hardware Model

```
"clock_mhz": {{ CLOCK_MHZ }},
"mem_units": {
    "DRAM": { "count": 1, "size_KiB": 1048576 },
    "SRAM": { "count": {{ NUM_SRAM }}, "size_KiB": {{ SRAM_SIZE_KIB }} },
},
"exec_units": {
    ”DSP": { "count": {{NUM_DSP}}, "ops_per_cycle": 64 },
    "CONV": { "count": {{NUM_CONV}}, "ops_per_cycle": 512, “pipeline_depth”: 2 }
},
"tx_units": {
    "DMA": { "count": 1 },
    "NOC": { "count": 1 },
},
"buses": [
    { "sources": ["DRAM[0]"], "sinks": ["DMA[0]"], "bytes_per_cycle": 64 },
    { "sources": ["DMA[0]"], "sinks": ["DRAM[0]"], "bytes_per_cycle": 64 },
    {
        "sources": ["DMA[0]"],
        "sinks": [{% for i in range(NUM_SRAM) %} "SRAM[{{i}}]"{endfor %}],
        "bytes_per_cycle": 64
    },
    {
        "sources": ["NOC[0]"],
        "sinks": [{% for i in range(NUM_SRAM) %} "SRAM[{{i}}]"{% endfor %}],
        "bytes_per_cycle": 512
    },
```

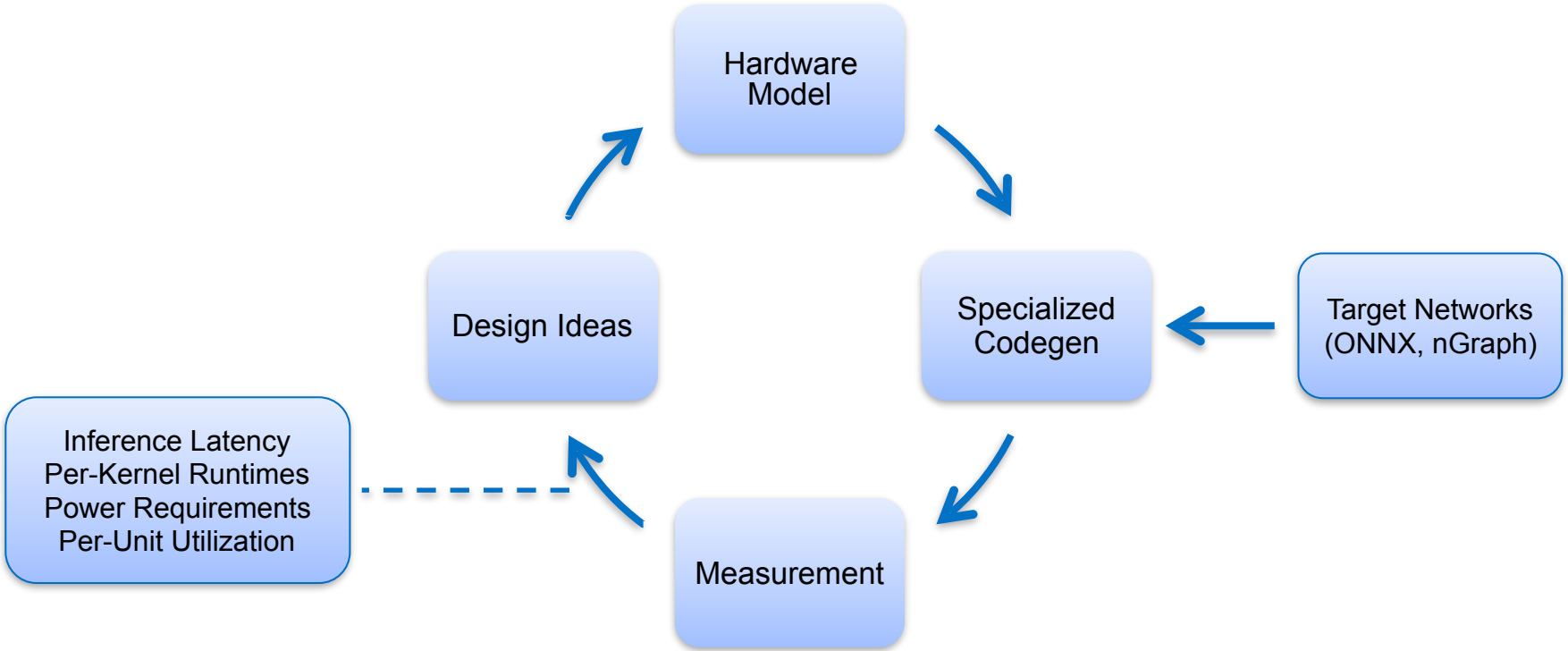
.
.
.



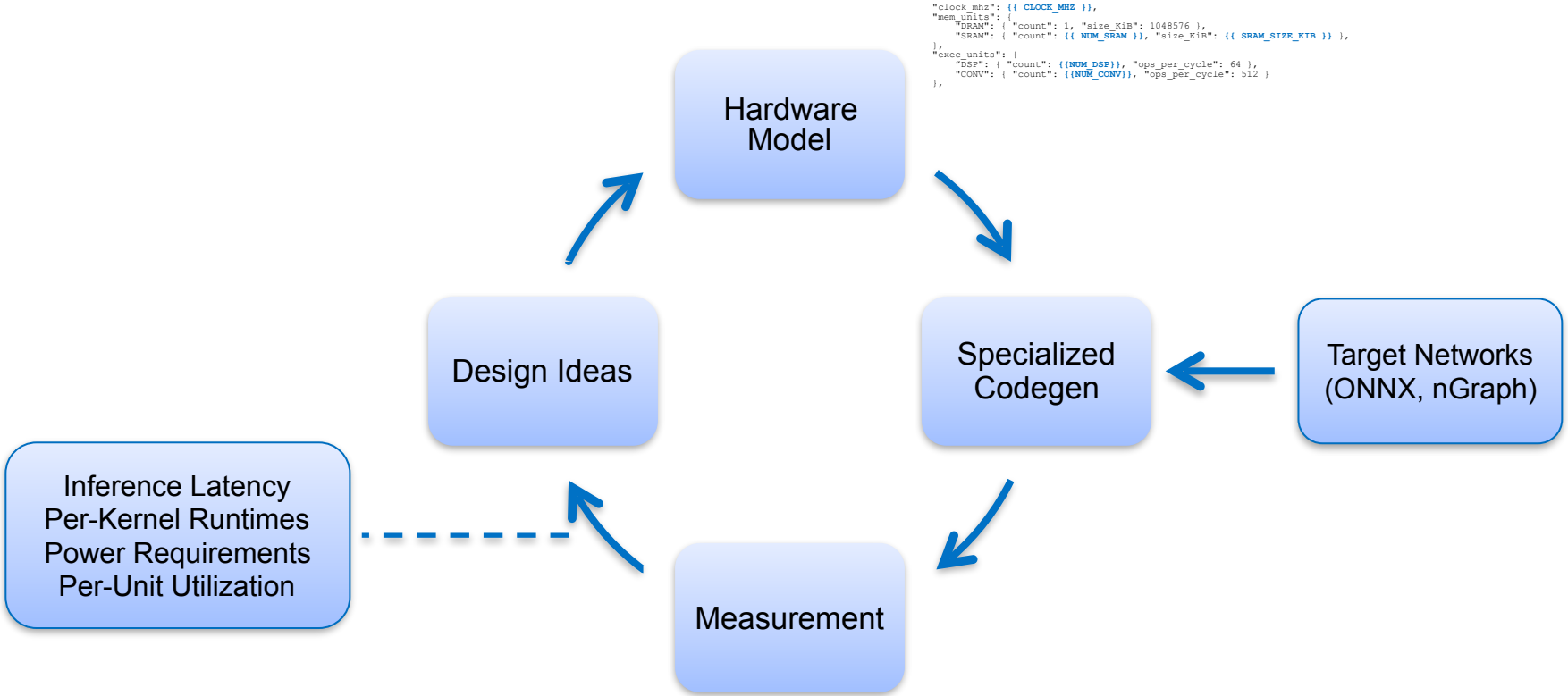
Stripe: Optimizer Config

```
{ "name": "fuse_CONV_add", "fusion": { "a_reqs": ["CONV"], "b_reqs": ["eltwise_add"], "fused_set": ["CONV"] } },
{ "name": "fuse_CONV_zelu", "fusion": { "a_reqs": ["CONV"], "b_reqs": ["eltwise_zelu"], "fused_set": ["CONV"] } },
{ "name": "fuse_CONV", "fusion": { "parent_reqs": ["CONV"], "fused_set": [”CONV_inner"] } },
{ "name": "localize_main", "localize": { "reqs": ["main"] } },
{ "name": "scalarize_main", "scalarize": { "reqs": ["main"] } },
{ "name": "loc_CONV", "locate_block": { "reqs": [”CONV"], "loc": { "name": ”CONV" } } },
{ "name": "loc_pool", "locate_block": { "reqs": ["agg_op_max"], "loc": { "name": "DSP" } } },
{ "name": "loc_eltwise", "locate_block": { "reqs": ["eltwise"], "loc": { "name": "DSP" } } },
 …
 …
 …
{ "name": "deps_main", "compute_deps": { "reqs": ["main"] } },
{
    "name": "schedule_main",
    "schedule": {
        "reqs": ["main"],
        "mem_loc": { "name": ”SRAM" },
        "mem_KiB": {{ SRAM_SIZE_KIB / NUM_SRAM }},
        "alignment": 16,
        "xfer_loc": { "name": "DMA" },
        "allow_out_of_range_accesses": true,
        "num_banks": {{ NUM_SRAM }}
    }
},
{ "name": "place_program", "memory_placement": { "reqs": ["program"], "locs": [{ "name": "DRAM" }], "alignment": 4 } }
```

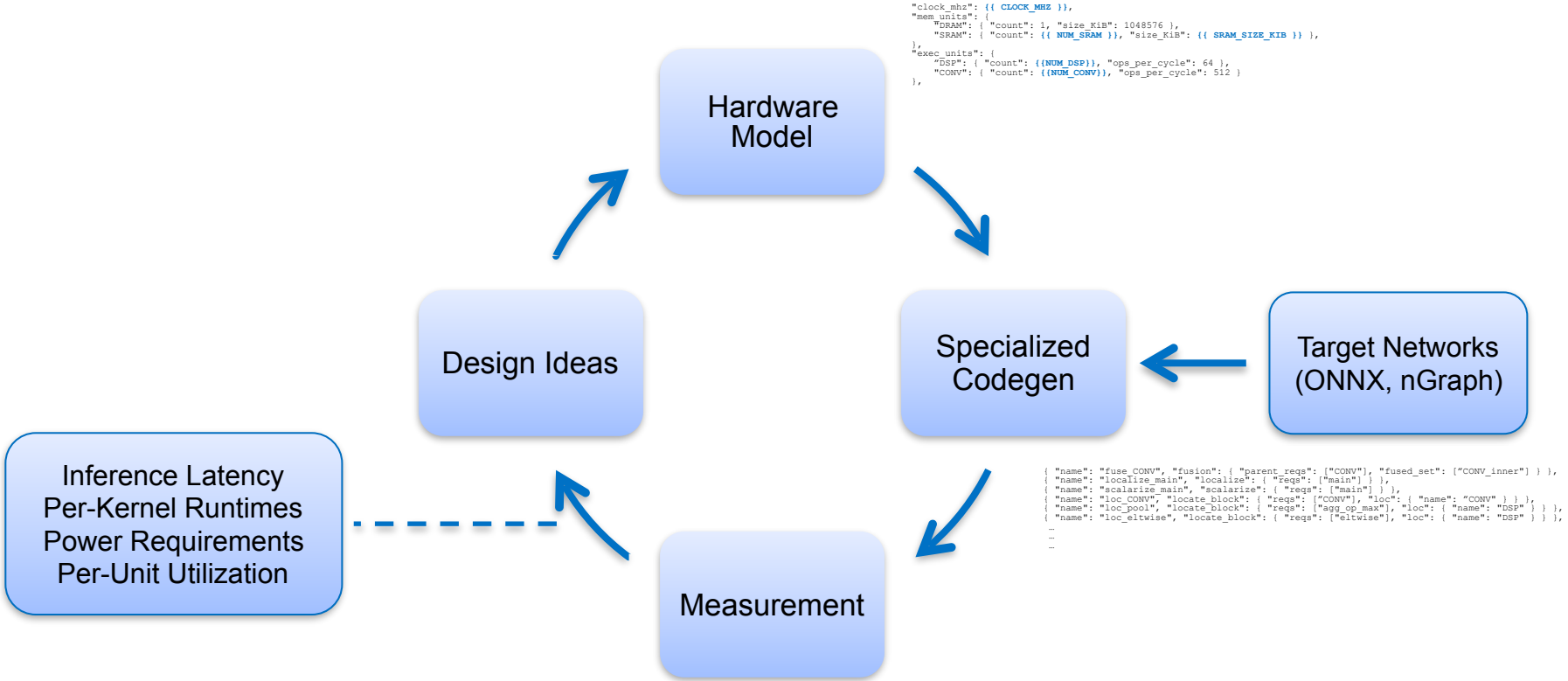
Stripe: Enabling Hardware / Software Co-Design



Stripe: Enabling Hardware / Software Co-Design



Stripe: Enabling Hardware / Software Co-Design

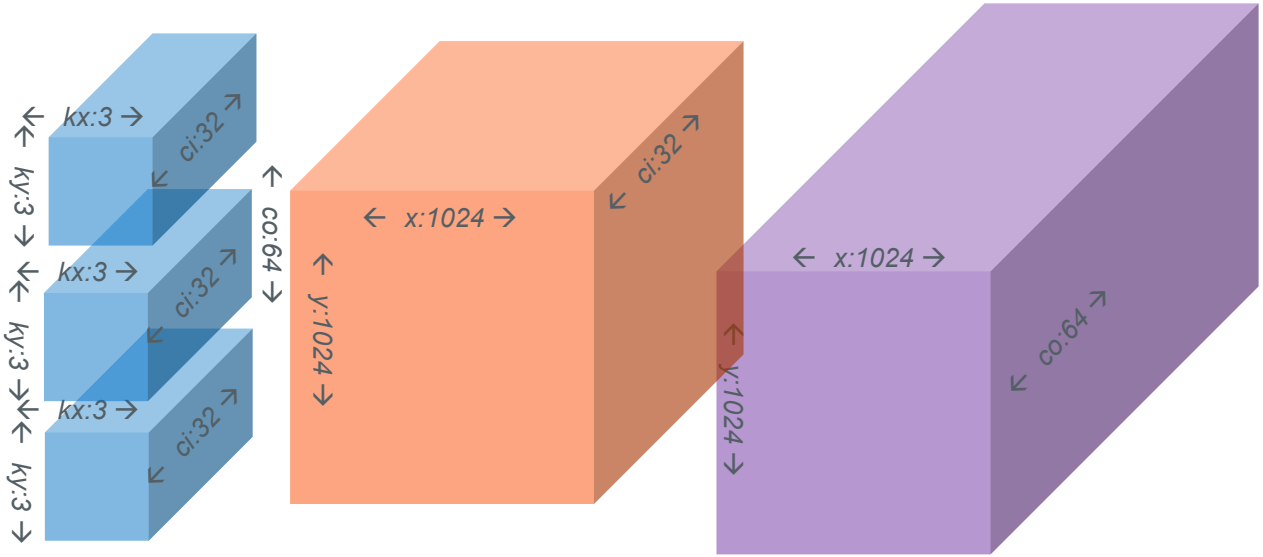


Stripe: Tensorization

```
”tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": ”i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": ”c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": ”i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": ”i2", "size": 4, "outs": [-1], "ins": [ 0, -1] },
…  ]},] } },
```

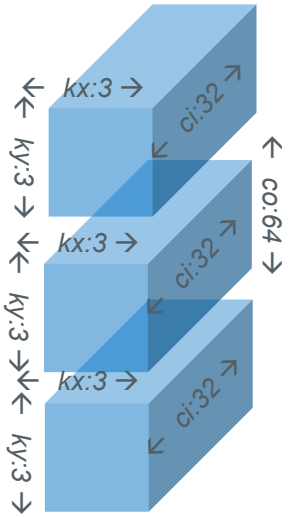
Stripe: Tensorization

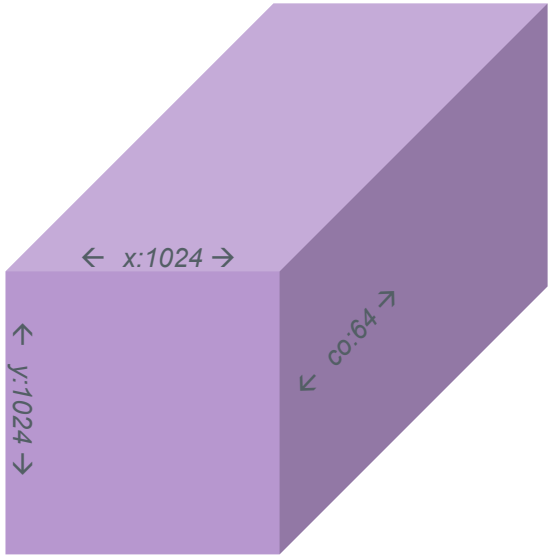
```
”tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": ”i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": ”c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": ”i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": ”i2", "size": 4, "outs": [-1], "ins": [ 0, -1] },
…  ]},] } },
```



Stripe: Tensorization

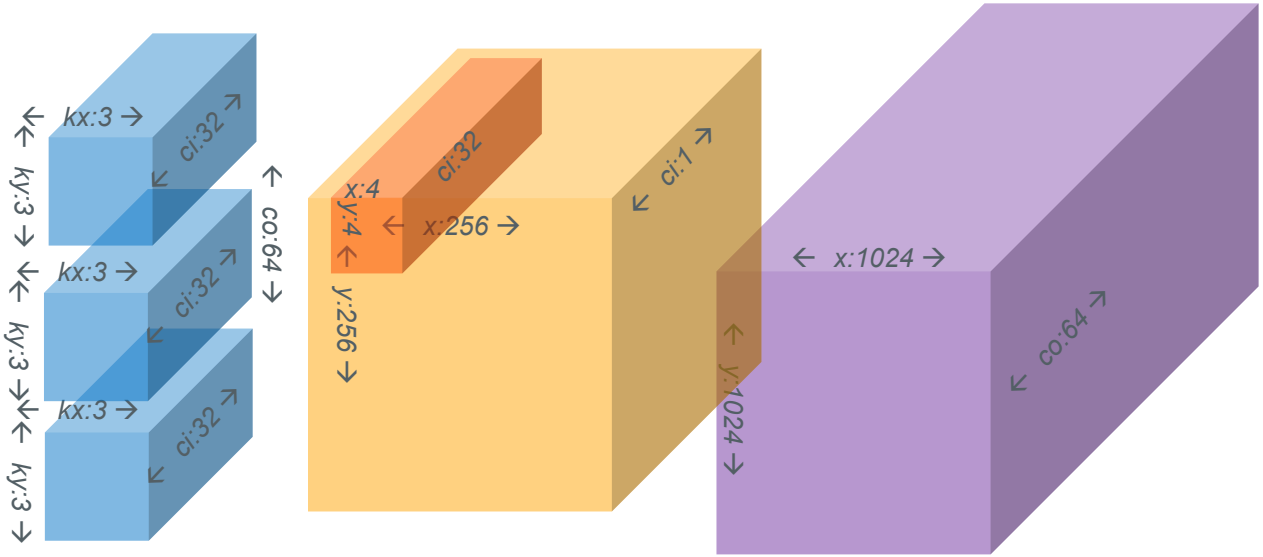
```
”tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": ”i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": ”c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": ”i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": ”i2", "size": 4, "outs": [-1], "ins": [ 0, -1] },
…  ]},] } },
```





Stripe: Tensorization

```
”tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": ”i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": ”c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": ”i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": ”i2", "size": 4, "outs": [-1], "ins": [ 0, -1] },
…  ]},] } },
```



Stripe: Tensorization

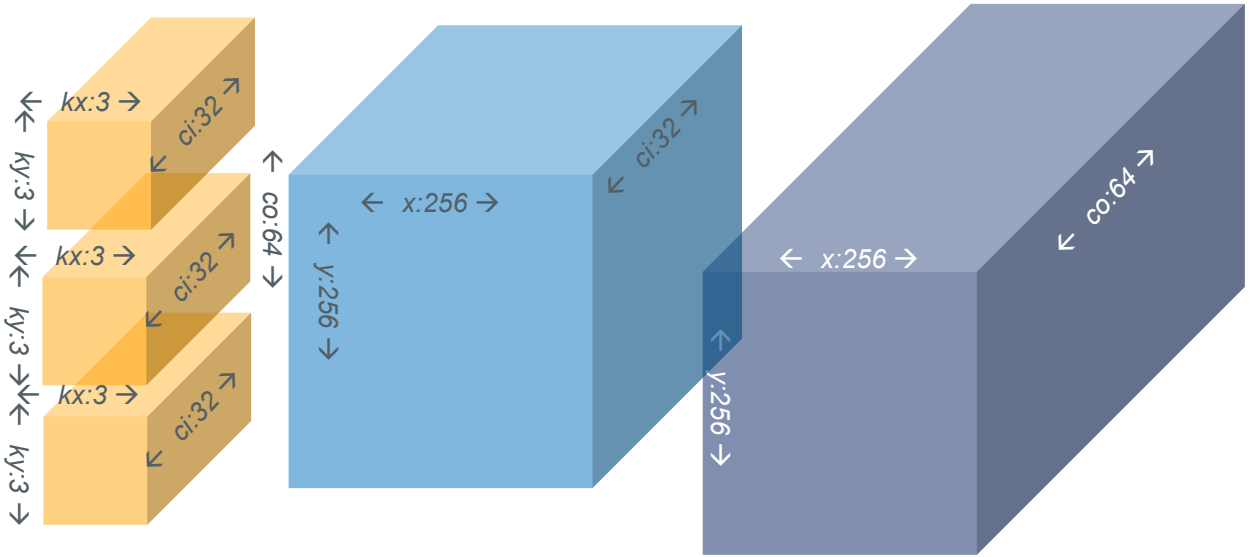
```
”tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": ”i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": ”c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": ”i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": ”i2", "size": 4, "outs": [-1], "ins": [ 0, -1] },
…  ]},] } },

BEFORE:
0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] ( // kernel_0
        // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
        out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
        in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
        in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
    }

AFTER:
0: #agg_op_add #comb_op_mul #contraction #CONV #kernel block [ci:1, co:1, kx:1, ky:1, x:256, y:256] ( // kernel_0
        // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
        out<DRAM[0]> O1[4*x, 4*y, 16*co]:add i8(4:65536, 4:64, 16:1)
        in<DRAM[0]> I[kx + 4*x, ky + 4*y, 32*ci] i8(4:32768, 4:32, 32:1)
        in<DRAM[0]> K1[kx, ky, 32*ci, 16*co] i8(1:6144, 1:2048, 32:1, 16:32)
    ) {
      0: #CONV_inner block [ci:32, co:64, kx:3, ky:3, x:4, y:4] ( // kernel_0
          out<DRAM[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
          in<DRAM[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
          in<DRAM[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:1, 1:32)
      ) {
      0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
    }}
```

Stripe: Auto-Tile

```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : false,
  "memory" : "SRAM" // ”pipeline_depth” : 2
}
```

Stripe: Auto-Tile

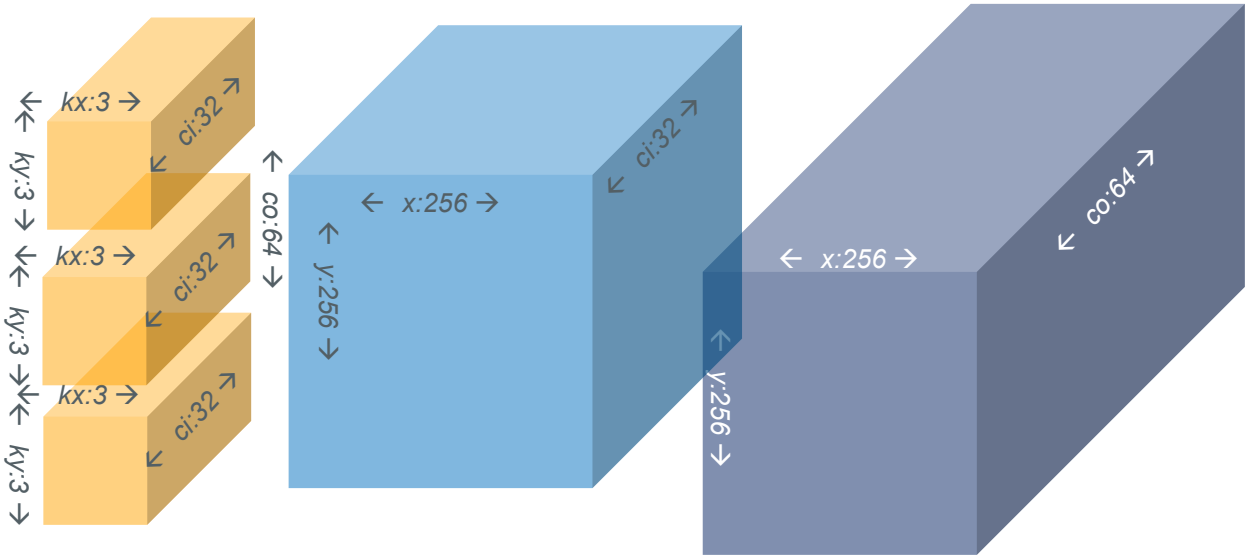
```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : false,
  "memory" : "SRAM" // ”pipeline_depth” : 2
}
```



Stripe: Auto-Tile

```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : false,
  "memory" : "SRAM" // ”pipeline_depth” : 2
}
```

kx	ky	ci	co	x	y	cost
1	1	32	4	8	8	120
1	1	16	8	8	8	140
1	1	32	5	4	4	270
3	3	32	1	6	6	310
3	3	16	1	9	9	340



Stripe: Auto-Tile

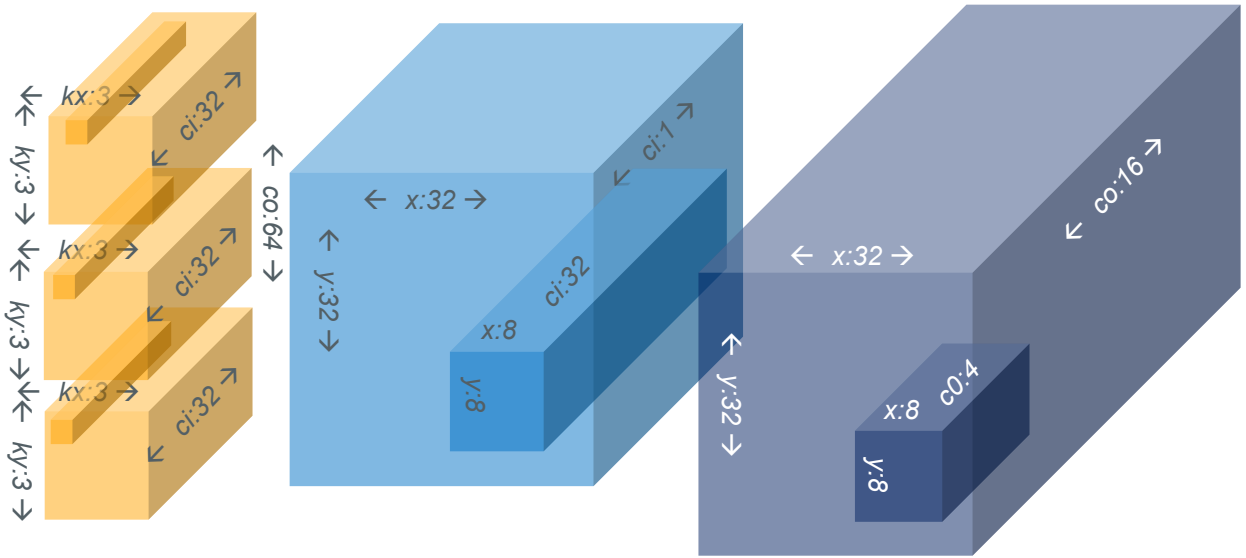
```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : false,
  "memory" : "SRAM" // ”pipeline_depth” : 2
}
```

kx	ky	ci	co	x	y	cost
1	1	32	4	8	8	120
1	1	16	8	8	8	140
1	1	32	5	4	4	270
3	3	32	1	6	6	310
3	3	16	1	9	9	340

Stripe: Auto-Tile

```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : false,
  "memory" : "SRAM" // "pipeline_depth" : 2
}
```

kx	ky	ci	co	x	y	cost
1	1	32	4	8	8	120
1	1	16	8	8	8	140
1	1	32	5	4	4	270
3	3	32	1	6	6	310
3	3	16	1	9	9	340



Stripe: Auto-Tile

```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : true,
  "memory" : "SRAM" // "pipeline_depth" : 2
}
```

BEFORE:

```
0: #conv block<CONV[0]> [ci:32, co:64, kx:3, ky:3, x:256, y:256] (
       out<DRAM[0]> O1[co, x, y]:add i8(64:65536, 4:64, 4:1)
       in<DRAM[0]> I[kx + x - 1, ky + y - 1, ci] i8(4:32768, 4:32, 32:1)
       in<DRAM[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 32:1, 64:32)
) {
   0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
}
```

AFTER:

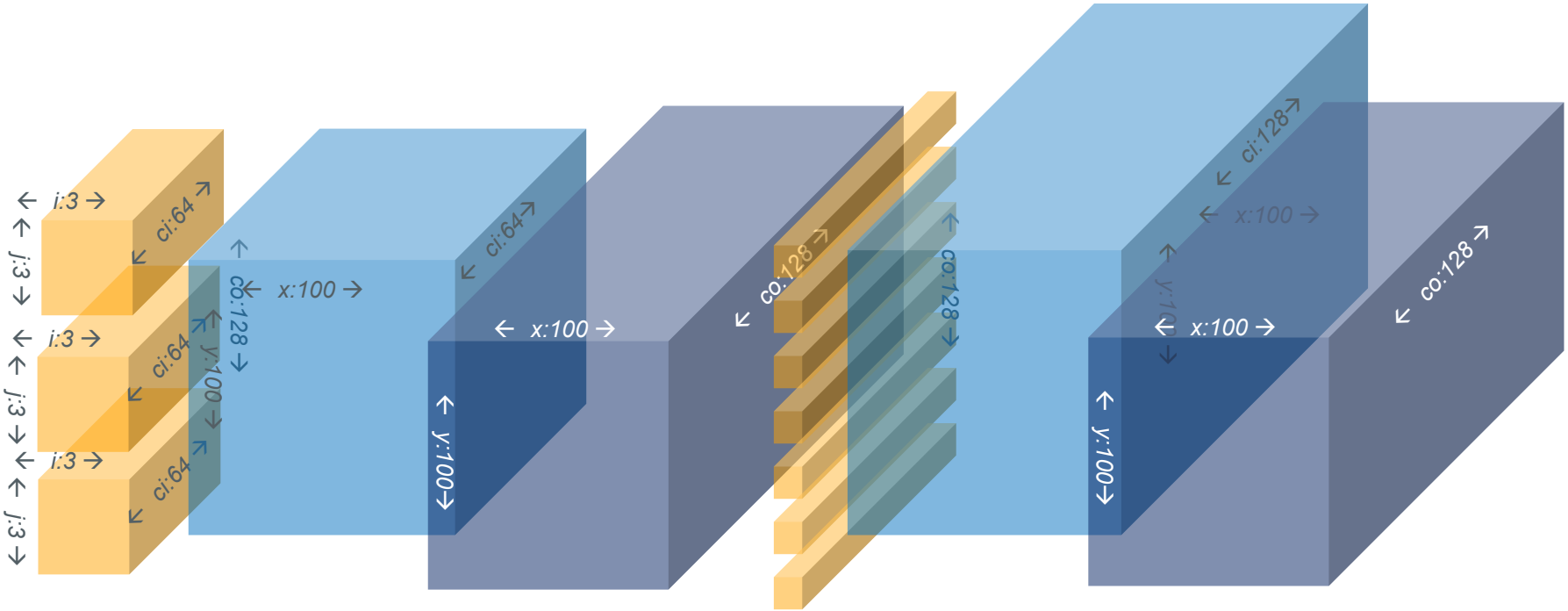
```
0: #conv block<CONV[0]> [ci:1, co:16, kx:3, ky:3, x:32, y:32] ( // kernel_0
    out<DRAM[0]> O1[8*x, 8*y, 4*co]:add i8(16:65536, 16:64, 64:1)
    in<DRAM[0]> I[kx + 8*x, ky + 8*y, ci] i8(16:32768, 16:32, 32:1)
    in<DRAM[0]> K1[kx, ky, ci, 4*co] i8(1:6144, 1:2048, 32:1, 64:32)
) {
    0: <Elided memory xfers>
    1: #conv_inner block<CONV[0]> [ci:32, co:4, kx:1, ky:1, x:8, y:8] ( // No halos as the tiling makes lots of 1x1 convolutions
        out<SRAM[0]> O1[co, x, y]:add i8(4:65536, 8:64, 8:1)
        in<SRAM[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 32:1)
        in<SRAM[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 32:1, 4:32)
    ) {
      0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
    }
}
```

Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```

Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```

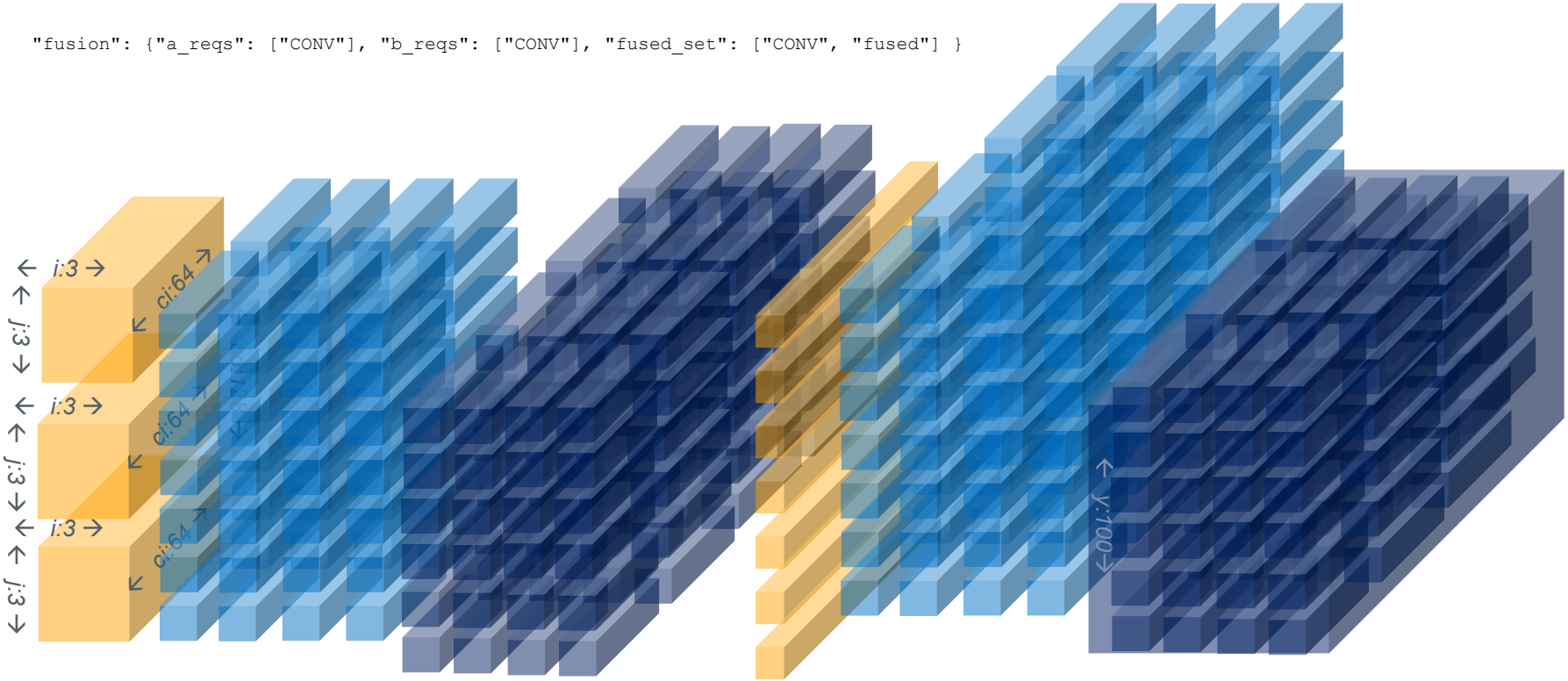


Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```

Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```



Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```

BEFORE:

```
0: #agg_op_add #comb_op_mul #CONV #contraction #kernel block [ci:64, co:128, i:3, j:3 x:100, y:100] ( // kernel_0
  // O1[x, y, co : X, Y, CO1] = +(In[-1 + i + x, -1 + j + y, ci] * K1[i, j, ci, co])
  ) {
    0: $In = load(In);  1: $K1 = load(K1);  2: $O1 = mul($In, $K1);  3: O1 = store($O1)
  }
1: #agg_op_add #comb_op_mul #CONV #contraction #kernel block [ci:128, co:128, x:100, y:100] ( // kernel_1
  // O2[x, y, co : X, Y, CO2] = +(O1[i + x, j + y, ci] * K2[i, j, ci, co])
  ) {
    0: $O1 = load(O1);  1: $K2 = load(K2);  2: $O2 = mul($O1, $K2);  3: O2 = store($O2)
  }
```

AFTER:

```
0: #fused block [co:8, x:100, y:100] ( // kernel_0+kernel_1 … ) {
  0: block [ci:64, co:16, i:3, j:3, x:1, y:1] (…){
    out<SRAM[0]> O1[x, y, co]:add fp32(1:16, 1:16, 1:16, 1:1)
    in<[0]> In[-1 + i + x, -1 + j + y, ci] fp32(1:640000, 1:6400, 1:64, 1:1)
    in<[0]> K1[i, j, ci, co] fp32(1:24576, 1:8192, 1:128, 1:1)
  ) {
    0: $In = load(In);  1: $K1 = load(K1);  2: $O1 = mul($In, $K1);  3: O1 = store($O1)
  }
  1: block [ci:64, co:16, x:1, y:1] (…) {
    out<[0]> O2[x, y, co]:add fp32(1:1280000, 1:12800, 1:128, 1:1)
    in<SRAM[0]> O1[x, y, ci] fp32(1:16, 1:16, 1:16, 1:1)
    in<[0]> K2[0, 0, ci, co] fp32(1:16384, 1:16384, 1:128, 1:1)
  ) {
    0: $O1 = load(O1);  1: $K2 = load(K2);  2: $O2 = mul($O1, $K2);  3: O2 = store($O2)
  }
}
```

PlaidML v1.x / Stripe : Status

- Initial code upstreamed to public as of 0.5
  - Available at https://github.com/plaidml/plaidml, inside the tile/stripe and tile/codegen directories
- Configurations for GPUs, CPUs & porting v0 to Stripe in progress
- Extensions for conditionals, loops, and indirection (scatter / gather) coming in v1
- Paper coming out early next year
- Specification available on request to: tim.zerrell@intel.com