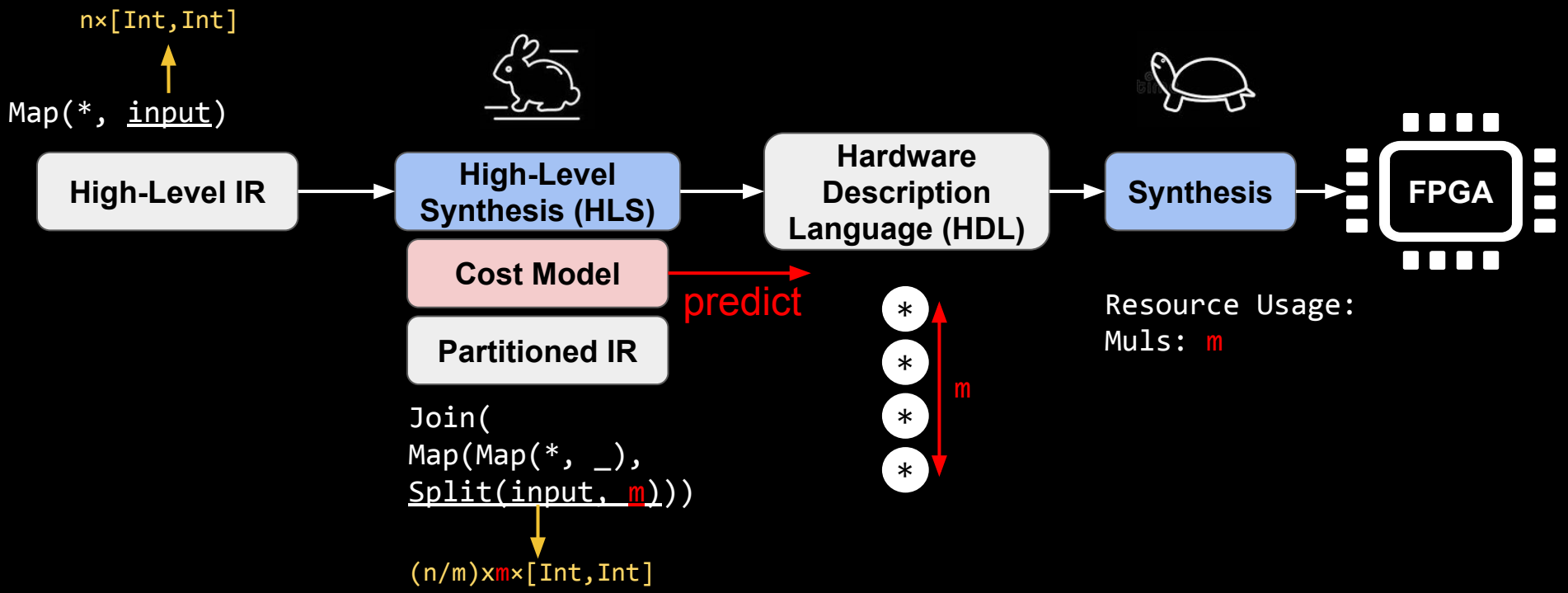
Toward Automatic Hardware and Data Partitioning in HLS



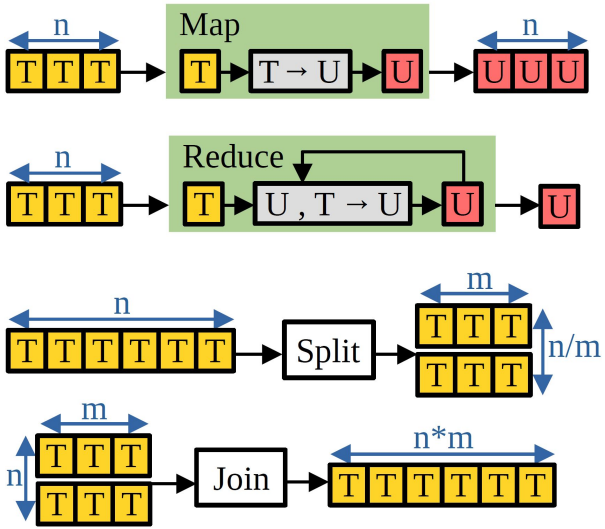
McGill

Tzung-Han Juang, Jonathan Van der Cruysse, and Christophe Dubach

McGill University – CDP '24 – Tuesday November 12, 2024

Functional Intermediate Representation (IR) [1]

```
Map(f: T→U, in: n×T): n×U

Reduce(f: (U,T)→U, in: n×T): U

Split(in: n×T, size: m): (n/m)×m×T

Join(in: n×m×T): (nm)×T
```




```
DotProd(in: nx[Int,Int]): Int = Reduce(+, Map(*, in))
```

[1]: Christof Schlaak et al. Memory-Aware Functional IR for Higher-Level Synthesis of Accelerators. . **TACO 2022**

Lower Functional IR

Algo Level:

```
Map(*3, in: n×Int): n×Int
```

Lowering

Arch Level:

```
MapStm(*3, in: Stm[Int,n]): Stm[Int,n]
```

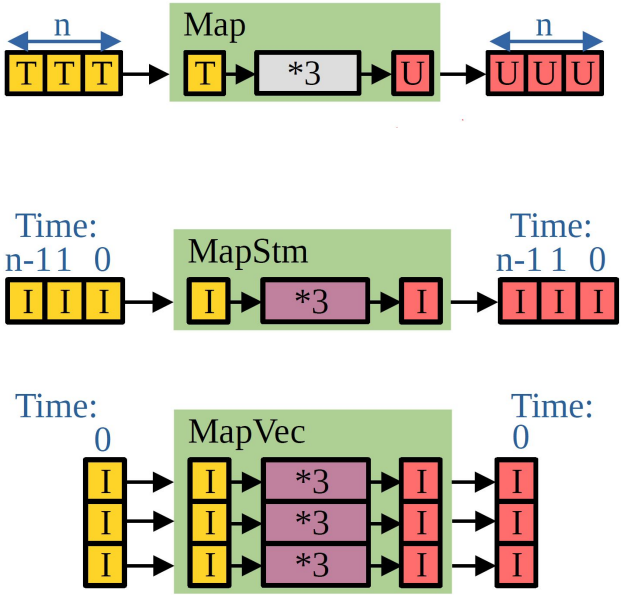
Optimizing

```
MapVec(*3, in: Vec[Int,n]): Vec[Int,n]
```

Lowering

HDL Level:

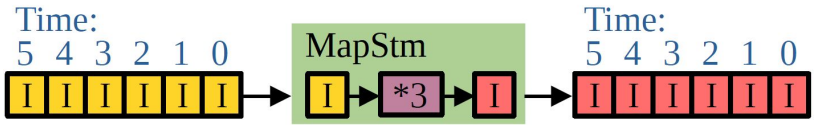
```
entity MapVec is port(input in std_logic_vector ...)
```



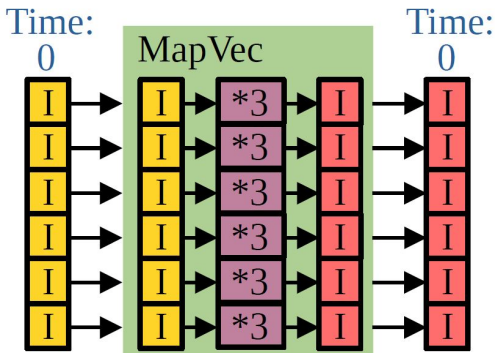
Motivation: Determining Partition Size

Original:

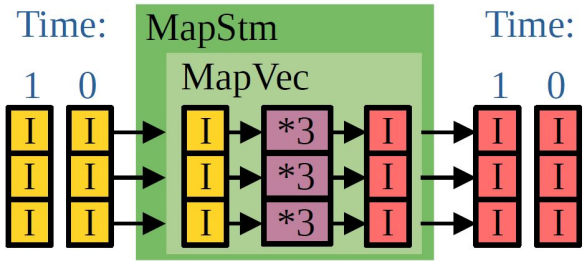
```
MapStm(*3, in: Stm[Int,6]): Stm[Int,6]
```



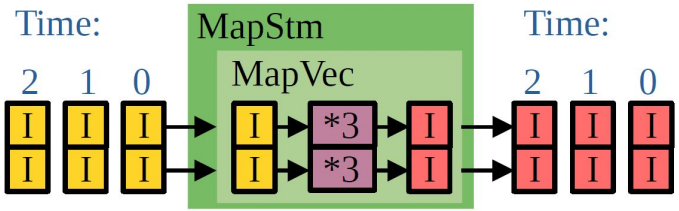
After Optimizing



1 cycle
6 resources



2 cycles
3 resources



3 cycles
2 resources

Preserve Tuning Parameters for Partitioning

Algo Level:

```
Map(*3, in: 6×Int): 6×Int
```

↓ Express Partitioning

```
Join(Map(Map(*3, _), Split(in, tp)))
```

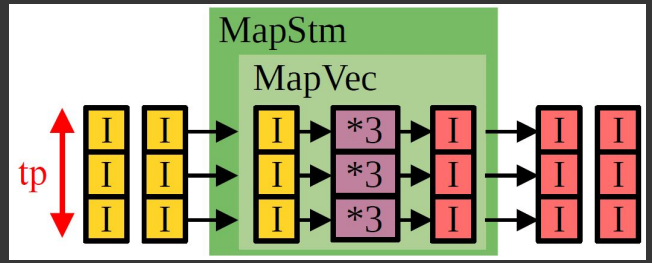
↓ Lowering

Arch Level:

```
JoinStm(MapStm(MapStm(*3, _), SplitStm(in, tp)))
```

↓ Optimizing

```
JoinStm(MapStm(Vec2Stm,
  MapStm(MapVec(*3, _),
    MapStm(Stm2Vec, SplitStm(in, tp)))))
```



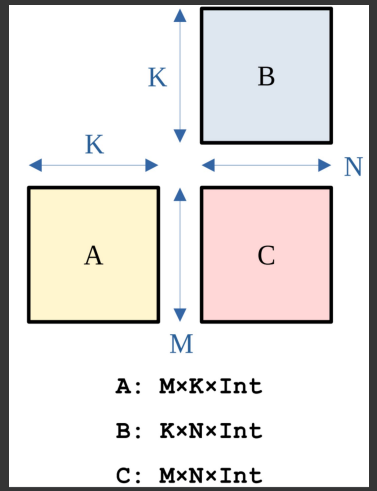
Insert Tuning Parameters

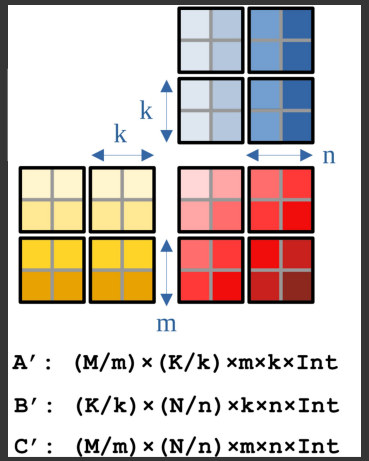
```
MatMul(A: MxKxInt, B: KxNxInt): MxNxInt =
  Map(a -> Map(b -> DotProd(a, b), Transpose(B)), A)
```

Add Tuning Parameters: m, n, k

```
A’ = Transpose(Split(A, (m, k))): (M/m)x(K/k)xmxkxInt
B’ = Transpose(Split(B, (k, n))): (K/k)x(N/n)xkxnxInt
C’ = Map(a’ =>
       Map(b’ =>
         Reduce(+,
           Map(tup =>
             MatMul(tup.0: mxkxInt, tup.1: kxnxInt)
           Zip(a’, b’)
         )), Transpose(B’)
       ), A’): (M/m)x(N/n)xmxnxInt
```

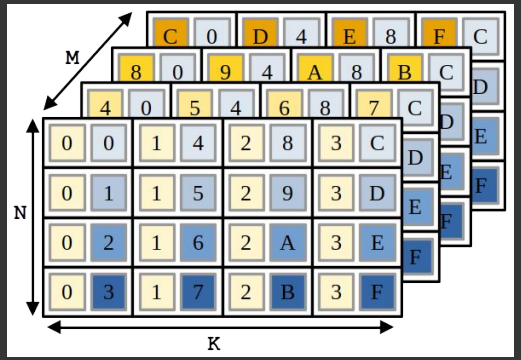
Linked tuning parameter

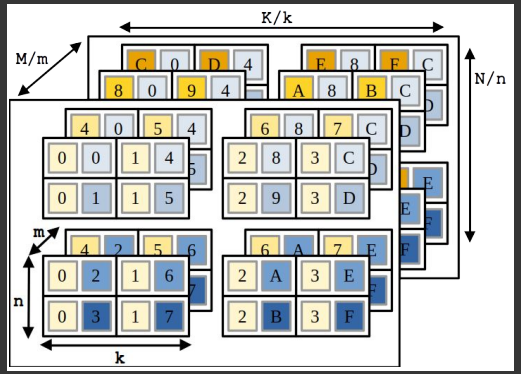




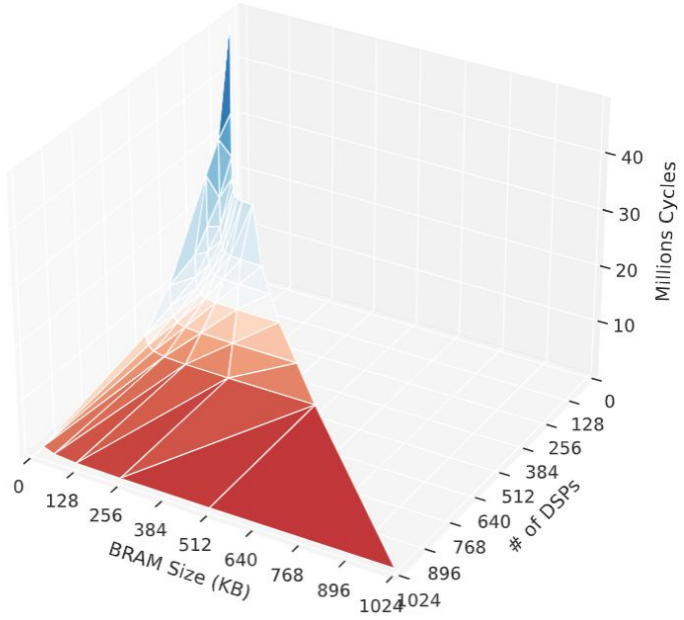
Partitioning with Single Input

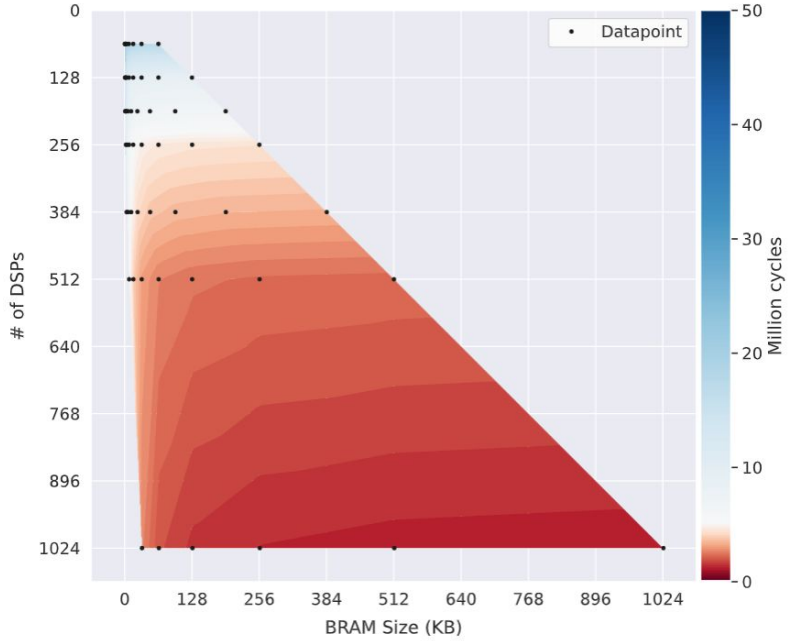
```
MatMul(AB: MxNxKx[Int,Int]): MxNxInt

DCM(divideFunc, conquerFunc, mergeFunc, AB)

divided: (M/m)x(N/n)x(K/k)xmxnxk×[Int,Int]
       = Transpose(Split(AB, (m, n, k)))

conquered: (M/m)x(N/n)x(K/k)xmxn×Int
         = Map(Map(Map(MatMul(_), _), _), divided)

merged: (M/m)x(N/n)xmxn×Int
       = Map(Map(Reduce(+, _), _), conquered)
```



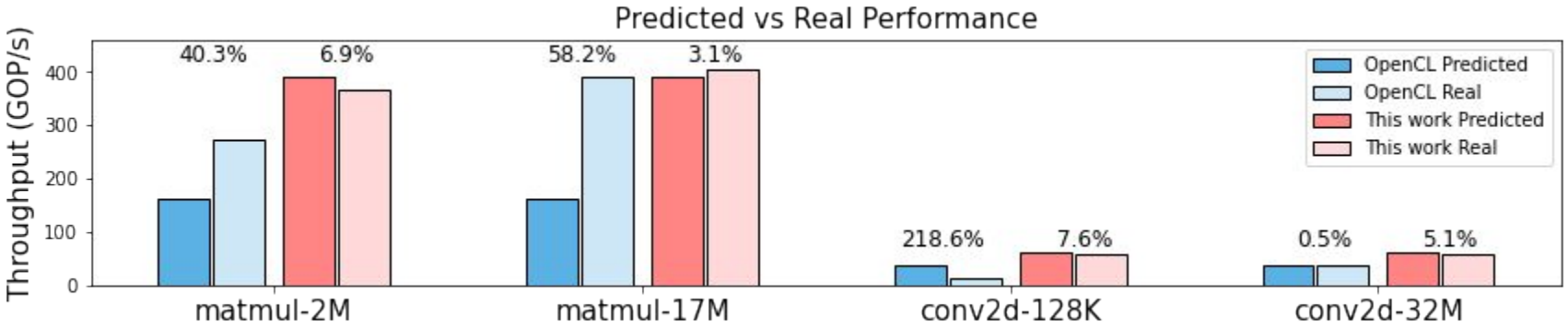


Exploration for Matrix Multiplication

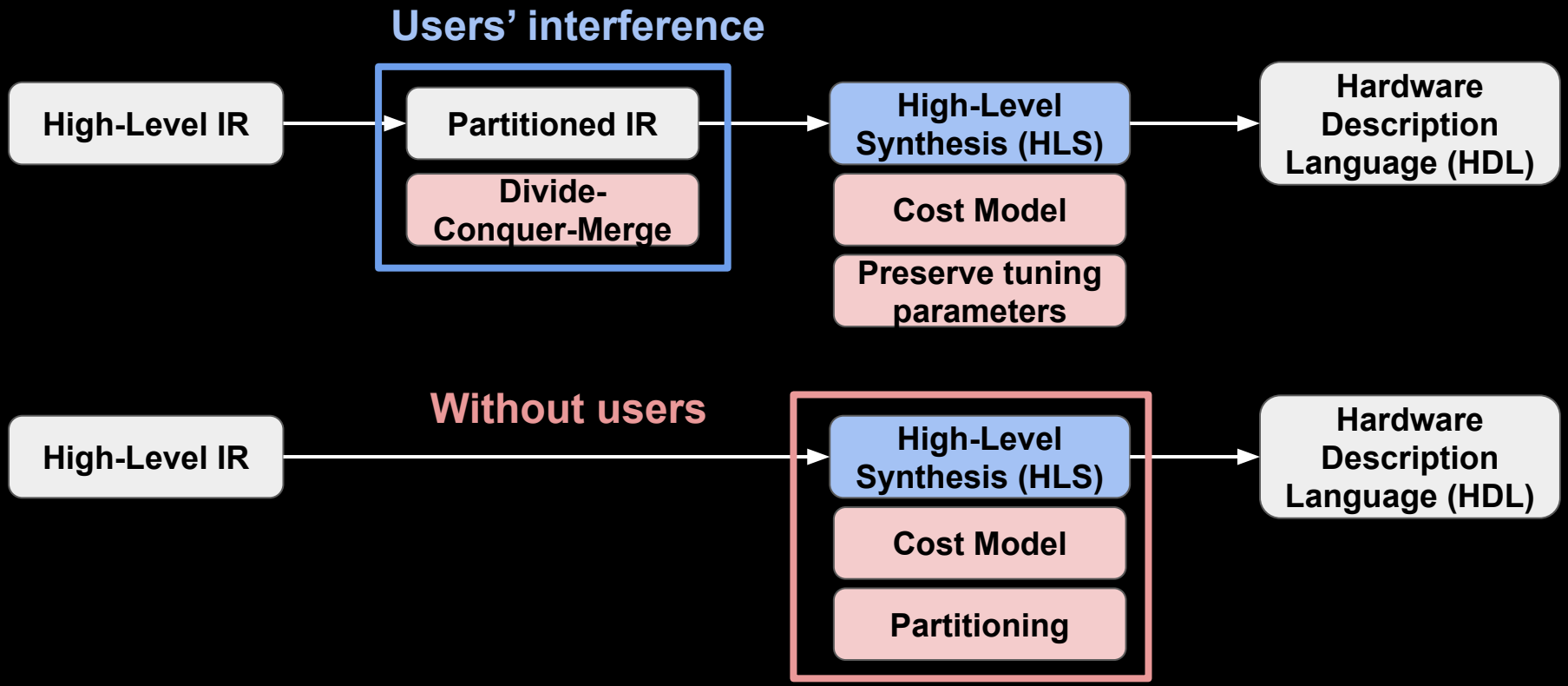




Performance Prediction against Intel OpenCL

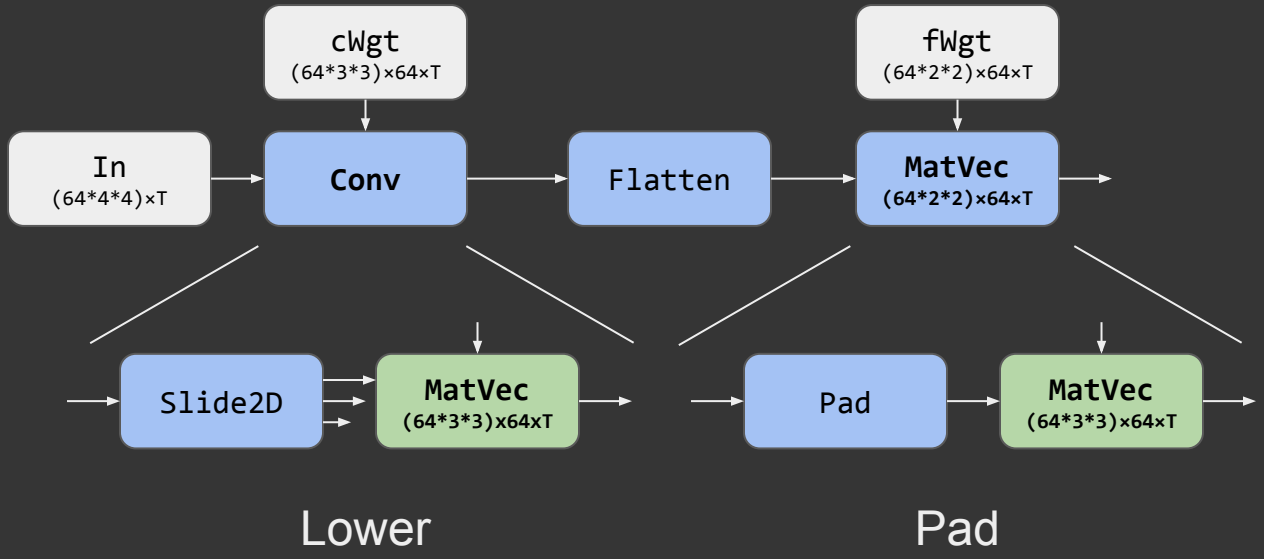


From Semi-Automatic to Automatic Partitioning



Automatic Partitioning with Resource Sharing

`MatVec(fWgt, Flatten(Conv(cWgt, In)))`

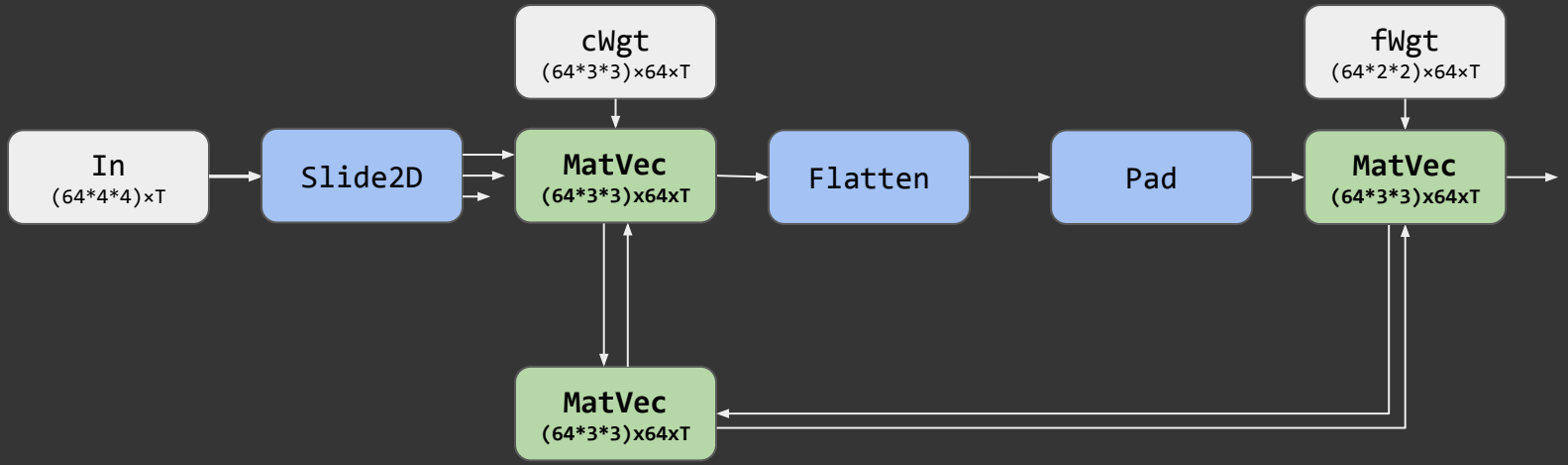


Resource Sharing

Ingredients:

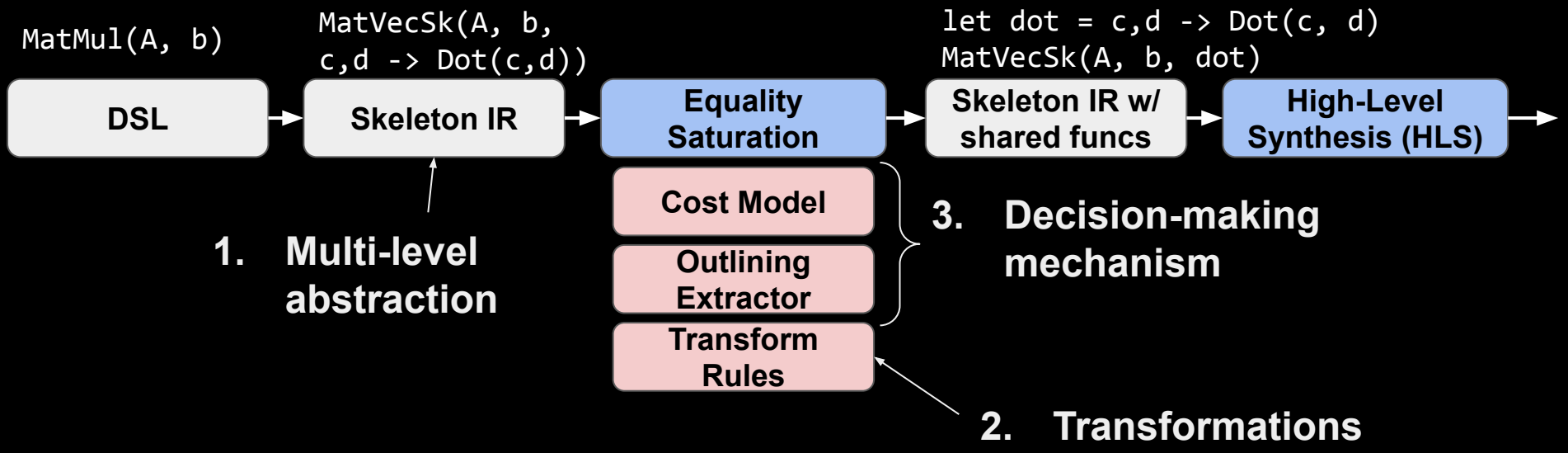
1. Multi-level abstraction
2. Transformations
3. Decision-making mechanism

```
MatVec(fWgt, Flatten(Conv(cWgt, In)))
```



[2]: Tzung-Han Juang et al. Let Coarse-Grained Resources Be Shared: Mapping Entire Neural Networks on FPGAs. **TECS 2023**

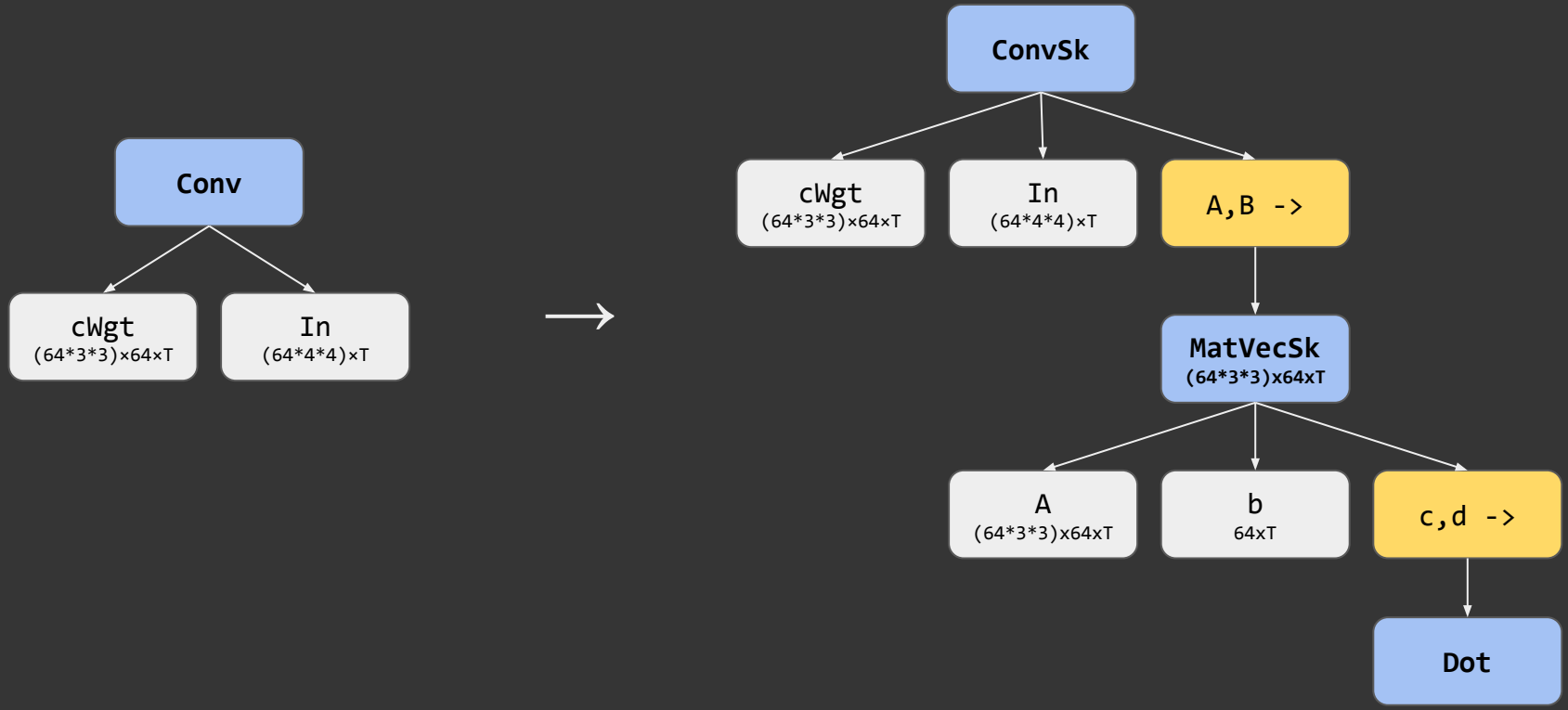
Overview


MatMul(A, b)
DSL
MatVecSk(A, b,
c,d -> Dot(c,d))
Skeleton IR
Equality Saturation
Cost Model
Outlining Extractor
Transform Rules
let dot = c,d -> Dot(c, d)
MatVecSk(A, b, dot)
Skeleton IR w/ shared funcs
High-Level Synthesis (HLS)
1. Multi-level abstraction
2. Transformations
3. Decision-making mechanism

Ingredient #1: Skeleton IR

DSL		Skeleton IR
`MatVec(A, b)`	→	`MatVecSk(A, b, c,d -> Dot(c, d))`
`Conv(Wgt, In)`	→	`ConvSk(Wgt, In, A,b -> MatVecSk(A, b, …))`

Skeleton IR


Conv
cWgt
(64*3*3)×64×T
In
(64*4*4)×T
ConvSk
cWgt
(64*3*3)×64×T
In
(64*4*4)×T
A,B ->
MatVecSk
(64*3*3)x64xT
A
(64*3*3)x64xT
b
64xT
c,d ->
Dot

Ingredient #2: Transformations

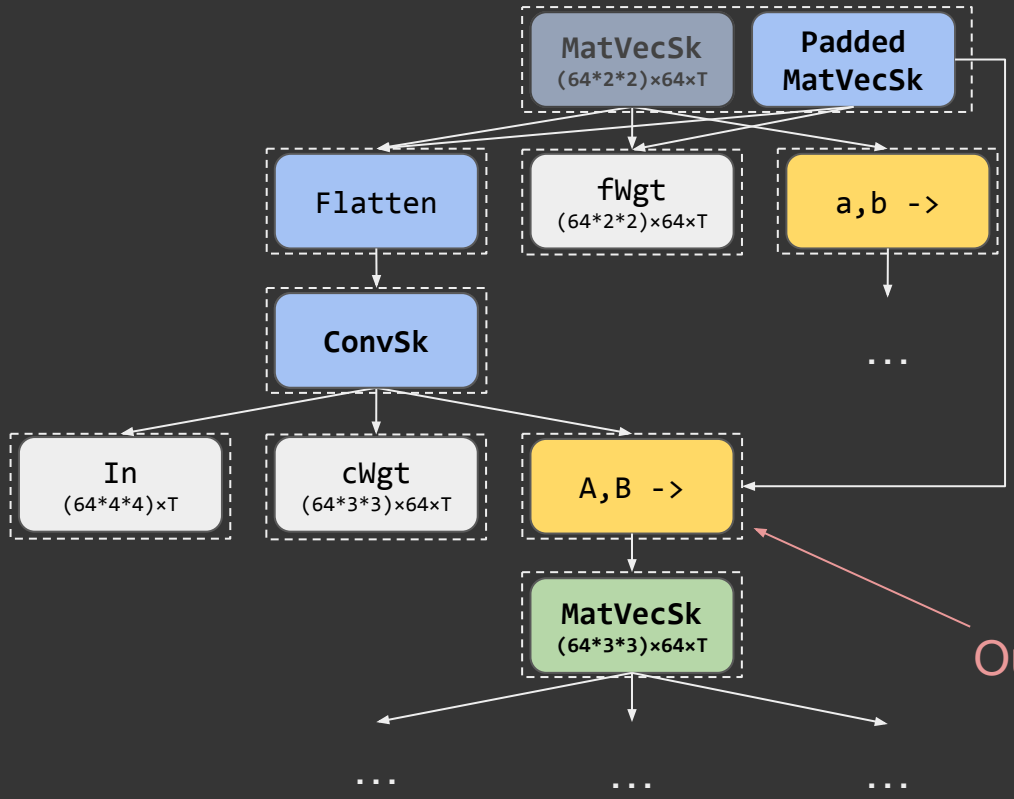
`Conv`	● Tile ○ Width/height ○ Output channels ○ Input channels ● Pad width/height
`MatVec`	● Tile ● Pad
`Dot`	● Change parallelism

Transformations

DSL		Skeleton IR
`MatVec(A, b)`	→	`MatVecSk(A, b, c,d -> Dot(c, d))`
`Conv(Wgt, In)`	→	`ConvSk(Wgt, In, A,B -> MatVecSk(A, B, ...))`
		`TiledMatVecSk(A, B, C,D -> MatVecSk(C, D, ...))`
		`PaddedMatVecSk(A, B, C,D -> MatVecSk(C, D, ...))`

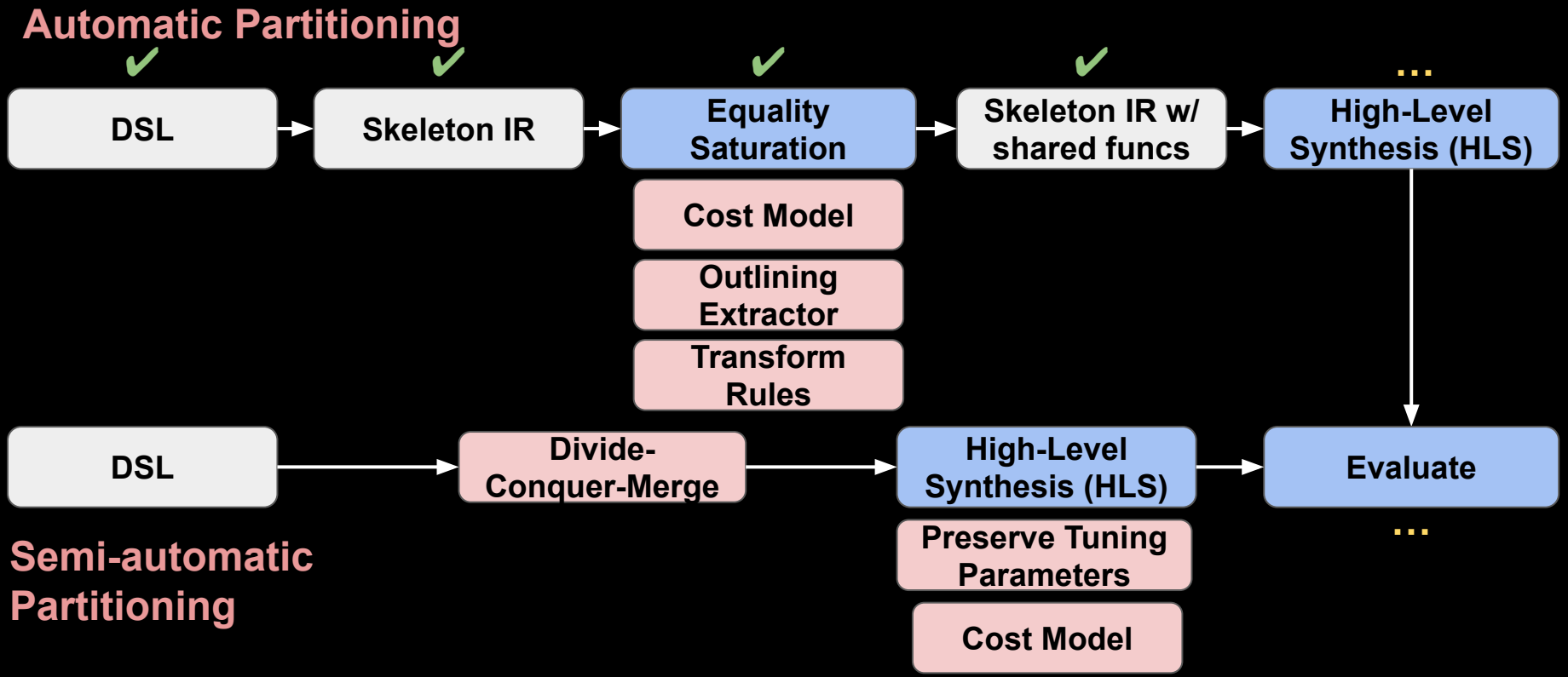
Ingredient #3: Equality Saturation

- Extraction
- Construction
- MatVec padding rule



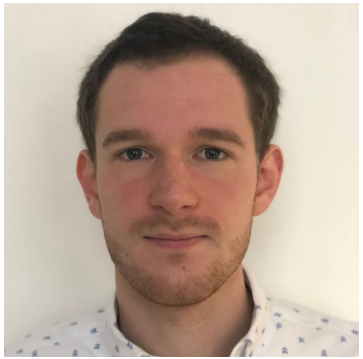

```
let mv = A,B ->
  MatVecSk(64*3*3)x64xT(...)

PaddedMatVecSk(
  fWgt,
  Flatten(ConvSk(
    cWgt, In, mv)),
  mv)
```

Summary + Future Work



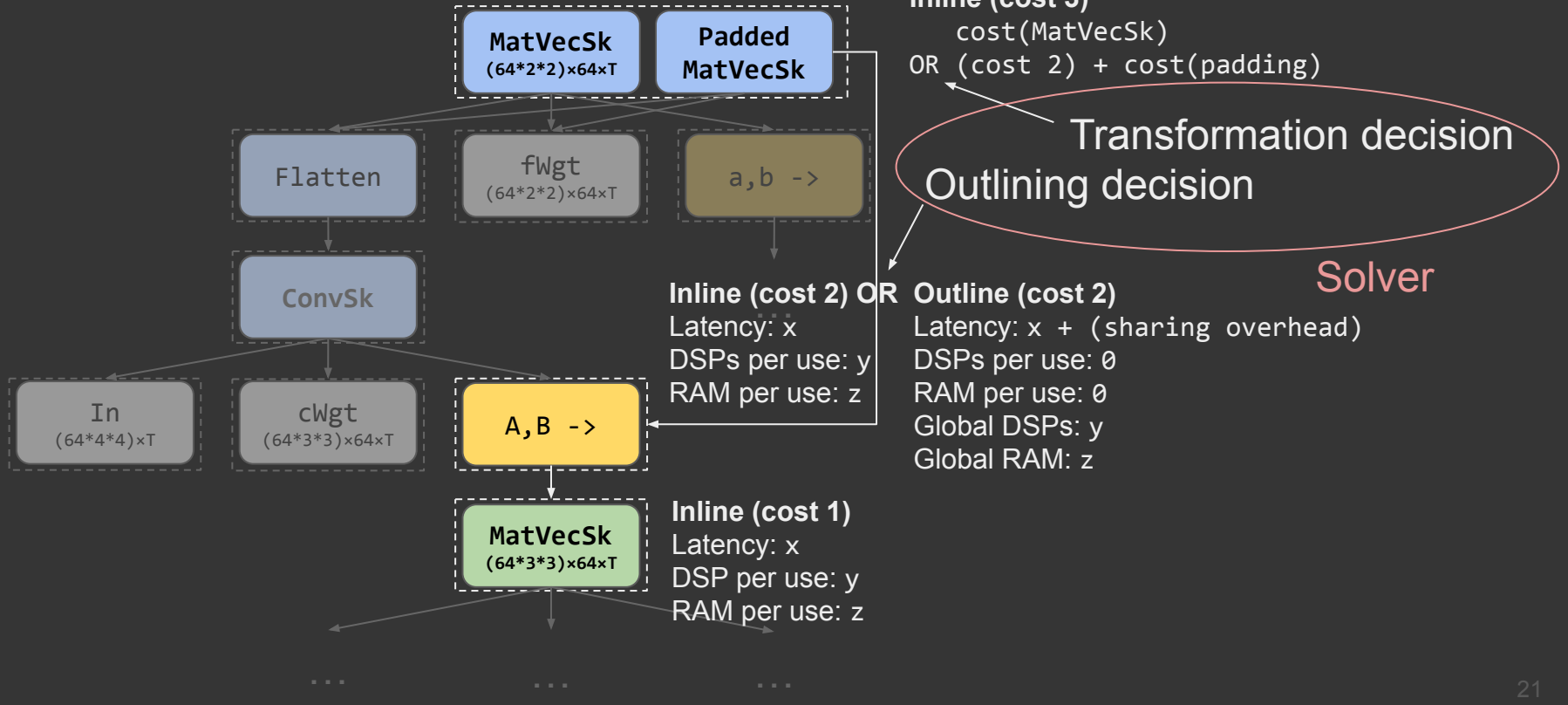
Toward Automatic Hardware and Data Partitioning in HLS

Tzung-Han Juang
McGill University
tzung-han.juang@mail.mcgill.ca



Jonathan Van der Cruysse
McGill University
jonathan.vandercruysse@mail.mcgill.ca

Equality Saturation: Extraction and Outlining

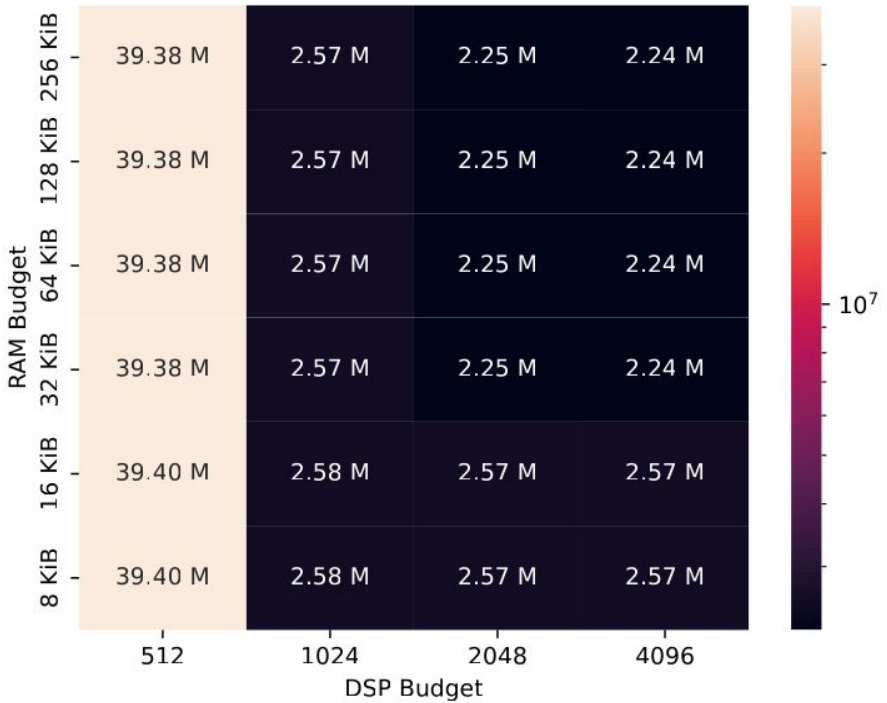


MatVec Skeleton

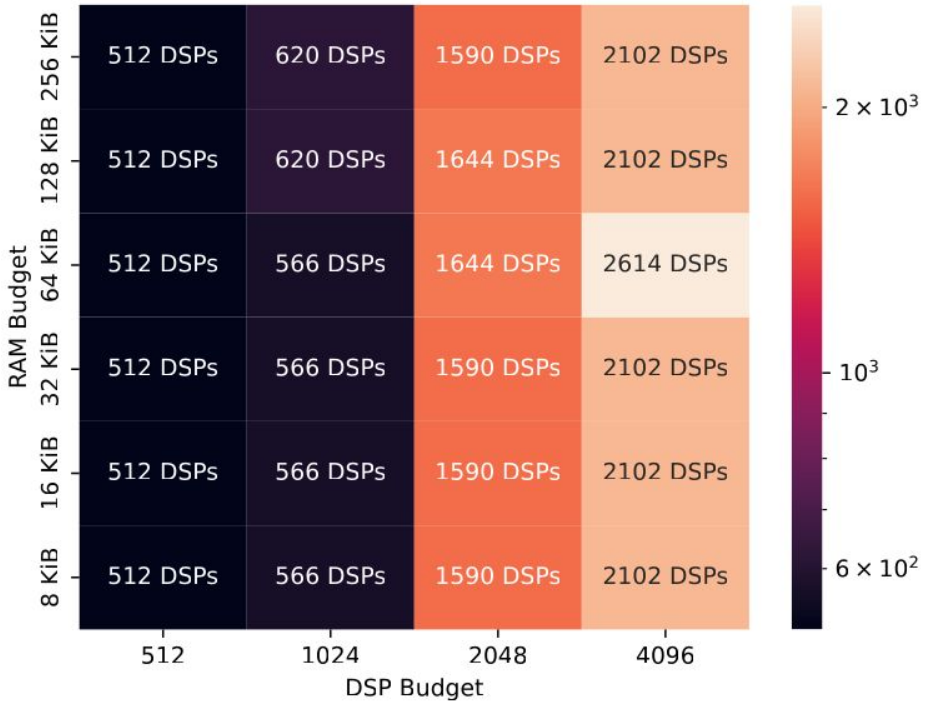
```
MatVec(A, b) = map(
  (a, b’) -> Dot(a, b’),
  zip(A, repeat(B)))
```

```
MatVecSk(A, b, f) = map(
  f,
  zip(A, repeat(B)))
```

Latency (in steps) per resource budget, first 4 layer of VGG



DSP use per resource budget, first 4 layer of VGG



RAM use per resource budget, first 4 layer of VGG