CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

C4CAM: A Compiler for CAM-based In-Memory Accelerators

Hamid Farzaneh, João Paulo C. de Lima, Mengyuan Li, Asif Ali Khan, X. Sharon Hu, Jeronimo Castrillon

Mondays in Memory (MiM) Webinars
January 20, 2025
Previously presented by Asif Ali Khan at ASPLOS'24

CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

C4CAM: A Compiler for CAM-based In-Memory Accelerators

Hamid Farzaneh, João Paulo C. de Lima, Mengyuan Li, Asif Ali Khan, X. Sharon Hu, Jeronimo Castrillon

Mondays in Memory (MiM) Webinars
January 20, 2025
Previously presented by Asif Ali Khan at ASPLOS'24

CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

C4CAM: A Compiler for CAM-based In-Memory Accelerators

Hamid Farzaneh, João Paulo C. de Lima, Mengyuan Li, Asif Ali Khan, X. Sharon Hu, Jeronimo Castrillon

Mondays in Memory (MiM) Webinars
January 20, 2025
Previously presented by Asif Ali Khan at ASPLOS'24

ScaDS.AI DRESDEN LEIPZIG

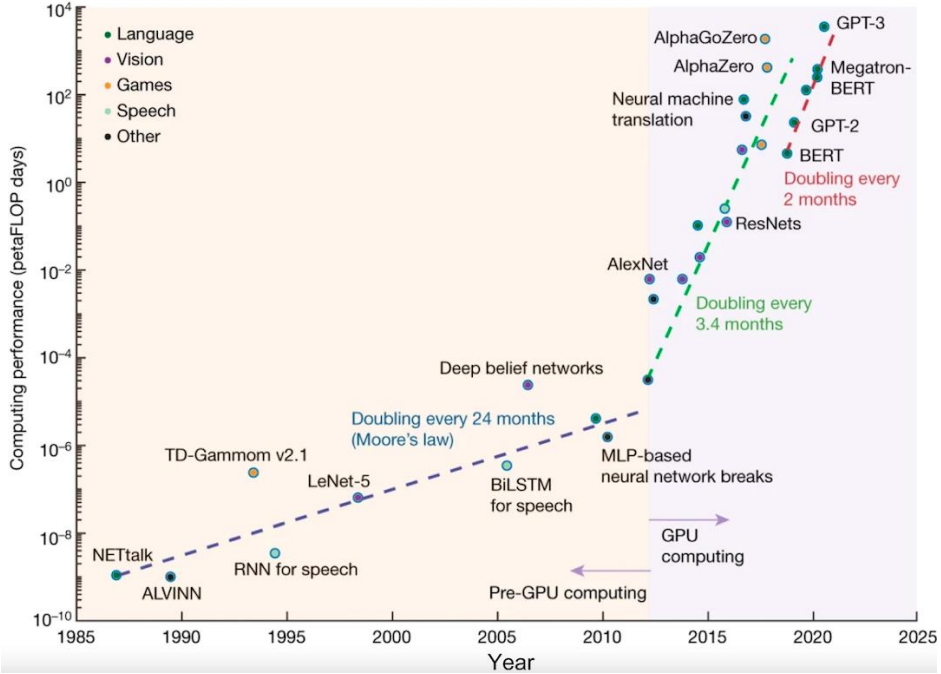
UNIVERSITY OF NOTRE DAME

TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

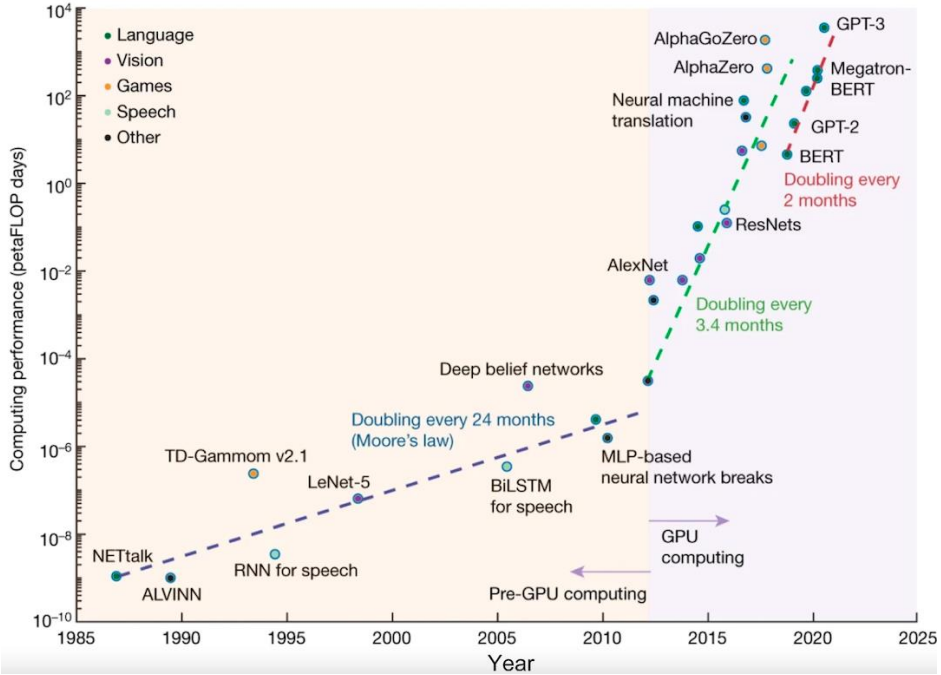
DFG WR WISSENSCHAFTSRAT

The rise in compute power demand



Aguirre et al, Nature Communications, 2024

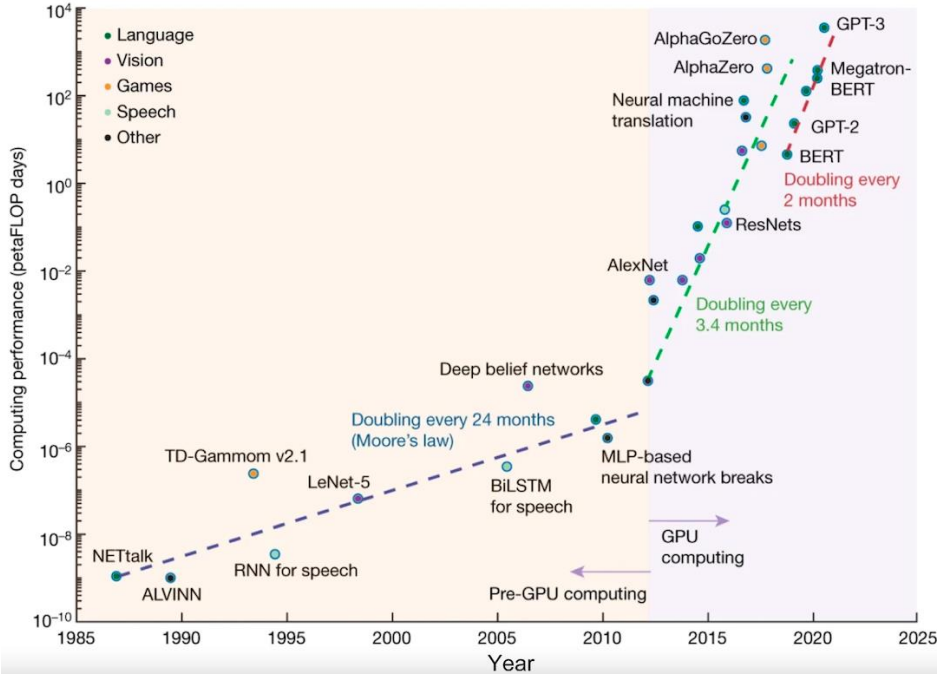
The rise in compute power demand



Aguirre et al, Nature Communications, 2024

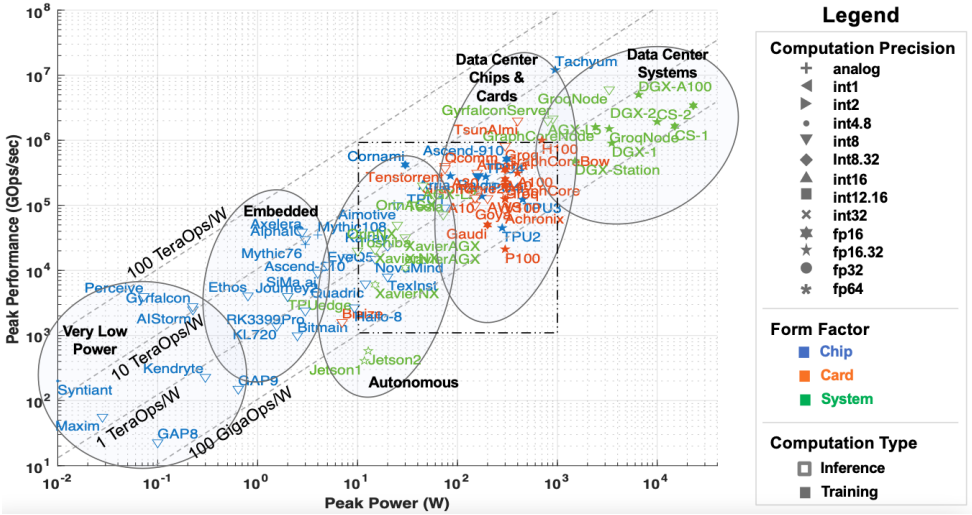
This unprecedented rise in computing power demand has fueled the emergence of novel architectures

The rise in compute power demand



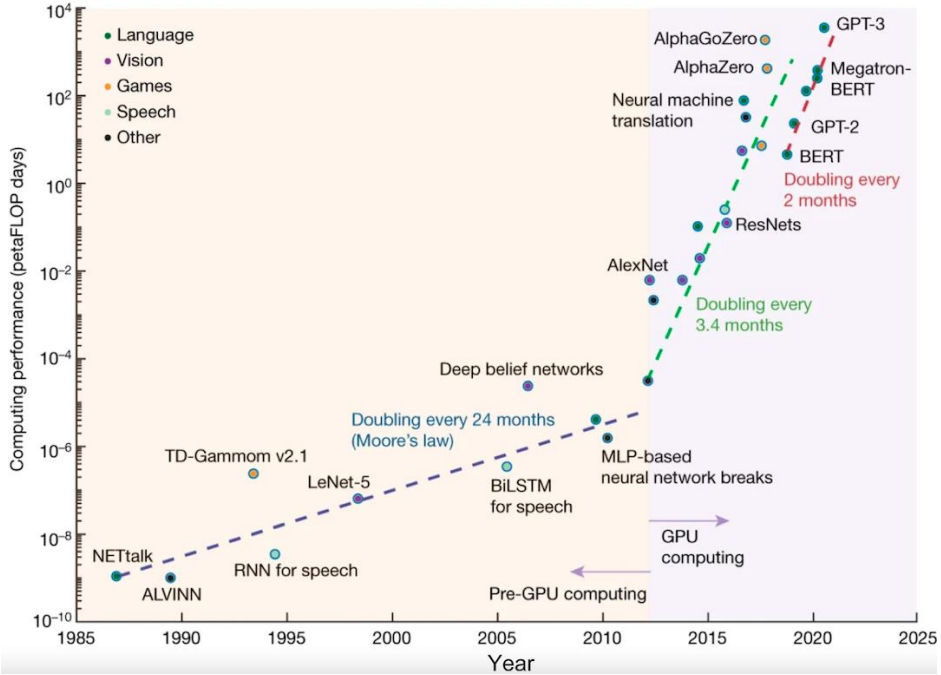
Aguirre et al, Nature Communications, 2024

This unprecedented rise in computing power demand has fueled the emergence of novel architectures

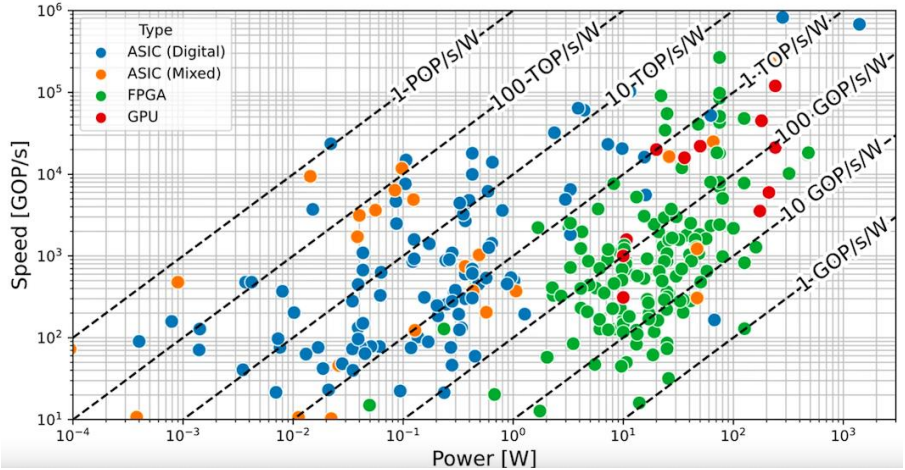


Reuther et al. IEEE HPEC, 2022

The rise in compute power demand

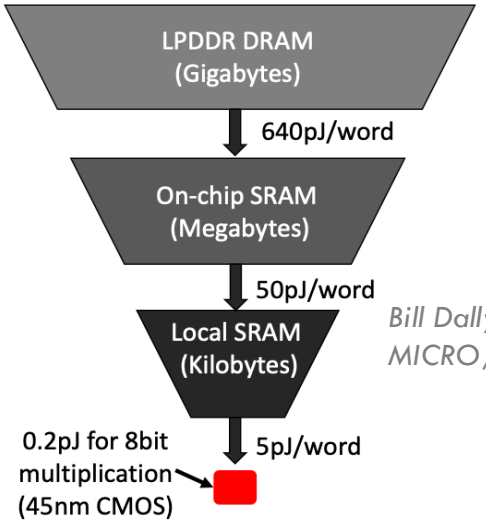


This unprecedented rise in computing power demand has fueled the emergence of novel architectures



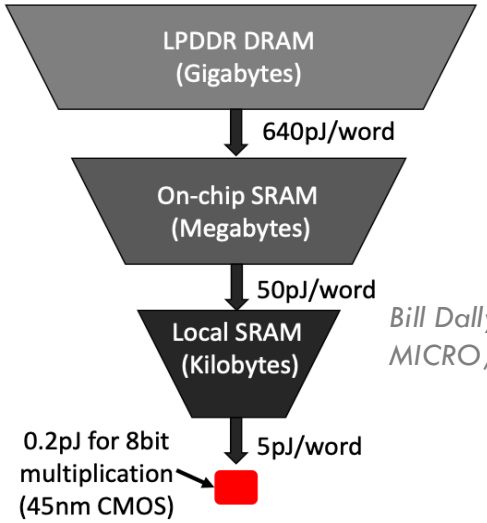
Aguirre et al, Nature Communications, 2024

Cost-driven accelerators design



Bill Dally, MICRO, 2019

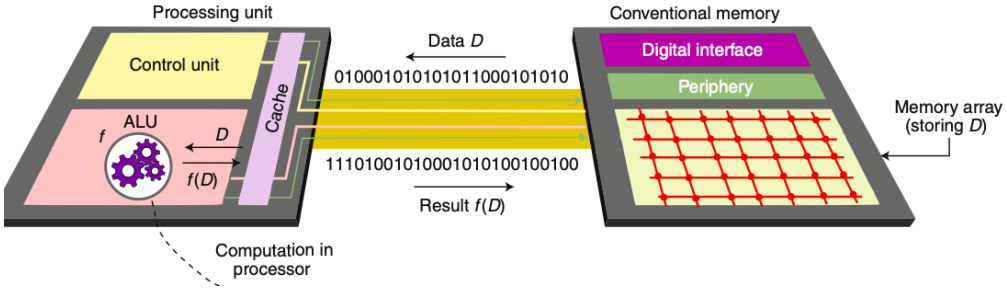
Cost-driven accelerators design

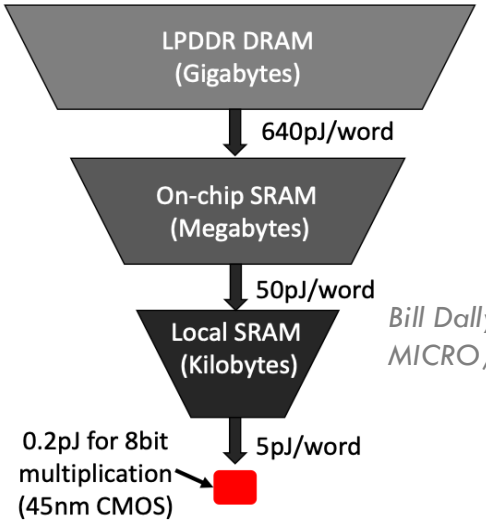


Bill Dally, MICRO, 2019

Computation is almost for free

Cost-driven accelerators design

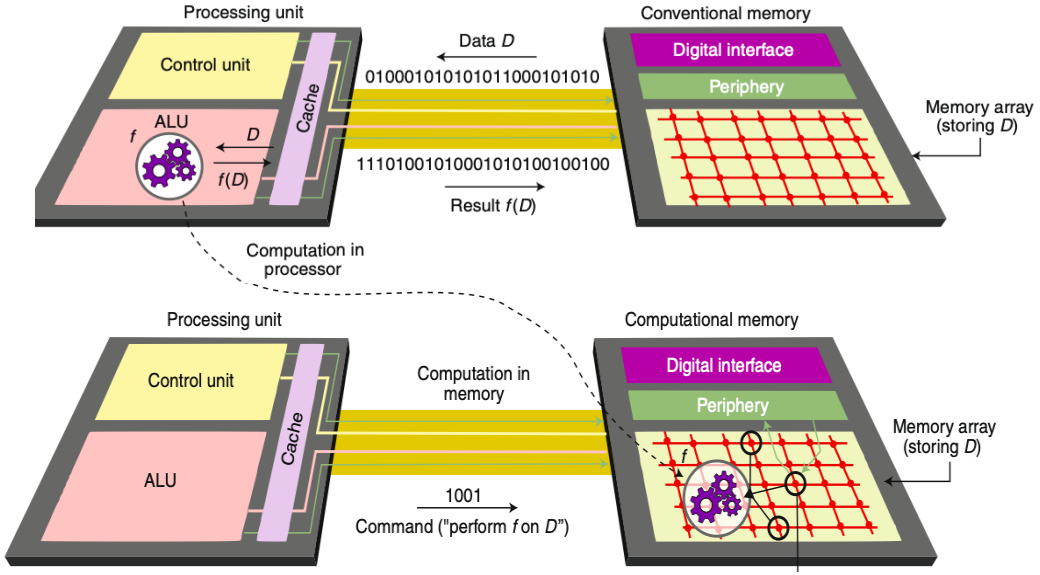




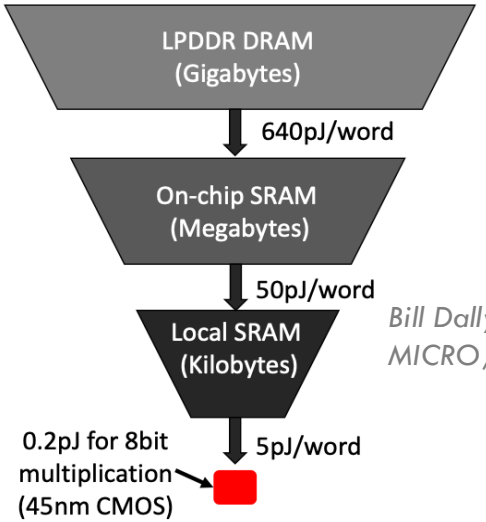
Bill Dally, MICRO, 2019

Computation is almost for free

Cost-driven accelerators design



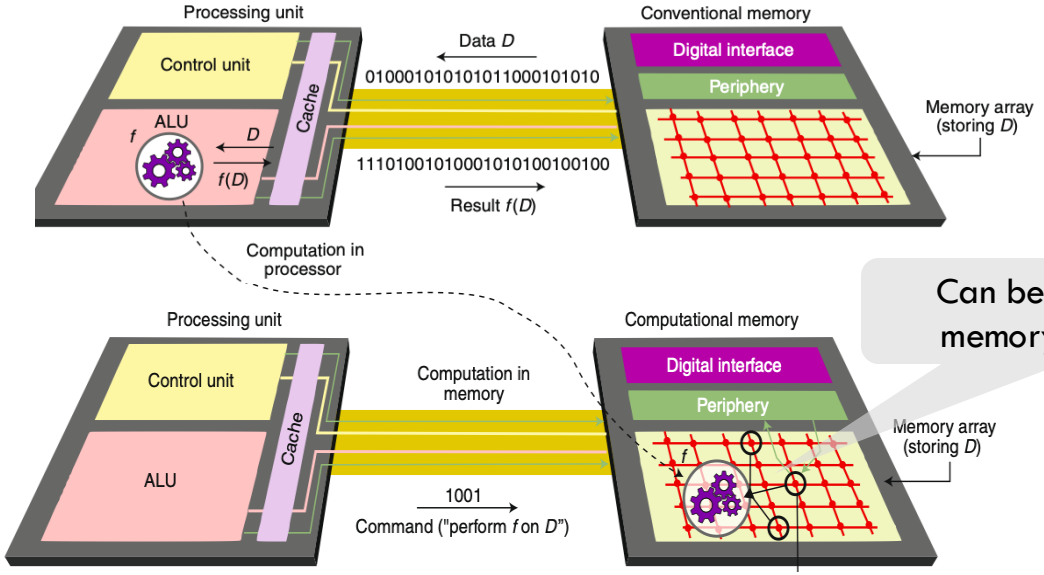
Abu Sebastian et al, Nature Nanotechnology, 2020



Bill Dally, MICRO, 2019

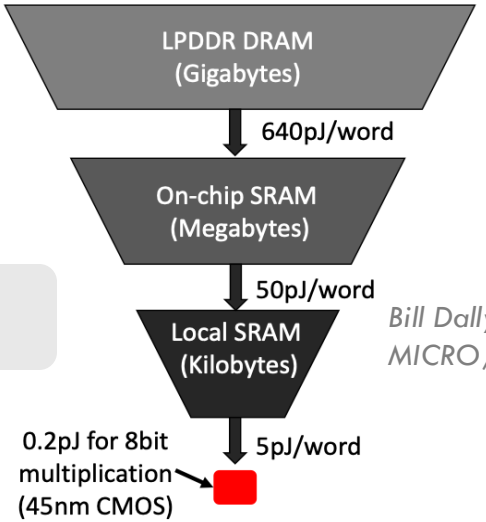
Computation is almost for free

Cost-driven accelerators design



Abu Sebastian et al, Nature Nanotechnology, 2020

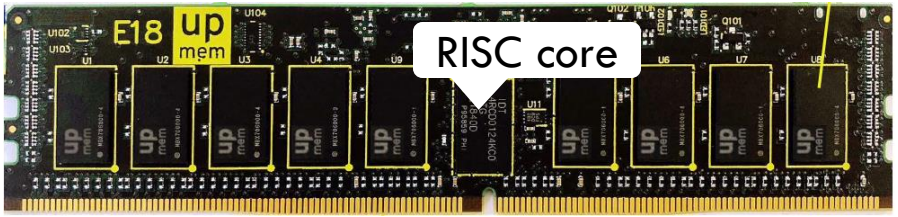
Can be compute near-memory or in-memory



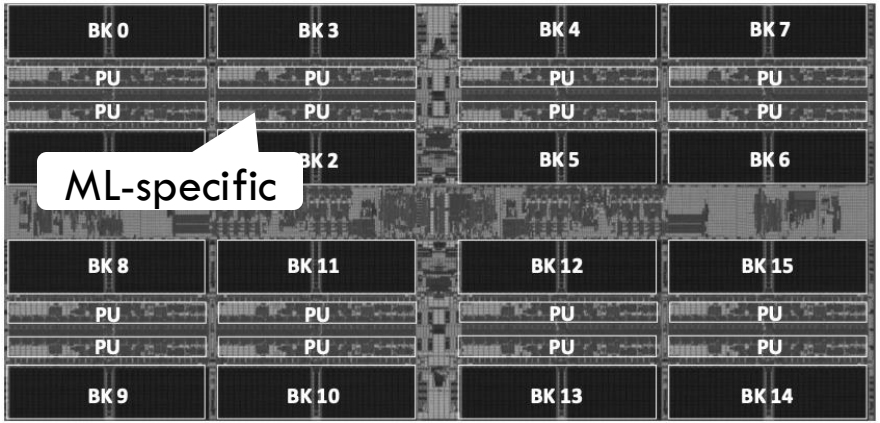
Bill Dally, MICRO, 2019

Computation is almost for free

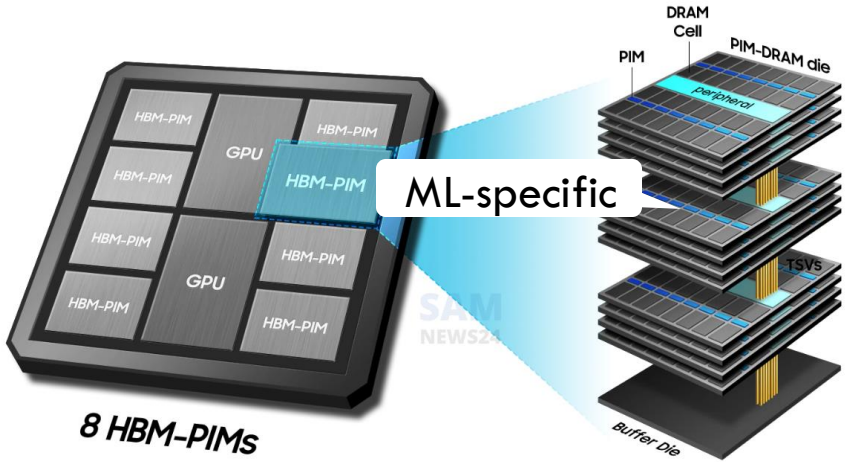
Computation near memory (CNM)



J. G. Luna et al, IEEE Access 2022 (UPMEM)



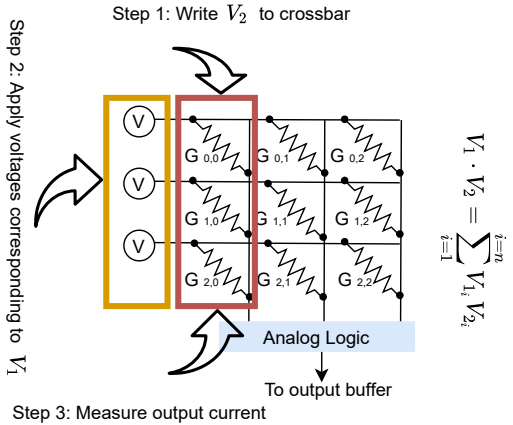
S. Lee et al., ISSCC 2022 (SK Hynix)



Samsung

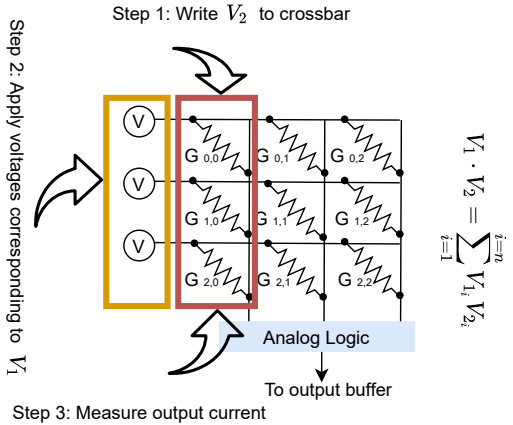
Computation in memory (CIM)

Memristive crossbar

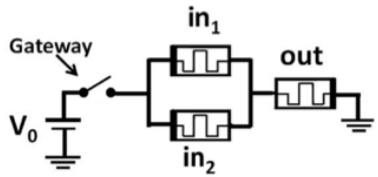


Computation in memory (CIM)

Memristive crossbar



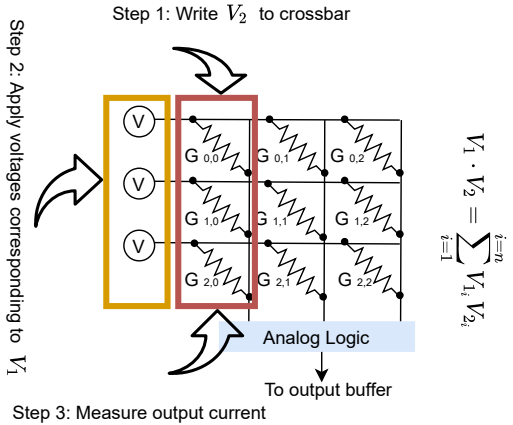
Boolean Logic



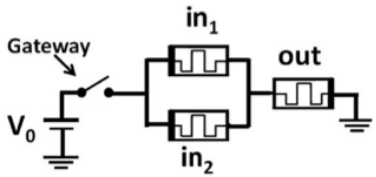
Kvatinsky et al, MAGIC, IEEE TCAS-II, 2014

Computation in memory (CIM)

Memristive crossbar

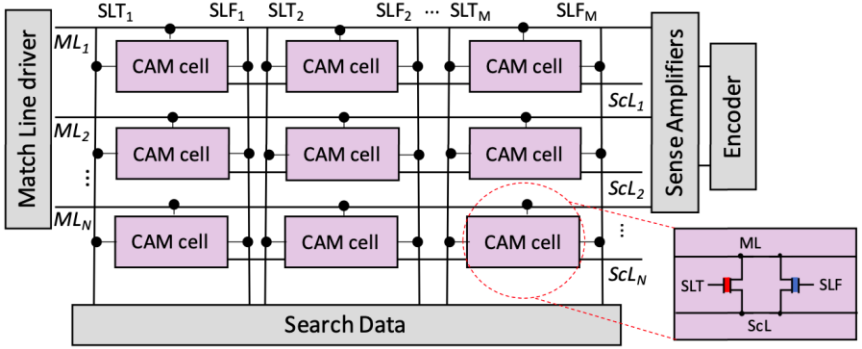


Boolean Logic



Kvatinsky et al, MAGIC, IEEE TCAS-II, 2014

Content Addressable Memory (CAM)

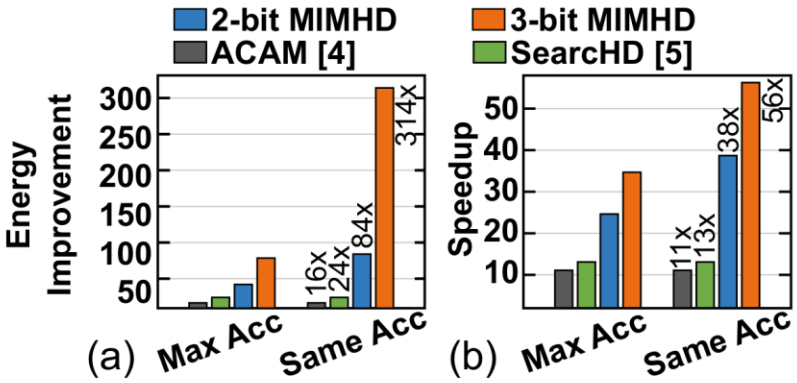


- Takes data, *searches* it in memory and outputs the matching address
- Many applications in ML and other domains

Asif Ali Khan, ASPLOS 2024.

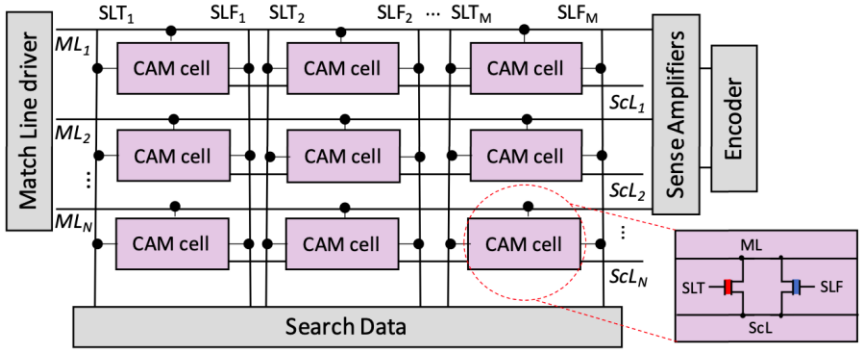
Computation in memory (CIM)

Orders of magnitude improvements



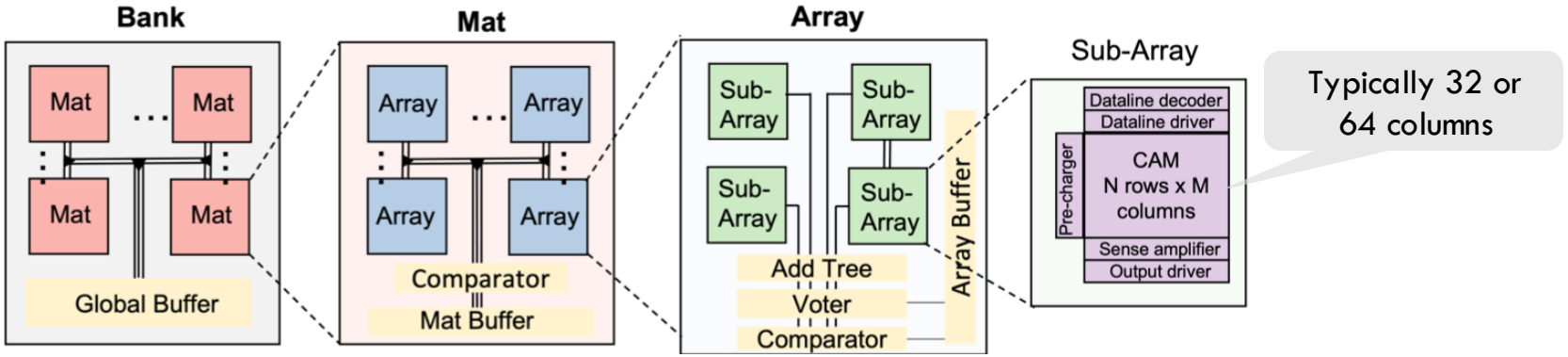
Kazemi, et al., Cross-Layer Design with Emerging Devices for Machine Learning Applications. 2023

Content Addressable Memory (CAM)



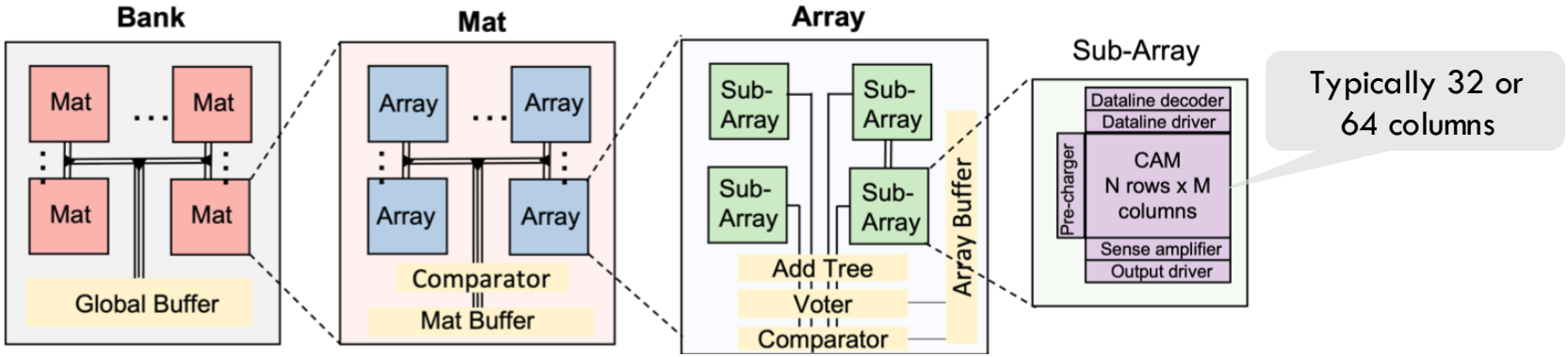
- Takes data, *searches* it in memory and outputs the matching address
- Many applications in ML and other domains

Content addressable memories



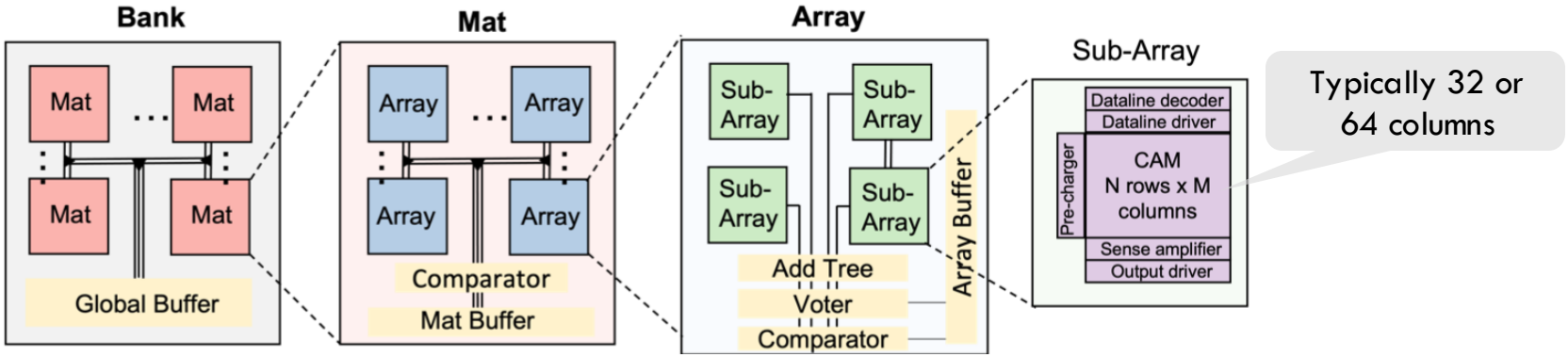
- Can be implemented with different memory technologies (FeFET in this work)

Content addressable memories



- Can be implemented with different memory technologies (FeFET in this work)
- Larger search vectors are first split into small chunks based on subarray sizes

Content addressable memories

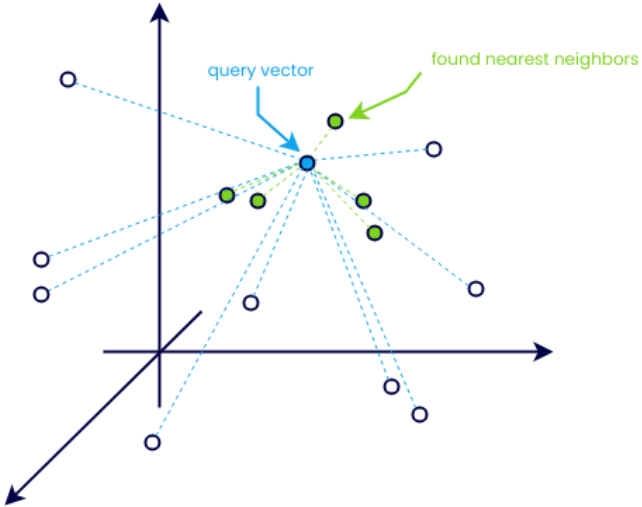


- Can be implemented with different memory technologies (FeFET in this work)
- Larger search vectors are first split into small chunks based on subarray sizes
- These chunks are then mapped to subarrays

CAMs programmability challenges

What the user writes

```
def forward(self, input: Tensor, dot: bool = False)
    ↪ -> Tensor:
    others = self.weight.transpose(-2, -1)
    matmul = torch.matmul(input, (others))
    values, indices = torch.ops.aten.topk(matmul, 1,
        ↪ largest=False)
    return indices
```



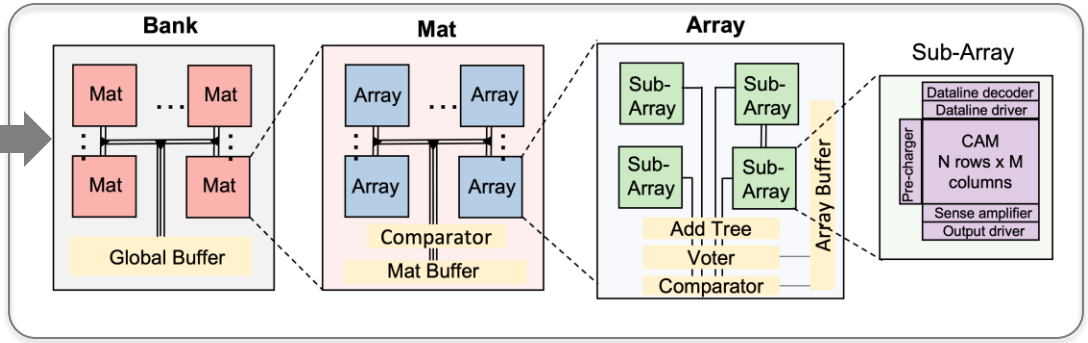
CAMs programmability challenges

What the user writes

```
def forward(self, input: Tensor, dot: bool = False)
    ↪ -> Tensor:
    others = self.weight.transpose(-2, -1)
    matmul = torch.matmul(input, (others))
    values, indices = torch.ops.aten.topk(matmul, 1,
        ↪ largest=False)
    return indices
```

What the device expects

```
…
cam_config = load_config()
cam = CAMASim(cam_config)
cam.write(CAM_Data)
CAM_pred_ids, _, _ = cam.query(CAM_Query)
return CAM_pred_ids
…
```



CAMs programmability challenges

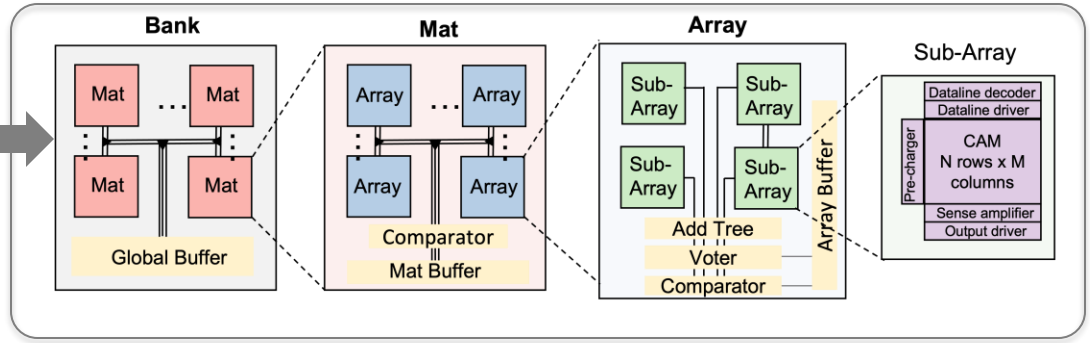
What the user writes

```
def forward(self, input: Tensor, dot: bool = False)
    ↪ -> Tensor:
    others = self.weight.transpose(-2, -1)
    matmul = torch.matmul(input, (others))
    values, indices = torch.ops.aten.topk(matmul, 1,
        ↪ largest=False)
    return indices
```

Only manual translation and mapping

What the device expects

```
…
cam_config = load_config()
cam = CAMASim(cam_config)
cam.write(CAM_Data)
CAM_pred_ids, _, _ = cam.query(CAM_Query)
return CAM_pred_ids
…
```



Challenges with manual designs

- Different CAM types and similarity metrics
  - TCAM, MCAM, ACAM
  - Hamming/Euclidean distance, dot-product/cosine similarity

Challenges with manual designs

- Different CAM types and similarity metrics
  - TCAM, MCAM, ACAM
  - Hamming/Euclidean distance, dot-product/cosine similarity
- Different cell precisions (single bit vs. multi bit): Tradeoffs in accuracy/perf

Challenges with manual designs

- Different CAM types and similarity metrics
  - TCAM, MCAM, ACAM
  - Hamming/Euclidean distance, dot-product/cosine similarity
- Different cell precisions (single bit vs. multi bit): Tradeoffs in accuracy/perf
- Different mappings have different impact on utilization, performance, energy

Challenges with manual designs

- Different CAM types and similarity metrics
  - TCAM, MCAM, ACAM
  - Hamming/Euclidean distance, dot-product/cosine similarity
- Different cell precisions (single bit vs. multi bit): Tradeoffs in accuracy/perf
- Different mappings have different impact on utilization, performance, energy
- Different merge operations

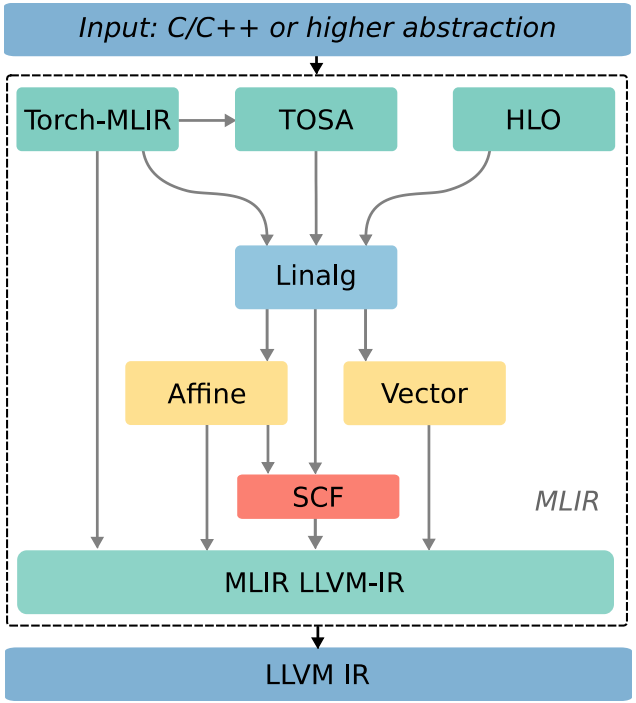
Challenges with manual designs

- Different CAM types and similarity metrics
  - TCAM, MCAM, ACAM
  - Hamming/Euclidean distance, dot-product/cosine similarity
- Different cell precisions (single bit vs. multi bit): Tradeoffs in accuracy/perf
- Different mappings have different impact on utilization, performance, energy
- Different merge operations
- Presently all of this is handled manually with low-level APIs, restricting CAMs usability to device experts

C4CAM to the rescue

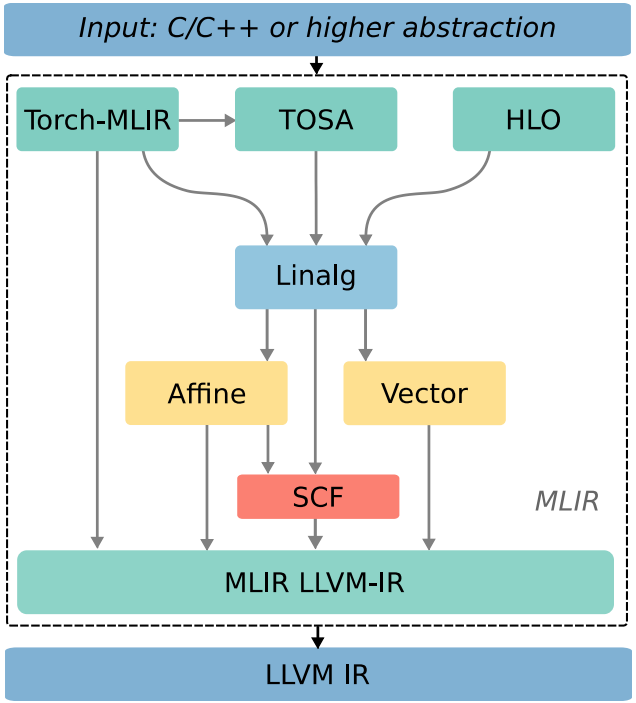
The MLIR ecosystem

- Allows representing and transforming intermediate representations at different abstraction levels (dialects)



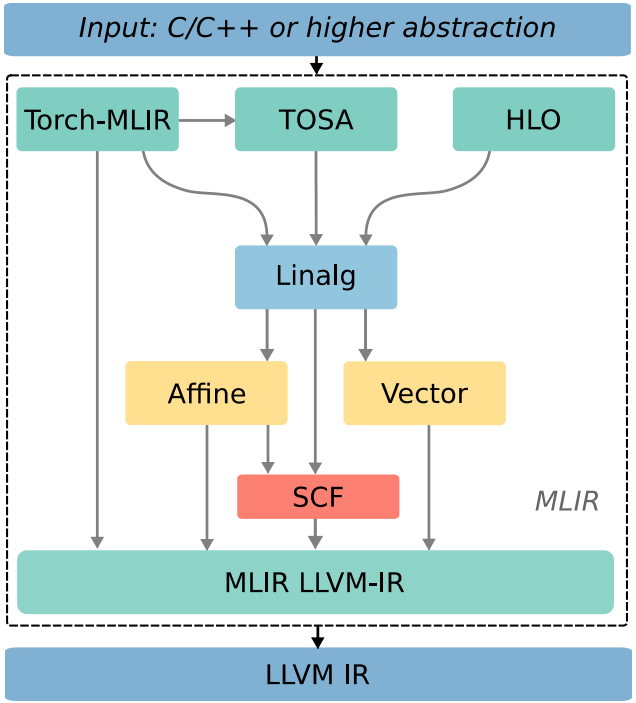
The MLIR ecosystem

- Allows representing and transforming intermediate representations at different abstraction levels (dialects)
- Allows defining your own operations and dialects

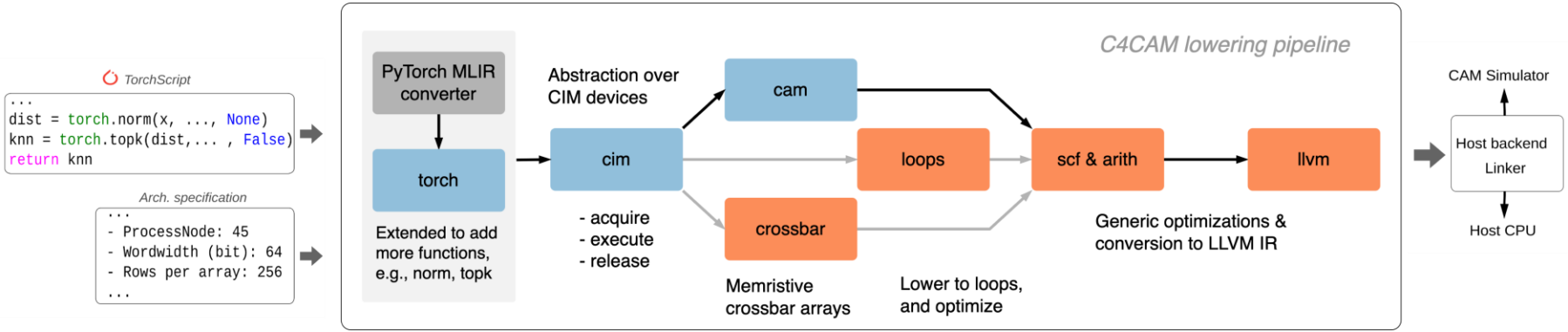


The MLIR ecosystem

- Allows representing and transforming intermediate representations at different abstraction levels (dialects)
- Allows defining your own operations and dialects
- Significantly simplifies compilation for heterogeneous hardware

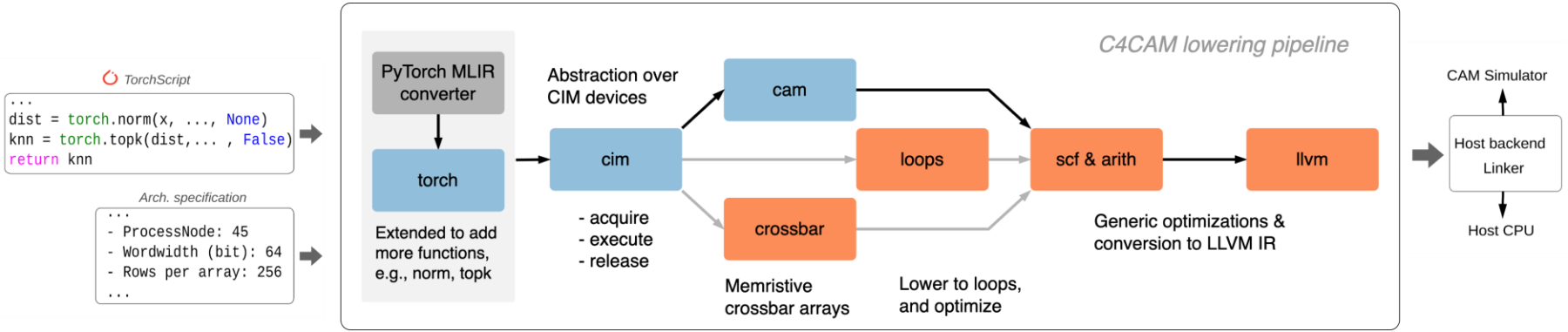


C4CAM: An end-to-end compilation flow for CAMs



- Takes a high-level device-agnostic representation and arch. specification

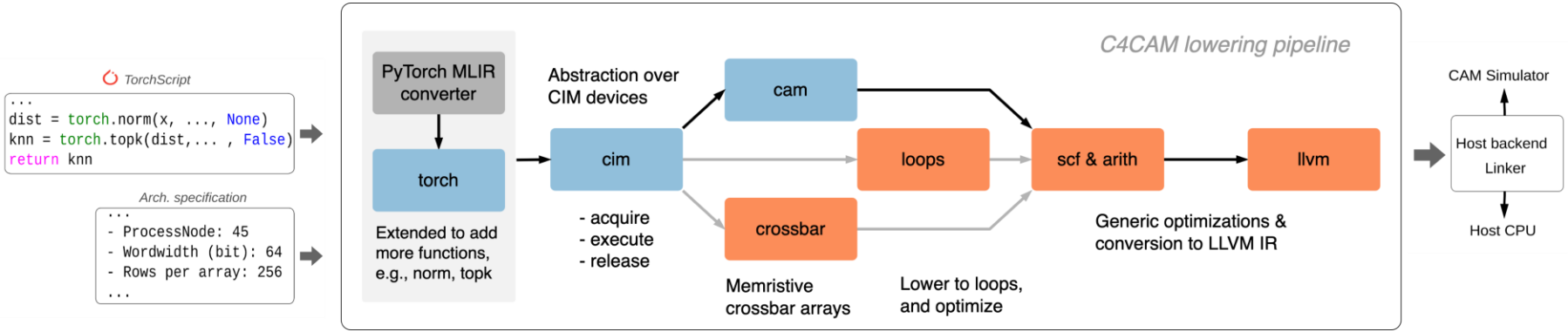
C4CAM: An end-to-end compilation flow for CAMs



- Takes a high-level device-agnostic representation and arch. specification
- `cim*` performs pattern-matching and rewriting

*Khan et al. Cinm (cinnamon): A compilation infrastructure for heterogeneous compute in-memory and compute near-memory paradigms. To appear in ASPLOS'25.

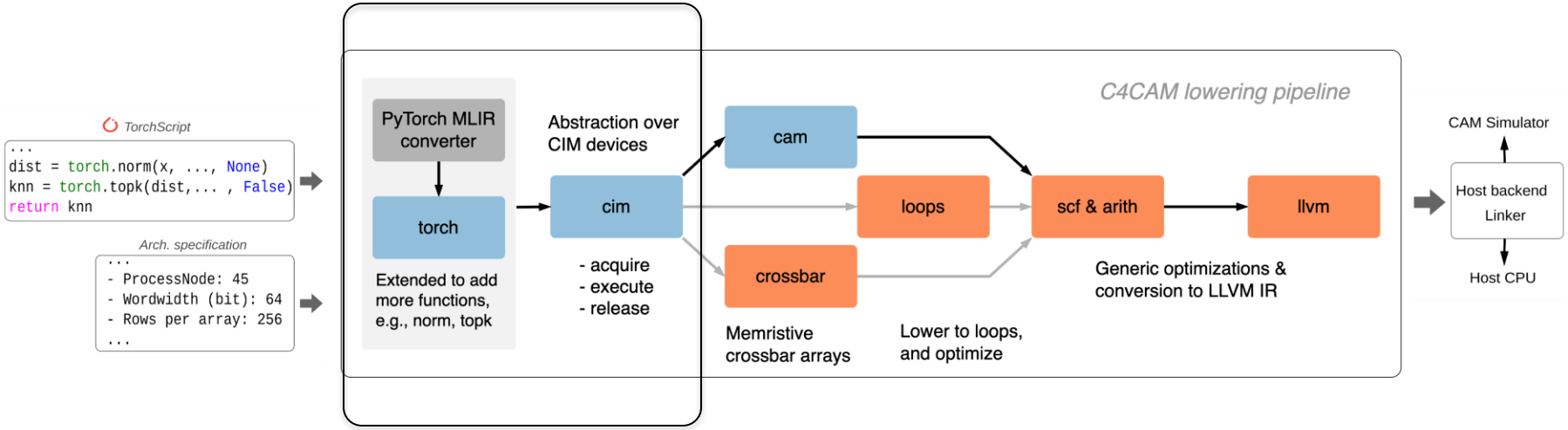
C4CAM: An end-to-end compilation flow for CAMs



- Takes a high-level device-agnostic representation and arch. specification
- `cim*` performs pattern-matching and rewriting
- `cam` leverages device experts' knowledge and optimize for it

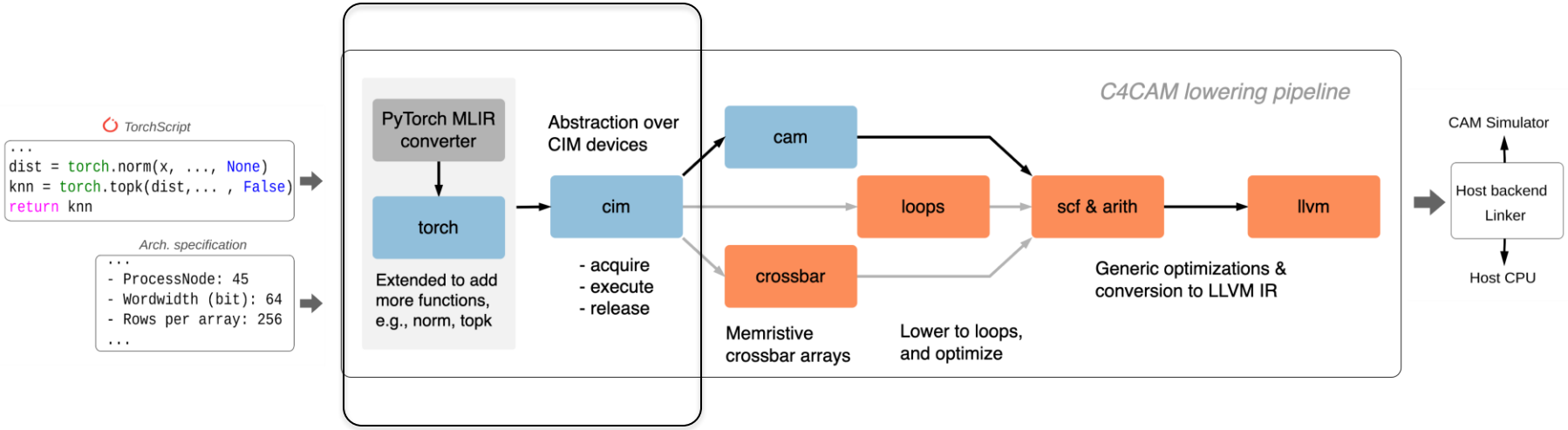
*Khan et al. Cinm (cinnamon): A compilation infrastructure for heterogeneous compute in-memory and compute near-memory paradigms. To appear in ASPLOS'25.

C4CAM: Rewriting and transformations



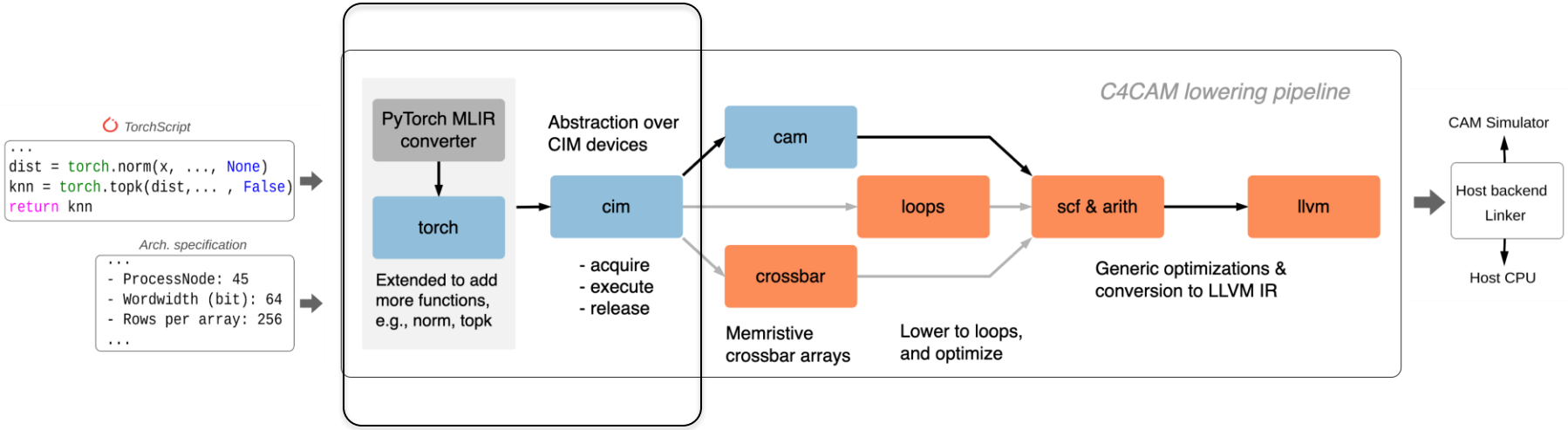
- Extended `torch-mlir` to support `topk` and `norm` operations

C4CAM: Rewriting and transformations



- Extended `torch-mlir` to support `topk` and `norm` operations
- The `cim` dialect finds search patterns and rewrite them

C4CAM: Rewriting and transformations



- Extended `torch-mlir` to support `topk` and `norm` operations
- The `cim` dialect finds search patterns and rewrites them
- Partition operands to fit onto CAM arrays

Rewriting similarity search operations

```
...
%4 = cim.acquire : index
%5:2 = cim.execute(%4, %2, %0, %3) ({
  %7 = cim.transpose %2 : tensor<10x8192xf32>
    -> tensor<8192x10xf32>
  %8 = cim.matmul %0, %7 : tensor<10x8192xf32>,
    tensor<8192x10xf32>
     -> tensor<10x10xf32>
  %values, %indices = cim.topk %8, %3 : tensor<10x10xf32>
    , i64 -> tensor<10x1xf32>, tensor<10x1xf32>
  cim.yield %values, %indices : tensor<10x1xf32>, tensor<10x1xf32>
}) : (index, tensor<10x8192xf32>, tensor<10x8192xf32>, i64)
  -> (tensor<10x1xf32>, tensor<10x1xf32>)
cim.release %4 : index
...
```

Rewriting similarity search operations

```
...
%4 = cim.acquire : index
%5:2 = cim.execute(%4, %2, %0, %3) ({
  %7 = cim.transpose %2 : tensor<10x8192xf32>
    -> tensor<8192x10xf32>
  %8 = cim.matmul %0, %7 : tensor<10x8192xf32>,
    tensor<8192x10xf32>
     -> tensor<10x10xf32>
  %values, %indices = cim.topk %8, %3 : tensor<10x10xf32>
    , i64 -> tensor<10x1xf32>, tensor<10x1xf32>
  cim.yield %values, %indices : tensor<10x1xf32>, tensor<10x1xf32>
}) : (index, tensor<10x8192xf32>, tensor<10x8192xf32>, i64)
  -> (tensor<10x1xf32>, tensor<10x1xf32>)
cim.release %4 : index
...
```

/* Pattern matching for dot product similarity */

Replace op<*topk*> (op<*matmul*> (arg2, op<*transpose*> (arg1)), arg3)
with op <*similarity*> (*dot*, arg1, arg2, arg3);

/* Pattern matching for Euclidean distance */

Replace op <*topk*>(op<*norm*> (op<*sub*> (arg1, arg2)), arg3)
with op <*similarity*> (*euc*, arg1, arg2, arg3);

/* Pattern matching for cosine similarity */

Replace op <*smulmat*>(op <*div*> (cons1, op<*mul*>(op<*norm*>(arg1), op<*norm*> (arg2))), op<*matmul*>(arg1, op<*transpose*>(arg2)))
with op <*similarity*> (*cos*, arg1, arg2);

/* Pattern matching for Hamming distance */

Replace op <*nonzero*>(op <*cmp*>(*lt*, op <*popcount*>(op <*xor*>(arg1, arg2), arg3)
with op <*similarity*> (*ham*, arg1, arg2, arg3);

Rewriting similarity search operations

```
...
%4 = cim.acquire : index
%5:2 = cim.execute(%4, %2, %0, %3) ({
  %7 = cim.transpose %2 : tensor<10x8192xf32>
    -> tensor<8192x10xf32>
  %8 = cim.matmul %0, %7 : tensor<10x8192xf32>,
    tensor<8192x10xf32>
     -> tensor<10x10xf32>
  %values, %indices = cim.topk %8, %3 : tensor<10x10xf32>
    , i64 -> tensor<10x1xf32>, tensor<10x1xf32>
  cim.yield %values, %indices : tensor<10x1xf32>, tensor<10x1xf32>
}) : (index, tensor<10x8192xf32>, tensor<10x8192xf32>, i64)
  -> (tensor<10x1xf32>, tensor<10x1xf32>)
cim.release %4 : index
...
```

```
...
%4 = cim.acquire : index
%5:2 = cim.execute(%4, %2, %0, %3) ({
  %values, %indices = cim.similarity dot %2,
    %0, %3 : tensor<10x8192xf32>, tensor<10x8192xf32>,
    i64 -> tensor<10x1xf32>, tensor<10x1xf32>
  cim.yield %values, %indices : tensor<10x1xf32>, tensor<10x1xf32>
}) : (index, tensor<10x8192xf32>, tensor<10x8192xf32>, i64)
  -> (tensor<10x1xf32>, tensor<10x1xf32>)
cim.release %4 : index
...
```

```
/* Pattern matching for dot product similarity */
Replace op<topk> (op<matmul> (arg2, op<transpose> (arg1)), arg3)
  with op <similarity> (dot, arg1, arg2, arg3);

/* Pattern matching for Euclidean distance */
Replace op <topk>(op<norm> (op<sub> (arg1, arg2)), arg3)
  with op <similarity> (euc, arg1, arg2, arg3);

/* Pattern matching for cosine similarity */
Replace op <smulmat>(op <div> (cons1, op<mul>(op<norm>(arg1),
  op<norm> (arg2))), op<matmul>(arg1, op<transpose>(arg2)))
  with op <similarity> (cos, arg1, arg2);

/* Pattern matching for Hamming distance */
Replace op <nonzero>(op <cmp>(lt, op <popcount>(op <xor>(arg1,
  arg2), arg3)
  with op <similarity> (ham, arg1, arg2, arg3);
```

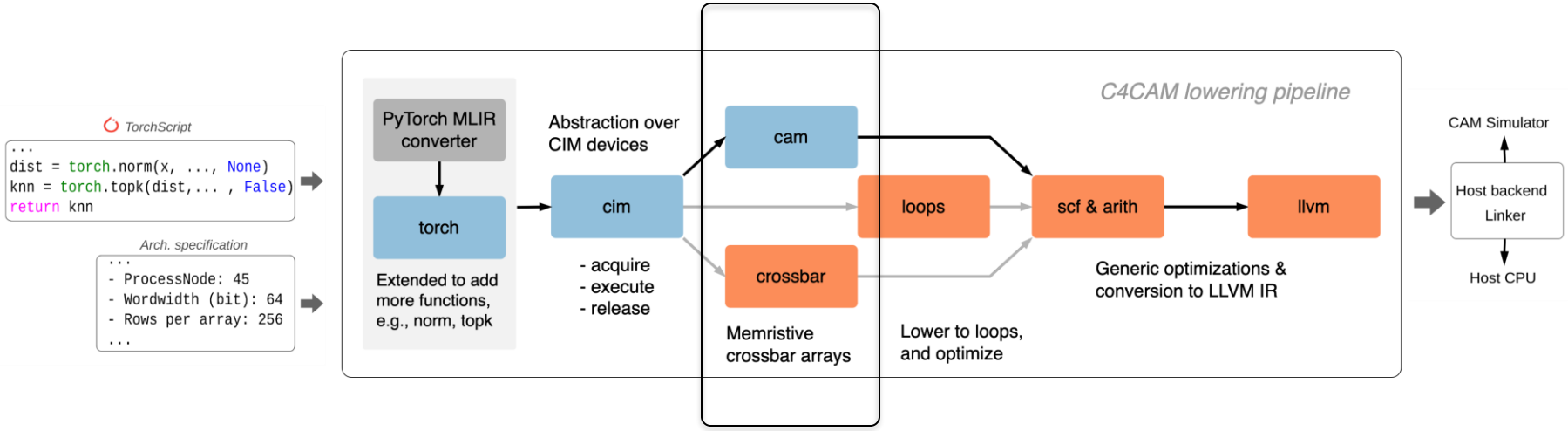
Rewriting similarity search operations

```
...
%4 = cim.acquire : index
%5:2 = cim.execute(%4, %2, %0, %3) ({
  %7 = cim.transpose %2 : tensor<10x8192xf32>
    -> tensor<8192x10xf32>
  %8 = cim.matmul %0, %7 : tensor<10x8192xf32>,
    tensor<8192x10xf32>
     -> tensor<10x10xf32>
  %values, %indices = cim.topk %8, %3 : tensor<10x10xf32>
    , i64 -> tensor<10x1xf32>, tensor<10x1xf32>
  cim.yield %values, %indices : tensor<10x1xf32>, tensor<10x1xf32>
}) : (index, tensor<10x8192xf32>, tensor<10x8192xf32>, i64)
  -> (tensor<10x1xf32>, tensor<10x1xf32>)
cim.release %4 : index
...
```

```
...
%4 = cim.acquire : index
%5:2 = cim.execute(%4, %2, %0, %3) ({
  %values, %indices = cim.similarity dot %2,
    %0, %3 : tensor<10x8192xf32>, tensor<10x8192xf32>,
    i64 -> tensor<10x1xf32>, tensor<10x1xf32>
  cim.yield %values, %indices : tensor<10x1xf32>, tensor<10x1xf32>
}) : (index, tensor<10x8192xf32>, tensor<10x8192xf32>, i64)
  -> (tensor<10x1xf32>, tensor<10x1xf32>)
cim.release %4 : index
...
```

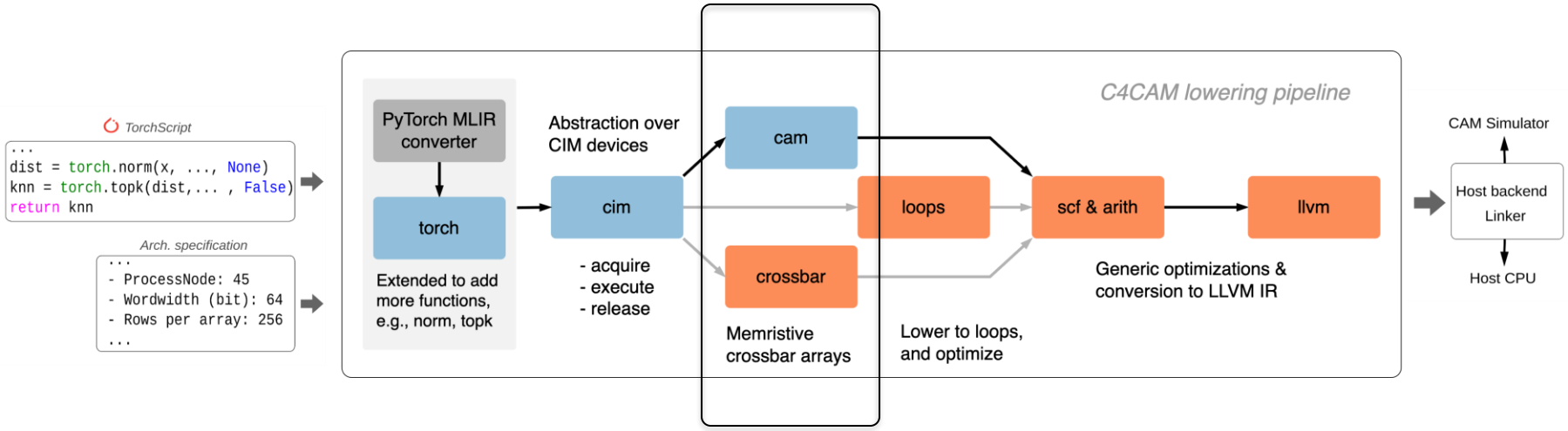
```
scf.for %arg1 = %c0 to %c8192 step %c32 {
  %extr_slice = tensor.extract_slice %2[0, %arg1] [10, 32]
    [1, 1] : tensor<10x8192xf32> to tensor<10x32xf32>
  %extr_slice_0 = tensor.extract_slice %0[0, %arg1] [10, 32]
    [1, 1] : tensor<10x8192xf32> to tensor<10x32xf32>
  %7 = cim.acquire : index
  %8:2 = cim.execute(%7, %extr_slice, %extr_slice_0, %3) ({
    %values, %indices = cim.similarity dot  %extr_slice,
      %extr_slice_0, %3 : tensor<10x32xf32>, tensor<10x32xf32>,
      i64 -> tensor<10x1xf32>, tensor<10x1xf32>
    cim.yield %values, %indices : tensor<10x1xf32>,
      tensor<10x1xf32>}) : (index, tensor<10x32xf32>,
      tensor<10x32xf32>, i64) -> (tensor<10x1xf32>,
      tensor<10x1xf32>)
  %9 = cim.merge_partial values  similarity dot  horizontal
    %7, %4, %8#0 : index, tensor<10x1xf32>, tensor<10x1xf32>
    -> tensor<10x1xf32>
  ...
  cim.release %7 : index
}
```

C4CAM: Mapping to CAM arrays



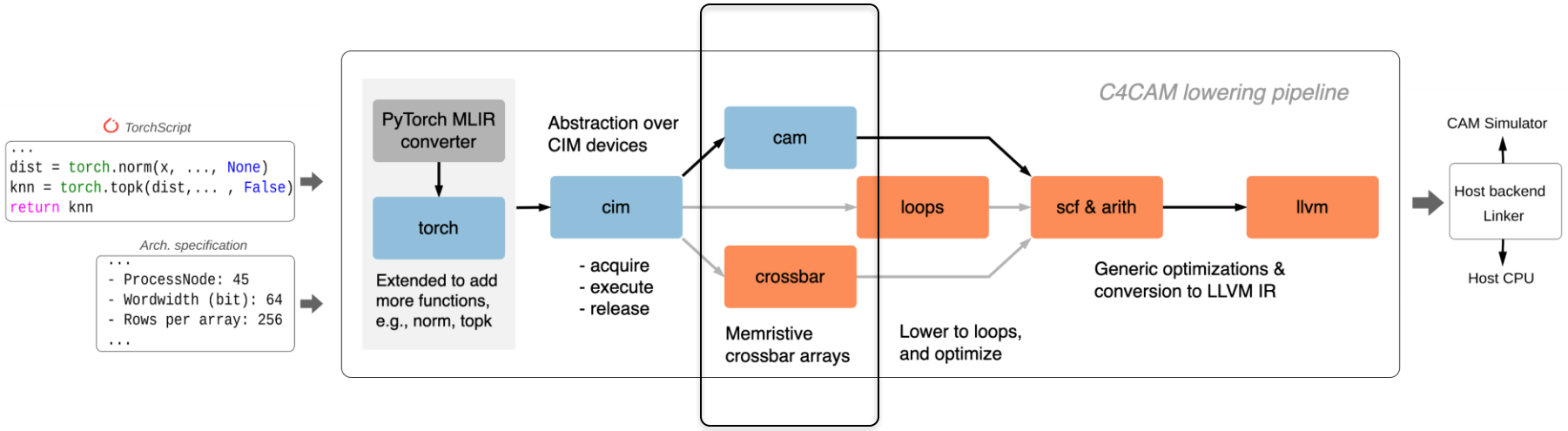
- `cam` maps the split search vectors to CAM arrays

C4CAM: Mapping to CAM arrays



- `cam` maps the split search vectors to CAM arrays
- Mapping chunks of the a vector to the same subarray requires multi cycles

C4CAM: Mapping to CAM arrays



- `cam` maps the split search vectors to CAM arrays
- Mapping chunks of the a vector to the same subarray requires multi cycles
- Also implements merge operation

C4CAM: Mapping to CAM arrays

```
%bank_values_buffer = memref.alloc() : memref<2x10x1xf32>
%bank_indices_buffer = memref.alloc() : memref<2x10x1xf32>
...
scf.parallel (%arg1) = (%c0) to (%c8192) step (%c4096) {
  %1 = cam.alloc_bank %c32, %c32 : index, index -> cam.bank_id
  ...
  scf.parallel (%arg2) = (%c0) to (%c4096) step (%c1024) {
    %4 = cam.alloc_mat %1 : cam.bank_id -> cam.mat_id
    ...
    scf.parallel (%arg3) = (%c0) to (%c1024) step (%c256) {
      %7 = cam.alloc_array %4 : cam.mat_id -> cam.array_id
      ...
      scf.parallel (%arg4) = (%c0) to (%c256) step (%c32) {
        ...
        %12 = cam.alloc_subarray %7 : cam.array_id -> cam.
            ↪ subarray_id
        cam.write_value %12, %subarray_data_buffer :
          cam.subarray_id, memref<10x32xf32>
        cam.search  exact  eucl %12, %subarray_query_buffer :
          cam.subarray_id, memref<1x32xf32>
        %13:2 = cam.read exact %12 : cam.subarray_id
          -> memref<10x1xf32>, memref<10x1xf32>
        ... }
      ...}
    ...}
  ...}
...
%value_res = cam.merge_partial bank values horizontal %1,
  %bank_values_buffer :cam.bank_id, memref<2x10x1xf32>
   -> memref<10x1xf32>
...
```

- One possible mapping (optimized for latency)
- All chunks goes to different subarrays that can run in parallel

Evaluation

Experimental setup

- Simulation environment: CAMASim
- NVIDIA RTX 3090 GPU

Table 1. Simulator configuration

Architecture & Circuit configuration			
Type	**HDC**	**KNN**	**DNA**
Horizontal merge	Voting	Voting	Counter
Vertical merge	Comparator	Comparator	Gather
Cell	TCAM	T/MCAM	TCAM
Sensing circuit	BE	BE	TH

Cost of additional circuits		
Type	**Latency**	**Energy**
Adder	0.25 ns	1.3 fJ/bit
Register	0.5 ns	4.5 fJ/bit
Comparator	0.25 ns	0.4 fJ/bit
Decoder/Encoder	0.25 ns	29 fJ

Experimental setup

- Simulation environment: CAMASim
- NVIDIA RTX 3090 GPU
- **Use-cases:** K-NN, DNA mapping, Hyperdimensional computing (HDC)

Table 1. Simulator configuration

Architecture & Circuit configuration			
Type	**HDC**	**KNN**	**DNA**
Horizontal merge	Voting	Voting	Counter
Vertical merge	Comparator	Comparator	Gather
Cell	TCAM	T/MCAM	TCAM
Sensing circuit	BE	BE	TH

Cost of additional circuits		
Type	**Latency**	**Energy**
Adder	0.25 ns	1.3 fJ/bit
Register	0.5 ns	4.5 fJ/bit
Comparator	0.25 ns	0.4 fJ/bit
Decoder/Encoder	0.25 ns	29 fJ

Experimental setup

- Simulation environment: CAMASim
- NVIDIA RTX 3090 GPU
- **Use-cases:** K-NN, DNA mapping, Hyperdimensional computing (HDC)
- **Datasets:** MNIST (HDC), human genome (DNA mapping), Iris-Wine-Cancer-WineQuality (K-NN)
- **Configurations:** Multiple bit-precisions, different opt targets

Table 1. Simulator configuration

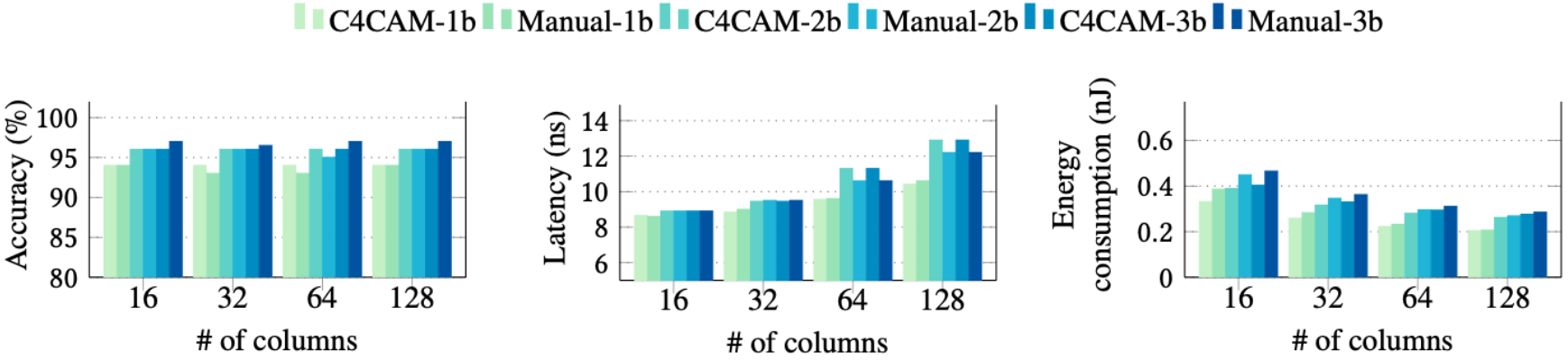
Architecture & Circuit configuration

Type	HDC	KNN	DNA
Horizontal merge	Voting	Voting	Counter
Vertical merge	Comparator	Comparator	Gather
Cell	TCAM	T/MCAM	TCAM
Sensing circuit	BE	BE	TH

Cost of additional circuits

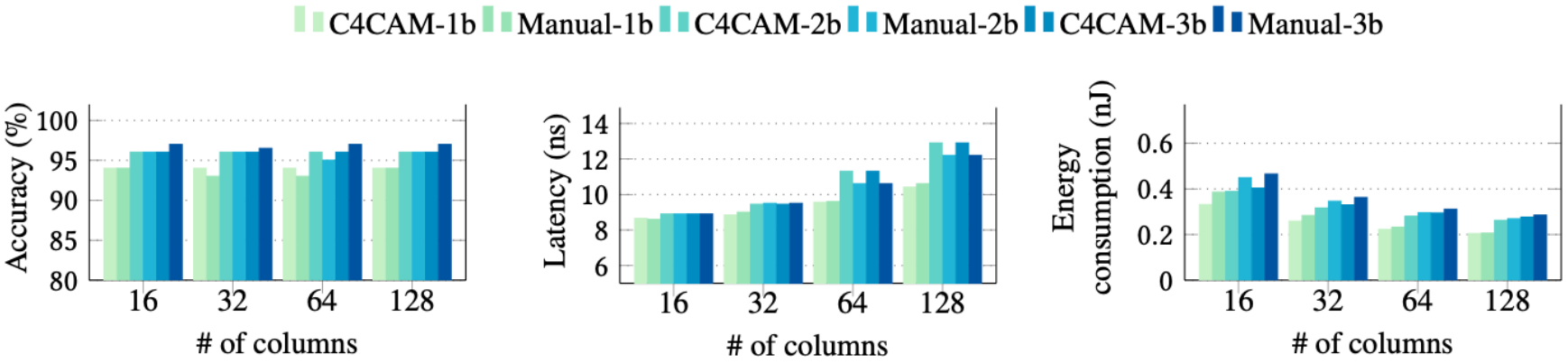
Type	Latency	Energy
Adder	0.25 ns	1.3 fJ/bit
Register	0.5 ns	4.5 fJ/bit
Comparator	0.25 ns	0.4 fJ/bit
Decoder/Encoder	0.25 ns	29 fJ

Results validation



Results validation

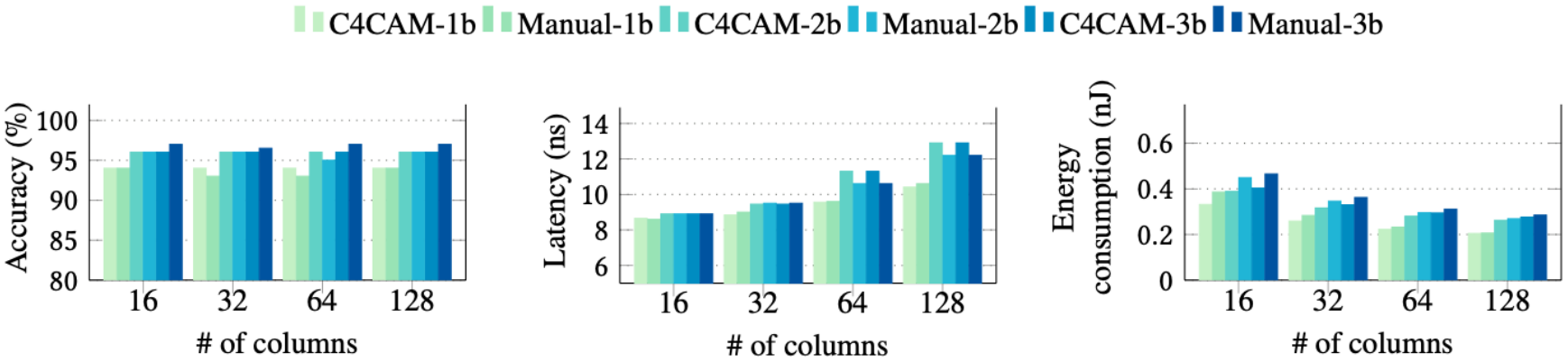
C4CAM vs manual



Results validation

C4CAM vs manual

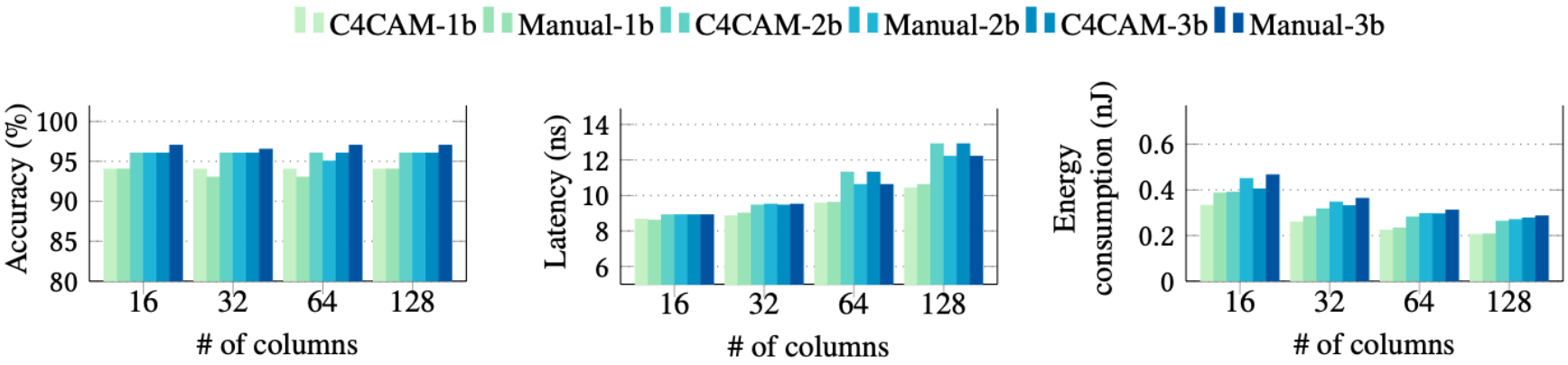
Different precisions



Results validation

C4CAM vs manual

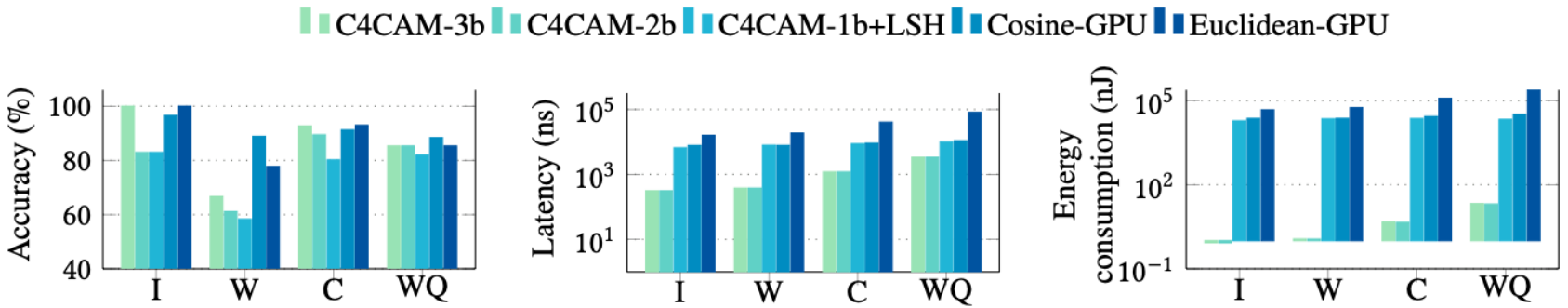
Different precisions



- Comparable accuracy (1% difference)
- Latency increases with the array size
- 1 bit/cell and larger array sizes config consume less energy

GPU comparison

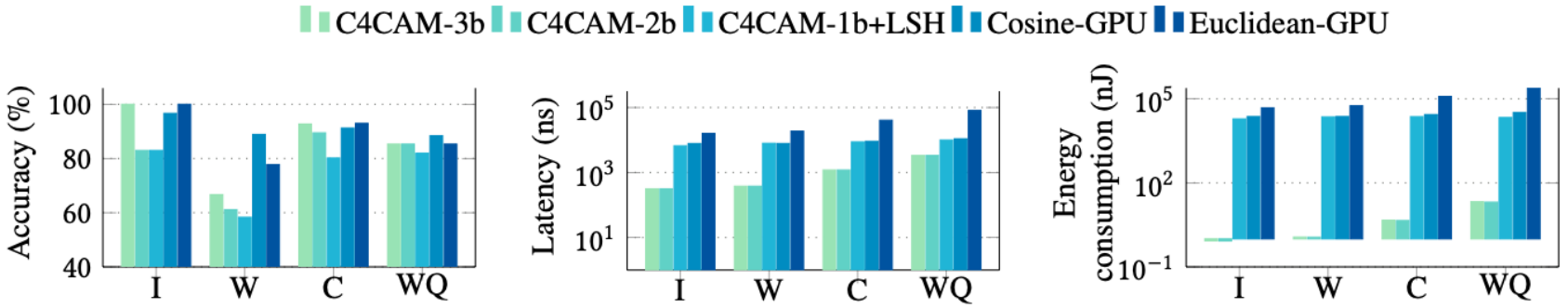
- K-NN: CAM array size is 128x128



Asif Ali Khan, ASPLOS 2024.

GPU comparison

- K-NN: CAM array size is 128x128

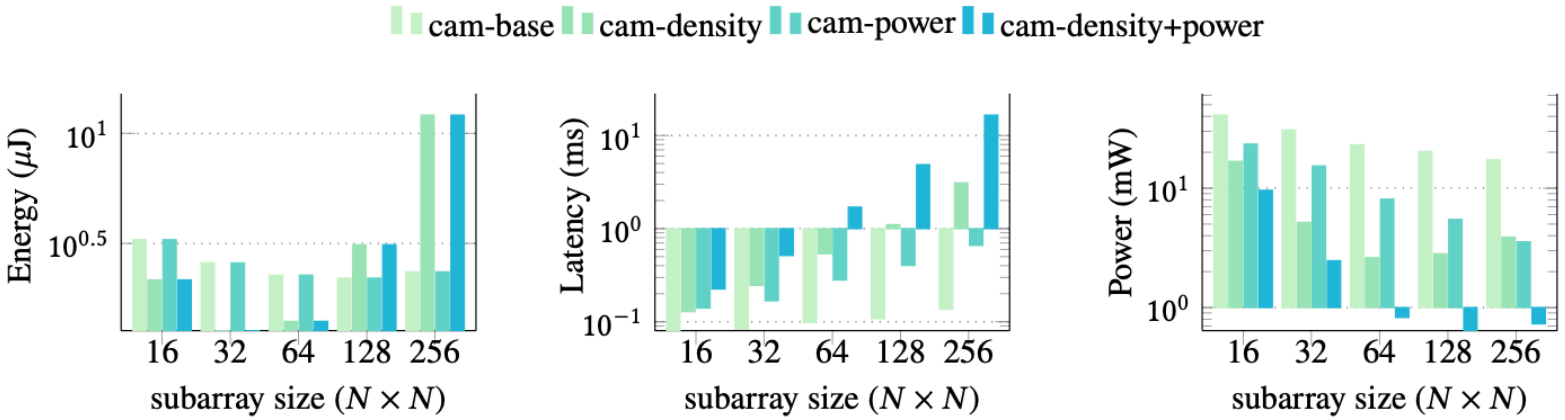


- 3 bit CAMs achieve GPU comparable accuracy
- Exhibits around 14x less latency
- 5 orders of magnitude less energy

Asif Ali Khan, ASPLOS 2024.

Design space exploration

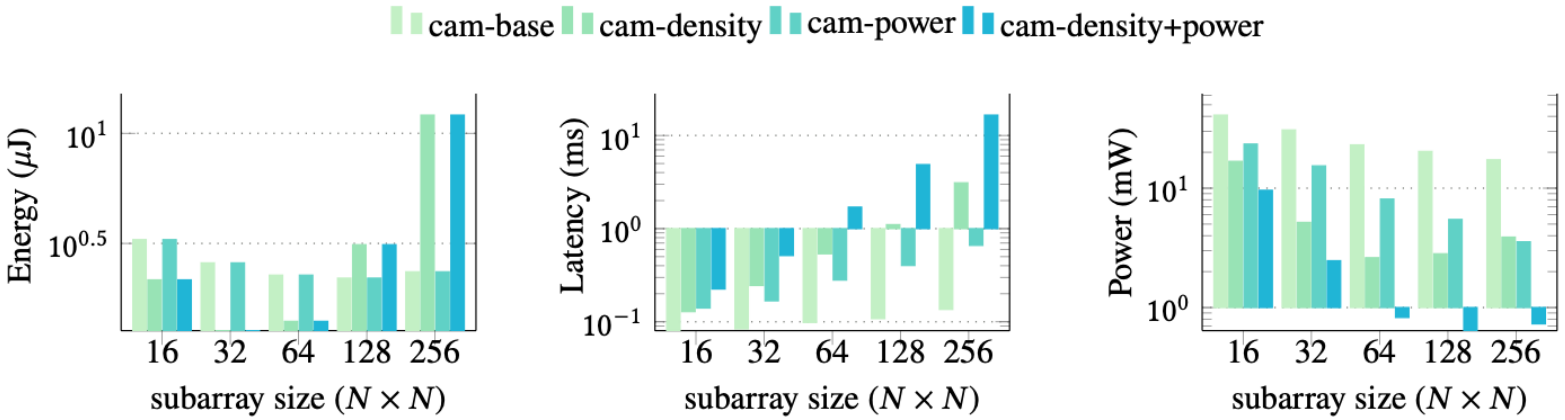
- *cam-base* (optimized for latency), *cam-power* (optimized for power), *cam-density* (optimized for density)



Asif Ali Khan, ASPLOS 2024.

Design space exploration

- *cam-base* (optimized for latency), *cam-power* (optimized for power), *cam-density* (optimized for density)

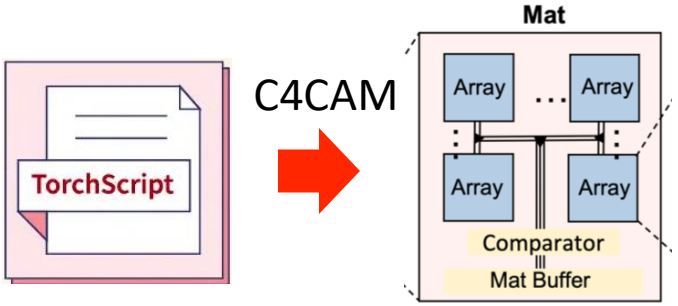


- *cam-power* can reduce the power considerably vs. *cam-base*
- The latency remains the same

Asif Ali Khan, ASPLOS 2024.

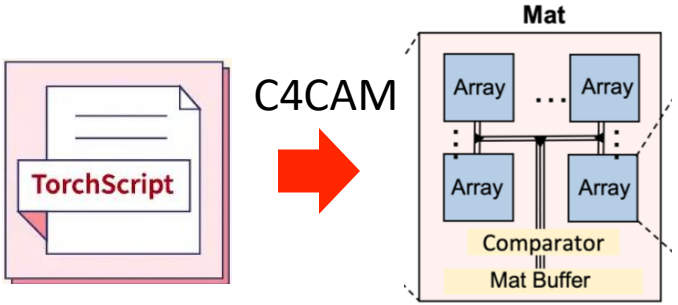
Conclusions

- CAMs, and other CIM designs, have demonstrated great potential



Conclusions

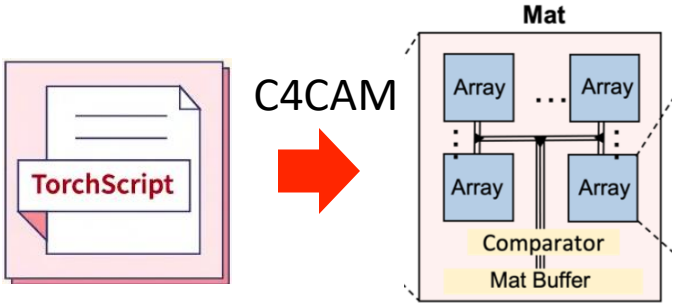
- CAMs, and other CIM designs, have demonstrated great potential



- C4CAM bridges the abstraction gap between **CAM APIs** and **high-level application descriptions**

Conclusions

- CAMs, and other CIM designs, have demonstrated great potential

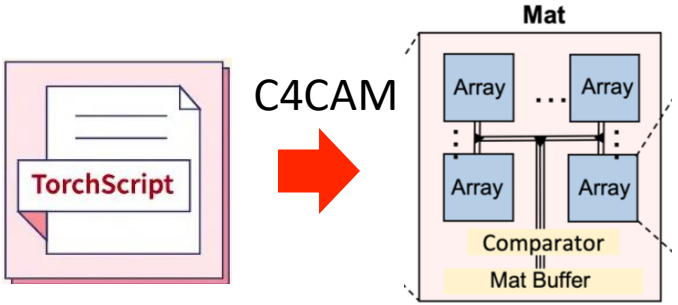


- C4CAM bridges the abstraction gap between **CAM APIs** and **high-level application descriptions**

- It enables rapid **design space exploration**

Conclusions

- CAMs, and other CIM designs, have demonstrated great potential



- C4CAM bridges the abstraction gap between **CAM APIs** and **high-level application descriptions**
- It enables rapid **design space exploration**
- **Future work:** Supporting heterogeneous CAM structures, integrating C4CAM in CINM, a generalized framework for CINM targets

Thank you

joao.lima@tu-dresden.de

cfaed.tu-dresden.de/ccc-about