CHAIR FOR COMPILER CONSTRUCTION

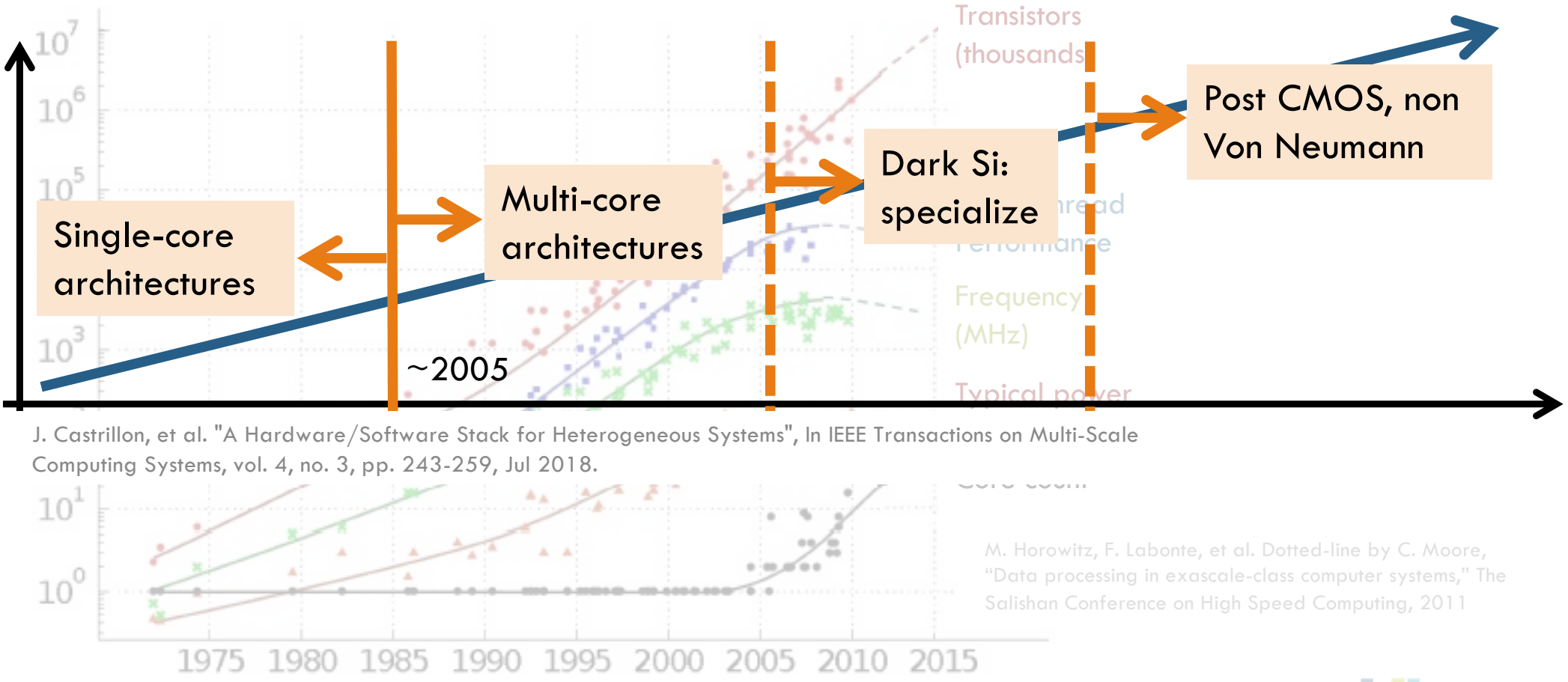
cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Next-generation compilers for emerging systems

Jeronimo Castrillon
Chair for Compiler Construction (CCC), TU Dresden,
SCADS.AI Dresden/Leipzig & Center for Advancing Electronics (cfaed) Dresden

Keynote: Workshop on Compilers, Deployment, and Tooling for Edge AI (CODAI'23)
Hamburg, Germany
September 21, 2023

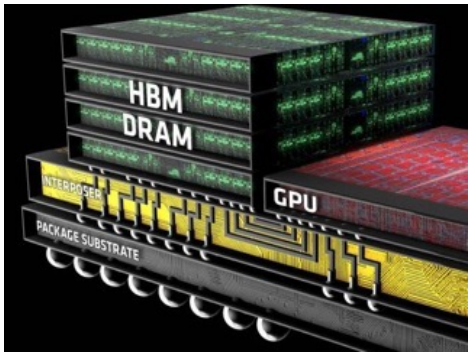
Evolution of computing: Breaking walls



J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 243-259, Jul 2018.

M. Horowitz, F. Labonte, et al. Dotted-line by C. Moore, "Data processing in exascale-class computer systems," The Salishan Conference on High Speed Computing, 2011

Emerging systems: Examples

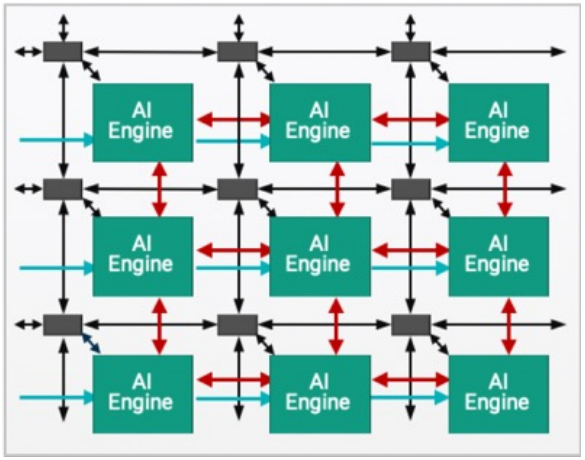


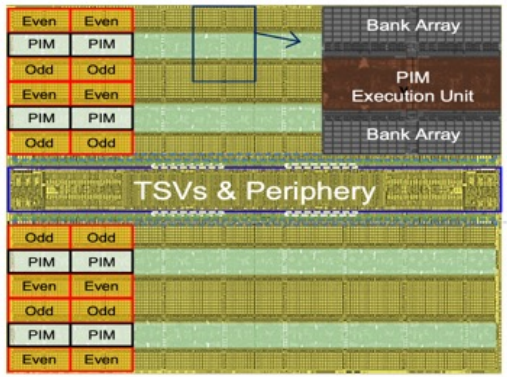
Source: AMD, AnandTech

High-bandwidth memory

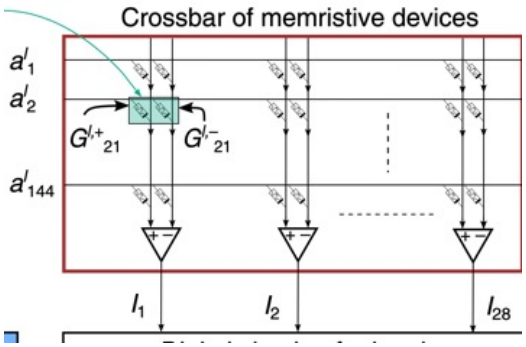
AI accelerators + Prog. logic

Source: AMD

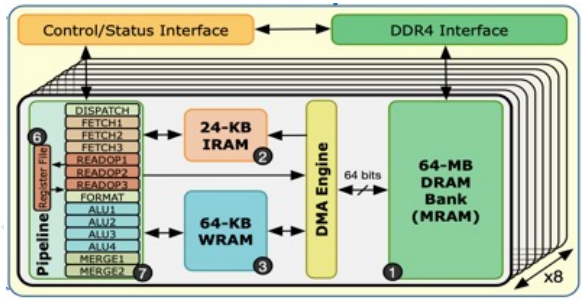




Source: Samsung

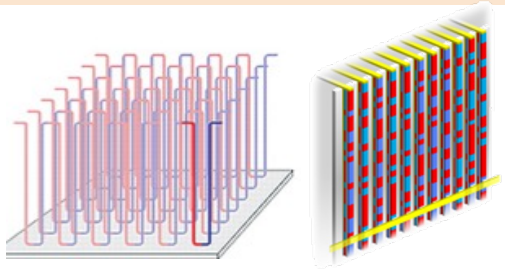


Source: IBM



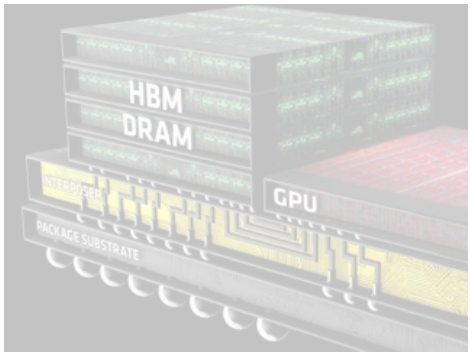
Source: UPMEM

Near-memory computing



Emerging memories + in-memory computing

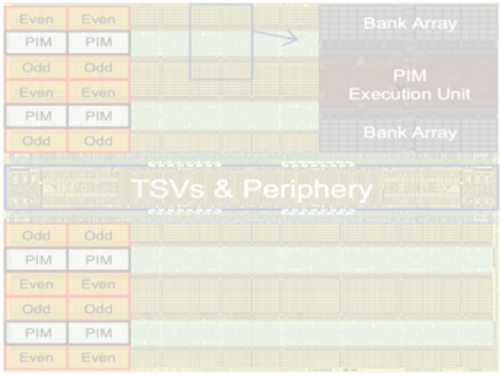
Emerging systems: Examples



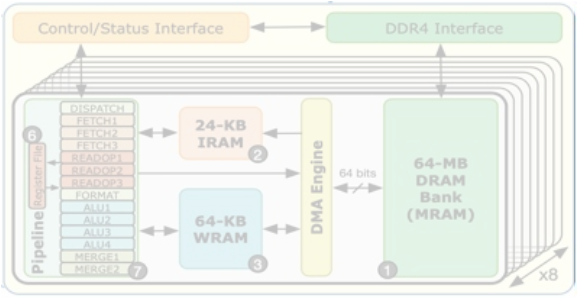
Source: AMD, AnandTech

High-bandwidth memory

AI accelerators + Prog. logic



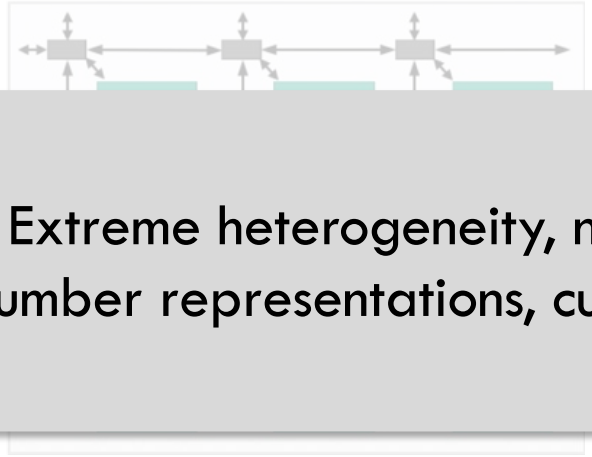
Source: Samsung



Source: UPMEM

Near-memory computing

Source: AMD



Extreme heterogeneity, non Von Neumann paradigms, custom number representations, custom data mapping, complex APIs, …

Abstractions and compilation

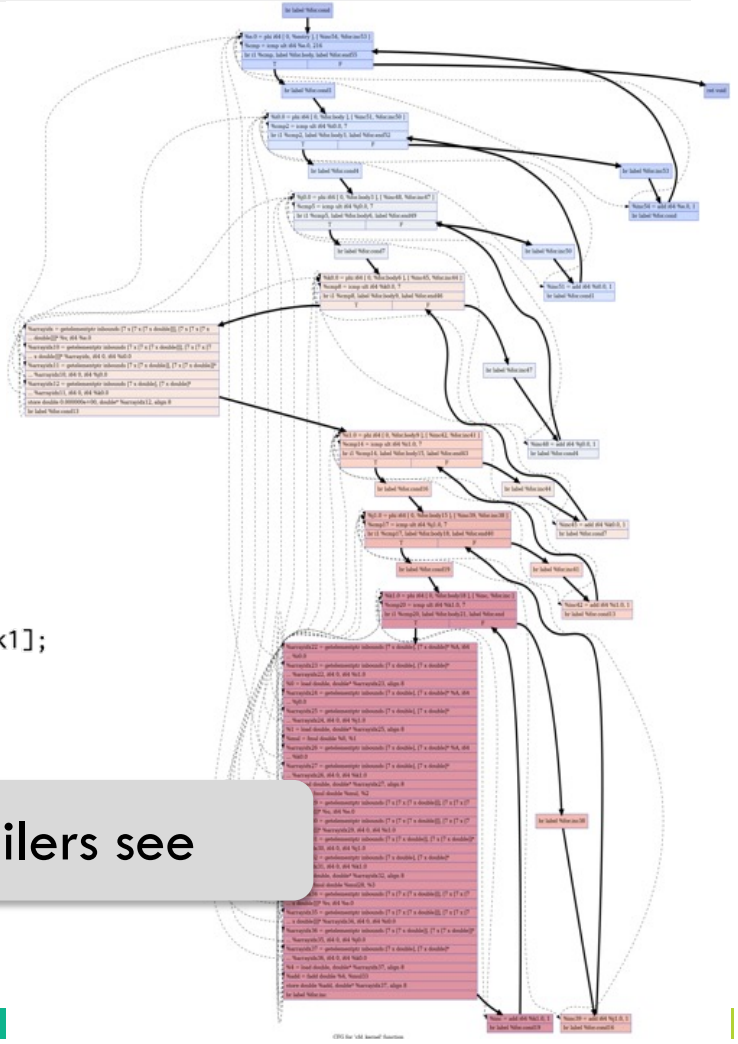
$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

What we want

What we (naively) code

```
1  void cfd_kernel(
2    double A[restrict 7][7],
3    double u[restrict 216][7][7][7],
4    double v[restrict 216][7][7][7])
5  {
6    /* element loop: */
7    for(int e = 0; e < 216; e++) {
8      for(int i0 = 0; i0 < 7; i0++) {
9      for(int j0 = 0; j0 < 7; j0++) {
10     for(int k0 = 0; k0 < 7; k0++) {
11       v[e][i0][j0][k0] = 0.0;
12       for(int i1 = 0; i1 < 7; i1++) {
         for(int j1 = 0; j1 < 7; j1++) {
         for(int k1 = 0; k1 < 7; k1++) {
           v[e][i0][j0][k0] += A[i0][i1]
                             * A[j0][j1]
17                           * A[k0][k1]
18                           * u[e][i1][j1][k1];
19     } } } } } }
20   } /* end of element loop */
21 }
```

What compilers see



Abstractions and compilation

$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

What we want

What we (naively) code

```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  for(int e = 0; e < 216; e++) {
    for(int i0 = 0; i0 < 7; i0++) {
    for(int j0 = 0; j0 < 7; j0++) {
    for(int k0 = 0; k0 < 7; k0++) {
      v[e][i0][j0][k0] = 0.0;
      for(int i1 = 0; i1 < 7; i1++) {
      for(int j1 = 0; j1 < 7; j1++) {
      for(int k1 = 0; k1 < 7; k1++) {
        v[e][i0][j0][k0] += A[i0][i1]
                          * A[j0][j1]
                          * A[k0][k1]
                          * u[e][i1][j1][k1];
    } } } } } }
  } /* end of element loop */
}
```

100X

What performance experts code

```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  #pragma omp for
  for (int e = 0; e < 216; e++) {
    double t6[7][7][7];
    /* 1st contraction: */
    #pragma simd
    for (int i0 = 0; i0 < 7; i0++) {
    for (int i1 = 0; i1 < 7; i1++) {
    /* #pragma simd */
    for (int i2 = 0; i2 < 7; i2++) {
      double t8 = 0.0;
      for (int i3 = 0; i3 < 7; i3++)
        t8 += A[i0][i3] * u[e][i1][i2][i3];
      t6[i0][i1][i2] = t8;
    } } } /* end of 1st contraction */
    double t7[7][7][7];
    /* 2nd contraction: */
    #pragma simd
    for (int i4 = 0; i4 < 7; i4++) {
    for (int i5 = 0; i5 < 7; i5++) {
    /* #pragma simd */
    for (int i6 = 0; i6 < 7; i6++) {
      double t9 = 0.0;
      for (int i7 = 0; i7 < 7; i7++)
        t9 += A[i4][i7] * t6[i5][i6][i7];
      t7[i4][i5][i6] = t9;
    } } } /* end of 2nd contraction */
    /* 3rd contraction: */
    #pragma simd
    for (int i8 = 0; i8 < 7; i8++) {
    for (int i9 = 0; i9 < 7; i9++) {
    /* #pragma simd */
    for (int i10 = 0; i10 < 7; i10++) {
      double t10 = 0.0;
      for (int i11 = 0; i11 < 7; i11++)
        t10 += A[i8][i11] * t7[i9][i10][i11];
      v[e][i8][i9][i10] = t10;
    } } } /* end of third contraction */
  } /* end of element loop */
```

Abstractions and compilation

$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

What we want

```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  for(int e = 0; e < 216; e++) {
    for(int i0 = 0; i0 < 7; i0++) {
    for(int j0 = 0; j0 < 7; j0++) {
    for(int k0 = 0; k0 < 7; k0++) {
      v[e][i0][j0][k0] = 0.0;
      for(int i1 = 0; i1 < 7; i1++) {
      for(int j1 = 0; j1 < 7; j1++) {
      for(int k1 = 0; k1 < 7; k1++) {
        v[e][i0][j0][k0] += A[i0][i1]
                          * A[j0][j1]
```

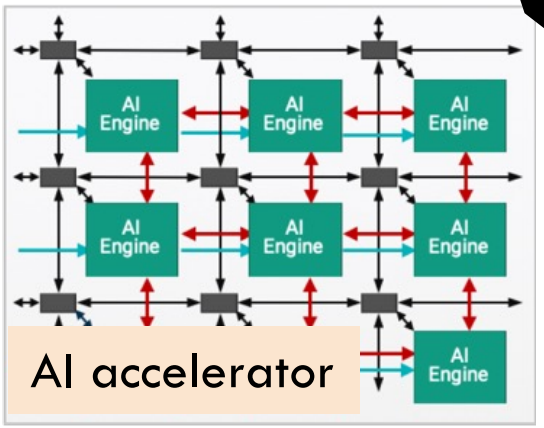
???

???

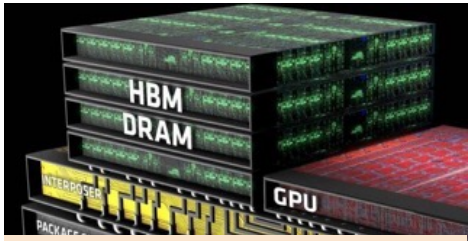
100X

???

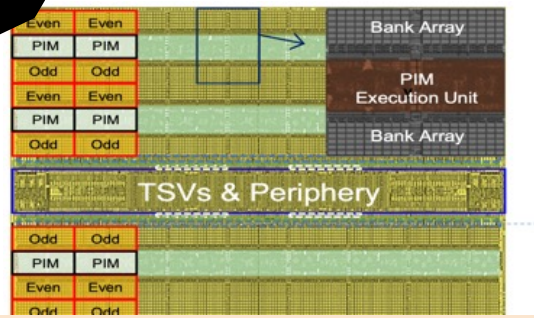
```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  #pragma omp for
  for (int e = 0; e < 216; e++) {
    double t6[7][7][7];
    /* 1st contraction: */
    #pragma simd
    for (int i0 = 0; i0 < 7; i0++) {
    for (int i1 = 0; i1 < 7; i1++) {
    /* #pragma simd */
    for (int i2 = 0; i2 < 7; i2++) {
      double t8 = 0.0;
      for (int i3 = 0; i3 < 7; i3++)
        t8 += A[i0][i3] * u[e][i1][i2][i3];
      t6[i0][i1][i2] = t8;
    } } } /* end of 1st contraction */
    double t7[7][7][7];
    /* 2nd contraction: */
    #pragma simd
    for (int i4 = 0; i4 < 7; i4++) {
    for (int i5 = 0; i5 < 7; i5++) {
    /* #pragma simd */
```



AI accelerator



HBM+FPGA



Near-memory computing

Abstractions and compilation

$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

What we want

```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  for(int e = 0; e < 216; e++) {
    for(int i0 = 0; i0 < 7; i0++) {
    for(int j0 = 0; j0 < 7; j0++) {
    for(int k0 = 0; k0 < 7; k0++) {
      v[e][i0][j0][k0] = 0.0;
      for(int i1 = 0; i1 < 7; i1++) {
      for(int j1 = 0; j1 < 7; j1++) {
      for(int k1 = 0; k1 < 7; k1++) {
        v[e][i0][j0][k0] += A[i0][i1]
```

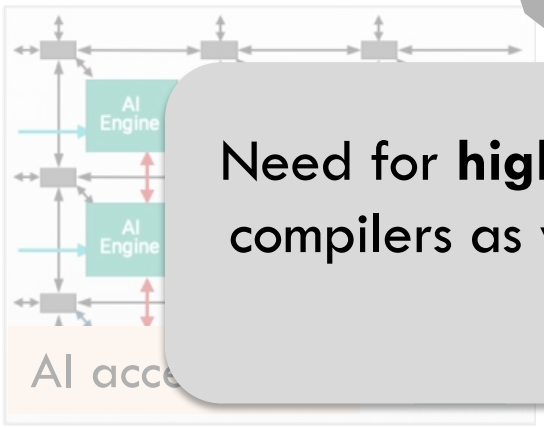
100X

???

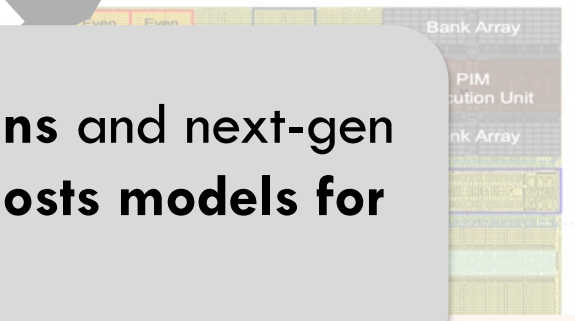
???

???

```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  #pragma omp for
  for (int e = 0; e < 216; e++) {
    double t6[7][7][7];
    /* 1st contraction: */
    #pragma simd
    for (int i0 = 0; i0 < 7; i0++) {
    for (int i1 = 0; i1 < 7; i1++) {
    /* #pragma simd */
    for (int i2 = 0; i2 < 7; i2++) {
      double t8 = 0.0;
      for (int i3 = 0; i3 < 7; i3++)
        t8 += A[i0][i3] * u[e][i1][i2][i3];
      t6[i0][i1][i2] = t8;
    } } } /* end of 1st contraction */
    double t7[7][7][7];
    /* 2nd contraction: */
    #pragma simd
    for (int i4 = 0; i4 < 7; i4++) {
    for (int i5 = 0; i5 < 7; i5++) {
    /* #pragma simd */
```



AI acce



HBM+FPGA

Near-memory computing

Need for **higher-level programming abstractions** and next-gen compilers as well as novel **computational and costs models for emerging accelerators**

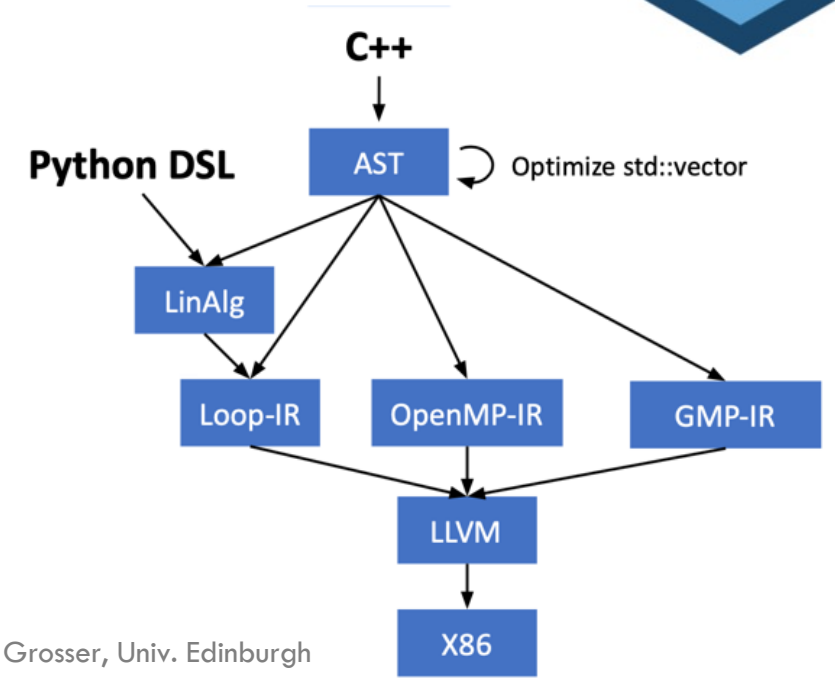
The power of abstractions

MLIR: Vehicle to capture abstractions

- Started by Google ~2018, now in public domain

Lattner, Chris, et al. "Mlir: Scaling compiler infrastructure for domain specific computation." 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2021.

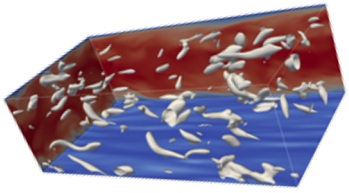
- Not an IR, but an extensible framework
  - to describe intermediate abstractions (called **dialects**),
  - to optimize representations between dialects (**transform, lower** or **raise**),
  - that builds on the success of LLVM to build community/infrastructure and reuse



Source: T. Grosser, Univ. Edinburgh

Example: Tensor expressions (Physics, ML)

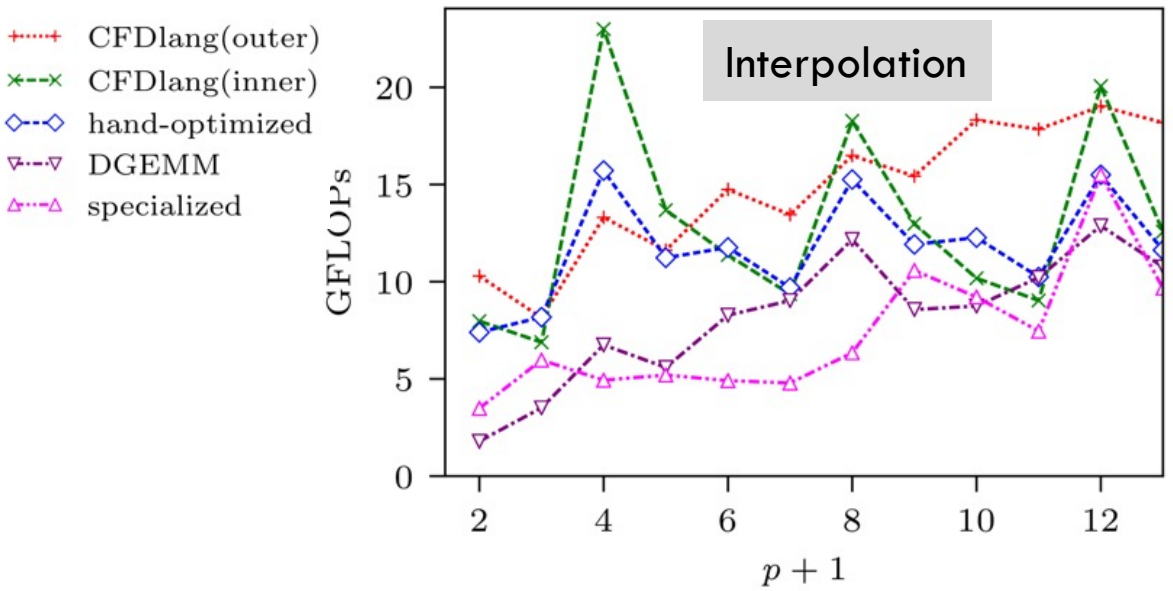
- CFDlang



$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

```
source =  ...
var input A   : matrix           &
var input u   : tensorIN         &
var input output v  : tensorOUT &
var input alpha : []             &
var input beta  : []             &
v = alpha * (A # A # A # u .
    [[5 8] [3 7] [1 6]]) + beta * v
```

```
auto A = Matrix(m, n), B = Matrix(m, n),
     C = Matrix(m, n);
auto u = Tensor<3>(n, n, n);
auto v = (A*B*C)(u);
```



N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL'18.
N.A. Rink, N. A. and J. Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ARRAY'19, pp. 57-68

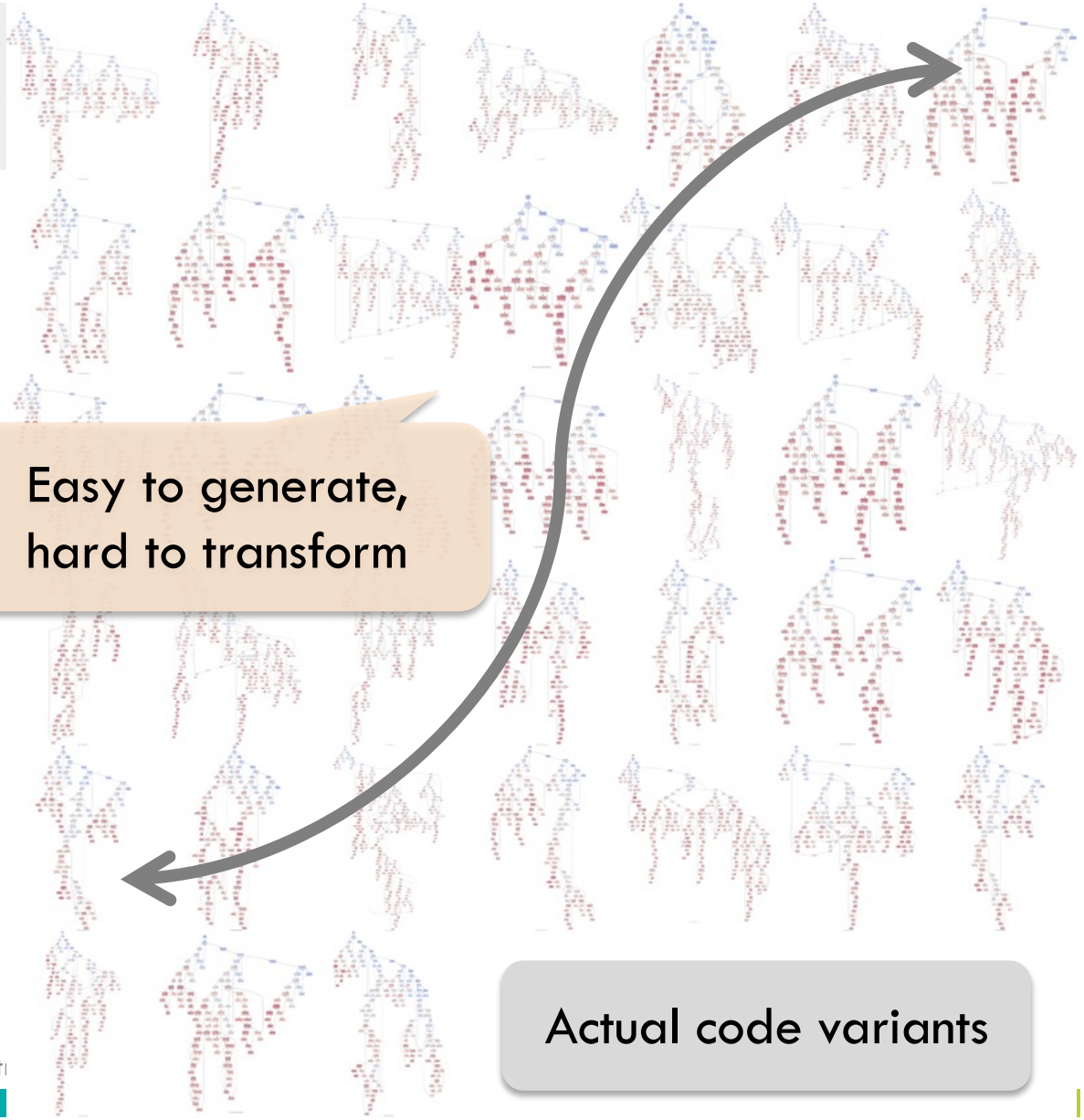
Closing the performance gap

- Not really optimization magic
  - Leverage expert knowledge
  - Algebraic identities

$$v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot (A_{jm} \cdot (A_{il} \cdot u_{lmn}))$$

$$v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot A_{jm}) \cdot (A_{il} \cdot u_{lmn})$$

$$v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))$$

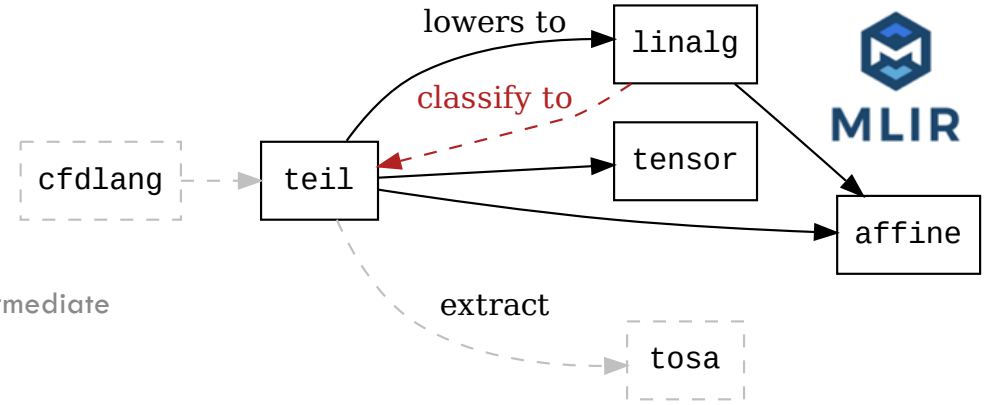


N. A. Rink, et al. “CFDlang: High-level code generation for high-order methods in fluid dynamics”. RWDSL’18.
A. Susungi, et al., "Meta-programming for Cross-Domain Tensor Optimizations", GPCE’18 pp. 79-92.

Tensor intermediate language (TeIL) in MLIR

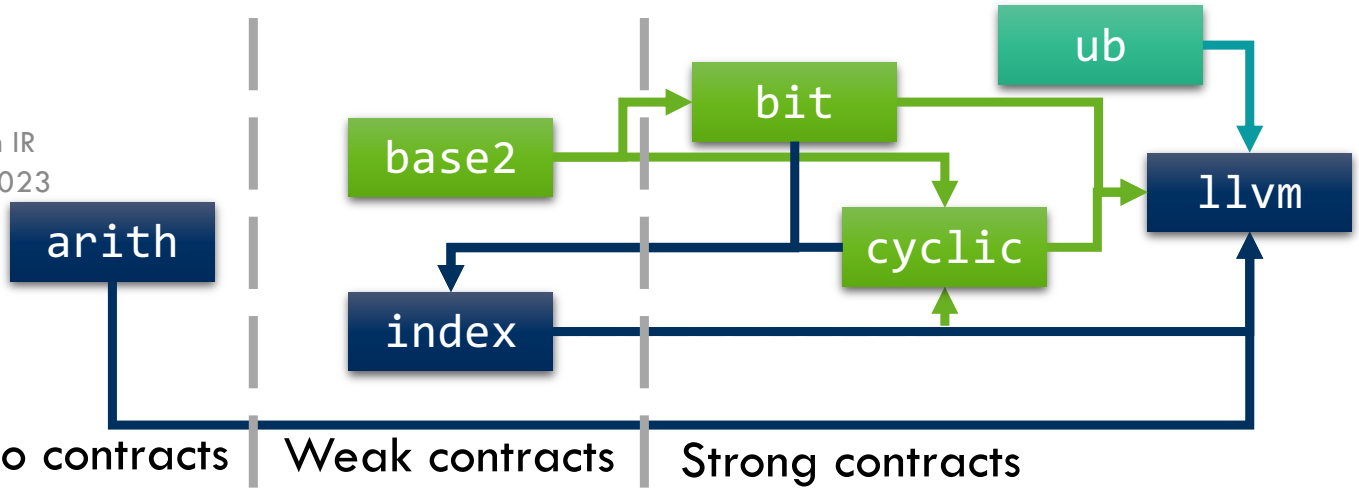
- Primitive ops instead of index maps
  - Easier to express identities (big-O trfs)
  - Uses symbolic math, infinite precision

N.A. Rink, N. A. and J. Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ARRAY'19, pp. 57-68



- Specialization path to custom hardware

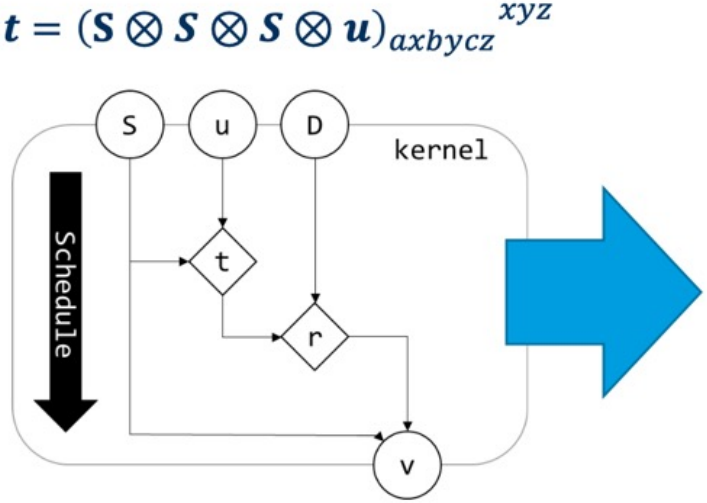
K. F. A. Friebel, J. Bi, J. Castrillon, "BASE2: An IR for Binary Numeral Types" In ACM HEART 2023



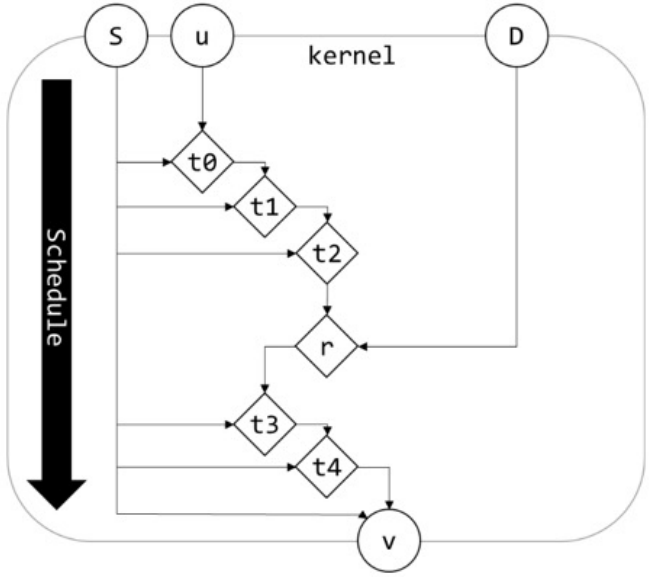
Domain-specific optimization

- Encode algebraic transformations
- Direct feedback to expert via DSL export

$$t = (S \otimes S \otimes S \otimes u)_{axbycz}{}^{xyz}$$



$$t = \left(S \otimes \left(S \otimes \left(S \otimes u\right)_{cz}{}^{xyz}\right)_{by}{}^{cxy}\right)_{ax}{}^{bcx}$$



FPGA code generation: Bus-attached FPGAs

- H2020 EU Project: Convergence HPC, Big Data and ML

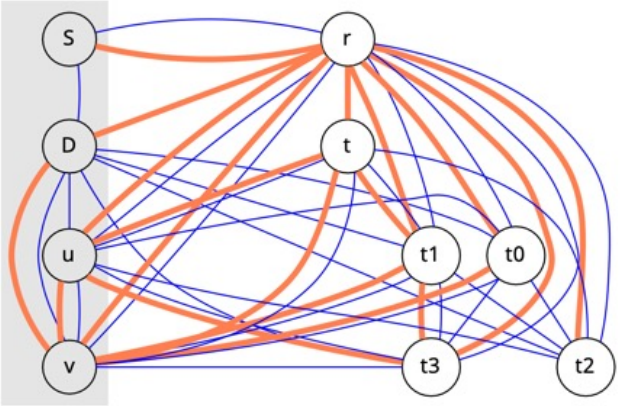
  C. Pilato, et al. "EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms", DATE 2021
- Inverse Helmholtz Kernel

https://everest-h2020.eu

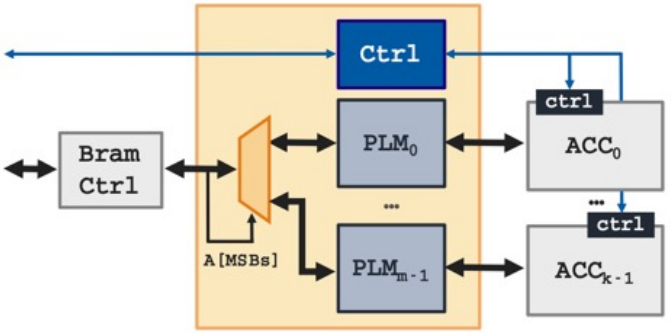
$$\mathbf{v}_e = (\mathbf{S} \otimes \mathbf{S} \otimes \mathbf{S})\mathbf{D}_e^{-1}(\mathbf{S}^T \otimes \mathbf{S}^T \otimes \mathbf{S}^T)\,\mathbf{u}_e$$

```
t = S # S # S # u . [[1
r = D * t
v = S # S # S # r . [[0
```

Lifetime analysis
(polyhedral analysis)



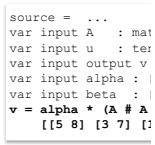
Menosyne
mem-subsystem gen (buffer sharing)

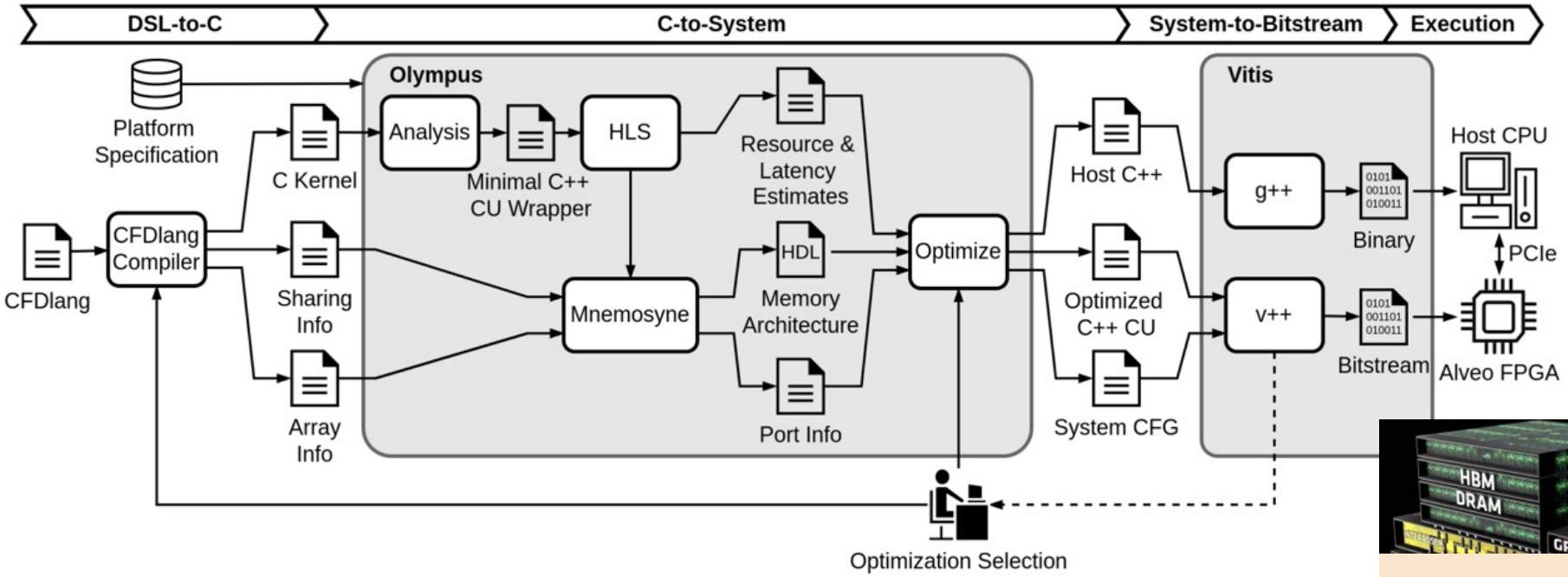


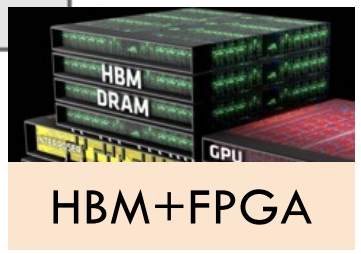
K. F. A. Friebel, S. Soldavini, G. Hempel, C. Pilato, J. Castrillon, "From Domain-Specific Languages to Memory-Optimized Accelerators for Fluid Dynamics", Proceedings of the FPGA for HPC Workshop, held in conjunction with IEEE Cluster 2021, Sep 2021

Putting it all together

- Complex compilation/design flow from DSL to system-level architecture





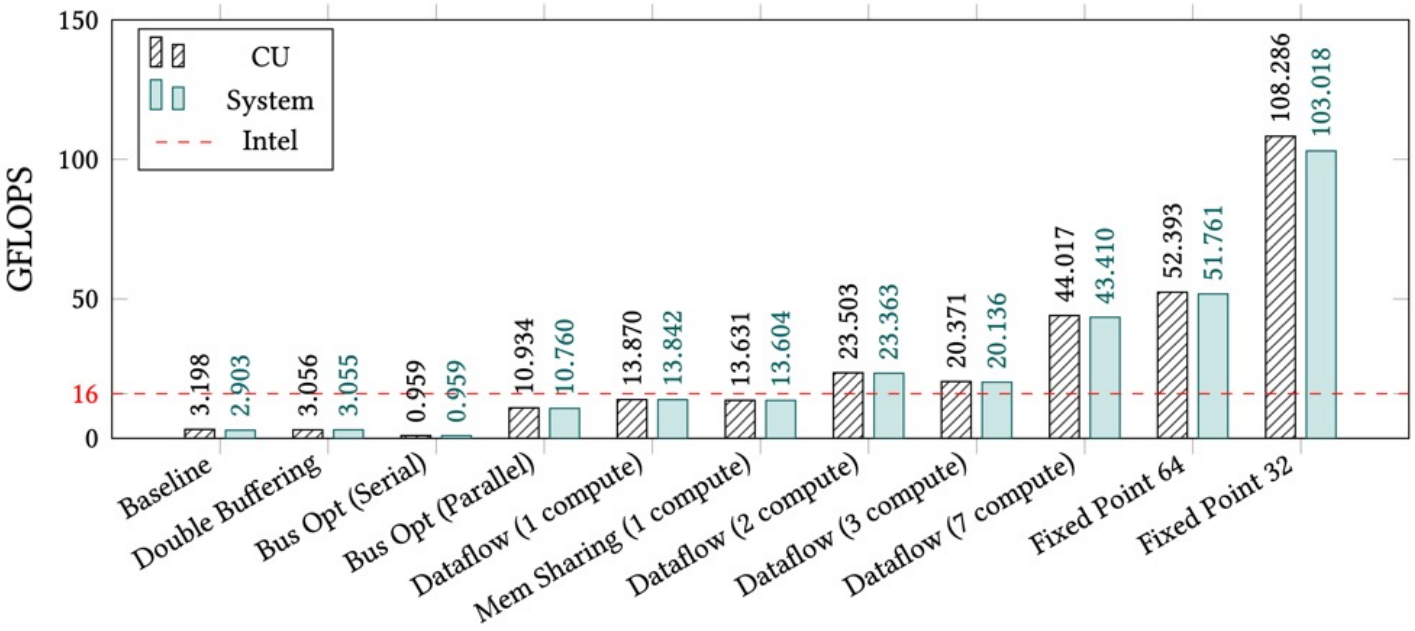


HBM+FPGA

S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, and C. Pilato. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics". In: ACM TRETS, Sept. 2022.

FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- Transformations for a **17x speedup** (same precision)



EVEREST

https://everest-h2020.eu

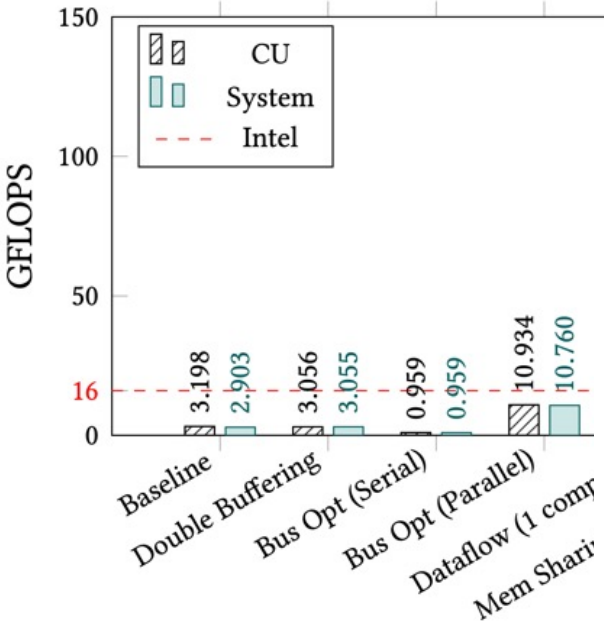
HBM+FPGA

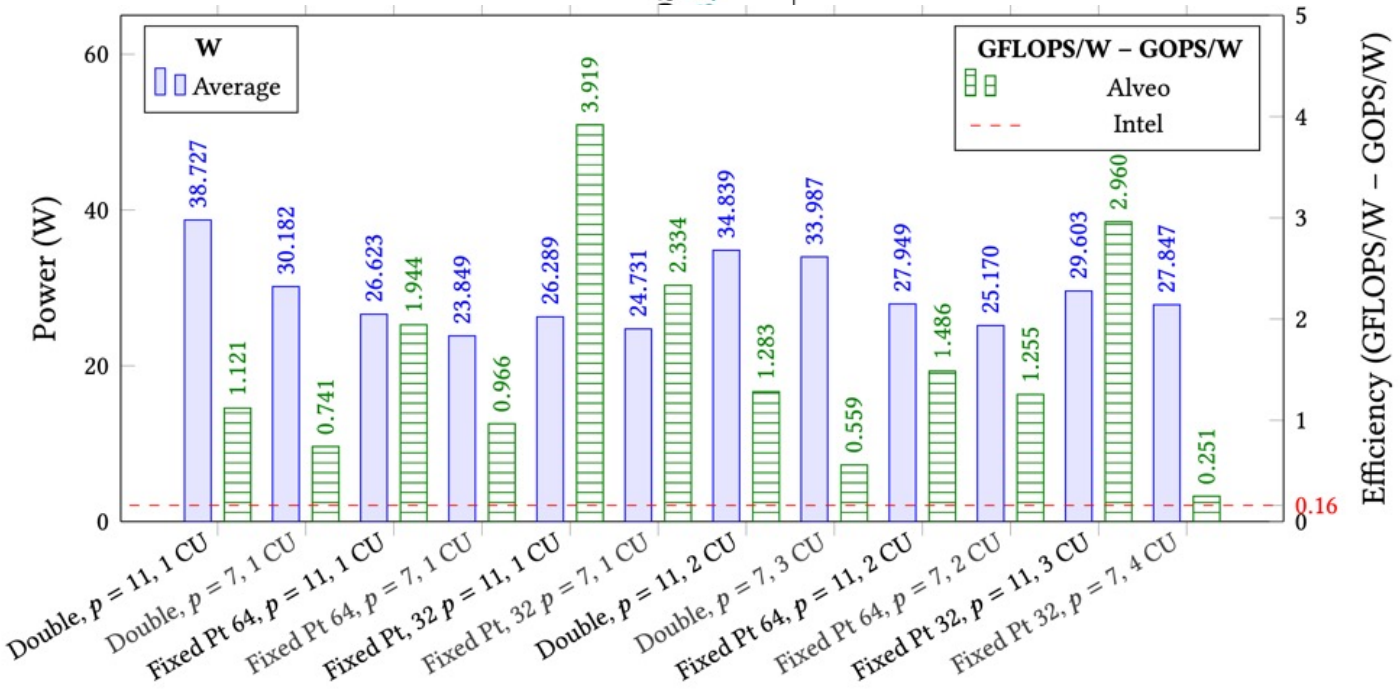
S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, and C. Pilato. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics". In: ACM TRETS, Sept. 2022.

FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- Variants with up to **24x better energy efficiency**

https://everest-h2020.eu

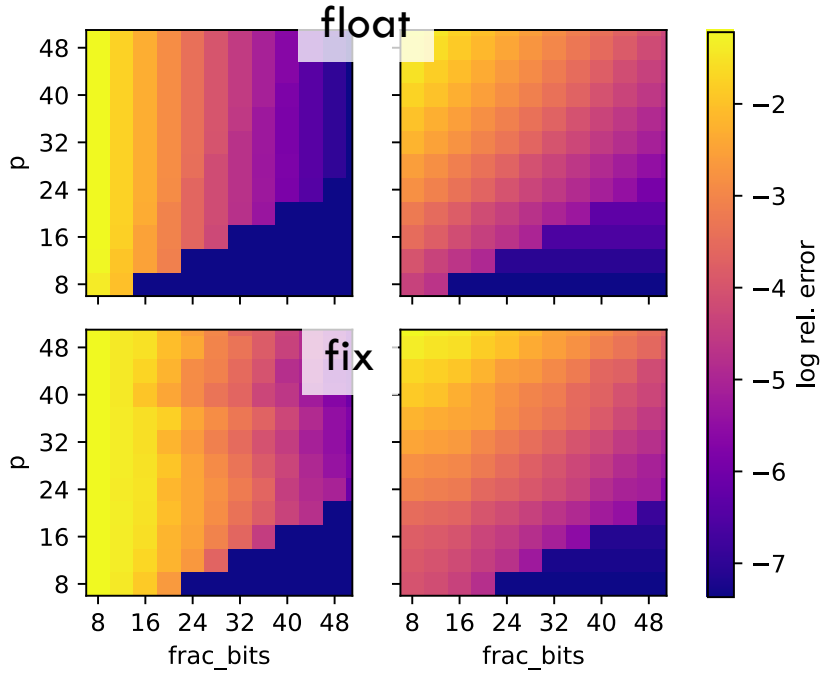




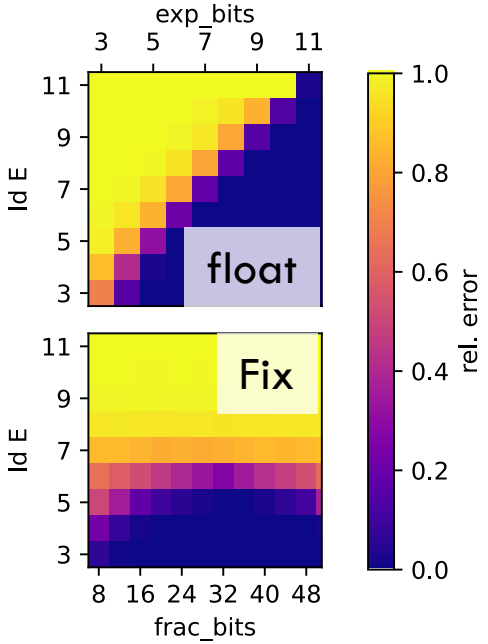
S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Her
Architectures from Domain-Specific Languages:

Base2: Custom precision analysis

- Interpolation $v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$



Significand precision p
(exp_bits = 6)



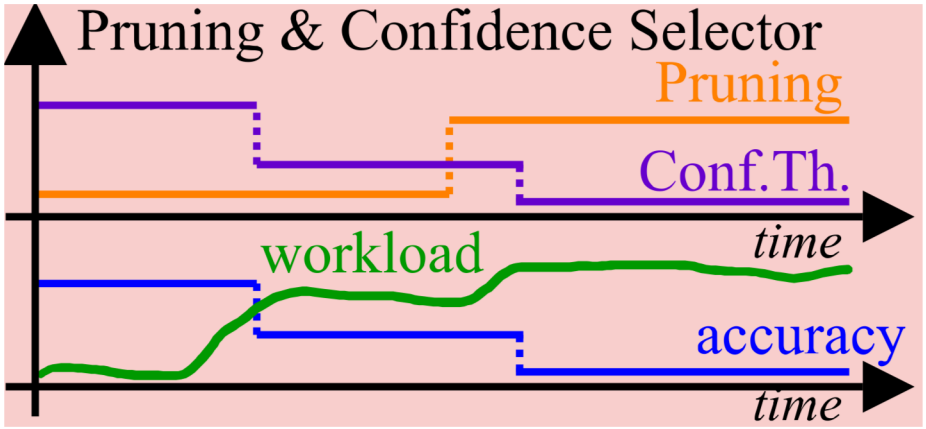
Exponent range ld E
(frac_bits = 32)

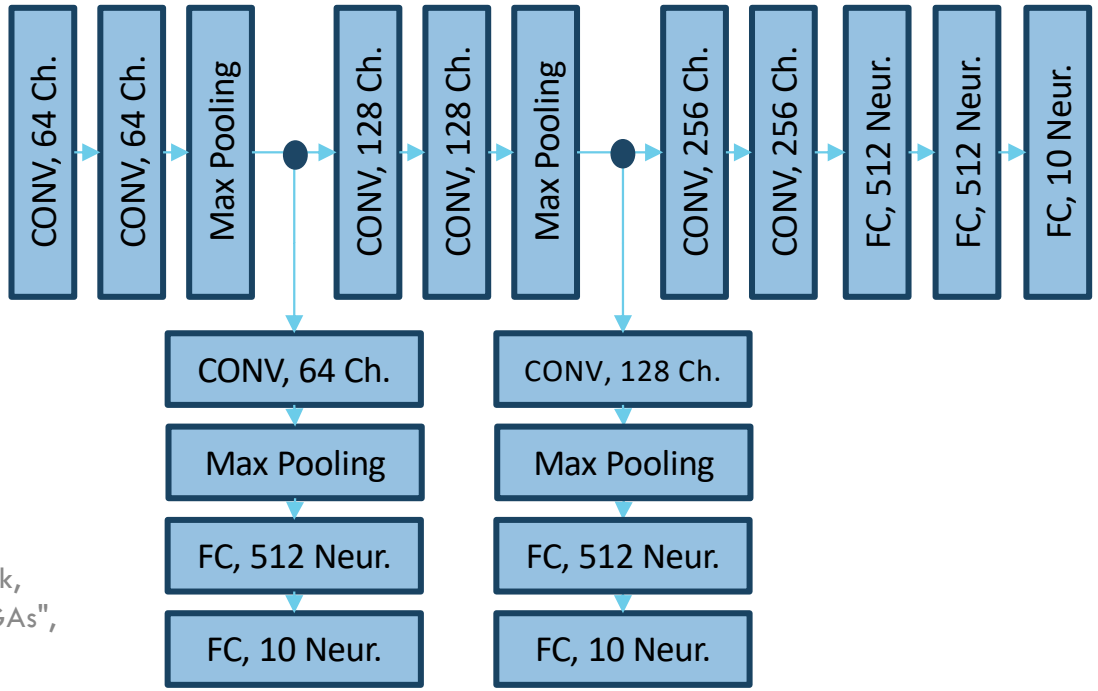
K. F. A. Friebel, J. Bi, J. Castrillon, "BASE2: An IR for Binary Numeral Types", In ACM HEART 2023

Inference: Reconfigurable HW & emerging memories

Adaptable inference

- High-level operator graphs + quantized data types (Brevitas, FINN)
- Trade-off: Pruning, early exit and confidence threshold

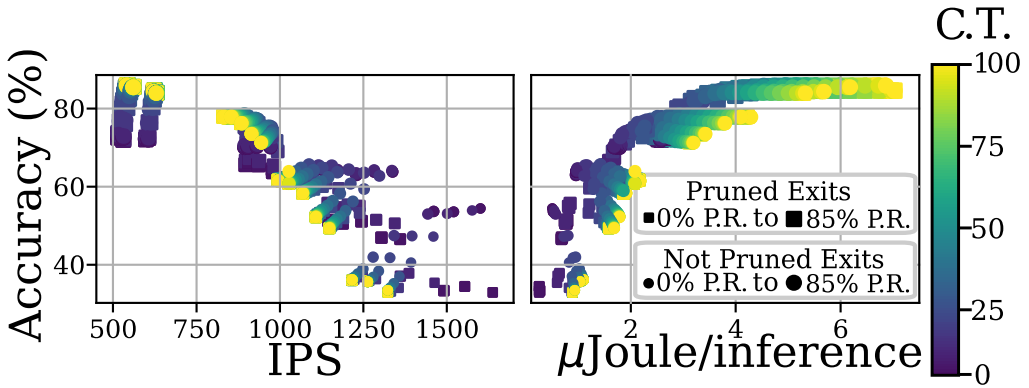




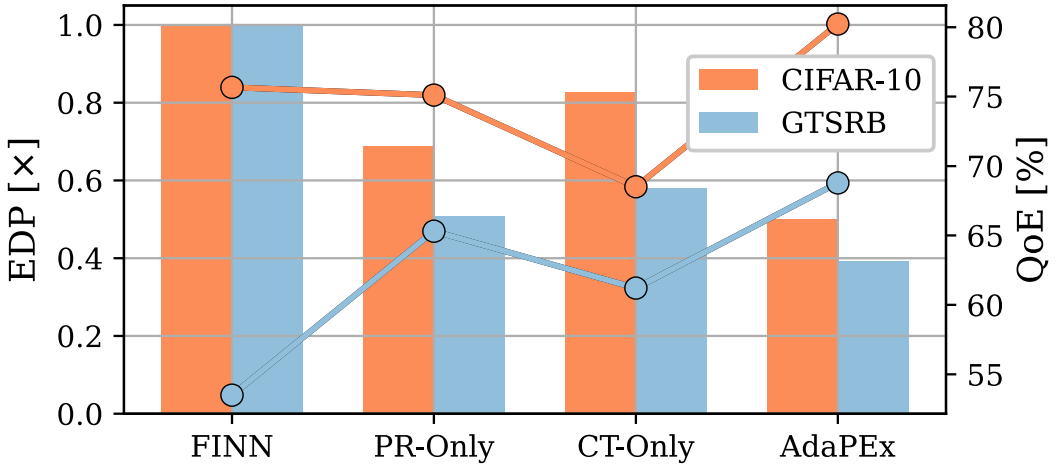
G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, A. C. Schneider Beck, "Pruning and Early-Exit Co-Optimization for CNN Acceleration on FPGAs", DATE 2023

Adaptable inference: Results

- Trade-off space @ design time for CNVW2A2 on CIFAR10



- Combined effect of jointly adapting pruning and early exits

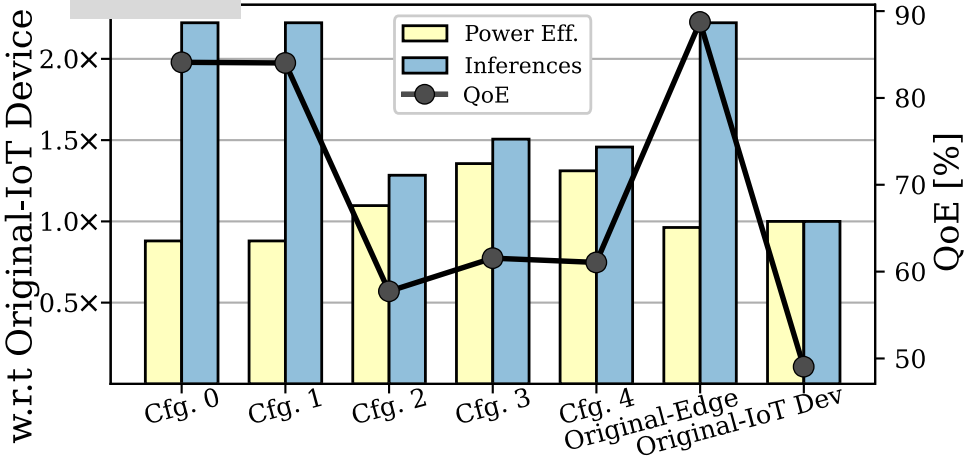


G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, A. C. Schneider Beck, "Pruning and Early-Exit Co-Optimization for CNN Acceleration on FPGAs", DATE 2023

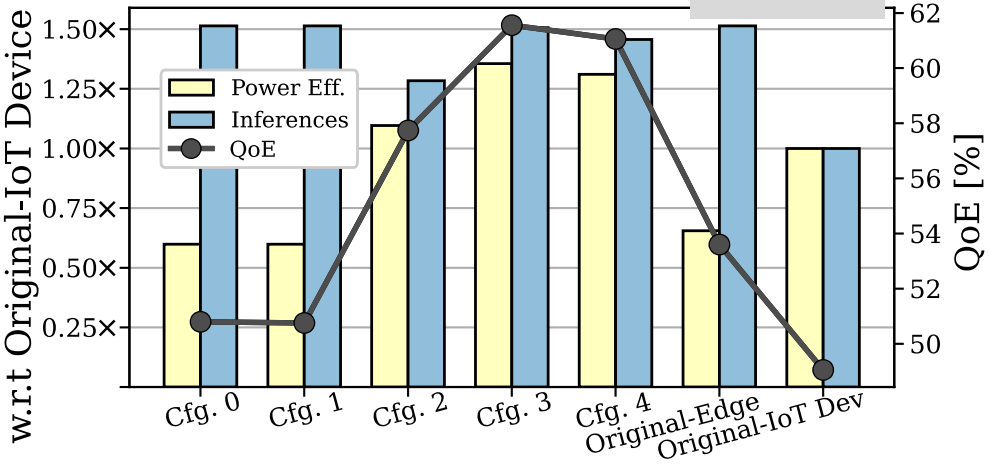
Offloading to the edge

- Balance local and edge compute: Accuracy, throughput, energy
  - Edge: Higher confidence, more parallelism, communication overhead
  - Device: Local compute, resource constrained
- Metrics vary with scenario: Video decoding **stable** (walking) or **instable** (tram)

Stable



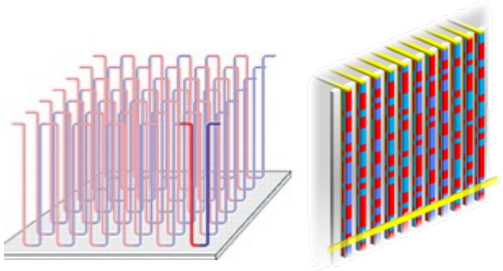
Instable

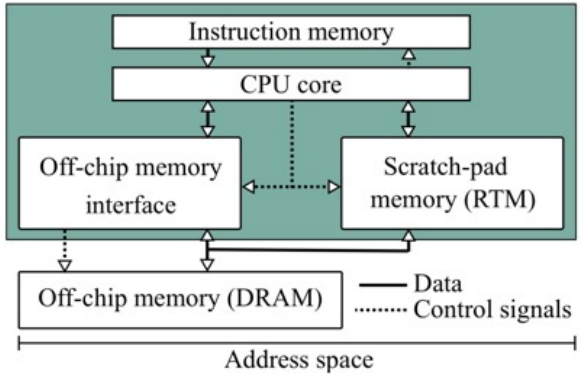


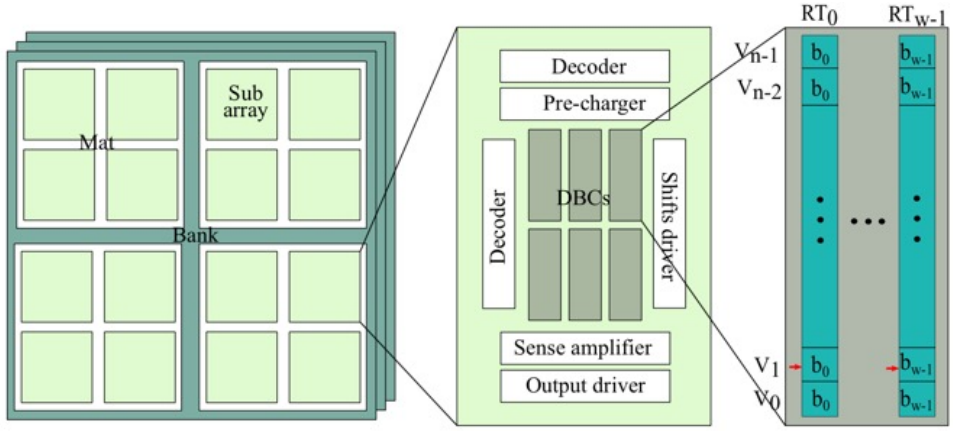
G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, A. C. Schneider Beck, ""Design Space Exploration for CNN Offloading to FPGAs at the Edge", ISVLSI 2023

Higher abstractions and data layout

- The case of racetrack memories (RTMs)
  - Density of DRAM & size/latency of SRAM!
  - Memory cell stores up to 100 bit sequentially (in tracks)
  - Latency highly depends on allocation and address traces





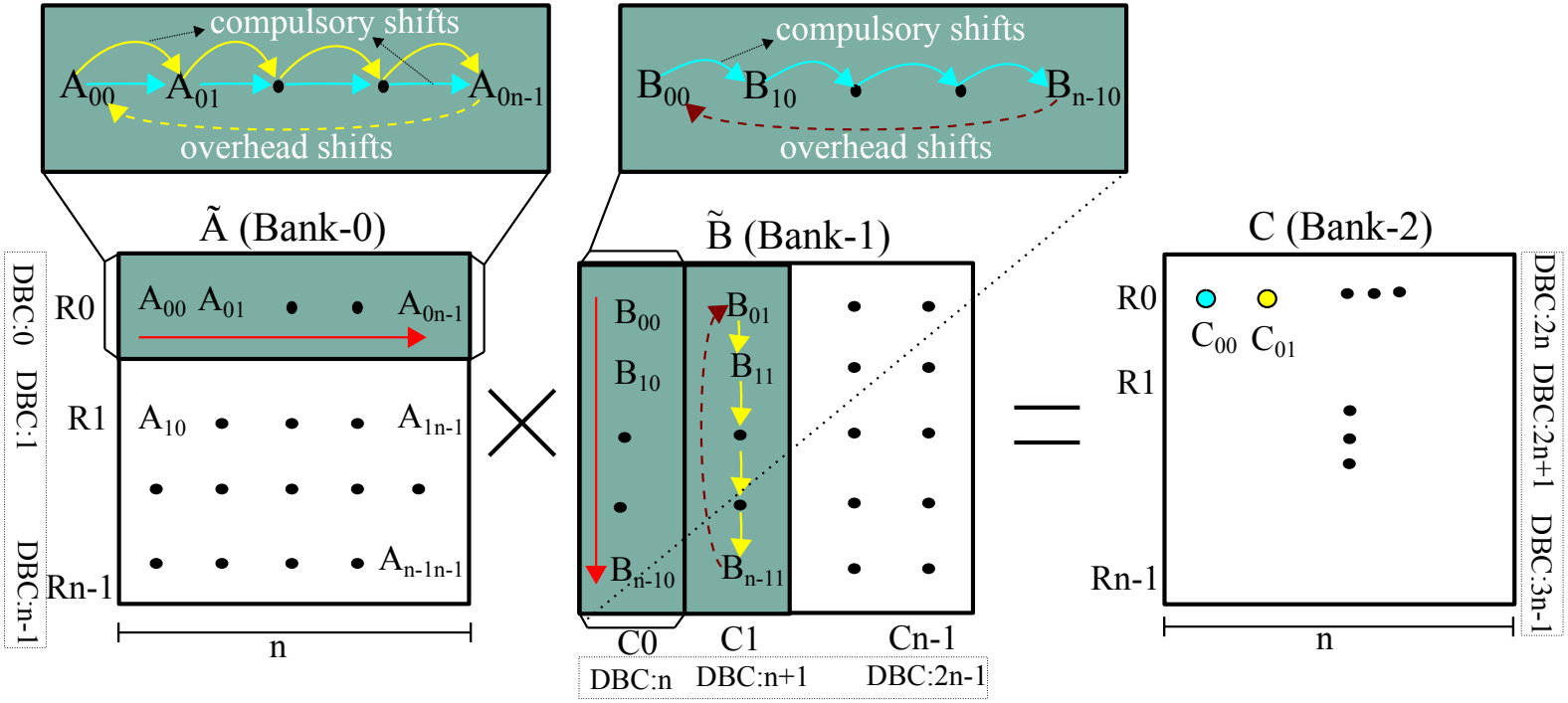


Khan, et al. "RTSim: A Cycle-accurate Simulator for Racetrack Memories", In IEEE Computer Architecture Letters, 2019

R. Bläsing, et al. "Magnetic Racetrack Memory: From Physics to the Cusp of Applications within a Decade", In Proceedings of the IEEE, vol. 108, no. 8, pp. 1303-1321, Mar 2020.

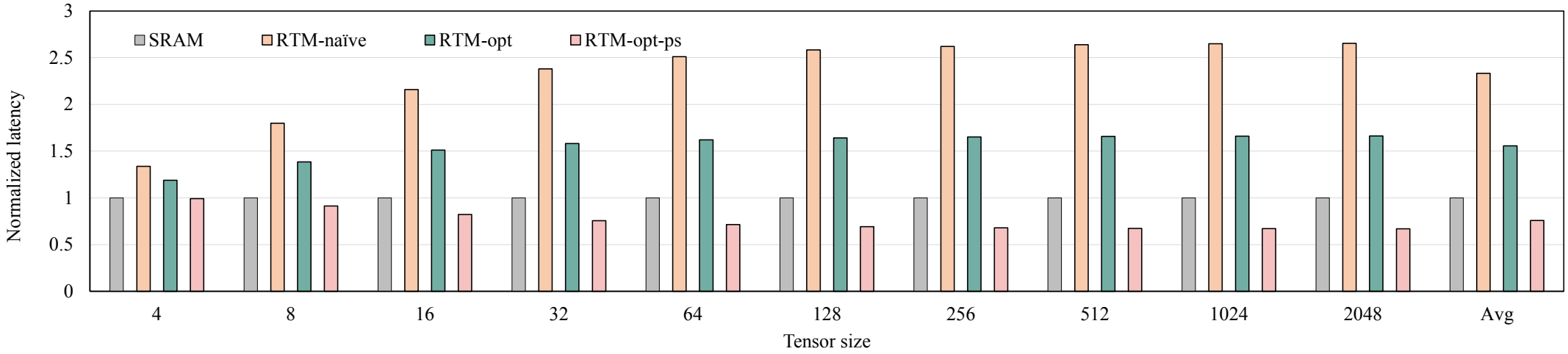
Tensor contractions on RTMs

- Consecutive accesses can be pre-shifted
- Zig-zagging: Avoid "rewinding the tape"



Latency comparison vs SRAM

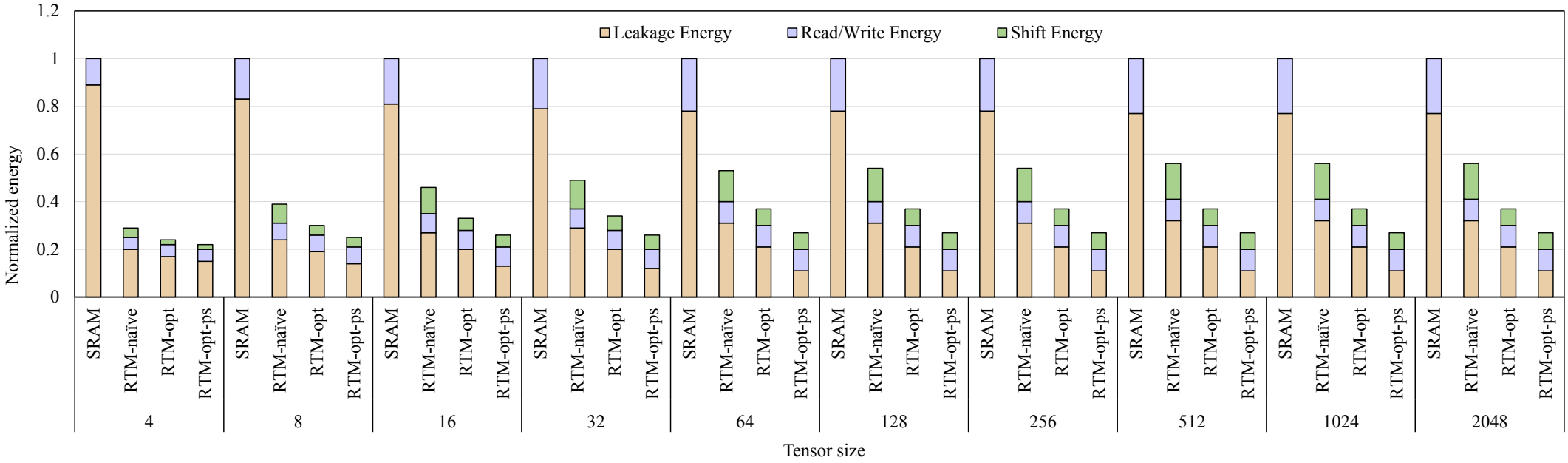
- Un-optimized and naïve mapping: Even worse latency than SRAM
- **24% faster** (even with very conservative circuit simulation)



A. A. Khan, et al, "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", LCTES'19, pp. 5-18, 2019
A. A. Khan, et al. "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories". ACM TECS 2020

Energy comparison vs SRAM

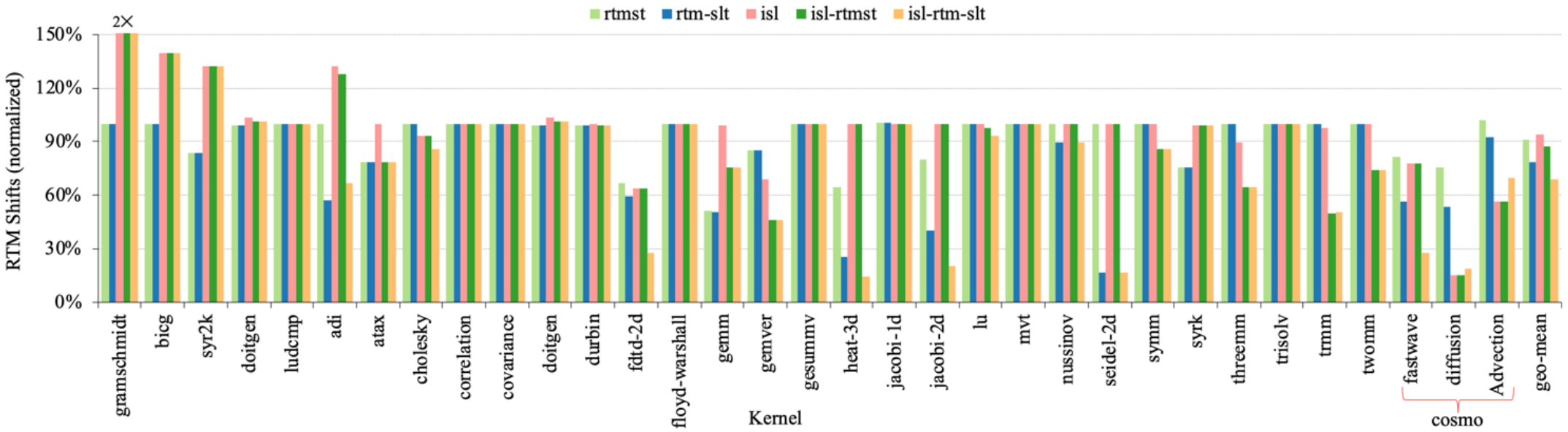
- Higher savings due to less leakage power
- **74% less energy** (in addition to savings due to DRAM placement)



A. A. Khan, et al, "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", LCTES'19, pp. 5-18, 2019
A. A. Khan, et al. "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories". ACM TECS 2020

Generalization: Optimizations for RTM

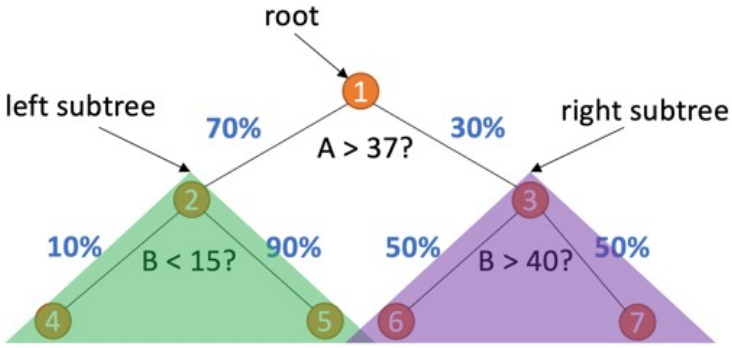
- Average improvements in performance (~20%) and energy consumption (~40%)

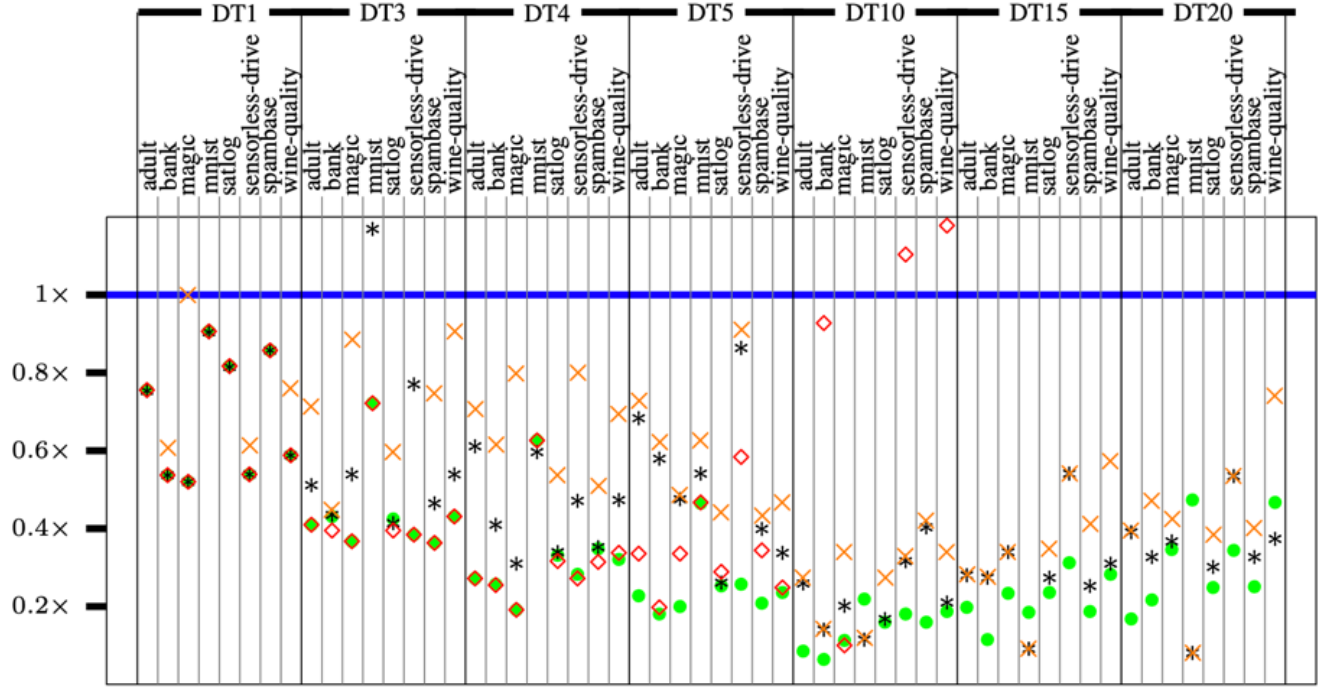


Khan, et al. "Polyhedral Compilation for Racetrack Memories". IEEE TCAD 2020

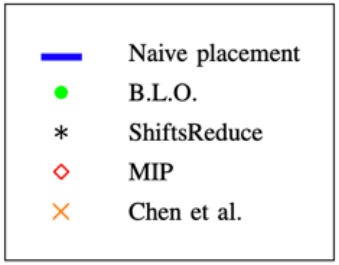
Random forests: Irregular access patterns

- RFs Popular for decision making @ the edge
- Use training statistics for tree placement





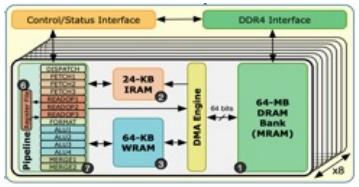
C. Hakert, "BLOwing Trees to the Ground: Layout Optimization of Decision Trees on Racetrack Memory", DAC 2021

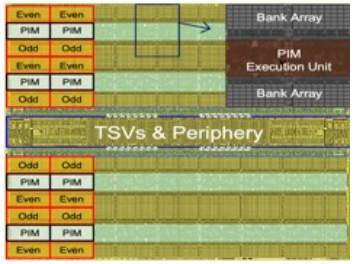


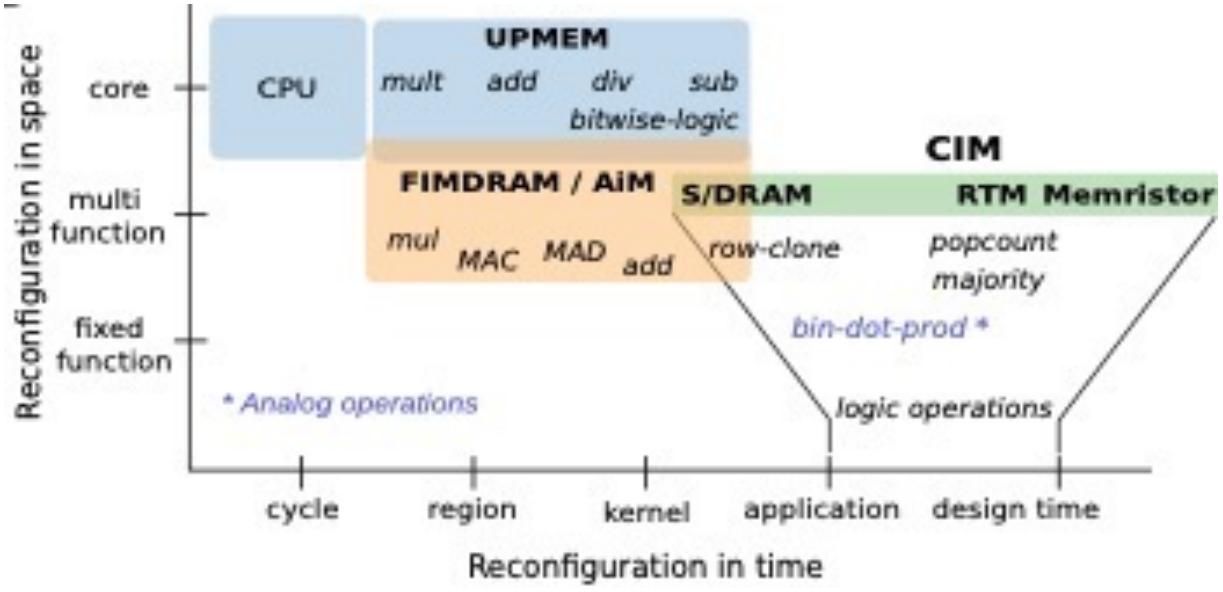
Near and in-memory computing

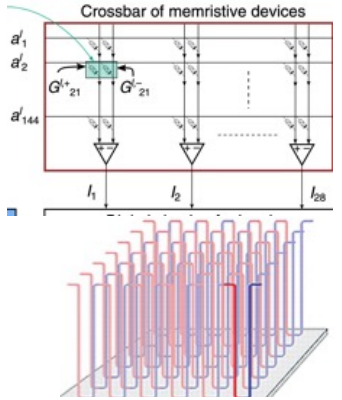
Rich landscape of designs

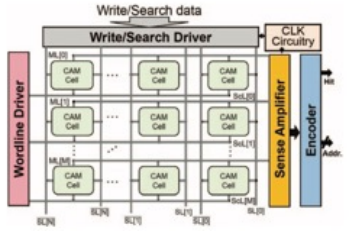
- Near-memory: Processors, logic close to memory
- In-memory (aka processing using memory): Leverage device properties







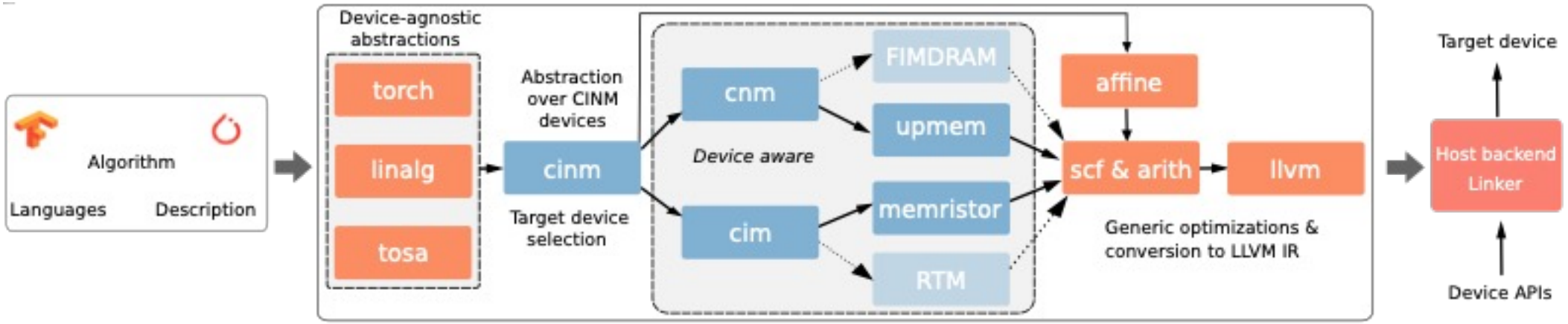




Samsung, Lee, Sukhan, et al. ISCA 2021
UPMEM by Gómez-Luna, Juan, et al. arXiv:2105.03814 (2021)
In-PCM Computing: Joshi, Vinay, et al. Nature Communications 11.1 (2020): 1-13.
CAM accelerators: Hu, Sharon, et al. 2021 IEDM

CINM: Generalized MLIR infrastructure

- From linear algebra abstractions (common to ML frameworks and beyond)
- Intermediate languages for **in and near memory computing**
- **Pattern recognition,** target-specific **models and optimizations**

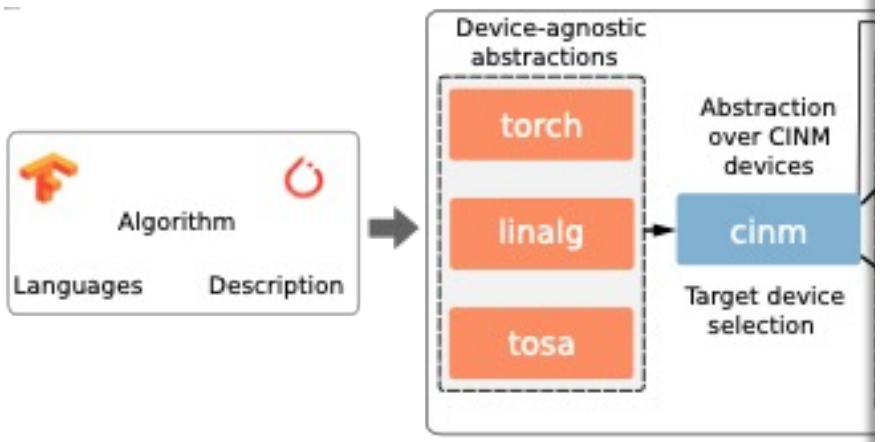


A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", arXiv, Aug 2023

DISRUPTIVE MEMORY TECHNOLOGIES
DFG PRIORITY PROGRAM 2377

CINM: Generalized MLIR infrastructure

- From linear algebra abstractions (common to ML frameworks and beyond)
- Intermediate languages for **in and near memory computing**
- **Pattern recognition,** target-specific **models and optimizations**

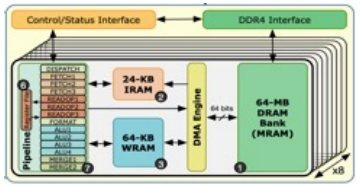


cinm.op.add/sub cinm.op.min/max	tensor<?xT> —"—	*same as lhs* —"—
cinm.op.and/or/xor cinm.op.popcount cinm.op.majority	—"— —"— —"—	—"— —"— —"—
cinm.op.sum cinm.op.exclusive_scan	—"— —"—	— —
cinm.op.transpose cinm.op.gemm cinm.op.gemv cinm.op.histogram	tensor<?x?xT> —"— —"— —"—	— tensor<?x?xT> tensor<?xT> —"—
cinm.op.similarity<enum> cinm.op.topk	tensor<?x?x?> tensor<?x?>	tensor<?x?x?> index

A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructu
Heterogeneous Compute In-Memory and Compute Near-M
Paradigms", arXiv, Aug 2023

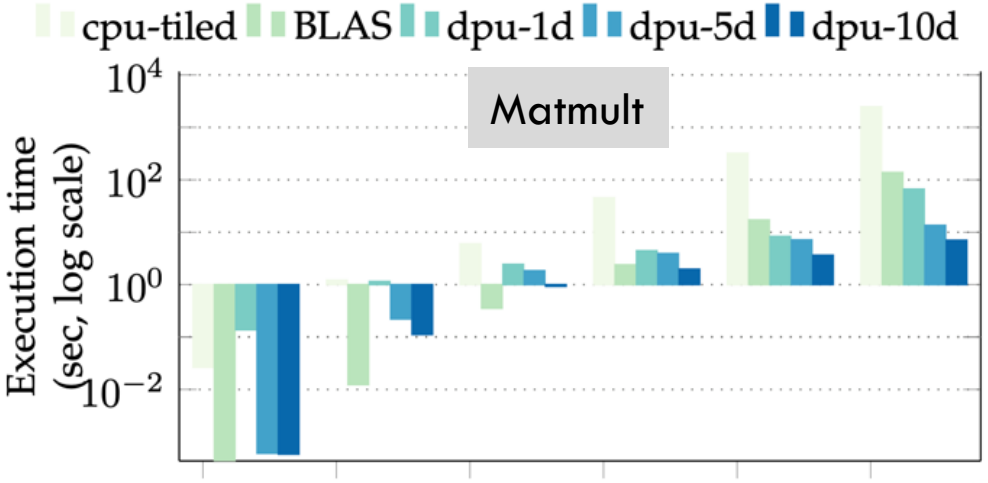
UPMEM example: Matmult

```
def mm(int32(64, 64) A, int32(64, 64) B) -> (int32(64, 64) C) {
  C(i,j) += A(i,k) * B(k,j)
            where i in 0:64, k in 0:64, j in 0:64
}
```



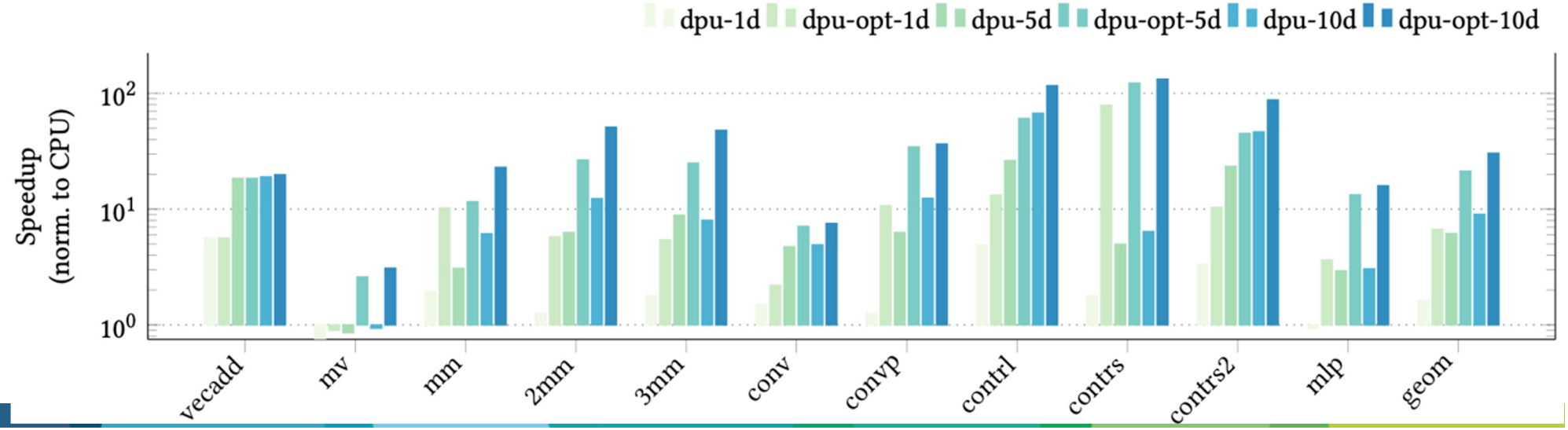

```
    uint32_t mram_base_addr_A = (uint32_t) (DPU_MRAM_HEAP_POINTER );
    uint32_t mram_base_addr_B = (uint32_t) (DPU_MRAM_HEAP_POINTER + ROWS * COLS *
↪   sizeof(T));
    uint32_t mram_base_addr_C = (uint32_t) (DPU_MRAM_HEAP_POINTER + 2 * ROWS * COLS
↪   * sizeof(T));
    for(int i = (tasklet_id * point_per_tasklet) ; i < (
↪   (tasklet_id+1)*point_per_tasklet ) ; i++) {
        if( new_row != row ){
            ...
            mram_read((__mram_ptr void const*) (mram_base_addr_A + mram_offset_A),
↪   cache_A, COLS * sizeof(T));
        }
        mram_read((__mram_ptr void const*) (mram_base_addr_B + mram_offset_B),
↪   cache_B, COLS * sizeof(T));
        dot_product(cache_C, cache_A, cache_B, number_of_dot_products);
        ...
    }
    ...
        mram_write( cache_C, (__mram_ptr void *) (mram_base_addr_C + mram_offset_C),
↪   point_per_tasklet * sizeof(T));
}
```

UPMEM example: Results



1-DIMM 128 DPUs
5-DIMMs 640 DPUs
10-DIMMs 1280-DPUs

6.1×, (1 DIMM)
21.3× (5 DIMM) and
30.4× (10 DIMM) wrt
host CPU



CIM: Lowering examples

```
def contr(int16(K,L,M) A, int16(L,K,N) B)
    -> (int16(M,N) C)
{
  C(m,n) += A(k,l,m) * B(l,k,n)
}
```

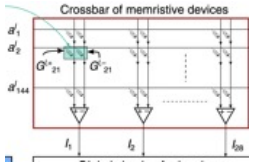
⇓ *lowers to*

```
%0 = linalg.transpose(%A, {2, 0, 1})
%1 = linalg.transpose(%B, {1, 0, 2})
%2 = linalg.reshape(%0, {0, {1, 2}})
%3 = linalg.reshape(%1, {{0, 1}, 2})
// eligible for offloading to CIM
linalg.matmul(%2, %3, %C)
```

```
// loop interchanged GEMM
scf.for %k = %c0 to %numTiles step %c1 {
  scf.for %j = %c0 to %tiledCols step %c1 {
    %tileB = cim.copyTile(%B, %k, %j)
    cim.write(%id, %tileB)
    scf.for %i = %c0 to %tiledRows step %c1 {
      %tileC = cim.copyTile(%C, %i, %j)
      ...
      cim.storeTile(%tileC, %C, %i, %j)
    }
  }
}
```

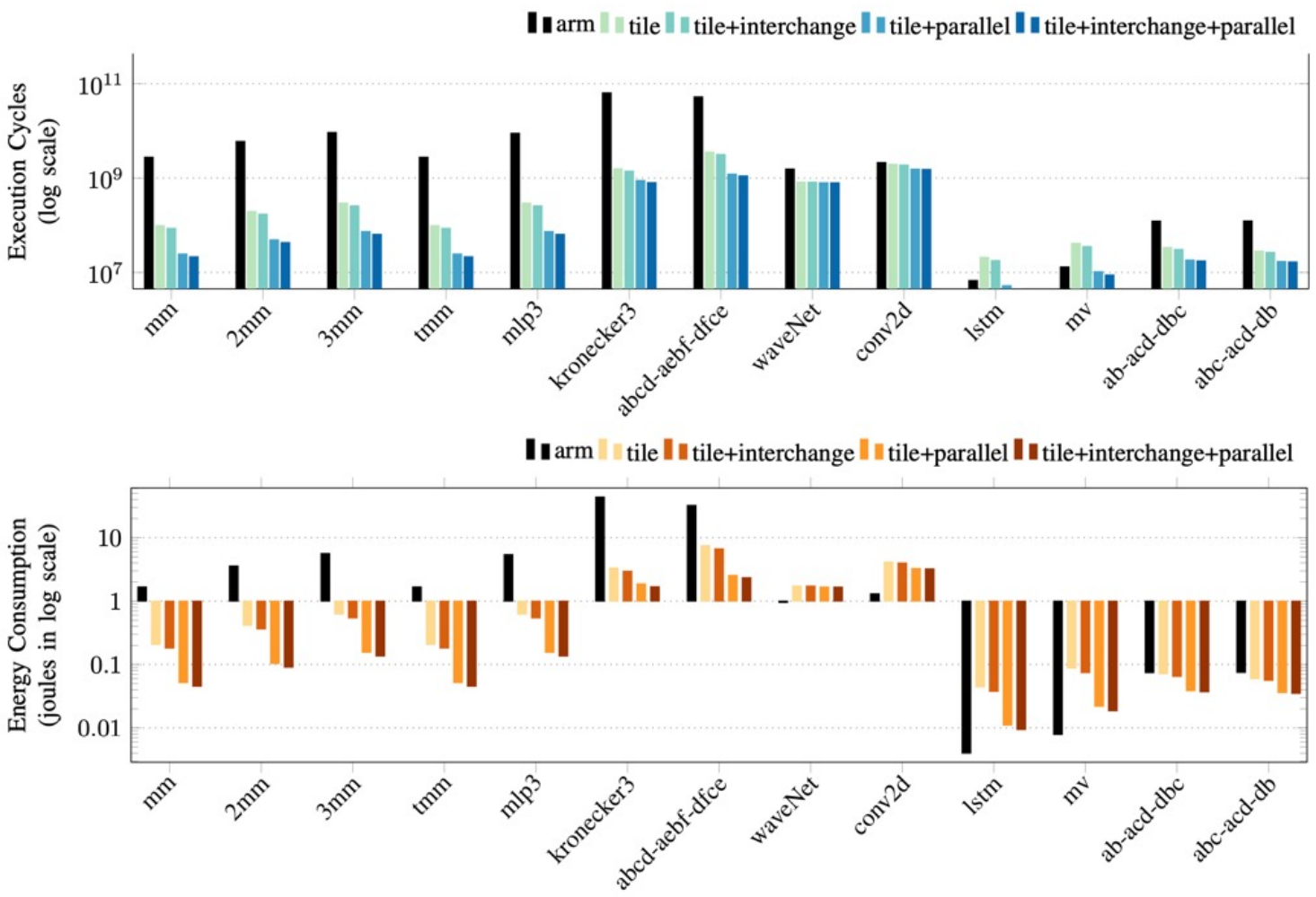
```
linalg.matmul(%A, %B, %C)
```

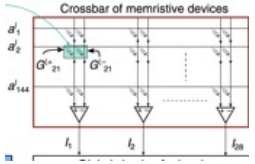
⇓ *lowers to*




```
// tiled GEMM in the CIM dialect
%c0 = constant 0 : i32
%c1 = constant 1 : i32
%id = constant 0 : i32 // tile id
scf.for %i = %c0 to %tiledRows step %c1 {
  scf.for %j = %c0 to %tiledCols step %c1 {
    %tileC = cim.copyTile(%C, %i, %j)
    %tempTile = cim.allocDuplicate(%tileC)
    scf.for %k = %c0 to %numTiles step %c1 {
      %tileA = cim.copyTile(%A, %i, %k)
      %tileB = cim.copyTile(%B, %k, %j)
      cim.write(%id, %tileB)
      cim.matmul(%id, %tileA, %tempTile)
      cim.barrier(%id)
      // tileC += tempTile
      cim.accumulate(%tileC, %tempTile)
    }
    cim.storeTile(%tileC, %C, %i, %j)
  }
}
```

Optimization results: Crossbars beyond matmult



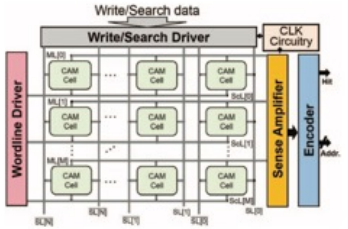


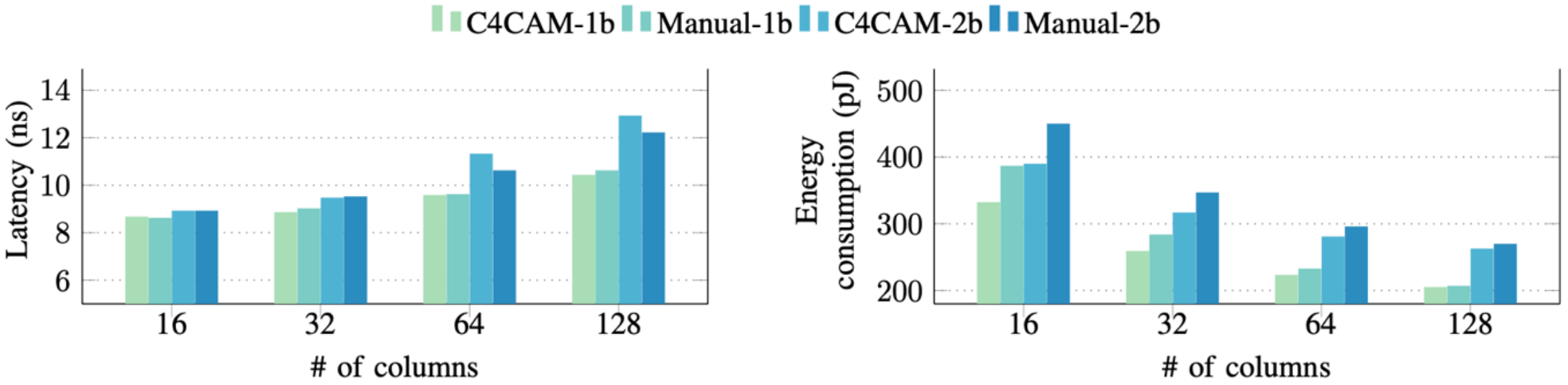
A. Siemieniuk, L
Learning Optim

Machine

Content addressable memories (CAMs)

- NVM-based CAMs: Great for KNNs, One-shot learning, …
- CINM support for **similarity** and **CAM arch exploration**
- Automatic flow from TorchScript **matches manual designs**

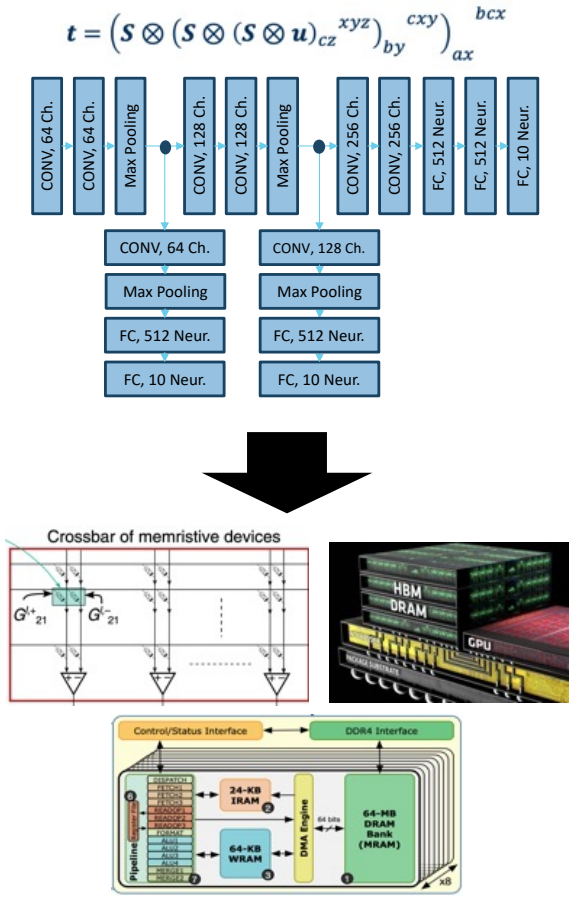




H. Farzaneh, et al. "C4CAM: A Compiler for CAM-based In-memory Accelerators", eprint arXiv:2309.06418, Sep 2023

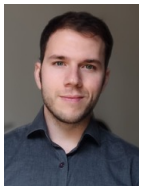
Summary

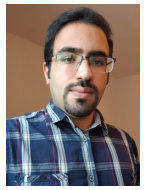
- Next generation programming for extreme heteogeneity
  - Domain-specific abstractions, compilation flows, …
  - Reconfigurable HW, HBM, data placement, near and in-memory computing
- Challenges
  - Understanding and modeling primitives from down below
  - Simulators, prototypes in interdisciplinary research efforts
  - Optimization/DSE: ML? simpler heuristics useful again?
  - Joint work across stack layers will be key!



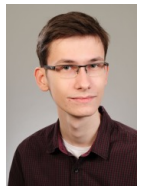
Thanks! & Acknowledgements


Hasna Bouraoui


João P. de Lima


Hamid Farzaneh


Clément Fournier


Karl Friebel


Dr. Asif Khan


Robert Khasanov


Alexander Brauckmann

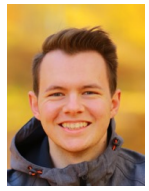

Nesrine Khouzami


Dr. Steffen Köhler


Christian Menard


Julian Robledo


Lars Schütze


Felix Wittwer


Dr. Fazal Hameed

...., and previous members of the group (**Norman Rink,** Sven Karol, Sebastian Ertel, **Andres Goens**), and collaborators (**J. Fröhlich,** I. Sbalzarini, **A. Cohen, T. Grosser, T. Hoefler,** H. Härtig, **H. Corporaal, C. Pilato, S. Parkin, P. Jääskeläinen, J-J. Chen, A. Jones, X.S. Hu**)

Funded by DFG

DISRUPTIVE MEMORY TECHNOLOGIES
DFG PRIORITY PROGRAM 2377

CO4RTM (450944241)
HetCIM (502388442)

https://everest-h2020.eu
EVEREST
GA: 957269

ScaDS.AI DRESDEN LEIPZIG
BMBF (01IS18026A-D)

AMD

STAATSMINISTERIUM FÜR WISSENSCHAFT KULTUR UND TOURISMUS
Freistaat SACHSEN

References

[TMSCS'18] J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems, 2018

[RWDSL'18] N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL'18.

[GPCE'18] A. Susungi, et al. "Meta-programming for cross-domain tensor optimizations" GPCE'18, 79-92.

[Array'19] N.A. Rink, N. A. and J. Castrillon. "TelL: a type-safe imperative Tensor Intermediate Language", ARRAY'19, pp. 57-68

[DATE'21] C. Pilato, et al. "EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms", DATE 2021

[FPGAHPC'21] K. F. A. Friebel, et al. "From Domain-Specific Languages to Memory-Optimized Accelerators for Fluid Dynamics", FPGA for HPC @ IEEE Cluster 2021

[TRETS'22] S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, and C. Pilato. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics". In: ACM TRETS, Sept. 2022.

[HEART'23] K. F. A. Friebel, J. Bi, J. Castrillon, "BASE2: An IR for Binary Numeral Types" (to appear), In ACM HEART 2023

[DATE'23] G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, A. C. Schneider Beck, "Pruning and Early-Exit Co-Optimization for CNN Acceleration on FPGAs", DATE 2023

[ISLVLSI'23] G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, A. C. Schneider Beck, ""Design Space Exploration for CNN Offloading to FPGAs at the Edge", ISVLSI 2023

[ICAL'19] Khan, et al. "RTSim: A Cycle-accurate Simulator for Racetrack Memories", In IEEE Computer Architecture Letters, 2019

[IPROC'20] R. Bläsing, et al. "Magnetic Racetrack Memory: From Physics to the Cusp of Applications within a Decade", In Proceedings of the IEEE, vol. 108, no. 8, pp. 1303-1321, Mar 2020.

[LCTES'19] A. A. Khan, et al. "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", Proceedings of the 20th ACM SIGPLAN/SIGBED LCTES'19, pp. 5-18, Jun 2019

[TECS'20] A. A. Khan, et al. "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories". ACM TECS 2020

[TACO'19] Khan, et al. "ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0", ACM TACO 2019

[TCAD'20] A. A. Khan, et al., "Polyhedral Compilation for Racetrack Memories", In IEEE TCAD'20, vol. 39, no. 11, pp. 3968-3980, Oct 2020.

[DAC'21] C. Hackert, "BLOwing Trees to the Ground: Layout Optimization of Decision Trees on Racetrack Memory", DAC 2021

[CINM'23] A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", arXiv, Aug 2023

[TCAD'21] A. Siemieniuk, et al. "OCC: An Automated End-to-End Machine Learning Optimizing Compiler for Computing-In-Memory", IEEE TCAD, 2021

[C4CAM'23] H. Farzaneh, et al. "C4CAM: A Compiler for CAM-based In-memory Accelerators", eprint arXiv:2309.06418, Sep 2023