CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

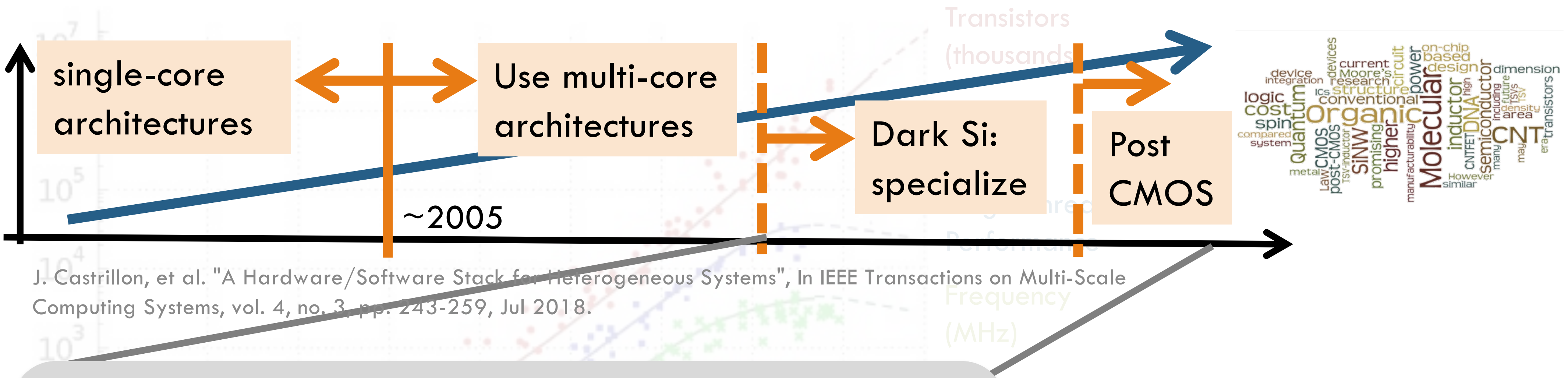
Domain-specific programming methodologies for domain-specific and emerging computing systems

Jeronimo Castrillon
Chair for Compiler Construction (CCC)
TU Dresden, Germany

Keynote: 23rd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2022).
San Diego CA, USA
June 14, 2022

TECHNISCHE UNIVERSITÄT DRESDEN
DRESDEN concept
DFG
WR WISSENSCHAFTSRAT

Evolution of computing: Breaking walls

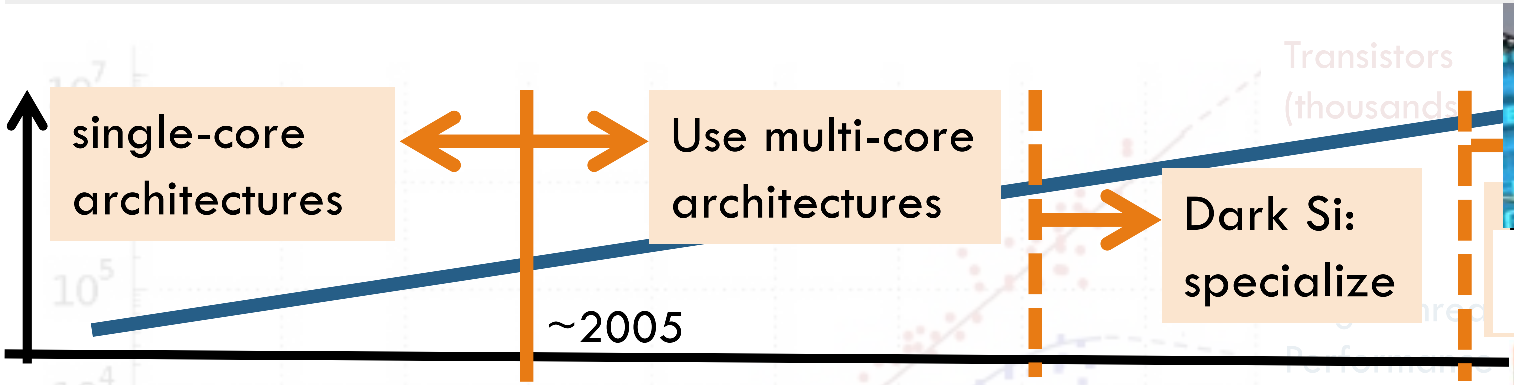


J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 243-259, Jul 2018.

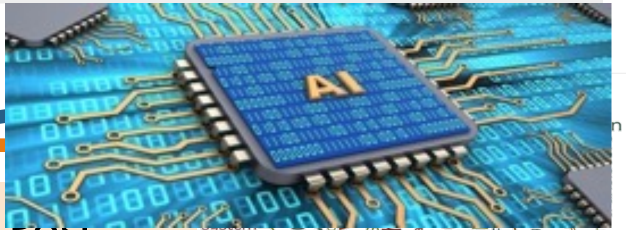
- ❑ Massive parallel and heterogeneous systems
- ❑ Specialization: TPUs, AI engines, Bio-inf, …
- ❑ Novel interconnects & distributed computing
- ❑ Emerging memories & in-memory computing

…al power
…count
…Horowitz, F. Labonte, et al. Dotted-line by C. Moore,
…ta processing in exascale-class computer systems," The
…shan Conference on High Speed Computing, 2011
1975 1980 1985 1990 1995 2000 2005 2010 2015

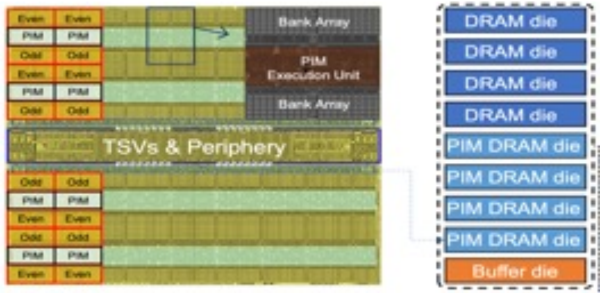
Evolution of computing: Breaking walls



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

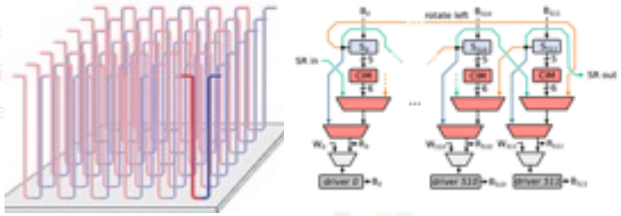


https://www.hpcwire.com/2017/04/10/nvidia-responds-google-tpu-benchmarking/

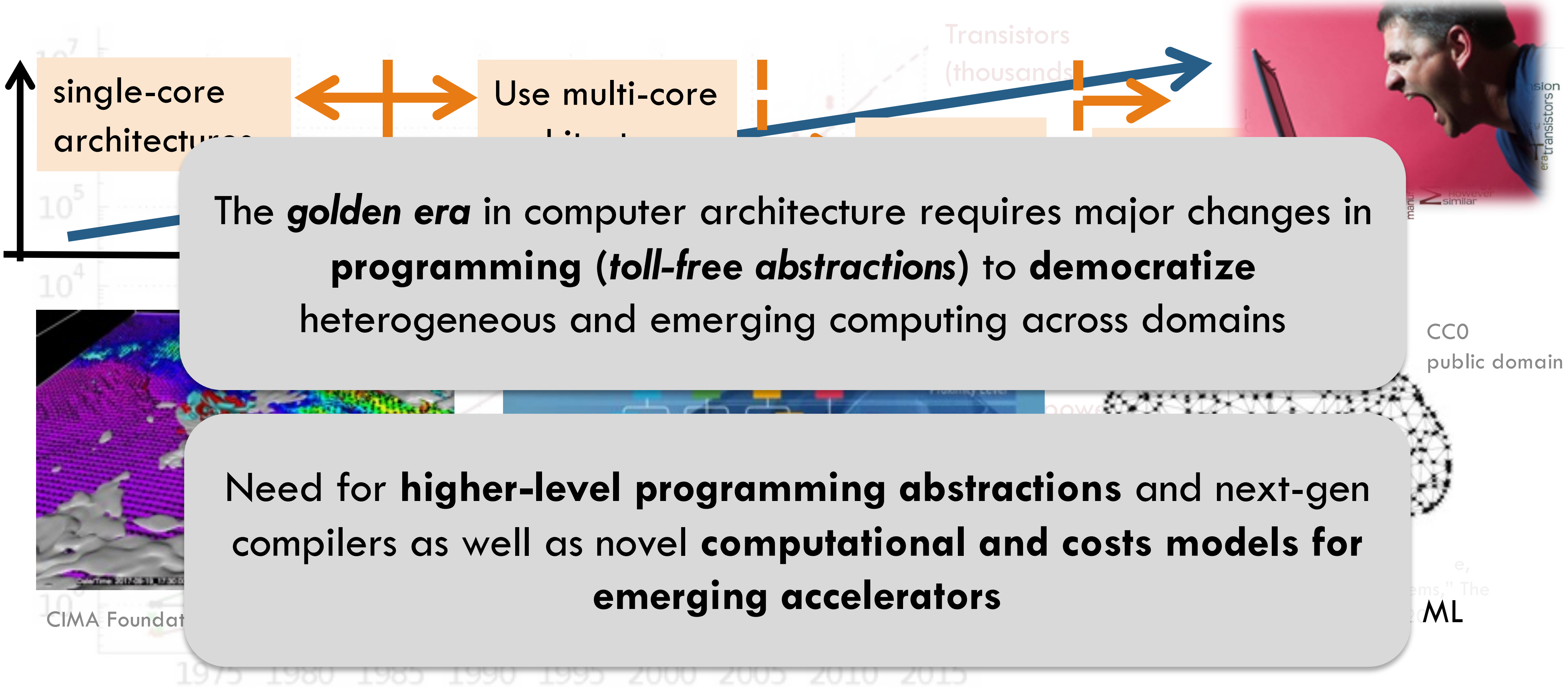


Samsung @ ISCA 2021

- □ Massive parallel and heterogeneous systems
- □ Specialization: TPUs, AI engines, Bio-inf, …
- □ Novel interconnects & distributed computing
- □ Emerging memories & in-memory computing



Evolution of computing: Software



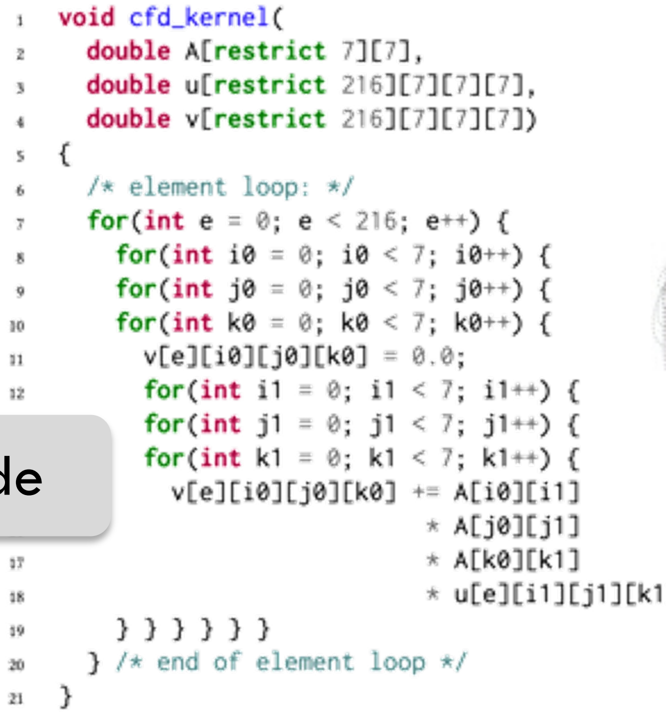
The ***golden era*** in computer architecture requires major changes in **programming (*toll-free abstractions*)** to **democratize** heterogeneous and emerging computing across domains

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

Why new abstractions?

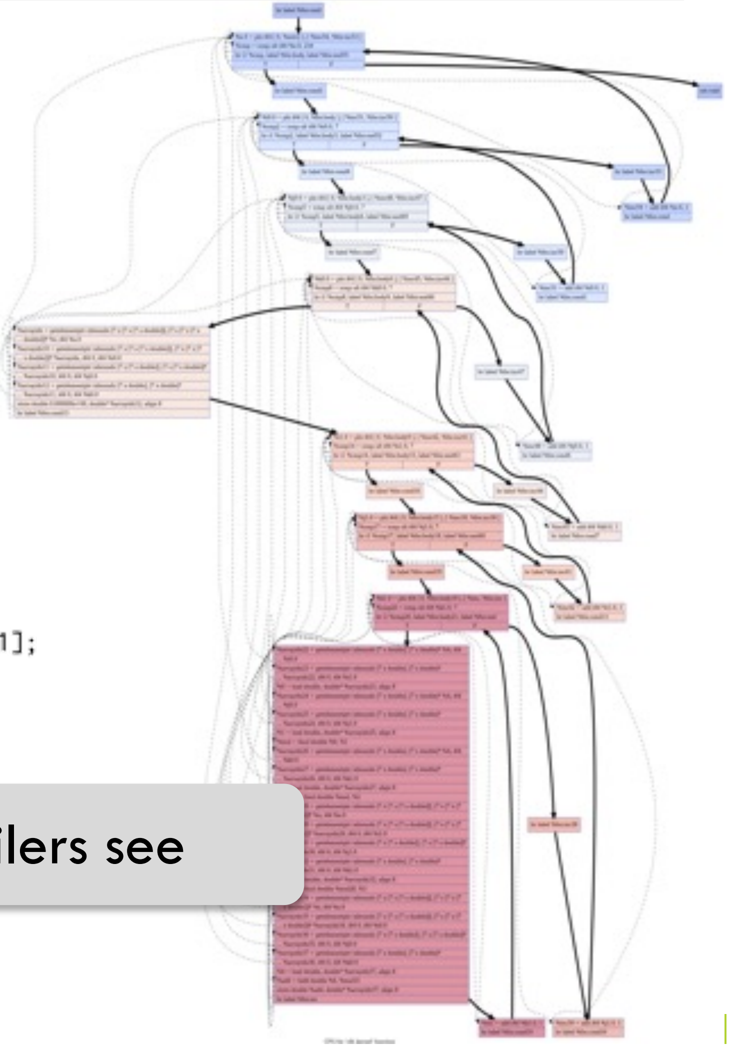
$$v_{ijk,e} = \sum_{i'=0}^{p}\sum_{j'=0}^{p}\sum_{k'=0}^{p} A_{kk'}A_{jj'}A_{ii'}u_{i'j'k'e}$$

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]
                          * A[k0][k1]
                          * u[e][i1][j1][k1];
    } } } } } }
  } /* end of element loop */
}
```

What we (naively) code

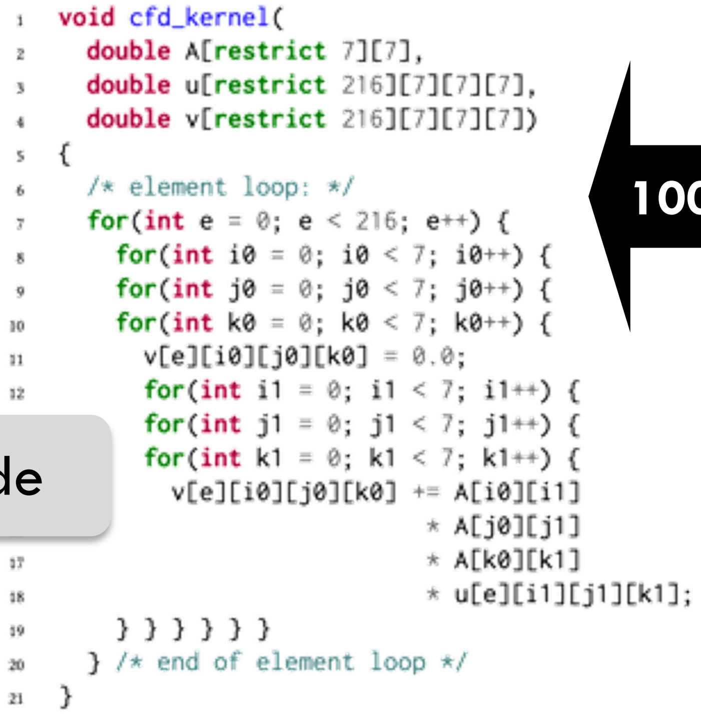


What compilers see

Semantic gap → performance gap

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

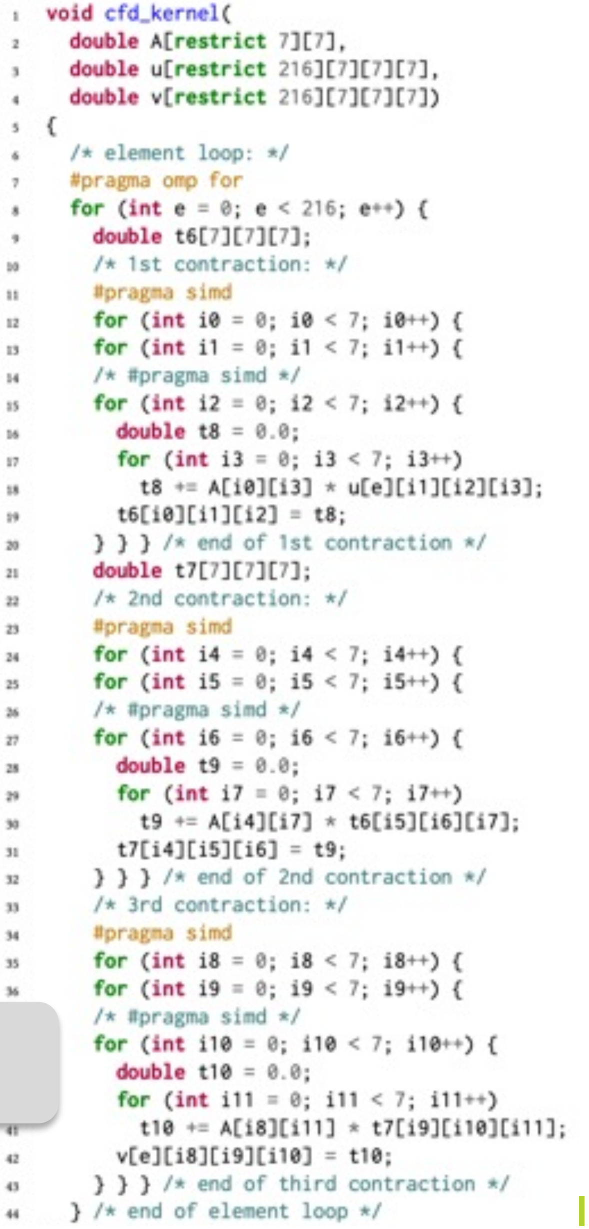
What we want



```
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]
                          * A[k0][k1]
                          * u[e][i1][j1][k1];
    } } } } } }
  } /* end of element loop */
}
```

What we (naively) code

100X



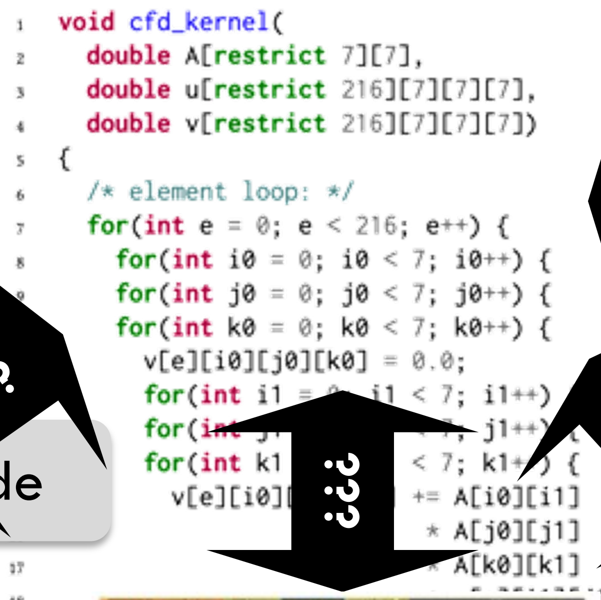
```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  #pragma omp for
  for (int e = 0; e < 216; e++) {
    double t6[7][7][7];
    /* 1st contraction: */
    #pragma simd
    for (int i0 = 0; i0 < 7; i0++) {
    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 */
    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 */
```

What performance experts code

Semantic gap → performance gap

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

What we want

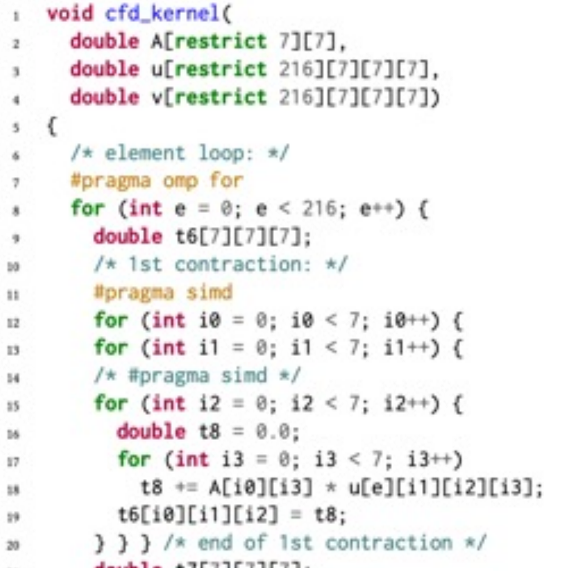


```
void cfd_kernel(
  double A[restrict 7][7],
  double u[restrict 216][7][7][7],
  double v[restrict 216][7][7][7])
{
  /* element loop: */
  for(int e = 0; e < 216; e++) {
    for(int i0 = 0; i0 < 7; i0++) {
      for(int j0 = 0; j0 < 7; j0++) {
        for(int k0 = 0; k0 < 7; k0++) {
          v[e][i0][j0][k0]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



}
```

y) 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 */
```

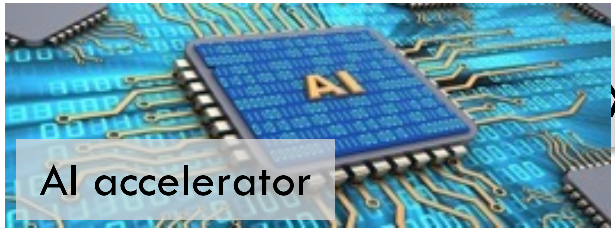
```
      double t10 = 0.0;
      for (int i11 = 0; i11 < 7; i11++)
        t10 += A[i8][i11] * t7[i9][i10][i11];
      v[e][i8][i9][i10] = t10;
    } } } /* end of third contraction */
  } /* end of element loop */
```

100X

???

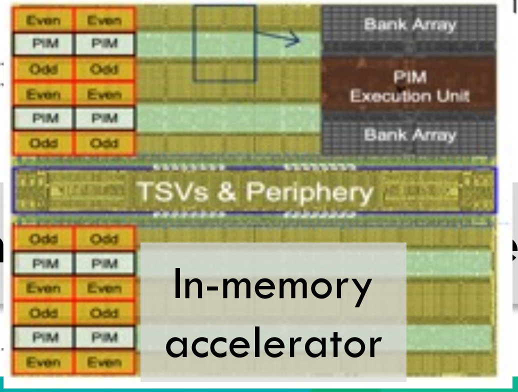
???

???



AI accelerator

https://www.hpcwire.com/2017/04/10/nvidia-responds-google-tpu-benchmarking/



In-memory accelerator

Lee, Sukhan, et al. "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product." ISCA 2021.

HBM-FPGA

Wh

rts code

© Pro

Example for tensors expressions (CFD, ML)

- Expression-language for tensor operations and optimizations
  - Originally for spectral element methods in computational fluid dynamics

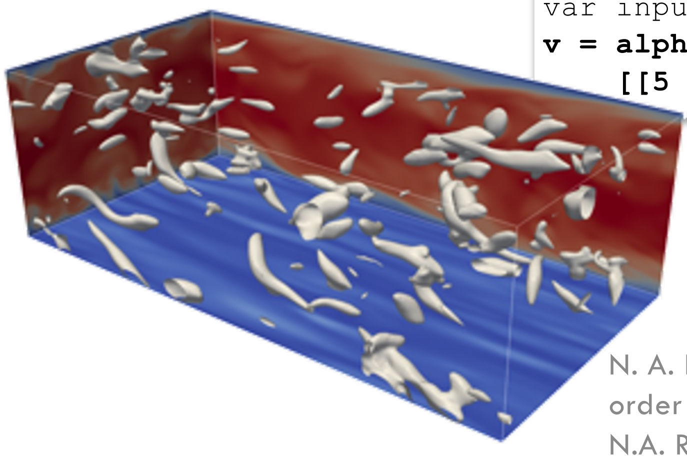
$$\mathbf{v}_e = (\mathbf{A} \otimes \mathbf{A} \otimes \mathbf{A})\, \mathbf{u}_e$$

Interpolation kernel

```
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);
```

Fortran and C++ integration



CC0
public domain

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

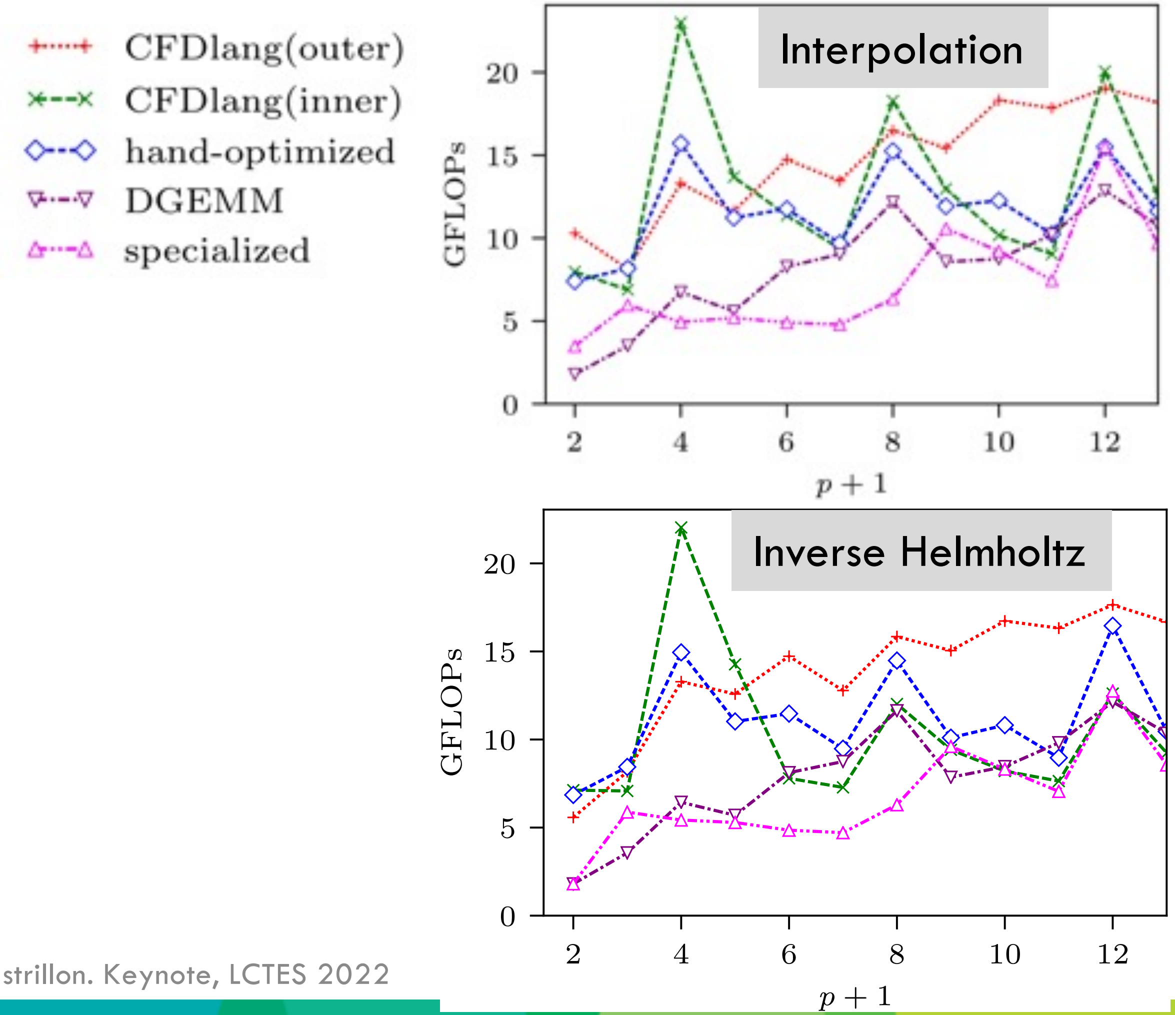
Closing the performance gap

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

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

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

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



Interpolation

Inverse Helmholtz

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.

Closing the performance gap

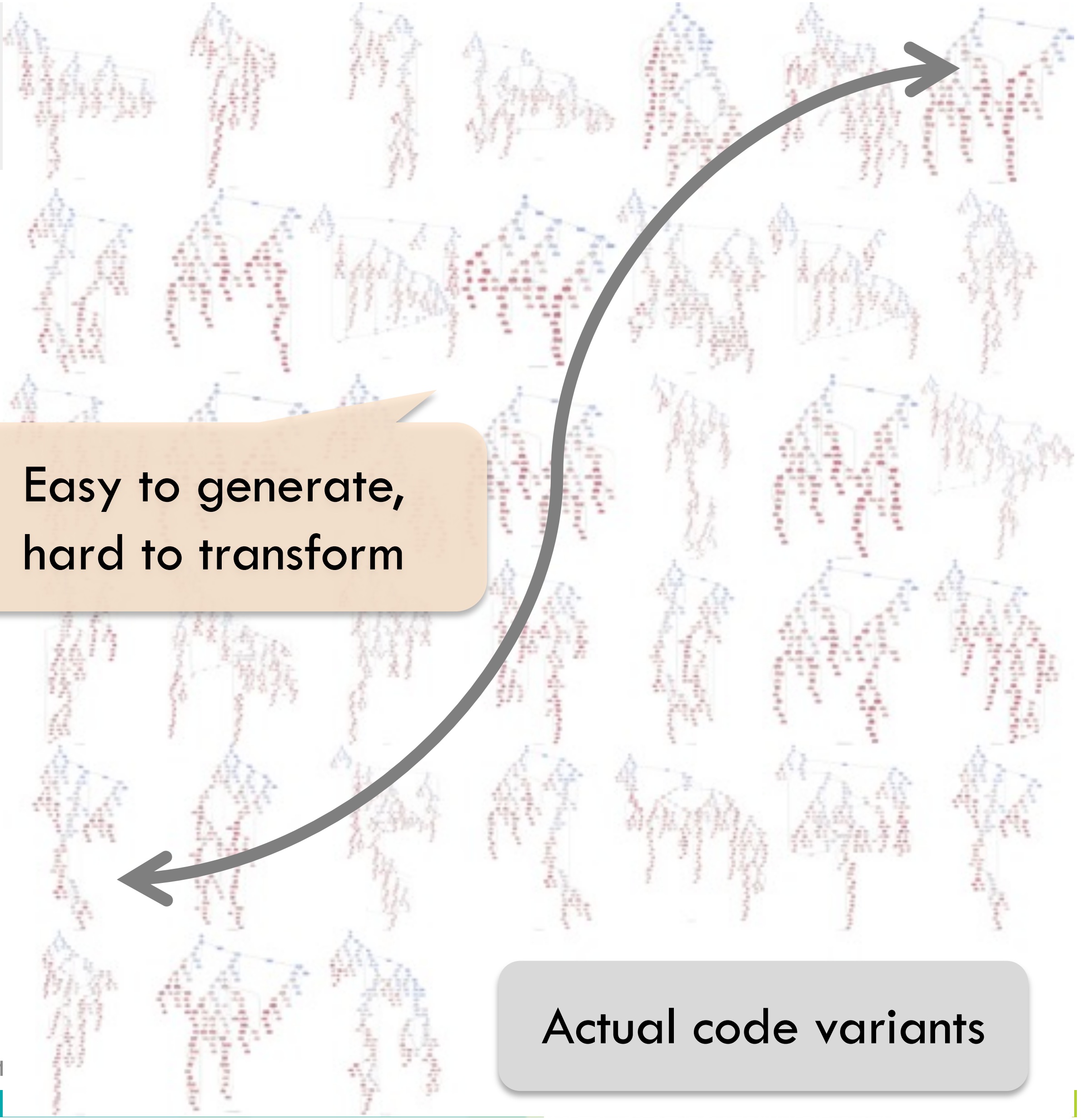
- ❑ Not really optimization magic
  - ❑ Leverage expert knowledge
  - ❑ Algebraic identities

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

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

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

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.

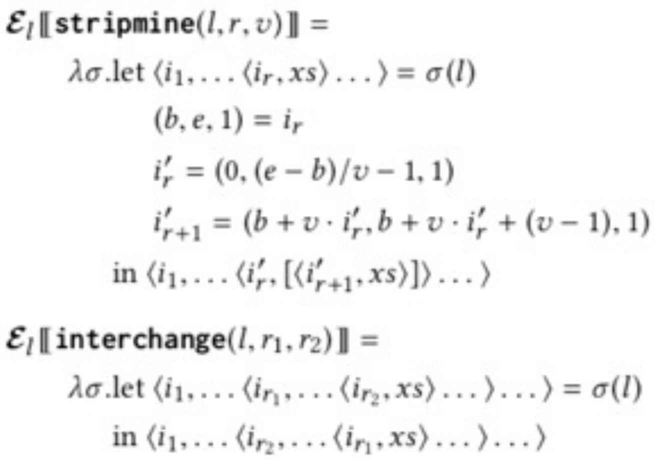


Easy to generate, hard to transform

Actual code variants

TeML: Meta-programming for tensor optimizations

- ❑ Generalize for cross-domain tensor expressions
- ❑ Formal semantics and composition of transformations

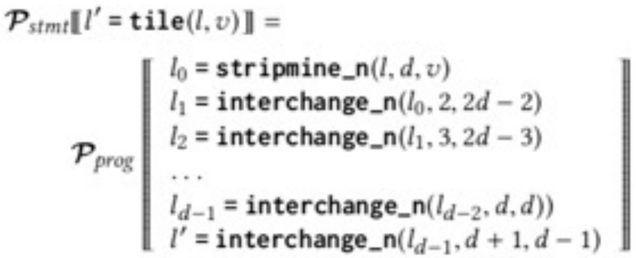


$$\mathcal{E}_l[\![\texttt{stripmine}(l, r, v)]\!] =$$

$$\begin{aligned}
&\lambda\sigma.\text{let } \langle i_1, \ldots \langle i_r, xs\rangle \ldots\rangle = \sigma(l)\\
&\qquad (b, e, 1) = i_r\\
&\qquad i'_r = (0, (e - b)/v - 1, 1)\\
&\qquad i'_{r+1} = (b + v \cdot i'_r, b + v \cdot i'_r + (v - 1), 1)\\
&\quad \text{in } \langle i_1, \ldots \langle i'_r, [\langle i'_{r+1}, xs\rangle]\rangle \ldots\rangle
\end{aligned}$$

$$\mathcal{E}_l[\![\texttt{interchange}(l, r_1, r_2)]\!] =$$

$$\begin{aligned}
&\lambda\sigma.\text{let } \langle i_1, \ldots \langle i_{r_1}, \ldots \langle i_{r_2}, xs\rangle \ldots\rangle \ldots\rangle = \sigma(l)\\
&\quad \text{in } \langle i_1, \ldots \langle i_{r_2}, \ldots \langle i_{r_1}, xs\rangle \ldots\rangle \ldots\rangle
\end{aligned}$$

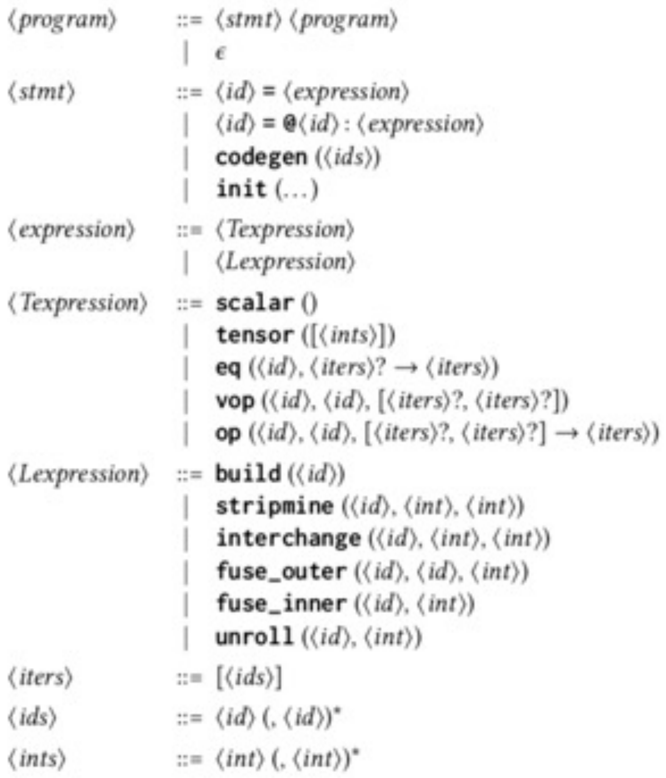


$$\mathcal{P}_{stmt}[\![l' = \texttt{tile}(l, v)]\!] =$$

$$\mathcal{P}_{prog}\left[\!\!\left[\begin{array}{l}
l_0 = \texttt{stripmine\_n}(l, d, v)\\
l_1 = \texttt{interchange\_n}(l_0, 2, 2d - 2)\\
l_2 = \texttt{interchange\_n}(l_1, 3, 2d - 3)\\
\ldots\\
l_{d-1} = \texttt{interchange\_n}(l_{d-2}, d, d))\\
l' = \texttt{interchange\_n}(l_{d-1}, d + 1, d - 1)
\end{array}\right]\!\!\right]$$

Formally defined transformation primitives

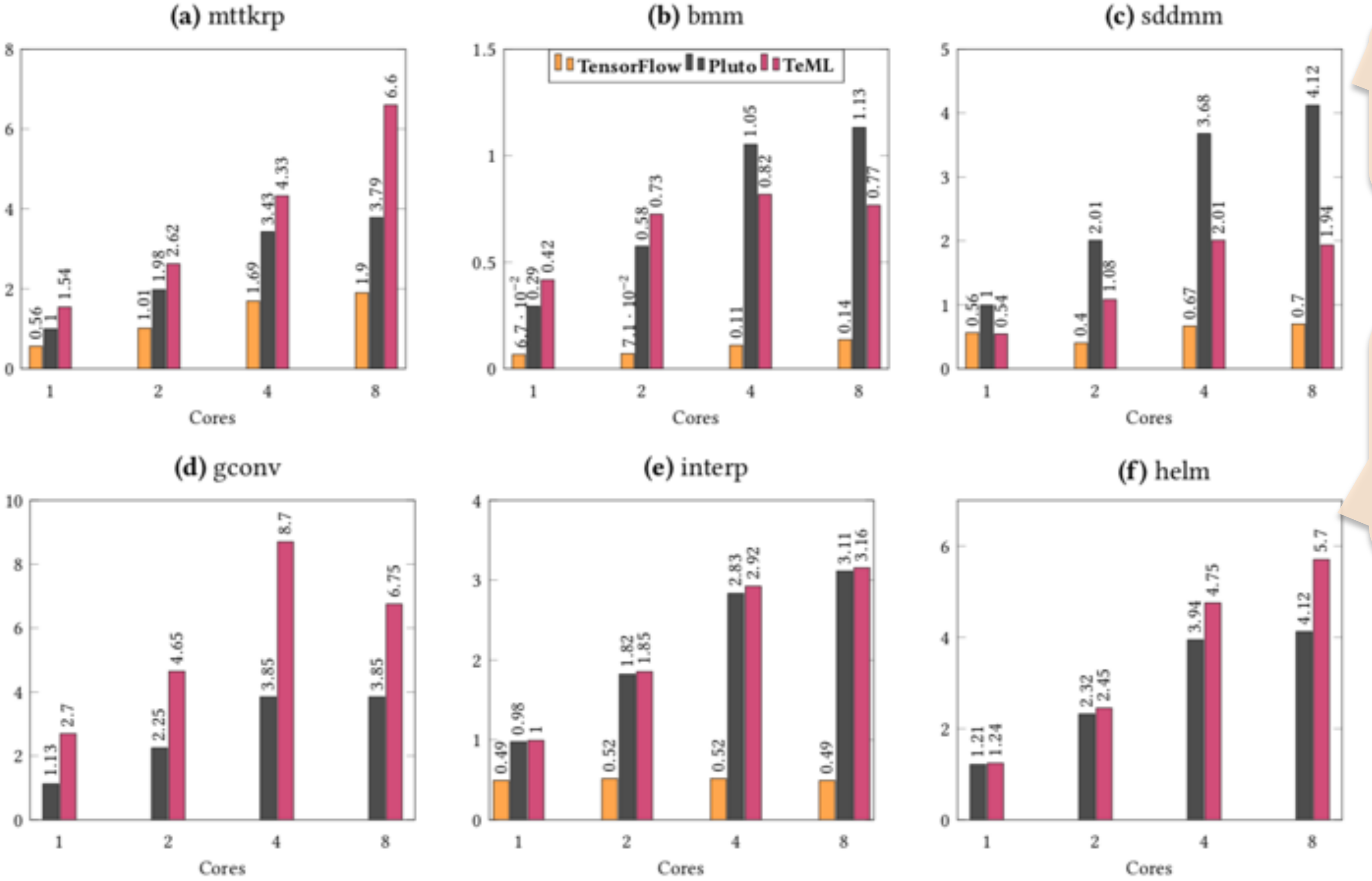
Higher-level transformations via composition



```
⟨program⟩       ::= ⟨stmt⟩ ⟨program⟩
                 |   ε
⟨stmt⟩          ::= ⟨id⟩ = ⟨expression⟩
                 |   ⟨id⟩ = @⟨id⟩ : ⟨expression⟩
                 |   codegen (⟨ids⟩)
                 |   init (...)
⟨expression⟩    ::= ⟨Texpression⟩
                 |   ⟨Lexpression⟩
⟨Texpression⟩   ::= scalar ()
                 |   tensor ([⟨ints⟩])
                 |   eq (⟨id⟩, ⟨iters⟩? → ⟨iters⟩)
                 |   vop (⟨id⟩, ⟨id⟩, [⟨iters⟩?, ⟨iters⟩?])
                 |   op (⟨id⟩, ⟨id⟩, [⟨iters⟩?, ⟨iters⟩?] → ⟨iters⟩)
⟨Lexpression⟩   ::= build (⟨id⟩)
                 |   stripmine (⟨id⟩, ⟨int⟩, ⟨int⟩)
                 |   interchange (⟨id⟩, ⟨int⟩, ⟨int⟩)
                 |   fuse_outer (⟨id⟩, ⟨id⟩, ⟨int⟩)
                 |   fuse_inner (⟨id⟩, ⟨int⟩)
                 |   unroll (⟨id⟩, ⟨int⟩)
⟨iters⟩         ::= [⟨ids⟩]
⟨ids⟩           ::= ⟨id⟩ (, ⟨id⟩)*
⟨ints⟩          ::= ⟨int⟩ (, ⟨int⟩)*
```

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

Cross-domain tensor optimization



(a) mttkrp

(b) bmm

(c) sddmm

(d) gconv

(e) interp

(f) helm

Performance of Pluto could be reproduced

Higher abstraction → more optimization potential

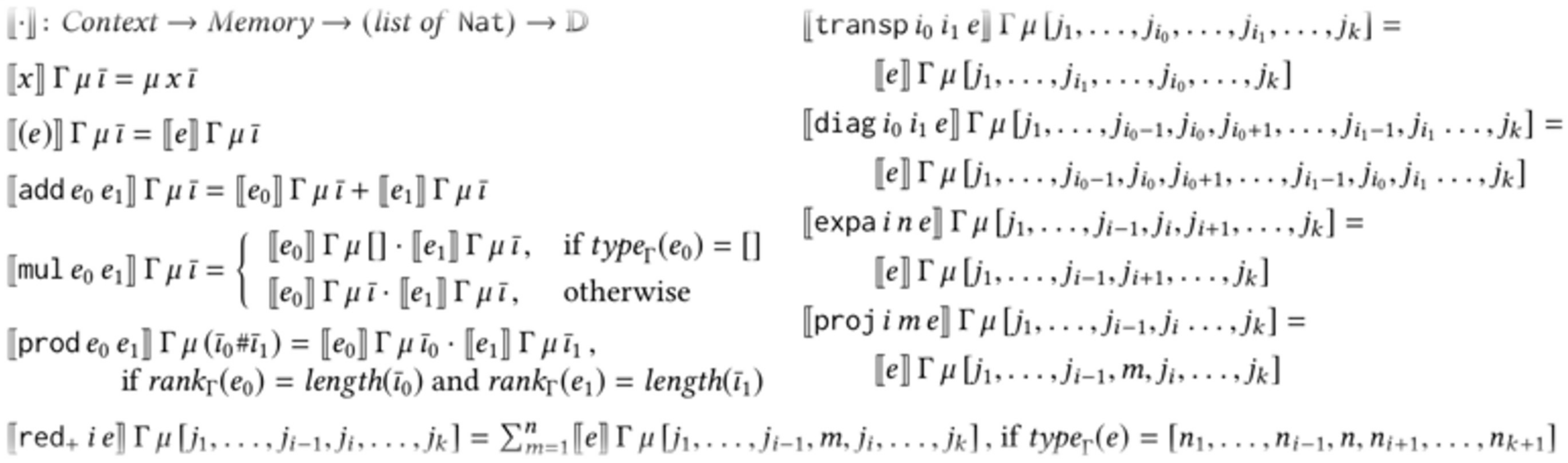
A. Susungi, et al. "Meta-programming for cross-domain tensor optimizations", GPCE'18, 79-92

TeIL: Formal language – added value

- ❑ Core common to multiple tensor languages
- ❑ Index-free notation and strong type system
- ❑ **Provably** no out-of-bound accesses

```
A = placeholder((m,h), name='A')
B = placeholder((n,h), name='B')
k = reduce_axis((0,h), name='k')
C = compute((m,n), lambda i, j:
      sum(A[k, i] * B[k, j], axis=k))
```

$$C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj}$$



$$\llbracket \cdot \rrbracket : \mathit{Context} \longrightarrow \mathit{Memory} \longrightarrow (\mathit{list\ of}\ \mathsf{Nat}) \longrightarrow \mathbb{D}$$

$$\llbracket x \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \mu\, x\, \bar{\imath}$$

$$\llbracket (e) \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \llbracket e \rrbracket\, \Gamma\, \mu\, \bar{\imath}$$

$$\llbracket \mathsf{add}\, e_0\, e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath} + \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}$$

$$\llbracket \mathsf{mul}\, e_0\, e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \begin{cases} \llbracket e_0 \rrbracket\, \Gamma\, \mu\, [\,] \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}, & \text{if } \mathit{type}_\Gamma(e_0) = [\,] \\ \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath} \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}, & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{prod}\, e_0\, e_1 \rrbracket\, \Gamma\, \mu\, (\bar{\imath}_0 \# \bar{\imath}_1) = \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath}_0 \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}_1\,, \quad \text{if } \mathit{rank}_\Gamma(e_0) = \mathit{length}(\bar{\imath}_0) \text{ and } \mathit{rank}_\Gamma(e_1) = \mathit{length}(\bar{\imath}_1)$$

$$\llbracket \mathsf{transp}\, i_0\, i_1\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_0}, \ldots, j_{i_1}, \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_1}, \ldots, j_{i_0}, \ldots, j_k]$$

$$\llbracket \mathsf{diag}\, i_0\, i_1\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_0-1}, j_{i_0}, j_{i_0+1}, \ldots, j_{i_1-1}, j_{i_1} \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_0-1}, j_{i_0}, j_{i_0+1}, \ldots, j_{i_1-1}, j_{i_0}, j_{i_1} \ldots, j_k]$$

$$\llbracket \mathsf{expa}\, i\, n\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i, j_{i+1}, \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_{i+1}, \ldots, j_k]$$

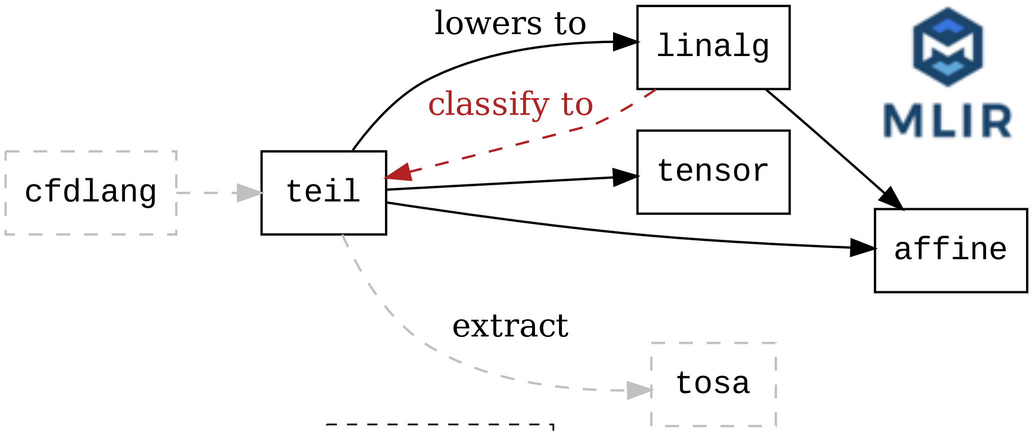
$$\llbracket \mathsf{proj}\, i\, m\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, m, j_i, \ldots, j_k]$$

$$\llbracket \mathsf{red}_+\, i\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i, \ldots, j_k] = \textstyle\sum_{m=1}^{n} \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, m, j_i, \ldots, j_k]\,, \text{ if } \mathit{type}_\Gamma(e) = [n_1, \ldots, n_{i-1}, n, n_{i+1}, \ldots, n_{k+1}]$$

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

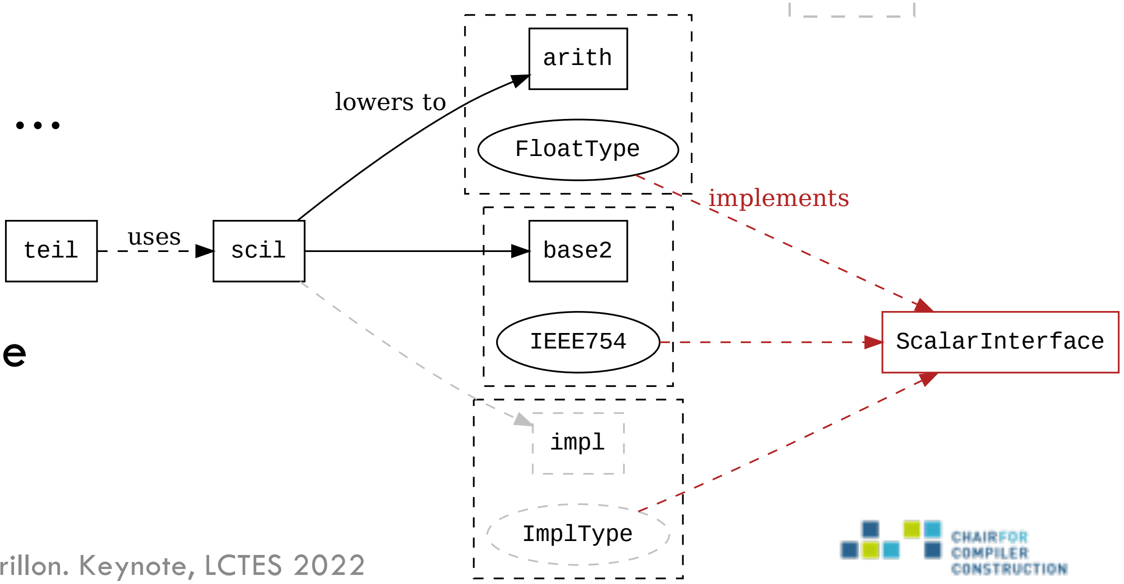
TeIL in MLIR

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



- ❑ Scalar types
  - ❑ SclL provides scalar operators
  - ❑ SclL provides Rationals, Neutrals, …
  - ❑ Base2 provides parametric

    binary number types
  - ❑ Based2 models (custom) hardware



Multi-level lowering with MLIR

- Encoding transformations

```
%y = teil.contr 6 9 %x : ...
```

Fission →

```
%1 = teil.diag 6 9 %x : ...
%y = teil.red "add" 6 %1 : ...
```

```
%1 = teil.prod %S %S : ...
%2 = teil.prod %1 %S : ...
%3 = teil.prod %2 %u : ...
%y = teil.red "add" 6 %3
```

Commutativity
Distributivity →

```
%1 = teil.prod %S %u : ...
%2 = teil.diag 2 5 %1 : ...
%3 = teil.red "add" 2 %2 : ...
%4 = teil.prod %S %3 : ...
%y = teil.prod %S %4 : ...
```

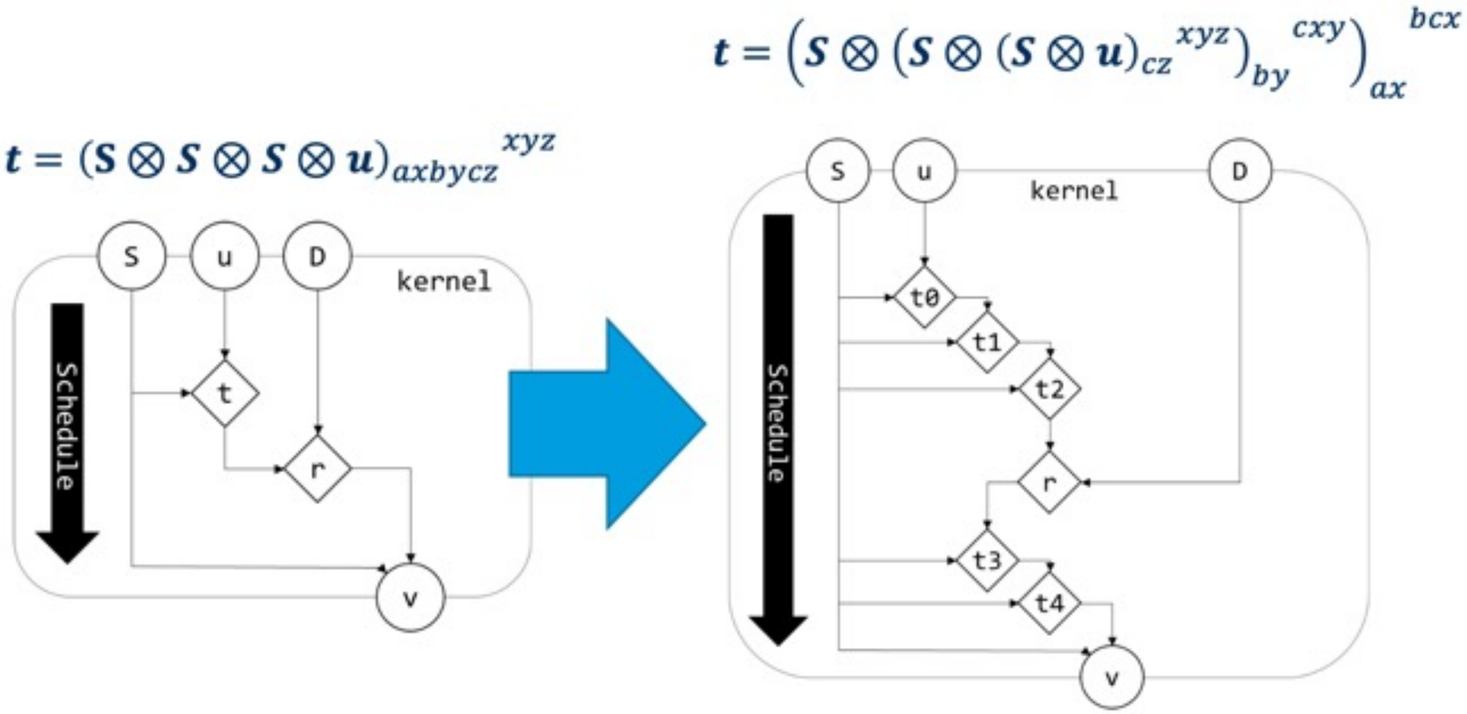
```
%1 = teil.prod %A, %B :
    tensor<?x?xf64>,
    tensor<?x?x?xf64>
%2 = teil.diag 2 3 %1 : ...
%y = teil.red 2 %2 : ...
```

Fusion →

```
%b = linalg.tensor_reshape %B
    [(i,j,k)->(i), (i,j,k)->(j,k)]
    ...
linalg.matmul
    ins(%A, %b : ...)
    outs(%y : ...)
    ...
```

Domain-specific optimization

- ❑ Encode algebraic transformations (Interpolation as example)
- ❑ Direct feedback to expert via DSL export



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

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

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

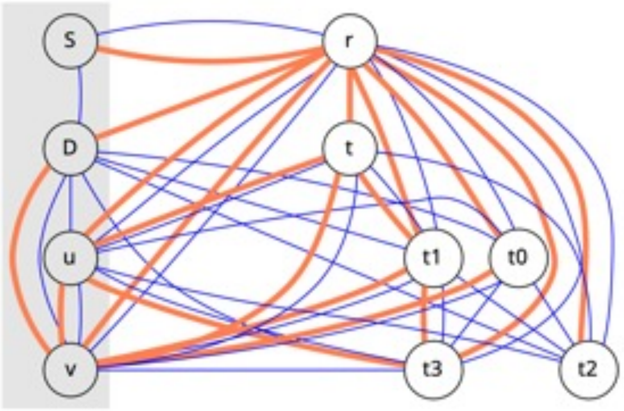
- ❑ Inverse Helmholtz Kernel

https://everest-h2020.eu

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

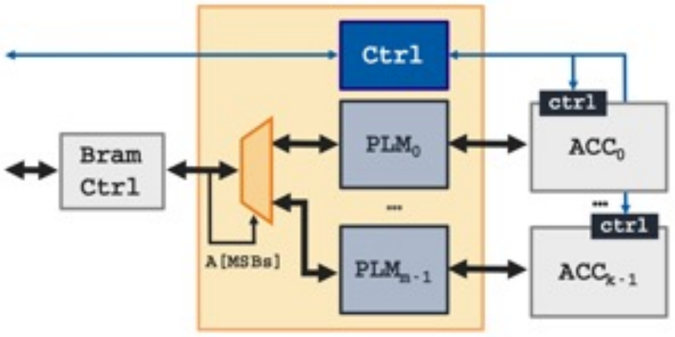
```
t = S # S # S # u . [[1
r = D * t
v = S # S # S # r . [[0
```

Lifetime analysis (polyhedral analysis)



Menosyne

mem-subsystem gen (buffer sharing)

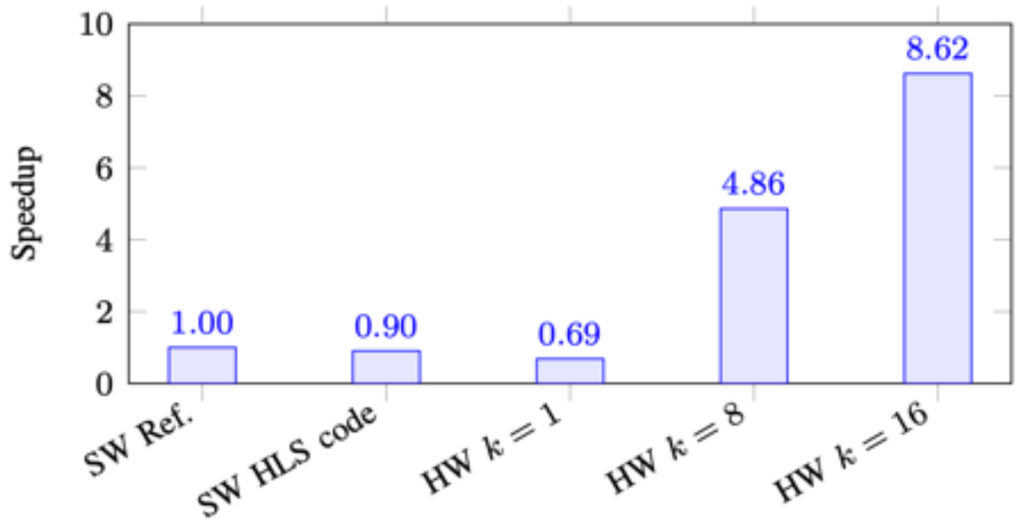


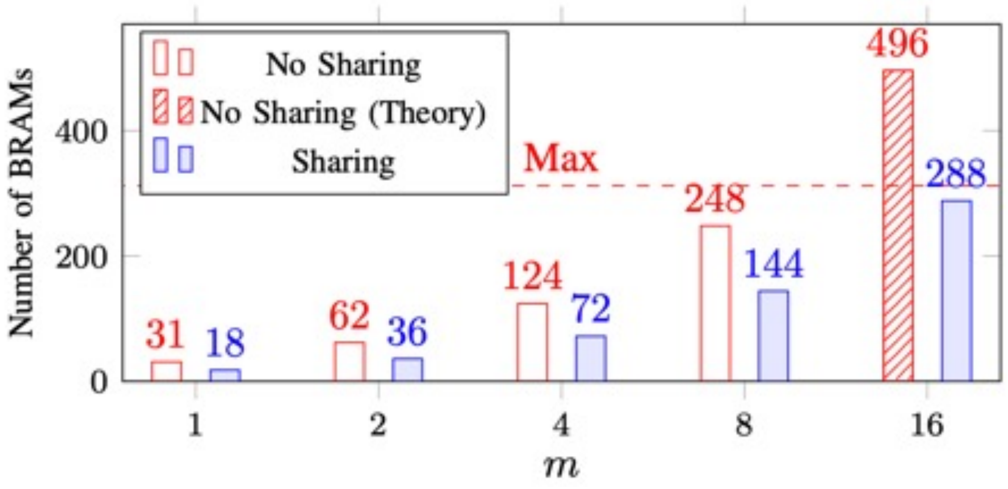
K. F. A. Friebel, S. Soldavini, G. Hempel, C. Pilato, J. Castrillon, "From Domain-Specific Languages to Memory-Optimized Accelerators for Fluid Dynamics", Proceedings of the FPGA for HPC Workshop, held in conjunction with IEEE Cluster 2021, Sep 2021

FPGA code generation: Bus-attached FPGAs

- ❑ H2020 EU Project: Convergence HPC, Big Data and ML
- ❑ Exploring configurations on small FPGAs
- ❑ Example on Inverse Helmholtz kernel

https://everest-h2020.eu



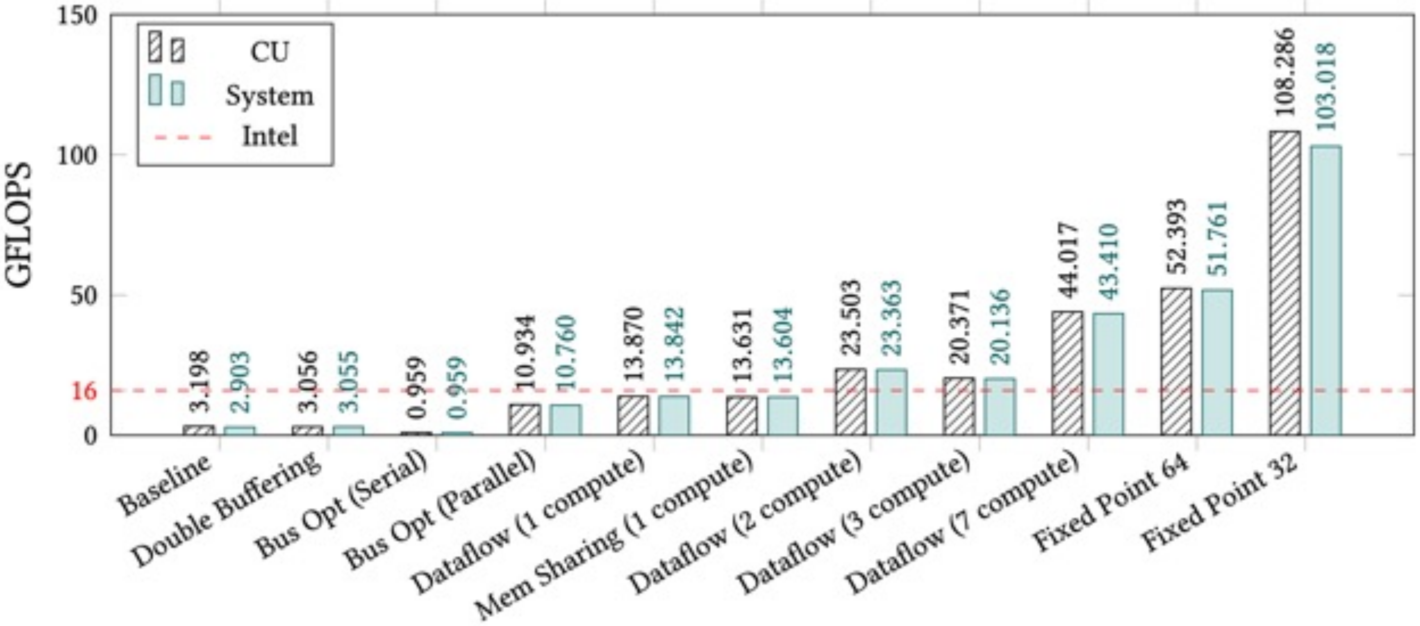


K. F. A. Friebel, S. Soldavini, G. Hempel, C. Pilato, J. Castrillon, "From Domain-Specific Languages to Memory-Optimized Accelerators for Fluid Dynamics", Proceedings of the FPGA for HPC Workshop, held in conjunction with IEEE Cluster 2021, Sep 2021

FPGA code generation: HBM FPGA

- ❑ H2020 EU Project: Convergence HPC, Big Data and ML
- ❑ HBM-FPGA and Cloud FPGA (ongoing)

https://everest-h2020.eu

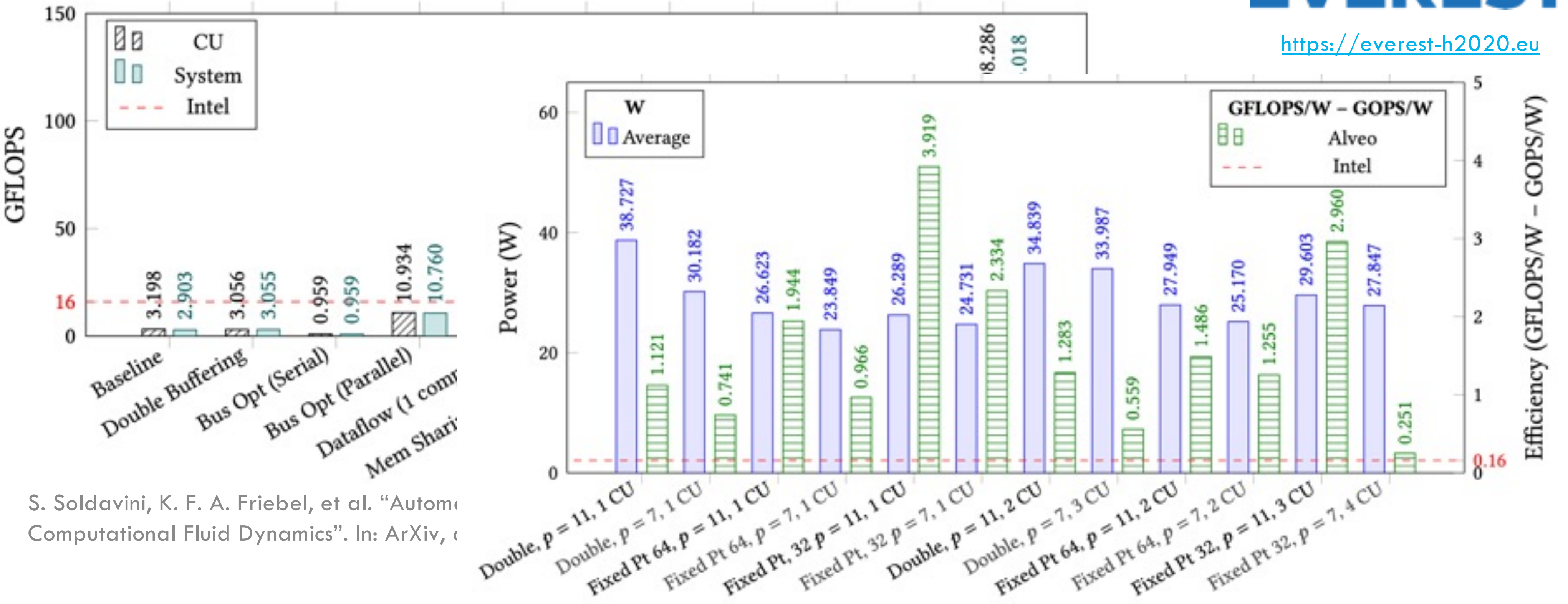


S. Soldavini, K. F. A. Friebel, et al. “Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics”. In: ArXiv, arXiv:2203.10850 (Mar. 2022)

FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- HBM-FPGA and Cloud FPGA (ongoing)

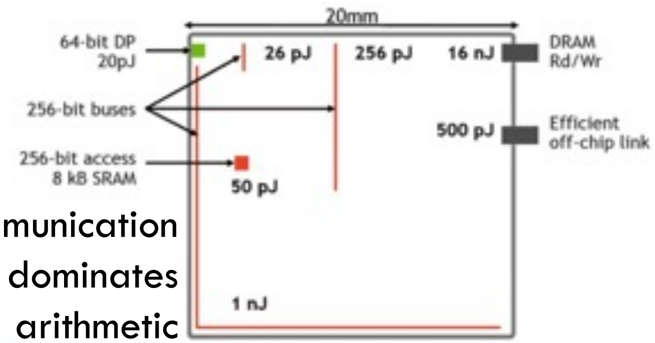
https://everest-h2020.eu



S. Soldavini, K. F. A. Friebel, et al. "Autom
Computational Fluid Dynamics". In: ArXiv,

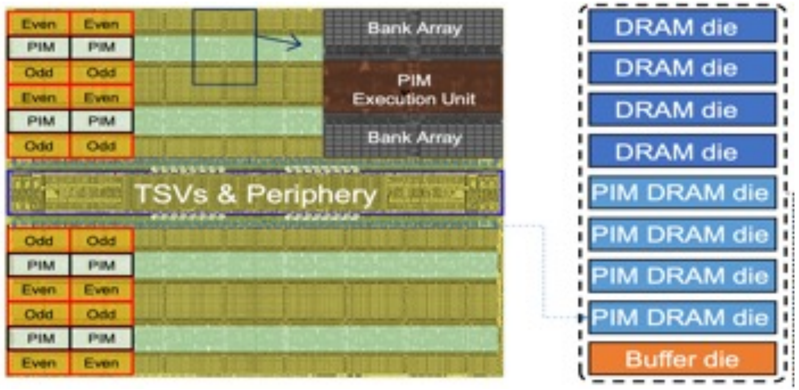
Emerging data-centric architectures

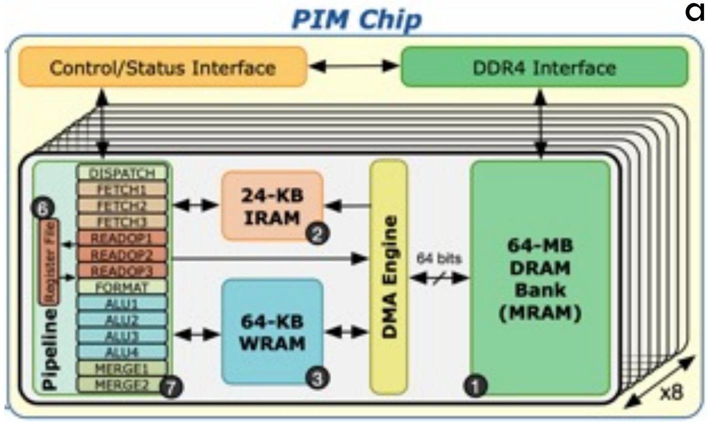
- ❑ Compute (almost) in-place, avoid data movement, transformations to match primitives
- ❑ Novel architectures for near-memory and in-memory computing



Communication dominates arithmetic

Bartolini, S., and Biagio P. "Parallel Programming in Cyber-Physical Systems." Cyber-Physical Systems Security. Springer 2018





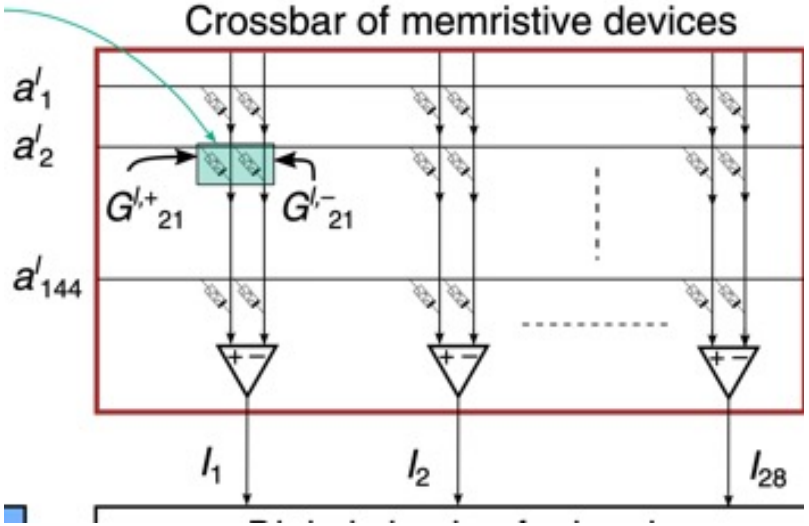
Samsung, Lee, Sukhan, et al. "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product." ISCA 2021.

UPMEM by Gómez-Luna, Juan, et al. "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture." arXiv preprint arXiv:2105.03814 (2021).

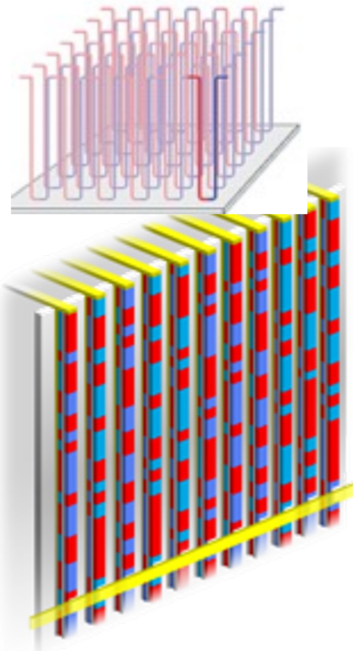
Emerging memories and in-memory computing

- ❑ Compute in-place, avoid data movement, transformations to match primitives

In-PCM Computing

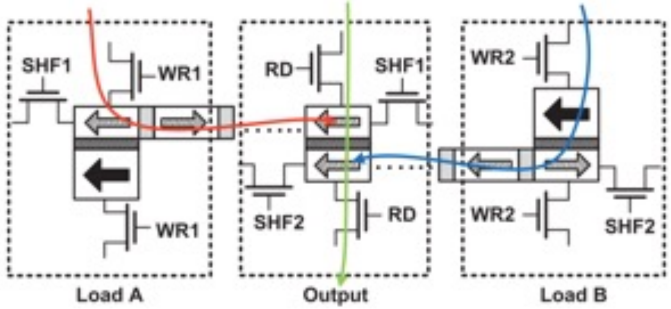


Joshi, Vinay, et al. "Accurate deep neural network inference using computational phase-change memory." Nature Communications 11.1 (2020): 1-13.



Parkin, Stuart SP, Masamitsu Hayashi, and Luc Thomas. "Magnetic domain-wall racetrack memory." Science 320.5873 (2008): 190-194.

In-RTM Computing

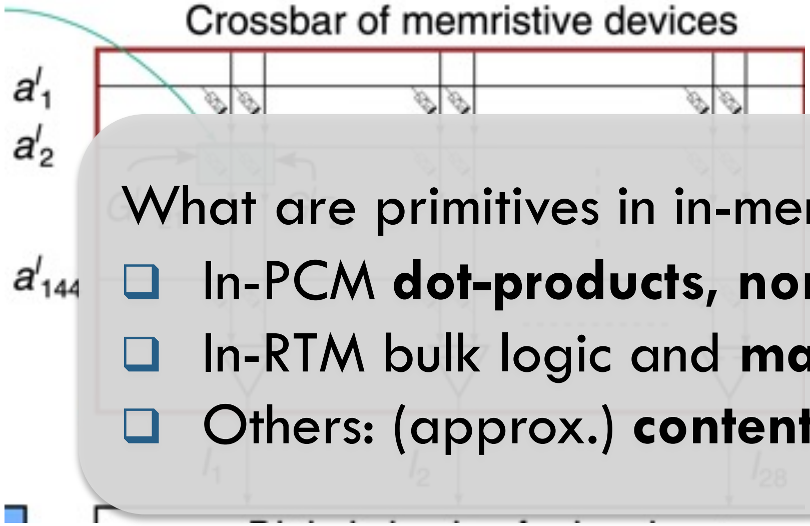


Y. Wang et al., "An Energy-Efficient Nonvolatile In-Memory Computing Architecture for Extreme Learning Machine by Domain-Wall Nanowire Devices," in IEEE Transactions on Nanotechnology, 2015.

Emerging memories and in-memory computing

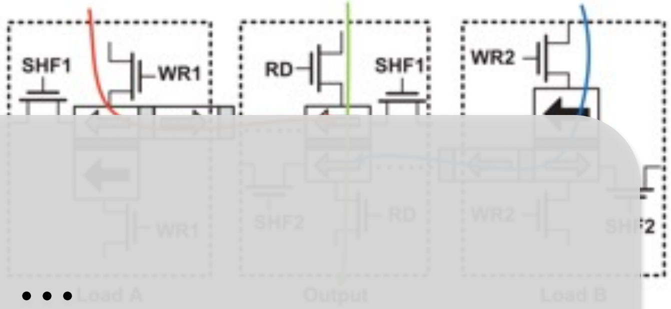
- ❑ Compute in-place, avoid data movement, transformations to match primitives

In-PCM Computing



Joshi, Vinay, et al. "Accurate deep neural network inference using computational phase-change memory." Nature Communications 11.1 (2020): 1-13.

In-RTM Computing



...nt Nonvolatile In-Memory Computing Architecture for Extreme Learning ... wire Devices," in IEEE Transactions on Nanotechnology, 2015.

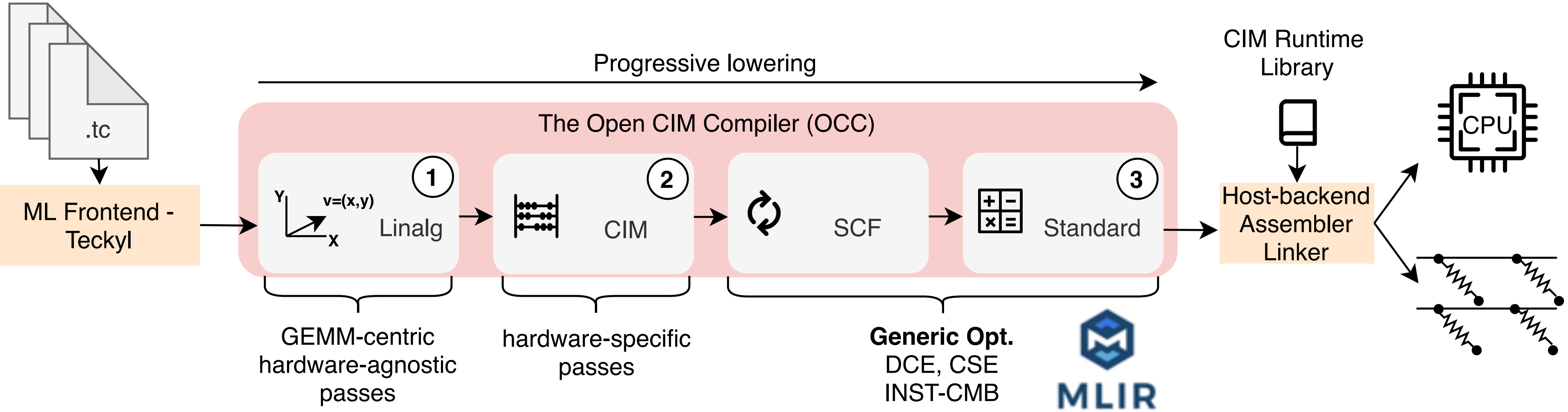
What are primitives in in-memory computing?

- ❑ In-PCM **dot-products, non-linear functions, …**
- ❑ In-RTM bulk logic and **majority operations, efficient counting, …**
- ❑ Others: (approx.) **content-addressable** memories (FeFETs, …)

Parkin, Stuart SP, Masamitsu Hayashi, and Luc Thomas. "Magnetic domain-wall racetrack memory." Science 320.5873 (2008): 190-194.

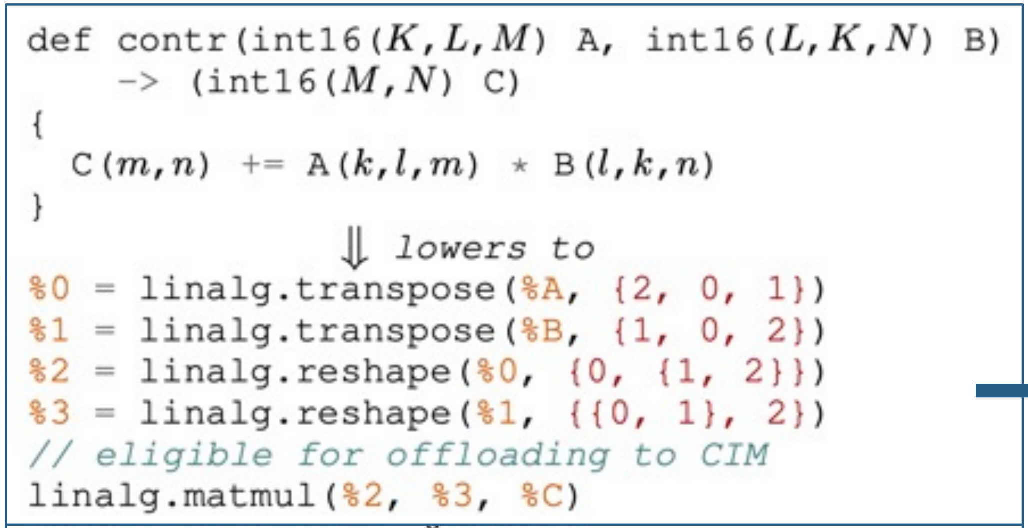
End-to-end compiler for PCM acceleration

- MLIR frontend for general tensor expressions
  - Reuse GEMM transformations from **linalg** (in MLIR)
  - High-level transformations and lowering to **CIM dialect**

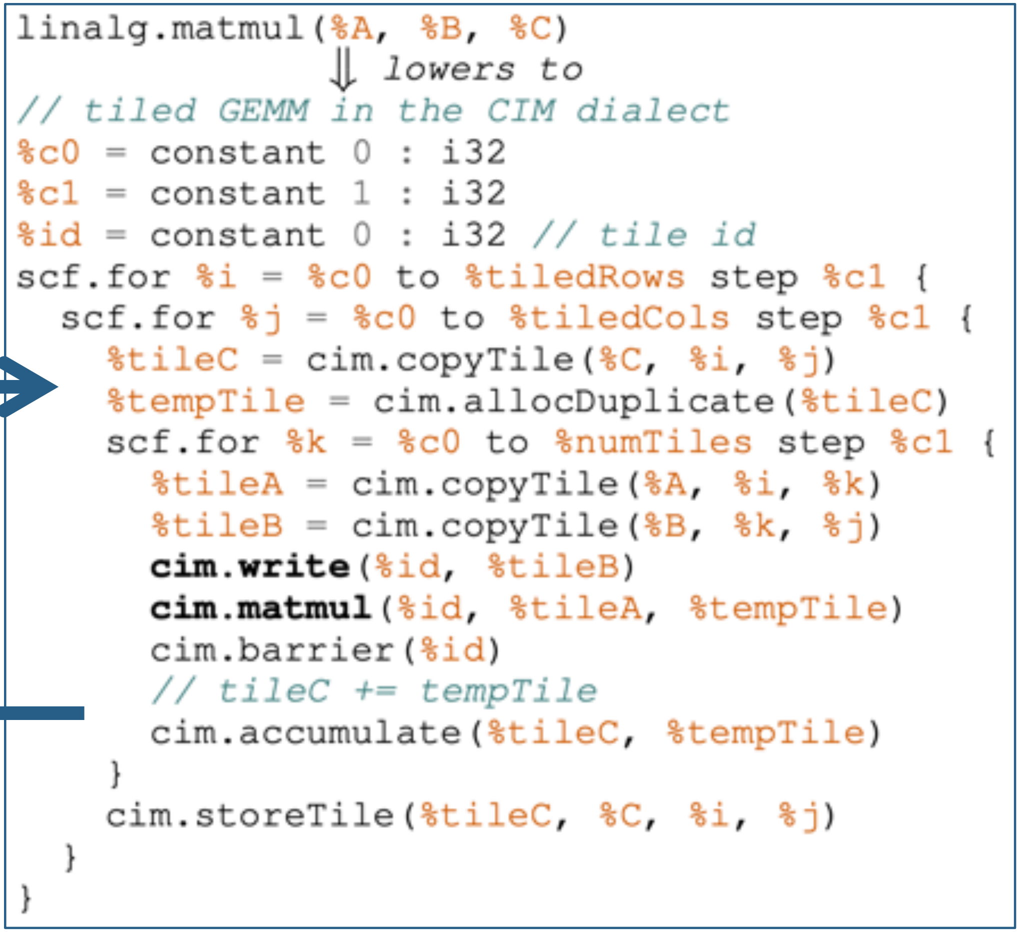


A. Siemieniuk, L. Chelini, A. A. Khan, J. Castrillon, A. Drebes, H. Corporaal, T. Grosser, M. Kong, "OCC: An Automated End-to-End Machine Learning Optimizing Compiler for Computing-In-Memory", In IEEE TCAD 2021

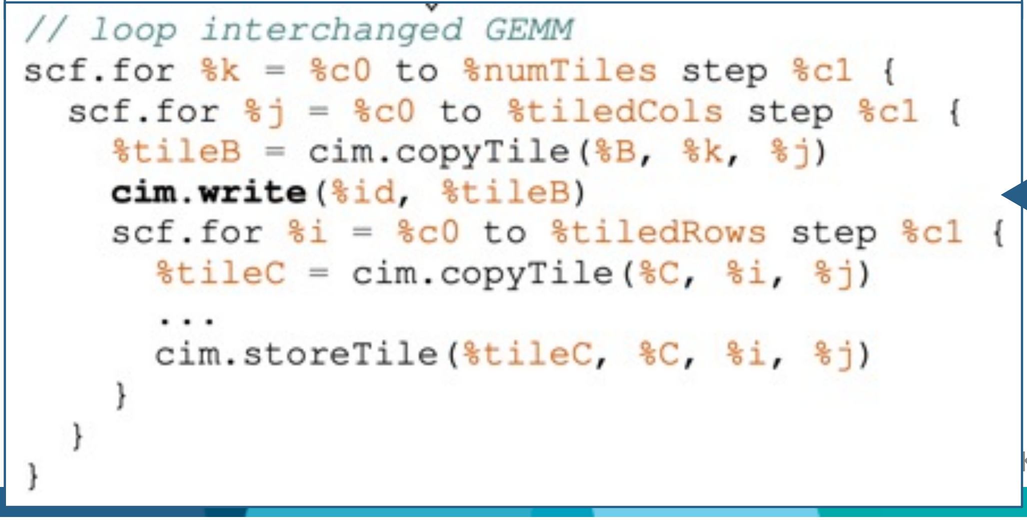
Lowering examples



```
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)
```

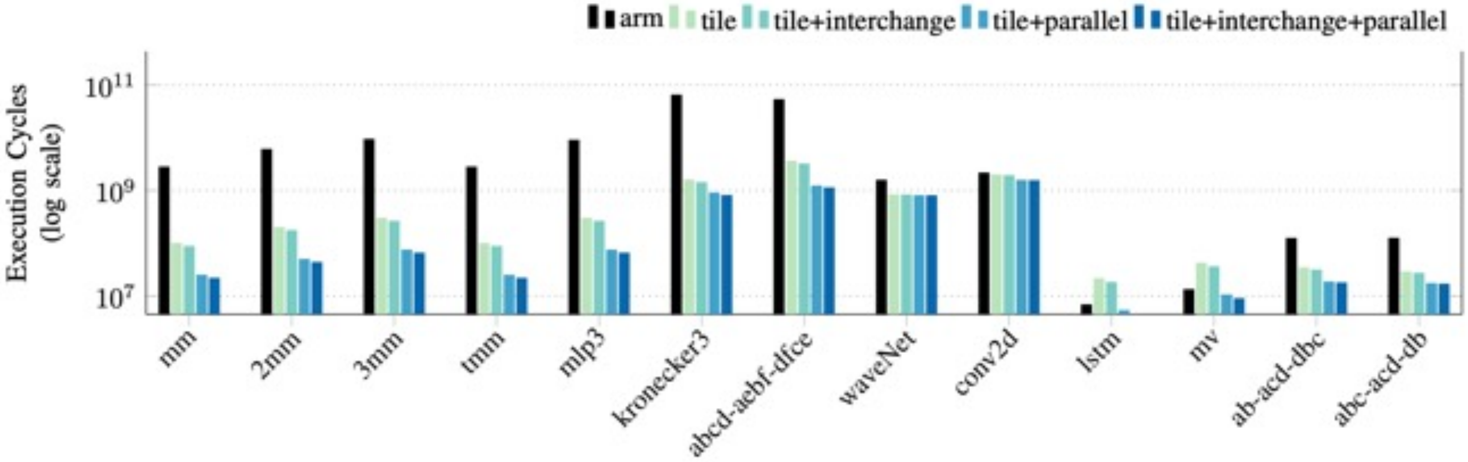


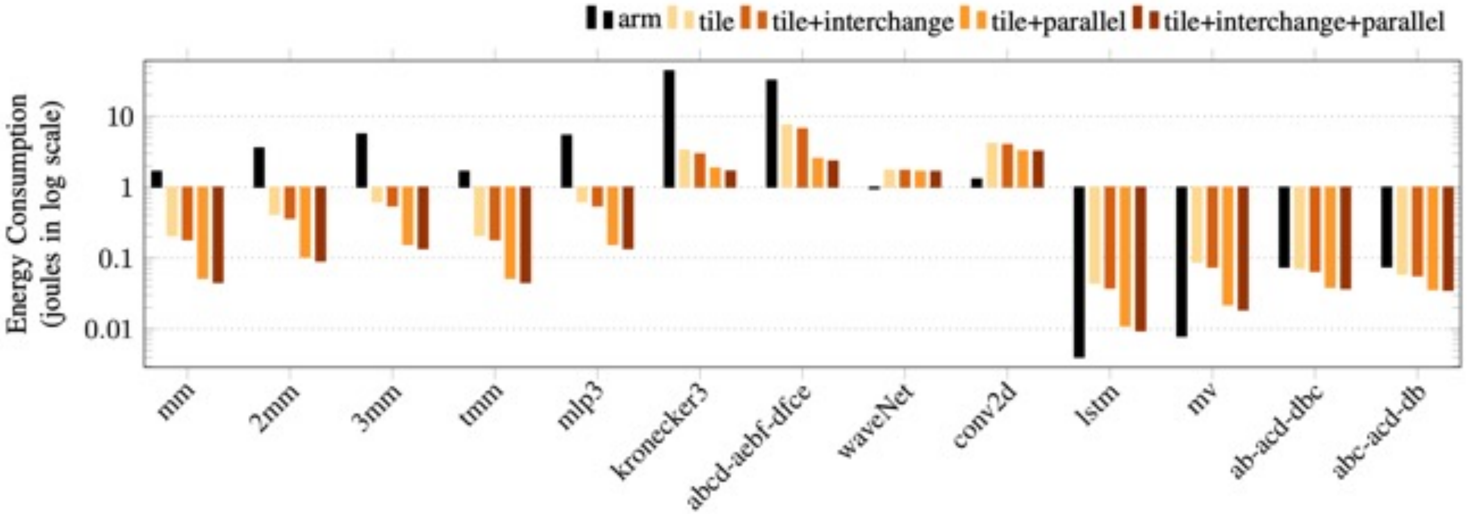
```
linalg.matmul(%A, %B, %C)
              ⇓ lowers to
// tiled GEMM in the CIM dialect
%c0 = constant 0 : i32
%c1 = constant 1 : i32
%id = constant 0 : i32 // tile id
scf.for %i = %c0 to %tiledRows step %c1 {
  scf.for %j = %c0 to %tiledCols step %c1 {
    %tileC = cim.copyTile(%C, %i, %j)
    %tempTile = 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)
  }
}
```



```
// 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)
    }
  }
}
```

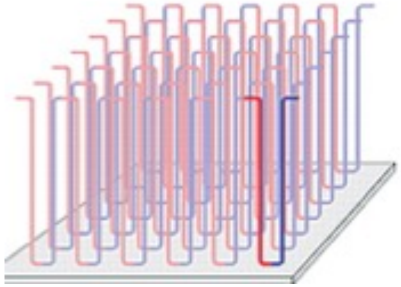
Optimization results: Beyond Matmult

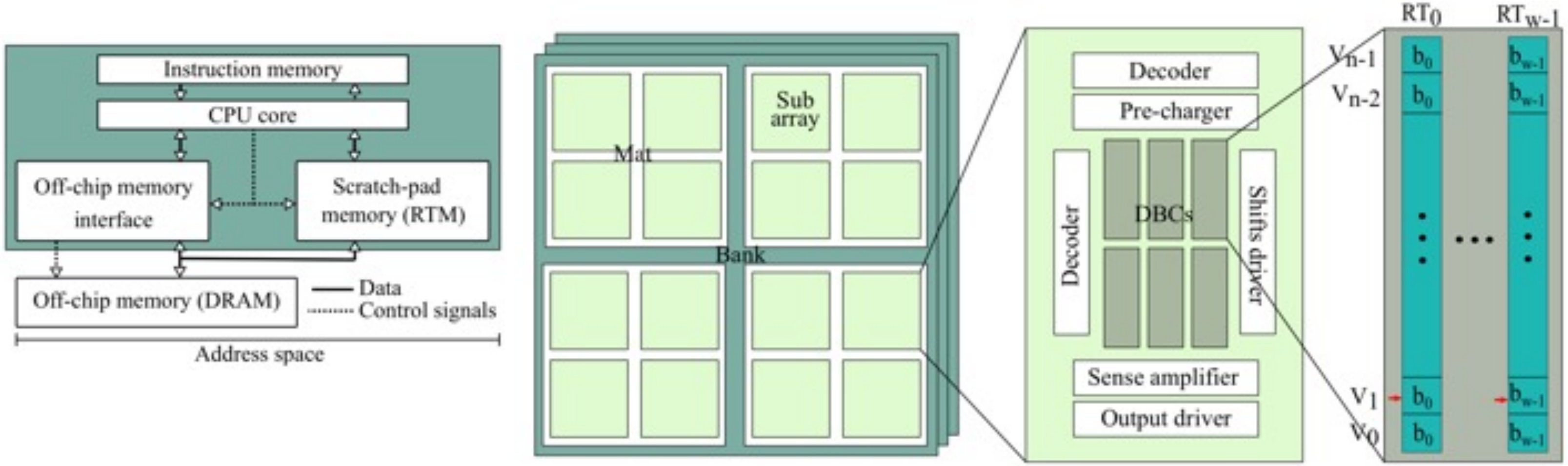




Racetrack memories and shifts

- ❑ Latency highly depends on allocation and address traces
- ❑ System-level simulators (interoperable w/ e.g. gem5)
- ❑ Compiler optimizations for scalars, arrays and instructions

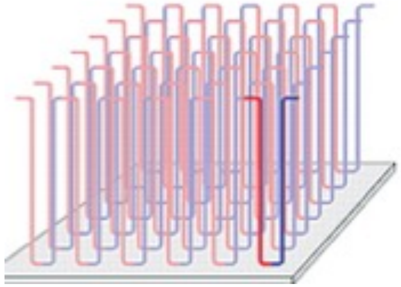


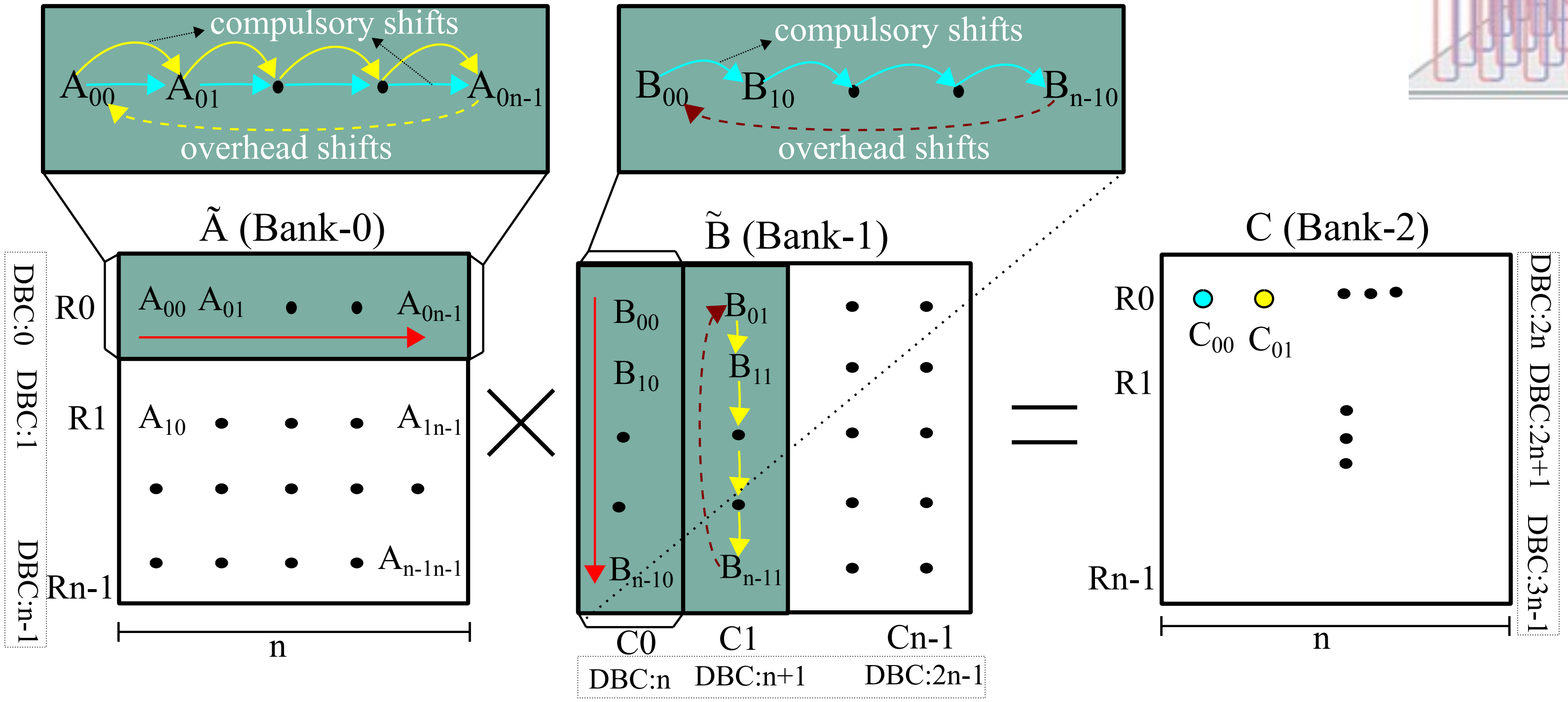


Khan, et al. "RTSim: A Cycle-accurate Simulator for Racetrack Memories", In IEEE Computer Architecture Letters, 2019
Khan, et al. "Generalized Data Placement Strategies for Racetrack Memories", DATE 2020
Khan, et al. "ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0", ACM TACO 2019
Multanen, et al. "Energy-Efficient Instruction Delivery in Embedded Systems with Domain Wall Memory", IEEE ToC 2021

Tensor expressions on RTMs

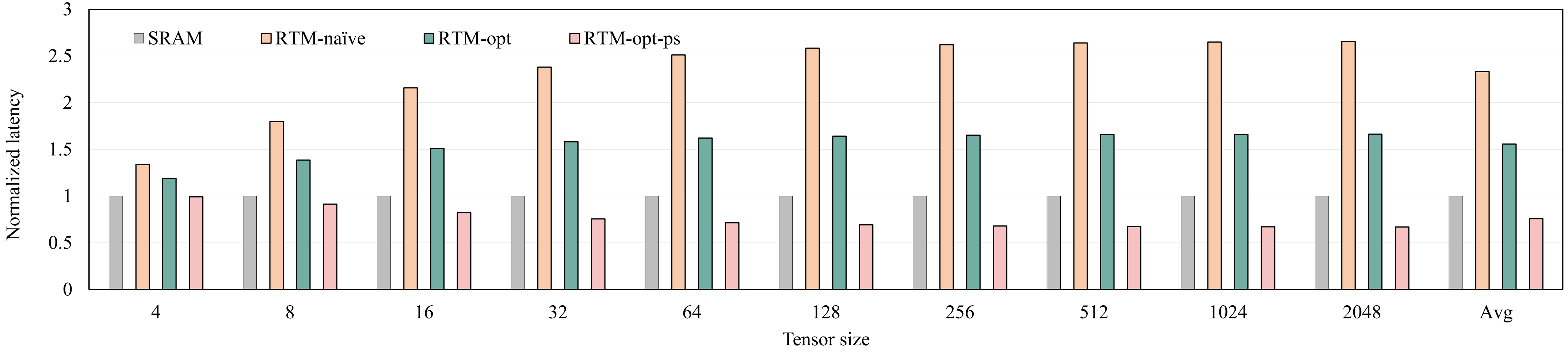
- ❑ Consecutive accesses can be pre-shifted
- ❑ Avoid “rewinding the tape”





Latency comparison vs SRAM

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

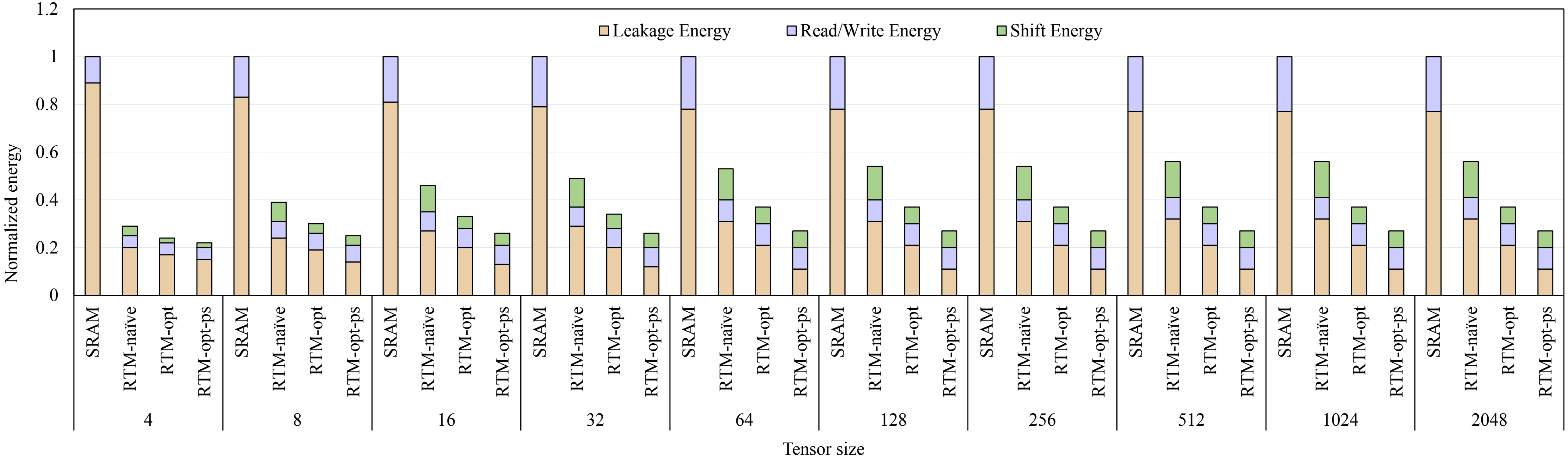


A. A. Khan, et al, "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", LCTES'19, pp. 5-18, 2019

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% average improvement (in addition to savings due to DRAM placement)

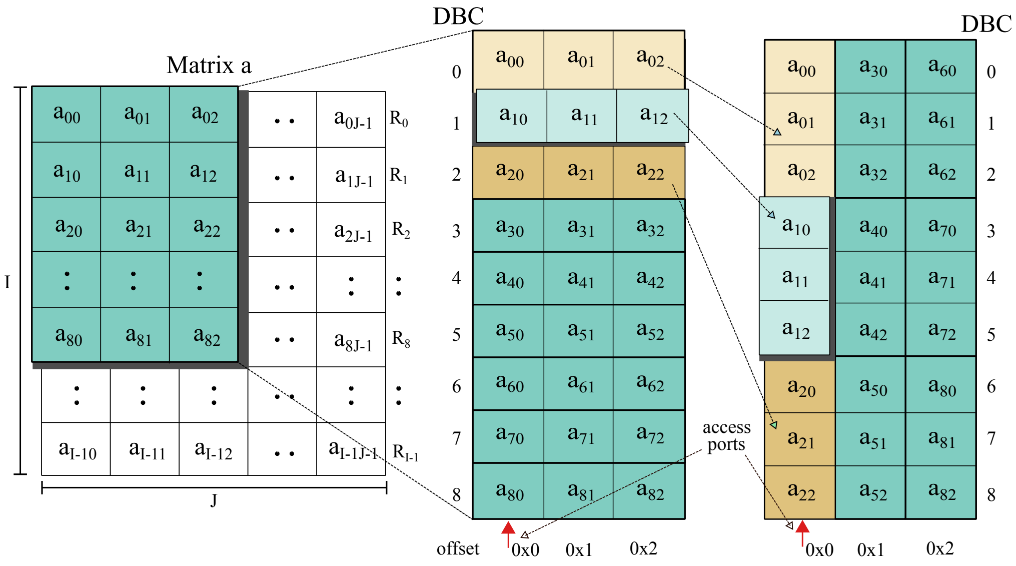


A. A. Khan, et al, "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", LCTES'19, pp. 5-18, 2019

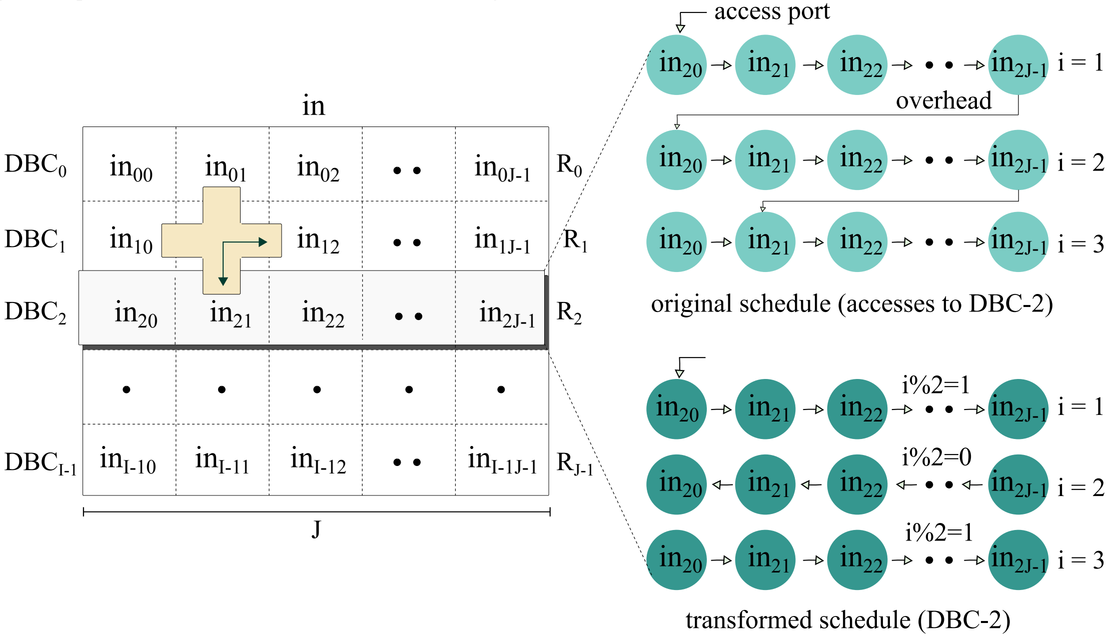
A. A. Khan, et al. "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories". ACM TECS 2020

Generalization: RTM optimizing compiler

- ❑ Jointly optimize layout and operation scheduling
- ❑ Interplay with other (polyhedral) loop optimizations: Tiling, fusion, …



Matrix a
DBC
I
J
a_{00} a_{01} a_{02} • • a_{0J-1} R_0
a_{10} a_{11} a_{12} • • a_{1J-1} R_1
a_{20} a_{21} a_{22} • • a_{2J-1} R_2
a_{80} a_{81} a_{82} • • a_{8J-1} R_8
a_{I-10} a_{I-11} a_{I-12} • • a_{I-1J-1} R_{I-1}
0 1 2 3 4 5 6 7 8
access ports
offset
0x0 0x1 0x2

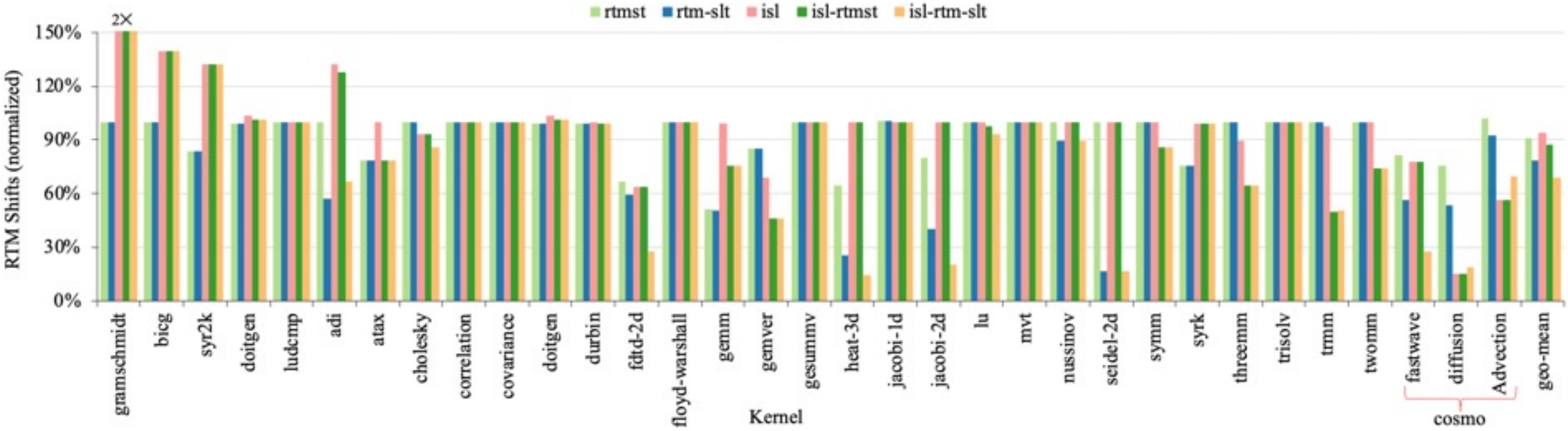


in
DBC_0 in_{00} in_{01} in_{02} • • in_{0J-1} R_0
DBC_1 in_{10} in_{12} • • in_{1J-1} R_1
DBC_2 in_{20} in_{21} in_{22} • • in_{2J-1} R_2
DBC_{I-1} in_{I-10} in_{I-11} in_{I-12} • • in_{I-1J-1} R_{J-1}
J
access port
overhead
i = 1
i = 2
i = 3
original schedule (accesses to DBC-2)
i%2=1
i%2=0
i%2=1
transformed schedule (DBC-2)

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

Example: Optimizations for strencils for RTM

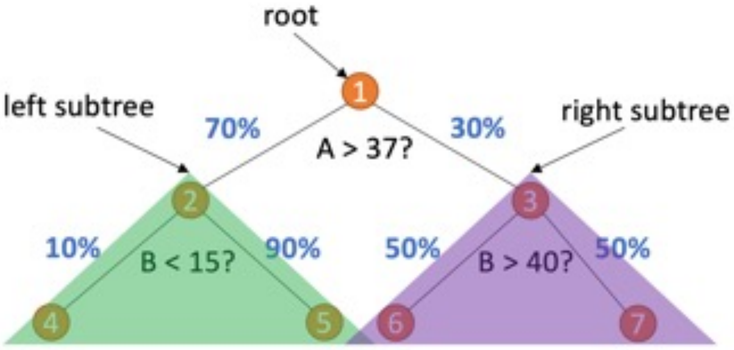
- ❑ Average improvements in performance (~20%) and energy consumption (~40%)

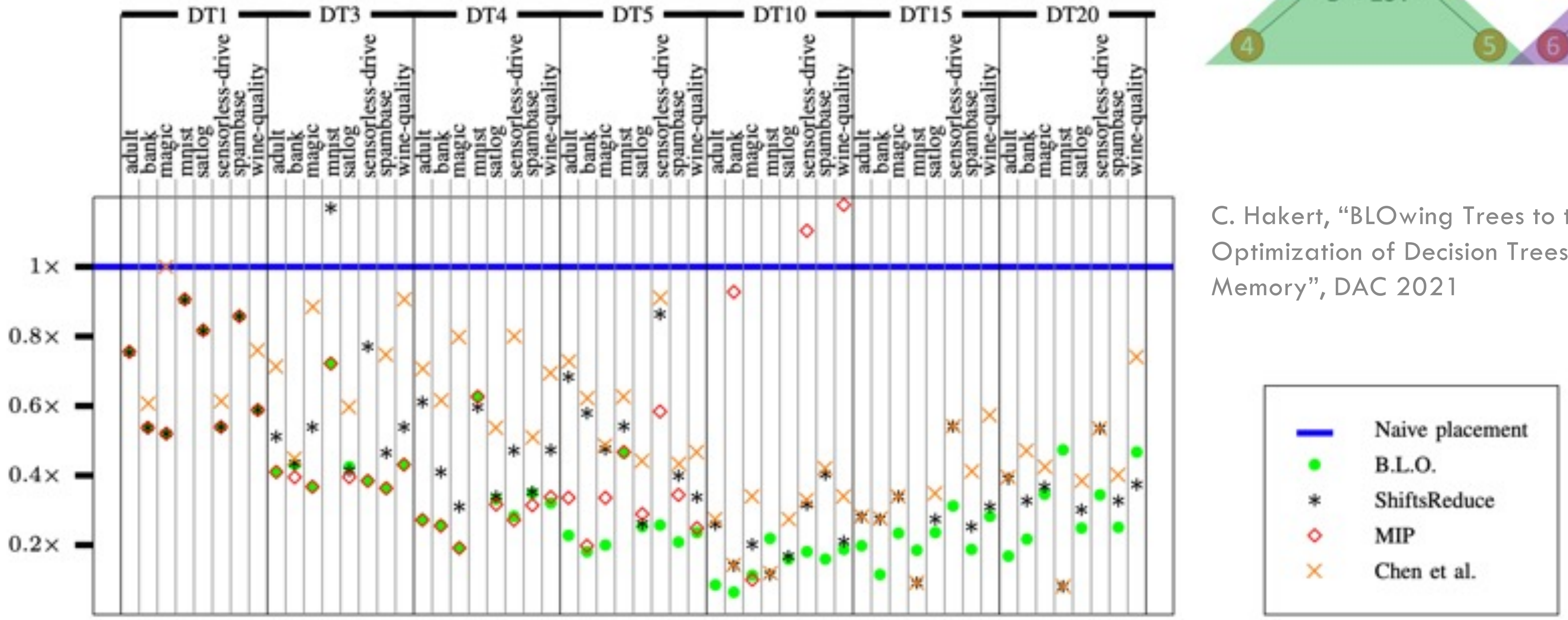


Khan, et al. "Polyhedral Compilation for Racetrack Memories". IEEE TCAD 2020

Random forests

- ❑ Popular way for decision making @ edge
- ❑ Example of less-predictable access
- ❑ Use training statistics for tree placement

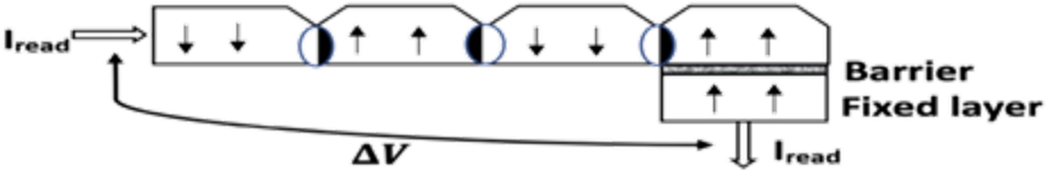




C. Hakert, “BLOwing Trees to the Ground: Layout Optimization of Decision Trees on Racetrack Memory”, DAC 2021

In-RTM computing

- ❑ Transverse reads
  - ❑ Pass current through nanowire (not for shifting)
  - ❑ Sensed resistance correlates with the amount of ones (group XOR)

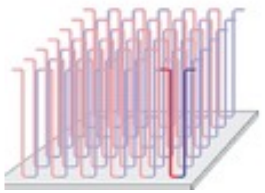


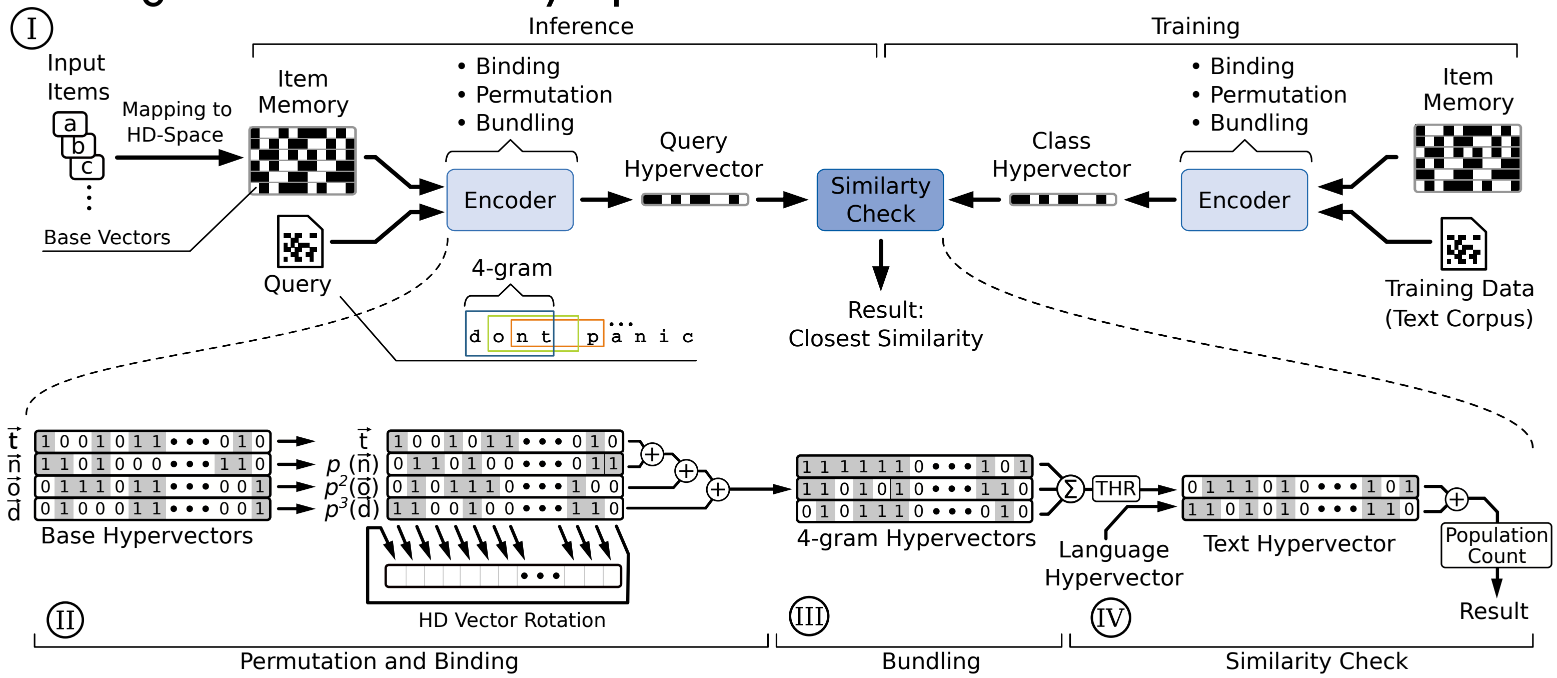
K. Roxy, IEEE T Nano 2020

- ❑ Studying how to generalize from this through hand-crafted designs…

Example: Hyper dimensional computing (HDC)

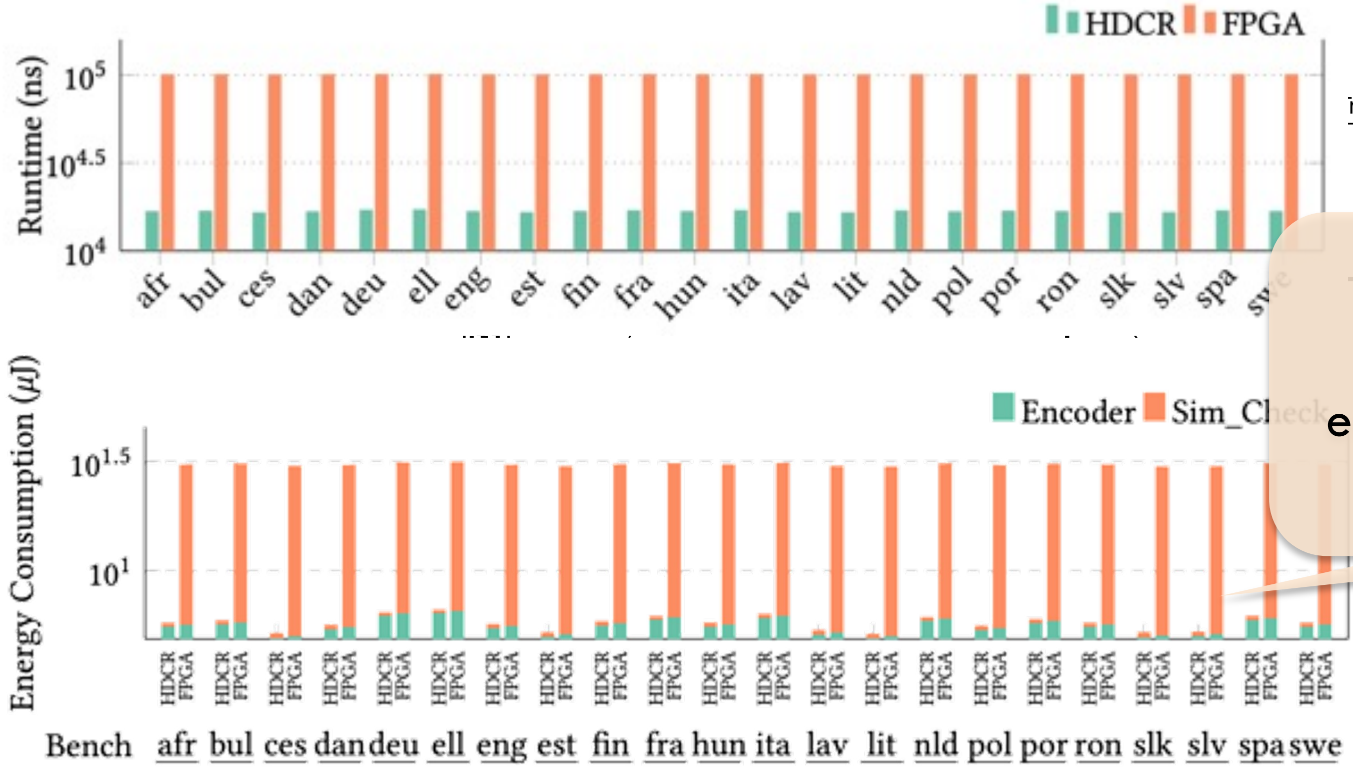
- HDC: Embed data in 10 k-dimensions – Von-Neuman Bottleneck!
  - Leverage bulk-wise binary operations

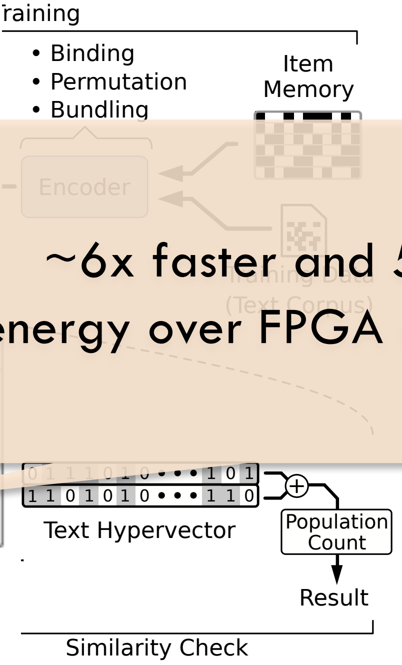




Khan, A. A., Ollivier, S., Longofono, S., Hempel, G., Castrillon, J., & Jones, A. K. (2022). Brain-inspired Cognition in Next Generation Racetrack Memories. In ACM TECS 2022

Example: Hyper dimensional computing (HDC)





~6x faster and 5.3x less energy over FPGA accelerator

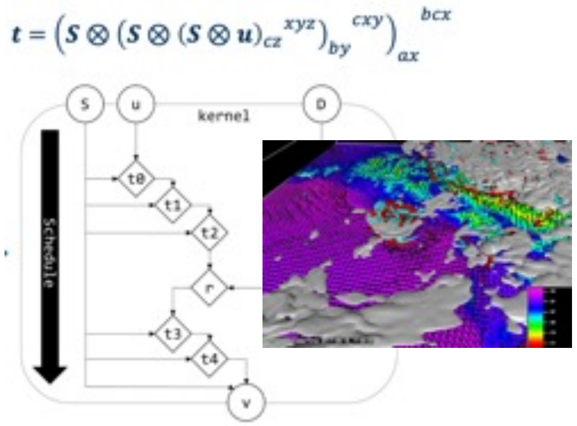
Khan, A. A., Ollivier, S., Longofono, S., Hempel, G., Castrillon, J., & Jones, A. K. (2022). “Brain-inspired Cognition in Next Generation Racetrack Memories” In ACM TECS 2022

Summary

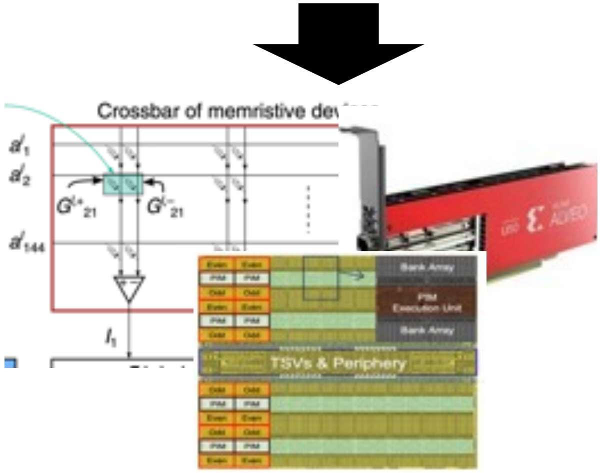
- ❑ Tame ever-increasing system complexity
  - ❑ Domain-specific abstractions, compilation flows, …
  - ❑ Example for tensor expressions
  - ❑ Optimization for CPU, in-memory, RTM placement, …

- ❑ 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!

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



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



Thanks! & Acknowledgements

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Hasna Bouraoui

João Cardoso

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**)

Funded by

DFG Deutsche Forschungsgemeinschaft

CO4RTM (Number 450944241)

EVEREST https://everest-h2020.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957269

STAATSMINISTERIUM FÜR WISSENSCHAFT KULTUR UND TOURISMUS | Freistaat SACHSEN

References

[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. "TeIL: 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

[Arxiv'22] S. Soldavini, K. F. A. Friebel, et al. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics". In: ArXiv, arXiv:2203.10850 (Mar. 2022)

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

[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

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

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

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

[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.

[DATE'20] Khan, et al. "Generalized Data Placement Strategies for Racetrack Memories", DATE 2020

[ToCS21] Multanen, et al. "Energy-Efficient Instruction Delivery in Embedded Systems with Domain Wall Memory", IEEE ToC 2021

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

[TECS'22] Khan, A. et al. "Brain-inspired Cognition in Next Generation Racetrack Memories" In ACM TECS 2022

[TMSCS'18] J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems, pp. 243-259, 2018.