Programming abstractions for in and near-memory computing

Jeronimo Castrillon

Chair for Compiler Construction (CCC)

TU Dresden, Germany

In-Memory Architectures and Computing Applications Workshop (iMACAW'23)
Design Automation Conference (DAC'23). San Francisco, USA
July 9, 2023

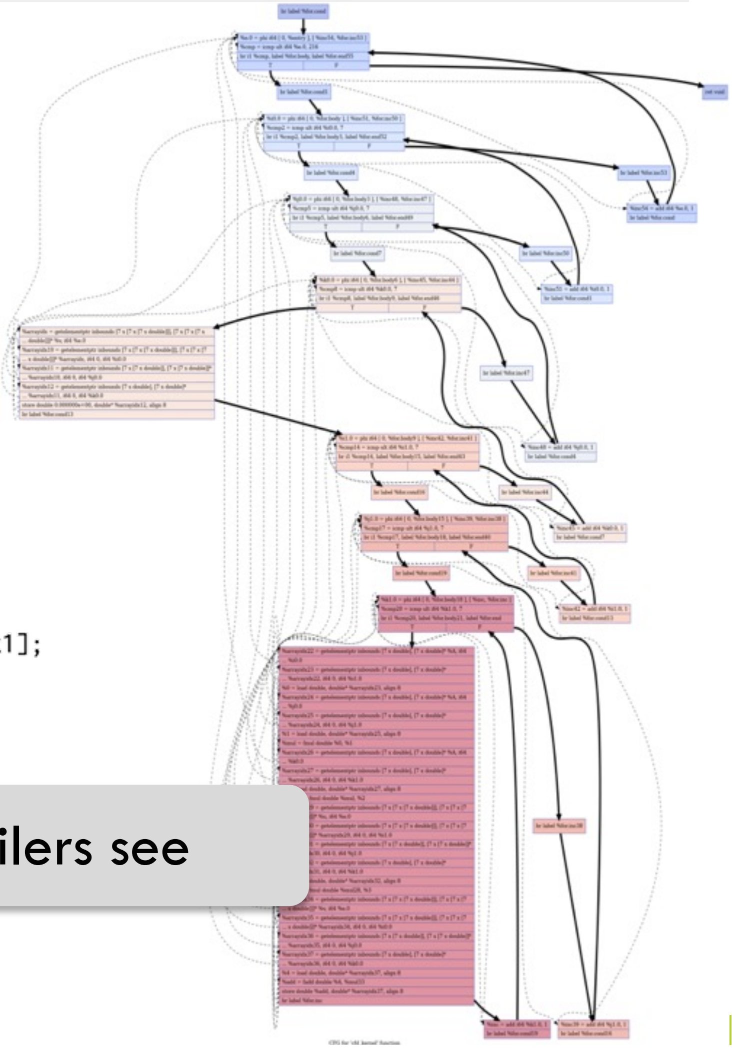
Why new abstractions

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

What we want

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 */
}
```



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

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 */
```

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]
                          ...][k1];
}
```

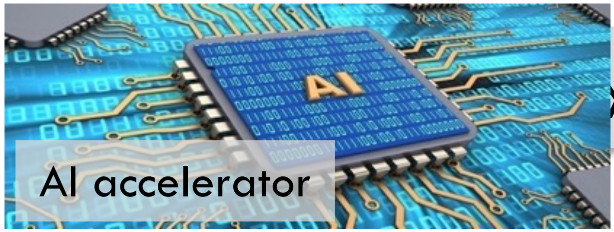
y) code

???

```
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

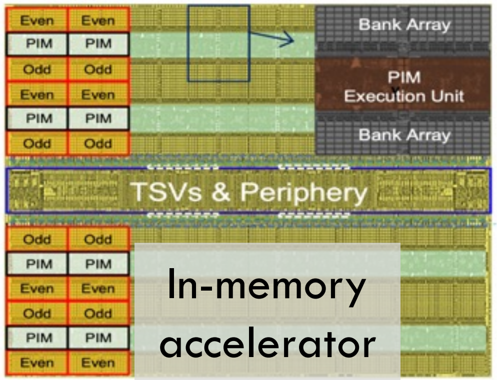
???



AI accelerator

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

???



In-memory accelerator

HBM-FPGA

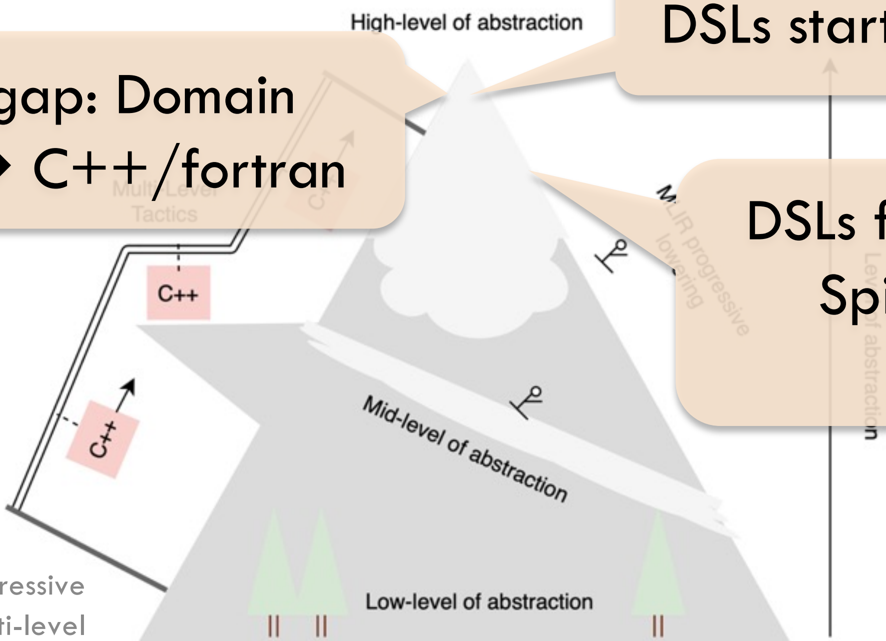
Lee, Sukhan, et al. "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product." ISCA 2021.

Wh

rts code

There is only su much we can do/reconstruct…

- Lots of progress: polyhderal compilers, trace-driven dynamic parallelization, patterns/ idiom extraction, …



Bridge gap: Domain experts → C++/fortran

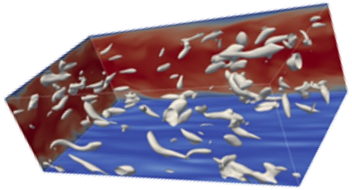
DSLs start here!

DSLs for performance: Halide, Spiral, TVM, TensorFlow, Firedrake…

L. Chelini, et al. "Progressive Raising in Multi-level IR." CGO 2021

Sample DSLs

- CFDlang

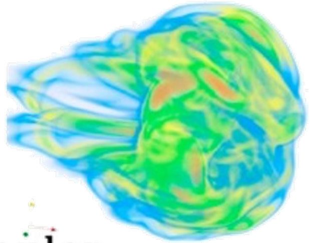


$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

```
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);
```

- OpenPME



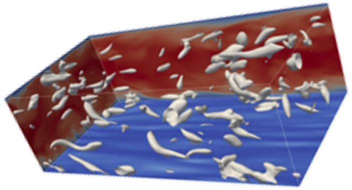
```
time loop
  start: 0  stop: 1000
  temporal method: explicit_euler
  spatial method: DC-PSE
```

$$\frac{\partial u}{\partial t} = \mathrm{Du} * \nabla^2 u - \mathrm{u} * v^2 + \mathrm{F} * (1 - \mathrm{u})$$

$$\frac{\partial v}{\partial t} = \mathrm{Dv} * \nabla^2 v + \mathrm{u} * v^2 - \mathrm{v} * (\mathrm{F} + \mathrm{k})$$

Sample DSLs

- CFDlang

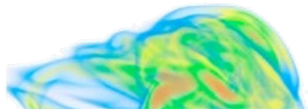


$$v_{ijk,e} = \sum_{i'=0}^{p}\sum_{j'=0}^{p}\sum_{k'=0}^{p} A_{kk'}A_{jj'}A_{ii'}u_{i'j'k'e}$$

```
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
```

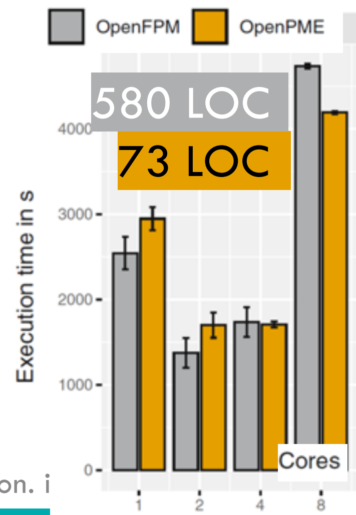
```
auto A = Matrix(m, n), B = Matrix(m, n),
     C = Matrix(m, n);
auto u = Tensor<3>(n, n, n);
auto v = (A*B*C)(u);
```

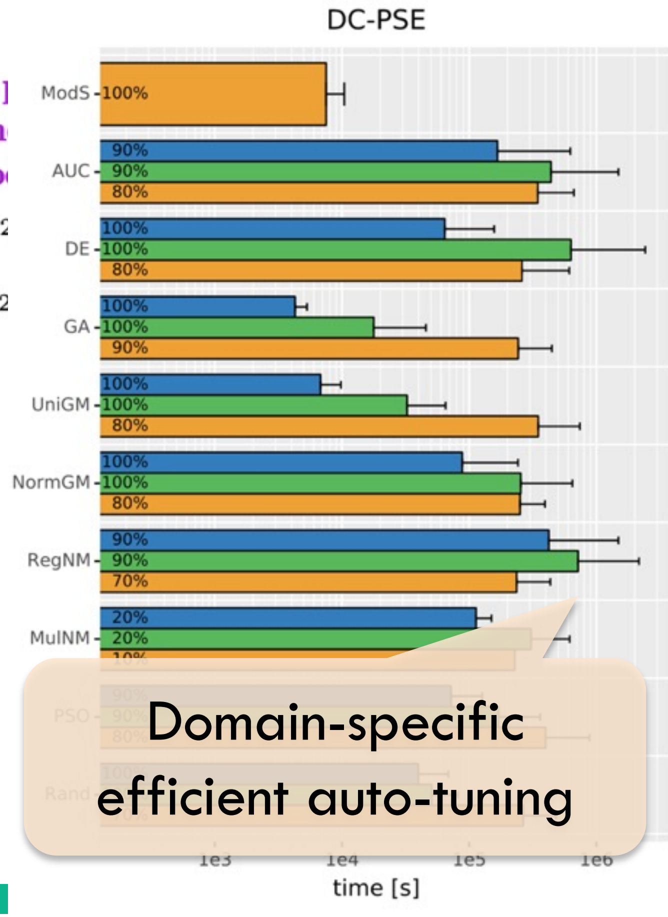
- OpenPME



```
time loop
 start: 0   sto
 temporal meth
 spatial metho
```

$$\frac{\partial u}{\partial t} = \mathrm{Du} * \nabla^2$$
$$\frac{\partial v}{\partial t} = \mathrm{Dv} * \nabla^2$$





Domain-specific efficient auto-tuning

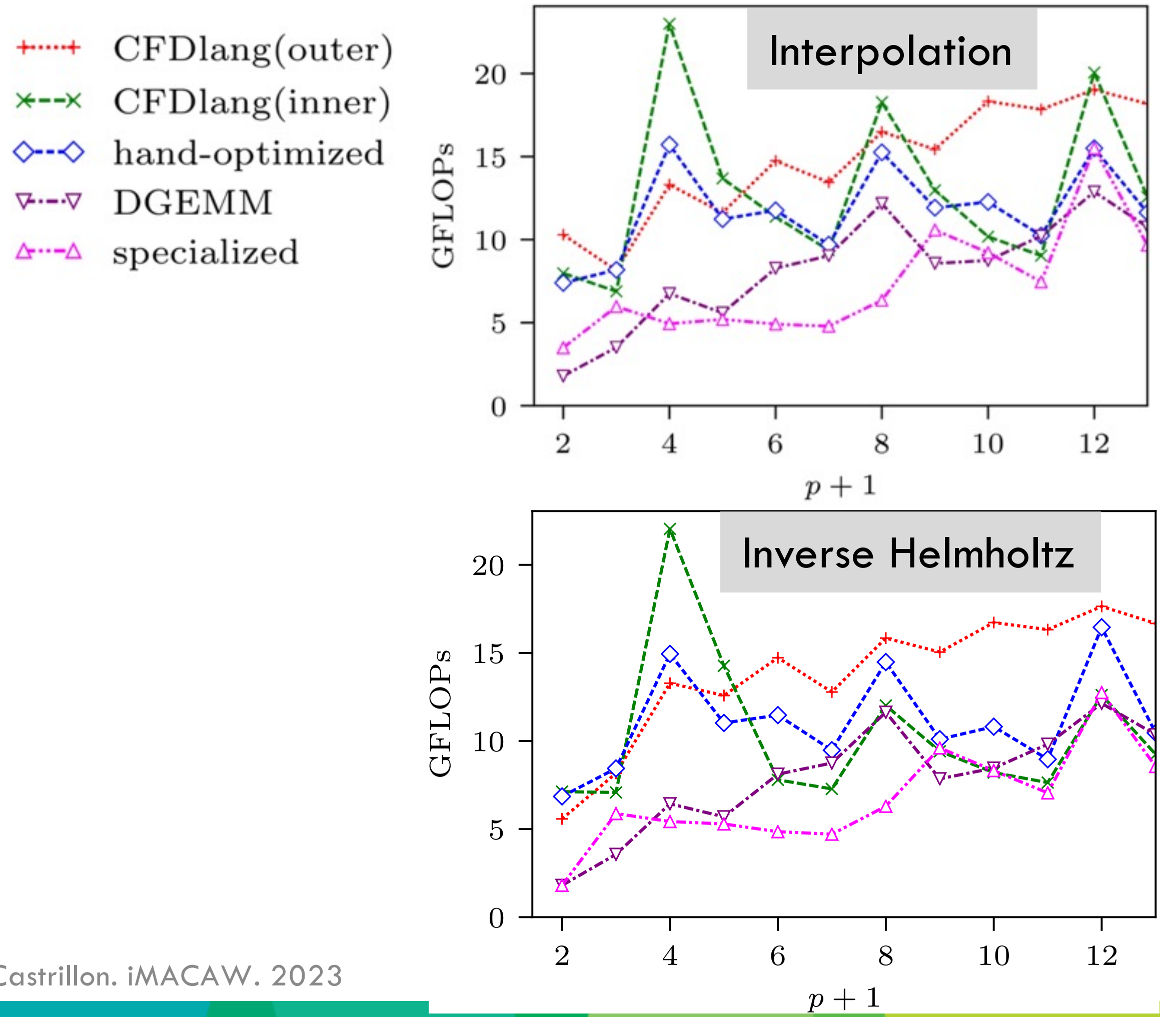
Closing the performance gap

- 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.

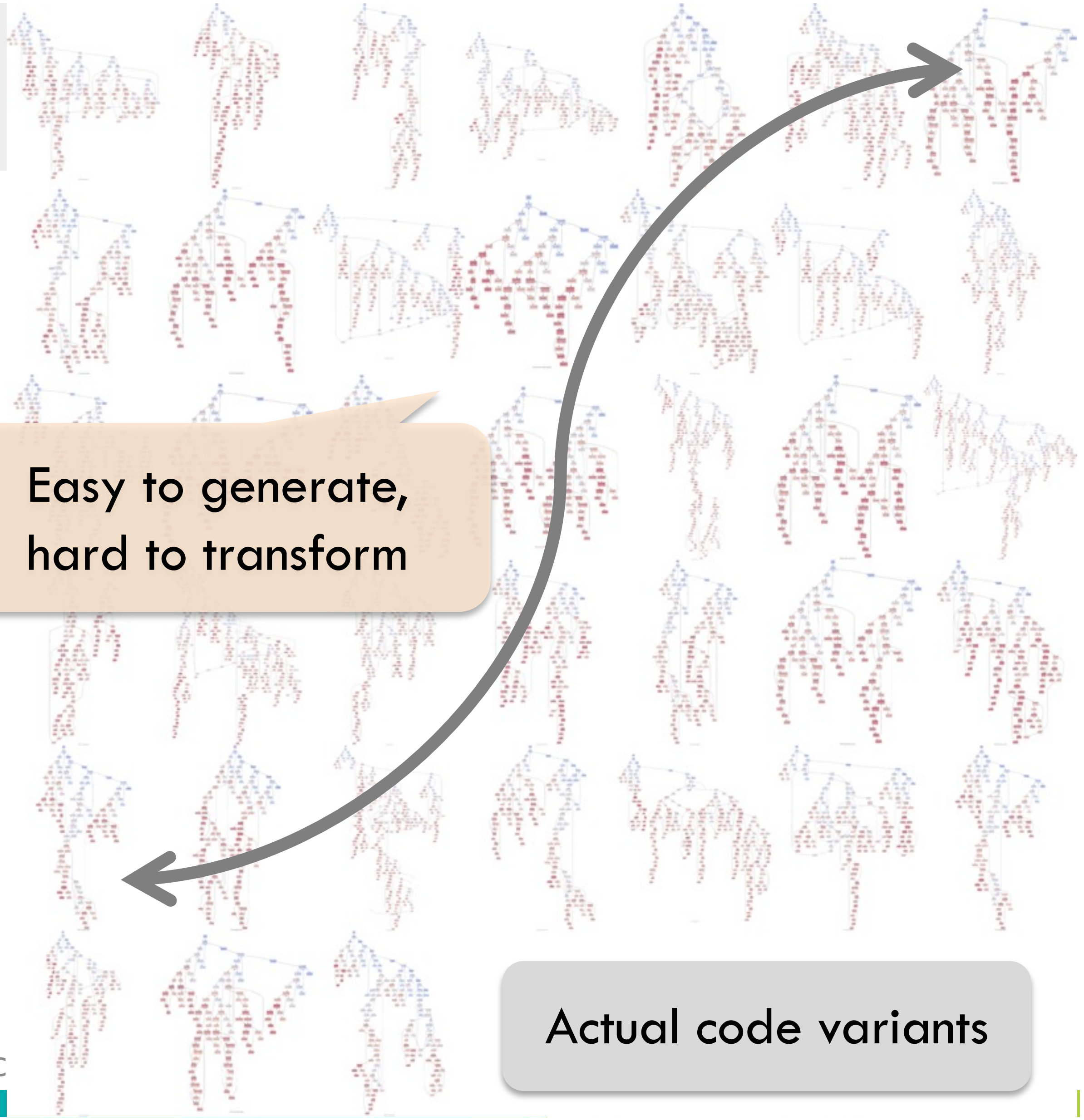
Closing the performance gap

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

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

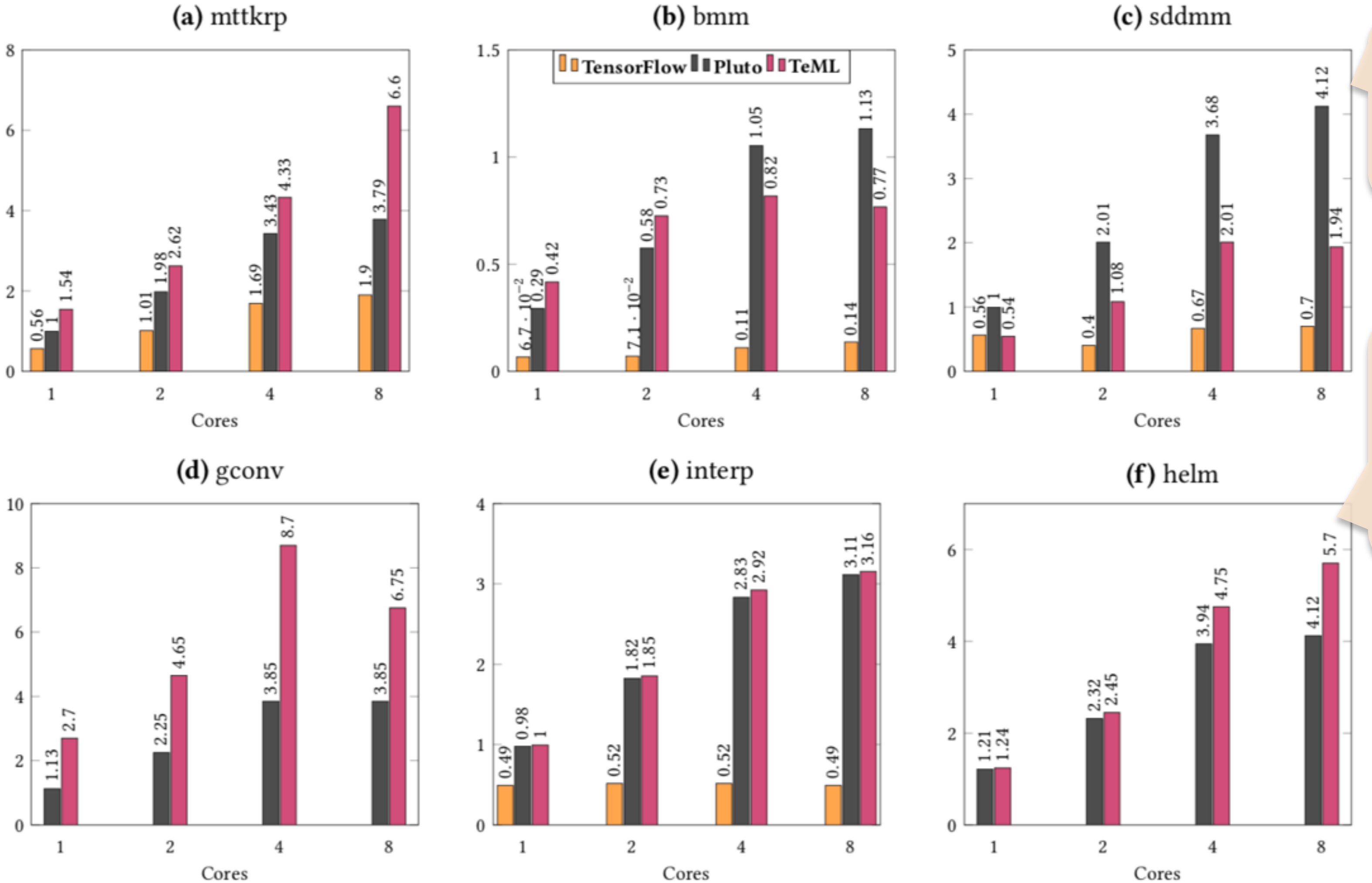
$$v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot A_{jm}) \cdot (A_{il} \cdot u_{lmn})$$

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



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.

Cross-domain optimizations



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: Tensor Intermediate language

- Formalized 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))
```

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

$$[\![\cdot]\!] : \mathit{Context} \to \mathit{Memory} \to (\mathit{list\ of}\ \mathsf{Nat}) \to \mathbb{D}$$

$$[\![x]\!]\,\Gamma\,\mu\,\bar{\imath} = \mu\,x\,\bar{\imath}$$

$$[\![(e)]\!]\,\Gamma\,\mu\,\bar{\imath} = [\![e]\!]\,\Gamma\,\mu\,\bar{\imath}$$

$$[\![\mathsf{add}\ e_0\ e_1]\!]\,\Gamma\,\mu\,\bar{\imath} = [\![e_0]\!]\,\Gamma\,\mu\,\bar{\imath} + [\![e_1]\!]\,\Gamma\,\mu\,\bar{\imath}$$

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

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

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

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

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

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

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

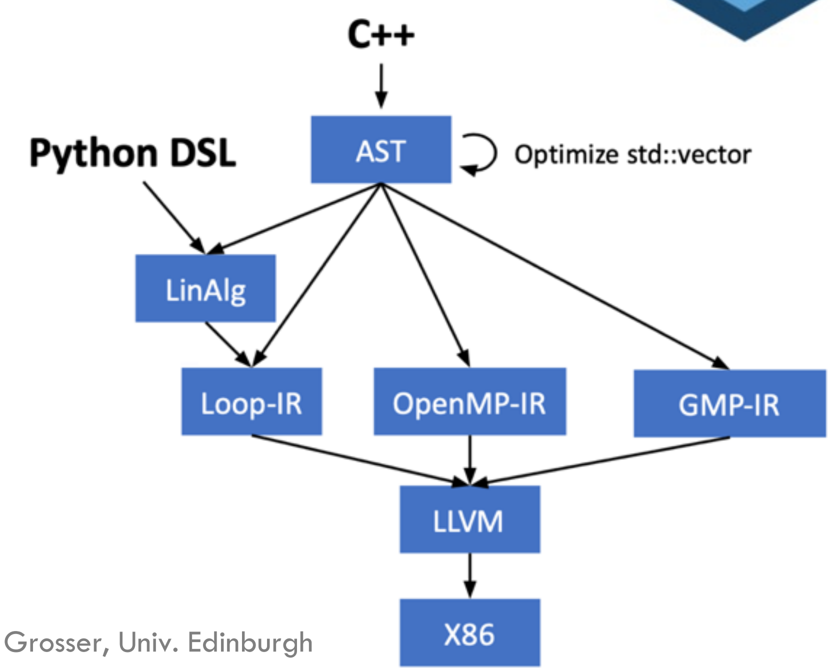
N.A. Rink, N. A. and J. Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ARRAY'19, pp. 57-68

Intermediate abstractions in MLIR

- 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.

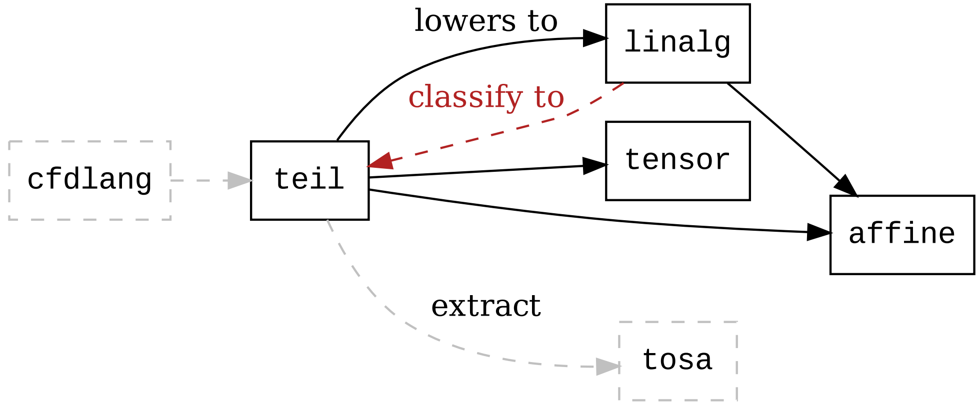
- Not an IR, but an extensible framework
  - to describe intermediate abstractions (called **dialects**),
  - to optimize representations between dialects (**transform, lower** or **raise**),
  - that builds on the success of LLVM to build community/infrastructure and reuse (“LLVM-quality” all the way)



Source: T. Grosser, Univ. Edinburgh

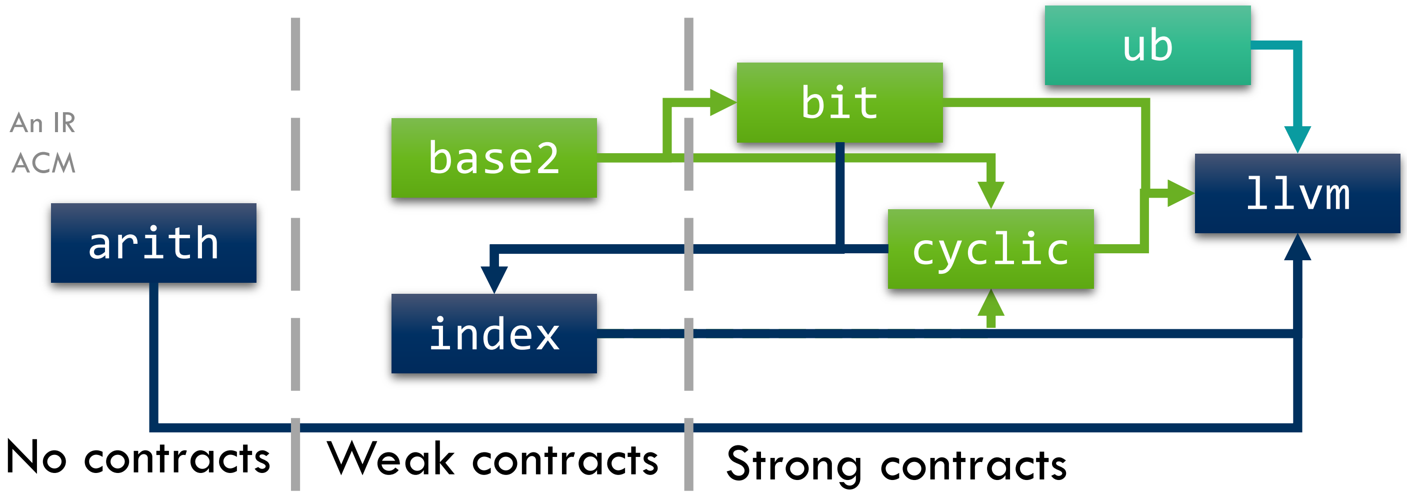
TeIL in MLIR

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



- Specialization path to custom hardware

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

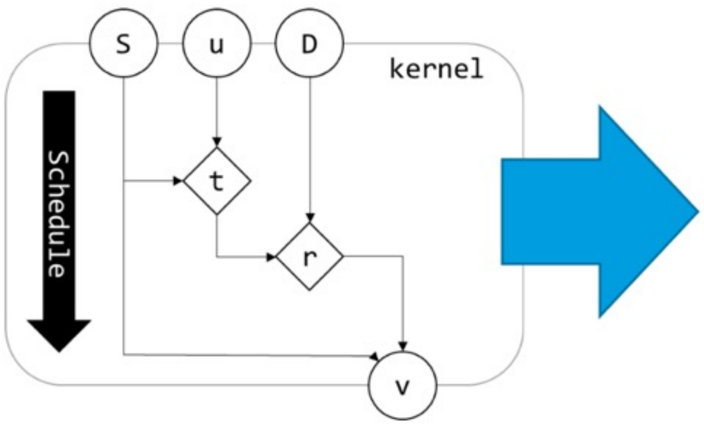


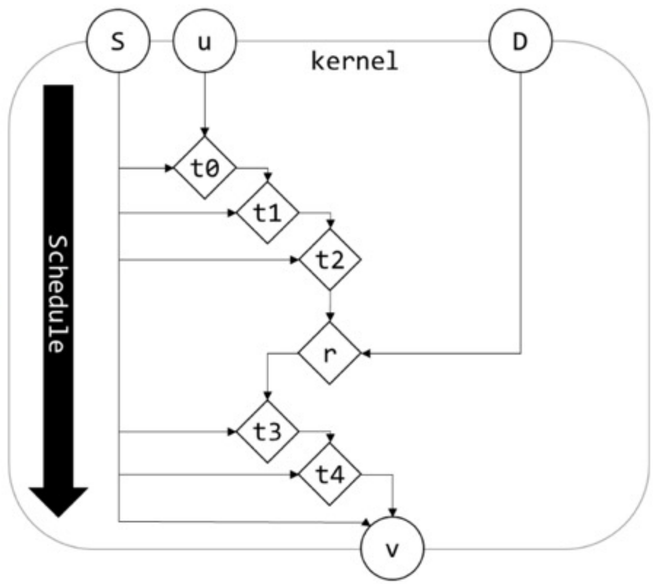
Domain-specific optimization

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

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

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





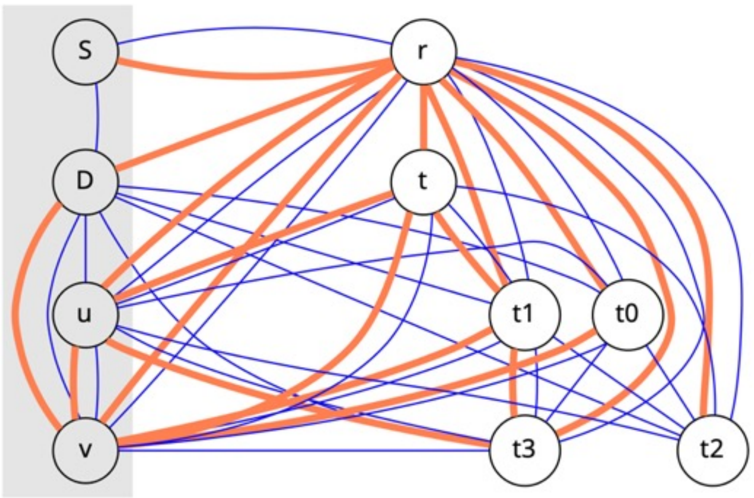
CHAIR FOR COMPILER CONSTRUCTION

High-level buffer re-use

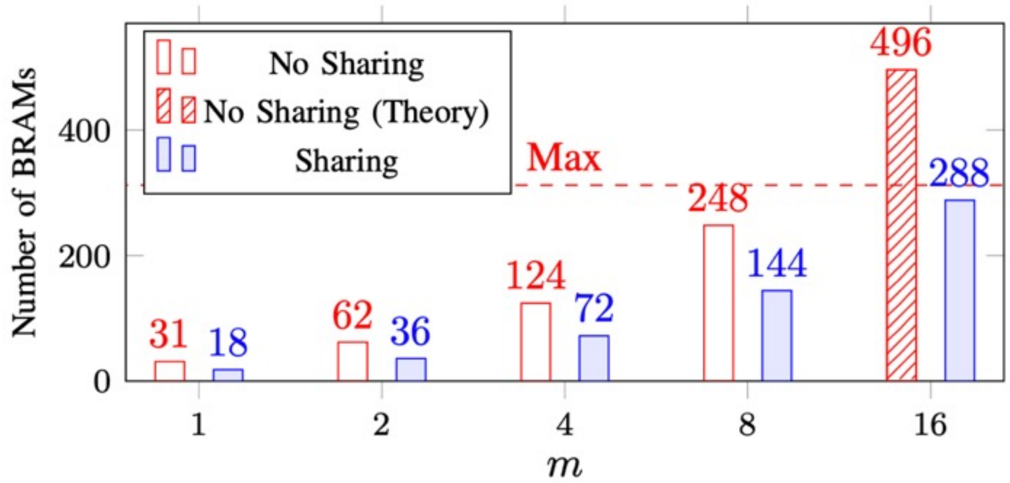
- Generate host code and accelerator code (for HLS)
- Generate liveness info (buffering, memory subsystem gen)

https://everest-h2020.eu

```
t = S # S # S # u . [[1 6] [3 7] [5 8]]
r = D * t
v = S # S # S # r . [[0 6] [2 7] [4 8]]
```



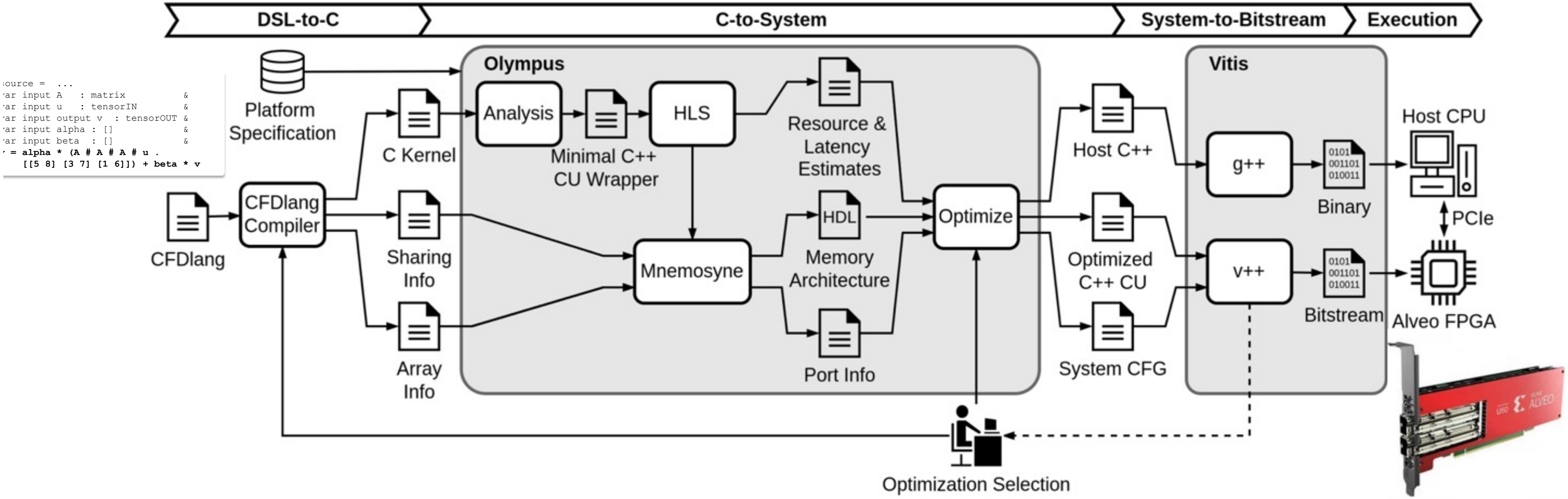
memory-interface and address-space compatibilities



K. F. A. Friebel, et al., "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

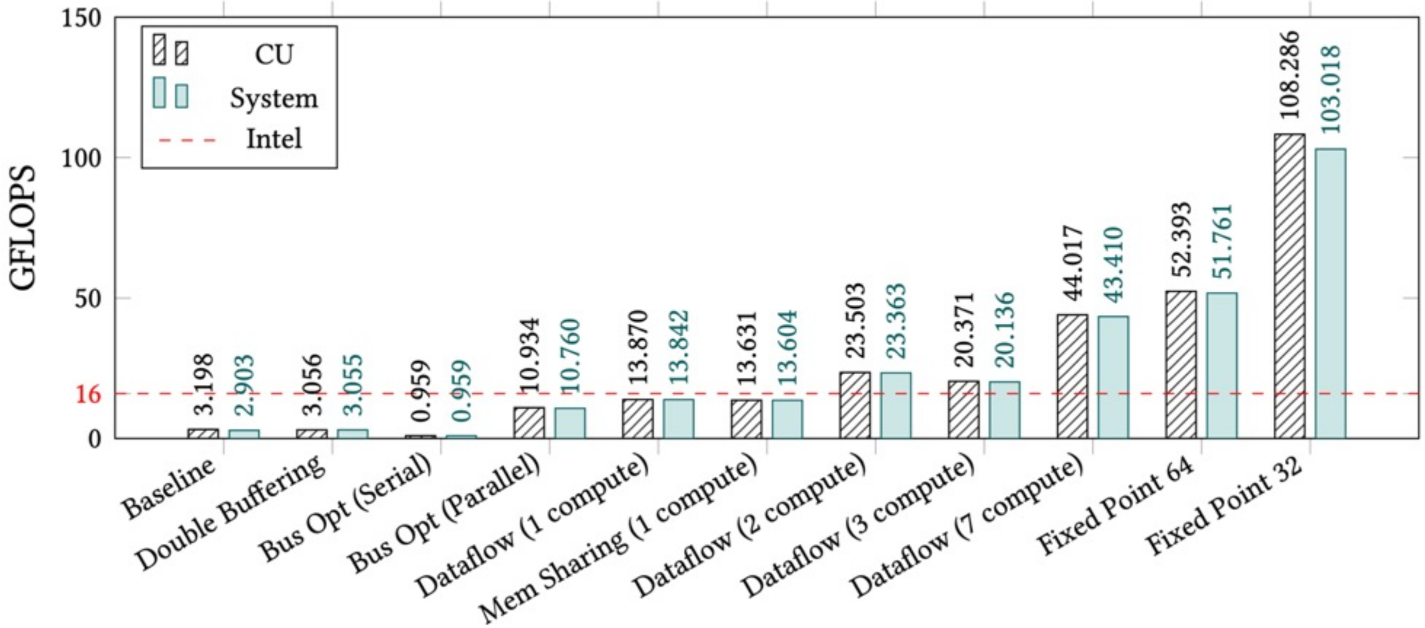
- Complex compilation/design flow from DSL to system-level architecture



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

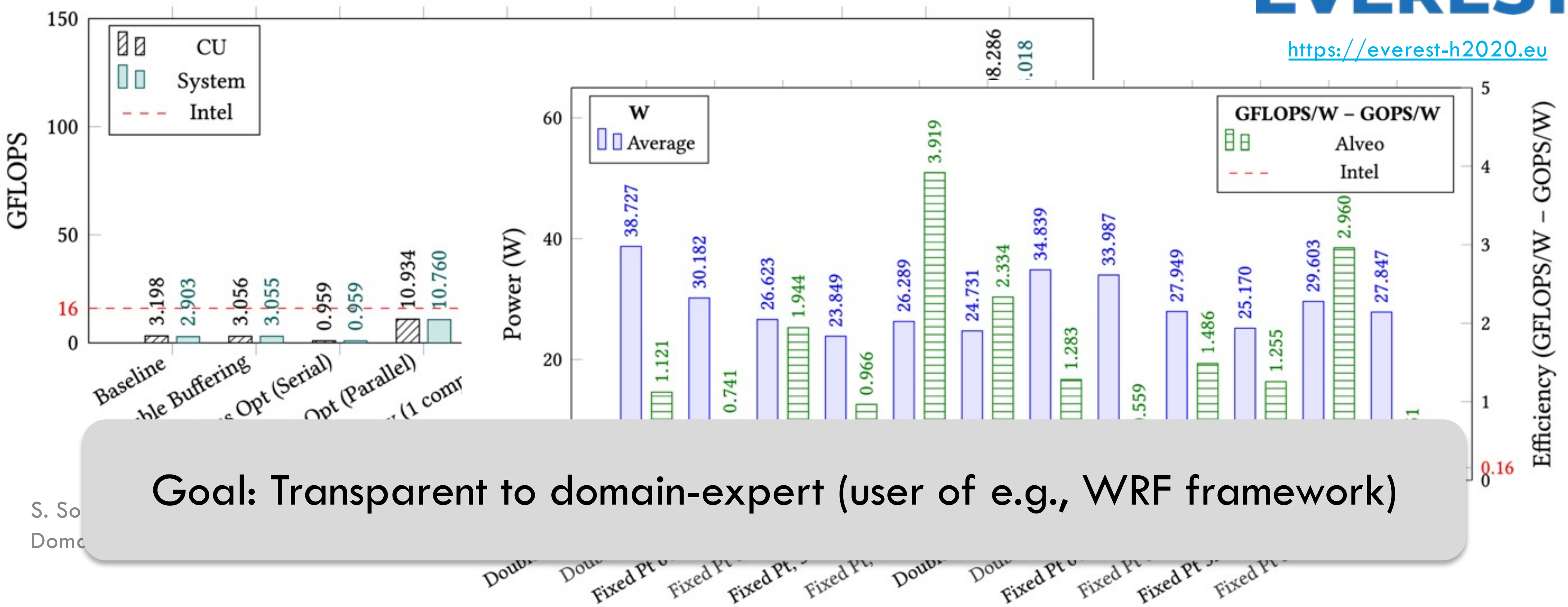
S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, and C. Pilato. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics". In: ACM TRETS, Sept. 2022.

FPGA code generation: HBM FPGA

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

EVEREST

https://everest-h2020.eu

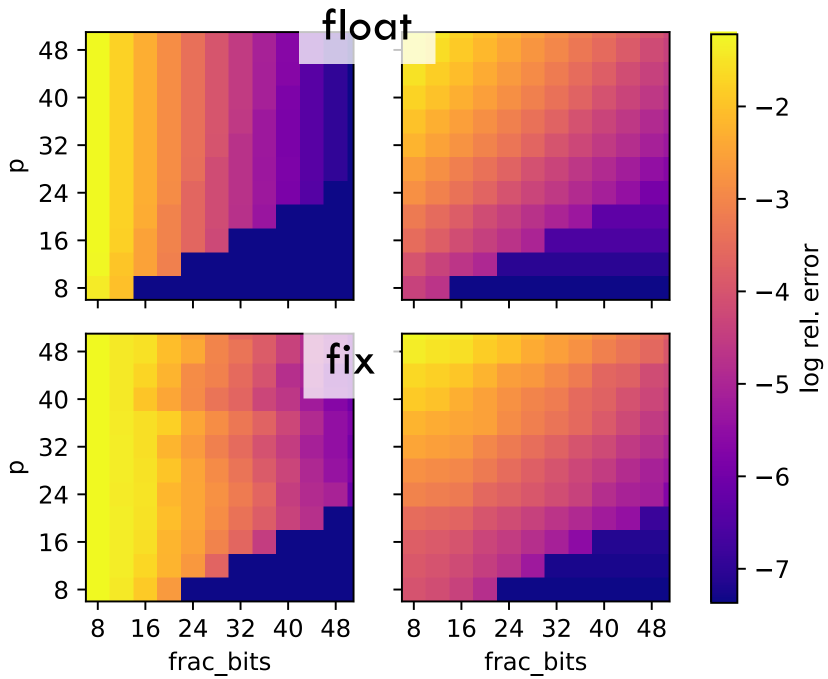


Goal: Transparent to domain-expert (user of e.g., WRF framework)

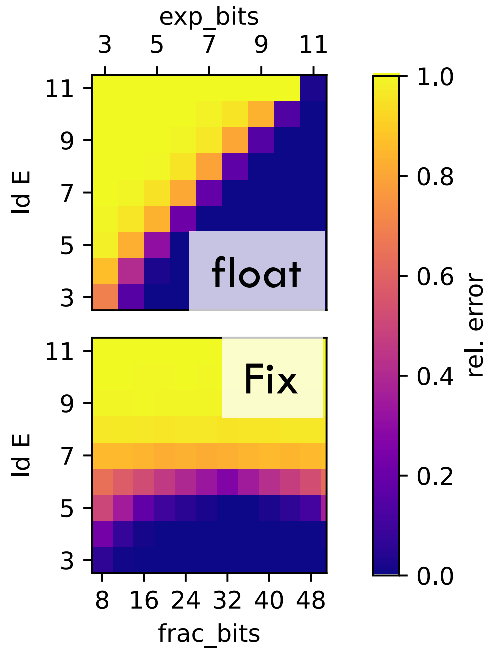
S. So
Domc

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)

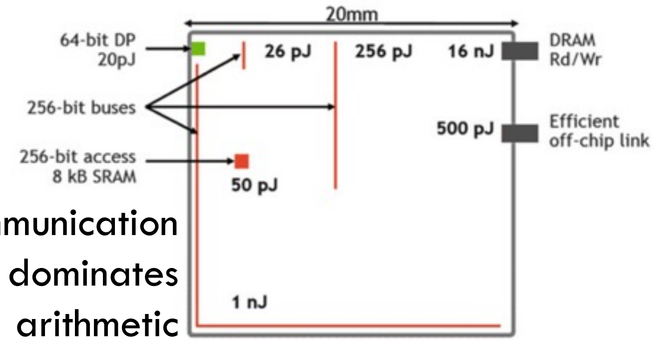


Exponent range ld E
(frac_bits = 32)

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

Emerging data-centric architectures

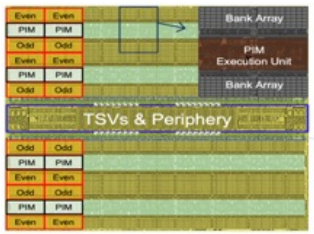
- Compute (almost) in-place, avoid data movement, transformations to match primitives
- Novel architectures for near-memory and in-memory computing

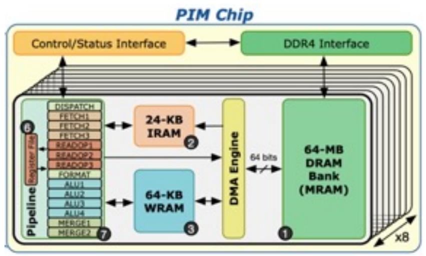


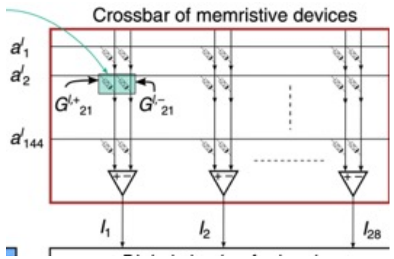
Communication dominates arithmetic

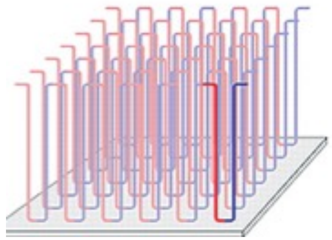
Source: Dally, NVIDIA

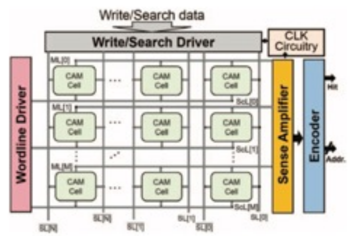
Samsung, Lee, Sukhan, et al. ISCA 2021
UPMEM by Gómez-Luna, Juan, et al. arXiv:2105.03814 (2021)
In-PCM Computing: Joshi, Vinay, et al. Nature Communications 11.1 (2020): 1-13.
CAM accelerators: Hu, Sharon, et al. 2021 IEDM





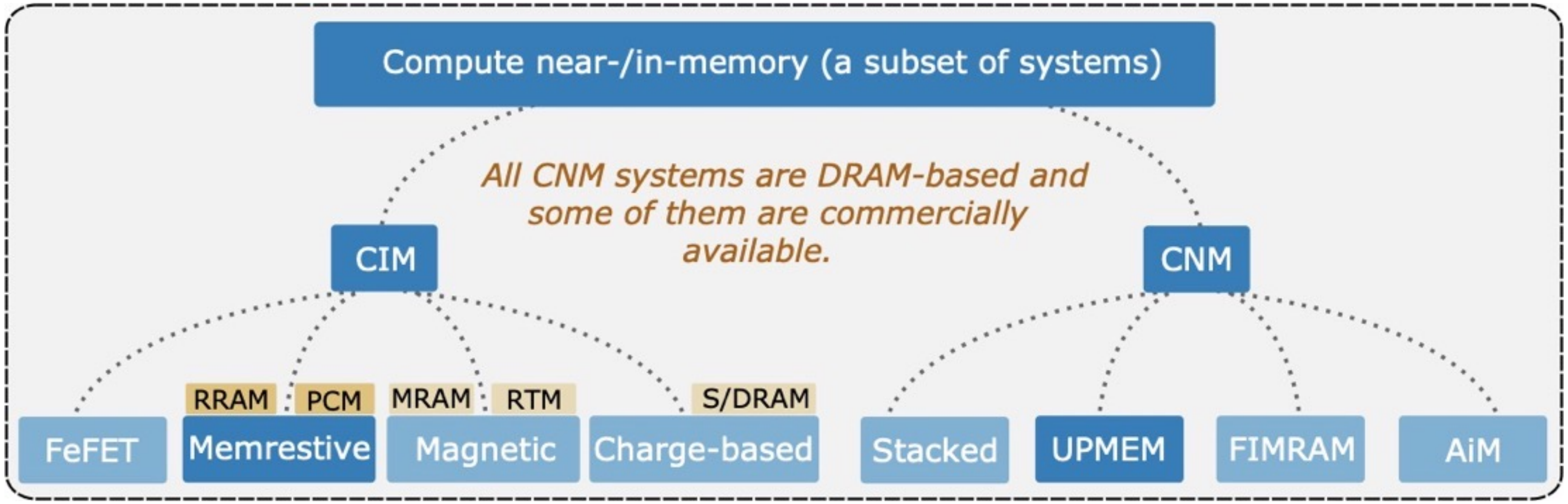






Compilation for heterogeneous CIM/CNM systems

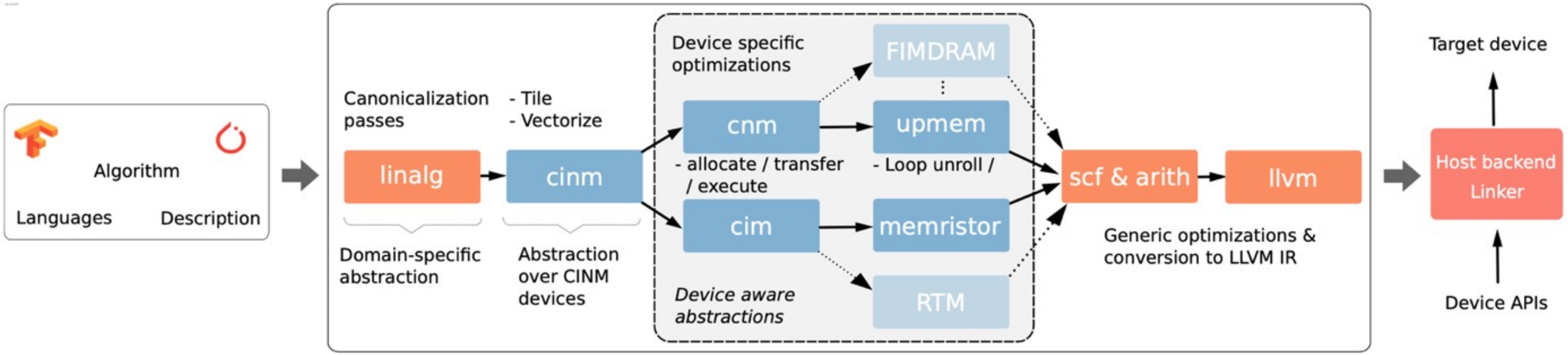
- A partial landscape/taxonomy of the CIM and CNM systems



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

Towards a generalized MLIR infrastructure

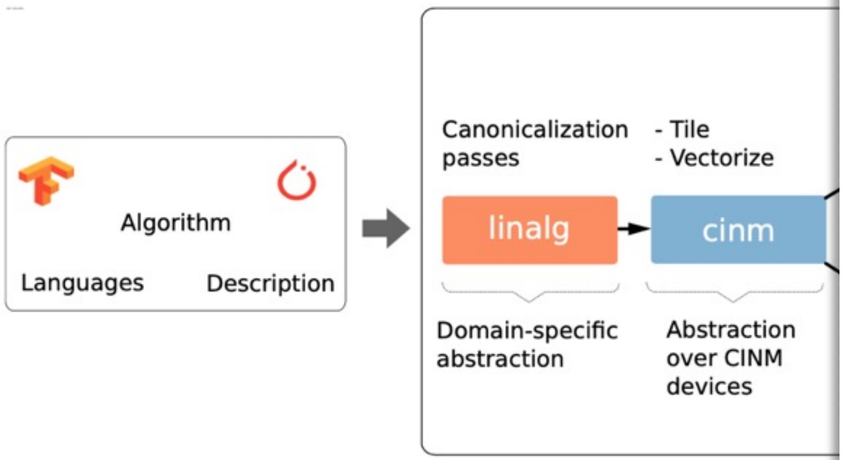
- Entry: linear algebra abstraction (common to ML frameworks and beyond)
- Intermediate languages for **in and near memory computing**
- Target-specific models and optimizations



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

Towards a generalized MLIR infrastructure

- Entry: linear algebra abstraction (common to ML frameworks and beyond)
- Intermediate languages for **in and near memory computing**
- Target-specific models and optimizations



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

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

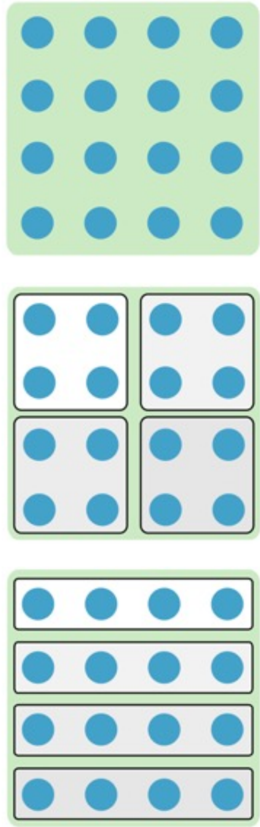
```
    uint32_t mram_base_addr_A = (uint32_t) (DPU_MRAM_HEAP_POINTER );
    uint32_t mram_base_addr_B = (uint32_t) (DPU_MRAM_HEAP_POINTER + ROWS * COLS *
↪   sizeof(T));
    uint32_t mram_base_addr_C = (uint32_t) (DPU_MRAM_HEAP_POINTER + 2 * ROWS * COLS
↪   * sizeof(T));
    for(int i = (tasklet_id * 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: Matmult

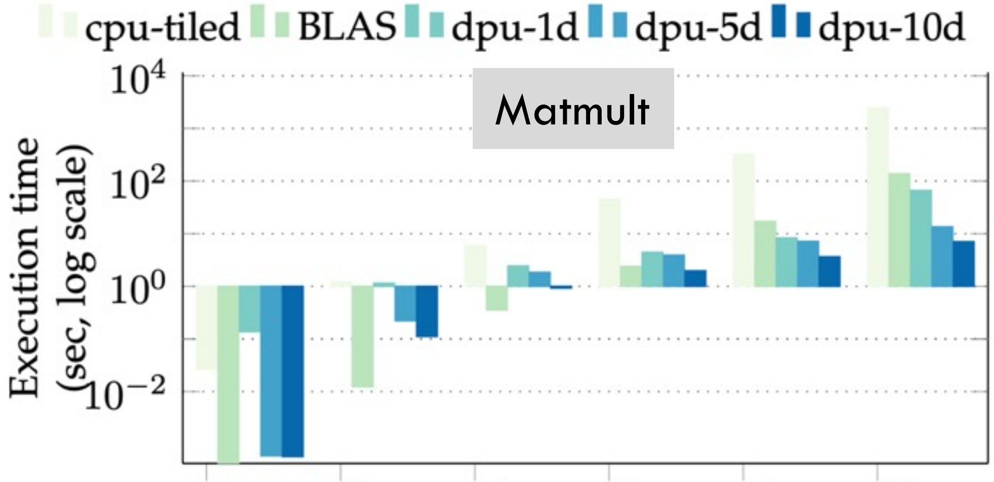
```
def mm(int32(64, 64) A, int32(64, 64) B) -> (int32(64, 64) C) {
  C(i,j) += A(i,k) * B(k,j)
           where i in 0:64, k in 0:64, j in 0:64
}
```

```
%D = linalg.matmul ins(%A, %B : tensor<64x64xi32>, tensor<64x64xi32>)
     outs(%C: tensor<64x64xi32>)
return %D : tensor<64x64xi32>
```

```
%C = scf.for %i = %cst0_i to %cst64_i step %cst16_i
<...>
    %B_tile = tensor.extract_slice %B[%o2, %o1][16, 16][1, 1]:
            tensor<64x64xi32> to tensor<16x16xi32>
    %A_t_b = bufferization.to_memref %A_tile: memref<16x16xi32>
    %B_t_b = bufferization.to_memref %B_tile: memref<16x16xi32>
    %A_dev = cnm.load_matrix %A_t_b[%c0, %c0] {leadDimension = 16: index}
            : memref<16x16xi32> -> !cnm.matrix<16x16xi32>
    %B_dev = cnm.load_matrix %B_t_b[%c0, %c0] {leadDimension = 16: index}
            : memref<16x16xi32> -> !cnm.matrix<16x16xi32>
    %C_part = cnm.op.gemm %A_dev, %B_dev: tensor<16x16xi32>, tensor<16x16xi32>
    %out_tile = arith.addf %in_tile, %C_part: tensor<16x16xi32>
    scf.yield %out_tile : tensor<16x16xi32>
   }
%out_result = tensor.insert_slice %C_tile, %in_result[%o0, %o1][16, 16][1, 1]:
<...>
```

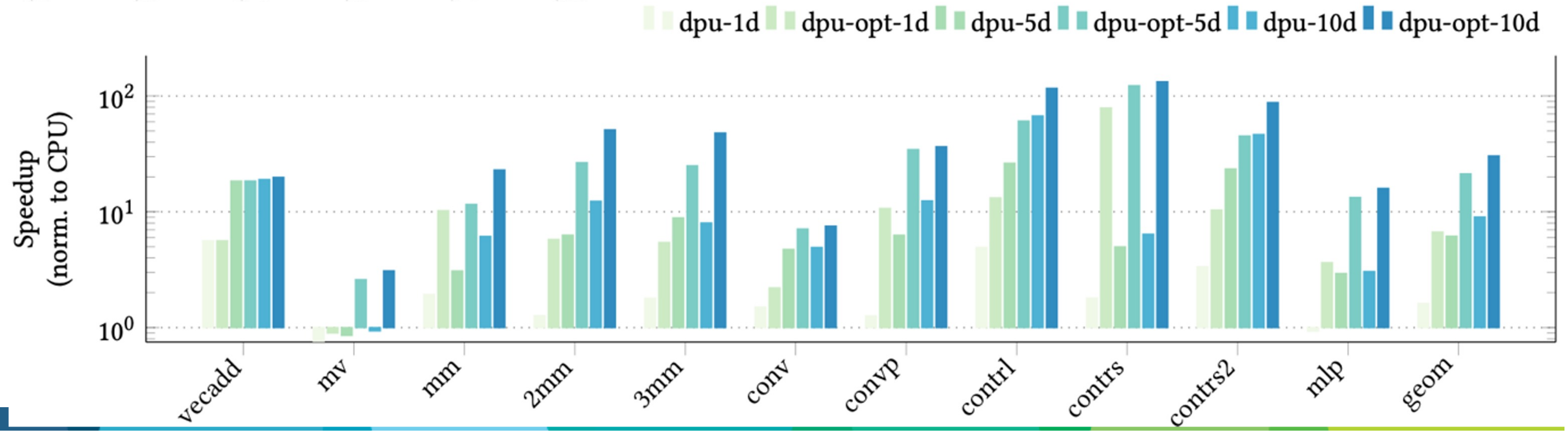


UPMEM example: Results



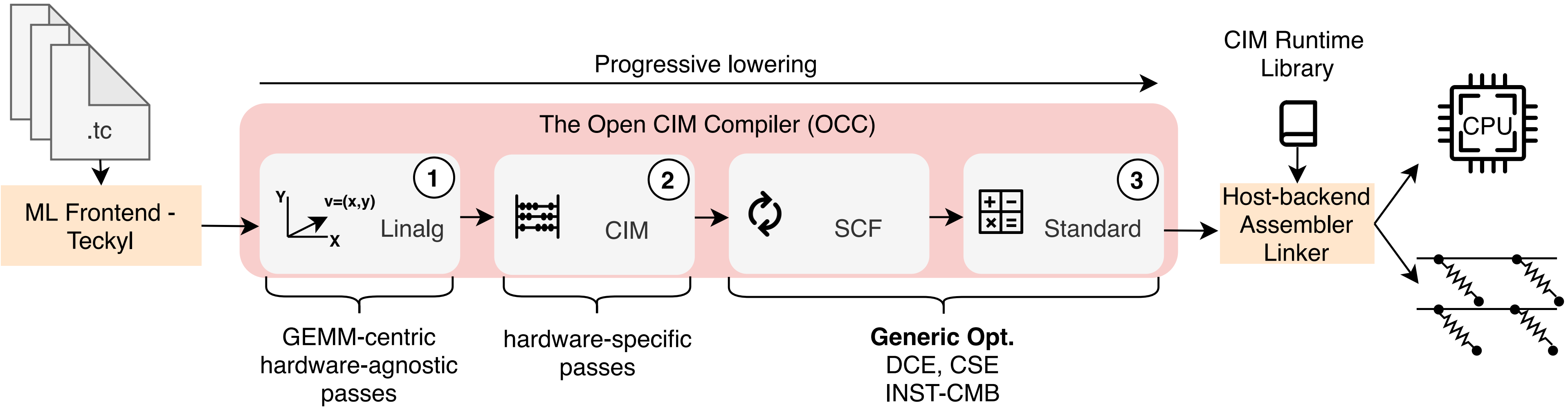
1-DIMM 128 DPUs
5-DIMMs 640 DPUs
10-DIMMs 1280-DPUs

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



CIM Pipeline: Cross-bar example

- MLIR dialect for general tensor expressions (Tensor Comprehensions)
  - Reuse GEMM transformations from linalg
  - Lower to CIM dialect (co-existing with SCF and Standard)
  - Lower CIM dialect to runtime APIs



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

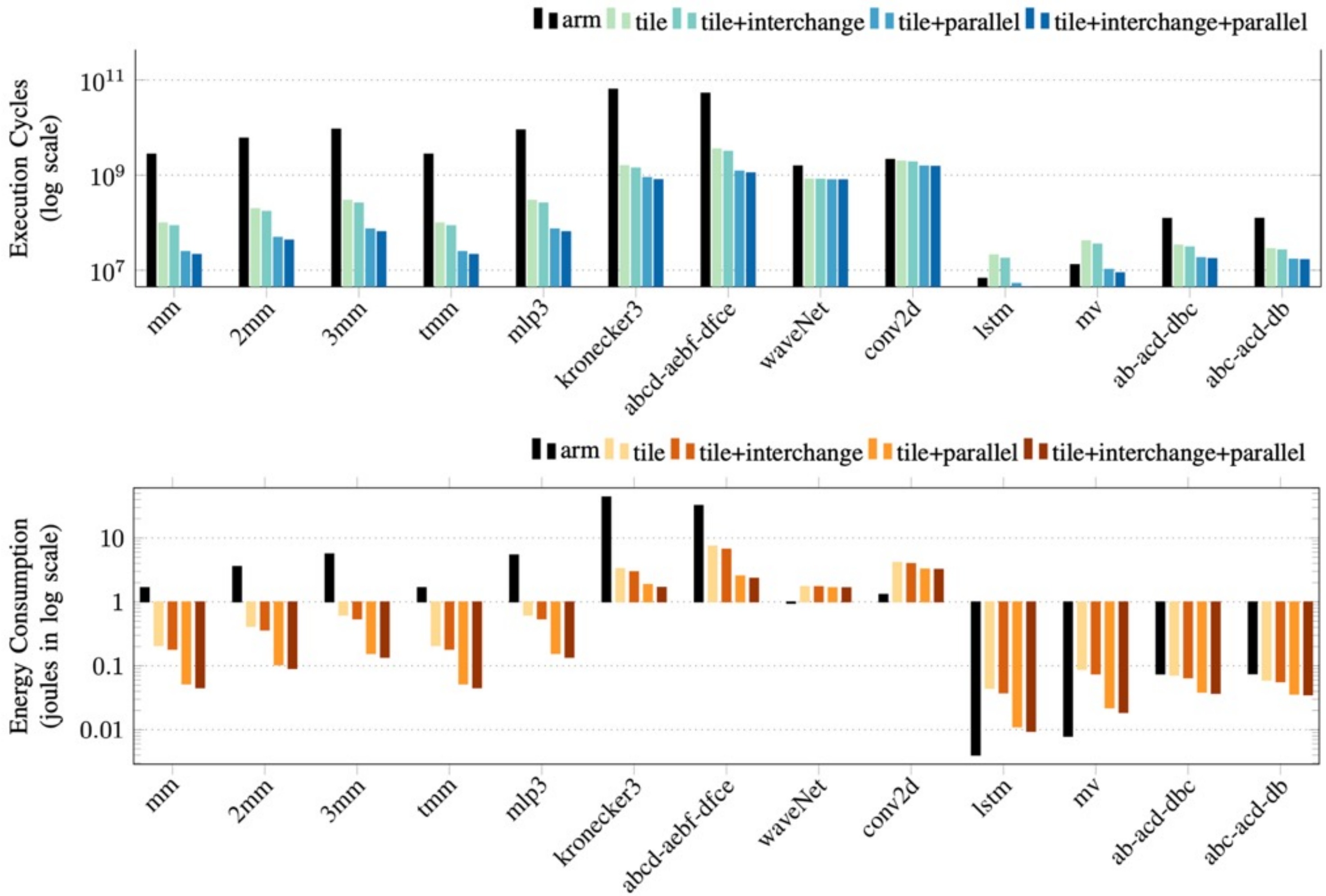
Lowering examples (somewhat beyond matmult)

```
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: Crossbars beyond matmult

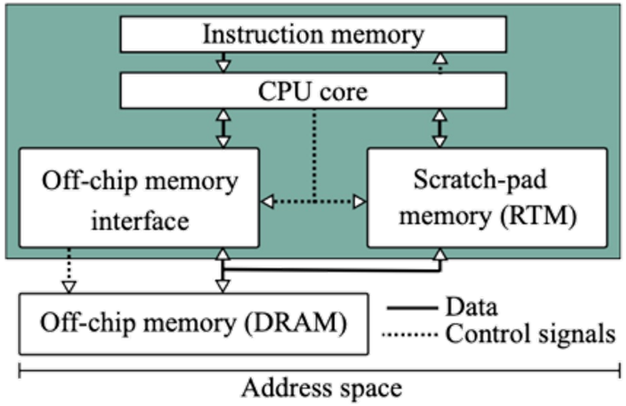


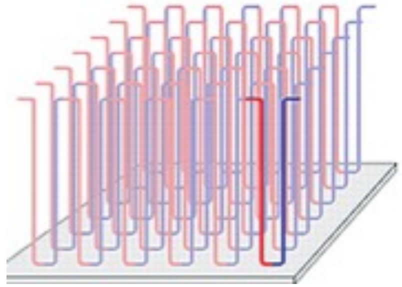
A. Siemieniuk, L ... Machine
Learning Optim

CHAIR FOR COMPILER CONSTRUCTION

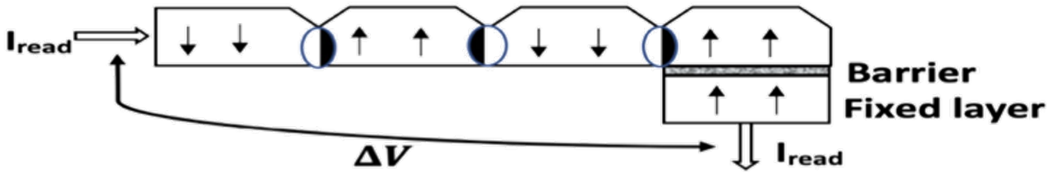
Racetrack memories

- Racetrack memories (RTM)
  - Extreme density
  - Sequential bit access per cell
  - Sequentiality on top of locality
  - Ex.: Placement for tensor contraction





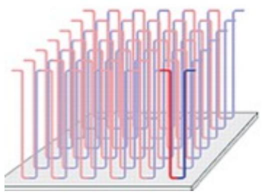
- Different approaches for in-RTM computing proposed
  - Example: Transverse reads
  - Interesting data-allocation problems

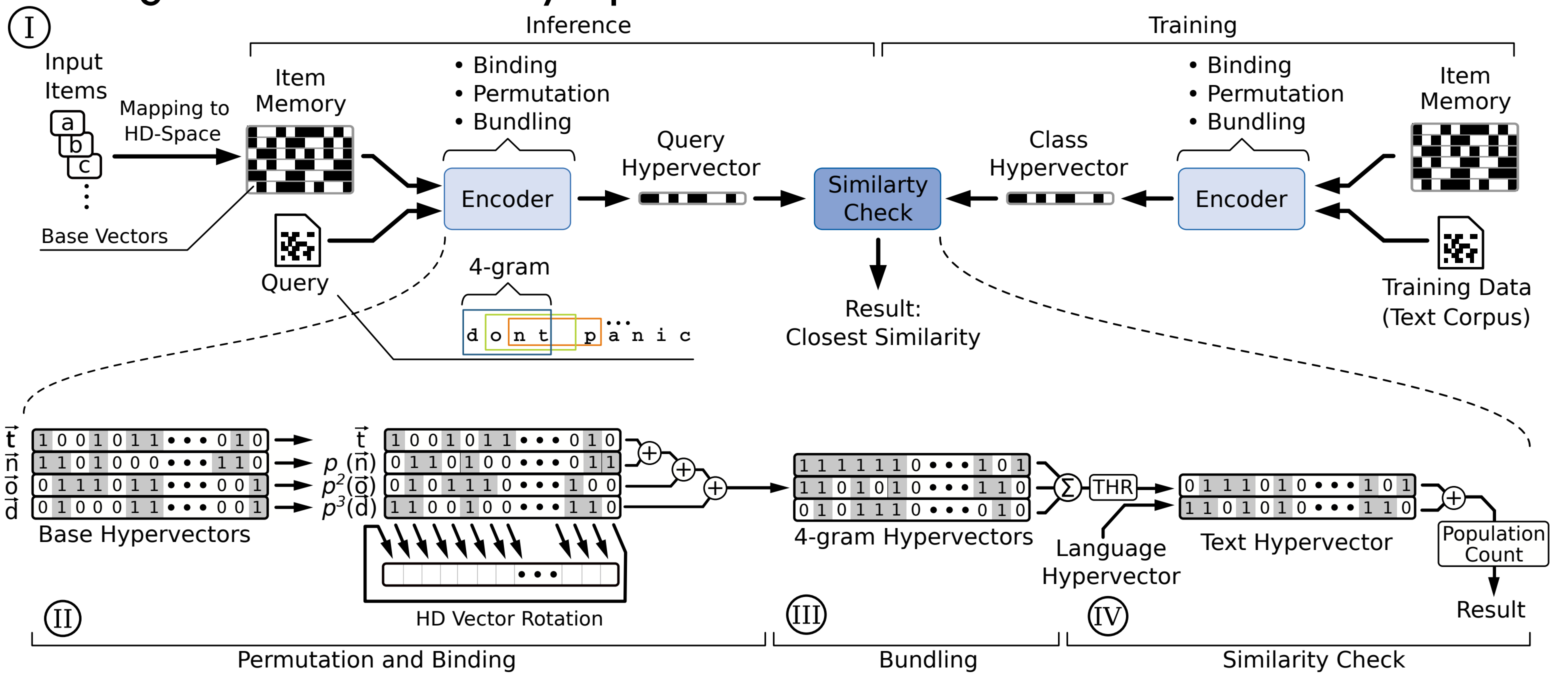


K. Roxy, IEEE T Nano 2020

Example: Hyper dimensional computing (HDC)

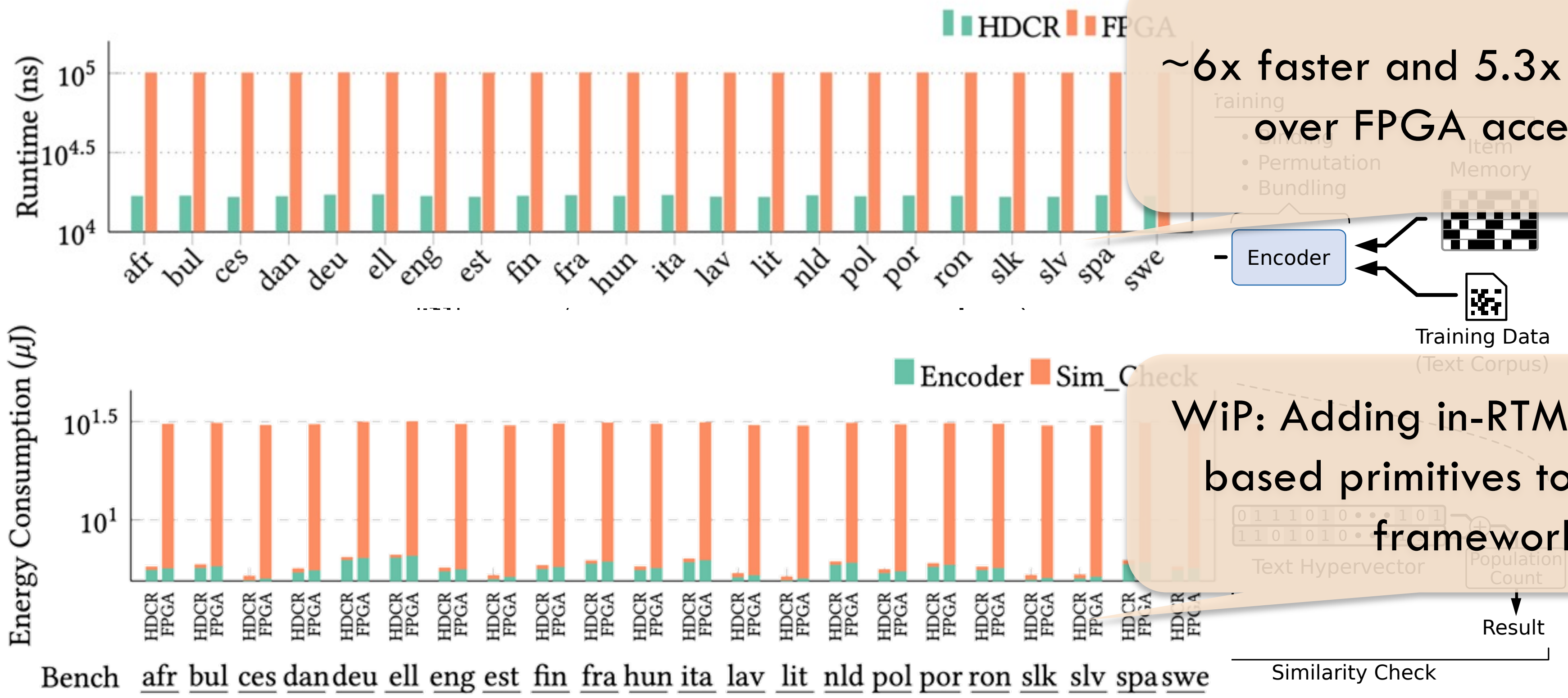
- HDC: Embed data in 10 k-dimensions – Von-Neuman Bottleneck!
  - Leverage bulk-wise binary operations





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)



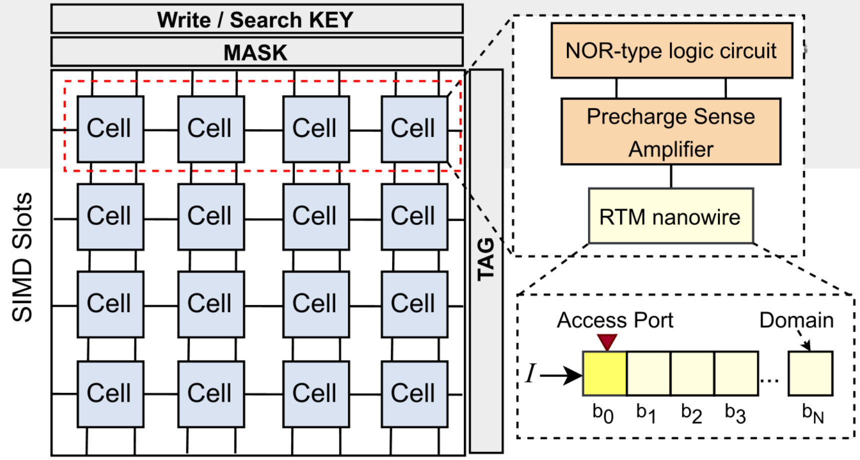
~6x faster and 5.3x less energy over FPGA accelerator

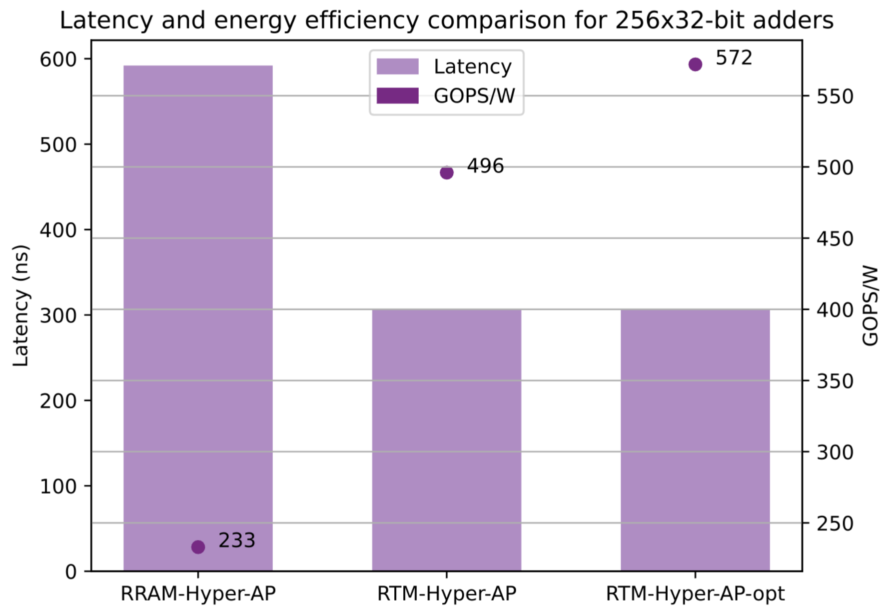
WiP: Adding in-RTM and CAM-based primitives to compiler framework

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

Associative Processors with RTMs

- Search-write computational paradigm with CAM-based memories
- Use RTM nanowires for multi-bit serial storage in a cell
- Improved performance and energy efficiency!

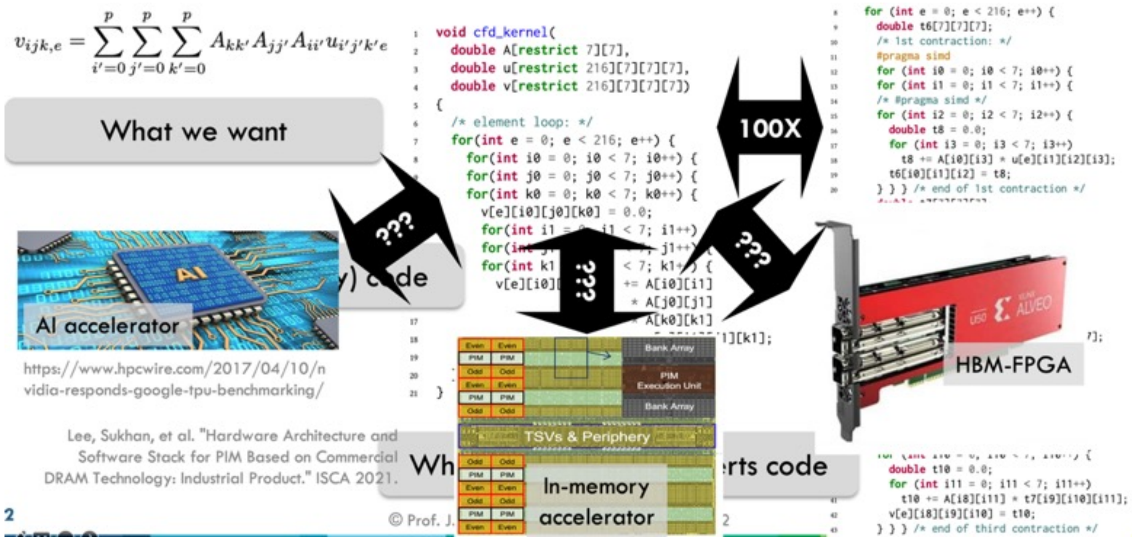




J. P. C. de Lima, A. A. Khan, H. Farzaneh, J. Castrillon, "Efficient Associative Processing with RTM-TCAMs" 1st in-Memory Architectures and Computing Applications Workshop (iMACAW), 2pp, Jul 2023

Summary

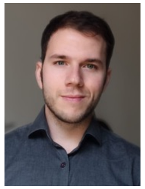
- Challenging & exciting computing landscape!
- Abstractions important to target emerging and domain-specific systems
- Beyond infrastructure: Need execution models of non Von Neumann
- Beyond energy efficiency: undersstanding of full-life-cycle sustainability

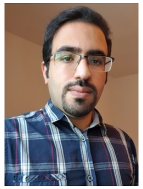


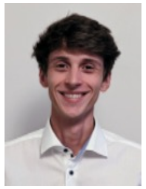
CHAIR FOR COMPILER CONSTRUCTION

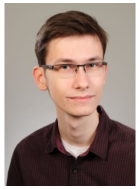
Thanks! & Acknowledgements


Hasna Bouraoui


João P. de Lima

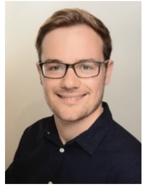

Hamid Farzaneh


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 (Andres Goens, Norman Rink, Sven Karol, Sebastian Ertel), and collaborators (J. Fröhlich, I. Sbalzarini, T. Grosser, C. Pilato, S. Parkin)

Funded by

Deutsche Forschungsgemeinschaft

HetCIM (502388442), CO4RTM (450944241)

(01IS18026A)

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

References

[RWDSL'18] N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". 3rd International Workshop on Real World Domain Specific Languages (RWDSL 2018), Feb 2018.

[GPCE'18] A. Susungi, et al. " "Meta-programming for cross-domain tensor optimizations". Proceedings of 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'18), Nov. 2018.

[ACM TOMS'18] Karol, S., Nett, T., Castrillon, J., Sbalzarini, I. F. "A Domain-Specific Language and Editor for Parallel Particle Methods", ACM Transactions on Mathematical Software (TOMS), ACM, 2018, 44, 34:1-34:32.

[JCS'21] N. Khouzami, F. Michel, P. Incardona, J. Castrillon, and I. F. Sbalzarini. "Model-based Autotuning of Discretization Methods in Numerical Simulations of Partial Differential Equations". In: Journal of Computational Science 57, 2021.

[ICCS'21] N. Khouzami, L. Schütze, P. Incardona, L. Kraaz, T. Subic, J. Castrillon, and I. F. Sbalzarini. "The OpenPME Problem Solving Environment for Numerical Simulations". In: International Conference on Computational Science (ICCS'21).

[Array'19] N.A. Rink, N. A. and J. Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY), ACM, 2019, pp. 57-68

[Cluster'21] 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

[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, and J. Castrillon. "BASE2: An IR for Binary Numeral Types". In: 13th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2023). June 2023.

[Arxiv'23] A. A. Khan, H. Farzaneh, K. F. A. Friebel, L. Chelini, and J. Castrillon. "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In- Memory and Compute Near-Memory Paradigms". In Arxiv, Jan 2023.

[TCAD'21] 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.

[TECS'22] 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

[iMACAW'23] J. P. C. de Lima, A. A. Khan, H. Farzaneh, J. Castrillon, "Efficient Associative Processing with RTM-TCAMs" 1st in-Memory Architectures and Computing Applications Workshop (iMACAW), 2pp, Jul 2023