

CPS Summer School: Designing Cyber-Physical Systems – From concepts to implementation

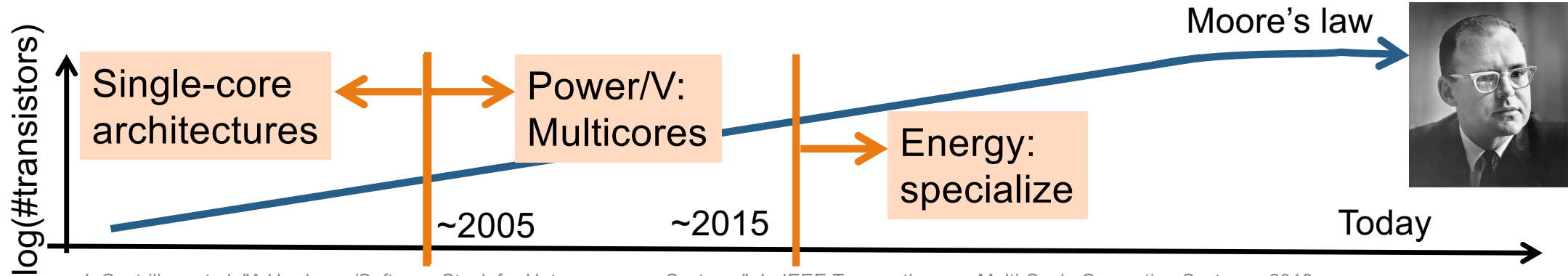
Jeronimo Castrillon

Chair for Compiler Construction (CCC) TU Dresden

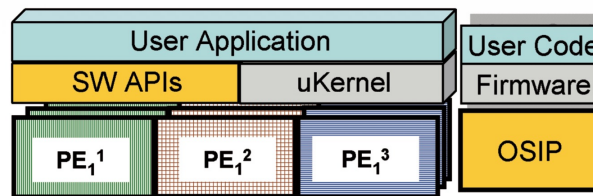
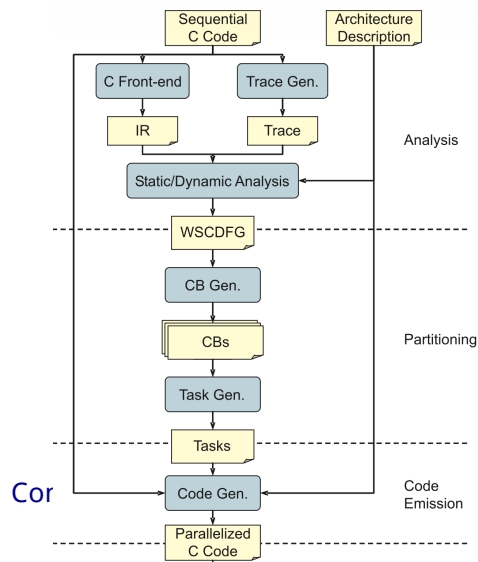
Compilers for Emerging Heterogeneous Systems

September 23, 2026. Alghero, Italy

HW evolution: The thermal and energy walls

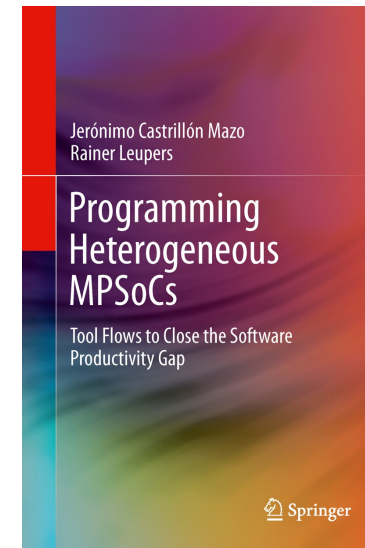


J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems. 2018.

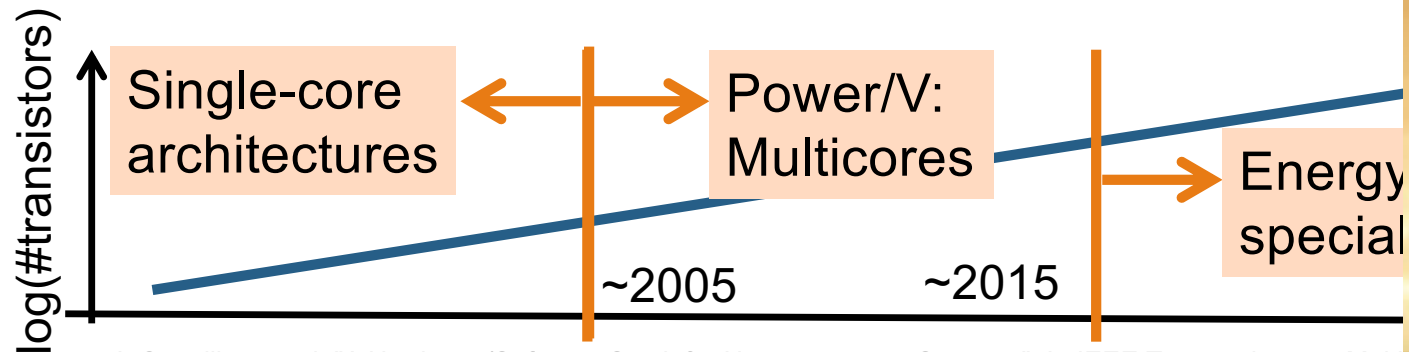


J. Ceng, J. Castrillon, et al. "MAPS: An Integrated Framework for MPSoC Application Parallelization", **DAC 2008**
 J. Castrillon, et al. "Task Management in MPSoCs: an ASIP Approach", **ICCAD 2009**

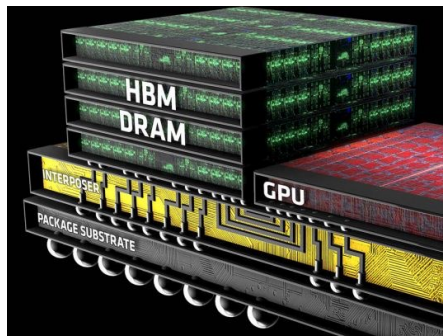
PSS. 2026 • © Prof. J. Castrillon • Slide 2



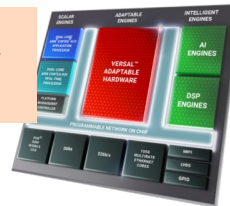
HW evolution: The thermal and energy walls



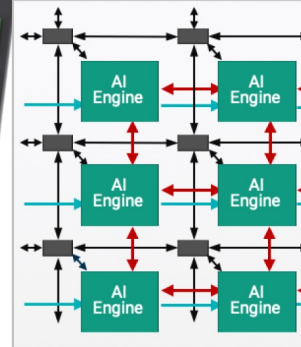
J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi



Accelerators & Prog. logic



CPU+GPU and High-bandwidth memory

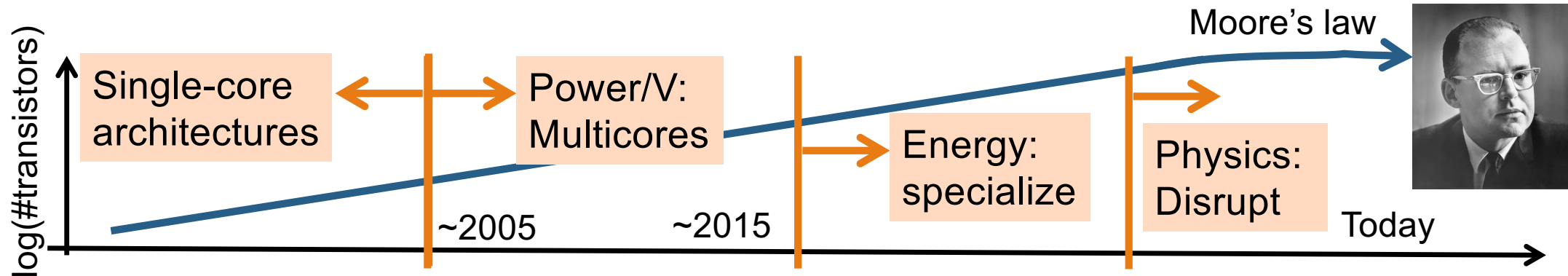


Source: AMD

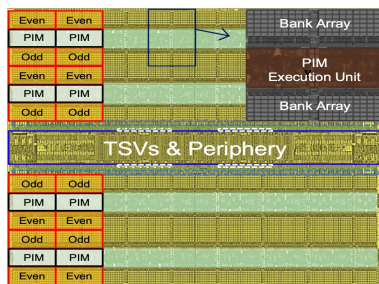


Source: AMD, AnandTech

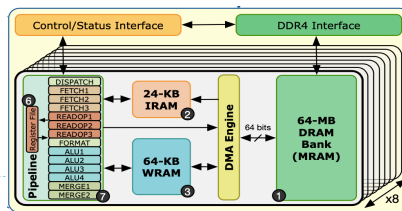
HW evolution: The thermal, energy and physics walls



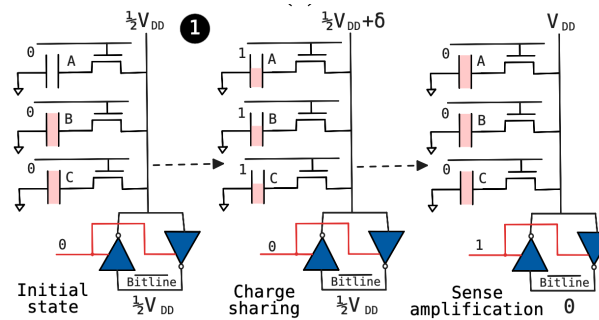
J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 243-259, Jul 2018.



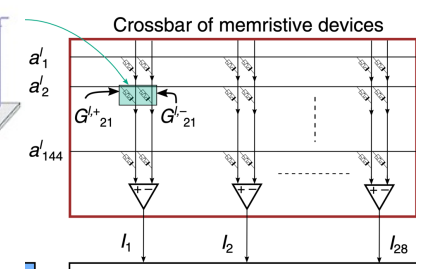
Source: Samsung



Source: UPMEM



Source: Ambit, O. Mutlu

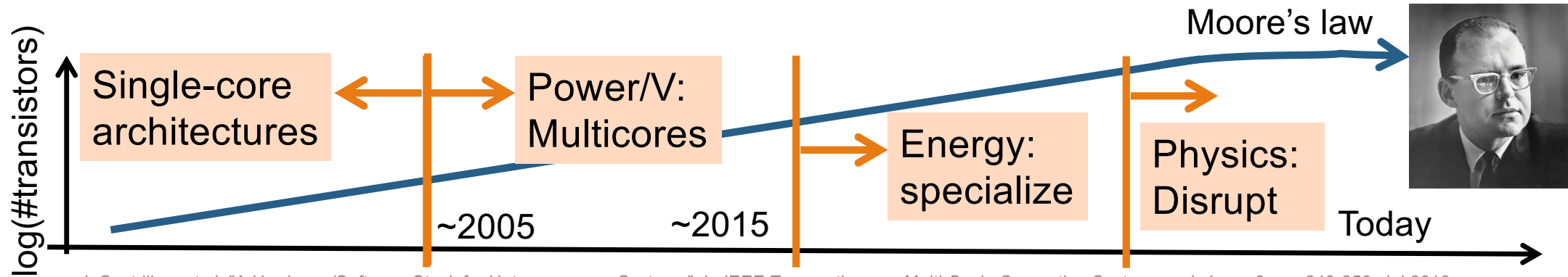


Source: IBM

Near-memory computing

Emerging memories + in-memory computing

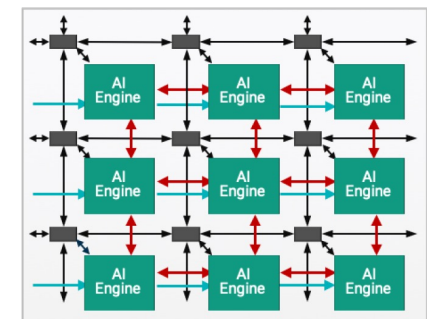
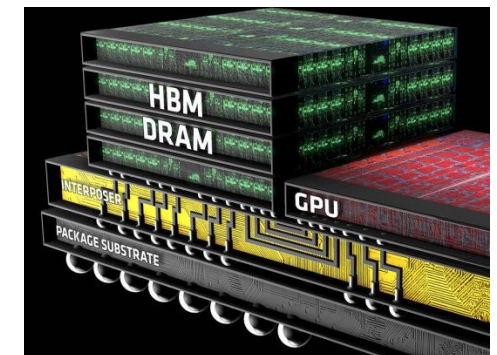
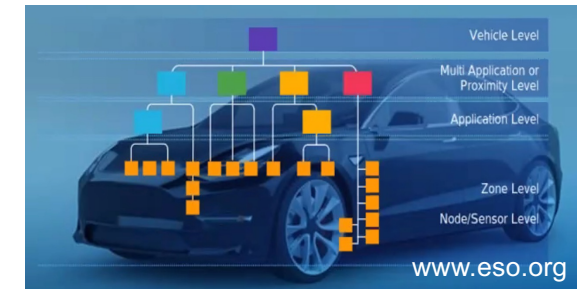
HW evolution: The thermal, energy and physics walls



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.

Need for higher-level **programming abstractions**, next-generation **compilers** and novel **computational models** for emerging computing systems

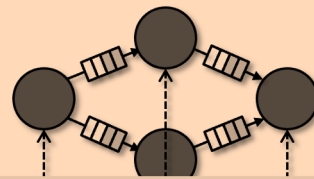
Coordination languages



“Recall” from a previous CPSS

Dataflow programming

- ❑ Graph representation of applications: Long history
 - ❑ Implicit repetitive execution of tasks
 - ❑ Good model for streaming applications
 - ❑ Good match for signal processing & multi-media
- ❑ The why
 - ❑ Explicit parallelism
 - ❑ Often: Determinism
 - ❑ Better analyzability (scheduling, mapping, optimization)

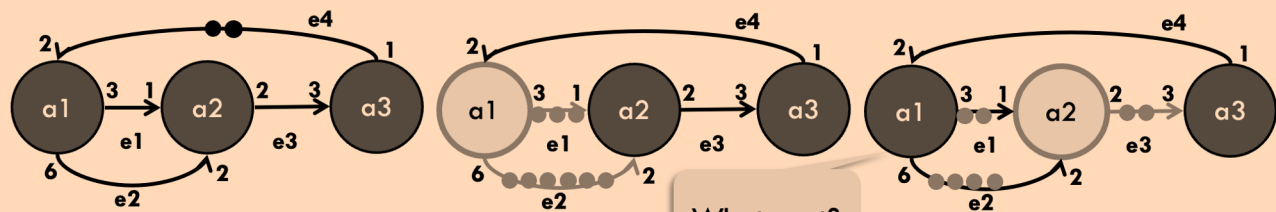


Example: Synchronous dataflow graph (SDF)

Def.: An **SDF** is an annotated multi-graph $G=(V,E,W)$. V is the set of actors and $E \subseteq V \times V$ the set of channels. $W = \{w_1, \dots, w_{|E|}\} \subset \mathbb{N}^3$ is a set of annotations for every channel $e = (a_1, a_2)$, $w_e = (p_e, c_e, d_e)$: p_e is the number of tokens produced by a firing of a_1 , c_e the tokens consumed by a_2 and d_e the amount of delay tokens

Example

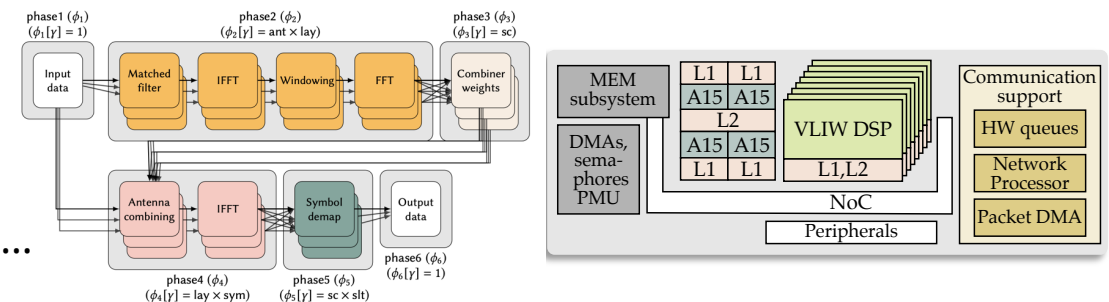
Originally in [Lee'87]



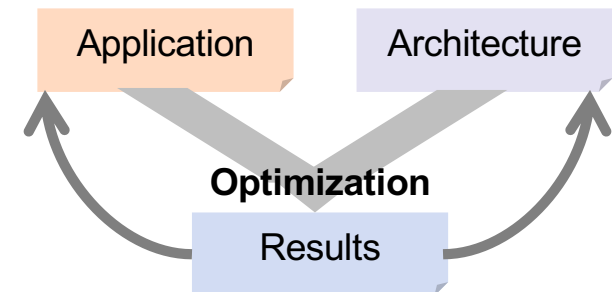
Coordination to ease design and programming

□ Model-based approach

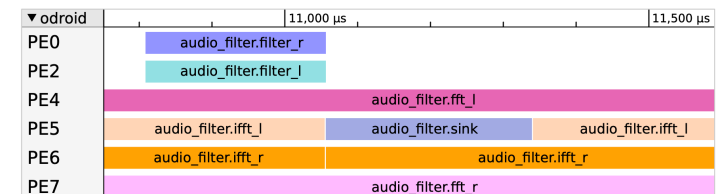
- Graph model (HW and SW)
- Early estimation of performance/energy/...



□ Successful iterative co-design methodologies (since the 80s)



J. Castrillon, et al. "Dataflow Models of Computation for Programming Heterogeneous Multicores", Springer Nature Singapore, pp. 1–40, Singapore, Jan 2023



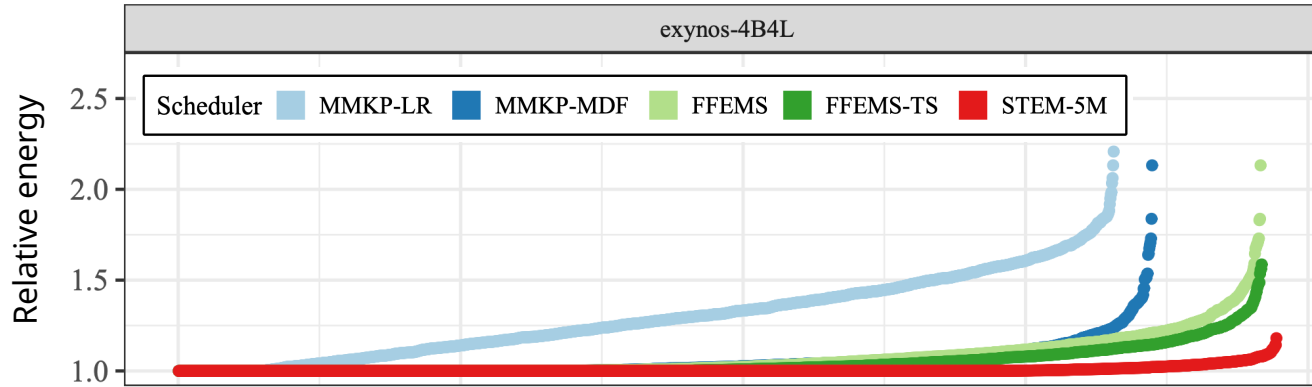
Adaptive dataflow mapping

□ Defining symmetries on mapping spaces for *rotations*: Paying Tetris with mappings at runtime

A. Goens, et al. "Symmetry in Software Synthesis", In ACM Transactions on Architecture and Code Optimization (TACO), ACM, vol. 14, no. 2, pp. 20:1–20:26, New York, NY, USA, Jul 2017

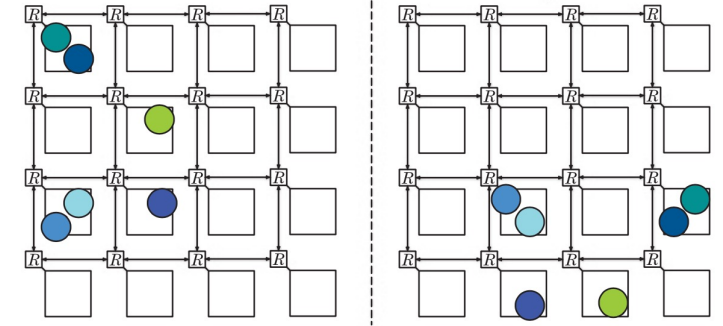
A. Goens, "TETRIS: a Multi-Application Run-Time System for Predictable Execution of Static Mappings". In: SCOPES '17.

□ Leveraging symmetries for efficient selection at runtime



R. Khasanov, J. Castrillon, "Energy-efficient Runtime Resource Management for Adaptable Multi-application Mapping", DATE'20

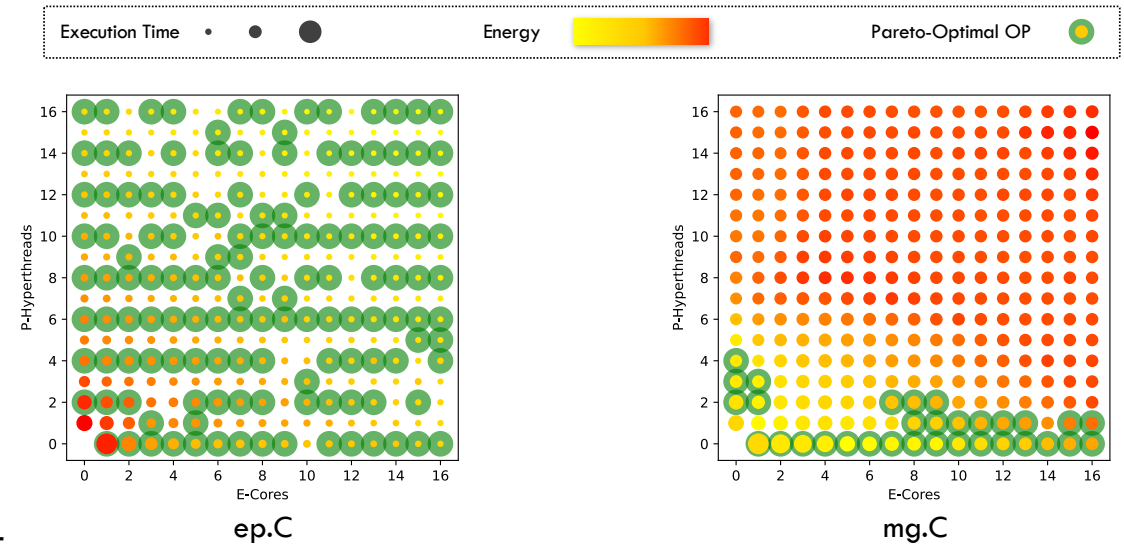
R. Khasanov, M. Dietrich, J. Castrillon, "Flexible Spatio-Temporal Energy-Efficient Runtime Management", ASP-DAC'24



$$\begin{aligned}
 & \text{minimize } \sum_{\mu \in \kappa} \sum_{\nu \in \mu} c_{\nu[\lambda]}^{\nu[j]}[\xi] \frac{|\Delta_{\mu}|}{c_{\nu[\lambda]}^{\nu[j]}[\tau]} \\
 & \text{subject to } \sum_{\nu \in \mu} c_{\nu[\lambda]}^{\nu[j]}[\bar{\theta}] \leq \bar{\Theta}, \quad \forall \mu \in \kappa \\
 & \quad \nu_1[\sigma] \neq \nu_2[\sigma], \quad \forall \nu_1, \nu_2 \in \mu, \nu_1 \neq \nu_2 \\
 & \quad \sum_{\substack{\mu \in \kappa \\ \nu \in \mu \\ \nu[\sigma] = \sigma}} \frac{|\Delta_{\mu}|}{c_{\nu[\lambda]}^{\nu[j]}[\tau]} = \sigma[\rho], \quad \forall \sigma \in \Sigma \\
 & \quad \max_{\substack{\mu \in \kappa \\ \nu \in \mu \\ \nu[\sigma] = \sigma}} \bar{\Delta}_{\mu} \leq \sigma[\delta], \quad \forall \sigma \in \Sigma.
 \end{aligned}$$

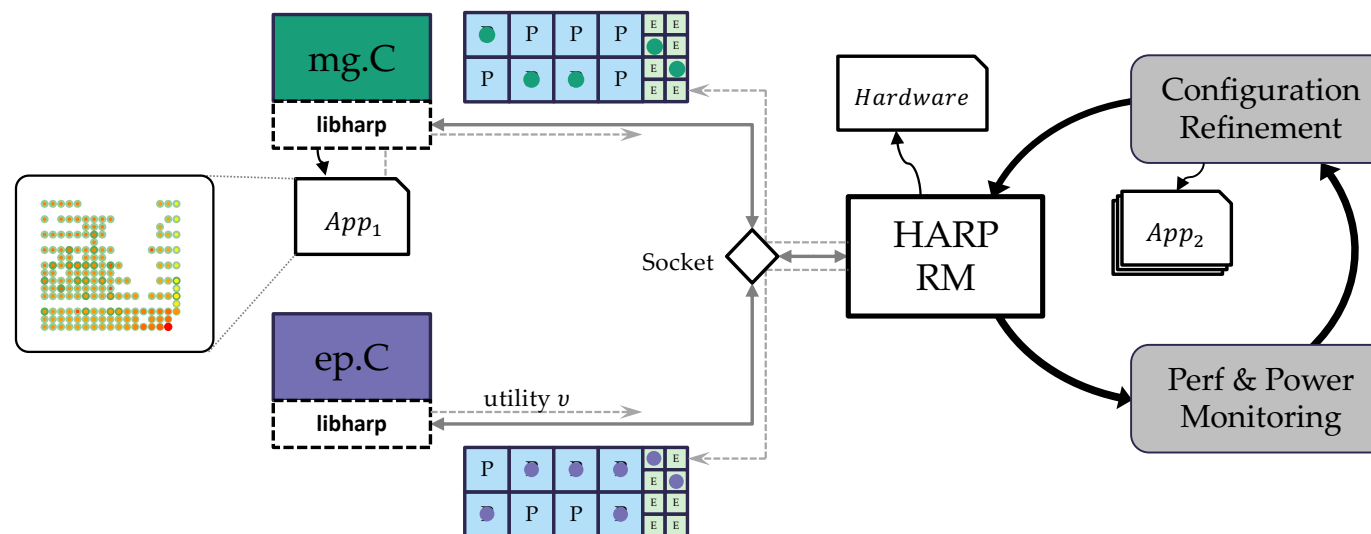
Adaptive compile/runtime mapping and scheduling

- Generalization for big/little, P/E cores
- Simplified symmetries for adaptation
- Hybrid adaptation
 - Offline characterization
 - Pareto-front approximation
 - Runtime selection and remapping
 - Jointly optimize multiple applications for energy efficiency



HARP approach

- ❑ Tight interaction within the Linux OS for switching and monitoring operating points

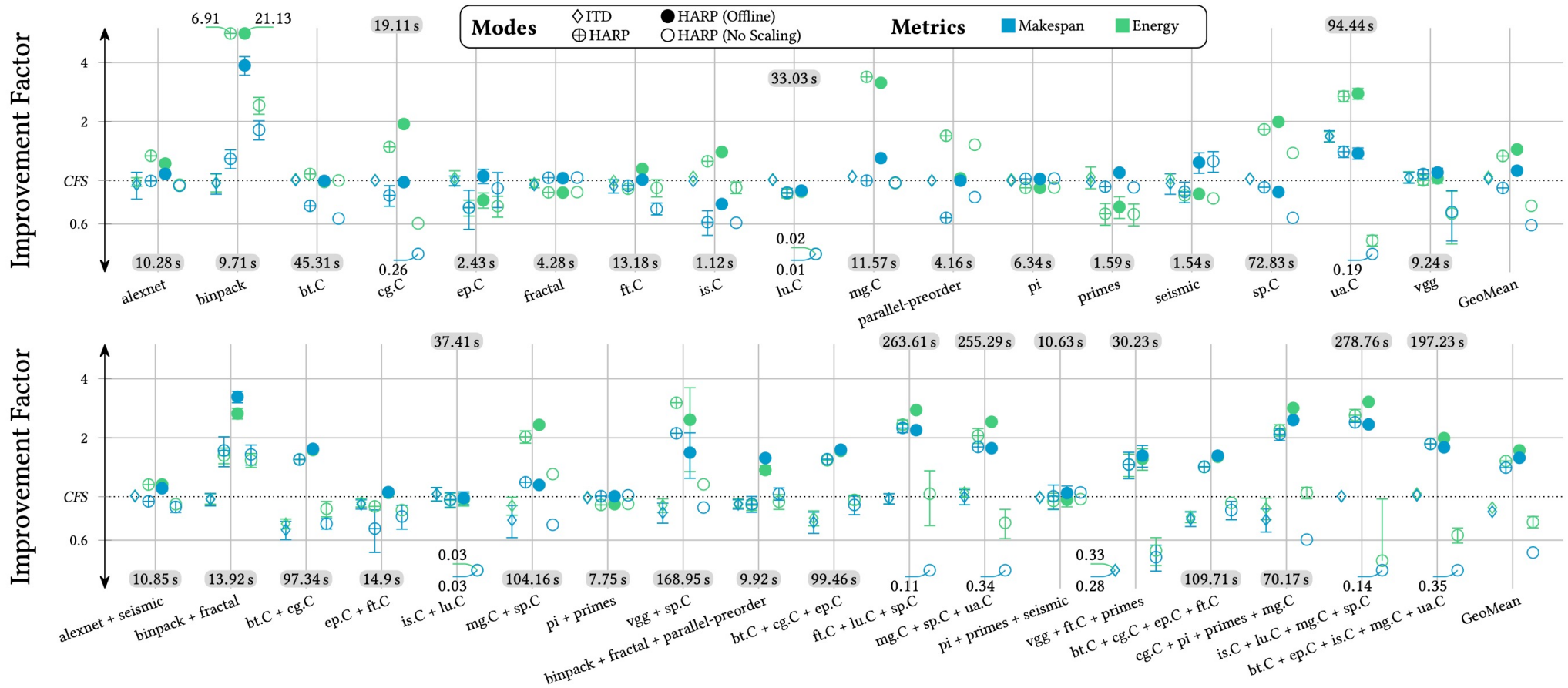


Best Paper Award

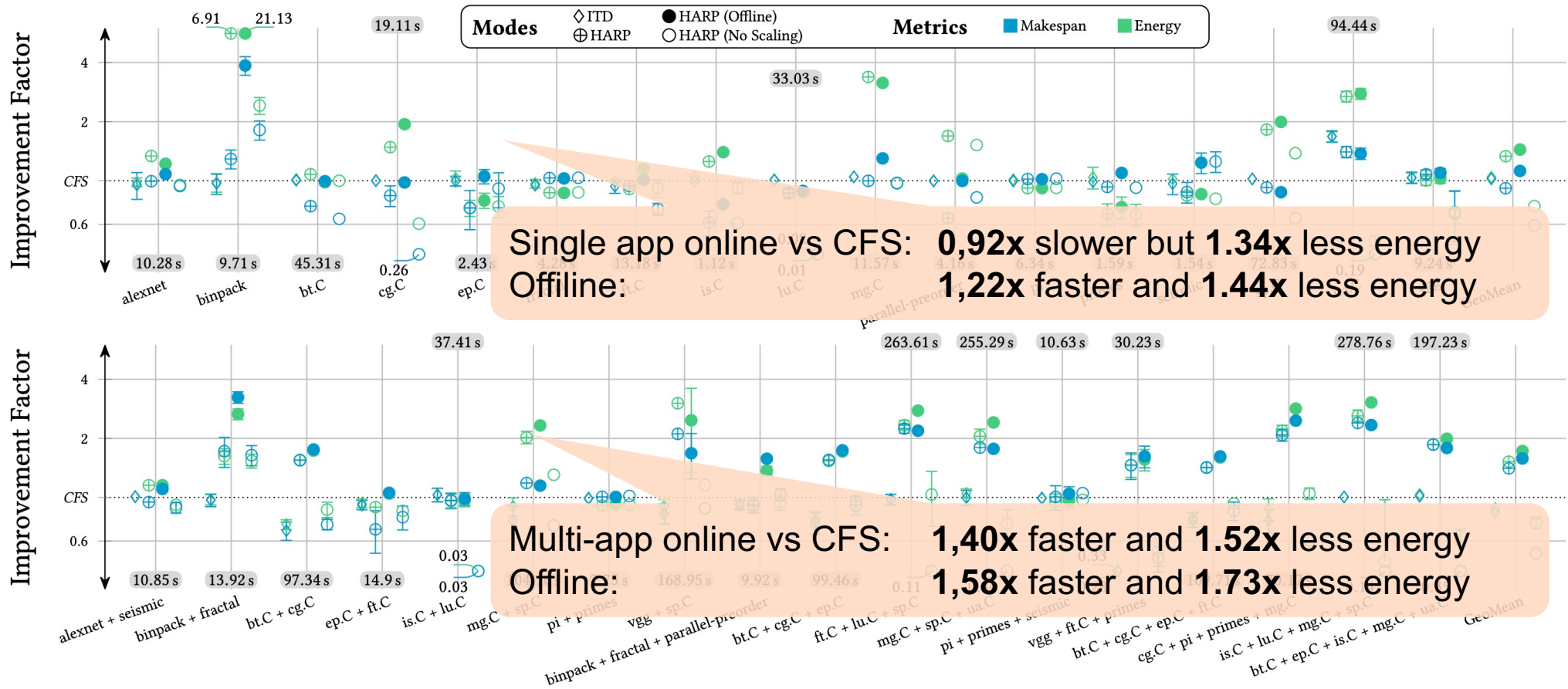
hon. mention

T. Smejkal, et al. "HARP: Energy-Aware and Adaptive Management of Heterogeneous Processors". Middleware'25

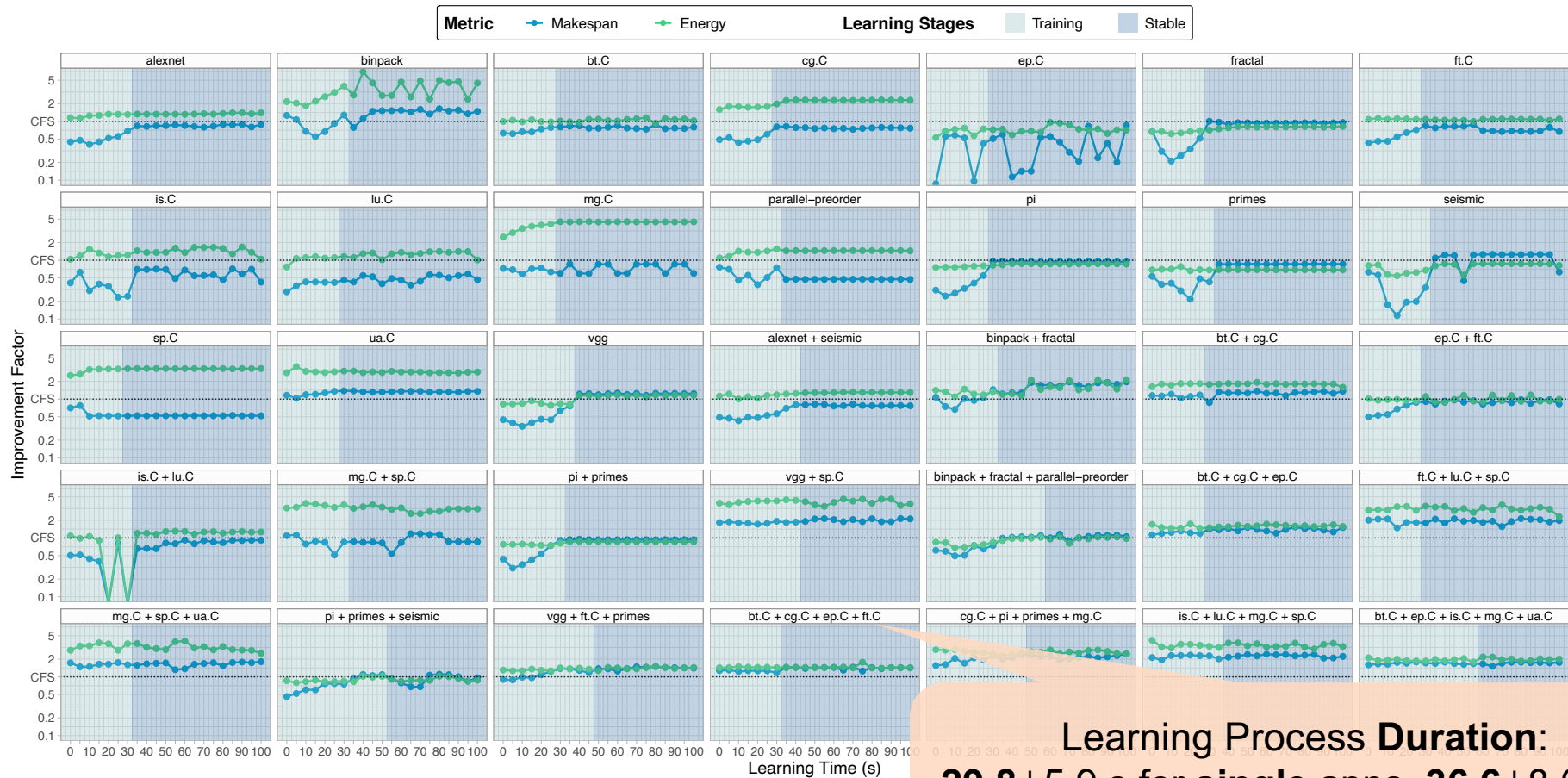
HARP on Intel Raptor Lake



HARP on Intel Raptor Lake



HARP monitoring for unknown applications



Learning Process Duration:
 29.8 ± 5.9 s for **single** apps, 36.6 ± 8.0 s for
 scenarios with **multiple** apps

Reactive cyber-physical systems (CPSs): Beyond dataflow

□ CPSs have to

- React to incoming sources
- Execute (different dataflow) within deadline
- Coordinate across subsystems
- Behave predictably and deterministically

Ongoing work: Autonomous driving

□ Currently analyzing Adaptive Autosar: **Suitable models?**

- Service-oriented architecture
- Test application for Breaking Assistant

□ Analysis, Debugging, Testing larger applications?

Looking into models that combine dataflow and discrete events [CyPhy'19]

```
int main() {  
    s = ServiceProxy();  
    s.set_value(1);  
    s.add(2);  
    result = s.get_value();  
    std::cout << result.get();  
    return 0;  
}
```

Emergency Brake Assist

Adaptive AUTOSAR

Linux with RT Patch

HPC

Ethernet / SOME-IP

Video Provider

Video Adapter

Pre Processing

Computer Vision

EBA

OTA

Brake SWC

UI

Probability

Printed Value

0 1 2 3

Giovanni Vergine, et al. Franca / ARA:COM Interoperability. Technology Brief. Dec. 2018

© Prof. J. Castrillon. CPS Summer School. Alghero, Italy, 2019



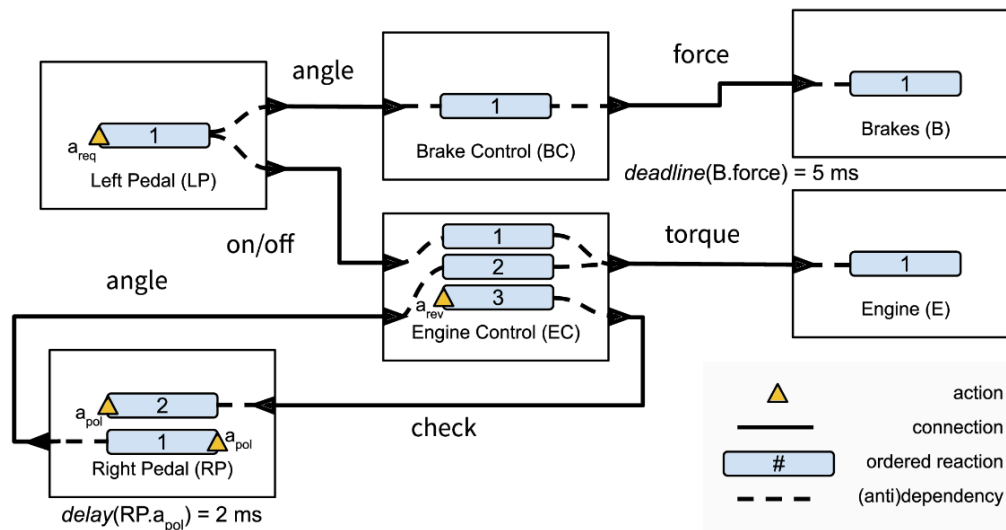
Ingenuity helicopter flying on Mars

Source: NASA/JPL-Caltech/ASU/MSSS

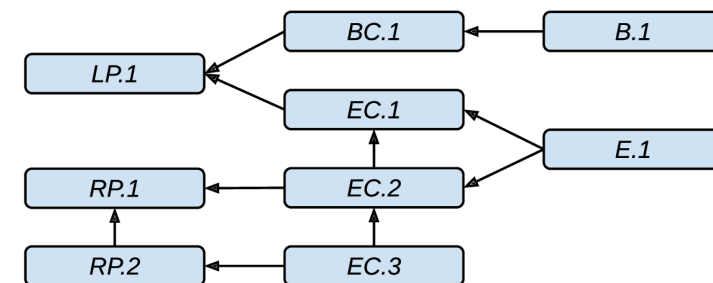
<https://science.nasa.gov/resource/mastcam-z-video-of-ingenuity-taking-off-and-landing/>

Time semantics in CPS: Lingua Franca

- Time semantics in the model of computation
- Reactos/LinguaFranca for time-deterministic execution
- Supports different languages: C/C++, Rust, Python, ...



Dependency Graph



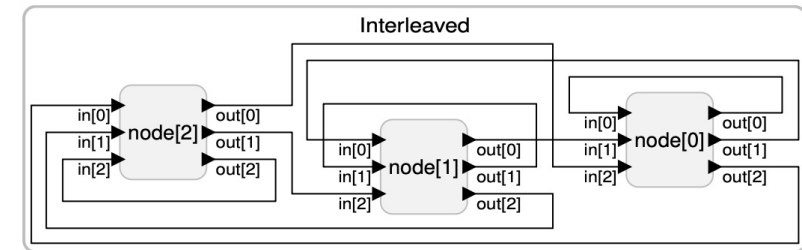
Marten Lohstroh, et al. "Reactors: A Deterministic Model for Composable Reactive Systems", CyPhy 2019
 M. Lohstroh, et al. "Deterministic Coordination Across Multiple Timelines", In ACM TECS, 2023.

Patterns, abstractions and runtime

Convenience syntax extensions

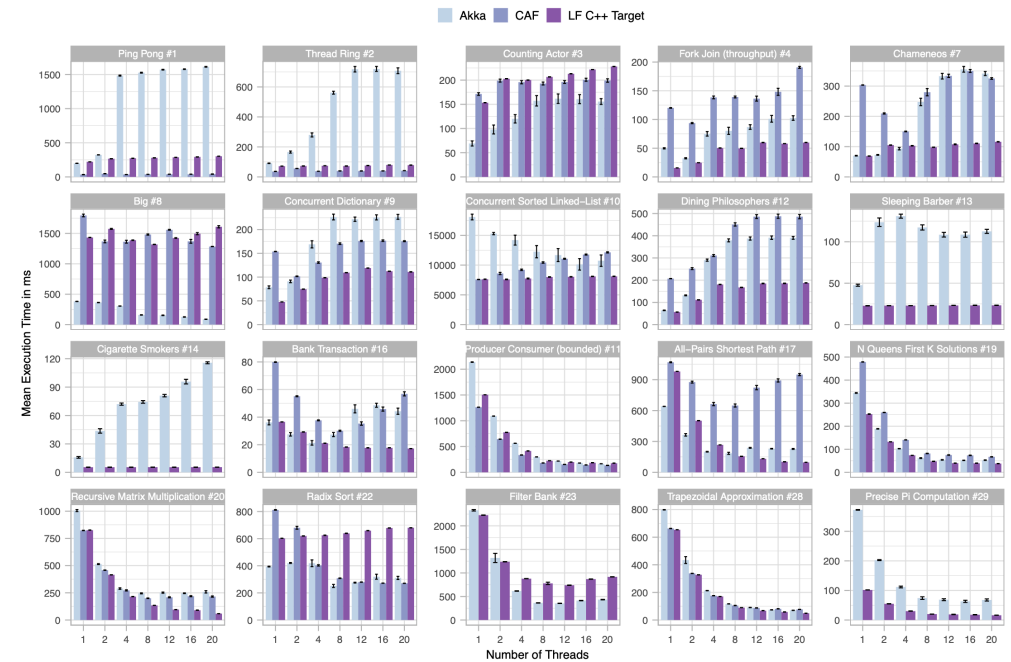
```
1 main reactor(w: int(3)) {  
2   node = new[w] Node(w=w)  
3   node.out -> interleaved(node.in)  
4 }
```

C. Menard, et al "High-Performance Deterministic Concurrency using Lingua Franca", ACM TACO 2023



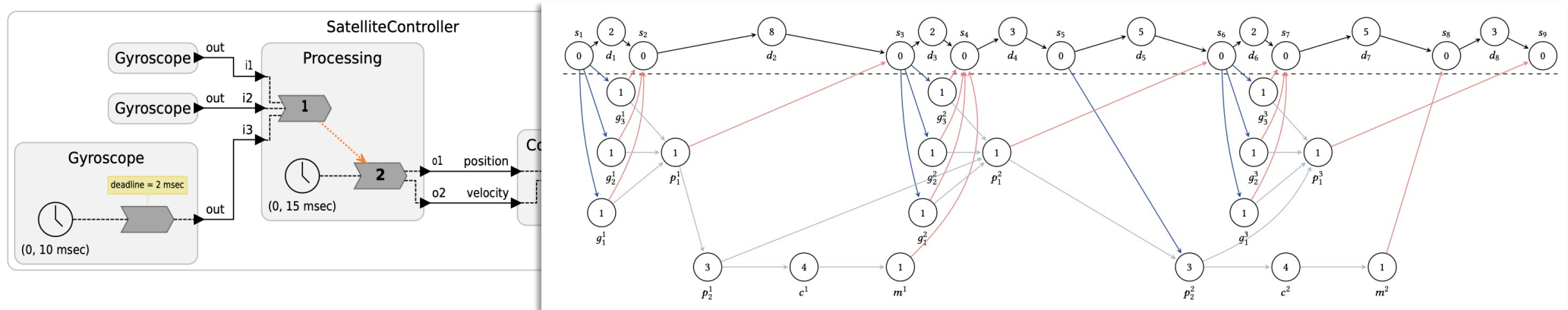
Efficient runtime for dynamic scheduling

1.85x over Akka
and **1.42x over CAF**



Quasi-static scheduling of LF programs

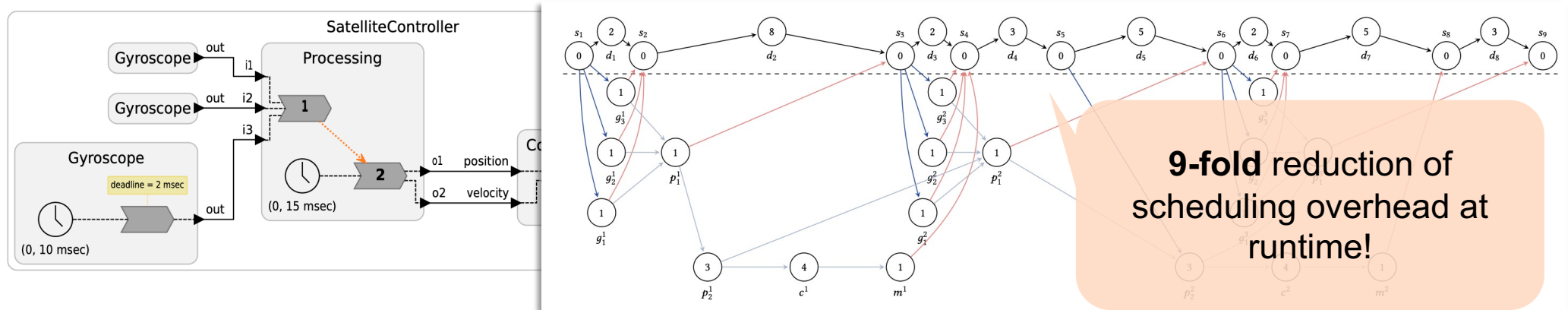
- Capture dynamic/reactive behavior in a State-space Finite State Automaton (SSFA)



Shaokai Lin, et al. "Quasi-Static Scheduling for Deterministic Timed Concurrent Models on Multi-Core Hardware", In ACM Transactions on Embedded Computing Systems (TECS). Special issue, International Conference on Embedded Software (EMSOFT'25), Association for Computing Machinery, New York, NY, USA, Sep 2025..

Quasi-static scheduling of LF programs

- Capture dynamic/reactive behavior in a State-space Finite State Automaton (SSFA)



Shaokai Lin, et al. "Quasi-Static Scheduling for Deterministic Timed Concurrent Models on Multi-Core Hardware", In ACM Transactions on Embedded Computing Systems (TECS). Special issue, International Conference on Embedded Software (EMSOFT'25), Association for Computing Machinery, New York, NY, USA, Sep 2025..

Advertising a spin-off: Check it out!



Marten Lohstroh

CO-FOUNDER & CHIEF EXECUTIVE OFFICER



Christian Menard

CO-FOUNDER & PRINCIPAL ARCHITECT



<https://www.xronos.com/>

Build **Intelligent Systems** with Xronos

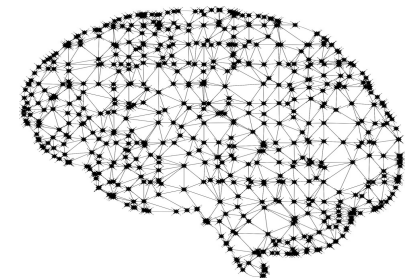
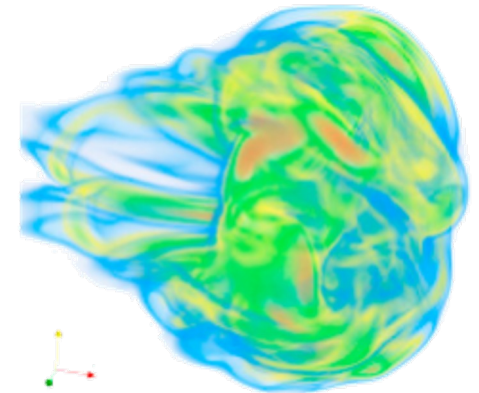
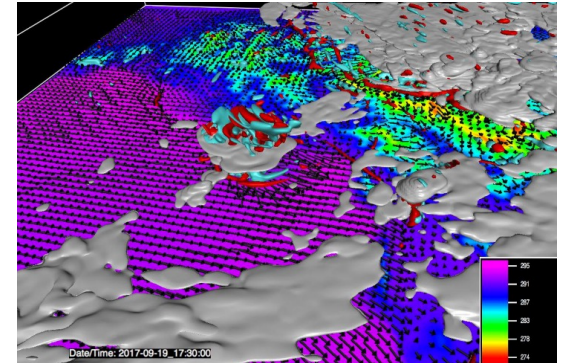
From prototype to production, our ready-to-deploy software infrastructure powers scalable and safe AI-driven robotics and mobility solutions.

BOOK A DEMO →

XRONOS AI ✨

High-level abstractions and MLIR

(for the “nodes”)



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

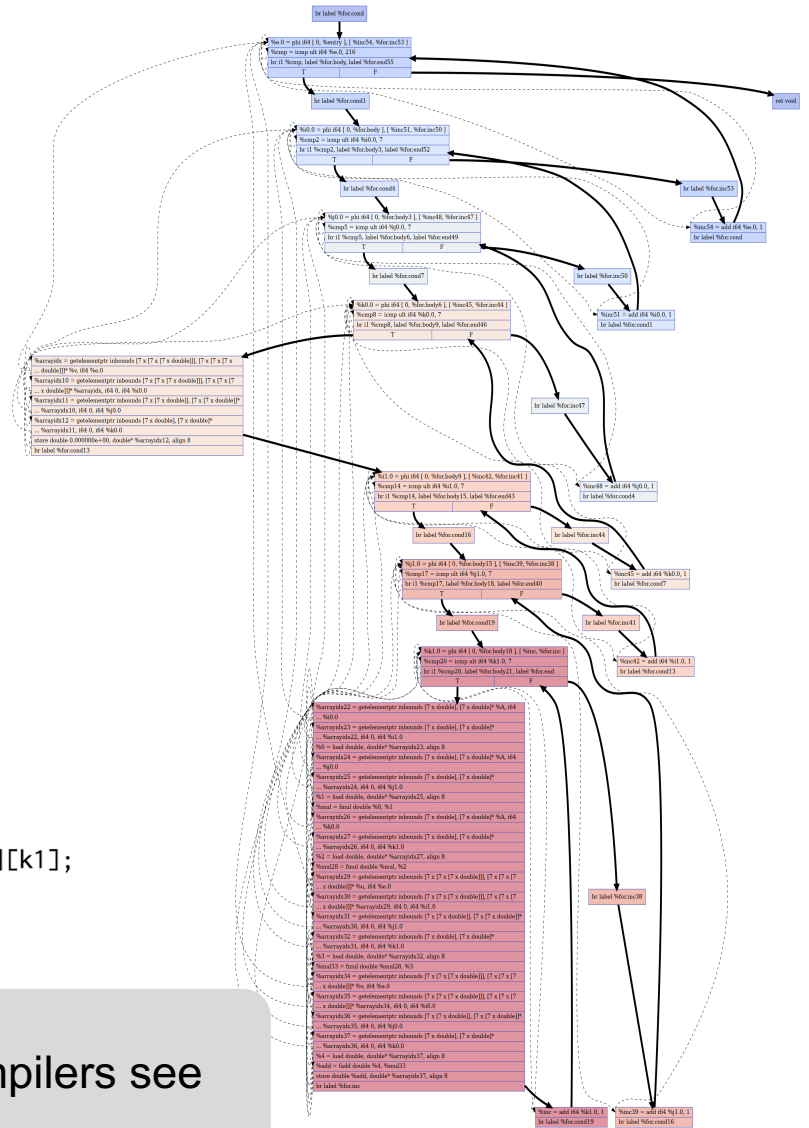
What we (naively) code

```

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++) {
13                        for(int j1 = 0; j1 < 7; j1++) {
14                            for(int k1 = 0; k1 < 7; k1++) {
15                                v[e][i0][j0][k0] += A[i0][i1]
16                                                            * A[j0][j1]
17                                                            * A[k0][k1]
18                                                            * u[e][i1][j1][k1];
19                            } } } } }
20                } /* end of element loop */
21            }
22        }
23    }

```

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

```

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++) {
13                        for(int j1 = 0; j1 < 7; j1++) {
14                            for(int k1 = 0; k1 < 7; k1++) {
15                                v[e][i0][j0][k0] += A[i0][i1]
16                                                         * A[j0][j1]
17                                                         * A[k0][k1]
18                                                         * u[e][i1][j1][k1];
19                            } } } } }
20                } /* end of element loop */
21            }

```

100X

```

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     #pragma omp for
8     for (int e = 0; e < 216; e++) {
9         double t6[7][7][7];
10        /* 1st contraction: */
11        #pragma simd
12        for (int i0 = 0; i0 < 7; i0++) {
13            for (int i1 = 0; i1 < 7; i1++) {
14                /* #pragma simd */
15                for (int i2 = 0; i2 < 7; i2++) {
16                    double t8 = 0.0;
17                    for (int i3 = 0; i3 < 7; i3++)
18                        t8 += A[i0][i3] * u[e][i1][i2][i3];
19                    t6[i0][i1][i2] = t8;
20                } } } /* end of 1st contraction */
21        double t7[7][7][7];
22        /* 2nd contraction: */
23        #pragma simd
24        for (int i4 = 0; i4 < 7; i4++) {
25            for (int i5 = 0; i5 < 7; i5++) {
26                /* #pragma simd */
27                for (int i6 = 0; i6 < 7; i6++) {
28                    double t9 = 0.0;
29                    for (int i7 = 0; i7 < 7; i7++)
30                        t9 += A[i4][i7] * t6[i5][i6][i7];
31                    t7[i4][i5][i6] = t9;
32                } } } /* end of 2nd contraction */
33        /* 3rd contraction: */
34        #pragma simd
35        for (int i8 = 0; i8 < 7; i8++) {
36            for (int i9 = 0; i9 < 7; i9++) {
37                /* #pragma simd */
38                for (int i10 = 0; i10 < 7; i10++) {
39                    double t10 = 0.0;
40                    for (int i11 = 0; i11 < 7; i11++)
41                        t10 += A[i8][i11] * t7[i9][i10][i11];
42                    v[e][i0][i9][i10] = t10;
43                } } } /* end of third contraction */
44            } } } /* end of element loop */

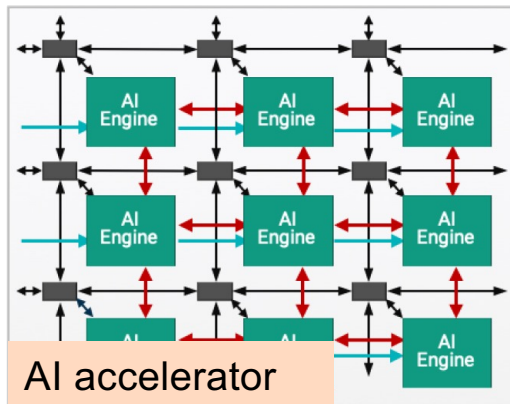
```

What performance experts code

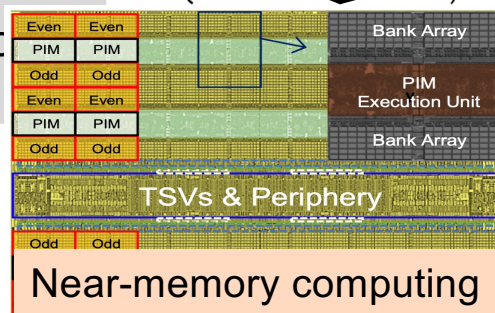
Abstractions and compilation

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

What we want

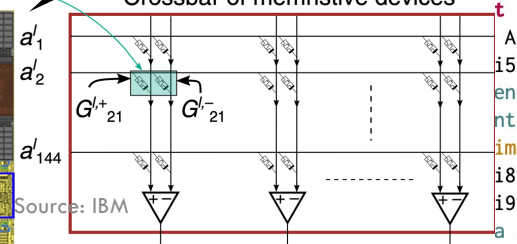


```
1 void cfd_kernel(
2   double A[restrict 7][7],
3   double u[restrict 216][7][7][7],
4   double v[restrict 216][7][7][7])
5 {
6   /* element loop: */
7   for(int e = 0; e < 216; e++) {
8     for(int i0 = 0; i0 < 7; i0++) {
9       for(int j0 = 0; j0 < 7; j0++) {
10        for(int k0 = 0; k0 < 7; k0++) {
11          v[e][i0][j0][k0] = 0.0;
12          for(int i1 = 0; i1 < 7; i1++) {
13            for(int j1 = 0; j1 < 7; j1++) {
14              for(int k1 = 0; k1 < 7; k1++) {
15                t8 += A[i0][i1][j1][k1] * u[e][i1][j1][k1];
16              }
17            }
18          }
19          t6[i0][j0][k0] = t8;
20        }
21      }
22    }
23    /* 2nd contraction: */
24    #pragma simd
25    for (int i4 = 0; i4 < 7; i4++) {
26      for (int i5 = 0; i5 < 7; i5++) {
27        for (int i6 = 0; i6 < 7; i6++) {
28          t9 = 0.0;
29          for (int i7 = 0; i7 < 7; i7++) {
30            for (int i8 = 0; i8 < 7; i8++) {
31              for (int i9 = 0; i9 < 7; i9++) {
32                t10 += A[i8][i9][i7][i6] * t7[i9][i8][i7][i5];
33              }
34            }
35          }
36          v[e][i4][i5][i6] = t10;
37        }
38      }
39    }
40  }
```



100X

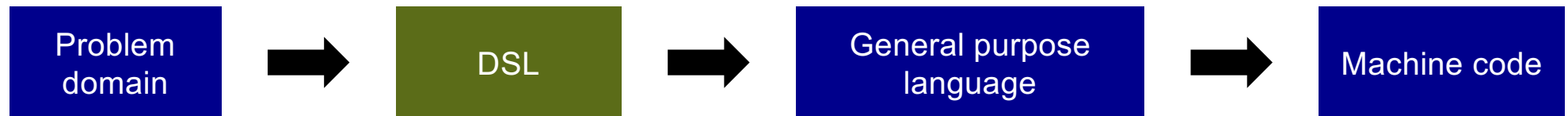
Crossbar of memristive devices



Emerging memories + in-memory computing

```
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   #pragma omp for
8   for (int e = 0; e < 216; e++) {
9     double t6[7][7][7];
10    /* 1st contraction: */
11    #pragma simd
12    for (int i0 = 0; i0 < 7; i0++) {
13      for (int i1 = 0; i1 < 7; i1++) {
14        /* #pragma simd */
15        for (int i2 = 0; i2 < 7; i2++) {
16          double t8 = 0.0;
17          for (int i3 = 0; i3 < 7; i3++) {
18            t8 += A[i0][i1][i2][i3] * u[e][i1][i2][i3];
19          }
20          t6[i0][i1][i2] = t8;
21        }
22      }
23    } /* end of 1st contraction */
24    double t7[7][7][7];
25    /* 2nd contraction: */
26    #pragma simd
27    for (int i4 = 0; i4 < 7; i4++) {
28      for (int i5 = 0; i5 < 7; i5++) {
29        /* #pragma simd */
30        for (int i6 = 0; i6 < 7; i6++) {
31          double t9 = 0.0;
32          for (int i7 = 0; i7 < 7; i7++) {
33            for (int i8 = 0; i8 < 7; i8++) {
34              for (int i9 = 0; i9 < 7; i9++) {
35                t10 += A[i8][i9][i7][i6] * t7[i9][i8][i7][i5];
36              }
37            }
38          }
39          v[e][i4][i5][i6] = t10;
40        }
41      }
42    } /* end of third contraction */
43  } /* end of element loop */
```

Enter domain-specific languages/abstractions



- A **DSL** is a programming language or executable specification language that offers, through **appropriate notations and abstractions**, expressive power focused on, and usually restricted to, a particular problem domain

A. Van Deursen, et al. "Domain-specific languages: An annotated bibliography." ACM Sigplan Notices 35.6 (2000)

- Limits to what can be reconstructed from lower-level implementations
- Limited expressiveness → more powerful analysis and optimizations
- Plenty of successful examples: ML frameworks, Halide, Spiral, Firedrak, Lift, ...

Examples (1): Particle-mesh simulations

Vortex ring

- Particle-mesh simulations in computational biology

- Discrete/continuous
- Deterministic/stochastic

time loop

start: 0 stop: 1000

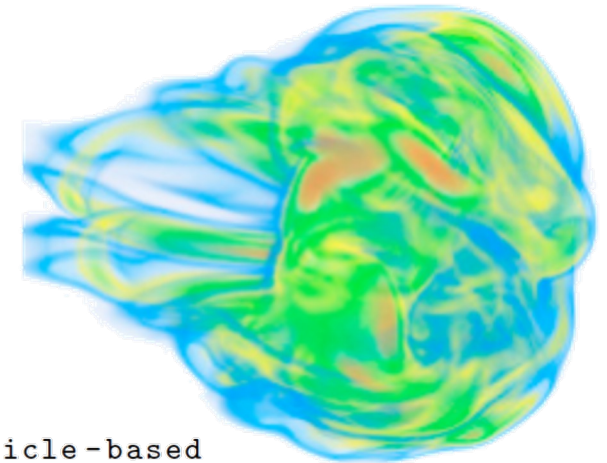
temporal method: explicit_euler

spatial method: DC-PSE

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

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

P. Incardona, et al "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers", Computer Physics Communications, 2019



```
type of simulation: particle-based
Particle sets:
  name particles
  properties
    velocity d:3
    force d:3
  Define interact in particles with self as
    p_force->force += 24.0 * 2.0 * sigma/r^7
    diff(p_force,q_force)
  Define evolve in particles with self as p_
    p_velocity = p_velocity + 0.5 * dt * p_force
```

S. Karol, et al. "A Domain-Specific Language and Editor for Parallel Particle Methods", In ACM TOMS'18, vol. 44, no. 3, pp. 32, Mar 2018.
N Khouzami, et al., "The OpenPME Problem Solving Environment for Numerical Simulations", In ICCS'21 pp. 614–627, Jun 2021.

Examples (1): Codegen and optimization

```
rhs -> vortex_stretching_m =
(vorticity_mesh -> vorticity_m.v)
velocity_mesh -> velocity_m +
nu * Δ vorticity_mesh -> vorticity_m
```

```
for mesh node decl <no type> loopNodeM in rhs
loopNodeM -> vortex_stretching_m [ 0 ] = ...
loopNodeM -> vortex_stretching_m [ 1 ] = ...
loopNodeM -> vortex_stretching_m [ 2 ] = ...
```

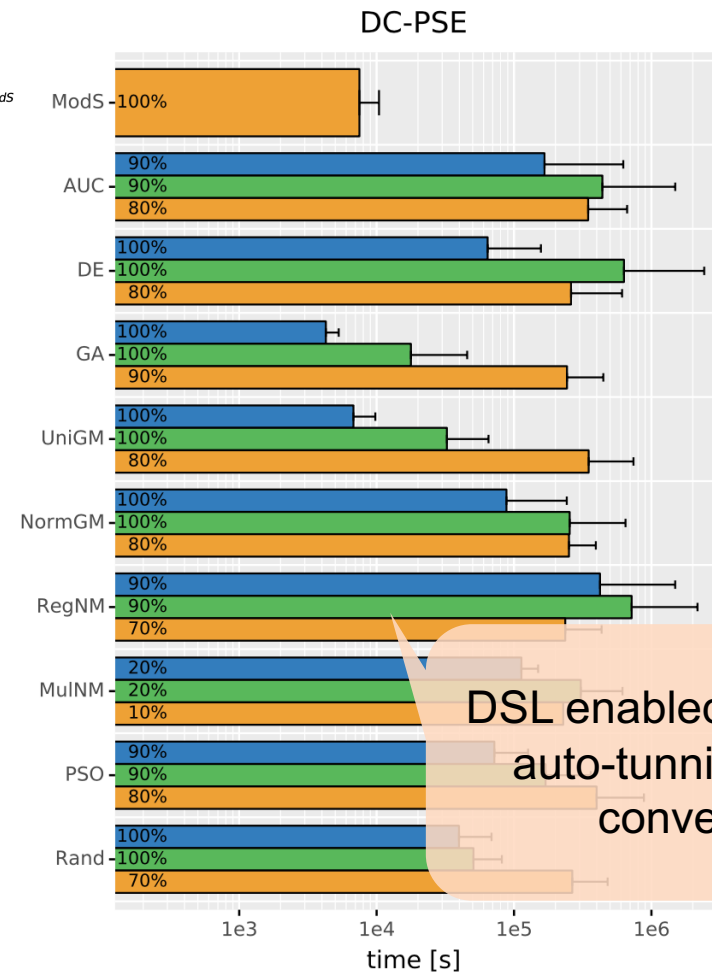
IR

```
while (mloop_iterator_h5a0.isNext())
```

```
{
  g_dwp.template get<rhs>(key)[x]=
  fac1*(g_vort.template get<vorticity>(key.move(x,1))[x]+
  g_vort.template get<vorticity>(key.move(x,-1))[x])+
  fac2*(g_vort.template get<vorticity>(key.move(y,1))[x]+
  g_vort.template get<vorticity>(key.move(y,-1))[x])+
  fac3*(g_vort.template get<vorticity>(key.move(z,1))[x]+
  g_vort.template get<vorticity>(key.move(z,-1))[x]) -
  2.0f*(fac1+fac2+fac3)*
  g_vort.template get<vorticity>(key)[x]+
  fac4*g_vort.template get<vorticity>(key)[x]*
  (g_vel.template get<velocity>(key.move(x,1))[x]-
  g_vel.template get<velocity>(key.move(x,-1))[x])+
  fac5*g_vort.template get<vorticity>(key)[y]*
  (g_vel.template get<velocity>(key.move(y,1))[x]-
  g_vel.template get<velocity>(key.move(y,-1))[x])+
  fac6*g_vort.template get<vorticity>(key)[z]*
  (g_vel.template get<velocity>(key.move(z,1))[x]-
  g_vel.template get<velocity>(key.move(z,-1))[x]);
}
g_dwp.template get<rhs>(key)[y]=
g_dwp.template get<rhs>(key)[z]=
```

tuning time limit

1 · t_{ModS} 8 · t_{ModS} 16 · t_{ModS}



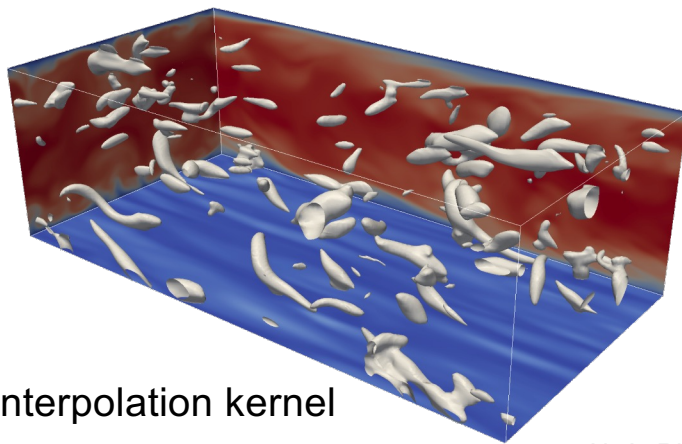
DSL enabled model-aware
auto-tuning for faster
convergence!

N Khouzami, et al., "The OpenPME Problem Solving Environment for Numerical Simulations", In ICCS'21 pp. 614–627, Jun 2021

N Khouzami, et al., "Model-based autotuning of discretization methods in numerical simulations of partial differential equations", In Journal of Computational Science 57, 2022

Examples (2): Tensors expressions (CFD)

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



Interpolation kernel

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

```
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);
```

Fortran and C++ integration

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. "TeLL: a type-safe imperative Tensor Intermediate Language", ARRAY'19, pp. 57-68

Example (2): Closing the performance gap

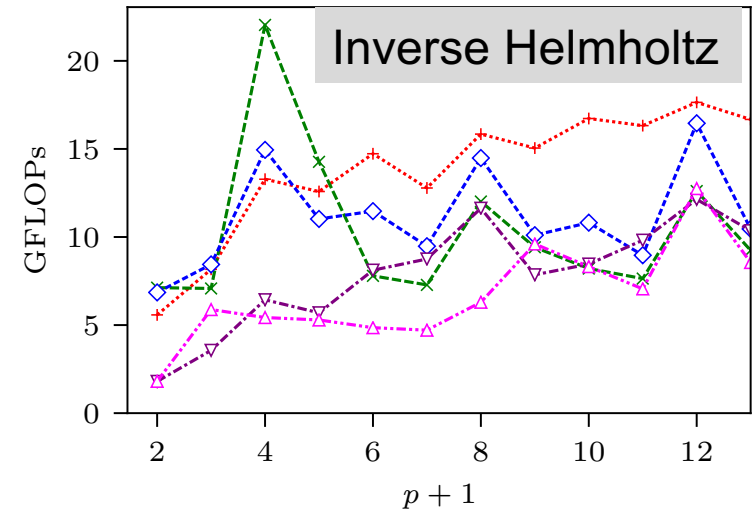
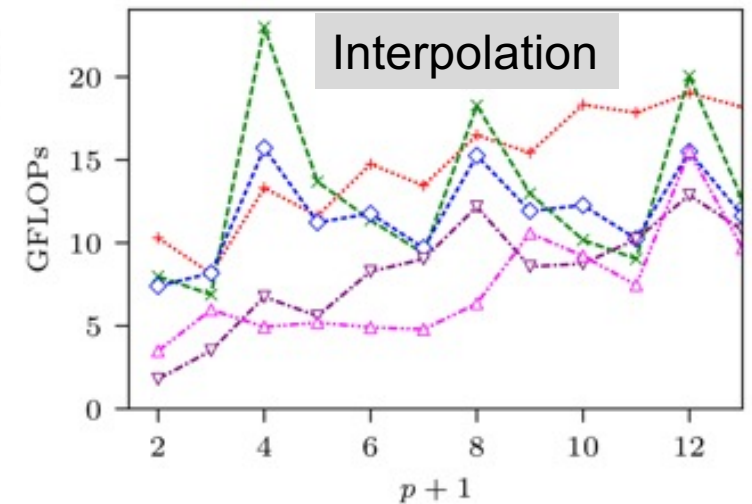
□ Not really optimization magic

- Leverage expert knowledge
- Algebraic identities

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

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

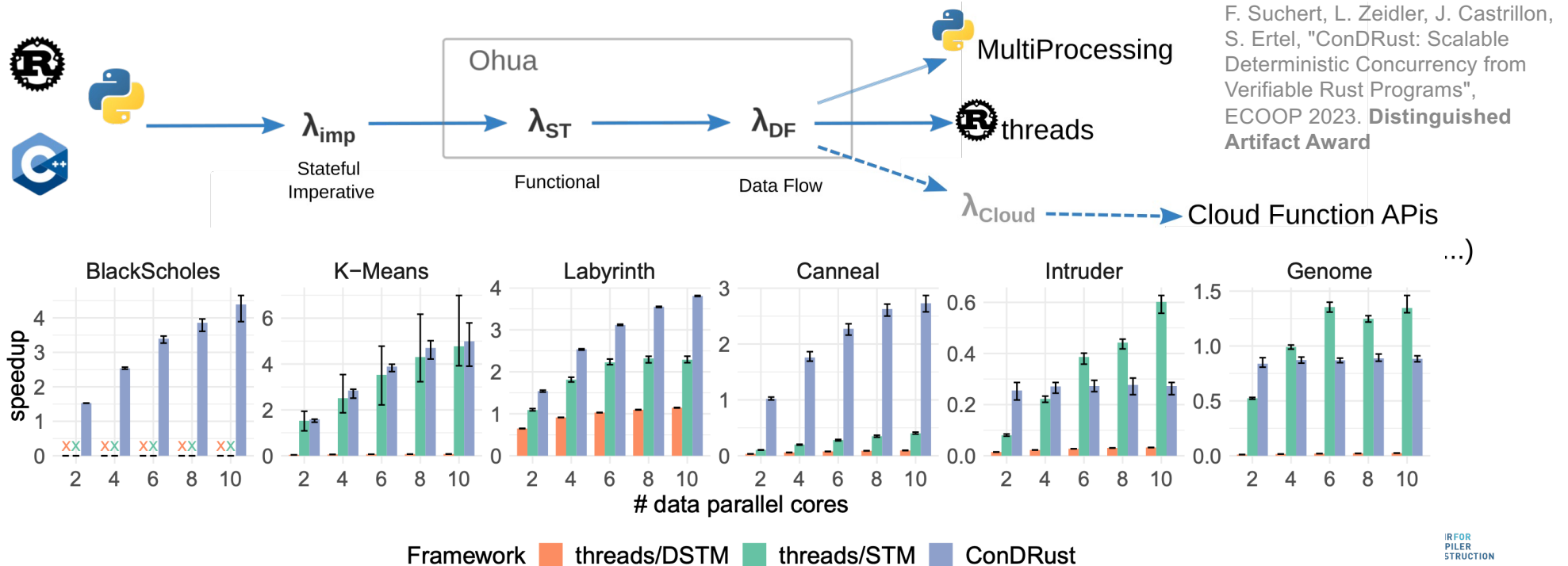
$$v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))$$



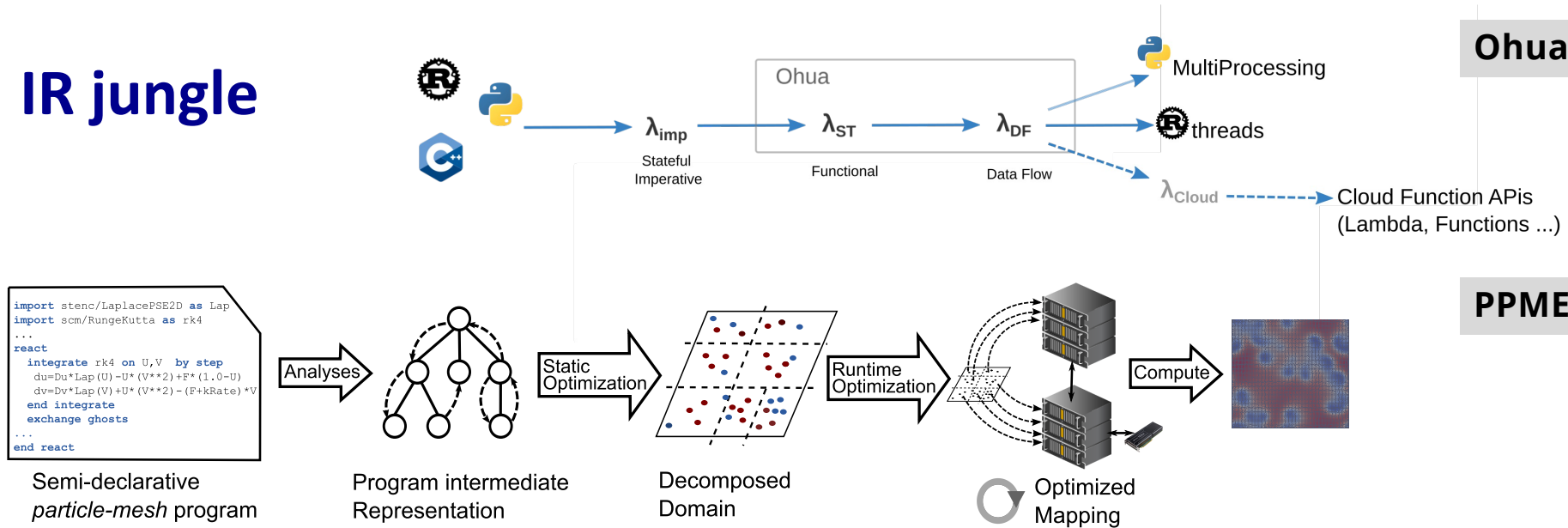
N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL'18.
 A. Susungi, et al., "Meta-programming for Cross-Domain Tensor Optimizations", GPCE'18 pp. 79-92.

Example (3): Implicit dataflow

- Functional programming abstraction to implicitly create dataflow graphs
- Faster than deterministic threads, comparable with unsafe code

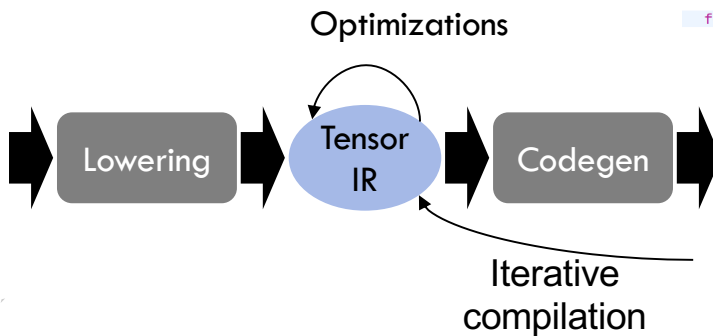


IR jungle



```
source =
type matrix      : [mp np]      &
type tensorIN    : [np np np ne] &
type tensorOUT   : [mp mp mp me] &
&
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
```

Fortran embedding



```
for (unsigned i0 = 0; i0 < 1000; i0++) {
  double t6[18];
  for (unsigned i3 = 0; i3 < 3; i3++) {
    for (unsigned i2 = 0; i2 < 3; i2++) {
      for (unsigned i1 = 0; i1 < 2; i1++) {
        t6[(i1 + 2*(i2 + 3*(i3)))] = 0.0;
        for (unsigned i4_contr = 0; i4_contr < 3; i4_contr++) {
          t6[(i1 + 2*(i2 + 3*(i3)))] += A[(i1 + 2*(i4_contr))]
            * u[(i2 + 3*(i3 + 3*(i4_contr + 3*(i0))))];
        }
      }
    }
  }
  double t7[12];
  for (unsigned i7 = 0; i7 < 3; i7++) {
    for (unsigned i6 = 0; i6 < 2; i6++) {
      for (unsigned i5 = 0; i5 < 2; i5++) {
        t7[(i5 + 2*(i6 + 2*(i7)))] = 0.0;
        for (unsigned i8_contr = 0; i8_contr < 3; i8_contr++) {
          t7[(i5 + 2*(i6 + 2*(i7)))] += A[(i5 + 2*(i8_contr))]
            * t6[(i6 + 2*(i7 + 3*(i8_contr))];
        }
      }
    }
  }
  double t8[1];
  double t9[1];
  for (unsigned i11 = 0; i11 < 2; i11++) {
    for (unsigned i10 = 0; i10 < 2; i10++) {
      t8[0] = t7[(i11 + 2*(i10))];
      t9[0] = t8[0];
    }
  }
}
```

Linkable C code

CFDLang/
TEML/Teil

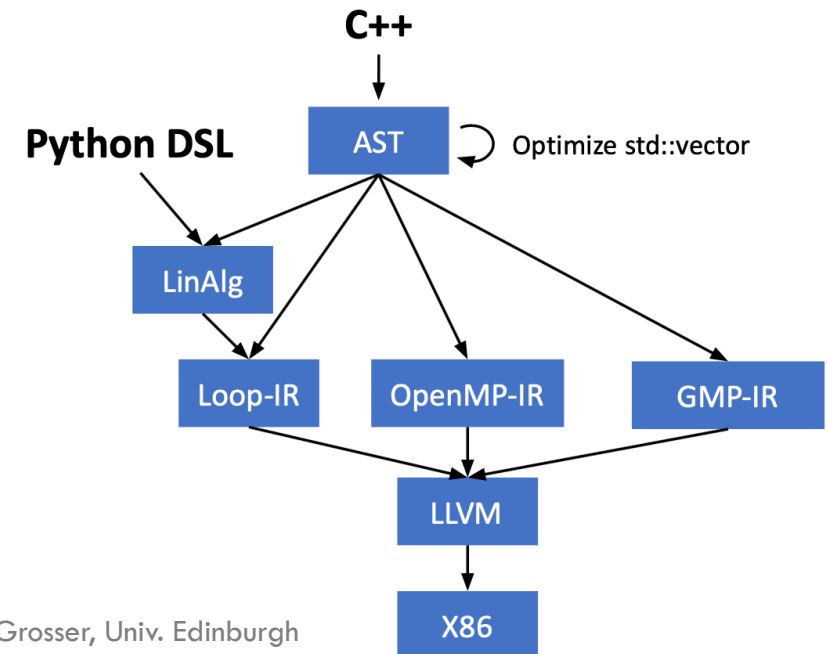
Enter MLIR



- 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.

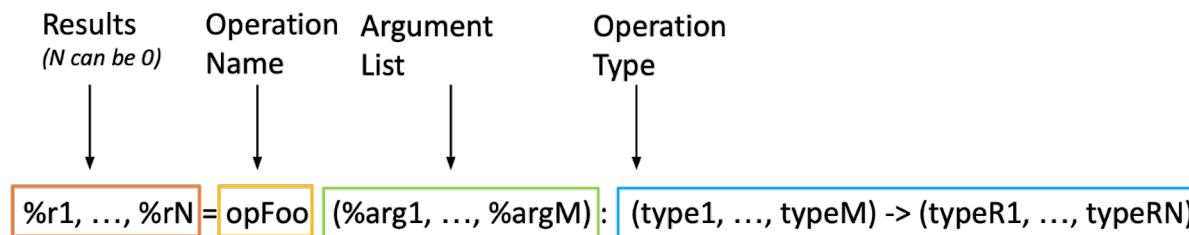
- Not an IR, but an extensible framework
 - to describe intermediate abstractions (called **dialects**),
 - to optimize representations between dialects (**transform**, **lower** or **raise**),
 - that builds on the success of LLVM to build community/infrastructure and reuse ("LLVM-quality" all the way)



Source: T. Grosser, Univ. Edinburgh

MLIR basics

- Similar structure to LLVM instructions, but much more generic and flexible



Source: T. Grosser, Univ. Edinburgh

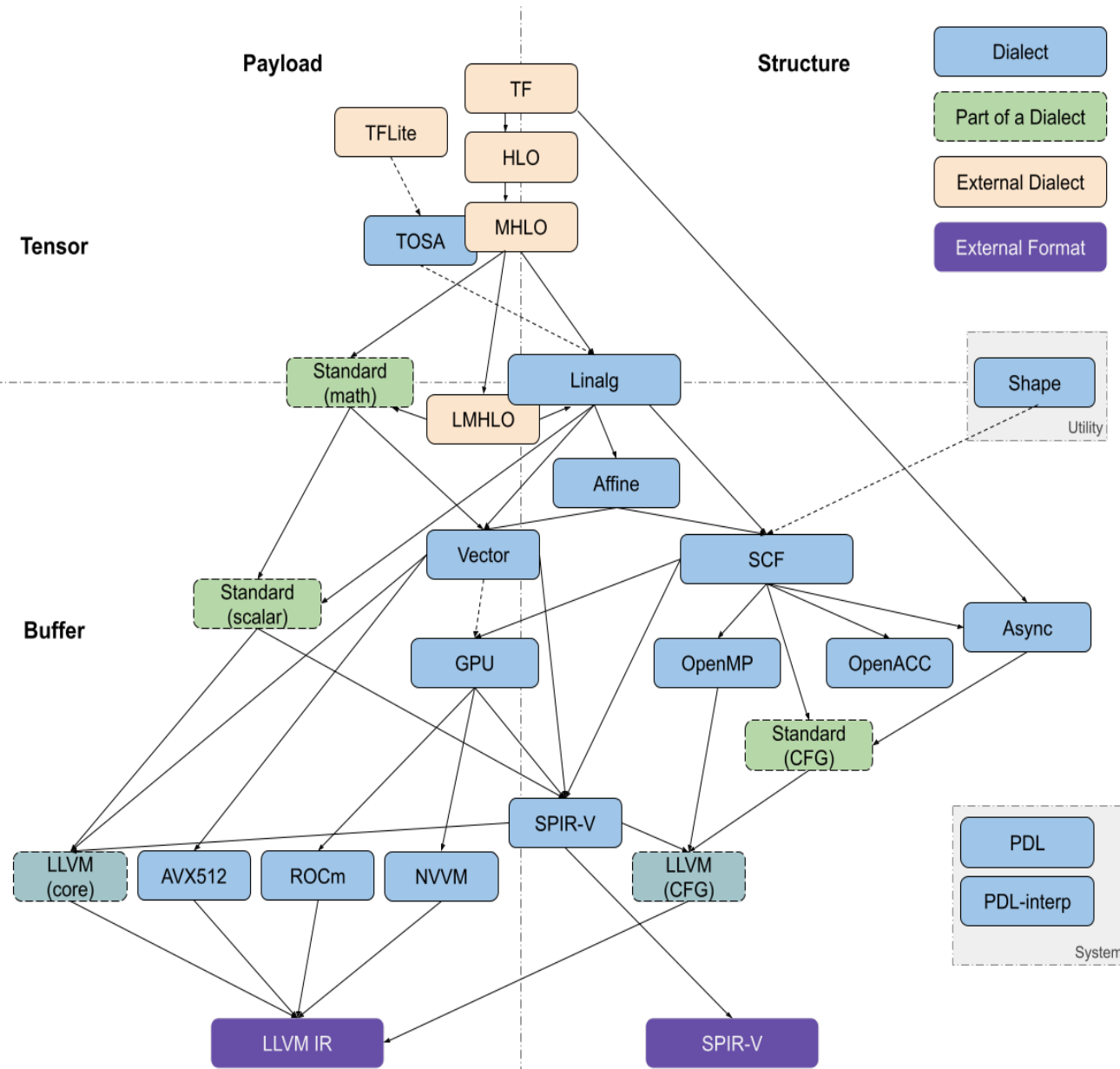
```
%val = get_random() : () -> i64
%sum = add (%val, %val) : (i64, i64) -> i64
print(%sum) : (i64) -> ()
```

- Constructs for “Blocks” (!= basic blocks) and functions
- Blocks take arguments as opposed to Phi nodes
- Dialect: Set of operations (kind of a namespace)

An MLIR language stack

□ Tensors (ML)

- Tensors: Abstract mathematical expressions
- Buffers: Materialization (reuse, alias, ...)



<https://aws1.discourse-cdn.com/free1/uploads/llvm/original/1X/4e51f8ba1dbf21f4dd171342b420178b00d4dbe8.png>

Tensor expressions: CFD and weather models

- More complex expressions: Subscripted subscripts, dependent types, ...

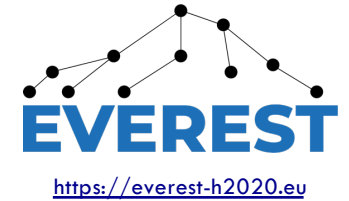
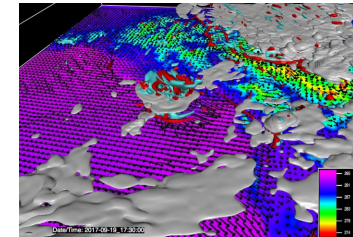
$$\tau_g^M = \sum_{dT} \sum_{dp} \sum_{d\eta} n_{\sigma_x, x, d\eta} \alpha_{\sigma_x, x, dT, dp, d\eta} k_{T+dT, \bar{p}+dp, \bar{\eta}+d\eta, g}$$

```

1  // Interpolation (snippet)
2  let p_prime = (log(p) - C_LOG_MAX_P_REF) / C_DELTA_LOG_P_REF
3  let p_prime = clamp(0, p_prime, 58)
4  let i_strato = select(p < C_P_TROPO, 1, 0)
5  let j_p, f_p = intfrac(p_prime)
6  let i_p      = [j_p, j_p + 1] + i_strato
7
8  // Major contributors
9  let tau_g_M = (n_mix[i_flav[x], x, deta]
10               * f_major[i_flav[x], x, dT, dp, deta]
11               * k_major[i_T[x, dT], i_p[x, dp], i_eta[x, deta], g])

```

VS



```

subroutine taumol16
  integer(kind=4) :: ig, ind0, ind1, inds, indf, is, lay, laycolfr
  real(kind=8) ::
    fac001, fac101, fac011, fac111, colh2o, selffac, selfref, forfac, forfrac, tauray

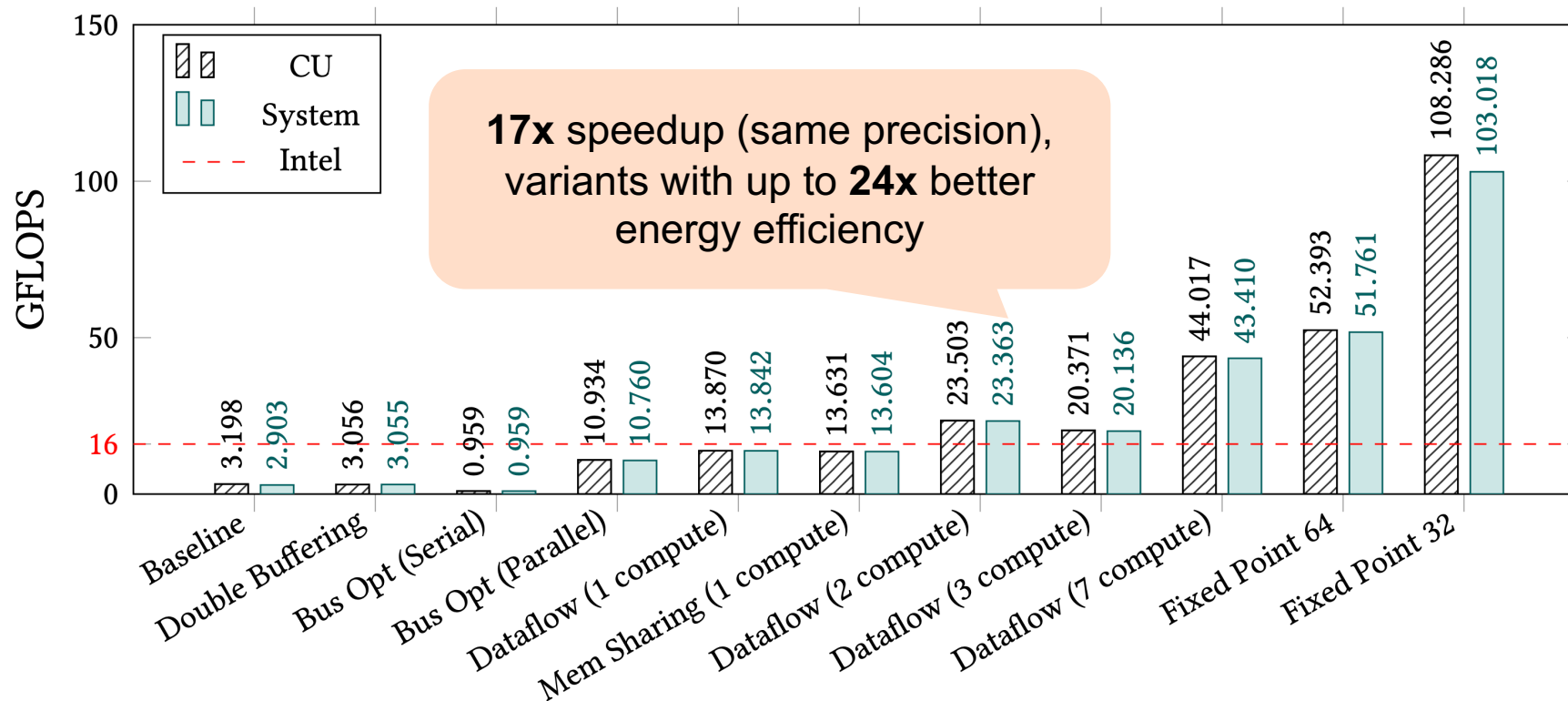
  do lay = 1, laytrc
    speccomb = colh
    specparm = colh
    if (specparm .eq. 8) then
      js = 1 + int(speccomb)
      fs = mod(speccomb, 8)
      fac000 = (1. - fs)
      fac100 = fs
      fac110 = fs
      fac001 = (1. - fs)
      fac011 = (1. - fs)
      fac101 = fs
      fac111 = fs
      ind0 = ((jp(lay)-13)*5 + (jt(lay)-1)*nsnb(16) + 1
      ind1 = ((jp(lay)-12)*5 + (jti(lay)-1)*nsnb(16) + 1
      tauray = colmol(lay) * rayl

      do ig = 1, ng16
        taug(lay, ig) = colch4(lay) * &
          (fac00(lay) * absb(ind0, ig) + &
           fac10(lay) * absb(ind0+1, ig) + &
           fac01(lay) * absb(ind1, ig) + &
           fac11(lay) * absb(ind1+1, ig))
        if (lay .eq. laysolfr) sfluxzen(ig) = sfluxref(ig)
        taur(lay, ig) = tauray
      enddo
    enddo
  enddo
end subroutine taumol16

```

K. Friebe, et al. "Demonstrating a Future for MLIR-native DSL Compilers on a NumPy-like Example", Arxiv. 2026

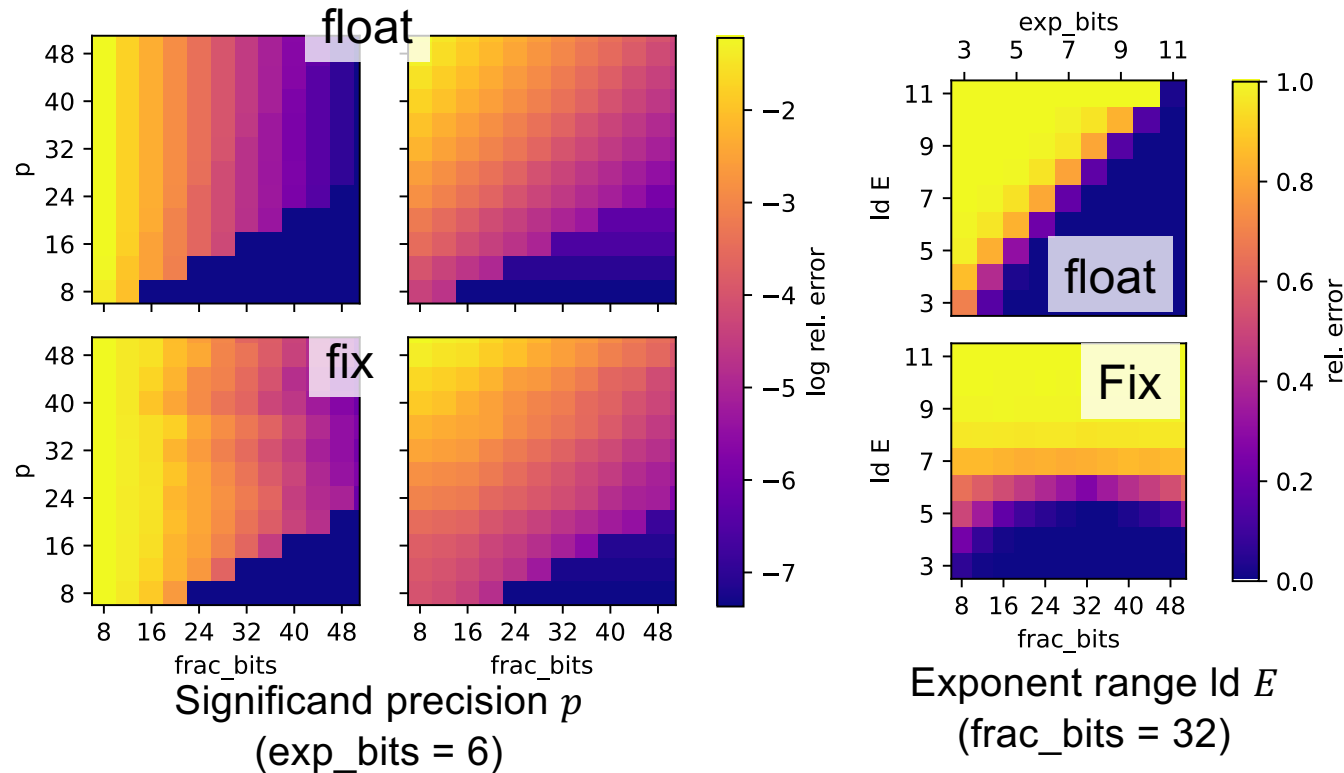
FPGA code generation: HBM FPGA (U280)



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: ACM TRET, Sept. 2022.

Base2: Custom precision analysis

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

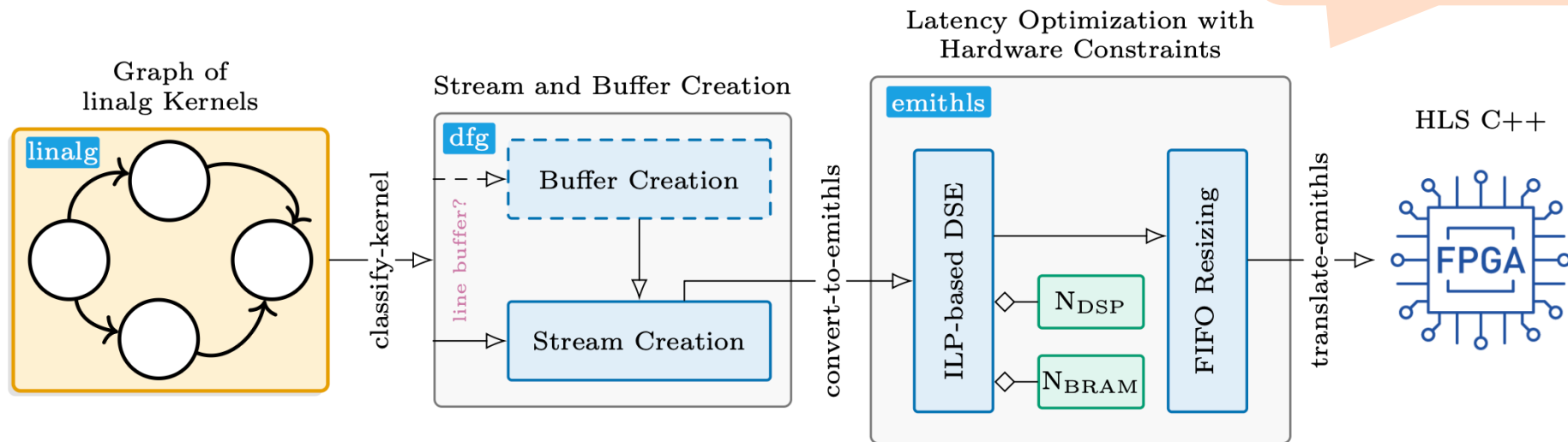


K. F. A. Friebe, J. Bi, J. Castrillon, "BASE2: An IR for Binary Numeral Types", In ACM HEART 2023

LAKSA: Coordination and synthesis

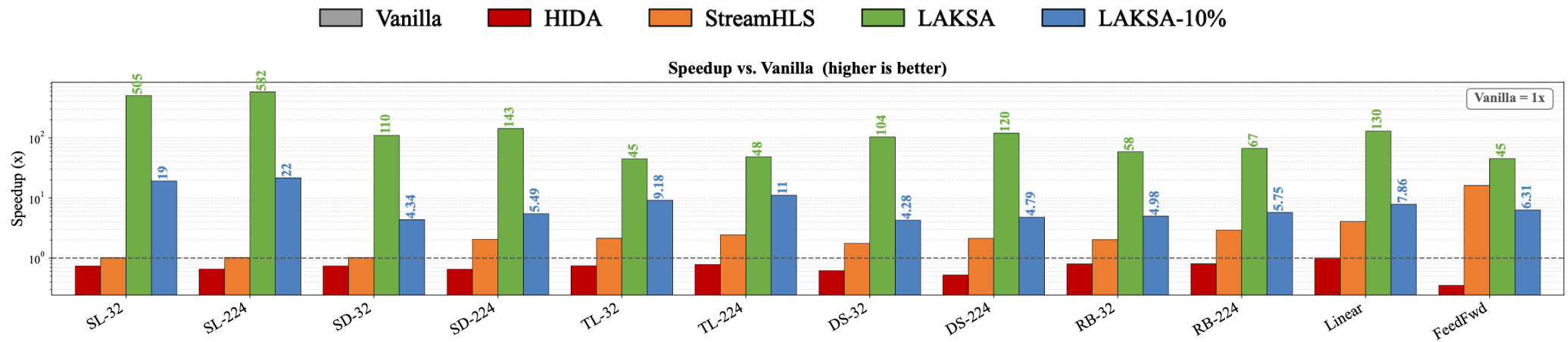
- Improve synthesis to FPGAs at the edge
- Dataflow for coordination & linear algebra for kernels

More on this in the tutorial

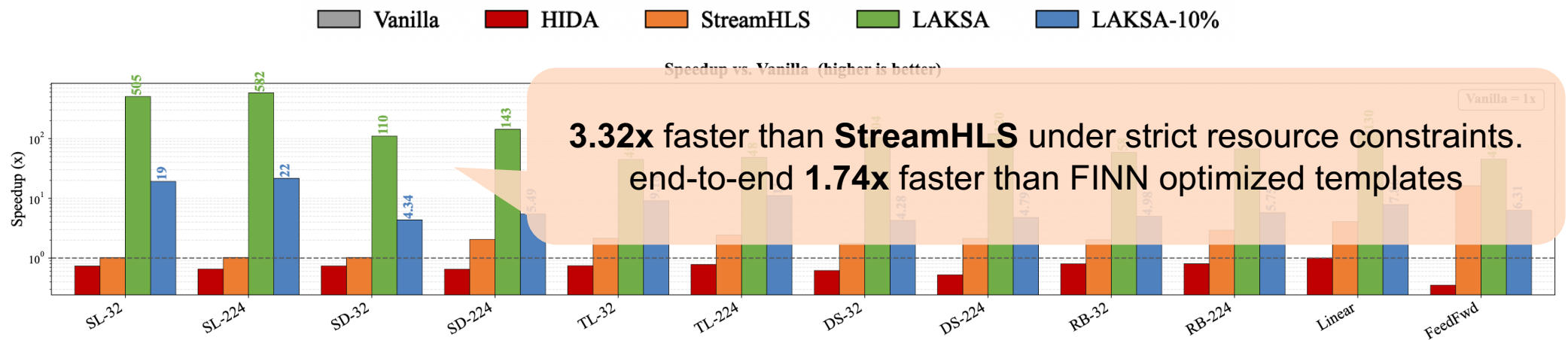


J. Bi, et al. LAKSA: An MLIR-based Higher-Level Synthesis Compiler for Edge Inference. In: IEEE TCAD, Oct. 2026

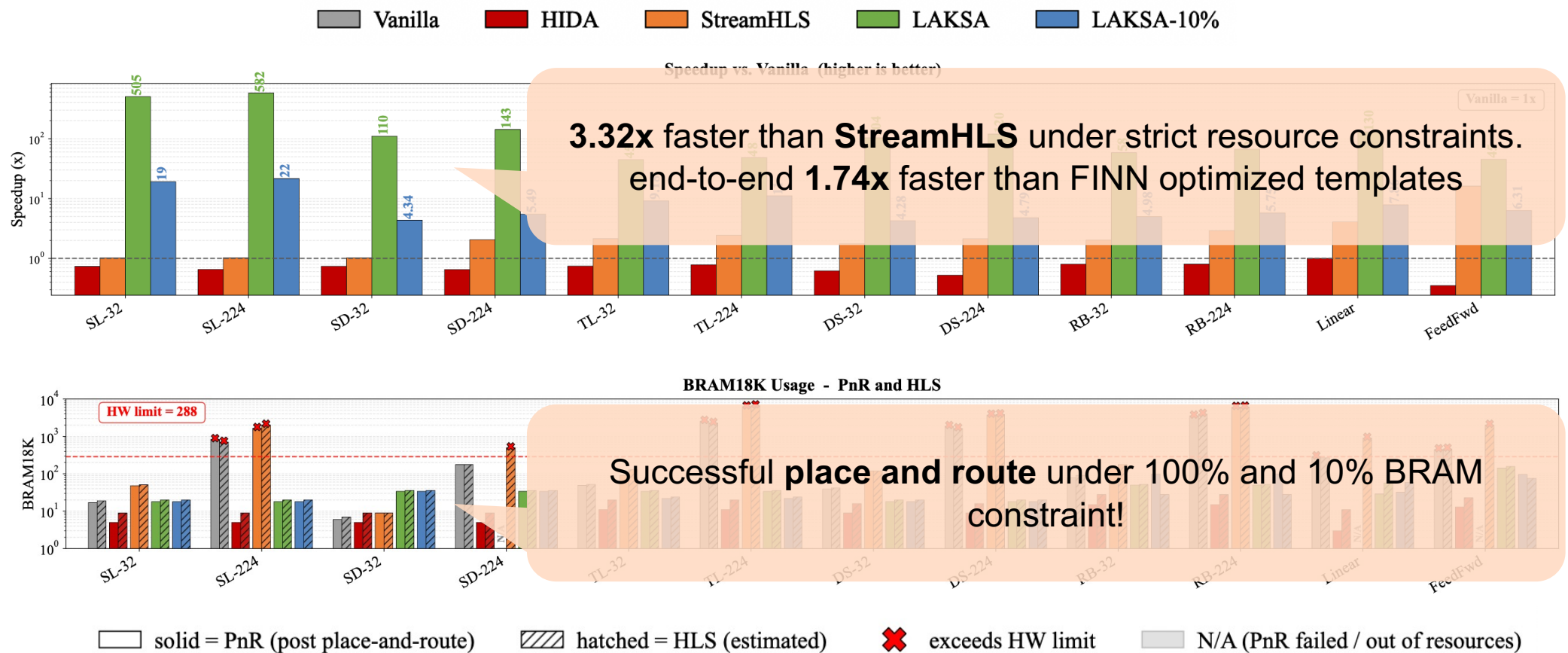
LAKSA: Coordination and synthesis



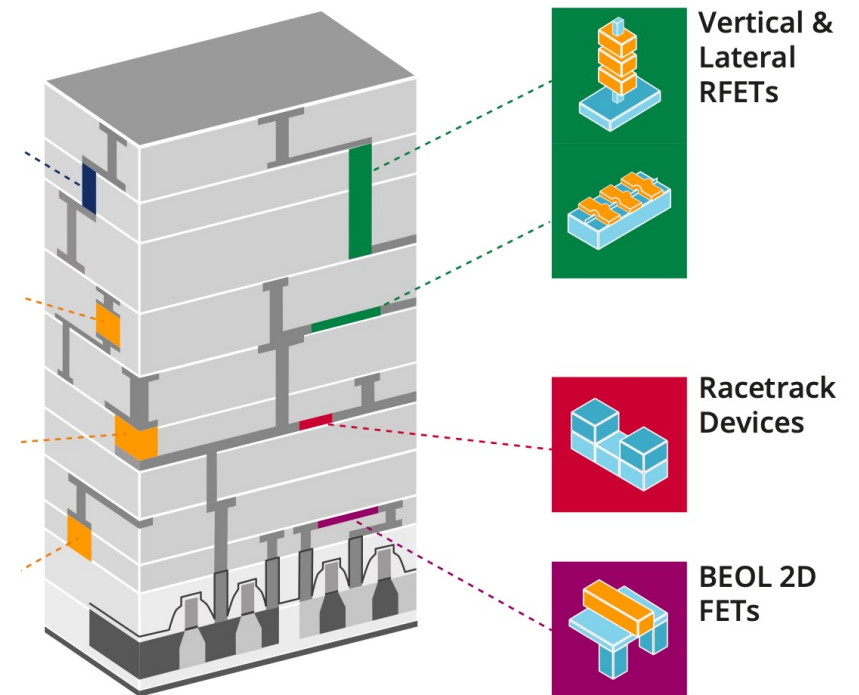
LAKSA: Coordination and synthesis



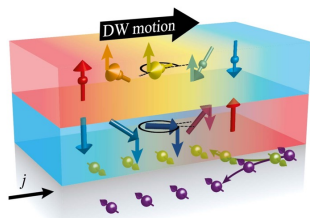
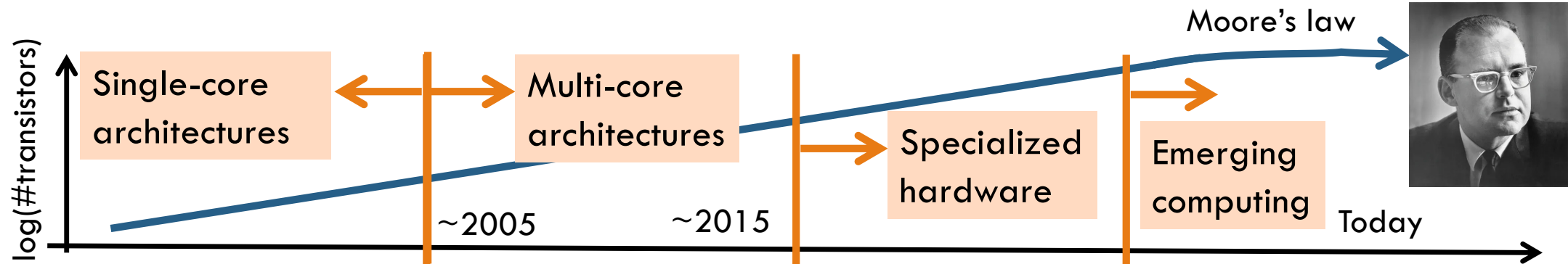
LAKSA: Coordination and synthesis



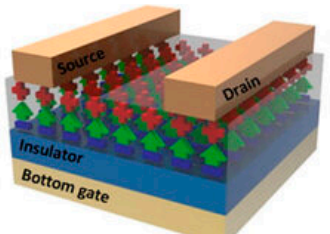
Emerging systems



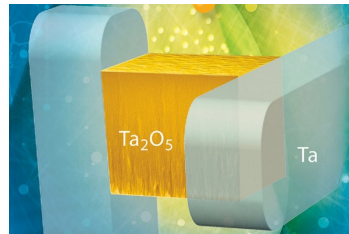
HW evolution: Back to the walls



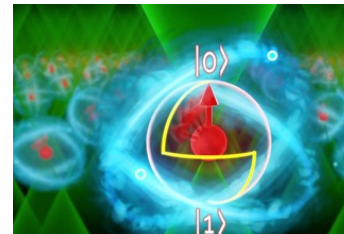
Spintronics



Ferro-electric



Memristive

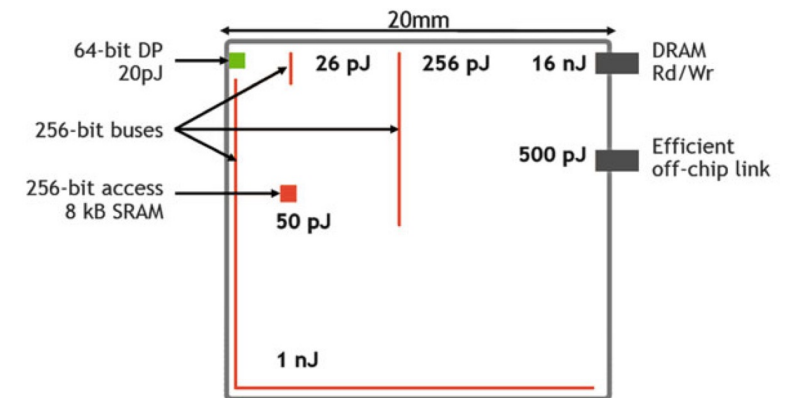


Quantum

Physical limits of classic technology
 → **Extreme heterogeneity!**

Emerging data-centric architectures

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

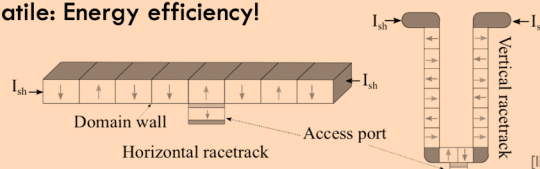


Bartolini, S., "Parallel Programming in Cyber-Physical Systems." Cyber-Physical Systems Security. Springer 2018

Emerging architectures



- ❑ Lots of research on tensor computations on tensor accelerators! e.g., [Kim'19]
- ❑ Recently looking into emerging memory architectures
 - ❑ Hybrid architectures STT/Re-RAM [TVLSI'18, TVLSI'19]
 - ❑ Racetrack memories
- ❑ Racetrack memories: Predicted extreme density at low latency
 - ❑ Multiple bits stored in nano-wires with magnetic domains
 - ❑ Non-volatile: Energy efficiency!



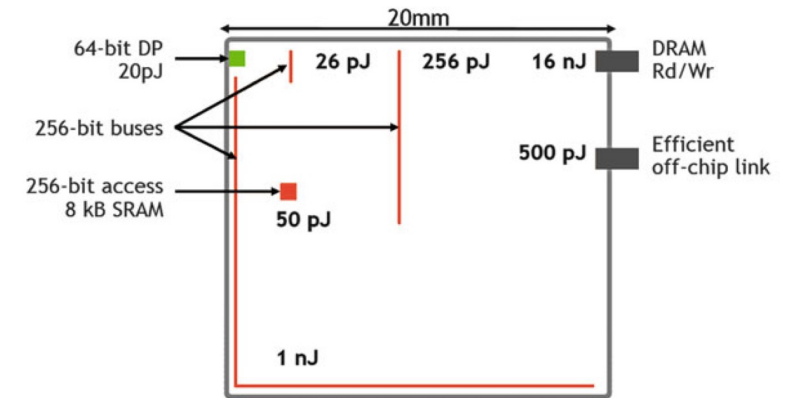
Need to exploit sequentially on top of locality

[IEEE CAL'19]

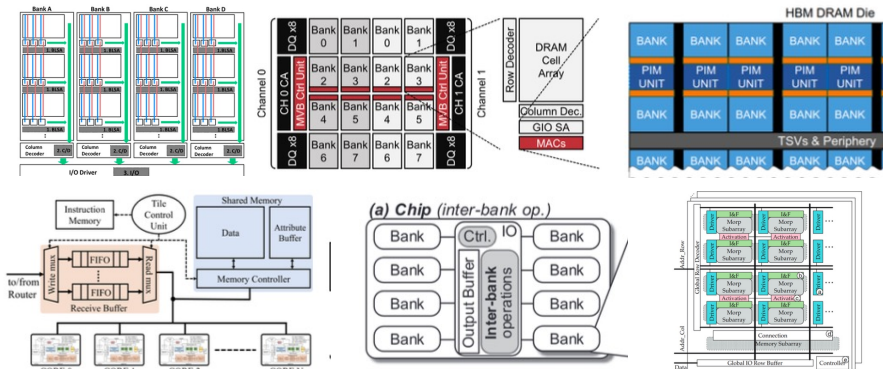


Emerging data-centric architectures

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



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



The Landscape of Compute-near-memory and Compute-in-memory: A Research and Commercial Overview

ASIF ALI KHAN, TU Dresden, Germany

JOÃO PAULO C. DE LIMA, TU Dresden and ScaDS.AI, Germany

HAMID FARZANEH, TU Dresden, Germany

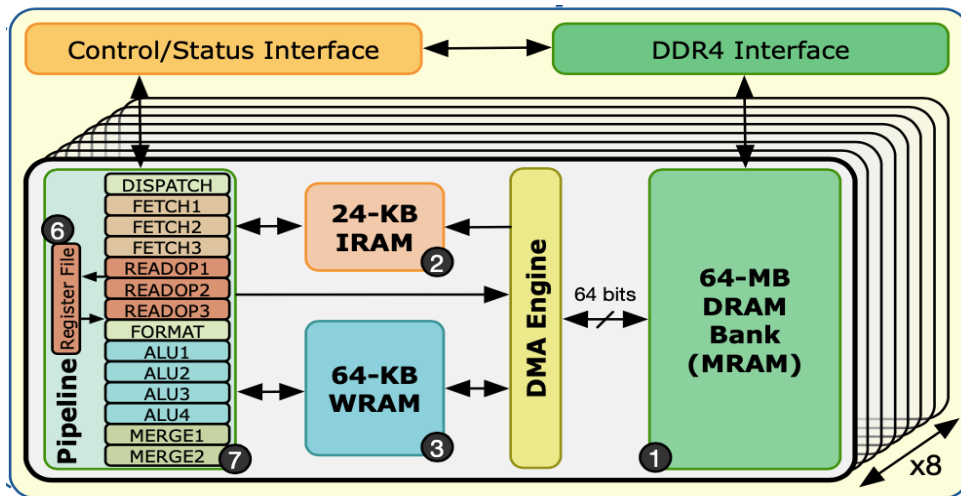
JERONIMO CASTRILLON, TU Dresden and ScaDS.AI, Germany

In today's data-centric world, where data fuels numerous application domains, with machine learning at the

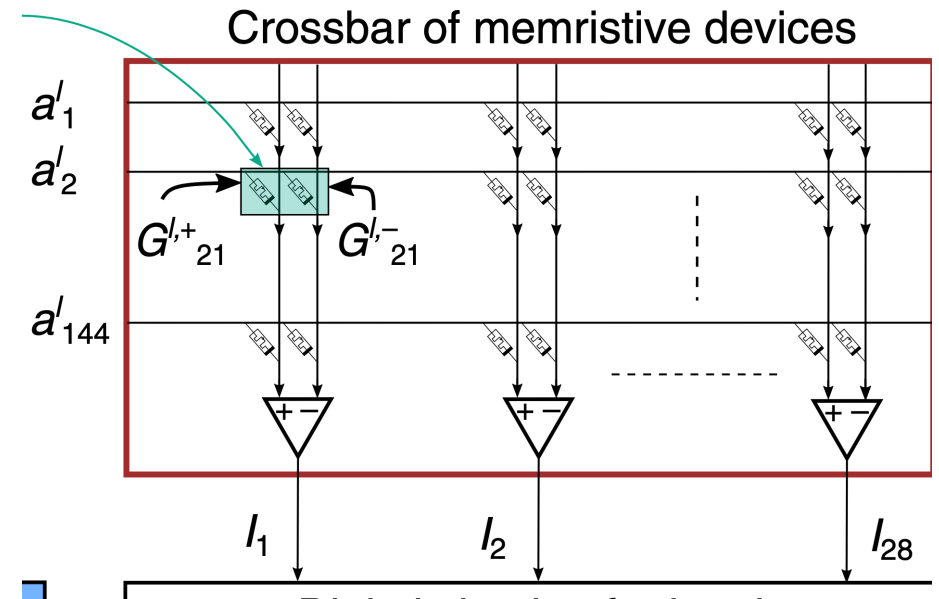
2024

A. Khan, et al "The Landscape of Compute-near-memory and Compute-in-memory: A Research and Commercial Overview." arXiv:2401.1442 (2024)

Near and in-memory computing: Examples



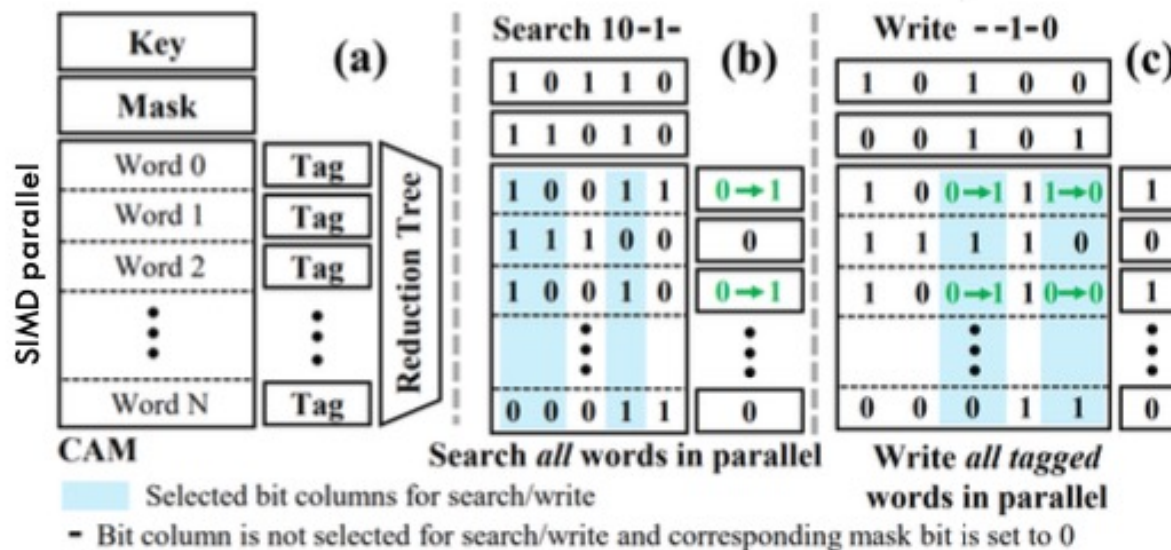
UPMEM: Near DRAM



E.g., IBM Research Zürich

Example: Content-addressable memories (CAMs)

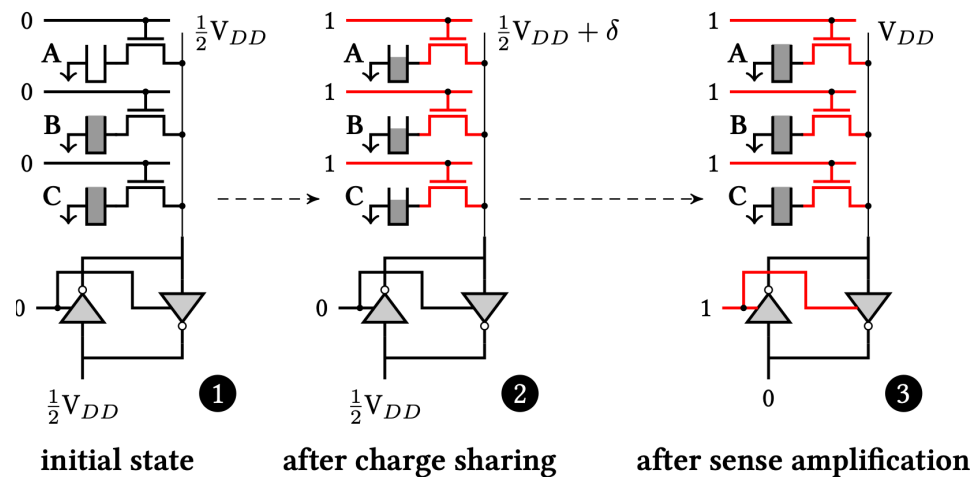
- Emerging Content-addressable memories (CAMs) and associative processors (APs)
- APs: Different programming paradigm based on **search and write**



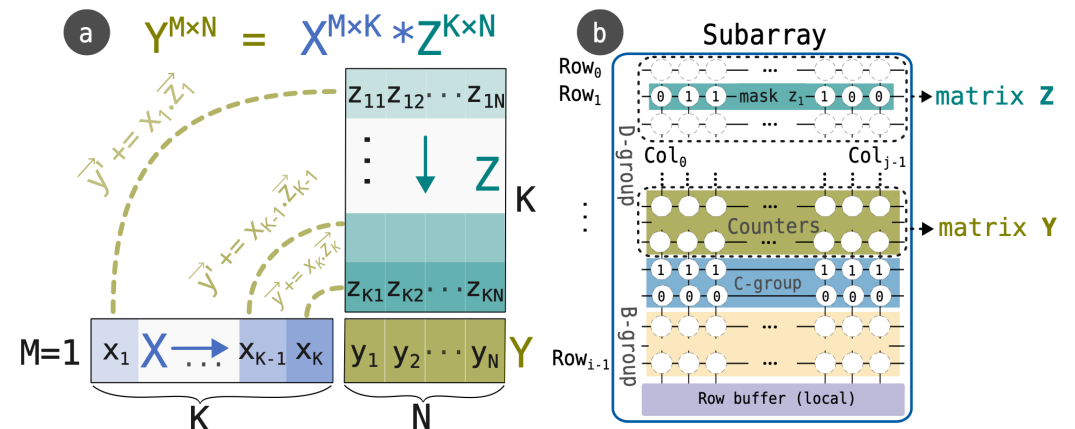
Y. Zha, ISCA'20

In-DRAM computing

- Multiple approaches leverage commercial DRAM for bulk-bitwise CIM
- Bulk-bitwise operations, in a bit-serial, world-parallel manner

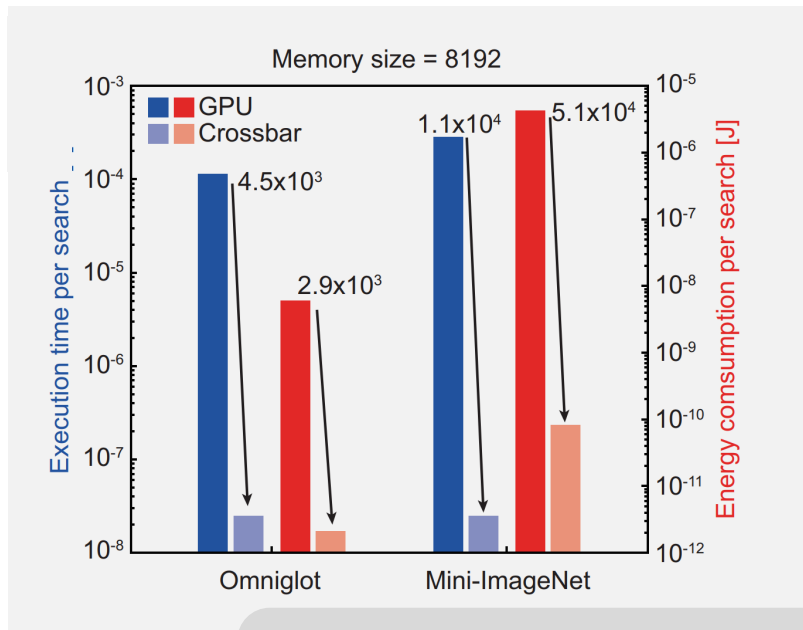


Sishadri et al, MICRO, 2017

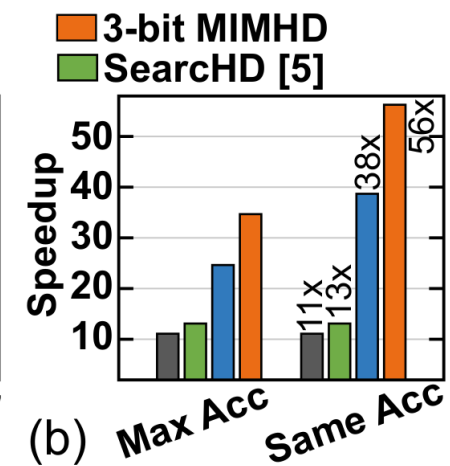
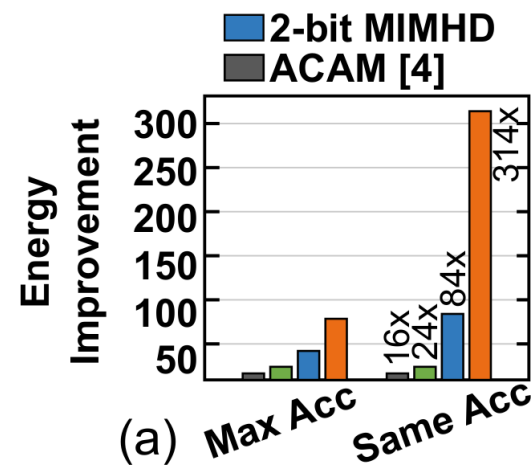


J. de Lima, et al. "Count2Multiply: Reliable In-memory High-Radix Counting", HPCA 2026

Emerging systems: Great potential!

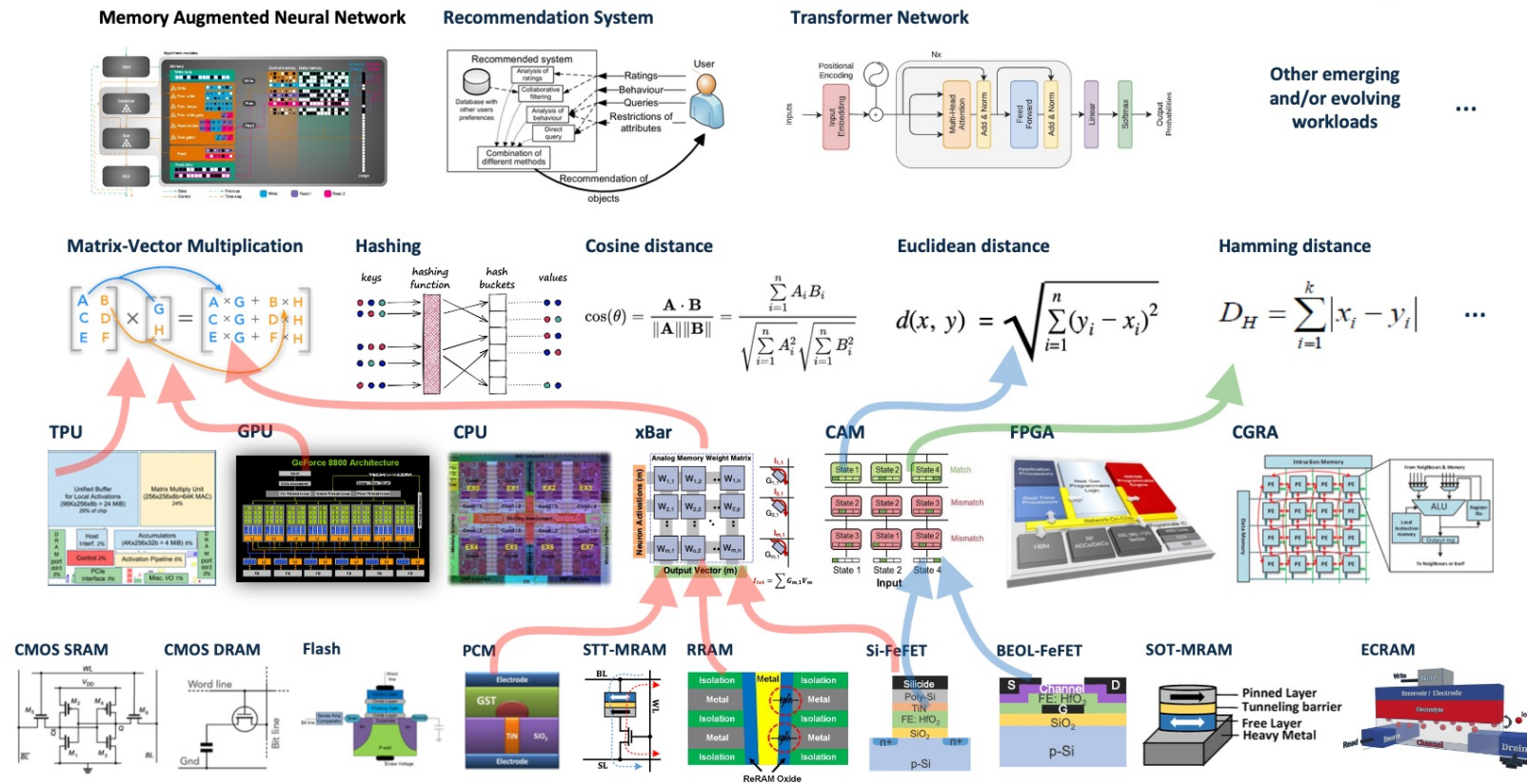


Mao, Ruibin, et al. "E
augmented neural ne



Great designs!! But need automation to **generalize** to other patterns and **optimize around kernels**

CIM: Identifying, generalizing, modeling and optimizing

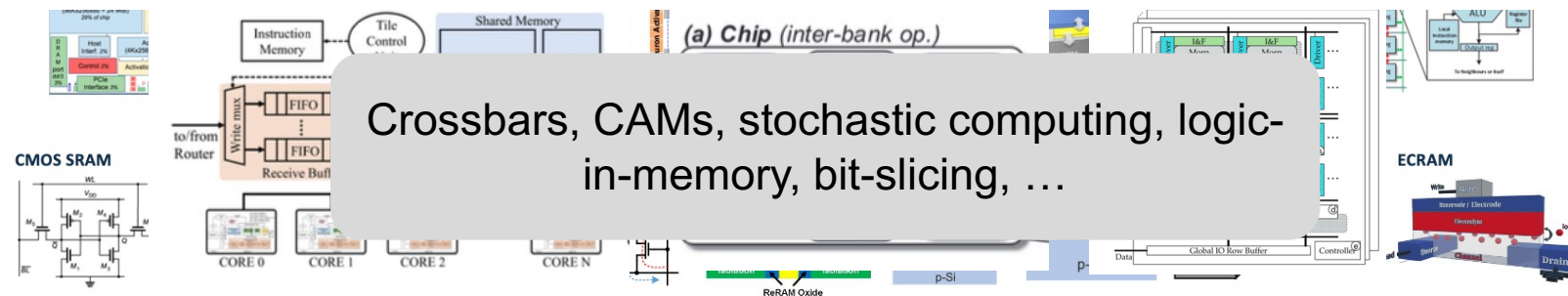


Source: M. Niemier, Notre Dame

CIM: Identifying, generalizing, modeling and optimizing

Operation	Type	Description	CIM	CNM
<code>cinm.{add,sub,mul,div,min,max}(%lhs, %rhs)</code>	$T \times T \rightarrow T$	Element-wise arithmetic	✓	✓
<code>cinm.{and,or,xor,not}(%lhs, %rhs)</code>	$T \times T \rightarrow T$	Element-wise bit-wise logic	✓	✓
<code>cinm.gemv(%lhs, %rhs)</code>	$S^{m \times n} \times S^n \rightarrow S^m$	Matrix-vector product	✓	✓
<code>cinm.gemm(%lhs, %rhs)</code>	$S^{m \times k} \times S^{k \times n} \rightarrow S^{m \times n}$	Matrix-matrix product	✓	✓
<code>cinm.transpose(%in, %perms)</code>	$S^n \times N^n \rightarrow S'$	Transposition	✗	✓
<code>cinm.{histogram,majority}(%in)</code>	$S^n \rightarrow S^k$	Histogram and bit-wise majority	✗	✓
<code>cinm.topk(%in, %k)</code>	$S^n \times N \rightarrow S^k \times N^k$	Finds k largest values & their indices	✗	✓
<code>cinm.simSearch #E, #k (%in1, %in2)</code>	$E \times N^k \times S^n \times S^n \times N \rightarrow S^k$	Finds k most similar values & their indices with metric E	✓	✓
<code>cinm.mergePartial #op #dir (%lhs, %rhs)</code>	$E \times D \times T \times T \rightarrow T$	Hardware-defined operation that merges partial results of E	✓	✓
<code>cinm.popCount(%in)</code>	$T \rightarrow N$	Counts 1s in a bit vector	✓	✗
<code>cinm.reduce #op (%in)</code>	$E \times S^n \rightarrow S$	Performs reduction in group (S, E)	✗	✓
<code>cinm.scan #op (%in)</code>	$E \times S^n \rightarrow S^n$	Performs inclusive scan in group (S, E)	✗	✓

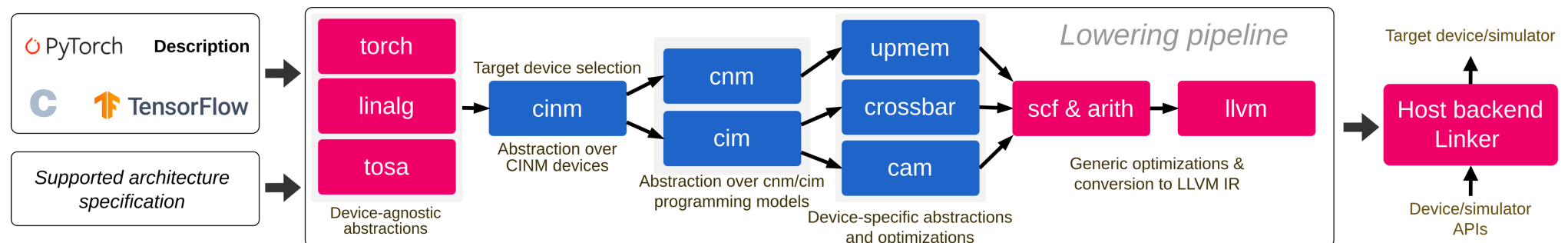
A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", ASPLOS'25



Source: M. Niemier, Notre Dame

CINM: Generalized MLIR infrastructure

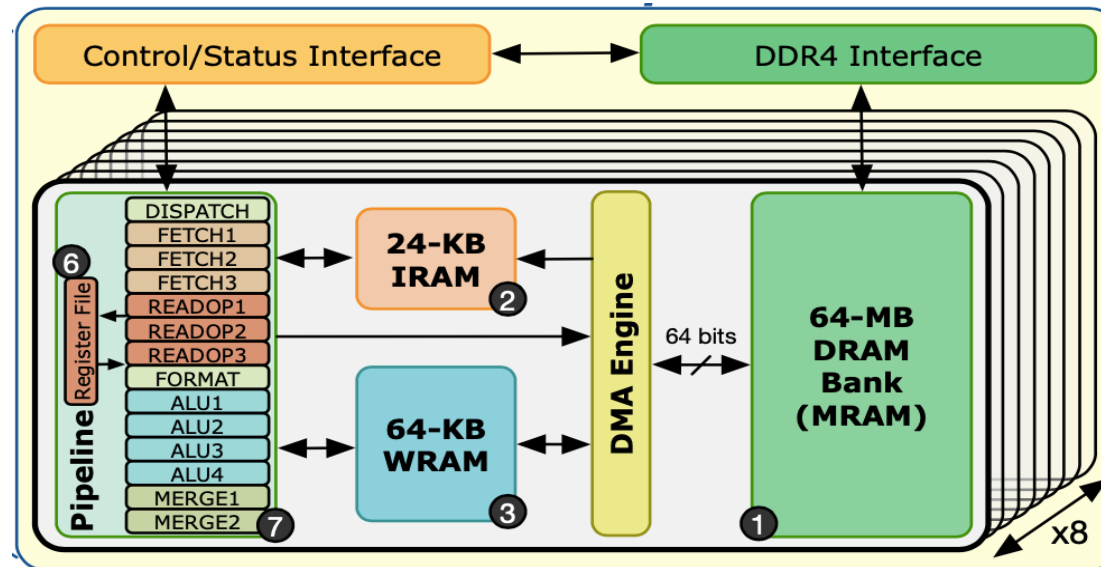
- ❑ From linear algebra abstractions (common to ML frameworks and beyond)
- ❑ Intermediate languages for **in and near memory computing**
- ❑ **Pattern recognition**, target-specific **models and optimizations**



A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", ASPLOS'25
H. Farzaneh et al. "C4CAM: A Compiler for CAM-based In-memory Accelerators", ASPLOS'24

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  
}
```

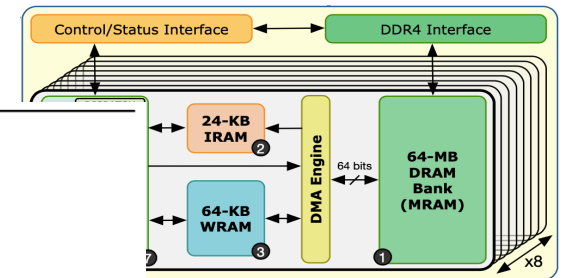


UPMEM example: Matmult

```

def mm(int32(64, 64) A, int32(64, 64) B) -> (int32(64,
64) C)
    C(i,j) BARRIER_INIT(my_barrier, NR_TASKLETS);
    int main() {
        ...
        barrier_wait(&my_barrier);
        int32_t point_per_tasklet = (ROWS*COLS)/NR_TASKLETS;
        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

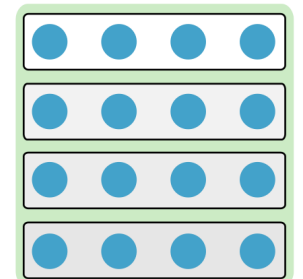
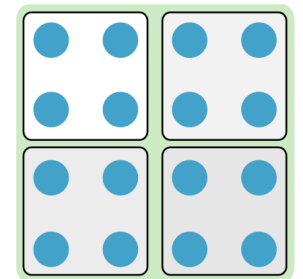
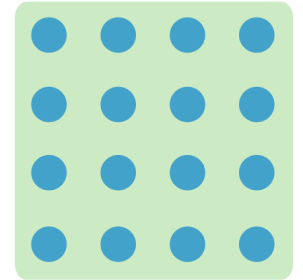
```
def mm(int32(64, 64) A, int32(64, 64) B) -> (int32(64, 64) C)
{
  C(i,j) += A(i,k) * B(k,j)
    where i in 0:64, k in 0:64, j in 0:64
}
```



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

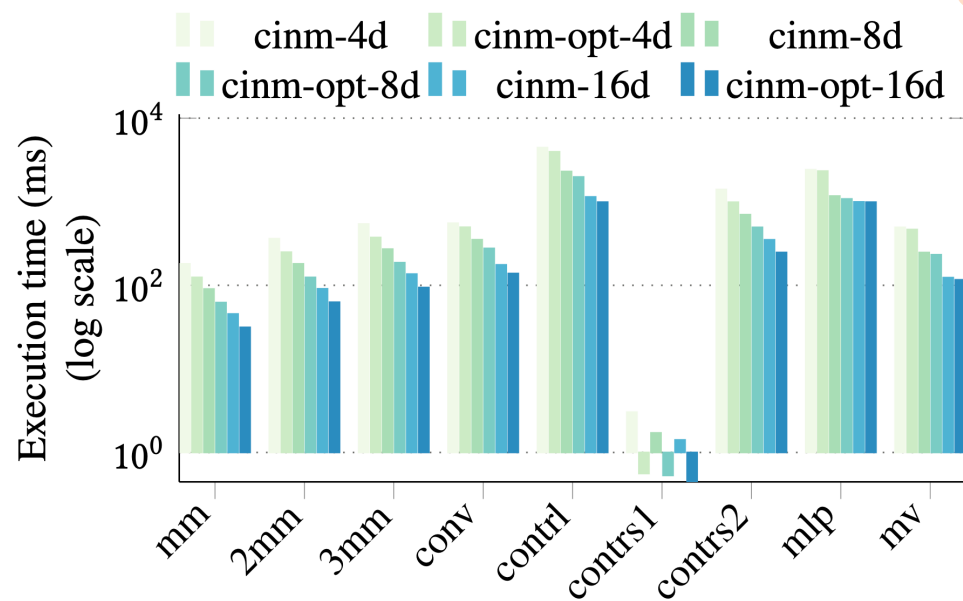


```
%C = scf.for %i = %cst0_i to %cst64_i step %cst16_i
<...>
  %B_tile = tensor.extract_slice %B[%o2, %o1][16, 16][1, 1]:
    tensor<64x64xi32> to tensor<16x16xi32>
  %A_t_b = bufferization.to_memref %A_tile: memref<16x16xi32>
  %B_t_b = bufferization.to_memref %B_tile: memref<16x16xi32>
  %A_dev = 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]: <...>
```



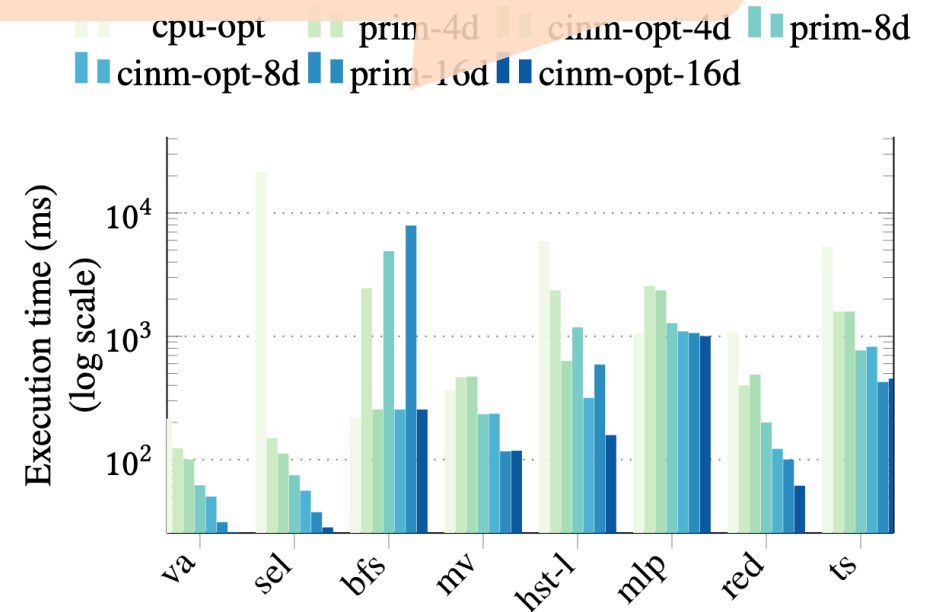
UPMEM example: Results

Manual designs 2-5x faster than CPU.
CINM 1.5-2x faster than hand-optimized code



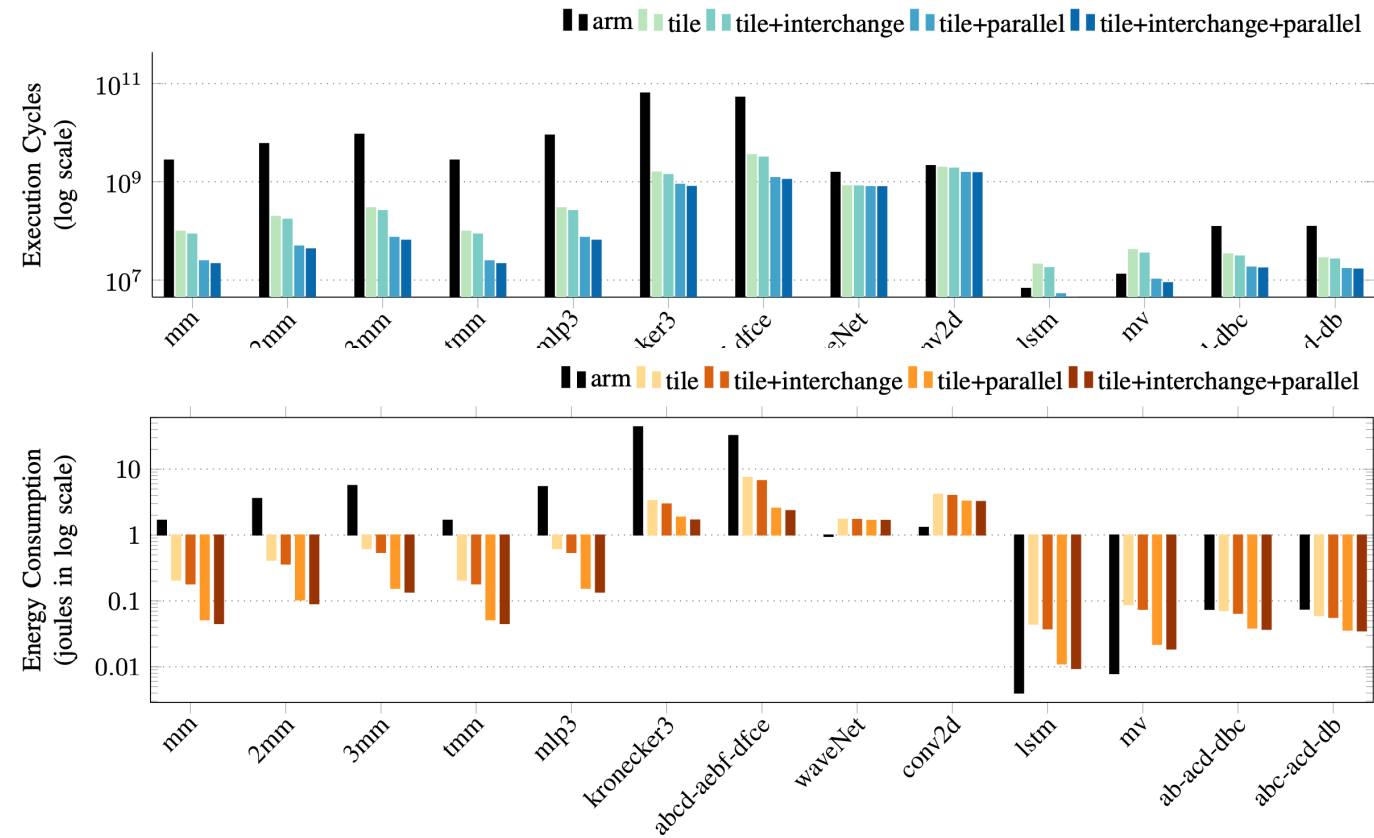
1-DIMM 128 DPUs
 4-DIMMs 512 DPUs
 8-DIMMs 1024 DPUs
 16-DIMMs 2048 DPUs

Optimizations achieve **40%-50% speedup** (geomean)



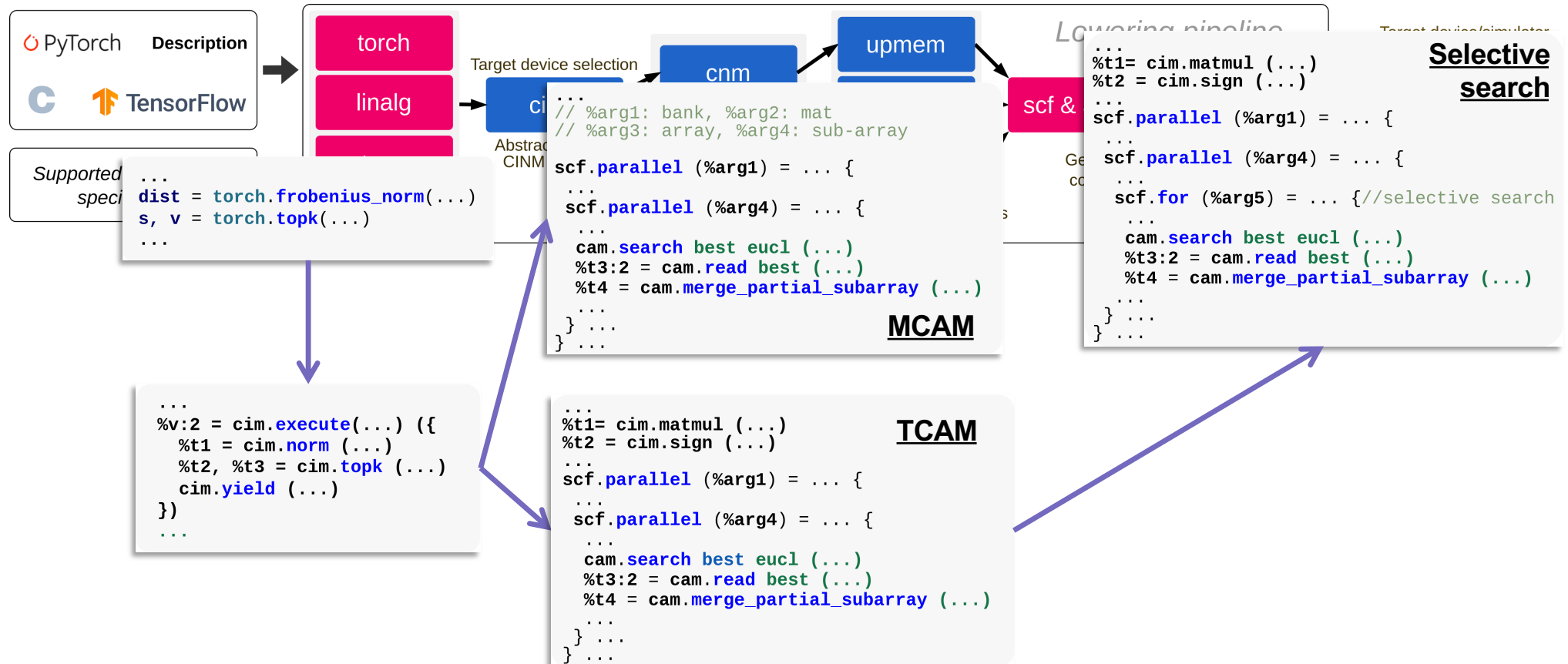
10x-40x less lines of code

Optimization results: Crossbars beyond matmult



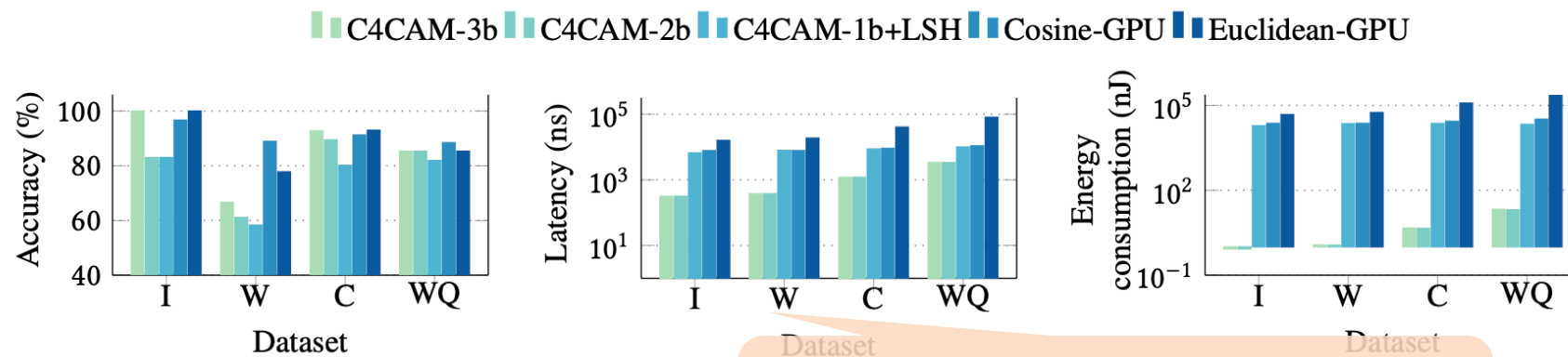
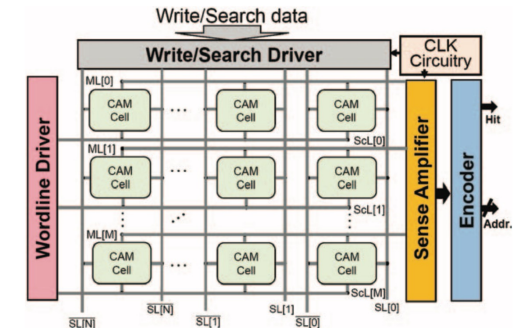
A. Siemieniuk, L. Chelini, A. A. Khan, J. Castrillon, A. Drebes, H. Corporaal, T. Grosser, M. Kong, "OCC: An Automated End-to-End Machine Learning Optimizing Compiler for Computing-In-Memory", In IEEE TCAD, 2021

CINM: Lowering for different CAM-based acc.



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**

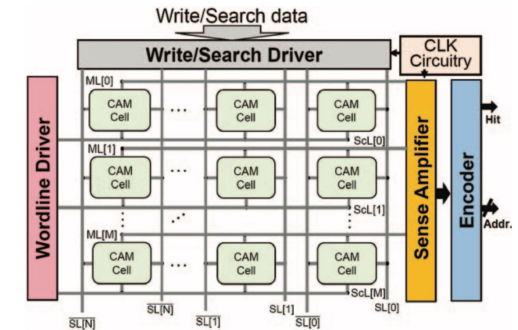


H. Farzaneh, et al. "C4CAM: A Compiler for CAM-based In-memory Accelerators", ASPLOS, 2024

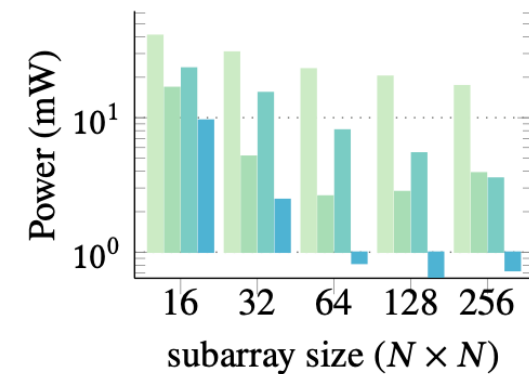
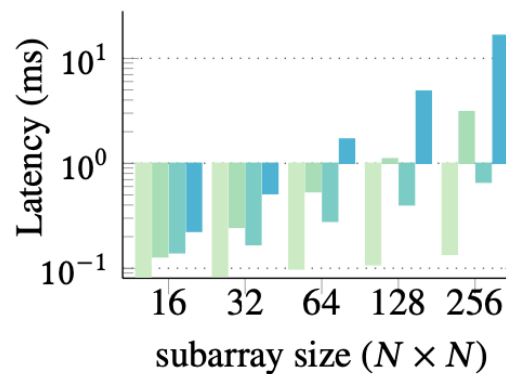
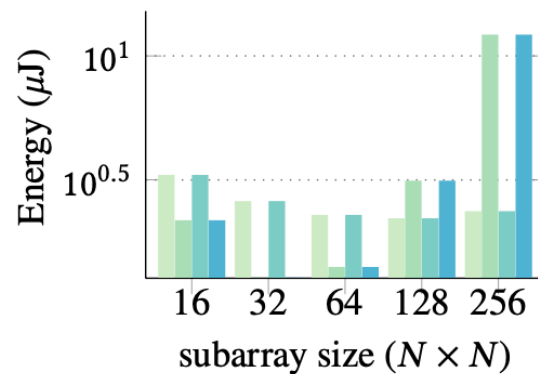
KNN results (128x128 CAM): **14x faster** and **~10⁴ less energy** compared to GPU

CAMs: Design space exploration

- Retargetable compiler for CAM exploration: Sizes and features
- Compiler flags for optimization target



cam-base cam-density cam-power cam-density+power

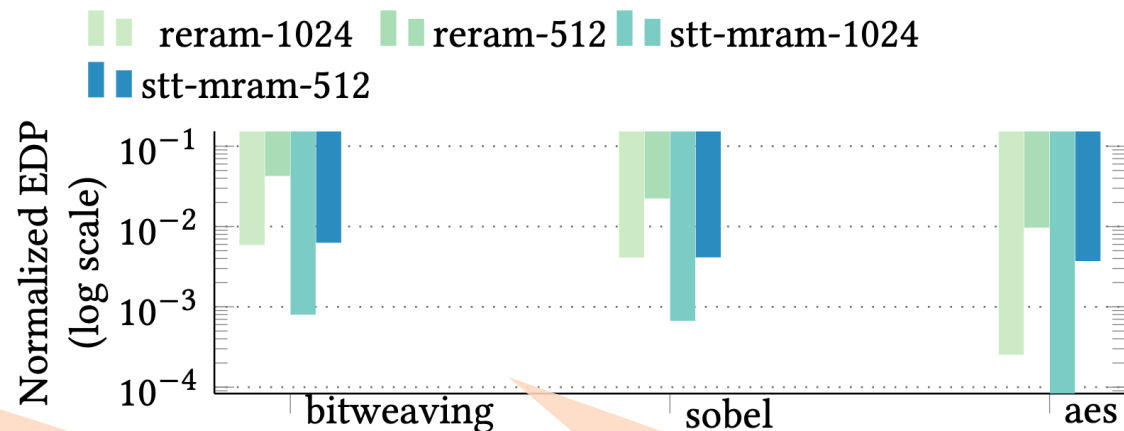
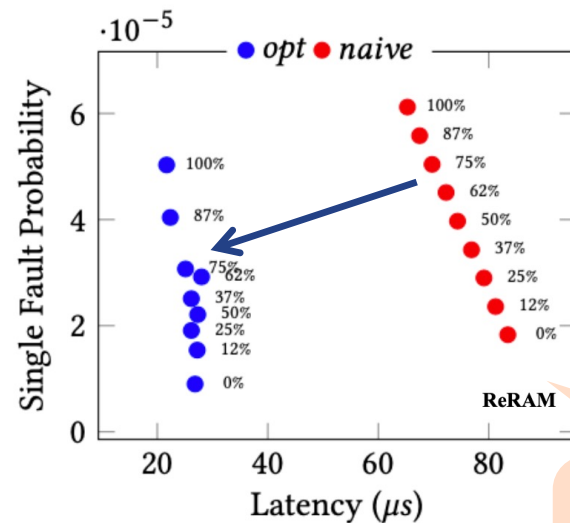


H. Farzaneh, et al. "C4CAM: A Compiler for CAM-based In-memory Accelerators", ASPLOS, 2024

Different flags expose trade-offs w/o manual re-coding. Next: include **device-level parameters**

Logic-in-memory in NVMs

- Massively parallel multi-operand bit-wise operations in-memory
- Complex mapping of operands, operations and temporaries to columns



Optimized mapping: **Less latency (3x), better reliability (~1.4x)**

Orders of magnitude better EDP vs CPU baseline

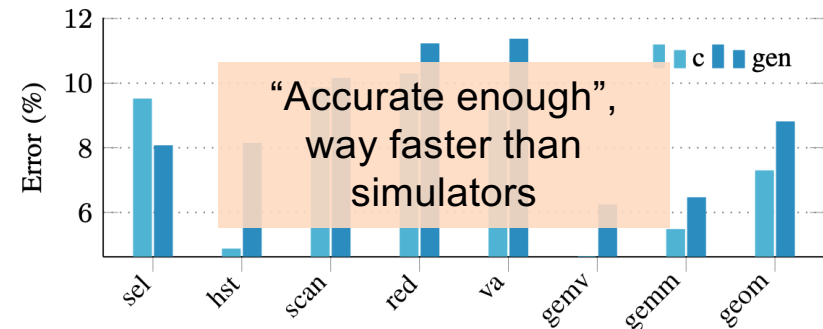
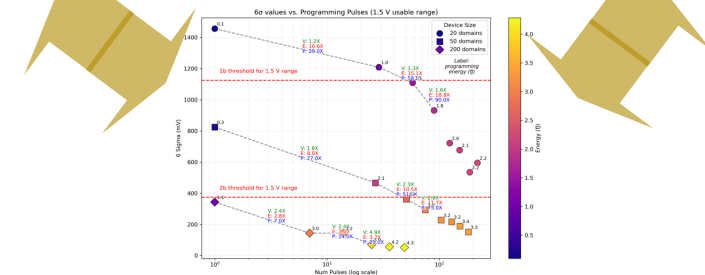
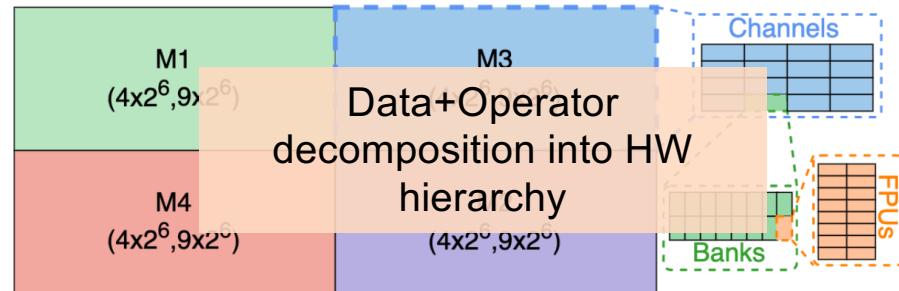
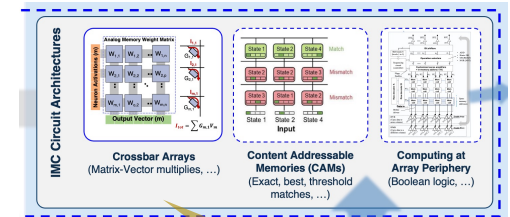
H. Farzaneh, et al. "SHERLOCK: Scheduling Efficient and Reliable Bulk Bitwise Operations in NVMs", DAC 2024

Models for compilers

- Abstract models for latency, energy, ...
- Algorithm – Architecture – *Atoms*
- Integration in CINM for generalized **non-von-Neumann native exploration**

```

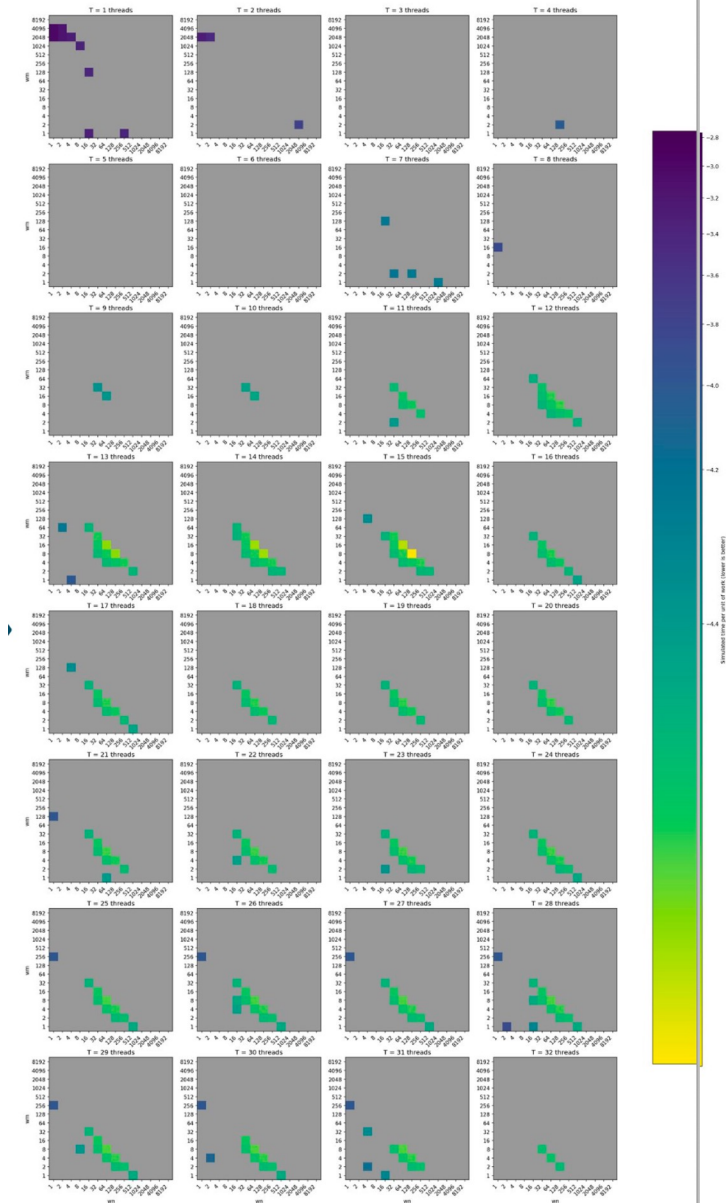
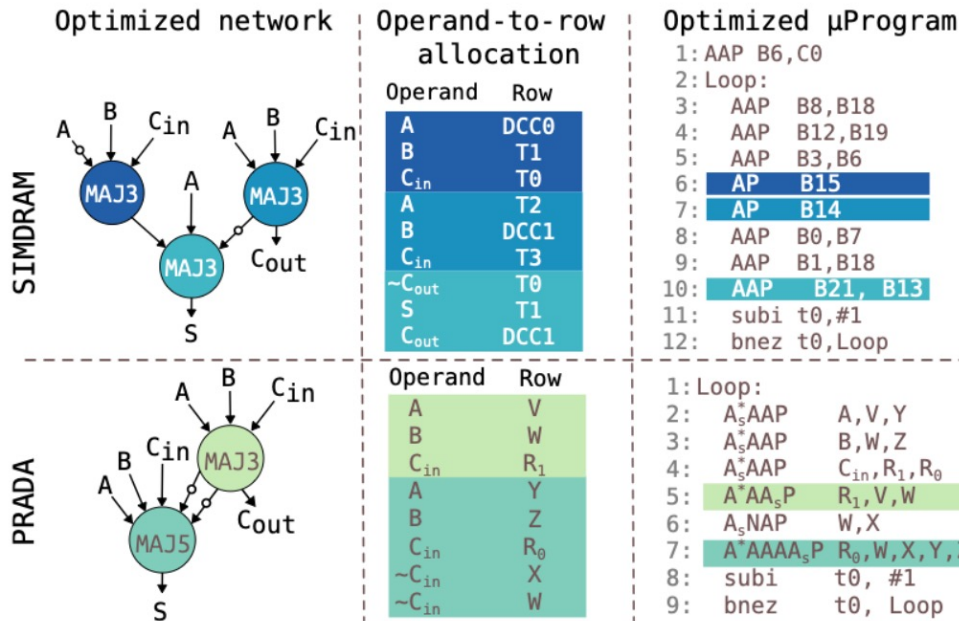
1 i_flav = spectra.bnd_to_flav[i_strato][:,i_bnd]
2 i_eta = j_eta[i_lay,i_flav,...]
3 i_eta = np.stack((i_eta,i_eta+1), -1)[:,:None,:,:)
4 a = n_prime_mix[None,i_lay,i_flav,None,:,:)
5 b = f_major[None,i_lay,i_flav,:,:)
6 c = spectra.k_major[gptS:gptE,i_p,i_T,i_eta]
7 result[gptS:gptE,:] = np.sum(a * b * c, (2,3,4))
    
```



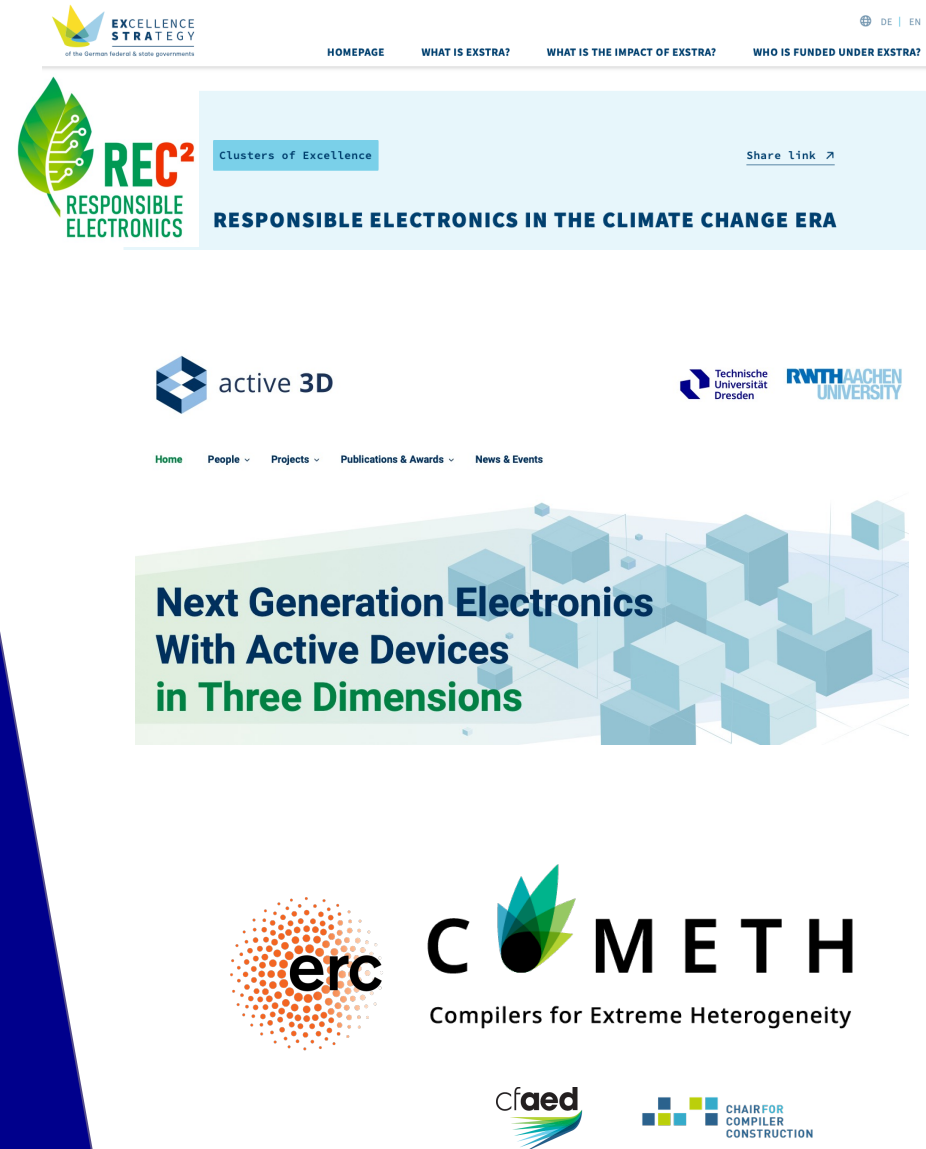
H, Farzaneh, et al, "CoMoNM: A Cost Modeling Framework for Compute-Near-Memory Systems", ACM TACO, Aug 2026

Compiler models and exploration

- Working on methods to efficiently prune parameter space
- Integrate models in synthesis to in-memory primitives
- Architecture description for retargetability



Moving forward



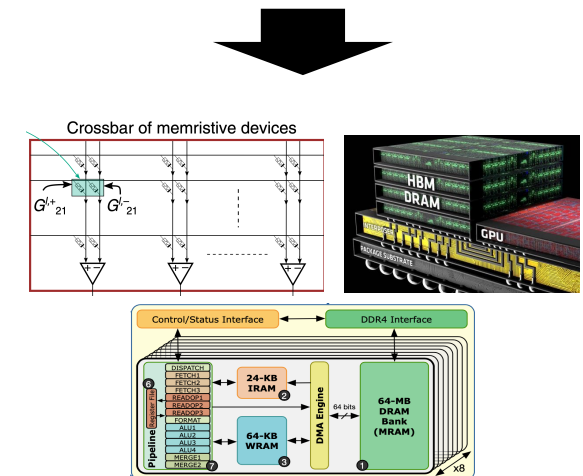
Summary

- Coordination + abstractions to tackle heterogeneity
 - Coordination and domain-specific languages
 - Optimizations for different types of systems and app domains
 - Understanding emerging computing architectures/models
 - Building compilers for non-von Neumann systems
- Lots of challenges remain:
 - Simulators, prototypes in interdisciplinary research efforts
 - Further understanding and modeling of primitives
 - Building *faithful* models for emerging computing...

```

7  
$$t = \left( s \otimes (s \otimes (s \otimes u)_{cz}^{xyz})_{by}^{cxy} \right)_{ax}^{bcx}$$

8  // Major contributors
9  let tau_g_M = (n_mix[i_flav[x], x, deta]
10                * f_major[i_flav[x], x, dT,
11                * k_major[i_T[x], dT], i_p[x,
```



Challenges in multi-level compilation and CIM

- Great approach to close the semantic gap
- Higher semantics → more powerful optimization

□ Under the hood: Von Neumann

```
for i in 0..100
  z[i]=x[i]+y[i];
```

Von Neumann

```
...
%1 = load i32* %xi
%2 = load i32* %yi
%3 = %1 + %2
store i32 %3, i32* zi
...
```

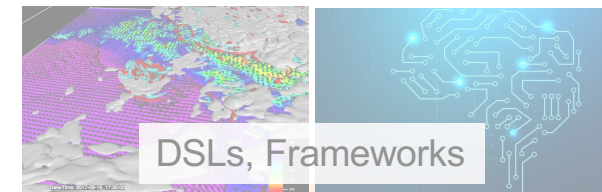
Word-based

VS

Logic in-DRAM

```
notX = NOT(Xi)
write notX
Out1 = MAJ(notX, Yi, Cr)
write out1
Cr = MAJ(Xi, Yi, Cr)
write Cr
notCr = NOT(Cr)
write notCr
Zi = MAJ(X, out1,
notCr)
write Zi
```

Bit-sliced, massively parallel



High: tensor product,
convolution

search-write in CAMs

```
search 011--
write ---10
search 001-
write ----01
search 100--
write ---01
search 110--
write ---10
```

Multi-level compilation and CIM

- ❑ Great approach to close the semantic gap
- ❑ Higher semantics → more powerful optimization
- ❑ **Under the hood: Von Neumann**

```
for i in 0..100
    z[i]=x[i]+y[i];
```

Von Neumann

```
...
%1 = load i32* %xi
```

VS

Need **bottom up** computational models to build **faithful abstractions** for non Von Neuman systems

Word-based

Logic in-DRAM

```
notX = NOT(Xi)
write notX
Out1 = MAJ(notX, Yi, Cr)
write out1
Cr = MAJ(Xi, Yi, Cr)
```

```
notCr)
write Zi
```

Bit-sliced, massively parallel

DSLs, Frameworks

High: tensor product, convolution

search-write in CAMs

```
search 011--
write  ---10
search 001-
write  ----01
```

search 100--



COMETH
Compilers for Extreme Heterogeneity

Compilers for Emerging Heterogeneous Systems

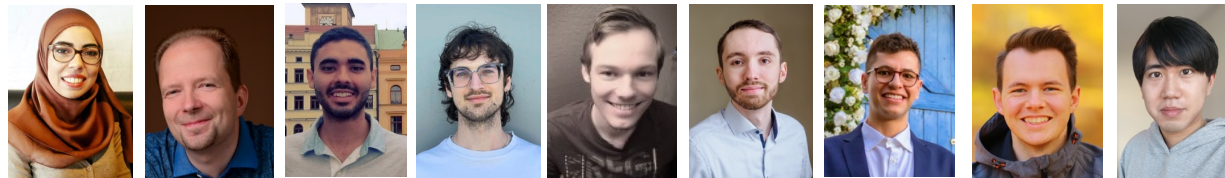
Road works ahead



Thanks! & Acknowledgements



Ali Ansari Tauseef Ahmad Jiahong Bi Dr. João P. de Lima Hamid Farzaneh Clément Fournier Karl Friebe Dr. Jiaxin Huang Dr. Asif Khan Dr. Robert Khasanov



Nesrine Khouzami Dr. Steffen Köhler Julian Robledo Luis Schoppik Dr. Lars Schütze Christian Schwarz Guilherme Silva Felix Suchert Zhiyang Wang

And collabortors (e.g., J-J. Chen, A. Jones, X.S. Hu, M. Niemier, M. Tahoori, E. A. Lee, A. Silva, H. Härtig)

STAATSMINISTERIUM
FÜR WISSENSCHAFT
KULTUR UND TOURISMUS



Freistaat
SACHSEN



BMBF (01IS18026A-D)

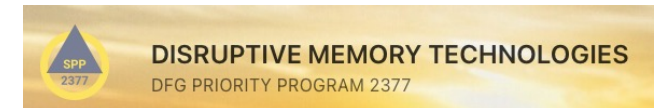


Funded by

HetCIM-I, II (502388442,
502388442)
Active-3D TRR (528378584)
REC2 EXC 3035
(533607596)



active 3D



101135183

MYRTUS



ERC CoG (101230491)

