CHAIR FOR COMPILER CONSTRUCTION

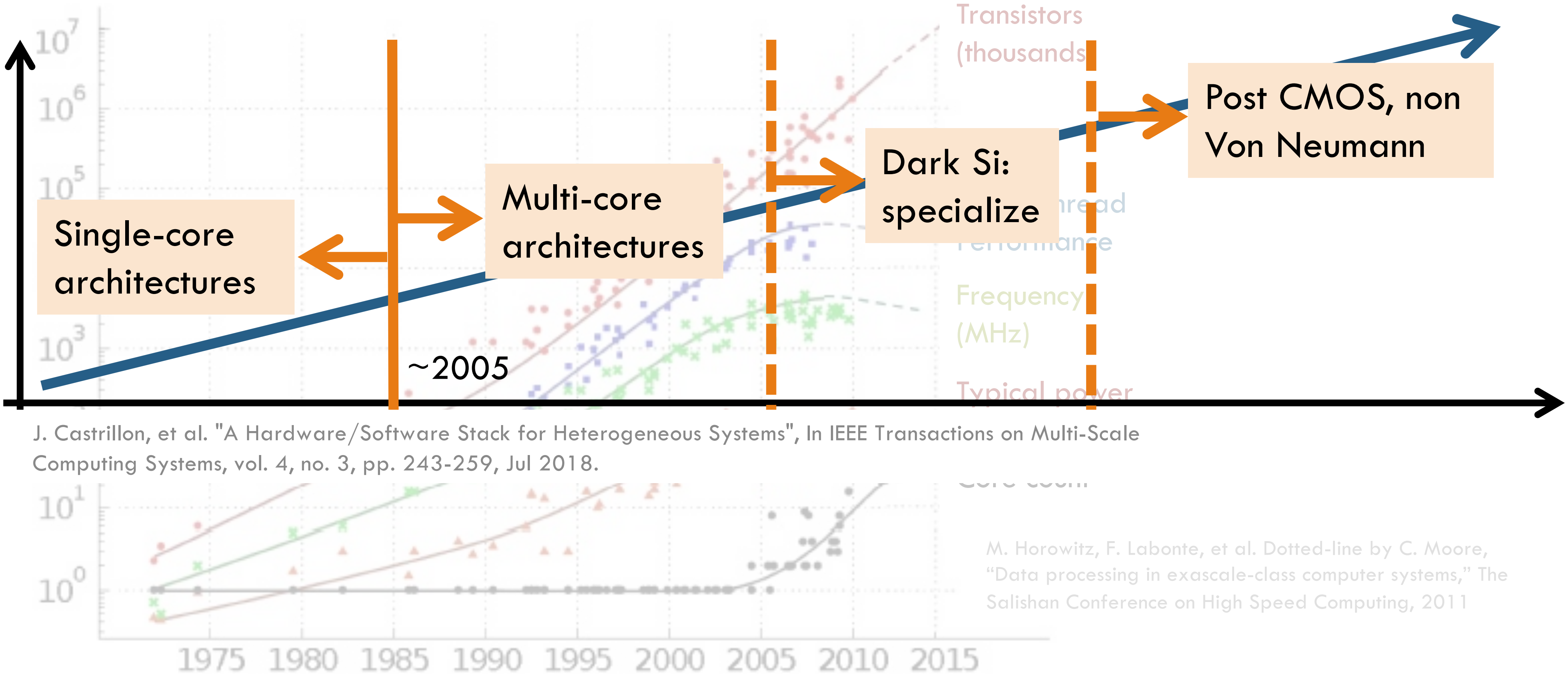
cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

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

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

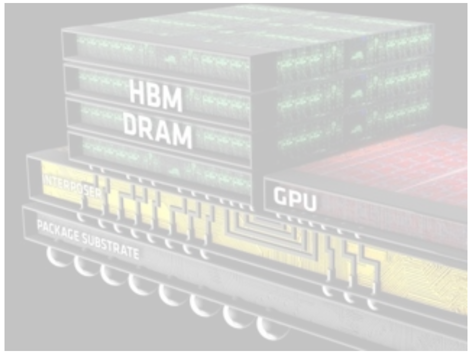
8th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)
SC23: The International Conference on High Performance Computing, Networking, Storage and Analysis, Denver, USA
November 13, 2023

Evolution of computing: Breaking walls



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

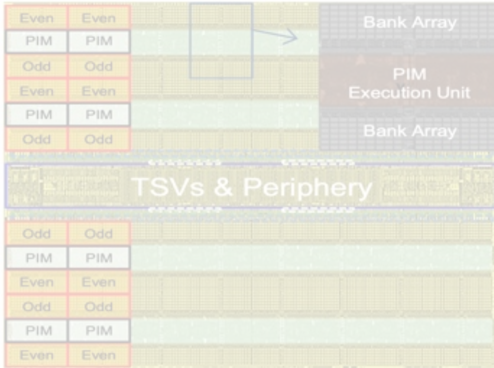
Emerging systems: Examples



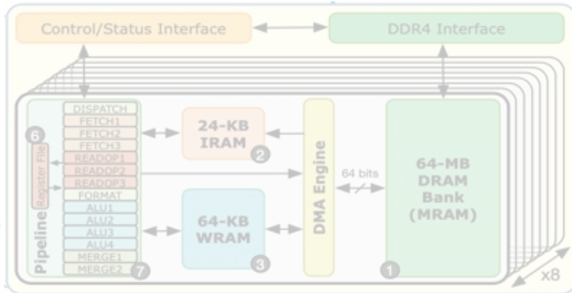
Source: AMD, AnandTech

High-bandwidth memory

AI accelerators + Prog. logic



Source: Samsung



Source: UPMEM

Near-memory computing

Source: AMD

es +

ting

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

Abstractions and compilation

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

What we want

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

What we (naively) code

100X

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

What performance experts code

Abstractions and compilation

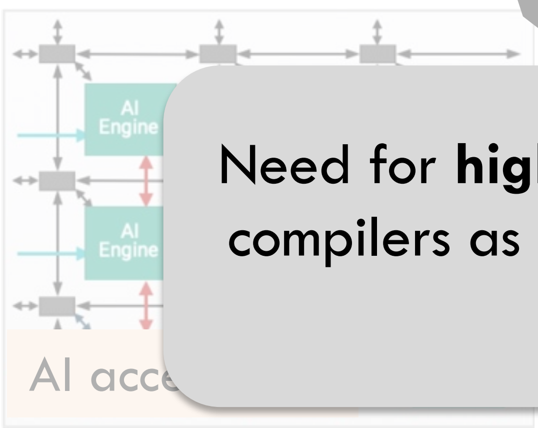
$$v_{ijk,e} = \sum_{i'=0}^{p}\sum_{j'=0}^{p}\sum_{k'=0}^{p} A_{kk'}A_{jj'}A_{ii'}u_{i'j'k'e}$$

What we want

```c
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 < 7; j1++) {
      for(int k1 < 7; k1++) {
        v[e][i k0] += A[i0][i1]
```

```c
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 */
```

100X



HBM+FPGA

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

CHAIR FOR COMPILER CONSTRUCTION

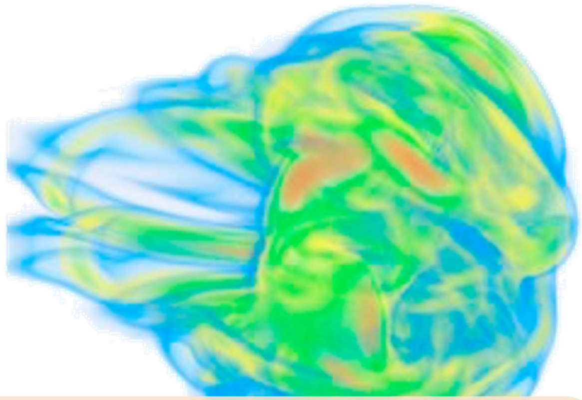
The power of abstractions

Example: Particle-mesh simulations

- Particle-mesh simulations in computational biology
  - Discrete/continuous
  - Deterministic/stochastic

Vortex ring

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



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

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

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

```
type of simulation:
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_
```

Syntax for interact, evolve, automatic insertion of interpolation, ghost sync., …

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.

Semantic gap → Debugging gap

- OpenFPM library
  - Modern C++ template library (for CPUs and GPUs)
  - Support for dynamic load-balancing, checkpointing and communication abstractions

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

- Template meta-programming

$$\frac{D\omega}{Dt} = (\omega\,.\,\nabla)u + \nu\Delta\omega$$

What we want

```
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])+
```

```
g_dwp.template get<rhs>(key)[y]=
  fac1*(g_vort.template get<vorticity>(key.move(x,1))[y]+
  g_vort.template get<vorticity>(key.move(x,-1))[y])+
```

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

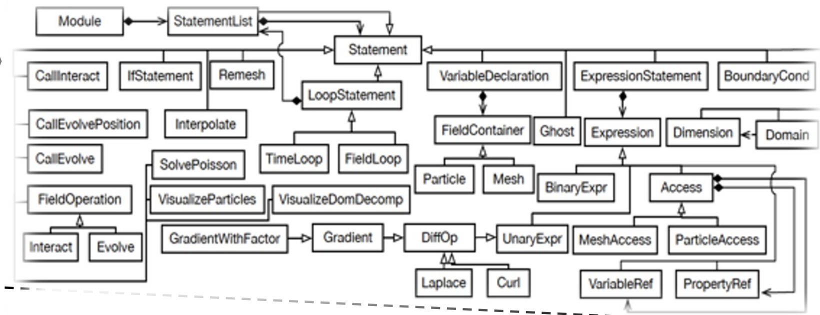
3D

What we code
(already quite abstracted!)

Model-to-model code generation

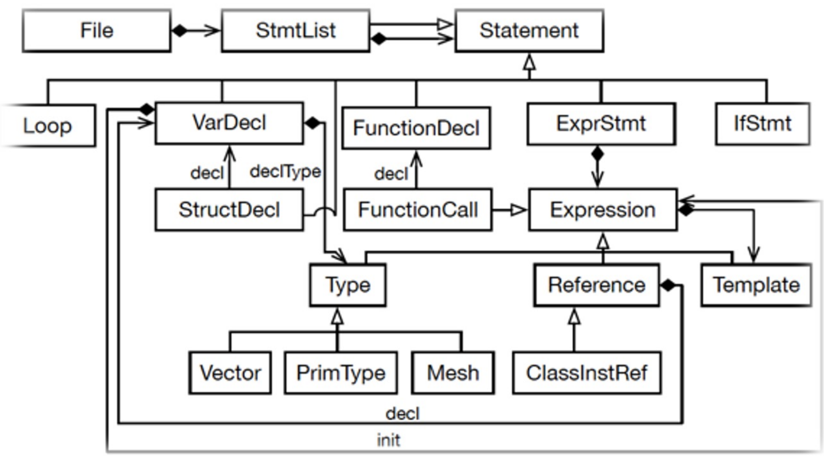
OpenPME DSL

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



Intermediate representation (IR)

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



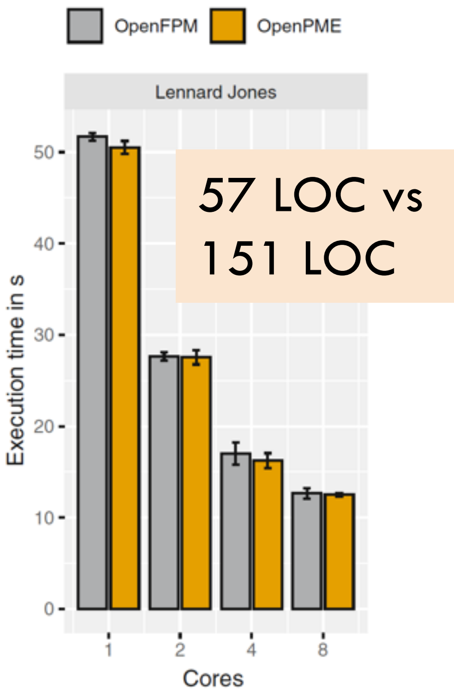

```
while (mloop_iterator_h5a0.isNext())
{   g_dwp.template get<rhs>(key)[x]=
      fac1*(g_vort.template get<vorticity>(key.move(x,1))[x]+
      g_vort.template get<vorticity>(key.move(x,-1))[x])+
      fac2*(g_vort.template get<vorticity>(key.move(y,1))[x]+
      g_vort.template get<vorticity>(key.move(y,-1))[x])+
      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]=
}
```

OpenFPM

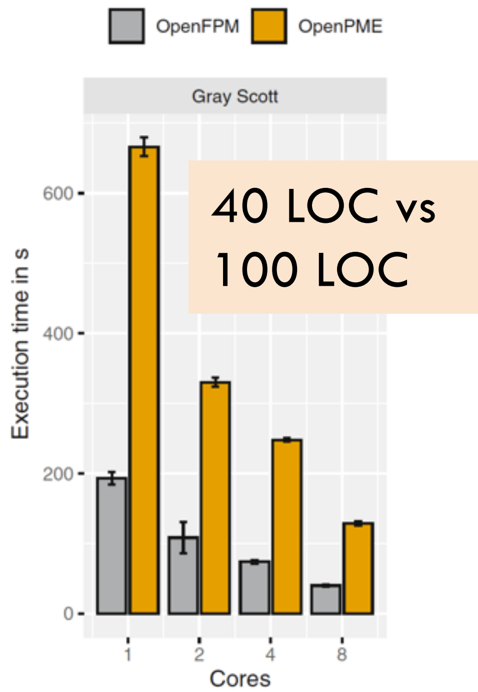
N Khouzami, et al., "The OpenPME Problem Solving Environment for Numerical Simulations", In ICCS'21 pp. 614–627, Jun 2021

Closing the performance gap

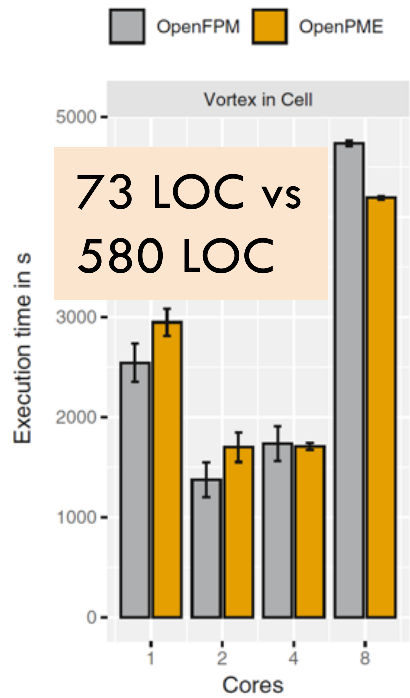
Lennard Jones
(particles, discrete)



Gray-Scott
(mesh, continuous)



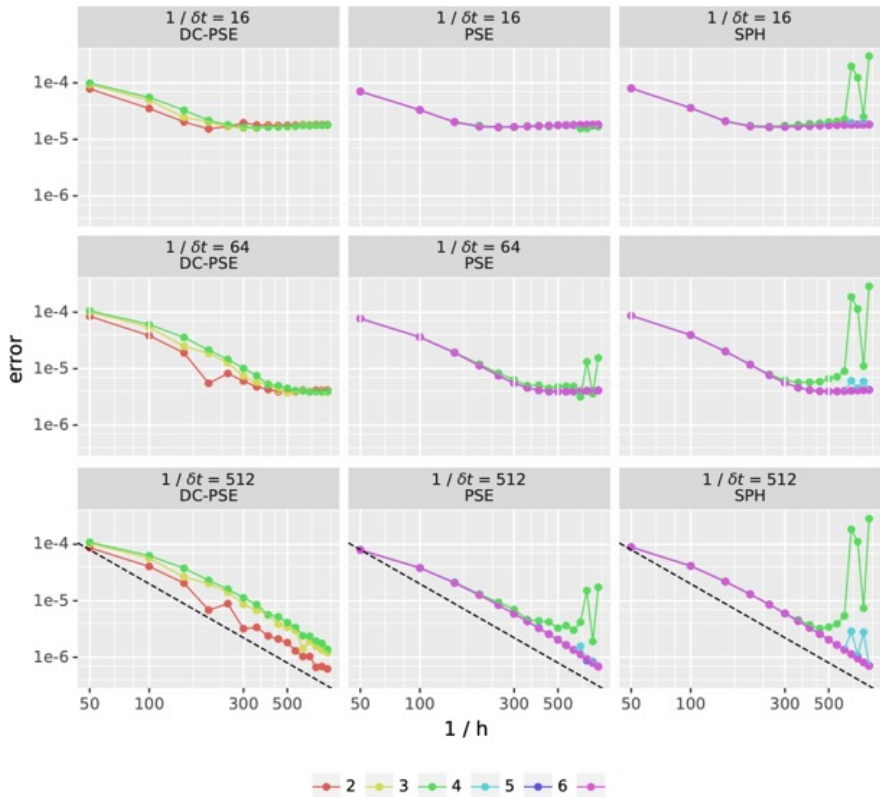
Vortex in Cell
(hybrid, continuous)

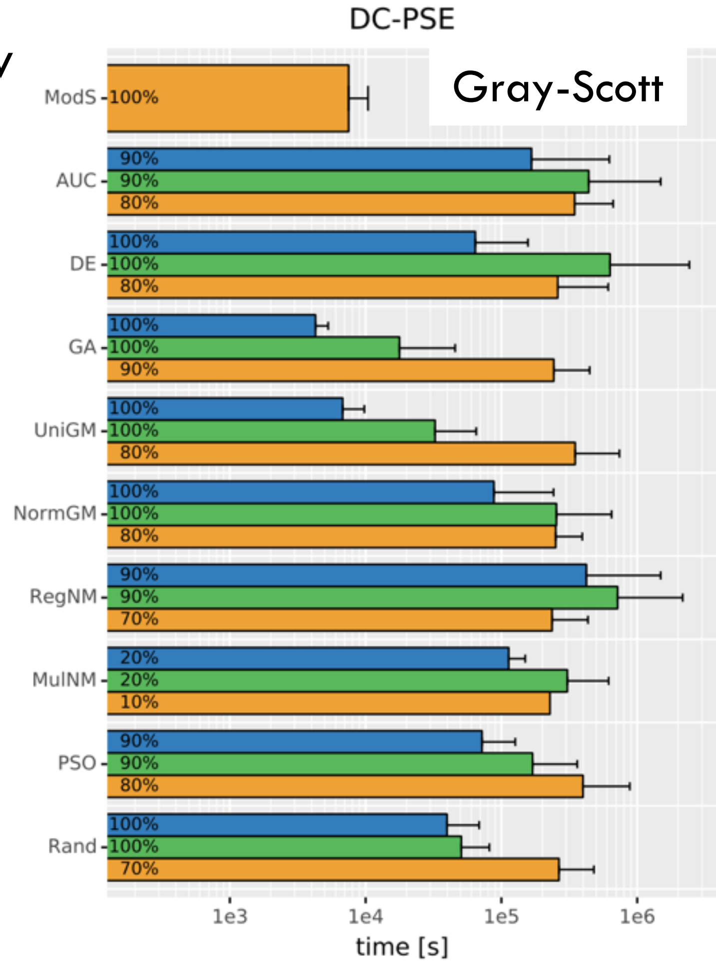


N Khouzami, et al., "The OpenPME Problem Solving Environment for Numerical Simulations", In ICCS'21 pp. 614–627, Jun 2021

Higher-level optimizations

- Insertion of ghost-gets, based on high-level dataflow
- Model-based auto-tunning for discretization
- Theoretical convergence to steer search



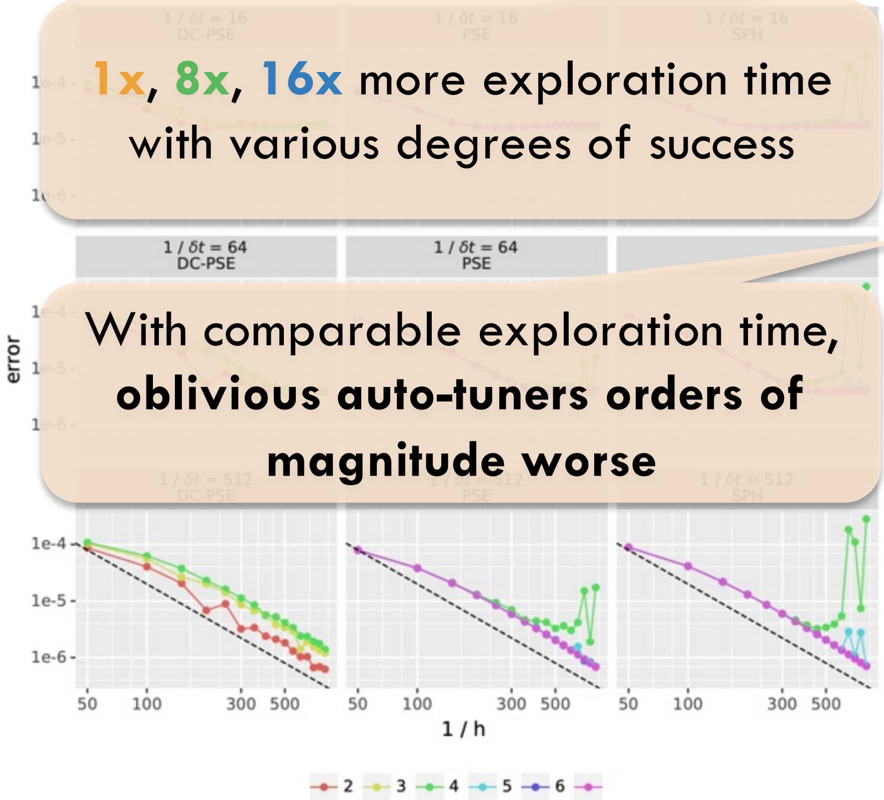


N. Khouzami, et al. "Model-based Autotuning of Discretization Methods in Numerical Simulations of Partial Differential Equations", In Journal of Computational Science, 2021.

Higher-level optimizations

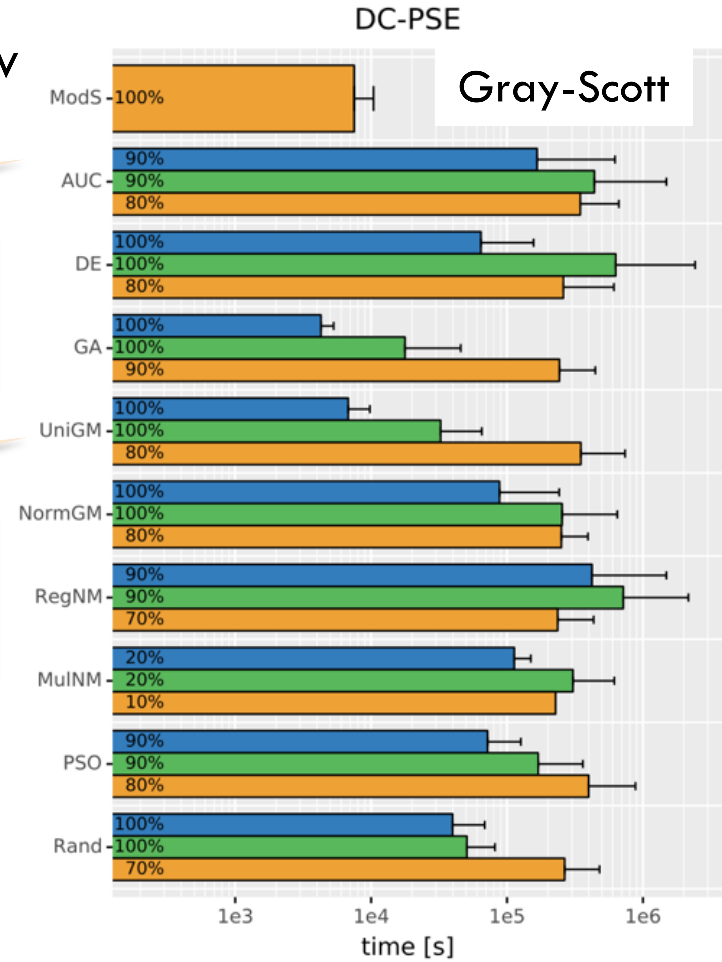
- ❑ Insertion of ghost-gets, based on high-level dataflow
- ❑ Model-based auto-tunning for discretization
- ❑ Theoretical convergence to steer search

N. Khouzami, et al. "Model-based Autotuning of Discretization Methods in Numerical Simulations of Partial Differential Equations", In Journal of Computational Science, 2021.



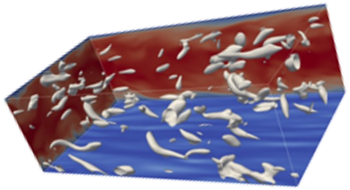
1x, 8x, 16x more exploration time with various degrees of success

With comparable exploration time, **oblivious auto-tuners orders of magnitude worse**



Example: Tensor expressions (Physics, ML)

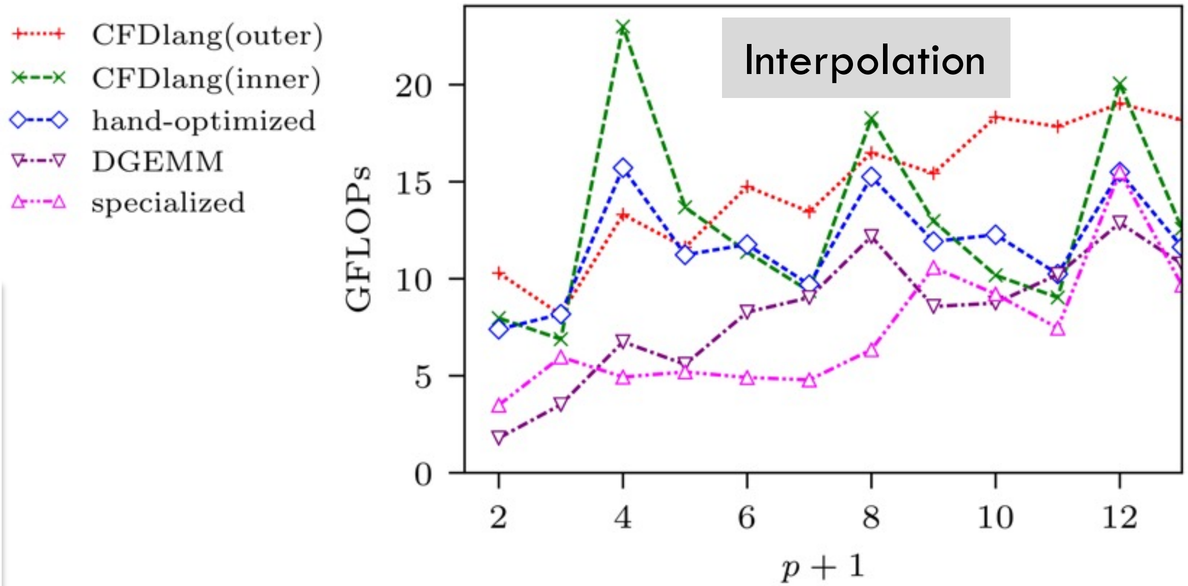
- CFDlang



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

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

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

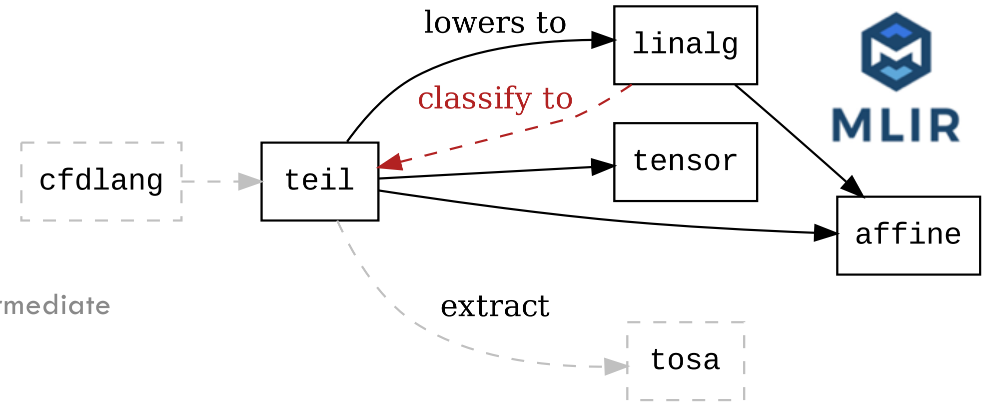


N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL'18.
N.A. Rink, N. A. and J. Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ARRAY'19, pp. 57-68

Tensor intermediate language (TeIL) in MLIR

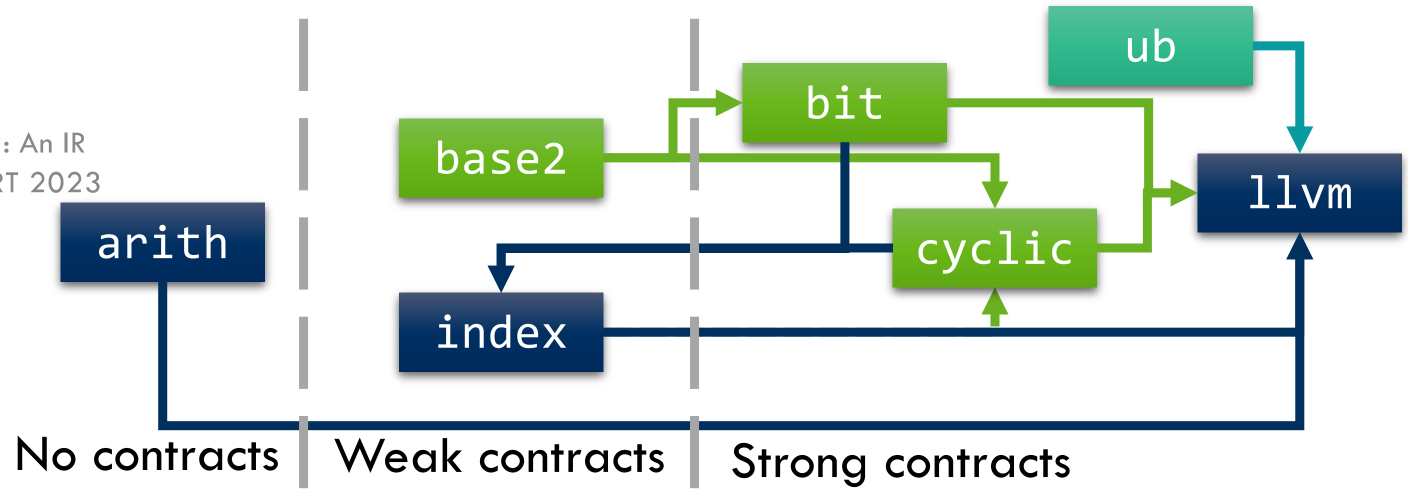
- Primitive ops instead of index maps
  - Easier to express identities (big-O trfs)
  - Uses symbolic math, infinite precision

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



- Specialization path to custom hardware

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



Flow from DSL to system-level architecture

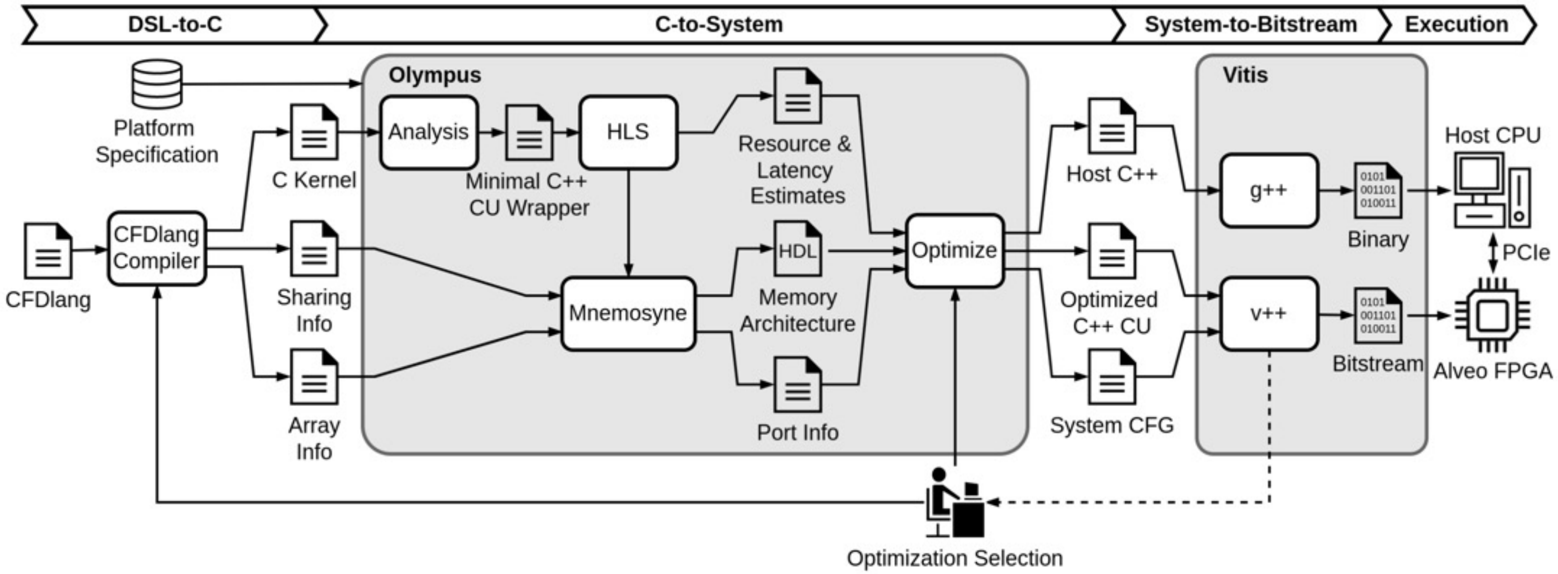
- H2020 EU Project: Convergence HPC, Big Data and ML

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

EVEREST

https://everest-h2020.eu

```
source =  ...
var input A    : mat
var input u    : ten
var input output v
var input alpha : [
var input beta  : [
v = alpha * (A # A
    [[5 8] [3 7] [1
```

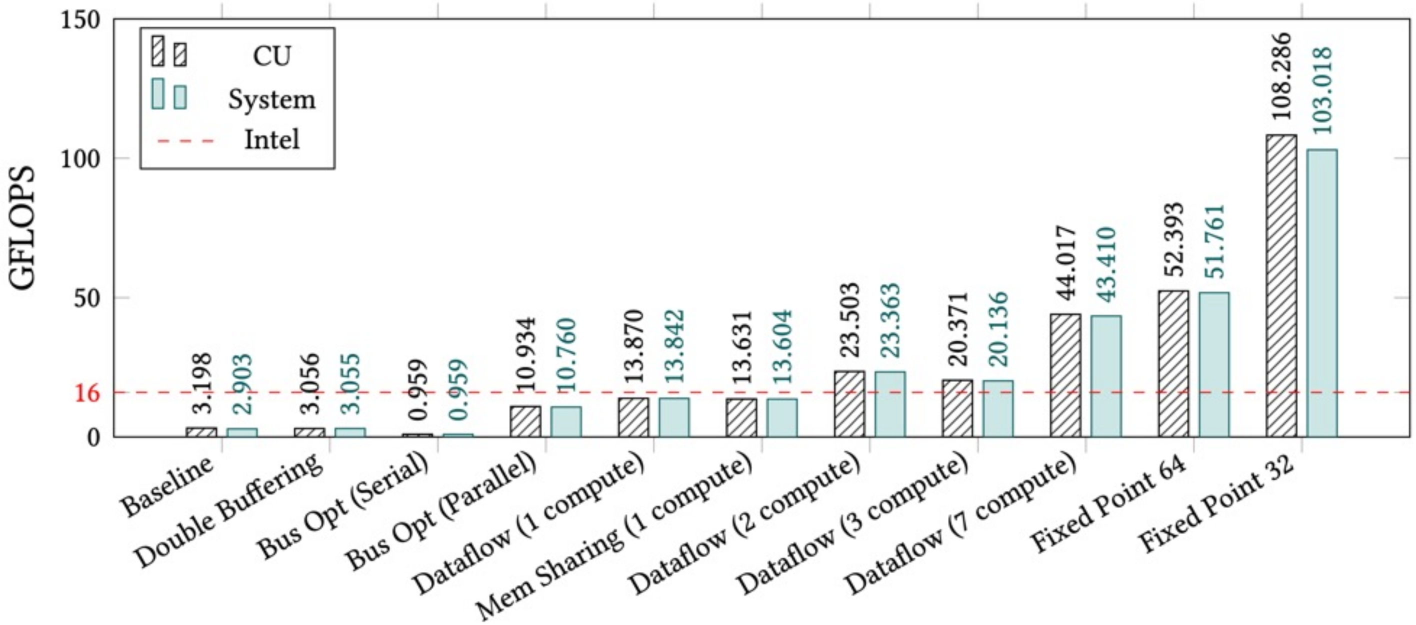


HBM+FPGA

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

FPGA code generation: HBM FPGA

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



EVEREST

https://everest-h2020.eu

HBM+FPGA

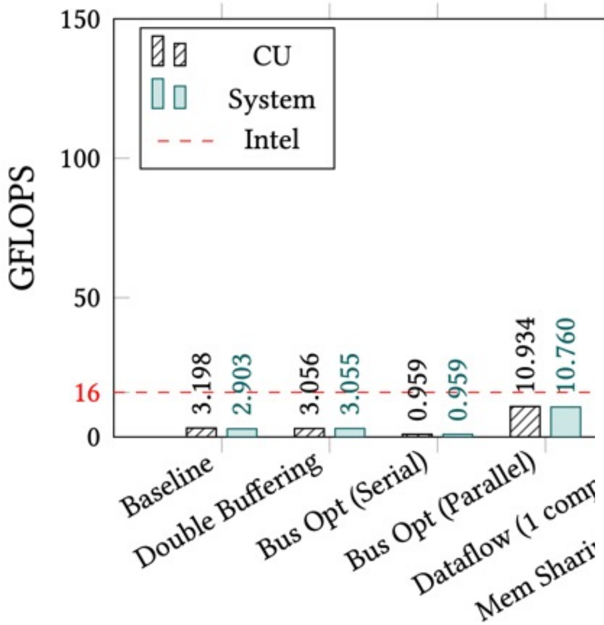
S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, and C. Pilato. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics". In: ACM TRETS, Sept. 2022.

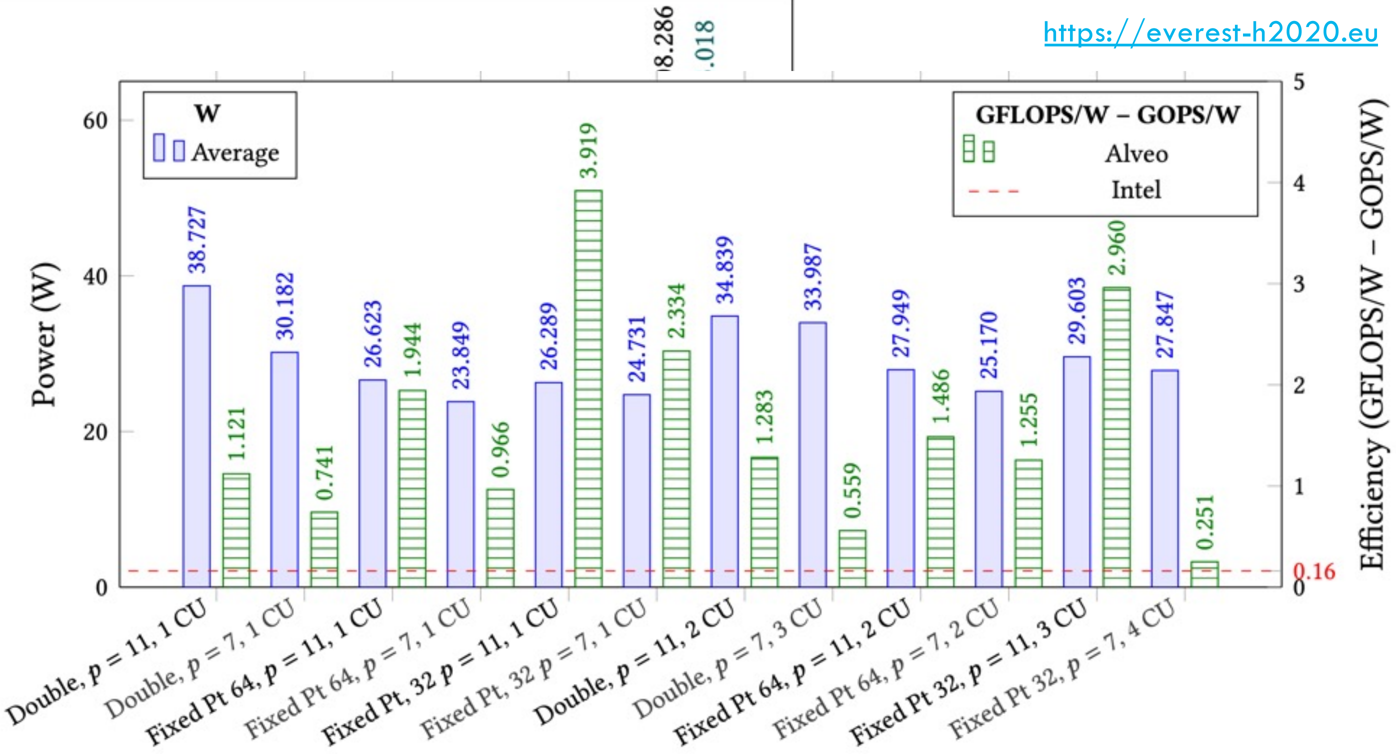
FPGA code generation: HBM FPGA

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

EVEREST

https://everest-h2020.eu

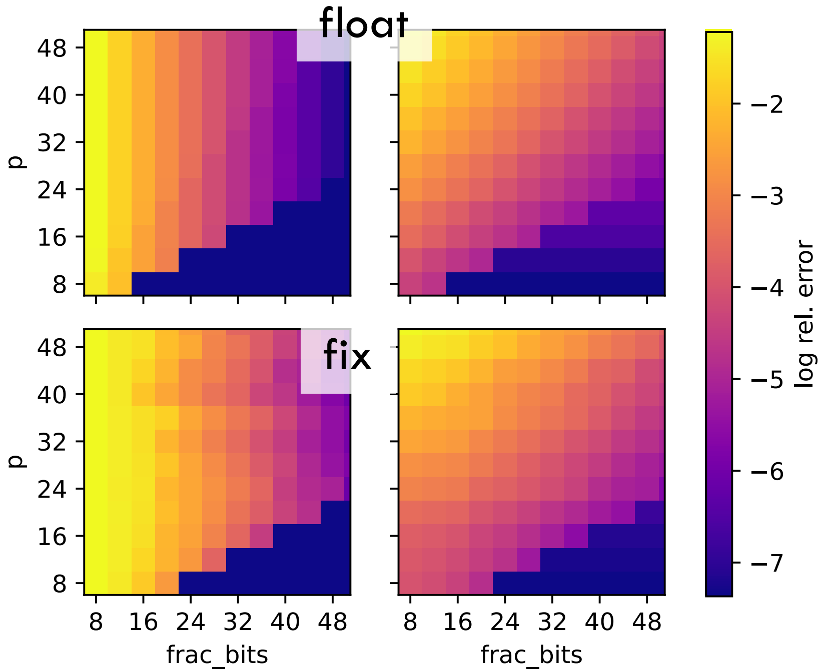




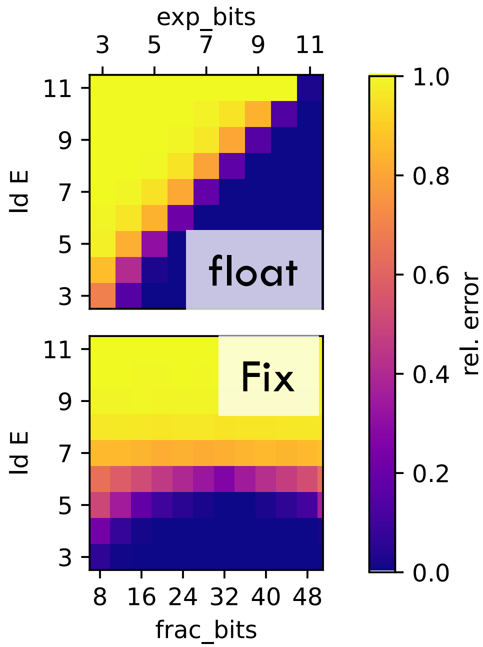
S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hem
Architectures from Domain-Specific Languages:

Base2: Custom precision analysis

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



Significand precision p
(exp_bits = 6)



Exponent range ld E
(frac_bits = 32)

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

Towards a system development kit (SDK)

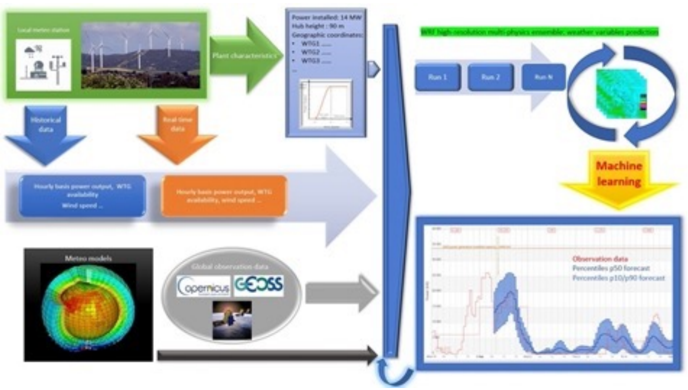
- Kernel integration into legacy (e.g., WRF)
- High-level (implicit) dataflow

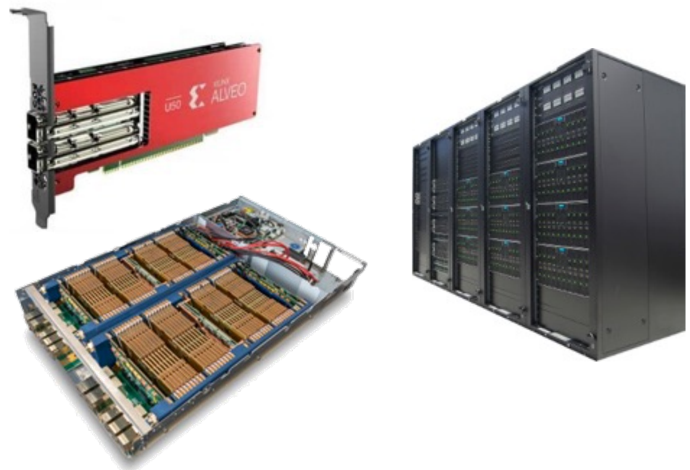
F. Suchert, et al. ECOOP'23

- MLIR for heterogeneous integration: System-level design and rich tool interfacing
- Convergence: HPC, Big data and machine learning

EVEREST

https://everest-h2020.eu



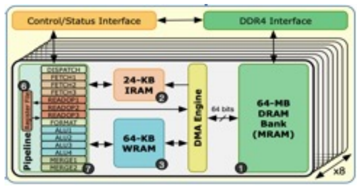


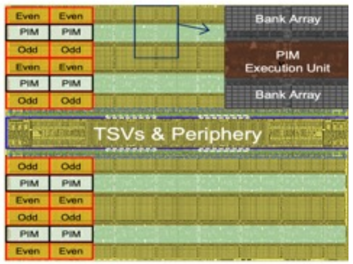
Webinar: https://youtu.be/h7sG9_JFqwk?si=UoM4CRCUvhJgUt3m

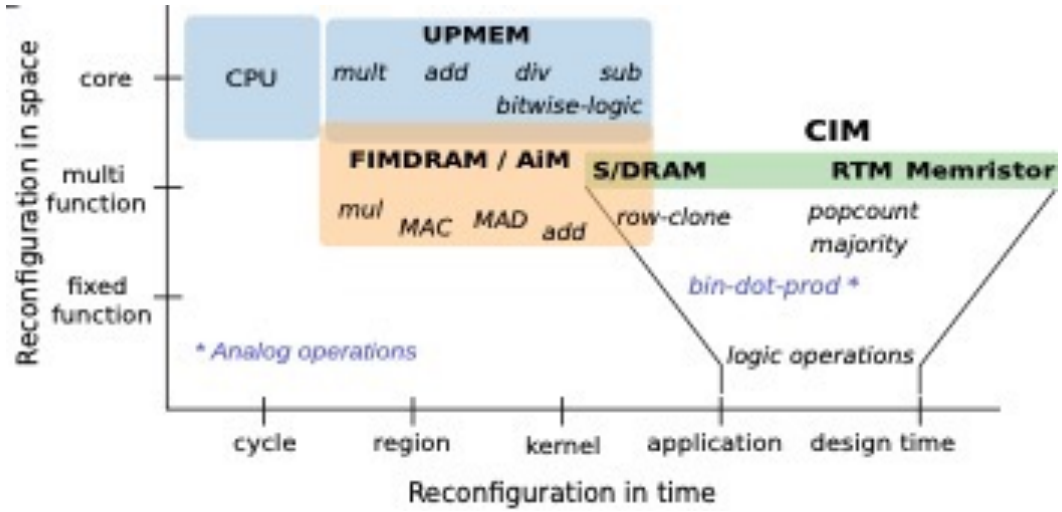
Near and in-memory computing

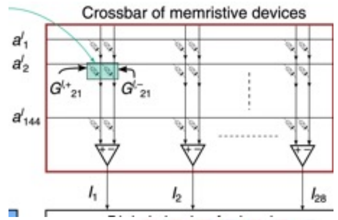
Rich landscape of designs

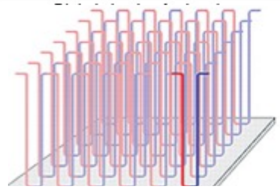
- Near-memory: Processors, logic close to memory
- In-memory (aka processing using memory): Leverage device properties

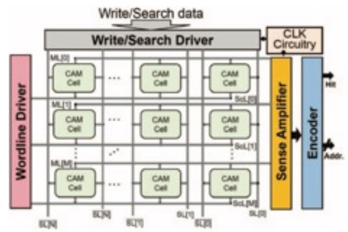








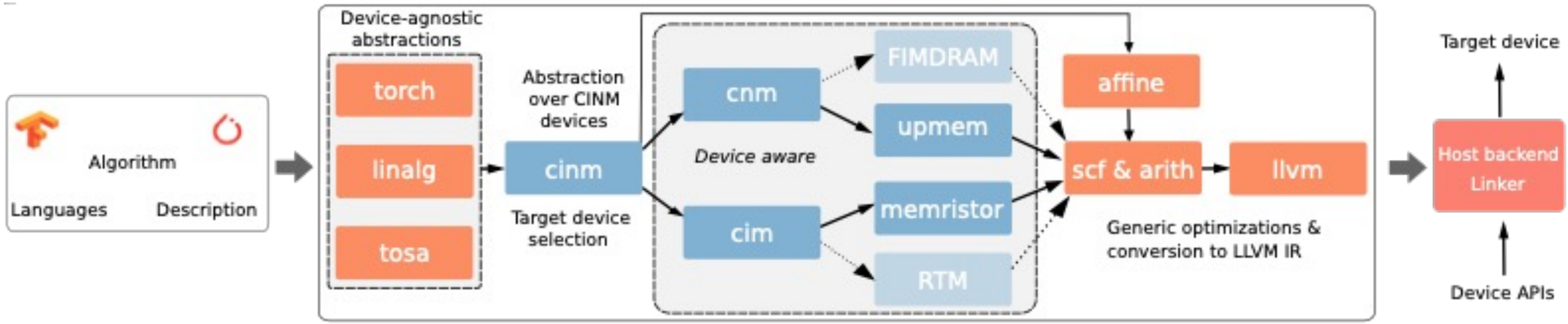




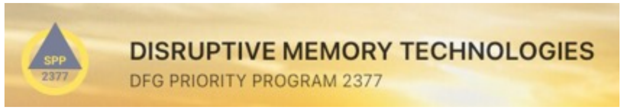
Samsung, Lee, Sukhan, et al. ISCA 2021
UPMEM by Gómez-Luna, Juan, et al. arXiv:2105.03814 (2021)
In-PCM Computing: Joshi, Vinay, et al. Nature Communications 11.1 (2020): 1-13.
CAM accelerators: Hu, Sharon, et al. 2021 IEDM

CINM: Generalized MLIR infrastructure

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

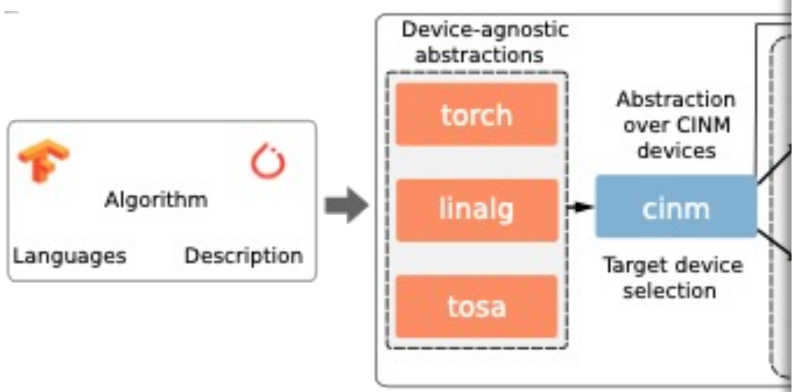


A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", arXiv, Aug 2023



CINM: Generalized MLIR infrastructure

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

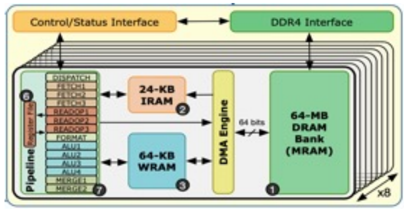


cinm.op.add/sub cinm.op.min/max	tensor<?xT> —"—	*same as lhs* —"—
cinm.op.and/or/xor cinm.op.popcount cinm.op.majority	—"— —"— —"—	—"— —"— —"—
cinm.op.sum cinm.op.exclusive_scan	—"— —"—	— —
cinm.op.transpose cinm.op.gemm cinm.op.gemv cinm.op.histogram	tensor<?x?xT> —"— —"— —"—	— tensor<?x?xT> tensor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

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

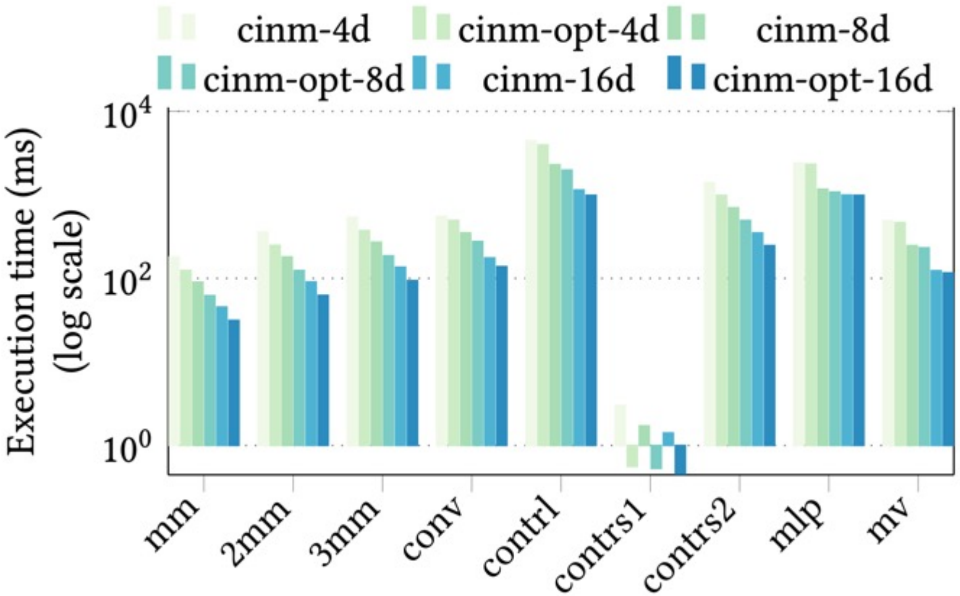
UPMEM example: Matmult

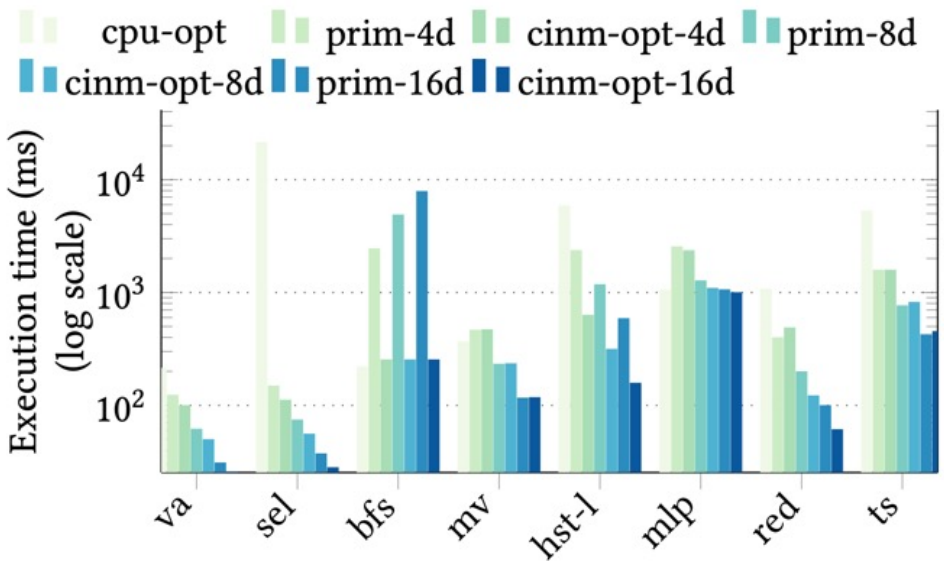
```
def mm(int32(64, 64) A, int32(64, 64) B) -> (int32(64, 64) C) {
  C(i,j) += A(i,k) * B(k,j)
           where i in 0:64, k in 0:64, j in 0:64
}
```




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

UPMEM example: Results



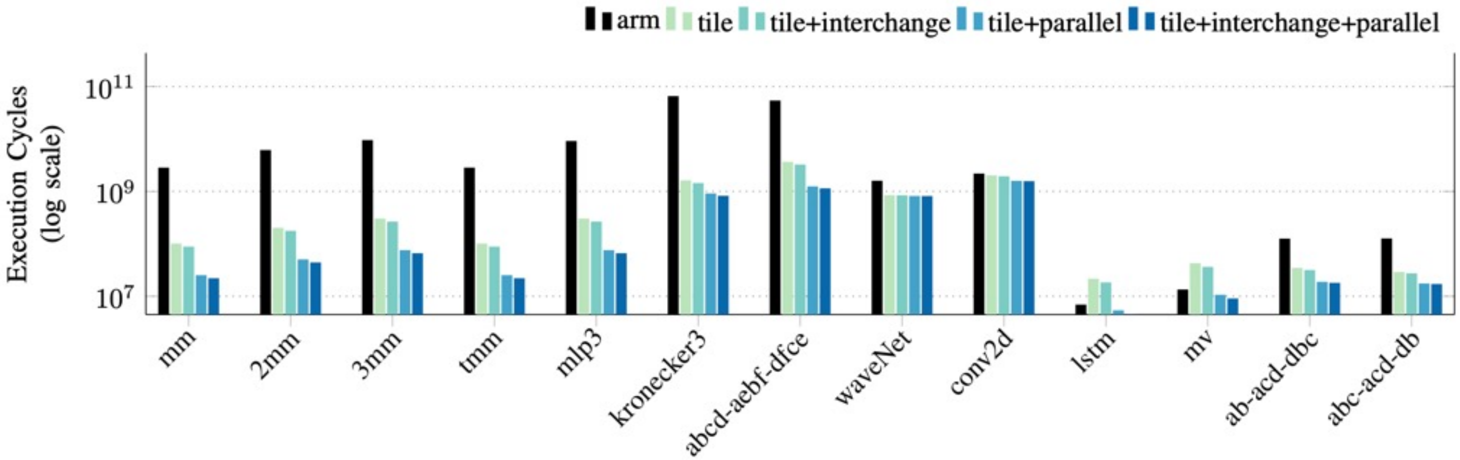


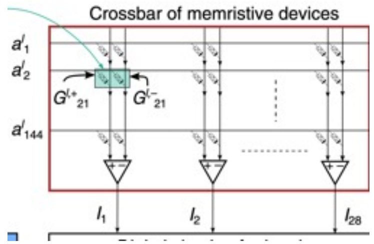
1-DIMM	128 DPUs
Xd	X x 128 DPUs

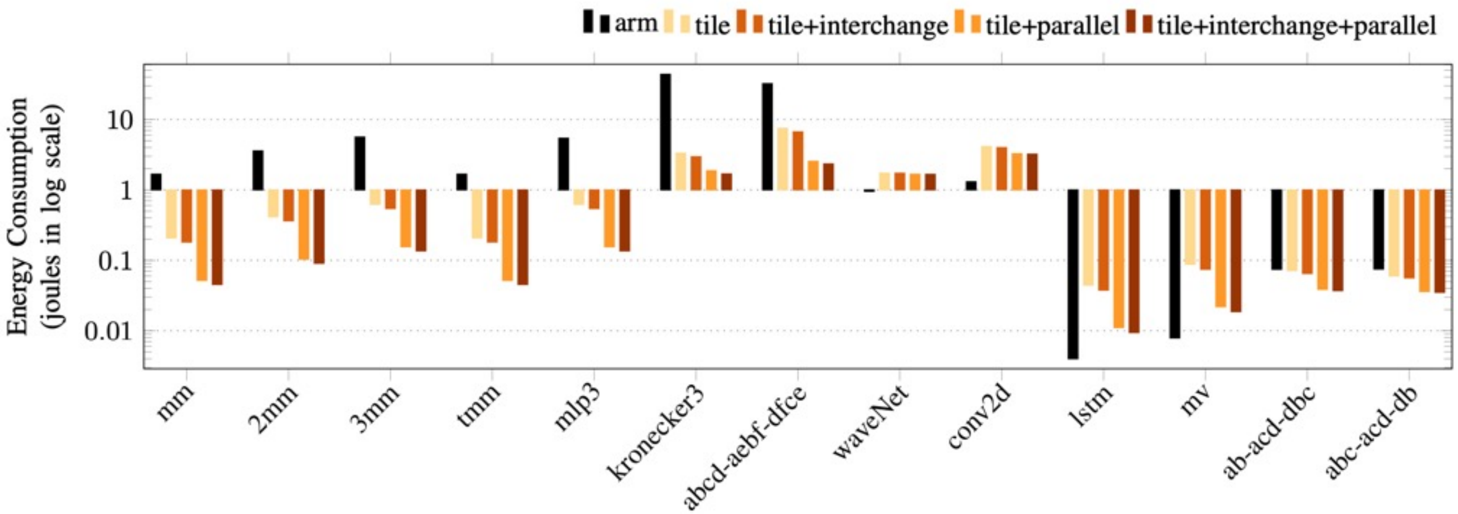
1.4x-1.5x faster than un-optimized versions (geomean)

2x-5x faster that CPU optimized
1.6x-2x faster than PrIM (manually optimized for UPMEM)

Optimization results: Crossbars beyond matmult



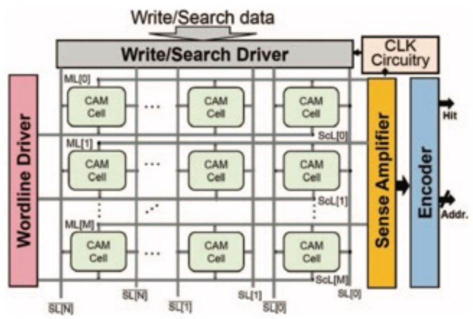


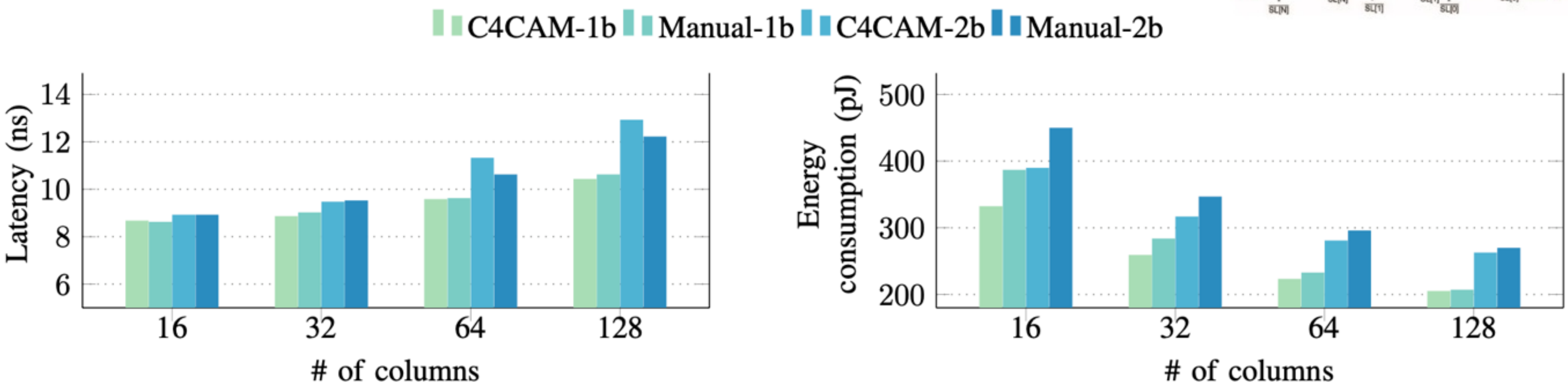


A. Siem … -to-End Machine
Learnin…

Content addressable memories (CAMs)

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





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

Summary

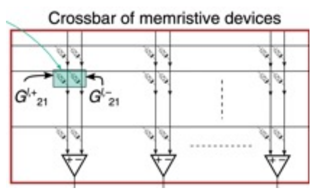
- Next generation programming for extreme heteogeneity
  - Domain-specific abstractions (**implicit**), compilation flows, …
  - Reconfigurable HW, HBM, data placement, near and in-memory computing

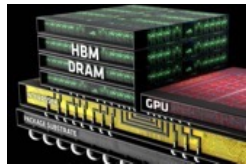
$$t = \left(s \otimes \left(s \otimes (s \otimes u)_{cz}{}^{xyz}\right)_{by}{}^{cxy}\right)_{ax}{}^{bcx}$$

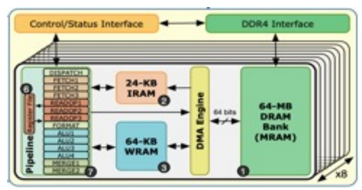
```
time loop
  start: 0   stop: 1000
  temporal method: explicit_euler
  spatial method: DC-PSE
```

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

- Challenges
  - Understanding and modeling primitives from down below
  - Maintainability and interoperability with low-level programming models
  - Optimization/DSE

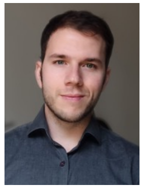


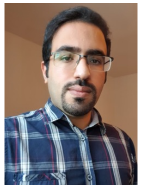


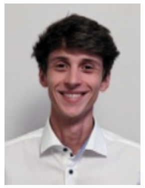


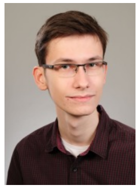
Thanks! & Acknowledgements


Hasna Bouraoui


João P. de Lima


Hamid Farzaneh


Clément Fournier


Karl Friebel


Dr. Asif Khan


Robert Khasanov


Alexander Brauckmann


Nesrine Khouzami


Dr. Steffen Köhler


Christian Menard


Julian Robledo


Lars Schütze


Felix Wittwer


Dr. Fazal Hameed

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

Funded by DFG

DISRUPTIVE MEMORY TECHNOLOGIES
DFG PRIORITY PROGRAM 2377

CO4RTM (450944241)
HetCIM (502388442)

EVEREST GA: 957269

AMD

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

6G-life
BMBF (16KISK001K)

STAATSMINISTERIUM FÜR WISSENSCHAFT KULTUR UND TOURISMUS
Freistaat SACHSEN

CHAIR FOR COMPILER CONSTRUCTION

References

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

[TOMS'18] 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.

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

[JCS'21] N. Khouzami, F. Michel, P. Incardona, J. Castrillon, I. F. Sbalzarini, "Model-based Autotuning of Discretization Methods in Numerical Simulations of Partial Differential Equations", In Journal of Computational Science, vol. 57, pp. 1–11, Dec 2021.

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

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

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

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

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

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

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

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

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

[ECOOP'23] Felix Suchert, et al. "ConDRust: Scalable Deterministic Concurrency from Verifiable Rust Programs", In Proceeding: 37th European Conference on Object-Oriented Programming (ECOOP 2023). Jul 2023.