CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Domain Specific Languages to Tame Heterogeneous and Emerging Computing Systems

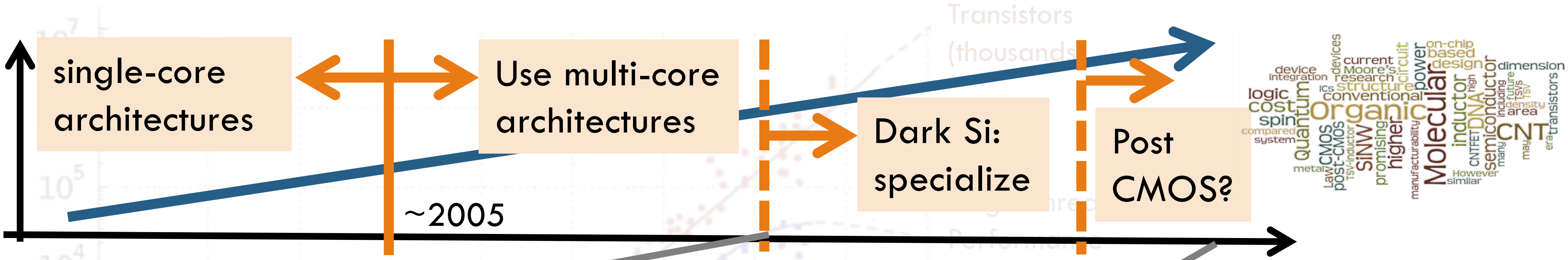
Jeronimo Castrillon

Chair for Compiler Construction (CCC)

TU Dresden, Germany

The Platform for Advanced Scientific Computing (PASC) Conference
Geneva (virtual), Switzerland
July 9, 2021

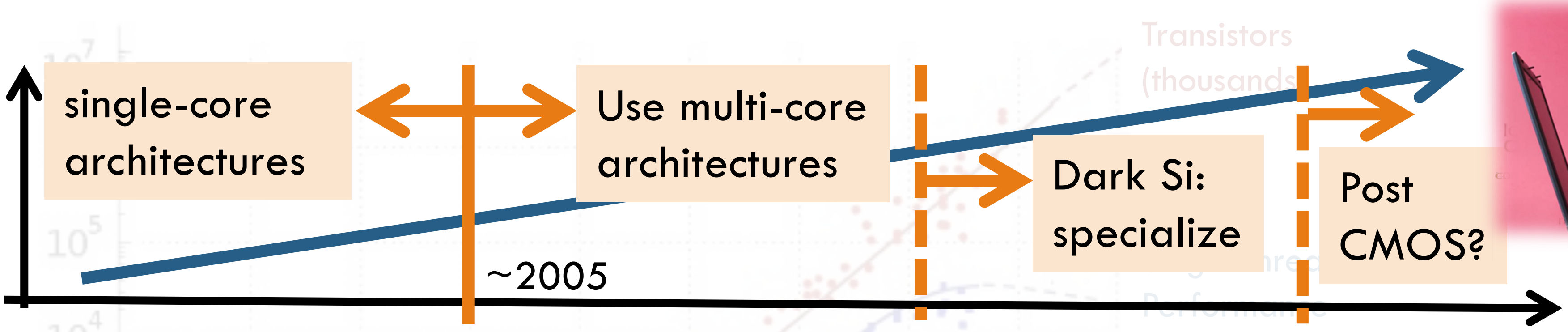
Evolution of computing

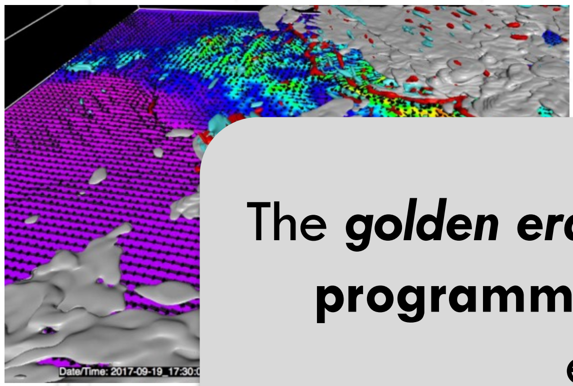


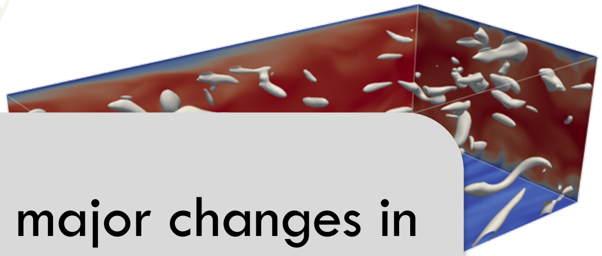
J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems", In IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 243-259, Jul 2018.

- Massive parallel and heterogeneous systems
- Specialization: TPUs, AI engines, PCM, …
- Interconnected & distributed computing
- Emerging non-volatile memories

Evolution of computing: Programming







The ***golden era*** in computer architecture requires major changes in **programming methods** to **democratize** heterogeneous and emerging high-performance computing

CIMA Founda

amics

What's wrong with good old sequential languages?

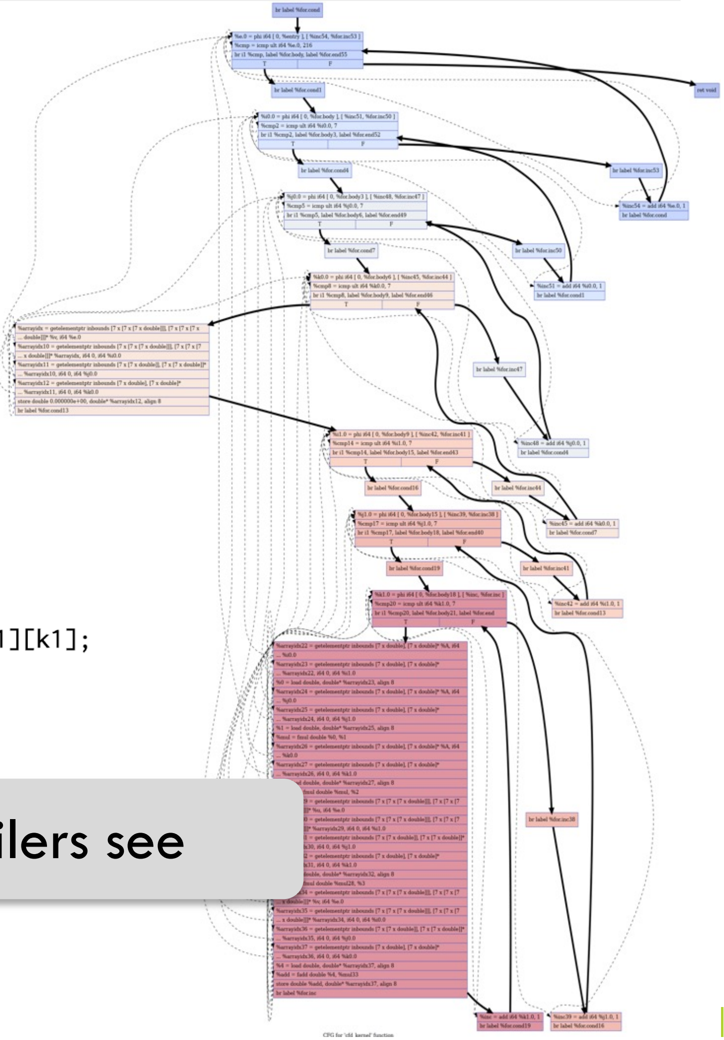
$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$

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 */
}
```

How many more times should we **optimize this manually?**



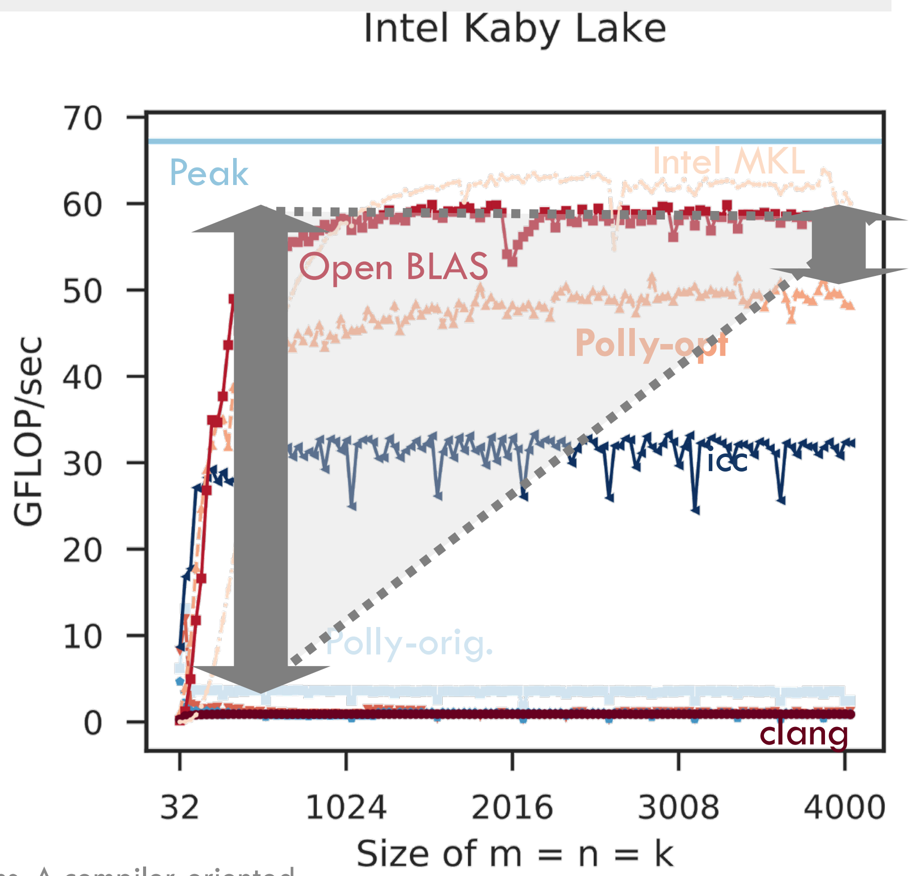
What compilers see

Polyhedral compilation: Hope for regular loops

- Recognize high-level patterns like **matrix multiply-and-add operation** (MMA)

"Our method attained the performance of vendor optimized BLAS libraries"

Complex and sensitive **pattern recognition** to help close the performance gap



R. Gareev, T. Grosser, M. Kruse. "High-performance generalized tensor operations: A compiler-oriented approach." ACM Transactions on Architecture and Code Optimization (TACO) 15.3 (2018): 34.

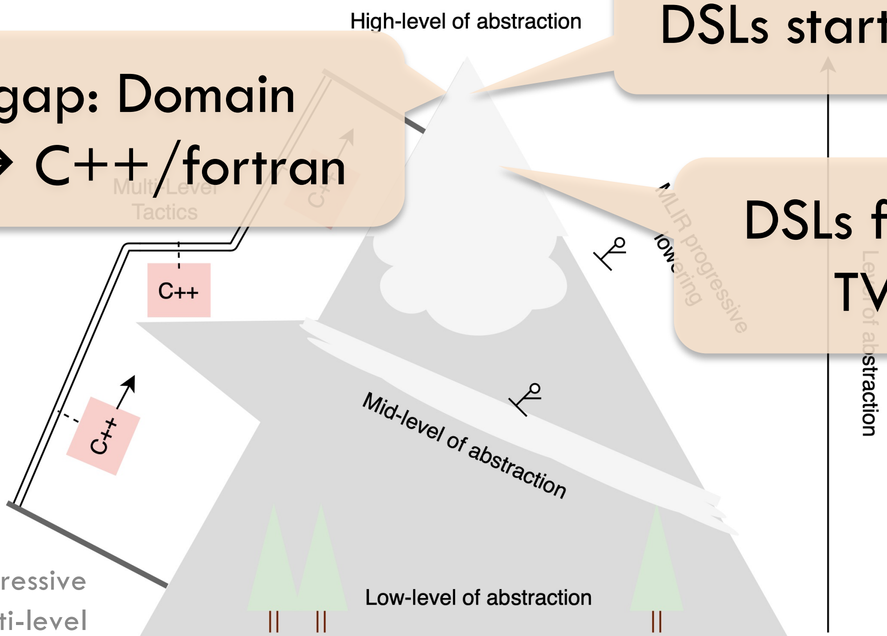
There is only su much we can do/reconstruct…

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

DSLs start here!

Bridge gap: Domain experts → C++/fortran

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



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

```
while(!queue.empty())
{
  // Dequeue a vertex from queue
  s = queue.front();
  queue.pop_front();

  // Apply function f to s, accumulate values
  result += f(s);

  // Get all adjacent vertices of s.
  // If an adjacent node hasn't been visited,
  // then mark it as visited and enqueue it
  for(i=adj[s].begin(); i!=adj[s].end(); ++i)
  ...
        queue.push_back(*i);
      }
    }
  }

  return result;
}
```

S. Manilov, C. Vasiladiotis, B. Franke. "Generalized profile-guided iterator recognition." CC 2018.

Examples (1): Tensors expressions (CFD, ML)

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

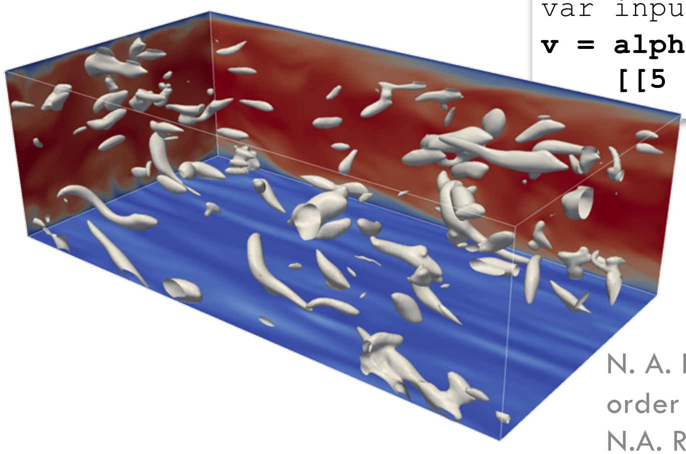
$$\mathbf{v}_e = (\mathbf{A} \otimes \mathbf{A} \otimes \mathbf{A})\, \mathbf{u}_e$$

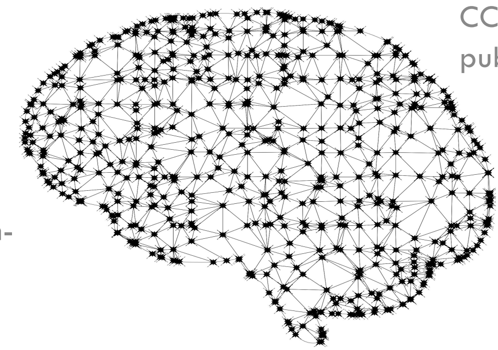
Interpolation kernel

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

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

Fortran and C++ integration





CC0
public domain

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

Semantic gap → performance gap

```
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
```

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

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

What we (naively) code

```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 */
}
```

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

Closing the performance gap

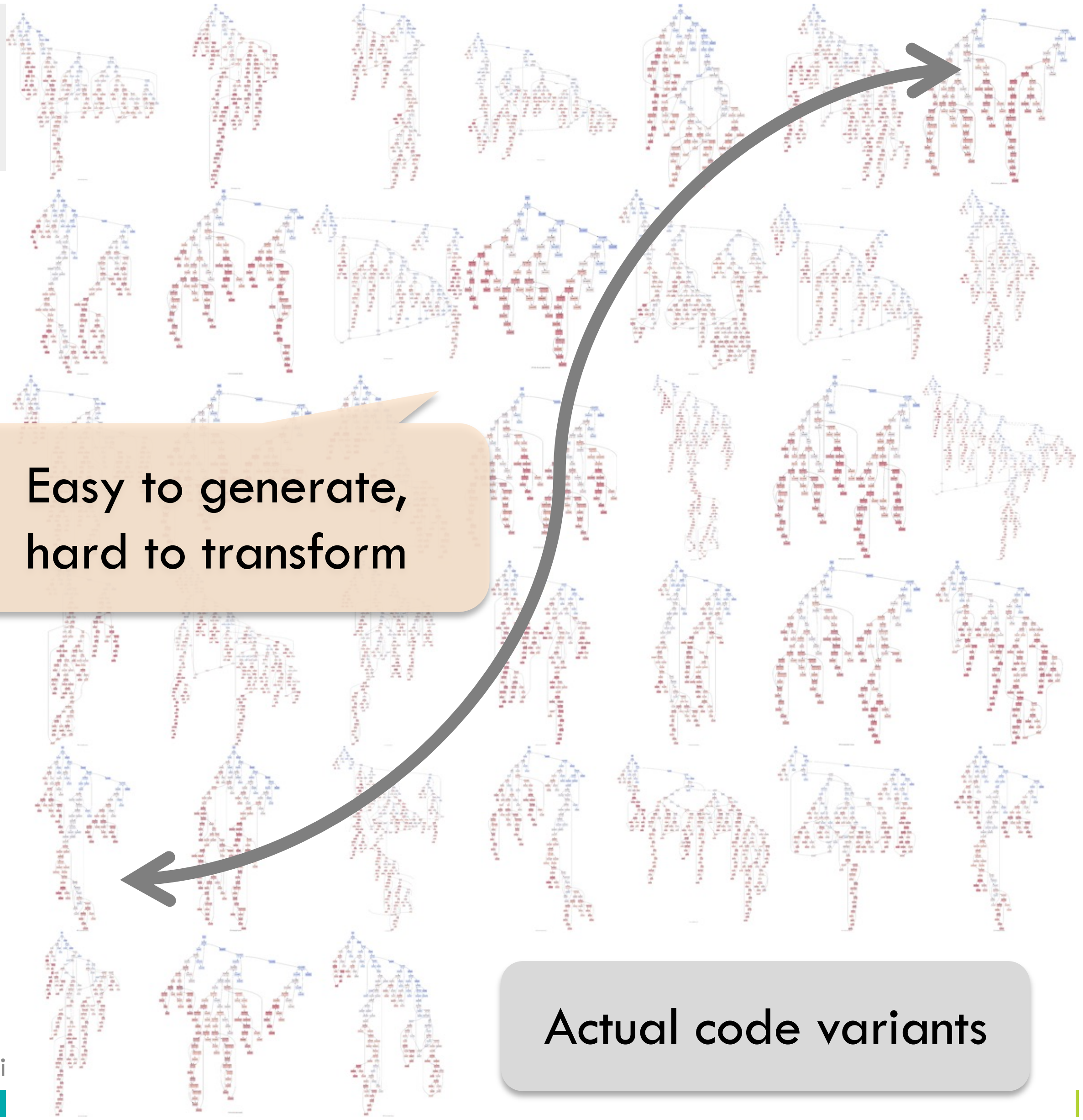
- Not really optimization magic
  - Leverage expert knowledge
  - Algebraic identities

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

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

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

N. A. Rink, et al. “CFDlang: High-level code generation for high-order methods in fluid dynamics”. RWDSL’18.
A. Susungi, et al., "Towards Compositional and Generative Tensor Optimizations", GPCE’17 pp. 169–175.



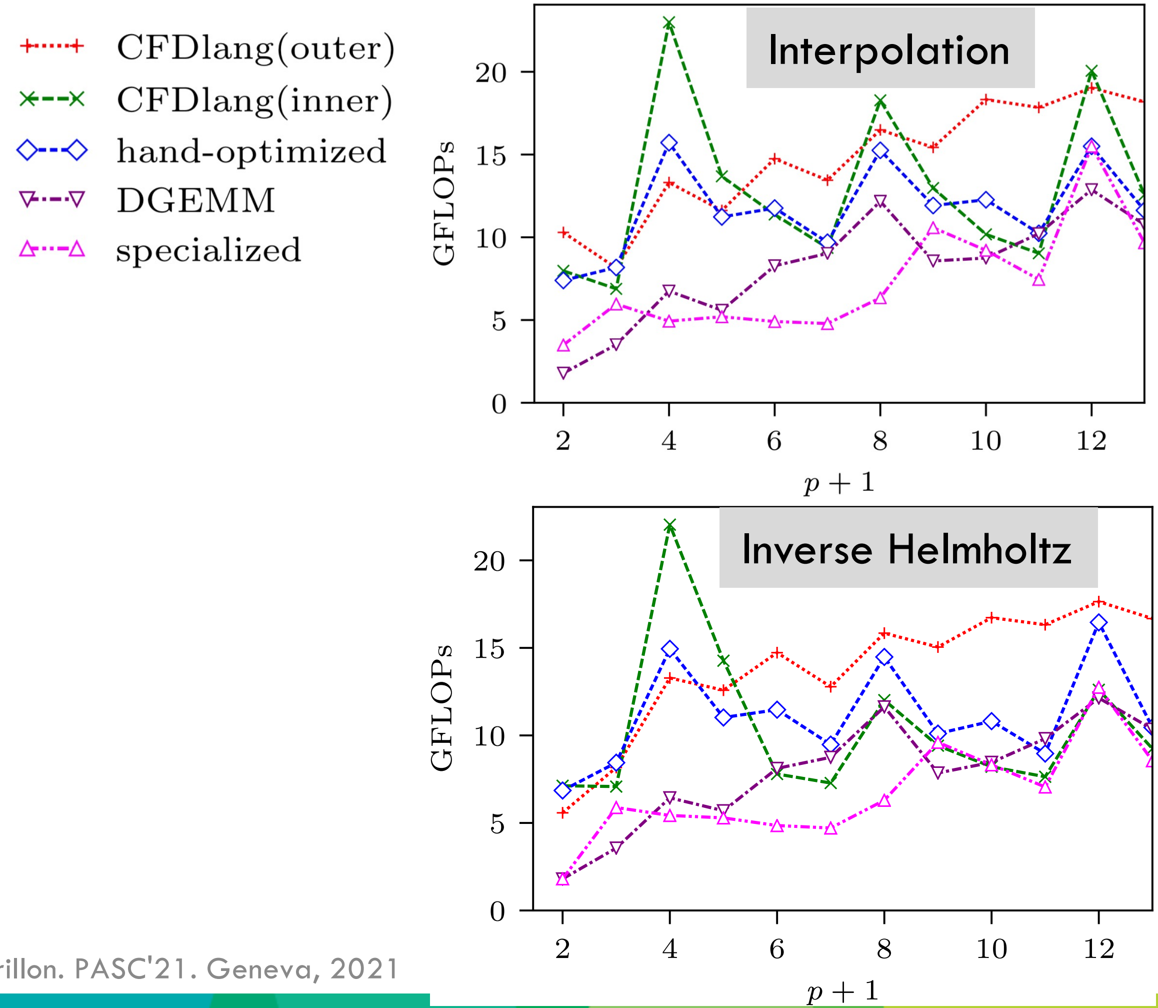
Closing the performance gap

- 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., "Towards Compositional and Generative Tensor Optimizations", GPCE’17 pp. 169–175.

TeML: Meta-programming for tensor optimizations

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

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

$$
\begin{aligned}
&\mathcal{E}_l[\![\texttt{interchange}(l, r_1, r_2)]\!] = \\
&\quad \lambda\sigma.\text{let } \langle i_1, \ldots \langle i_{r_1}, \ldots \langle i_{r_2}, xs \rangle \ldots \rangle \ldots \rangle = \sigma(l) \\
&\quad \text{in } \langle i_1, \ldots \langle i_{r_2}, \ldots \langle i_{r_1}, xs \rangle \ldots \rangle \ldots \rangle
\end{aligned}
$$

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

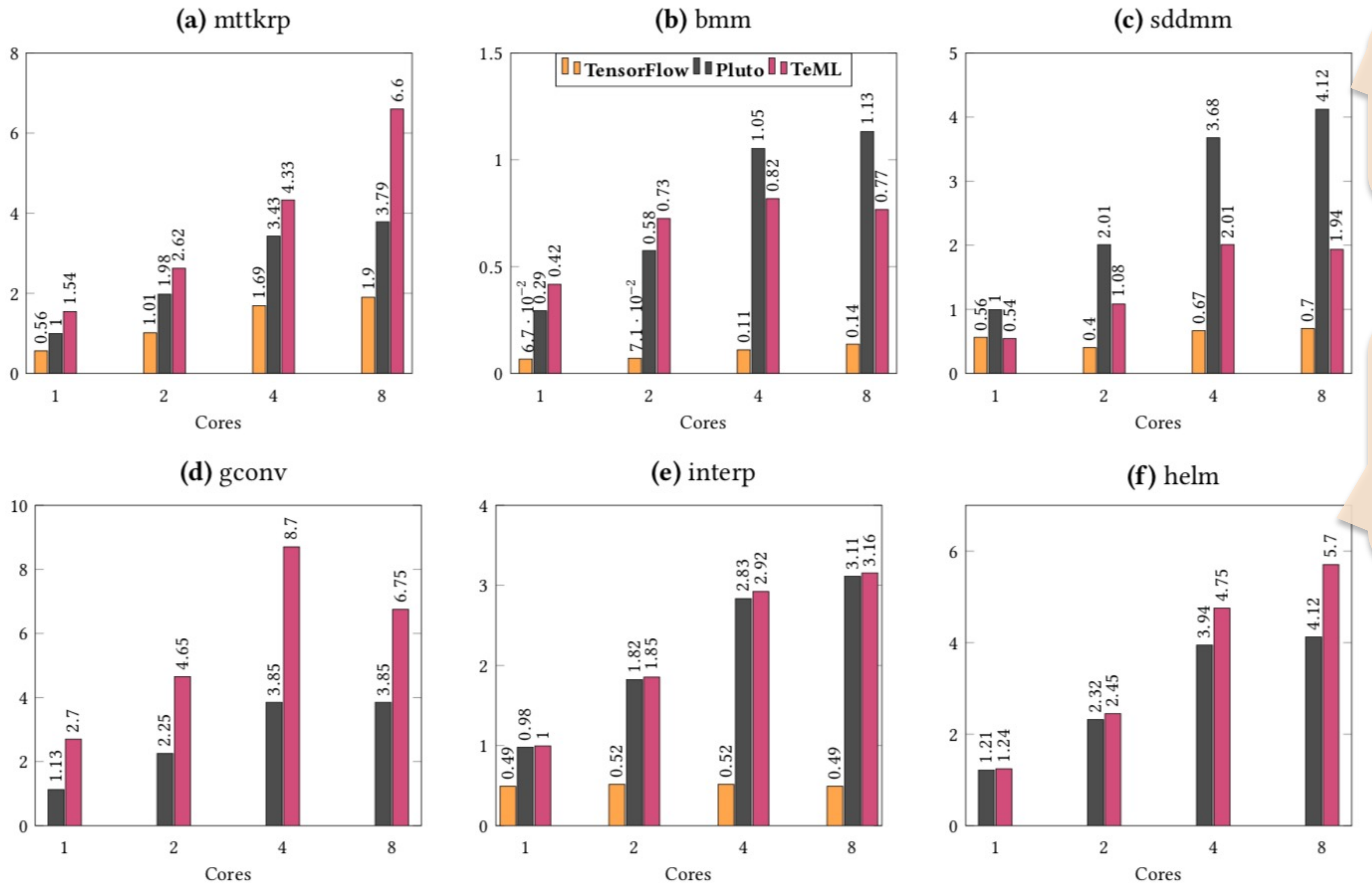
Formally defined transformation primitives

Higher-level transformations via composition

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

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

Meta-programming for optimizations: Results



Performance of Pluto could be reproduced

Higher abstraction → more optimization potential

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

TeIL: Formal language – added value

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

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

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

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

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

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

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

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

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

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

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

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

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

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

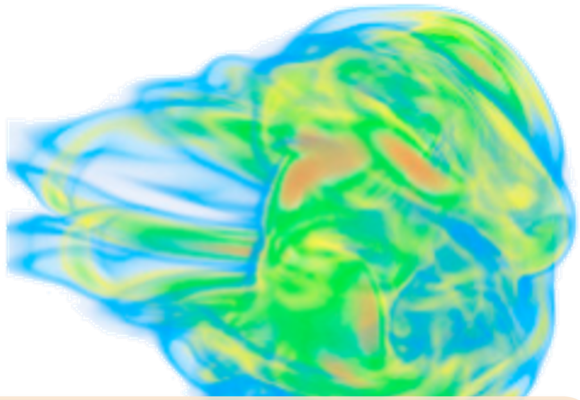
N.A. Rink, N. A. and J. Castrillon. “TeIL: a type-safe imperative Tensor Intermediate Language”, ARRAY’19, pp. 57-68

Examples (2): 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: pa
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, …

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

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

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

- Template meta-programming

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

What we want

What we code
(already quite abstracted!)

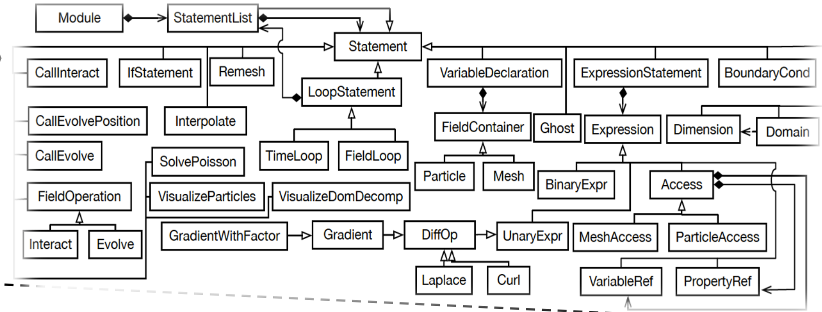
Model-to-model code generation

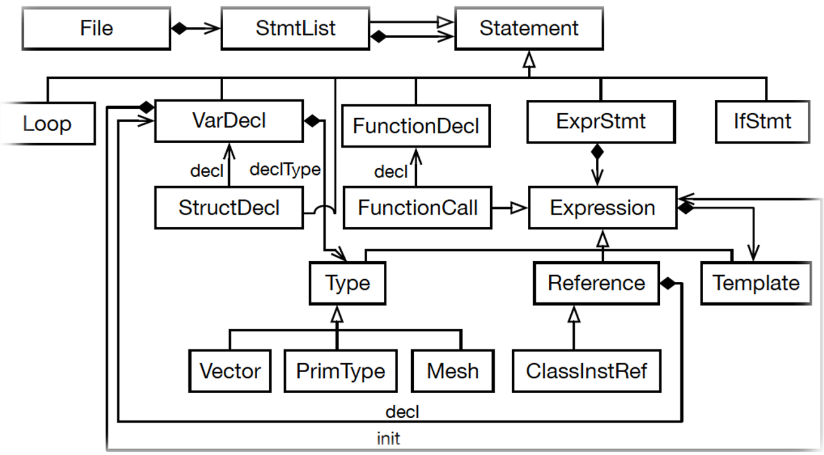
OpenPME DSL

```
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 ] = ...
```





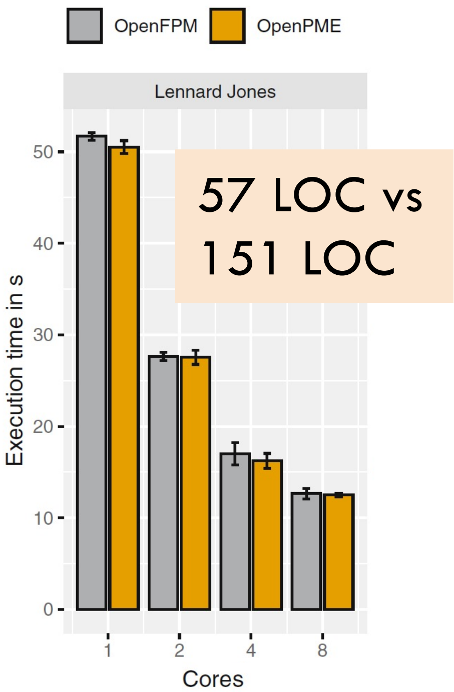
OpenFPM

```
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]=
}
```

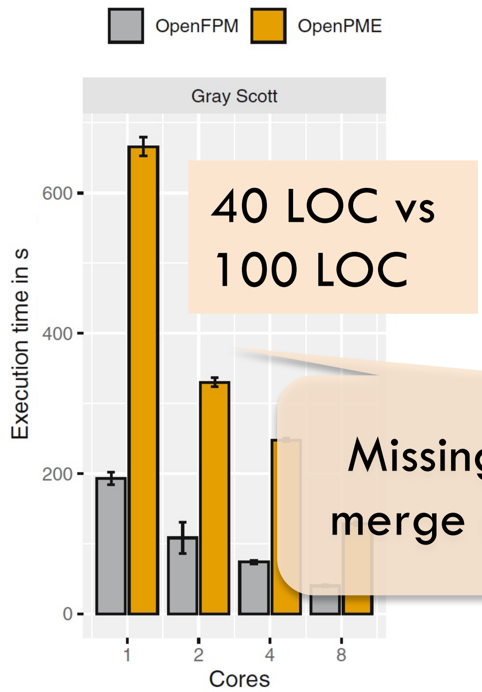
N Khouzami, et al., "The OpenPME Problem Solving Environment for Numerical Simulations", In ICCS'21 pp. 614–627, Jun 2021

Closing the performance gap

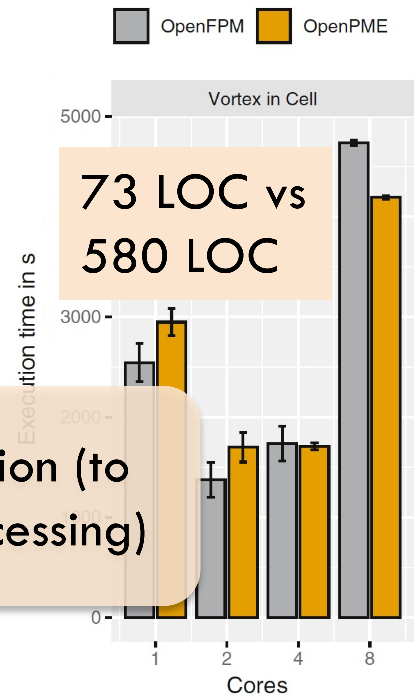
Lennard Jones
(particles, discrete)



Gray-Scott
(mesh, continuous)



Vortex in Cell
(hybrid, continuous)

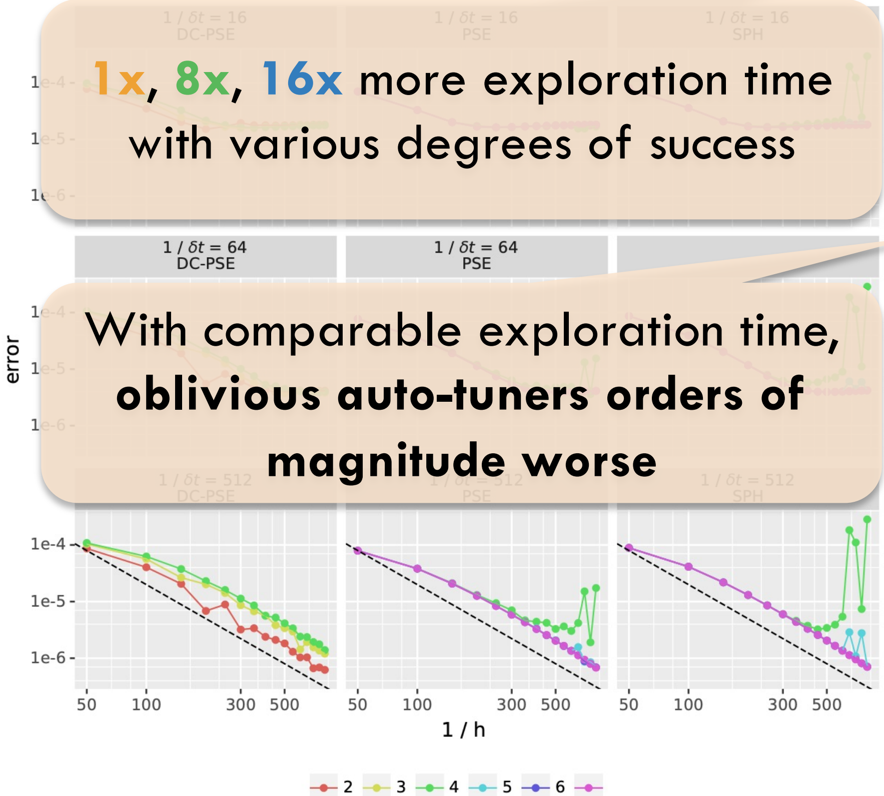


Missing loop fusion (to merge mesh processing)

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

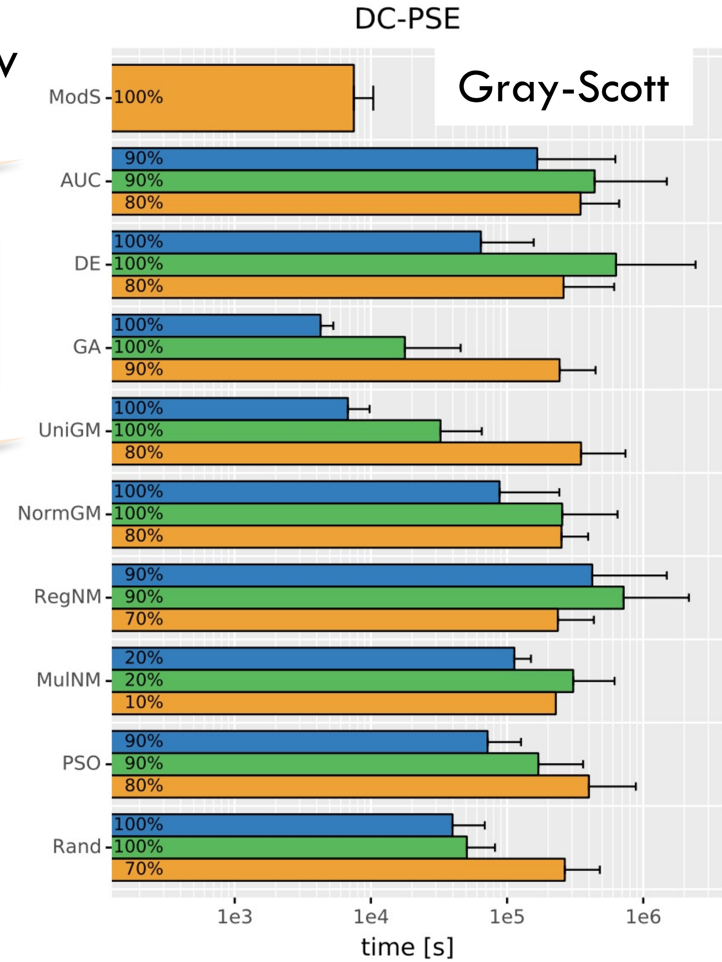
Higher-level optimizations

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



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

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



Formal language – added value

- Mathematical expressions: Possible to explore performance-accuracy trade-offs
- Type system: High-level semantics checks (e.g., units)

$$\textsc{Var}\ \frac{\Gamma(v)=\tau}{\Gamma\vdash v:\tau}\qquad \textsc{VarDecl}\ \frac{}{\Gamma\vdash \tau\ x:\Gamma\cup\{x=\tau\}}\qquad \textsc{VarInit}\ \frac{\Gamma\vdash e:\tau' \quad \tau'\leq\tau}{\Gamma\vdash \tau\ x=e:\Gamma\cup\{x=\tau\}}$$

$$\textsc{Paren}\ \frac{\Gamma\vdash e:\tau}{\Gamma\vdash (e):\tau}\qquad \textsc{Assign}\ \frac{\Gamma\vdash x:\tau \quad \Gamma\vdash e:\tau' \quad \tau'\leq\tau}{\Gamma\vdash x=e:\tau}$$

$$\textsc{VecAcc}\ \frac{\Gamma\vdash v:\mathbb{V}\langle\tau\rangle \quad \Gamma\vdash i:\mathbb{Z} \quad i\geq 0}{\Gamma\vdash v[i]:\tau}\qquad \textsc{MatAcc}\ \frac{\Gamma\vdash m:\mathbb{M}\langle\tau\rangle \quad \Gamma\vdash i,j:\mathbb{Z} \quad i,j\geq 0}{\Gamma\vdash m[i][j]:\tau}$$

$$\textsc{PartScaAcc}\ \frac{\Gamma\vdash p:\mathbb{P} \quad \Gamma\vdash f:\mathcal{E}\langle\tau,1\rangle}{\Gamma\vdash p\to f:\tau}\qquad \textsc{PartVecAcc}\ \frac{\Gamma\vdash p:\mathbb{P} \quad \Gamma\vdash f:\mathcal{E}\langle\tau,n\rangle, n\geq 2}{\Gamma\vdash p\to f:\mathbb{V}\langle\tau\rangle}$$

$$\textsc{Unary}\ \frac{\Gamma\vdash e:\tau \quad \tau_\ominus(\tau)\neq\bot}{\Gamma\vdash \ominus e:\tau_\ominus(\tau)}\qquad \textsc{BinLog}\ \frac{\Gamma\vdash e_1:\mathbb{B} \quad \Gamma\vdash e_2:\mathbb{B}}{\Gamma\vdash e_1\otimes_{log} e_2:\mathbb{B}}$$

$$\textsc{BinRel}\ \frac{\Gamma\vdash e_1:\tau_1 \quad \Gamma\vdash e_2:\tau_2 \quad \tau_\otimes(\tau_1,\tau_2)\neq\bot}{\Gamma\vdash e_1\otimes_{rel} e_2:\mathbb{B}}\qquad \textsc{BinAri}\ \frac{\Gamma\vdash e_1:\tau_1 \quad \Gamma\vdash e_2:\tau_2 \quad \tau_\otimes(\tau_1,\tau_2)\neq\bot}{\Gamma\vdash e_1\otimes_{arith} e_2:\tau_\otimes(\tau_1,\tau_2)}$$

$$\textsc{ErrUnary}\ \frac{\Gamma\vdash e:\tau \quad \tau_\ominus(\tau)=\bot}{\Gamma\vdash \ominus(e):\mathbb{E}}\qquad \textsc{ErrBin}\ \frac{\Gamma\vdash e_1:\tau_1 \quad \Gamma\vdash e_2:\tau_2 \quad \tau_\otimes(\tau_1,\tau_2)=\bot}{\Gamma\vdash e_1\otimes e_2:\mathbb{E}}$$

$$\ominus\in\{-,!,\sqrt{\ }\},\ \otimes_{arith}\in\{+,-,*,/,a^b\},\ \otimes_{log}\in\{\&\&,||\},\ \otimes_{rel}\in\{==,!=,<,>,<=,>=\}$$

$$\mathbb{Z}=\mathit{Integer},\ \mathbb{R}=\mathit{Real},\ \mathbb{P}=\mathit{Particle},\ \mathbb{V}=\mathit{Vector},\ \mathbb{M}=\mathit{Matrix},\ \mathcal{E}=\mathit{Field/Property},\ \mathbb{E}=\mathit{Error}$$

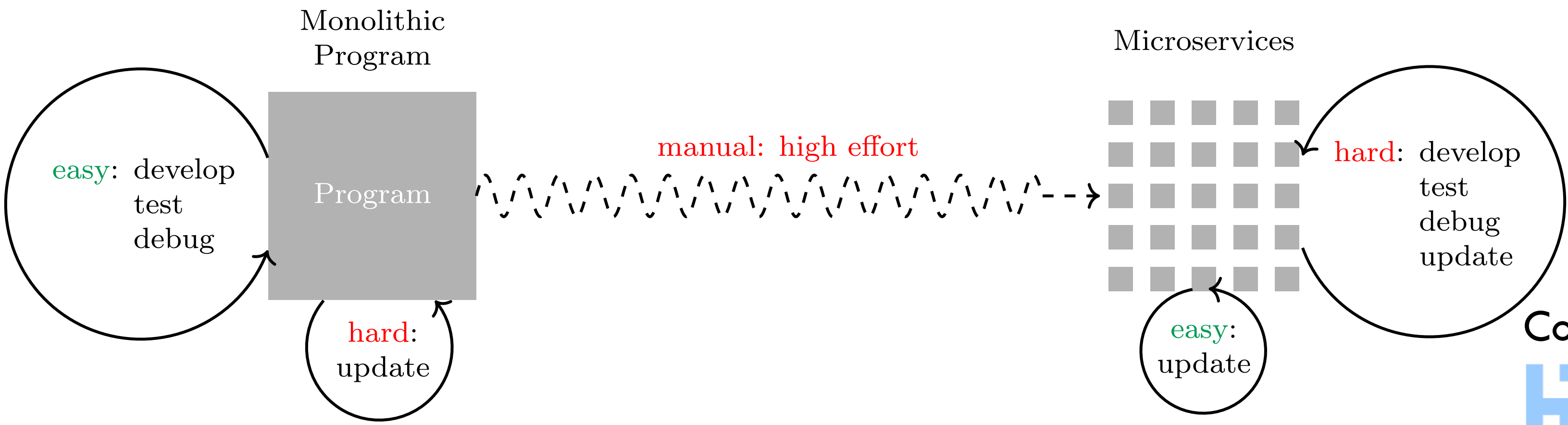
S. Karol, et al. "A Domain-Specific Language and Editor for Parallel Particle Methods", In ACM TOMS'18, vol. 44, no. 3, pp. 32, Mar 2018.

Examples (3): Big data (only briefly)

- Dataflow IR from a seqentual syntax (Rust or Java-like)

```
let v1_1_0 = join(customer, sales, cc_sk, c_sk);
let v1_1 = join(v1_1_0, date_dim, sd_sk, d_d_sk);
let gs = group_by(v1_1, gb_key);
let mut v1 = Vec::new();
for group in gs {
    let result = compute(group, sale_type);
    v1.push(result);
}
```

Sequential code (implicit parallelism)



Collaboration with barkhausen institut

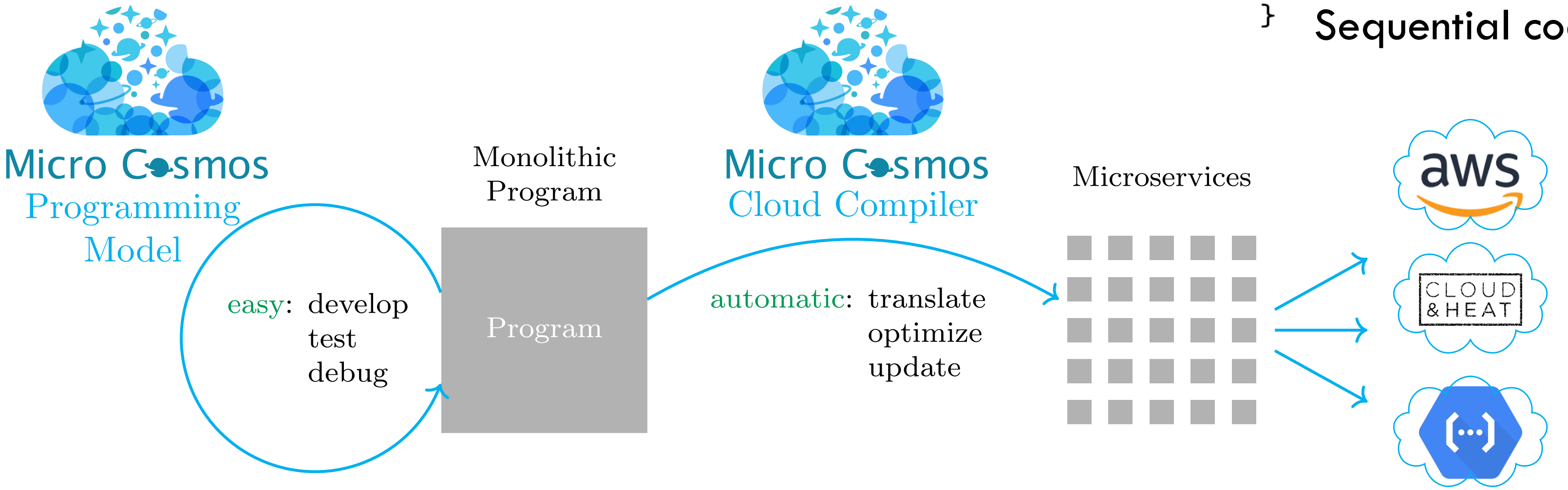
Chair for Compiler Construction

Examples (3): Big data (only briefly)

- Dataflow IR from a seqentual syntax (Rust or Java-like)
- IR to abstract from **“cloud ISAs”**

```
let v1_1_0 = join(customer, sales, cc_sk, c_sk);
let v1_1 = join(v1_1_0, date_dim, sd_sk, d_d_sk);
let gs = group_by(v1_1, gb_key);
let mut v1 = Vec::new();
for group in gs {
    let result = compute(group, sale_type);
    v1.push(result);
}
```

Sequential code (implicit parallelism)



Collaboration with barkhausen institut

S. Ertel, A. Goens, J. Adam, J. Castrillon, "Compiling for Concise Code and Efficient I/O", Proceedings of the 27th International Conference on Compiler Construction (CC 2018), ACM, pp. 104–115,

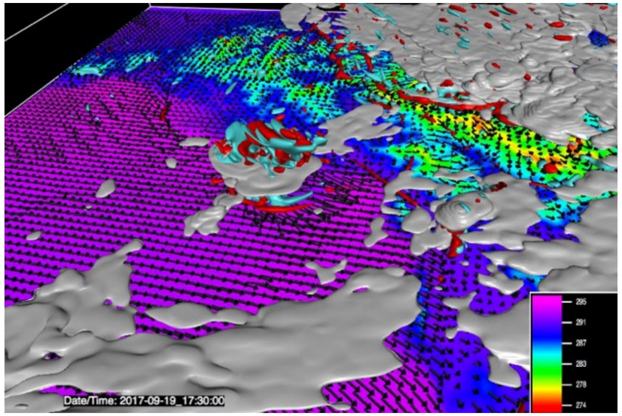
CHAIR FOR COMPILER CONSTRUCTION

EVEREST: Efficient large-scale heterogeneous computing

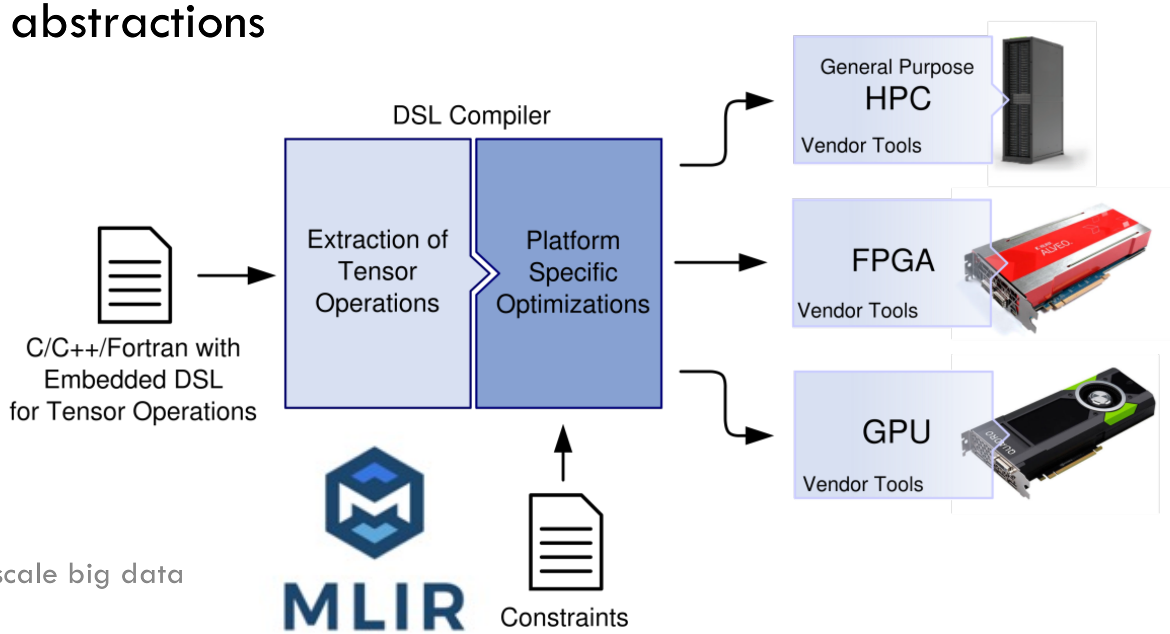
- Current work on large-scale EU H2020 project Everest
  - Stencil and Tensor Operations in Weather Modelling (WRF)
  - Interplay orchestration (dataflow) and kernels
  - MLIR framework for reusable abstractions

EVEREST

https://everest-h2020.eu



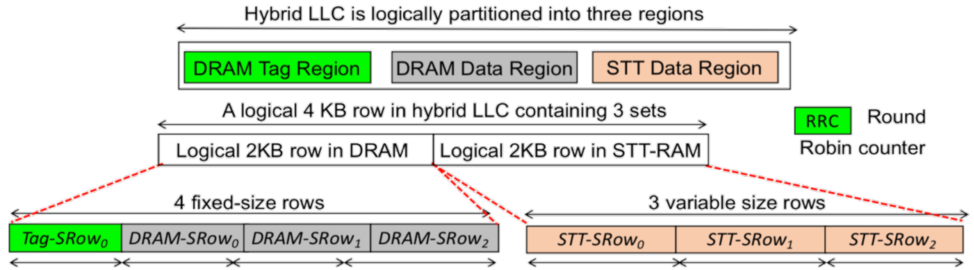
CIMA Foundation



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

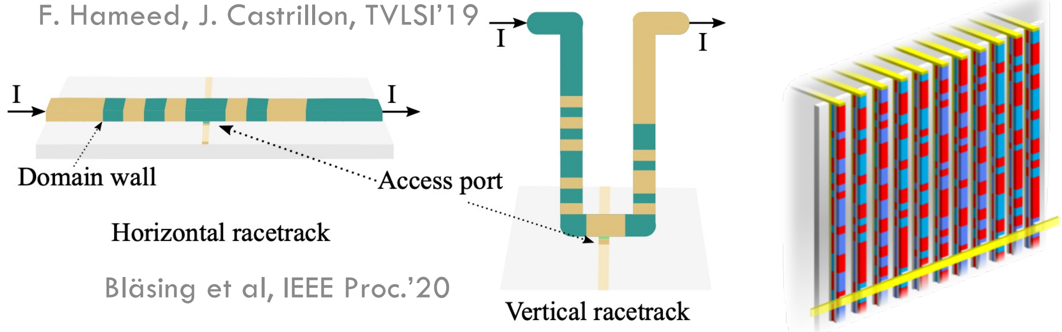
Challenges ahead: Emerging memories

- Example: Hybrid STT-/DRAM
  - Placement and layout optimization
  - Hints for memory controllers



F. Hameed, J. Castrillon, TVLSI'19

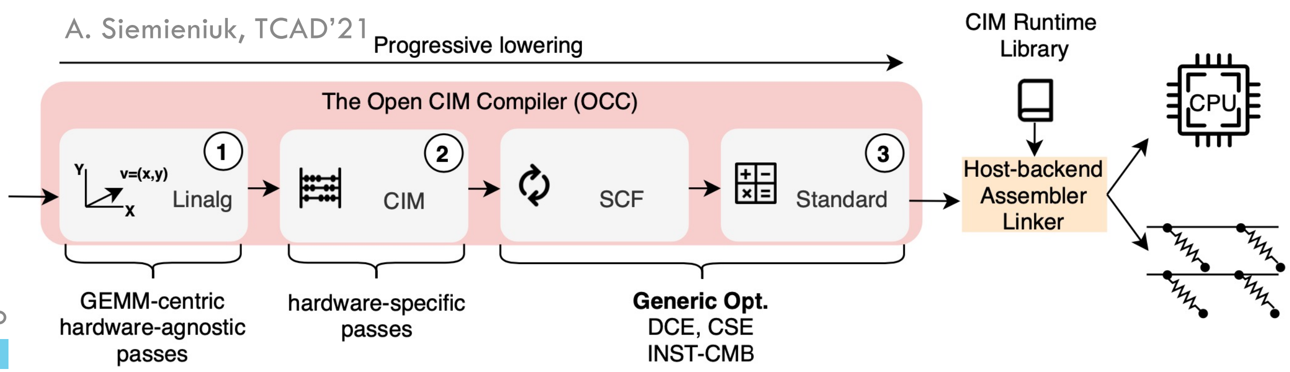
- Racetrack memories
  - Extreme density
  - Sequential bit access per cell



Bläsing et al, IEEE Proc.'20

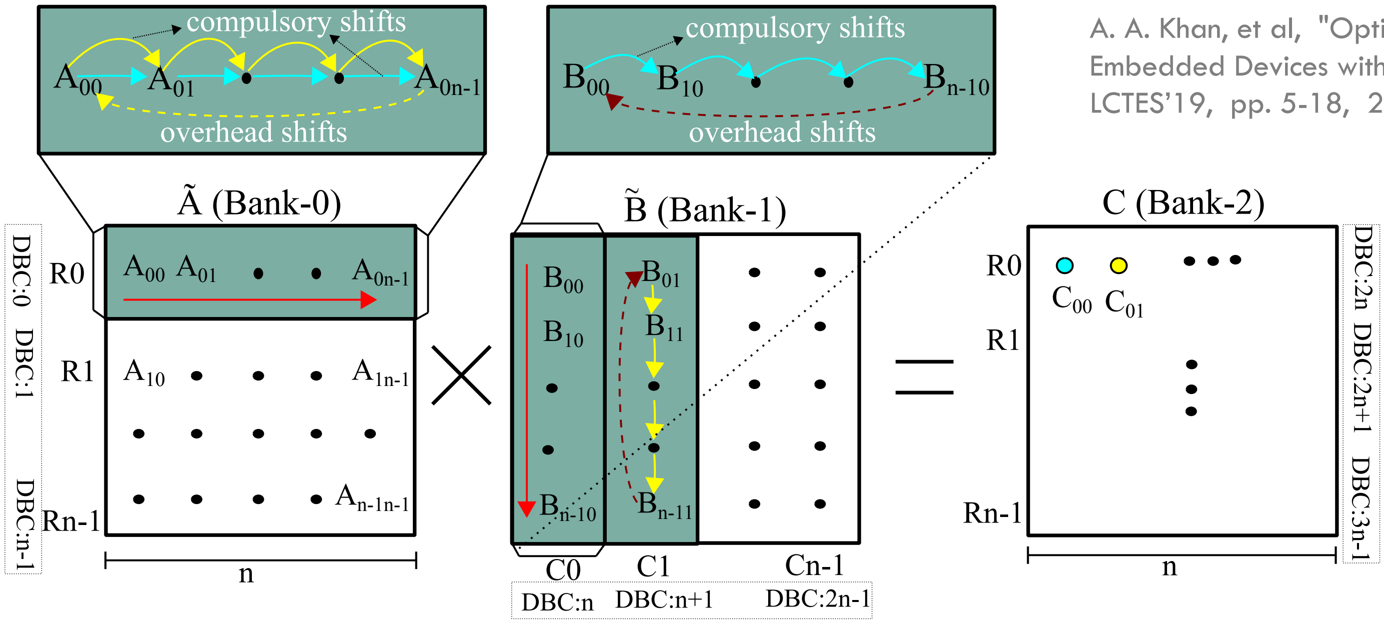
- Memristive accelerators
  - In-memory computing
  - Compiler abstractions

A. Siemieniuk, TCAD'21



Architecture and data layout optimization

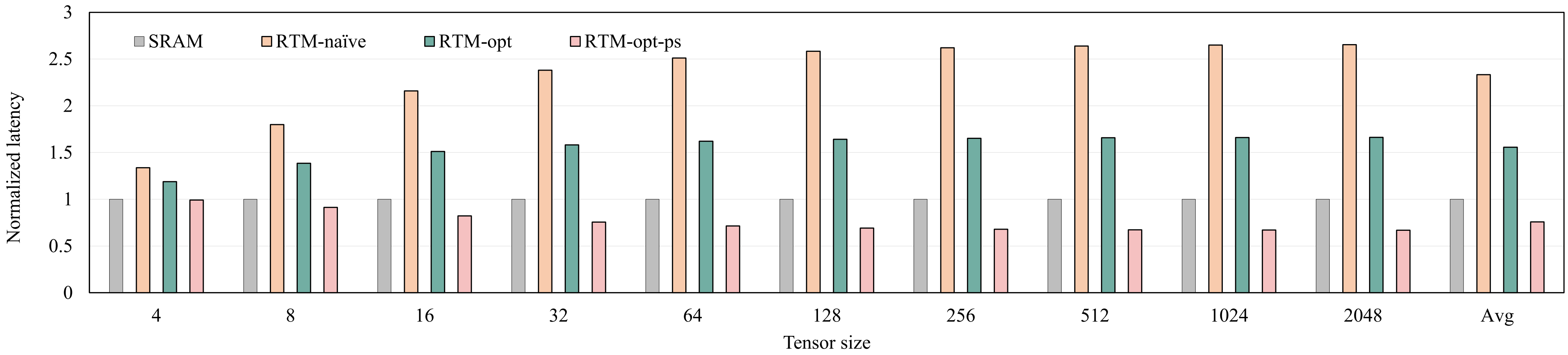
- Underlying idea: Zig-zag through data to reduce number of shifts
  - Exploit explicit patterns in high-level DSLs
  - Recognize patterns with polyhedral compilers



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

Latency comparison vs SRAM

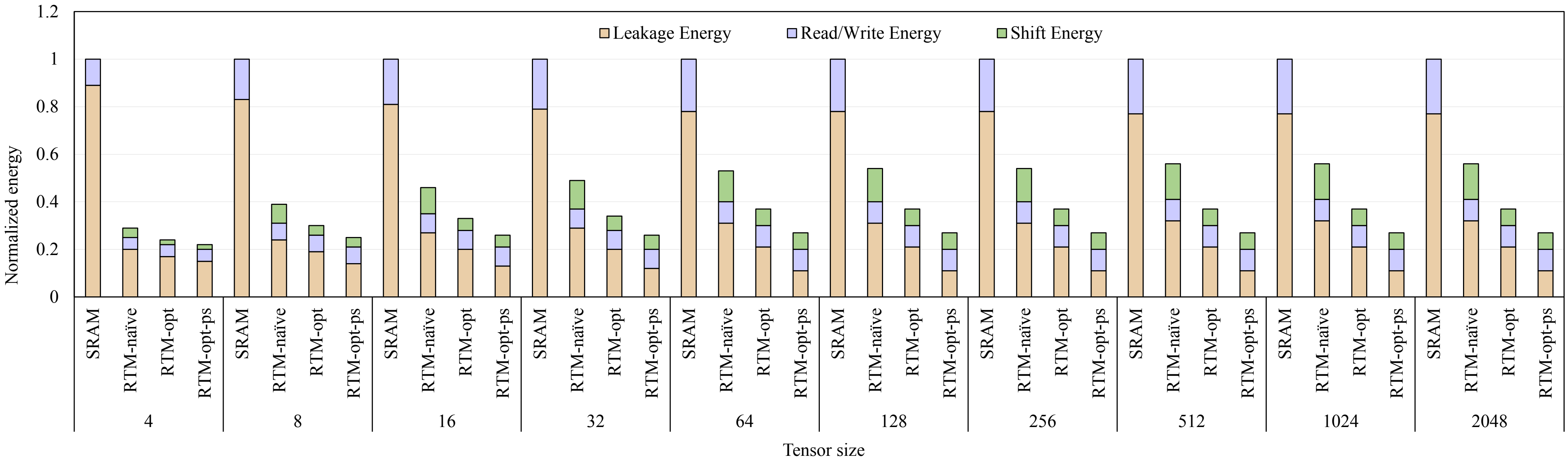
- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)



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

Energy comparison vs SRAM

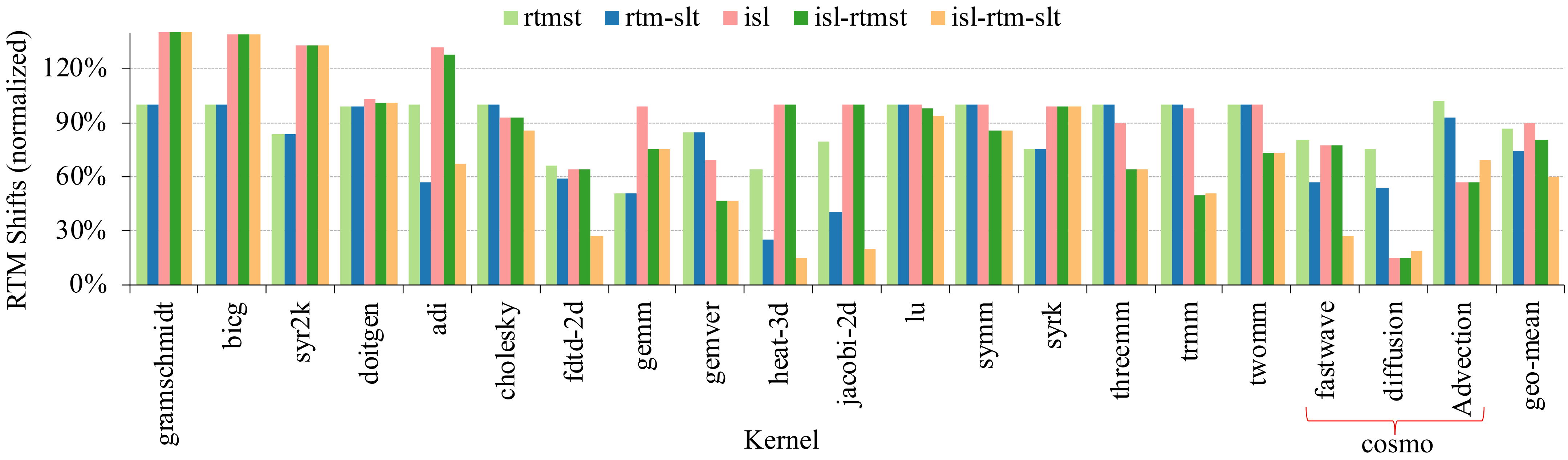
- Higher savings due to less leakage power
- 74% average improvement (in addition to savings due to DRAM placement)



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

Generalization to stencils and other kernels

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



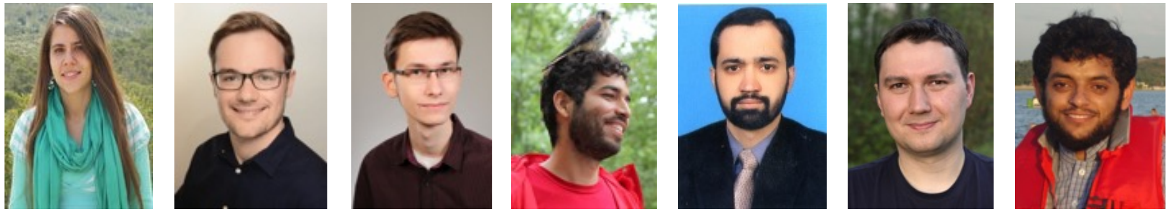
A. A. Khan, et al., "Polyhedral Compilation for Racetrack Memories", In IEEE TCAD'20, vol. 39, no. 11, pp. 3968-3980, Oct 2020.

Summary

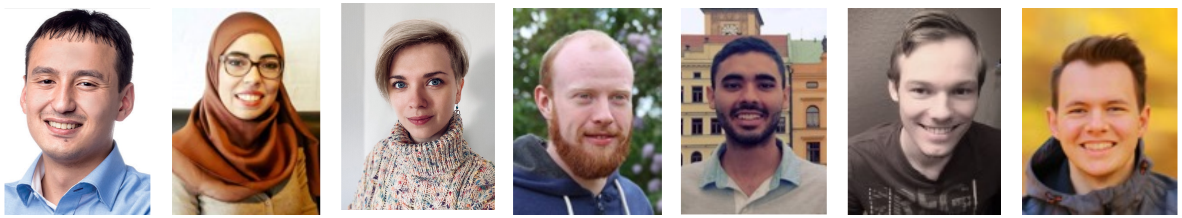
- Tame ever-increasing system complexity
  - Still highly-relevant optimizing compilers (polyhedral, …)
  - DSL examples: expose higher semantics (efficiency, productivity)
  - Higher semantics key for emerging accelerators/systems!

- Moving forward
  - Semantic-preserving transformations
  - Larger use cases (e.g., WRF in the context of EVEREST)
  - Common abstraction across novel paradimgs (e.g., as MLIR dialects across in-memory computing architectures)

Thanks! & Acknowledgements



Hasna Bouraoui | Alexander Brauckmann | Karl Friebel | Andrés Goens | Fazal Hameed | Gerald Hempel | Asif Ali Khan



Robert Khasanov | Nesrine Khouzami | Galina Kozyreva | Christian Menard | Julian Robledo | Lars Schütze | Felix Wittwer

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

Funded by

DFG Deutsche Forschungsgemeinschaft

CO4RTM (Number 450944241)
OpenPME (Number 350008342)

EVEREST https://everest-h2020.eu

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

STAATSMINISTERIUM FÜR WISSENSCHAFT KULTUR UND TOURISMUS

Freistaat SACHSEN

Summary

- Tame ever-increasing system complexity
  - Still highly-relevant optimizing compilers (polyhedral, …)
  - DSL examples: expose higher semantics (efficiency, productivity)
  - Higher semantics key for emerging accelerators/systems!

- Moving forward
  - Semantic-preserving transformations
  - Larger use cases (e.g., WRF in the context of EVEREST)
  - Common abstraction across novel paradimgs (e.g., as MLIR dialects across in-memory computing architectures)

References

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

[Manilov'18] S. Manilov, C. Vasiladiotis, B. Franke. "Generalized profile-guided iterator recognition." CC 2018.

[Chelini'21] L. Chelini, et al. "Progressive Raising in Multi-level IR." CGO 2021

[Gareev'18] R. Gareev, T. Grosser, M. Kruse. "High-performance generalized tensor operations: A compiler-oriented approach." ACM TACO 15.3 (2018): 34.

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

[GPCE'17] A. Susungi, et al., "Towards Compositional and Generative Tensor Optimizations", GPCE'17 pp. 169–175.

[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

[Incardona'19] P. Incardona, et al "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers", Computer Physics Communications, 2019

[ACM 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.

[CC'18] S. Ertel, A. Goens, J. Adam, J. Castrillon, "Compiling for Concise Code and Efficient I/O", Proceedings of the 27th International Conference on Compiler Construction (CC 2018), ACM, pp. 104–115

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

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

[TVLSI'19] F. Hameed, J. Castrillon, "A Novel Hybrid DRAM/STT-RAM Last-Level-Cache Architecture for Performance, Energy and Endurance Enhancement" , In IEEE TVLSI, 27, pp. 2375-2386, Oct 2019.

[IEEE Proc.'20] R. Bläsing, et al. "Magnetic Racetrack Memory: From Physics to the Cusp of Applications within a Decade". In: Proceedings of the IEEE 2020.

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

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

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