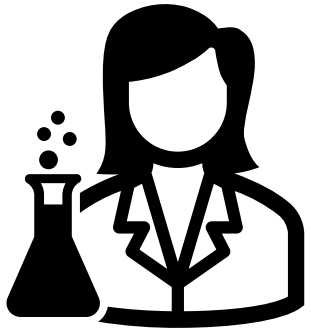
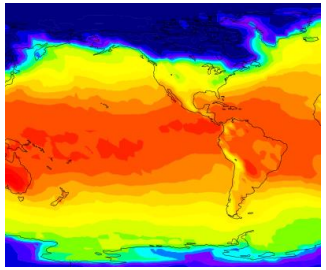
Tal Ben-Nun and the DaCe Team at SPCL

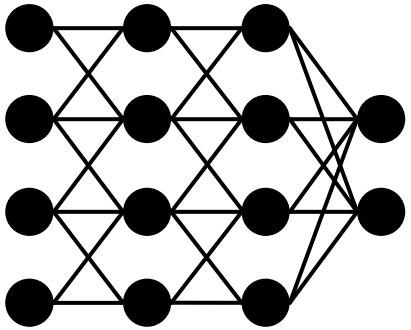
Scaling Machine Learning and Scientific Computing: A Data-Centric Perspective

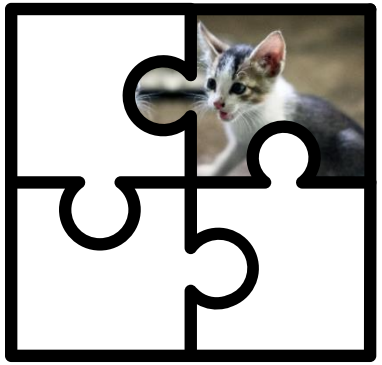
Efficient Simulations on GPU Hardware, October 2022

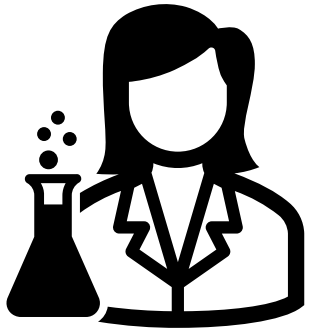
This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 program (grant agreements MAELSTROM, No. 955513 and DEEP-SEA, No. 955606).

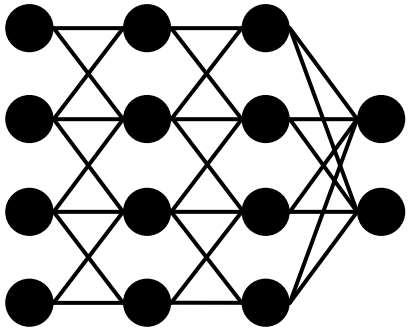


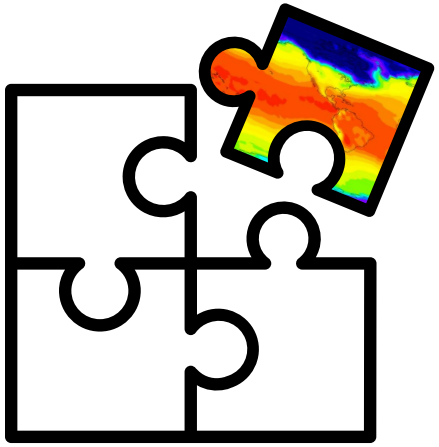


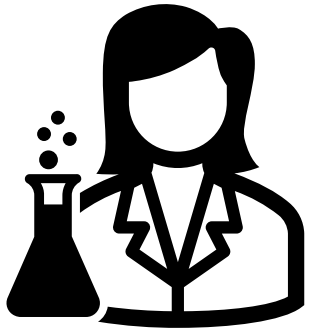


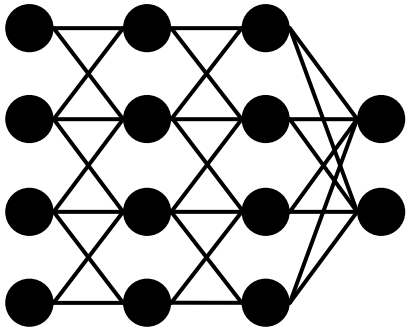


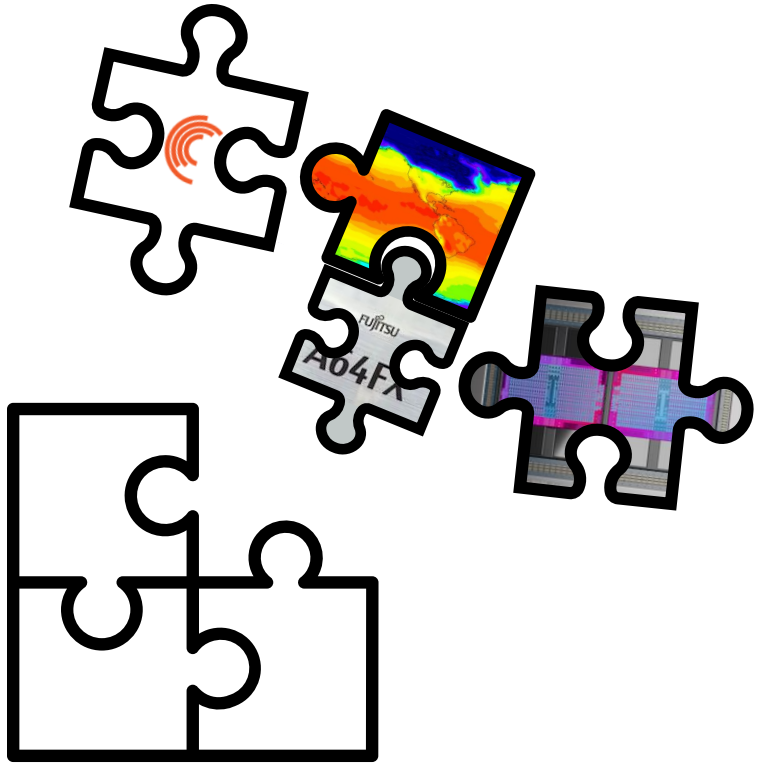


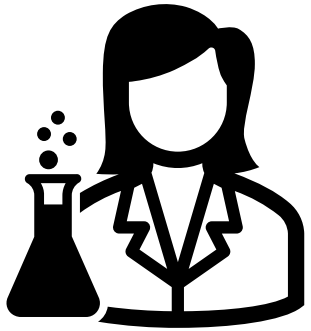


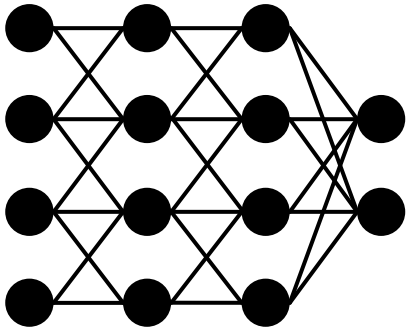


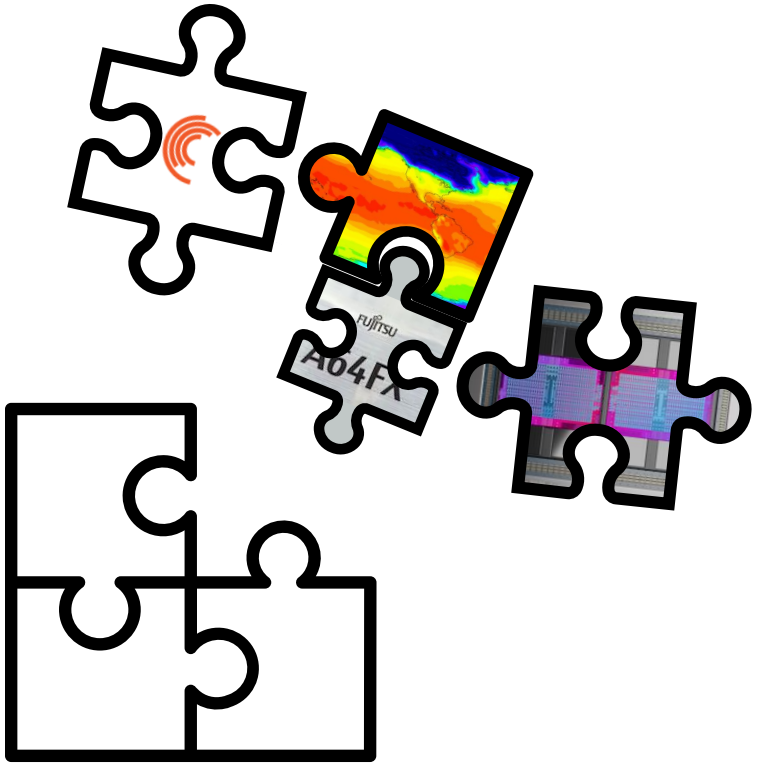


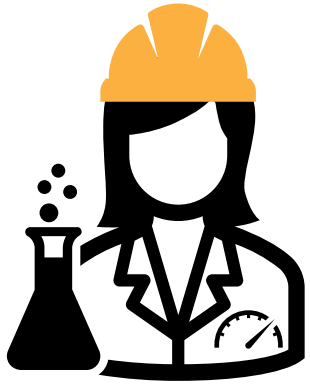


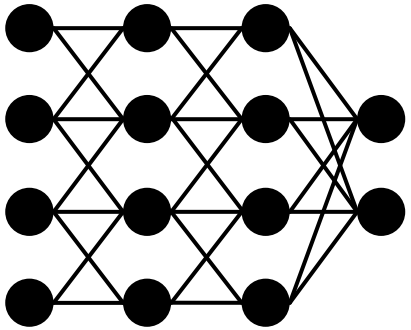

FUJITSU
A64FX

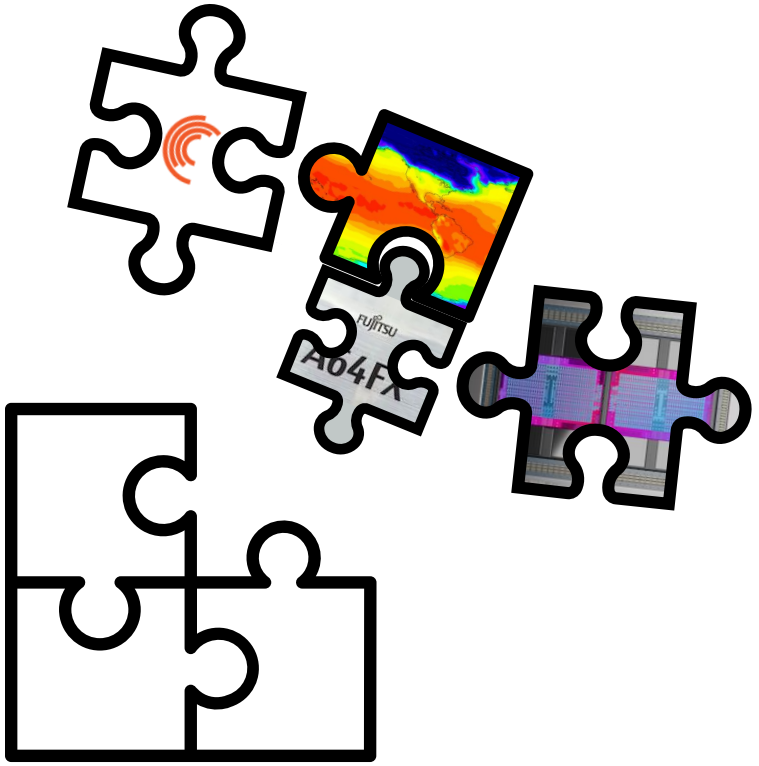


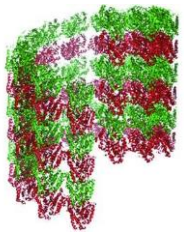


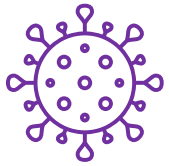

FUJITSU
A64FX



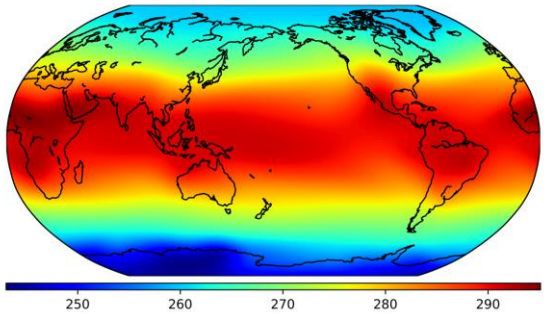



FUJITSU
A64FX





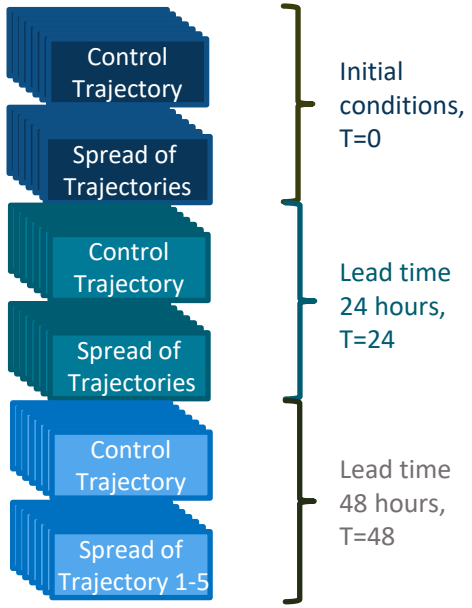
ENS-10 and Ensemble Post-Processing



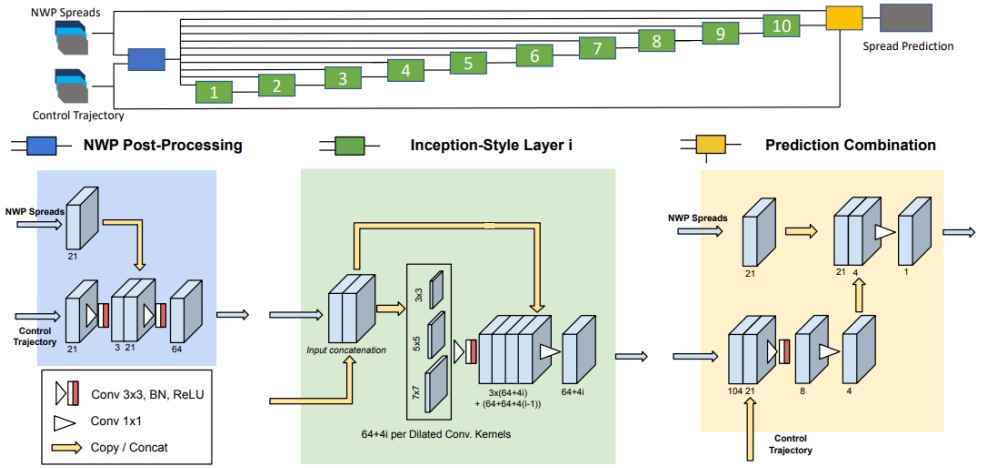
MAELSTROM

ENS-10: **~3 TB** re-forecast (hindcast) dataset for ensemble weather forecasts:

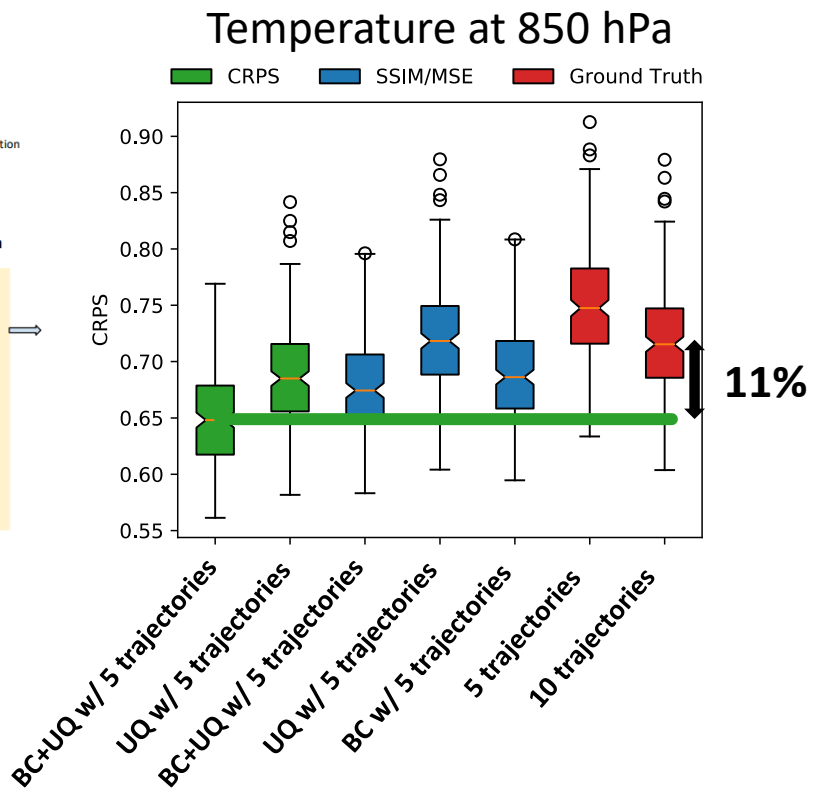
- 10-member ensemble + control
- 0.5° latitude / longitude resolution
- Years: 1999 – 2017



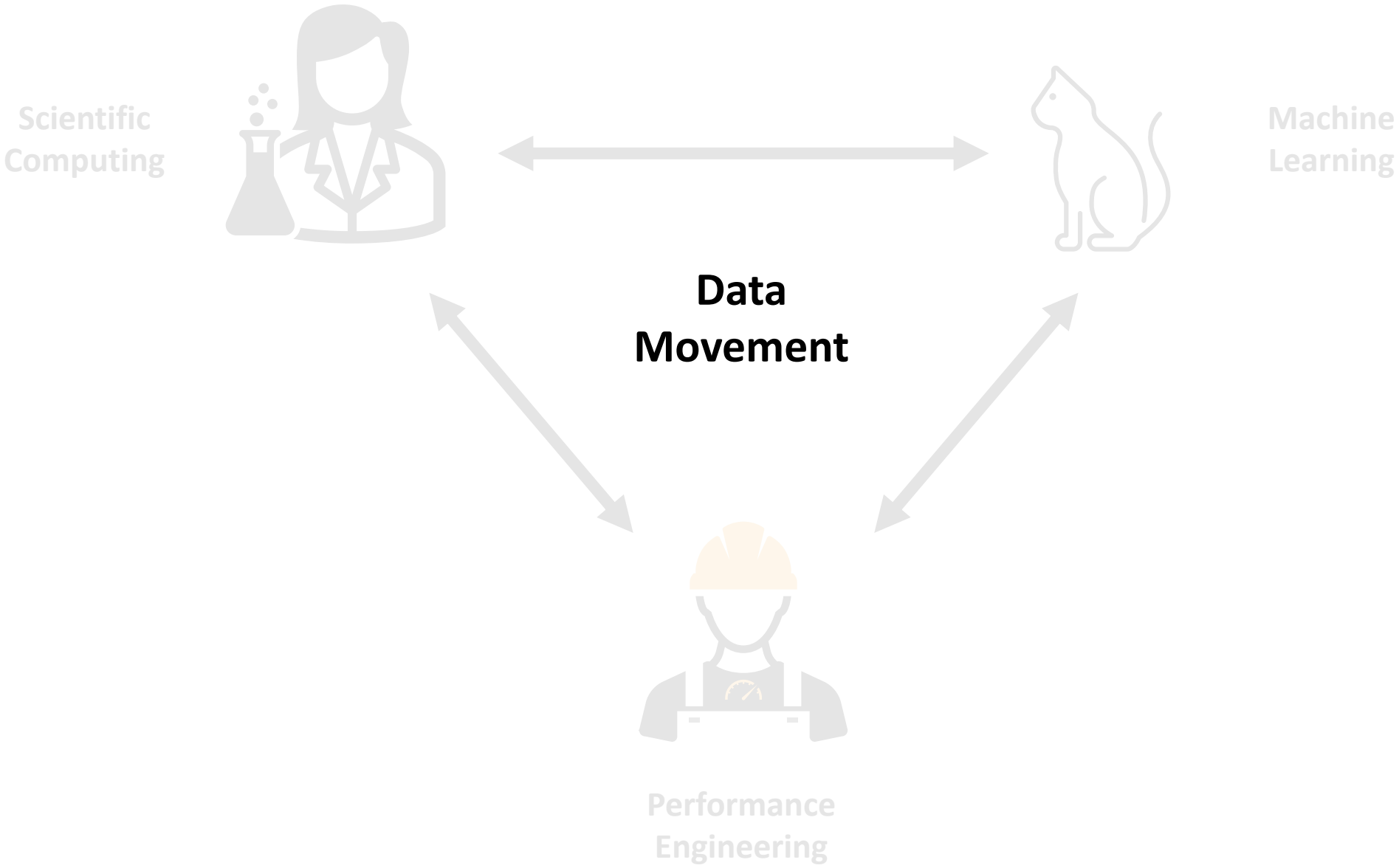
Sample dimensions:
10 x 11 x 7 x 361 x 720



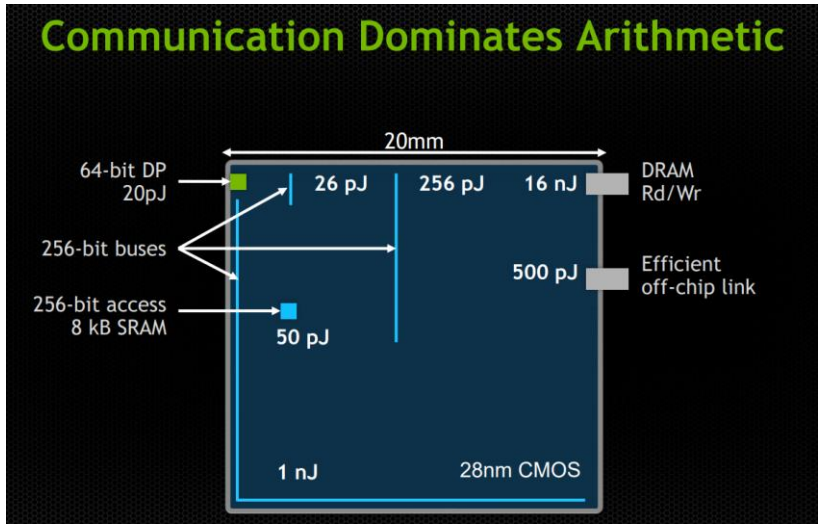
Training time: several days
Some models infeasible



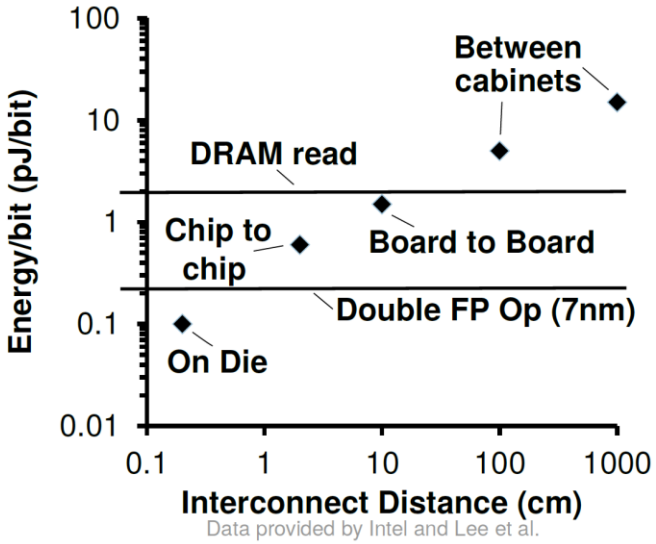
Grönquist, Yao, Ben-Nun et al., Deep learning for post-processing ensemble weather forecasts, RSTA'21.


Scientific Computing
Machine Learning
Data Movement
Performance Engineering

Data Movement is All You Need



Slide courtesy of NVIDIA

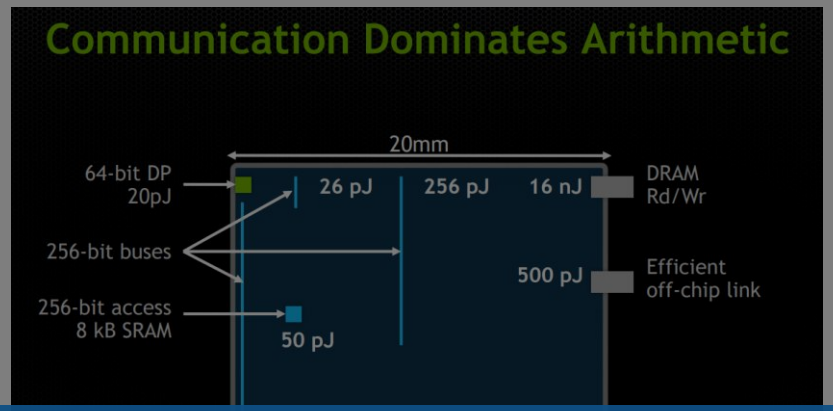


333
LoC

```
__syncthreads();

// Compute a grid of C matrix tiles in each warp.
#pragma unroll
for (int k_step = 0; k_step < CHUNK_K; k_step++) {
    wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a[WARP_COL_TILES];
    wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b[WARP_ROW_TILES];

    #pragma unroll
    for (int i = 0; i < WARP_COL_TILES; i++) {
        size_t shmem_idx_a = (warpId/2) * M * 2 + (i * M);
        const half *tile_ptr = &shmem[shmem_idx_a][k_step * K];

        wmma::load_matrix_sync(a[i], tile_ptr, K * CHUNK_K + SKEW_HALF);

        #pragma unroll
        for (int j = 0; j < WARP_ROW_TILES; j++) {
            if (i == 0) {
                // Load the B matrix fragment once, because it is going to be reused
                // against the other A matrix fragments.
                size_t shmem_idx_b = shmem_idx_b_off + (WARP_ROW_TILES * N) * (warpId%2)
                                   + (j * N);
                const half *tile_ptr = &shmem[shmem_idx_b][k_step * K];

                wmma::load_matrix_sync(b[j], tile_ptr, K * CHUNK_K + SKEW_HALF);
            }

            wmma::mma_sync(c[i][j], a[i], b[j], c[i][j]);
        }
    }
}

__syncthreads();
```

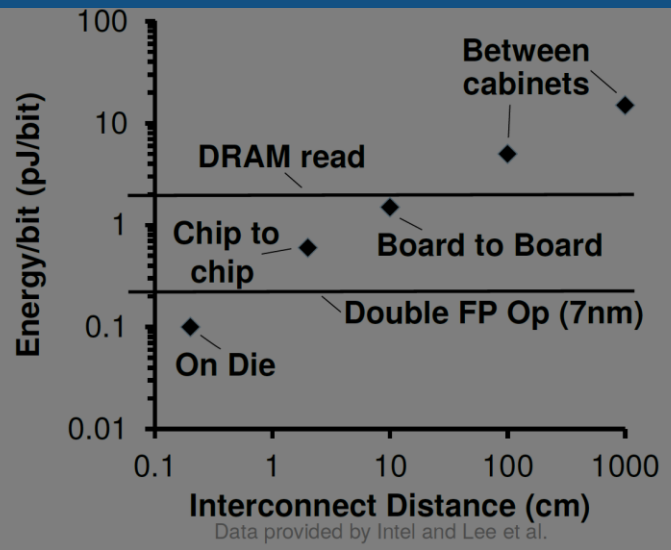
Tensor Core NVIDIA Code Sample

Data Movement is All You Need



Slide courtesy of NVIDIA

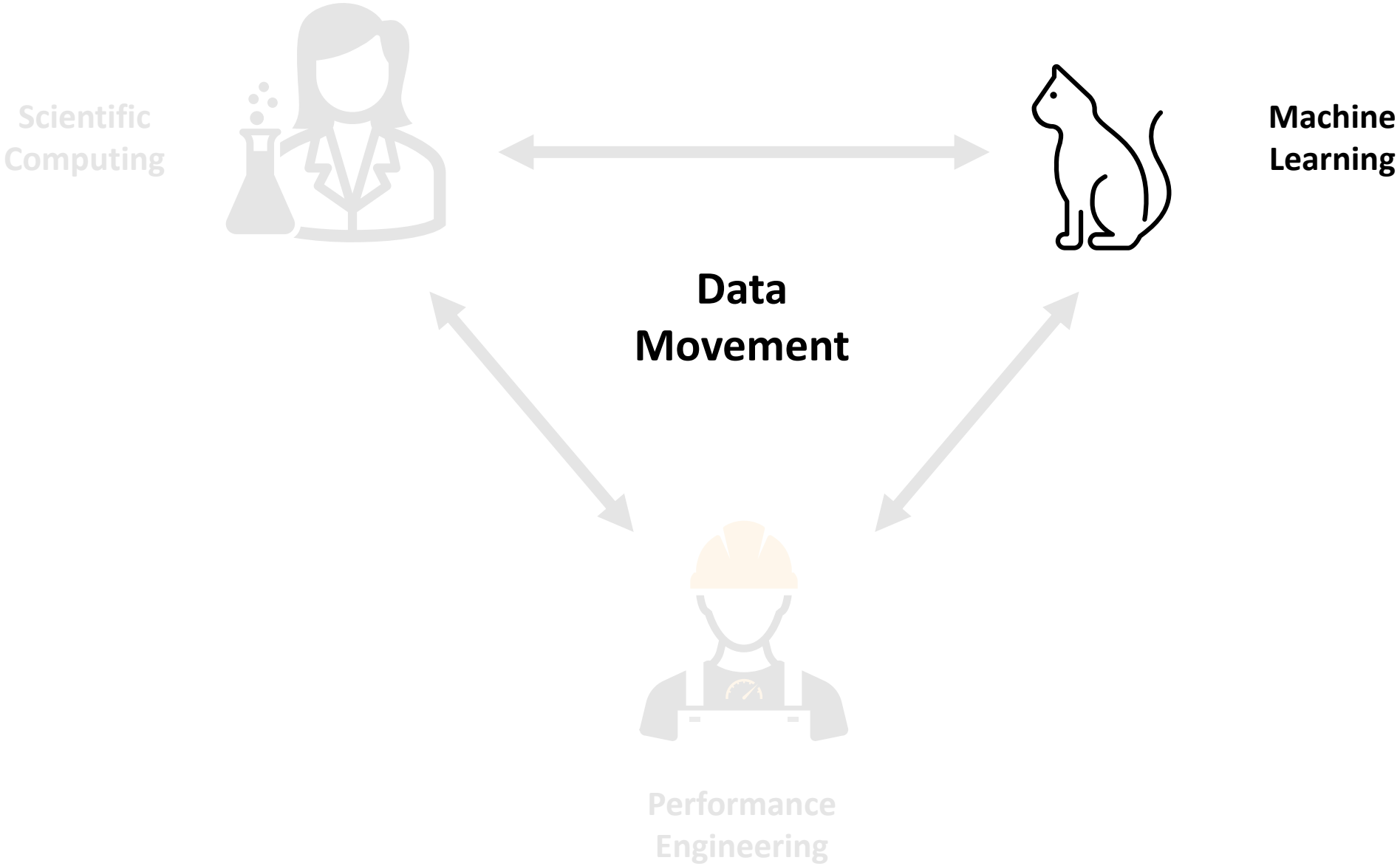
High-performance optimization = data movement reduction



Data provided by Intel and Lee et al.

```
__syncthreads();

// Compute a grid of C matrix tiles in each warp.
#pragma unroll
for (int k_step = 0; k_step < CHUNK_K; k_step++) {
    wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a[WARP_COL_TILES];
    wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b[WARP_ROW_TILES];

    #pragma unroll
    for (int i = 0; i < WARP_COL_TILES; i++) {
        size_t shmem_idx_a = (warpId/2) * M * 2 + (i * M);
        const half *tile_ptr = &shmem[shmem_idx_a][k_step * K];

        #pragma unroll
        for (int j = 0; j < WARP_ROW_TILES; j++) {
            if (i == 0) {
                // Load the B matrix fragment once, because it is going to be reused
                // against the other A matrix fragments.
                size_t shmem_idx_b = shmem_idx_b_off + (WARP_ROW_TILES * N) * (warpId%2)
                                     + (j * N);
                const half *tile_ptr = &shmem[shmem_idx_b][k_step * K];

                wmma::load_matrix_sync(b[j], tile_ptr, K * CHUNK_K + SKEW_HALF);
            }

            wmma::mma_sync(c[i][j], a[i], b[j], c[i][j]);
        }
    }
}

__syncthreads();
```

Tensor Core NVIDIA Code Sample


Scientific Computing
Machine Learning
Data Movement
Performance Engineering

State of the Practice

- **DNN compilers are on the rise**

Operator-centric view:

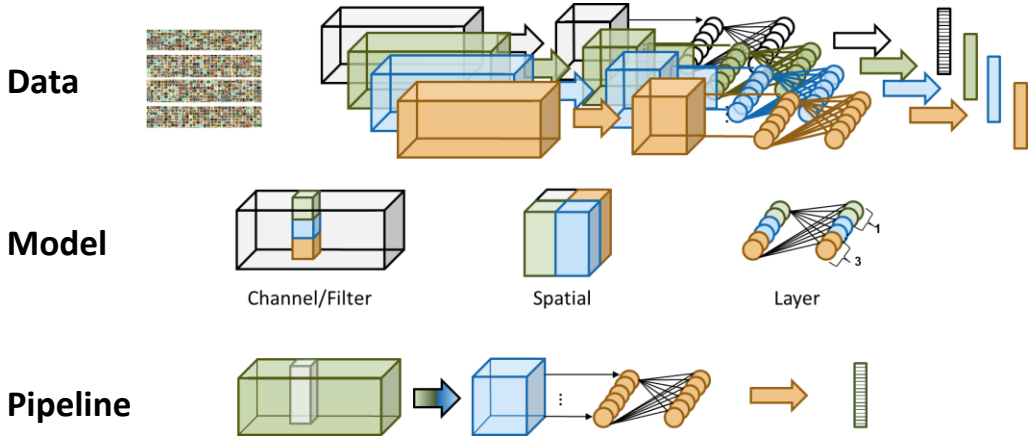
- **Most are affected by "library jail"**
  - Library calls (Convolution, GEMM) never decomposable
- **Convolutions with higher FLOP count underperform**
- **Even within the "standard" realm, odd performance**
- **Hardcoded sharding schemes for distribution**

Workload	Size	%Peak FLOP/s
Conv1d (3->16, 3)	B=128, W=128	8.36
Conv2d (3->16, 3x3)	B=128, W=H=128	75.45
Conv3d (3->16, 3x3x3)	B=128, D=W=H=128	46.26

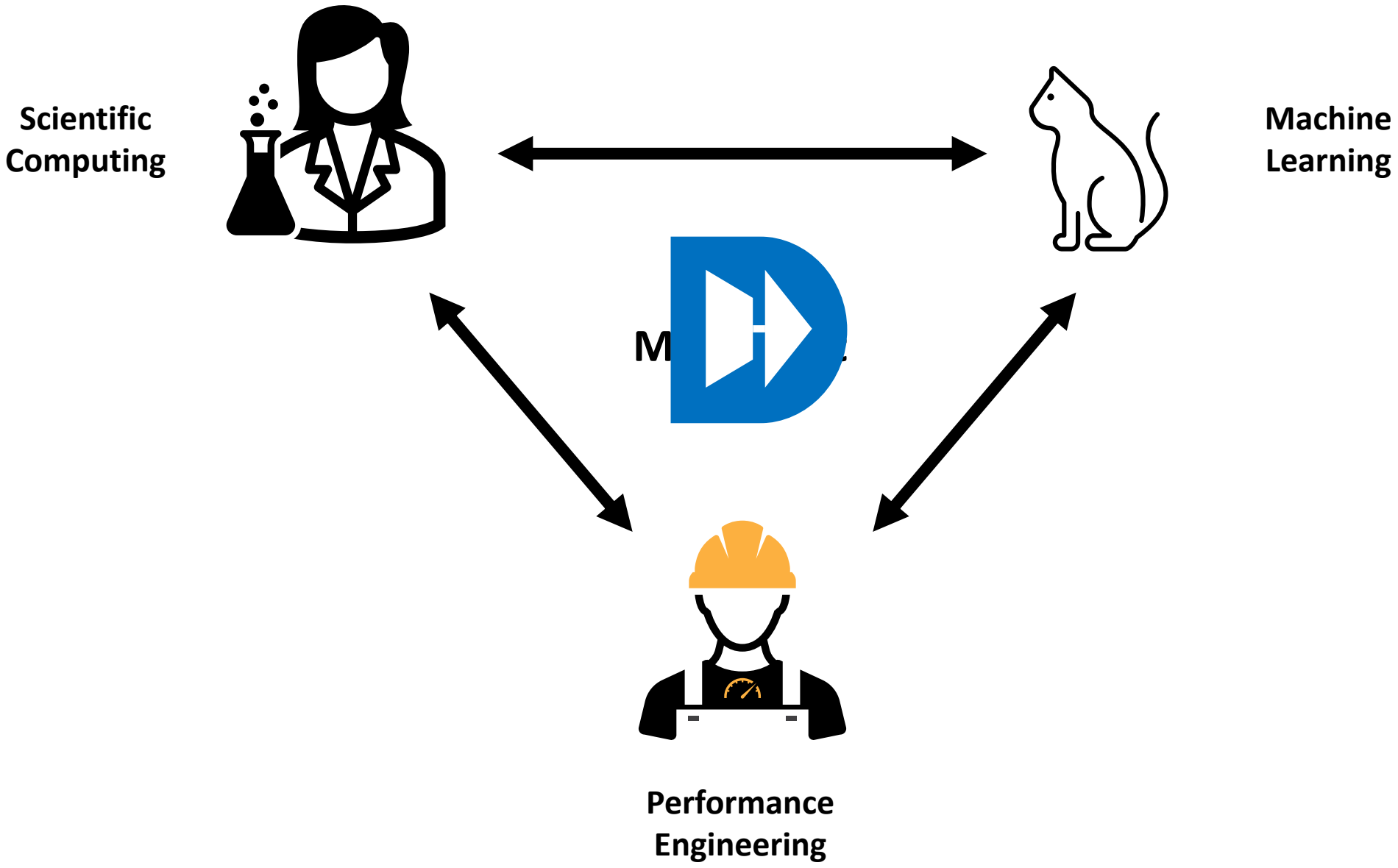
CUDA 11.4, CUDNN 8.0.5, PyTorch 1.8

Workload	PyTorch	`torch.jit`	JAX	TF+XLA
ResNet-50	32.04	31.94	33.93	35.57
Wide ResNet-50-2	70.94	70.83	98.13	99.06
Ratio (ideal: 2x)	**2.21x**	**2.22x**	**2.89x**	**2.78x**

Forward+backprop time, in ms



Ben-Nun & Hoefler, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, CSUR'19.
Dryden et al., Channel and filter parallelism for large-scale CNN training, SC'19.
Rausch*, Ben-Nun* et al., A Data-Centric Optimization Framework for Machine Learning, ICS'22.


Scientific Computing
Machine Learning
Performance Engineering

DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

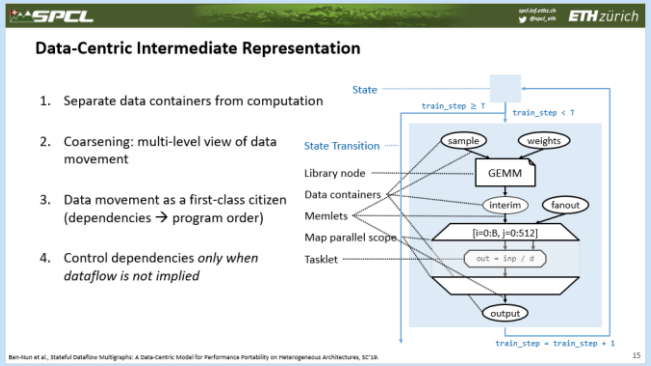
Python | DSLs

PyTorch | C

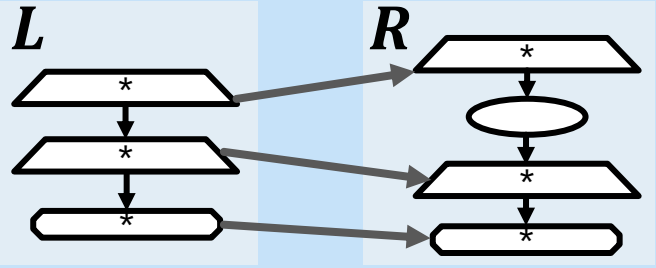
...

Scientific Frontend

Performance Engineer



Data-Centric Intermediate Representation (SDFG)



Graph Transformations

Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

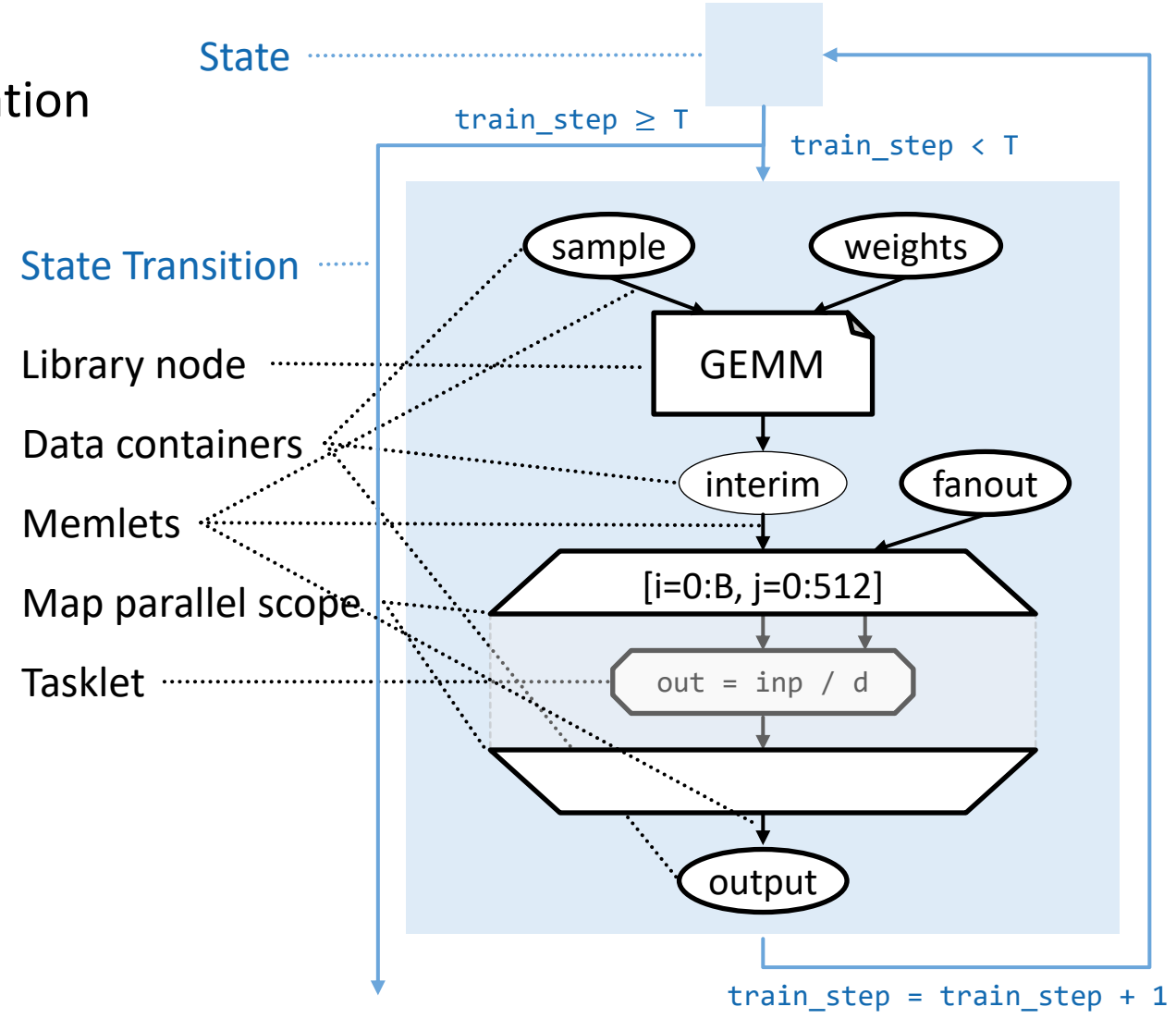
CPU Binary

GPU Binary

FPGA Modules

Data-Centric Intermediate Representation

1. Separate data containers from computation
2. Coarsening: multi-level view of data movement
3. Data movement as a first-class citizen (dependencies → program order)
4. Control dependencies *only when dataflow is not implied*



Ben-Nun et al., Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures, SC'19.

DaCe Overview

Domain Scientist

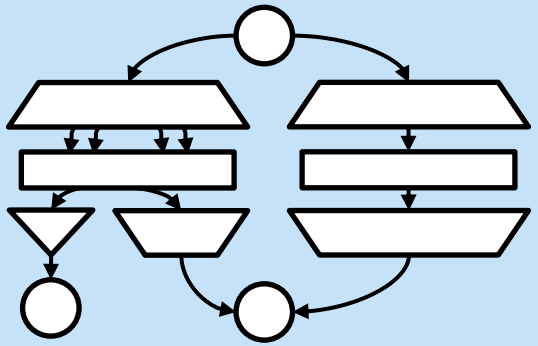
Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

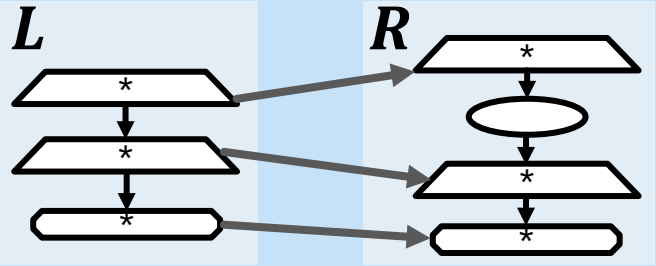
Python | DSLs
PyTorch | C
...

Scientific Frontend

Performance Engineer



Data-Centric Intermediate Representation (SDFG)



Graph Transformations

Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

CPU Binary

GPU Binary

FPGA Modules

DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

Python	DSLs
PyTorch	C

...

Scientific Frontend

```
glob_b = ...
class ClassA:
    def __init__(self, arr):
        self.q = arr

    @dace.method
    def __call__(self, a):
        return a * self.q + glob_b
```

Python JIT

```
@dace_module
class MyModule(nn.Module):
  def __init__(self, in_features,
               out_features):
    super().__init__()
    self.linear = nn.Linear(in_features,
                            out_features)
    self.fanout = out_features

  def forward(self, x):
    return self.linear(x) / self.fanout
```

PyTorch (DaCeML)

```
for (int i = 0; i < N ; i++)
    for (int j = row_ptr[i]; j < row_ptr[i+1]; j++)
        y[i] += A[col_idx[j]] * x[j];
```

C

```
N = dace.symbol()

@dace.program
def jacobi_1d(tsteps: dace.int32,
              a: dace.float64[N],
              b: dace.float64[N]):

    for _ in range (1, tsteps):
        b[1: -1] = 0.33333 * (a[:-2] + a[1:-1] + a[2:])
        a[1: -1] = 0.33333 * (b[:-2] + b[1:-1] + b[2:])
```

Python AOT

```
sdfg.sdfg{entry=@state_0} @sdfg_0 {
    %A = sdfg.alloc() : !sdfg.array<2x6x8xi32>
    %B = sdfg.alloc() : !sdfg.array<2x6x8xi32>
    %C = sdfg.alloc() : !sdfg.array<2x6x8xi32>
    sdfg.state @state_0 {
        sdfg.map (%i, %j, %g) = (0, 0, 0) to (2, 2, 2) step (1, 1, 1) {
            %a_ijg = sdfg.load %A[%i, %j, %g] : !sdfg.array<2x6x8xi32> -> i32
            %b_ijg = sdfg.load %B[%i, %j, %g] : !sdfg.array<2x6x8xi32> -> i32
            %res = sdfg.tasklet @add(%a_ijg: i32, %b_ijg: i32) -> i32 {
                %z = arith.addi %a_ijg, %b_ijg : i32
                sdfg.return %z : i32
            }
            sdfg.store %res, %C[%i, %j, %g] : i32 -> !sdfg.array<2x6x8xi32>
        }
    }
}
```

MLIR dialect

```
do i=is,ie+1
  if (cx(i,j,k) > 0.) then
    xfx(i,j,k) = cx(i,j,k)*dxa(i-1,j)*dy(i,j)*sin_sg(i-1,j,3)
  else
    xfx(i,j,k) = cx(i,j,k)*dxa(i,  j)*dy(i,j)*sin_sg(i,  j,1)
  endif
enddo
```

FORTRAN (coming soon...)

From source code to SDFG

Code

```
glob_b = ...
class ClassA:
    def __init__(self, arr):
        self.q = arr

    @dace.method
    def __call__(self, a):
        return a * self.q + glob_b
```

From source code to SDFG

Code → **Preprocess**

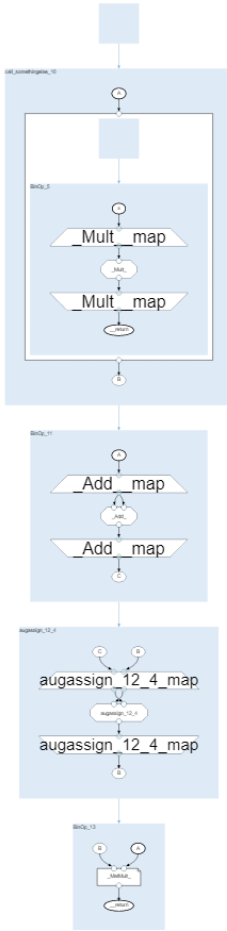
```
glob_b = ...
class ClassA:
    def __init__(self, arr):
        self.q = arr

    @dace.method
    def __call__(self, a):
        return a * self.q + glob_b
```

```
@dace.program
def ClassA__call__(a, __g_self_q, __g_glob_b):
    return a * __g_self_q + __g_glob_b
```

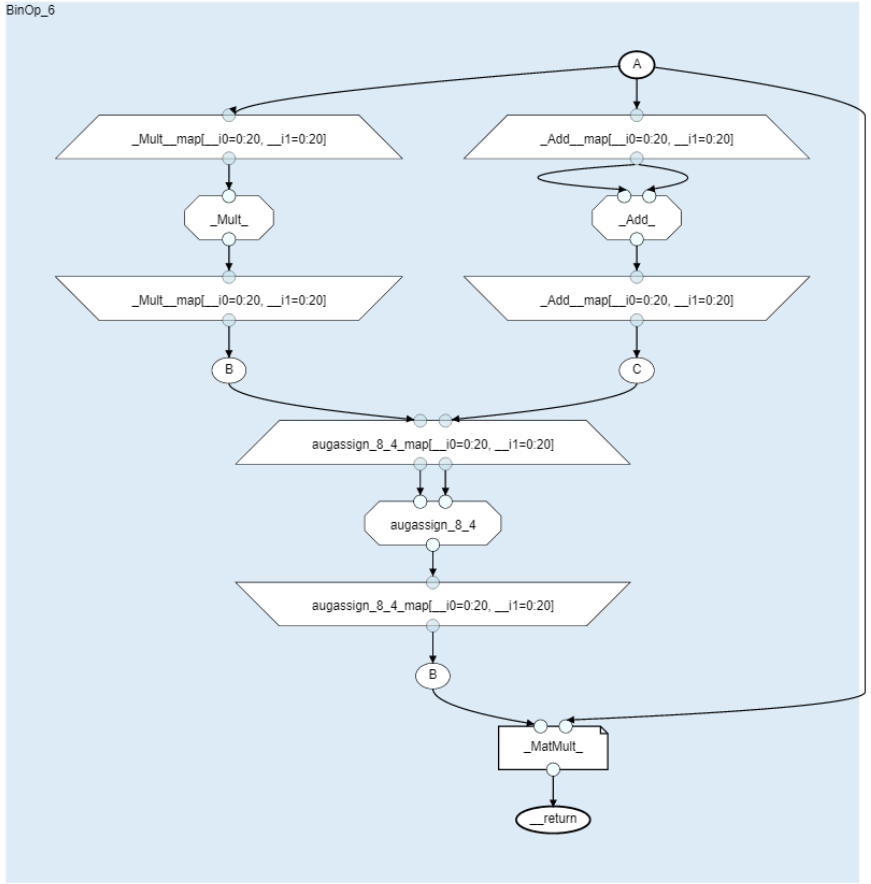
Closure Resolution → Constant Folding → Conditional/Dead Code Elimination → Call Tree Evaluation → Array Consolidation

From source code to SDFG

Code → **Preprocess** → **Simplify**

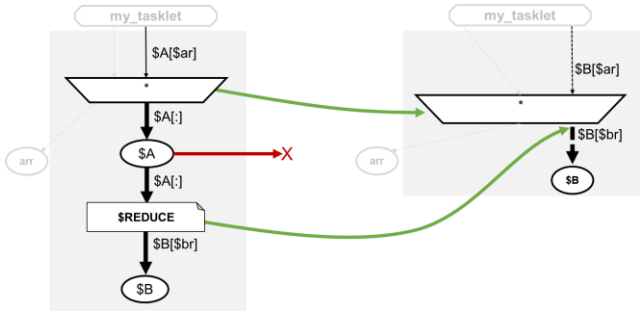


Control flow coarsening passes

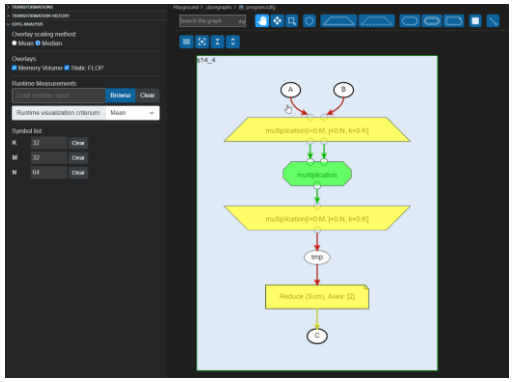


From source code to SDFG

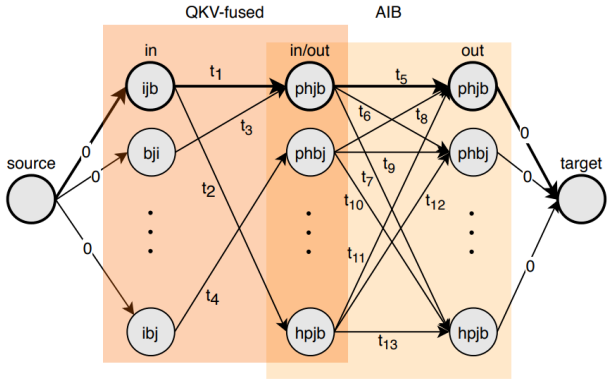
Code	Preprocess	Simplify	Optimize



Graph Rewriting Transformations



Interactive Transformation and Instrumentation

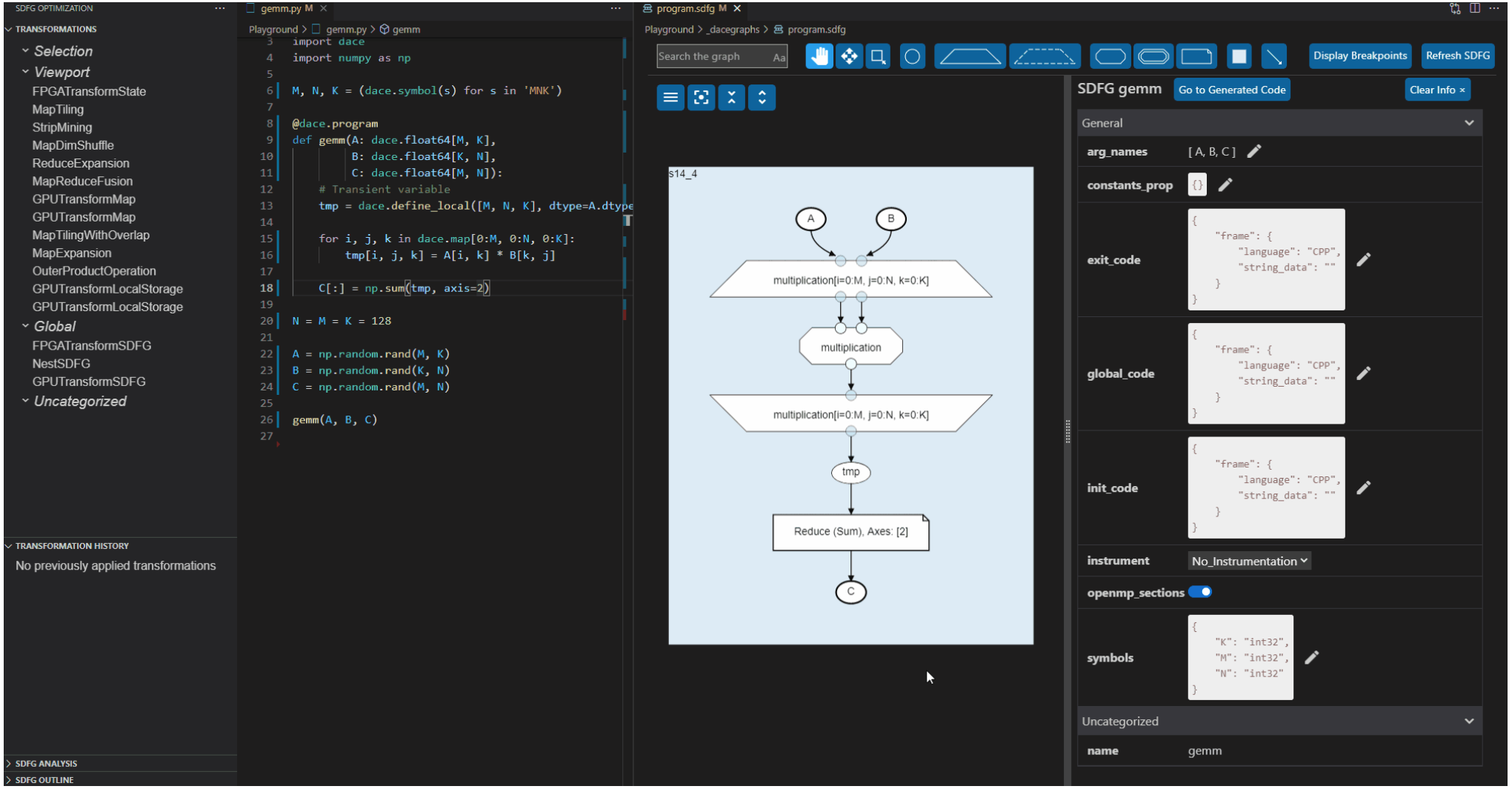


Local and Global Tuning Interface

```
class ClassA:
    def __init__(self, arr):
        self.q = arr

    @dace.method(auto_optimize=True, device=dace.DeviceType.GPU)
    def __call__(self, a):
        return a * self.q + glob_b
```

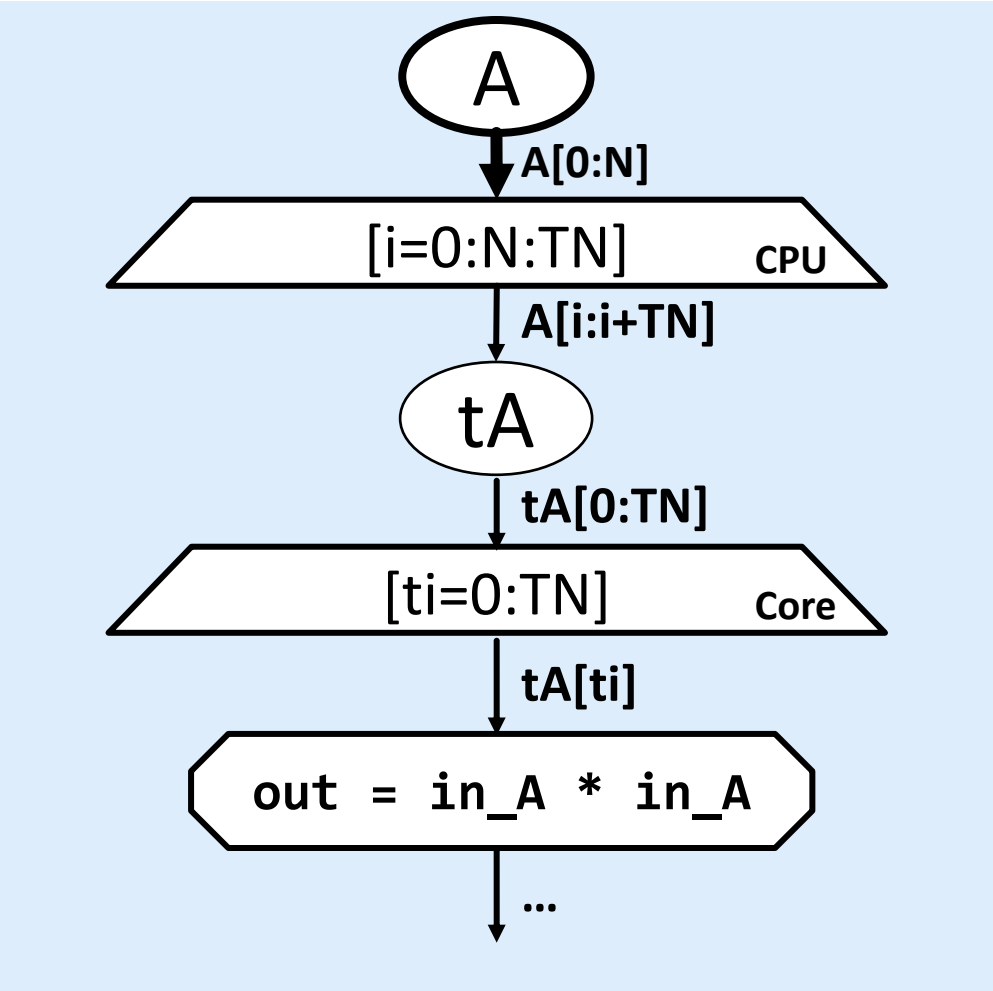
Automatic Optimization Heuristics

Visual Studio Code Integration



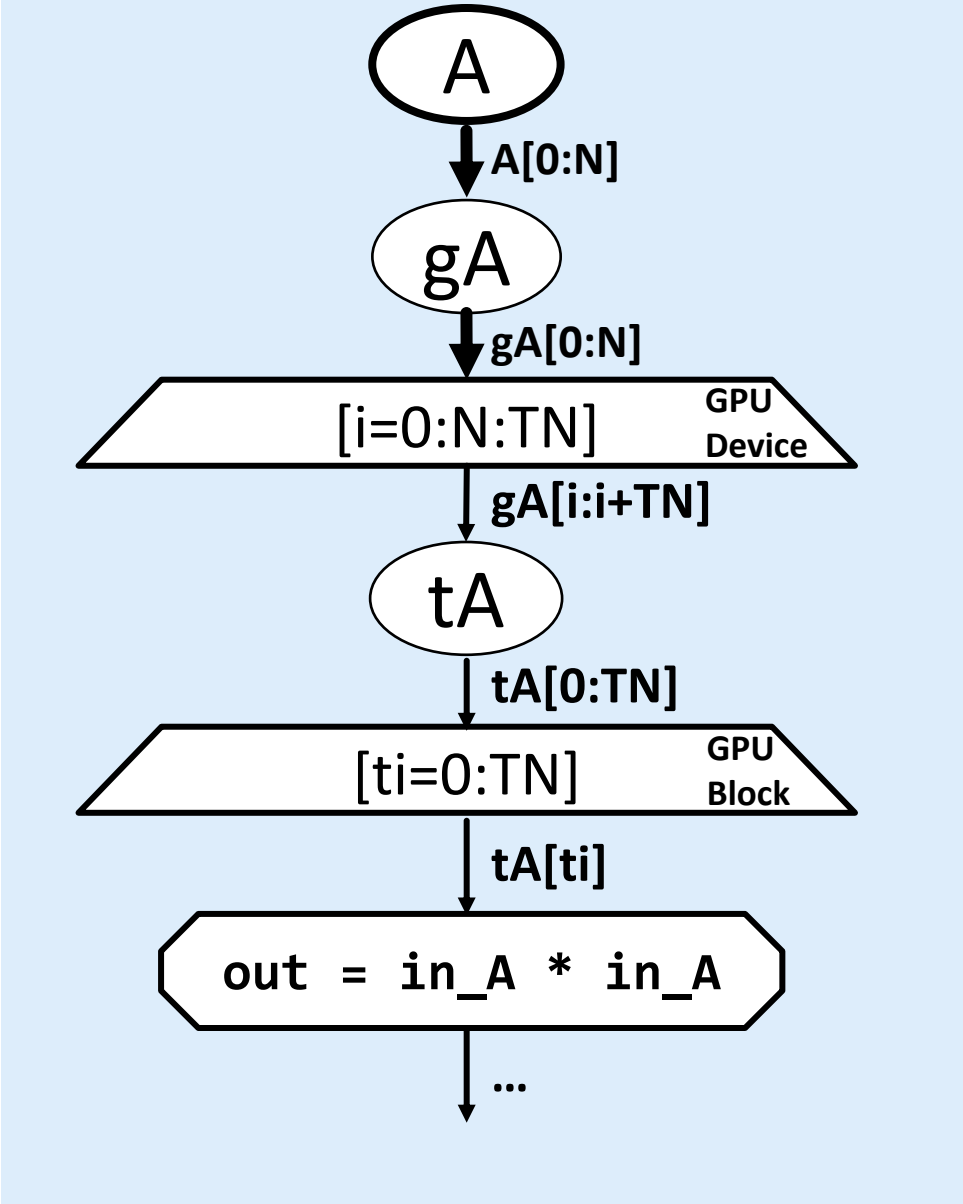
Hierarchical Parallelism and Heterogeneity

- **Maps have schedules, arrays have storage locations**



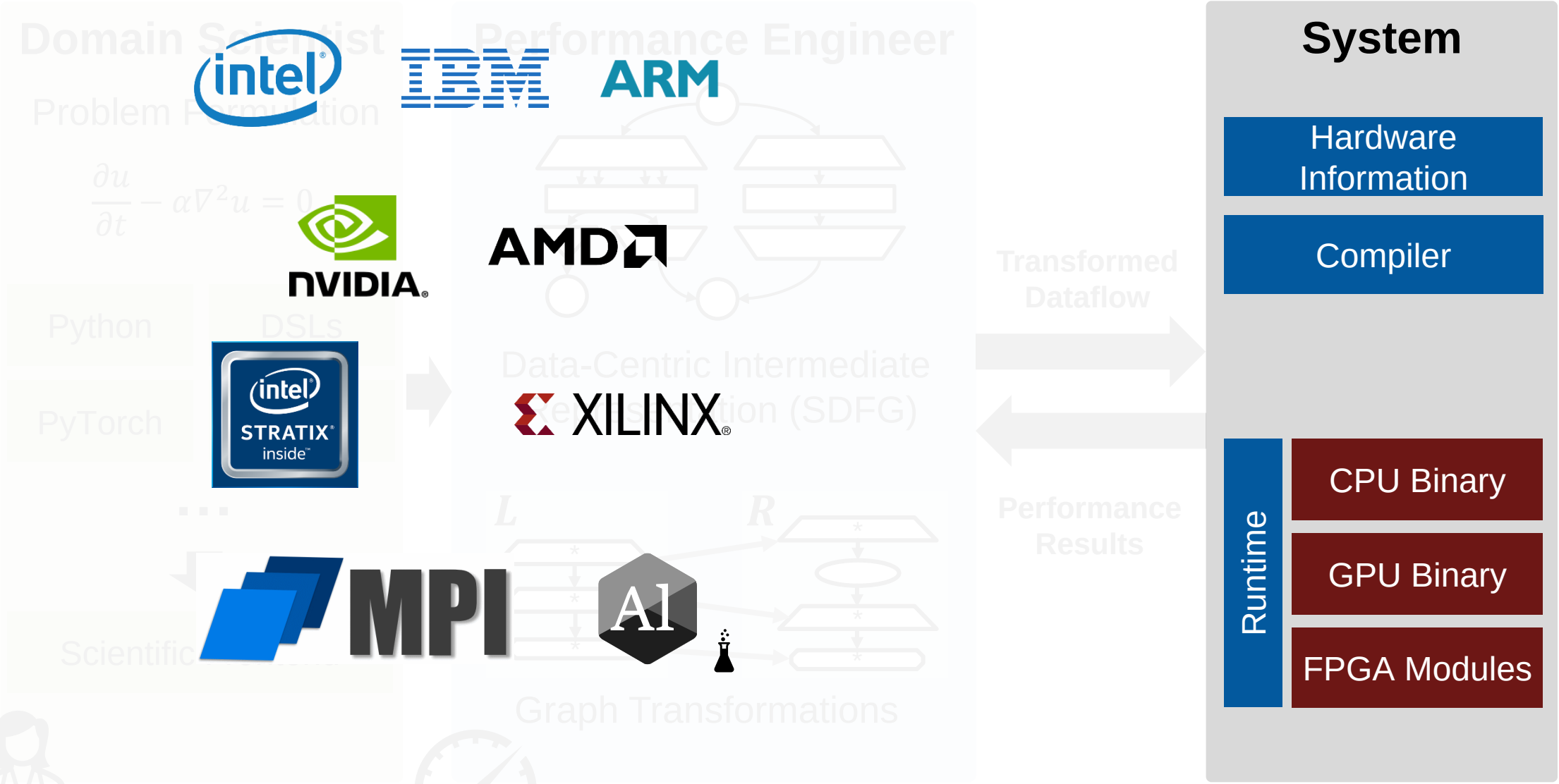

```
// ...
#pragma omp parallel for
for (int i = 0; i < N; i += TN) {
    vec<double, 4> tA[TN];
    Global2Stack_1D<double, 4, 1>(&A[i], min(N – i, TN), tA);

    for (int ti = 0; ti < TN; ti += 1) {
        vec<double, 4> in_A = tA[ti];
        auto out = (in_A * in_A);
        tC[ti] = out;
    }
}
```

Hierarchical Parallelism and Heterogeneity




```
__global__ void multiplication_1(...) {
    int i = blockIdx.x * TN;
    int ti = threadIdx.y + 0;
    if (i+ti >= N)  return;

    __shared__ vec<double, 2> tA[TN];
    GlobalToShared1D<double, 2, TN, 1, 1, false>(gA, tA);

    vec<double, 2> in_A = tA[ti];
    auto out = (in_A * in_A);
    tC[ti] = out;
}
```

DaCe Overview

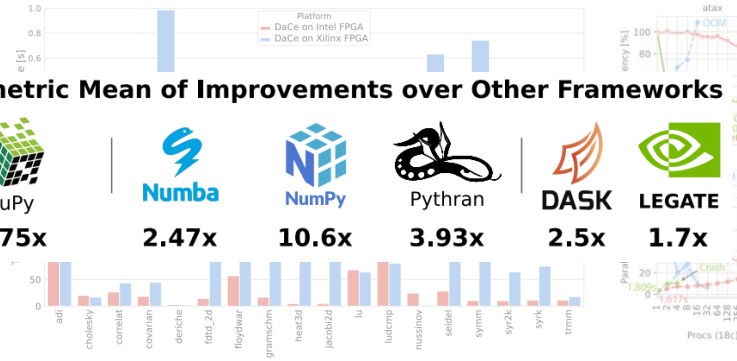


DaCe Performance

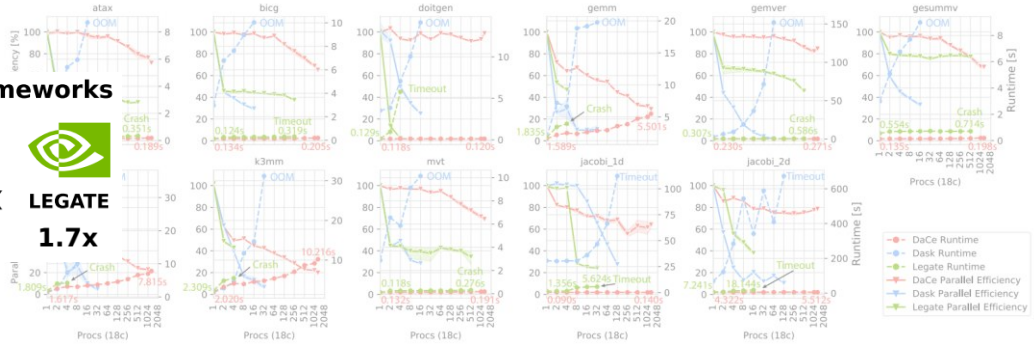
https://github.com/spcl/npbench

Domains	DaCe	Numba	Pythran	NumPy
Total	↑10.4	↑4.3	↑3.2	
Chemistry	↑36.3	↑6.2	↑4.1	0.71 s
Graphs	↑61.7	↑38.6	↑33.5	6.79
Learning	↑9.2	↑2.6	↓1.3	0.62
LinAlg	↑5.0	↑3.2	↑1.8	0.78
Other	↑17.5	↑19.4	↑28.6	2.11
Physics	↑16.0	↑4.1	↑2.6	5.19
Signals	↑8.1	↑3.6	↑1.9	0.92
Solver	↑5.7	↑3.3	↑6.3	1.01 s
Weather	↑33.8	↑4.3	↑1.2	1.14 s

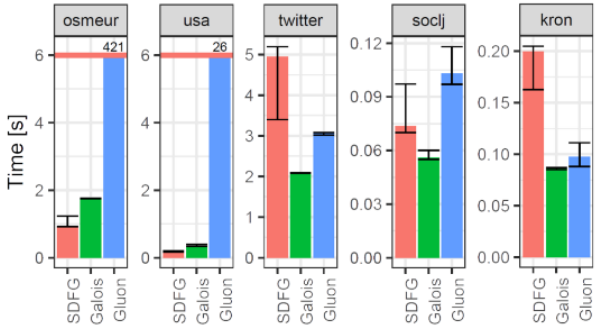
NumPyBench



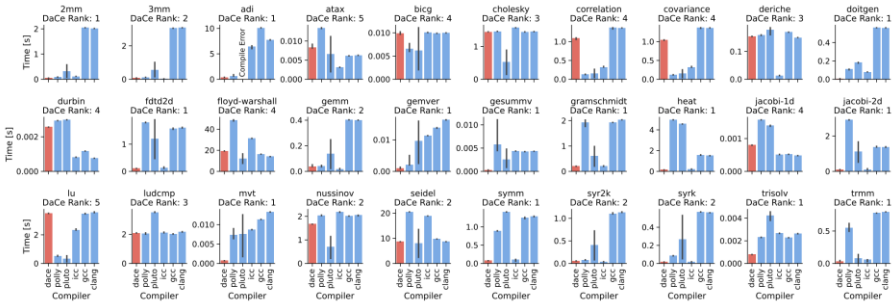
NumPyBench FPGA



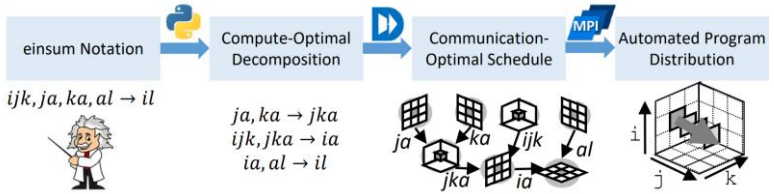
NumPyBench Distributed



Graph Analytics



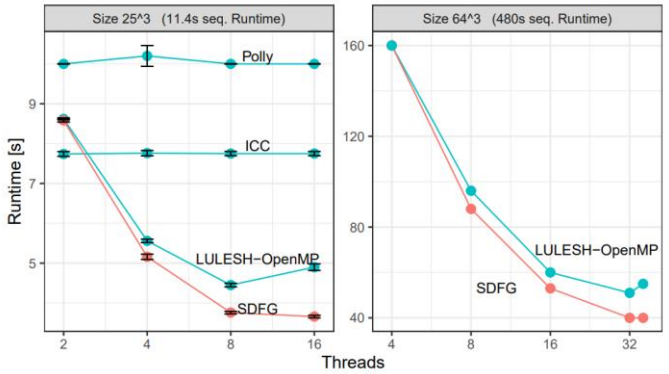
PolyBench/C



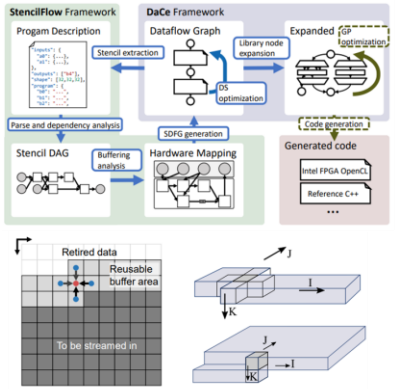
Distributed Tensor Operations

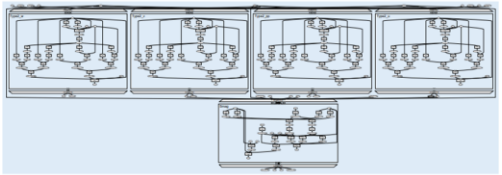
Ziogas, Schneider, Ben-Nun et al., Productivity, Portability, Performance: Data-Centric Python, SC'21.
Calotoiu, Ben-Nun et al., Lifting C semantics for dataflow optimization, ICS'22.
Ziogas, Kwasniewski, Ben-Nun et al., Deinsum: Practically I/O Optimal Multilinear Algebra, SC'22.

DaCe Performance



Unstructured Hydrodynamics (LULESH)

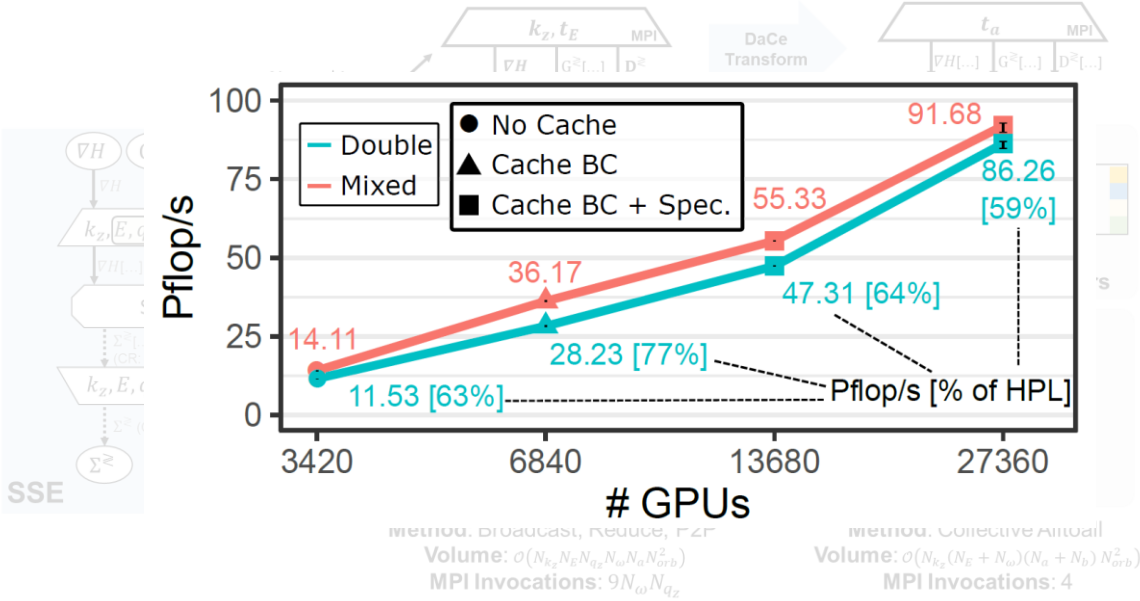




	Runtime	Performance	Peak BW.	%Roof.
Stratix 10	1,178 µs	145 GOp/s	77 GB/s	52%
Stratix 10*	332 µs	513 GOp/s	∞ GB/s	–
Xeon 12C	5,270 µs	32 GOp/s	68 GB/s	13%
P100	810 µs	210 GOp/s	732 GB/s	8%
V100	201 µs	849 GOp/s	900 GB/s	26%

*Without memory bandwidth constraints.

Numerical Weather Prediction (CPU, GPU, spatial)



Quantum Transport Simulation

Calotoiu, Ben-Nun et al. Lifting C semantics for dataflow optimization, ICS'22.
de Fine Licht, Kuster, De Matteis, Ben-Nun et al., StencilFlow: Mapping large stencil programs to distributed spatial computing systems, CGO'21.
Ziogas, Ben-Nun et al., A data-centric approach to extreme-scale ab initio dissipative quantum transport simulations, SC'19 (Gordon Bell Prize).

DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

Python | DSLs

PyTorch | C

...

Scientific Frontend

Performance Engineer

Data-Centric Intermediate Representation (SDFG)

Graph Transformations

Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

CPU Binary

GPU Binary

FPGA Modules

DaCeML

PyTorch

Rethink training as a data-centric program

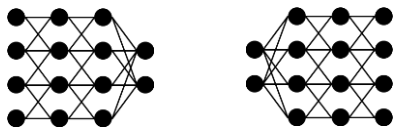
```
@dace_module
class MyModule(nn.Module):
  def __init__(self, in_features,
               out_features):
    super().__init__()
    self.linear = nn.Linear(in_features,
                            out_features)
    self.fanout = out_features

  def forward(self, x):
    return self.linear(x) / self.fanout
```

Goals:

- **No dependency on operators**
  - Rewritten in NumPy

```
@python_pure_op_implementation
def Softplus(X, Y):
  Y[:] = numpy.log(1 + numpy.exp(X))
```

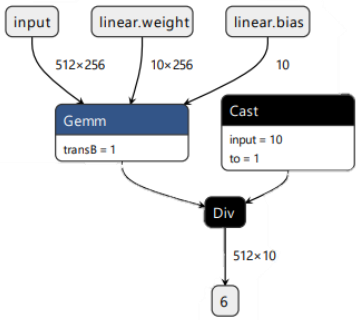
- **Redesign Automatic Differentiation (AD)**
  - Symbolically
- **Automate pipeline**
  - Before and after AD
  - Guided still an option



Rausch*, Ben-Nun* et al., A Data-Centric Optimization Framework for Machine Learning, ICS'22.

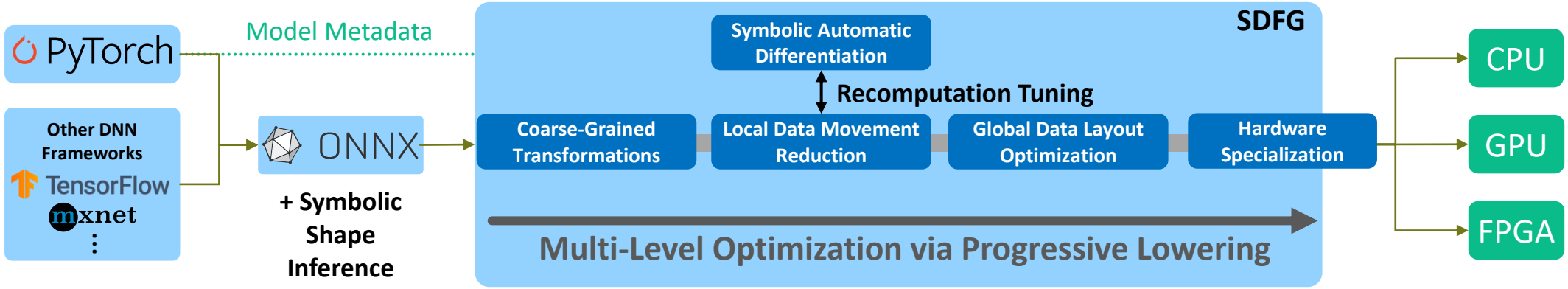
Optimization Pipeline

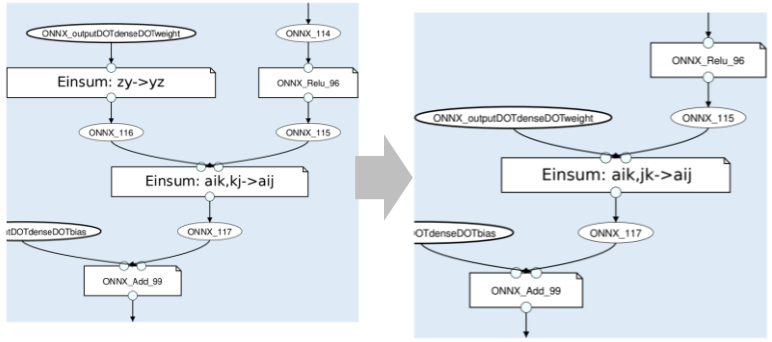
```
@dace_module
class MyModule(nn.Module):
  def __init__(self, in_features,
               out_features):
    super().__init__()
    self.linear = nn.Linear(in_features,
                            out_features)
    self.fanout = out_features

  def forward(self, x):
    return self.linear(x) / self.fanout
```



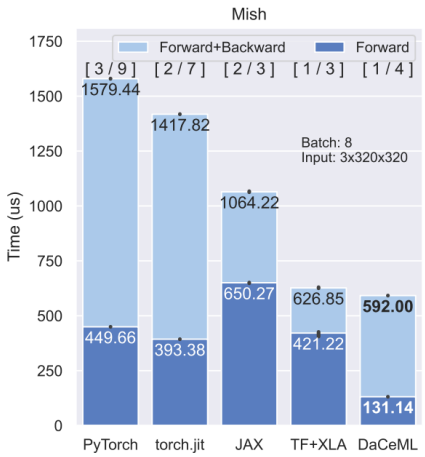
Rausch*, Ben-Nun* et al., A Data-Centric Optimization Framework for Machine Learning, ICS'22.

Optimization Pipeline

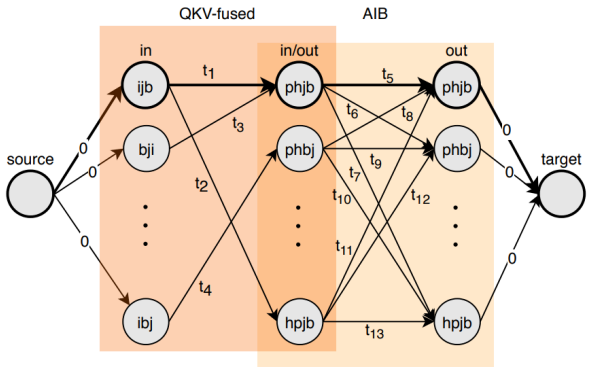




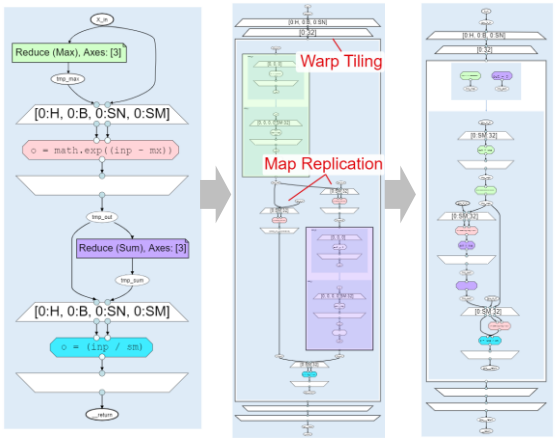
Algebraic Fusion



Pre/Post-AD Optimization



Optimal Data Layout Path



Statistical Normalization via Warp-Level Intrinsics

Rausch*, Ben-Nun* et al., A Data-Centric Optimization Framework for Machine Learning, ICS'22.
Ivanov, Dryden, Ben-Nun et al., Data Movement Is All You Need: A Case Study on Optimizing Transformers, MLSys '21.

DaCeML Results – Networks vs. PyTorch, `torch.jit`, JAX, TF+XLA; with a Tesla V100 GPU

	ResNet-50	Wide ResNet-50-2	MobileNet v2	WaveNet	DLRM
State of the Art:	31.94 ms	70.83 ms	15.53 ms	41.49 ms	126.55 ms
DaCeML:	32.45 ms	2.09x → **67.99 ms**	**14.77 ms**	**41.07 ms**	**126.42 ms**

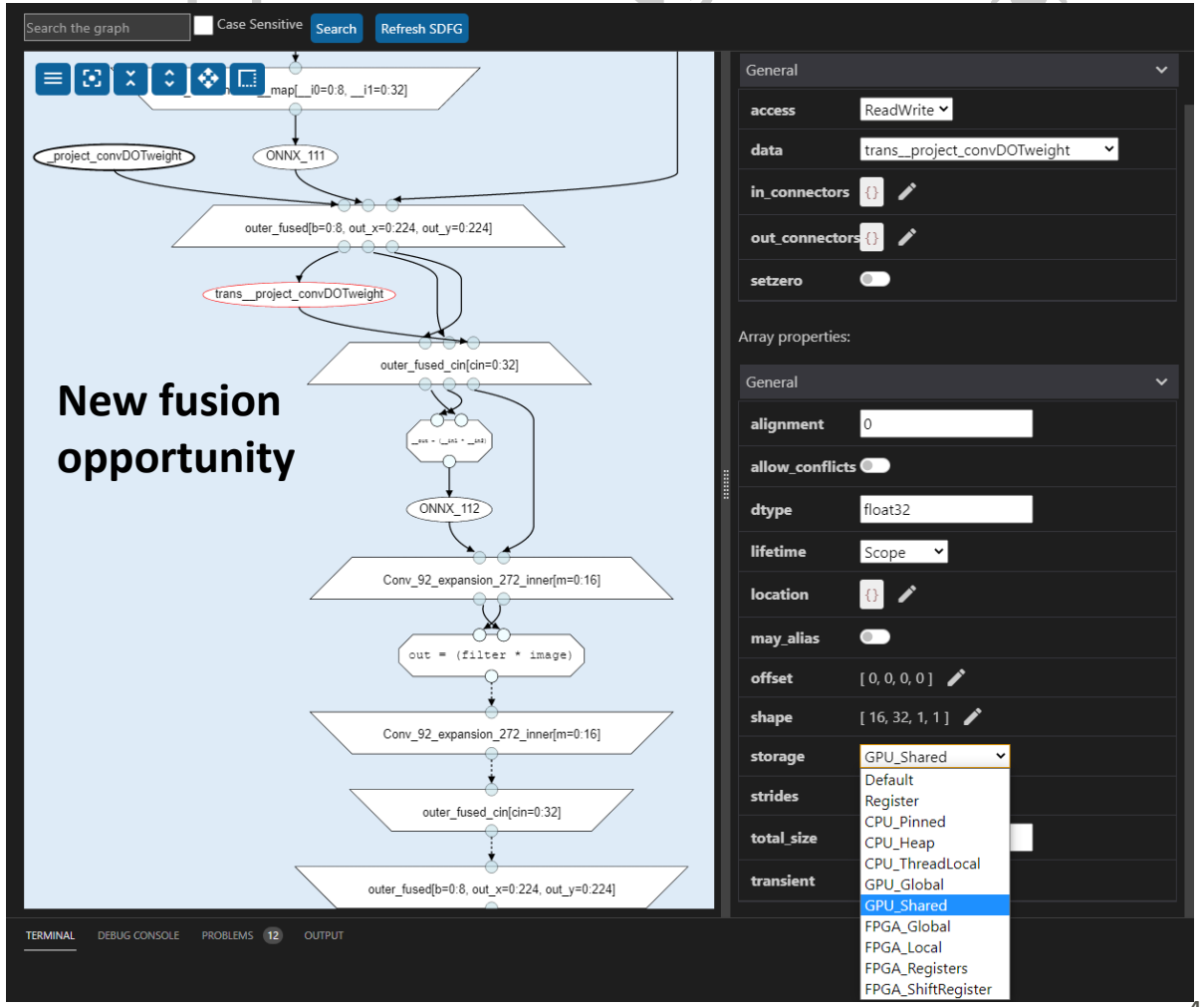
	EfficientNet	$BERT_{LARGE}$
State of the Art:	6.37 ms	8.11 ms
DaCeML **Guided**:	**5.97 ms**	**7.62 ms**

Rausch*, Ben-Nun* et al., A Data-Centric Optimization Framework for Machine Learning, ICS'22.

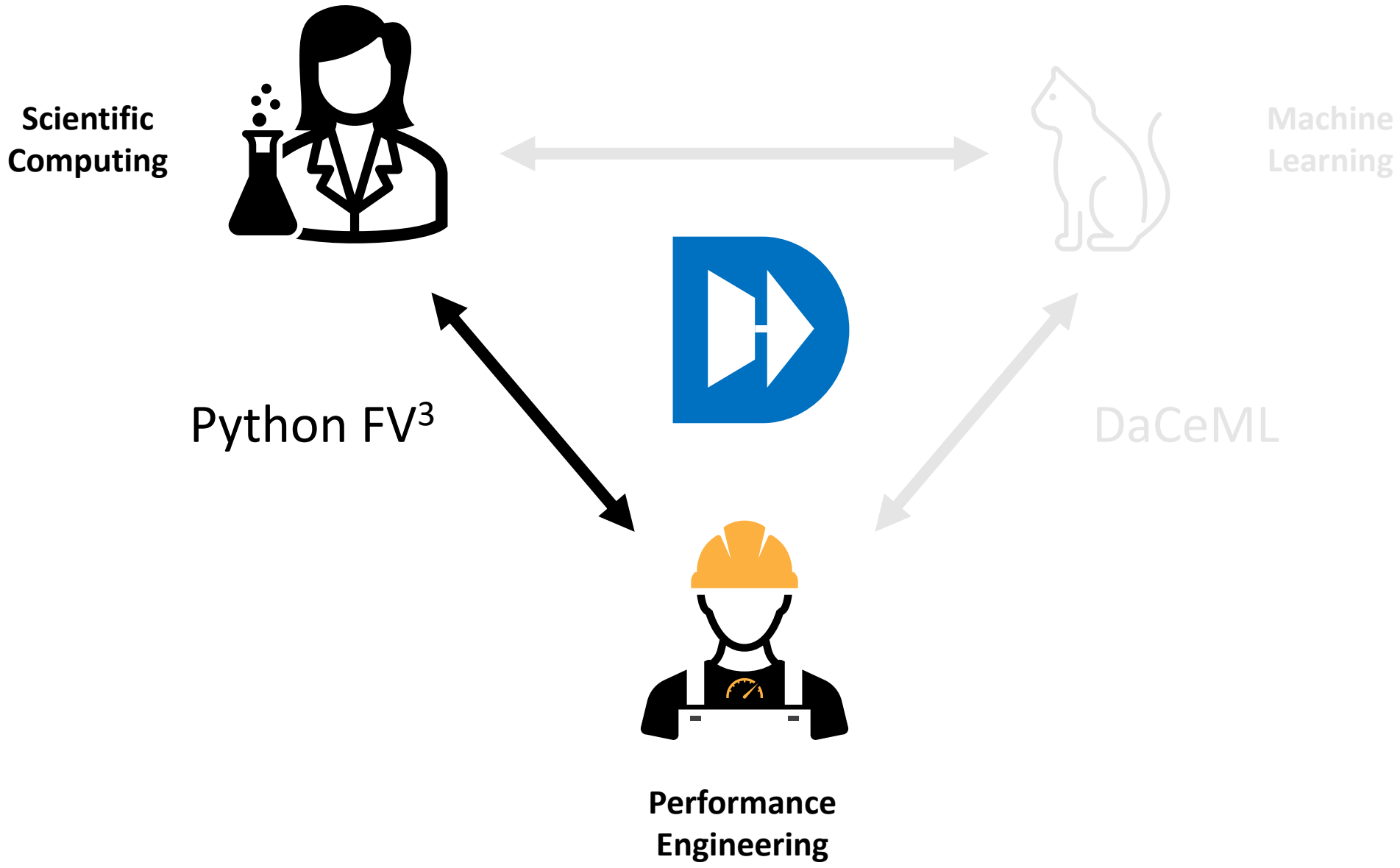
DaCeML Results – Networks vs. PyTorch, `torch.jit`, JAX, TF+XLA; with a Tesla V100 GPU

	ResNet-50		Wide ResNet-50-2
State of the Art:	31.94 ms		70.83 ms
DaCeML:	32.45 ms	2.09x	**67.99 ms**

	EfficientNet
State of the Art:	6.37 ms
DaCeML **Guided**:	**5.97 ms**

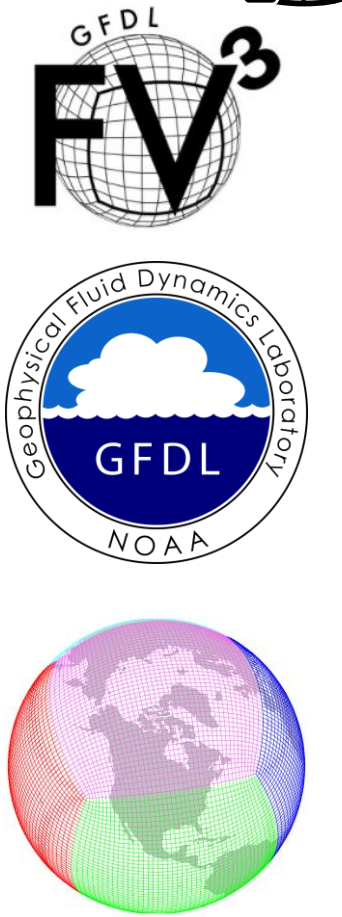


Rausch*, Ben-Nun* et al., A Data-Centric Optimization Framework for Machine Learning, ICS'22.


Scientific Computing
Machine Learning
Python FV³
DaCeML
Performance Engineering

The Pace Project

- **Build a high-resolution atmospheric model entirely in Python that can run at scale on modern supercomputers**
- **Model of choice: Finite-Volume Cubed-Sphere (FV3GFS) global climate model**
- **Distributed across at least 6 nodes (faces of the cubed sphere)**
- **Full dynamical core: 12,450 Python LoC across 36 modules**
- **Stencils powered by an embedded DSL: GridTools for Python (GT4Py)**
- **Baseline: x86-optimized production FORTRAN**



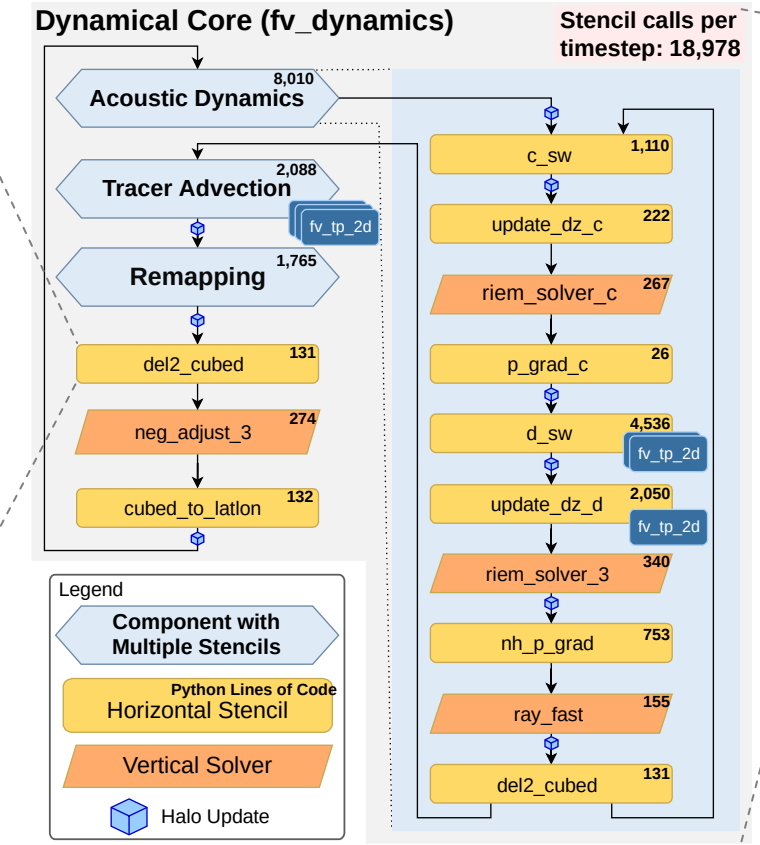
```
Usage: python -m pace.driver.run [OPTIONS] CONFIG_PATH

  Run the driver.

  CONFIG_PATH is the path to a DriverConfig yaml file.

Options:
  ...
```

Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

FV3 Dynamical Core

```python
class HyperdiffusionDamping:
  # ...
  def __call__(self, qdel: FloatField, cd: float):
    # ...
    for n in range(self._ntimes):
      nt = self._ntimes - (n + 1)
      self._corner_fill(qdel, self._q)

      if nt > 0:
        self._copy_corners_x(self._q)

      self._compute_zonal_flux[n](
          self._fx, self._q, self._del6_v)
      # ...
```

```python
@gtscript.stencil
def compute_zonal_flux(flux: FloatField,
                       a_in: FloatField,
                       del_term: FloatFieldIJ):
    with computation(PARALLEL), interval(...):
      flux = del_term * (a_in[-1, 0, 0] - a_in)
```



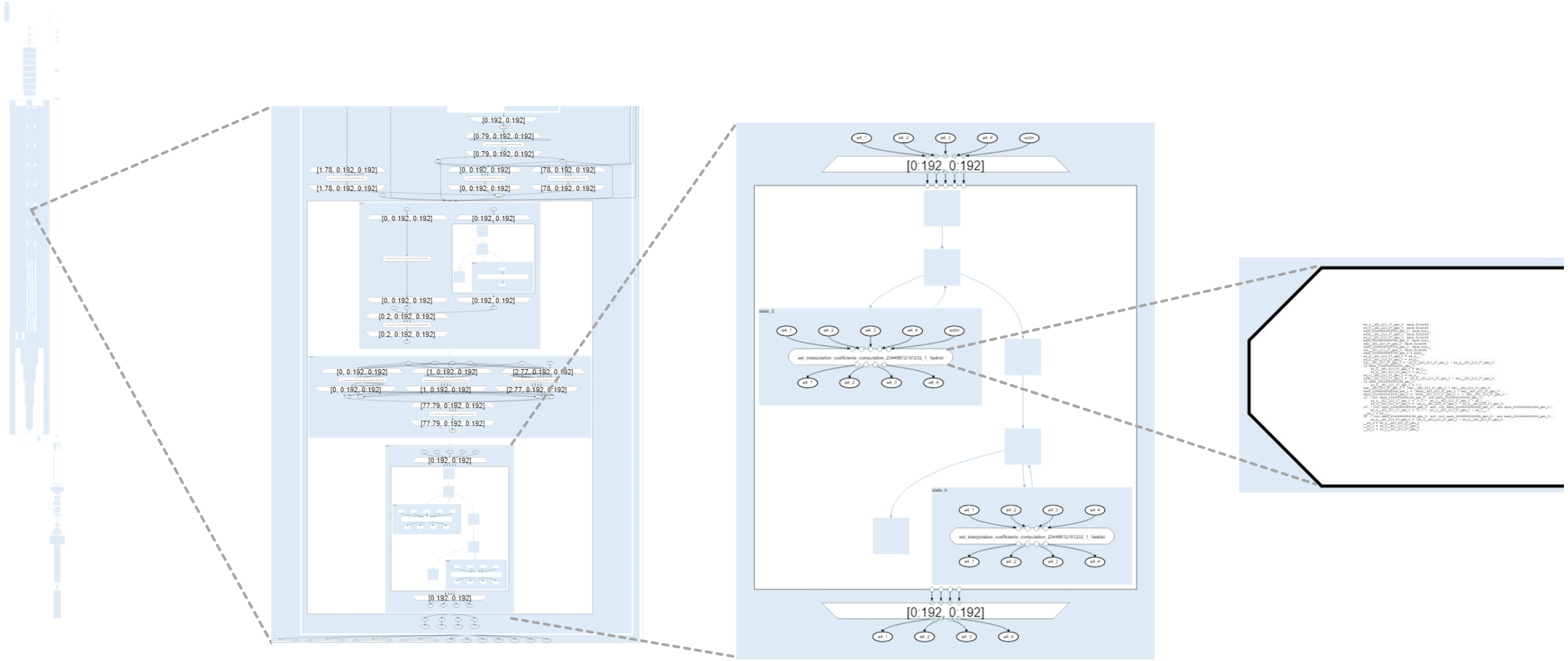

```python
@dace
def dycore_loop(state, dycore, time_steps):

  for _ in range(time_steps):
    dycore.step_dynamics(state)
```

Compile
libdycore_loop.so

```python
state = initialize_state(...)  # Data loading
dycore = fv_dynamics.DynamicalCore(...)

# Invoke compiled function
dycore_loop(state, dycore, T)

validate(state)

plot_on_map(state.x_wind)
```

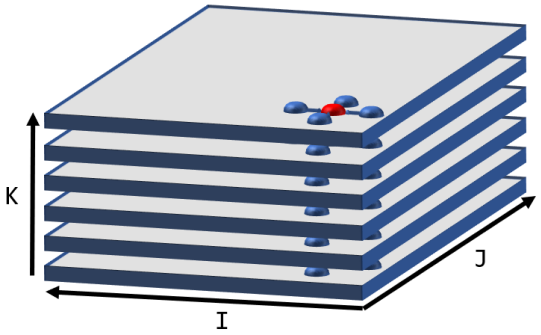
dynamics.py

Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

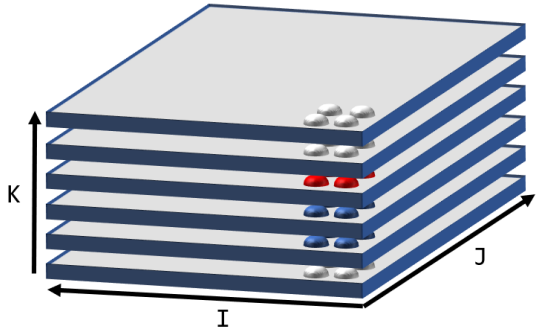


Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

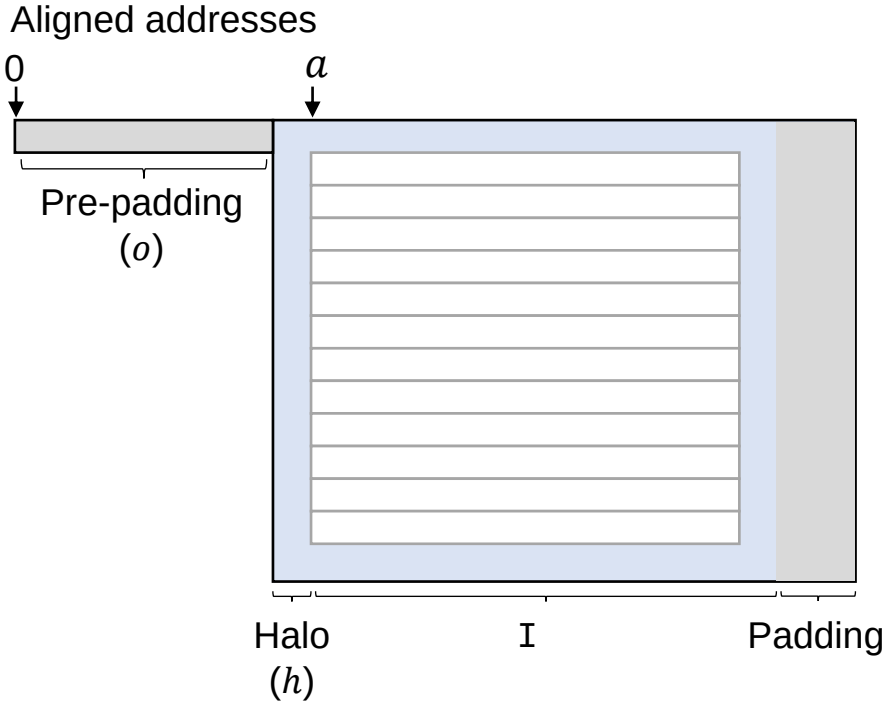
Initial Heuristics



Interval, Operation, K, J, I



J, I, Interval, Operation, K



Shape: $(I + 2h,\ J + 2h,\ K)$

Start offset: $o = a - h$

Strides:

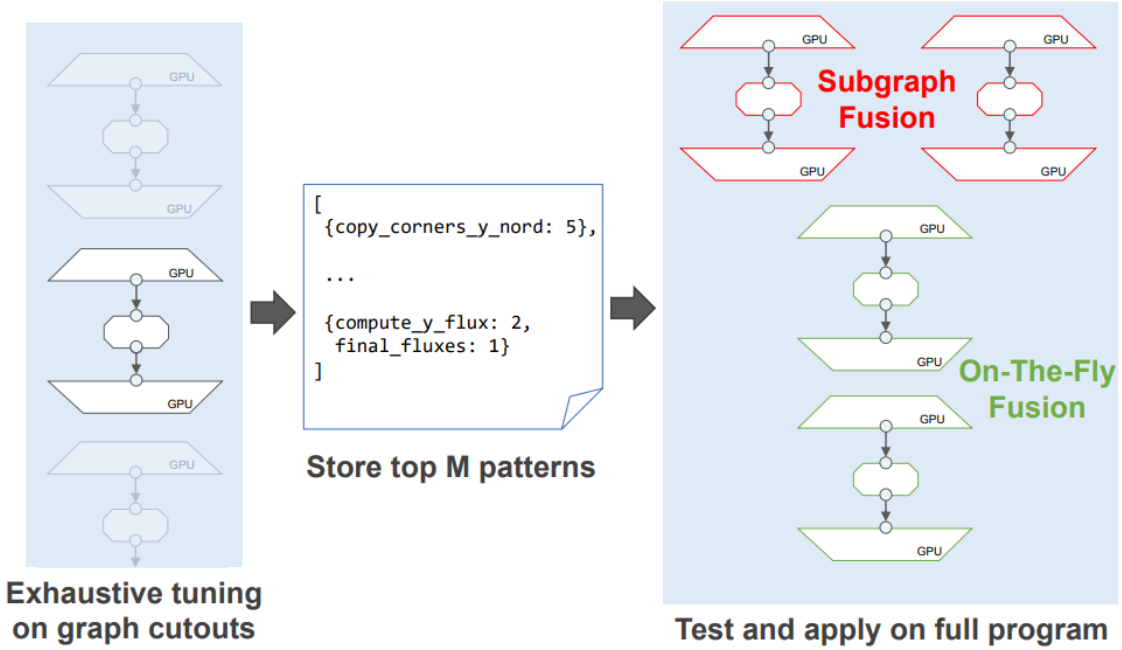
$$s_i = 1$$

$$s_j = a \left\lceil \frac{I + 2h}{a} \right\rceil$$

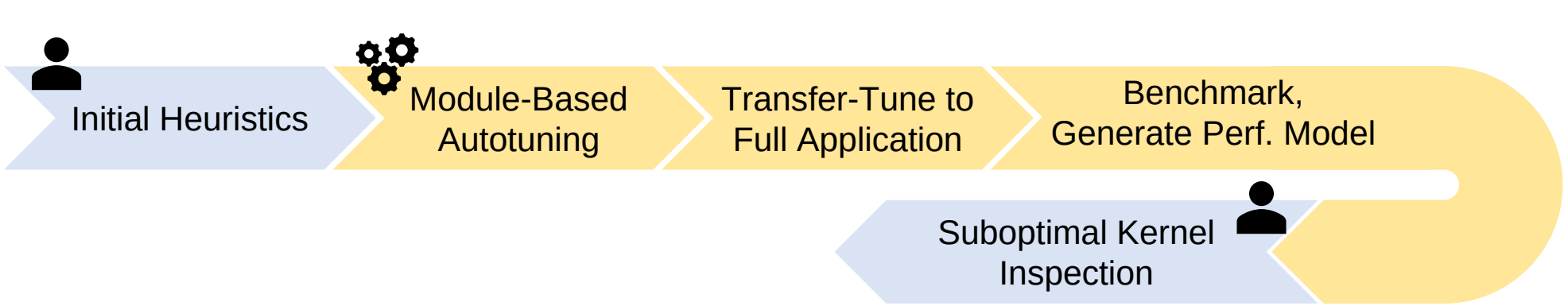
$$s_k = s_j \cdot (J + 2h)$$

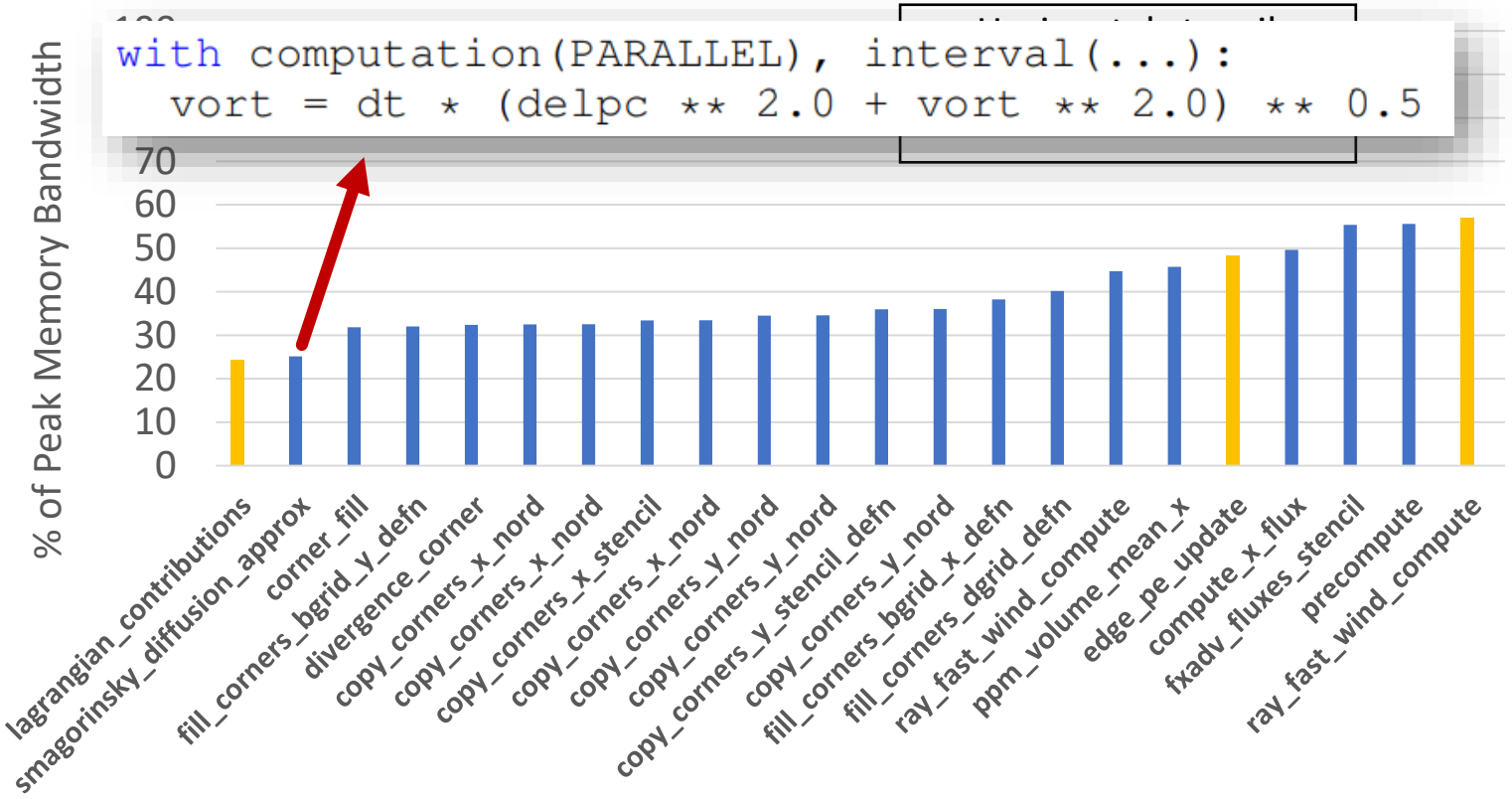
Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

Initial Heuristics → Module-Based Autotuning → Transfer-Tune to Full Application

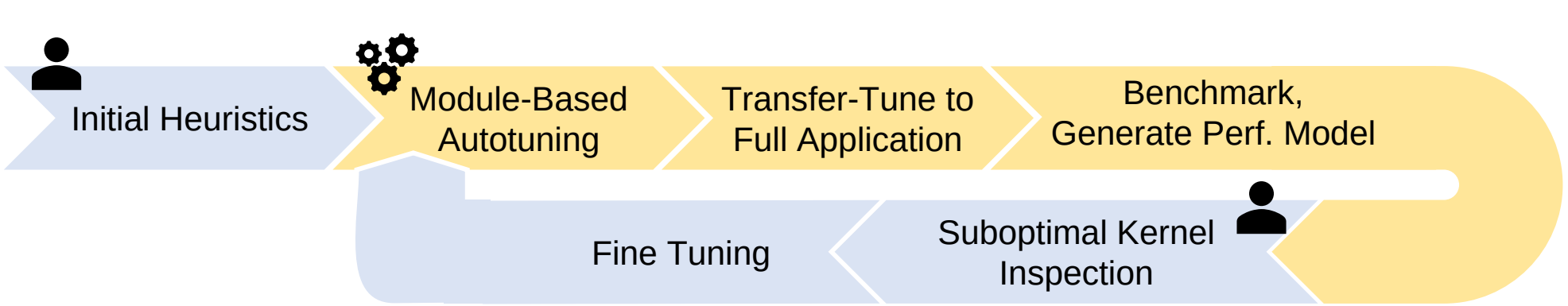


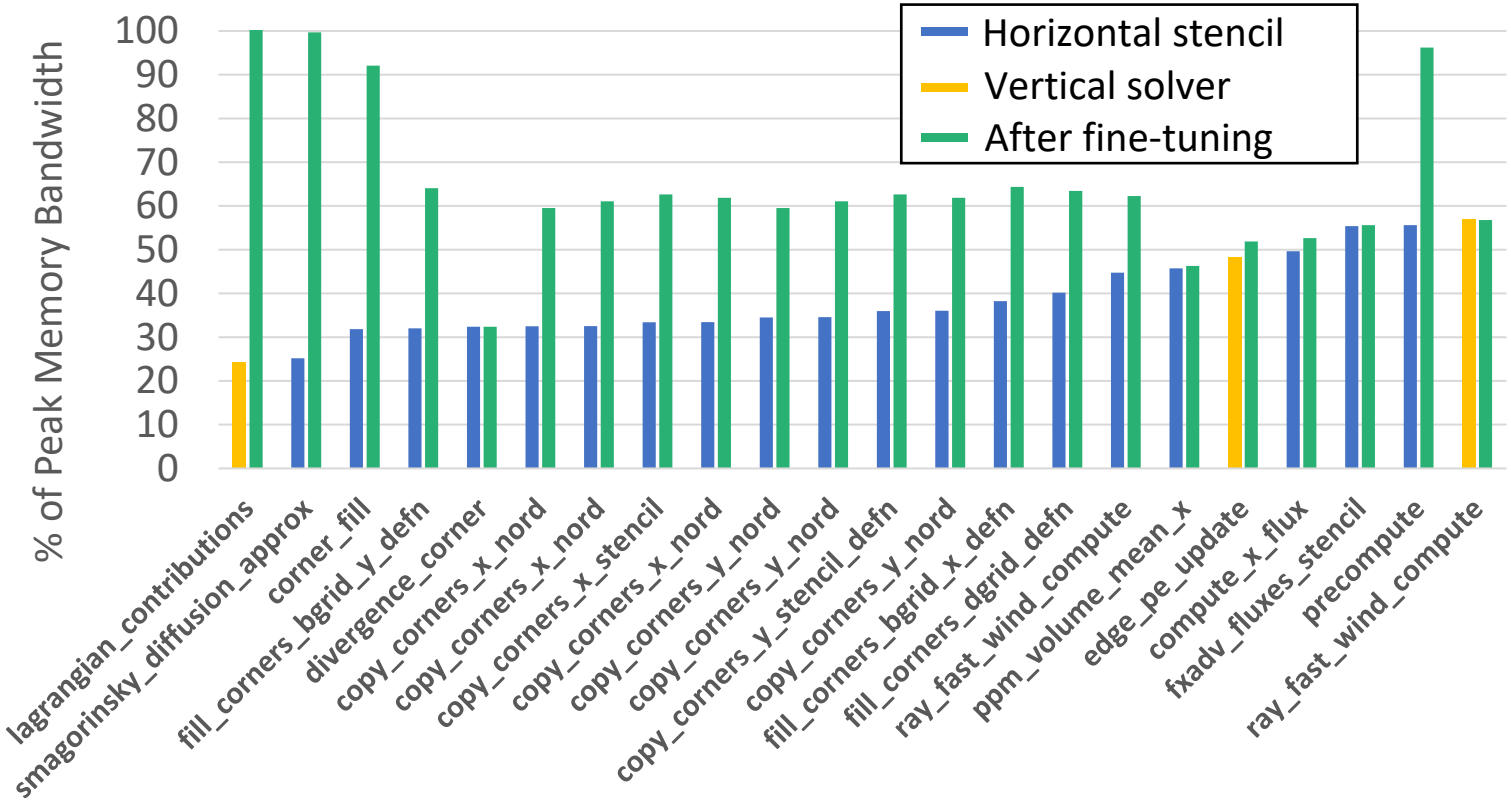
Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.





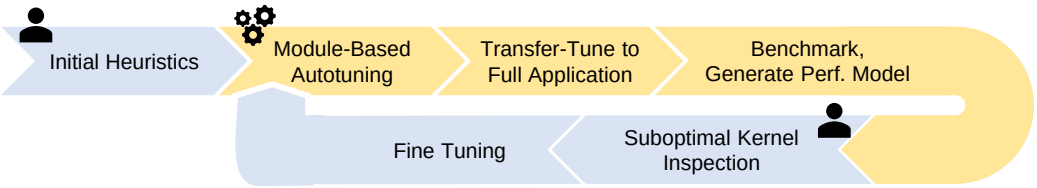
Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

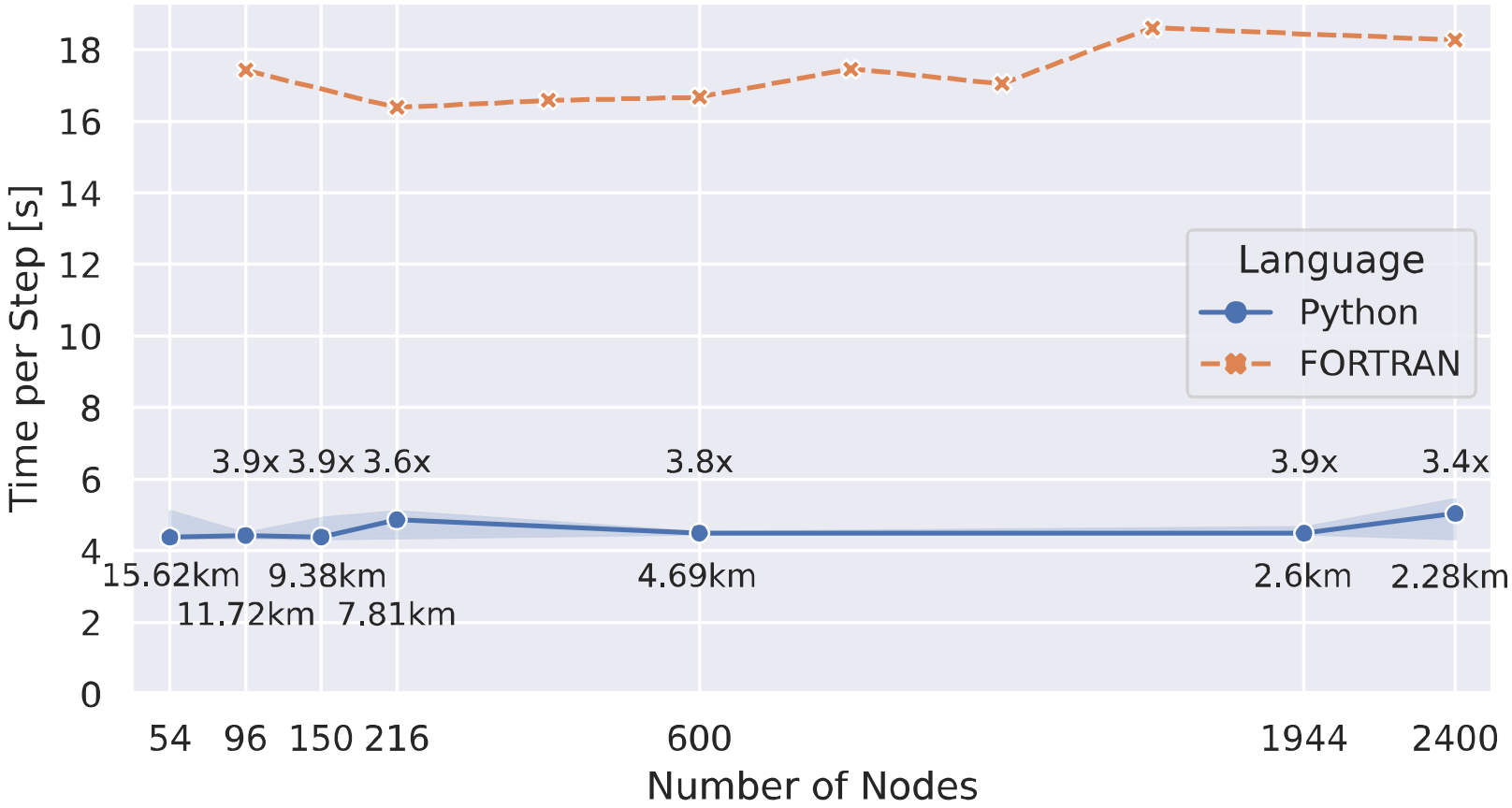




Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

Weak Scaling





Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.

6
weeks of
work

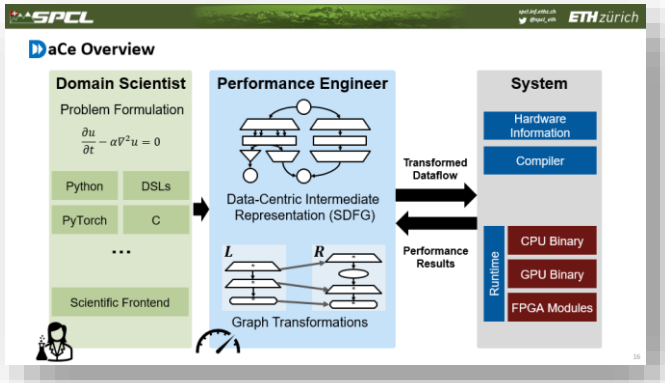
10
code
revisions

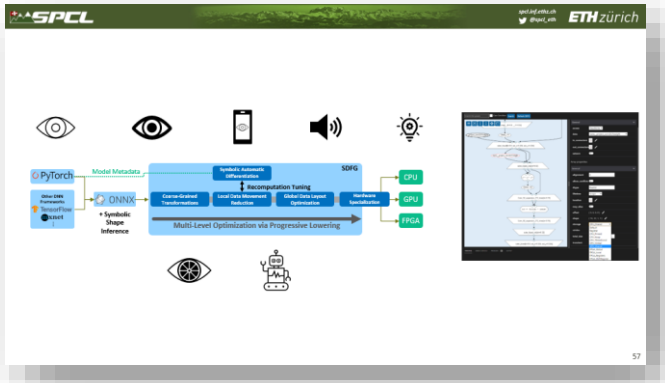
4
performance
engineers

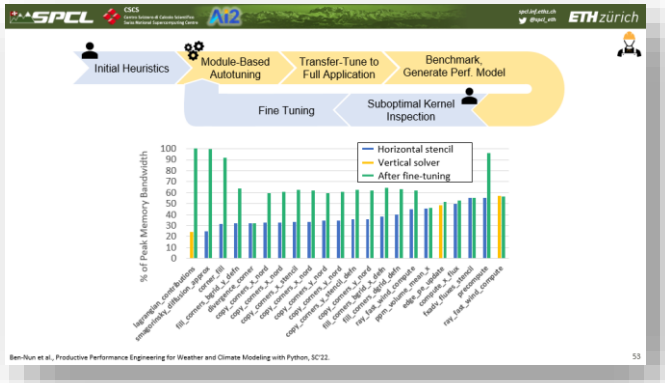
3.92 – 8.48x
speedup vs.
production FORTRAN

0
model
changes

Ben-Nun et al., Productive Performance Engineering for Weather and Climate Modeling with Python, SC'22.







https://www.github.com/spcl/dace

```
pip install dace
pip install daceml
```