DaCe.Python: Performance Metaprogramming Towards Spatial Computers

Torsten Hoefler – Keynote at PERMAVOST'23 in conjunction with FCRC'23, Orlando, FL, June 2023

Tal Ben-Nun, Alexandros Ziogas, Johannes de Fine Licht, Tiziano de Matteis, Timo Schneider, Andreas Kuster, Manuel Burger, Philip Schaad, Dominic Hofer and the whole DAPP team @ SPCL as well as many external and industry collaborators

Why Python as Programmer Frontend?

Python is the lingua franca of computational, data sciences, and AI

TIOBE Index for June 2023

June Headline: Will Python remain number 1?

Python has been the TIOBE index annual award winner for 3 times in the last 5 years. It has grown in popularity like crazy, due to boosts in the fields of data sciences and artificial intelligence. The rise started somewhere in the autumn of 2017 with a share of 3% and ended at the end of last year with a share of 17%. This year, Python couldn't keep this all time high of 17% and dropped back to 13%. The other 3 contenders for the first position, C, Java and C++, are getting closer now. Will Python remain number 1? This depends, I think, mainly on the popularity of AI. If tools such as ChatGPT remain the talk of the day, it will attrack new comers and then Python is definitely here to stay. If not, Python should fear for its first position. Apart from this battle for the first place, we see two interesting new languages entering the top 50 for the first time this month: X++ (the language used by Microsoft Dynamics) and Raku (the fork/successor of Perl). -- *Paul Jansen CEO TIOBE Software*

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Jun 2023	Jun 2022	Change	Programming Language	Ratings	Change
1	1		Python	12.46%	+0.26%
2	2		C	12.37%	+0.46%
3	4	▲	C++	11.36%	+1.73%
4	3	▼	Java	11.28%	+0.81%
5	5		C#	6.71%	+0.59%
6	6		Visual Basic	3.34%	-2.08%

DaCe also supports C and Fortran but not as part of this talk ☺

It's All About the Ecosystem – and NumPy (forget about lists & dictionaries if you're after performance)

IPython

MD ANALYSIS

GridTools

jupyter

SciPy

seaborn

pandas

TensorFlow

matplotlib

SymPy

python Package Index

NetworkX
Network Analysis in Python

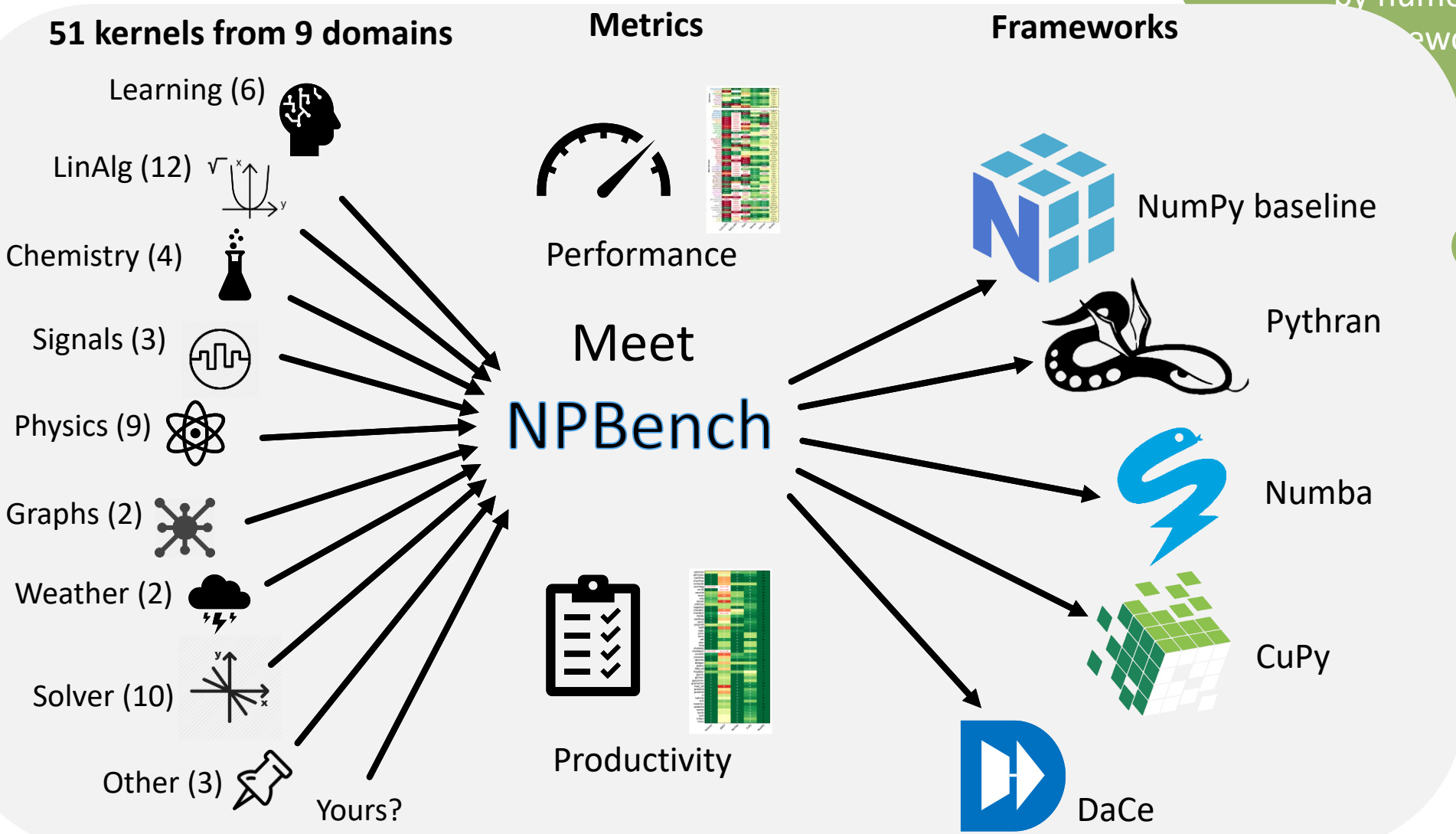
PyTorch

439,100 projects

Scientific Python – a Long History of Optimization

NumPy is accelerated by numerous frameworks!

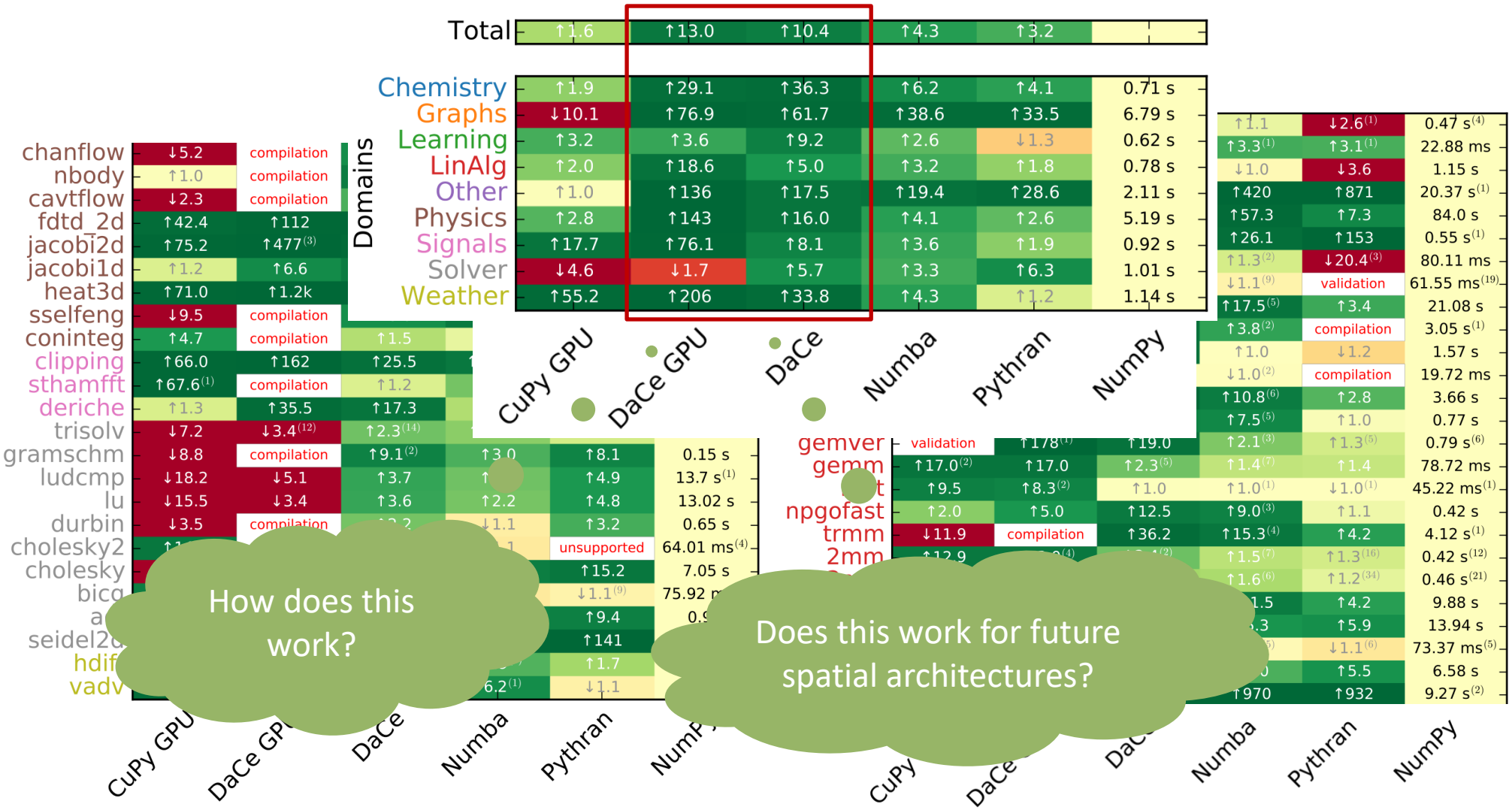
We need a fair comparison between those



https://github.com/spcl/npbench

NPBench results

Machine with two 16-core Intel Xeon Gold 6130 processors and an Nvidia V100 GPU with 32GB of memory



Changing hardware constraints and the physics of computing

How to address locality challenges on standard architectures and programming?

D. Unat et al.: “**Trends in Data Locality Abstractions for HPC Systems**”

IEEE Transactions on Parallel and Distributed Systems (TPDS). Vol 28, Nr. 10, IEEE, Oct. 2017

Three Ls of modern computing:

Spatial Locality

Temporal Locality

Control Locality

[1]: Ma e can do about it), ISSC 2014, plenary

[2]: Moore: Landauer Limit Demonstrated, IEEE Spectrum 2012

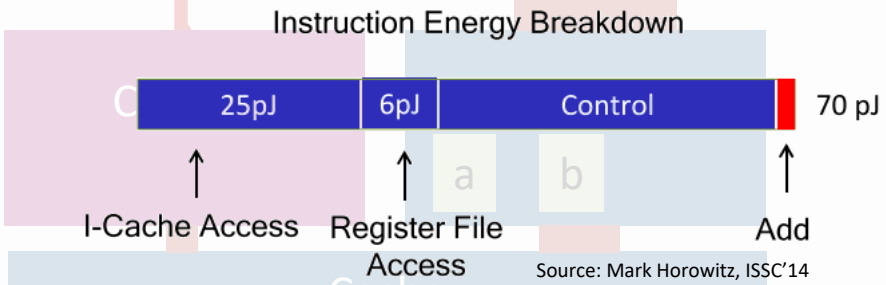
Control in Load-store vs. Dataflow

Turing Award 1977 (Backus): "Surely there must be a less primitive way of making big changes in the store than pushing vast numbers of words back and forth through the von Neumann bottleneck."

Load-store ("von Neumann")

x=a+b

Energy per instruction: 70pJ



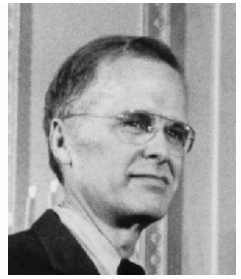
Very Low

st r1, x r2

Memory

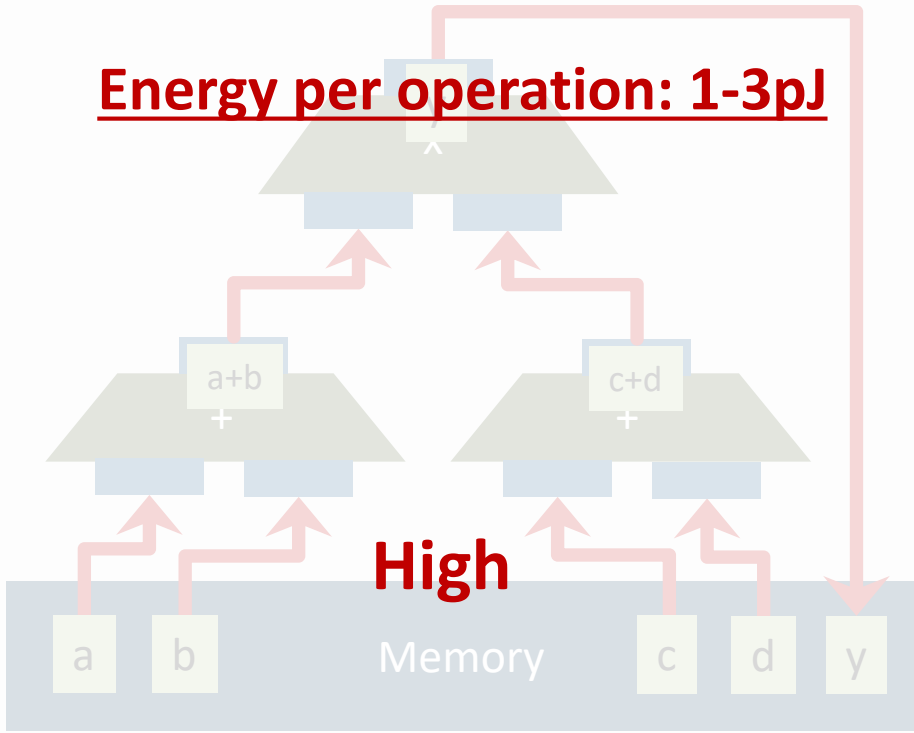
a b

Static Dataflow ("non von Neumann")



y=(a+b)*(c+d)

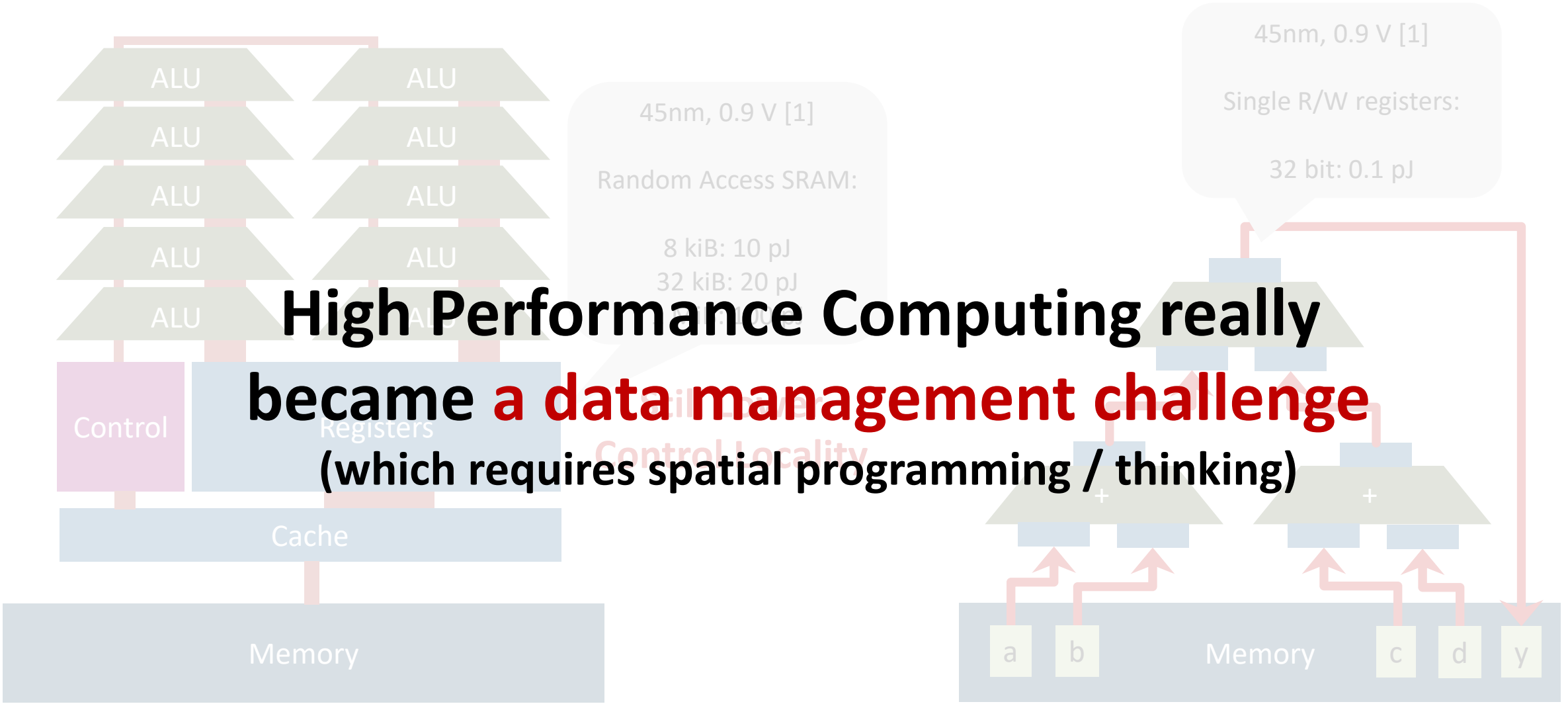
Energy per operation: 1-3pJ



High

Control Locality

Single Instruction Multiple Data/Threads (SIMD - Vector CPU, SIMT - GPU)

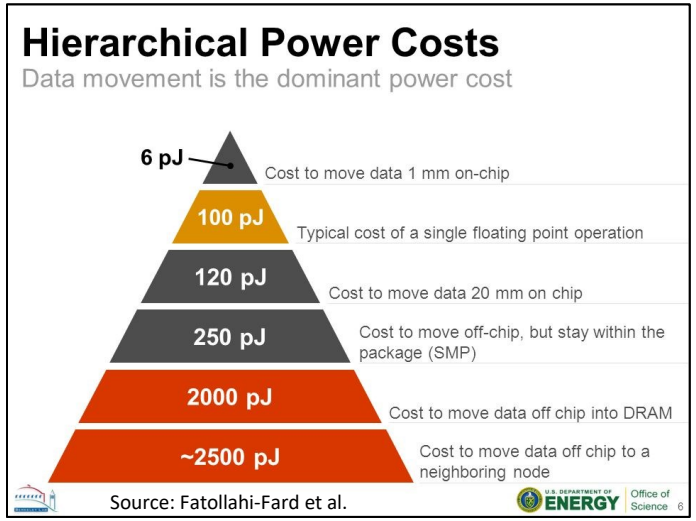


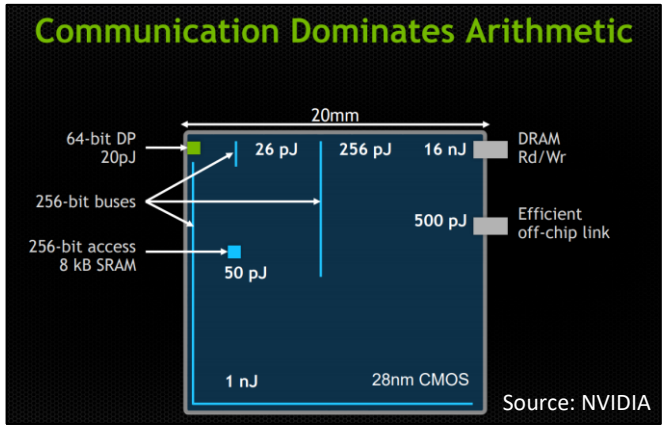
High Performance Computing really became a data management challenge

(which requires spatial programming / thinking)

[1]: Marc Horowitz, Computing's Energy Problem (and what we can do about it), ISSC 2014, plenary

Data movement dominates everything!





Data Movement Is All You Need: A Case Study on Optimizing Transformers

Andrei Ivanov*, Nikoli Dryden*, Tal Ben-Nun, Shigang Li, Torsten Hoefler
ETH Zürich
firstname.lastname@inf.ethz.ch
* Equal contribution

Outstanding paper at MLSys'21

[cs.LG] 2 Jul 2020

Abstract—Transformers have become widely used for language modeling and sequence learning tasks, and are one of the most important machine learning workloads today. Training one is a very compute-intensive task, often taking days or weeks, and significant attention has been given to optimizing transformers. Despite this, existing implementations do not efficiently utilize GPUs. We find that data movement is the key bottleneck when training. Due to Amdahl's Law and massive improvements in compute performance, training has now become memory-bound. Further, existing frameworks use suboptimal data layouts. Using these insights, we present a recipe for globally optimizing data movement in transformers. We reduce data movement by up to 22.91% and overall achieve a 1.30× performance improvement over state-of-the-art frameworks when training BERT. Our approach is applicable more broadly to optimizing deep neural networks, and offers insight into how to tackle emerging performance bottlenecks.

Index Terms—Data movement, high-performance computing, deep learning, transformers

challenges such as artificial general intelligence [27]. Thus, improving transformer performance has been in the focus of numerous research and industrial groups.

Significant attention has been given to optimizing transformers: local and fixed-window attention [28]–[32], more general structured sparsity [33], learned sparsity [34]–[36], and other algorithmic techniques [19], [37] improve the performance of transformers. Major hardware efforts, such as Tensor Cores and TPUs [38] have accelerated tensor operations like matrix-matrix multiplication (MMM), a core transformer operation. Despite this, **existing implementations do not efficiently utilize GPUs**. Even optimized implementations such as Megatron [18] report achieving only 30% of peak GPU flop/s.

We find that **the key bottleneck when training transformers is data movement**. Improvements in compute performance have reached the point that, due to Amdahl's Law and the

DOI:10.1145/1941487.1941507

Energy efficiency is the new fundamental limiter of processor performance, way beyond numbers of processors.

BY SHEKHAR BORKAR AND ANDREW A. CHIEN

The Future of Microprocessors

- "In future microprocessors, the energy expended for data movement will have a critical effect on achievable performance."
- "… movement consumes almost 58 watts with hardly any energy budget left for computation."
- "…the cost of data movement starts to dominate."
- "…data movement over these networks must be limited to conserve energy…"
- the phrase "data movement" appears 18 times on 11 pages (usually in concerning contexts)!
- "Efficient data orchestration will increasingly be critical, evolving to more efficient memory hierarchies and new types of interconnect tailored for locality and that **depend on sophisticated software to place computation and data so as to minimize data movement**."

“Sophisticated software”: How do we program today?

- **Well, to a good approximation how we programmed yesterday**
  - Or last year?
  - Or four decades ago?

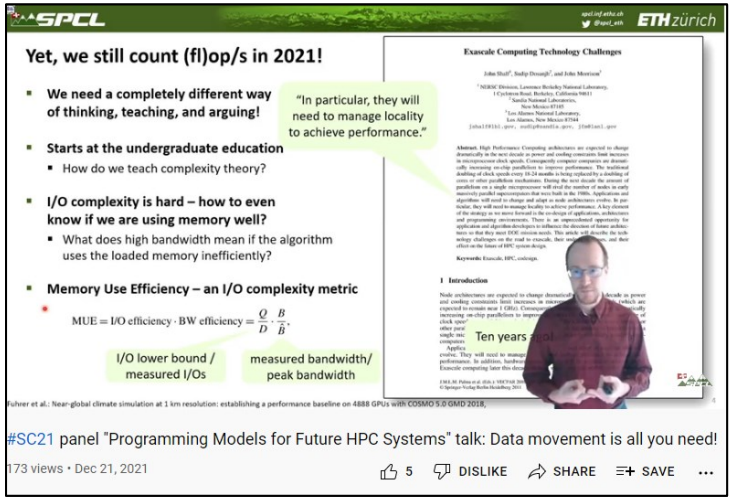
- **Control-centric programming**
  - Worry about operation counts (flop/s is **the metric**, isn’t it?)
  - Data movement is at best implicit (or invisible/ignored)
  - In fact, this is how we think about algorithms

    Do this then that

Backus ‘77: *“The assignment statement is the von Neumann bottleneck of programming languages and keeps us thinking in word-at-a-time terms in much the same way the computer’s bottleneck does.”*

- **How to cross the chasm between algorithms and data-centric computing?**
  - Domain scientists and algorithm developers remain in the control-centric land
  - Performance engineers enter the data-centric/spatial land
  - An intermediate representation connects the two – both may need to adapt!

    → *Performance Metaprogramming connects the two!*



Upleveling programming in the 21st century – Performance Metaprogramming

10s of SLOC

Domain Scientist

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

NumPy DSLs TensorFlow PyTorch MATLAB

Applied Scientist

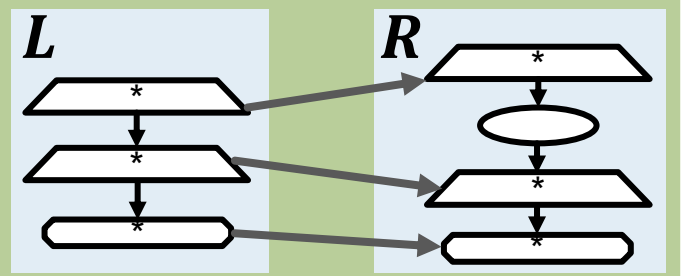
translate DSL into an Internal Representation

SDFG Builder API
Multi-Level Library Nodes

Performance Engineer

100s of reusable SLOC

IR – e.g., Parametric Dataflow Graphs (SDFG)



Graph Transformations (API, Interactive)

Transformed Dataflow

Performance Results

1000s of auto-generated SLOC

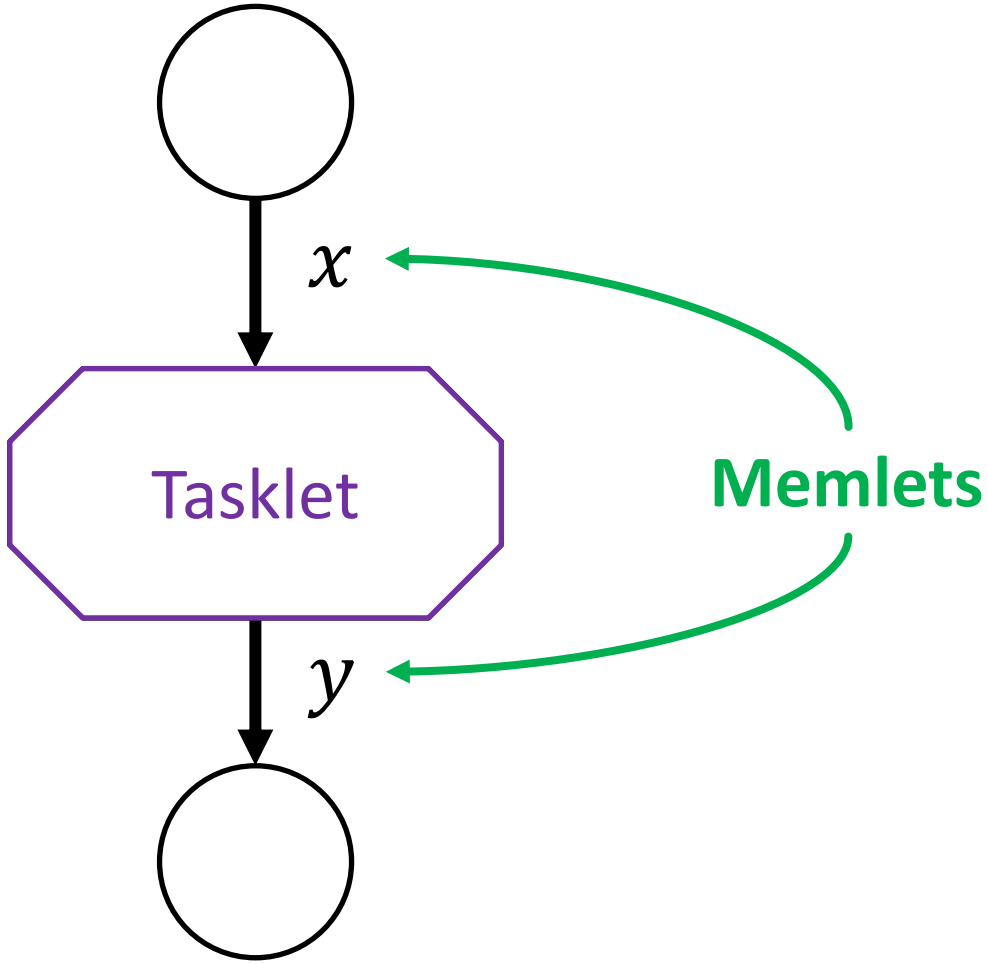
Specialized Code Generation

Runtime
CPU Code
GPU Code
FPGA Code

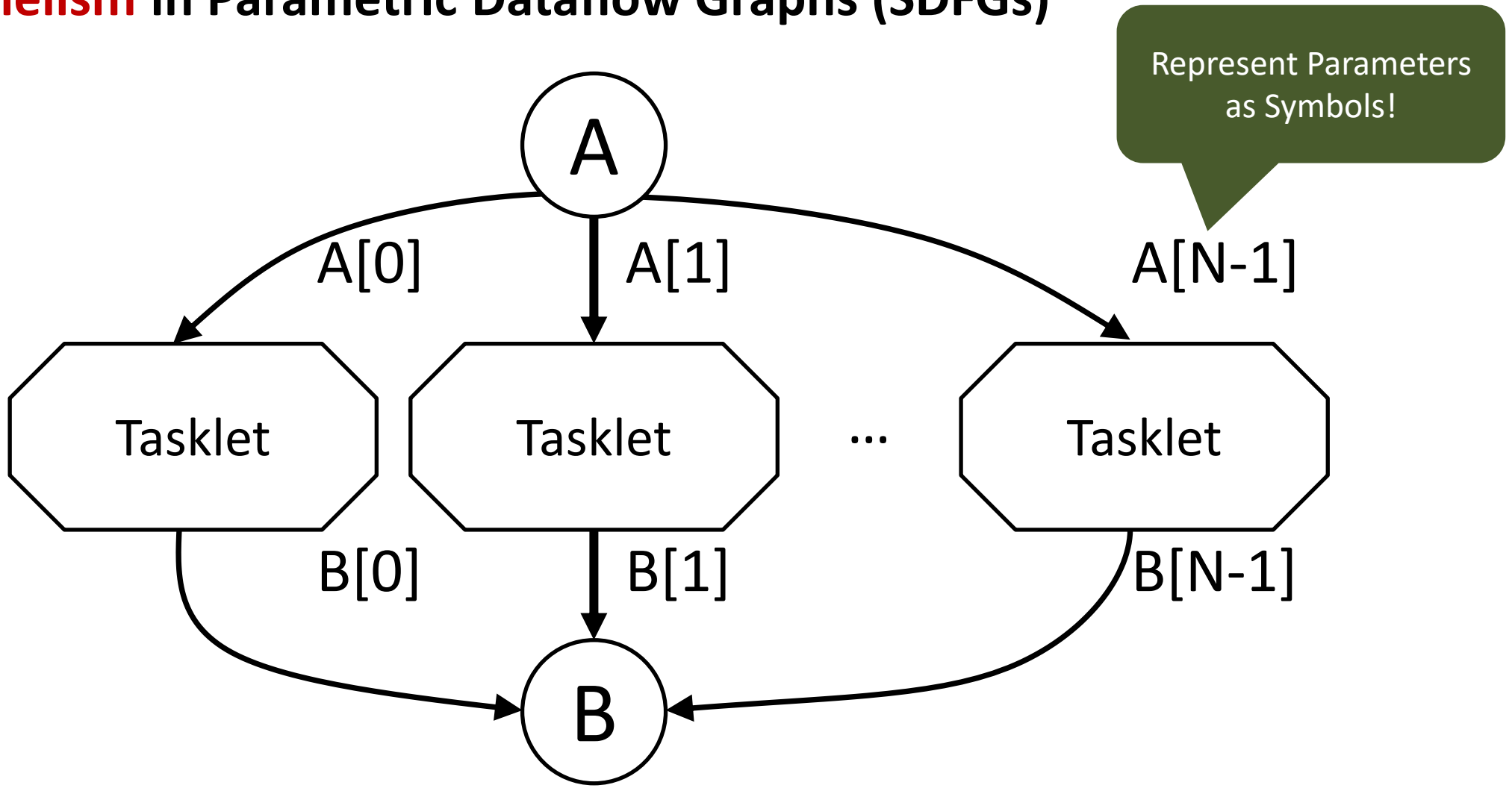
C++ code generation/runtime

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

DaCe.Python translates to Parametric Dataflow Graphs (SDFGs) as IR

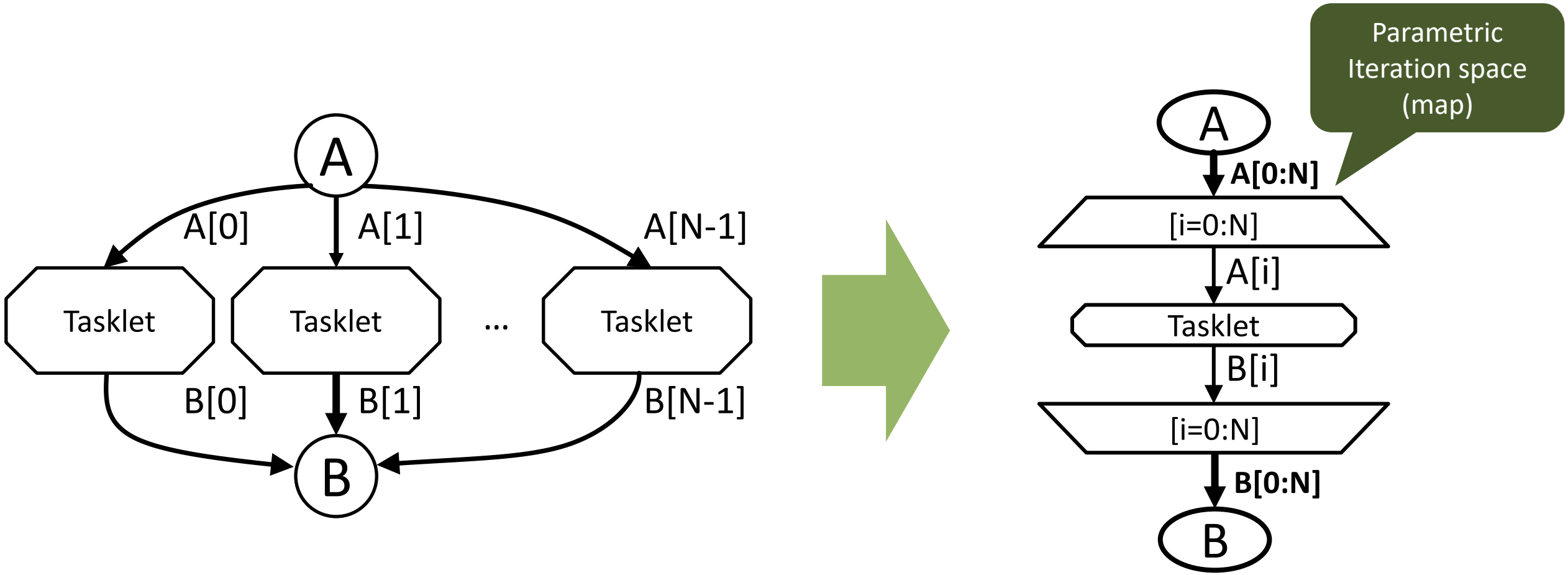


Data-Parallelism in Parametric Dataflow Graphs (SDFGs)

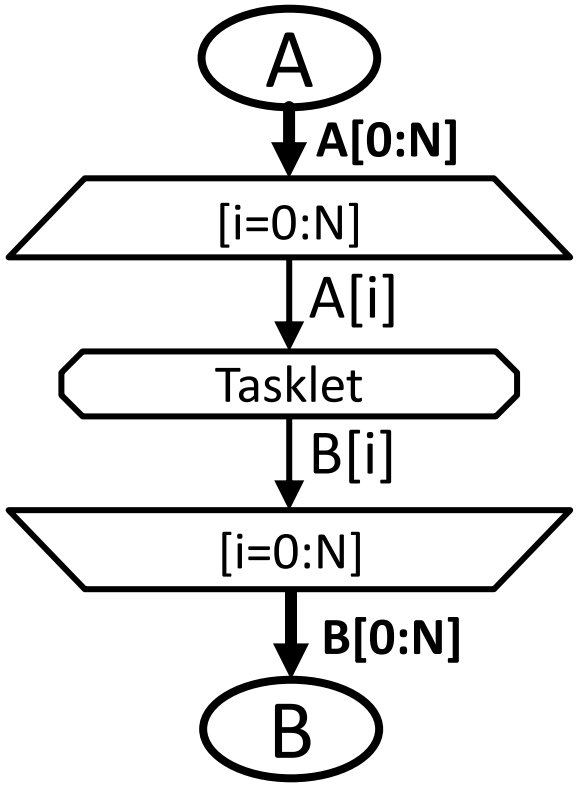


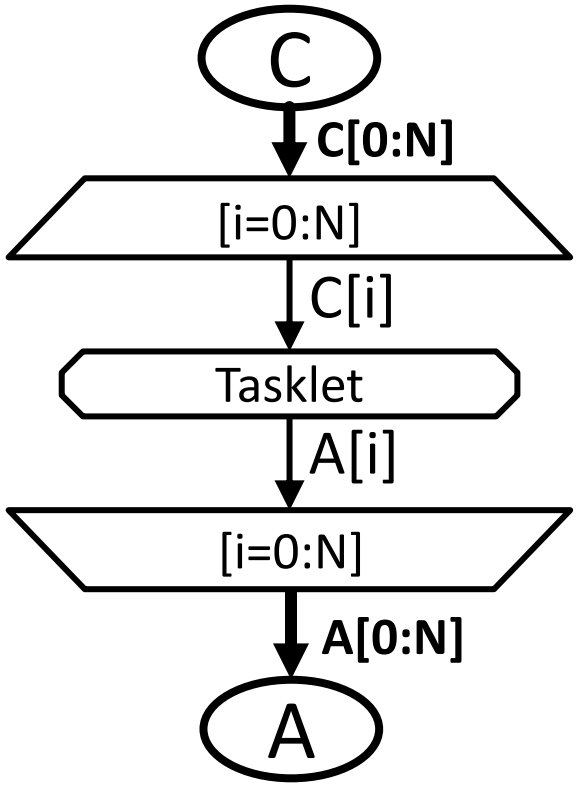
Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Data-Parallelism in Parametric Dataflow Graphs (SDFGs)



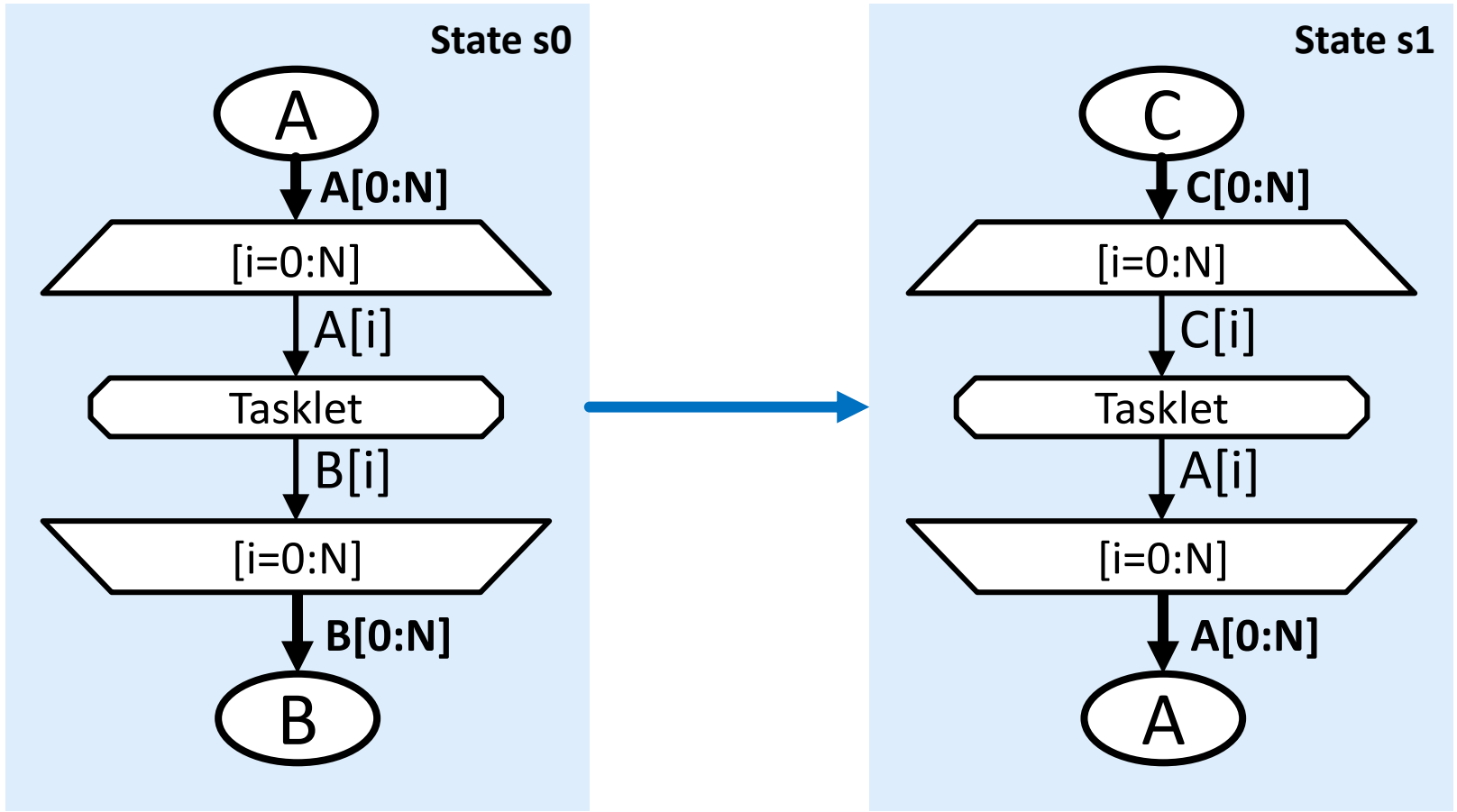
Non-dataflow ordering: States in Parametric Dataflow Graphs (SDFGs)





Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Non-dataflow ordering: States in Parametric Dataflow Graphs (SDFGs)



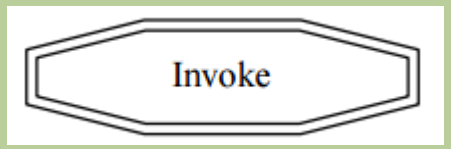
Parametric Dataflow Graphs - Concepts

Data Containers

- Store volatile (buffers, queues, RAM) and nonvolatile (files, I/O) information
- Can be sources or sinks of data

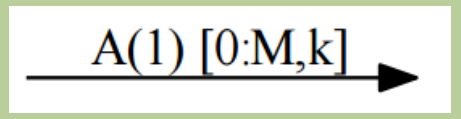
Computation

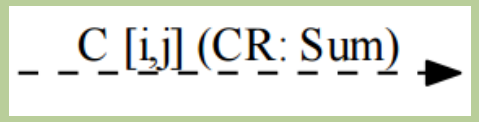
- Stateless functions that perform computations at any granularity
- Data access only through ports



Data Movement / Dependencies

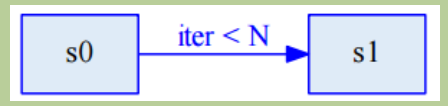
- Data flowing between containers and tasklets/ports
- Implemented as access, copies, streaming, …





Parallelism and States

- Map scopes provide parallelism
- States constrain parallelism outside of datatflow



Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance Engineer view: Visual Studio Code Integration & Transformations

SDFG OPTIMIZATION

TRANSFORMATIONS
- Selection (0)
- Viewport (14)
- Global (3)
- Uncategorized (0)

TRANSFORMATION HISTORY
- TaskletFusion — Current SDFG
- Original SDFG

gemm.py

GEMM > gemm.py > gemm > B

```python
import dace
import numpy as np

M, N, K = (dace.symbol(s) for s in 'MNK')

@dace.program
def gemm(A: dace.float64[M, K],
         B: dace.float64[K, N],
         C: dace.float64[M, N]):
    tmp = dace.define_local([M, N, K], dtype=A.dtype)
    for i, j, k in dace.map[0:M, 0:N, 0:K]:
        tmp[i, j, k] = A[i, k] * B[k, j]
    C[:] = np.sum(tmp, axis=2)

```

gemm.cpp 2

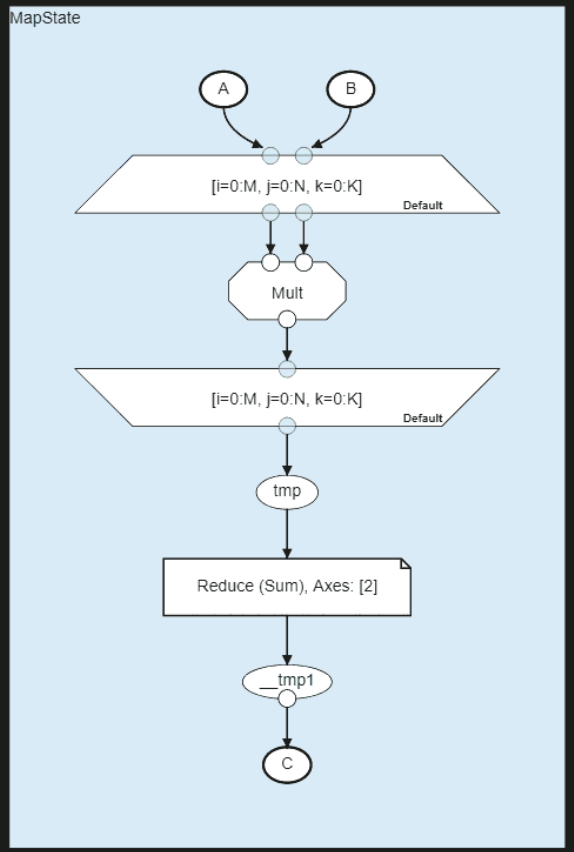
cpu > gemm.cpp > __program_gemm_internal(gemm_t *, double *__restrict__, double *__restrict__, double *__restrict__, in...

```cpp
        tmp = new double DACE_ALIGN(64)[((K * M) * N)];

        {
            #pragma omp parallel for
            for (auto i = 0; i < M; i += 1) {
                for (auto j = 0; j < N; j += 1) {
                    for (auto k = 0; k < K; k += 1) {
                        {
                            double __in1 = A[((K * i) + k)];
                            double __in2 = B[((N * k) + j)];
                            double __out;

                            ////////////////////
                            // Tasklet code (Mult)
                            __out = (__in1 * __in2);
                            ////////////////////

                            tmp[((((K * N) * i) + (K * j)) + k)] = __out;
                        }
                    }
                }
            }
        }
        reduce_0_0_8(__state, &tmp[0], &__tmp1[0], K, M, N);
        delete[] tmp;

    }
```

gemm.sdfg

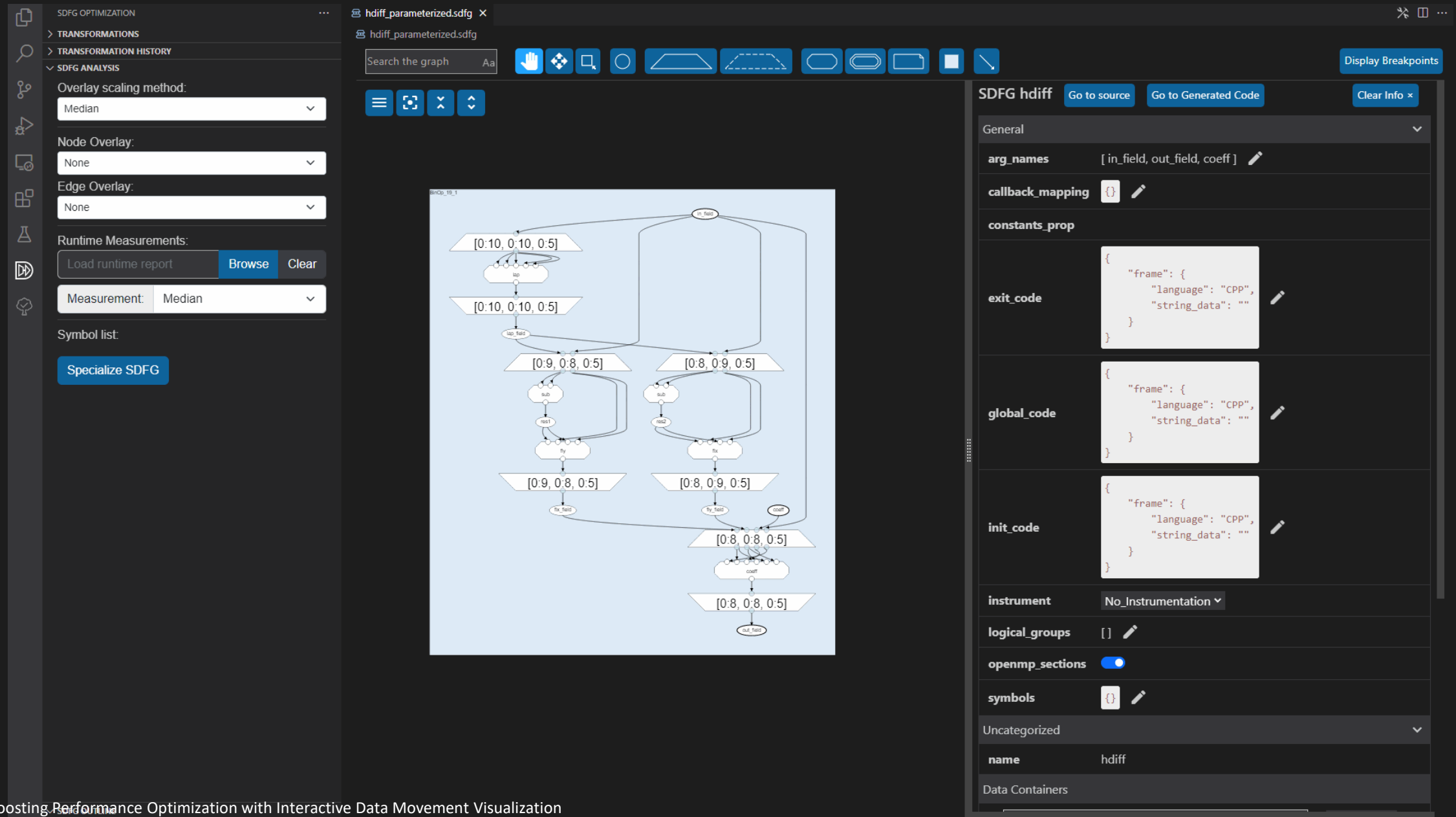
GEMM > gemm.sdfg

Search the graph Aa

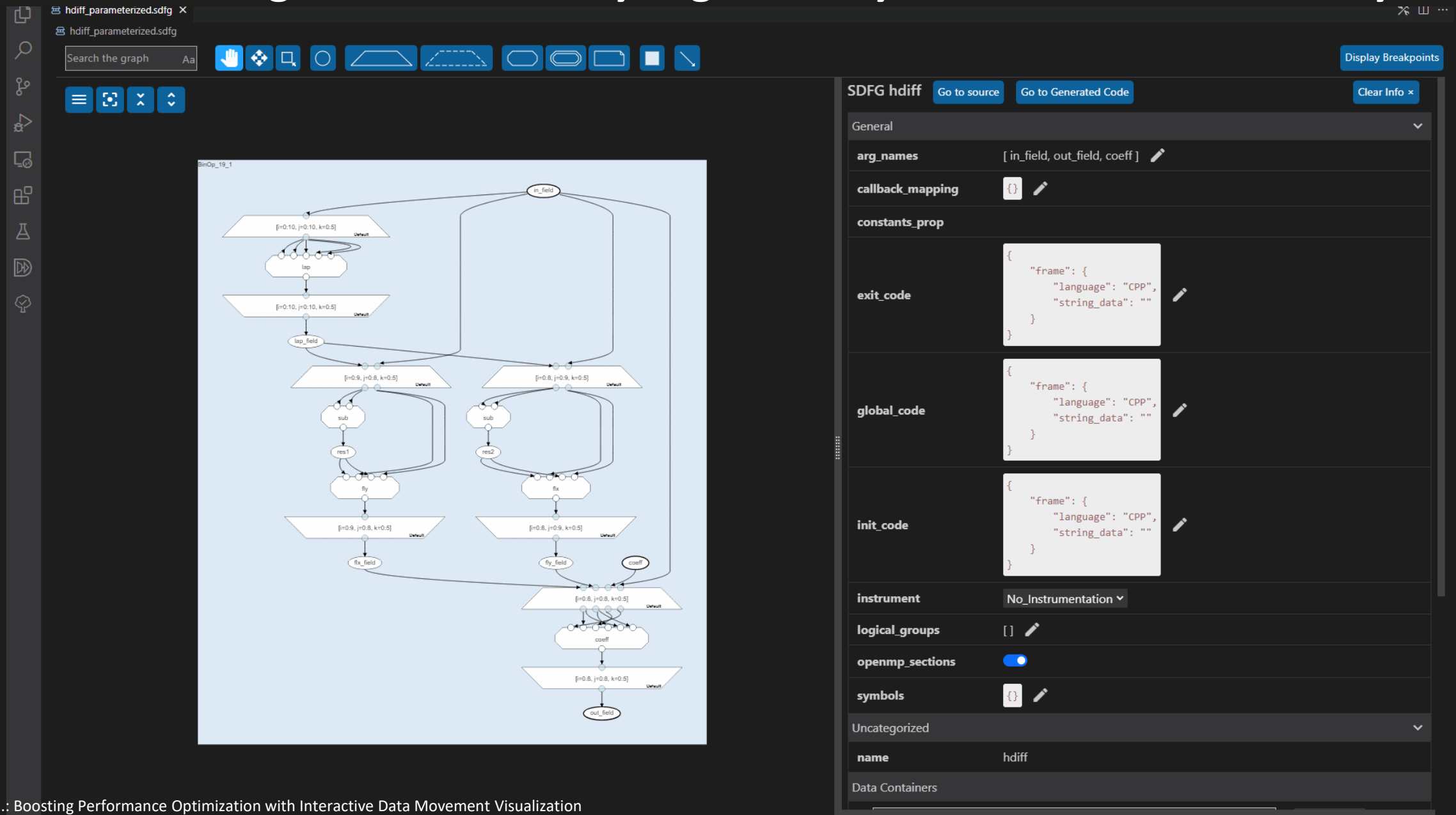


Schaad et al.: Boosting Performance Optimization with Interactive Data Movement Visualization

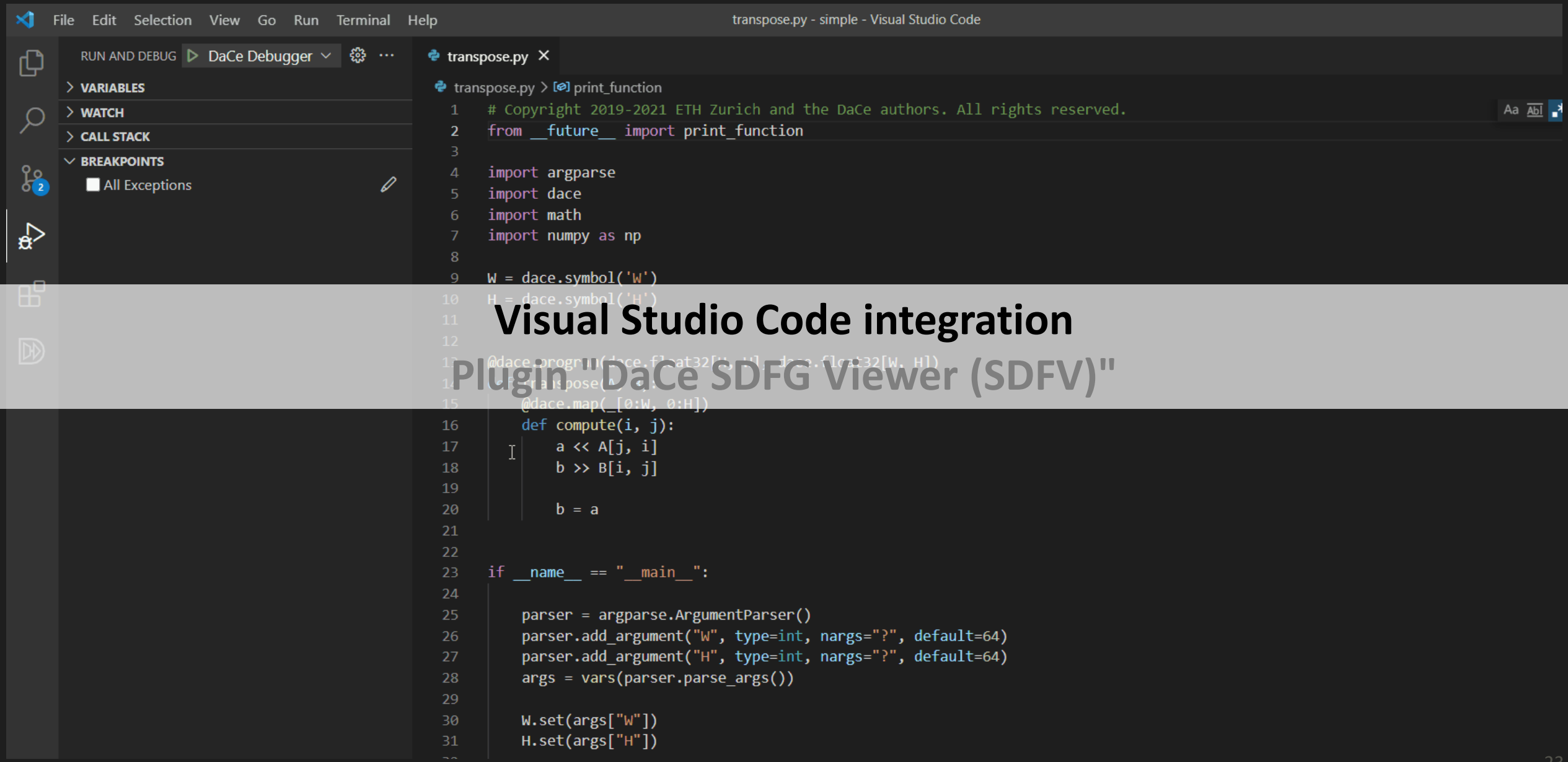
Performance Engineer view: Analyzing Data Flows & Computation Volume



Performance Engineer view: Analyzing Memory Access Patters & Locality



Programmer/Performance Engineer view: Debugging & Code Generation

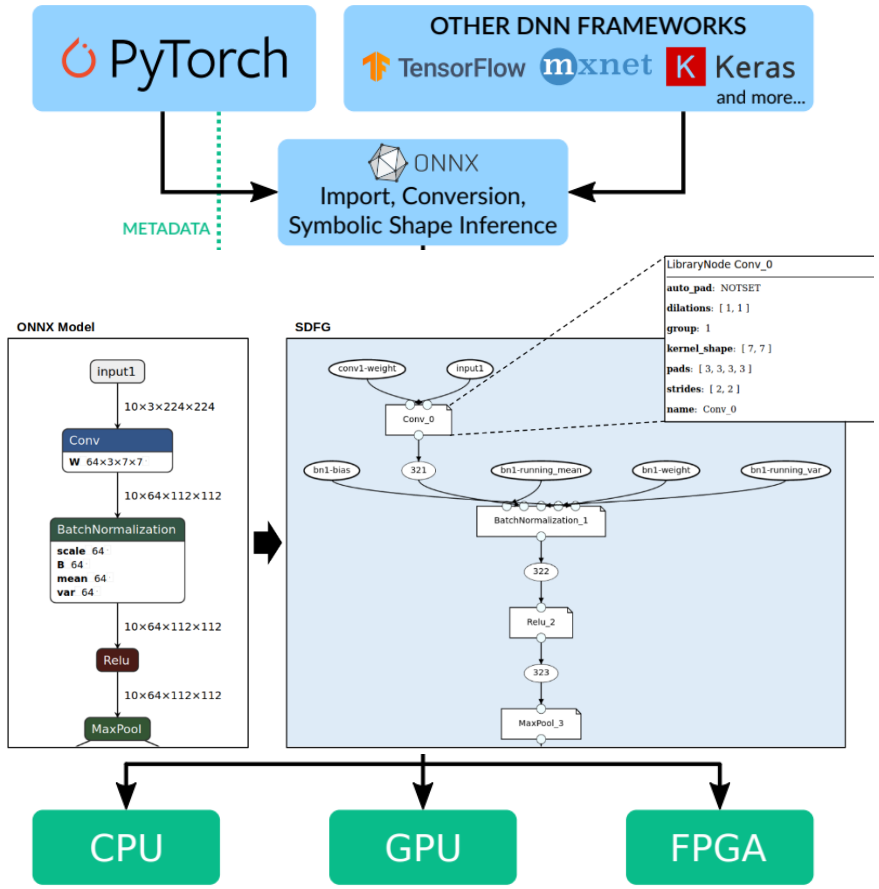


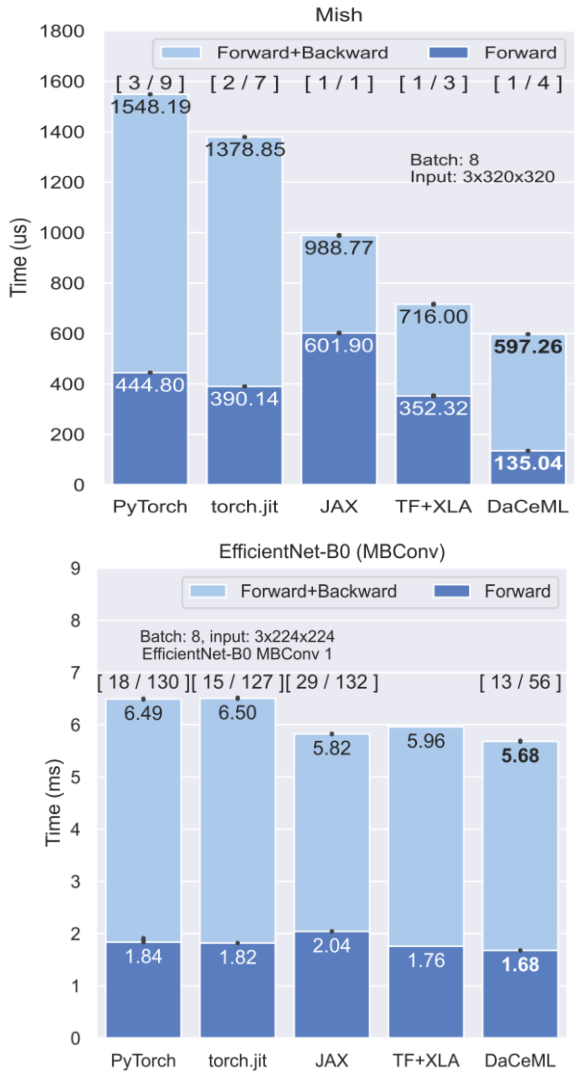
Visual Studio Code integration

Plugin "DaCe SDFG Viewer (SDFV)"

DaCe Offers Full Support for ML Pipelines Through ONNX

Key principle: minimize data movement and optimize data layout





Data Movement Is All You Need: A Case Study on Optimizing Transformers

Andrei Ivanov*, Nikoli Dryden*, Tal Ben-Nun, Shigang Li, Torsten Hoefler
ETH Zürich
firstname.lastname@inf.ethz.ch
* Equal contribution

MLSys'20

***Abstract*—Transformers have become widely used for language modeling and sequence learning tasks, and are one of the most important machine learning workloads today. Training one is a very compute-intensive task, often taking days or weeks, and significant attention has been given to optimizing transformers. Despite this, existing implementations do not efficiently utilize GPUs. We find that data movement is the key bottleneck when training. Due to Amdahl's Law and massive improvements in compute performance, training has now become memory-bound. Further, existing frameworks use suboptimal data layouts. Using these insights, we present a recipe for globally optimizing data movement in transformers. We reduce data movement by up to 22.91% and overall achieve a 1.30× performance improvement over state-of-the-art frameworks when training BERT. Our approach is applicable more broadly to optimizing deep neural networks, and offers insight into how to tackle emerging performance bottlenecks.**

***Index Terms*—Data movement, high-performance computing, deep learning, transformers**

I. Introduction

Transformers [1] are a class of deep neural network architecture for sequence transduction [2], similar to recurrent neural networks [3] and LSTMs [4]. They have recently had a major impact on natural language processing (NLP), including language modeling [5]–[7], question-answering [8], translation [1], and many other applications. The significant improvement in accuracy brought by transformers to NLP tasks is comparable to the improvement brought to computer vision by AlexNet [9] and subsequent convolutional neural networks. Transformers have also begun to be applied to domains beyond NLP where RNNs would previously have been used, including speech recognition [10], reinforcement learning [11], molecular property prediction [12], and symbolic mathematics [13].

Significant attention has been given to optimizing transformers: local and fixed-window attention [28]–[32], more general structured sparsity [33], learned sparsity [34]–[36], and other algorithmic techniques [19], [37] improve the performance of transformers. Major hardware efforts, such as Tensor Cores and TPUs [38] have accelerated tensor operations like matrix-matrix multiplication (MMM), a core transformer operation. Despite this, **existing implementations do not efficiently utilize GPUs**. Even optimized implementations such as Megatron [18] report achieving only 30% of peak GPU flop/s.

We find that **the key bottleneck when training transformers is data movement**. Improvements in compute performance have reached the point that, due to Amdahl's Law and the acceleration of tensor contractions, training is now memory-bound. Over a third (37%) of the runtime in a BERT training iteration is spent in memory-bound operators: While tensor contractions account for over 99% of the flop performed, they are only 61% of the runtime. By optimizing these, we show that the overhead of data movement can be reduced by up to 22.91%. Further, while MMM is highly tuned by BLAS libraries and hardware, we also find that **existing frameworks use suboptimal data layouts**. Using better layouts enables us to speed up MMM by up to 52%. Combining these insights requires moving beyond peephole-style optimizations and **globally optimizing data movement**, as selecting a single layout is insufficient. Overall, we achieve at least 1.30× performance improvements in training over general-purpose deep learning frameworks, and 1.08× over DeepSpeed [39], the state of the art manually-tuned implementation of BERT.

Rausch et al. "A Data-Centric Optimization Framework for Machine Learning", in ICS'22

Gordon Bell Prize 2019 on ORNL's Summit (Top-1 Machine)

- **Gordon Bell Prize 2019**
  - Optimized twice-finalist code OMEN
  - Quantum Nano Transport simulation

    Design of future micro-processors

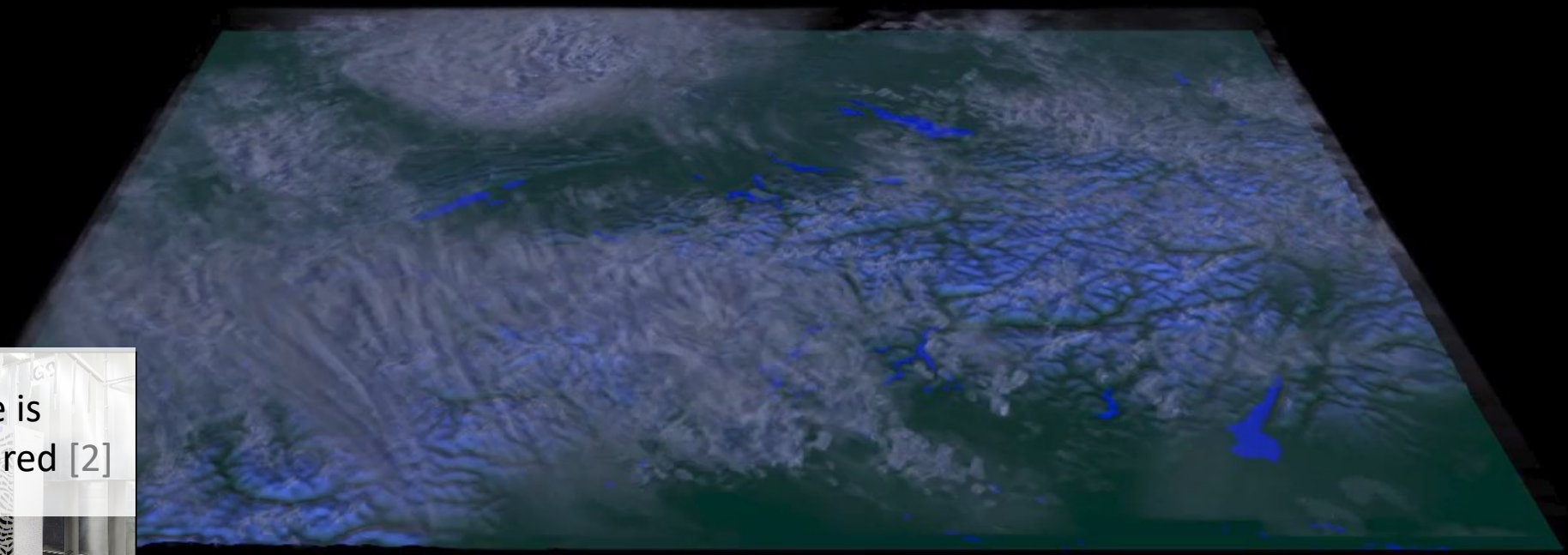
- **Now working on large-scale:**
  - Deep Learning (transformers)
  - Climate (COSMO, icon, fv3)
  - Green's functions solvers
  - … your project?

http://spcl.inf.ethz.ch/DAPP

Case Study: Weather and Climate Code FV3

[1] COSMO 1.1 km

2018-05-29 00:00 UTC+2



[1] Institute for Atmospheric and Climate Science and Computer Graphics Laboratory, ETH Zürich [https://vimeo.com/389292423]
[2] Swiss National Supercomputing Center (CSCS) [https://www.cscs.ch/computers/arolla-tsa-meteoswiss]

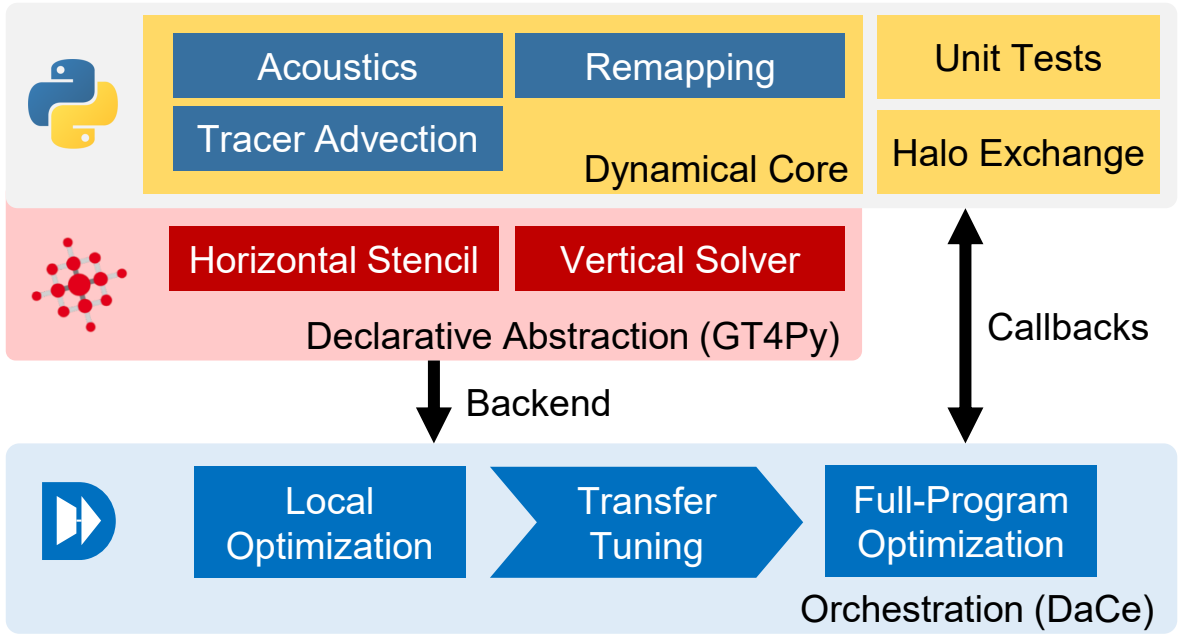
The Pace Project – Rewriting FV3 in Python

https://github.com/ai2cm/pace

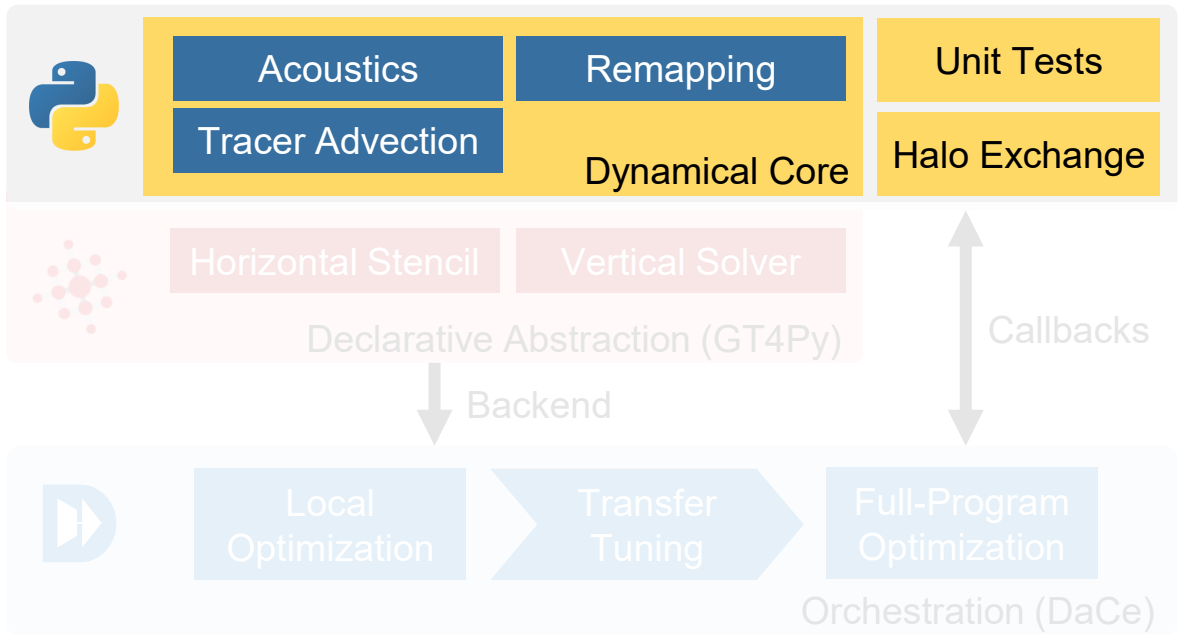
```
Usage: python -m pace.driver.run [OPTIONS] CONFIG_PATH

  Run the driver.

  CONFIG_PATH is the path to a DriverConfig yaml file.

Options:
  ...
```

- **FV3 is a highly optimized atmospheric model in Fortran**
  - Rewrite in Python to run it at scale on modern supercomputers
  - No FORTRAN involved – move to 21st century programming + devops + package management (with similar syntax!)
- **Full dynamical core: 12,450 Python LoC across 36 modules**

  vs. 29,458 in the baseline FORTRAN implementation



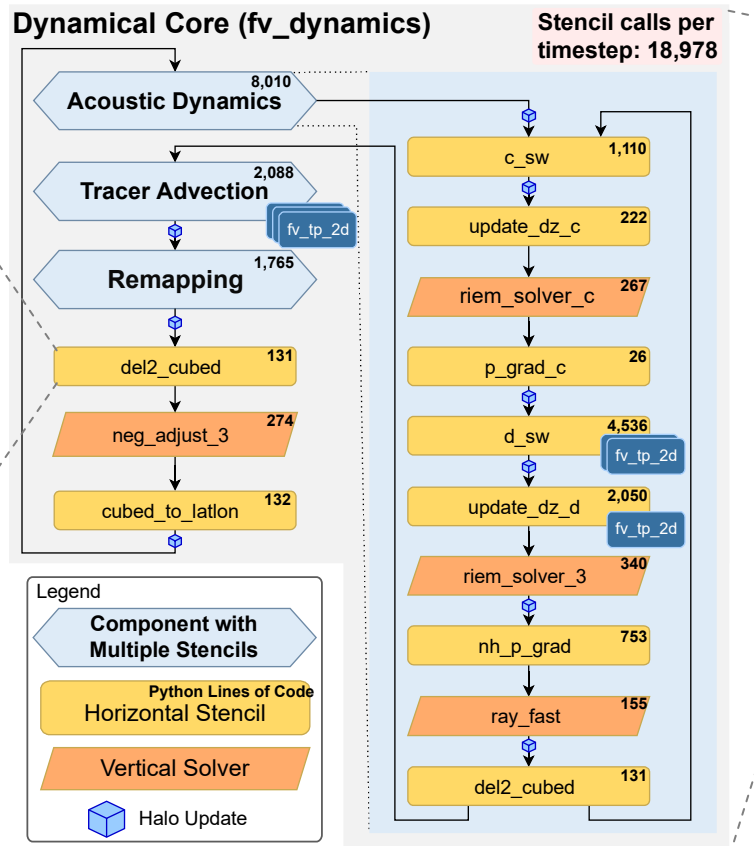
J. Dahm et al., "Pace v0.1: A Python-based Performance-Portable Implementation of the FV3 Dynamical Core". EGUSphere'22



Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

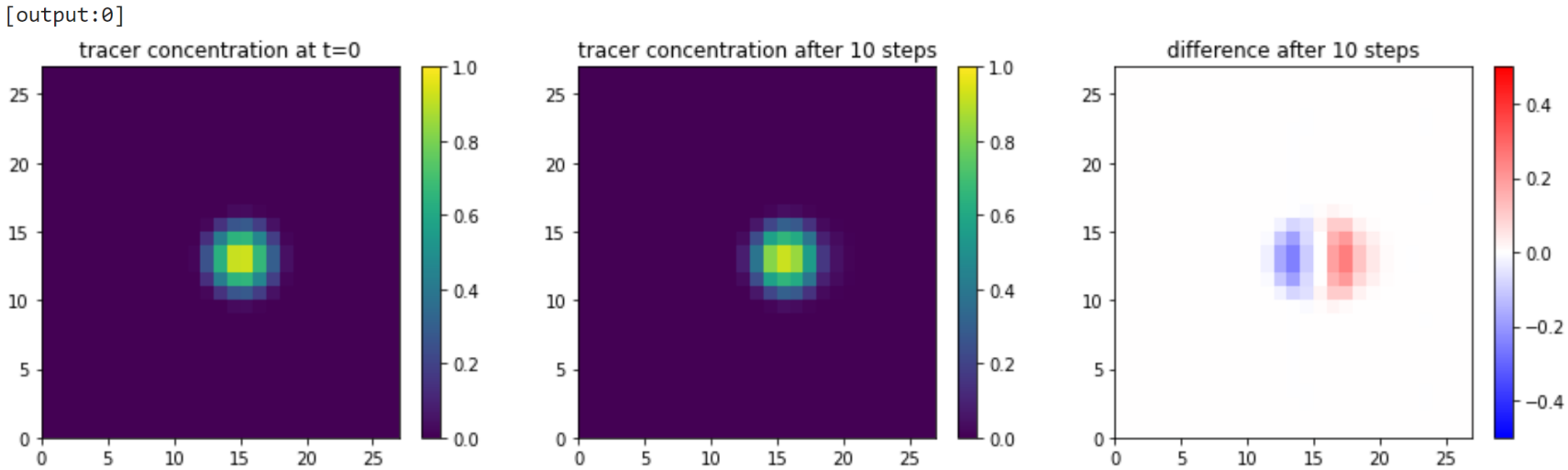
```python
class HyperdiffusionDamping:
  # ...
  def __call__(self, qdel: FloatField, cd: float):
    # ...
    for n in range(self._ntimes):
      nt = self._ntimes - (n + 1)
      self._corner_fill(qdel, self._q)

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

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

del2cubed.py




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

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



# ...



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

# Invoke function
dycore_loop(state, dycore, T)

validate(state)

plot_on_map(state.x_wind)
```

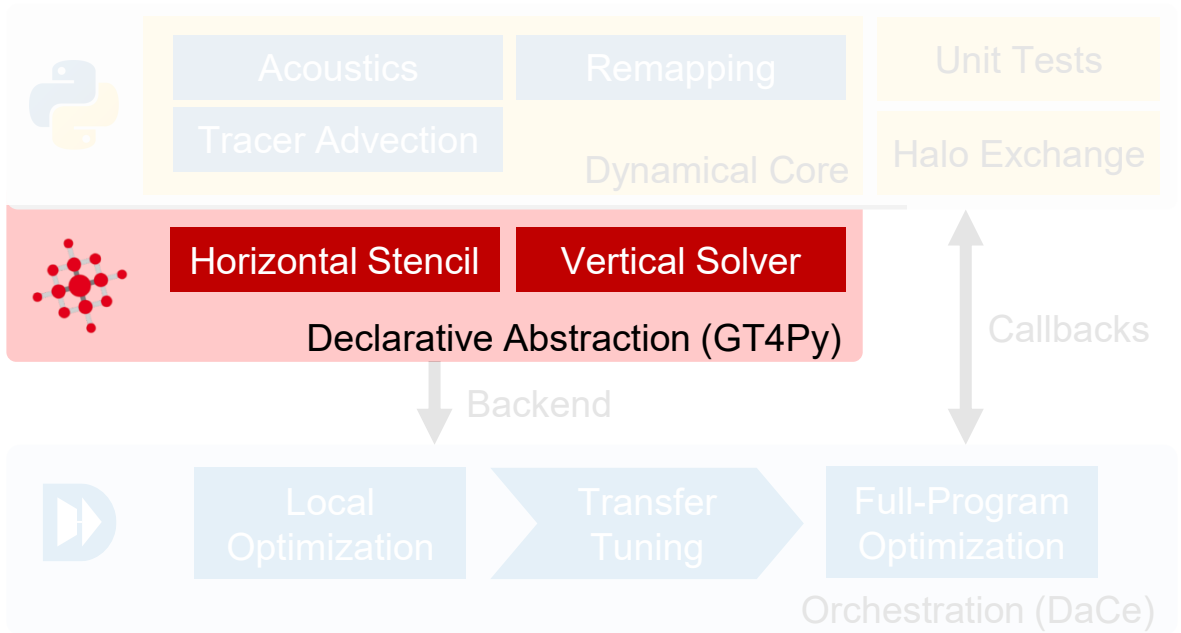
dynamics.py




```
plot_on_map(state.x_wind)
```

dynamics.py

https://github.com/ai2cm/pace/blob/main/examples/notebooks/stencil_definition.ipynb



Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

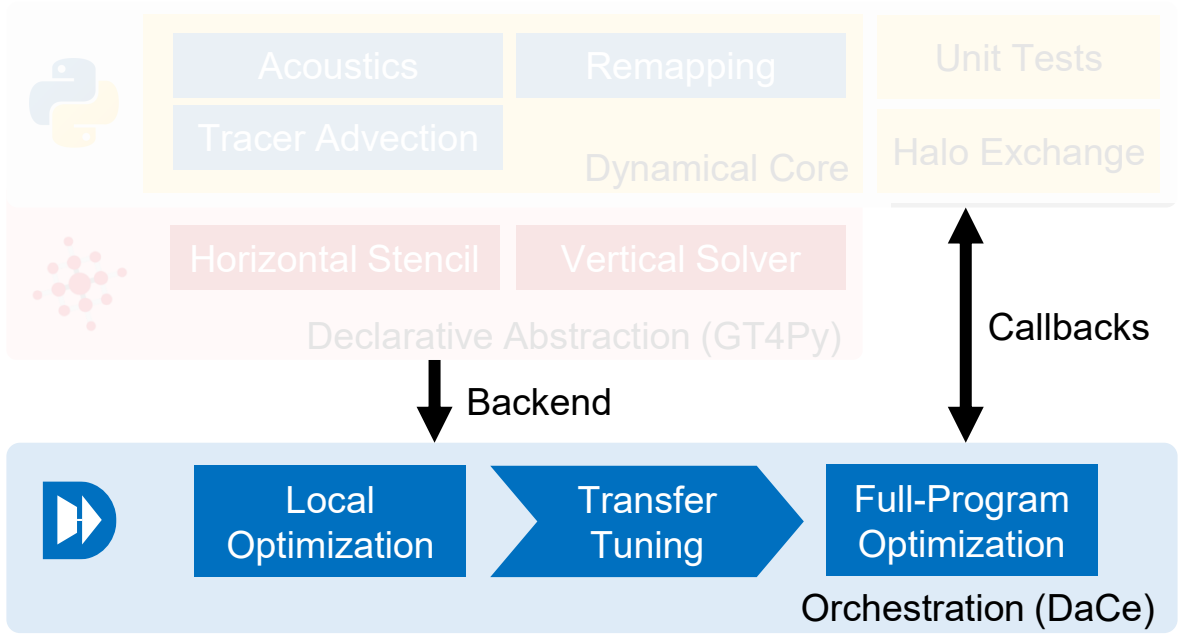
GridTools for Python (GT4Py)

https://github.com/GridTools/gt4py

- **Domain Specific Language (DSL) for Weather and Climate**
- **A declarative approach to define stencils (“what”, not “how”)**
  - 3D stencils and vertical solvers
- **Computation domain is abstracted**
  - Relative indexing
  - Automatic iteration ranges and halo regions
- **Implementation concerns are delegated to backends**
  - Execution schedules
  - Memory allocation
  - Target language

```
@gtscript.stencil(backend='dace:gpu')
def q_j_stencil(q: FloatField, area: FloatFieldIJ,
                x_area_flux: FloatField, fx2: FloatField,
                q_j: FloatField):

    with computation(PARALLEL), interval(...):

        fx1 = x_area_flux * fx2

        area_with_x_flux = area + x_area_flux - x_area_flux[1, 0, 0]

        q_j = (q * area + fx1 - fx1[1, 0, 0]) / area_with_x_flux
```

Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

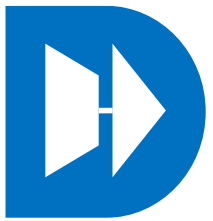


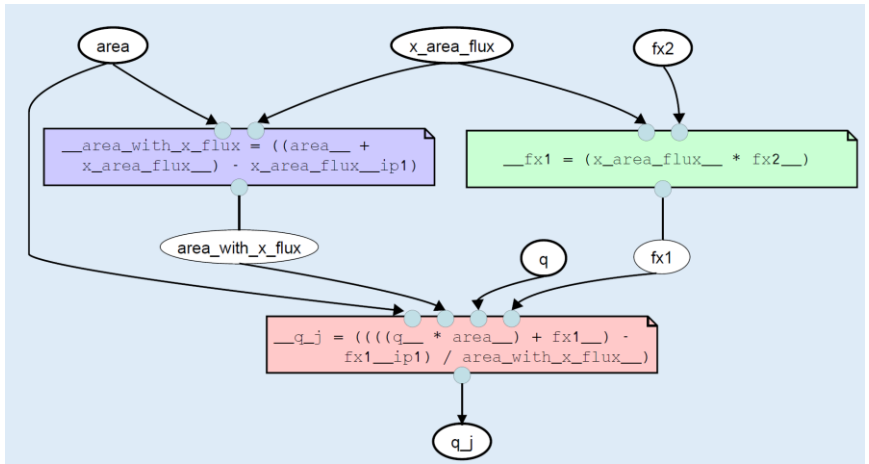
Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

```python
@gtscript.stencil(backend='dace:gpu')
def q_j_stencil(q: FloatField, area: FloatFieldIJ,
                x_area_flux: FloatField, fx2: FloatField,
                q_j: FloatField):
    with computation(PARALLEL), interval(...):
        fx1 = x_area_flux * fx2
        area_with_x_flux = area + x_area_flux - x_area_flux[1, 0, 0]
        q_j = (q * area + fx1 - fx1[1, 0, 0]) / area_with_x_flux
```

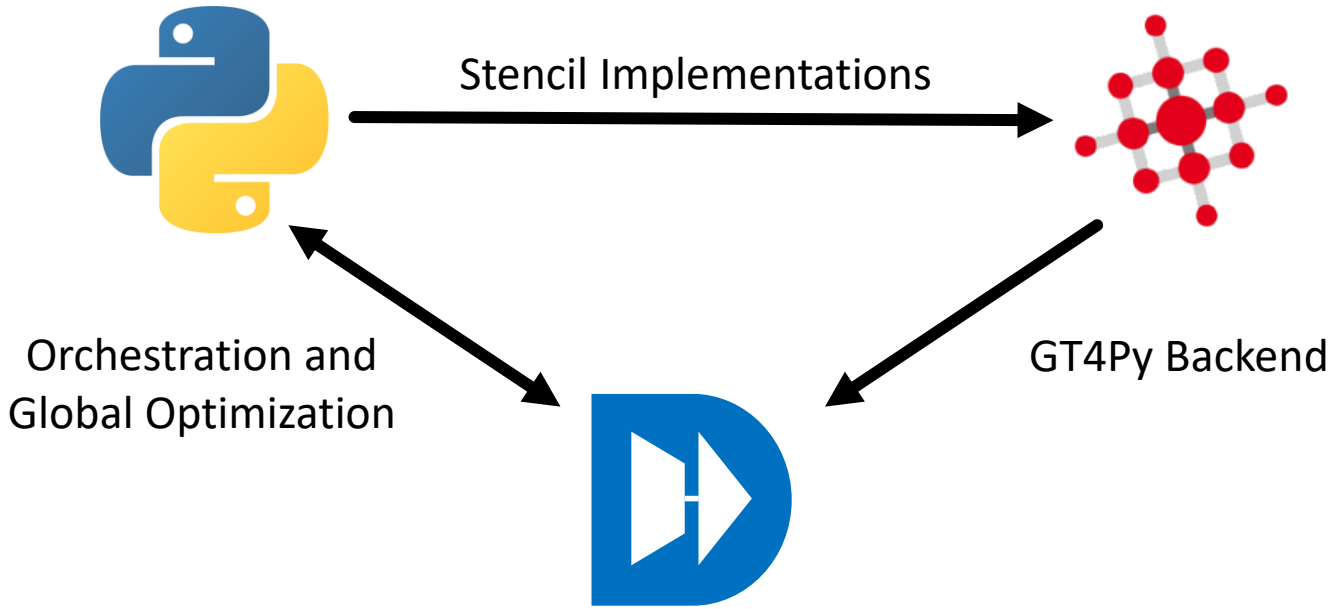
Stencil Implementations

GT4Py Backend





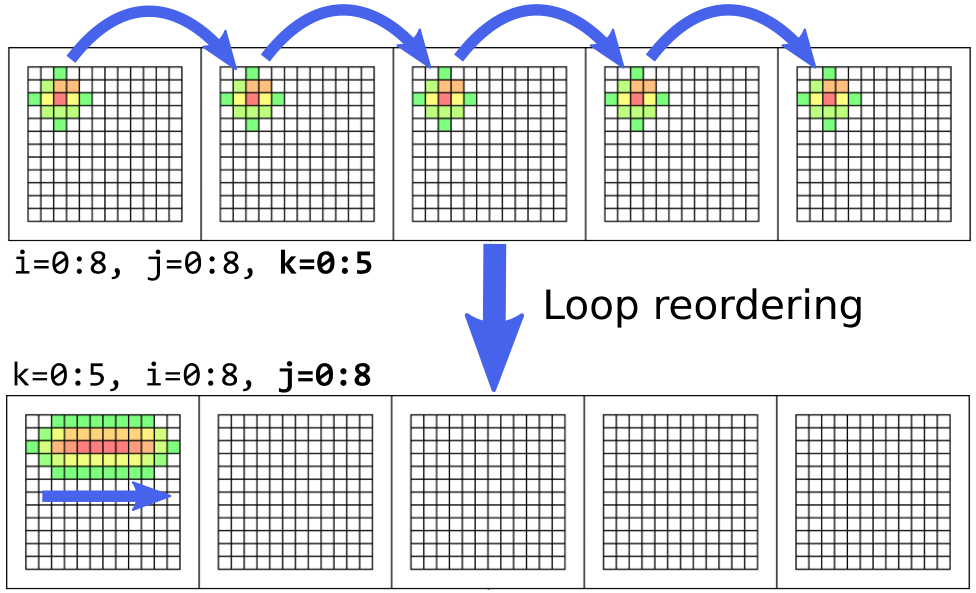
Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22


Stencil Implementations
Orchestration and
Global Optimization
GT4Py Backend

Characterizing the optimization space

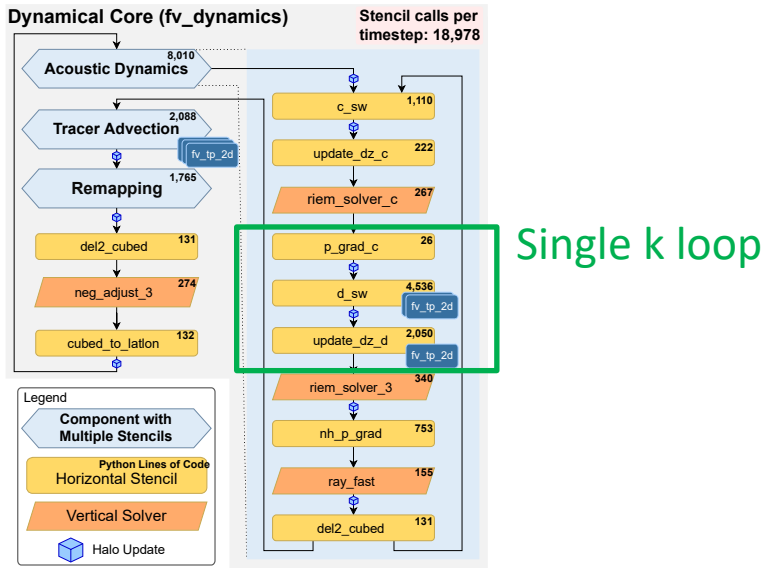
Within each stencil

- Computational layout
- Data layout
- Other rescheduling passes in GT4Py (e.g., branch → predication)

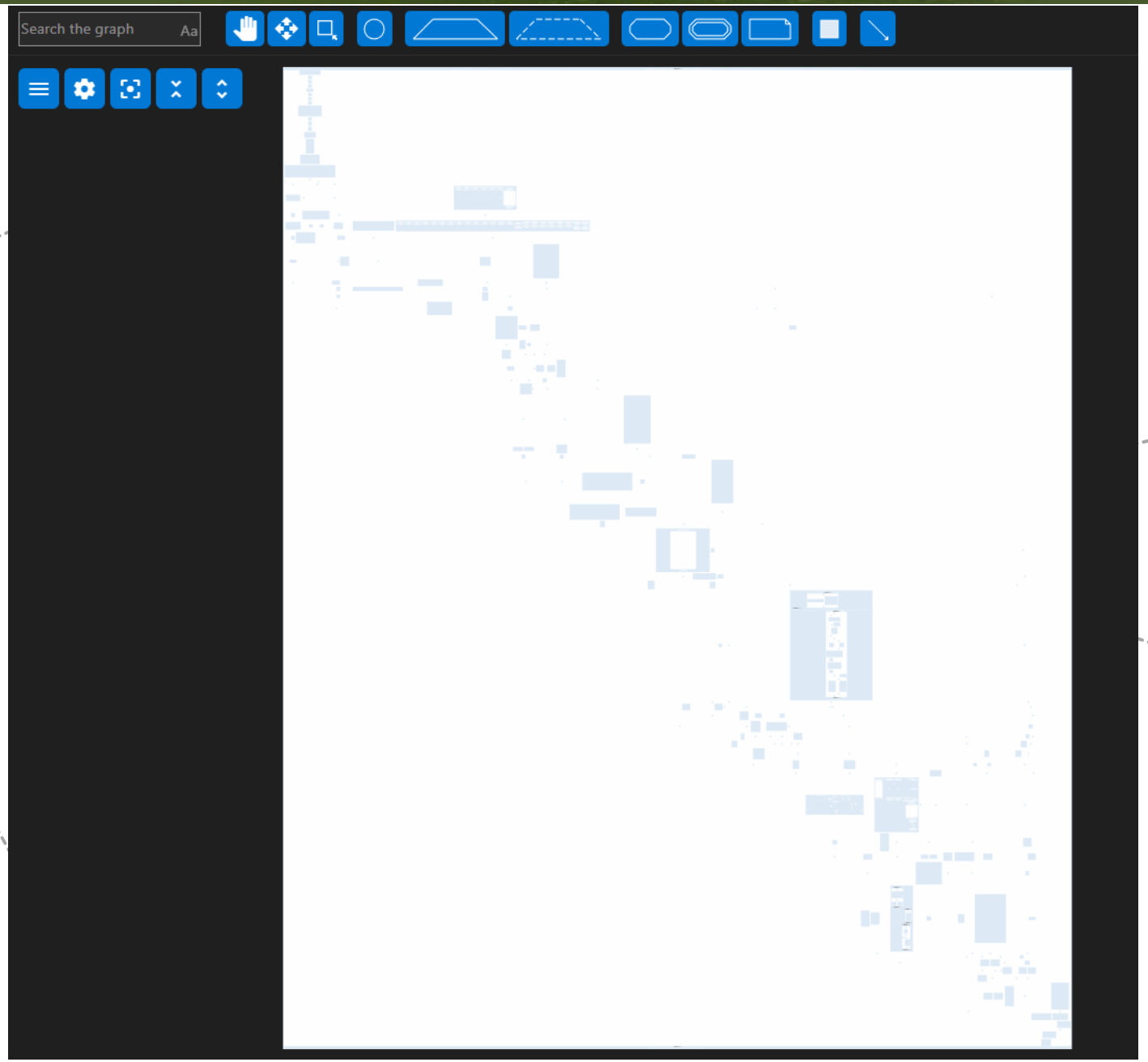


Between stencils

- Fusion
- Macro scheduling
- Pre-allocation (memory pool, static)
- Data layout "path"

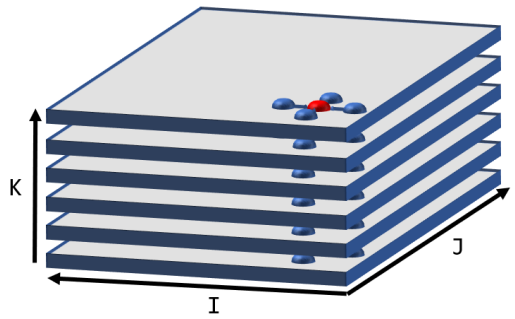


Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

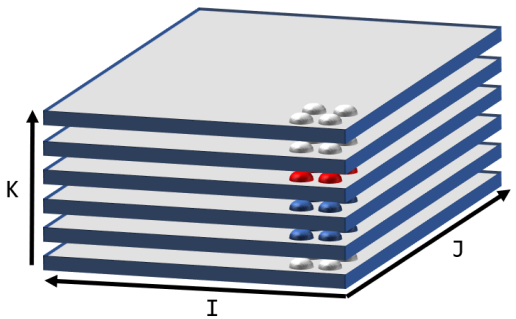


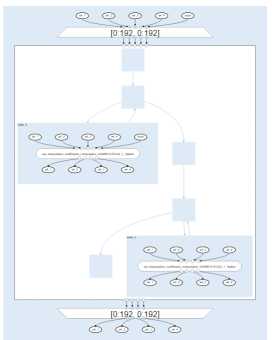
Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

Initial Heuristics



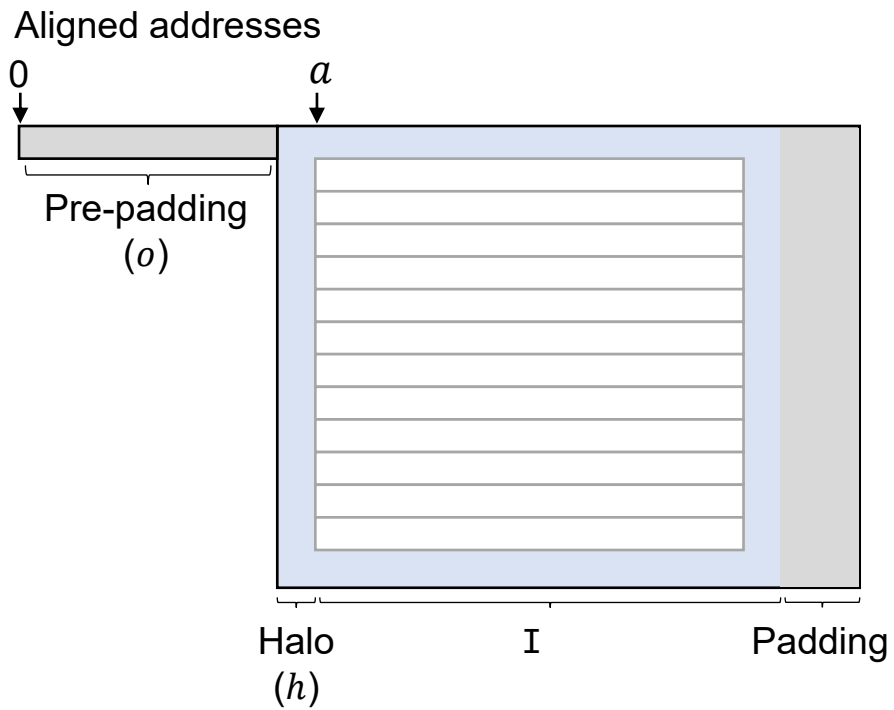
`Interval, Operation, K, J, I`





`J, I, Interval, Operation, K`

Initial Heuristics



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

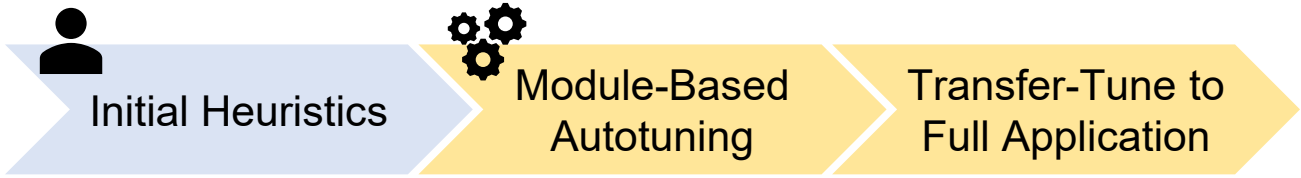
Start offset: $o = a - h$

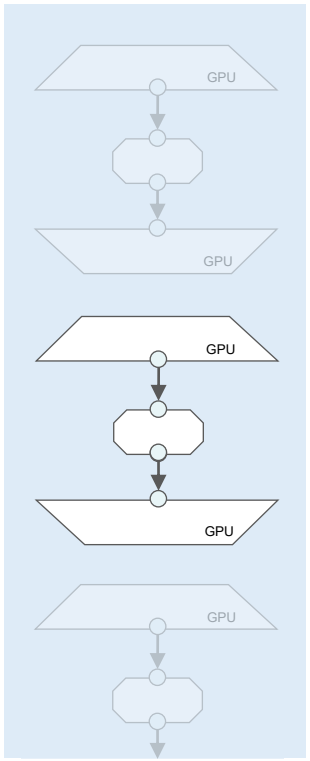
Strides:

$$s_i = 1$$

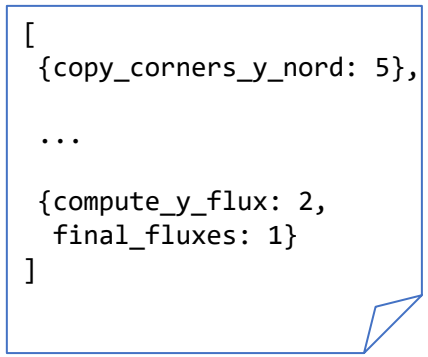
$$s_j = a\left\lceil \frac{I + 2h}{a} \right\rceil$$

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

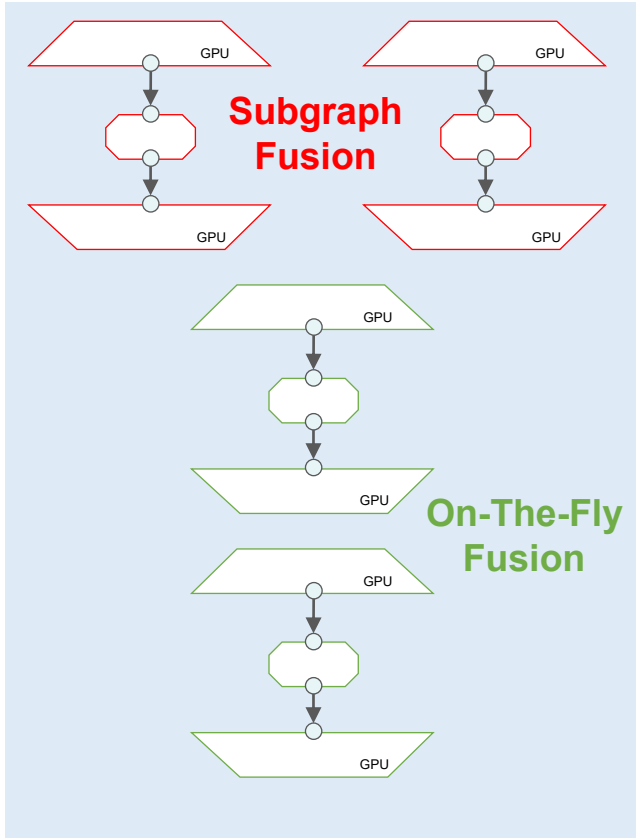




Exhaustive tuning on graph cutouts



Store top-k patterns



Test and apply on full program

2:42 hours on Piz Daint

8:24 hours

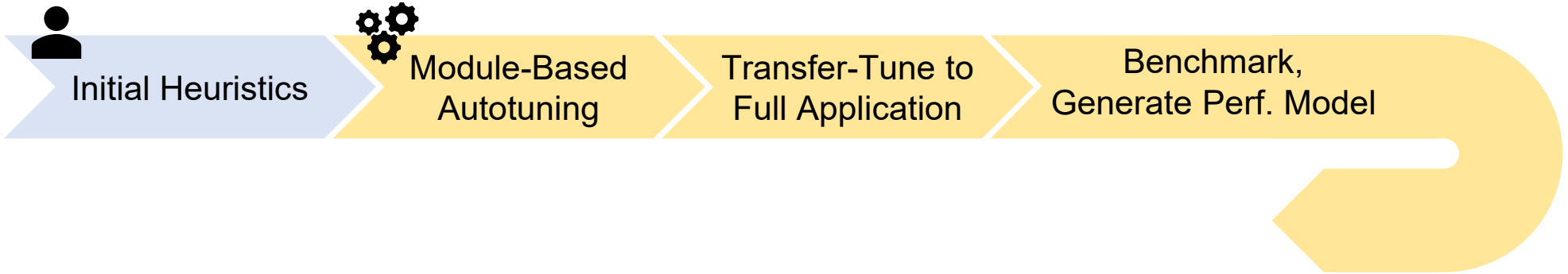
Without transfer tuning:

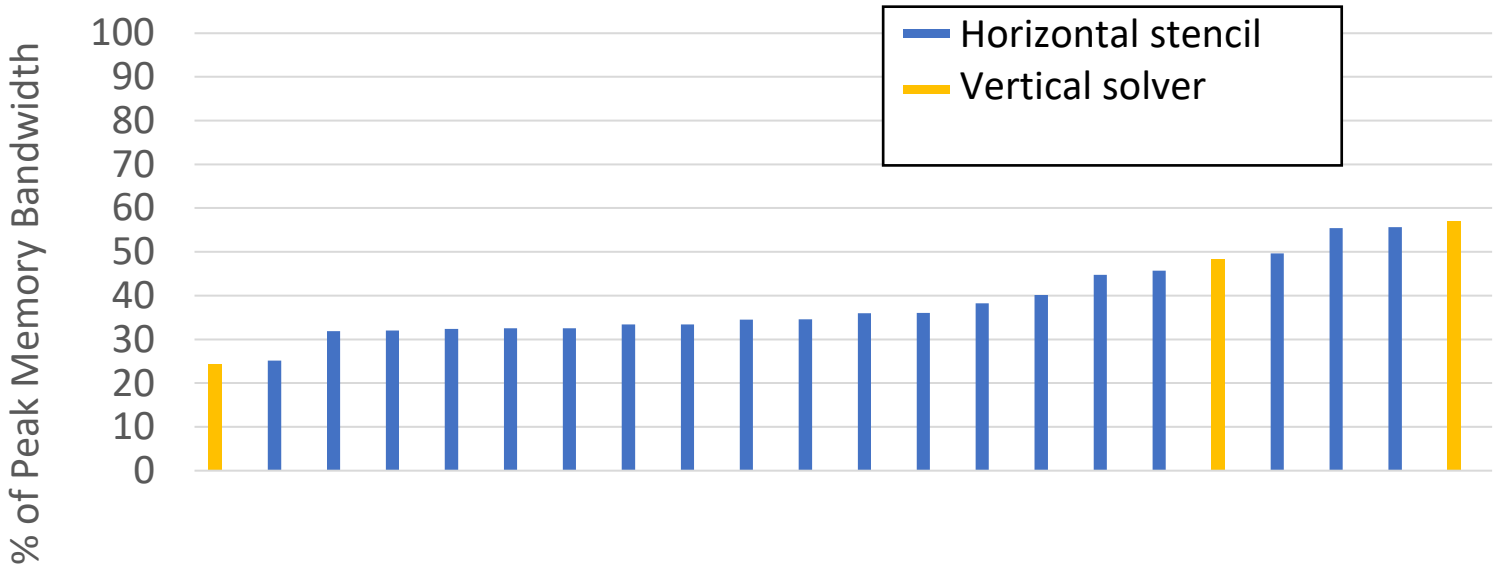
≥30,302,185 configurations

With transfer tuning:

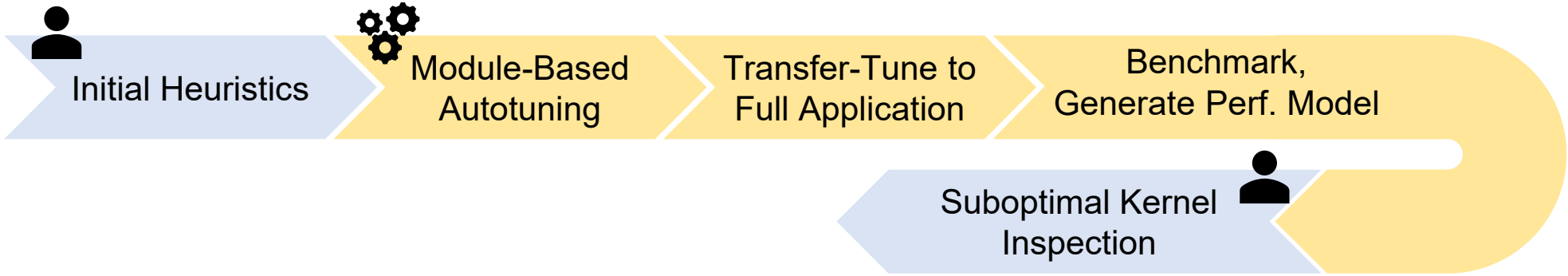
603

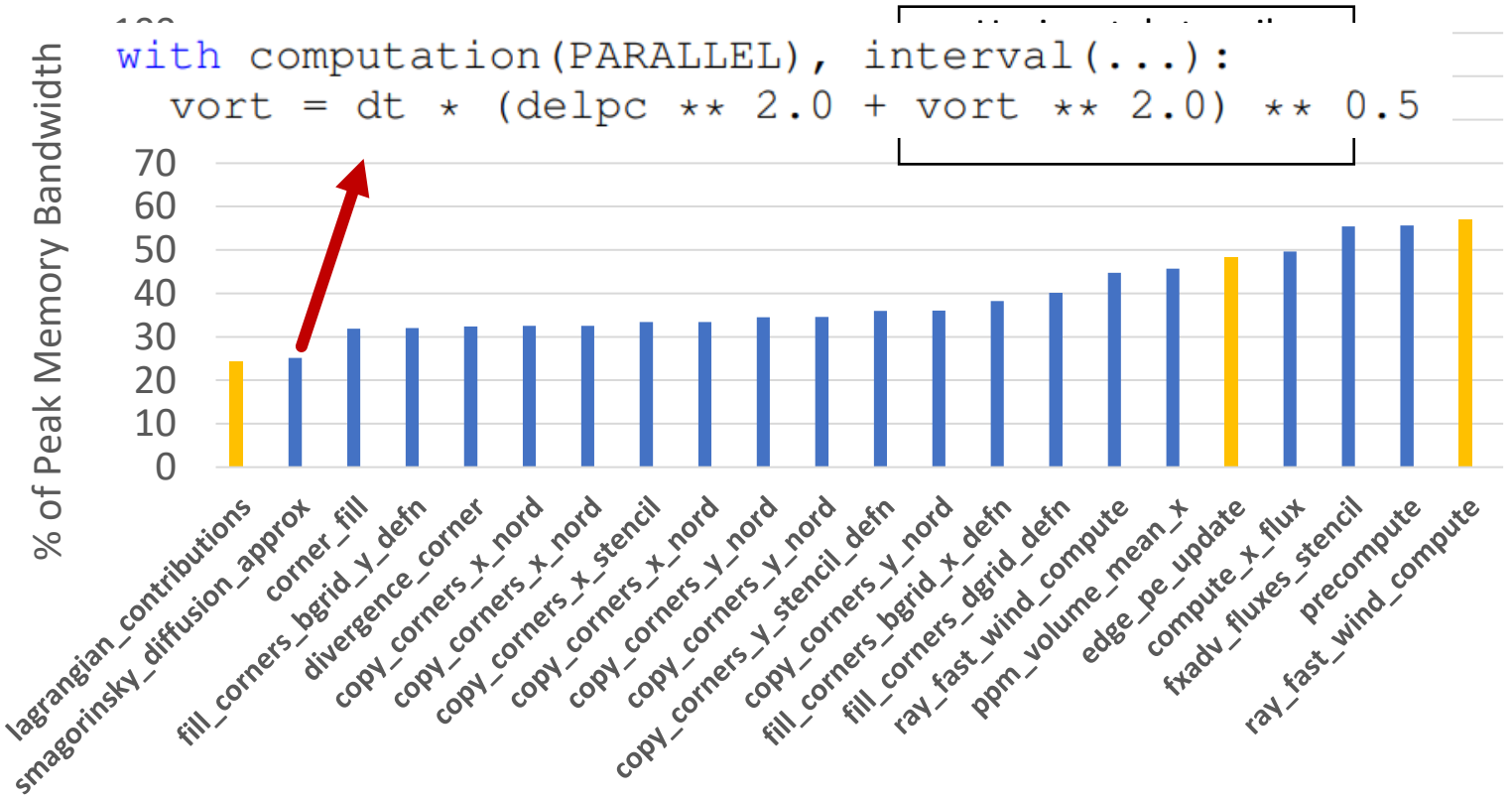
Truemper et al.: Performance Embeddings: A Similarity-based Transfer Tuning Approach to Performance Optimization, ICS'23



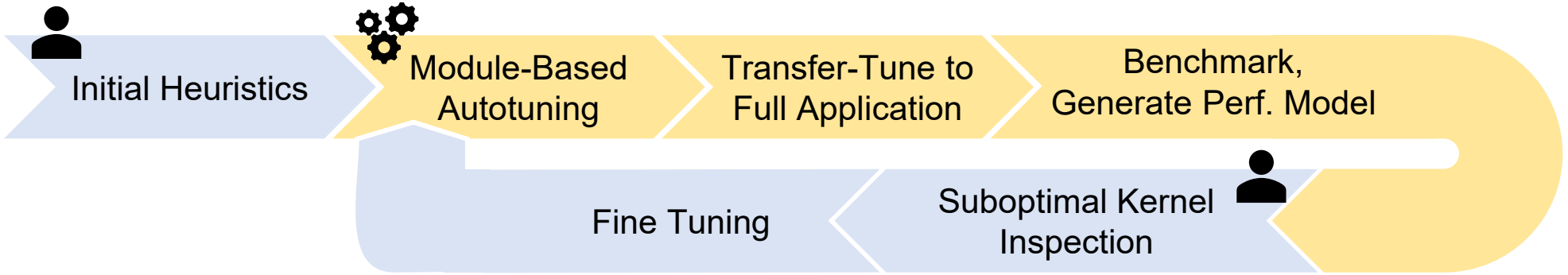


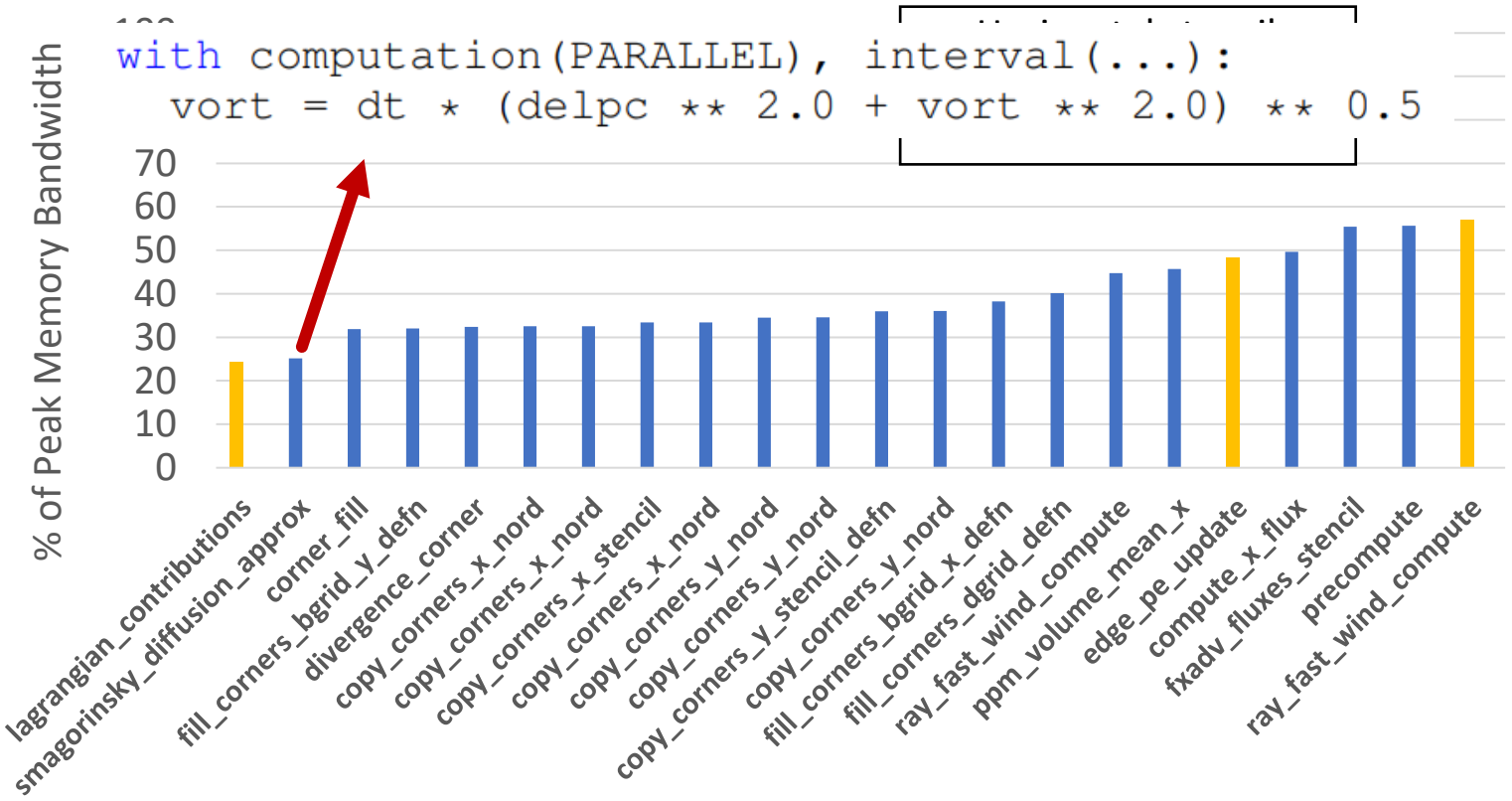
Truemper et al.: Performance Embeddings: A Similarity-based Transfer Tuning Approach to Performance Optimization, ICS'23



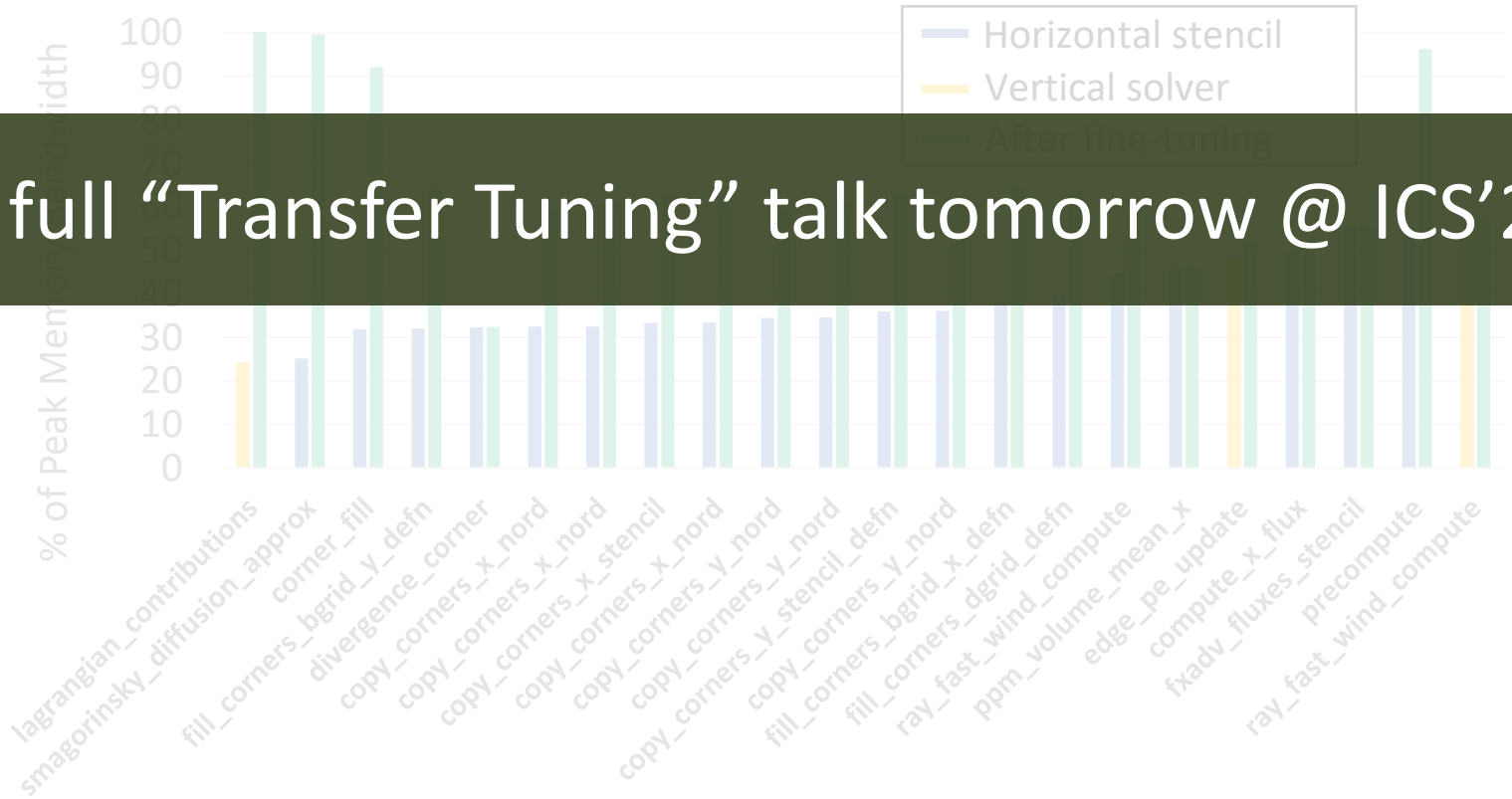


Truemper et al.: Performance Embeddings: A Similarity-based Transfer Tuning Approach to Performance Optimization, ICS'23





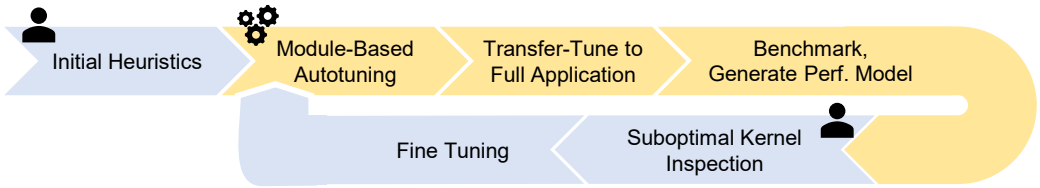
Truemper et al.: Performance Embeddings: A Similarity-based Transfer Tuning Approach to Performance Optimization, ICS'23



Watch the full “Transfer Tuning” talk tomorrow @ ICS’23, 4:20pm!

Truemper et al.: Performance Embeddings: A Similarity-based Transfer Tuning Approach to Performance Optimization, ICS’23

Evaluated Systems



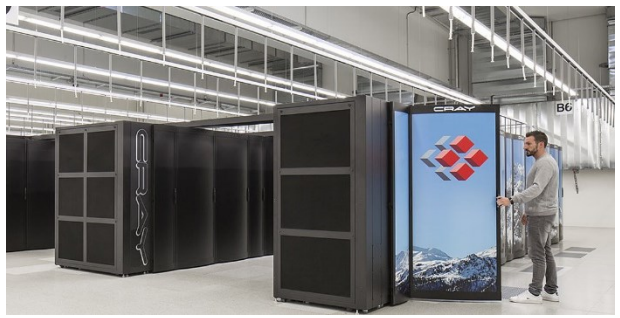


Photo courtesy of the Swiss National Supercomputing Centre

Photo courtesy of Forschungszentrum Jülich

Piz Daint:

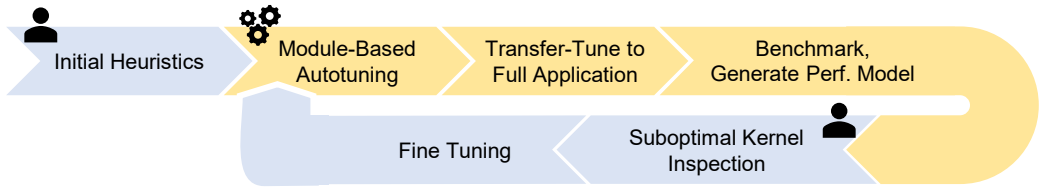
- GPU: 1 x NVIDIA Tesla P100 / Node
- CPU: Intel Xeon E5-2690 v3 (12 cores)

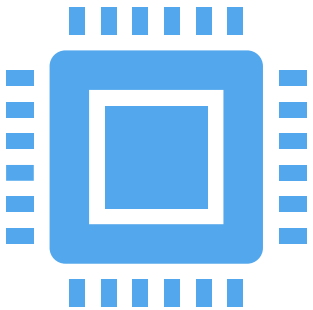
JUWELS Booster:

- GPU: 4 x NVIDIA Tesla A100 / Node
- CPU: AMD EPYC 7402 (2 sockets, 24 cores)

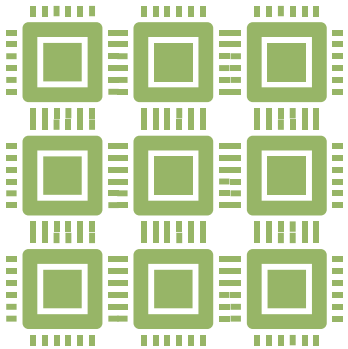
Domain size: 192x192x80

Memory Bounds





43.77 GB/s



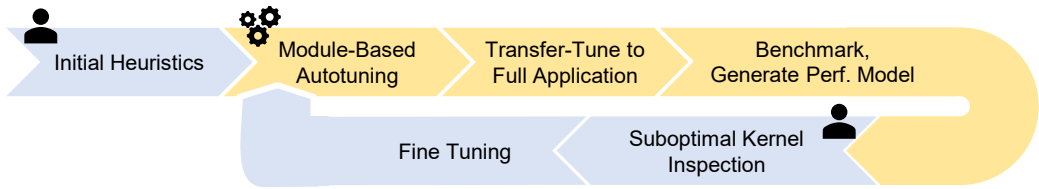
501.1 GB/s

Potential Speedup $\leq$ **11.45x**

Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

Representative Vertical Solver

Riemann Solver (`riem_solver_c`)



Semi-implicit solver for nonhydrostatic terms of vertical velocity and pressure perturbation

CPU

GPU

	FORTRAN		GT4Py+DaCe		
Domain Size (relative size)	Time [ms]	Scaling	Time [ms]	Scaling	Speedup
128×128×80 (1x)	12.27	—	1.85	—	6.63×
192×192×80 (2.25x)	27.94	2.28	3.86	2.08	7.25×
256×256×80 (4x)	52.40	4.27	6.96	3.76	7.53×
384×384×80 (9x)	121.80	9.92	15.31	8.26	7.96×

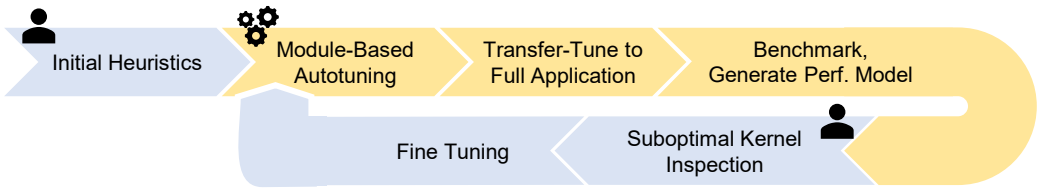
CPU cache runs out, data layout not ideal

Not enough parallelism

Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

Representative Horizontal Stencil

Finite Volume Transport (`fv_tp_2d`)



FORTRAN runs on a **single slice**, GT4Py/DaCe runs on entire 3D domain

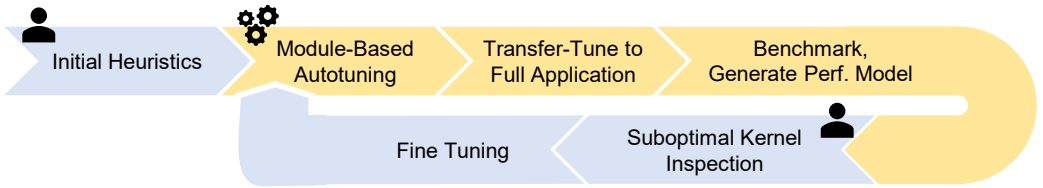
	FORTRAN		GT4Py+DaCe		
Domain Size (relative size)	Time [ms]	Scaling	Time [ms]	Scaling	Speedup
128×128×80 (1x)	3.41	—	1.81	—	1.88×
192×192×80 (2.25x)	12.31	3.61	3.41	1.88	3.61×
256×256×80 (4x)	35.79	10.49	5.67	3.13	6.31×
384×384×80 (9x)	106.66	31.27	13.10	7.23	8.14×

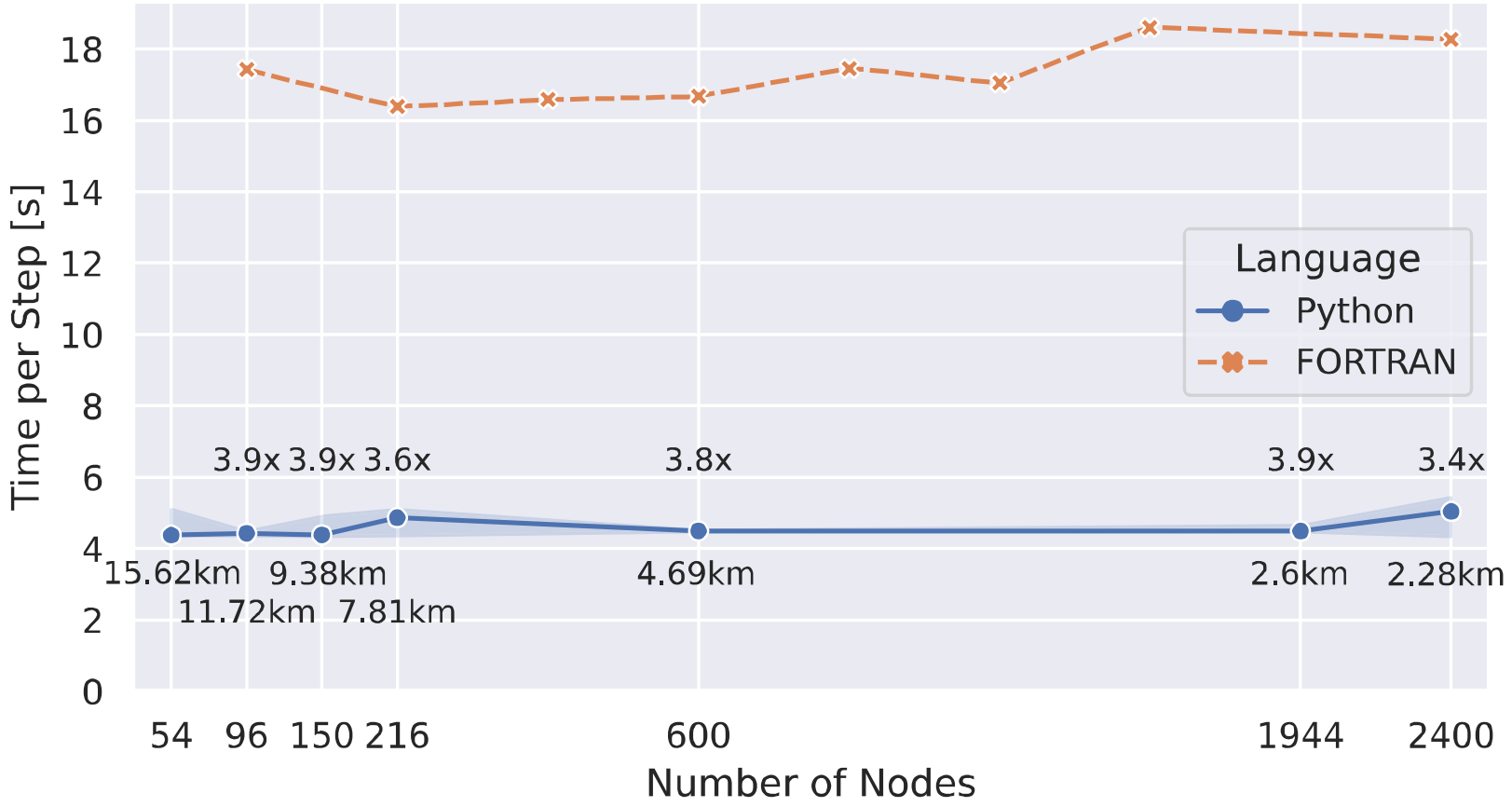
0.13% of load/stores are L3 misses

Closing gap to ideal memory bandwidth factor

Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

Weak Scaling





Simulation throughput of **0.12 SYPD** at 2.6 km grid spacing

FV3 Summary:

6 weeks of work

10 optimization revisions

4 performance engineers

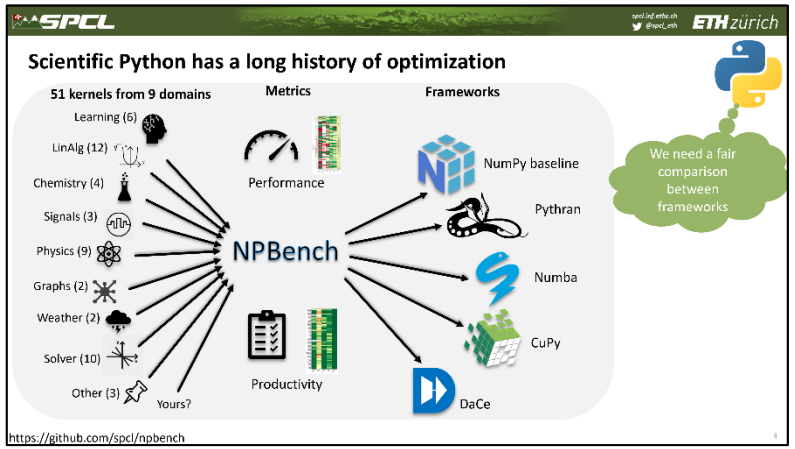
3.92 – 8.48x speedup vs. FORTRAN

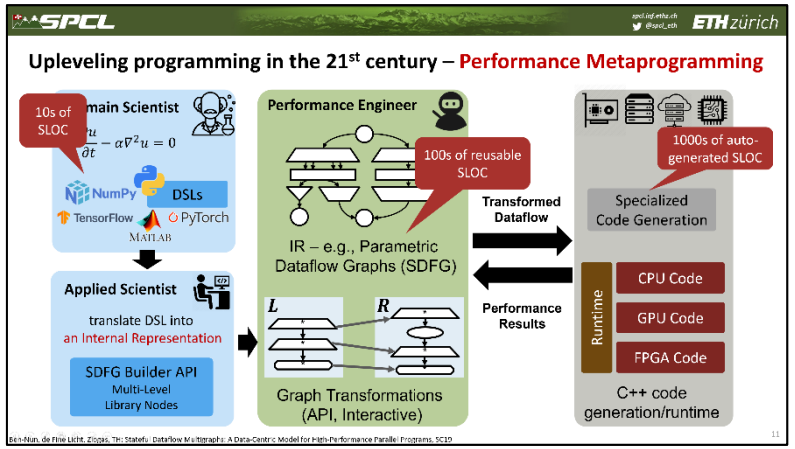
0 model changes

Ben-Nun et al.: Productive Performance Engineering for Weather and Climate Modeling with Python, SC22

Key Points and Conclusions

Fortran to DaCe coming soon!





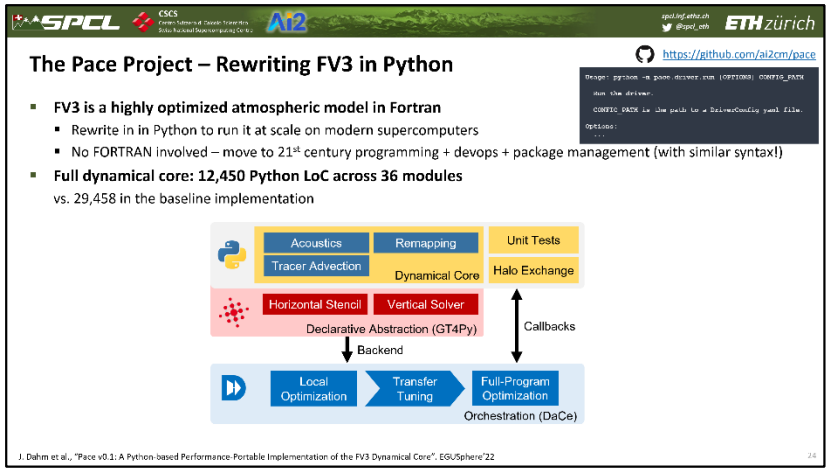
More of SPCL's research:

youtube.com/@spcl — 170+ Talks

twitter.com/spcl_eth — 1.3K+ Followers

github.com/spcl — 2K+ Stars

… or spcl.ethz.ch



Spatial Devices are the Future!

More on Mapping with some Theory

Friday, 10am @ HPDC'23

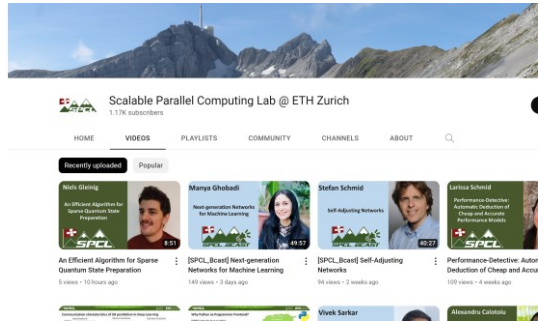
De Matteis et al.: "Streaming Task Graph Scheduling for Dataflow Architectures"

Want to join our efforts?
We're looking for excellent
Postdocs, PhD students, and Visitors.
Talk to me!

https://github.com/ai2cm/pace
https://github.com/GridTools/gt4py
https://github.com/spcl/dace

Want to know more?

youtube.com/@spcl



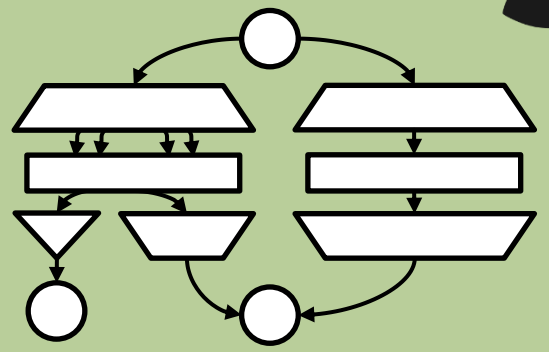
twitter.com/spcl_eth

spcl.inf.ethz.ch

github.com/spcl

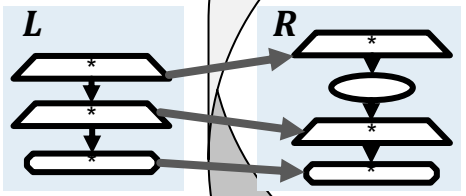
Generating hardware descriptions

Performance Engineer



Parametric Dataflow Graphs (SDFG)

Graph Transformations



Stateful DataFlow multiGraph (SDFG)

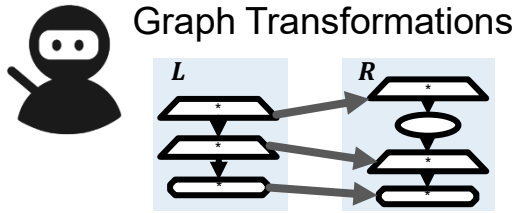
Code generated for CPU, GPU, FPGA.

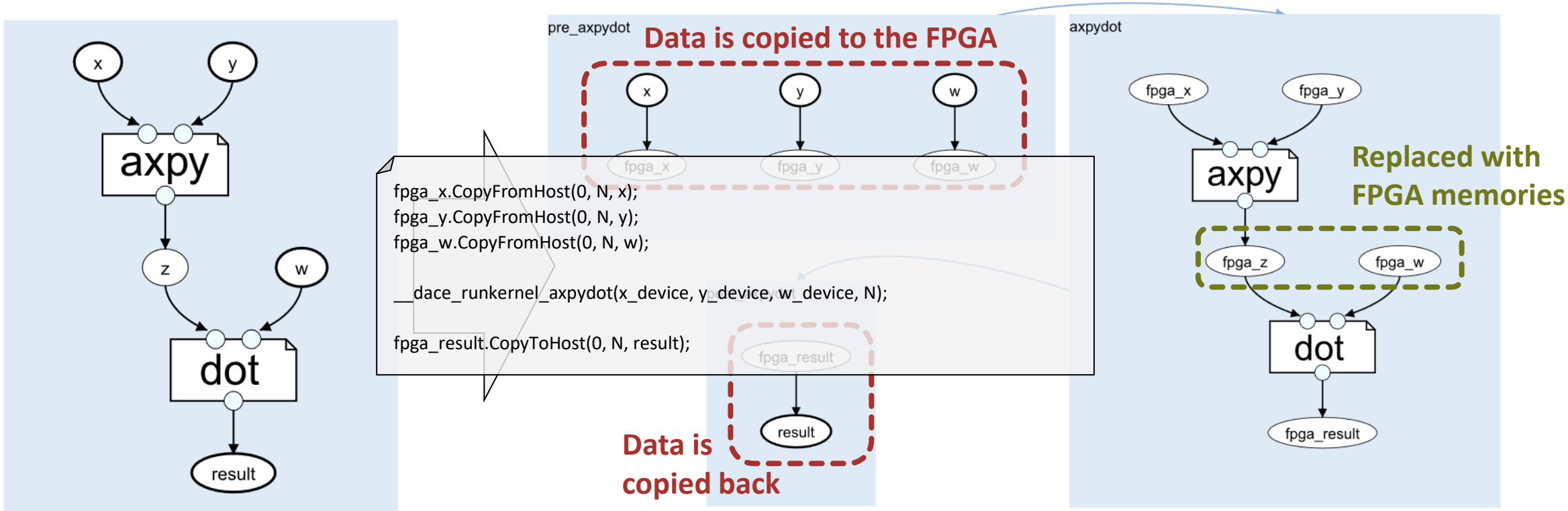
```
for (int n0 = 0; n0 < (N / P); n0 += 1) {
  dace::vec<float, 4> C_buffer[(M / 4)];
  for (int k = 0; k < K; k += 1) {
    float A_reg;
    for (int n1 = 0; n1 < P; n1 += 1) {
      #pragma HLS PIPELINE II=1
      #pragma HLS LOOP_FLATTEN
      {
        float a_in = (A_pipe[p]).pop();

        ////////////////////
        // Tasklet code (buffer_a)
        if ((n1 == ((P - p) - 1))) {
          A_reg = a_in;
        }
        if ((p < (P - 1))) {
          A_pipe[(p + 1)].push(a_in);
        }
        ////////////////////
      }
    }
    for (int m = 0; m < (M / 4); m += 1) {
      #pragma HLS PIPELINE II=1
      #pragma HLS LOOP_FLATTEN
      {
        float a_in = A_reg;
        dace::vec<float, 4> b_in = (B_pipe[p]).pop();
        dace::vec<float, 4> c_in = C_buffer[m];
        ...
```

All data movement is explicit

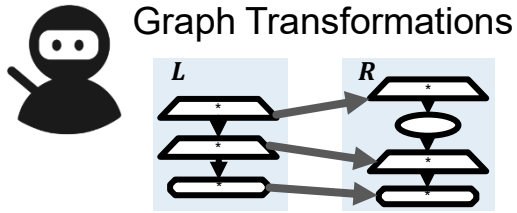
Pure dataflow sections.

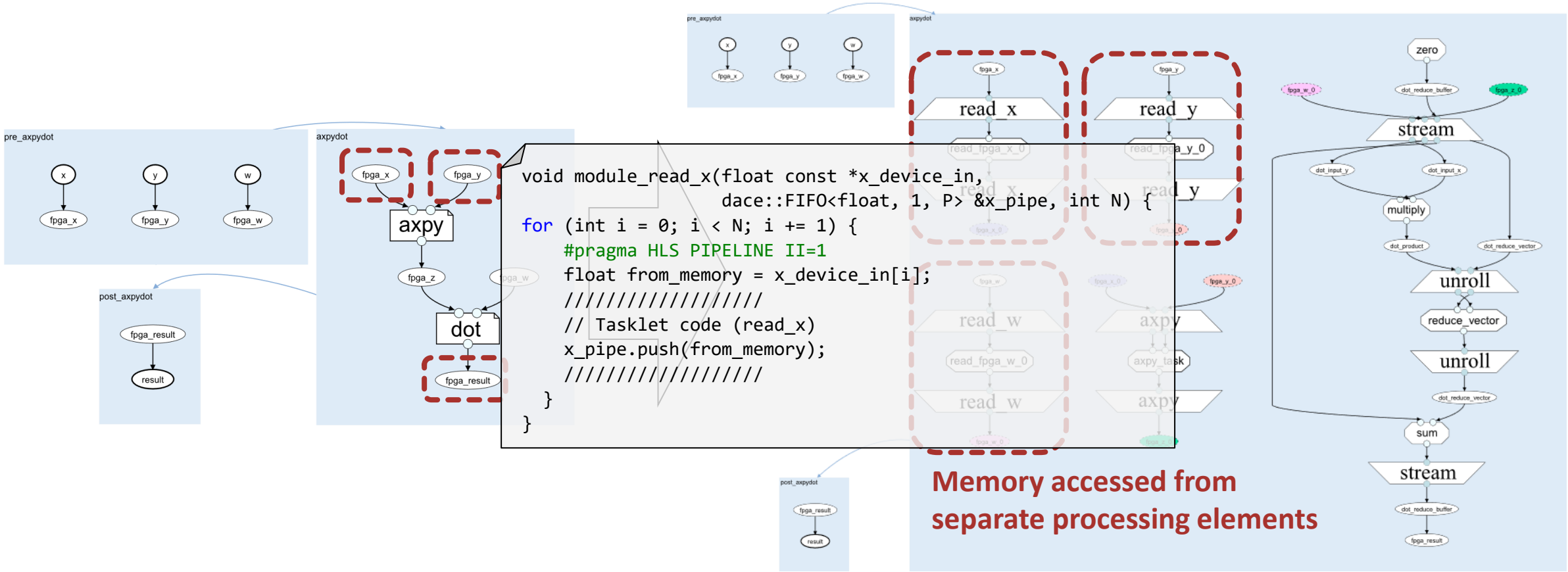
Example transformation #0: Offload to FPGA





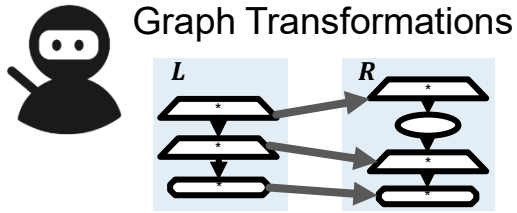
Example transformation #1: Stream memory accesses

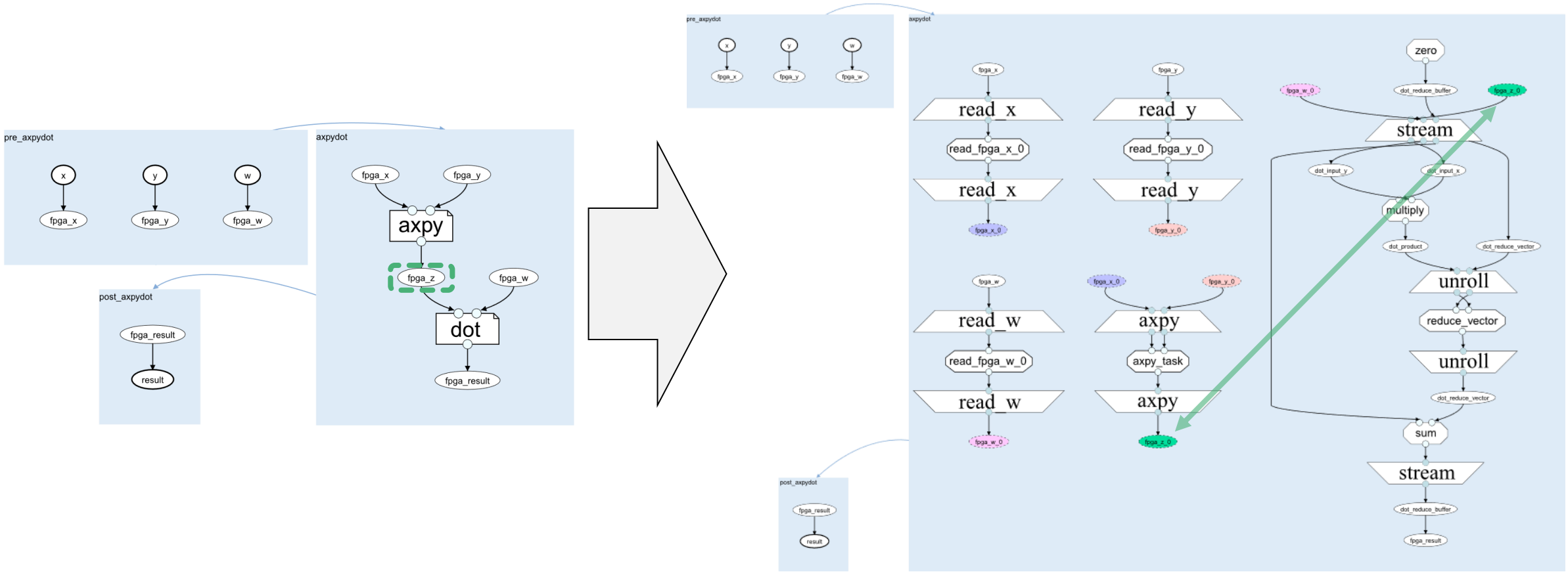




de Fine Licht et al.: Transformations of High-Level Synthesis Codes for High-Performance Computing IEEE Transactions on Parallel and Distributed Systems (TPDS). 2021

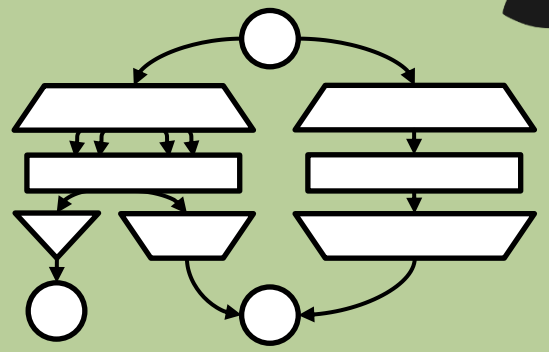
Example transformation #2: Stream between operators





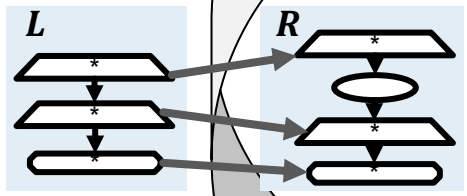
de Fine Licht et al.: Transformations of High-Level Synthesis Codes for High-Performance Computing IEEE Transactions on Parallel and Distributed Systems (TPDS). 2021

Performance Engineer



Parametric Dataflow Graphs (SDFG)

Graph Transformations



Stateful DataFlow multiGraph (SDFG)

Code generated for CPU, GPU, FPGA.

```
for (int n0 = 0; n0 < (N / P); n0 += 1) {
  dace::vec<float, 4> C_buffer[(M / 4)];
  for (int k = 0; k < K; k += 1) {
    float A_reg;
    for (int n1 = 0; n1 < P; n1 += 1) {
      #pragma HLS PIPELINE II=1
      #pragma HLS LOOP_FLATTEN
      {
        float a_in = (A_pipe[p]).pop();

        ////////////////////
        // Tasklet code (buffer_a)
        if ((n1 == ((P - p) - 1))) {
          A_reg = a_in;
        }
        if ((p < (P - 1))) {
          A_pipe[(p + 1)].push(a_in);
        }
        ////////////////////
      }
    }
    for (int m = 0; m < (M / 4); m += 1) {
      #pragma HLS PIPELINE II=1
      #pragma HLS LOOP_FLATTEN
      {
        float a_in = A_reg;
        dace::vec<float, 4> b_in = (B_pipe[p]).pop();
        dace::vec<float, 4> c_in = C_buffer[m];
        ...
```

de Fine Licht et al.: Transformations of High-Level Synthesis Codes for High-Performance Computing IEEE Transactions on Parallel and Distributed Systems (TPDS). 2021

Cross-vendor support

Code generation

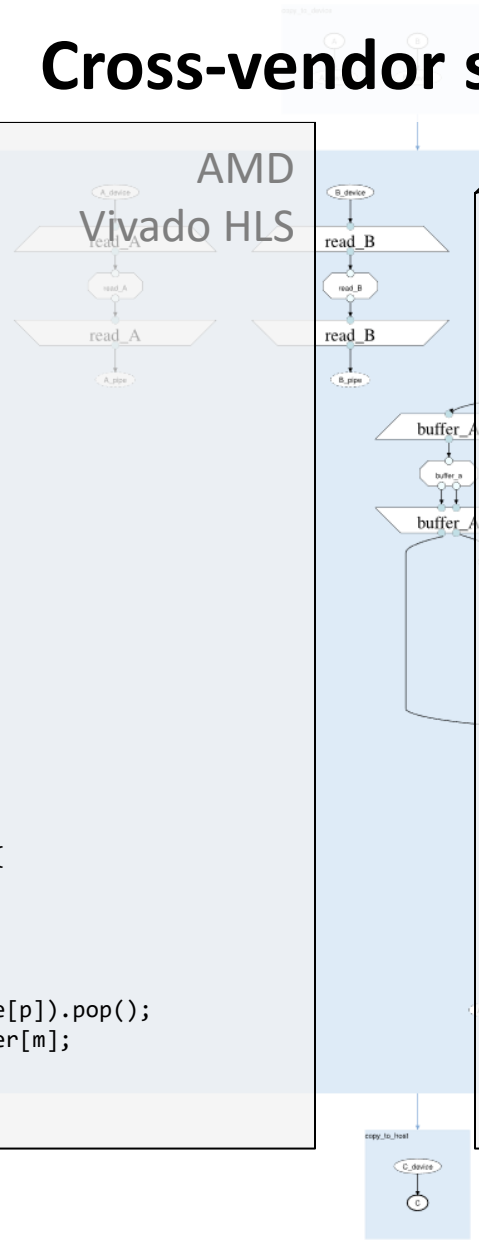
...

AMD Vivado HLS

```
for (int n0 = 0; n0 < (N / P); n0 += 1) {
  dace::vec<float, 4> C_buffer[(M / 4)];
  for (int k = 0; k < K; k += 1) {
    float A_reg;
    for (int n1 = 0; n1 < P; n1 += 1) {
      #pragma HLS PIPELINE II=1
      #pragma HLS LOOP_FLATTEN
      {
        float a_in = (A_pipe[p]).pop();

        ////////////////////
        // Tasklet code (buffer_a)
        if ((n1 == ((P - p) - 1))) {
          A_reg = a_in;
        }
        if ((p < (P - 1))) {
          A_pipe[(p + 1)].push(a_in);
        }
        ////////////////////
      }
    }
    for (int m = 0; m < (M / 4); m += 1) {
      #pragma HLS PIPELINE II=1
      #pragma HLS LOOP_FLATTEN
      {
        float a_in = A_reg;
        dace::vec<float, 4> b_in = (B_pipe[p]).pop();
        dace::vec<float, 4> c_in = C_buffer[m];
        ...
```

Intel OpenCL

```
for (int n0 = 0; n0 < (N / P); n0 += 1) {
  float4 C_buffer[(M / 4)];
  #pragma ivdep
  for (int k = 0; k < K; k += 1) {
    float A_reg;
    #pragma loop_coalesce
    for (int n1 = 0; n1 < P; n1 += 1) {
      float a_in = read_channel_intel(A_pipe[p]);
      float *a_reg = &A_reg;
      #define a_out A_pipe[(p + 1)]  // God save us

      ////////////////////
      // Tasklet code (buffer_a)
      if ((n1 == ((P - p) - 1))) {
        *a_reg = a_in;
      }
      if ((p < (P - 1))) {
        write_channel_intel(a_out, a_in);
      }
      ////////////////////

      #undef a_out
    }
    #pragma loop_coalesce
    #pragma ivdep
    for (int m = 0; m < (M / 4); m += 1) {
      float a_in = A_reg;
      float4 b_in = read_channel_intel(B_pipe[p]);
      float4 c_in = C_buffer[m];
      ...
```



de Fine Licht et al.: Transformations of High-Level Synthesis Codes for High-Performance Computing IEEE Transactions on Parallel and Distributed Systems (TPDS). 2021

Code generation

...

Boilerplate code elimination

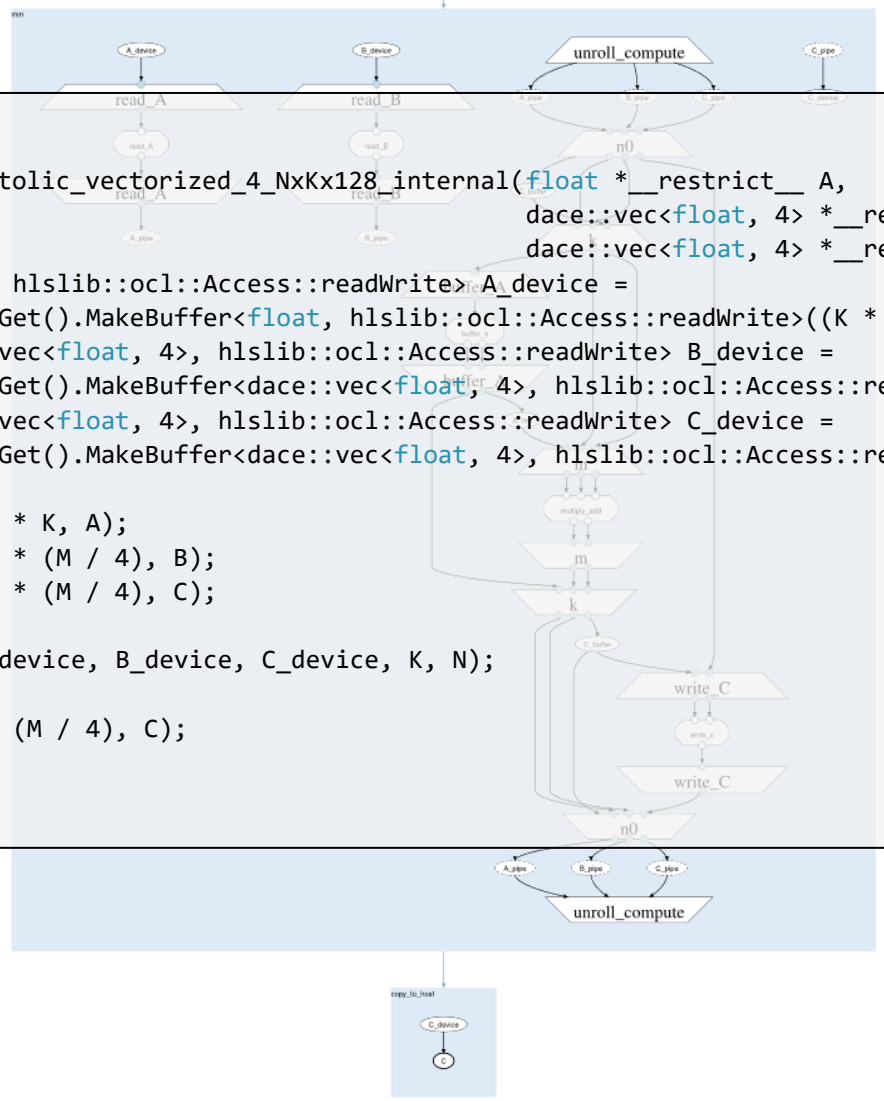
```
dace::fpga::Context *dace::fpga::_context;

int __dace_init_intel_fpga(float *__restrict__ A, dace::vec<float, 4> *__restrict__ B,
                           dace::vec<float, 4> *__restrict__ C, int K, int N) {
  dace::fpga::_context = new dace::fpga::Context();
  dace::fpga::_context->Get().MakeProgram(
      DACE_BINARY_DIR "/gemm_fpga_systolic_vectorized_4_NxKx128.aocx");
  return 0;
}

void __dace_runkernel_gemm_0(hlslib::ocl::Buffer<float, hlslib::ocl::Access::readWrite> &A_device,
                             hlslib::ocl::Buffer<dace::vec<float, 4>, hlslib::ocl::Access::readWrite> &B_device,
                             hlslib::ocl::Buffer<dace::vec<float, 4>, hlslib::ocl::Access::readWrite> &C_device, int K,
                             int N) {
  hlslib::ocl::Program program = dace::fpga::_context->Get().CurrentlyLoadedProgram();
  std::vector<hlslib::ocl::Kernel> kernels;
  kernels.emplace_back(program.MakeKernel("mod_0_read_A", A_device, K, N));
  kernels.emplace_back(program.MakeKernel("mod_0_read_B", B_device, K, N));
  kernels.emplace_back(program.MakeKernel("mod_0_gemm_0", K, N));
  kernels.emplace_back(program.MakeKernel("mod_0_gemm_1", K, N));
  kernels.emplace_back(program.MakeKernel("mod_0_gemm_2", K, N));
  kernels.emplace_back(program.MakeKernel("mod_0_gemm_3", K, N));
  kernels.emplace_back(program.MakeKernel("mod_0_write_C", C_device, K, N));
  std::vector<cl::Event> events;
  for (auto &k : kernels) {
    events.emplace_back(k.ExecuteTaskFork());
  }
  cl::Event::waitForEvents(events);
}
```

Code generation

...

Host/device interaction

```
void __program_gemm_fpga_systolic_vectorized_4_NxKx128_internal(float *__restrict__ A,
                                                                dace::vec<float, 4> *__restrict__ B,
                                                                dace::vec<float, 4> *__restrict__ C, int K, int N) {

  hlslib::ocl::Buffer<float, hlslib::ocl::Access::readWrite> A_device =
      dace::fpga::_context->Get().MakeBuffer<float, hlslib::ocl::Access::readWrite>((K * N));
  hlslib::ocl::Buffer<dace::vec<float, 4>, hlslib::ocl::Access::readWrite> B_device =
      dace::fpga::_context->Get().MakeBuffer<dace::vec<float, 4>, hlslib::ocl::Access::readWrite>(((K * M) / 4));
  hlslib::ocl::Buffer<dace::vec<float, 4>, hlslib::ocl::Access::readWrite> C_device =
      dace::fpga::_context->Get().MakeBuffer<dace::vec<float, 4>, hlslib::ocl::Access::readWrite>(((M * N) / 4));

  A_device.CopyFromHost(0, N * K, A);
  B_device.CopyFromHost(0, K * (M / 4), B);
  C_device.CopyFromHost(0, N * (M / 4), C);

  __dace_runkernel_gemm_0(A_device, B_device, C_device, K, N);

  C_device.CopyToHost(0, N * (M / 4), C);
}
```



Code generation

...

```python
N = dace.symbol("N")
K = dace.symbol("K")
M = dace.symbol("M")
P = dace.symbol("P")
W = dace.symbol("W")

def make_sdfg():
    # ...do stuff...
    return sdfg

if __name__ == "__main__":
    # Matrix of size 1024x1024x1024
    K.set(1024)
    N.set(1024)

    gemm = make_sdfg()

    # Initialize arrays: Randomize A and B, zero C
    A = np.ndarray([N.get(), K.get()], dtype=np.float32)
    B = np.ndarray([K.get(), M.get()], dtype=np.float32)
    C = np.ndarray([N.get(), M.get()], dtype=np.float32)
    A[:] = np.random.rand(N.get(), K.get()).astype(np.float32)
    B[:] = np.random.rand(K.get(), M.get()).astype(np.float32)
    C[:] = np.random.rand(N.get(), M.get()).astype(np.float32)

    gemm(A=A, B=B, C=C, N=N, K=K)  # M is fixed at compile-time
```

SDFGs defined or loaded using Python API

Interfaces with NumPy arrays

DaCe programs are exposed as Python functions

DaCe successfully runs 30/30 Polybench applications directly from NumPy (Xilinx + Intel)!

Case study: Weather simulation at MeteoSwiss

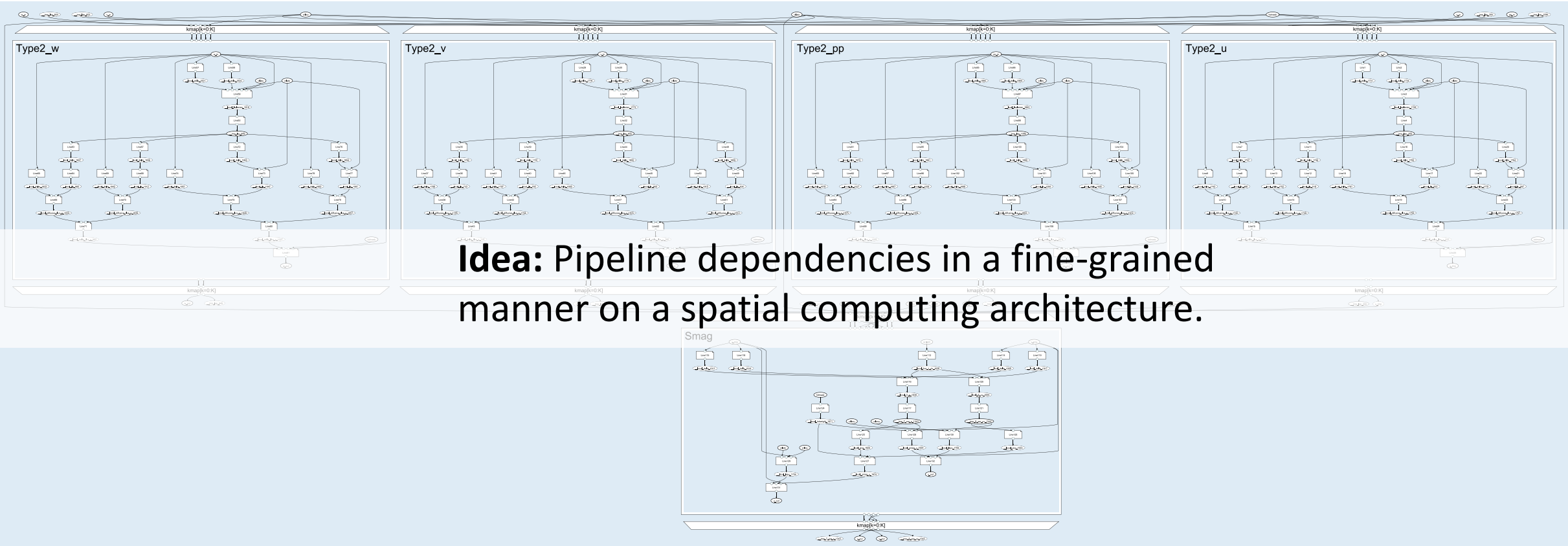
[1] COSMO 1.1 km

2018-05-29 00:00 UTC+2

Weather service is currently GPU-powered [2]

[1] Institute for Atmospheric and Climate Science and Computer Graphics Laboratory, ETH Zürich [https://vimeo.com/389292423]
[2] Swiss National Supercomputing Center (CSCS) [https://www.cscs.ch/computers/arolla-tsa-meteoswiss]

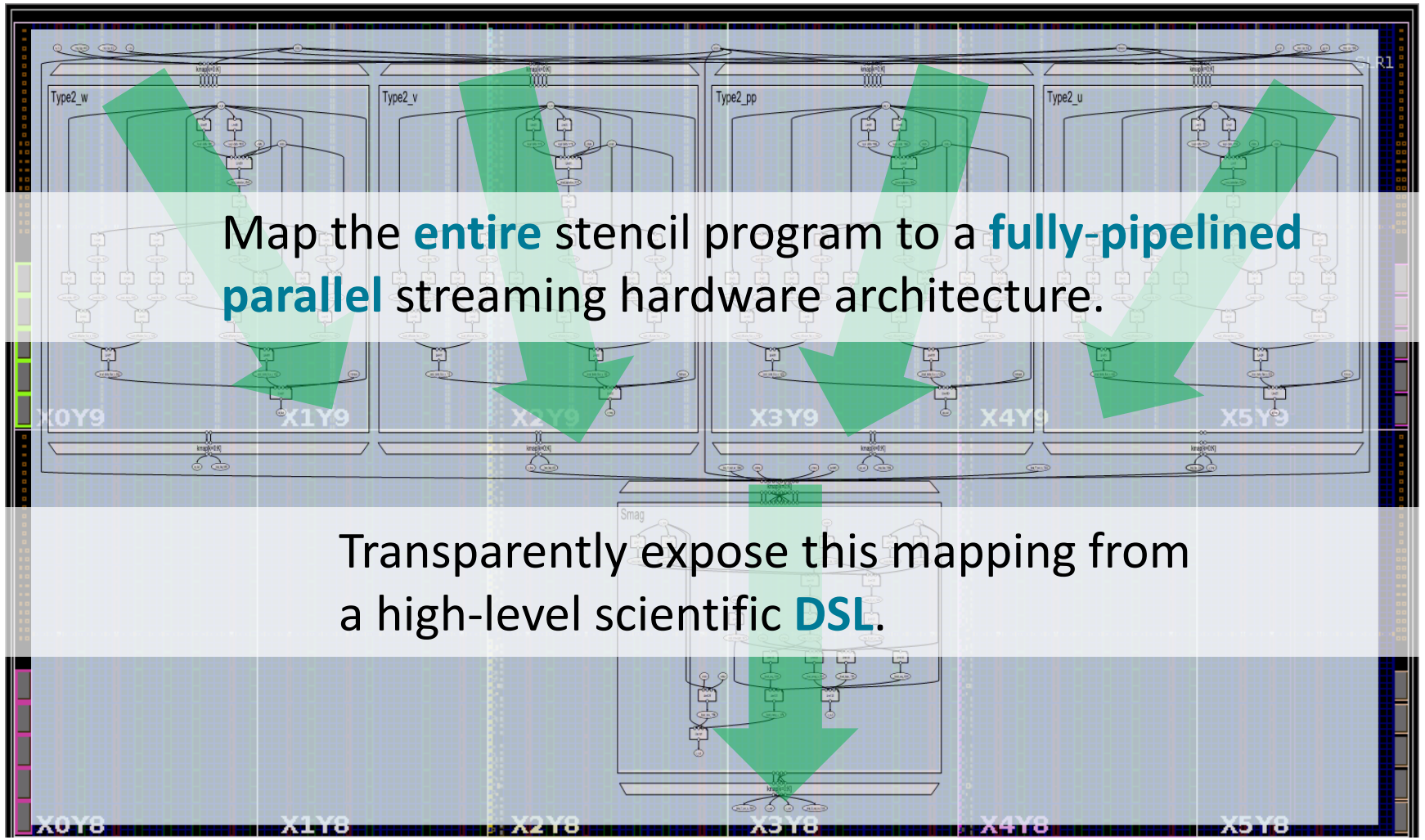
Weather stencil programs



Idea: Pipeline dependencies in a fine-grained manner on a spatial computing architecture.

Gysi et al.: MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures, 29th International Conference on Supercomputing (ICS'15)

StencilFlow



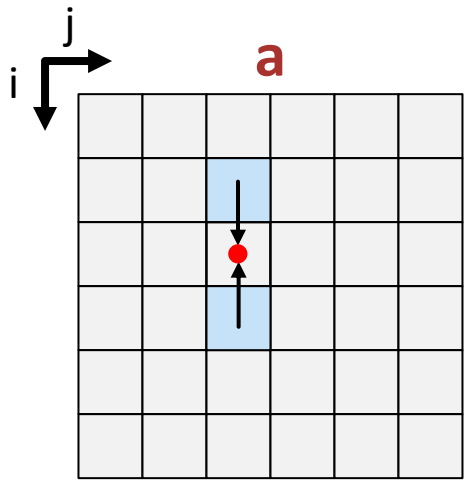
Map the **entire** stencil program to a **fully-pipelined parallel** streaming hardware architecture.

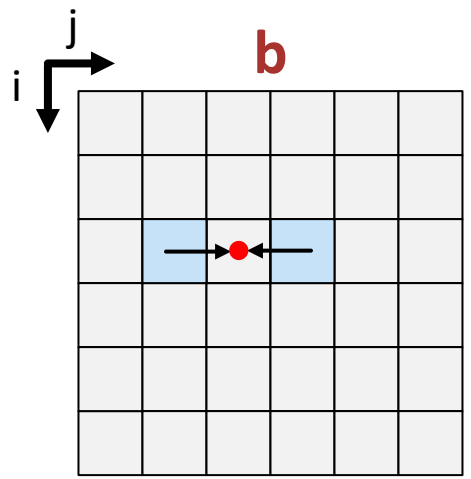
Transparently expose this mapping from a high-level scientific **DSL**.

Gysi et al.: MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures, 29th International Conference on Supercomputing (ICS'15)

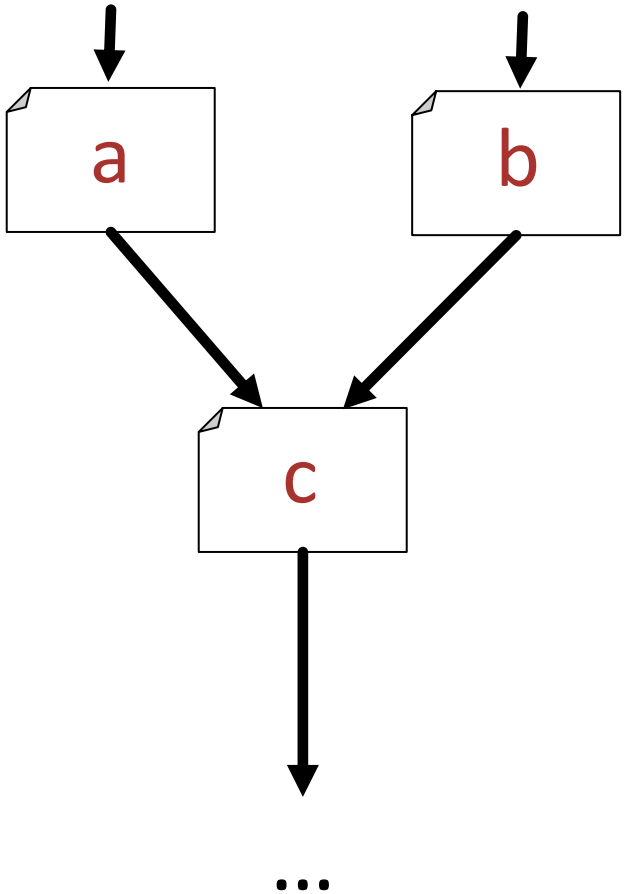
Stencil Operation

```
for (int i = 1; i < N - 1; ++i) {
  for (int j = 1; j < M - 1; ++j) {
    c[i, j] = (a[i-1, j] + a[i+1, j]) -
              (b[i, j-1] + b[i, j+1]);
  }
}
```





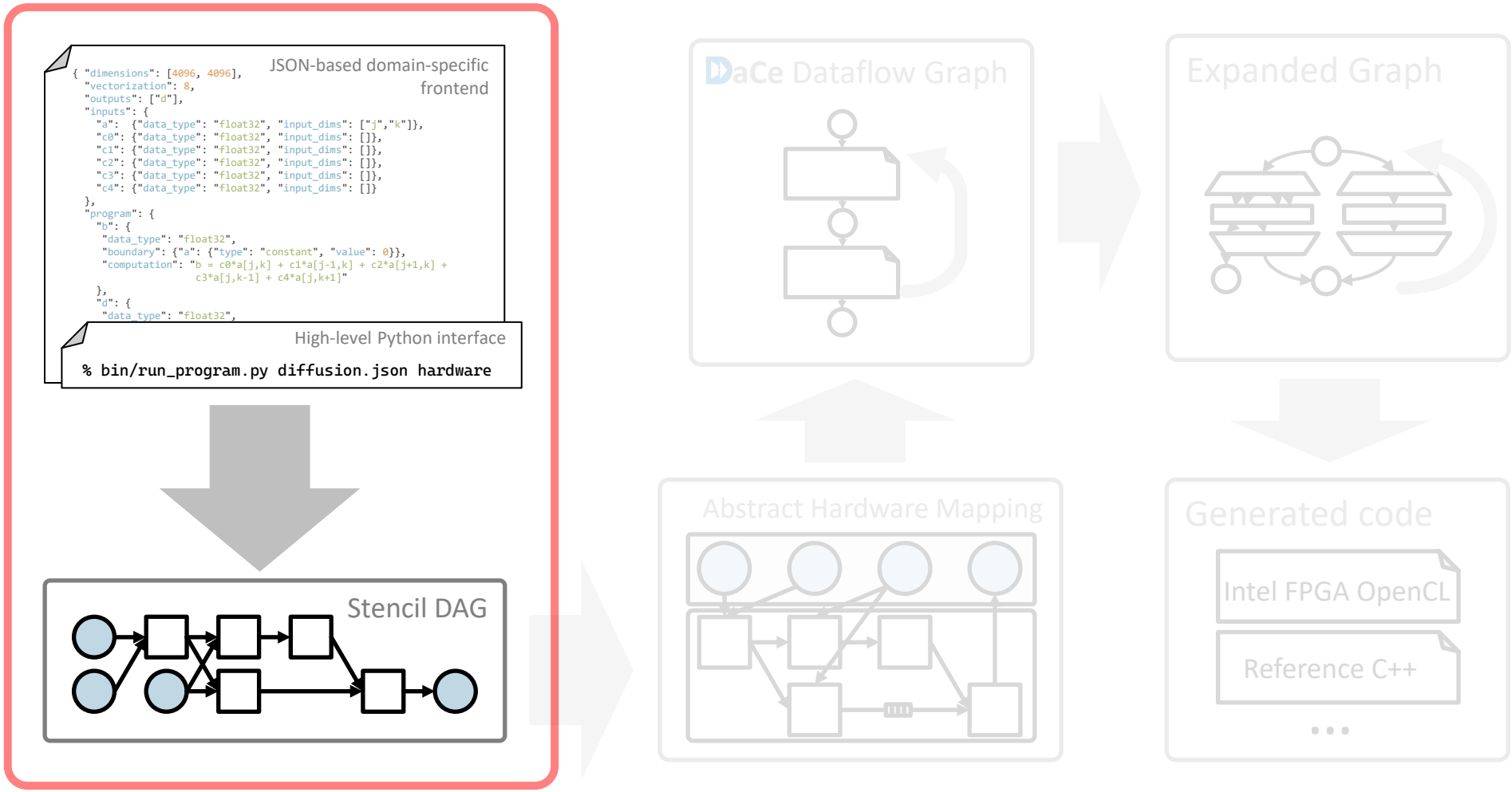
Stencil Program



A stencil **program** is a **DAG** of stencil operations working on the **same grid**.

Gysi et al.: MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures, 29th International Conference on Supercomputing (ICS'15)

Prototype: The StencilFlow Stack



de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Frontend

GridTools

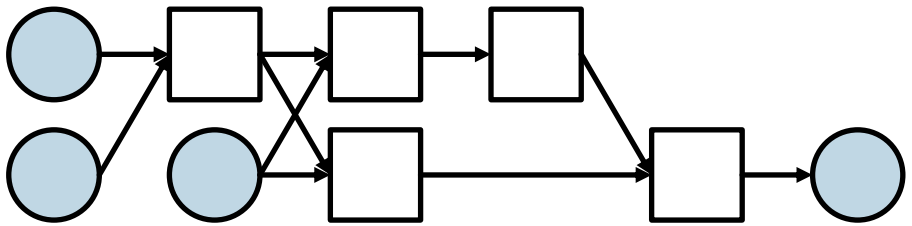
support on the way
in collaboration with

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

JSON-based domain-specific frontend

```
{ "dimensions": [4096, 4096],
  "vectorization": 8,
  "outputs": ["d"],
  "inputs": {
    "a":  {"data_type": "float32", "input_dims": ["j","k"]},
    "c0": {"data_type": "float32", "input_dims": []},
    "c1": {"data_type": "float32", "input_dims": []},
    "c2": {"data_type": "float32", "input_dims": []},
    "c3": {"data_type": "float32", "input_dims": []},
    "c4": {"data_type": "float32", "input_dims": []}
  },
  "program": {
    "b": {
      "data_type": "float32"
```

Stencil DAG

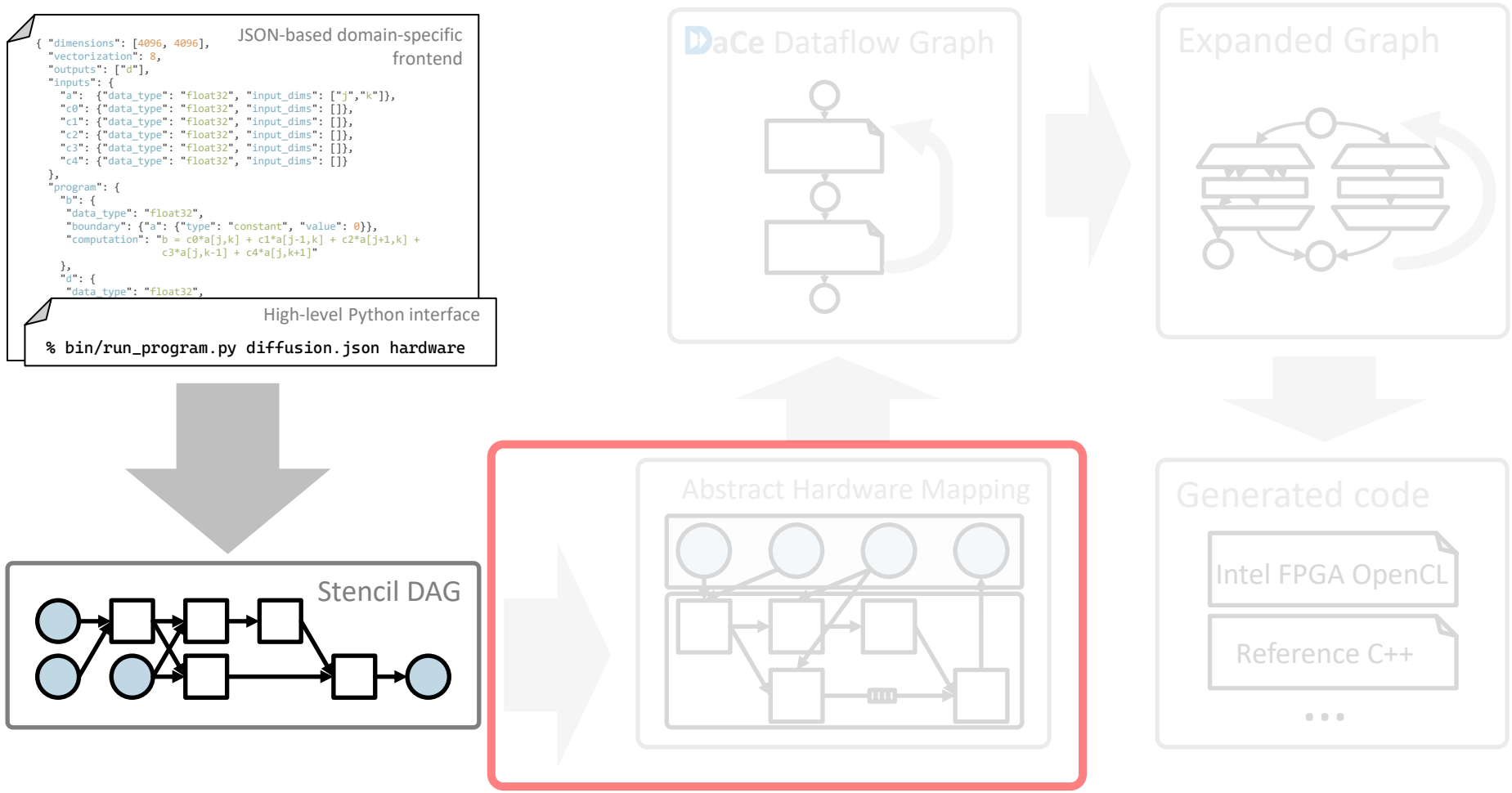


```
} }
```

High-level Python interface

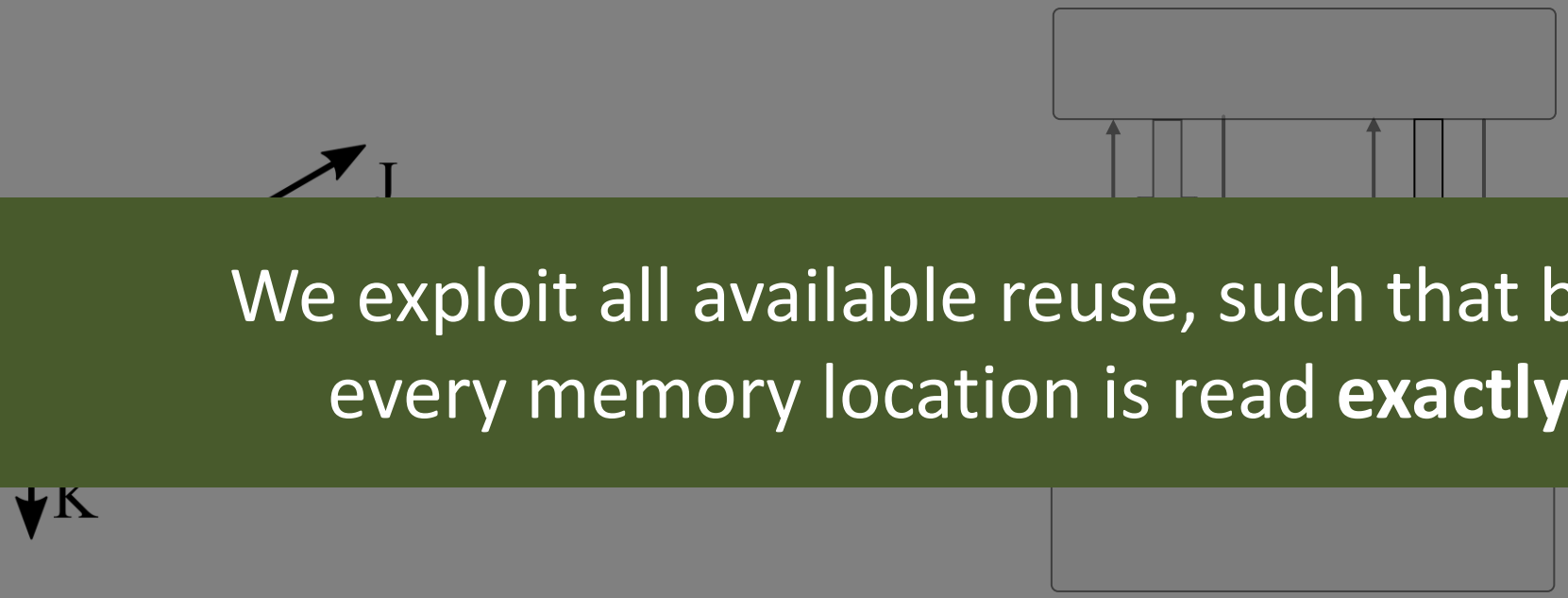
```
% bin/run_program.py diffusion.json hardware
```

The StencilFlow Stack



de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Two classes of buffers



We exploit all available reuse, such that by design, every memory location is read **exactly once**.

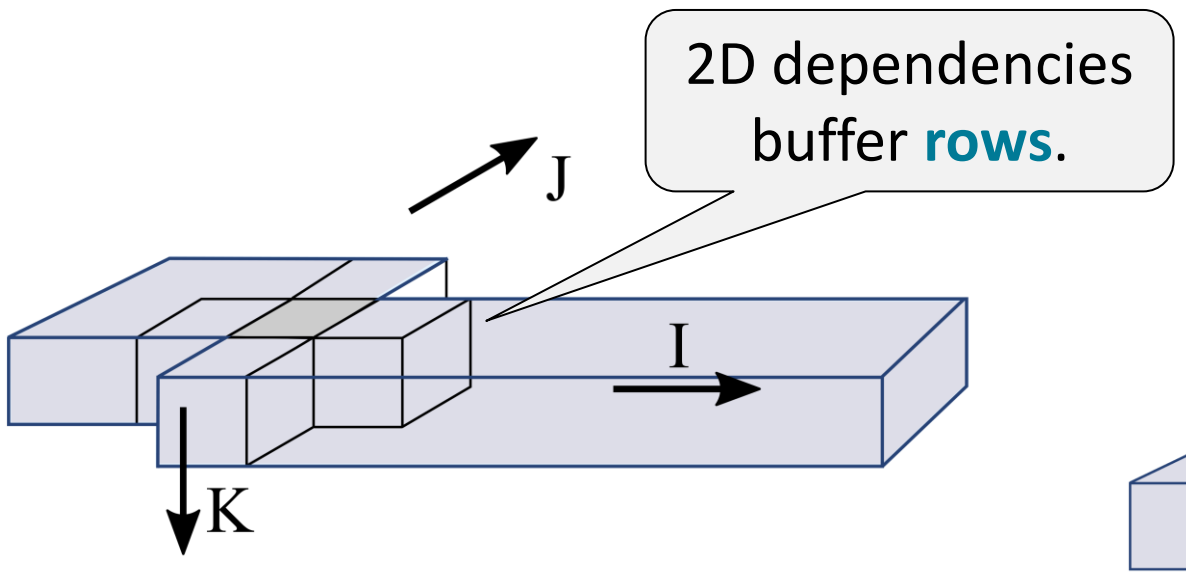
Internal buffers maximize data reuse within each stencil operation.

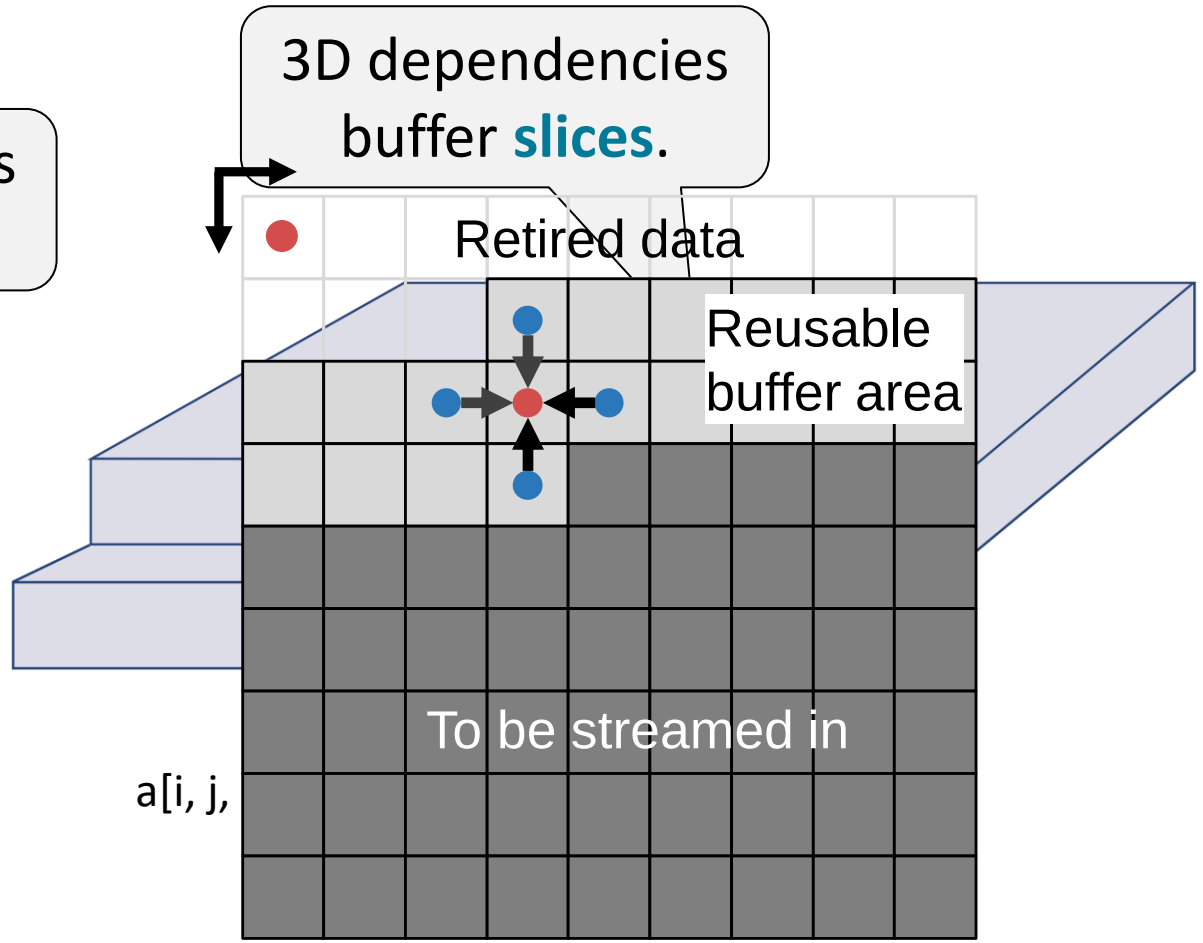
Delay buffers maximize data reuse between stencil operations.

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Internal buffers

(in general, buffers proportional to up to D-1)

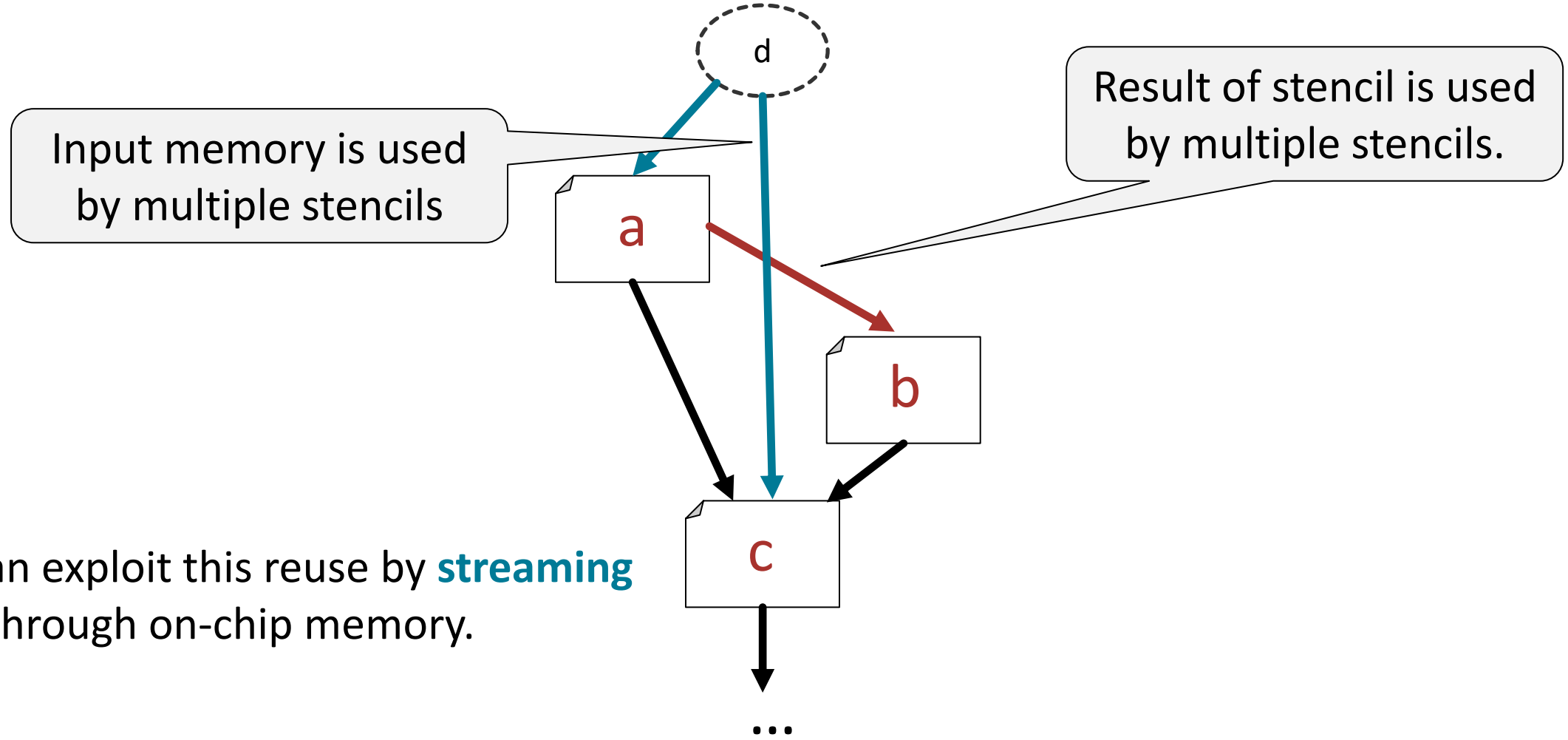




a[i-1, j, k] + a[i+1, j, k] +
a[i, j-1, k] + a[i, j+1, k]

Every cell used by a stencil operation is **only read once**.

Global reuse



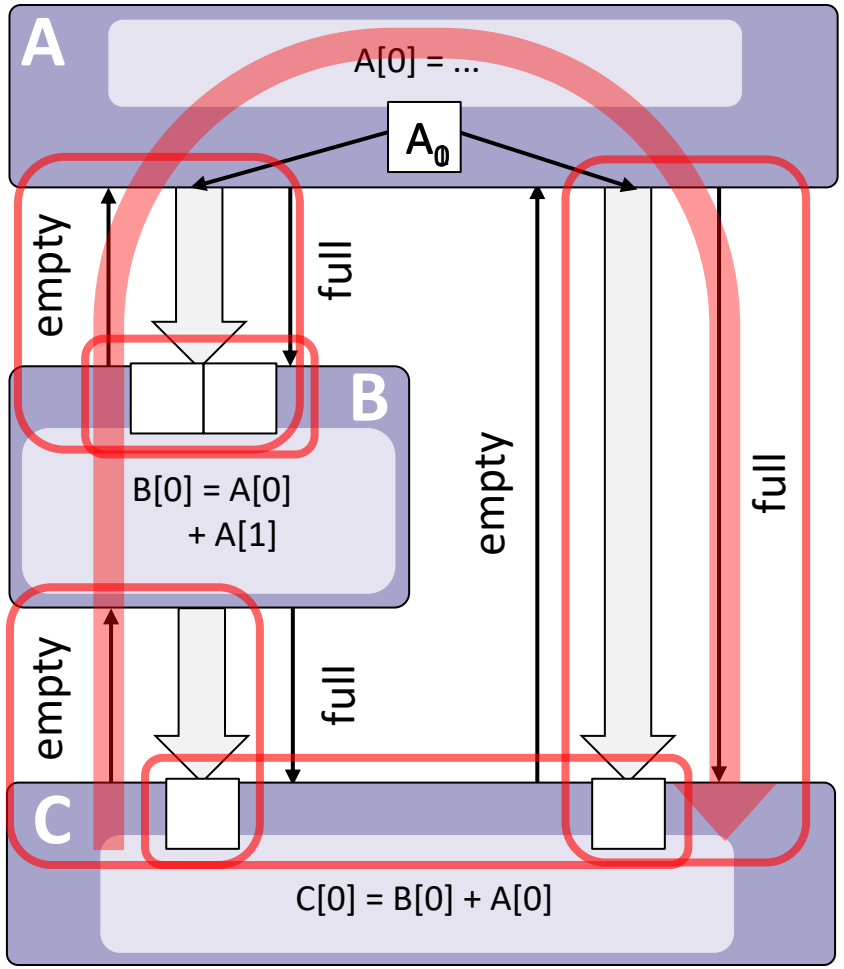
We can exploit this reuse by **streaming** data through on-chip memory.

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Delay buffers

Intermediate buffers represent physical hardware

Sizes are fixed at compile time

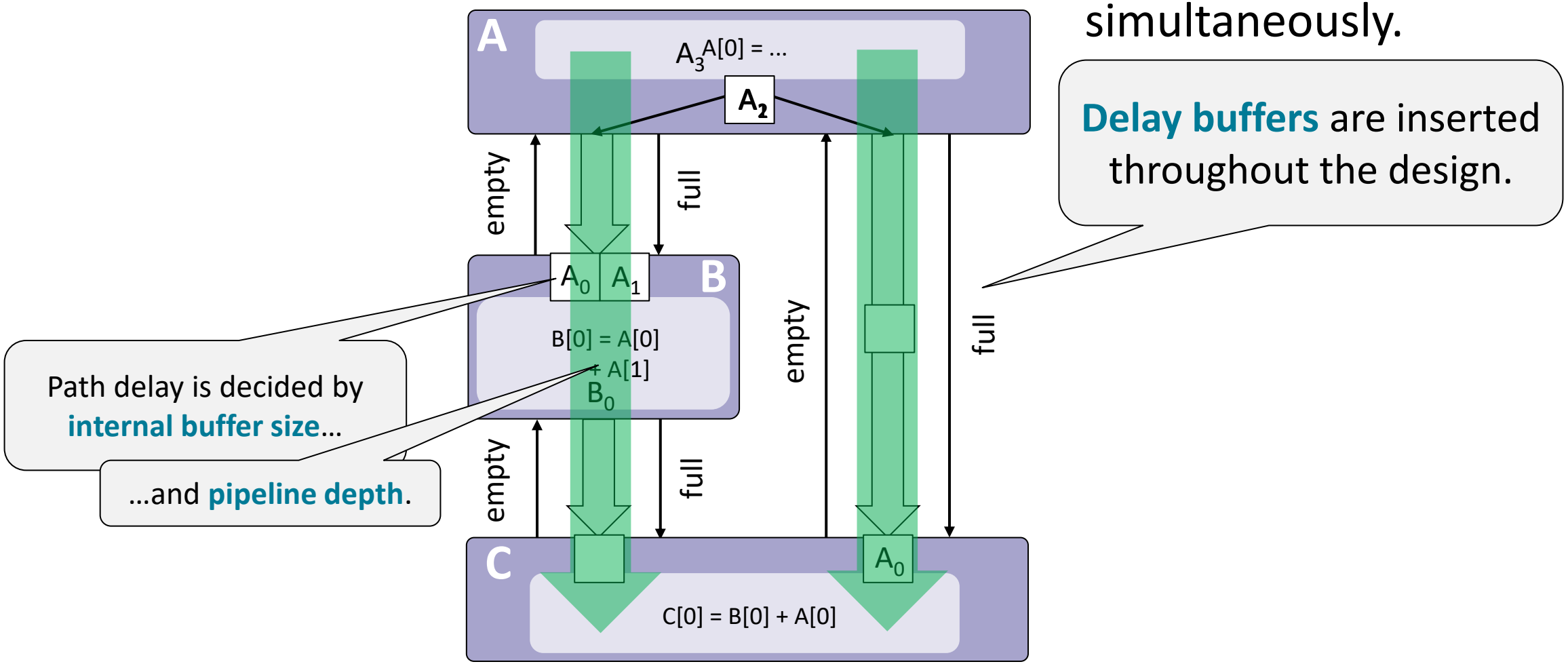


Design can **deadlock** when path is full.

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Delay buffers

No deadlock.
Both paths **stream** simultaneously.



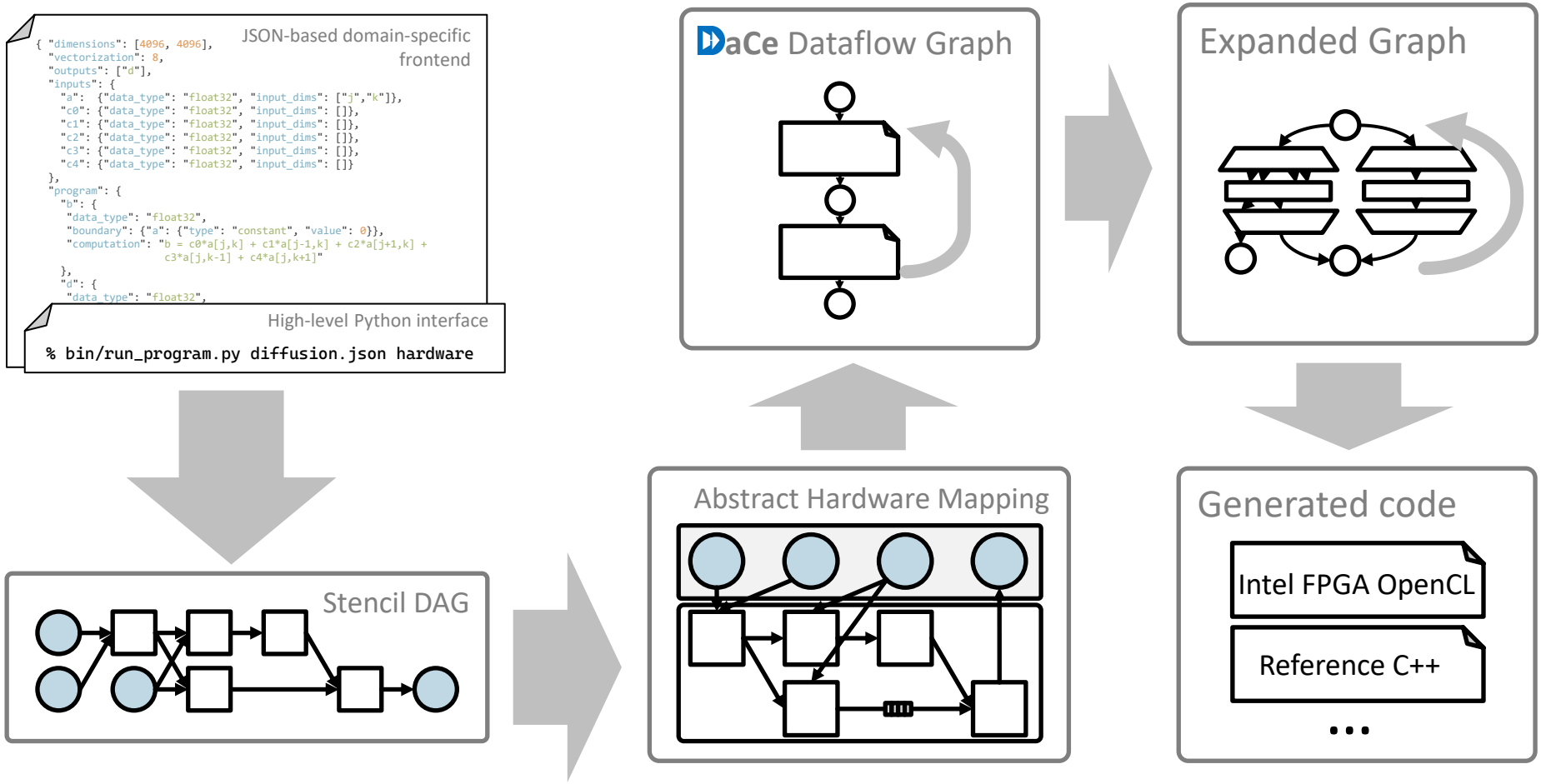
Delay buffers are inserted throughout the design.

Path delay is decided by **internal buffer size**…

…and **pipeline depth**.

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

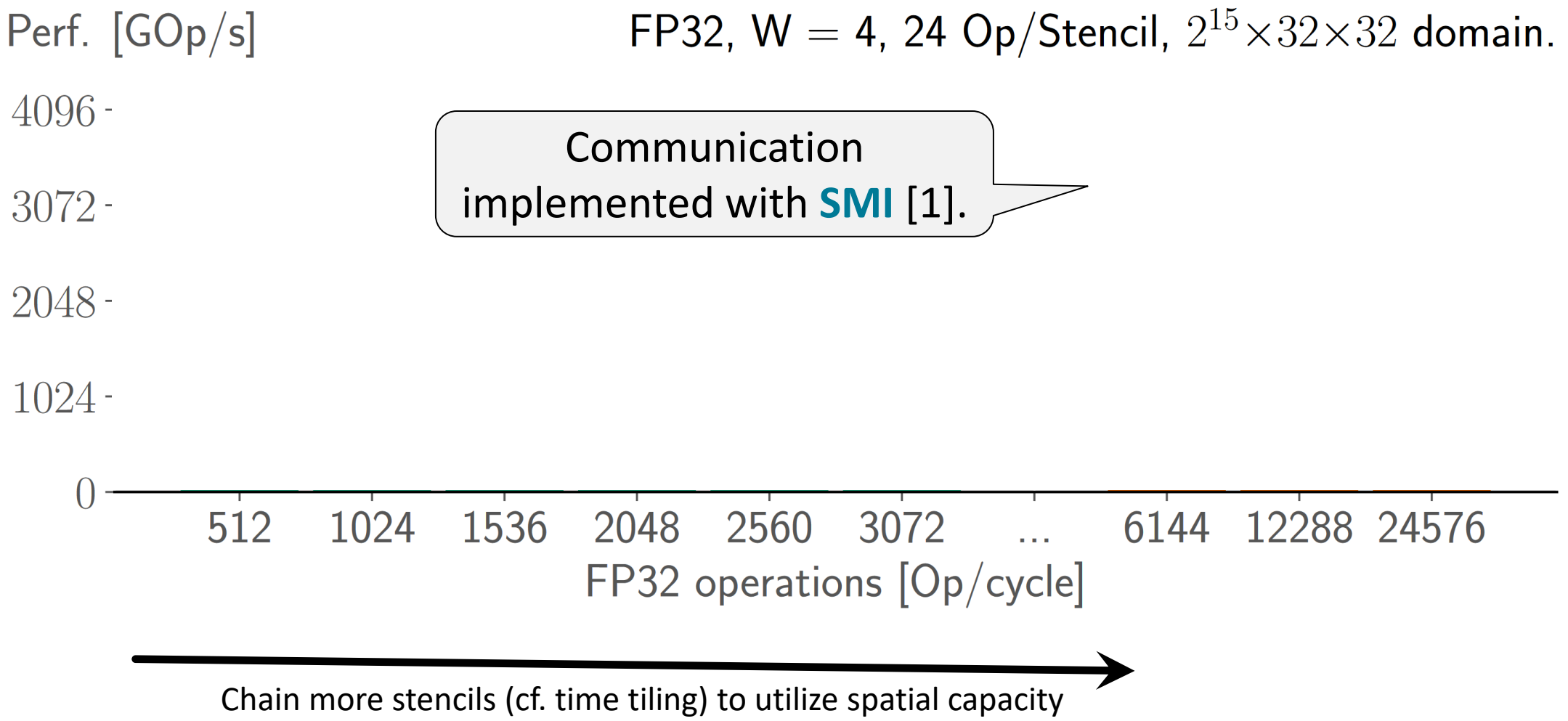
The StencilFlow Stack - Results



de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

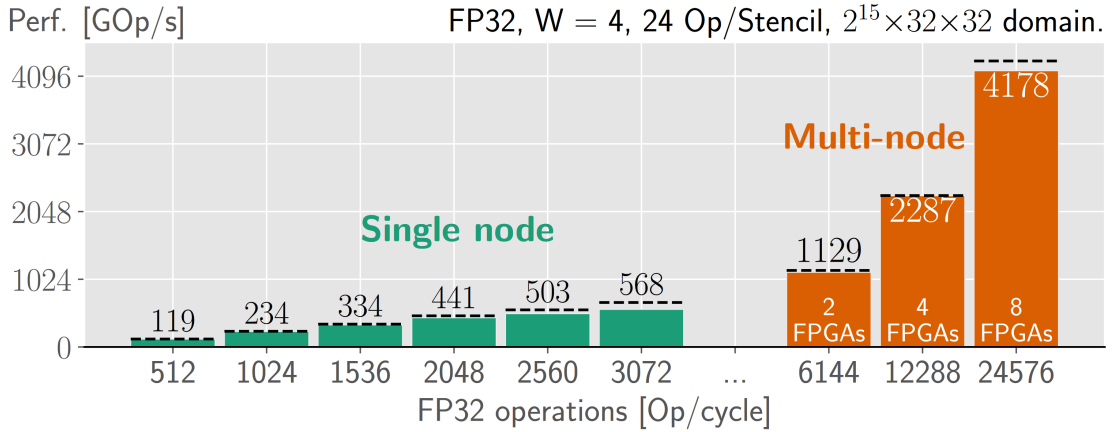
Simple iterative stencils

BittWare 520N, 1-4x Stratix 10 GX 2800, 4x DDR4 at 2400 MHz, 4x 40 Gbit/s Ethernet, p520_max_sg280l shell, Intel FPGA OpenCL SDK/Quartus 19.1.0.



[1] De Matteis et al. "Streaming Message Interface: High-performance distributed memory programming on reconfigurable hardware." *Supercomputing 2019 (SC'19)*

Simple iterative stencils

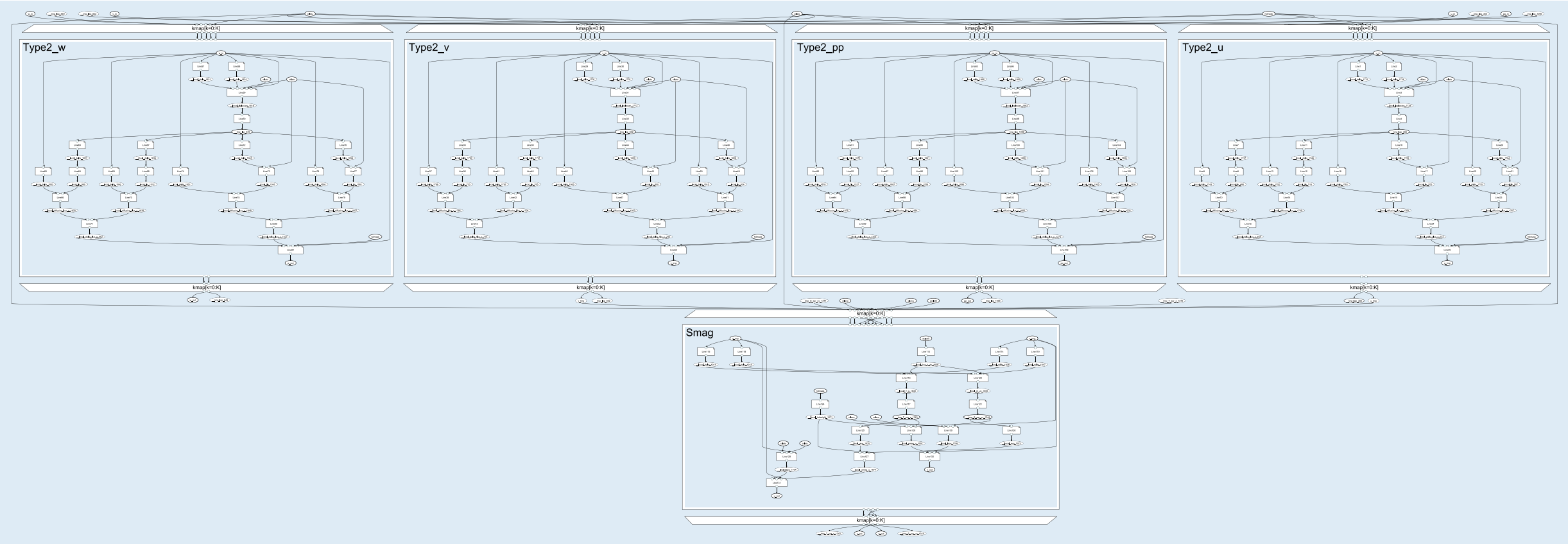


	Performance	ALM	FF	M20K	DSP
Total		103 M	3.7 M	11.7 K	5760
Avail.		692 K	2.8 M	8.9 K	4468
Jacobi 3D (Ours)	265 GOp/s	233 K 33.6%	534 K 19.3%	1495 16.7%	784 17.6%
Jacobi 3D W=8 (Ours)	921 GOp/s	437 K 63.1%	1207 K 43.6%	2285 25.5%	3072 68.8%
Diffusion 2D W=8 (Ours)	1,313 GOp/s	449 K 64.8%	1329 K 48.0%	2565 28.6%	2304 51.6%
Diffusion 3D W=8 (Ours)	1,152 GOp/s	567 K 81.9%	1606 K 57.9%	5357 59.8%	3072 68.8%
Diffusion 2D (Zohouri et. al. [8])	913 GOp/s	471.4 K 68.0%	1173.6 K 42.3%	2204 24.6%	3844 86.0%
Diffusion 3D (Zohouri et. al. [8])	934 GOp/s	450.5 K 65.0%	1078.2 K 38.9%	8684 97.0%	3592 80.4%

```
% bin/run_program.py diffusion_3d_vec8.json hardware
```

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Taming the weather



Horizontal diffusion program obtained from generic weather frontend.

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Taming the weather

Program is recovered as StencilFlow **DSL**.

```
"inputs": {
 "pp_in": {
  "data": "pp_in_128x64x128_float64.dat",
  "data_type": "float64",
  "dimensions": [
   "i",
   "j",
   "k"
  ]
 },
 "crlato": {
  "data": "crlato_128_float64.dat",
  "data_type": "float64",
  "dimensions": [
   "i"
  ]
 },
 "crlatu": {
  "data": "crlatu_128_float64.dat",
  "data_type": "float64",
  "dimensions": [
   "i"
  ]
 },
 "hdmask": {
  "data": "hdmask_128x64x128_float64.dat",
  "data_type": "float64",
  "dimensions": [
   "i",
   "j",
   "k"
  ]
 },
 "w_in": {
  "data": "w_in_128x64x128_float64.dat",
  "data_type": "float64",
  "dimensions": [
   "i",
   "j",
   "k"
  ]
 },
 ...
```

GridTools

support on the way
in collaboration with

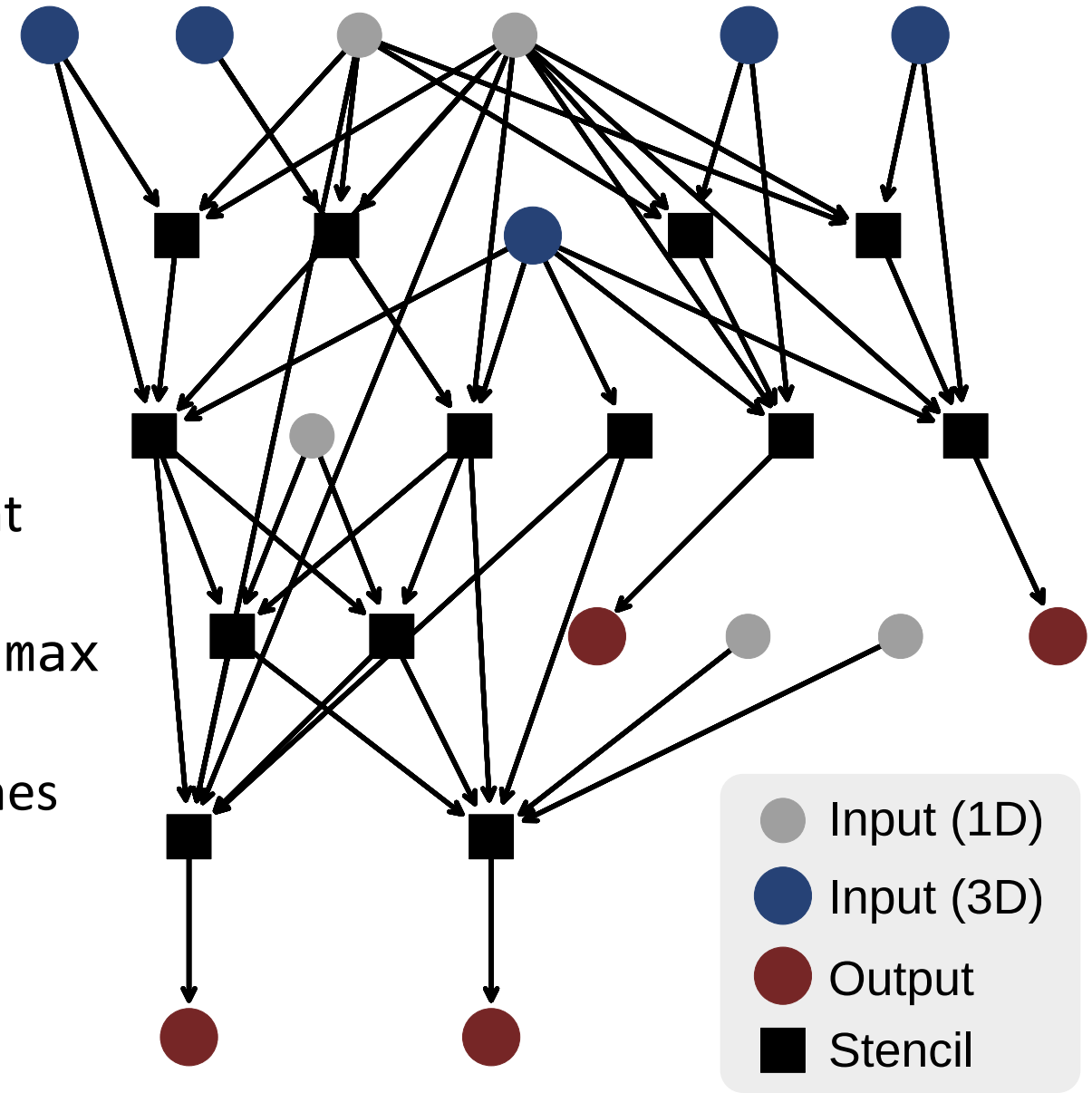
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Taming the weather

We can now generate and process the stencil **DAG**.

- Performs **130** floating point operations per grid point (including `sqrt`, `min`, and `max` operations).
- 20 data-dependent branches
- Arithmetic intensity of **65/18 Op/Byte**.



de Fine Licht et al.: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems 19th ACM/IEEE International Symposium on Code Generation and Optimization (CGO'21), 2021

Taming the weather

CPU and GPU results are highly optimized codes produced by the DAWN framework [1].

318 MHz at 48% DSP utilization

all memory bound

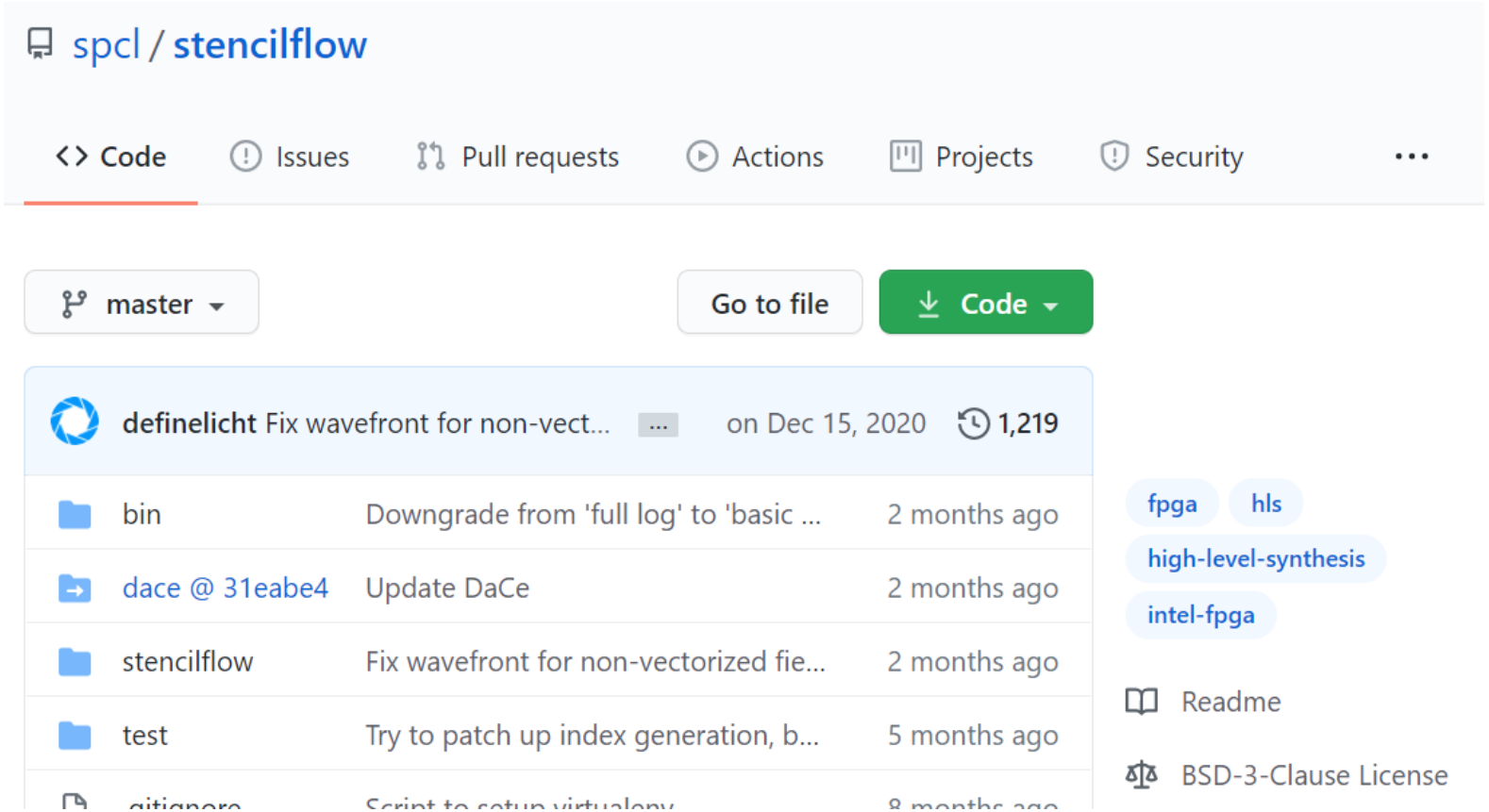
	Runtime	Performance	Peak BW.	%Roof.
Stratix 10	1,178 µs	145 GOp/s	77 GB/s	52%

The Stratix 10 is held back by insufficient bandwidth.

FPGAs are good at deterministically **exploiting bandwidth**, but require a lot of **pipeline parallelism**.

[1] Osuna et al.: Dawn: a High Level Domain-Specific Language Compiler Toolchain for Weather and Climate Applications, Supercomputing Frontiers and Innovation 2020

Reproducibility artifact



We would love it if you **break it**!

https://github.com/spcl/stencilflow

Maybe we can get a full weather/climate code on FPGAs

New Study: Scientists Create Earth's Highly-Accurate 'Digital Twin' to Project Climate Event

By Precious Smith Mar 01, 2021 11:09 AM EST

Computer scientists at ETH (Swiss Federal Institute of Technology, Zurich) are trying to create a detailed digital twin of our Big Blue Marble in a kind of pixelated cloning experiment intended to serve as a model of experiment for Earth's climatic changes.

GPU: The Most Encouraging Option

Presently, researchers have the believe that super computers based on graphics processing units (GPU) seem to be the most encouraging option for the creation of their digital earth.

They appraise that running a full-scale digital twin, matching hardware with advanced algorithms, would need a system operating approximately 20,000 GPUs and consuming nearly 20MW of total power.

Perspective | Published: 22 February 2021

The digital revolution of Earth-system science

Peter Bauer, Peter D. Dueben, Torsten Hoefler, Tiago Quintino, Thomas C. Schulthess & Nils P. Wedi

Nature Computational Science **1**, 104–113 (2021) | Cite this article

13k Accesses | **2** Citations | **276** Altmetric | Metrics

Abstract

Computational science is crucial for delivering reliable weather and climate predictions. However, despite decades of high-performance computing experience, there is serious concern about the sustainability of this application in the post-Moore/Dennard era. Here, we discuss the present limitations in the field and propose the design of a novel infrastructure that is scalable and more adaptable to future, yet unknown computing architectures.

https://www.nature.com/articles/s43588-021-00023-0

Another observation: modularity of frequency

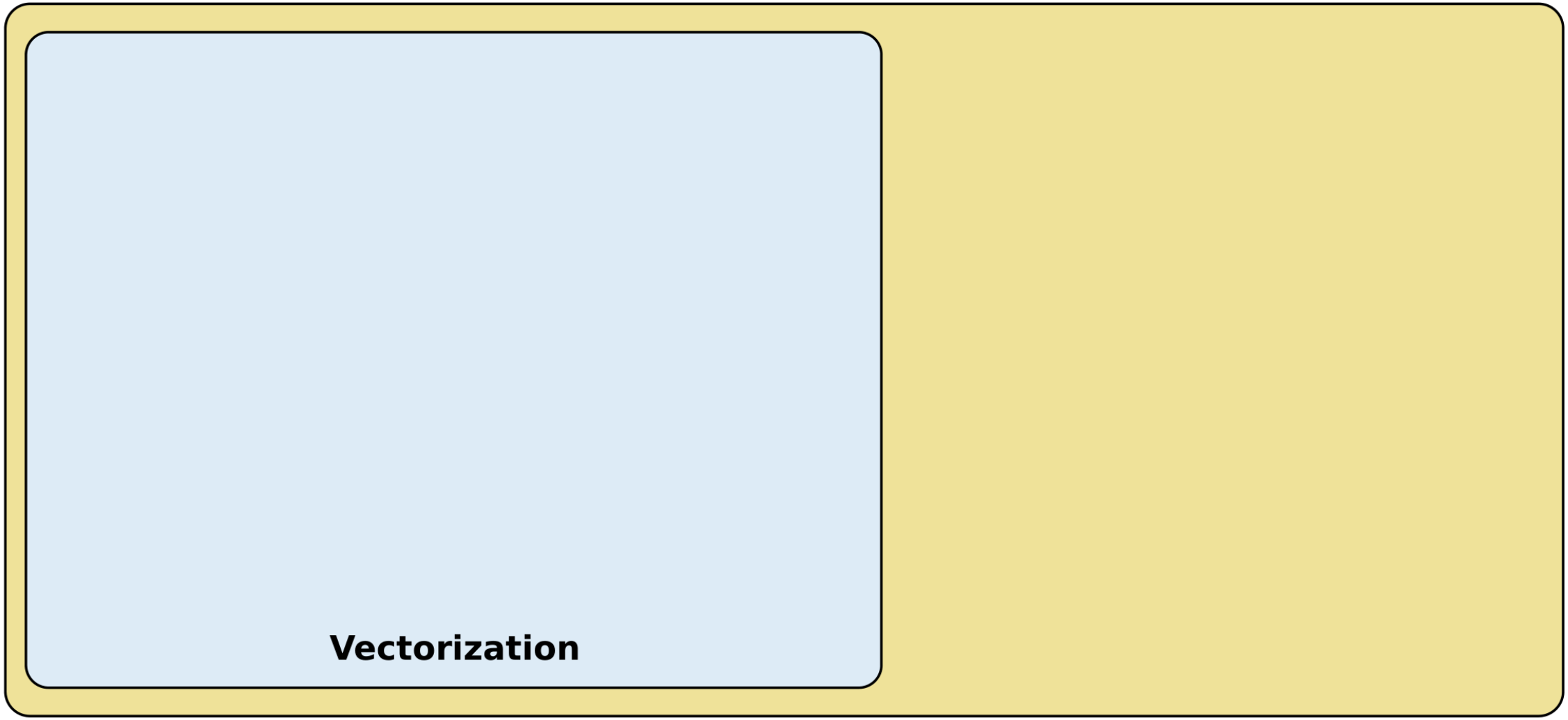
- Despite all data-movement optimizations, our frequencies were far from the target (often <200 MHz)

```
int i, k;
for(i=0; i<k; ++i)
  n += n*obj;
  k = n*n
```

```
int i, k;
for(i=0; i<k; ++i)
  n += n*obj;
  k = n*n
```

```
int i, k;
for(i=0; i<k; ++i)
  n += n*obj;
  k = n*n
```

Another trick: temporal vectorization aka. automatic multi-pumping

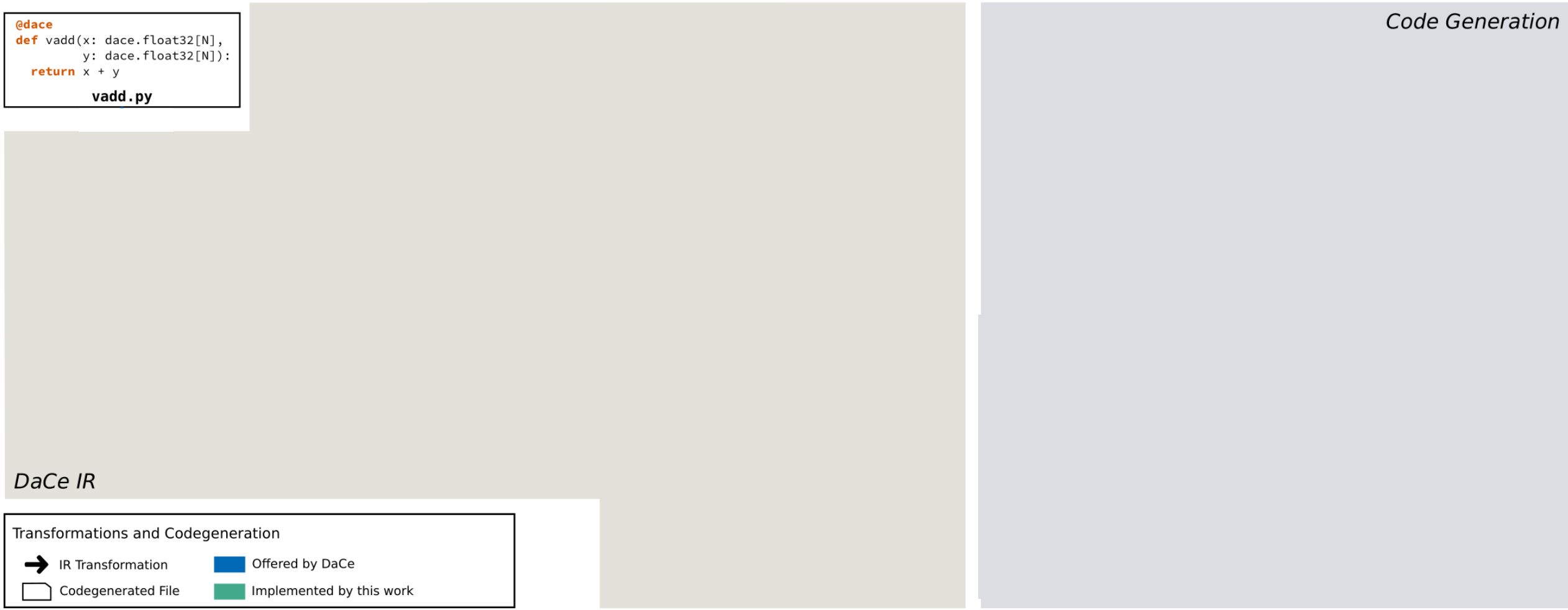


[1] Johnsen et al.: Temporal Vectorization: A Compiler Approach to Automatic Multi-Pumping, ICCAD 2022

How can temporal vectorization be implemented?

```
@dace
def vadd(x: dace.float32[N],
         y: dace.float32[N]):
  return x + y
```

vadd.py



[1] Johnsen et al.: Temporal Vectorization: A Compiler Approach to Automatic Multi-Pumping, ICCAD 2022

Performance of temporal vectorization on a ~~Xilinx~~ AMD Alveo U280

	32 PEs		
	CA [10]	O	DP
Freq CL0 [MHz]	250	268	261.4
Freq CL1 [MHz]	-	-	452.8
Perf [GOp/s]	253.2	256.1	219.1
LUT Logic [%]	43.9	44.8	32.1
LUT Memory [%]	6.9	13	10.1
Registers [%]	44.5	44.3	36.6
BRAM [%]	81.4	80.3	47
DSP [%]	88.9	90	45.6
MOp/s per DSP	98.9	98.8	167.0

Matrix Multiplication

	S=8	
	O	DP
Freq CL0 [MHz]	307.6	322.4
Freq CL1 [MHz]	-	510.4
Perf [GOp/s]	101.4	96.9
LUT Logic [%]	20.25	14.2
LUT Memory [%]	6.21	6.89
Registers [%]	22.48	19.14
BRAM [%]	15.33	10.57
DSP [%]	28.89	14.44
MOp/s per DSP	121.9	232.8

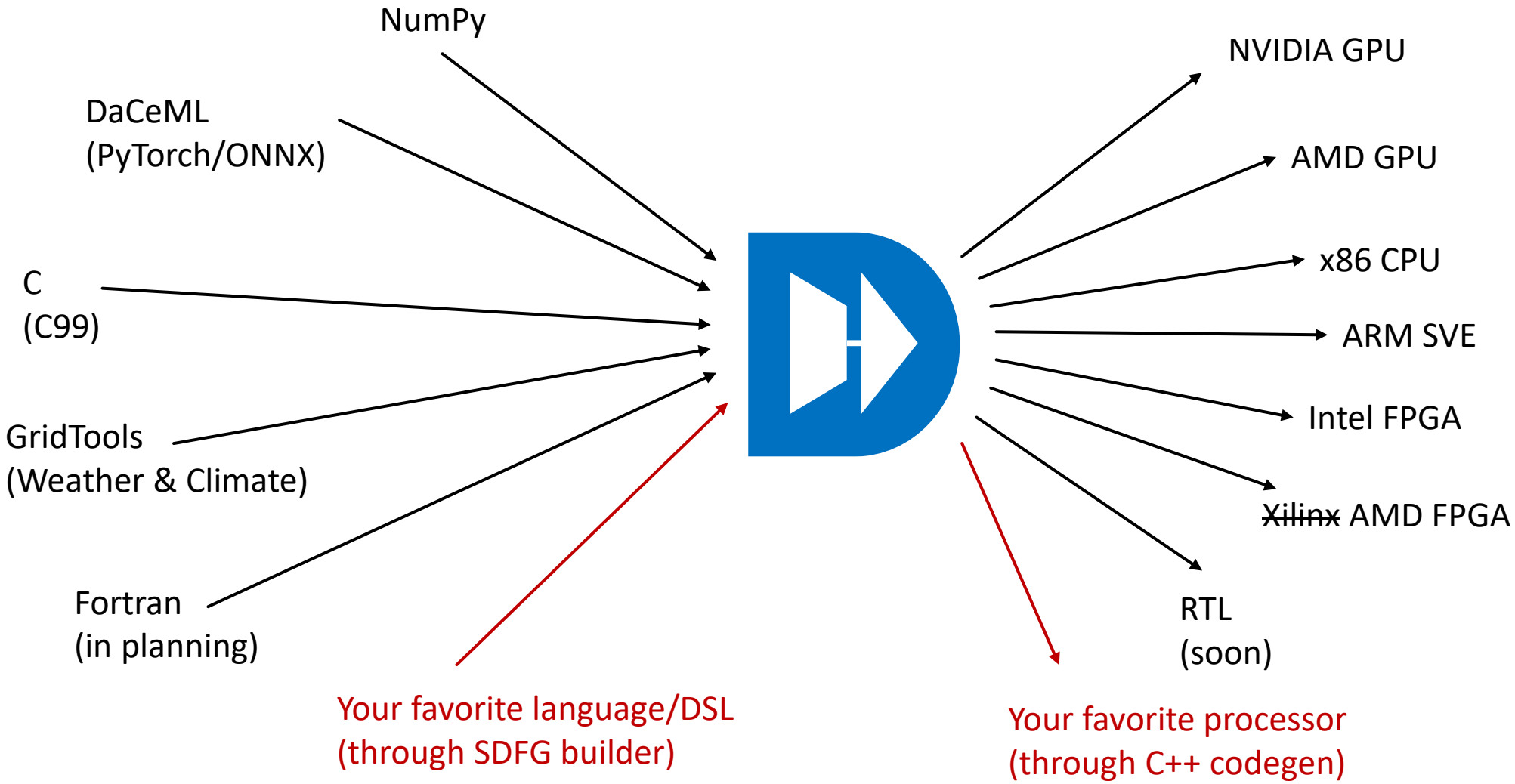
Jacobi 3D stencil

	S=8	
	O	DP
Freq CL0 [MHz]	309.1	329.4
Freq CL1 [MHz]	-	537.3
Perf [GOp/s]	110.4	102.8
LUT Logic [%]	16.55	12.08
LUT Memory [%]	4.85	5.27
Registers [%]	18.25	15.88
BRAM [%]	10.57	8.18
DSP [%]	31.67	16.67
MOp/s per DSP	121.0	214.2

Diffusion 3D stencil

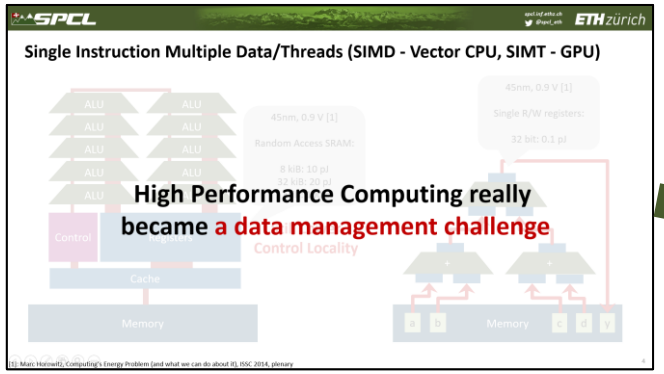
[1] Johnsen et al.: Temporal Vectorization: A Compiler Approach to Automatic Multi-Pumping, ICCAD 2022

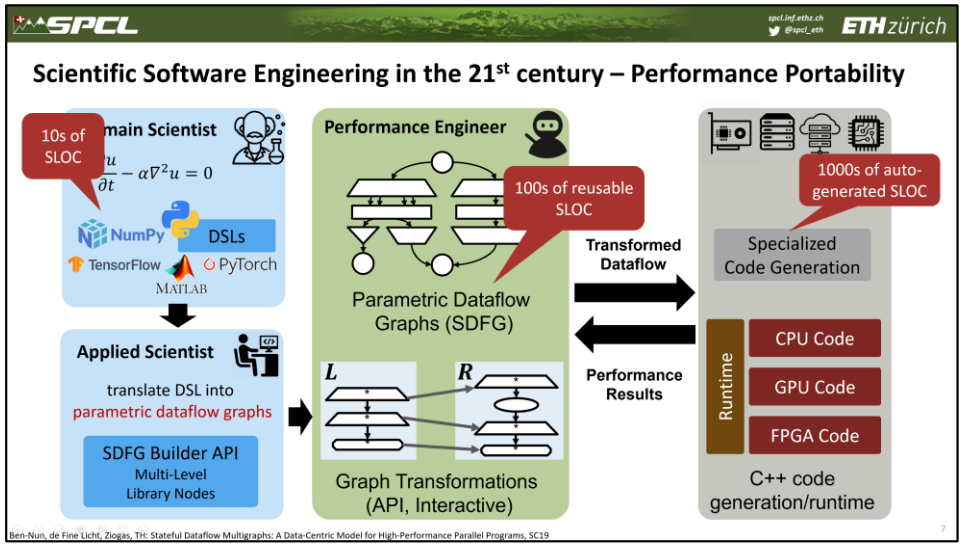
DaCe is a versatile platform

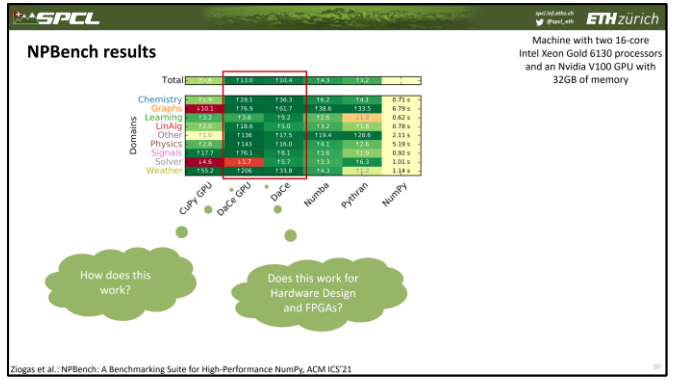


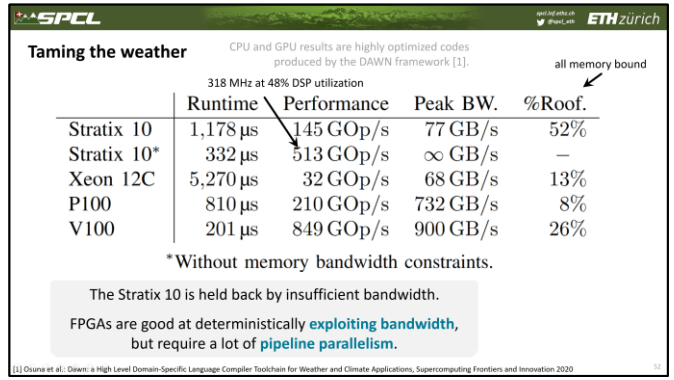
Ben-Nun et al. "Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures." *Supercomputing 2019 (SC'19)*.

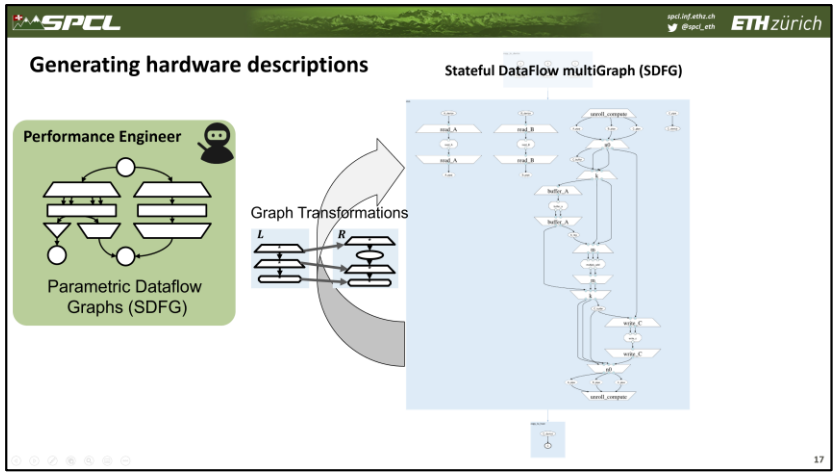
Overview and wrap-up

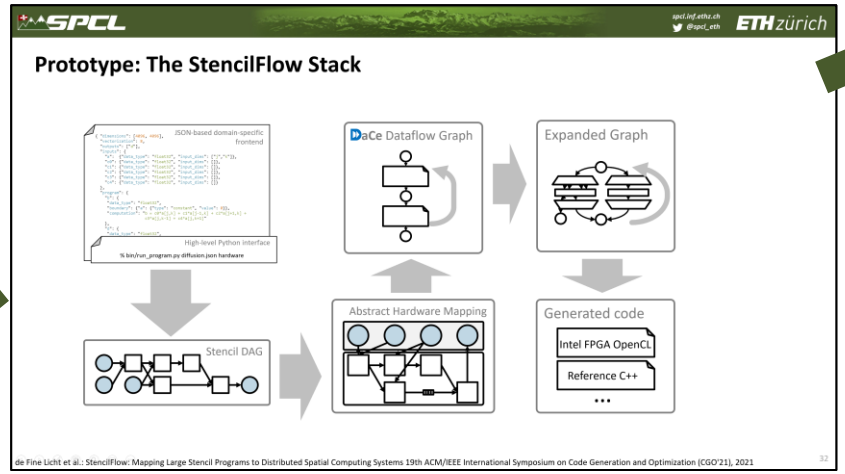


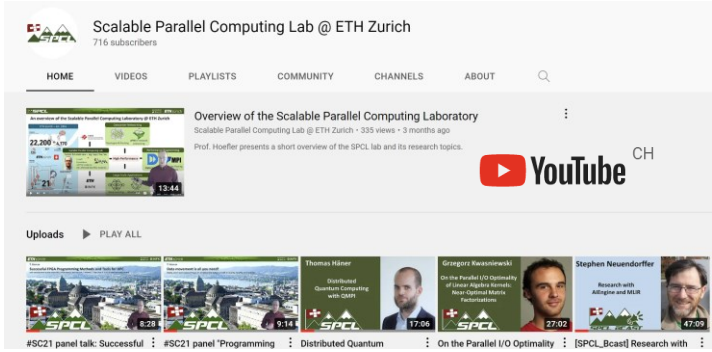








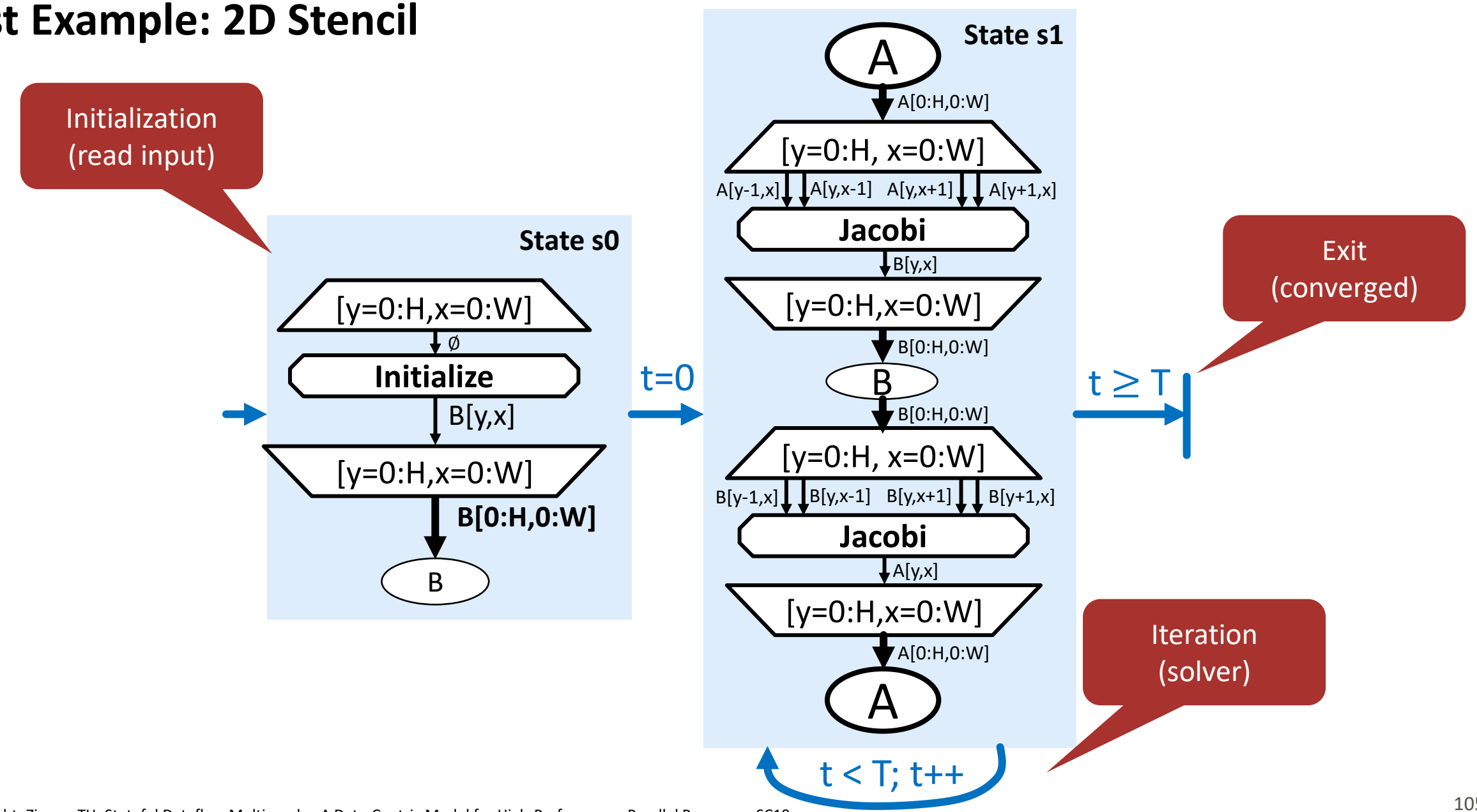




https://www.youtube.com/c/ScalableParallelComputingLabETHZurich

This project has received funding from the European Research Council (ERC) under grant agreements "DAPP and PSAP (PI: T. Hoefler)".

First Example: 2D Stencil

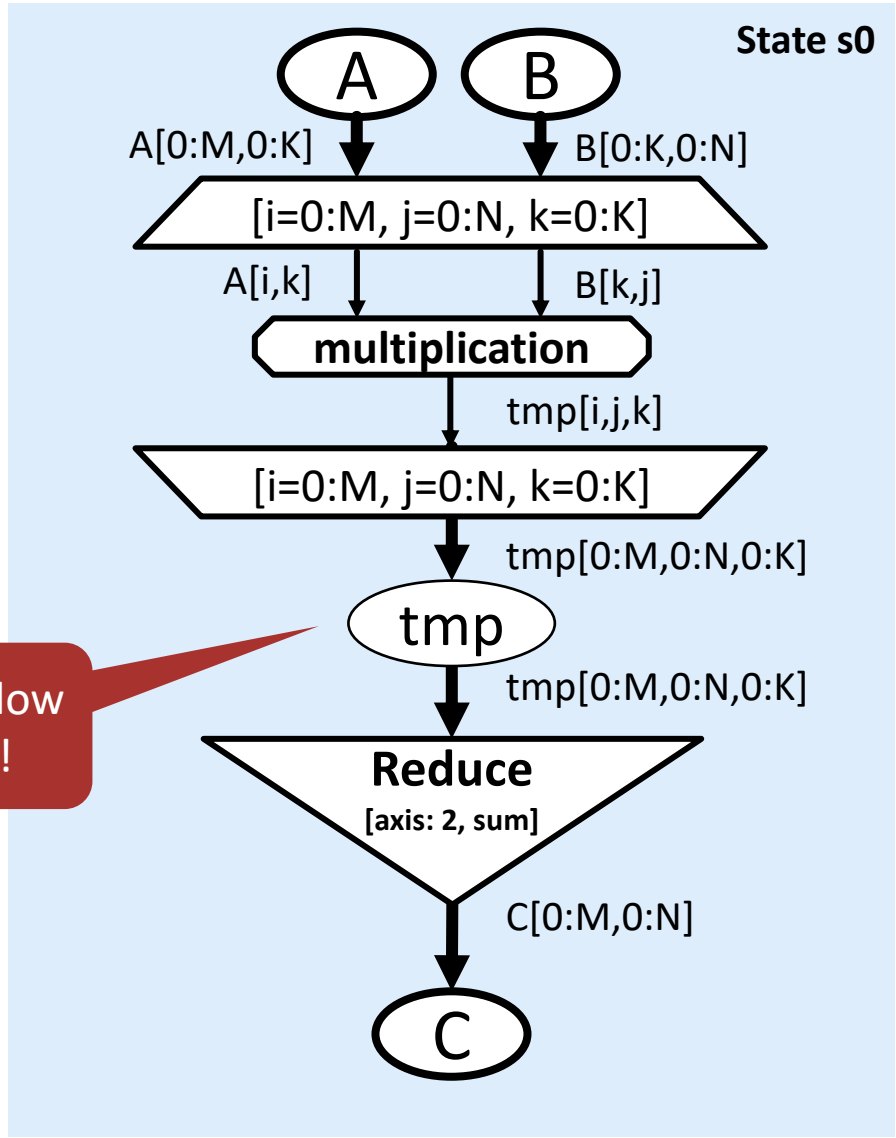


Second Example: MatMul

DaCe-Python
Explicit (minimal)
side-effect code

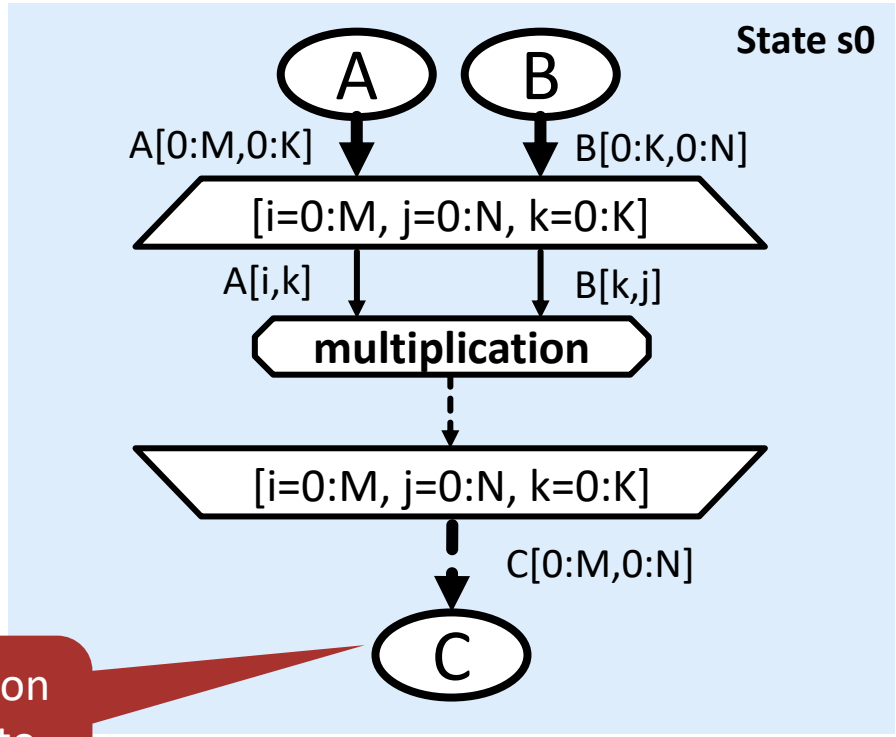
```
@dace.program
def gemm(A, B, C):
    # Transient variable
    tmp = dace.define_local([M, N, K], dtype=A.dtype)

    @dace.map
    def multiplication(i: _[0:M], j: _[0:N], k: _[0:K]):
        in_A << A[i,k]
        in_B << B[k,j]
        out  >> tmp[i,j,k]

        out = in_A * in_B

    dace.reduce(lambda a, b: a + b, tmp, C, axis=2)
```

N^3 size dataflow temporary!



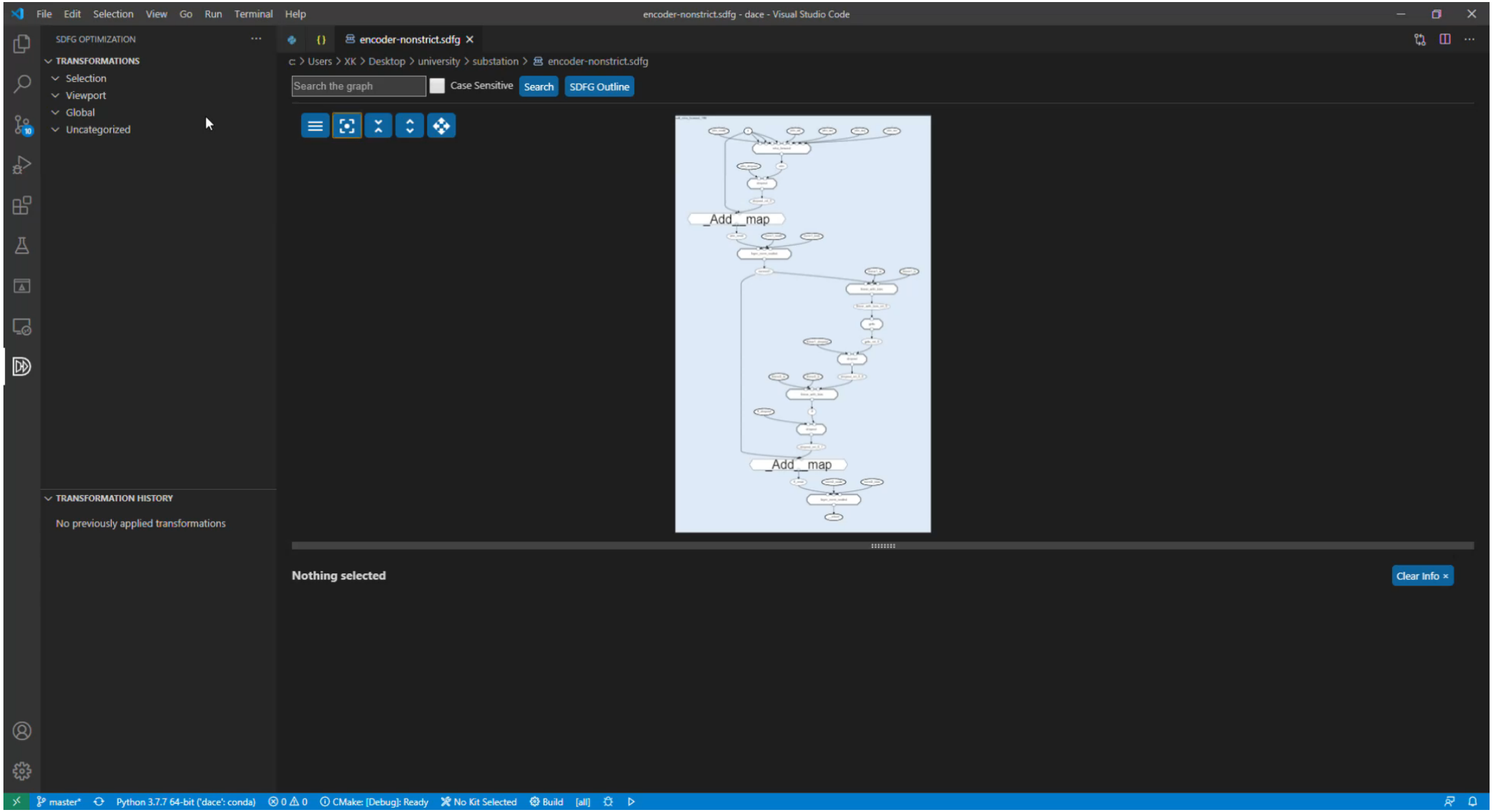
Second Example: MatMul

```
@dace.program
def gemm(A, B, C):
    # Transient variable
    tmp = dace.define_local([M, N, K], dtype=A.dtype)

    @dace.map
    def multiplication(i: _[0:M], j: _[0:N], k: _[0:K]):
        in_A << A[i,k]
        in_B << B[k,j]
        out  >> tmp[i,j,k]

        out = in_A * in_B

    dace.reduce(lambda a, b: a + b, tmp, C, axis=2)
```



MapReduce Fusion transformation to N^2 size

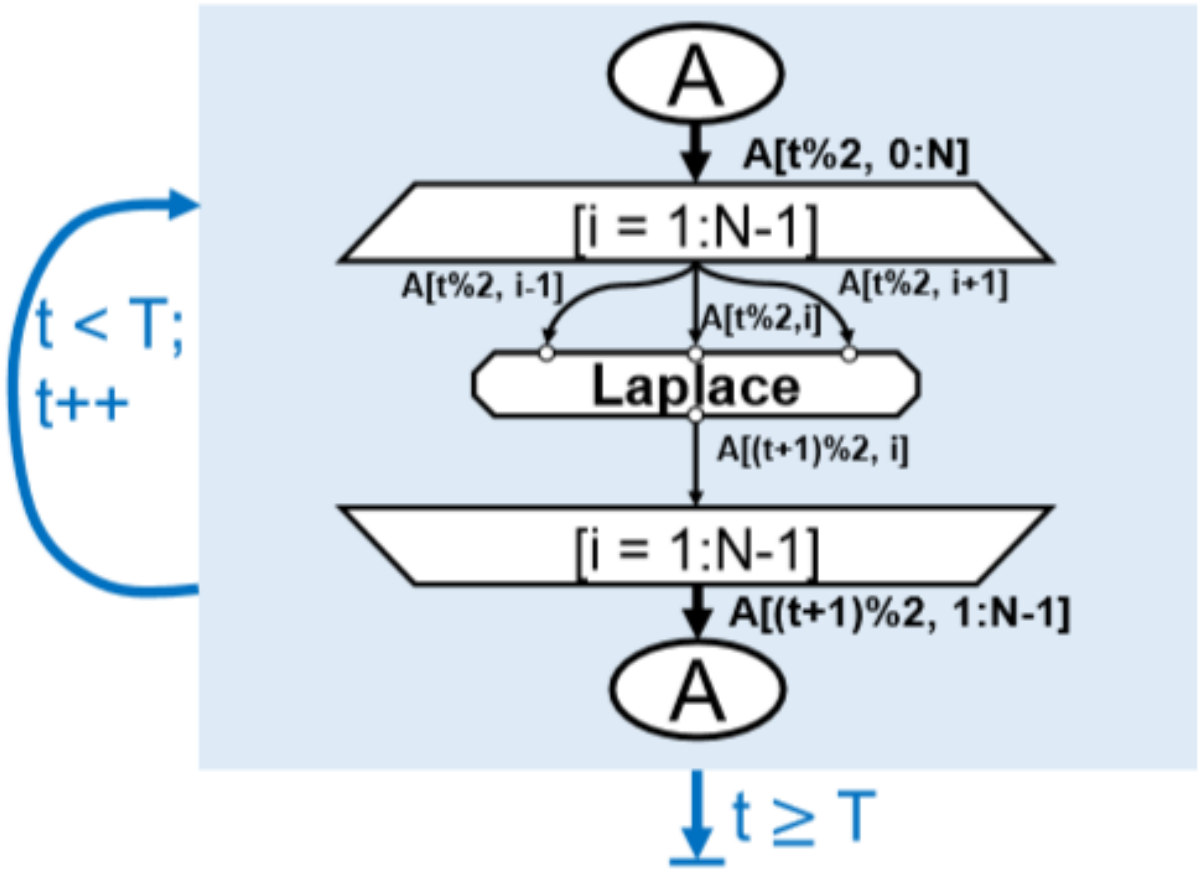
Optimizing Transformer Deep Neural Networks



Ivanov et al.: Data Movement Is All You Need: A Case Study on Optimizing Transformers, https://arxiv.org/abs/2007.00072

Back to our first example – Laplace in DaCe Python

```
@dace.program
def Laplace(A: dace.float64[2,N],
            T: dace.uint32):
  for t in range(T):
    for i in dace.map[1:N-1]:
      # Data dependencies
      in_l << A[t%2,      i-1]
      in_c << A[t%2,      i]
      in_r << A[t%2,      i+1]
      out  >> A[(t+1)%2,  i]
      # Computation
      out = in_l - 2*in_c + in_r
```

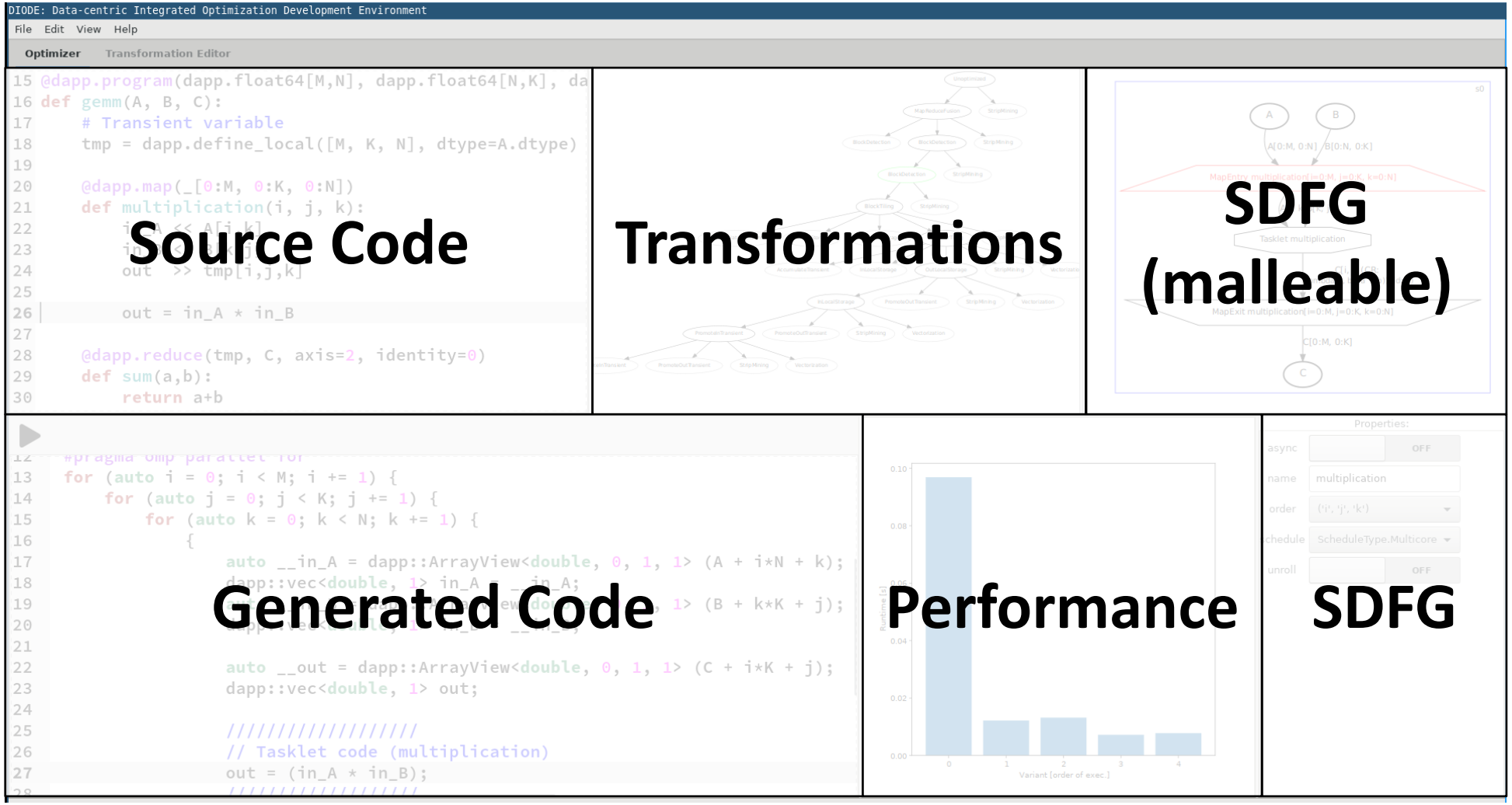



```
var << A(1, WCR)[0:N]
```

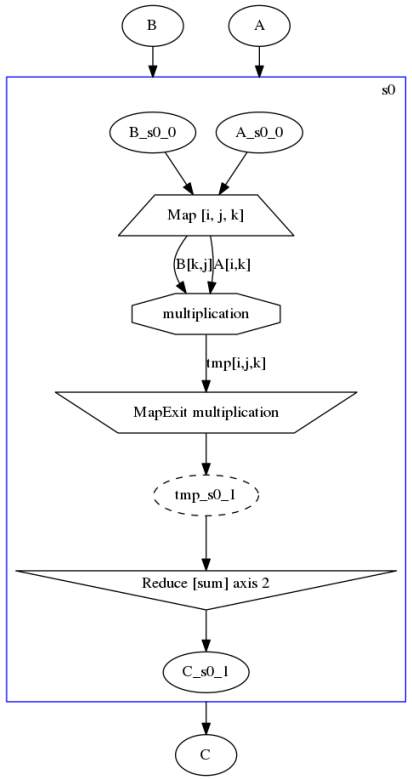
Local variable name
Direction (<<, >>)
Data
Number of accesses
Conflict Resolution
Location/Range

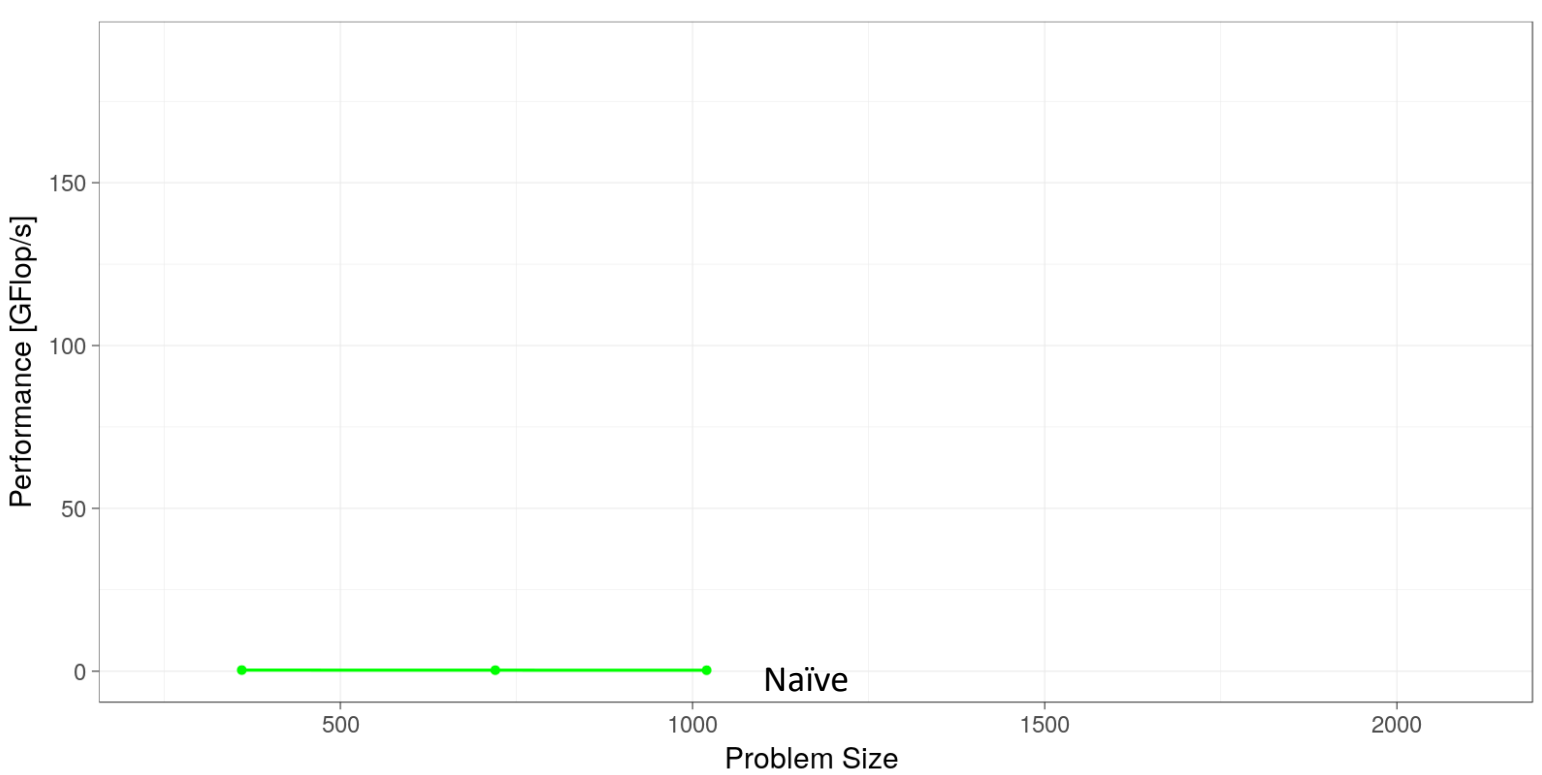
Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

DIODE User Interface (moving into vscode)



Performance for matrix multiplication on x86

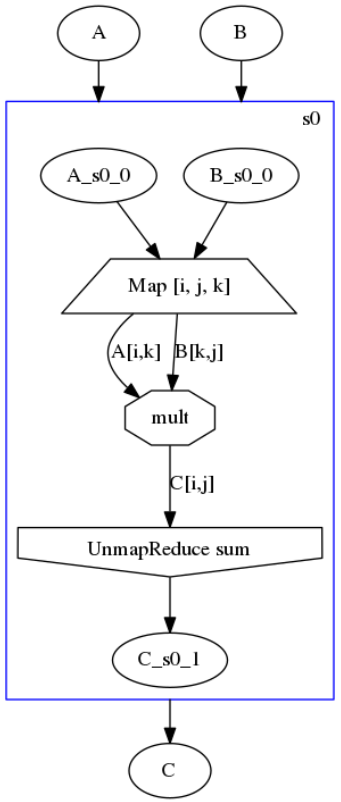




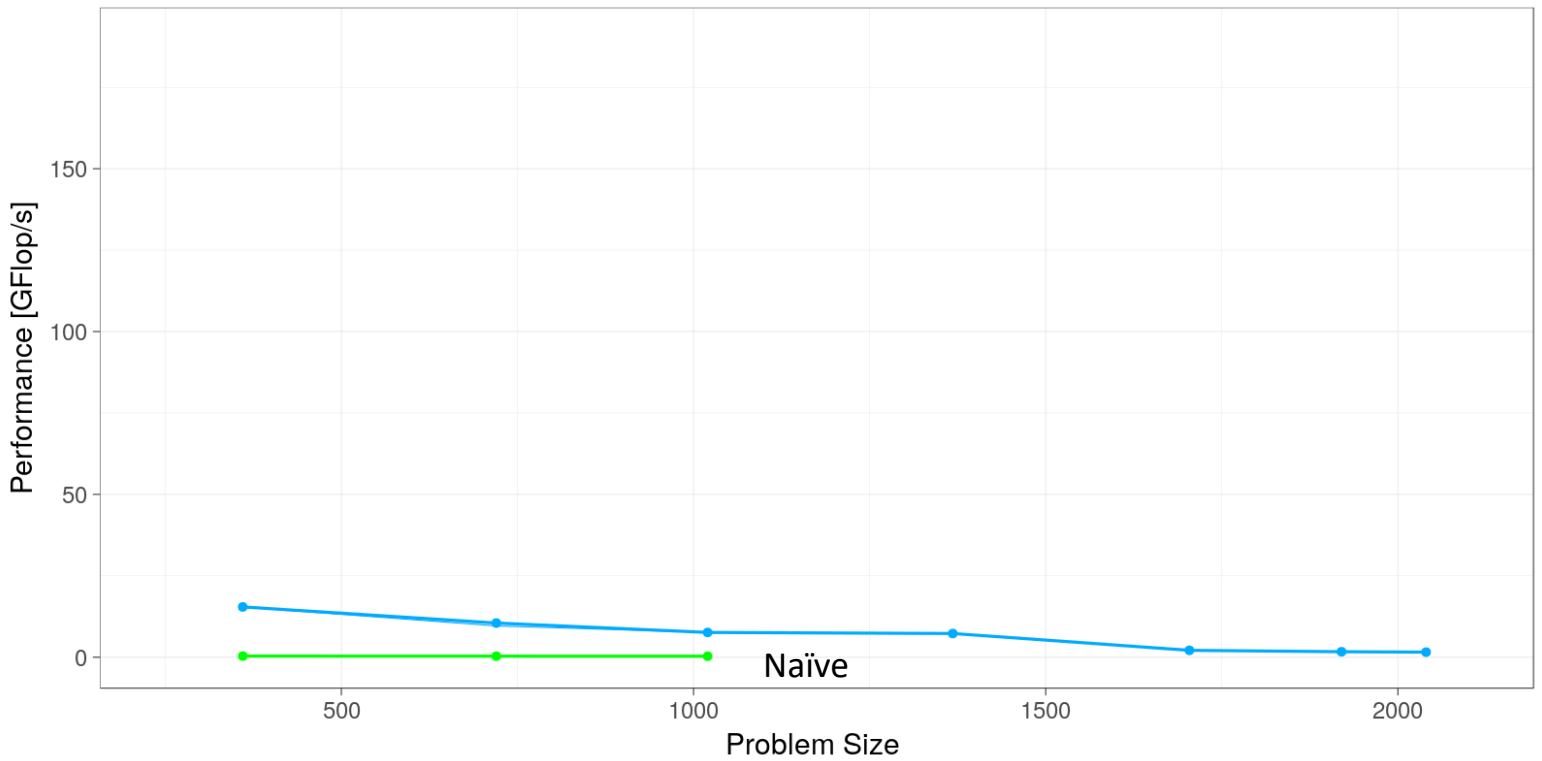
SDFG

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86



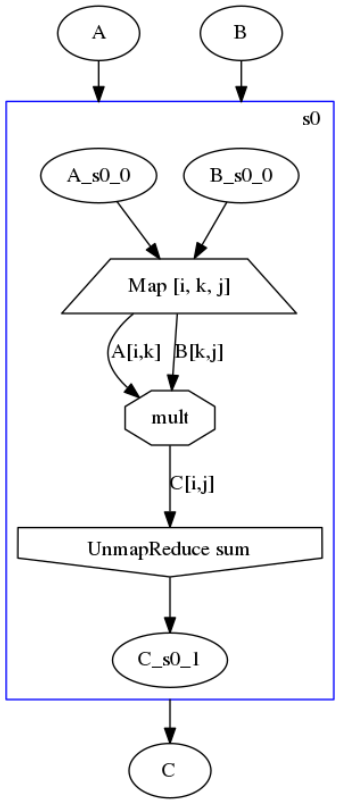
SDFG



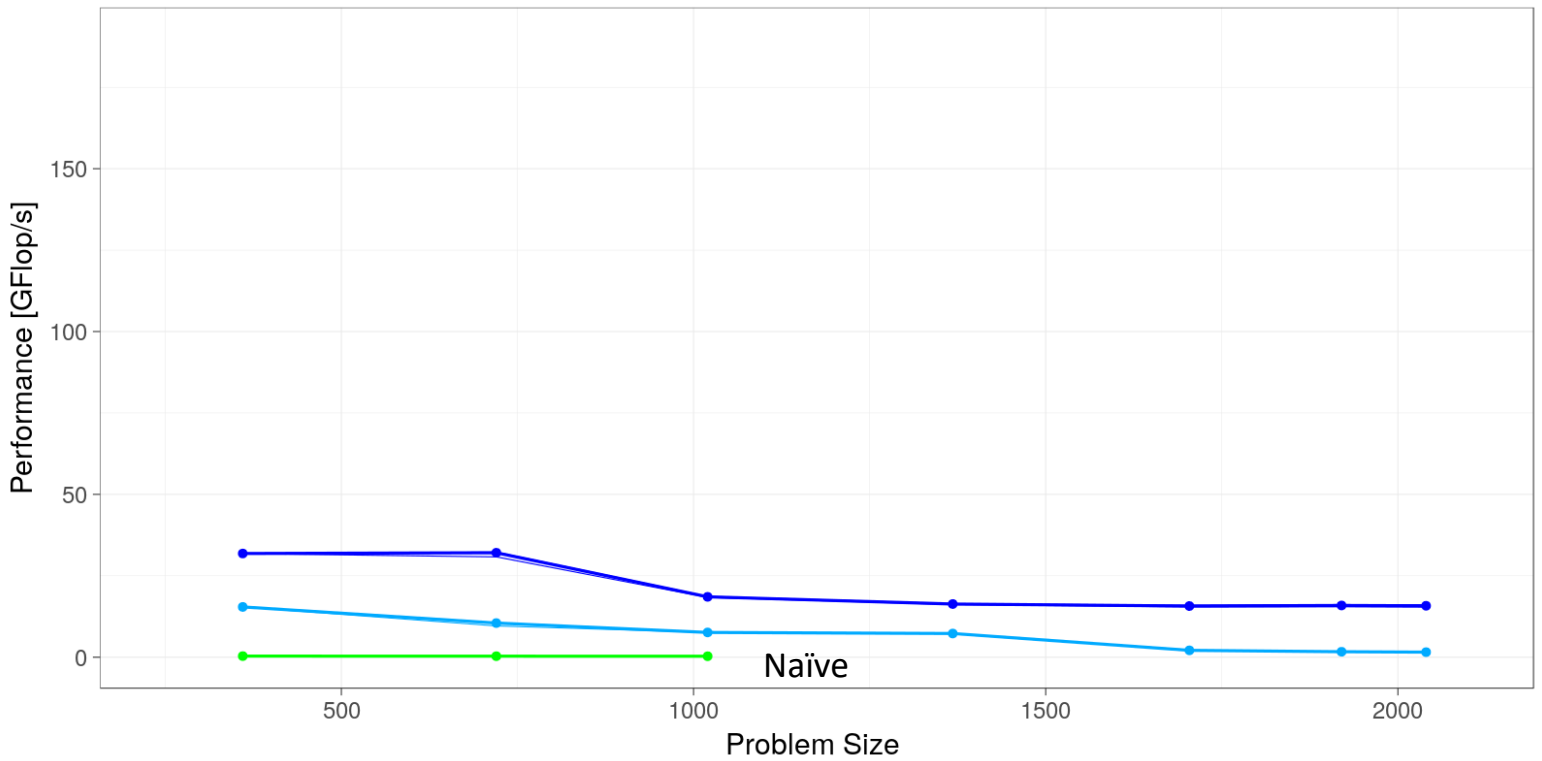
MapReduceFusion (27 SLOC)

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86



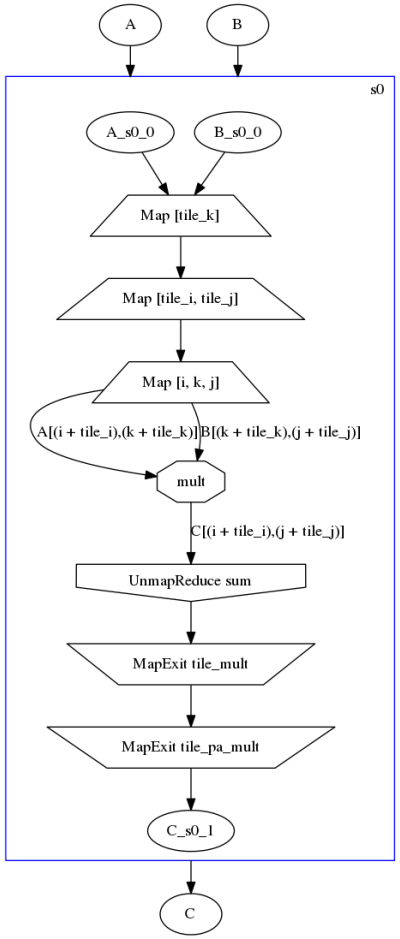
SDFG



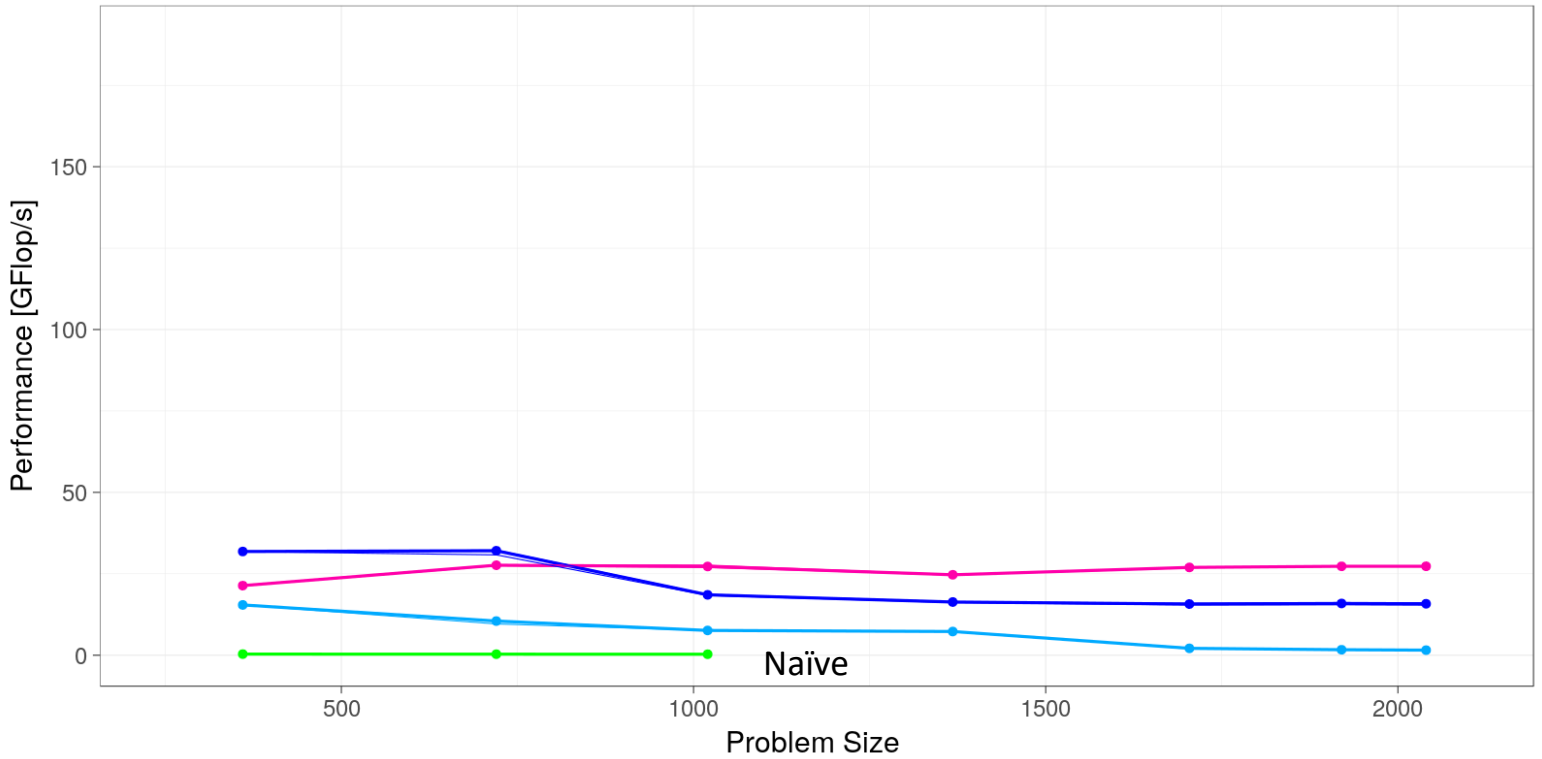
LoopReorder (27 SLOC)
MapReduceFusion (27 SLOC)

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86



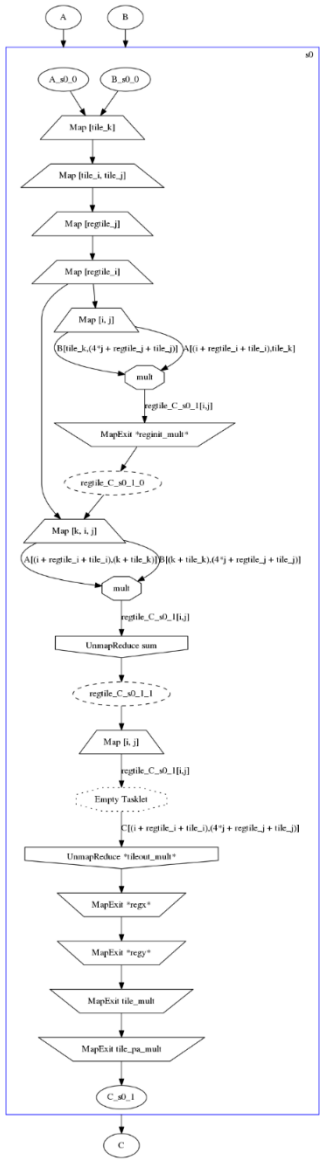
SDFG

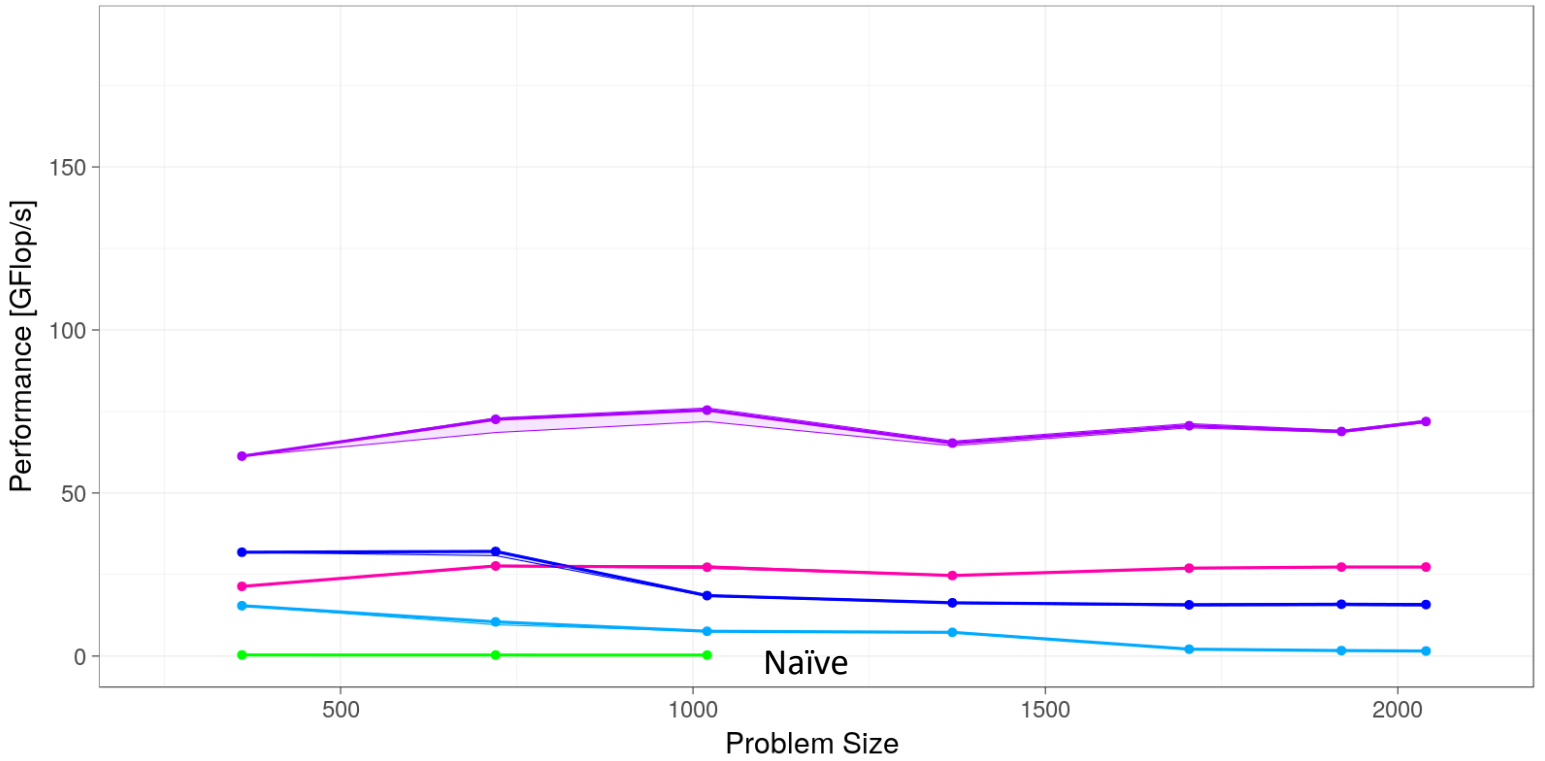


BlockTiling (39 SLOC)
LoopReorder (27 SLOC)
MapReduceFusion (27 SLOC)

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86



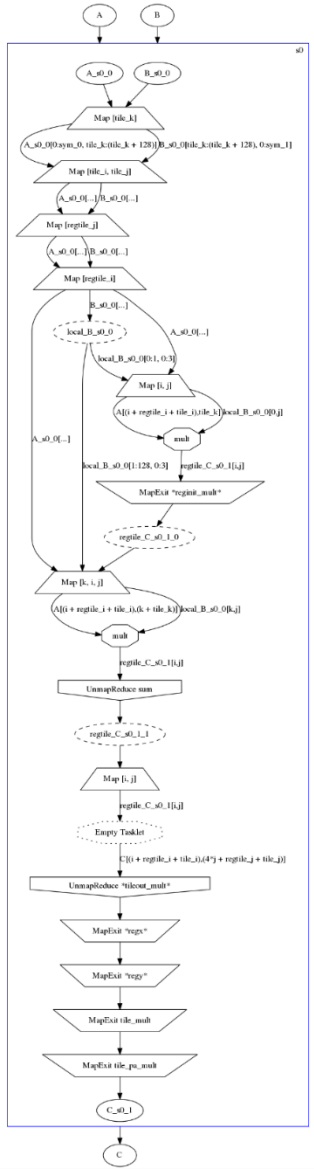


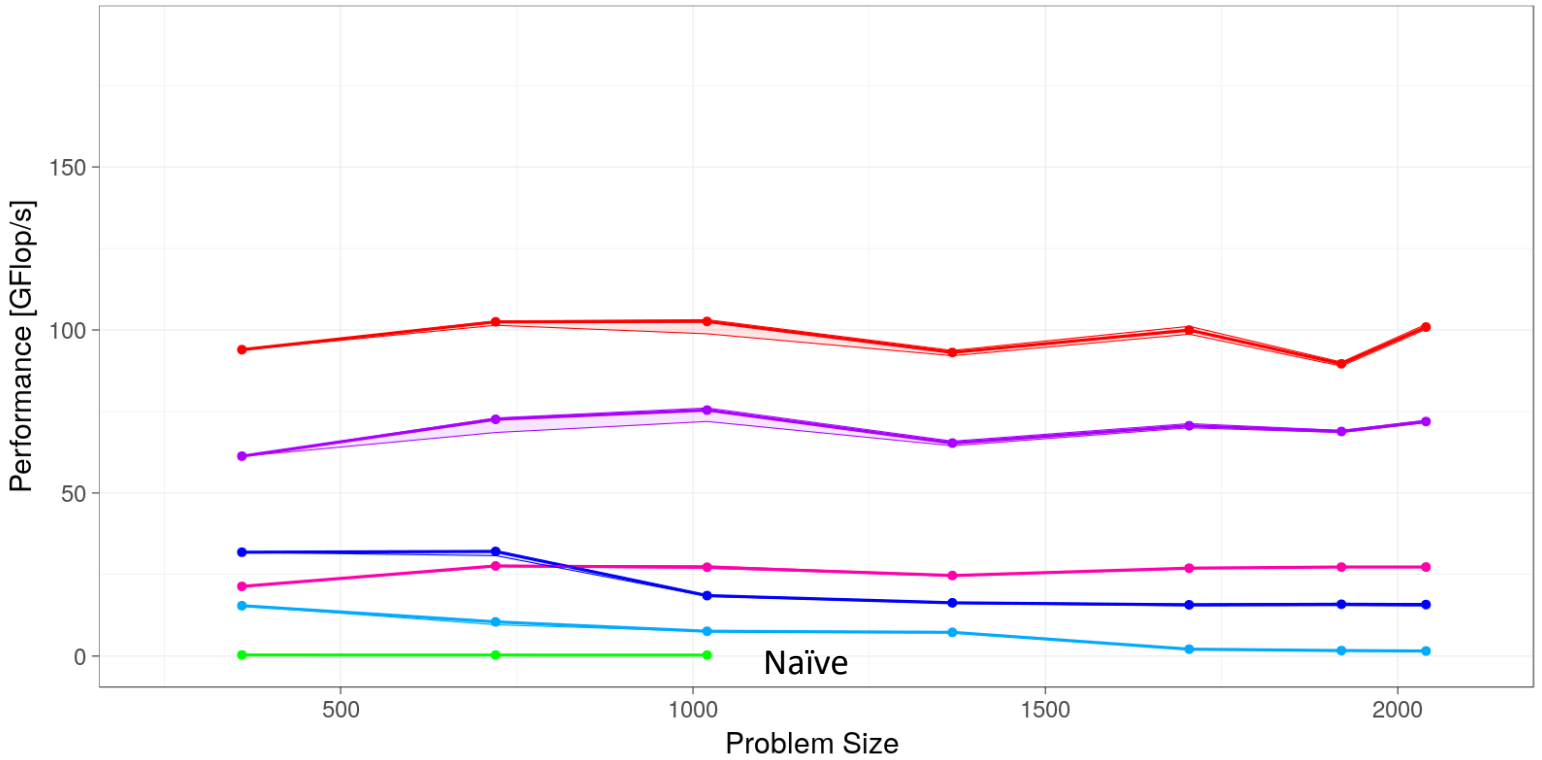
RegisterTiling (47 SLOC)

BlockTiling (39 SLOC)
LoopReorder (27 SLOC)
MapReduceFusion (27 SLOC)

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86





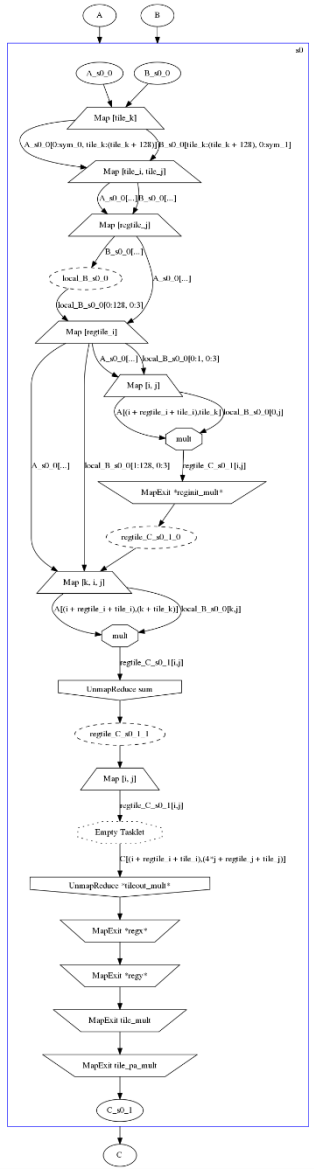
LocalStorage (50 SLOC)

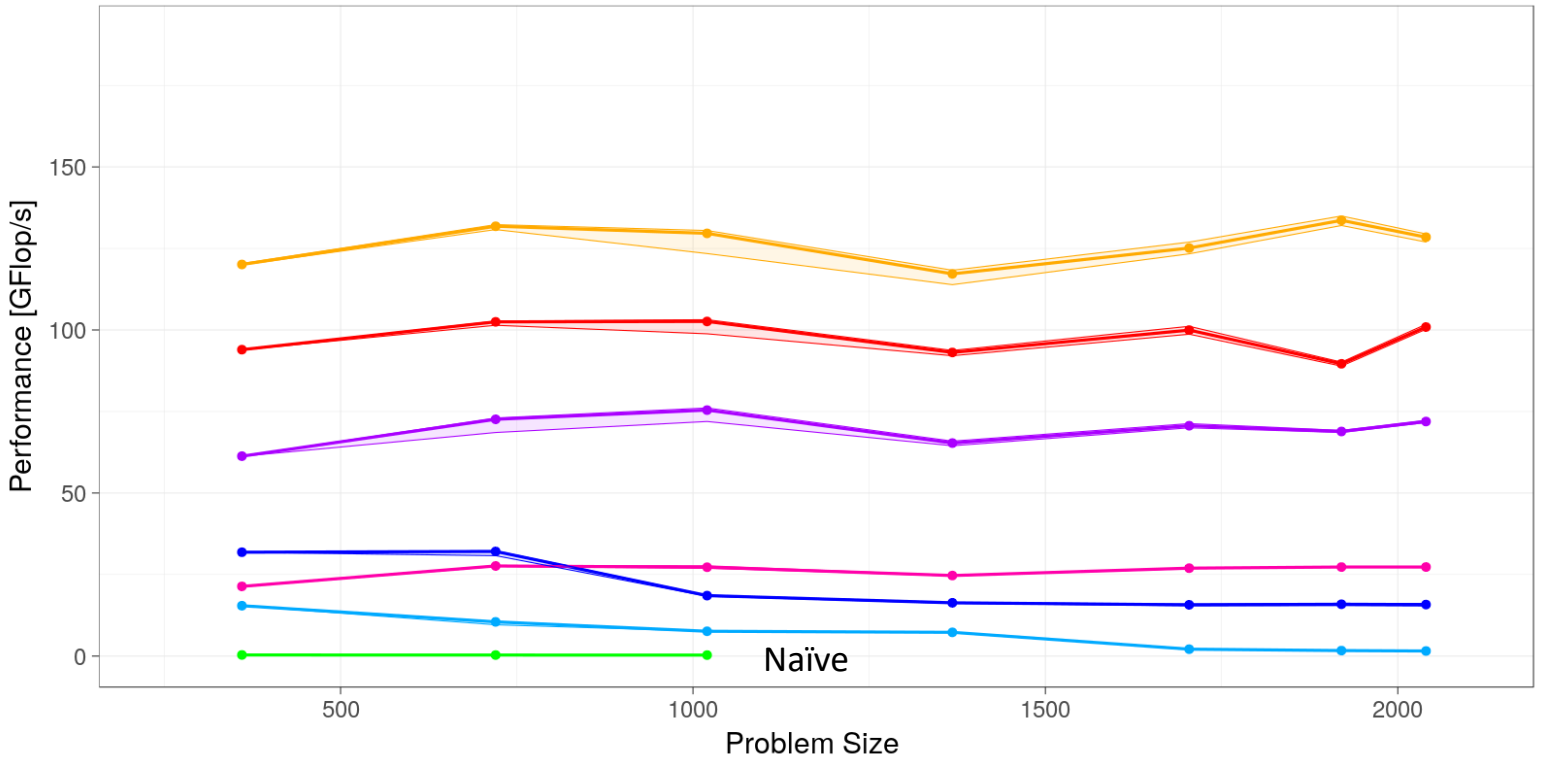
RegisterTiling (47 SLOC)

BlockTiling (39 SLOC)
LoopReorder (27 SLOC)
MapReduceFusion (27 SLOC)

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86





PromoteTransient (51 SLOC)

LocalStorage (50 SLOC)

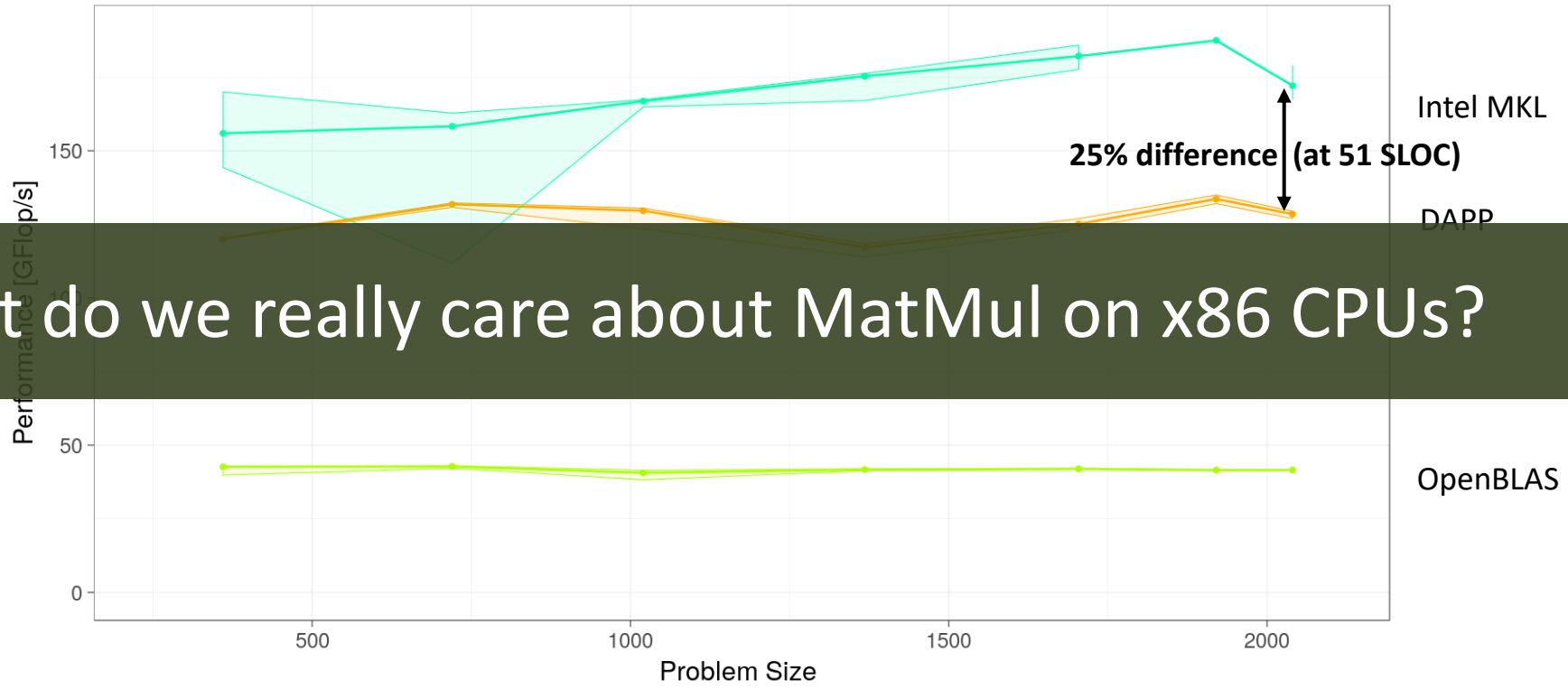
RegisterTiling (47 SLOC)

BlockTiling (39 SLOC)
LoopReorder (27 SLOC)
MapReduceFusion (27 SLOC)

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance for matrix multiplication on x86

With more tuning: 98.6% of MKL for specific inputs (587 SLOC)

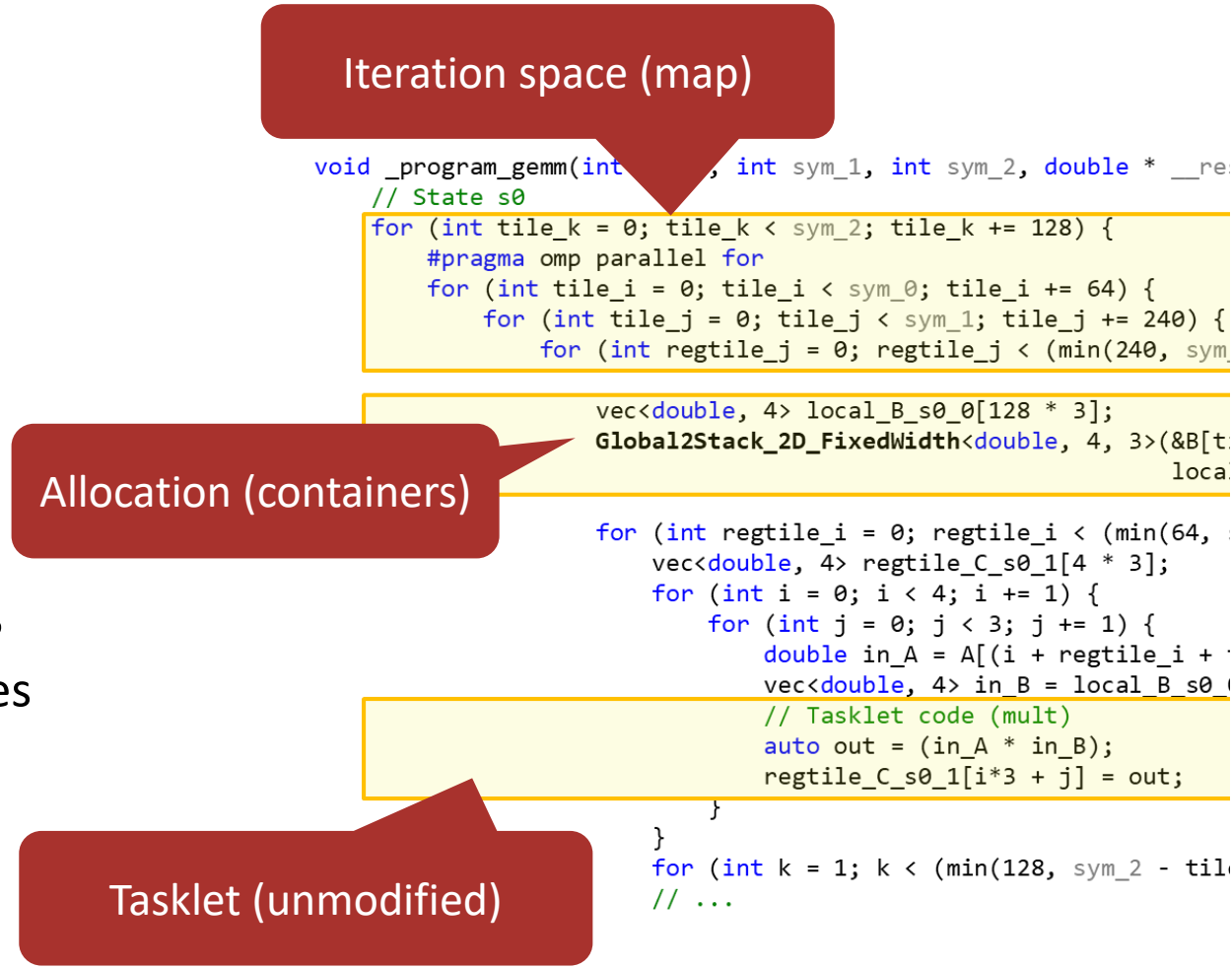


But do we really care about MatMul on x86 CPUs?

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

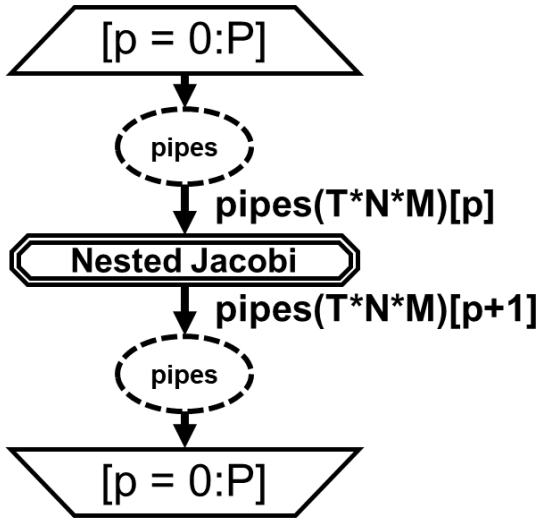
Code Generation for Load/Store Architectures

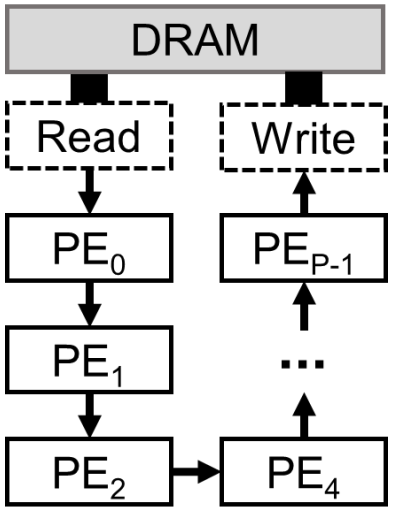
- **Recursive code generation (C++, CUDA)**
  - **Control flow:** Construct detection and gotos
- **Parallelism**
  - **Multi-core CPU**: OpenMP, atomics, and threads
  - **GPU**: CUDA kernels and streams
  - Connected components run concurrently
- **Memory and interaction with accelerators**
  - Array-array edges create intra-/inter-device copies
  - Memory access validation on compilation
  - Automatic CPU SDFG to GPU transformation
- **Tasklet code immutable**



Code Generation for Pipelined Architectures

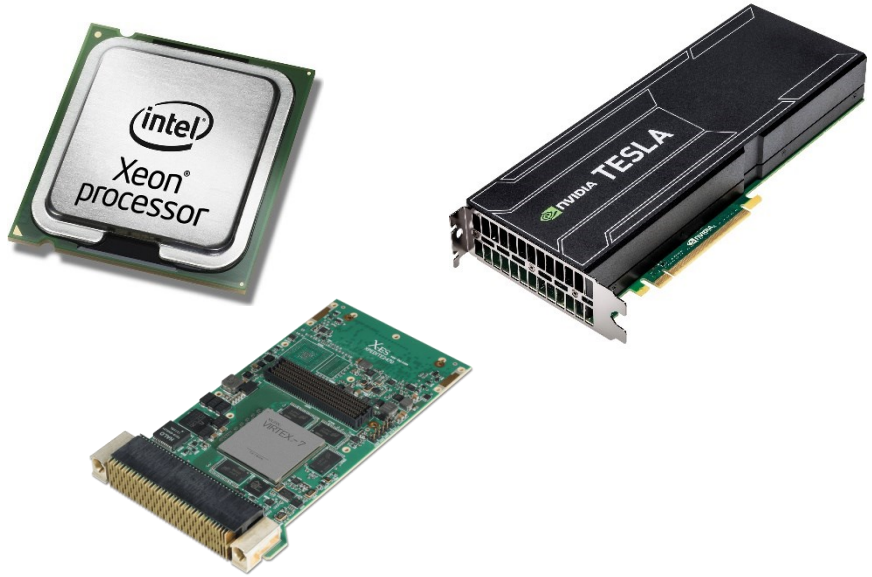
- **Module generation with HDL and HLS**
  - Integration with Xilinx SDAccel or OpenCL (RTL in development)
  - Nested SDFGs become FPGA state machines
- **Parallelism**
  - Exploiting temporal locality: Pipelines
  - Exploiting spatial locality: Vectorization, replication
- **Replication**
  - Enables parametric systolic array generation
- **Memory access**
  - Burst memory access, vectorization
  - Streams for inter-PE communication





Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance (Portability) Evaluation



- **Three platforms:**
  - Intel Xeon E5-2650 v4 CPU (2.20 GHz, no HT)
  - Tesla P100 GPU
  - Xilinx VCU1525 hosting an XCVU9P FPGA

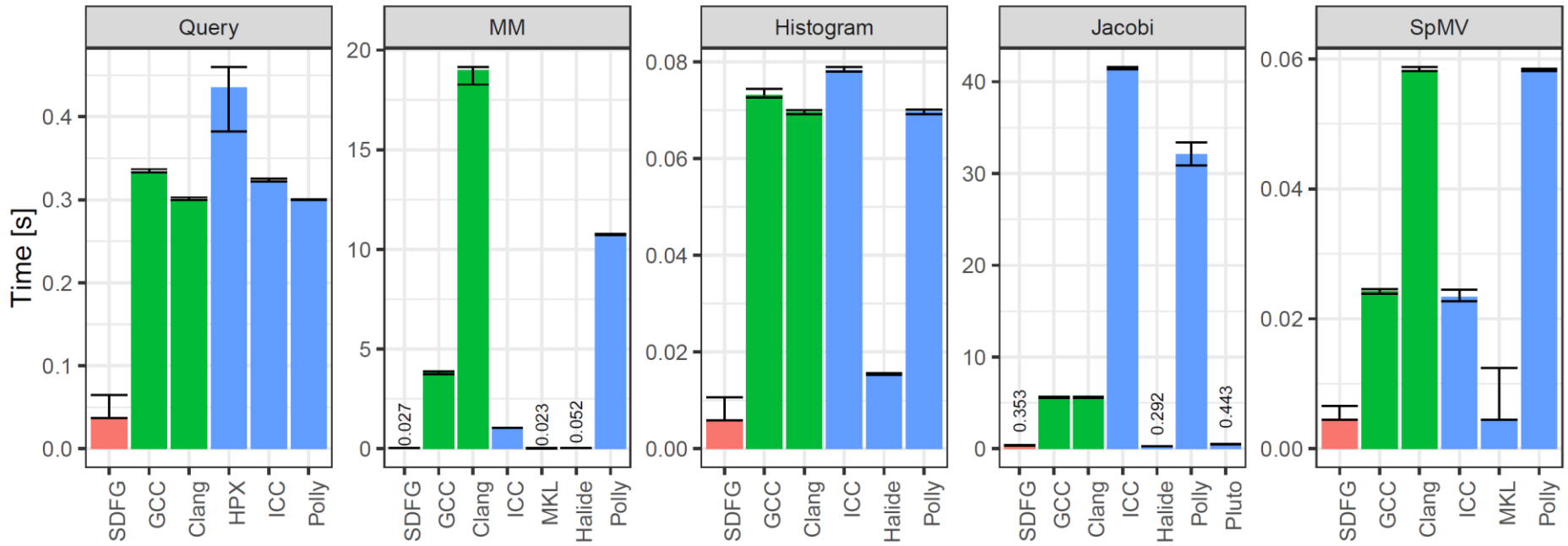
- **Compilers and frameworks:**
  - Compilers:
    GCC 8.2.0
    Clang 6.0
    icc 18.0.3
  - Polyhedral optimizing compilers:
    Polly 6.0
    Pluto 0.11.4
    PPCG 0.8
  - GPU and FPGA compilers:
    CUDA nvcc 9.2
    Xilinx SDAccel 2018.2
  - Frameworks and optimized libraries:
    HPX
    Halide
    Intel MKL
    NVIDIA CUBLAS, CUSPARSE, CUTLASS
    NVIDIA CUB

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance Evaluation: Fundamental Kernels (CPU)

- **Database Query**: roughly 50% of a 67,108,864 column
- **Matrix Multiplication (MM)**: 2048x2048x2048
- **Histogram**: 8192x8192
- **Jacobi stencil**: 2048x2048 for T=1024
- **Sparse Matrix-Vector Multiplication (SpMV)**: 8192x8192 CSR matrix (nnz=33,554,432)

8.12x faster | 98.6% of MKL | 2.5x faster | 82.7% of Halide | 99.9% of MKL

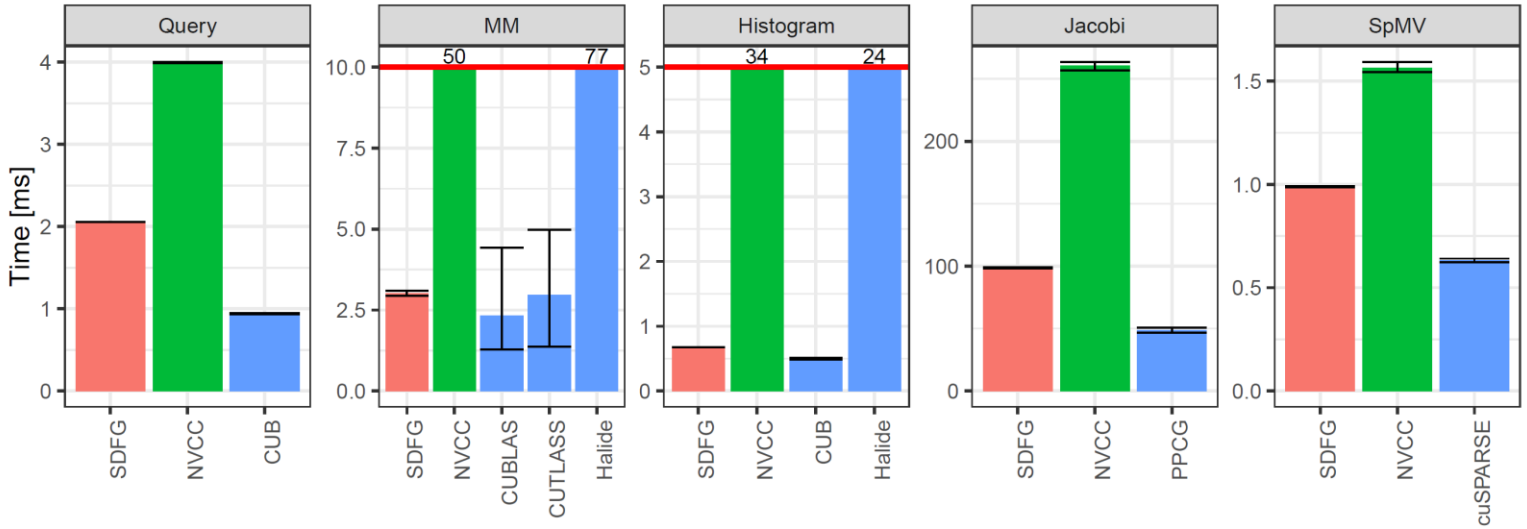


Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance Evaluation: Fundamental Kernels (GPU, FPGA)

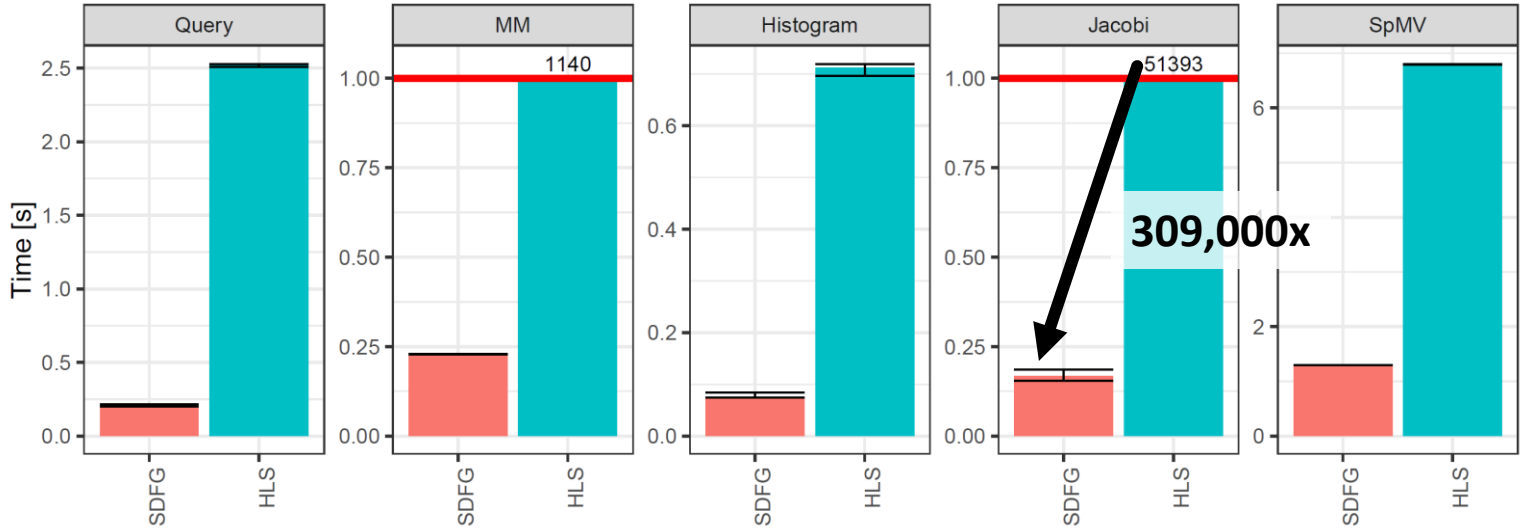
GPU

90% of CUTLASS



FPGA

19.5x of Spatial

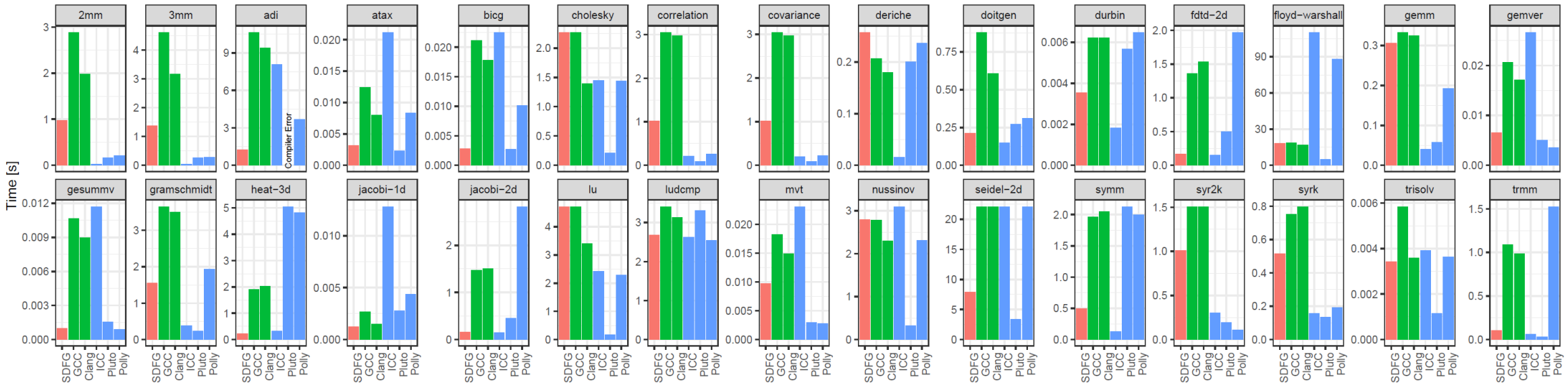


Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Performance Evaluation: Polybench (CPU)

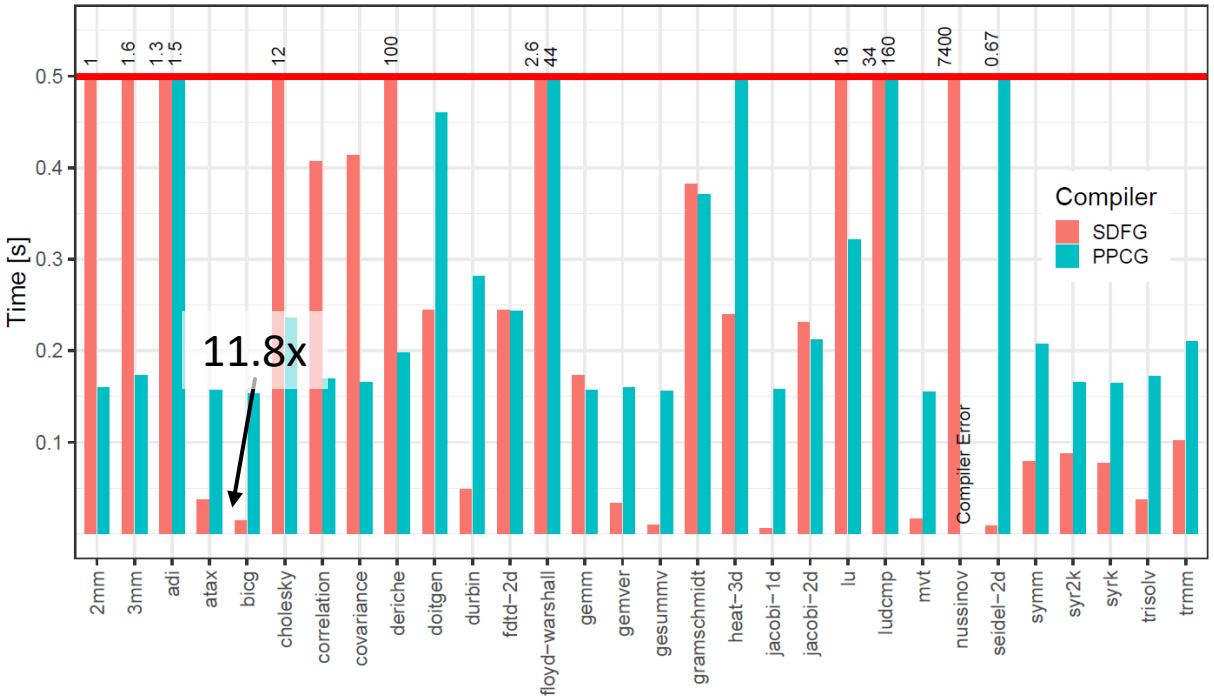
- Polyhedral benchmark with 30 applications

- Without any transformations, achieves 1.43x (geometric mean) over general-purpose compilers



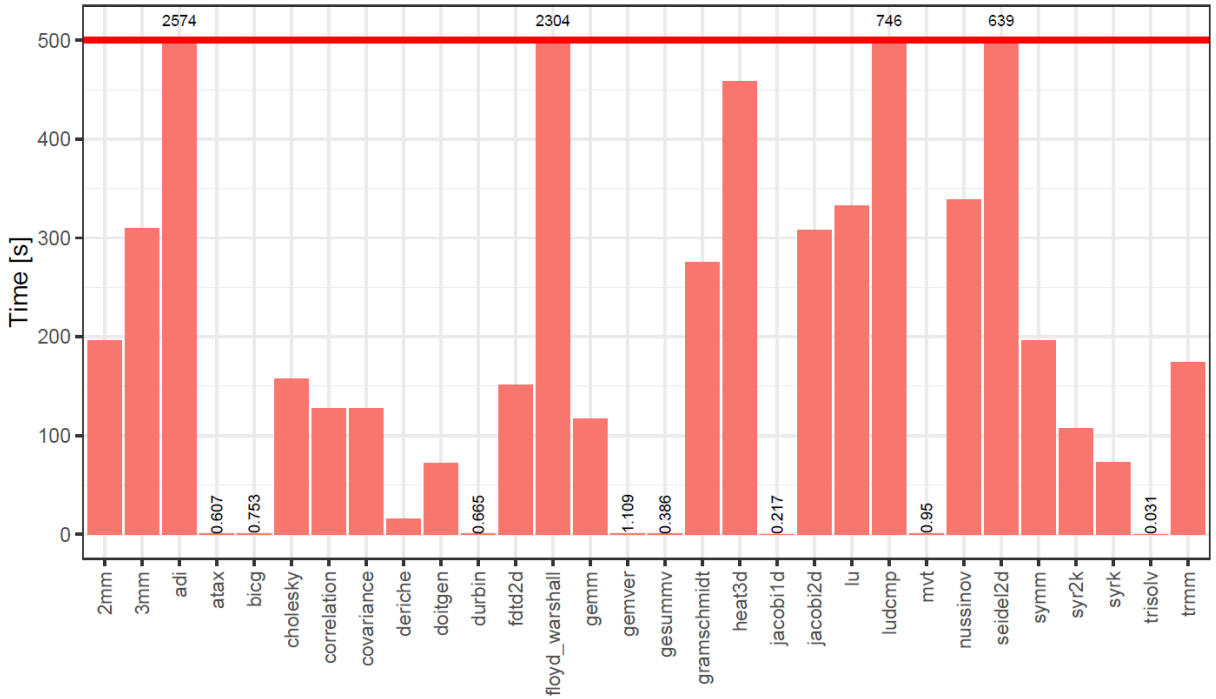
Performance Evaluation: Polybench (GPU, FPGA)

- **Automatically transformed from CPU code**



GPU

(1.12x geomean speedup)



FPGA

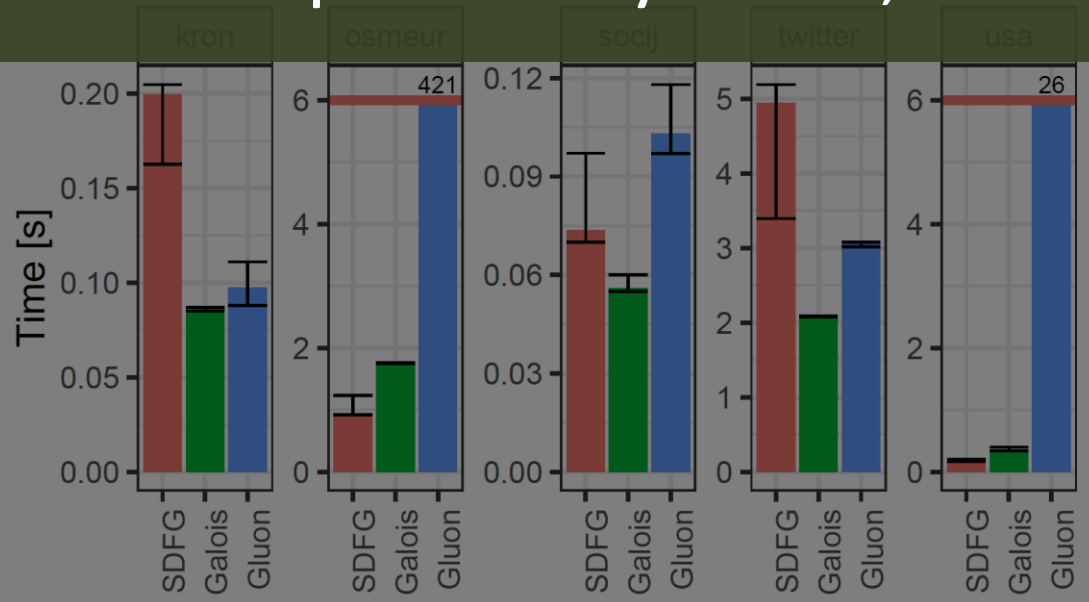
The **first** full set of placed-and-routed Polybench

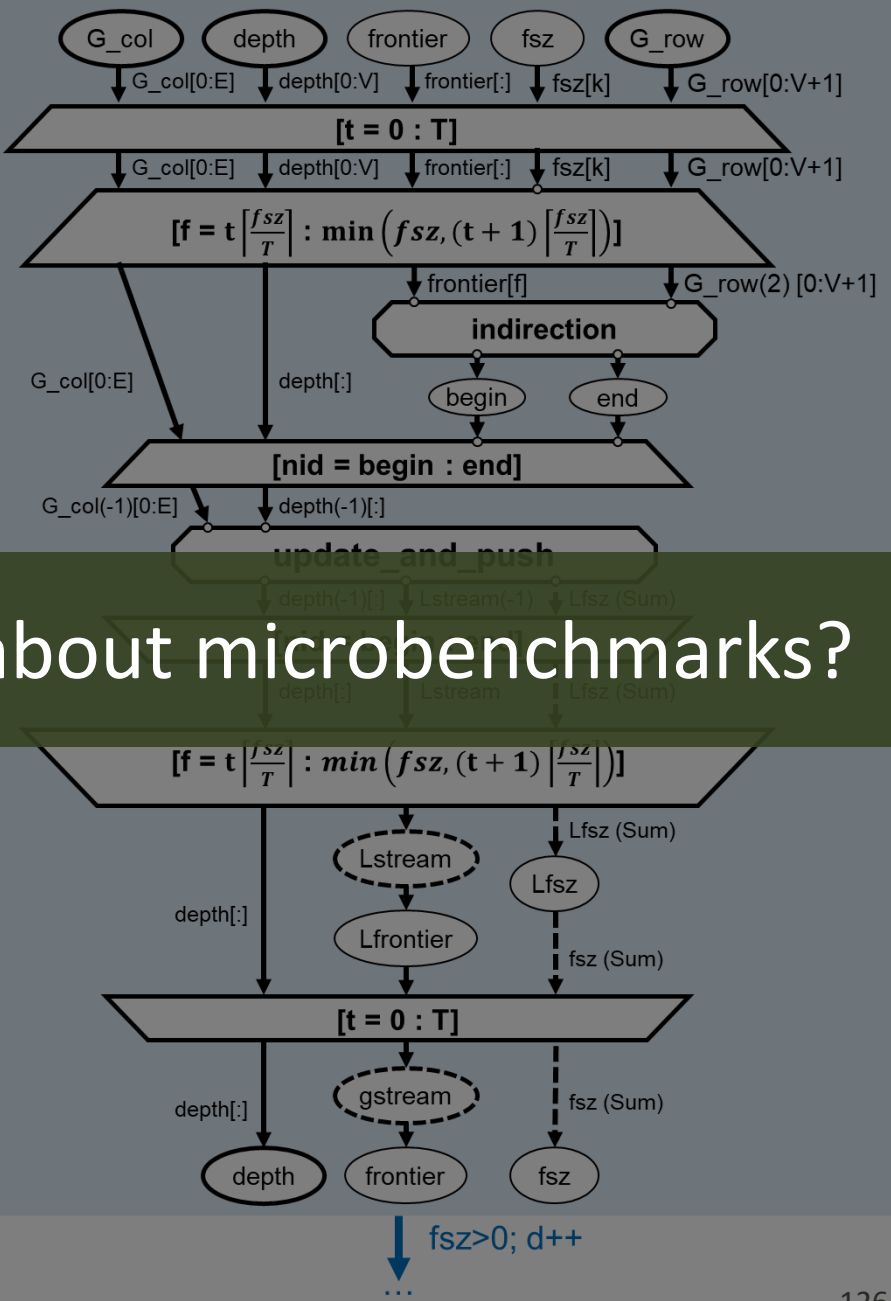
Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Case Study: Parallel Breadth-First Search

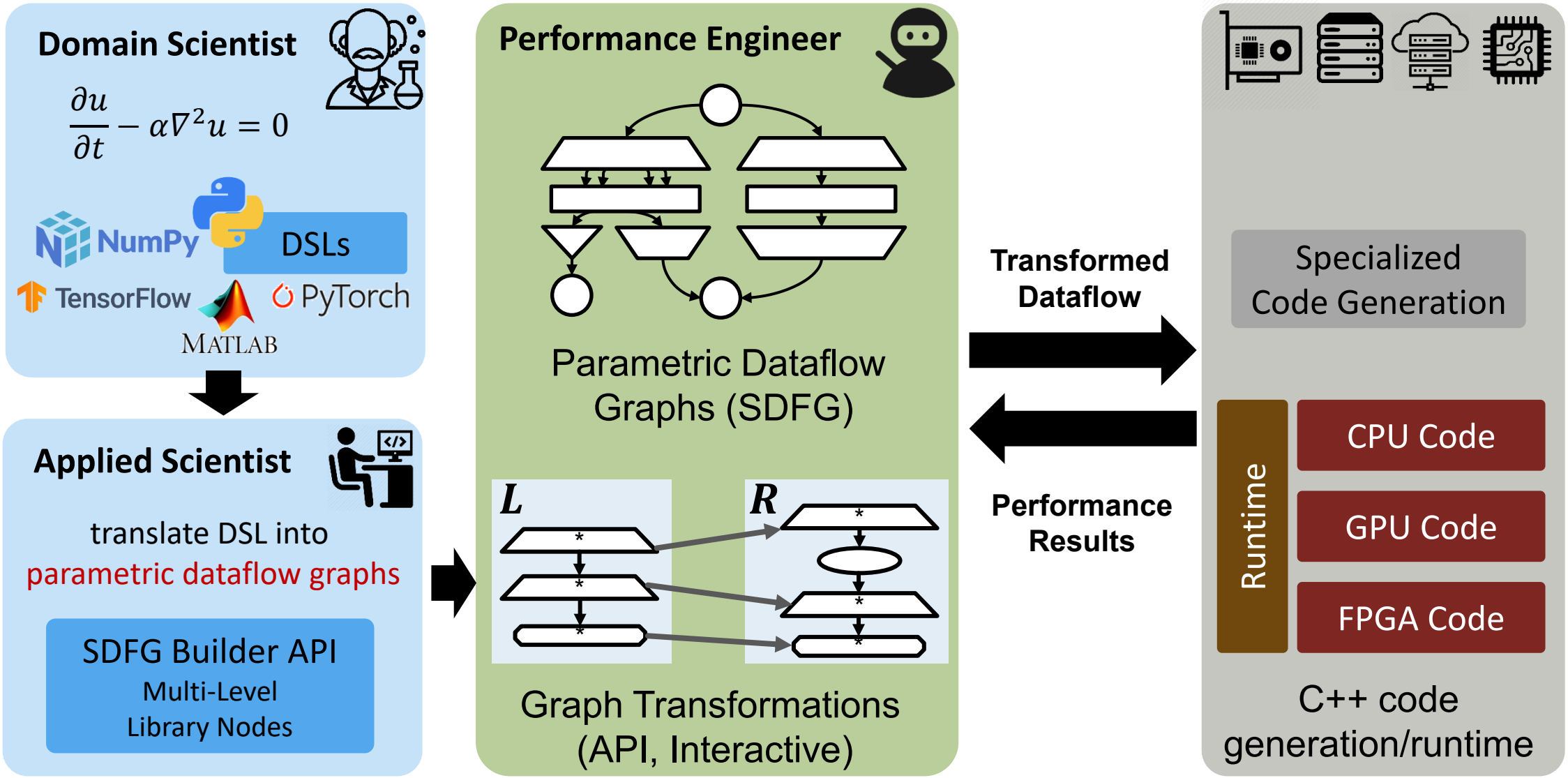
- **Compared with Galois and Gluon**
  - State-of-the-art graph processing frameworks on CPU
- **Graphs:**
  - Road maps: USA, OSM-Europe
  - Social networks: Twitter, LiveJournal

Performance portability – fine, but who cares about microbenchmarks?



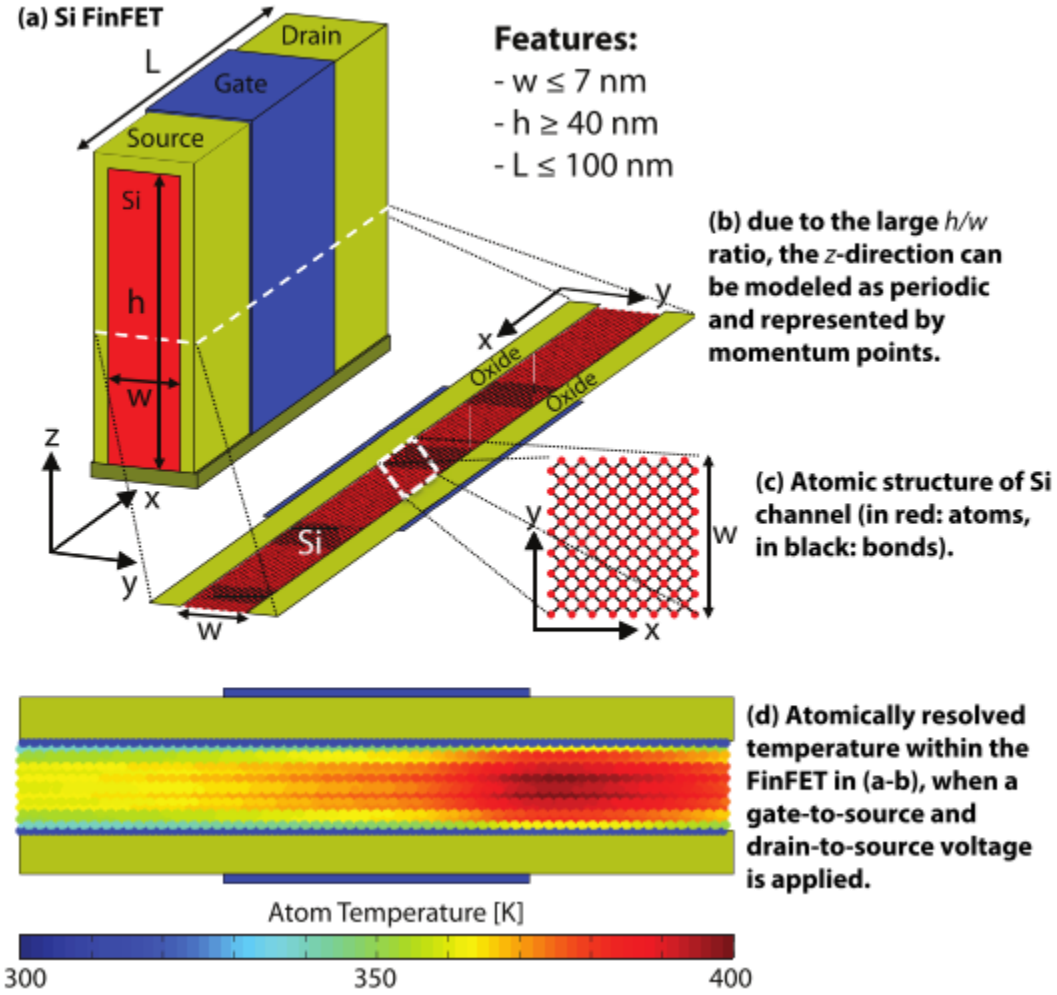


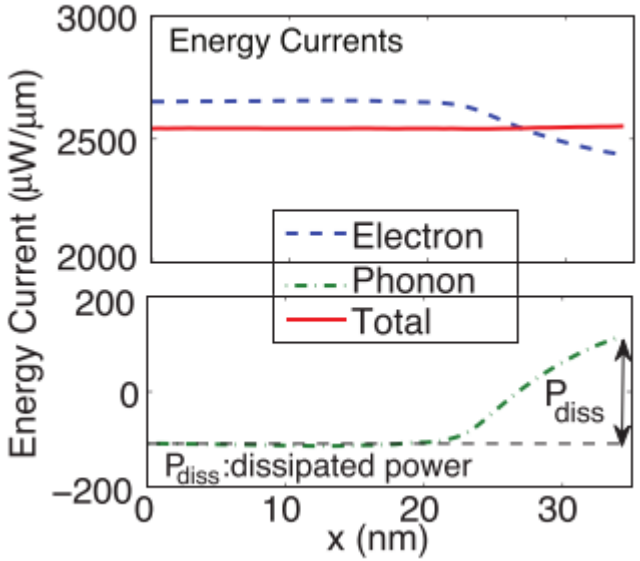
Remember: Scientific Software Engineering in the 21st century

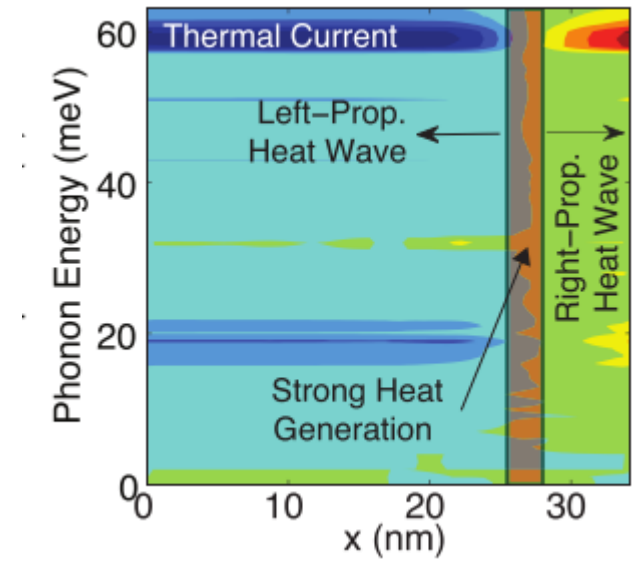


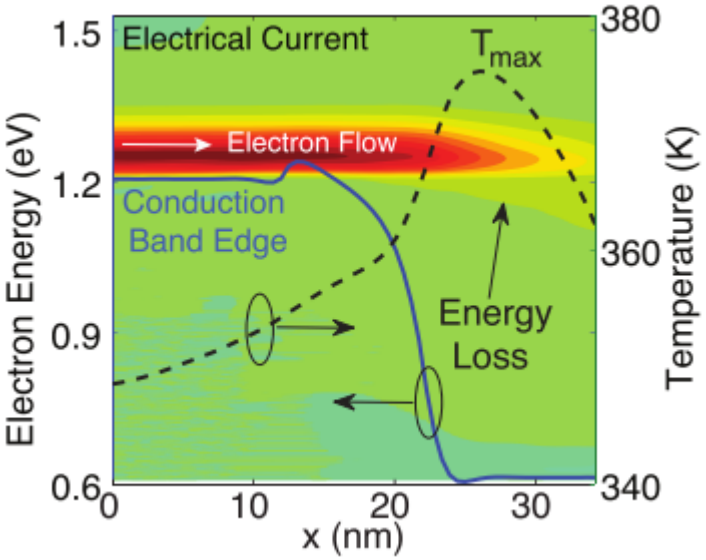
Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19

Next-Generation Transistors need to be cooler – addressing self-heating





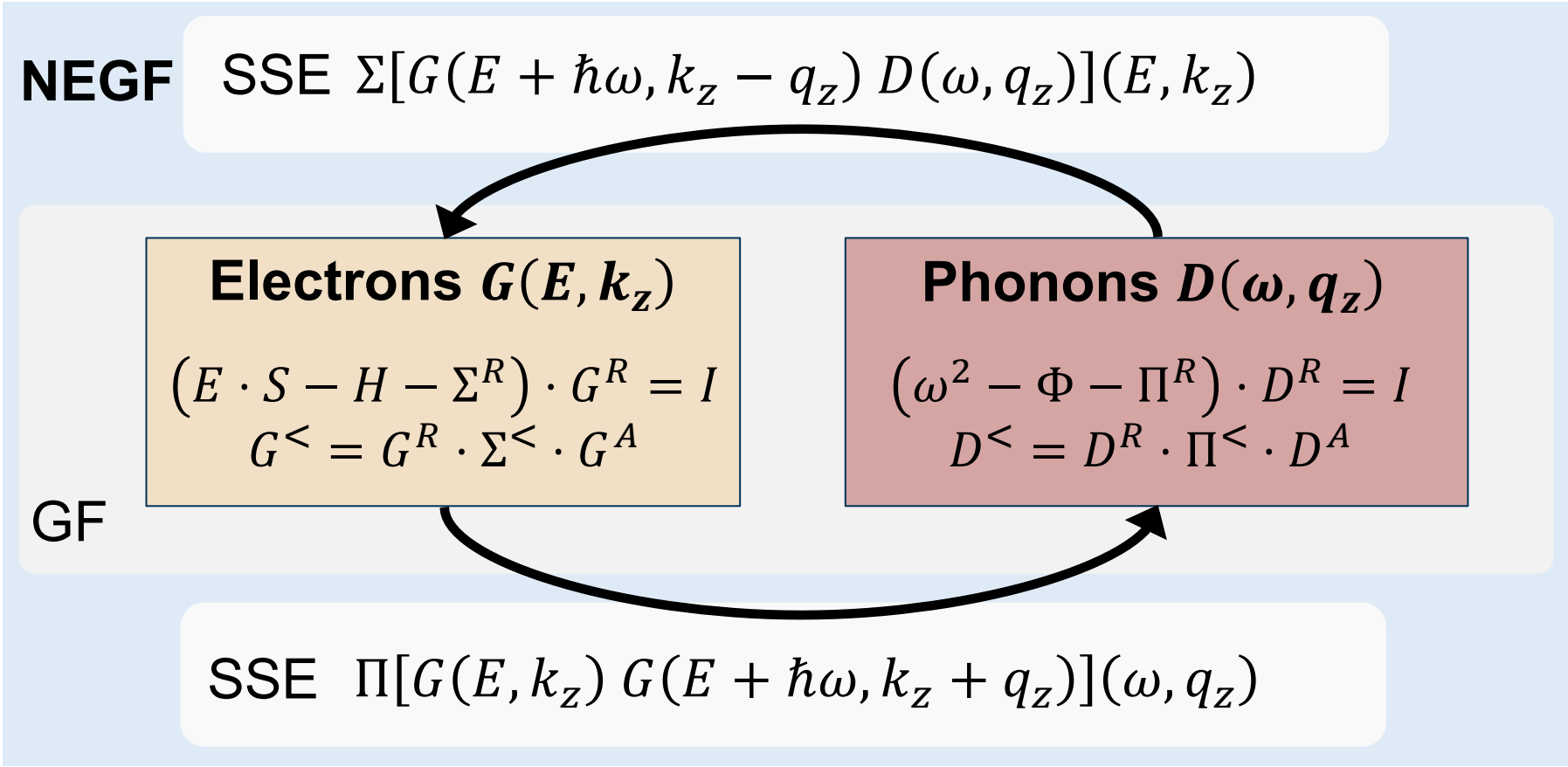




Ziogas et al. "A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations", SC19 (Gordon Bell Prize 2019).

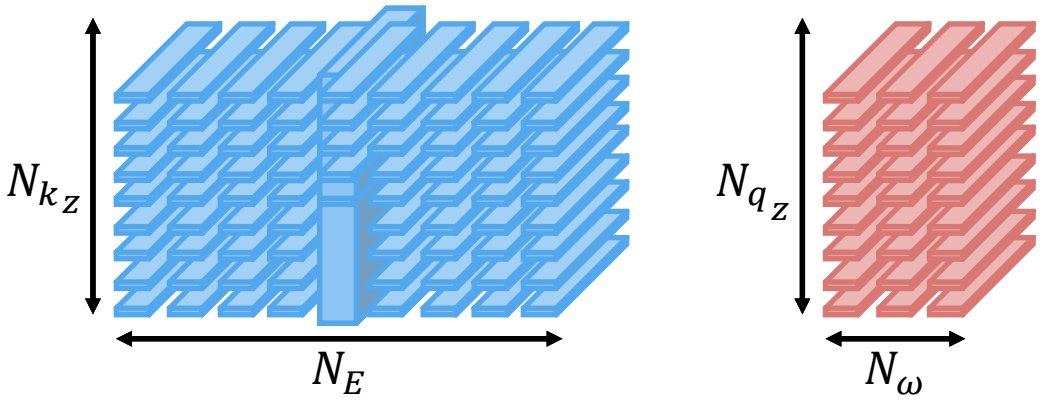
Quantum Transport Simulations with OMEN

- **OMEN Code (Luisier et al., Gordon Bell award finalist 2011 and 2015)**
  - 90k SLOC, C, C++, CUDA, MPI, OpenMP, …



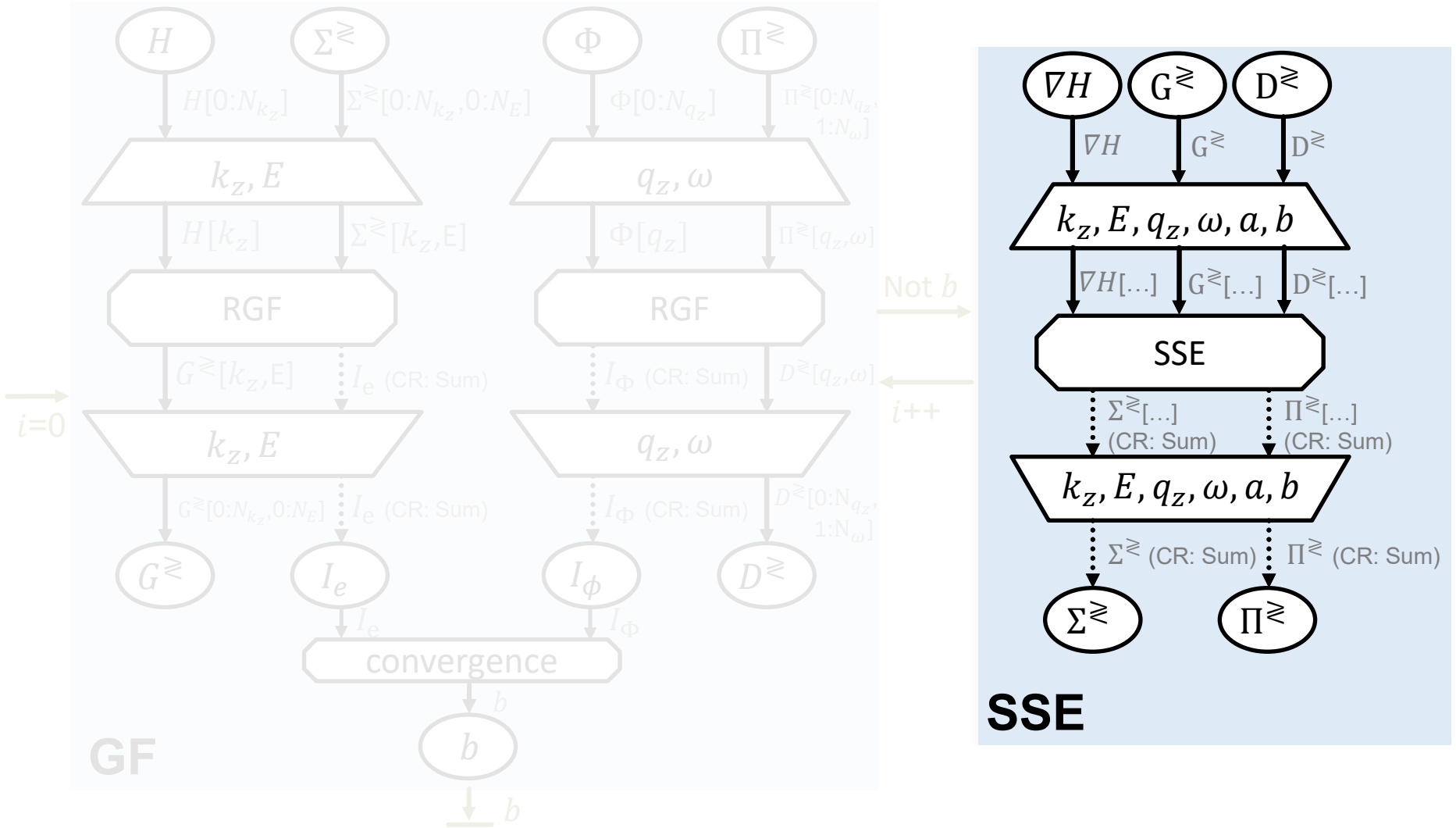
Ziogas et al. “A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations”, SC19 (Gordon Bell Prize 2019).

OMEN

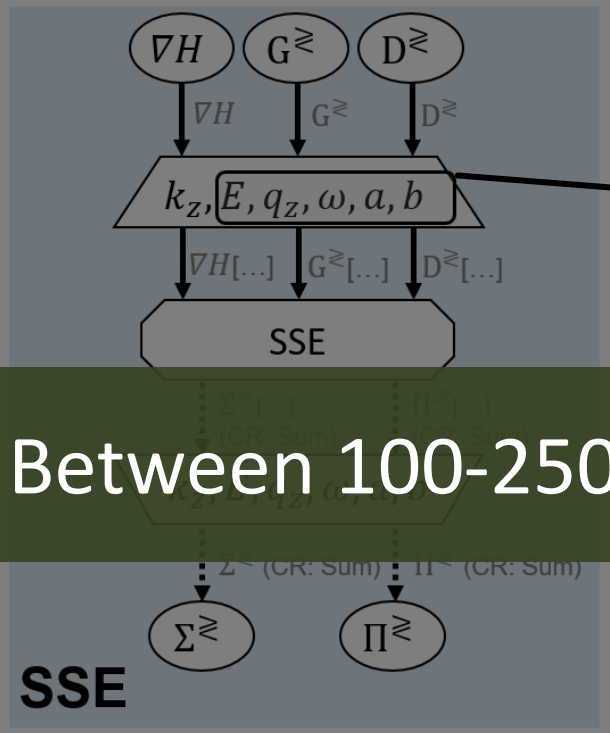


Ziogas et al. “A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations”, SC19 (Gordon Bell Prize 2019).

All of OMEN (90k SLOC) in a single SDFG – (collapsed) tasklets contain more SDFGs



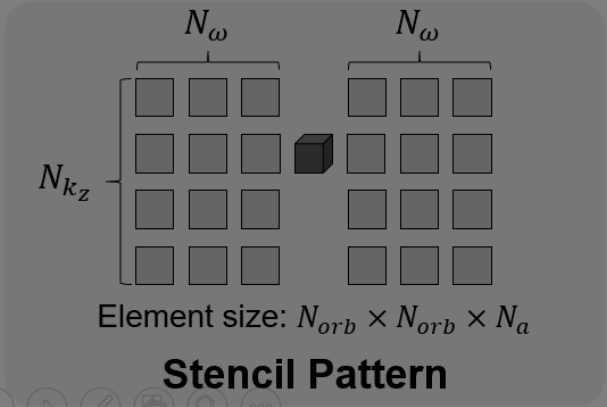
Zooming into SSE (large share of the runtime)



Natural Loop Tiling

DaCe Transform

Between 100-250x less communication at scale! (from PB to TB)



Additional interesting performance insights

Python is slow! Ok, we knew that – but compiled can be fast!

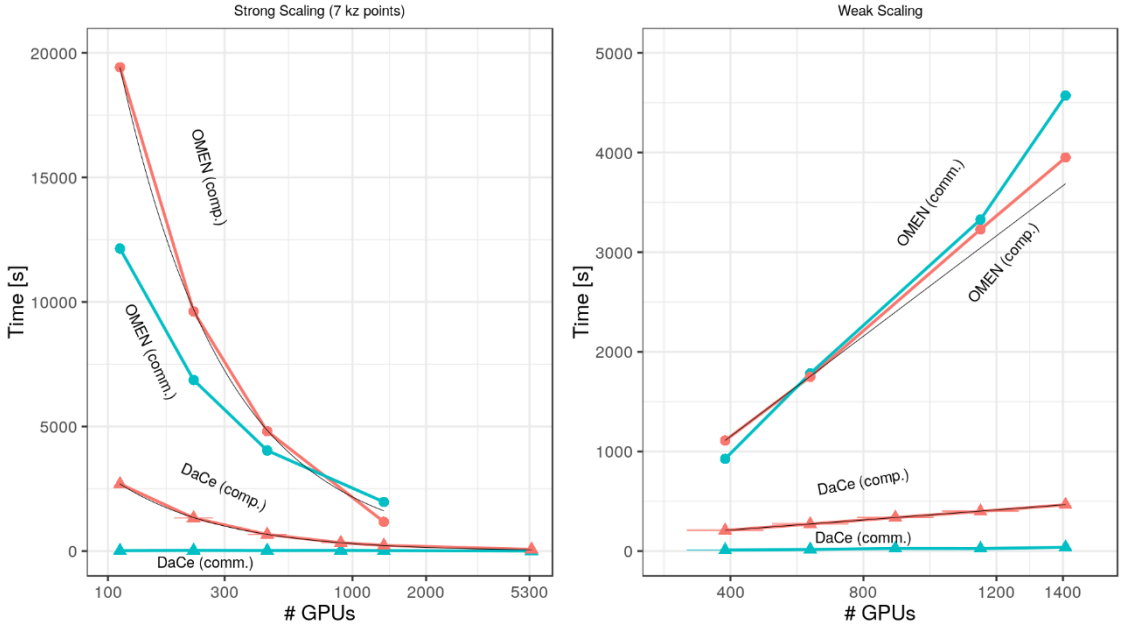
Variant	Phase					
	GF			SSE		
	Tflop	Time [s]	% Peak	Tflop	Time [s]	% Peak
OMEN	174.0	144.14	23.2%	63.6	965.45	1.3%
Python	174.0	1,342.77	2.5%	63.6	30,560.13	0.2%
DaCe	174.0	**111.25**	**30.1%**	31.8	**29.93**	**20.4%**

Piz Daint single node (P100)

cuBLAS can be very inefficient (well, unless you floptimize)

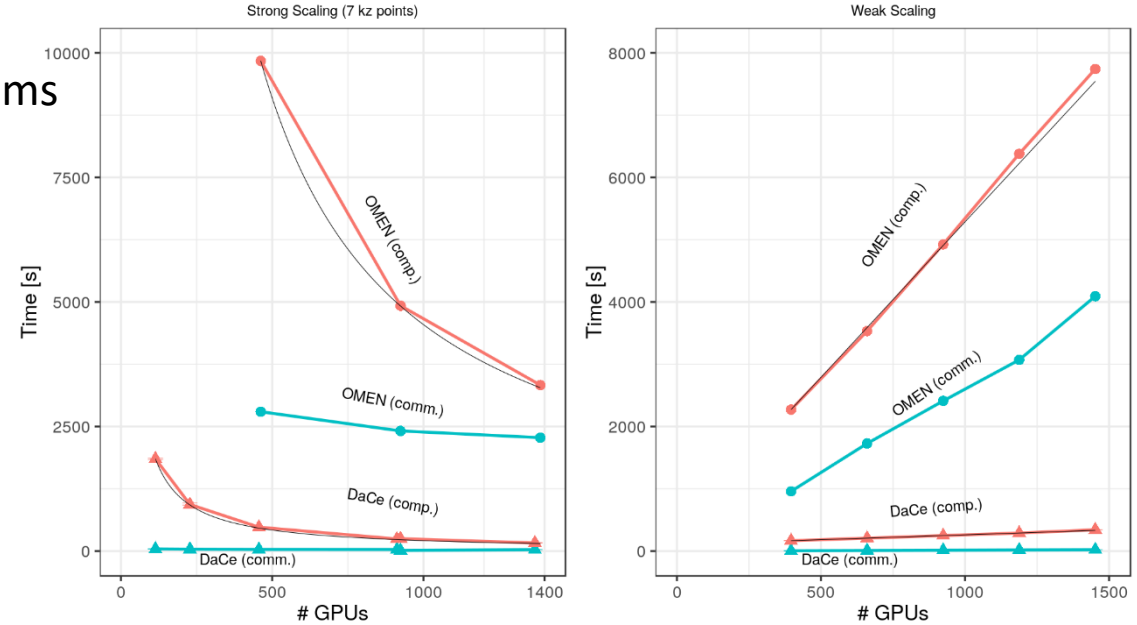
GPU	cuBLAS			DaCe (SBSMM)		
	Gflop	Time	% Peak (Useful)	Gflop	Time	% Peak
P100	27.42	6.73 ms	86.6% (6.1%)	1.92	4.03 ms	10.1%
V100	27.42	4.62 ms	84.8% (5.9%)	1.92	0.97 ms	28.3%

Basic operation in SSE (many very small MMMs)



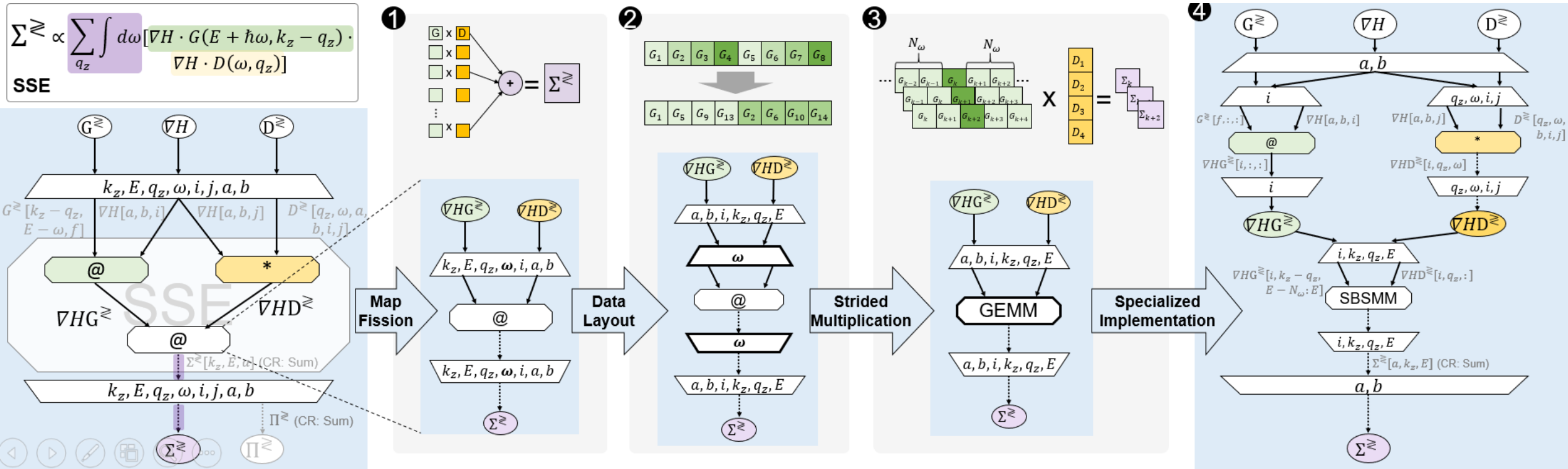
Piz Daint

5k atoms



Summit

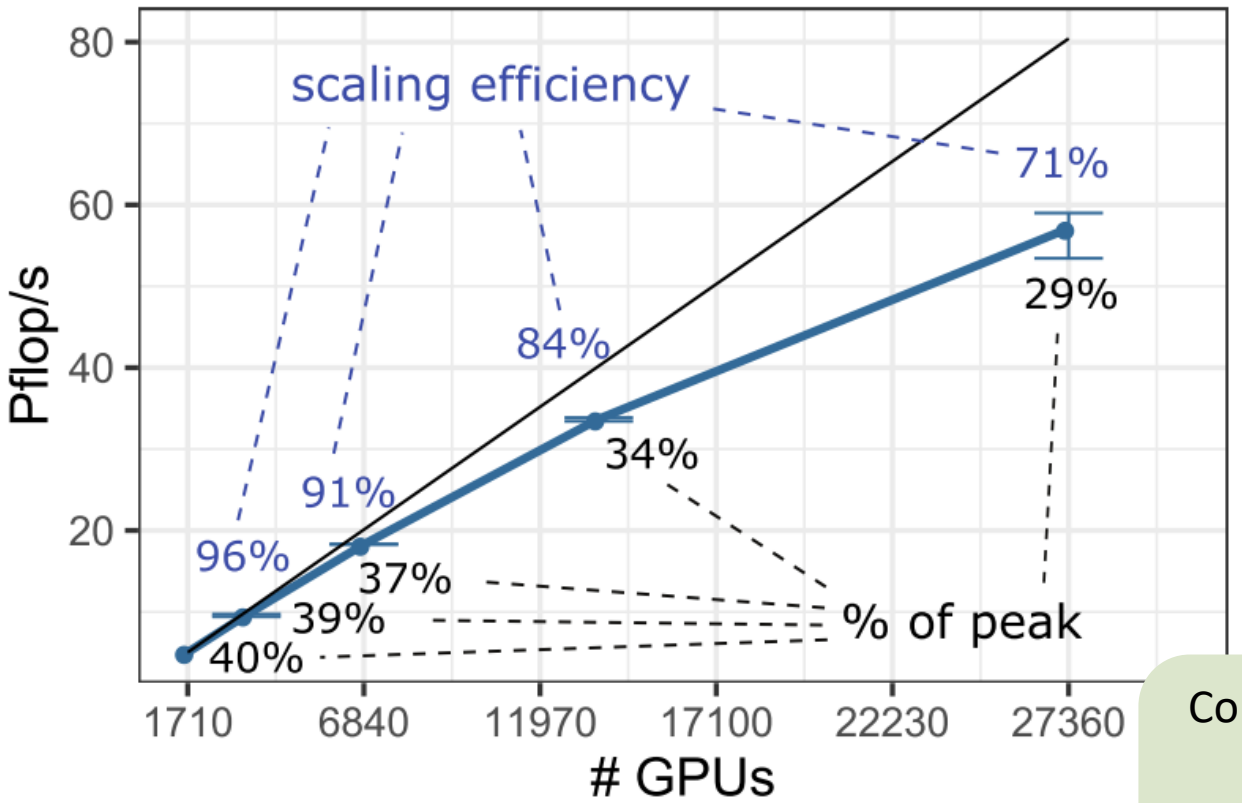
An example of fine-grained data-centric optimization (i.e., how to vectorize)



Ziogas et al. "A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations", SC19 (Gordon Bell Prize 2019).

10,240 atoms on 27,360 V100 GPUs (full-scale Summit)

- 56 Pflop/s with I/O (28% peak)



Variant	N_a	Time [s]	Time/Atom [s]	Speedup
OMEN	1,064	4695.70	4.413	1.0x
DaCe	10,240	489.83	0.048	92.3x

$P = 6,840$, $N_b = 34$, $N_{orb} = 12$, $N_E = 1,220$, $N_\omega = 70$.

Communication time reduced by 417x on Piz Daint!

Volume on full-scale Summit from 12 PB/iter → 87 TB/iter

Already ~100x speedup on 25% of Summit – the original OMEN does not scale further!

Ziogas et al. "A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations", SC19 (Gordon Bell Prize 2019).