Data-Centric Parallel Programming

Torsten Hoefler, invited talk at ROSS'19 at HPDC'19 in conjunction with ACM FCRC

Alexandros Ziogas, Tal Ben-Nun, Guillermo Indalecio, Timo Schneider, Mathieu Luisier, and Johannes de Fine Licht and the whole DAPP team @ SPCL

SPCL

EuroMPI'19
September 11-13 2019
Zurich, Switzerland
https://eurompi19.inf.ethz.ch

How to address locality challenges on standard architectures and programming?

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

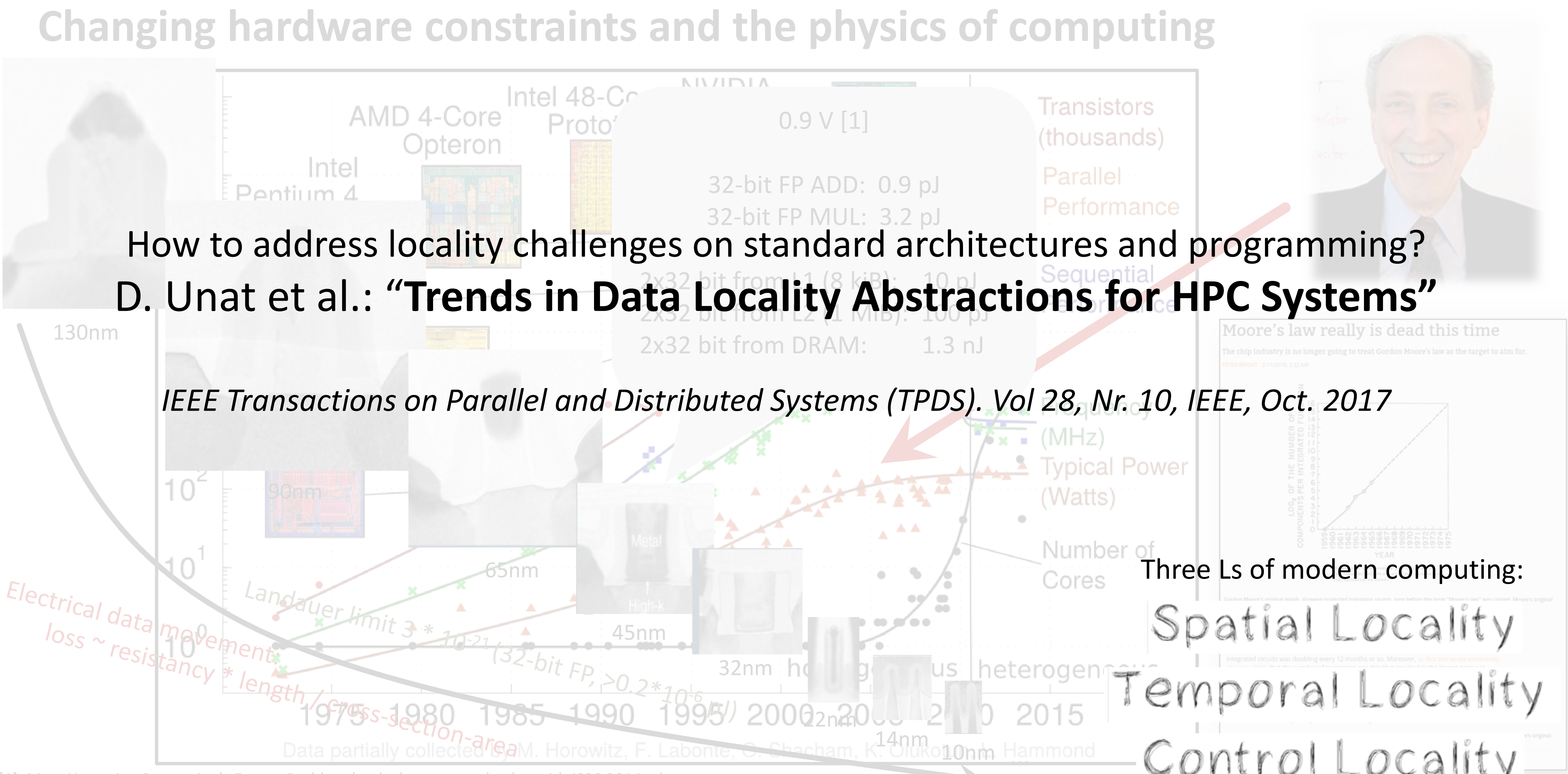
IEEE Transactions on Parallel and Distributed Systems (TPDS). Vol 28, Nr. 10, IEEE, Oct. 2017

Three Ls of modern computing:

Spatial Locality

Temporal Locality

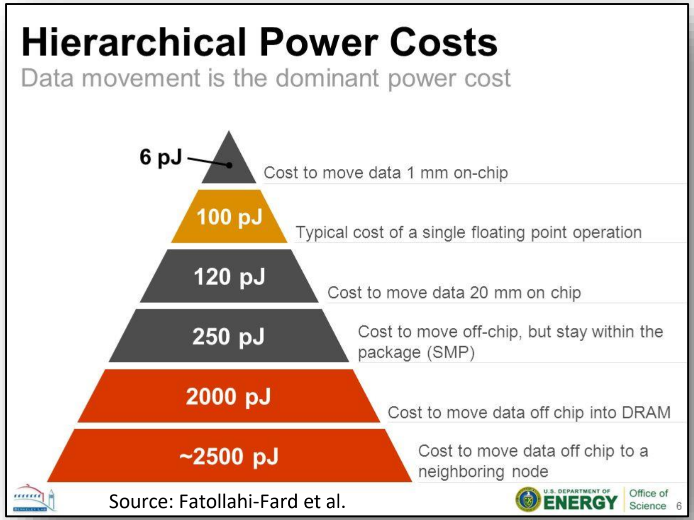
Control Locality

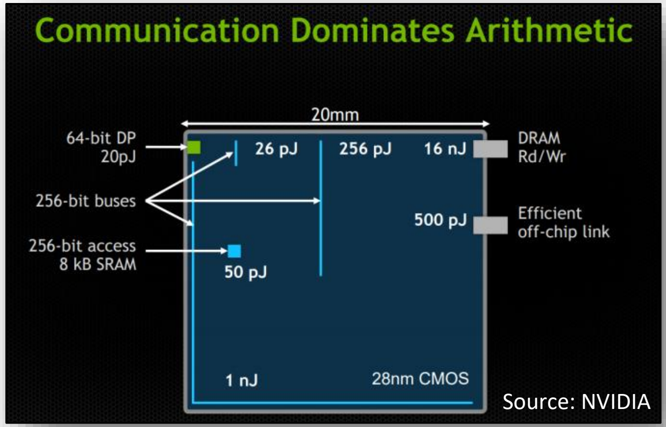


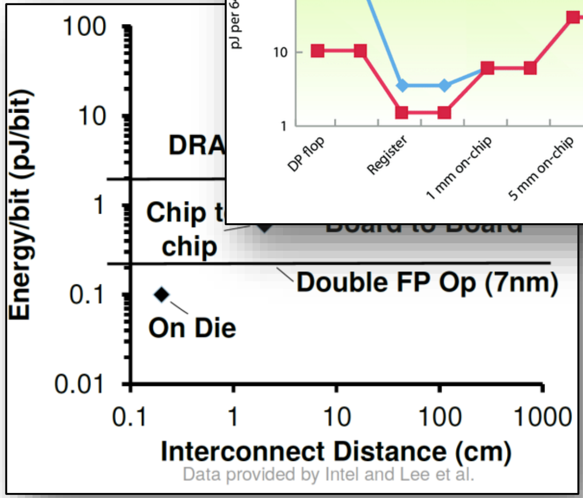
[1]: Marc Horowitz, Computing's Energy Problem (and what we can do about it), ISSC 2014, plenary

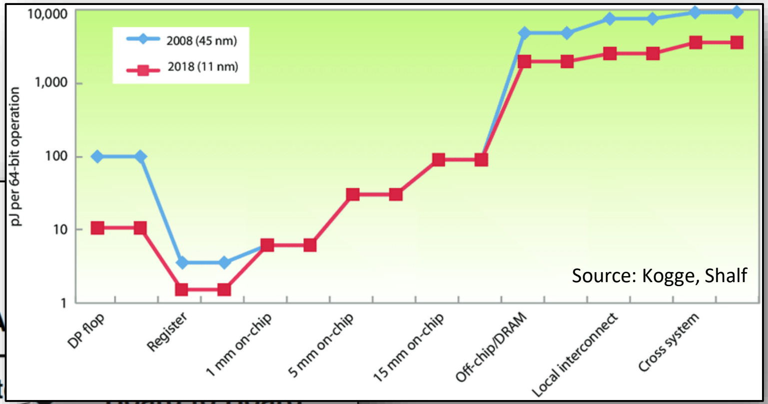
[2]: Moore: Landauer Limit Demonstrated, IEEE Spectrum 2012

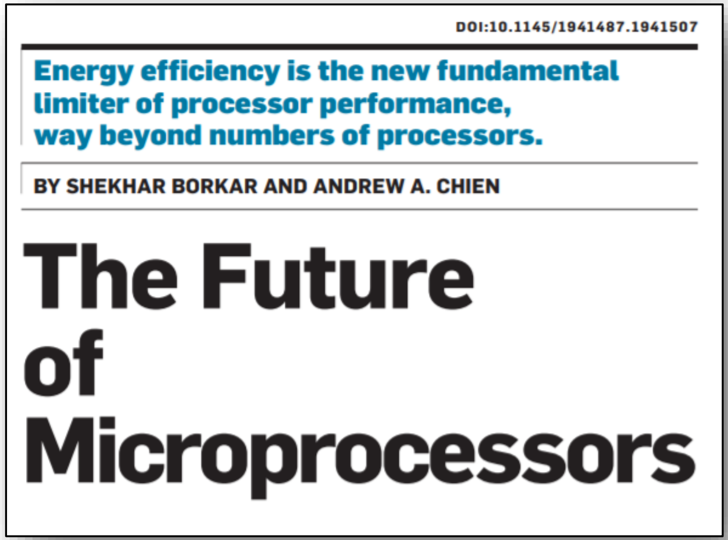
Data movement will dominate everything!










DOI:10.1145/1941487.1941507

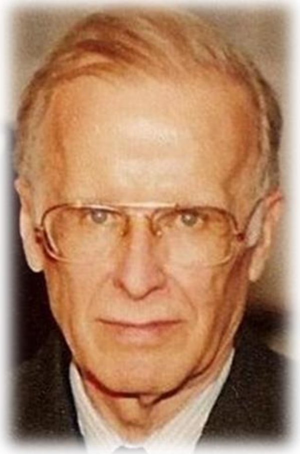
Energy efficiency is the new fundamental limiter of processor performance, way beyond numbers of processors.

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)

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."*

- **Legion [1] is taking a good direction towards data-centric**
  - Tasking relies on data placement but not really dependencies (not visible to tool-chain)
  - But it is still control-centric in the tasks – not (performance) portable between devices!

- **Let's go a step further towards an explicitly data-centric viewpoint**
  - For performance engineers at least!

[1]: Bauer et al.: "Legion: expressing locality and independence with logical regions", SC12, 2012

Performance Portability with DataCentric (DaCe) Parallel Programming

Domain Scientist

Problem Formulation

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

Python / NumPy

DSLs

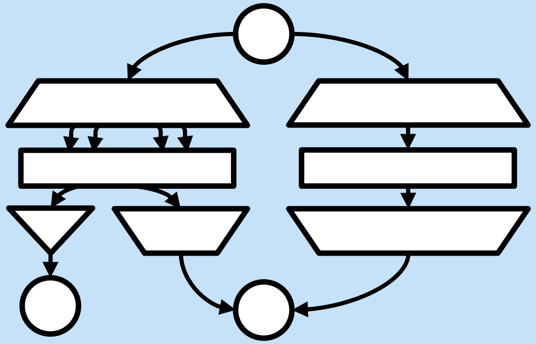
TensorFlow

MATLAB

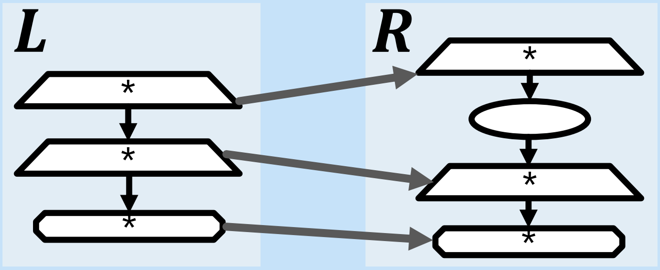
SDFG Builder API

High-Level Program

Performance Engineer



Data-Centric Intermediate Representation (SDFG, §3)



Graph Transformations (API, Interactive, §4)

Transformed Dataflow

Performance Results

System

Hardware Information

SDFG Compiler

Runtime

CPU Binary

GPU Binary

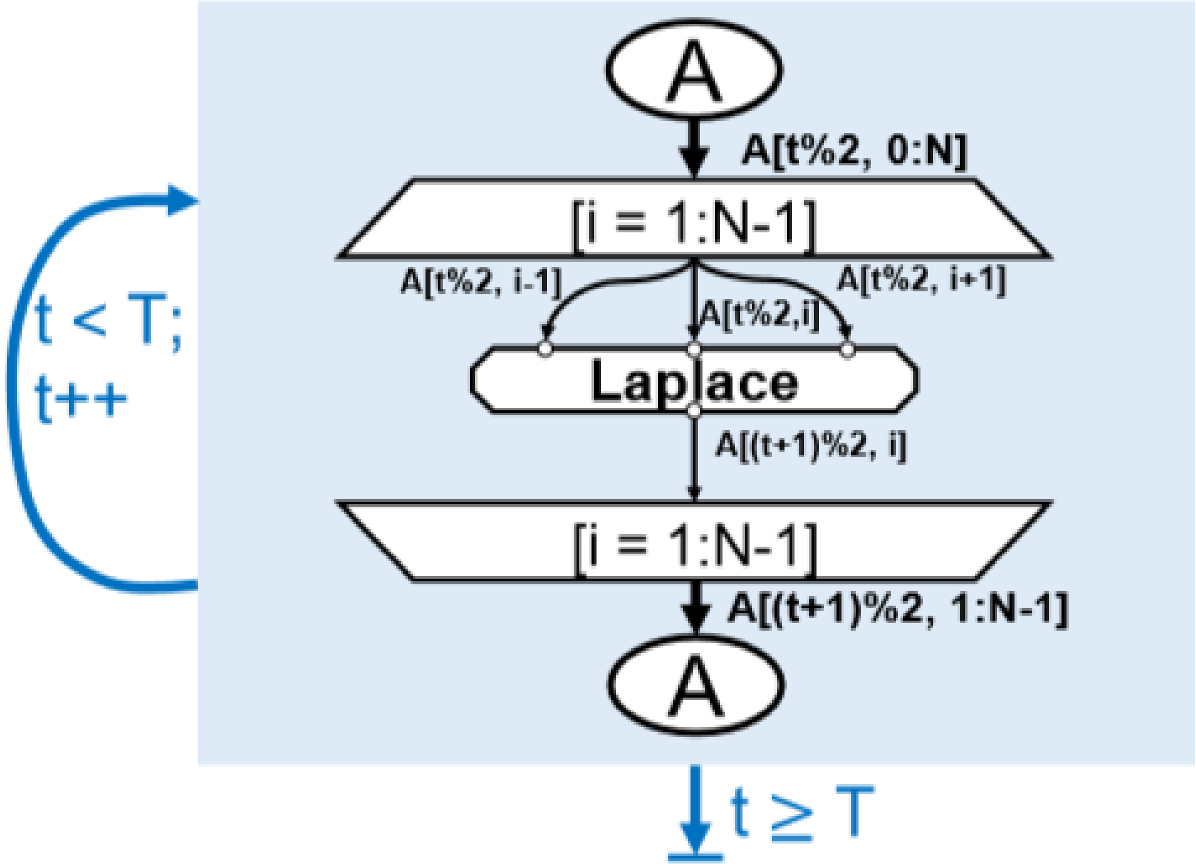
FPGA Modules

Thin Runtime Infrastructure

Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

A first example in DaCe Python

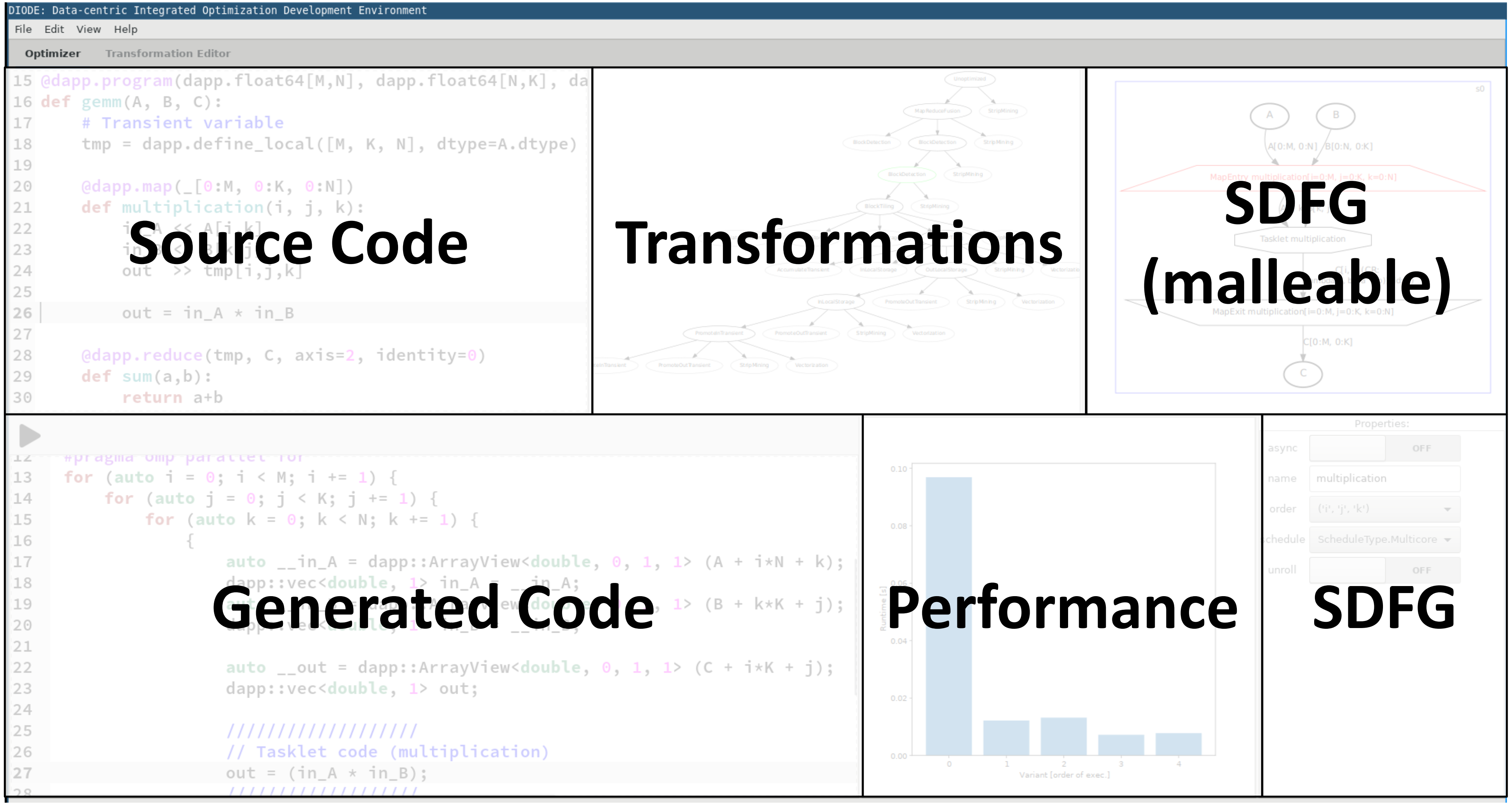
```python
@dace.program
def Laplace(A: dace.float64[2,N],
            T: dace.uint32):
  for t in range(T):
    for i in dace.map[1:N-1]:
      # Data dependencies
      in_l << A[t%2,     i-1]
      in_c << A[t%2,     i]
      in_r << A[t%2,     i+1]
      out  >> A[(t+1)%2, i]
      # Computation
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



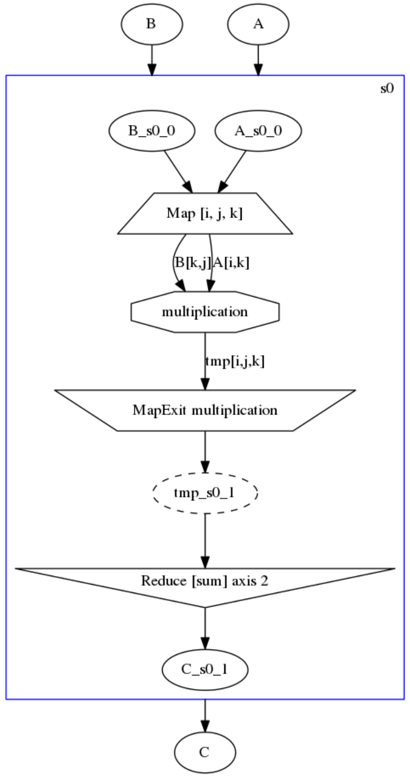

```
var << A(1, WCR)[0:N]
```

- var: Local variable name
- <<: Direction (<<, >>)
- A: Data
- 1: Number of accesses
- WCR: Conflict Resolution
- [0:N]: Location/Range

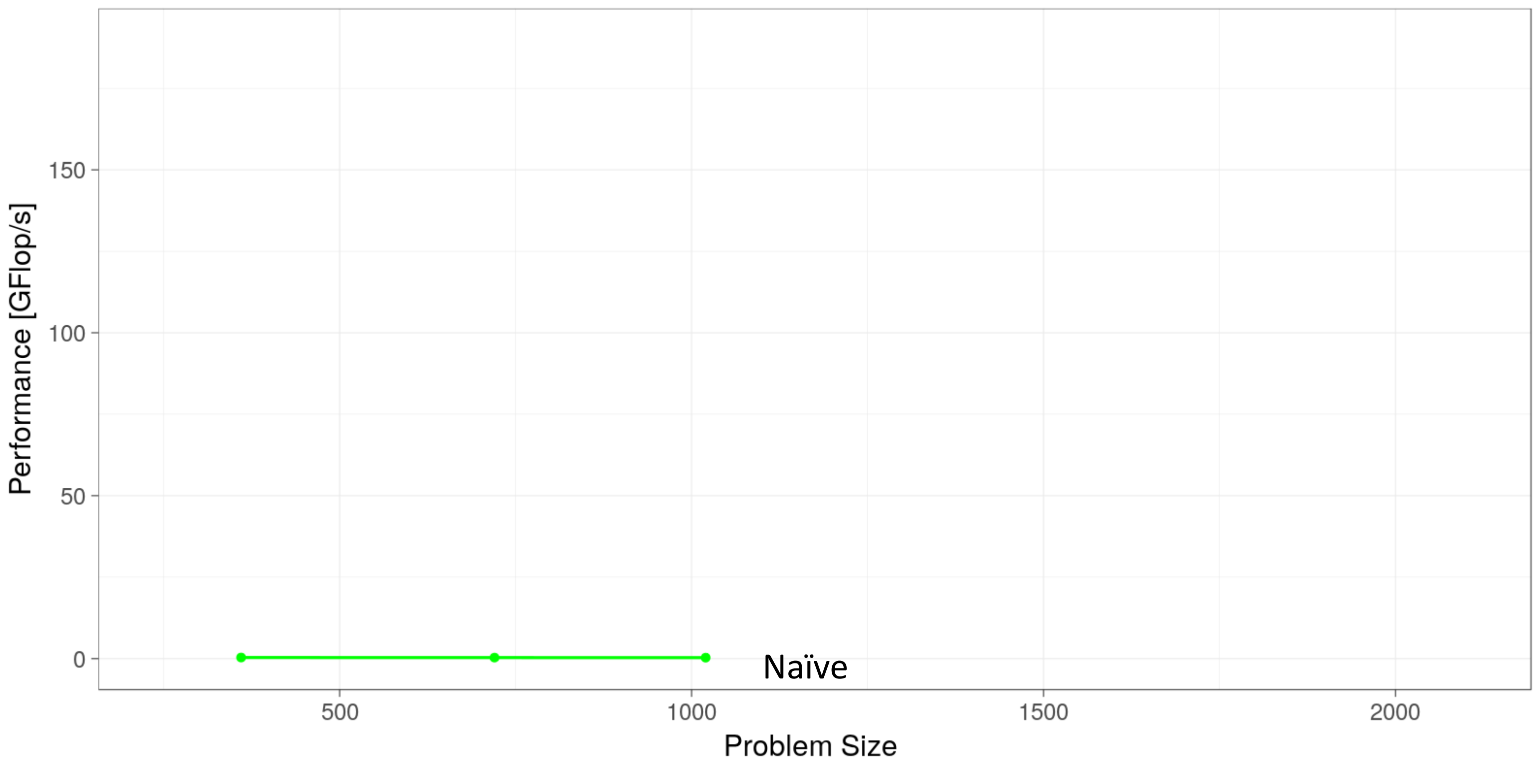
DIODE User Interface



Performance for matrix multiplication on x86

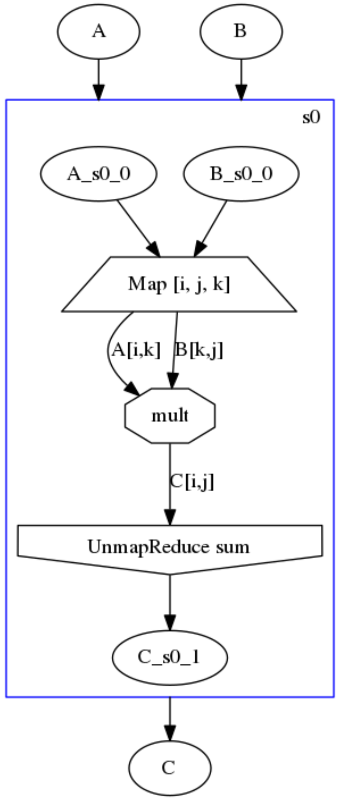


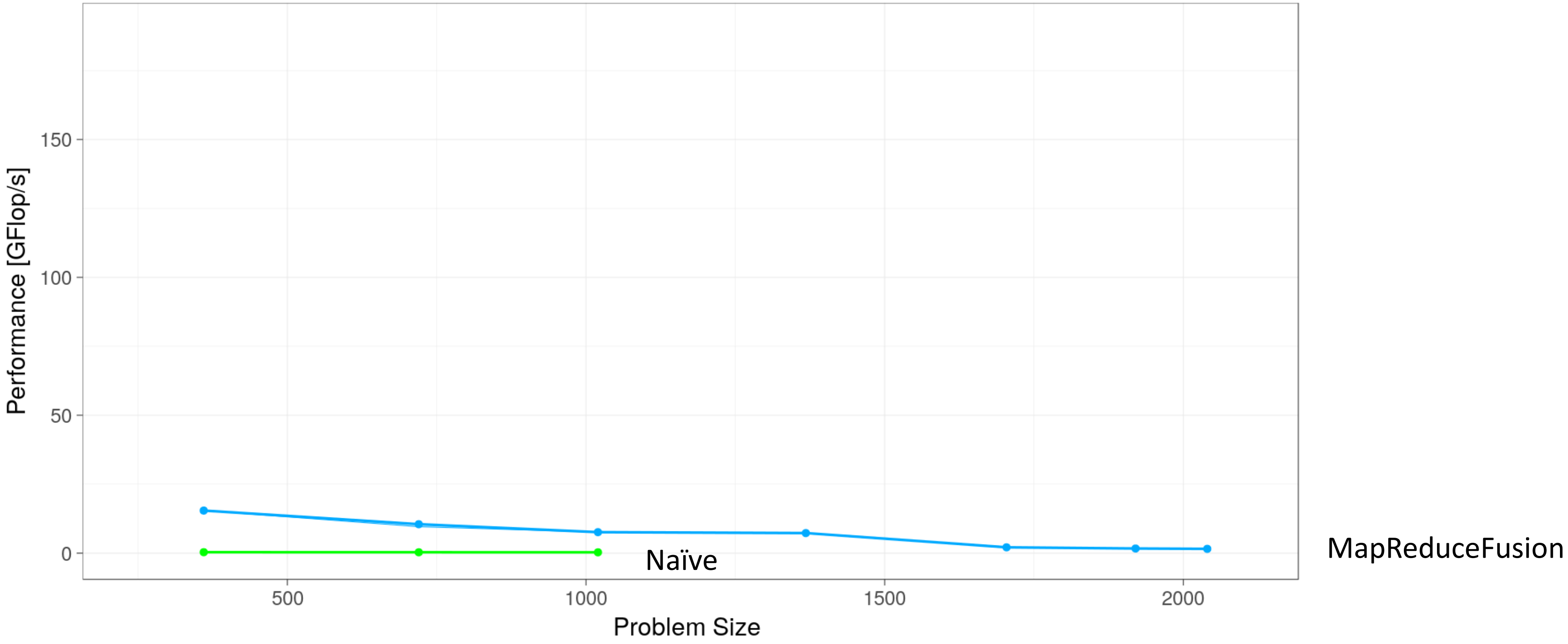
SDFG



Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

Performance for matrix multiplication on x86

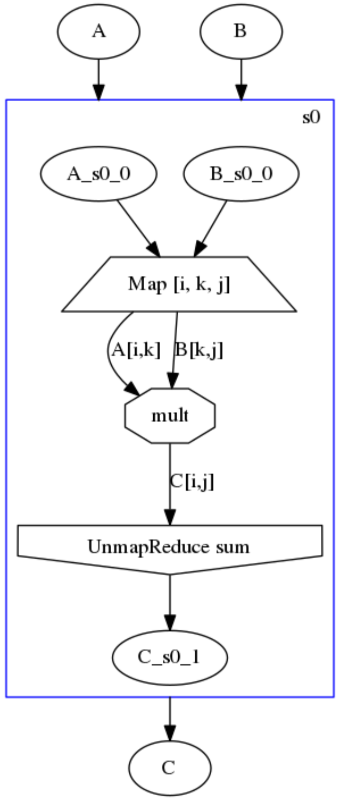


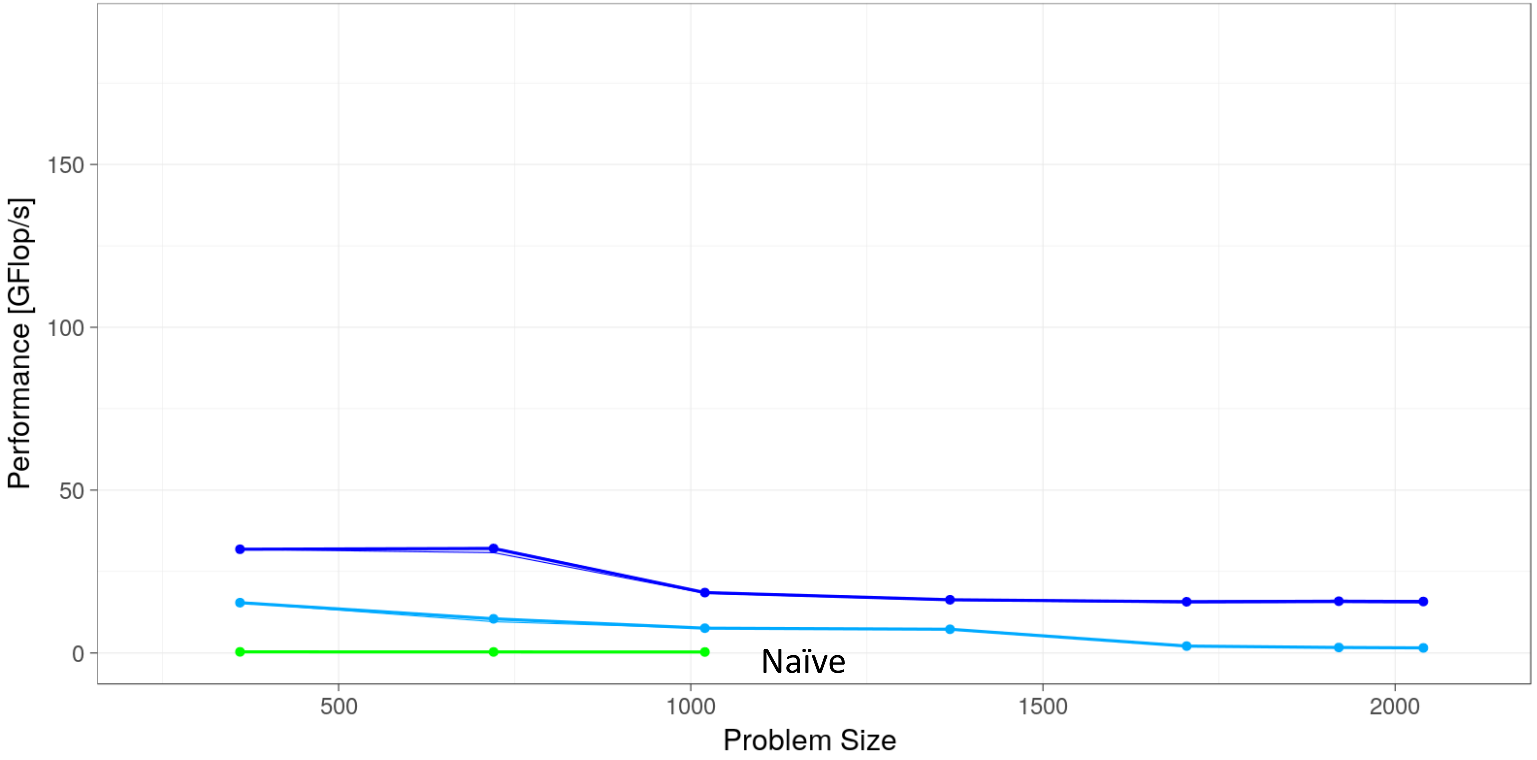


SDFG

Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

Performance for matrix multiplication on x86



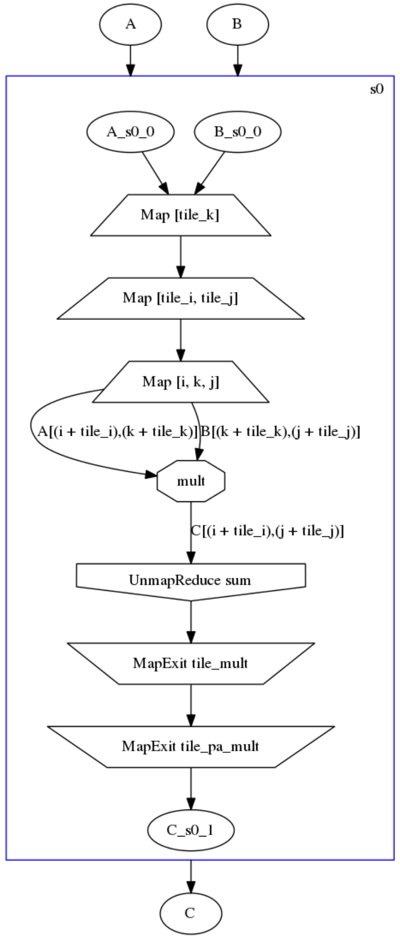


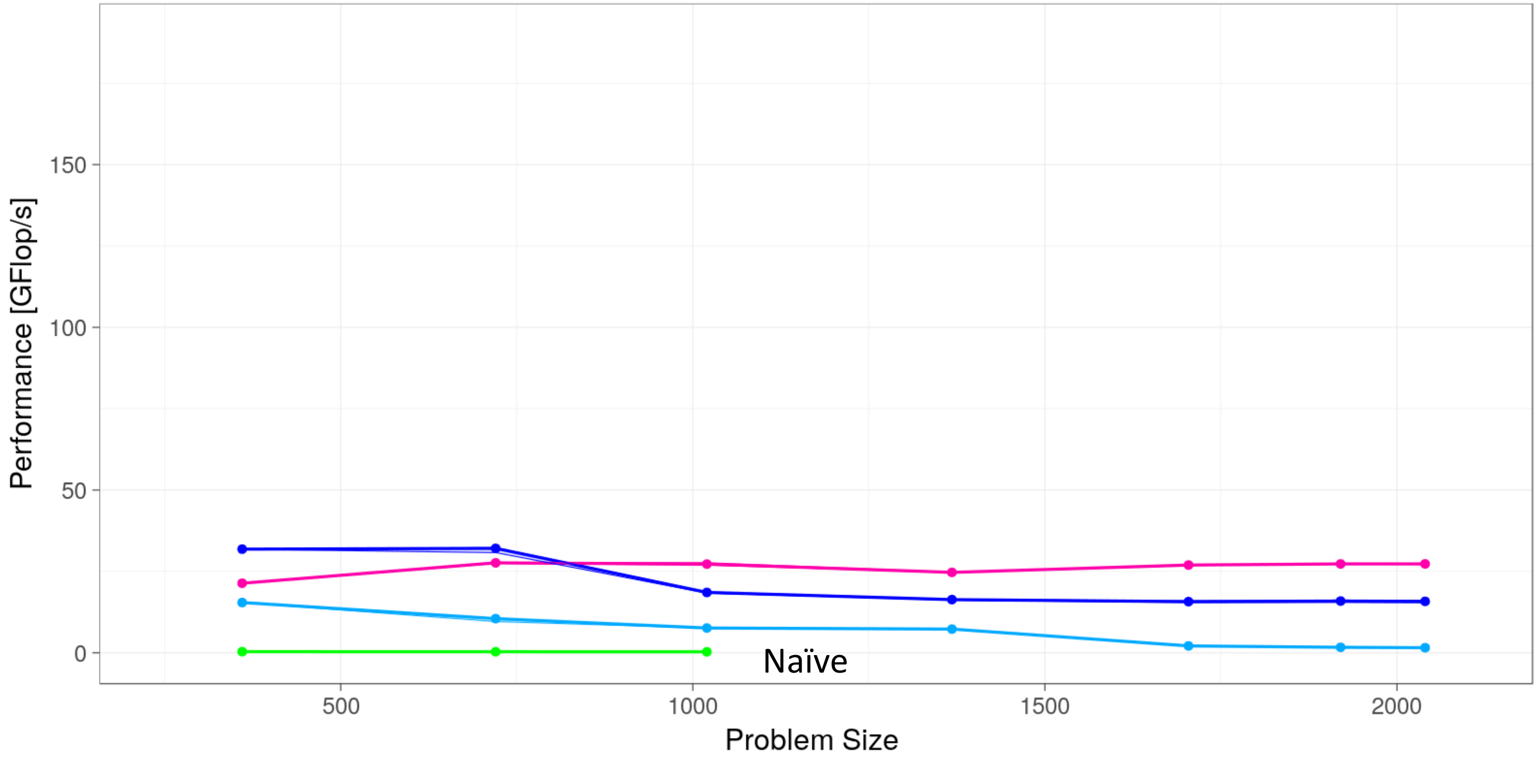
LoopReorder
MapReduceFusion

SDFG

Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

Performance for matrix multiplication on x86



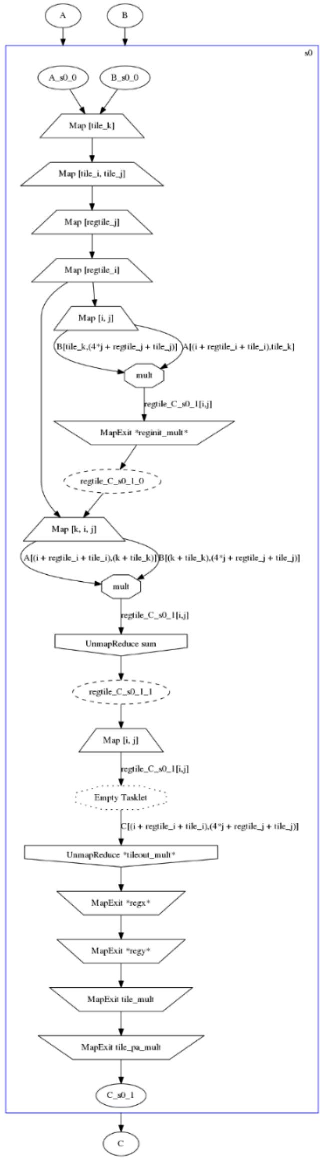


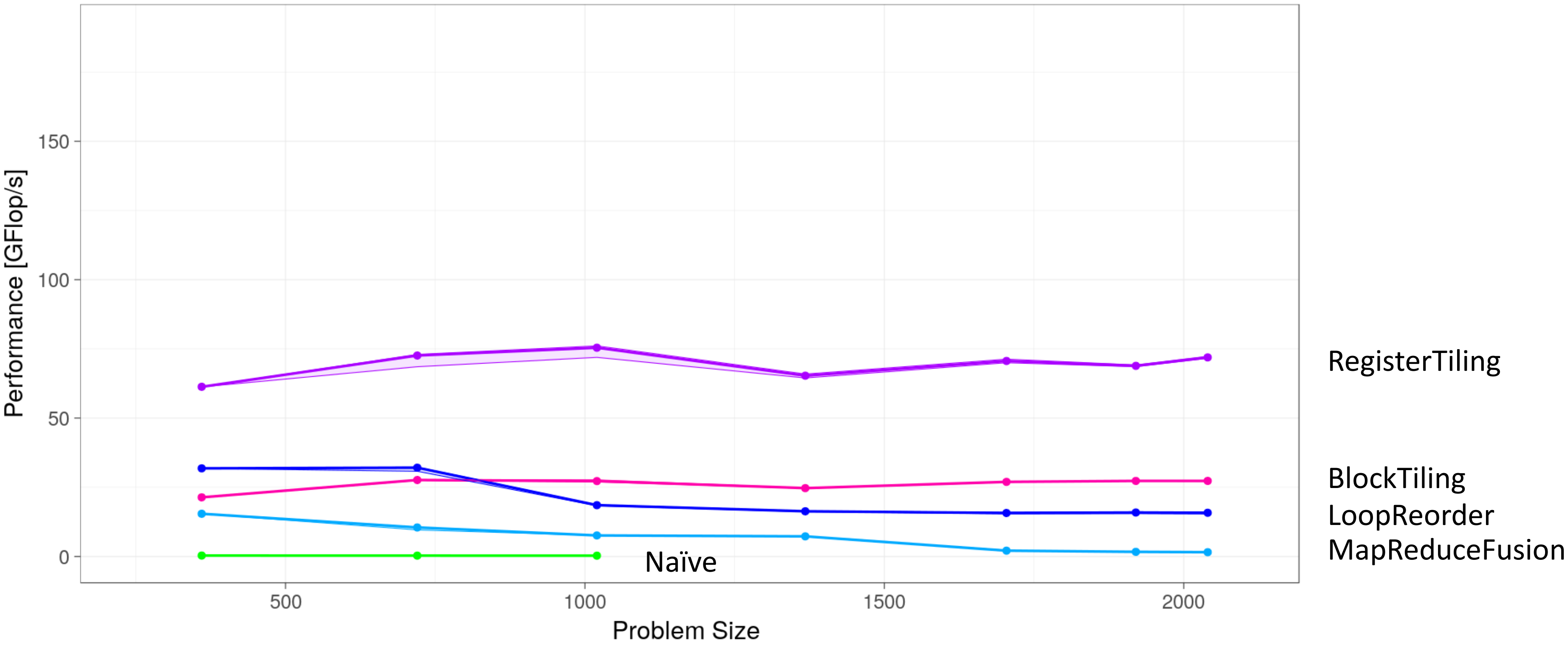
BlockTiling
LoopReorder
MapReduceFusion

SDFG

Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

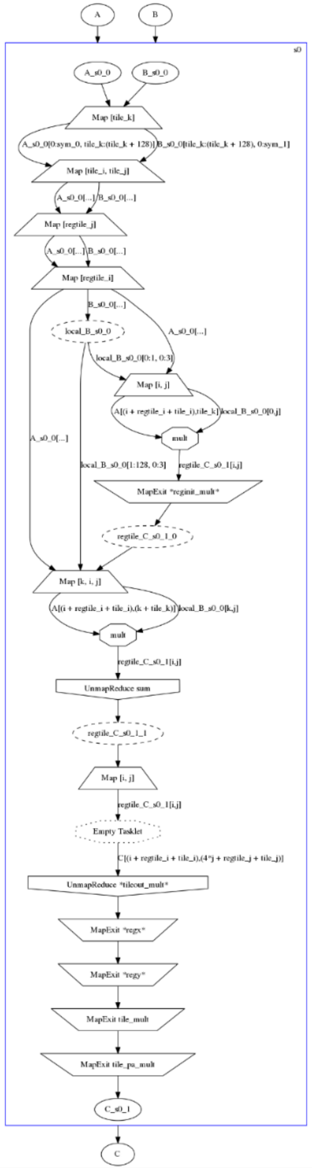
Performance for matrix multiplication on x86

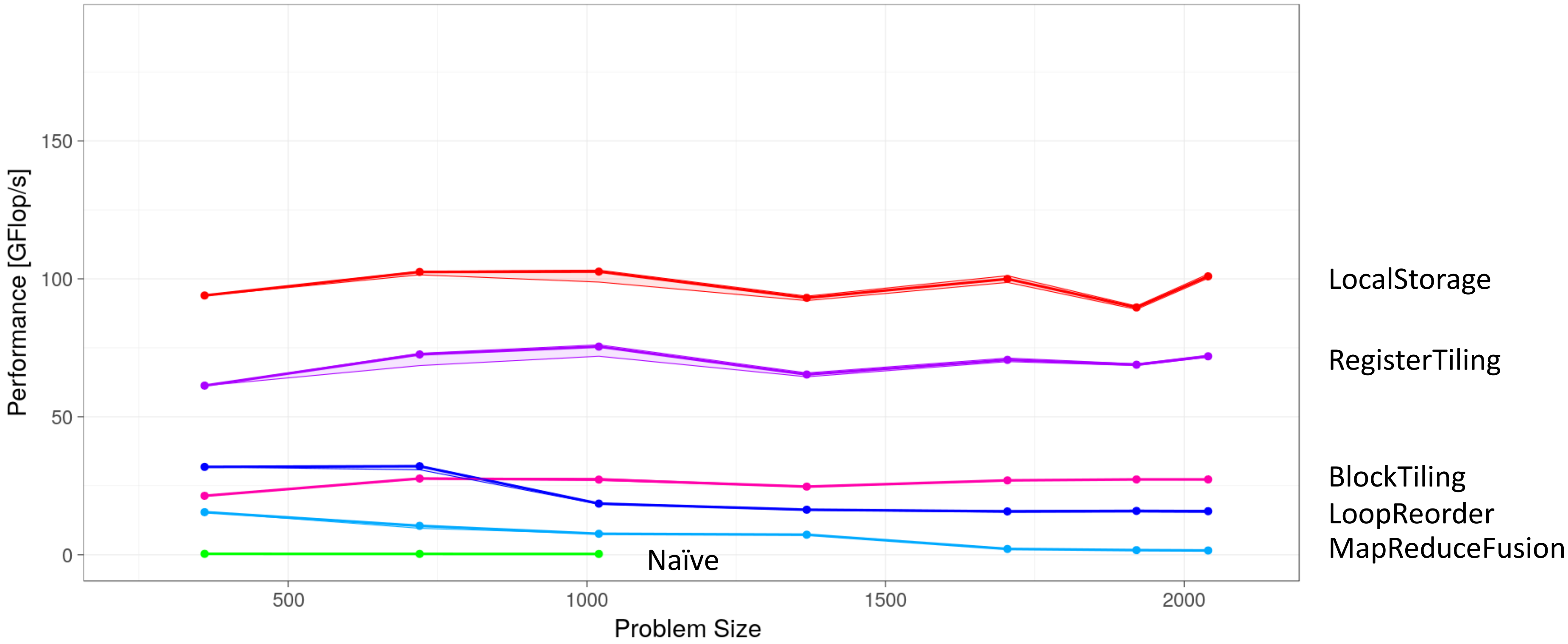




Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

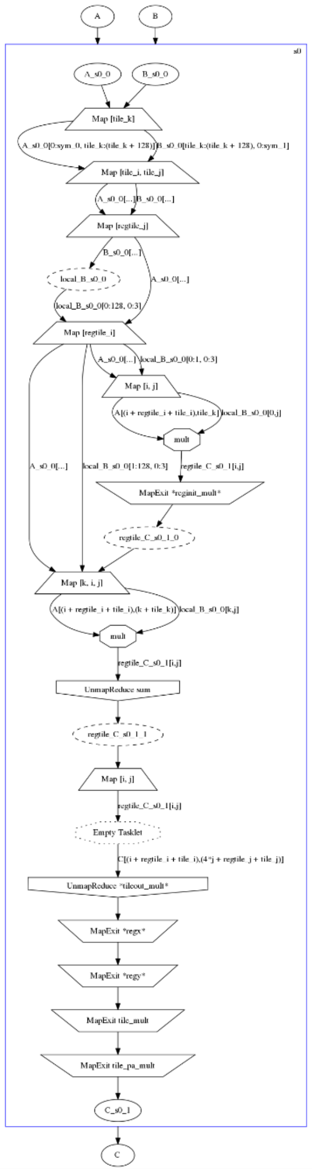
Performance for matrix multiplication on x86

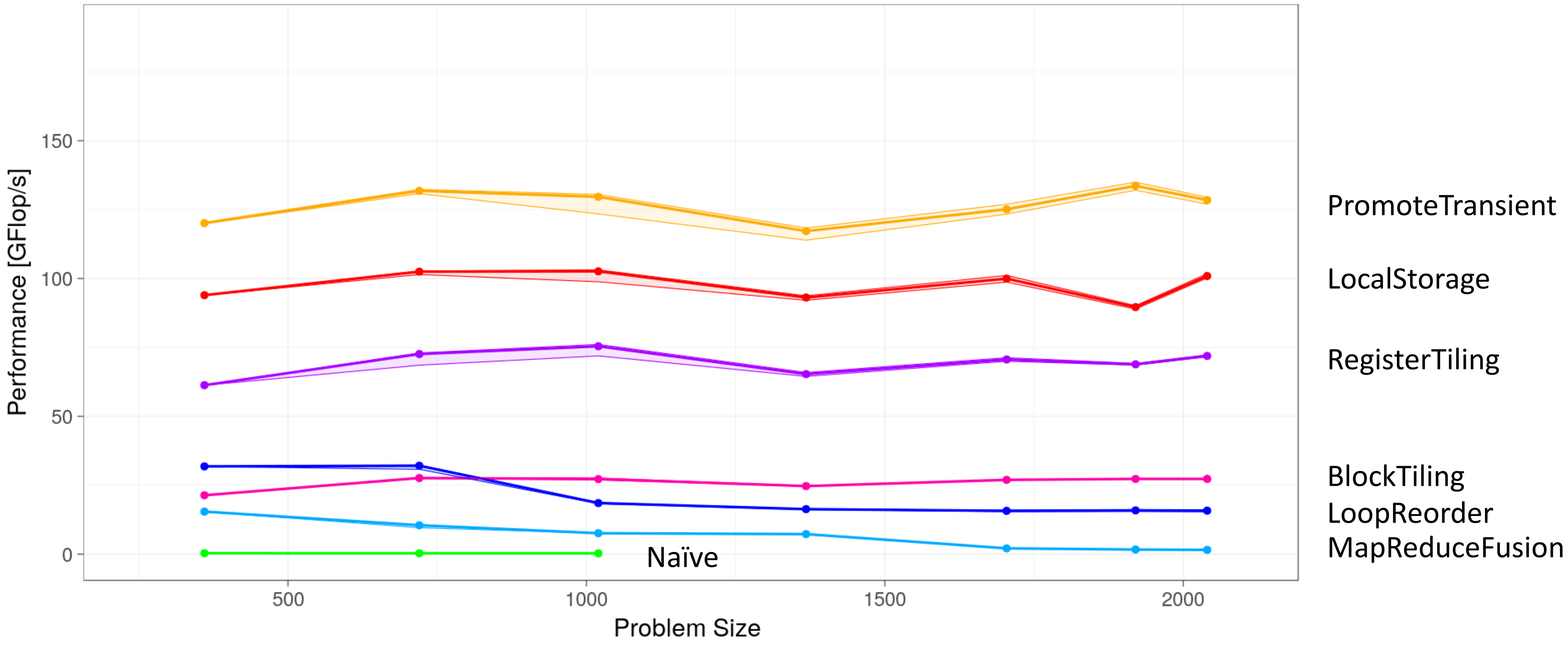




Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

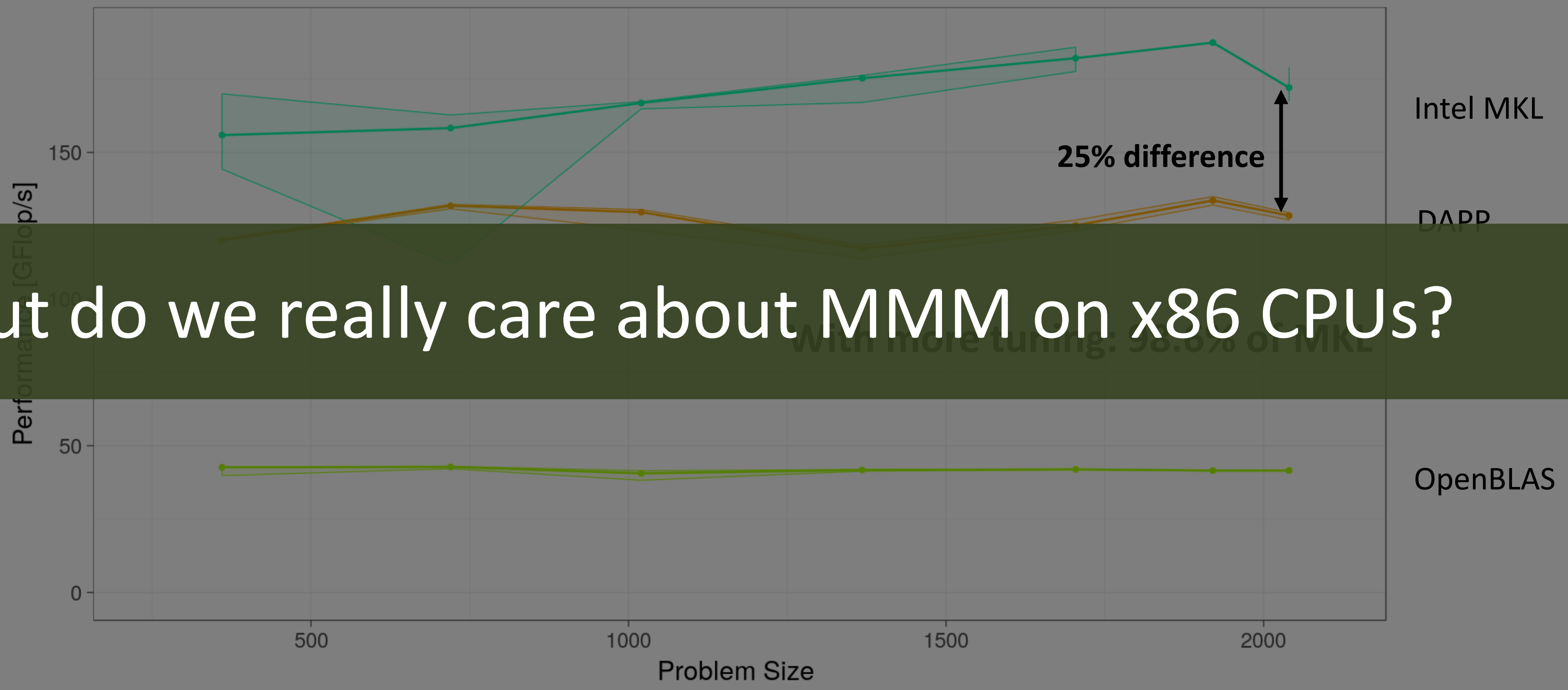
Performance for matrix multiplication on x86





Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at SC19

Performance for matrix multiplication on x86



But do we really care about MMM on x86 CPUs?

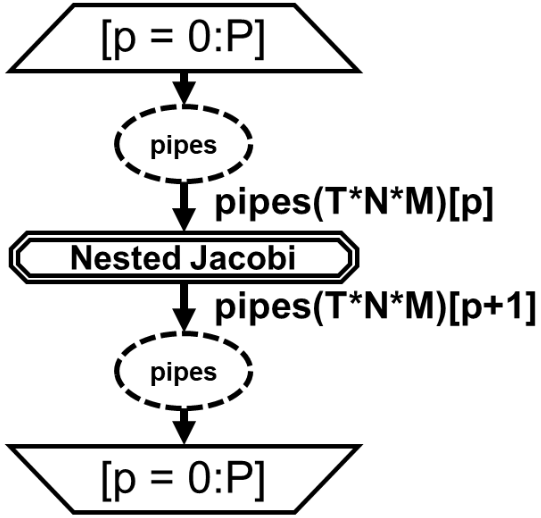
Hardware Mapping: Load/Store Architectures

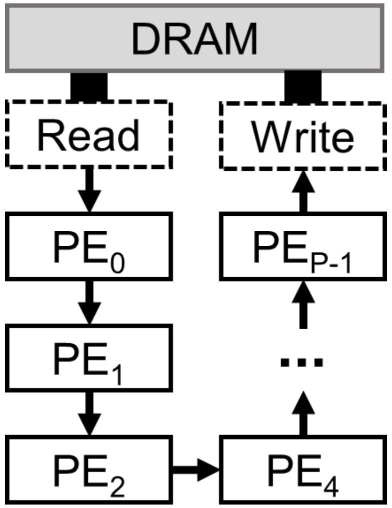
- **Recursive code generation (C++, CUDA)**
  - **Control flow:** Construct detection and gotos
- **Parallelism**
  - **Multi-core CPU**: OpenMP, atomics, and threads
  - **GPU**: CUDA kernels and streams
  - Connected components run concurrently
- **Memory and interaction with accelerators**
  - Array-array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

```
void _program_gemm(int sym_0, int sym_1, int sym_2, double * __re
    // State s0
    for (int tile_k = 0; tile_k < sym_2; tile_k += 128) {
        #pragma omp parallel for
        for (int tile_i = 0; tile_i < sym_0; tile_i += 64) {
            for (int tile_j = 0; tile_j < sym_1; tile_j += 240) {
                for (int regtile_j = 0; regtile_j < (min(240, sym_

                    vec<double, 4> local_B_s0_0[128 * 3];
                    Global2Stack_2D_FixedWidth<double, 4, 3>(&B[t
                                                             loca

                    for (int regtile_i = 0; regtile_i < (min(64,
                        vec<double, 4> regtile_C_s0_1[4 * 3];
                        for (int i = 0; i < 4; i += 1) {
                            for (int j = 0; j < 3; j += 1) {
                                double in_A = A[(i + regtile_i +
                                vec<double, 4> in_B = local_B_s0_
                                // Tasklet code (mult)
                                auto out = (in_A * in_B);
                                regtile_C_s0_1[i*3 + j] = out;
                            }
                        }
                        for (int k = 1; k < (min(128, sym_2 - til
                        // ...
```

Hardware Mapping: Pipelined Architectures

- **Module generation with HDL and HLS**
  - Integration with Xilinx SDAccel
  - 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





Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at 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

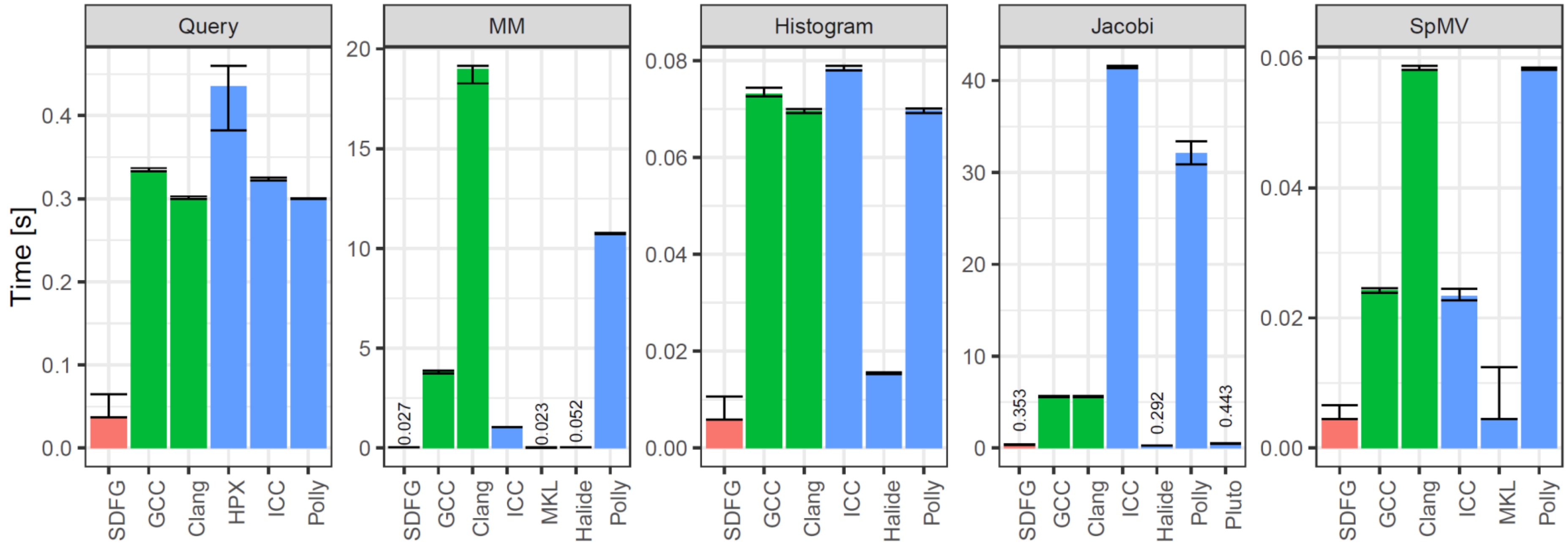
    NVIDIA CUB

Preprint (arXiv): Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, to appear at 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

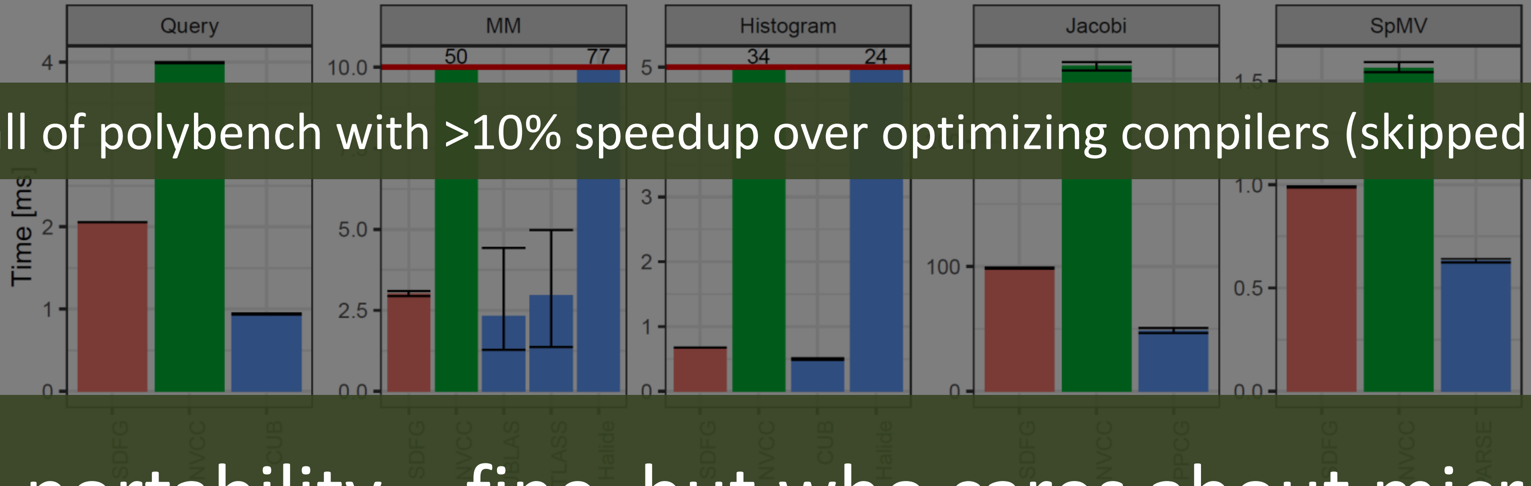


Performance Evaluation: Fundamental Kernels (GPU, FPGA)

90% of CUTLASS

We also have all of polybench with >10% speedup over optimizing compilers (skipped for time reasons)

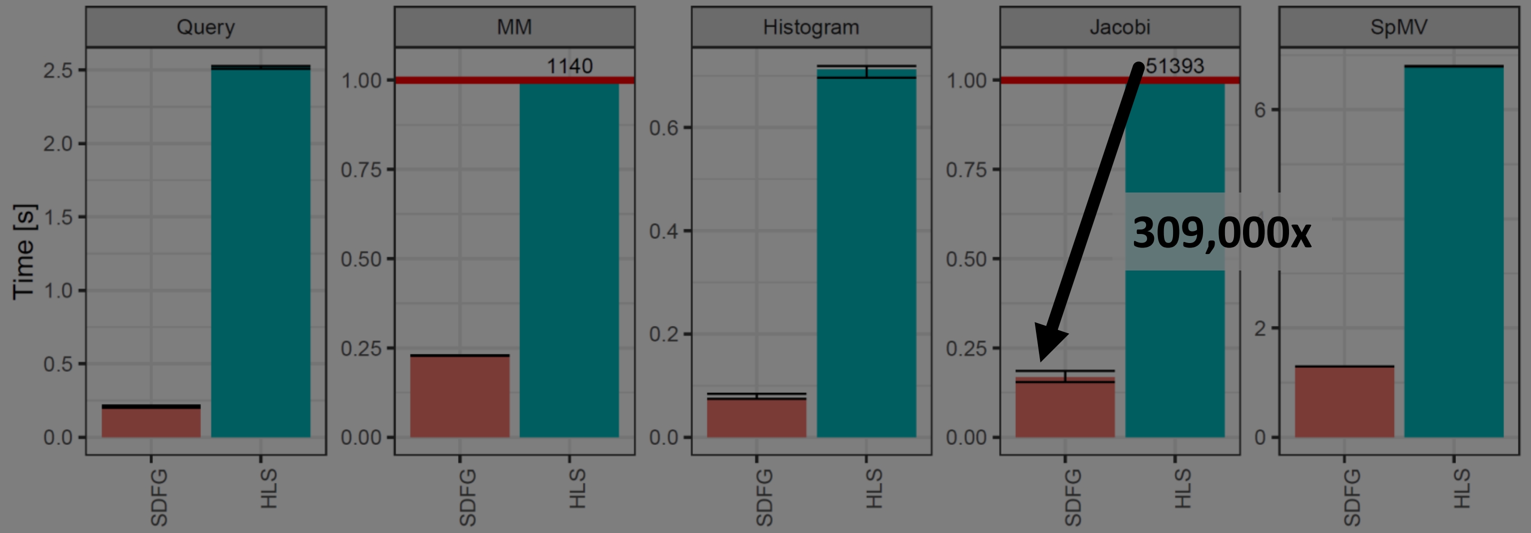
GPU



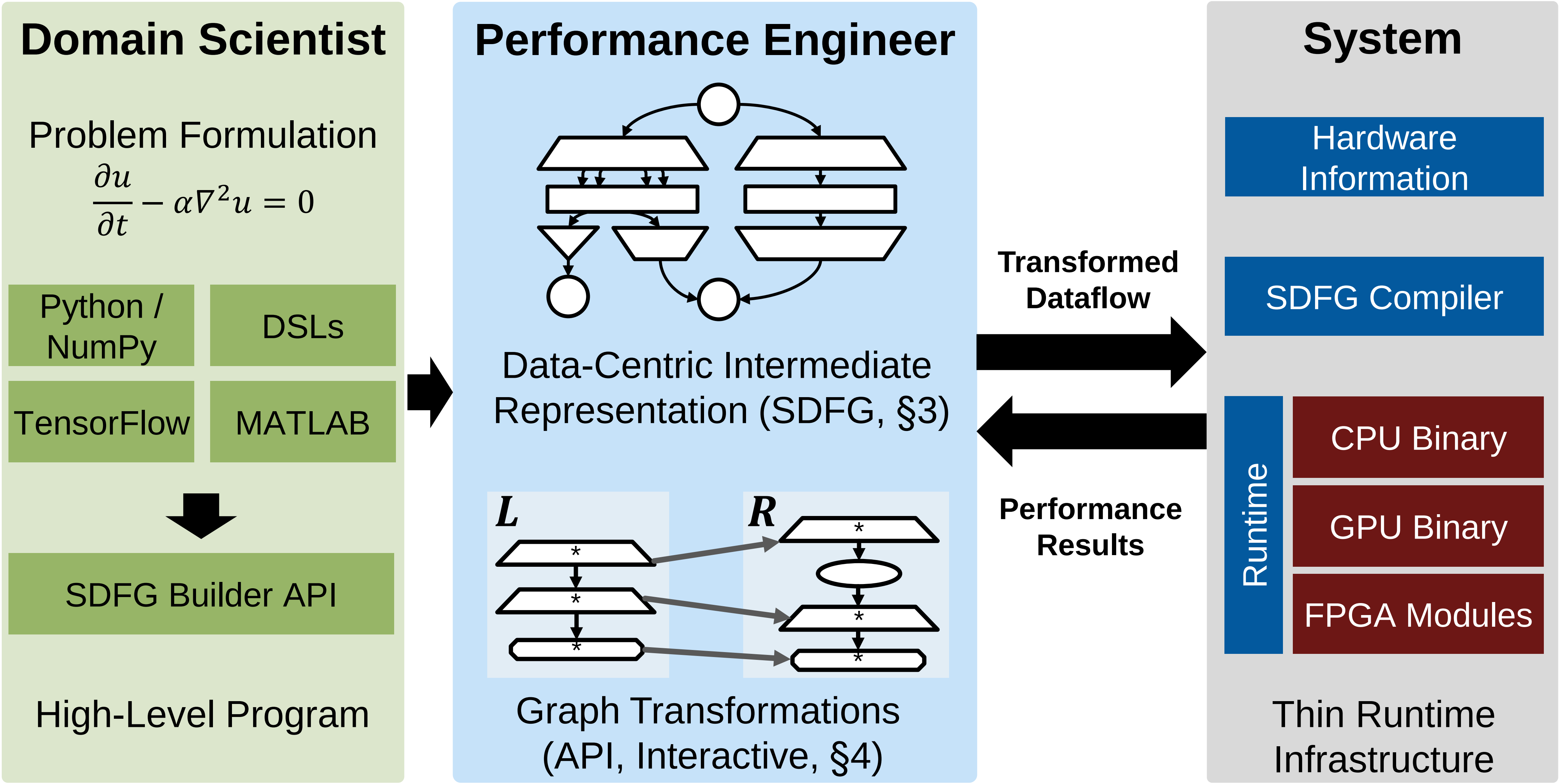
Performance portability – fine, but who cares about microbenchmarks?

19.5x of Spatial

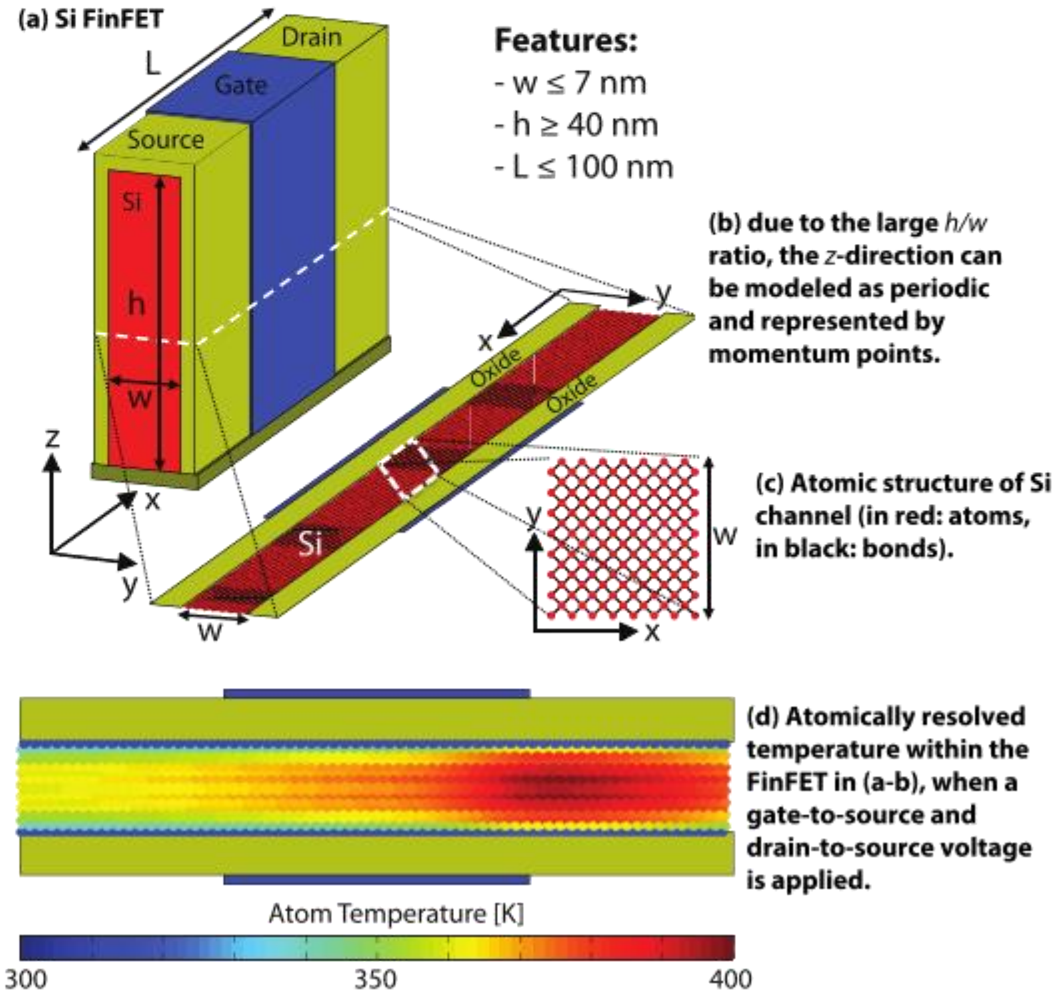
FPGA

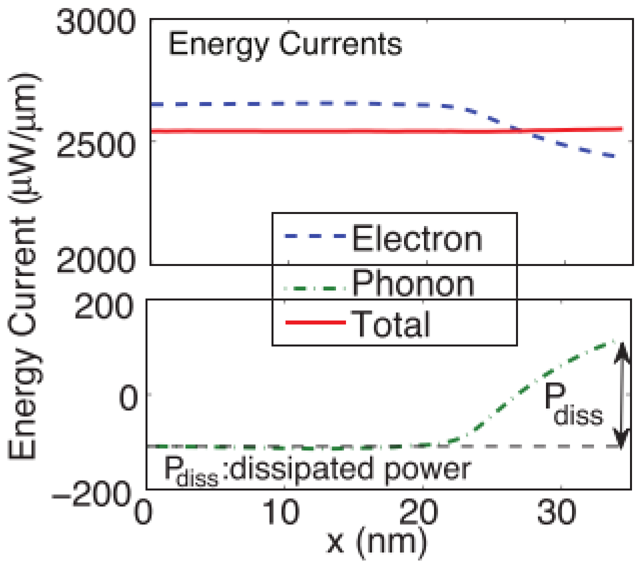


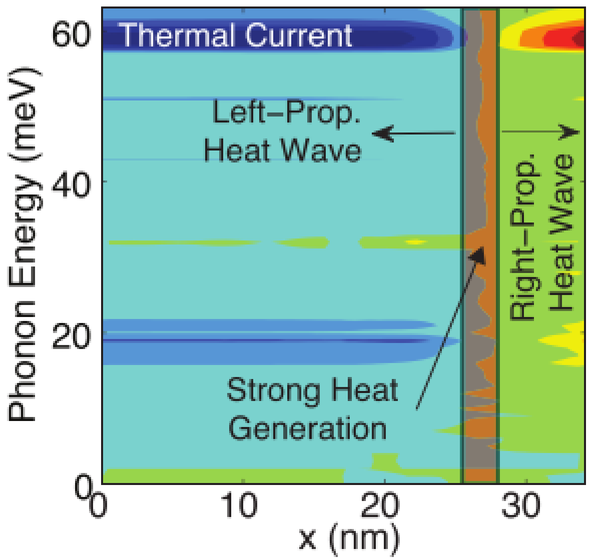
Remember the promise of DAPP – on to a real application!

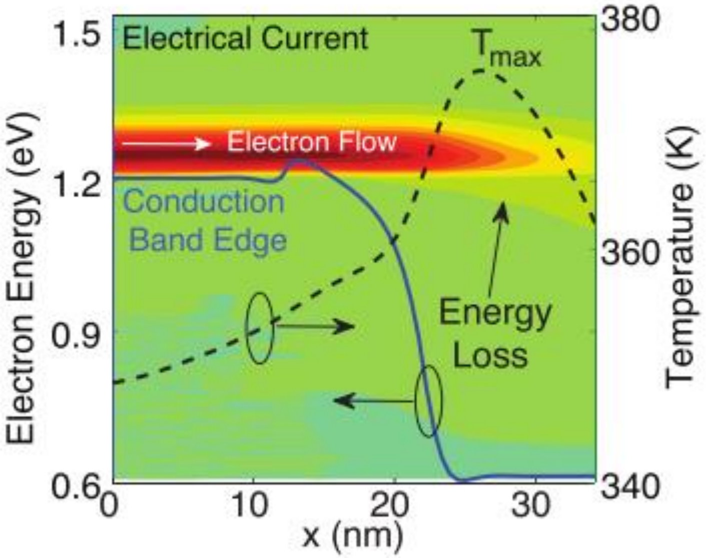


Next-Generation Transistors need to be cooler – addressing self-heating









Quantum Transport Simulations with OMEN

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

NEGF

SSE $\Sigma[G(E+\hbar\omega, k_z - q_z)\, D(\omega, q_z)](E, k_z)$

Electrons $G(E, k_z)$

$$(E \cdot S - H - \Sigma^R) \cdot G^R = I$$
$$G^< = G^R \cdot \Sigma^< \cdot G^A$$

Phonons $D(\omega, q_z)$

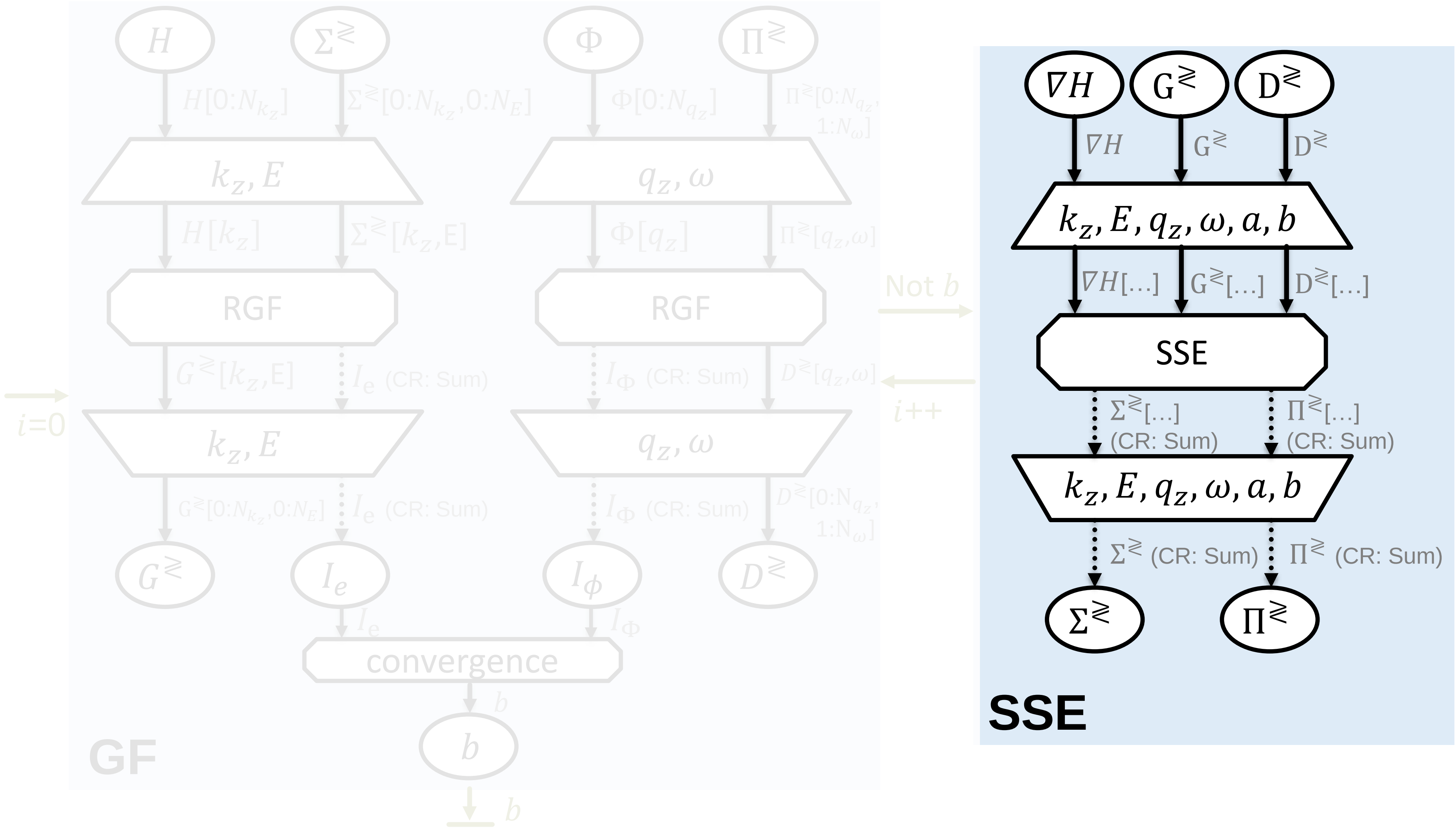
$$(\omega^2 - \Phi - \Pi^R) \cdot D^R = I$$
$$D^< = D^R \cdot \Pi^< \cdot D^A$$

GF

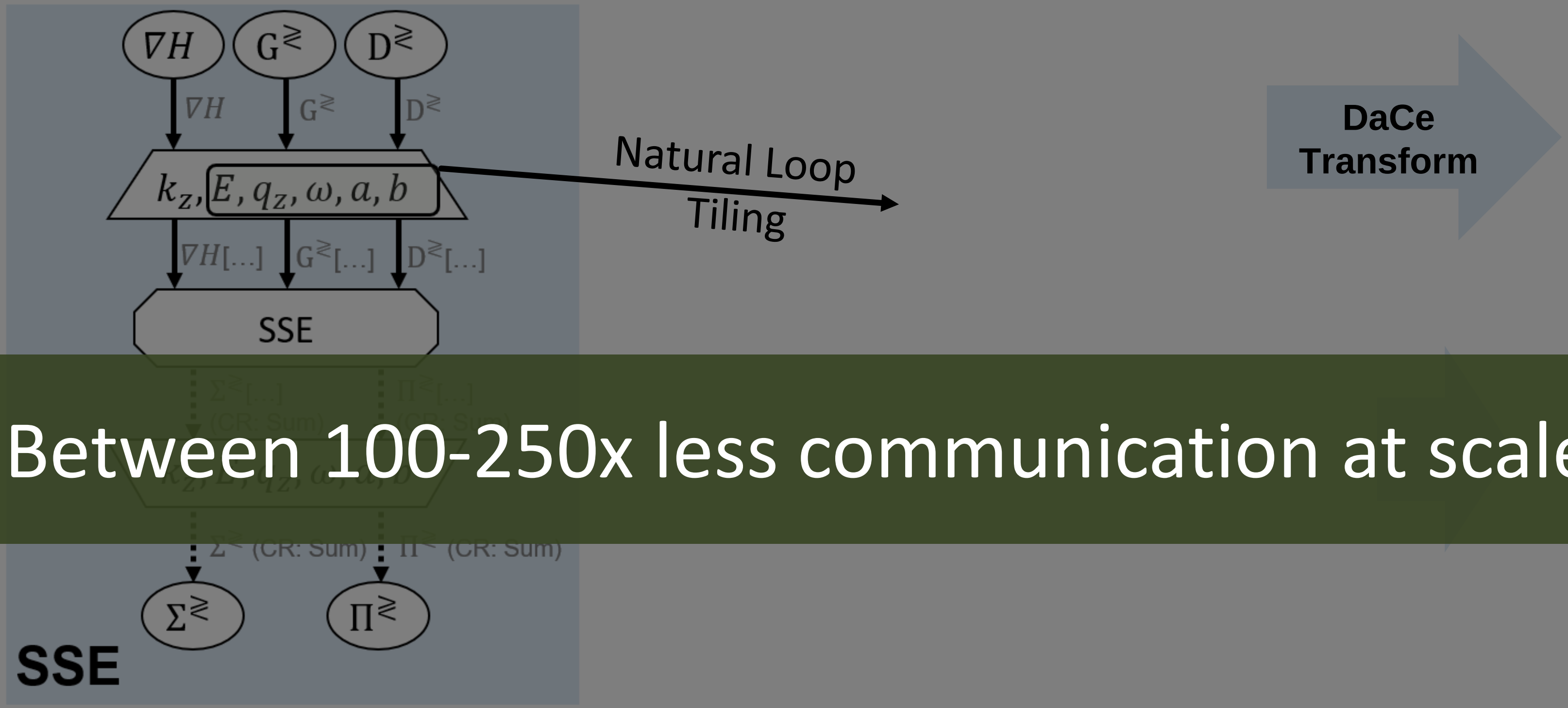
SSE $\Pi[G(E, k_z)\, G(E+\hbar\omega, k_z + q_z)](\omega, q_z)$

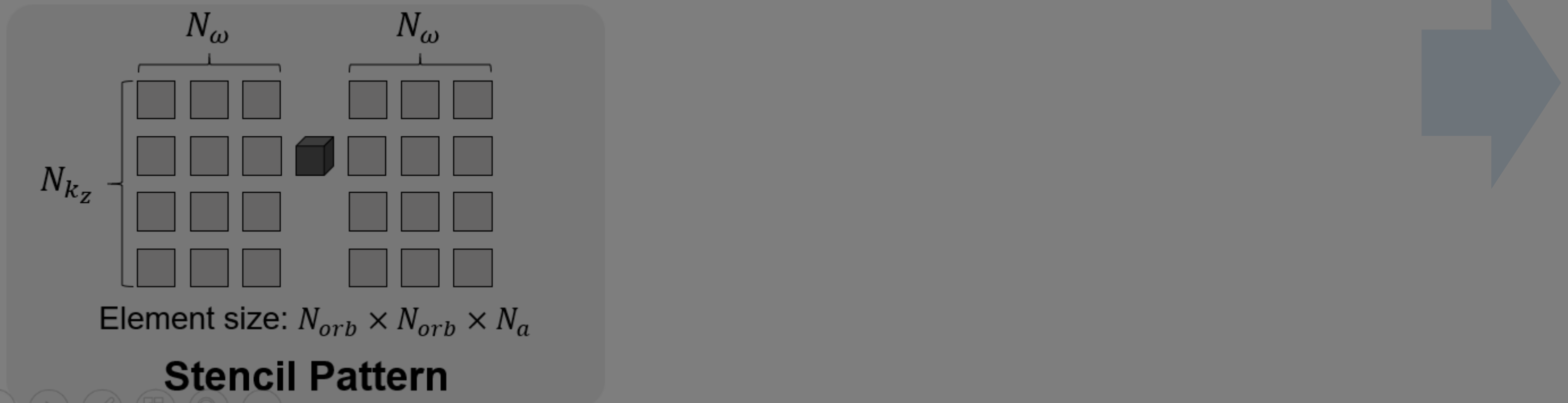
Ziogas, et al.: A Data-Centric Approach to Extreme-Scale Dissipative Quantum Transport Simulations, to appear at SC19

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



Zooming into SSE (large share of the runtime)





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

Ziogas, et al.: A Data-Centric Approach to Extreme-Scale Dissipative Quantum Transport Simulations, to appear at SC19

Additional interesting performance insights

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

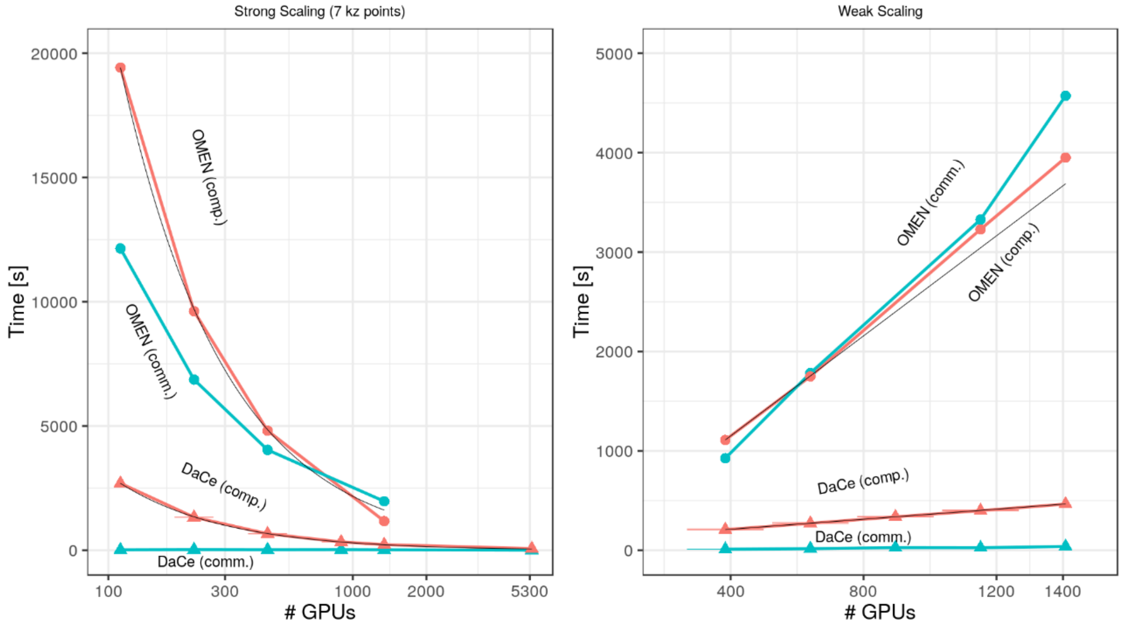
Variant	GF			SSE		
	Tflop	Time [s]	% Peak	Tflop	Time [s]	% Peak
OMEN	174.0	144.14	23.2%	63.6	965.45	1.3%
Python	174.0	1,342.77	2.5%	63.6	30,560.13	0.2%
DaCe	174.0	**111.25**	**30.1%**	31.8	**29.93**	**20.4%**

Piz Daint single node (P100)

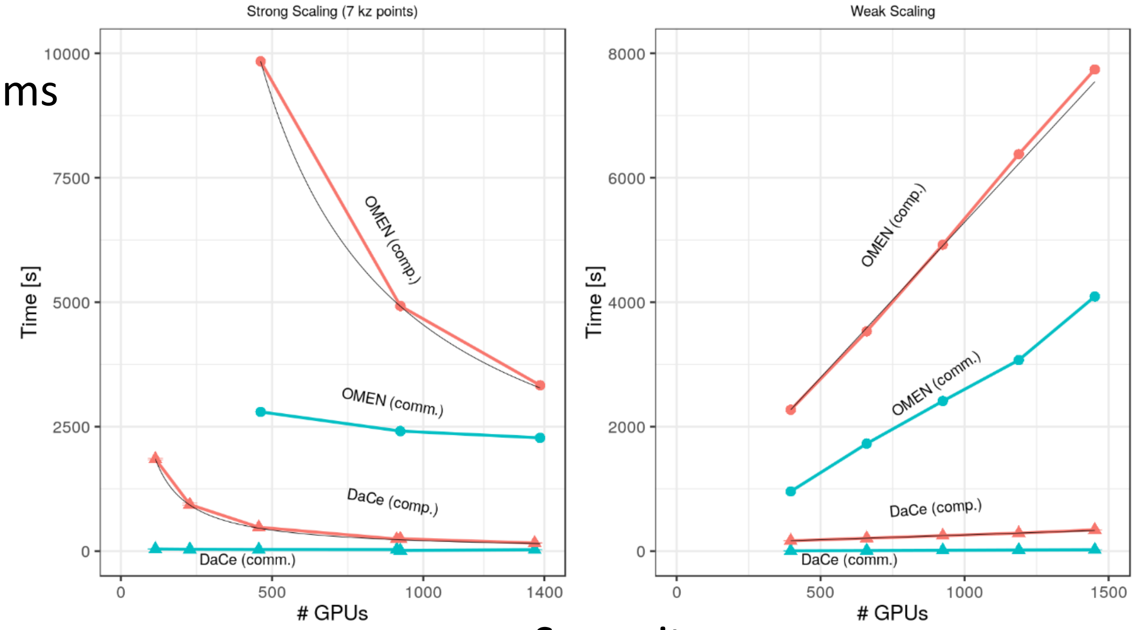
cuBLAS can be very inefficient (well, unless you floptimize)

GPU	cuBLAS Gflop	cuBLAS Time	cuBLAS % Peak (Useful)	DaCe (SBSMM) Gflop	DaCe (SBSMM) Time	DaCe (SBSMM) % 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)



5k atoms

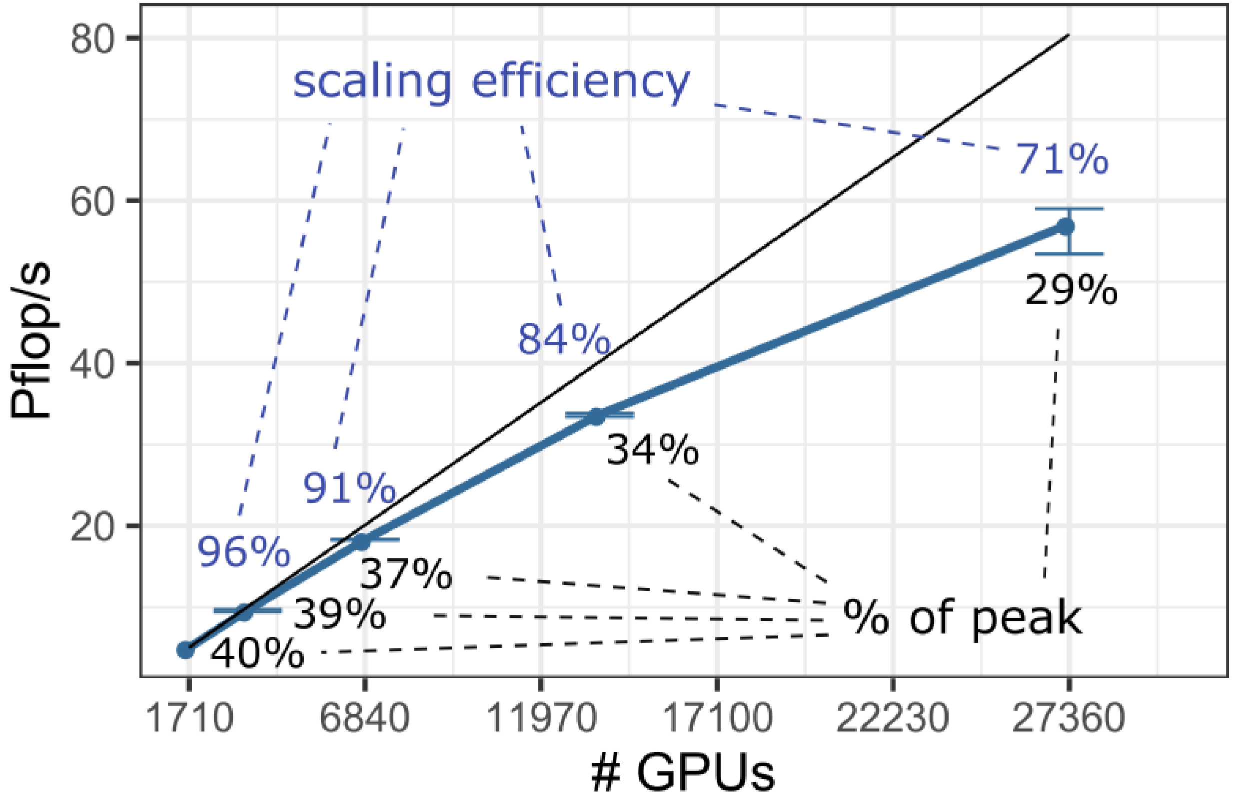


Piz Daint

Summit

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

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

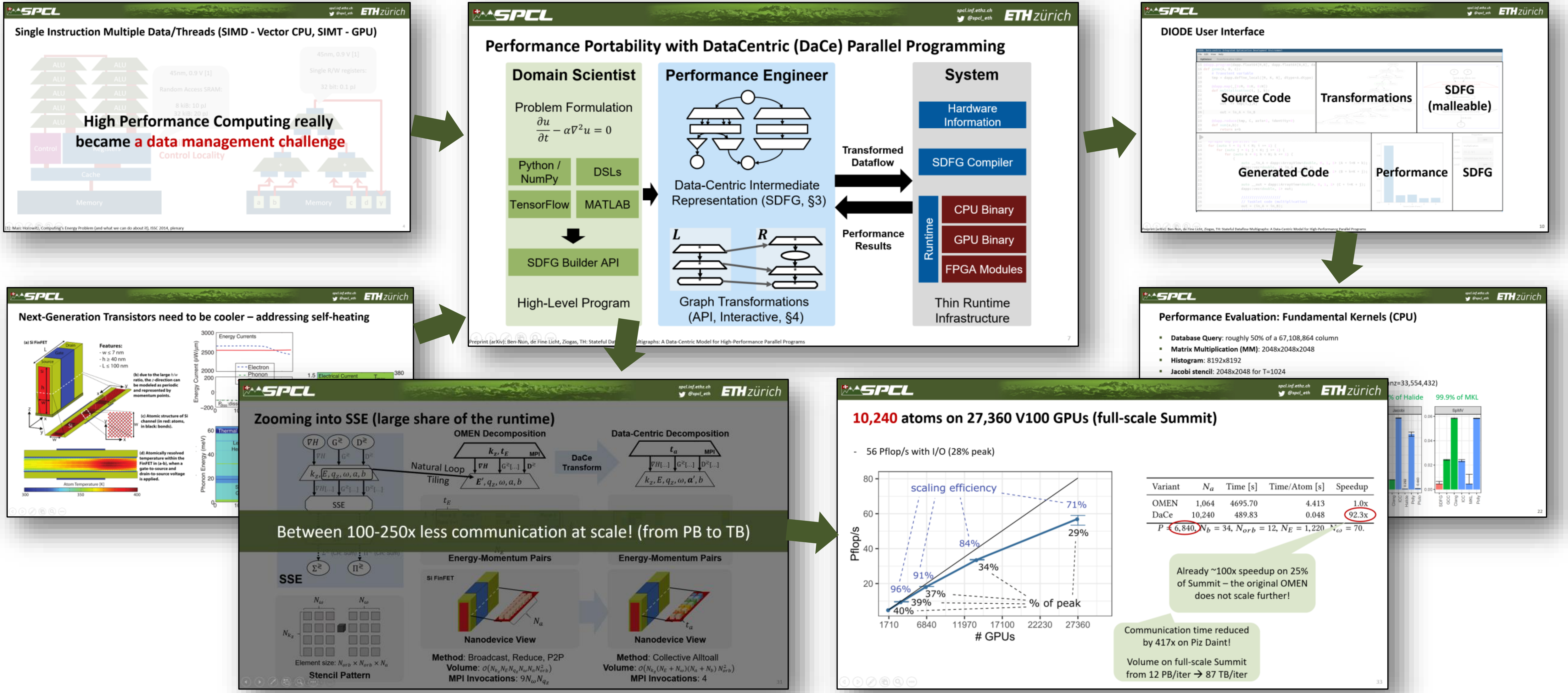
Already ~100x speedup on 25% of Summit – the original OMEN does not scale further!

Communication time reduced by 417x on Piz Daint!

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

Ziogas, et al.: A Data-Centric Approach to Extreme-Scale Dissipative Quantum Transport Simulations, to appear at SC19

Overview and wrap-up



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