A. N. ZIOGAS, T. SCHNEIDER, T. BEN-NUN, A. CALOTOIU, T. DE MATTEIS, J. DE FINE LICHT, L. LAVARINI, T. HOEFLER

Productivity, Portability, Performance: Data-Centric Python

Python: The Scientific Language

IP[y]:
IPython

jupyter

seaborn

matplotlib

SciPy

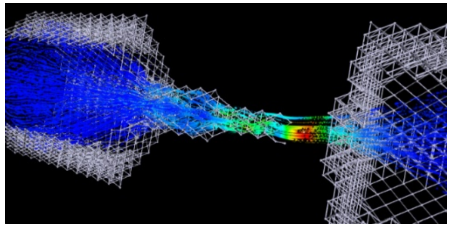
pandas

SymPy

NetworkX
Network Analysis in Python

MD ANALYSIS

GridTools



TensorFlow

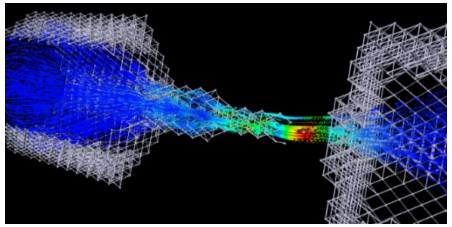
PyTorch

Vast software ecosystem → **Productivity**

Python: An HPC language?

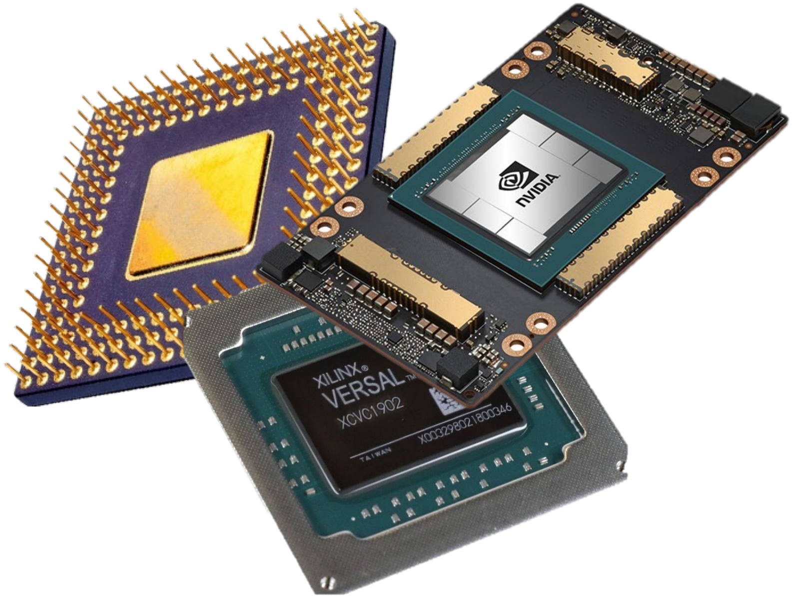
MD ANALYSIS

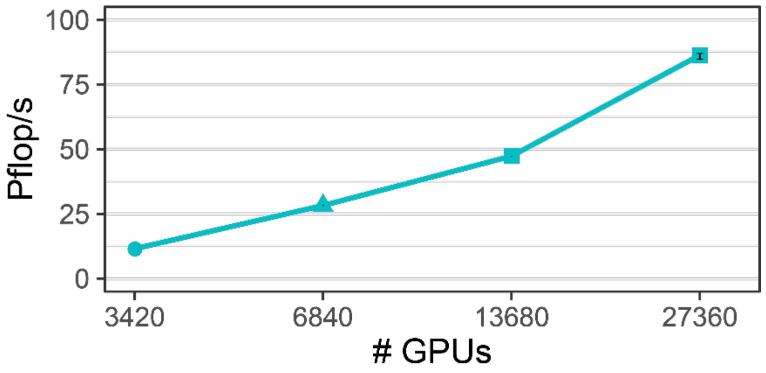
GridTools



TensorFlow

PyTorch





Productivity

Portability

Performance

Productivity and Performance

C Code

```c
int i, j, k;
for (i = 0; i < _PB_NI; i++) {
  for (j = 0; j < _PB_NJ; j++)
  C[i][j] *= beta;
  for (k = 0; k < _PB_NK; k++) {
    for (j = 0; j < _PB_NJ; j++)
    C[i][j] += alpha * A[i][k] * B[k][j];
  }
}
```

Python/NumPy Code

```python
C[:] = alpha * A @ B + beta * C
```

- 🟡 Explicit matrix product operator
- 🟠 Implicit scalar-array operations
- 🔴 Implicit array-array operations

Productivity and Performance

C Code

```c
int i, j, k;
for (i = 0; i < _PB_NI; i++) {
  for (j = 0; j < _PB_NJ; j++)
  C[i][j] *= beta;
  for (k = 0; k < _PB_NK; k++) {
    for (j = 0; j < _PB_NJ; j++)
    C[i][j] += alpha * A[i][k] * B[k][j];
  }
}
```

Python/NumPy Code

```python
C[:] = alpha * A @ B + beta * C
```

- ● Explicit matrix product operator
- ● Implicit scalar-array operations
- ● Implicit array-array operations

	CPython	Numba	Pythran	GCC	ICC		CuPy	
GEMM	1.0x	**1.4x**	**1.4x**	0.008x	0.2x		**17.0x**	
2MM	1.0x	**1.5x**	1.3x	0.001x	0.06x		**12.9x**	
3MM	1.0x	**1.6x**	1.2x	0.0006x	0.9x		**9.3x**	

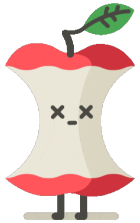
2x Intel Xeon Gold 6130 CPU (32 cores)

Nvidia V100 GPU

Productivity and Performance

C Code

```
int i, j, k;
for (i = 0; i < _PB_NI; i++) {
  for (j = 0; j < _PB_NJ; j++)
  C[i][j] *= beta;
  for (k = 0; k < _PB_NK; k++) {
    for (j = 0; j < _PB_NJ; j++)
    C[i][j] += alpha * A[i][k] * B[k][j];
  }
}
```

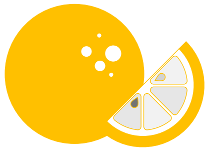


naive algorithm

Python/NumPy Code

```
C[:] = alpha * A @ B + beta * C
```

- ● Explicit matrix product operator
- ● Implicit scalar-array operations
- ● Implicit array-array operations



abstracts away BLAS call

	CPython	Numba	Pythran	GCC	ICC		CuPy	
GEMM	1.0x	**1.4x**	**1.4x**	0.008x	0.2x		**17.0x**	
2MM	1.0x	**1.5x**	1.3x	0.001x	0.06x		**12.9x**	
3MM	1.0x	**1.6x**	1.2x	0.0006x	0.9x		**9.3x**	

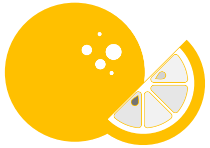
2x Intel Xeon Gold 6130 CPU (32 cores)

Nvidia V100 GPU

Productivity and Performance

C Code

```
cblas_dgemm(
  CblasRowMajor,
  CblasNoTrans, CblasNoTrans,
  _PB_NI, _PB_NJ, _PB_NK,
  alpha, A, _PB_NK, B, _PB_NJ,
  beta, C, _PB_NJ
);
```

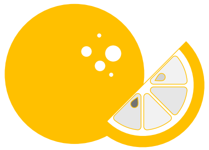


BLAS call

Python/NumPy Code

```
C[:] = alpha * A @ B + beta * C
```

- ● Explicit matrix product operator
- ● Implicit scalar-array operations
- ● Implicit array-array operations



abstracts away BLAS call

	CPython	Numba	Pythran	GCC	ICC		CuPy	
GEMM	1.0x	**1.4x**	**1.4x**	0.008x	0.2x		**17.0x**	
2MM	1.0x	**1.5x**	1.3x	0.001x	0.06x		**12.9x**	
3MM	1.0x	**1.6x**	1.2x	0.0006x	0.9x		**9.3x**	

2x Intel Xeon Gold 6130 CPU (32 cores)

Nvidia V100 GPU

Productivity or Performance

C Code

```c
int t, i, j;
for(t = 0; t < _PB_TMAX; t++) {
  for (j = 0; j < _PB_NY; j++)
    ey[0][j] = _fict_[t];
  for (i = 1; i < _PB_NX; i++)
    for (j = 0; j < _PB_NY; j++)
      ey[i][j] = ey[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i-1][j]);
  for (i = 0; i < _PB_NX; i++)
    for (j = 1; j < _PB_NY; j++)
      ex[i][j] = ex[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i][j-1]);
  for (i = 0; i < _PB_NX - 1; i++)
    for (j = 0; j < _PB_NY - 1; j++)
      hz[i][j] = hz[i][j] - SCALAR_VAL(0.7)*  (ex[i][j+1] - ex[i][j] +
                       ey[i+1][j] - ey[i][j]);
}
```

Python/NumPy Code

```python
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

Productivity or Performance

C Code

```
int t, i, j;
for(t = 0; t < _PB_TMAX; t++) {
  for (j = 0; j < _PB_NY; j++)
    ey[0][j] = _fict_[t];
  for (i = 1; i < _PB_NX; i++)
    for (j = 0; j < _PB_NY; j++)
      ey[i][j] = ey[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i-1][j]);
  for (i = 0; i < _PB_NX; i++)
    for (j = 1; j < _PB_NY; j++)
      ex[i][j] = ex[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i][j-1]);
  for (i = 0; i < _PB_NX - 1; i++)
    for (j = 0; j < _PB_NY - 1; j++)
      hz[i][j] = hz[i][j] - SCALAR_VAL(0.7)*  (ex[i][j+1] - ex[i][j] +
                       ey[i+1][j] - ey[i][j]);
}
```

Python/NumPy Code

```
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

	CPython	Numba	Pythran	GCC	ICC		CuPy	
FDTD-2D	1.0x	4.1x	1.3x	3.7x	**41.3x**		**42.4x**	
Jacobi-2D	1.0x	18.2x	21.8x	7.1x	**58.6x**		**75.2x**	
Heat-3D	1.0x	50.1x	2.3x	24.0x	**179.0x**		**71.0x**	

2x Intel Xeon Gold 6130 CPU (32 cores)

Nvidia V100 GPU

Productivity or Performance

C Code

```c
int t, i, j;
for(t = 0; t < _PB_TMAX; t++) {
  for (j = 0; j < _PB_NY; j++)
    ey[0][j] = _fict_[t];
  for (i = 1; i < _PB_NX; i++)
    for (j = 0; j < _PB_NY; j++)
      ey[i][j] = ey[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i-1][j]);
  for (i = 0; i < _PB_NX; i++)
    for (j = 1; j < _PB_NY; j++)
      ex[i][j] = ex[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i][j-1]);
  for (i = 0; i < _PB_NX - 1; i++)
    for (j = 0; j < _PB_NY - 1; j++)
      hz[i][j] = hz[i][j] - SCALAR_VAL(0.7)*  (ex[i][j+1] - ex[i][j] +
                                 ey[i+1][j] - ey[i][j]);
}
```

Python/NumPy Code

```python
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

```python
tmp1 = hz[:, 1:] - hz[:, :-1]
tmp2 = 0.5 * tmp1
ex[:, 1:] -= tmp2
```

extraneous intermediate results

operations can be fused as shown in C-code

Productivity or Performance

C Code

```
int t, i, j;
for(t = 0; t < _PB_TMAX; t++) {
  for (j = 0; j < _PB_NY; j++)
    ey[0][j] = _fict_[t];
  for (i = 1; i < _PB_NX; i++)
    for (j = 0; j < _PB_NY; j++)
      ey[i][j] = ey[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i-1][j]);
  for (i = 0; i < _PB_NX; i++)
    for (j = 1; j < _PB_NY; j++)
      ex[i][j] = ex[i][j] - SCALAR_VAL(0.5)*(hz[i][j]-hz[i][j-1]);
  for (i = 0; i < _PB_NX - 1; i++)
    for (j = 0; j < _PB_NY - 1; j++)
      hz[i][j] = hz[i][j] - SCALAR_VAL(0.7)*  (ex[i][j+1] - ex[i][j] +
                                   ey[i+1][j] - ey[i][j]);
}
```

Python/NumPy Code

```
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

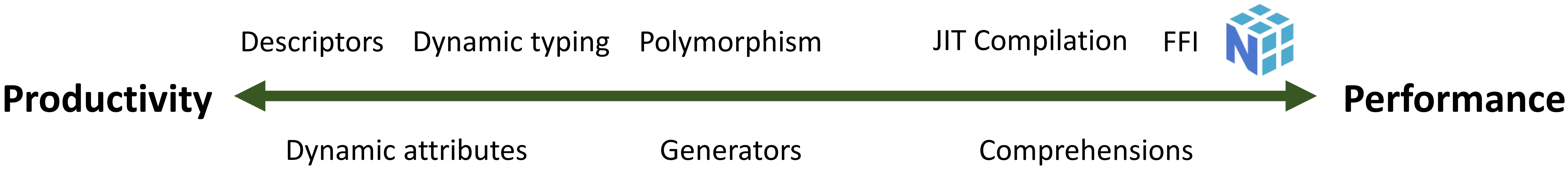
```
tmp1 = hz[:, 1:] - hz[:, :-1]
tmp2 = 0.5 * tmp1
ex[:, 1:] -= tmp2
```

extraneous intermediate results

operations can be fused as shown in C-code

Python doesn't analyze data movement

Python



Descriptors Dynamic typing Polymorphism JIT Compilation FFI

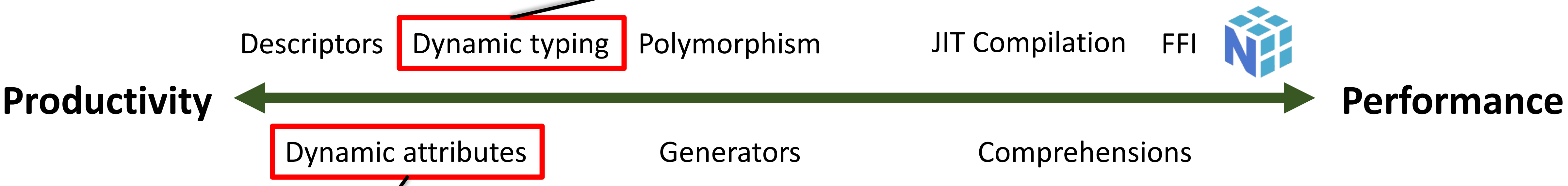
Productivity **Performance**

Dynamic attributes Generators Comprehensions

Ben-Nun et al. “Bridging Productivity, Portability, and Performance with Data-Centric Python”, PyCon Sweden 2021.

Python

```
def func(a_boolean):
    if a_boolean:
        a = 5  # int
    else:
        a = 5 + 6j  # complex
```



Descriptors

Dynamic typing

Polymorphism

JIT Compilation

FFI

Productivity

Performance

Dynamic attributes

Generators

Comprehensions

```
class A:
    pass

def func():
    b = A()
    b.c = 5
```

Inhibit Analyzability of Data Movement

Ben-Nun et al. “Bridging Productivity, Portability, and Performance with Data-Centric Python”, PyCon Sweden 2021.

Data-Centric (DaCe) Python

```python
def func(a_boolean):
    a: complex
    if a_boolean:
        a = 5  # int
    else:
        a = 5 + 6j  # complex
```

Descriptors   Dynamic typing   JIT Polymorphism   JIT Compilation   FFI

Productivity ⟷ **Performance**

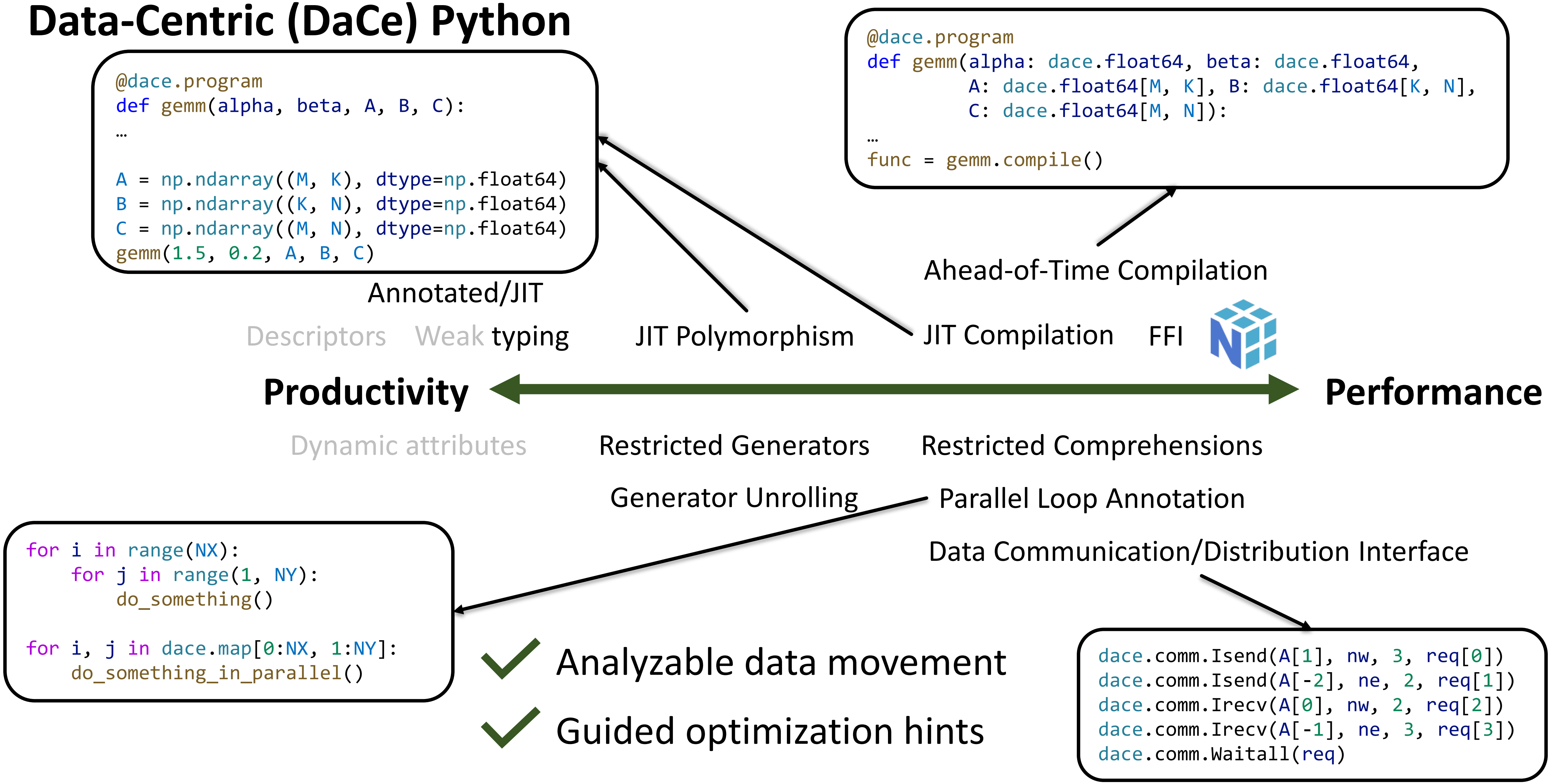
Dynamic attributes   Restricted Generators   Restricted Comprehensions

```python
class A:
  def __init__(self):
    self.c = 0
def func():
  b = A()
  b.c = 5
```

✓ Analyzable data movement

Ben-Nun et al. “Bridging Productivity, Portability, and Performance with Data-Centric Python”, PyCon Sweden 2021.

Data-Centric (DaCe) Python



```
@dace.program
def gemm(alpha, beta, A, B, C):
…

A = np.ndarray((M, K), dtype=np.float64)
B = np.ndarray((K, N), dtype=np.float64)
C = np.ndarray((M, N), dtype=np.float64)
gemm(1.5, 0.2, A, B, C)
```

```
@dace.program
def gemm(alpha: dace.float64, beta: dace.float64,
         A: dace.float64[M, K], B: dace.float64[K, N],
         C: dace.float64[M, N]):
…
func = gemm.compile()
```

Ahead-of-Time Compilation

Annotated/JIT

Descriptors Weak typing JIT Polymorphism JIT Compilation FFI

Productivity **Performance**

Dynamic attributes Restricted Generators Restricted Comprehensions

Generator Unrolling Parallel Loop Annotation

Data Communication/Distribution Interface

```
for i in range(NX):
    for j in range(1, NY):
        do_something()

for i, j in dace.map[0:NX, 1:NY]:
    do_something_in_parallel()
```

```
dace.comm.Isend(A[1], nw, 3, req[0])
dace.comm.Isend(A[-2], ne, 2, req[1])
dace.comm.Irecv(A[0], nw, 2, req[2])
dace.comm.Irecv(A[-1], ne, 3, req[3])
dace.comm.Waitall(req)
```

- ✓ Analyzable data movement
- ✓ Guided optimization hints

Ben-Nun et al. “Bridging Productivity, Portability, and Performance with Data-Centric Python”, PyCon Sweden 2021.

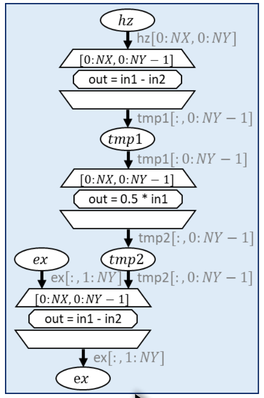
Data-Centric Workflow

Python/NumPy Code

```
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

Constrained to
Data-Centric Python

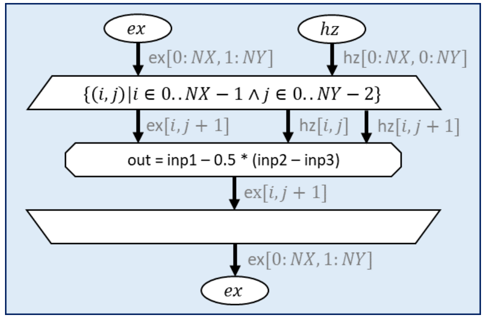
Data-Centric IR



Parsing the Python program’s AST

SDFG: Stateful DataFlow multiGraphs [1]

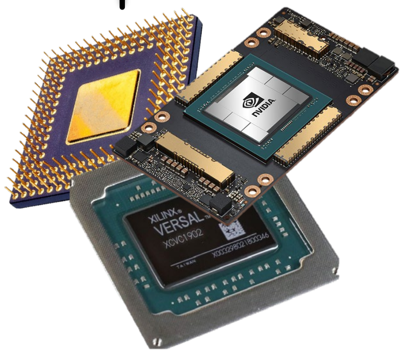
Optimized IR



Data-Centric Transformations
Device-specific Transformations
User-driven (API, GUI)
Auto-optimization heuristics

HW-specific code

```
#pragma omp parallel for
for (auto __i0 = 0; __i0 < NX; __i0 += 1) {
  for (auto __i1 = 0; __i1 < (NY - 1); __i1 += 1) {
    double __in1 = hz[(((NY * __i0) + __i1) + 1)];
    double __in2 = hz[((NY * __i0) + __i1)];
    double __in0 = ex[(((NY * __i0) + __i1) + 1)];
    double __out_1;
    ///////////////////
    // Tasklet code
    auto __out = (__in1 - __in2);
    auto __out_0 = (0.5 * __out);
    __out_1 = (__in0 - __out_0);
    ///////////////////
    ex[(((NY * __i0) + __i1) + 1)] = __out_1;
  }
}
```



---

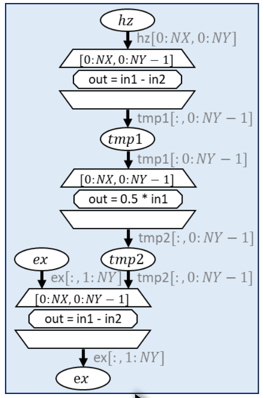
[1] Ben-Nun et al. “Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures”, SC’19.

Data-Centric Workflow

Python/NumPy Code

```
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

Data-Centric IR



Constrained to Data-Centric Python

Parsing the Python program's AST

SDFG: Stateful DataFlow multiGraphs [1]

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

Data-Centric Model and SDFG

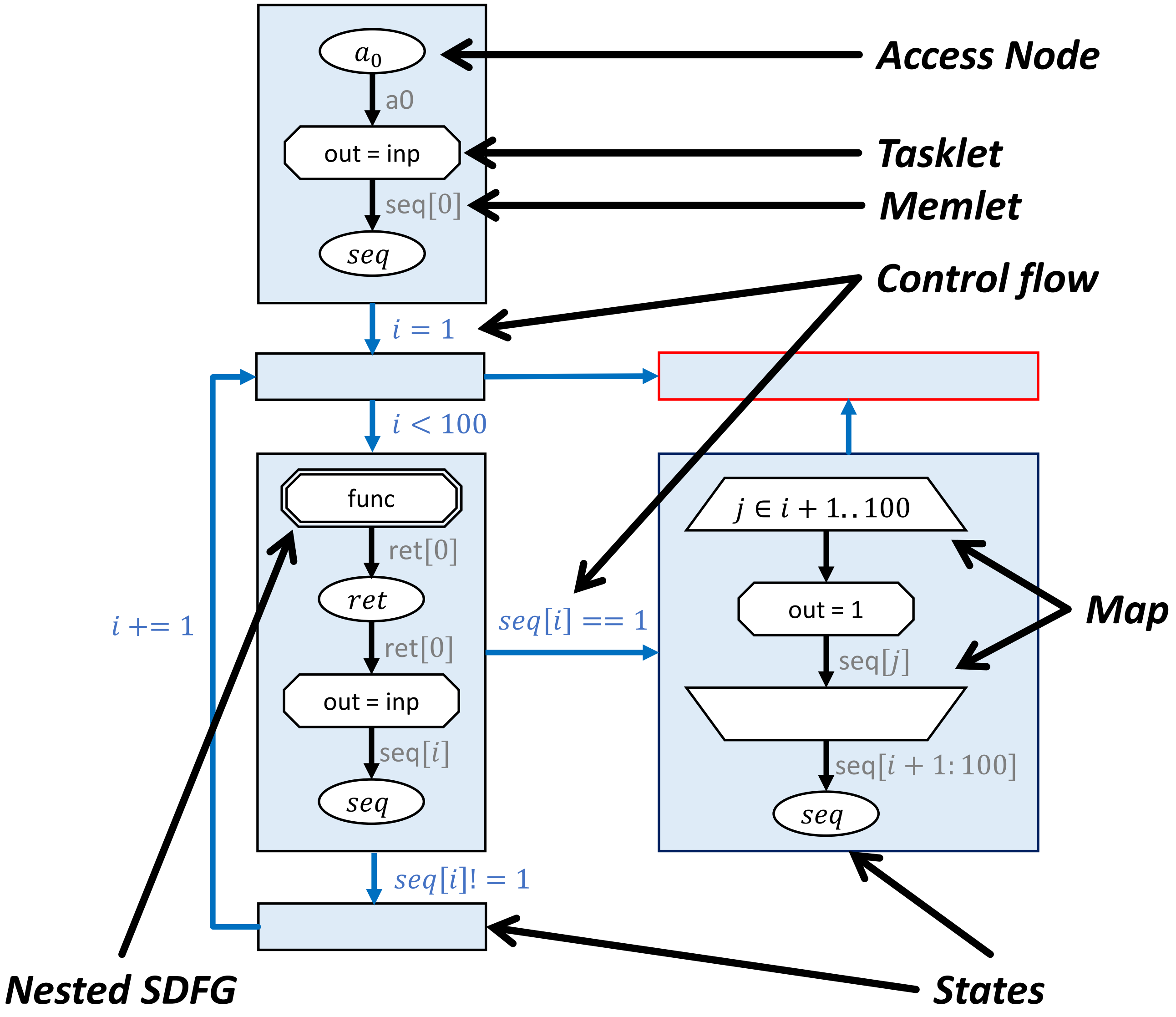
```
@dace.program
def top_level_func():
  seq = np.ndarray((100,), np.int32)
  seq[0] = a0
  for i in range(1, 100):
    seq[i] = func(seq[i-1])
    if seq[i] == 1:
      seq[i+1:] = 1
      break
return seq
```

Separating containers from computation

Explicit data movement

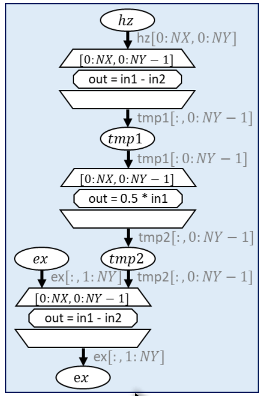
Control flow when necessary

Coarsening

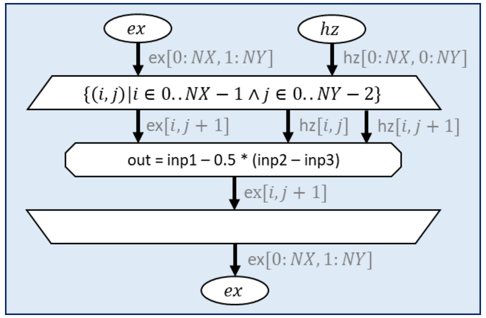


Data-Centric Workflow

Data-Centric IR



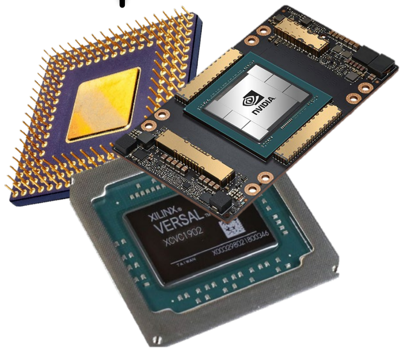
Optimized IR



HW-specific code

```
#pragma omp parallel for
for (auto __i0 = 0; __i0 < NX; __i0 += 1) {
  for (auto __i1 = 0; __i1 < (NY - 1); __i1 += 1) {
    double __in1 = hz[(((NY * __i0) + __i1) + 1)];
    double __in2 = hz[((NY * __i0) + __i1)];
    double __in0 = ex[(((NY * __i0) + __i1) + 1)];
    double __out_1;
    ///////////////////
    // Tasklet code
    auto __out = (__in1 - __in2);
    auto __out_0 = (0.5 * __out);
    __out_1 = (__in0 - __out_0);
    ///////////////////
    ex[(((NY * __i0) + __i1) + 1)] = __out_1;
  }
}
```

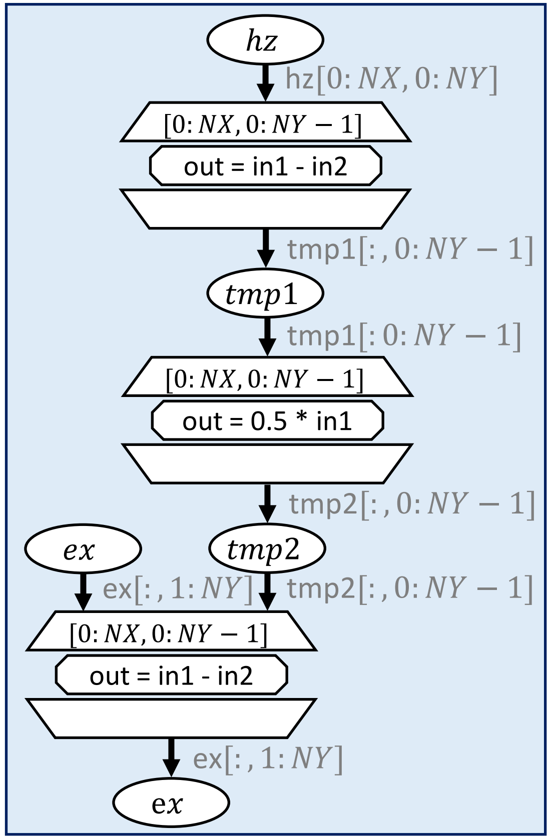
Data-Centric Transformations
Device-specific Transformations
User-driven (API, GUI)
Auto-optimization heuristics



SDFG: Stateful DataFlow multiGraphs

Data-Centric Optimizations

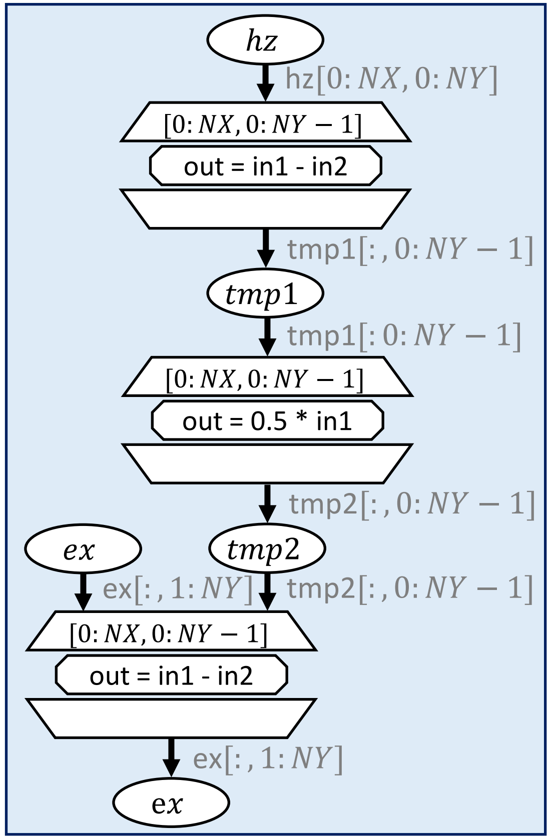
```
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```



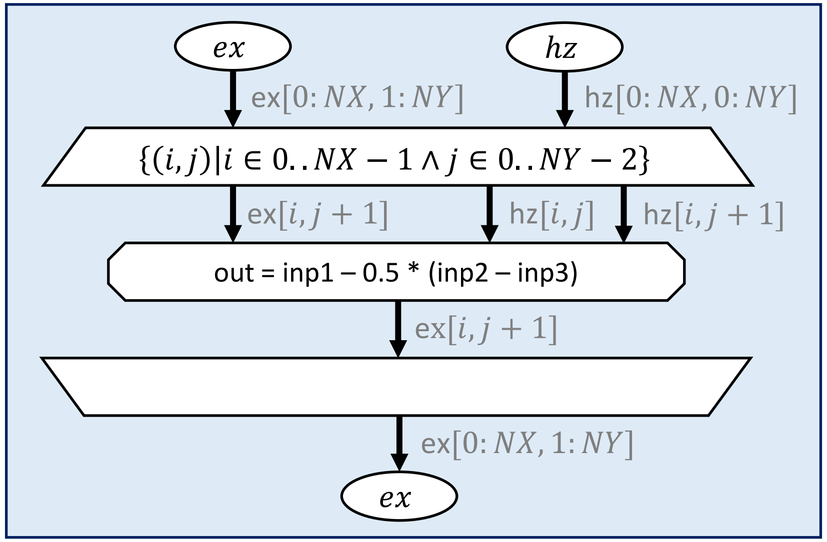
Data-Centric Optimizations

```
for t in range(TMAX):
  ey[0, :] = _fict_[t]
  ey[1:, :] -= 0.5 * (hz[1:, :] - hz[:-1, :])
  ex[:, 1:] -= 0.5 * (hz[:, 1:] - hz[:, :-1])
  hz[:-1, :-1] -= 0.7 * (
    ex[:-1, 1:] - ex[:-1, :-1] +
    ey[1:, :-1] - ey[:-1, :-1])
```

```
#pragma omp parallel for
for (auto __i0 = 0; __i0 < NX; __i0 += 1) {
  for (auto __i1 = 0; __i1 < (NY - 1); __i1 += 1) {
    double __in1 = hz[(((NY * __i0) + __i1) + 1)];
    double __in2 = hz[((NY * __i0) + __i1)];
    double __in0 = ex[(((NY * __i0) + __i1) + 1)];
    double __out_1;
    ///////////////////
    // Tasklet code
    auto __out = (__in1 - __in2);
    auto __out_0 = (0.5 * __out);
    __out_1 = (__in0 - __out_0);
    ///////////////////
    ex[(((NY * __i0) + __i1) + 1)] = __out_1;
  }
}
```

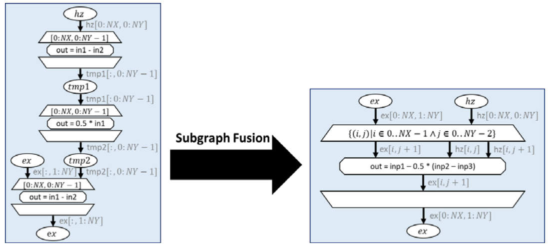


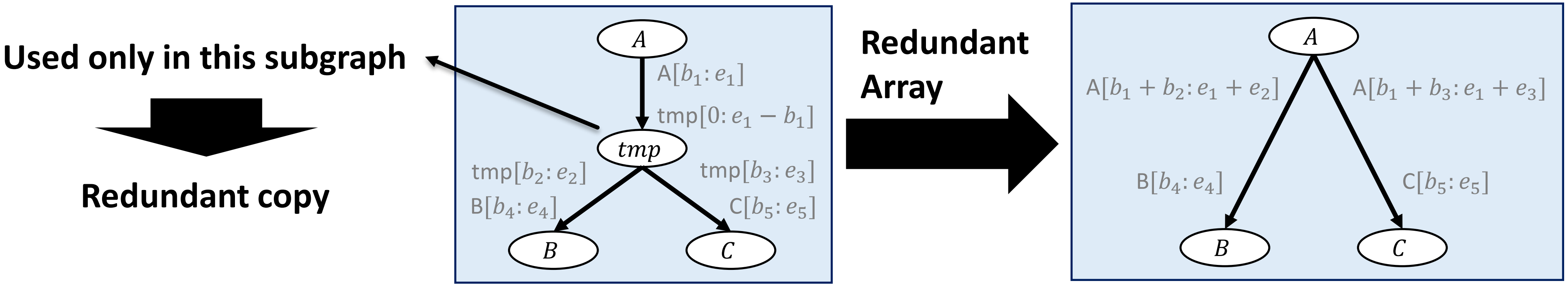
Subgraph Fusion



Data-Centric Optimizations

Subgraph Fusion





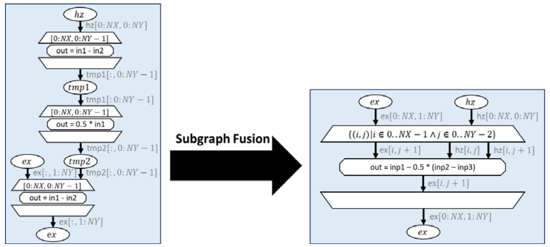
Used only in this subgraph

Redundant copy

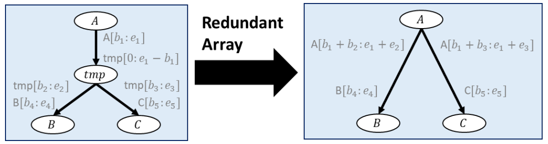
Redundant Array

Data-Centric Optimizations

Subgraph Fusion

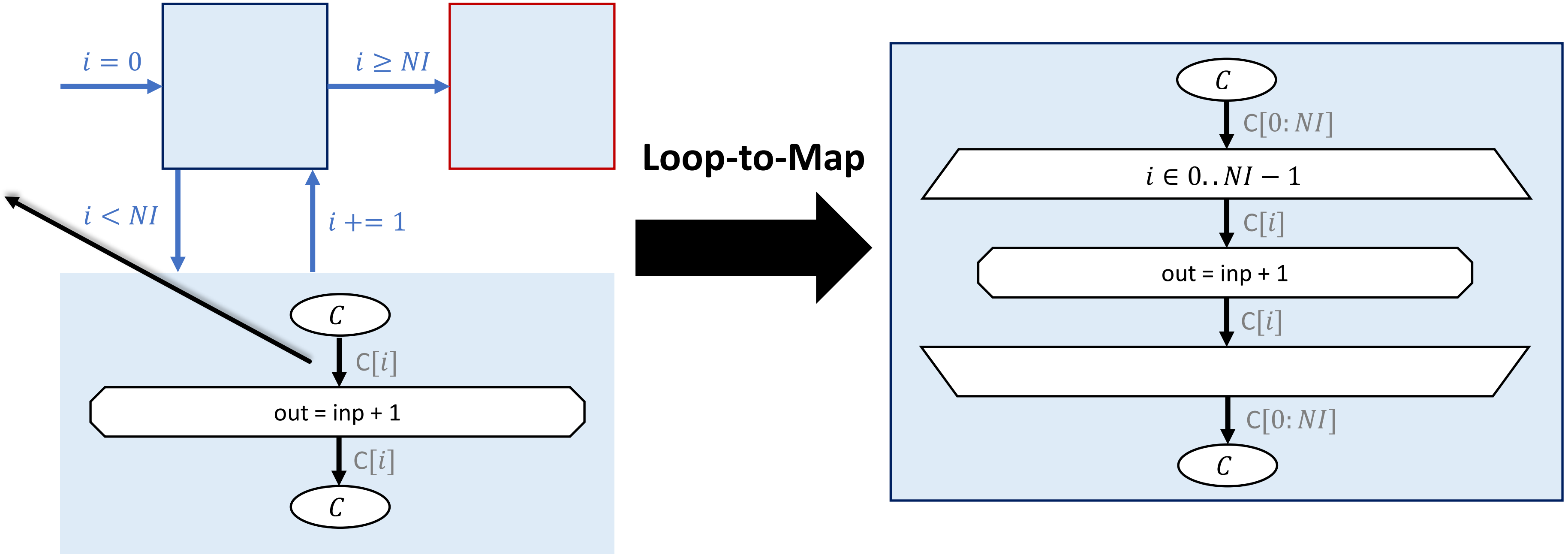


Redundant Array Elimination



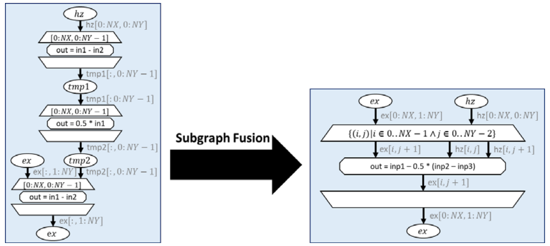
Independently parallel iterations

Can be executed in parallel

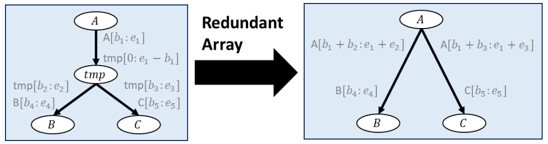


Data-Centric Optimizations

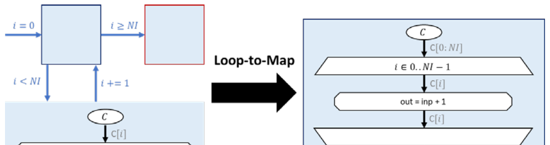
Subgraph Fusion

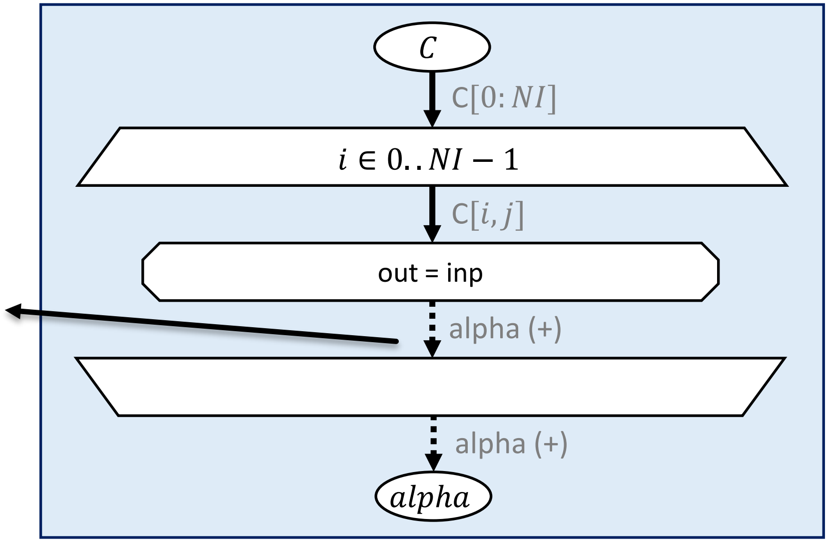


Redundant Array Elimination



Loop-to-Map

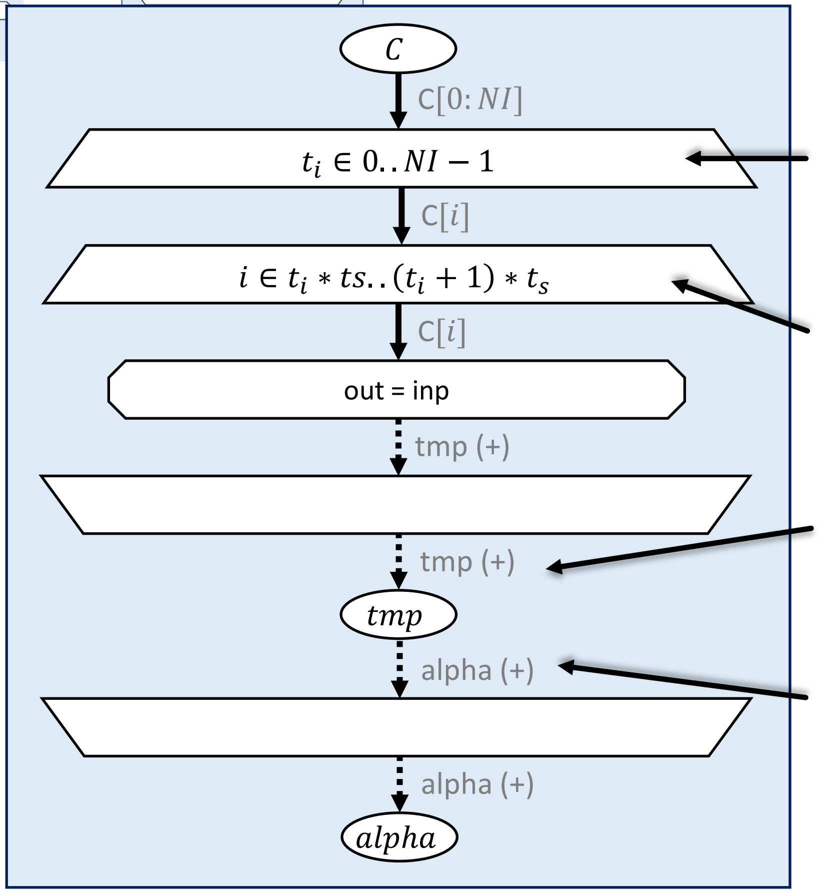




Write-Conflict Resolution Edge

Needs atomics Performance may be low

WCR Tiling



Parallel schedule

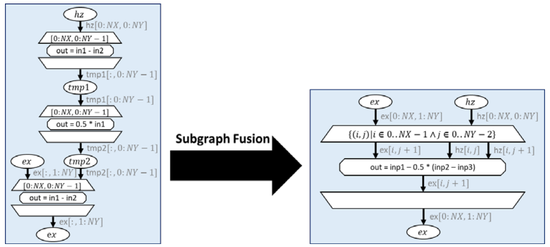
Sequential schedule

Doesn't need atomics

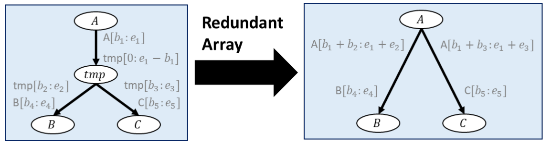
Needs atomics

Data-Centric Optimizations

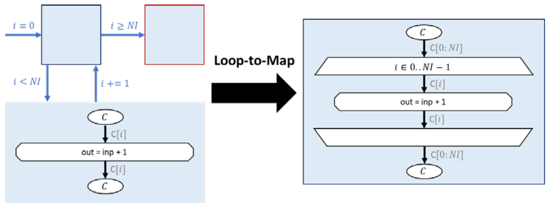
Subgraph Fusion



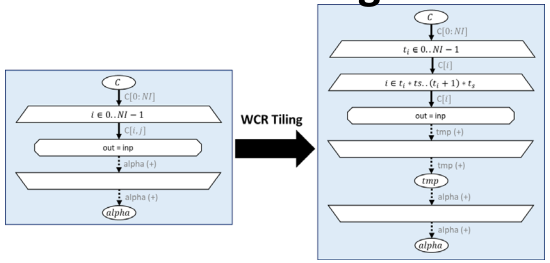
Redundant Array Elimination

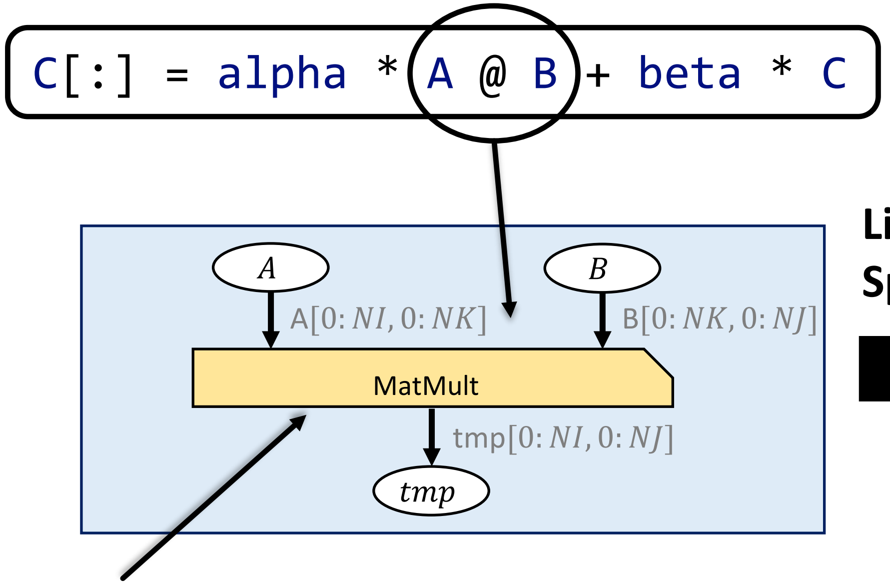


Loop-to-Map



WCR Tiling

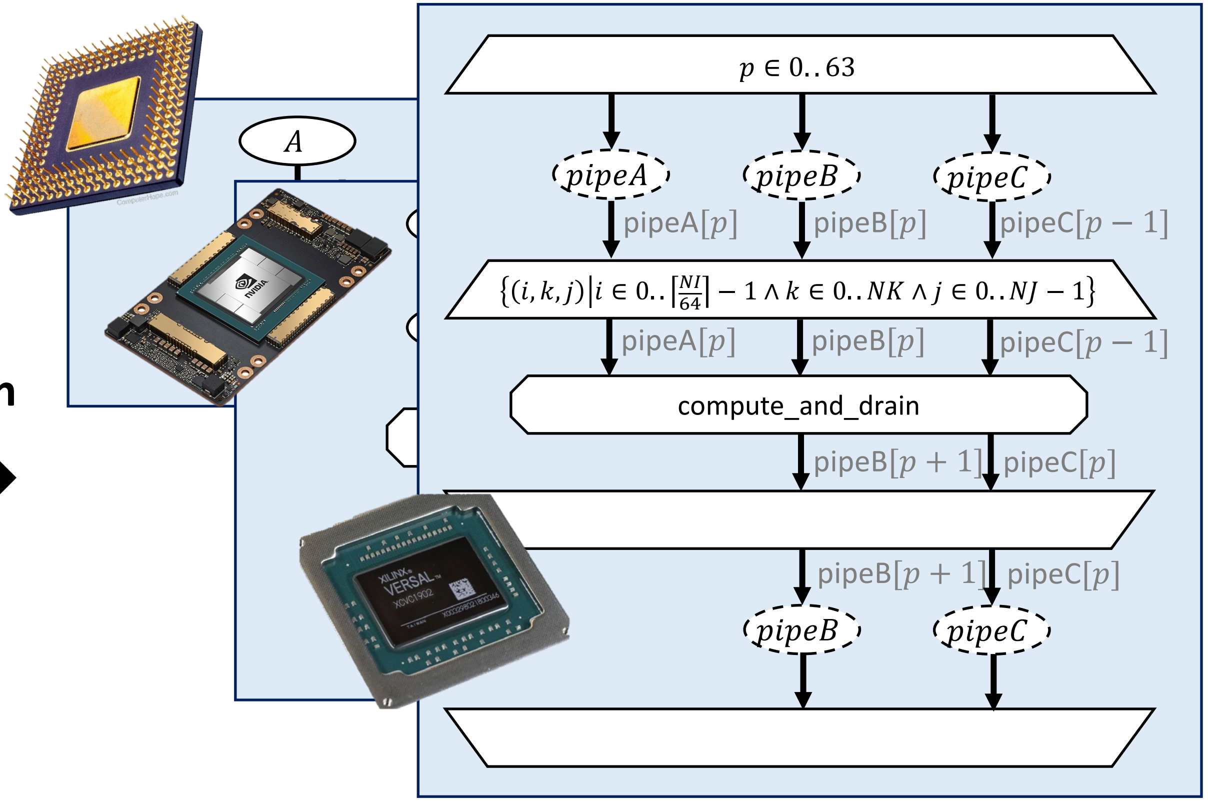




```
C[:] = alpha * A @ B + beta * C
```

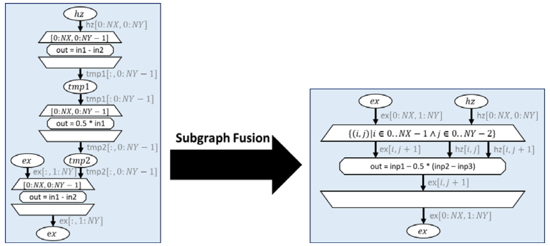
Library Node

Library Specialization

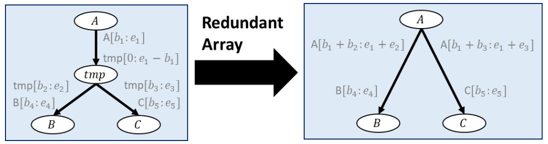


Data-Centric Optimizations

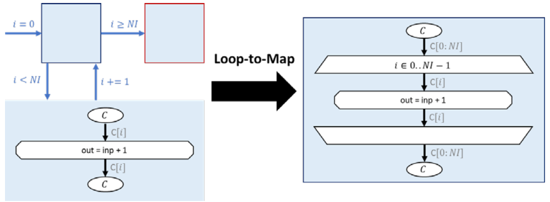
Subgraph Fusion



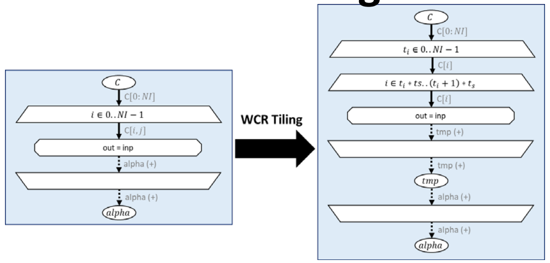
Redundant Array Elimination



Loop-to-Map

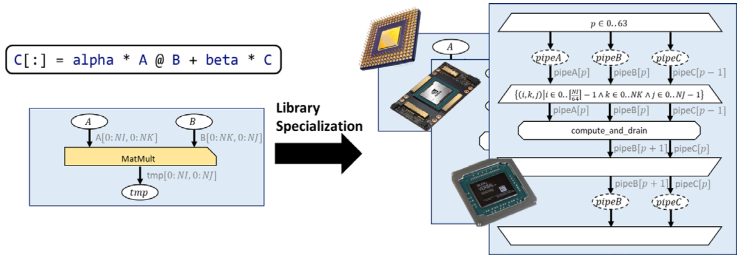


WCR Tiling



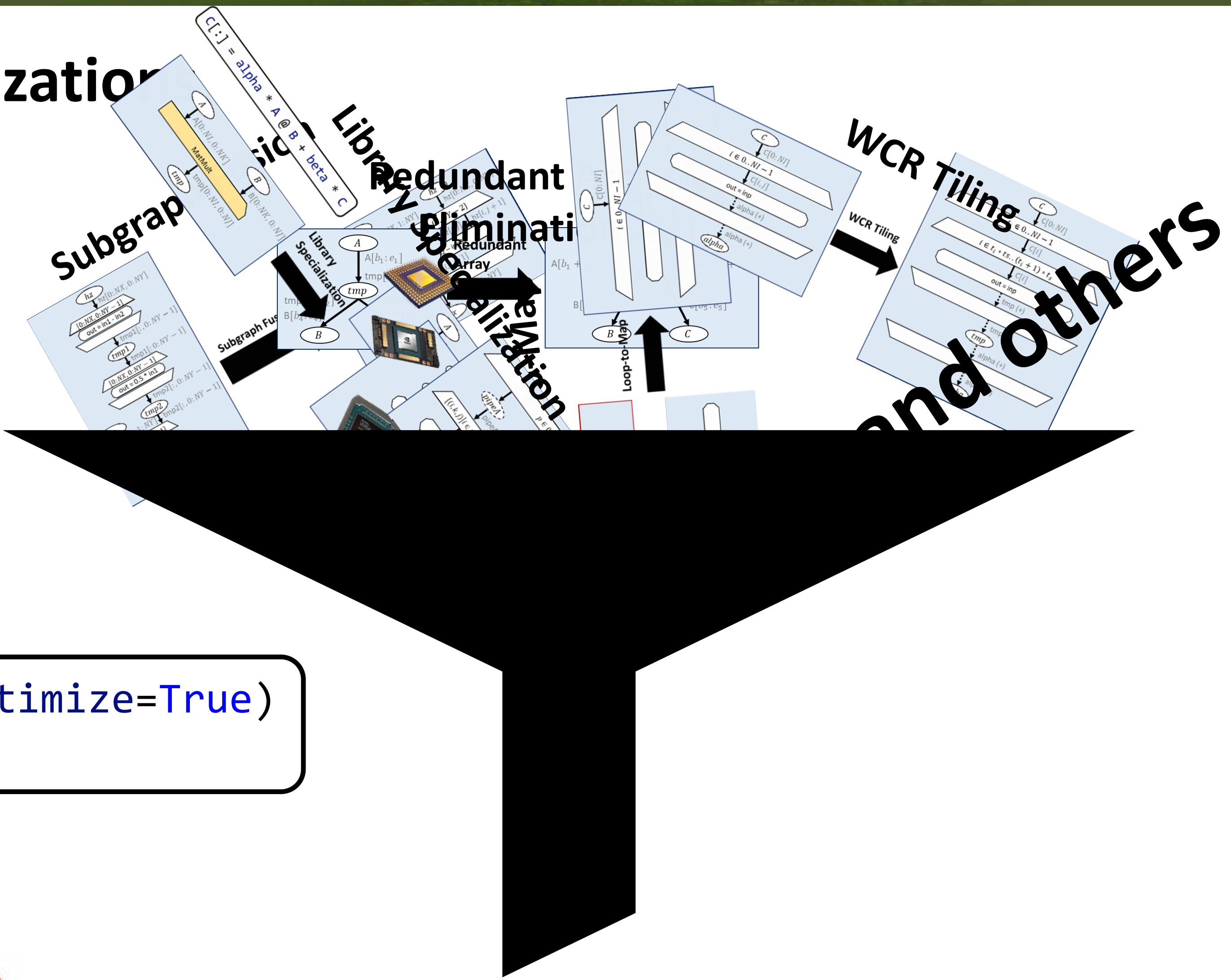
Library Specialization

```
C[:] = alpha * A @ B + beta * C
```

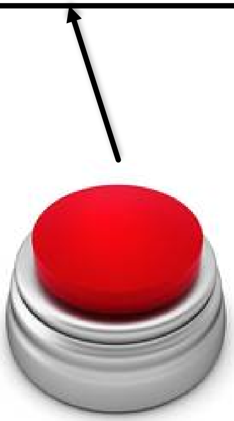


… and others

Data-Centric Optimization



```
@dace.program(auto_optimize=True)
def top_level_func():
```



Automatic Optimization Heuristics

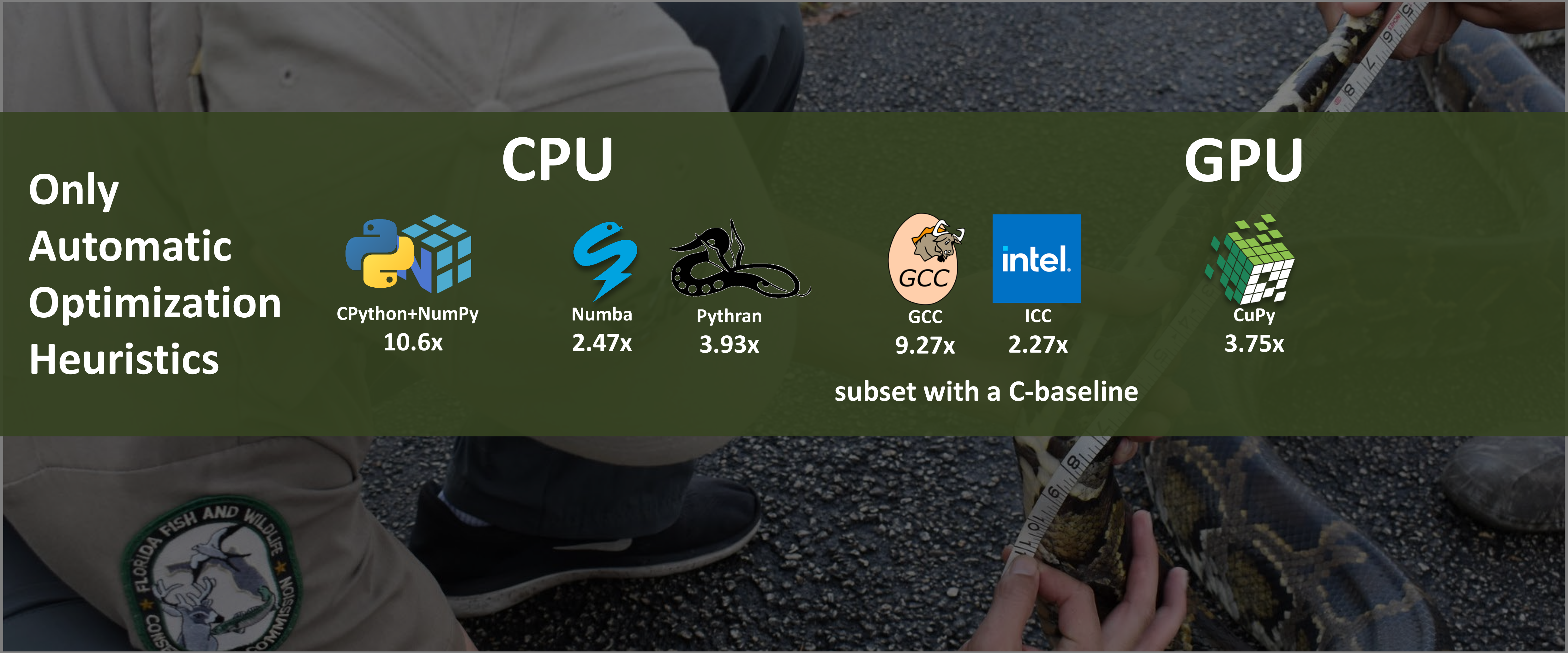
Auto-Optimizer Results

	CPython	Numba	Pythran	GCC	ICC	DaCe CPU	CuPy	DaCe GPU
GEMM	1.0x	1.4x	1.4x	0.008x	0.2x	2.3x	17.0x	17.0x
2MM	1.0x	1.5x	1.3x	0.001x	0.06x	2.4x	12.9x	13.0x
3MM	1.0x	1.6x	1.2x	0.0006x	0.9x	1.7x	9.3x	9.2x
FDTD-2D	1.0x	4.1x	1.3x	3.7x	41.3x	170.0x	42.4x	112.4x
Jacobi-2D	1.0x	18.2x	21.8x	7.1x	58.6x	56.2x	75.2x	477.0x
Heat-3D	1.0x	50.1x	2.3x	24.0x	179.0x	454.0x	71.0x	1200.0x

2x Intel Xeon Gold 6130 CPU (32 cores)

Nvidia V100 GPU

Measuring Python Performance with NPBench



Only Automatic Optimization Heuristics

CPU

CPython+NumPy	Numba	Pythran	GCC	ICC
10.6x	2.47x	3.93x	9.27x	2.27x

subset with a C-baseline

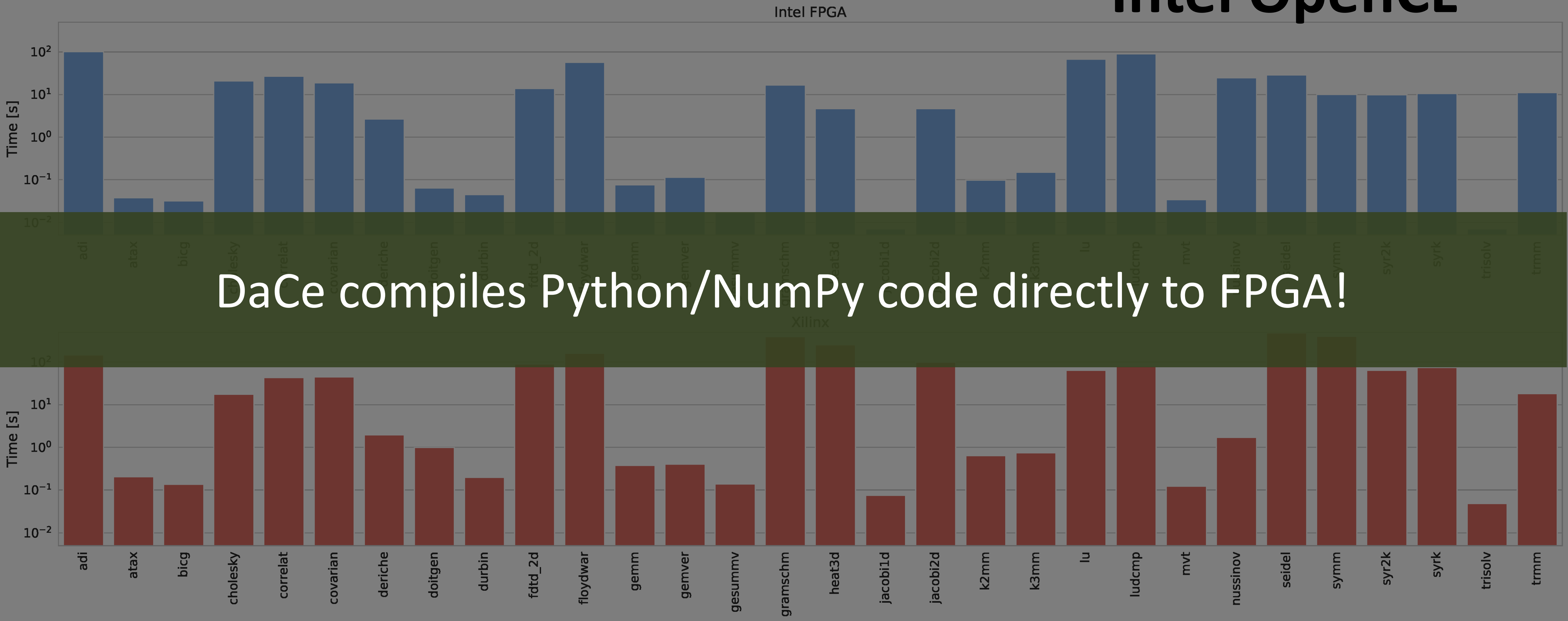
GPU

CuPy
3.75x

Measuring Burmese Python by Florida Fish and Wildlife licensed under CC BY-NC-ND 2.0

From Python to HLS and OpenCL

Intel OpenCL

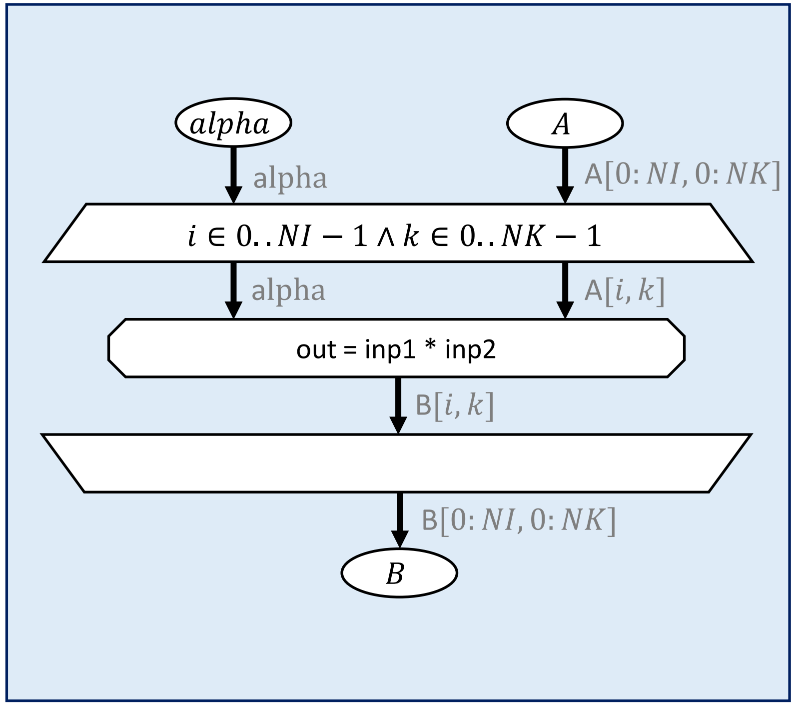


DaCe compiles Python/NumPy code directly to FPGA!

Xilinx HLS

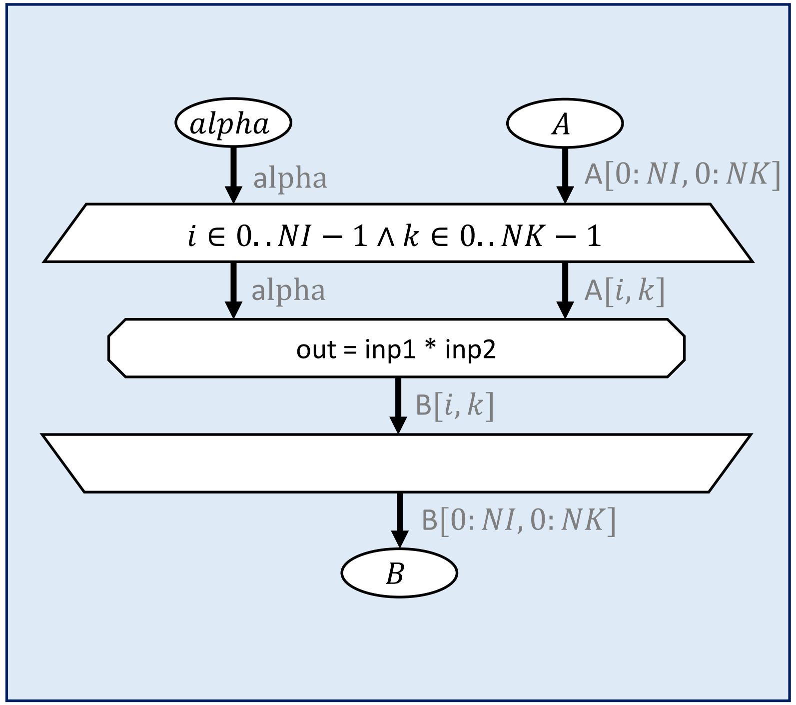
Transforming for Scale

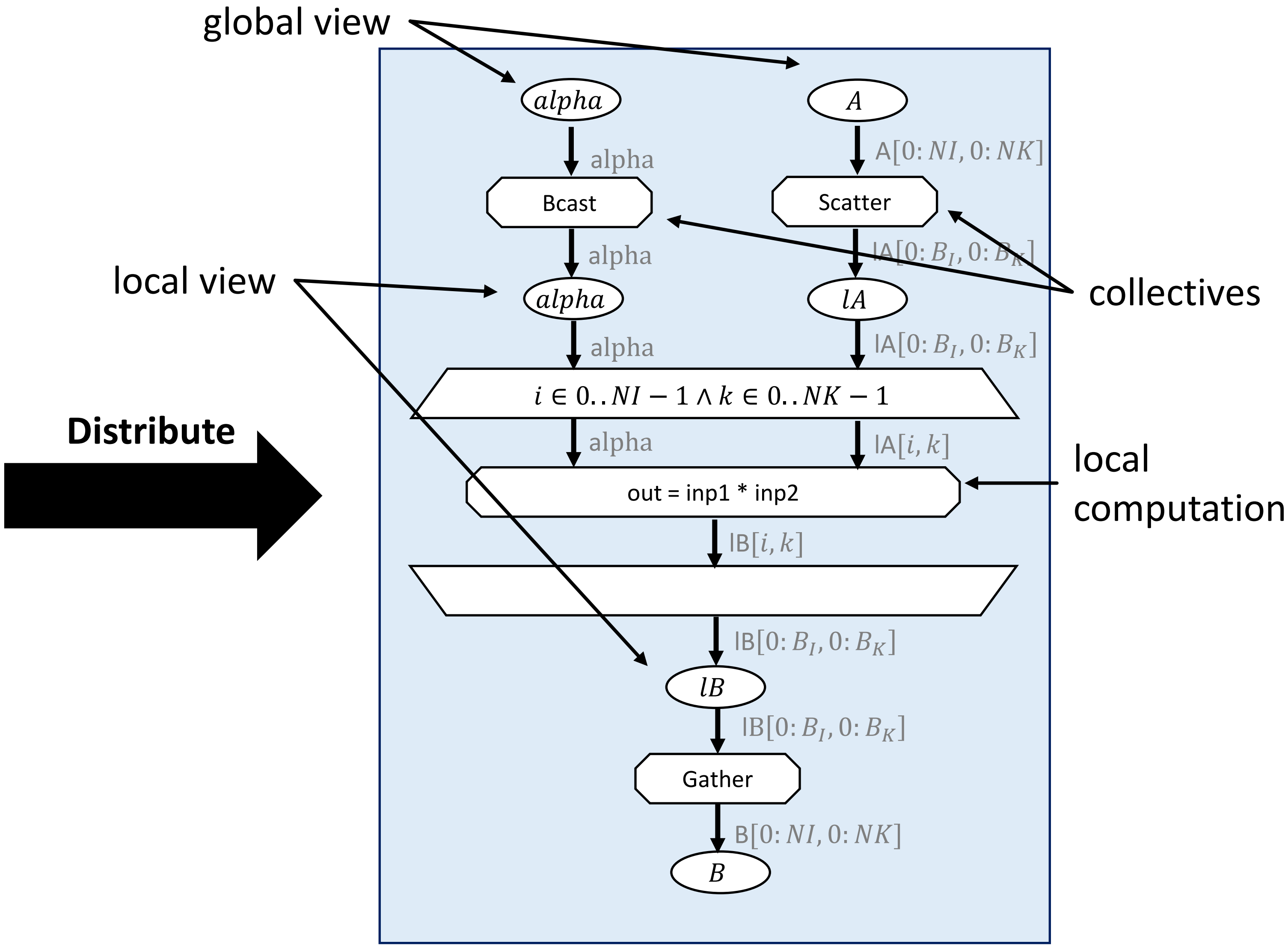
```
B[:] = alpha * A
```



Transforming for Scale

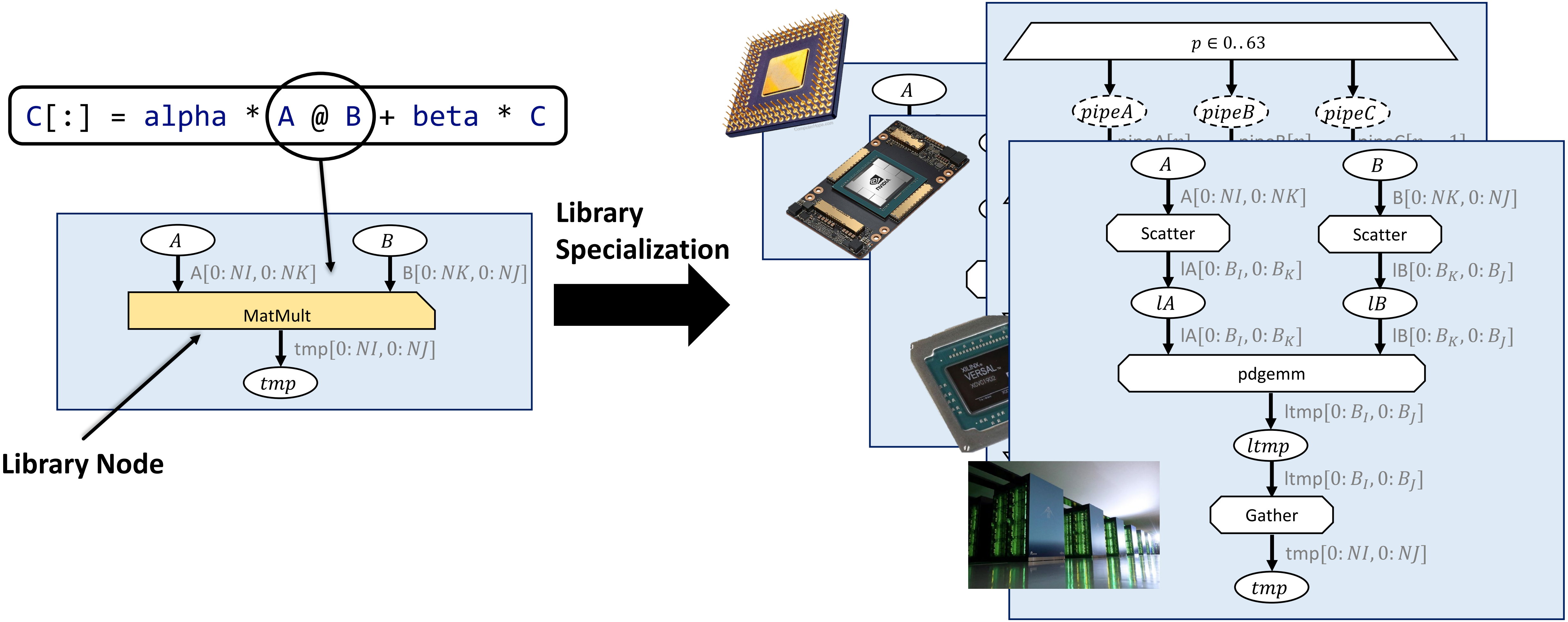
```
B[:] = alpha * A
```





Library Specialization for Scale

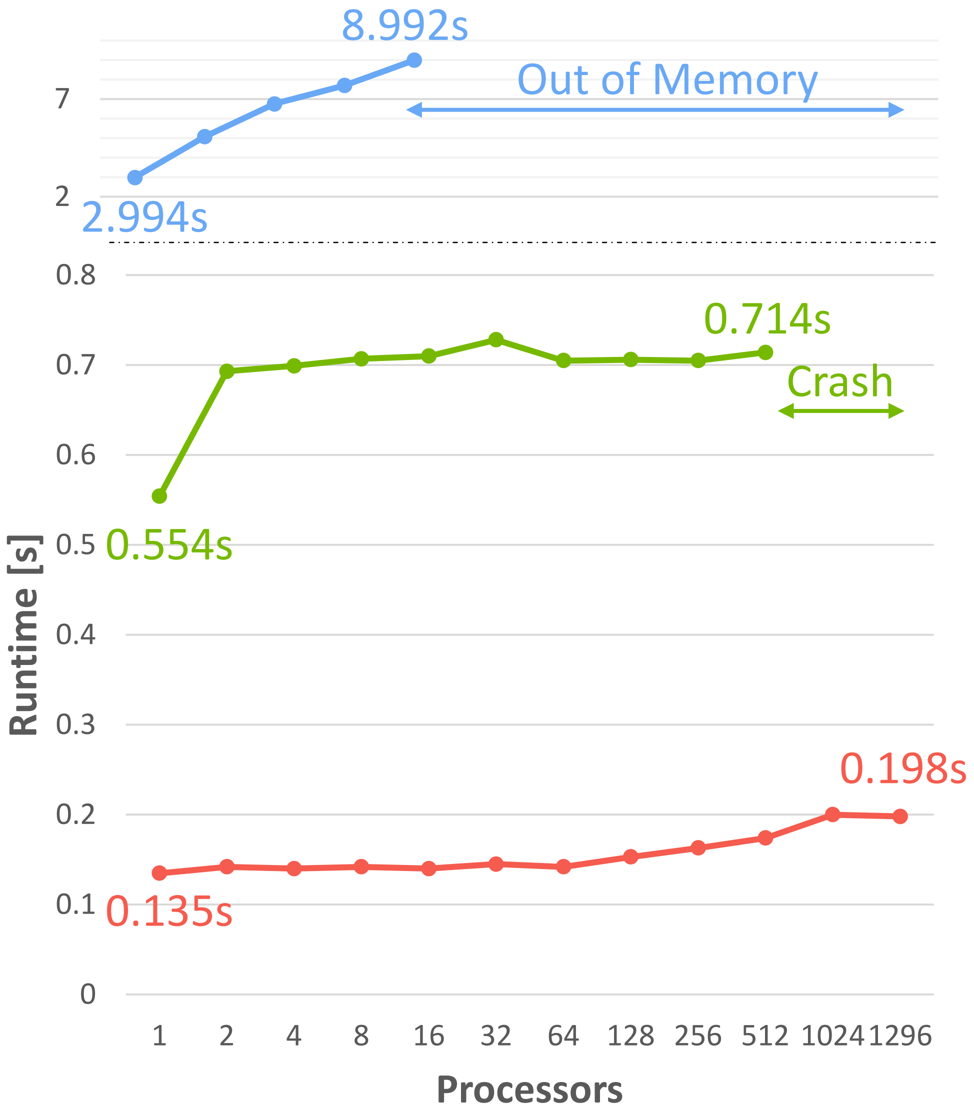
`C[:] = alpha * A @ B + beta * C`



Library Specialization

Library Node

Distributed GESUMMV

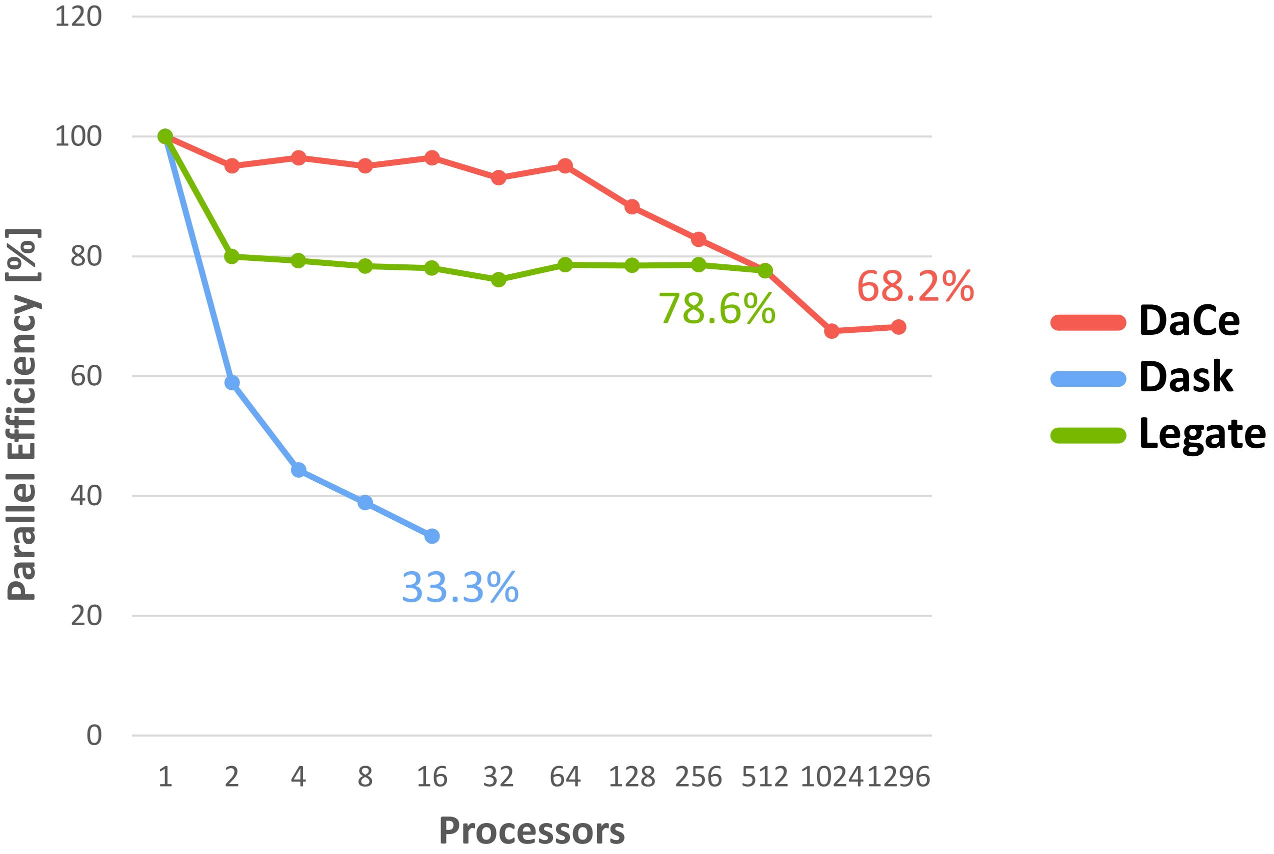


`alpha * A @ x + beta * B @ x`

Automatic Distribution

Weak scaling $N = 22400$ (half for Dask)

1-1296 Intel Xeon E5-2695 v4 CPUs



Direct Control via Local Views

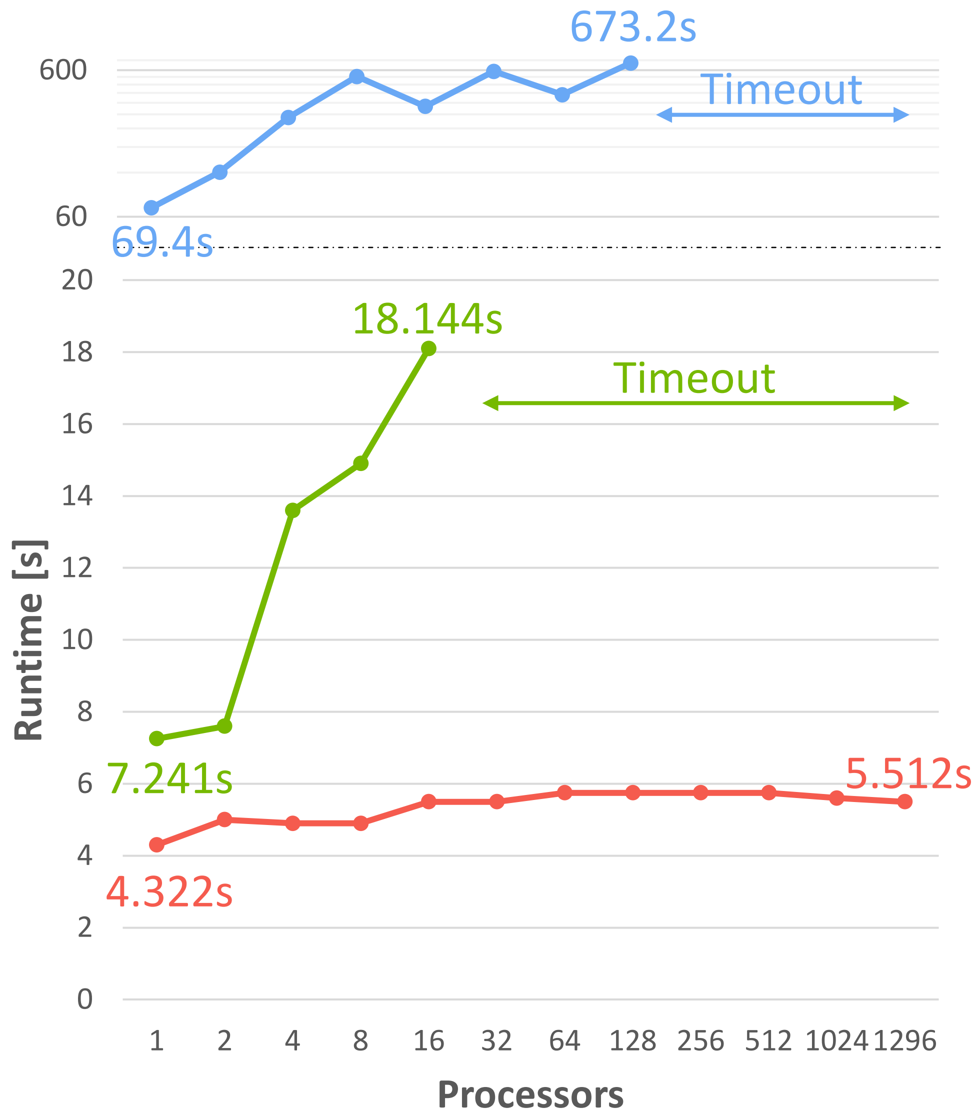
```
for t in range(1, TSTEPS):
  B[1:-1] = 0.33333 * (A[:-2] + A[1:-1] + A[2:])
```

Global to Local View

```
req = np.empty((4,), dtype=MPI_Request)
for t in range(1, TSTEPS):
  dc.comm.Isend(A[1], nw, 3, req[0])
  dc.comm.Isend(A[-2], ne, 2, req[1])
  dc.comm.Irecv(A[0], nw, 2, req[2])
  dc.comm.Irecv(A[-1], ne, 3, req[3])
  dc.comm.Waitall(req)
  lB[1+woff:-1-eoff] = 0.33333 * (
    lA[woff:-2-eoff] + lA[1+woff:-1-eoff] + lA[2+woff:-eoff])
```

User-driven Communication Pattern using the Data Communication Interface

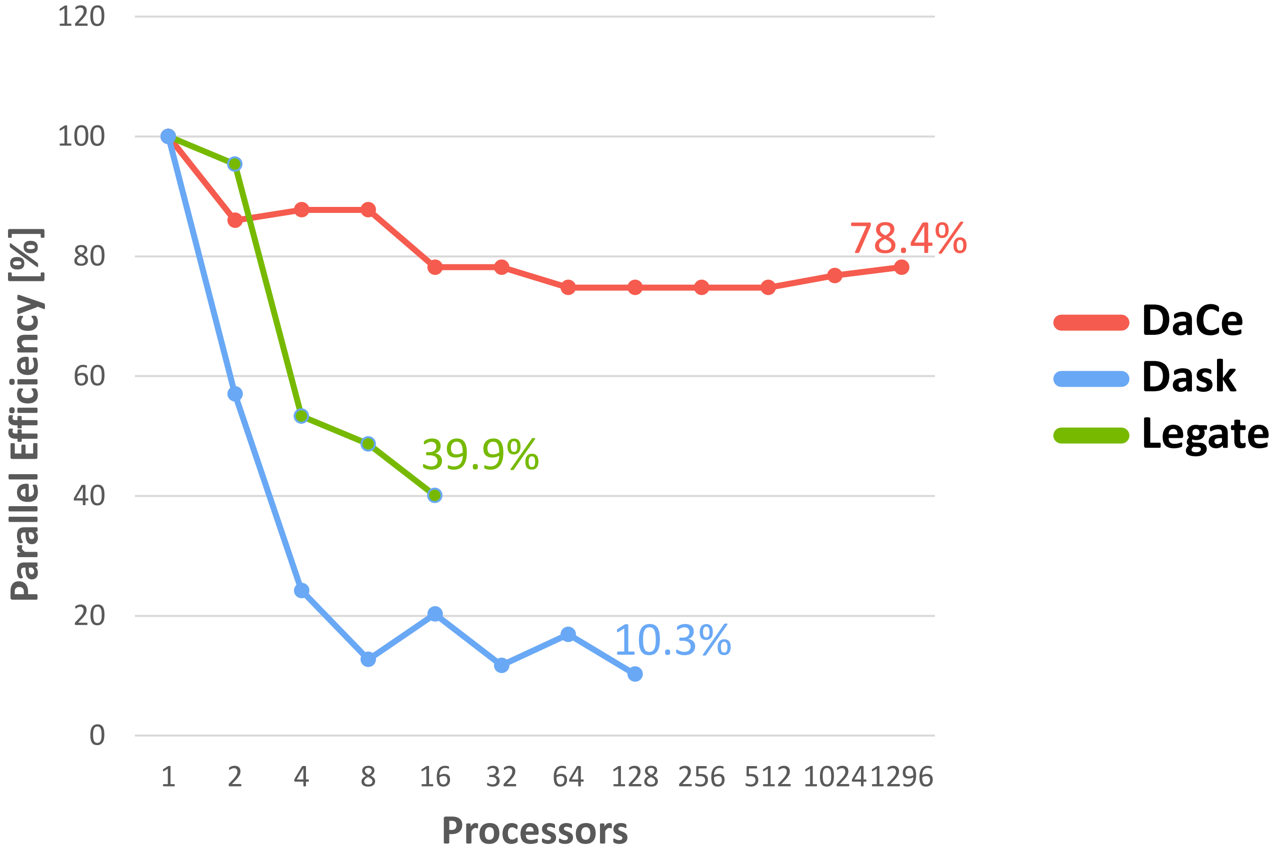
Distributed Jacobi-2D



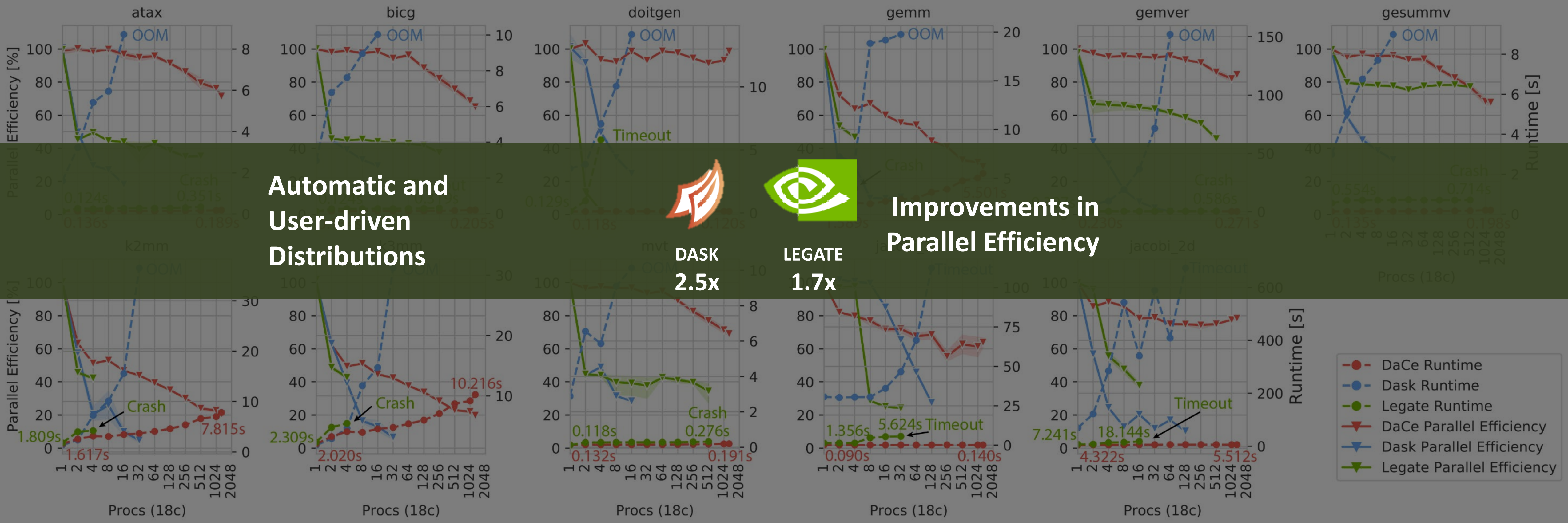
Data Communication Interface

Weak scaling $N = 1300$

1-1296 Intel Xeon E5-2695 v4 CPUs



Distributed Results



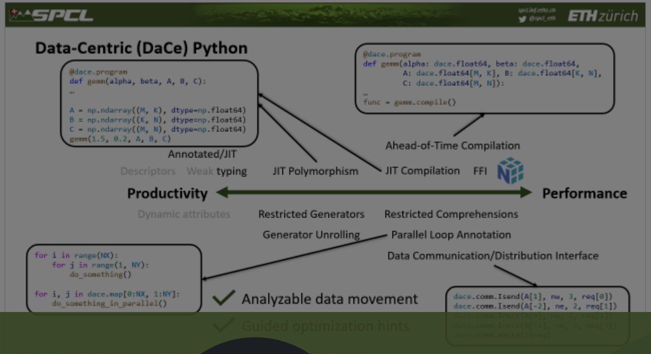
Automatic and User-driven Distributions

DASK
2.5x

LEGATE
1.7x

Improvements in Parallel Efficiency

Conclusions



Data-Centric (DaCe) Python

```
@dace.program
def gemm(alpha, beta, A, B, C):
…

A = np.ndarray((M, K), dtype=np.float64)
B = np.ndarray((K, N), dtype=np.float64)
C = np.ndarray((M, N), dtype=np.float64)
gemm(1.5, 0.2, A, B, C)
```

```
@dace.program
def gemm(alpha: dace.float64, beta: dace.float64,
         A: dace.float64[M, K], B: dace.float64[K, N],
         C: dace.float64[M, N]):
…
func = gemm.compile()
```

Annotated/JIT

Ahead-of-Time Compilation

Descriptors Weak typing JIT Polymorphism JIT Compilation FFI

Productivity ⟷ **Performance**

Dynamic attributes Restricted Generators Restricted Comprehensions

Generator Unrolling Parallel Loop Annotation

Data Communication/Distribution Interface

```
for i in range(NX):
    for j in range(1, NY):
        do_something()

for i, j in dace.map[0:NX, 1:NY]:
    do_something_in_parallel()
```

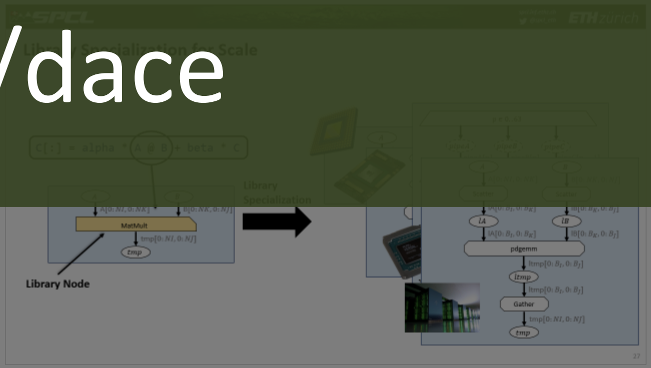
✓ Analyzable data movement

✓ Guided optimization hints

```
dace.comm.Isend(A[1], nw, 3, req[0])
dace.comm.Isend(A[-2], ne, 2, req[1])
```

Productivity

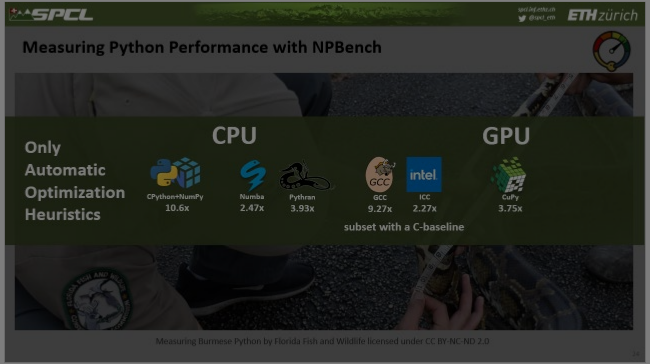
https://github.com/spcl/dace



Library Specialization for Scale

`C[:] = alpha * (A @ B) + beta * C`

Performance



Measuring Python Performance with NPBench

Only Automatic Optimization Heuristics

CPU					GPU
CPython+NumPy	Numba	Pythran	GCC	ICC	CuPy
10.6x	2.47x	3.93x	9.27x	2.27x	3.75x

subset with a C-baseline

Measuring Burmese Python by Florida Fish and Wildlife licensed under CC BY-NC-ND 2.0

Portability