Data-Centric Python - Productivity, portability and all with high performance!
Torsten Hoefler, keynote at the Russian Supercomputing Days (virtual)

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
Communication Dominates Arithmetic

- 64-bit DP: 20pJ
- 256-bit buses
- 256-bit access: 8 kB SRAM
- 50 pJ
- 26 pJ
- 256 pJ
- 16 nJ
- 500 pJ
- 1 nJ
- 28nm CMOS

Efficient off-chip link

Courtesy NVIDIA
How do we bring these ideas into productive performance computing?
Comparing fast python implementations

51 kernels from 9 domains
- Learning (6)
- LinAlg (12)
- Chemistry (4)
- Signals (3)
- Physics (9)
- Graphs (2)
- Weather (2)
- Solver (10)
- Other (3)

Metrics
- Performance
- Productivity

Frameworks
- NumPy baseline
- Pythran
- Numba
- CuPy
- DaCe

We need a fair comparison between frameworks

Python is the language of computational and data sciences

Weather (2)
Physics (9)
Chemistry (4)
Learning (6)
Signals (3)
Graphs (2)
Solver (10)
Other (3)

Meet NPBench

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**

<table>
<thead>
<tr>
<th>Domains</th>
<th>Total</th>
<th>Chemistry</th>
<th>Graphs</th>
<th>Learning</th>
<th>LinAlg</th>
<th>Other</th>
<th>Physics</th>
<th>Signals</th>
<th>Solver</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13.6</td>
<td>13.0</td>
<td>10.4</td>
<td>14.3</td>
<td>13.2</td>
<td>11.0</td>
<td>13.8</td>
<td>14.3</td>
<td>11.2</td>
<td>11.4</td>
</tr>
</tbody>
</table>

**How does this work?**

Does this work for Hardware Design and spatial architectures?

---

Ziogas et al.: NPBench: A Benchmarking Suite for High-Performance NumPy, ACM ICS’21
Data-Centric Python

```python
import dace

@dace.program
def somethingelse(x):
    return x * 5

@dace.program
def example(A: dace.float64[20, 20]):
    B = somethingelse(A)
    C = A + A
    B += C
    return np.dot(B, A)
```

imperative code

parametric dataflow representation
Data-Centric Python Vision – Performance Portability

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

10s of SLOC

Applied Scientist

translate DSL into parametric dataflow graphs

SDFG Builder API
Multi-Level Library Nodes

Performance Engineer

100s of reusable SLOC

Parametric Dataflow Graphs (SDFG)

Graph Transformations (API, Interactive)

Transformed Dataflow

Performance Results

Specialized Code Generation

CPU Code
GPU Code
FPGA Code
C++ code generation/runtime

1000s of auto-generated SLOC

Domain Scientist

\[ y = x^2 + \sin \frac{x}{\pi} \]

Applied Scientist

translate DSL into parametric dataflow graphs

SDFG Builder API
Multi-Level Library Nodes

Performance Engineer

100s of reusable SLOC

Transformed Dataflow

Performance Results

Specialized Code Generation

CPU Code
GPU Code
FPGA Code
C++ code generation/runtime

1000s of auto-generated SLOC

Domain Scientist

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

Applied Scientist

translate DSL into parametric dataflow graphs

SDFG Builder API
Multi-Level Library Nodes

Performance Engineer

100s of reusable SLOC

Transformed Dataflow

Performance Results

Specialized Code Generation

CPU Code
GPU Code
FPGA Code
C++ code generation/runtime

1000s of auto-generated SLOC

Domain Scientist

\[ y = x^2 + \sin \frac{x}{\pi} \]

Applied Scientist

translate DSL into parametric dataflow graphs

SDFG Builder API
Multi-Level Library Nodes

Performance Engineer

100s of reusable SLOC

Transformed Dataflow

Performance Results

Specialized Code Generation

CPU Code
GPU Code
FPGA Code
C++ code generation/runtime

1000s of auto-generated SLOC
Dataflow Programming in DaCe

\[ y = x^2 + \sin \frac{x}{\pi} \]
Dataflow Programming in DaCe

\[ y = x^2 + \sin(x) \pi \]
Parallel Dataflow Programming

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

Diagram showing a dataflow network with tasklets labeled A and B, and loops indicating iteration over an array A[0:N] and B[0:N].
Stateful Parallel Dataflow Programming

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

State s0

State s1
Meet the Nodes

**State**

- State machine element
- Fine-grained computational block
- N-dimensional data container
- Parametric graph abstraction for parallelism
- Streaming data container
- Dynamic mapping of computations on streams
- Defines behavior during conflicting writes
- Customizable computation with multiple implementations

**Tasklet**

- Nested SDFG

**Array**

**Map**

**Exit**

**Stream**

- Consume
- Exit

**Conflict Resolution**

- Reduce
- GEMM

---

Ben-Nun, de Fine Licht, Ziogas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19
Programmer/Performance Engineer view: Visual Studio Code Integration

Ben-Nun, de Fine Licht, Ziegas, TH: Stateful Dataflow Multigraphs: A Data-Centric Model for High-Performance Parallel Programs, SC19
Programmer/Performance Engineer view: Analyzing data flows
Programmer/Performance Engineer view: Debugging
DaCe is a versatile platform

- DaCeML (PyTorch/ONNX)
- C (C99)
- GridTools (Weather & Climate)
- Fortran (in planning)
- NumPy
- NVIDIA GPU
- AMD GPU
- x86 CPU
- ARM SVE
- Intel FPGA
- Xilinx FPGA
- RTL (soon)

Your favorite language/DSL (through SDFG builder)
Your favorite processor (through C++ codegen)
### Mapping NumPy to CPU

**Key principle:** keep dataflow of vectorized code around and use it for mapping

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>DaCe</th>
<th>GCC</th>
<th>ICC</th>
<th>Numba</th>
<th>Python</th>
<th>NumPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>110.6</td>
<td>14.5</td>
<td>14.3</td>
<td>12.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>syrk</td>
<td>10.0</td>
<td>19.3</td>
<td>233</td>
<td>13.0</td>
<td>15.5</td>
<td>6.58 s</td>
</tr>
<tr>
<td>gesumm</td>
<td>18.6(2)</td>
<td>11.8(3)</td>
<td>11.8(3)</td>
<td>17.5(3)</td>
<td>11.0(3)</td>
<td>0.77 s</td>
</tr>
<tr>
<td>mvt</td>
<td>11.0</td>
<td>52.6(3)</td>
<td>139</td>
<td>11.0(3)</td>
<td>1.0(3)</td>
<td>45.22 ms(3)</td>
</tr>
<tr>
<td>gemver</td>
<td>13.1(4)</td>
<td>12.4(3)</td>
<td>26.5(3)</td>
<td>12.1(3)</td>
<td>1.3(3)</td>
<td>0.79 s(3)</td>
</tr>
<tr>
<td>atax</td>
<td>12.2</td>
<td>12.0(3)</td>
<td>16.2(3)</td>
<td>11.0(3)</td>
<td>1.1(3)</td>
<td>73.37 ms(3)</td>
</tr>
<tr>
<td>floydwar</td>
<td>116(2)</td>
<td>13.9(3)</td>
<td>12.0</td>
<td>157.3</td>
<td>17.3</td>
<td>84.0 s</td>
</tr>
<tr>
<td>ftdt_2d</td>
<td>170(3)</td>
<td>13.7(3)</td>
<td>141(3)</td>
<td>14.1</td>
<td>11.3</td>
<td>7.4 s</td>
</tr>
<tr>
<td>durbin</td>
<td>12.2</td>
<td>15.0(3)</td>
<td>15.9(3)</td>
<td>4.1(3)</td>
<td>13.2</td>
<td>0.65 s</td>
</tr>
<tr>
<td>doitgen</td>
<td>139.7(4)</td>
<td>11.8(3)</td>
<td>12.3(3)</td>
<td>11.1</td>
<td>10.3(3)</td>
<td>0.47 s(3)</td>
</tr>
<tr>
<td>deriche</td>
<td>117.3</td>
<td>12.7(3)</td>
<td>155.1(3)</td>
<td>11.5</td>
<td>11.6</td>
<td>2.83 s</td>
</tr>
<tr>
<td>covarian</td>
<td>116(4)</td>
<td>121.4</td>
<td>15.0(3)</td>
<td>11.3(3)</td>
<td>120.4(3)</td>
<td>80.11 ms</td>
</tr>
<tr>
<td>correlat</td>
<td>11.6</td>
<td>27.8(3)</td>
<td>16.6(3)</td>
<td>11.1(3)</td>
<td>1.0(3)</td>
<td>61.55 ms(3)</td>
</tr>
<tr>
<td>cholesky</td>
<td>12.1</td>
<td>13.7</td>
<td>13.4</td>
<td>10.3</td>
<td>15.2</td>
<td>7.05 s</td>
</tr>
<tr>
<td>bicg</td>
<td>11.1</td>
<td>17.2(3)</td>
<td>17.5(3)</td>
<td>11.1(3)</td>
<td>11.1(3)</td>
<td>75.92 ms(3)</td>
</tr>
<tr>
<td>trisolv</td>
<td>2.3(4)</td>
<td>11.6(3)</td>
<td>11.8(3)</td>
<td>11.9(3)</td>
<td>11.3</td>
<td>0.13 s(4)</td>
</tr>
<tr>
<td>adi</td>
<td>16.5</td>
<td>19.1</td>
<td>13.7(3)</td>
<td>18.2</td>
<td>19.4</td>
<td>0.9 s</td>
</tr>
<tr>
<td>3mm</td>
<td>11.7(3)</td>
<td>17.1(3)</td>
<td>11.1(3)</td>
<td>11.6(3)</td>
<td>12.0(3)</td>
<td>0.46 s(3)</td>
</tr>
<tr>
<td>2mm</td>
<td>12.4(2)</td>
<td>17.1(3)</td>
<td>11.5(3)</td>
<td>11.3(3)</td>
<td>0.42 s(2)</td>
<td></td>
</tr>
<tr>
<td>gemm</td>
<td>12.3(3)</td>
<td>13.3(2)</td>
<td>14.1(3)</td>
<td>11.4(3)</td>
<td>11.4</td>
<td>78.72 ms</td>
</tr>
<tr>
<td>trmm</td>
<td>136.2</td>
<td>14.1</td>
<td>150.9(3)</td>
<td>15.3(3)</td>
<td>14.2</td>
<td>4.12 s(1)</td>
</tr>
</tbody>
</table>

NumPy can even beat the C versions of PolyBench codes!

DaCe outperforms CuPy in most cases

Zogas et al.: Productivity, Portability, Performance: Data-Centric Python, SC21
Mapping Weather/Climate Stencils (in a Python DSL) to FPGAs

Key principle: spatial layout and pipelining using streams and delay buffers

Combined Spatial and Temporal Blocking for High-Performance Stencil Computation on FPGAs Using OpenCL

Hamid Ria Zohouri, Artur Podeba, Satoshi Matsuoka
Tokyo Institute of Technology, Tokyo, Japan
{zohouri.bas@pm.tuat.ac.jp, matsuoka@nist.gov}

ABSTRACT
Recent developments in High-Level Synthesis tools have attracted widespread attention to accelerate data algorithms, which has prevented largescale adoption of FPGAs in the High Performance Computing (HPC) community. However, with the recent improvements in High-Level Synthesis (HLS), especially the

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

23-44% faster!

de Fine Licht et al: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems, CGO’21
Mapping Weather Stencils (in a Python DSL) to FPGAs

Key principle: spatial layout and pipelining using streams and delay buffers

318 MHz at 48% DSP utilization

<table>
<thead>
<tr>
<th>Stratix 10</th>
<th>Runtime</th>
<th>Performance</th>
<th>Peak BW.</th>
<th>%Roof.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,178 μs</td>
<td>145 GOP/s</td>
<td>77 GB/s</td>
<td>52%</td>
</tr>
</tbody>
</table>

The Stratix 10 is held back by insufficient bandwidth.

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

de Fine Licht et al: StencilFlow: Mapping Large Stencil Programs to Distributed Spatial Computing Systems, CGO’21
Mapping Deep Learning Codes to GPUs

Key principle: minimize data movement and optimize data layout

Bonus: Mapping C/C++ (not just Python!) to CPUs

Key principle: find parallel regions using parametric dataflow

LLNL’s LULESH C++ benchmark

dual-socket 2×18 core Intel Xeon Gold 6154

Calotoiu et al: High Performance C — Let Dataflow Do the Heavy Lifting, to appear
Mapping NumPy to large-scale CPU clusters

Key principle: communication-minimizing data mapping to nodes

Using MPI library nodes
(MPI-like interface)
Gordon Bell Prize 2019 on ORNL’s Summit (Top-1 machine run with >21k GPUs)

- pip install dace

- Gordon Bell Prize 2019
  - 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
Overview and wrap-up

This project has received funding from the European Research Council (ERC) under grant agreement "DAPP (PI: T. Hoefler)" and DEEP-SEA, No. 955606.

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

pip install dace

Open PhD and Postdoc positions: https://spcl.inf.ethz.ch/Jobs/
SPCL is hiring PhD students and highly-qualified postdocs to reach new heights!

https://spcl.inf.ethz.ch/Jobs/