A Data-Centric Approach to Performance Portability
Torsten Hoefler, Keynote at CIUK 2020, United Kingdom (virtual)

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

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."

Control in Load-store vs. Dataflow

Load-store ("von Neumann")

\[ x = a + b \]

Energy per instruction: 70pJ

Static Dataflow ("non von Neumann")

\[ y = (a + b) \times (c + d) \]

Energy per operation: 1-3pJ

Control Locality

Source: Mark Horowitz, ISSC'14
Single Instruction Multiple Data/Threads (SIMD - Vector CPU, SIMT - GPU)

High Performance Computing really became a data management challenge

[1]: Marc Horowitz, Computing’s Energy Problem (and what we can do about it), ISSC 2014, plenary
Data movement will dominate everything!

- "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)

- 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!

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.”

[1]: Bauer et al.: “Legion: expressing locality and independence with logical regions”, SC12, 2012
Scientific Software Engineering in the 21\textsuperscript{st} century – Performance Portability

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

\begin{itemize}
\item Main Scientist
\item Applied Scientist
\item Performance Engineer
\end{itemize}

Applied Scientist
- translate DSL into parametric dataflow graphs

Performance Engineer
- 100s of reusable SLOC
- Transformed Dataflow
- Performance Results

Specialized Code Generation
- CPU Code
- GPU Code
- FPGA Code
- C++ code generation/runtime

Transformed Dataflow Graphs (SDFG)

- \( L \)
- \( R \)

10s of SLOC

1000s of auto-generated SLOC

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

\[ y = x^2 + \sin x \pi \]

Tasklet

Memlets
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)

Tasklet
\[ A[i] \]
\[ B[i] \]

Tasklet
\[ A[0:N] \]
\[ B[0:N] \]

Iteration space (map)
States in Parametric Dataflow Graphs (SDFGs)

\[ A \]
\[ A[0:N] \]
\[ [i=0:N] \]
\[ A[i] \]
Tasklet
\[ B[i] \]
\[ [i=0:N] \]
\[ B[0:N] \]
\[ B \]

\[ C \]
\[ C[0:N] \]
\[ [i=0:N] \]
\[ C[i] \]
Tasklet
\[ A[i] \]
\[ [i=0:N] \]
\[ A[0:N] \]
\[ A \]
States in Parametric Dataflow Graphs (SDFGs)

State s0

A
\[ A[0:N] \]
\[ i=0:N \]
A[i]

Tasklet

B[i]
\[ i=0:N \]
B[0:N]

State s1

C
\[ C[0:N] \]
\[ i=0:N \]
C[i]

Tasklet

A[i]
\[ i=0:N \]
A[0:N]
First Example: 2D Stencil

Initialization (read input)

State $s_0$

$[y=0:H, x=0:W]$  
Initialize  
$B[y,x]$  
$[y=0:H, x=0:W]$  
$B[0:H,0:W]$  
$B$

$t = 0$

$t < T; t++$

$t \geq T$

Exit (converged)

Iteration (solver)

State $s_1$

$A$

$[y=0:H, x=0:W]$  
Jacobi  
$A[y-1,x]$  
$A[y+1,x]$  
$A[y,x]$  
$A[y,x-1]$  
$A[y,x+1]$

$B$

$[y=0:H, x=0:W]$  
Jacobi  
$B[y-1,x]$  
$B[y+1,x]$  
$B[y,x]$  
$B[y,x-1]$  
$B[y,x+1]$

$A$

$[y=0:H, x=0:W]$  
$A[y,x]$  
$A[0:H,0:W]$  
$B[0:H,0:W]$  
$B[0:H,0:W]$  
$B[0:H,0:W]$  
$A[0:H,0:W]$
Second Example: MatMul

```python
@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)
```

DaCe-Python Explicit (minimal) side-effect code

State s0

\[
\begin{align*}
A[0:M,0:K] & \rightarrow [i=0:M, j=0:N, k=0:K] \\
B[0:K,0:N] & \rightarrow [i=0:M, j=0:N, k=0:K] \\
A[i,k] & \rightarrow B[k,j] \\
multiplication & \rightarrow tmp[i,j,k] \\
tmp[0:M,0:N,0:K] & \rightarrow [i=0:M, j=0:N, k=0:K] \\
tmp & \rightarrow [0:M,0:N,0:K] \\
\text{Reduce} [\text{axis: 2, sum}] & \rightarrow C[0:M,0:N]
\end{align*}
\]

$N^3$ 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)
### Parametric Dataflow Graphs - Concepts

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

- ![Data](image1.png)
- ![Transient Data](image2.png)
- ![Stream](image3.png)

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

- ![Tasklet](image4.png)
- ![Invoke](image5.png)

#### Data Movement / Dependencies
- Data flowing between containers and tasklets/ports
- Implemented as access, copies, streaming, ...

- ![A(1) [0:M,k]](image6.png)
- ![C[i,j] (CR: Sum)](image7.png)

#### Parallelism and States
- Map scopes provide parallelism
- States constrain parallelism outside of datatflow

- ![s0](image8.png) \[\text{iter < N}\] \[\rightarrow\] ![s1](image9.png)

---

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)

Source Code

Transformations

SDFG (malleable)

Generated Code

Performance

SDFG

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

Naive

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, Ziqgas, 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)
Performance for matrix multiplication on x86

- Naïve
- LocalStorage (50 SLOC)
- RegisterTiling (47 SLOC)
- BlockTiling (39 SLOC)
- LoopReorder (27 SLOC)
- MapReduceFusion (27 SLOC)
Performance for matrix multiplication on x86

PromoteTransient (51 SLOC)
LocalStorage (50 SLOC)
RegisterTiling (47 SLOC)
BlockTiling (39 SLOC)
LoopReorder (27 SLOC)
MapReduceFusion (27 SLOC)
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
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
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)

![Performance Graph](image)

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

GPU

19.5x of Spatial

FPGA

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 geometric speedup)

**FPGA**

The **first** full set of placed-and-routed Polybench
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?
Parametric Dataflow Graphs (SDFG)

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

Graph Transformations (API, Interactive)

Transformed Dataflow

Performance Results

Specialized Code Generation

Runtime

CPU Code
GPU Code
FPGA Code

C++ code generation/runtime

Domain Scientist

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

NumPy
TensorFlow
PyTorch
MATLAB

DSLs

Applied Scientist

translate DSL into parametric dataflow graphs

SDFG Builder API
Multi-Level Library Nodes

Performance Engineer

Transformed Dataflow

Performance Results

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
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, ...

\[
\text{NEGF} \quad \text{SSE} \quad \Sigma [G(E + \hbar \omega, k_z - q_z) D(\omega, q_z)](E, k_z)
\]

\[
\begin{align*}
\text{Electrons} \quad G(E, k_z) \\
(E \cdot S - H - \Sigma^R) \cdot G^R &= I \\
G^< &= G^R \cdot \Sigma^< \cdot G^A
\end{align*}
\]

\[
\begin{align*}
\text{Phonons} \quad D(\omega, q_z) \\
(\omega^2 - \Phi - \Pi^R) \cdot D^R &= I \\
D^< &= D^R \cdot \Pi^< \cdot D^A
\end{align*}
\]

\[
\text{GF} \quad \text{SSE} \quad \Pi [G(E, k_z) G(E + \hbar \omega, k_z + q_z)](\omega, q_z)
\]

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

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!

<table>
<thead>
<tr>
<th>Variant</th>
<th>GF</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tilop</td>
<td>Time [s]</td>
</tr>
<tr>
<td>OMEN</td>
<td>174.0</td>
<td>144.14</td>
</tr>
<tr>
<td>Python</td>
<td>174.0</td>
<td>1,342.77</td>
</tr>
<tr>
<td>DaCe</td>
<td>174.0</td>
<td>111.25</td>
</tr>
</tbody>
</table>

Piz Daint single node (P100)

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

<table>
<thead>
<tr>
<th></th>
<th>cuBLAS</th>
<th>DaCe (SBSMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPU</td>
<td>Glop</td>
</tr>
<tr>
<td>P100</td>
<td>27.42</td>
<td>6.73 ms</td>
</tr>
<tr>
<td>V100</td>
<td>27.42</td>
<td>4.62 ms</td>
</tr>
</tbody>
</table>

Basic operation in SSE (many very small MMMs)

5k atoms
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 P flop/s with I/O (28% peak)

### Communication time reduced by 417x on Piz Daint!

Volume on full-scale Summit from 12 PB/iter \(\rightarrow\) 87 TB/iter

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

**Table:**

<table>
<thead>
<tr>
<th>Variant</th>
<th>(N_a)</th>
<th>Time [s]</th>
<th>Time/Atom [s]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMEN</td>
<td>1,064</td>
<td>4695.70</td>
<td>4.413</td>
<td>1.0x</td>
</tr>
<tr>
<td>DaCe</td>
<td>10,240</td>
<td>489.83</td>
<td>0.048</td>
<td>92.3x</td>
</tr>
</tbody>
</table>

\(P = 6.840, N_b = 34, N_{orb} = 12, N_E = 1,220, N_\omega = 70.\)

Ziogas et al. “A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations”, SC19 (Gordon Bell Prize 2019).
Gordon Bell Prize 2019 – High Performance on Top-1 Machine

- pip install dace

- **Gordon Bell Award 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)".
Optimizing Transformer Deep Neural Networks

Back to our first example – Laplace 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
```

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