Progress in automatic GPU compilation and why you want to run MPI on your GPU

Torsten Hoefler (most work by Tobias Grosser, Tobias Gysi, Jeremia Baer)
Presented at Saarland University, Saarbrücken, Germany
ETH Zurich, CS, Systems Group, SPCL

- ETH Zurich
  - Shanghai ranking ‘15 (Computer Science): #1 outside North America (#17 global)
  - 16 departments, 1.62 Bn $ federal budget
- Computer Science department
  - 28 tenure-track faculty, ~1k students
- Systems group (6 professors)
  - Onur Mutlu, Timothy Roscoe, Gustavo Alonso, Ankit Singla, Ce Zheng, TH
  - Focused on systems research of all kinds (data management, OS, …)
- SPCL focusses on performance/data/HPC
  - 3 postdocs
  - 10 PhD students (+2 external)
  - 20+ BSc and MSc students
  - [http://spcl.inf.ethz.ch](http://spcl.inf.ethz.ch) - ERC Starting Grant DAPP
  - Twitter: @spcl_eth
Evading various “ends” – the hardware view
Sequential Software

Fortran

C/C++

Parallel Hardware

Multi-Core CPU

Accelerator
Design Goals

Automatic accelerator mapping
- How close can we get?

“Regression Free”
High Performance
Theory
Tool: Polyhedral Modeling

Program Code

```c
for (i = 0; i <= N; i++)
    for (j = 0; j <= i; j++)
        S(i,j);
```

Iteration Space

\[ D = \{ (i,j) \mid 0 \leq i \leq N \land 0 \leq j \leq i \} \]

N = 4

(i, j) = (4,4)
Mapping Computation to Device

**Iteration Space**

\[ BID = \{ (i,j) \rightarrow \left( \left\lfloor \frac{i}{4} \right\rfloor \mod 2, \left\lfloor \frac{j}{3} \right\rfloor \mod 2 \} \]  

\[ TID = \{ (i,j) \rightarrow (i \mod 4, j \mod 3) \} \]  

**Device Blocks & Threads**

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3 \\
\end{array}
\]
Memory Hierarchy of a Heterogeneous System
Host-device data transfers
Host-device date transfers
Mapping onto fast memory
Mapping onto fast memory

Practice
LLVM Nightly Test Suite

for(int i=0; i<5; i++) {
    a[i] = 0;
}

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS'16
LLVM Nightly Test Suite

# Compute Regions / Kernels

SCoPs 0-dim 1-dim 2-dim 3-dim

No Heuristics Heuristics

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
Profitability Heuristic

Modeling

Execution

GPU

Insufficient Compute

static
dynamic

All Loop Nests

Trivial
(e.g., initialization)

Unsuitable
(e.g., sequential)

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
From kernels to program – data transfers

```c
void heat(int n, float A[n], float hot, float cold) {
    float B[n] = {0};
    initialize(n, A, cold);
    setCenter(n, A, hot, n/4);
    for (int t = 0; t < T; t++) {
        average(n, A, B);
        average(n, B, A);
        printf("Iteration %d done", t);
    }
}
```
void heat(int n, float A[n], ...) {
    initialize(n, A, cold);
    setCenter(n, A, hot, n/4);
    for (int t = 0; t < T; t++) {
        average(n, A, B);
        average(n, B, A);
        printf("Iteration \%d done", t);
    }
}
void heat(int n, float A[n], ...) {
    initialize(n, A, cold);
    setCenter(n, A, hot, n/4);
    for (int t = 0; t < T; t++) {
        average(n, A, B);
        average(n, B, A);
        printf("Iteration %d done", t);
    }
}
Evaluation

Workstation: 10 core SandyBridge
Mobile: 4 core Haswell
NVIDIA Titan Black (Kepler)
NVIDIA GT730M (Kepler)
Some results: Polybench 3.2

Xeon E5-2690 (10 cores, 0.5Tflop) vs. Titan Black Kepler GPU (2.9k cores, 1.7Tflop)

Speedup over icc –O3

- **Arithmetic mean:** ~30x
- **Geometric mean:** ~6x

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
Compiles all of SPEC CPU 2006 – Example: LBM

- Essentially my 4-core x86 laptop with the (free) GPU that’s in there.

- Xeon E5-2690 (10 cores, 0.5 Tflop) vs. Titan Black Kepler GPU (2.9k cores, 1.7 Tflop)

- ~20%

- ~4x
Cactus ADM (SPEC 2006)

**Mobile**

- Polly ACC cached
- Polly ACC
- icc parallel
- icc
- LLVM

**Workstation**

- Polly ACC cached
- Polly ACC
- icc parallel
- icc
- LLVM

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
Polly-ACC

Mapping Computation to Device

Automatic

"Regression Free"

High Performance

Profitability Heuristic

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16

http://spcl.inf.ethz.ch/Polly-ACC
Brave new/old compiler world!? 

- Unfortunately not ...
  - Limited to affine code regions
  - Maybe generalizes to oblivious (data-independent control) programs
  - No distributed anything!!

- Good news:
  - Much of traditional HPC fits that model
  - Infrastructure is coming along

- Bad news:
  - Modern data-driven HPC and Big Data fits less well
  - Need a programming model for distributed heterogeneous machines!
Distributed GPU Computing
GPU cluster programming using MPI and CUDA

node 1

device memory

PCI-Express

host memory

PCI-Express

interconnect

node 2

device memory

PCI-Express

host memory

PCI-Express

code

// run compute kernel
__global__
void mykernel( ... ) {
}

// launch compute kernel
mykernel<<<64,128>>>( ... );

// on-node data movement
cudaMemcpy(
    psize, &size,
    sizeof(int),
    cudaMemcpyDeviceToHost);

// inter-node data movement
mpi_send(
    pdata, size,
    MPI_FLOAT, ... );

mpi_recv(
    pdata, size,
    MPI_FLOAT, ... );
Disadvantages of the MPI-CUDA approach

**Complexity**
- two programming models
- duplicated functionality

**Performance**
- encourages sequential execution
- low utilization of the costly hardware

```
mykernel<<< >>>( ... );  
cudaMemcpy( ... );  
mpi_send( ... );  
mpi_recv( ... );  
mykernel<<< >>>( ... );
```

```
...  
mykernel( ... ) {  
    ...  
}
```

```
...  
mykernel( ... ) {  
    ...  
}
```

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Achieve high resource utilization using oversubscription & hardware threads

<table>
<thead>
<tr>
<th>code</th>
<th>thread 1</th>
<th>thread 2</th>
<th>thread 3</th>
<th>instruction pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld %r0,%r1</td>
<td>ready</td>
<td>ready</td>
<td>ready</td>
<td>ld</td>
</tr>
<tr>
<td>mul %r0,%r0,3</td>
<td>stall</td>
<td>1d %r0,%r1</td>
<td>ready</td>
<td>1d 1d</td>
</tr>
<tr>
<td>st %r0,%r1</td>
<td>ready</td>
<td>stall</td>
<td>1d %r0,%r1</td>
<td>1d 1d</td>
</tr>
<tr>
<td></td>
<td>mul %r0,%r0,3</td>
<td>ready</td>
<td>stall</td>
<td>mul 1d</td>
</tr>
<tr>
<td></td>
<td>stall</td>
<td>mul %r0,%r0,3</td>
<td>ready</td>
<td>mul mul</td>
</tr>
<tr>
<td></td>
<td>ready</td>
<td>stall</td>
<td>mul %r0,%r0,3</td>
<td>mul mul</td>
</tr>
<tr>
<td></td>
<td>st %r0,%r1</td>
<td>ready</td>
<td>stall</td>
<td>st mul</td>
</tr>
<tr>
<td></td>
<td>stall</td>
<td>st %r0,%r1</td>
<td>ready</td>
<td>st st</td>
</tr>
<tr>
<td></td>
<td>ready</td>
<td>stall</td>
<td>st %r0,%r1</td>
<td>st st</td>
</tr>
</tbody>
</table>

GPU cores use "parallel slack" to hide instruction pipeline latencies

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Use oversubscription & hardware threads to hide remote memory latencies

<table>
<thead>
<tr>
<th>code</th>
<th>thread 1</th>
<th>thread 2</th>
<th>thread 3</th>
<th>instruction pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>get …</td>
<td>get</td>
<td>ready</td>
<td>ready</td>
<td>get</td>
</tr>
<tr>
<td>mul %r0,%r0,3</td>
<td>stall</td>
<td>get</td>
<td>ready</td>
<td>get</td>
</tr>
<tr>
<td>put …</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>!</td>
</tr>
<tr>
<td></td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>!</td>
</tr>
<tr>
<td></td>
<td>ready</td>
<td>stall</td>
<td>stall</td>
<td>!</td>
</tr>
<tr>
<td>mul %r0,%r0,3</td>
<td>ready</td>
<td>stall</td>
<td>stall</td>
<td>mul</td>
</tr>
<tr>
<td></td>
<td>mul %r0,%r0,3</td>
<td>mul %r0,%r0,3</td>
<td>mul %r0,%r0,3</td>
<td>! mul</td>
</tr>
<tr>
<td>put …</td>
<td>ready</td>
<td>stall</td>
<td>mul %r0,%r0,3</td>
<td>put</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mul %r0,%r0,3</td>
<td>mul</td>
</tr>
</tbody>
</table>

introduce put & get operations to access distributed memory

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
How much “parallel slack” is necessary to fully utilize the interconnect?

Little’s law

\[
\text{concurrency} = \text{latency} \times \text{throughput}
\]

---

**device memory**

<table>
<thead>
<tr>
<th>latency</th>
<th>1(\mu)s</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth</td>
<td>200GB/s</td>
</tr>
<tr>
<td>concurrency</td>
<td>200kB</td>
</tr>
<tr>
<td>#threads</td>
<td>~12000</td>
</tr>
</tbody>
</table>
dCUDA (distributed CUDA) extends CUDA with MPI-3 RMA and notifications

```c
for (int i = 0; i < steps; ++i) {
    for (int idx = from; idx < to; idx += jstride)
        out[idx] = -4.0 * in[idx] +
                   in[idx + 1] + in[idx - 1] +
                   in[idx + jstride] + in[idx - jstride];

    if (lsend)
        dcuda_put_notify(ctx, wout, rank - 1,
                         len + jstride, jstride, &out[jstride], tag);
    if (rsend)
        dcuda_put_notify(ctx, wout, rank + 1,
                         0, jstride, &out[len], tag);

    dcuda_wait_notifications(ctx, wout,
                             DCUDA_ANY_SOURCE, tag, lsend + rsend);

    swap(in, out);
    swap(win, wout);
}
```

- iterative stencil kernel
- thread specific idx
- map ranks to blocks
- device-side put/get operations
- notifications for synchronization
- shared and distributed memory

---

Advantages of the dCUDA approach

**Performance**
- avoid device synchronization
- latency hiding at cluster scale

**Complexity**
- unified programming model
- one communication mechanism

---

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Implementation of the dCUDA runtime system

event handler

block manager

block manager

block manager

MPI

GPU direct

device-library

put( ... );
get( ... );
wait( ... );

device-library

put( ... );
get( ... );
wait( ... );

device-library

put( ... );
get( ... );
wait( ... );

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Evaluation

Cluster: 8 Haswell nodes, 1x Tesla K80 per node
Overlap of a copy kernel with halo exchange communication

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Weak scaling of MPI-CUDA and dCUDA for a stencil program

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Weak scaling of MPI-CUDA and dCUDA for a particle simulation

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)

![Graph showing execution time vs. number of nodes for MPI-CUDA and dCUDA with halo exchange.](image-url)
Weak scaling of MPI-CUDA and dCUDA for sparse-matrix vector multiplication

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
Conclusions

- unified programming model for GPU clusters
  - device-side remote memory access operations with notifications
  - transparent support of shared and distributed memory
- extend the latency hiding technique of CUDA to the full cluster
  - inter-node communication without device synchronization
  - use oversubscription & hardware threads to hide remote memory latencies
- automatic overlap of computation and communication
  - synthetic benchmarks demonstrate perfect overlap
  - example applications demonstrate the applicability to real codes
- [https://spcl.inf.ethz.ch/Research/Parallel_Programming/dCUDA/](https://spcl.inf.ethz.ch/Research/Parallel_Programming/dCUDA/)
SPCL is hiring PhD students and highly-qualified postdocs to reach new heights!

- Working in the area of High Performance Computing
  - Networking (InfiniBand, Aries etc.)
  - Middleware/Programming Models (MPI, CUDA etc.)
  - Compilation (LLVM, DSLs etc.)

- ERC Starting Grant DAPP
  - Develop programming model to replace MPI

- Many international contacts
  - US/Japan/China Academics + Universities
  - US National Labs
  - Industry

https://spcl.inf.ethz.ch/Jobs/
Backup slides
Hardware utilization of dCUDA compared to MPI-CUDA

traditional MPI-CUDA

device 1

rank 1 1
rank 2 1
rank 3 2
rank 4 2

device 2

rank 1 1
rank 2 1
rank 3 2
rank 4 2

dCUDA

device 1

rank 1 1
rank 2 1
rank 3 2
rank 4 2

device 2

rank 1 1
rank 2 1
rank 3 2
rank 4 2
Implementation of the dCUDA runtime system

event handler

block manager

MPI

host-side

device-side

more blocks

logging queue

command queue

ack queue

notification queue

device library

context
GPU clusters gained a lot of popularity in various application domains

- machine learning
- weather & climate
- molecular dynamics
Hardware supported overlap of computation & communication

traditional MPI-CUDA

device
compute core
active block

dCUDA
Traditional GPU cluster programming using MPI and CUDA

CUDA
- over-subscribe hardware
- use spare parallel slack for latency hiding

MPI
- host controlled
- full device synchronization
How can we apply latency hiding on the full GPU cluster?

**dCUDA (distributed CUDA)**
- unified programming model for GPU clusters
- avoid unnecessary device synchronization to enable system wide latency hiding
Questions

- relation to NV link
  - NV link is a transport layer in the first place
  - it should enable a faster implementation

- synchronized programming in NV link
  - single kernel on machine with multiple GPUs connected by NV link
  - single node at the moment
  - will probably not scale to 1000s of nodes as there is no explicit communication