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 University of Texas Austin, USA, Feb. 2017
ETH, CS, Systems Group, SPCL

- ETH Zurich – top university in central Europe
  - Shanghai ranking ‘15 (Computer Science): #17, best outside North America
  - Times higher education ‘16-17 (Computer Science): #1
  - 16 departments, 1.62 Bn $ federal budget

- Computer Science department
  - 28 tenure-track faculty, 1k students

- Systems group (7 professors)
  - Onur Mutlu, Timothy Roscoe, Gustavo Alonso, Ankit Singla, Ce Zheng, (Donald Kossmann), TH
  - Focused on systems research of all kinds (data management, OS, ...)

- SPCL focuses on performance/data/HPC
  - 3 postdocs
  - 8 PhD students (+2 external)
  - 15+ BSc and MSc students
  - [http://spcl.inf.ethz.ch](http://spcl.inf.ethz.ch)
  - 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

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

- \( i \leq N = 4 \)
- \( 0 \leq j \)
- \( j \leq i \)
- \( 0 \leq i \)
- \( (i, j) = (4,4) \)

N = 4
Mapping Computation to Device

Iteration Space

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

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

Device Blocks & Threads

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
Memory Hierarchy of a Heterogeneous System
Host-device data transfers
Host-device date transfers
Mapping onto fast memory

Main Memory
Device Memory
Shared Memory
Registers
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

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

Modeling

Execution

GPU

All Loop Nests

Trivial

Insensitive Compute

Static

Dynamic

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
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);
    }
}
Data Transfer – Per Kernel

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);
   }
}
Data Transfer – Inter Kernel Caching

```c
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)

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

Runtime (m:s)

- Xeon E5-2690 (10 cores, 0.5Tflop) vs. Titan Black Kepler GPU (2.9k cores, 1.7Tflop)
  - ~20%
  - ~4x

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS’16
Cactus ADM (SPEC 2006)

**Mobile**

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

Performance [iterations/second]

**Workstation**

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

Performance [iterations/second]

T. Grosser, TH: Polly-ACC: Transparent compilation to heterogeneous hardware, ACM ICS'16
Cactus ADM (SPEC 2006) - Data Transfer

**Mobile**

- H to D
- H to D (cached)
- D to H
- D to H (cached)

**Workstation**

- H to D
- H to D (cached)
- D to H
- D to H (cached)

---

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

http://spcl.inf.ethz.ch/Polly-ACC

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

// 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

```c
mykernel<<< >>>( ... );
cudaMemcpy( ... );

mykernel( ... ) {
    ...
}

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

```c
device sync
mpi_send( ... );
mpi_recv( ... );

cluster sync
mykernel<<< >>>( ... );
```

```c
host
```
Achieve high resource utilization using oversubscription & hardware threads

GPU cores use “parallel slack” to hide instruction pipeline latencies

<table>
<thead>
<tr>
<th>Time</th>
<th>Code</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
<th>Instruction Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ld %r0,%r1</td>
<td>ready</td>
<td>ready</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mul %r0,%r0,3</td>
<td>stall</td>
<td></td>
<td>ld %r0,%r1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>st %r0,%r1</td>
<td></td>
<td></td>
<td></td>
<td>mul %r0,%r0,3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>get …</td>
<td>!</td>
</tr>
<tr>
<td>mul %r0,%r0,3</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></td>
<td>mul %r0,%r0,3</td>
<td>ready</td>
<td>stall</td>
<td>mul</td>
</tr>
<tr>
<td></td>
<td>stall</td>
<td>mul %r0,%r0,3</td>
<td>ready</td>
<td>mul</td>
</tr>
<tr>
<td></td>
<td>ready</td>
<td>stall</td>
<td>mul %r0,%r0,3</td>
<td>mul</td>
</tr>
<tr>
<td></td>
<td>put …</td>
<td>ready</td>
<td>stall</td>
<td>put</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>

---

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

Device library

Put( ... );
Get( ... );
Wait( ... );
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)
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)
Weak scaling of MPI-CUDA and dCUDA for sparse-matrix vector multiplication

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)
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/)
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 3: 1

**dCUDA**

- **device 1**
  - rank 1: 1
  - rank 2: 1
  - rank 3: 2
  - rank 4: 2

- **device 2**
  - rank 3: 1
  - rank 4: 1
Implementation of the dCUDA runtime system

Event handler

Block manager

MPI

Host-side

Device-side

Logging queue
Command queue
Ack queue
Notification queue

Device library

Context

More blocks
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

dCUDA

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

device compute core active block
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