dCUDA: Distributed GPU Computing with Hardware Overlap

Tobias Gysi, Jeremia Bär, Lukas Kuster, and Torsten Hoefler
GPU computing gained a lot of popularity in various application domains

weather & climate  machine learning  molecular dynamics
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
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>ready</td>
<td>stall</td>
<td>mul %r0,%r0,3</td>
<td>mul</td>
</tr>
<tr>
<td>st %r0,%r1</td>
<td>stall</td>
<td>ready</td>
<td>st %r0,%r1</td>
<td>st</td>
</tr>
</tbody>
</table>

GPU cores use “parallel slack” to hide instruction pipeline latencies
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>get</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></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
How much “parallel slack” is necessary to fully utilize the interconnect?

Little’s law
\[ \text{concurrency} = \text{latency} \times \text{throughput} \]

<table>
<thead>
<tr>
<th></th>
<th>device memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>latency</td>
<td>~1(\mu)s</td>
</tr>
<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,
                         tag, lsend + rsend);

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

- iterative stencil kernel
- thread specific idx
- computation

- map ranks to blocks
- device-side put/get operations
- notifications for synchronization
- shared and distributed memory

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

Host-side:
- block manager

Device-side:
- device-library
  - put( ... );
  - get( ... );
  - wait( ... );

Direct communication:
- GPU direct

Event handler:
- MPI/IB

T. Gysi, J. Baer, TH: dCUDA: Hardware Supported Overlap of Computation and Communication, SC16
GPUDirect provides the NIC with direct device memory access

**idea**
- avoid copy to host memory
- host-side control

**performance**
- lower latency
- bandwidth penalty on Greina (2.7GB/s instead of 7.2GB/s)
System latencies of the IB-backend compared to the MPI-backend

benchmarked on Greina (4 Broadwell nodes with 1x Tesla K80 per node)

<table>
<thead>
<tr>
<th></th>
<th>IB-backend [µs]</th>
<th>MPI-backend [µs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>same device (put &amp; notify)</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>peer device (put &amp; notify)</td>
<td>6.7</td>
<td>23.7</td>
</tr>
<tr>
<td>remote device (put &amp; notify)</td>
<td>6.9</td>
<td>12.2</td>
</tr>
<tr>
<td>same device (notify)</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>peer device (notify)</td>
<td>3.4</td>
<td>5.0</td>
</tr>
<tr>
<td>remote device (notify)</td>
<td>3.6</td>
<td>5.4</td>
</tr>
</tbody>
</table>

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

benchmarked on Greina (4 Broadwell nodes with 1x Tesla K80 per node)
Weak scaling of IB-CUDA and dCUDA for a stencil program

benchmarked on Greina (4 Broadwell nodes with 1x Tesla K80 per node)
Weak scaling of IB-CUDA and dCUDA for sparse-matrix vector multiplication

benchmarked on Greina (4 Broadwell nodes with 1x Tesla K80 per node)

![Graph showing weak scaling of IB-CUDA and dCUDA for sparse-matrix vector multiplication. The graph compares execution time in milliseconds (ms) against the number of GPUs. The IB-CUDA and dCUDA lines show increasing execution time with the number of GPUs, highlighting the communication overhead. The IB-backend is indicated by a red triangle.]
Weak scaling of IB-CUDA and dCUDA for power iterations

benchmarked on Greina (4 Broadwell 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/)