dCUDA: Hardware Supported Overlap of Computation and Communication

Tobias Gysi, Jeremia Bär, 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

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>stall</td>
<td>stall</td>
<td></td>
<td>ld mul</td>
</tr>
<tr>
<td>st  %r0,%r1</td>
<td>ready</td>
<td></td>
<td>stall</td>
<td>st mul</td>
</tr>
<tr>
<td>mul %r0,%r0,3</td>
<td>stall</td>
<td></td>
<td>mul %r0,%r0,3</td>
<td>st mul</td>
</tr>
<tr>
<td>st  %r0,%r1</td>
<td>ready</td>
<td>stall</td>
<td></td>
<td>st mul</td>
</tr>
<tr>
<td></td>
<td>stall</td>
<td></td>
<td>st %r0,%r1</td>
<td>st st</td>
</tr>
<tr>
<td></td>
<td></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
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, get</td>
</tr>
<tr>
<td>put ...</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>!, get</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, 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>put ...</td>
<td>ready</td>
<td>stall</td>
<td>put, mul</td>
</tr>
</tbody>
</table>

introduce put & get operations to access distributed memory

![Instruction Pipeline Diagram](image-url)
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>
<th>interconnect</th>
</tr>
</thead>
<tbody>
<tr>
<td>latency</td>
<td>1µs</td>
<td>19µs</td>
</tr>
<tr>
<td>bandwidth</td>
<td>200GB/s</td>
<td>6GB/s</td>
</tr>
<tr>
<td>concurrency</td>
<td>200kB</td>
<td>114kB</td>
</tr>
<tr>
<td>#threads</td>
<td>~12000</td>
<td>&gt;&gt; ~7000</td>
</tr>
</tbody>
</table>
dCUDA (distributed CUDA) extends CUDA with MPI-3 RMA and notifications

```
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

<table>
<thead>
<tr>
<th>device 1</th>
<th>device 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank 1</td>
<td>rank 2</td>
</tr>
<tr>
<td>stencil( ... ); put( ... ); put( ... ); wait( ... );</td>
<td>stencil( ... ); put( ... ); put( ... ); wait( ... );</td>
</tr>
<tr>
<td>sync</td>
<td></td>
</tr>
<tr>
<td>stencil( ... ); put( ... ); put( ... ); wait( ... );</td>
<td>stencil( ... ); put( ... ); put( ... ); wait( ... );</td>
</tr>
<tr>
<td>sync</td>
<td></td>
</tr>
<tr>
<td>stencil( ... ); put( ... ); put( ... ); wait( ... );</td>
<td></td>
</tr>
</tbody>
</table>

performance
- avoid device synchronization
- latency hiding at cluster scale

complexity
- unified programming model
- one communication mechanism
Implementation of the dCUDA runtime system

- **event handler**
  - block manager
    - host-side
      - block manager
        - device-library
          - put( ... );
          - get( ... );
          - wait( ... );
    - device-library
      - put( ... );
      - get( ... );
      - wait( ... );
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)
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/)