Mapping MPI+X Applications to Multi-GPU Architectures
A Performance-Portable Approach

Edgar A. León
Computer Scientist

San Jose, CA
March 28, 2018
GPU Technology Conference
Application developers face greater complexity

- **Hardware architectures**
  - SMT
  - GPUs
  - FPGAs
  - NVRAM
  - NUMA, multi-rail

- **Programming abstractions**
  - MPI
  - OpenMP, POSIX threads
  - CUDA, OpenMP 4.5, OpenACC
  - Kokkos, RAJA

- **Applications**
  - Need the hardware topology to run efficiently
  - Need to run on more than one architecture
  - Need multiple programming abstractions

*Hybrid applications*
How do we map hybrid applications to increasingly complex hardware?

- Compute power is not the bottleneck
- Data movement dominates energy consumption
- HPC applications dominated by the memory system
  - Latency and bandwidth
  - Capacity tradeoffs (multi-level memories)
- Leverage local resources
  - Avoid remote accesses

More than compute resources, it is about the memory system!
The Sierra system that will replace Sequoia features a GPU-accelerated architecture

**Compute Node**
- 2 IBM POWER9 CPUs
- 4 NVIDIA Volta GPUs
- NVMe-compatible PCIe 1.6 TB SSD
- 256 GiB DDR4
- 16 GiB Globally addressable HBM2 associated with each GPU
- Coherent Shared Memory

**Compute Rack**
- Standard 19”
- Warm water cooling

**Compute System**
- 4320 nodes
- 1.29 PB Memory
- 240 Compute Racks
- 125 PFLOPS
- ~12 MW

**Components**

**IBM POWER9**
- Gen2 NVLink

**NVIDIA Volta**
- 7 TFlop/s
- HBM2
- Gen2 NVLink

**Mellanox Interconnect**
- Single Plane EDR InfiniBand
- 2 to 1 Tapered Fat Tree

**GPFS File System**
- 154 PB usable storage
- 1.54 TB/s R/W bandwidth
IBM Power8+
SL822LC

NVIDIA Pascal
Tesla P100

2017
CORAL EA

Figure generated with hwloc
Existing approaches and their limitations

- **MPI/RM approaches**
  - By thread
  - By core
  - By socket
  - Latency (IBM Spectrum MPI)
  - Bandwidth (IBM Spectrum MPI)

- **OpenMP approaches**
  - Policies
    - Spread, Close, Master
  - Predefined places
    - Threads, Cores, Sockets

- **Limitations**
  - Memory system is not primary concern
  - No coherent mapping across programming abstractions
  - No heterogeneous devices support
A portable algorithm for multi-GPU architectures: *mpibind*

- 2 IBM Power8+ processors
- Per socket
  - 10 SMT-8 cores
  - 1 InfiniBand NIC
  - 2 Pascal GPUs
- NVMe SSD
- Private L1, L2, L3 per core
mpibind's primary consideration: 
*The memory hierarchy*

Workers = 8
Vertices($l_2$) = 20

8 workers / 2 NUMA
4 workers / 2 GPUs
The algorithm

- Get hardware topology
- Devise memory tree $G$
  - Assign devices to memory vertices
- Calculate $\#$ workers $w$ (all processes and threads)
- Traverse tree to determine level $k$ with at least $w$ vertices
- Traverse subtrees selecting compute resources for each vertex
  \[ m': \text{vertices}(k) \rightarrow PU \]
- Map workers to vertices respecting NUMA boundaries
  \[ m: \text{workers} \rightarrow \text{vertices}(k) \rightarrow PU \]
Example mapping: One task per GPU
Evaluation: Synchronous collectives, GPU compute and bandwidth, app benchmark

<table>
<thead>
<tr>
<th>Machine</th>
<th>CORAL EA system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affinity</td>
<td>Spectrum-MPI default</td>
</tr>
<tr>
<td></td>
<td>Spectrum-MPI latency</td>
</tr>
<tr>
<td></td>
<td>Spectrum-MPI bandwidth</td>
</tr>
<tr>
<td></td>
<td>mpibind</td>
</tr>
<tr>
<td>Benchmarks</td>
<td>MPI Barrier</td>
</tr>
<tr>
<td></td>
<td>MPI Allreduce</td>
</tr>
<tr>
<td></td>
<td><strong>Bytes&amp;Flops</strong> Compute</td>
</tr>
<tr>
<td></td>
<td><strong>Bytes&amp;Flops</strong> Bandwidth</td>
</tr>
<tr>
<td></td>
<td><strong>SW4lite</strong></td>
</tr>
<tr>
<td>Number of Nodes</td>
<td>1, 2, 4, 8, 16</td>
</tr>
<tr>
<td>Processes (tasks) per node</td>
<td>4, 8, 20</td>
</tr>
</tbody>
</table>
Enabled uniform access to GPU resources

- Execute multiple instances concurrently
  - 4 and 8 PPN
- Measure GPU FLOPS
- Processes time-share GPUs by default
- Performance without mpibind severely limited because of GPU mapping

*kokkos/benchmarks/bytes_and_flops
Enabled access to the memory of all GPUs without user intervention

Memory bandwidth micro-benchmark*

- Execute multiple instances concurrently
  - 4 or 8 PPN
- Measure GPU global memory bandwidth
- Processes time-share GPUs by default
- Without mpibind some processes fail running out of memory

*kokkos/benchmarks/bytes_and_flops
Impact on SW4lite—Earthquake ground motion simulation

- Simplified version of SW4
  - Layer over half space (LOH.2)
  - 17 million grid points (h100)

- Multiple runs, calculate mean
  - 6 times for GPU
  - 10 times for CPU

- Performance speedup
  - CPU: mpibind over default: 3.7x
  - GPU: mpibind over bandwidth: 2.2x
  - GPU over CPU: 9.7x

Better

Mean Execution Time (secs)

<table>
<thead>
<tr>
<th>TPP</th>
<th>CPP x PPN</th>
<th>CPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>default</td>
<td>80</td>
<td>10</td>
</tr>
<tr>
<td>latency</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>mpibind</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

TPP: Threads per process, CPP: Cores per process, PPN: Processes per node, CPN: Cores per node
**mpibind**: A memory-driven mapping algorithm for multi-GPU architectures

- Focuses on hierarchical nature of memory system
- Provides portability and user transparency
  - Same algorithm on GPU-based, KNL-based, and commodity-based systems
- Encompasses
  - Hybrid programming abstractions
  - Heterogeneous devices
- Outperforms existing approaches without user intervention
  - Reduces runtime variability
  - Competitive performance on collective operations
  - Enables uniform access to all GPU resources
Bibliography and related GTC talks

- **mpibind**: A memory-centric affinity algorithm for hybrid applications. MEMSYS 2017.

- **System noise revisited: Enabling application scalability and reproducibility with SMT.** IPDPS 2016.

- **3D ground motion simulation in basins.** Final report to Pacific Earthquake Engineering Research Center, 2005.

- **SW4lite, Kokkos, and RAJA**
  - [github.com/geodynamics/sw4lite](https://github.com/geodynamics/sw4lite)
  - [github.com/kokkos/kokkos](https://github.com/kokkos/kokkos)
  - [github.com/LLNL/RAJA](https://github.com/LLNL/RAJA)

- **S8270** – Acceleration of an LLNL production Fortran application on the Sierra supercomputer

- **S8470** – Using RAJA for accelerating LLNL production applications on the Sierra supercomputer

- **S8489** – Scaling molecular dynamics across 25,000 GPUs on Sierra & Summit