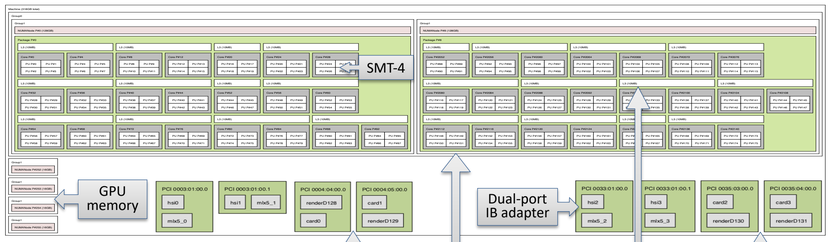
Cross-Architecture Affinity of Supercomputers

Edgar A. León, Lawrence Livermore National Laboratory

How to map hybrid applications portably to complex machines?

Emerging Systems		Application Needs
Many-core	Growing Gap	Portability
Heterogeneity		Productivity
Multi-level memory		Performance

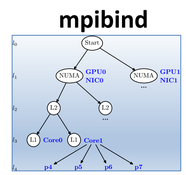
MPI
OpenMP, Pthreads
US DOE RAJA, Kokkos
CUDA, OpenMP 5.0, OpenACC



IBM Power CPUs + NVIDIA GPUs

Bridging the gap with *mpibind*: A memory-driven approach to map scientific applications

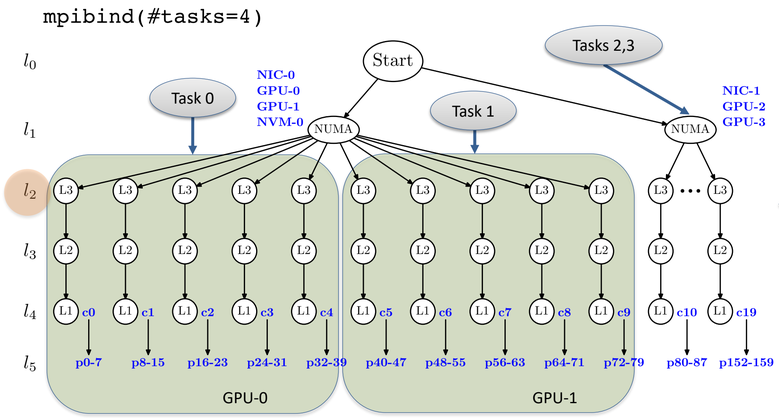
$$Perf \propto \frac{1}{Usability}$$ &



Performance portability

Memory-hierarchy driven

Primary consideration: *The memory hierarchy*



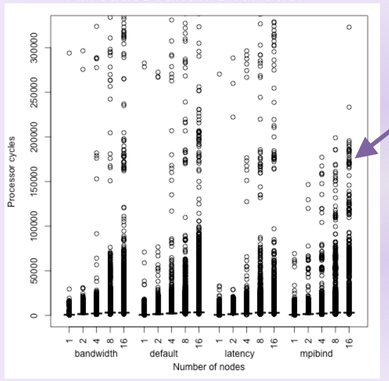

```
mpibind(#tasks, [#threads]):
        Mapping of MPI tasks, threads, GPU kernels
        to processing units and memory domains
```

1) Build memory tree
2) Assign compute resources to memory vertices
3) Distribute tasks over NUMA nodes
4) Traverse tree to match user workers to memory subtrees and associated compute resources

Key takeaways mapping parallel applications

- A memory-driven approach is key for performance portability
  - Many applications are memory bound, not compute bound
  - Memory tree can be derived on most systems
- Maintain a simple interface to improve productivity
  - Computational scientists can focus on science & engineering
- Avoid using hardware topology at the application level
  - May improve performance but may break portability
- **Provide a Rookie interface with Pro performance**
  - Provide a simple interface, but
  - Leverage advanced configurations to extract performance

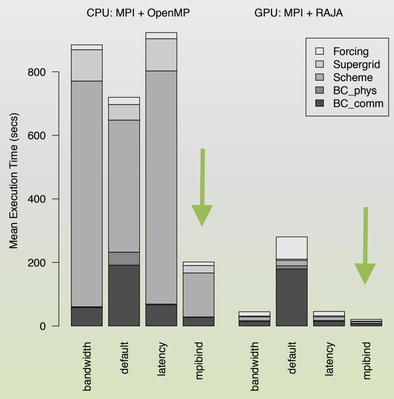
Allreduce runtime variation



Reduced variation range despite significant variability

Collective operations such as *Allreduce* and *Barrier* are sensitive to system noise

SW4lite execution time



Earthquake motion simulation: Improved performance on CPU and GPU variants

	TPP	CPP	PPN	*CPN*	
bandwidth	8	1	4	*4*	*under*
default	80	10	4	*40*	*over*
latency	8	1	4	*4*	*under*
mpibind	5	5	4	***20***	***balanced***

System with 20 SMT-8 cores and 4 GPUs

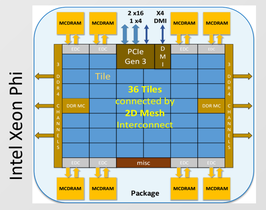
TPP = #threads per process
CPP = #cores per process
PPN = #processes per node
CPN = #cores per node

Evaluation—3 case studies

	Latency-Optimized Compute	Hybrid CPU + GPU	Many-Core Two-Level Memory
Architecture	IBM Power8+	NVIDIA Pascal GPUs w/ IBM Power8+	Intel Knights Landing
Memory	DDR	HBM + DDR	HBM + DDR
Affinity	Spectrum-MPI default	Spectrum-MPI default	OpenMPI default
	Spectrum-MPI latency	Spectrum-MPI latency	
	Spectrum-MPI bandwidth	Spectrum-MPI bandwidth	
	mpibind	mpibind	mpibind
Application Benchmarks	Allreduce: MPI	SW4lite: MPI+OpenMP, MPI+RAJA	Stream: MPI
			Kripke: MPI+OpenMP
Repetitions	1,000,000	10	10
Tasks per Node	4	4	64
Nodes	1, 2, 4, 8, 16	1	1
Case Study	System Noise Mitigation	Application Performance	Application Productivity

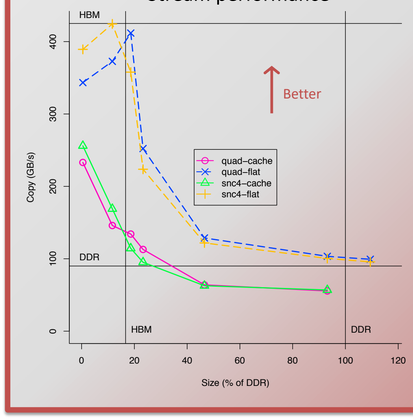
Application productivity w/2-level memory–HBM+DDR:

- **Bridge the gap between performance and ease-of-use**
- **Users can choose HW mode with exactly the same interface**

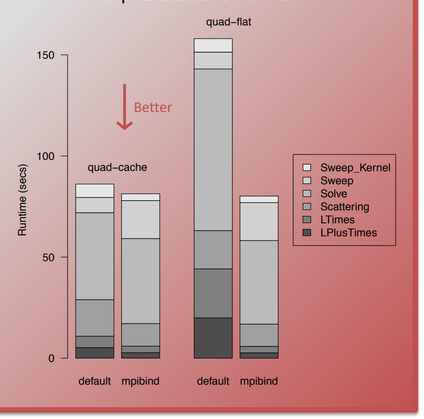


HW Mode		NUMA	Use	Perf.	Name
Quad	Cache	1	**High**	Low	Rookie
	Flat	1x2	Med	Med	Skilled
SNC4	Cache	4			
	Flat	4x2	Low	**High**	Pro

Stream performance



Kripke execution time



Prepared by LLNL under Contract DE-AC52-07NA27344. LLNL-POST-768850.