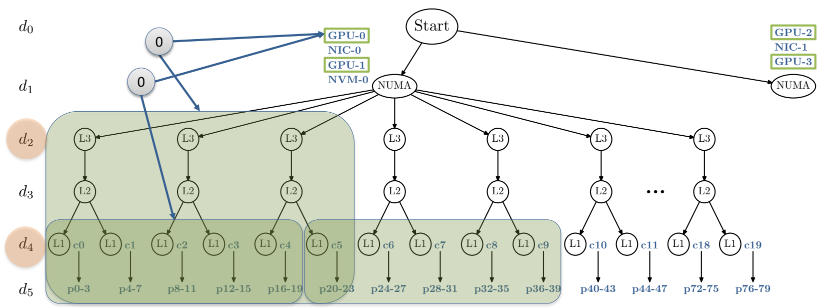
Memory-Based, Locality-Driven Supercomputing Affinity

mpibind

Edgar A. León
Livermore Computing

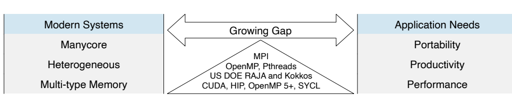
Scalable Tools Workshop
13 August 2024, Lake Tahoe, CA



Prepared by LLNL under contract DE-AC52-07NA27344. LLNL-PRES-862198.

Lawrence Livermore National Laboratory

HPC users face complex computing architectures



AMD Instinct MI300A

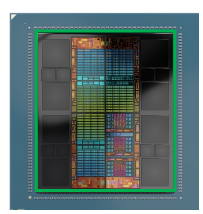


Image credit: AMD

AMD Instinct MI300X

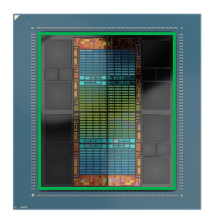
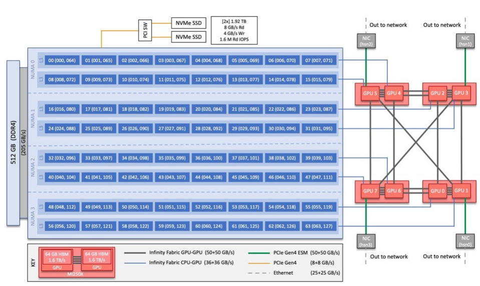


Image credit: AMD

AMD 3rd Gen EPYC CPU + AMD Instinct MI250X GPUs



https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html

NVIDIA GH200 Grace Hopper Superchip

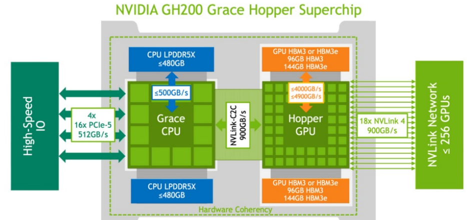
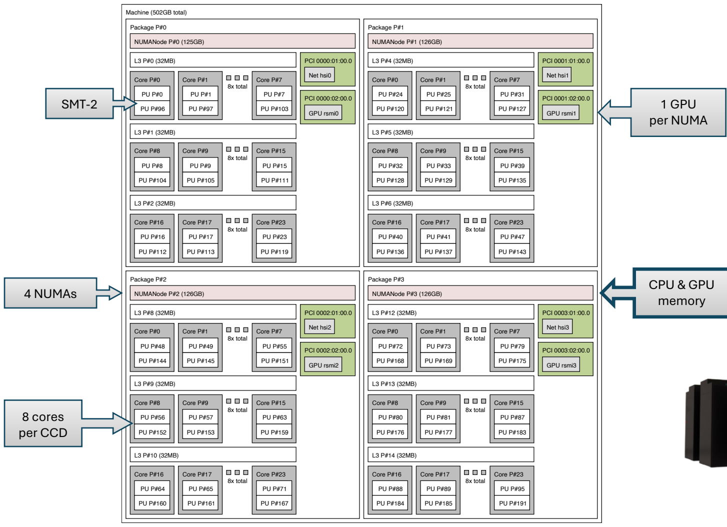
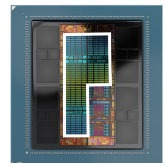


Image credit: NVIDIA

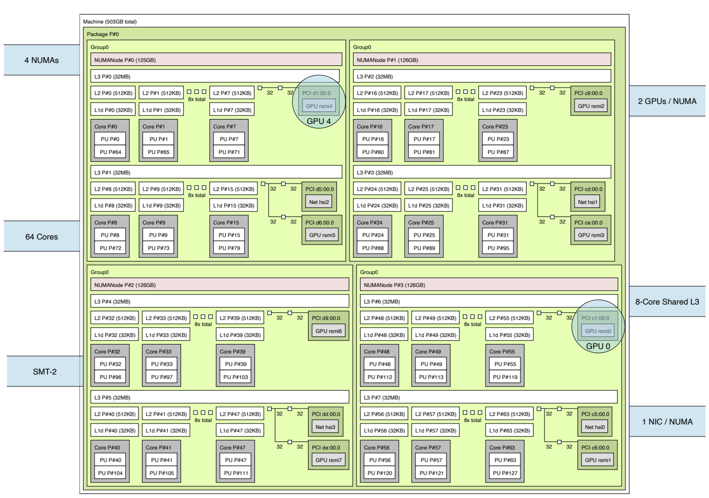
El Capitan's building block is AMD's MI300A APU





AMD Instinct MI300A APU

Mapping applications to the machine is hard



- What could go wrong?
  - Multiple threads running on a single core
  - Multiple GPU kernels sharing a GPU
  - Tasks launching kernels on non-local GPUs
  - Threads accessing remote memory
  - Etc.

mpibind is a memory-first, user-friendly algorithm

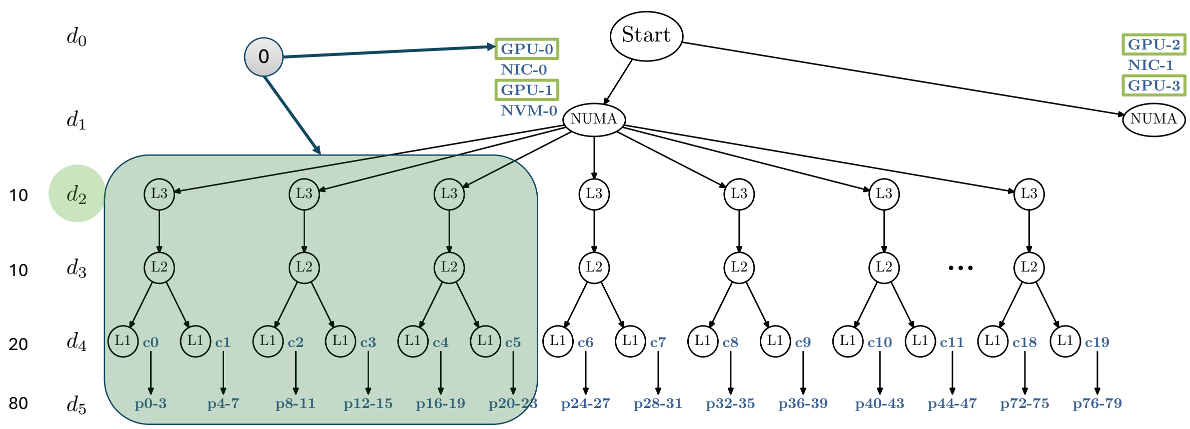
- Design principles
  - Driven by the memory system
  - Based on locality
  - Portable across architectures

mpibind

- Guiding principles
  - Provide a simple interface
  - Require minimal user input
  - Minimize remote memory accesses
  - Maximize cache per worker
  - Leverage compute/memory locality
  - Leverage architecture's features

mpibind uses the node topology to maximize memory and ensure locality

Tasks = 8
→Tasks/NUMA = 4
→ Workers = 4
→ Vertices(d_2) = 10



Provides a simple interface & Requires minimal user input

JSM

```
# 8 MPI tasks across all cores and all GPUs on a CORAL node
jsrun -a 2 -c 10 -g 1 -r 4 -d packed -b packed:5 <prog>
```

mpibind

```
# 8 MPI tasks across all cores and all GPUs on a CORAL node
srun -n8 <prog>
srun -n8 -mpibind=on <prog>
```

Required	Number of tasks
Optional	Number of threads
User options	Greedy (true or false)
	GPU optimized (true or false)
	SMT (1 to num. HW threads per core)

Provides portability across systems

- Counterexample

Intel MPI	I_MPI_PIN_DOMAIN	core, sock, numa, node, cache
	I_MPI_PIN_ORDER	range, scatter, compact, spread, bunch
MVAPICH2	MV2_CPU_BINDING_LEVEL	core, socket, numanode
	MV2_CPU_BINDING_POLICY	bunch, scatter, hybrid
	MV2_HYBRID_BINDING_POLICY	bunch, scatter, linear, compact, spread
OpenMPI	--bind-to, --rank-by	slot, hwthread, core, cache, socket, numa, board
	--map-by	+ node, sequential, distance, ppr:n:unit:pe=n
IBM Spectrum MPI	-aff	bandwidth, latency, cycle:unit

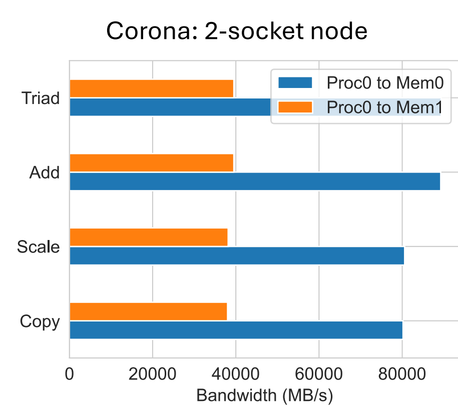
Provides portability across systems

- Relies on abstract memory-compute tree
  - Portable Hardware Locality (hwloc)
- Affinity algorithm is separate from applying affinity
  - C interface
    {Task} → {CPUs, GPUs, Thread mapping}
  - Slurm and Flux plugins
    - Use job info as input to mpibind
    - Get mpibind mapping
    - Bind tasks
    - Set environment variables
  - JSM wrapper (*lrun*)

mpibind

Minimizes remote memory accesses

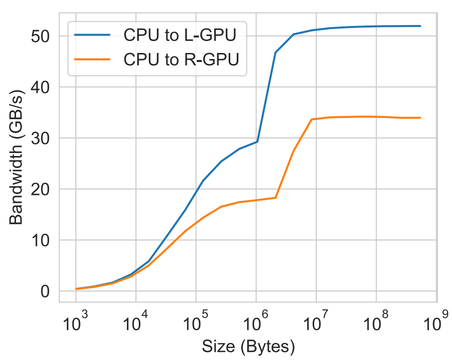
- Each task restricted to CPUs within a NUMA domain
- Leverage local memory
  - Spillover if necessary



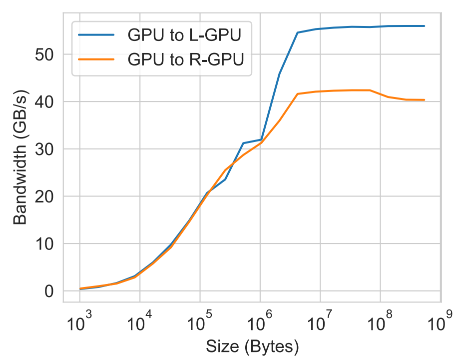
Selects local CPU-GPU sets

- Does it matter?

Up to 35% penalty for remote transfers



Up to 28% penalty for remote transfers



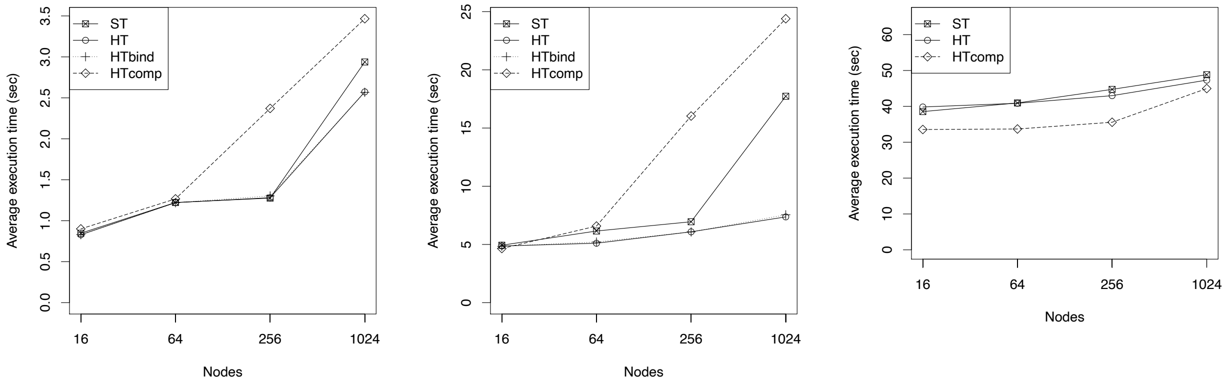
On LLNL systems with SMT we mitigate noise via thread specialization + mpibind

- Thread specialization
  - Application threads
  - System threads
- At boot time (TOSS)
  - Bind system processes to one HW thread of every core
  - *RHEL's tuna*
- At run time
  - Bind application to the other HW thread of every core
  - *LLNL's mpibind*
- But, applications have a choice!
  - `--mpibind=smt:<n>`
  - **Use all resources if needed**
- System noise studies
  - Must look at applications
  - Useful benchmarks
    - Parallel FWQ/FTQ
    - Synchronous collectives
  - Leon et al., IPDPS 2016, SC 2020

It matters how SMT is used by applications!

AMG and BLAST do not take advantage of additional HW threads, but significantly benefit from using them for system processes

pF3D can leverage the additional HW threads for compute



Leon et al., System Noise Revisited: Enabling Application Scalability and Reproducibility with SMT, IPDPS 2016

LLNL using mpibind in production since 2015

- Open source
  - MIT license
  - Written in C
  - Depends on hwloc
- Slurm SPANK plugin

  $ grep mpibind /etc/slurm/plugstack.conf
  required /usr/lib64/mpibind/mpibind_slurm.so default_off

- Building mpibind
  - GNU autotools

    bootstrap
    configure
    make
    make install
  - Spack

    spack install mpibind+rocm
    spack install mpibind+cuda

https://github.com/LLNL/mpibind

mpibind is an excellent initial policy, but…

- Not suited for apps with dynamically changing mappings
  - Static policy set at job start
- Not intended for benchmarking
  - Performance of remote memory, remote GPUs, etc.
- Does not replace custom mappings
  - Resource manager's affinity masks, etc.

- Advanced use cases will be covered by Quo Vadis
  https://github.com/hpc/quo-vadis

Documentation is available

- Tutorials
  - Flux affinity, Slurm affinity, mpibind
    https://github.com/LLNL/mpibind/tree/master/tutorials
- Articles

SC 2020	TOSS-2020: A commodity software stack for HPC
MEMSYS 2018	Achieving transparency mapping parallel applications: A memory hierarchy affair
GTC 2018	Mapping MPI+X applications to multi-GPU architectures: A performance-portable approach
MEMSYS 2017	Mpibind: A memory-centric affinity algorithm for hybrid applications
IPDPS 2016	System Noise Revisited: Enabling Application Scalability and Reproducibility with SMT

mpibind helps applications with performance, portability, and productivity

- Performance
  - Leverages local memory and local devices
  - Mitigates system noise via SMT
  - Maximizes cache per worker
- Productivity
  - Provides a simple interface
  - Requires minimal user input
- Portability
  - Same algorithm across system architectures, MPI libraries, and resource managers

mpibind

Fin