Supporting Many Task Workloads on Frontier using PMIx and PRRTE

Wael Elwasif, Thomas Naughton
Oak Ridge National Laboratory

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725

HPC Workflows Build Upon Task Management

- Workflow Engines coordinate the execution of compute tasks.
- Challenges using system software (PBS, Slurm, mpirun … etc):
  - Shell- based (one process per task)
  - Scalability
  - OS resource exhaustion
- Custom process management subsystem for each engine
  - Portability, maintenance, code bloat, …
- Use a standard
  - ***PMIx: Process Management Interface for Exascale***

Motivation

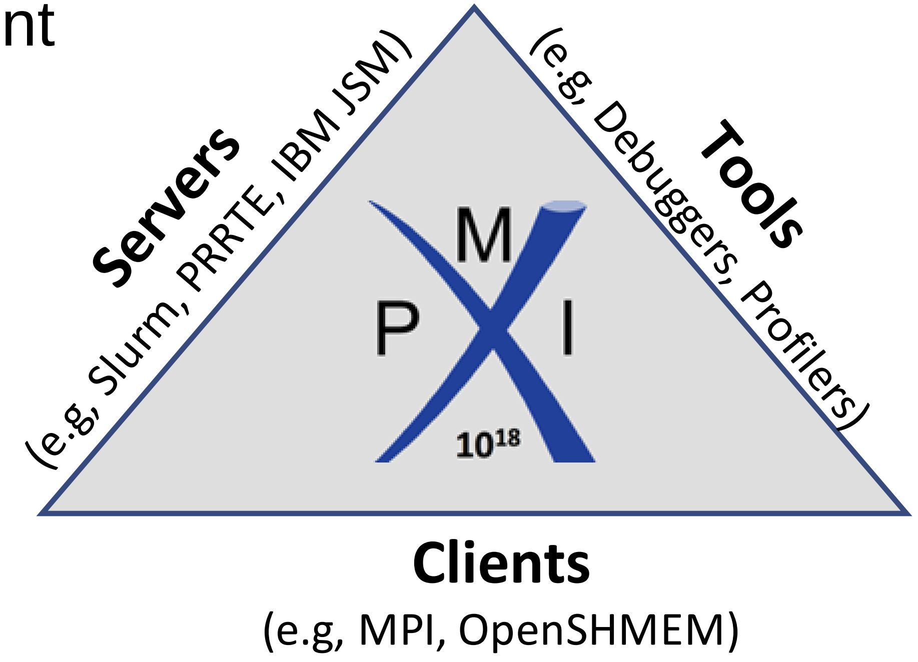
- Portability:
  - PMIx standard supported by several native system resource managers
    - Not all export the API to outside users
- Programmability
  - Using an API instead of command line tools for better control
- User-level deployment
  - Implementation deployed in user-space avoid impacting shared system resources

PMIx : **P**rocess **M**anagement **I**nterface for E**X**ascale

PMIx is a standard API providing libraries and programming models with portable and well-defined access to commonly available system services

- PMIx is messenger between software services (not a doer)
  - Facilitator of interactions between applications, tools & runtime environments
- Standard API for process and resource management
  - Specifications for Server/Client/Tool interfaces
- Open, community driven standard

https://PMIx.org



Source: https://PMIx.org

PMIx : **P**rocess **M**anagement **I**nterface for E**X**ascale

- Example use cases:
  - **Process wire-up** via either business card exchange or “instant on” (where supported)
  - **Tool connections** including debugger support
  - **Event notification** used by fault tolerant libraries
  - **Environment discovery** for Application/Job/Node information
  - **Job scheduler** interaction

- More information
  - Monthly status meetings & Quarterly voting meetings
  - Specification & Governance

https://github.com/pmix/pmix-standard
https://github.com/pmix/governance

PMIx 10^{18}

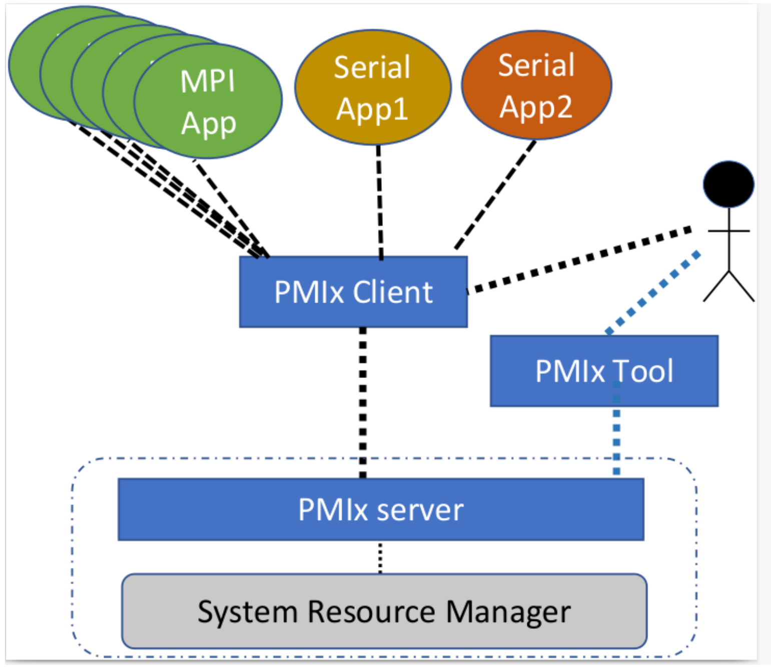
Source: https://PMIx.org

OpenPMIx

OpenPMIx is a feature complete implementation of PMIx Standard

- OpenPMIx provides C library implementation to connect PMIx-enabled clients (like Open MPI) with PMIx-enabled Tools (like debuggers) and PMIx-enabled Servers (like PRRTE, SLURM, IBM JSM)
  - Open, community supported, scalable implementation
  - Proving ground for new additions to PMIx Standard
  - Used on many large scale HPC systems
  - Cross-version compatibility allows clients to use a different version of OpenPMIx than the server or tool

https://OpenPMIx.org



PRRTE

PMIx Reference RunTime Environment (PRRTE) is a featureful, scalable, PMIx-enabled runtime environment

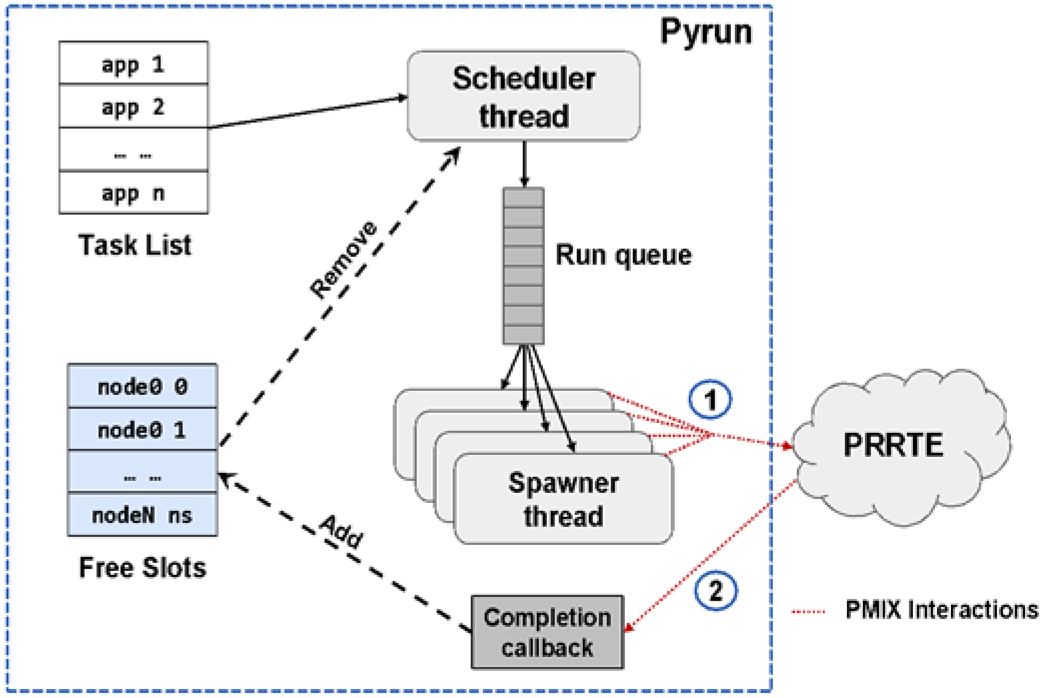
- PRRTE support interfaces needed for PMIx-enabled clients & tools to interact across HPC systems with a **portable** PMIx-enabled server
  - Offers PMIx support even if host environment is not PMIx-enabled
- Open, community supported, scalable implementation
  - Supports single instance jobs via **prterun** and multiple jobs via **prte/prun** (“DVM mode”)
  - Supports tools interface (to include replacement for MPIR)
- Origin: Based on the OpenRTE runtime from Open MPI, which evolved into a stand-alone project.

PMIx Python Binding

- Recent addition to PMIx (starting Ver. 4.1)
  - C bindings present a challenge for general adoption in many workflows
- Part of the OpenPMIx library
  - Use Cython to access C-layer in OPenPMIx
- Almost-direct translation of the C-API
  - Not all APIs are currently exported via the Python interface
- Maintain the callback-based design
  - C layer calls back into the Python binding/client code
- Python data structure simplify usage
  - List of dictionaries as the key argument type

Pyrun: Prototype Lightweight Python PMIx driver

- Target OpenPMIx and PRRTE
  - Rely on PRRTE persistent DVM functionality
  - PRRTE functionality only needed for daemons initialization
- Simple user-level FIFO scheduler with back-fill
- Multiple spawner threads for task injection
- Single PMIx tool connection to PRRTE
- Nonblocking, callback- based API



- Task List (applications)
  - Executable & arguments
  - Number of processes
    - Ex. MPI processes

- Scheduler
  - Free slot tracking
    - Generic counter, or Specific node(s)
  - FIFO queue with back-fill
  - Spawner threads consume tasks
  - Add/Remove slot tracking

- PMIx standard
  - Spawn tasks ①
  - Callbacks & Events ②

Pyrun PMIX interaction : Initialization

```
import pmix
tool = pmix.PMIxTool()
rc, my_proc = tool.init(
    [
        {
            "key": pmix.PMIX_SERVER_URI,
            "value": f"file:{dvm_file}",
            "val_type": pmix.PMIX_STRING,
        }
    ]
)
```

Initialize tools interface and connect to PRRTE daemons

- Pyrun presents itself as a PMIx tool
- Connect to a running PRRTE DVM using the URI in **`dvm_file`**

Pyrun PMIX interaction : callback

```
def done_cb(evhdlr: int, status: int, source: dict, info: list, results: list):
    with done_var:
        done_var.notify()

    return pmix.PMIX_EVENT_ACTION_COMPLETE, None

rc,myhandle = tool.register_event_handler([pmix.PMIX_EVENT_JOB_END], None, done_cb)
```

Define a job termination call back handler that notifies threads waiting on a condition variable

- Register an even handler for **`PMIX_EVENT_JOB_END`**
- Callback notify all threads waiting on a condition variable

Pyrun PMIX interaction: Spawn task(s)

- Spawn one (or more) PMIx apps on behalf of the calling process
- PMIx suport both blocking and non-blocking spawn
  - The Python client currently only supports blocking spawn
- `job_info` is shared across all apps
  - Specify mapping and binding, ..etc

```
job_info = [
    {
        "key": pmix.PMIX_MAPBY,
        "value": "core",
        "val_type": pmix.PMIX_STRING
    },
    {
        "key": pmix.PMIX_BINDTO,
        "value": "core",
        "val_type": pmix.PMIX_STRING
    },
]
exe = "/home/elwasif/bin/sleeper"
app = {
    "cmd": str(exe),
    "argv": [str(exe), "-n", "180"],
    "maxprocs": 8,
    "my_id": 1
    }
rc, nspace = tool.spawn(job_info, [app])
```

Define and spawn (nonblocking) a PMIx task on 8 cores – with binding and mapping to cores

PRRTE Binding, Mapping, and GPU affinity

- PRRTE is not GPU aware
  - Binding/mapping is done using CPU-only constructs
- Need binary wrappers to assign GPUs to processes
- Selection based on known CPU core/GPU NUMA affinity information
- Works for both parallel (MPI) and sequential tasks
- Supports:
  - Processes sharing the same GPU
  - Processes using more than one GPU

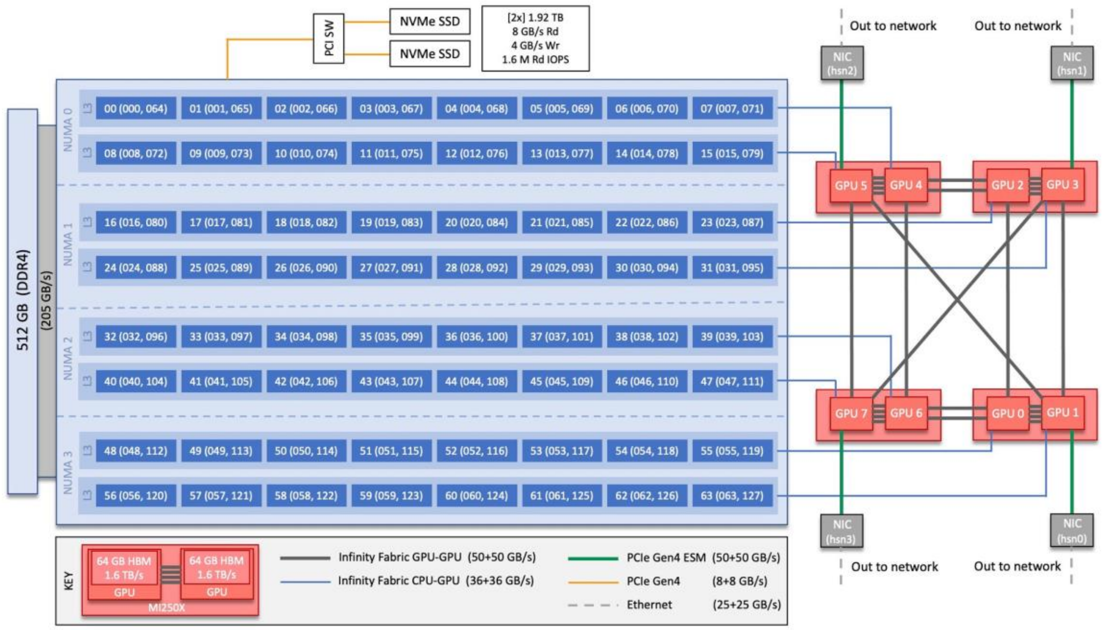
Wrapper Code

```bash
#!/usr/bin/bash

declare -A gpumap
declare -A visible

function map_gpu() {
    local c0=$1;   local c1=$2;
    local c2=$3;   local c3=$4;
    local gpu=$5;
    for c in $(seq $c0 $c1; seq $c2 $c3); do
        gpumap[$c]=$gpu
    done
}
#NUMA 0:
map_gpu 0 7 64 71 4;  map_gpu 8 15 72 79 5
#NUMA 1:
map_gpu 16 23 80 87 2; map_gpu 24 31 88 95 3
#NUMA 2:
map_gpu 32 39 96 103 6; map_gpu 40 47 104 111 7
#NUMA 3:
map_gpu 48 55 112 119 0; map_gpu 56 63 120 127 1
```

```bash
corelist=$(taskset -c -p $$ | awk '{print $NF}')
#Split the string based on the delimiter, ','
readarray -d , -t strarr <<<$(echo "$corelist")

length=${#strarr[*]}
for ((n = 0; n < $length; n++)); do
    entry="${strarr[$n]/$'\n'/}"
    #Split the string based on the delimiter, '-'
    readarray -d - -t cpus <<<"$entry"
    ntokens=${#cpus[*]}
    if [ $ntokens -eq 2 ]; then
        first=${cpus[0]};   last=${cpus[1]/$'\n’/};
        for c in $(seq $first $last); do
            visible[${gpumap[$c]}]=1
        done
    else
        visible[${gpumap[$entry]}]=1
    fi
done
devices="${!visible[@]}"
export ROCR_VISIBLE_DEVICES=${devices// /,}
export HIP_VISIBLE_DEVICES=$ROCR_VISIBLE_DEVICES
exec "$@"
```

Experimental Evaluations

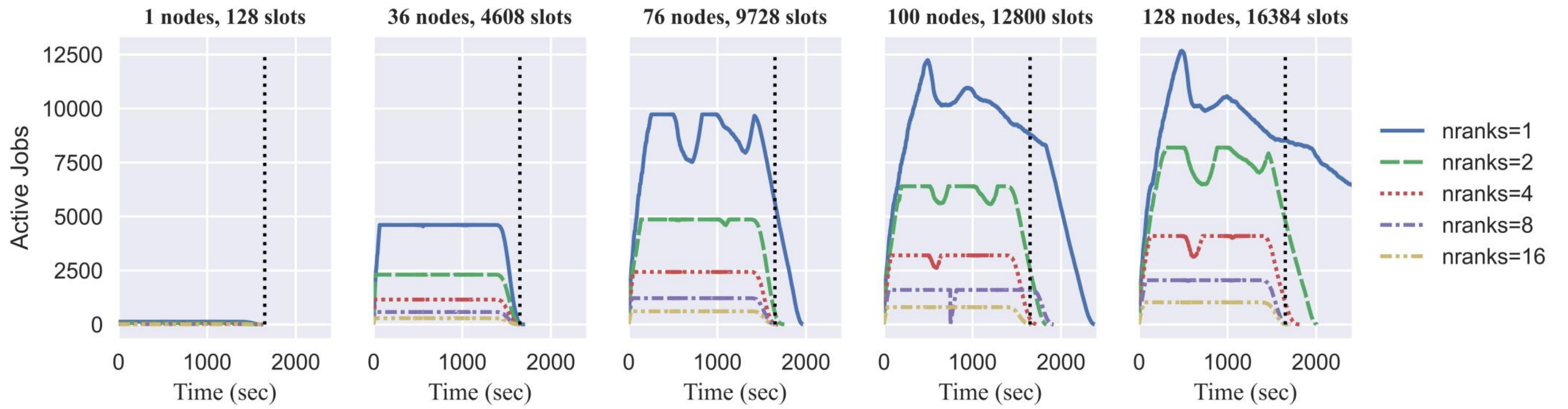
- Using prerelease OpenMPI 5.0, OpenPMIx 4.2
- Node architecture :
  - 64-Core AMD EPYC 7A53 (x 2 HW threads)
  - 512 GB DDR5 CPU memory
  - 4 AMD 250X GPUs (x2 GCD)
  - 128 HBM2E / GPU (64 GB/GCD)
  - Infinity Fabric CPU-GPU connection
  - 4x Slingshot 11 NIC
- Crusher : 192 Nodes
- Frontier : 9,048 Nodes

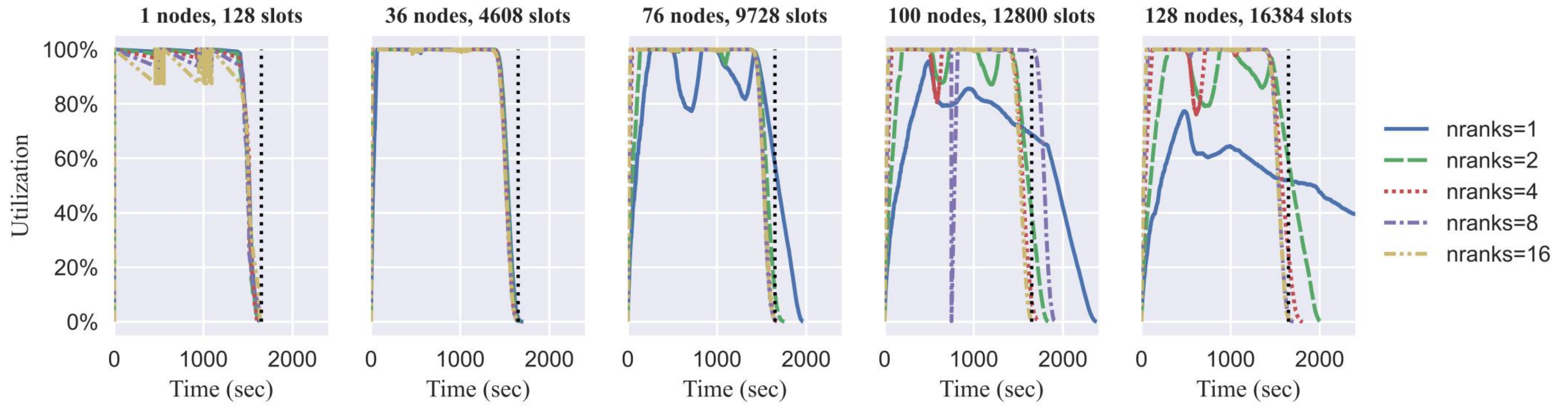


Crusher Scaling Experiments

- Assess ability of Pyrun/PRRTE to manage large number of independent tasks
- Many-task Workload:
  - MPI ring buffer with a random sleep duration in **`[450–550]`** Sec.
  - Prestage code to NVME on compute nodes to avoid FS issues
  - Use all 128 HW threads on compute nodes
  - ***NOT High Throughput workload***
- Pyrun uses a pool of available "slots" on which tasks are scheduled
- Sweep on different size allocations (<= 128 nodes)
- Number of tasks = 3 x N tasks to fill the entire allocation

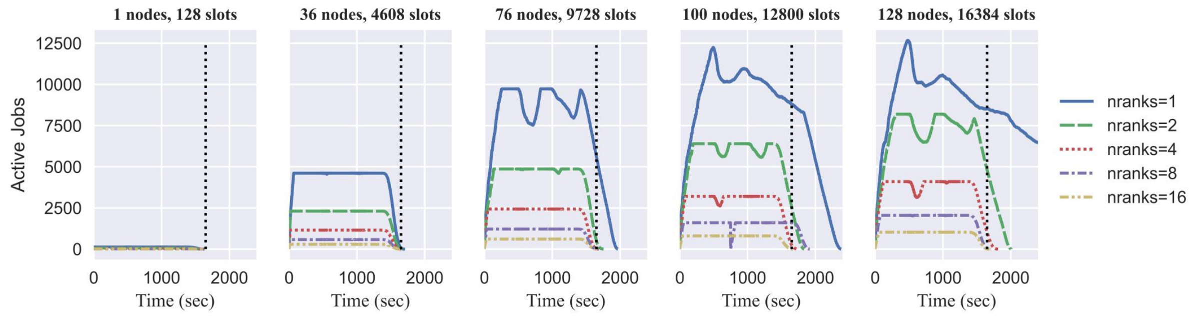
Crusher Scaling Experiments

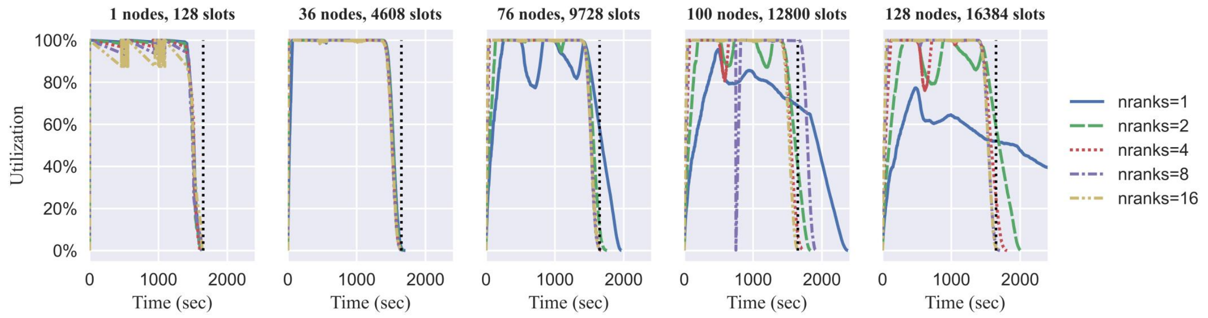




Crusher Scaling Experiments

- 128 nodes
  - **> 12500 concurrent tasks**
  - Utilization peaks at 80%
  - Cannot finish all tasks within allocation time
- Cannot sustain full utilization at high outstanding task count
  - Task launch overhead too large to keep up with terminating tasks

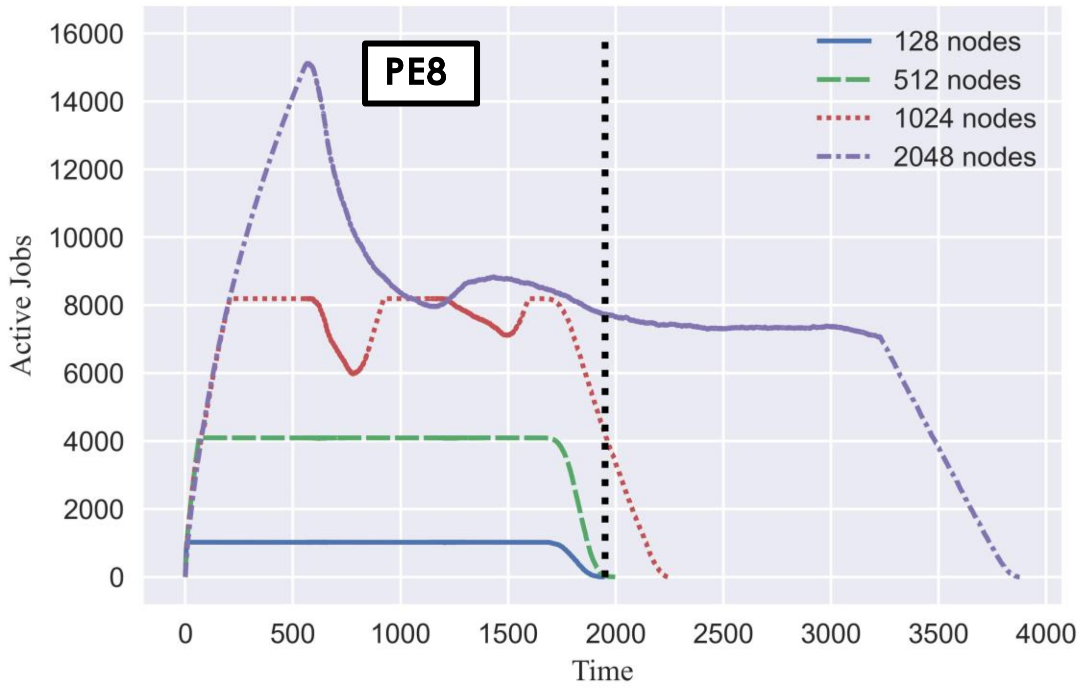


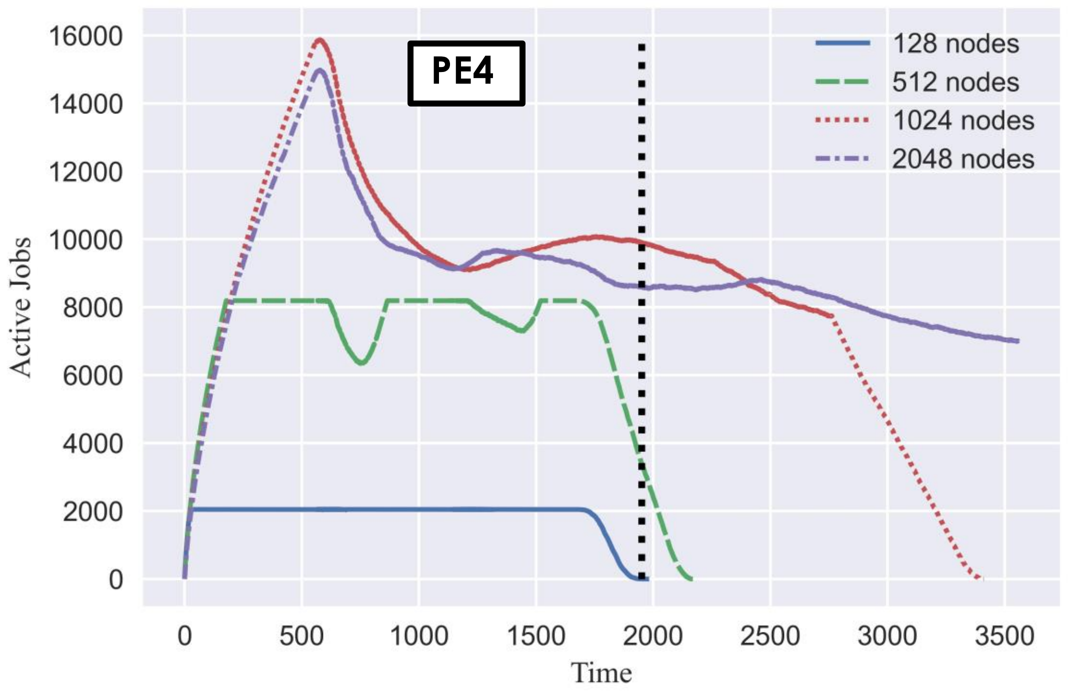


Frontier Experiments:

- Two Scenarios
  - **PE8**: **`1`** task per GPU - PRRTE mapby **`ppr:1:l3cache:corecpus:PE=8`**
  - **PE4**: **`2`** tasks per GPU - PRRTE mapby **`ppr:1:l3cache:corecpus:PE=4`**
- Workload:
  - Single rank, MPI ring buffer with a random sleep duration in **`[550–650]`** Sec.
  - Prestage code to NVME on compute nodes to avoid FS issues
  - Use all 64 cores on compute nodes
- Sweep on different size allocations (<= **2048** nodes)
- Number of tasks = 3 x N tasks to fill the entire allocation

Frontier Experiments: Active Jobs

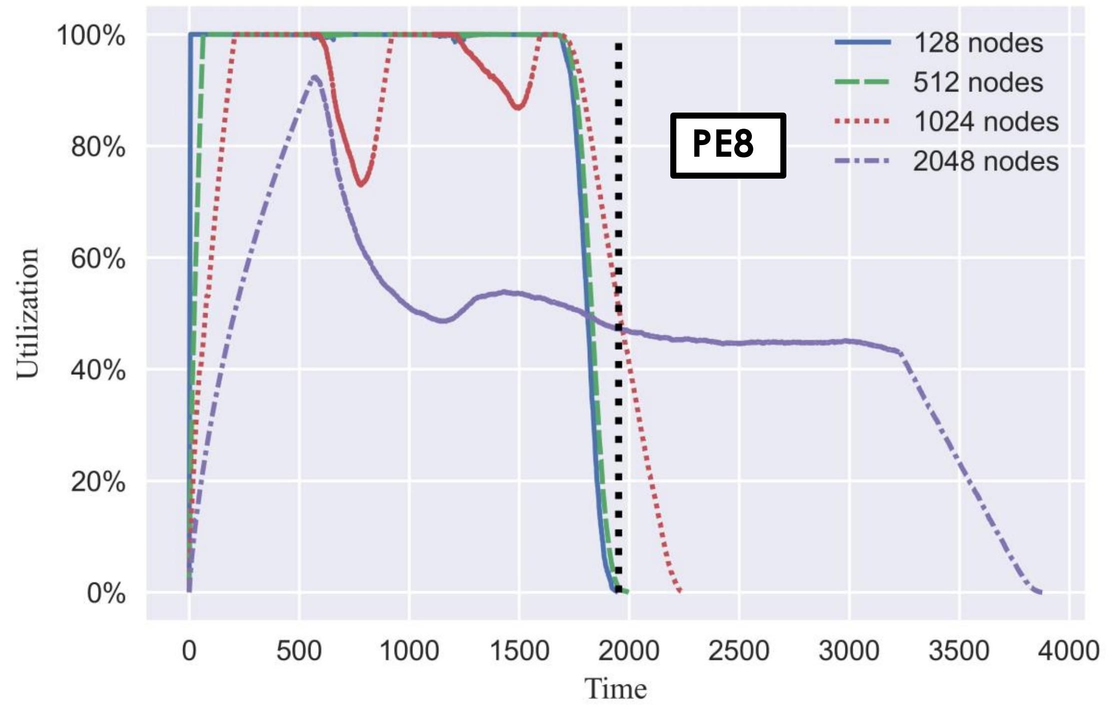


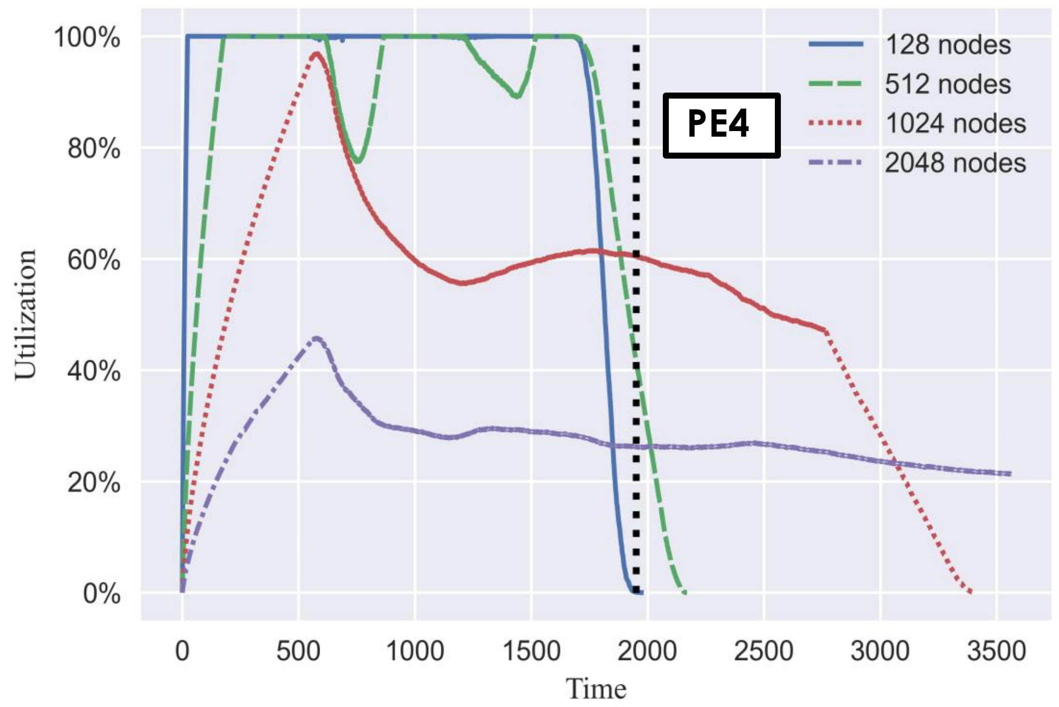


- Many tasks on PRRTE DVM scalable up to 2048 nodes ***(22.6 % System size)***
  - **O(16000) concurrent tasks**
  - Cannot finish all tasks within allocation time for **PE4** at 2048 nodes
- Cannot sustain full utilization at high outstanding task count
  - Task launch overhead too large to keep up with terminating tasks

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Frontier Experiments : Utilization

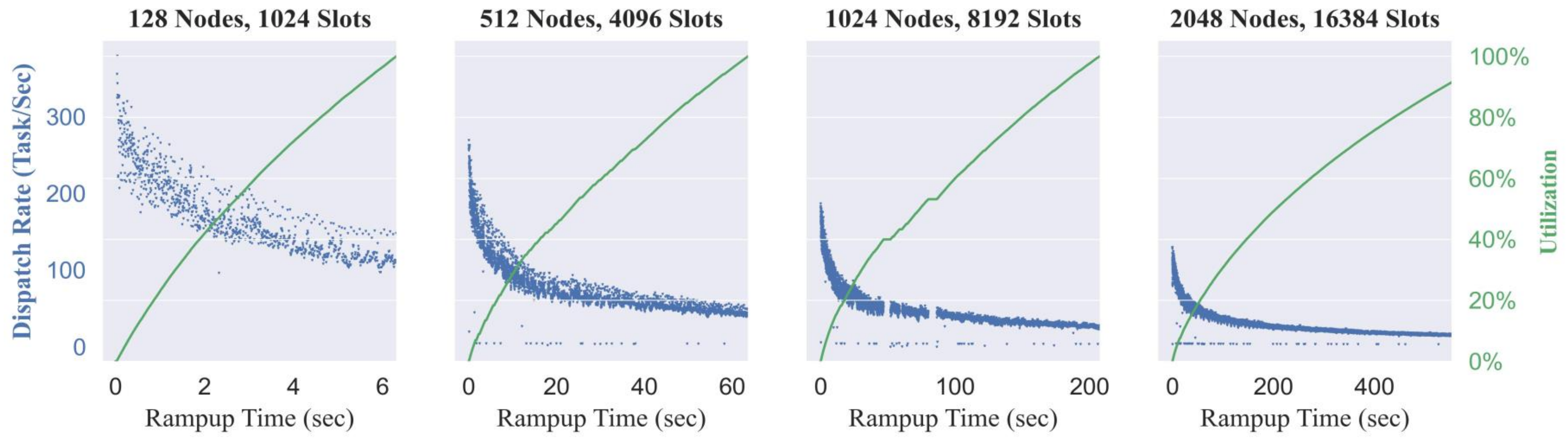




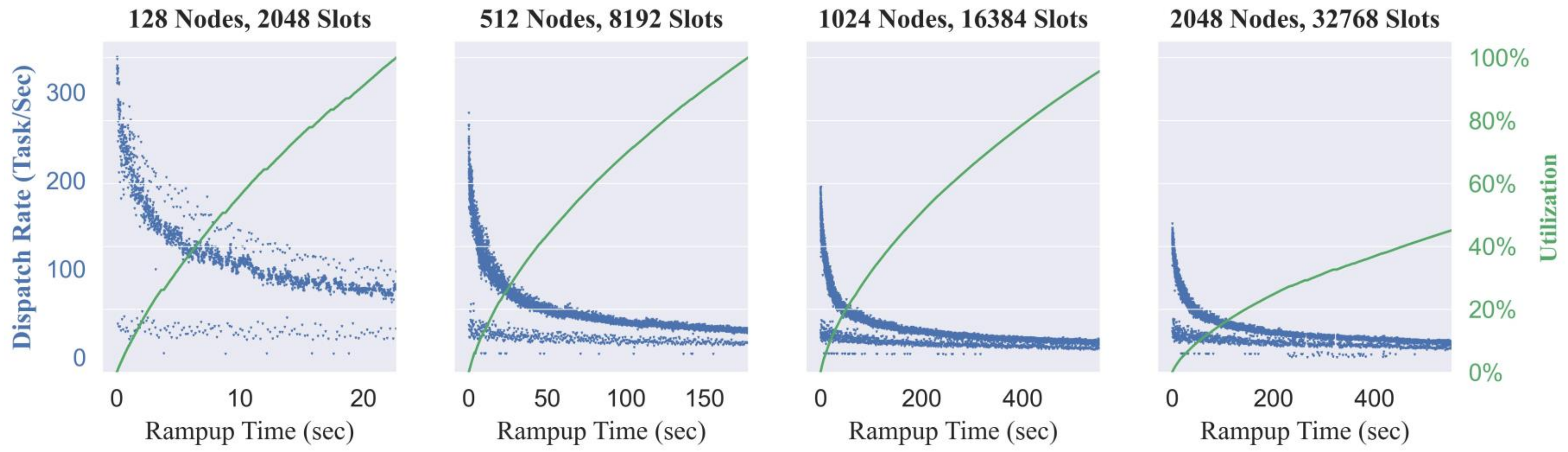
- Utilization drops with large number of outstanding tasks
  - Peak only **45%** for PE4 on 2048 nodes
- Ramp-up time too long for large task count
- Need to improve task launch latency to maintain high utilization

Frontier OpenPMIx/PRRTE Ramp Up Performance

```
t0 = time()
tool.spawn(...)
t1 = time()
y  = 1/(t1-t0)
```



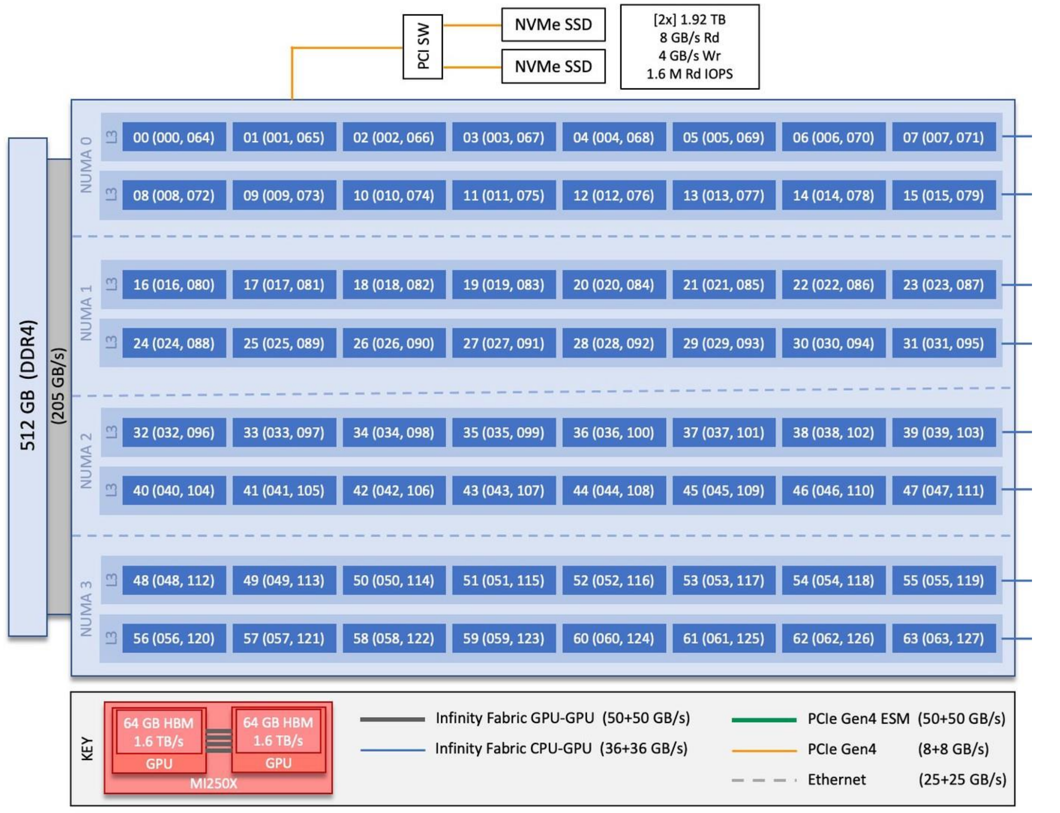
PE8: 8 tasks per node



PE4: 16 tasks per node

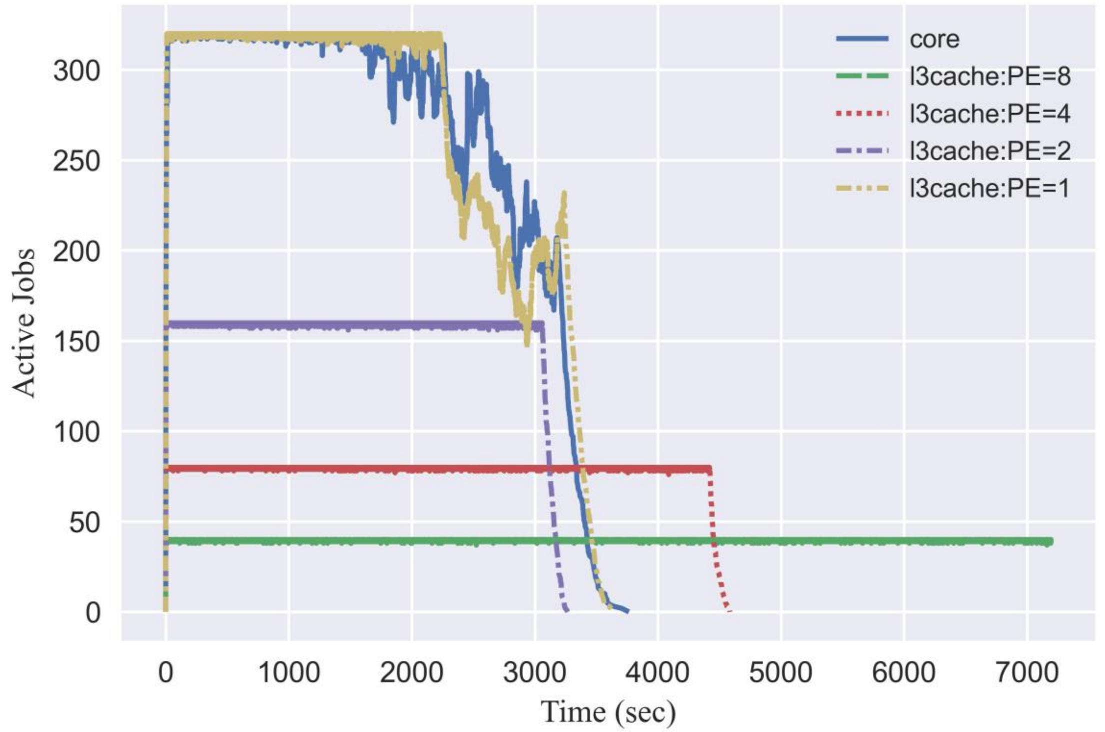
Crusher Experiments: NAS Benchmark

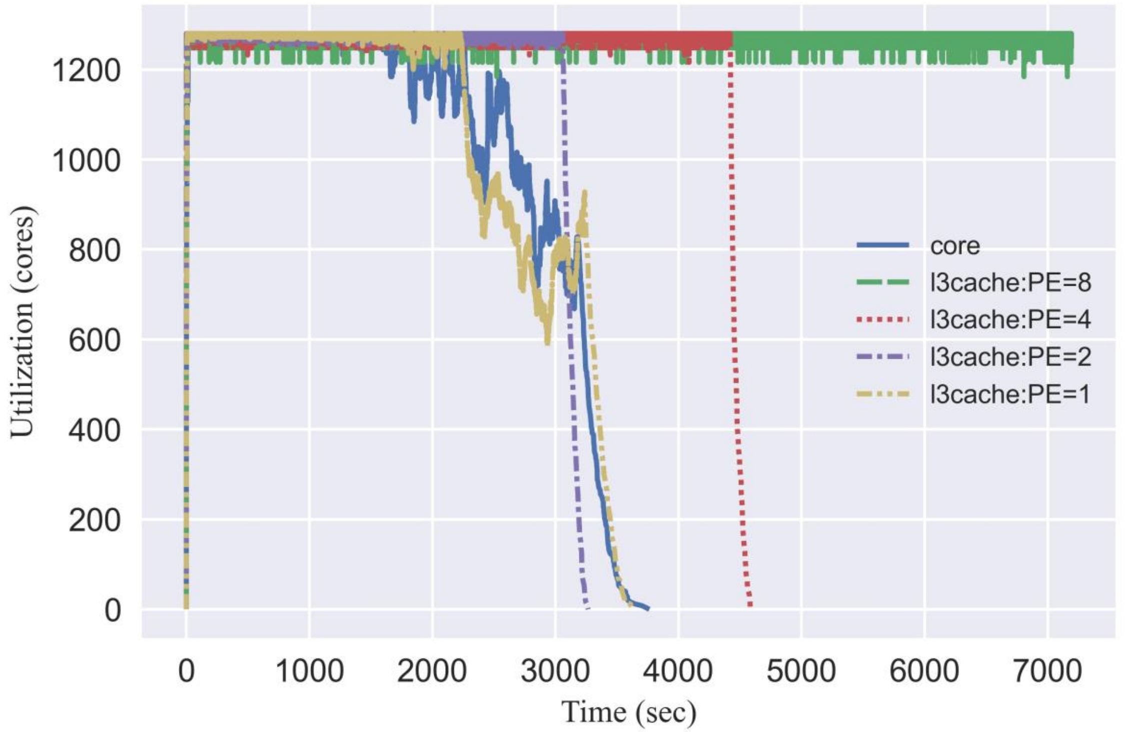
- Use Pyrun/PRRTE to drive hybrid many task workload
  - Exercise OpenMPI sub-node tasks for many task workloads
- Stude tradeoff between individual task performance and ensemble execution time
  - Improved utilization vs slow down due to sharing L3 cache
  - PE=8 : 1 rank per l3cache ….
    PE=1 : 8 ranks per l3cache
  - Ranks from the same MPI job spread on different l3caches
- Workload:
  - **20** Crusher nodes
  - **8000** 4-rank NAS benchmark class B, C problems
  - Variable runtime (range from **2 – 150** Seconds)
  - Prestage code to NVME on compute nodes to avoid FS issues
  - Use all 64 cores on compute nodes



OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Crusher NAS Experiments





- PE=8 fails to finish within allocated time
- PE=2 gives best ensemble execution time
- map by **core** places all 4-ranks from the same job on the same l3 cache domain
- More concurrent jobs makes up for degraded performance due to cache sharing

NAS Cache Sharing Effect

Code	Ntasks		μ		σ		σ/μ	
	PE=8	PE=2	PE=8	PE=2	PE=8	PE=2	PE=8	PE=2
bt.B.x	470	477	37.37	61.45	0.94	8.25	0.03	0.13
bt.C.x	511	525	150.53	242.94	2.10	26.59	0.01	0.11
cg.B.x	533	536	8.03	18.20	0.21	5.22	0.03	0.29
cg.C.x	490	495	22.28	50.66	0.46	12.14	0.02	0.24
ep.B.x	508	514	9.68	10.83	0.14	0.31	0.01	0.03
ep.C.x	514	518	34.95	39.25	0.57	0.20	0.02	0.01
ft.B.x	476	481	8.00	12.37	0.12	1.53	0.02	0.12
ft.C.x	483	492	28.95	46.12	0.30	5.79	0.01	0.13
is.B.x	490	492	2.19	2.83	0.07	0.31	0.03	0.11
is.C.x	503	508	4.99	7.32	0.07	1.00	0.01	0.14
lu.B.x	491	492	22.47	32.89	0.32	2.84	0.01	0.09
lu.C.x	498	510	93.60	129.83	1.18	9.64	0.01	0.07
mg.B.x	489	492	2.17	3.20	0.08	0.46	0.04	0.14
mg.C.x	479	483	8.46	16.74	0.13	3.10	0.02	0.19
sp.B.x	472	474	23.66	54.89	0.31	11.99	0.01	0.22
sp.C.x	501	511	115.71	255.39	0.86	41.84	0.01	0.16

Conclusions

- PMIx/PRRTE Python client usable as a standard process management layer for workflow engines.
  - Scalable to large node counts and large ensembles on PRRTE
- Provide a user-level programmable solution
  - Little or no impact on system resource and job managers
- Needed: Improvements in task launch latency
- Needed: Better PMIx vendor support
  - Export API for user-level access.

Questions?

Acknowledgement: This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research was partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborativ e effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY