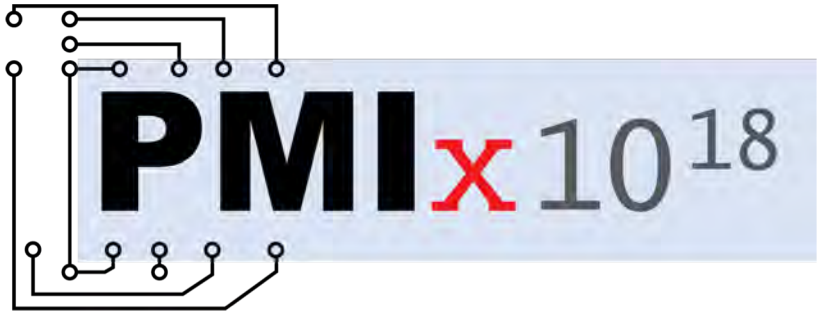


PMIx: State-of-the-Union

Ralph H. Castain
Intel
Joshua Hursey
IBM

Slack: pmix-workspace.slack.com

https://pmix.org

Agenda

- Quick status update
  - Library releases
  - Standards documents
- Application examples
  - PMIx Basics: Querying information
  - OpenSHMEM
  - Interlibrary coordination
  - MPI Sessions: Dynamic Process Grouping
  - Dynamic Programming Models (web only)

Overview Paper

PMIx: Process Management for Exascale Environments

Ralph H. Castain[a], Aurelien Bouteiller[b,1], Joshua Hursey[c], David Solt[c]

[a] *Intel, Inc.*
[b] *The University of Tennessee, Knoxville*
[c] *IBM*

Abstract

High-Performance Computing (HPC) applications have historically executed in static resource allocations, using programming models that ran independently from the resident system management stack (SMS). Achieving exascale performance that is both cost-effective and fits within site-level environmental constraints will, however, require that the application and SMS collaboratively orchestrate the flow of work to optimize resource utilization and compensate for on-the-fly faults. The Process Management Interface - Exascale (PMIx) community is committed to establishing scalable workflow orchestration by defining an abstract set of interfaces by which not only applications and tools can interact with the resident SMS, but also the various SMS components can interact with each other. This paper presents a high-level overview of the goals and current state of the PMIx standard, and lays out a roadmap for future directions.

Ralph H. Castain, Joshua Hursey, Aurelien Bouteiller, David Solt, “PMIx: Process management for exascale environments”, Parallel Computing, 2018.

https://doi.org/10.1016/j.parco.2018.08.002

Status Snapshot

- Standards Documents
  - v2.0 released Sept. 27, 2018
  - v2.1 (errata/clarifications, Dec.)
  - v3.0 (Dec.)

Current Library Releases

  - v4.0 (1Q19)

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

PMIx Standard – In Progress

- Clarify terms
  - Session, job, application
- New chapter detailing namespace registration data organization
- Data retrieval examples
  - Application size in multi-app scenarios
  - What rank to provide for which data attributes

Status Snapshot

- Standards Documents
  - v2.0 released Sept. 27, 2018
  - v2.1 (errata/clarifications, Dec.)
  - v3.0 (Dec.)

Current Library Releases

  - v4.0 (1Q19)
- Reference Implementation
  - v3.0 => v3.1 (Dec)
  - v2.1 => v2.2 (Dec)

New dstore implementation

Full Standard Compliance

What's in the Standard?

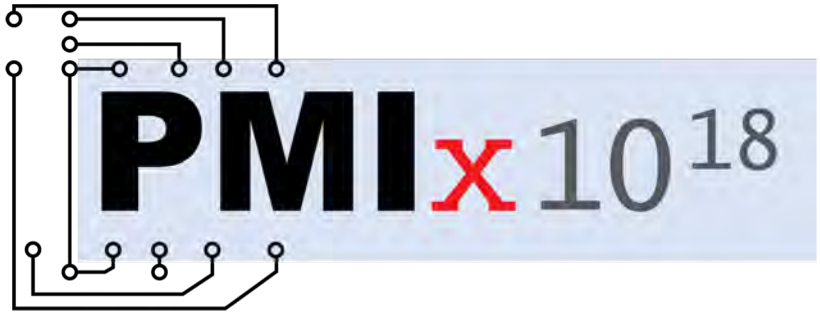
- v2 => workflow orchestration
  - (Ch.7) Job Allocation Management and Reporting, (Ch.8) Event Notification
- v3 => tools support
  - Security credentials, I/O management
- v4 => tools, network, dynamic programming models
  - Complete debugger attach/detach/re-attach
    - Direct and indirect launch
  - Fabric topology information (switches, NICs, …)
    - Communication cost, connectivity graphs, inventory collection
  - PMIx Groups
  - Bindings (Python, …)

Status Snapshot

- PRRTE
  - Formal releases to begin in Dec. or Jan.
  - Ties to external PMIx, libevent, hwloc installations
  - Support up thru current head PMIx master branch
- Cross-version support
  - Regularly tested, automated script
    - Help Wanted: Container developers for regular regression testing
  - v2.1.1 and above remain interchangeable
  - https://pmix.org/support/faq/how-does-pmix-work-with-containers/

Adoption Updates

- MPI use-cases
  - Re-ordering for load balance (UTK/ECP)
  - Fault management (UTK)
  - On-the-fly session formation/teardown (MPIF)
- MPI libraries
  - OMPI, MPICH, Intel MPI, HPE-MPI, Spectrum MPI, Fujitsu MPI
- Resource Managers (RMs)
  - Slurm, Fujitsu, IBM's Job Step Manager (JSM), PBSPro (2019), Kubernetes(?)
- Spark, TensorFlow
  - Just getting underway
- OpenSHMEM
  - Multiple implementations



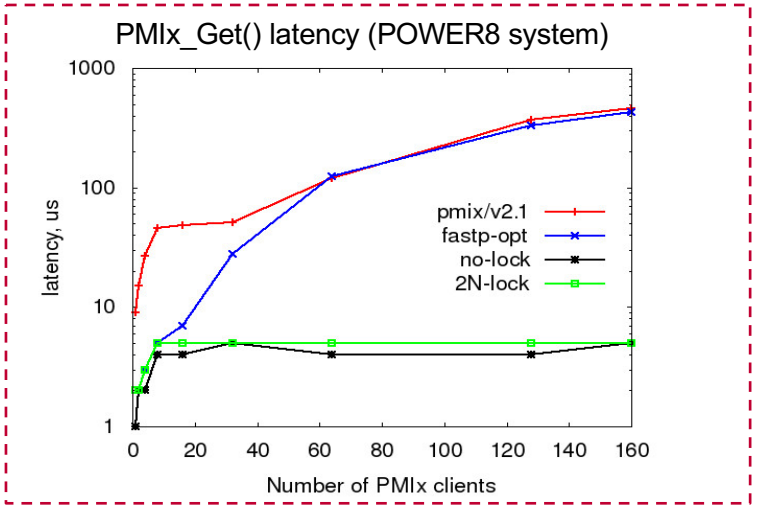
Shared Memory Optimizations (ds21)

Artem Y. Polyakov, Joshua S. Ladd, Boris I. Karasev

Mellanox Technologies

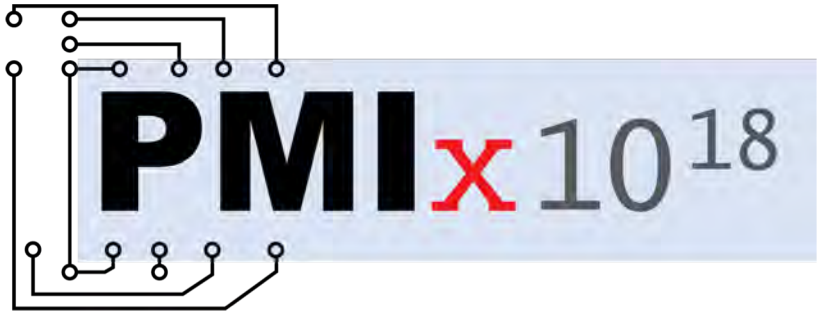
Mellanox update

- Improvements to intra-node performance
  - Highly optimized the intra-node communication component **dstor/ds21**[1]
    - Significantly improved the PMIx_Get latency using the *2N-lock* algorithm on systems with a large number of processes per node accessing many keys.
    - Designed for exascale: Deployed in production on CORAL systems, Summit and Sierra, POWER9-based systems.
    - Available in PMIx v2.2, v3.1, and subsequent releases.
      - Enabled by default.



- Improvements to the SLURM PMIx plug-in
  - Added a ring-based PMIx_Fence collective (Slurm v18.08)
  - Added support for high-performance RDMA point-to-point via UCX (Slurm v17.11) [2]
  - Verified PMIx compatibility through v3.x API (Slurm v18.08)

[1] *Artem Polyakov et al.* A Scalable PMIx Database (poster) EuroMPI'18: https://eurompi2018.bsc.es/sites/default/files/uploaded/dstore_empi2018.pdf
[2] *Artem Polyakov PMIx Plugin with UCX Support (SC17), SchedMD Booth talk: https://slurm.schedmd.com/publications.html*



PMIx Basics: Querying Information

Joshua Hursey
IBM

Harnessing PMIx capabilities

- https://pmix.org/support/faq/rm-provided-information/
- https://pmix.org/support/faq/what-information-is-an-rm-supposed-to-provide/

```
#include <pmix.h>
int main(int argc, char **argv) {
    pmix_proc_t myproc, proc_wildcard;
    pmix_value_t value;
    pmix_value_t *val = &value;

    PMIx_Init(&myproc, NULL, 0);
    PMIX_PROC_CONSTRUCT(&proc_wildcard);
    (void)strncpy(proc_wildcard.nspace, myproc.nspace, PMIX_MAX_NSLEN);
    proc_wildcard.rank = PMIX_RANK_WILDCARD;

    PMIx_Get(&proc_wildcard, PMIX_JOB_SIZE, NULL, 0, &val);
    job_size = val->data.uint32;
    PMIx_Get(&myproc, PMIX_HOSTNAME, NULL, 0, &val);
    strncpy(hostname, val->data.string, 256);

    printf("%d/%d) Hello World from %s\n", myproc.rank, job_size, hostname);

    PMIx_Get(&proc_wildcard, PMIX_LOCALLDR, NULL, 0, &val); // Lowest rank on this node
    printf("%d/%d) Lowest Local Rank: %d\n", myproc.rank, job_size, val->data.rank);
    PMIx_Get(&proc_wildcard, PMIX_LOCAL_SIZE, NULL, 0, &val); // Number of ranks on this node
    printf("%d/%d) Local Ranks: %d\n", myproc.rank, job_size, val->data.uint32);

    PMIx_Fence(&proc_wildcard, 1, NULL, 0); // Synchronize processes (not required, just for demo)
    PMIx_Finalize(NULL, 0); // Cleanup
    return 0;
}
```

Harnessing PMIx capabilities

- https://pmix.org/support/how-to/example-direct-launch-debugger-tool/
- https://pmix.org/support/how-to/example-indirect-launch-debugger-tool/

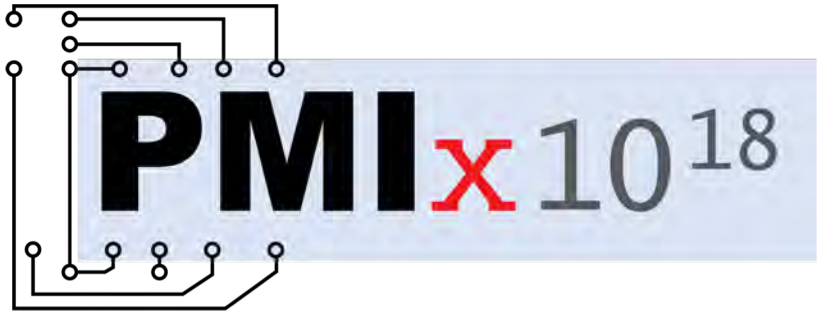
```
volatile bool isdone = false;
static void cbfunc(pmix_status_t status, pmix_info_t *info, size_t ninfo, void *cbdata,
                   pmix_release_cbfunc_t release_fn, void *release_cbdata)
{
    size_t n, i;
    if (0 < ninfo) {
        for (n=0; n < ninfo; n++) {
            // iterate trough the  info[n].value.data.darray->array of
/*
typedef struct pmix_proc_info {
    pmix_proc_t proc;
    char *hostname;
    char *executable_name;
    pid_t pid;
    int exit_code;
    pmix_proc_state_t state;
} pmix_proc_info_t;
*/
        }
    }

    if (NULL != release_fn) {
        release_fn(release_cbdata);
    }
    isdone = true;
}
```

```
#include <pmix.h>
int main(int argc, char **argv) {
    pmix_query_t *query;
    char clientspace[PMIX_MAX_NSLEN+1];
    // ... Initialize and setup - PMIx_tool_init()...

    /* get the proctable for this nspace */
    PMIX_QUERY_CREATE(query, 1);
    PMIX_ARGV_APPEND(rc, query[0].keys, PMIX_QUERY_PROC_TABLE);
    query[0].nqual = 1;
    PMIX_INFO_CREATE(query->qualifiers, query[0].nqual);
    PMIX_INFO_LOAD(&query->qualifiers[0], PMIX_NSPACE, clientspace, PMIX_STRING

    if (PMIX_SUCCESS != (rc = PMIx_Query_info_nb(query, 1, cbfunc, NULL))) {
        exit(42);
    }
    /* wait to get a response */
    while( !isdone ) { usleep(10); }
    // ... Finalize and cleanup
}
```



OSSS OpenSHMEM-on-UCX / Fault Tolerance

Tony Curtis <anthony.curtis@stonybrook.edu**>**
Dr. Barbara Chapman
Abdullah Shahneous Bari Sayket, Wenbin Lü, Wes Suttle
Stony Brook University

https://www.iacs.stonybrook.edu/

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- OpenSHMEM
  - Partitioned Global Address Space library
  - Take advantage of RDMA
    - *put/get* directly to/from remote memory
    - Atomics, locks
    - Collectives
    - And more…

http://www.openshmem.org/

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- OpenSHMEM: our bit
  - Reference Implementation
    - OSSS + LANL + SBU + Rice + Duke + Rhodes
    - Wireup/maintenance: PMIx
    - Communication substrate: UCX
      - shm, IB, uGNI, CUDA, …

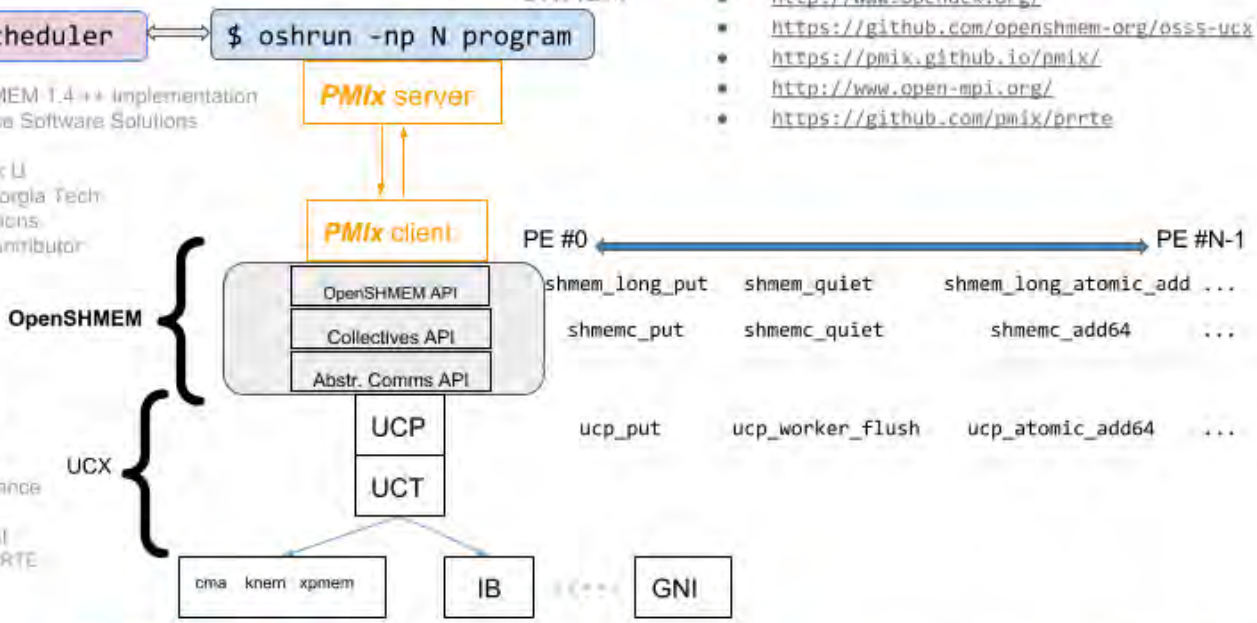
OSSS OpenSHMEM-on-UCX

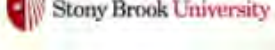
Tony Curtis (SBU), Howard Pritchard (LANL)

http://www.openshmem.org/

- http://www.openucx.org/
- https://github.com/openshmem-org/osss-ucx
- https://pmix.github.io/pmix/
- http://www.open-mpi.org/
- https://github.com/pmix/prrte

- Reference OpenSHMEM 1.4 ++ implementation
  - Open Source Software Solutions
  - LANL
  - Stony Brook U
  - Rice U / Georgia Tech
- UCX for communications
  - User and contributor
- PMIx for startup, resilience
- Program launch via
  - mpiexec:
  - Open-MPI
  - PMIx Reference RunTime Environment
    - PRRTE





PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- Reference Implementation
  - PMIx/UCX for OpenSHMEM ≥ 1.4
  - Also for < 1.4 based on GASNet
  - Open-Source @ github
    - Or my dev repo if you're brave
    - Our research vehicle

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- Reference Implementation
  - PMIx/PRRTE used as o-o-b startup / launch
  - Exchange of wireup info e.g. heap addresses/rkeys
  - Rank/locality and other “ambient” queries
  - Stays out of the way until OpenSHMEM finalized
  - Could kill it once we’re up but used for fault tolerance project

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- Fault Tolerance (NSF) #1
  - Project with UTK and Rutgers
  - Current OpenSHMEM specification lacks general fault tolerance (FT) features
  - PMIx has basic FT building blocks already
    - Event handling, process monitoring, job control
  - Using these features to build FT API for OpenSHMEM

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

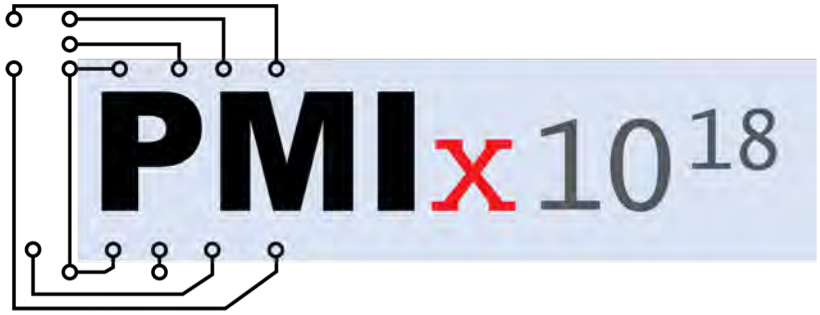
- Fault Tolerance (NSF) #2
  - User specifies
    - Desired process monitoring scheme
    - Application-specific fault mitigation procedure
  - Then our FT API takes care of:
    - Initiating specified process monitoring
    - Registering fault mitigation routine with PMIx server
  - PMIx takes care of the rest

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- Fault Tolerance (NSF) #3
  - Longer-term goal: automated selection and implementation of FT techniques
  - Compiler chooses from a small set of pre-packaged FT schemes
  - Appropriate technique selected based on application structure and data access patterns
  - Compiler implements the scheme at compile-time

PMIx@SC18: OSSS OpenSHMEM-on-UCX / Fault Tolerance

- PMIx @ github
  - We try to keep up on the bleeding edge of both PMIx & PRRTE for development
  - But make sure releases work too!
  - We've opened quite a few tickets
    - Always fixed or nixed quickly!



OSSS OpenSHMEM-on-UCX / Fault Tolerance

http://www.openshmem.org/

http://www.openucx.org/

Any (easy) Questions?

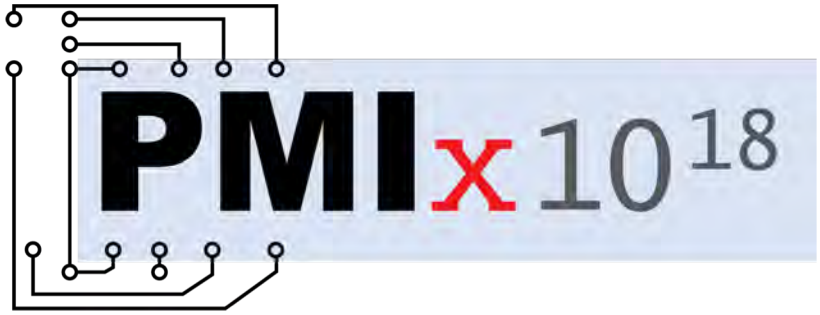
Tony Curtis <anthony.curtis@stonybrook.edu**>**

Dr. Barbara Chapman

Abdullah Shahneous Bari Sayket, Wenbin Lü, Wes Suttle

Stony Brook University

https://www.iacs.stonybrook.edu/



Runtime Coordination Based on PMIx

Geoffroy Vallee

Oak Ridge National Laboratory

OAK RIDGE National Laboratory

ECP EXASCALE COMPUTING PROJECT

Introduction

- Many pre-exascale systems show a fairly drastic hardware architecture change
  - Bigger/complex compute nodes
  - Summit: 2 IBM Power-9 CPUs, 6 GPUs per node
- Requires most scientific simulations to switch from pure MPI to a MPI+X paradigm
- MPI+OpenMP is a popular solution

Challenges

- MPI and OpenMP are independent communities
- Implementations un-aware of each other
  - MPI will deploy ranks without any knowledge of the OpenMP threads that will run under each rank
  - OpenMP assumes by default that all available resources can be used

It is very difficult for users to have a fine-grain control over application execution

Context

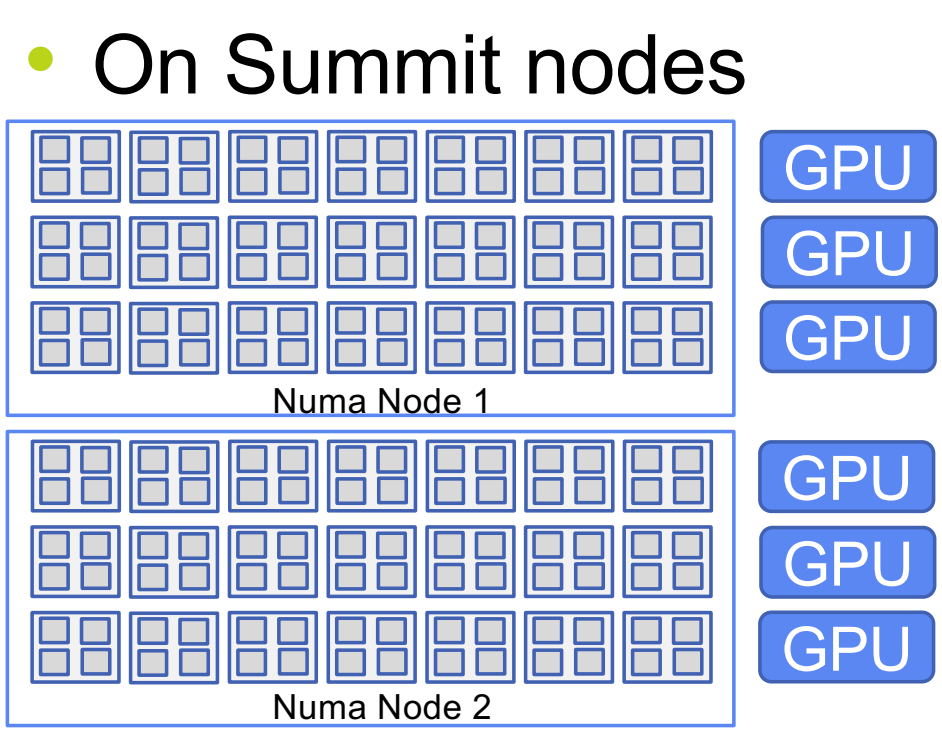
- U.S. DOE Exascale Computing Project (ECP)
- ECP OMPI-X project
- ECP SOLLVE project
- Oak Ridge Leadership Computing Facility (OCLF)

Challenges (2)

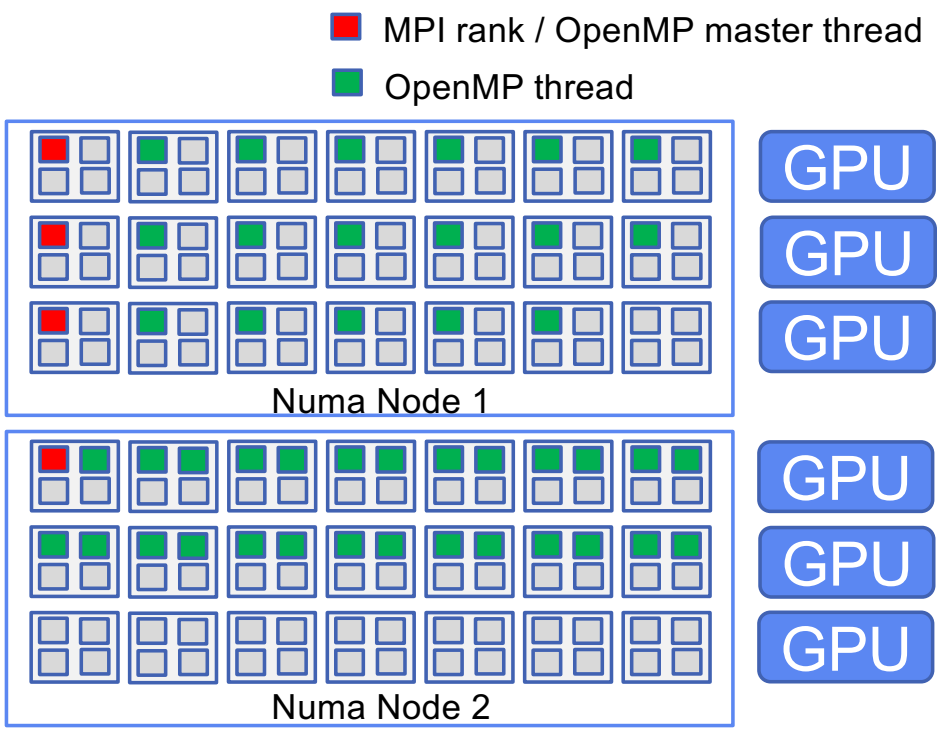
- Impossible to coordinate runtimes and optimize resource utilization
  - Runtimes independently allocate resources
  - No fine-grain & coordinated control over the placement of MPI ranks and OpenMP threads
  - Difficult to express affinity across runtimes
  - Difficult to optimize applications

Illustration

- On Summit nodes



Node architecture



Desired placement

Proposed Solution

- Make all runtimes PMIx aware for data exchange and notification through events
- Key PMIx characteristics
  - Distributed key/value store for data exchange
  - Asynchronous events for coordination
  - Enable interactions with the resource manager
- Modification of LLVM OpenMP and Open MPI

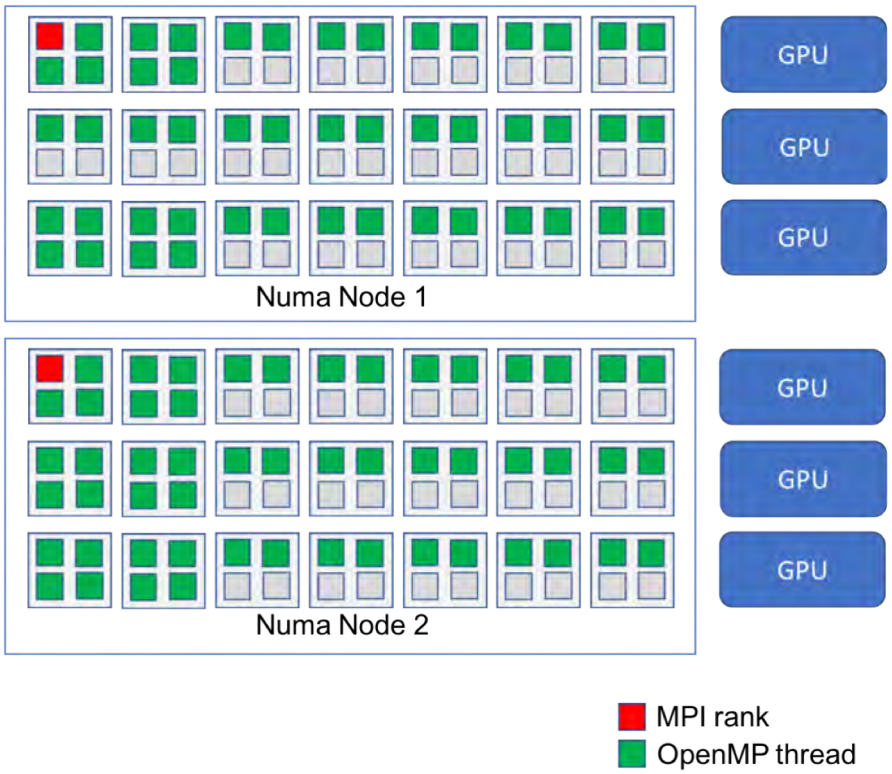
Two Use-cases

- Fine-grain placement over MPI ranks and OpenMP threads (prototyping stage)
- Inter-runtime progress (design stage)

Fine-grain Placement

- Goals
  - Explicit placement of MPI ranks
  - Explicit placement of OpenMP threads
  - Re-configure MPI+OpenMP *layout* at runtime
- Propose the concept of *layout*
  - Static layout: explicit definition of ranks and threads placement at initialization time only
  - Dynamic layouts: change placement at runtime

Layout: Example



→

```
[MPI,-,Cores,[[0, -, -, -, -, -, -], [-, -, -, -, -, -, -],
                [-, -, -, -, -, -, -], [1, -, -, -, -, -, -],
                [-, -, -, -, -, -, -], [-, -, -, -, -, -, -]]
[OpenMP,MPI-0,HT,[[0,1,2,3],[4,5,6,7],[8,9,-,-],
    [10,11,-,-],[12,13,-,-],[14,15,-,-],[16,17,-,-],
    [18,19,-,-],[20,21,-,-],[22,23,-,-],[24,25,-,-],
    [26,27,-,-],[28,29,-,-],[30,31,-,-],[32,33,34,35],
    [36,37,38,39],[40,41,-,-],[42,43,-,-],[44,45,-,-],
    [46,47,-,-],[48,49,-,-]]
[OpenMP,MPI-1,HT, [[0,1,2,3],[4,5,6,7],[8,9,-,-],
    [10,11,-,-],[12,13,-,-],[14,15,-,-],[16,17,-,-],
    [18,19,21,22],[23,24,25,26],[27,28,-,-],
    [29,30,-,-],[31,32,-,-],[33,34,-,-],[35,36,-,-],
    [37,38,39,40],[41,42,43,44],[45,46,-,-],
    [47,48,-,-],[49,50,-,-],[51,52,-,-],[53,54,-,-]]
```

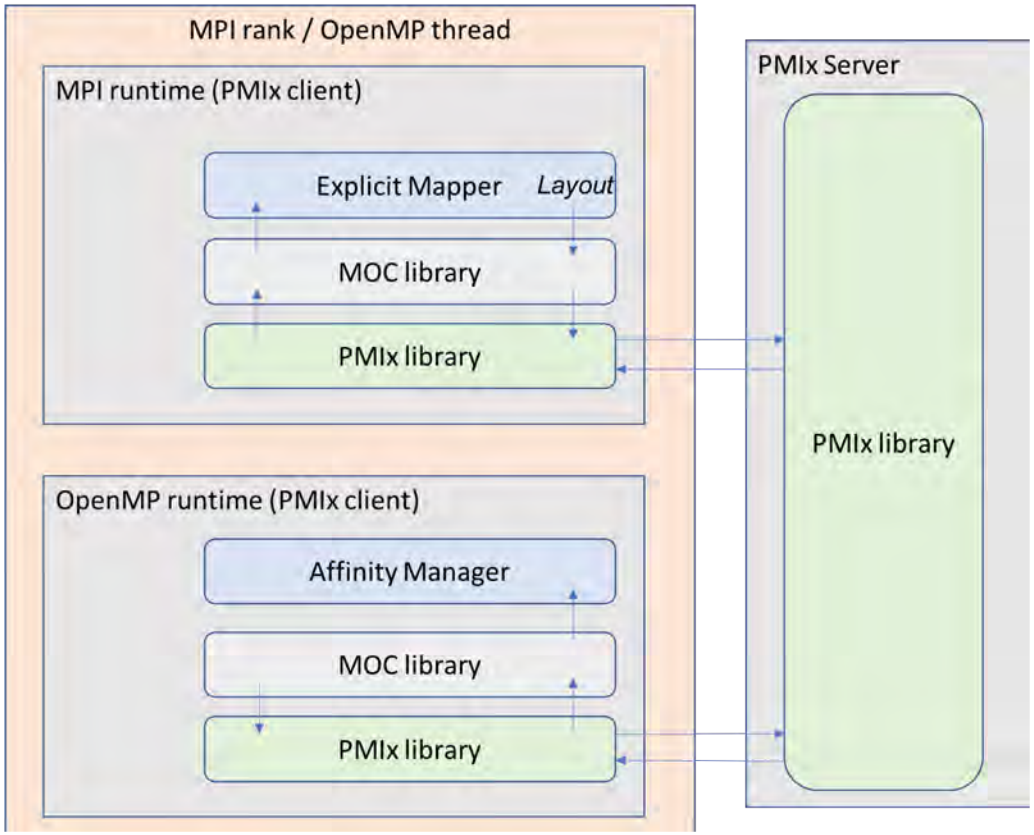
Static Layouts

- New MPI mapper
  - Get the layout specified by the user
  - Publish it through PMIx
- New MPI OpenMP Coordination (MOC) helper-library gets the layout and set OpenMP places to specify threads will be created placement
- No modification to OpenMP standard or runtime

Dynamic Layouts

- New API to define phases within the application
  - A static layout is deployed for each phase
  - Modification of the layout between phases (w/ MOC)
  - Well adapted to the nature of some applications
- Requires
  - OpenMP runtime modifications (LLVM prototype)
    - Get phases' layouts and set new places internally
  - OpenMP standard modifications to allow dynamicity

Architecture Overview



- MOC ensures inter-runtime data exchanged (MOC-specific PMIx keys)
- Runtimes are PMIx-aware
- Runtimes have a layout-aware mapper/affinity manager
- Respect the separation between standards, resource manager and runtimes

Inter-runtime Progress

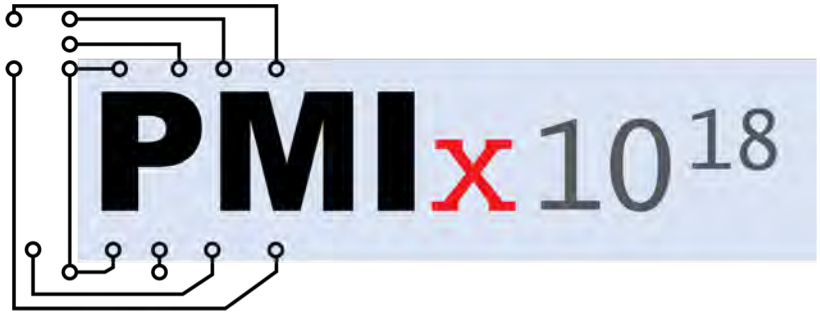
- Runtime usually guarantee internal progress but are not aware of other runtimes
- Runtimes can end up in a deadlock if mutually waiting for completion
  - MPI operation is ready to complete
  - Application calls and block in other runtimes

Inter-runtime Progress (2)

- Proposed solution
  - Runtimes use PMIx to *notify* of progress
  - When a runtime gets a progress notification, it yields once done with all tasks that can be completed
- Currently working on prototyping

Conclusion

- There is a real need for making runtimes aware of each other
- PMIx is a privileged option
  - Asynchronous / event based
  - Data exchange
  - Enable interaction with the resource manager
- Opens new opportunities for both research and development (e.g. new project focusing on QoS)



MPI Sessions: Dynamic Proc. Groups

Dan Holmes
EPCC University of Edinburgh

Howard Pritchard, Nathan Hjelm
Los Alamos National Laboratory

Los Alamos
NATIONAL LABORATORY
EST. 1943

Using PMIx to Help Replace MPI_Init

14th November 2018

Dan Holmes, Howard Pritchard, Nathan Hjelm

LA-UR-18-30830

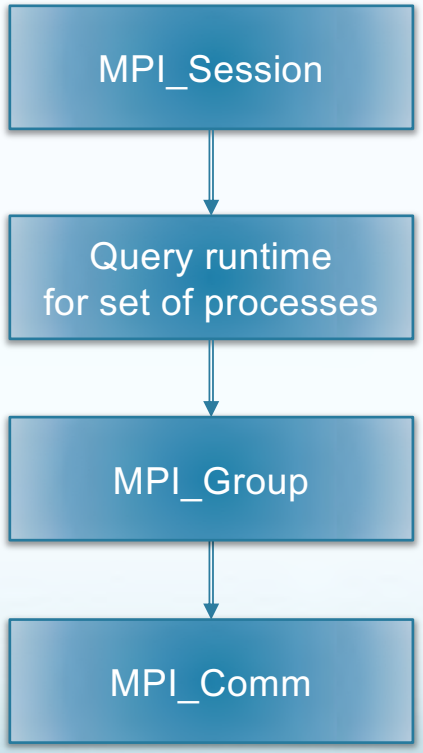
Problems with MPI_Init

- All MPI processes must initialize MPI exactly once
- MPI cannot be initialized within an MPI process from different application components without coordination
- MPI cannot be re-initialized after MPI is finalized
- Error handling for MPI initialization cannot be specified

Sessions – a new way to start MPI

- General scheme:
  - Query the underlying run-time system
    - Get a “set” of processes
  - Determine the processes you want
    - Create an MPI_Group
  - Create a communicator with just those processes
    - Create an MPI_Comm

Could be PMIx



MPI Sessions proposed API

- Create (or destroy) a session:
  - MPI_SESSION_INIT (and MPI_SESSION_FINALIZE)
- Get names of sets of processes:
  - MPI_SESSION_GET_NUM_PSETS, MPI_SESSION_GET_NTH_PSET
- Create an MPI_GROUP from a process set name:
  - MPI_GROUP_CREATE_FROM_SESSION
- Create an MPI_COMM from an MPI_GROUP:
  - MPI_COMM_CREATE_FROM_GROUP

PMIx groups helps here

MPI_COMM_CREATE_FROM_GROUP

```
MPI_Create_comm_from_group(IN MPI_Group group,
                           IN const char *uri,
                           IN MPI_Info info,
                           IN MPI_Erhandler hndl,
                           OUT MPI_Comm *comm);
```

The ‘uri’ is supplied by the application.
Implementation challenge: ‘group’ is a local object.
Need some way to synchronize with other “joiners” to the communicator. The ‘uri’ is different than a process set name.

Using PMIx Groups

- PMIx Groups - a collection of processes desiring a unified identifier for purposes such as passing events or participating in PMIx fence operations
  - Invite/join/leave semantics
- Sessions prototype implementation currently uses PMIX_Group_construct/PMIX_Group_destruct
- Can be used to generate a “unique” 64-bit identifier for the group. Used by the sessions prototype to generate a communicator ID.
- Useful options for future work
  - Timeout for processes joining the group
  - Asynchronous notification when a process leaves the group

Using PMIx_Group_Construct

```
PMIx_Group_Construct(const char id[],
                     const pmix_proc_t procs[],
                     const pmix_info_t info[],
                     size_t ninfo);
```

- 'id' maps to/from the 'uri' in MPI_Comm_create_from_group (plus additional Open MPI internal info)
- 'procs' array comes from information previously supplied by PMIx
  - "mpi://world" and "mpi://self" already available
  - mpiexec -np 2 --pset user://ocean ocean.x : \
    -np 2 --pset user://atmosphere atmosphere.x

Work in progress

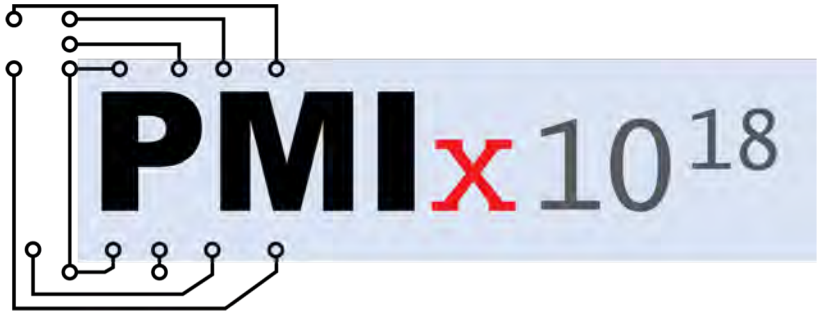
MPI Sessions Prototype Status

- All “MPI_*_from_group” functions have been implemented
  - Only pml/ob1 supported at this time
- Working on MPI_Group creation (from PSET) now
  - Will start with mpi://world and mpi://self
  - User-defined process sets need additional support from PMIx
- Up next: break apart MPI initialization
  - Goal is to reduce startup time and memory footprint

Summary

- PMIx Groups provides an OOB mechanism for MPI processes to bootstrap the formation of a communication context (MPI Communicator) from a group of MPI processes
- Functionality for future work
  - Handling (unexpected) process exit
  - User-defined process sets
  - Group expansion

Funding Acknowledgments



Q&A

Useful Links:

General Information: https://pmix.org/

PMIx Library: https://github.com/pmix/pmix

PMIx Reference RunTime Environment (PRRTE): https://github.com/pmix/prrte

PMIx Standard: https://github.com/pmix/pmix-standard

Slack: pmix-workspace.slack.com