PMI-1, PMI-2

Ken Raffenetti

Argonne National Laboratory

PMI API

- Common API Functions
  - Initialization and finalization
    - Init, Finalize, Abort
  - Information exchange
    - Put, Get, Fence (aka Barrier)
  - Process creation
    - Spawn

INFORMATION EXCHANGE

- Processes need to exchange connection info
- PMI uses a Key-Value database (KVS)
- At init, processes *Put* contact information
  - E.g., IP address and port
- Processes *Get* contact info when establishing connections
- Collective *Fence* operation to allow optimizations
- Some networks (BG/Q) do not require exchange to communicate

PMI-1 EXTRAS

- Pre-populated keys
  - PMI Process Mapping
    - Processes able to calculate “nodemap” locally

PMI-2 FEATURES

- Attribute query functionality
- Database scope
- Thread safety
- Dynamic processes
- Fault tolerance

PMI-2 ATTRIBUTE QUERY FUNCTIONALITY

- Process and resource managers have system-specific information
  - Node topology, network topology, etc.
- Without this, processes need to determine this themselves
  - Each process gets each other's contact-info to discover local processes
  - $O(p^2)$ queries

PMI-2 DATABASE SCOPE

- Previously KVS had only global scope
- PMI-2 adds node-level scoping
  - E.g., keys for shared memory segments
- Allows for optimized storage and retrieval of values

PMI-2 THREAD SAFETY

- PMI-1 is not thread safe
  - All PMI calls must be serialized
    - Wait for request and response
  - Can affect multithreaded programs
- PMI-2 adds thread safety
  - Multiple threads can call PMI functions
  - One call cannot block the completion of another

PMI-2 DYNAMIC PROCESSES

- In PMI-1 a separate database is maintained for each MPI_COMM_WORLD (process group)
  - Queries are not allowed across databases
  - Requires out-of-band exchange of databases
- PMI-2 allows cross-database queries
  - Spawned or connected process groups can now query each other's databases
  - Only process group ids need to be exchanged

PMI-2 FAULT TOLERANCE

- PMI-1 provides no mechanism for respawning a failed process
  - New processes can be spawned, but they have a unique rank and process group
- Respawn is critical for supporting fault-tolerance
  - Not just for MPI but other programming models

PMI-3 (DEAD)

- `int PMI_Init(int required, int *provided, int *max_keylen, int *max_vallen);`
- `int PMI_Initialized(int *initialized);`
- `int PMI_Finalize(void);`
- `int PMI_Finalized(int *finalized);`
- `int PMI_Get_attr(int attr, int *value);`
- `int PMI_Put(const char key[], const char value[], int scope);`
- `int PMI_Get(const char key[], char value[], int scope, int jobid);`
- `int PMI_Fence(void);`
- `int PMI_Abort(int flag, const char msg[]);`

BACKUP SLIDES

OAKFOREST-PACS

- Intel KNL 7250 (68 cores)
- 8,208 compute nodes
- Intel Omnipath Architecture

- Hello, World!
  - 512 nodes, 64 ppn took **over an hour!**

- To bring down launch times, we looked at
  - Hydra process manager (mpiexec)
  - Usage of the Process Management Interface (PMI)

- What is not covered in this talk
  - Initialization of the fabric (fi_av_insert)

KEY-VALUE SPACE OPTIMIZATIONS

- Existing optimization
  - PMI_KVS_Barrier caches key-value pairs at the node level
  - PMI_KVS_Get is a node-local operation

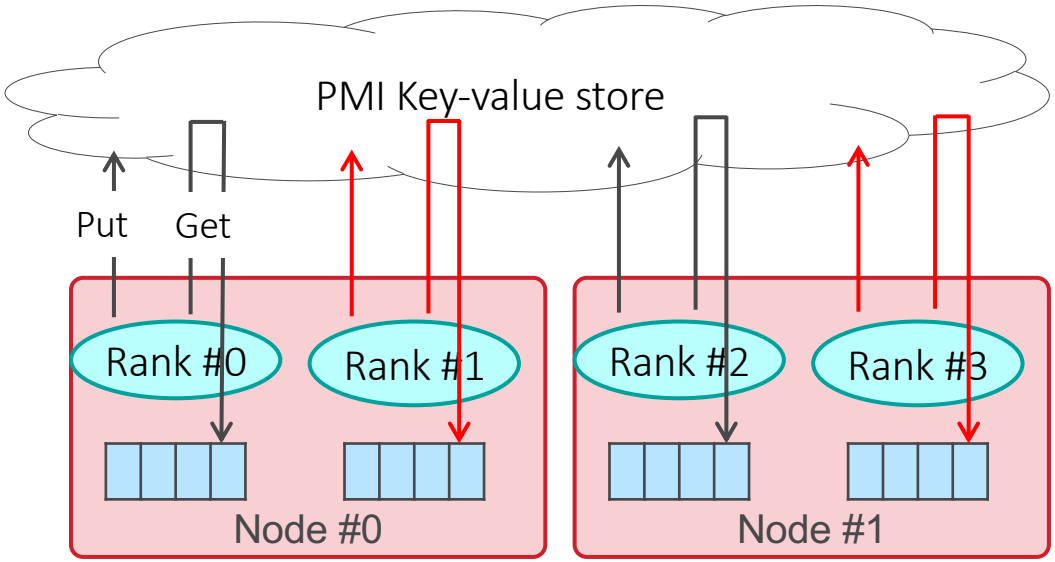
- New optimizations
  - Replaced linked-list implementation with a more scalable hash
    - Constant lookup time for PMI_KVS_Get
  - Eliminated checks for duplicate keys
    - Erroneous usage
    - Can be re-enabled for debugging

ORIGINAL CODE

Code

```
/* All ranks performs followings*/
PMI_KVS_Put(rank, myaddr);
PMI_KVS_Commit();
PMI_KVS_Barrier();
for (i = 0; i < size; i++)
   PMI_KVS_Get(i, &addrs[i]);
```

Data Flow



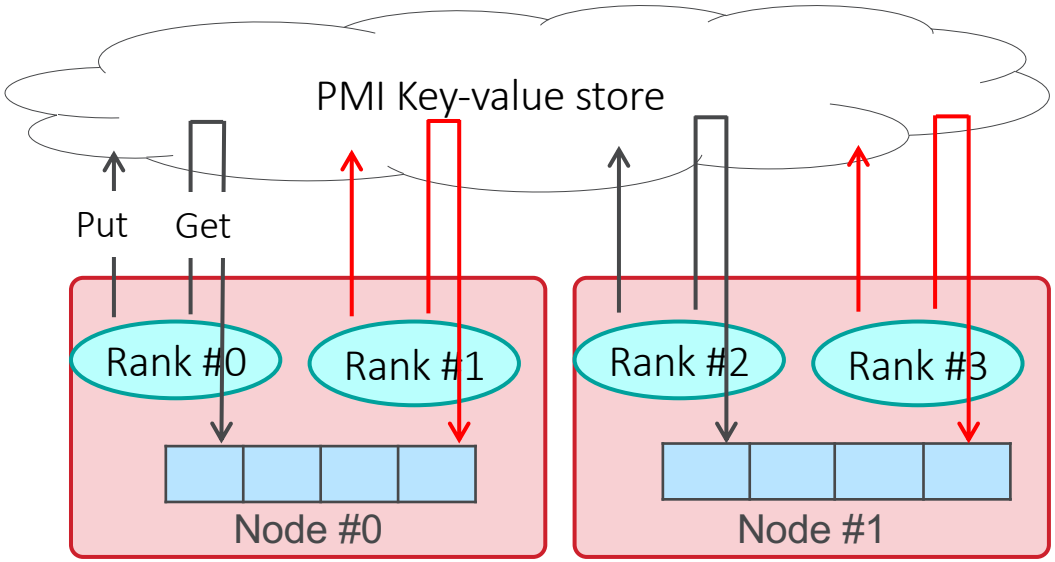
Too many get operations, many of which are redundant

SHM OPTIMIZATION

Code

```
addrs[rank] = myaddr; /* address table in shared memory */
/* All ranks performs followings*/
PMI_KVS_Put(rank, myaddr);
PMI_KVS_Commit();
PMI_KVS_Barrier();
int num_cards = (size / local_size);
for (i = local_rank * num_cards; i < (local_rank + 1) * num_cards; i++)
   PMI_KVS_Get(i, &addrs[i]);
```

Data Flow

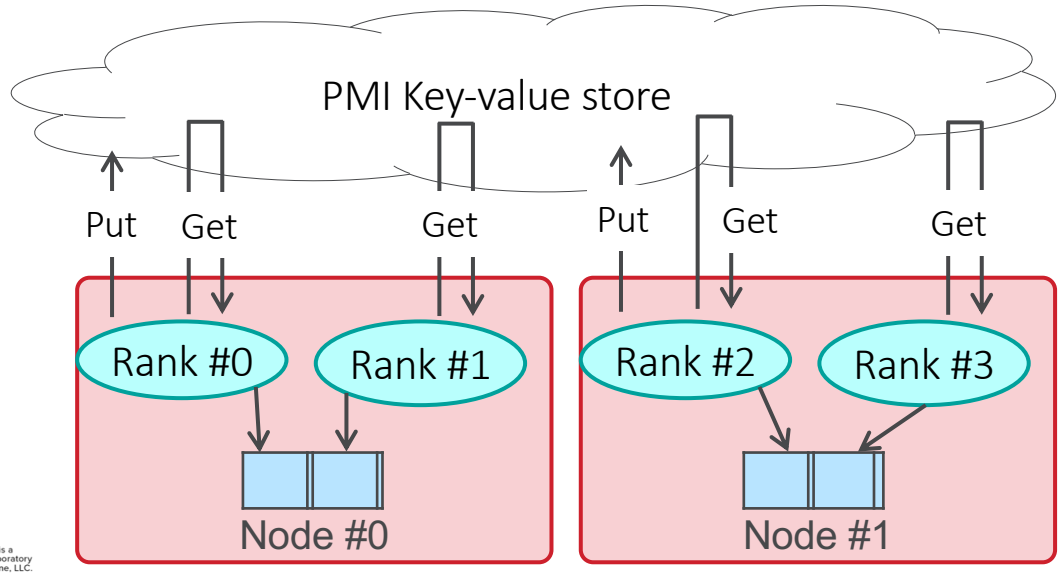


NODE ROOTS OPTIMIZATION

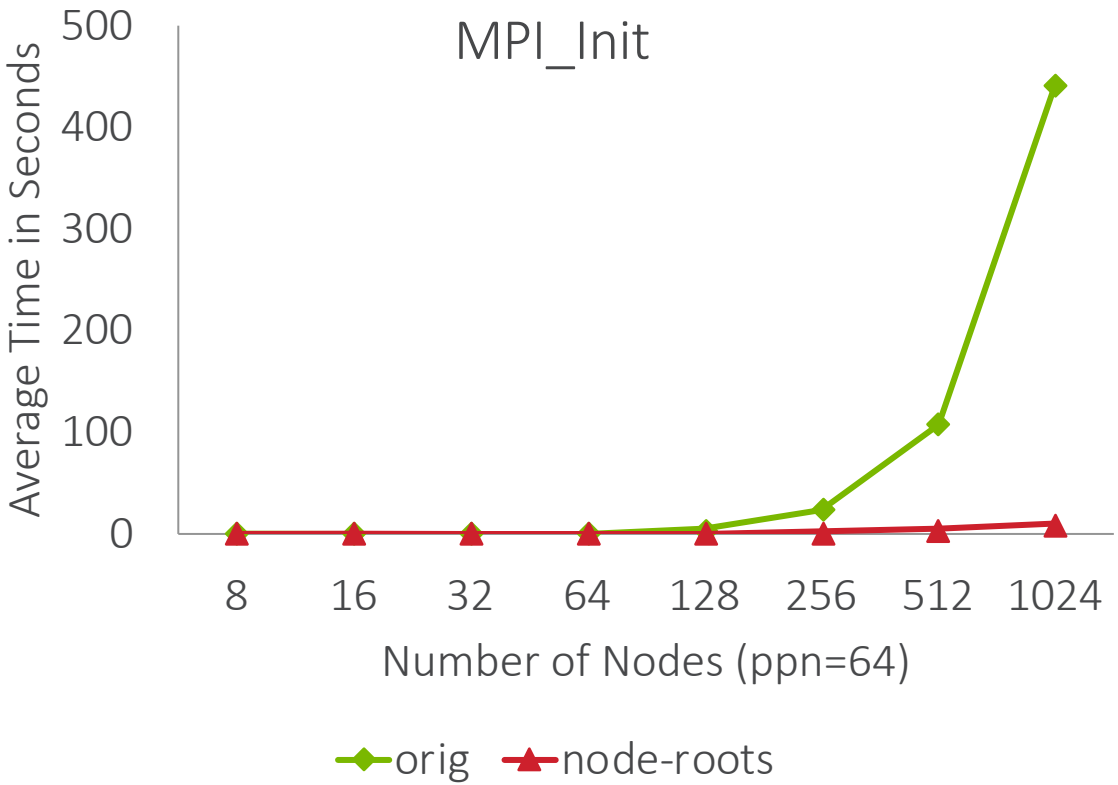
Code

```
addrs[rank] = myaddr;
if (node_root) {
   PMI_KVS_Put(rank, addr[i]);
   PMI_KVS_Commit();
}
PMI_KVS_Barrier();
int num_cards = num_roots / local_size;
for (i = local_rank * num_cards; i < (local_rank + 1) * num_cards; i++)
   PMI_KVS_Get(i, &addrs[i]);
MPIR_Allgather(..., addrs, node_roots_comm);
```

Data Flow



EVALUATION



- Measured on Intel Xeon Phi 7230 (Theta@ANL)
- Node-root algorithm can reduce MPI_Init time from 442 seconds to 10 seconds

PMI-3 (DEAD)

- `int PMI_Init(int *max_keylen, int *max_vallen);`
- `int PMI_Get_attr(int attr, int *value);`
- `int PMI_Put(const char key[], const char value[], int scope);`
- `int PMI_Get(const char key[], char value[], int scope, int jobid);`
- `int PMI_Fence(void);`
- `int PMI_Abort(int flag, const char msg[]);`