NETWORK-BASED COMPUTING LABORATORY

Impact of Node Level Caching in MPI Job Launch Mechanisms

Jaidev Sridhar and D. K. Panda

{sridharj,panda}@cse.ohio-state.edu

Presented by Pavan Balaji, Argonne National Laboratory

Network-Based Computing Lab

The Ohio State University

Columbus, OH USA

OHIO STATE

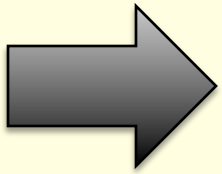
Presentation Outline

- **Introduction and Motivation**
- ScELA Design
- Impact of Node-Level Caching
- Experimental Evaluation
- Conclusions and Future Work

Introduction

- HPC Clusters continue to increase rapidly in size
  - Largest systems have hundreds of thousands of cores today
- As clusters grow, there has been increased focus on the scalability of programming models and libraries
  - MPI, PGAS models
  - “First class citizens”
- Job launch mechanisms have not received enough attention and have scaled poorly over the last few years
  - Traditionally ignored since the “percentage of time” for launching jobs on production runs is small
  - But increasingly becoming important, especially for extremely large-scale systems

Multi-Core Trend

Largest InfiniBand cluster in **2006**		Largest general purpose InfiniBand cluster in **2008**
Sandia Thunderbird		TACC Ranger
– 8,960 processing cores		– 62,976 processing cores
– **4,480** compute nodes		– **3,936** compute nodes

Courtesy Intel Corp.

- The total number of compute cores has increased by a factor of 7, however, the number of compute nodes has remained flat
- Job launchers must take advantage of multi-core compute nodes

Limitations

- MPI Job launch mechanisms scale poorly over large multi-core clusters
  - Over 3 minutes to launch a MPI job over 10,000 cores (in the early part of 2008)
  - Unable to launch larger jobs
    - Exponential increase in job launch time
- These designs run into system limitations
  - Limits on the number of open network connections
  - Delays due to simultaneous flooding of network

Job Launch Phases

- Typical parallel job launch involves two phases
  - Spawning processes on target cores
  - Communication between processes to discover peers
- In addition to spawning processes, job launcher must facilitate communication for job initialization
  - Point to point
  - Collective communication

Presentation Outline

- Introduction and Motivation
- **ScELA Design**
- Impact of Node-Level Caching
- Experimental Evaluation
- Conclusions and Future Work

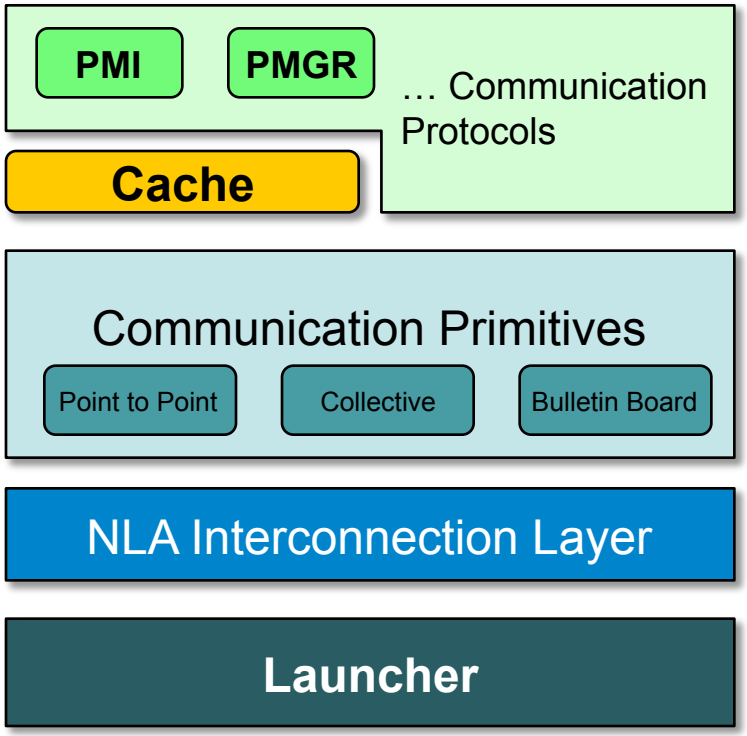
ScELA Design

- Designed a Scalable, Extensible Launching Architecture (ScELA) that takes advantage of increased use of multi-core compute nodes in clusters
  - Presented at Int'l Symposium on High Performance Computing (HiPC '08)
- Supported both PMGR_Collectives and PMI
- The design was incorporated into MVAPICH 1.0 and MVAPICH2 1.2
  - MVAPICH/MVAPICH2 - Popular MPI libraries for InfiniBand and 10GigE/iWARP, used by over 975 organizations worldwide (http://mvapich.cse.ohio-state.edu)
  - Significant performance benefits on large-scale clusters
- Many other MPI stacks have adopted this design for their job launching mechanisms

Design:

ScELA Architecture

- Hierarchical launch
  - Central launcher launches Node Launch Agents (NLA) on target nodes
  - NLAs launch processes on cores
- NLAs interconnect to form a k-ary tree to facilitate communication
- Common communication primitives built on NLA tree
- Libraries can implement their protocols (PMI, PMGR, etc.) over the basic framework

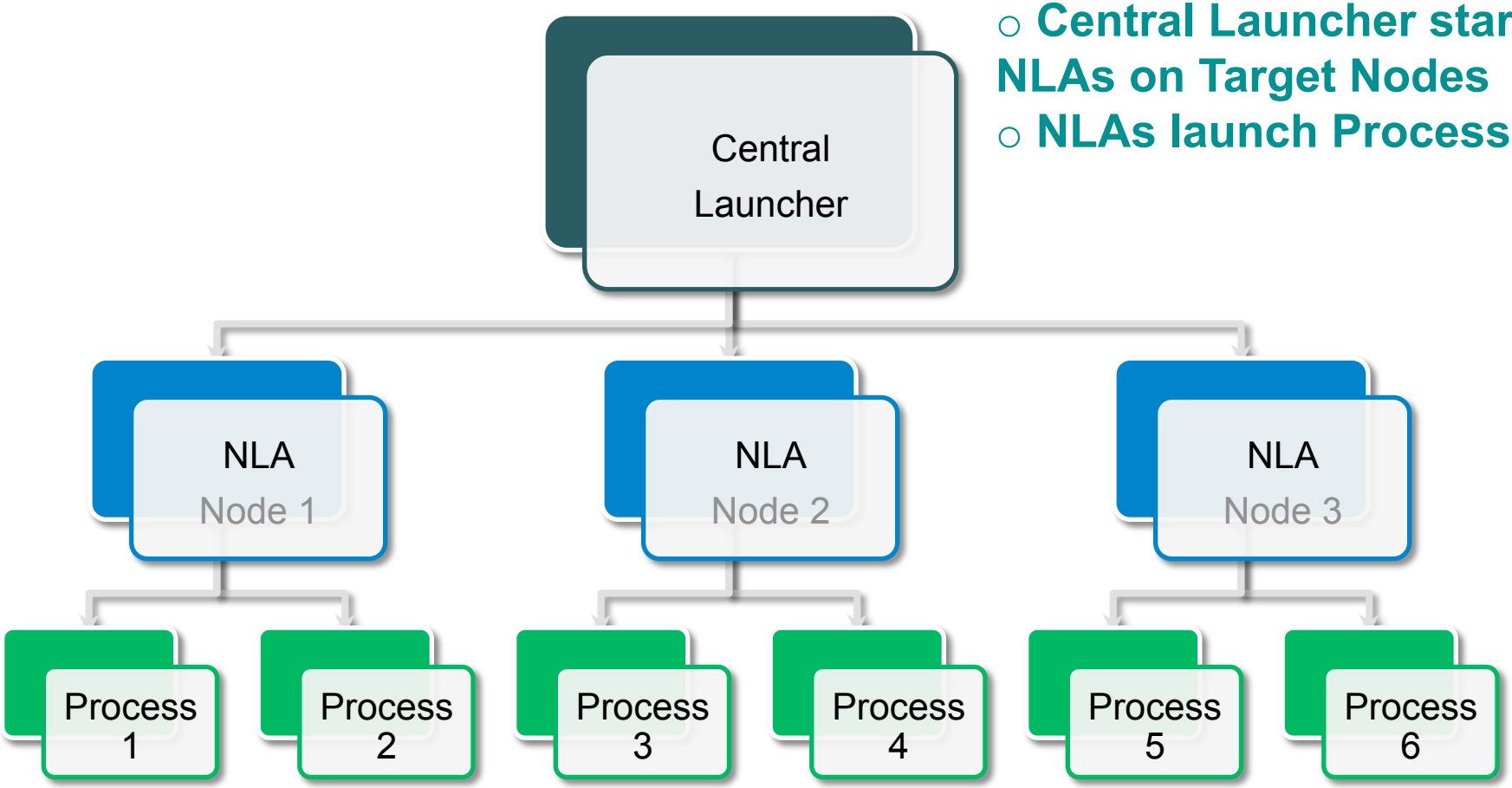


ScELA Architecture

Design:

Launch Mechanism

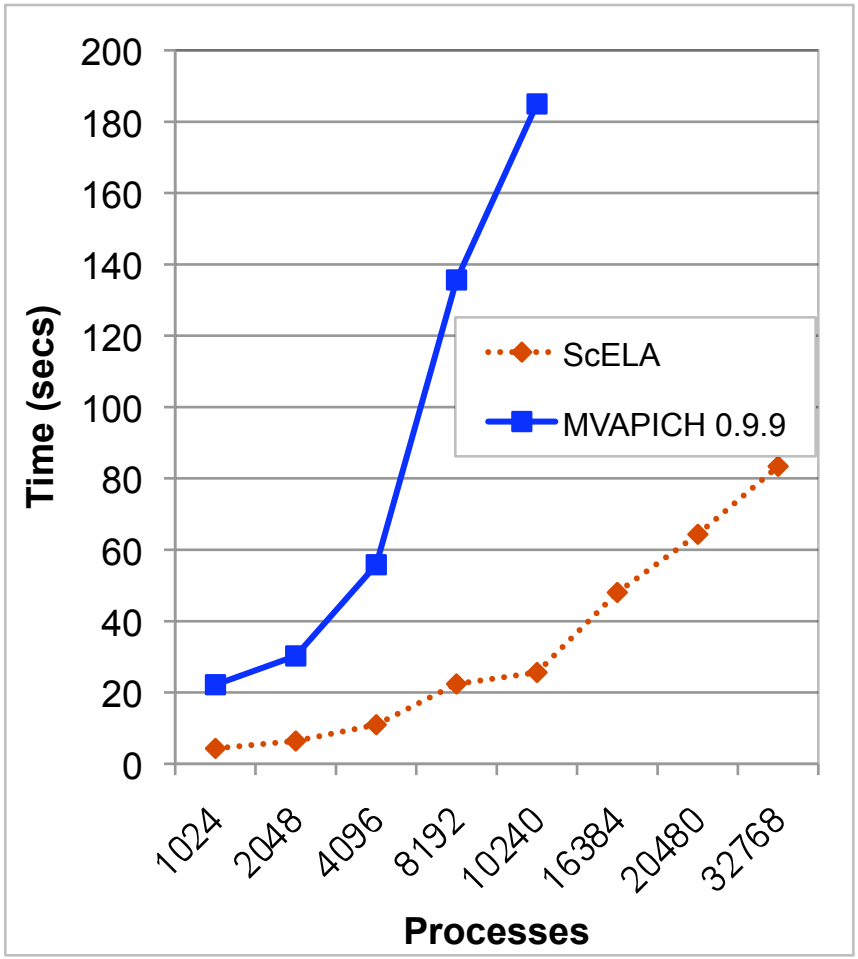
- Central Launcher starts NLAs on Target Nodes
- NLAs launch Processes



Evaluation:

Large Scale Cluster

- ScELA compared MVAPICH 0.9.9 on the TACC Ranger
  - 3,936 nodes with four 2.0 GHz Quad-Core AMD “Barcelona” Opteron processors
  - 16 processing cores per node
- Time to launch a simple MPI “Hello World” program
- Can scale at least 3X
- Order of magnitude faster



Presentation Outline

- Introduction and Motivation
- ScELA Design
- **Impact of Node-Level Caching**
- Experimental Evaluation
- Conclusions and Future Work

PMI Bulletin Board on ScELA

- PMI is a startup communication protocol used by MVAPICH2, MPICH2, etc.
- For process discovery, PMI defines a bulletin board protocol
  - PMI_Put (key, val) publishes a key, value pair
  - PMI_Get (key) fetches appropriate value
- We define similar operations NLA_Put and NLA_Get to facilitate a bulletin board over the NLA tree
- NLA level caches to speedup information access

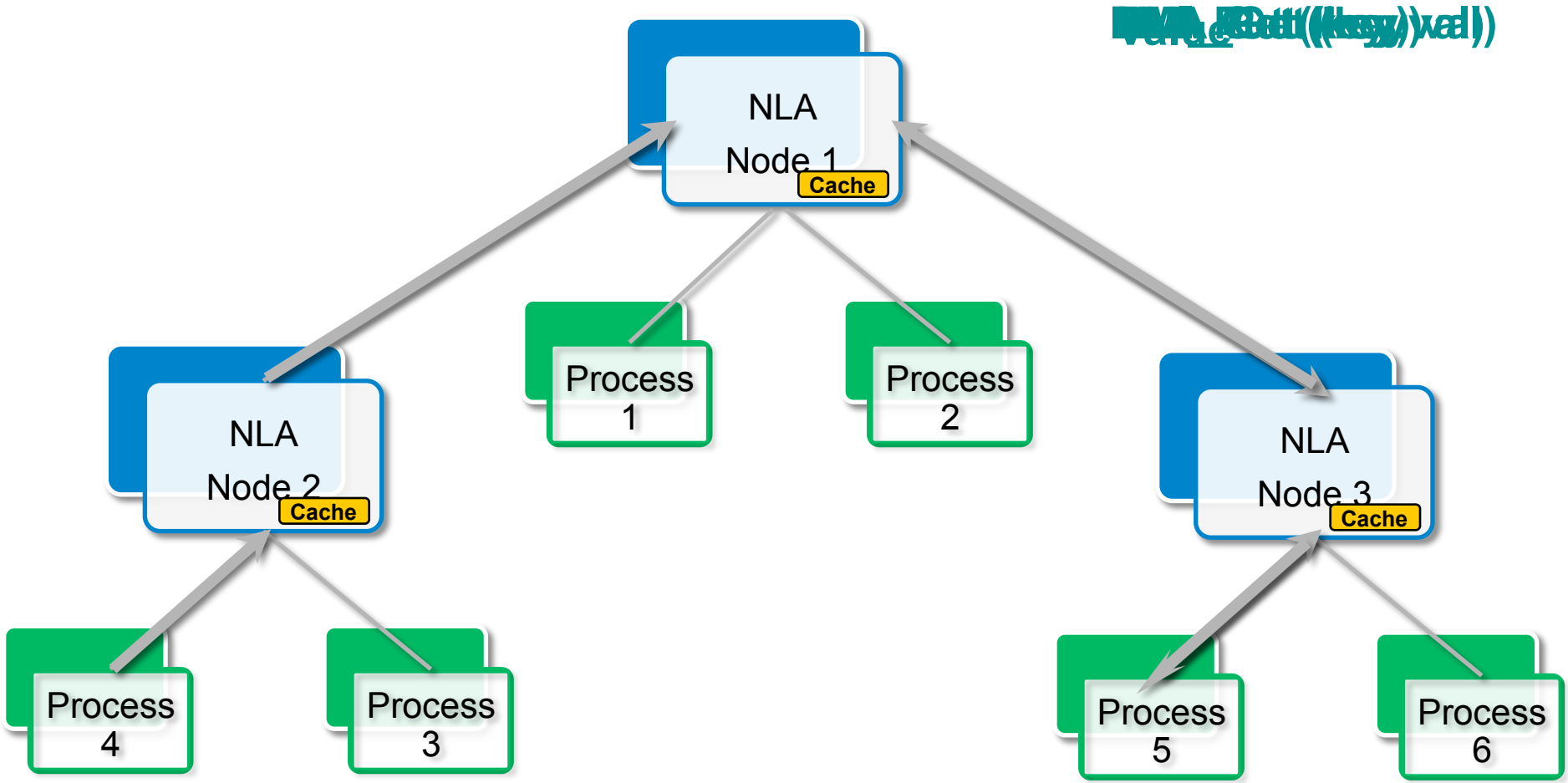
Focus in this Paper

- Is it beneficial to cache information in intermediate nodes in the NLA tree?
- How these caches need to be designed?
- What trade-offs exist in designing such caches?
- How much performance benefits can be achieved with such caching?

Four Design Alternatives for Caching

- Hierarchical Cache Simple (HCS)
- Hierarchical Cache with Message Aggregation (HCMA)
- Hierarchical Cache with Message Aggregation and Broadcast (HCMAB)
- Hierarchical Cache with Message Aggregation, Broadcast with LRU (HCMAB-LRU)

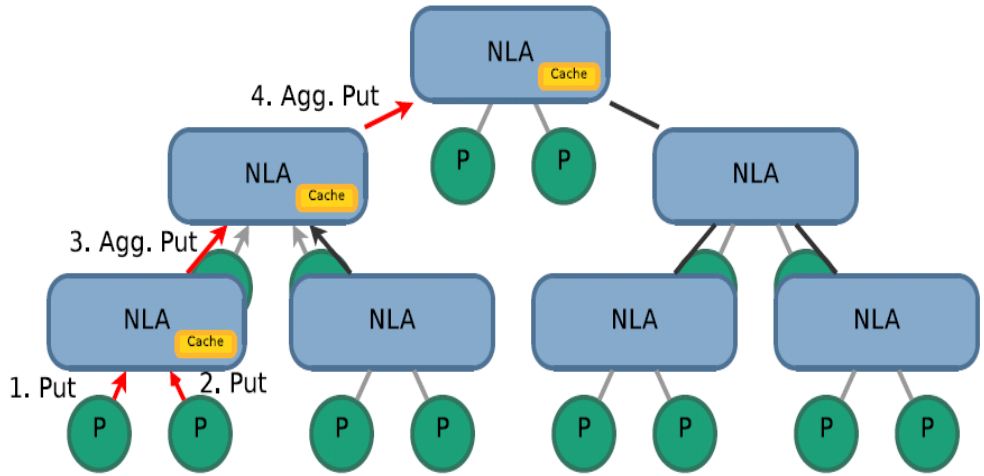
PMI Bulletin Board on ScELA with HCS



Better Caching Mechanisms

- We've seen a simple Hierarchical Cache (HCS)
  - Slow, due to number of messages
- Reduce number of messages with message aggregation – HCMA

```
PMI_Put (mykey, myvalue);
PMI_Barrier ();
...
val1 = PMI_Get (key1);
val2 = PMI_Get (key2);
...
```



Caching Mechanisms (contd)

- HCMA still has lots of messages over network during GETs
- Propose HCMAB
  - HCMA + Broadcast
- HCS, HCMA, HCMAB are memory inefficient
  - Information exchange is in stages – discard old information
- Propose HCMAB-LRU
  - Have a fixed size cache with LRU
  - HCMAB-LRU

Comparison of Memory usage

- For n (key, value) pairs exchanged by p processes

MPI Job Size (p)	Caching Mechanism			
	HCS	HCMA	HCMAB	HCMAB-LRU
64	$O(64 \times n)$	$O(64 \times n)$	$O(64 \times n)$	$O(64)$
256	$O(256 \times n)$	$O(256 \times n)$	$O(256 \times n)$	$O(256)$
1024	$O(1024 \times n)$	$O(1024 \times n)$	$O(1024 \times n)$	$O(1024)$
4096	$O(4096 \times n)$	$O(4096 \times n)$	$O(4096 \times n)$	$O(4096)$
16384	$O(16384 \times n)$	$O(16384 \times n)$	$O(16384 \times n)$	$O(16384)$
65536	$O(65536 \times n)$	$O(65536 \times n)$	$O(65536 \times n)$	$O(65536)$

Presentation Outline

- Introduction and Motivation
- ScELA Design
- Impact of Node-Level Caching
- **Experimental Evaluation**
- Conclusions and Future Work

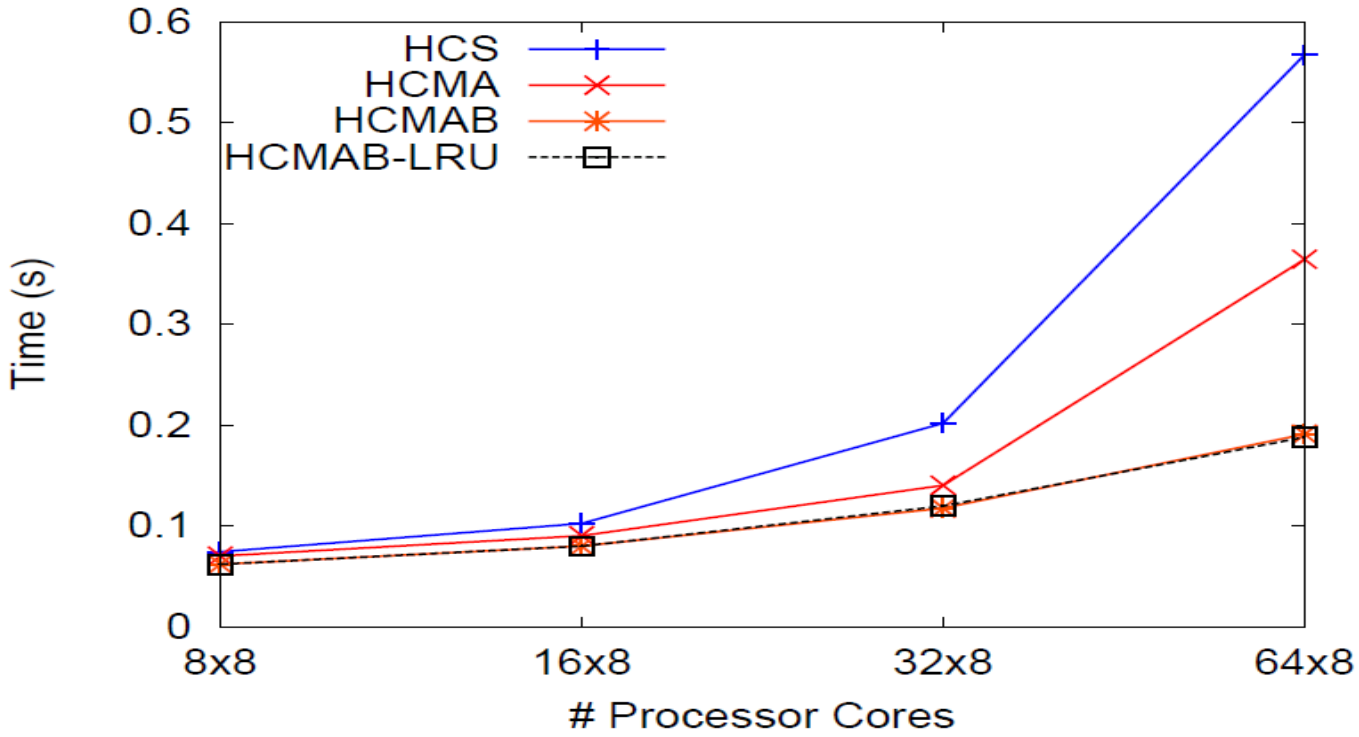
Evaluation:

Experimental Setup

- OSU Cluster
  - 512-core InfiniBand Cluster
  - 64 compute nodes
  - Dual 2.33 GHz Quad-Core Intel “Clovertown”
  - Gigabit Ethernet adapter for management traffic
- TACC Ranger (62,976-cores)
  - InfiniBand connectivity

Simple PMI Exchange (1:2)

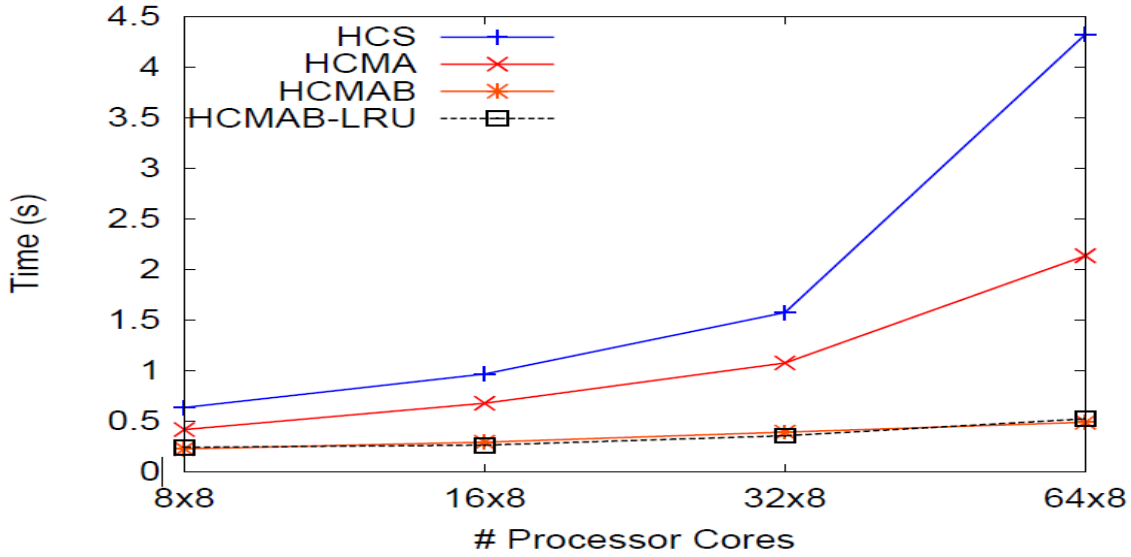
- Each MPI process publishes one (key, value) pair using PMI_Put
- Retrieves values published by two other MPI processes



- HCMAB and HCMAB-LRU are the best

Heavy PMI Exchange (1:p)

- Each MPI process publishes one (key, value) pair using PMI_Put
- All p processes read values published by all other p processes



- HCMAB and HCMAB-LRU are the best with significant performance improvement
- HCMAB and HCMAB-LRU demonstrate good scalability with increase in system size

Software Distribution

- Both HCS and HCMAB have been integrated into MVAPICH2 1.2 and available to the MPI community for some time
- Additional enhancements in terms of parallelizing the startup further have been carried out in MVAPICH2 1.4

Presentation Outline

- Introduction and Motivation
- ScELA Design
- Impact of Node-Level Caching
- Experimental Evaluation
- **Conclusions and Future Work**

Conclusion and Future Work

- Propose the impact of caching in scalable, hierarchical job launch mechanisms, especially for emerging multi-core clusters
- Demonstrate design alternatives and their impact on performance and scalability
- Integrated into the latest MVAPICH2 1.4 version
  - Basic enhancements are available in MVAPICH versions (1.0 and 1.1)
- Parallelize the job launch phase even further for even larger clusters with a million of processes

Questions?

{sridharj, panda}@cse.ohio-state.edu

http://mvapich.cse.ohio-state.edu