MIC-Check: A Distributed Checkpointing Framework for the Intel Many Integrated Cores Architecture

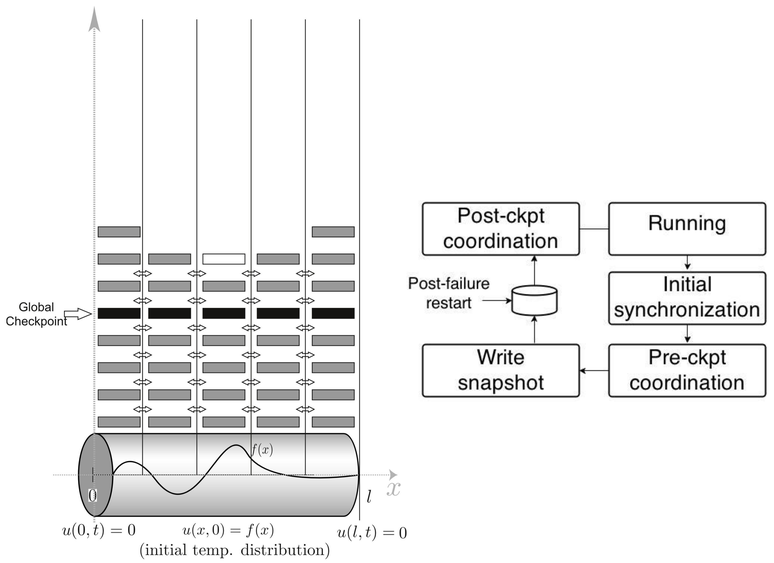
The Ohio State University
MVAPICH MPI, PGAS and Hybrid MPI+PGAS Library

Raghunath Rajachandrasekar, Sreeram Potluri, Akshay Venkatesh, Khaled Hamidouche and Dhabaleswar K. Panda

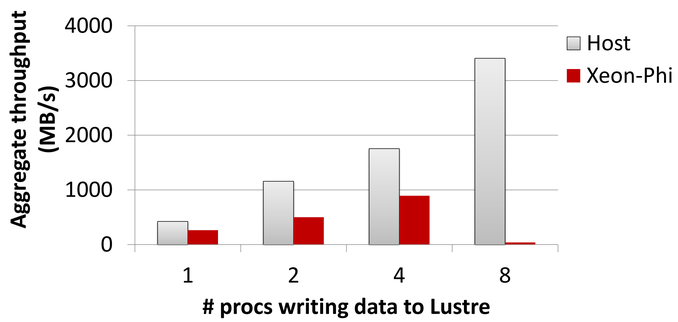
Abstract

Naïve checkpointing protocols, which are predominantly I/O-intensive, face severe performance bottlenecks on the Xeon Phi architecture due to several inherent limitations. This work explores these limitations, and proposes the architecture and design of a novel distributed checkpointing framework, namely MIC-Check, for HPC applications running on it.

Checkpointing in HPC



Disparity in Xeon Phi I/O Performance

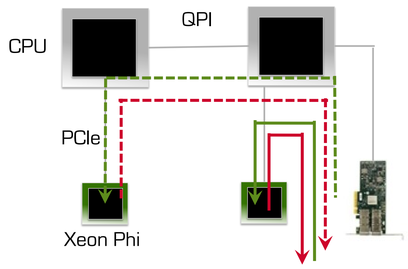


- IOZone benchmark run on the 1 node of Stampede@TACC
- Aggregate throughput as seen by host peaks at **3.4GB/s**
- Peak throughput as seen by Xeon Phi coprocessor: **893MB/s**
- Contention hurts throughput with just 8 MIC processes (**41MB/s**)

Factors Limiting I/O Performance on MICs

- Low-frequency processing units with reduced cache sizes
- VFS page-cache management overheads when per-CPU pool is depleted
  - Kernel page allocator invoked to request free page
  - Zone and LRU locking
  - Identifying pages that can be used to replenish per-CPU pool
- User-space ⇔ kernel-space data movement (copy_from/to_user routines)
  - Do not leverage vector-processing capabilities
- Page locking to maintain consistency
- 4-way multithreaded processing cores => round-robin arbitration (CPI=4)
- Not capable of branch-prediction, speculative or out-of-order execution

Peak bandwidth of various communication channels on Xeon Phi

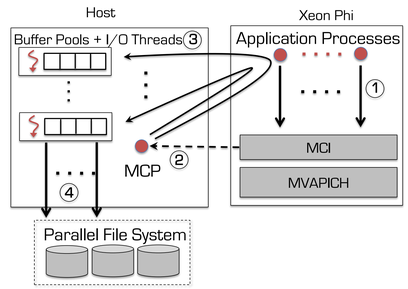


		Sandy Bridge	Ivy Bridge
Same Socket	Read from MIC	962 MB/s (15%)	3421 MB/s (54%)
	Write to MIC	5280 MB/s (83%)	6396 MB/s (100%)
Different Socket	Read from MIC	370 MB/s (6%)	247 MB/s (4%)
	Write to MIC	1075 MB/s (17%)	1179 MB/s (19%)

Peak IB FDR Bandwidth: **6397 MB/s**

Proposed Architecture and Design

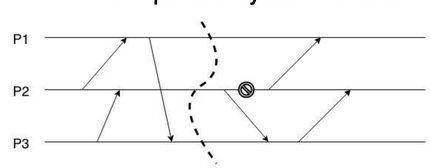
Application-level checkpointing for Native and Symmetric mode of execution



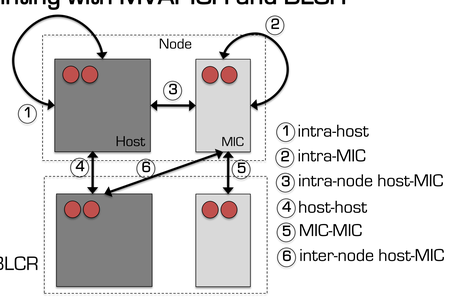
MCI = MIC-Check Interception Library
MCP = MIC-Check Proxy

1. MCI takes control of application I/O; initiates SCIF connection with MCP on host during *MPI_Init()*
2. During a checkpoint, MCI intercepts *open()* to register SCIF resources at host; intercepts *write()* to send control to host indicating checkpoint is ready
3. MCP spawns new thread to progress I/O on behalf of each MPI process connecting to it; pulls checkpoint data from MIC using SCIF RMA protocol
4. Checkpoint written to underlying parallel file system in a pipelined manner, as and when data is available from the SCIF RMA transfers

Transparent System-Level Checkpointing with MVAPICH and BLCR

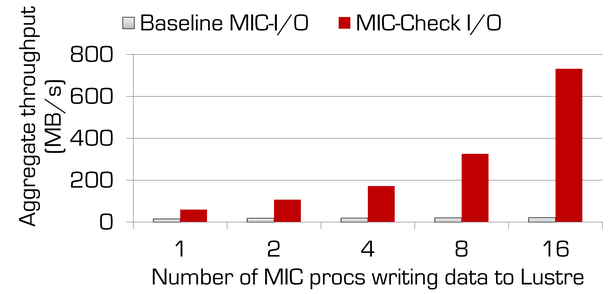


1. Drain in-flight messages and suspend network activity
2. Tear down communication channels (1)–(6)
3. Obtain snapshot of host and MIC MPI processes using BLCR
4. Re-establish communication channels (1)–(6)

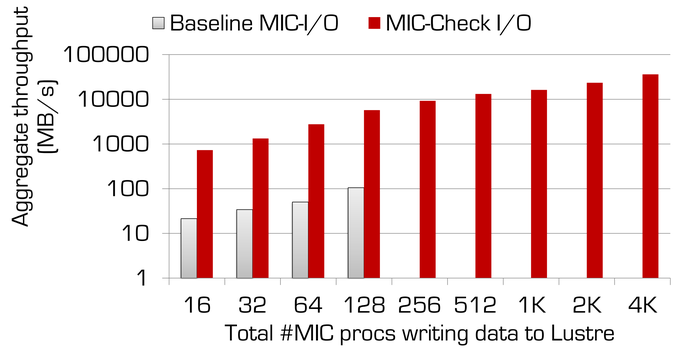


Performance Evaluation

Intra-node Scalability tests on 1 node of the Stampede supercomputer (@TACC)



Inter-node Scalability tests* on the Stampede supercomputer (@TACC)



*TACC staff requested us to limit our baseline runs to 128 processes, owing to failures caused by Lustre contention

Application-level evaluation with ENZO checkpoints

	Compute Time (s)	Checkpoint Time (s)
Baseline	91.2	44.8
MIC-Check	93.1	1.49

- Native mode of execution
- 128 MPI processes running on the TACC system
- 5.37GB of aggregate checkpoints
- 30x reduction in checkpointing time observed

Summary and Future Work

- Outlined and analyzed the inherent I/O limitations on MICs
- Proposed a novel checkpointing system that overcomes these limitations
- MIC-Check provides **35x improvement** in aggregate I/O throughput with 16 processes running on a single MIC; **54x improvement with 4,096 process** running on 256 MICs
- Adapter-based coprocessors are expected to be the mainstay – we will study the impact of MIC-Check on future architectures
- Extend MIC-Check to transparently checkpoint "offloaded" applications

This research is supported in part by NSF grants OCI-1148371 and CCF-1213084.