Native Mode-Based Optimizations of Remote Memory Accesses in OpenSHMEM for Intel Xeon Phi

Naveen Namashivayam, Sayan Ghosh, Dounia Khaldi, Deepak Eachempati and Barbara Chapman

Department of Computer Science

University of Houston

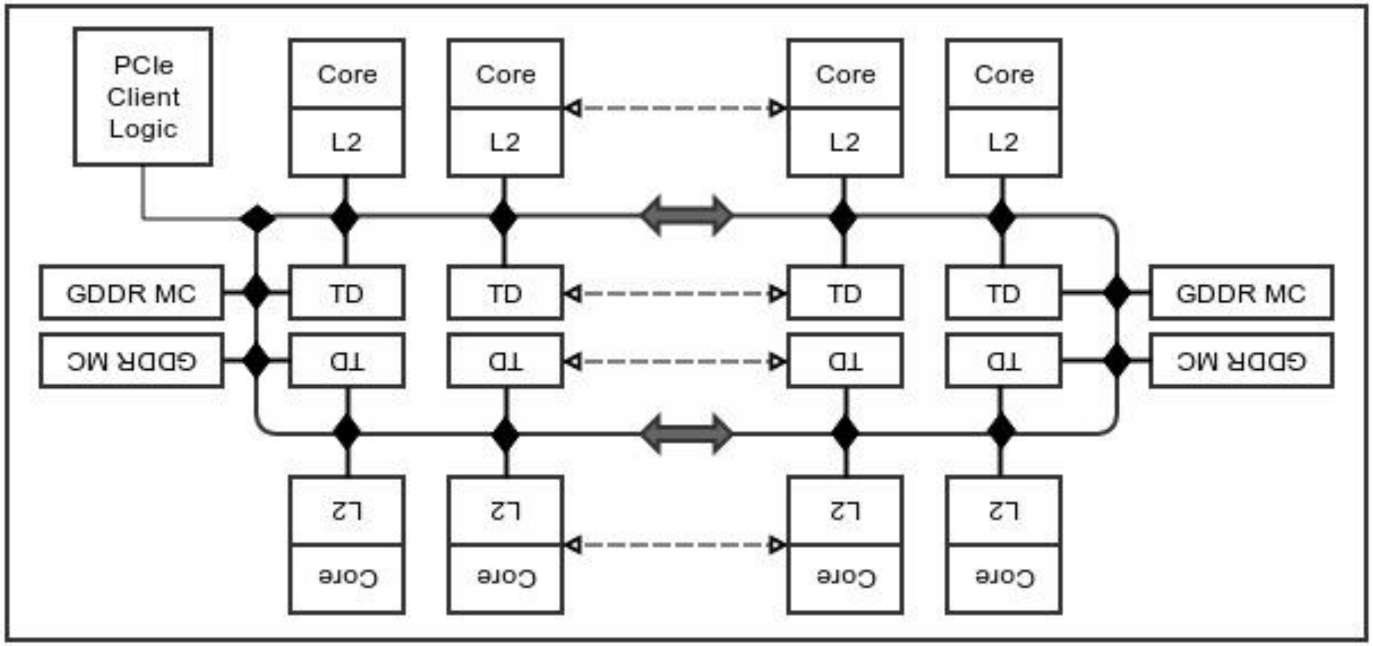
Motivation

- PGAS
  - An emerging programming model to be explored
  - OpenSHMEM on MIC in Native Mode
- Intel Xeon Phi: Many Integrated Core (MIC)
- Knight’s Corner (KNC)
  - Current generation
  - Modified x86 co-processor
  - Unlike GPGPUs, porting applications is straightforward
- Knight’s Landing (KNL)
  - Future Generation
  - Main CPU

Contents

- **Xeon Phi Architecture**
- Why OpenSHMEM?
- RMA put/get vs. Local load/store
- Optimizing Collectives: Reduction as a use case
- Experimental Results: NAS parallel benchmarks

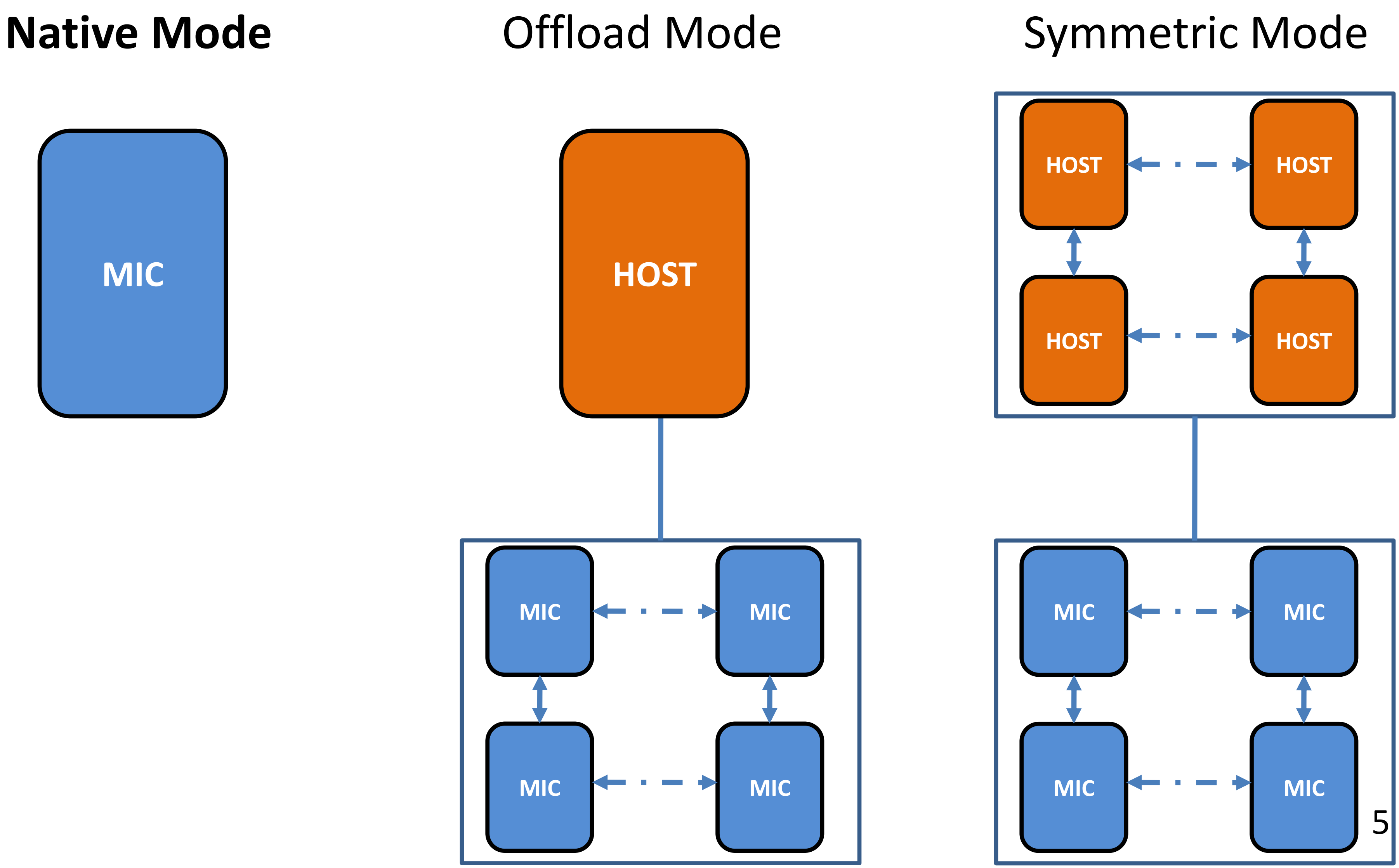
Xeon Phi Architectural Overview



- 61 * 4 = 244 cores
- Ring Interconnect
- Bidirectional
- x86 compatibility
- L1(32KB), L2(512KB)
- 8 Memory Controllers
- 16 GDDR5 memory
- PCI Express System

Xeon Phi – Modes of Programming

Native Mode | Offload Mode | Symmetric Mode



Contents

- Xeon Phi Architecture
- **Why OpenSHMEM?**
- RMA put/get vs. Local load/store
- Optimizing Collectives: Reduction as a use case
- Experimental Results: NAS parallel benchmarks

Why OpenSHMEM ?

- Bottleneck in selecting other PGAS models
  – Intel Compiler and MIC dependency
  – Fortran Coarray vs. OpenSHMEM
- Shared and distributed memory machines
- Reference implementation* developed by University of Houston is based on GASNet
- OpenSHMEM version for Intel Xeon Phi
- Test Bed -> Stampede Super Computer

* http://openshmem.org/

Contents

- Xeon Phi Architecture
- Why OpenSHMEM?
- **RMA put/get vs. Local load/store**
- Optimizing Collectives: Reduction as a use case
- Experimental Results: NAS parallel benchmarks

RMA put/get → Local load/store

- *shmem_ptr:* address of a data object on a specific PE
- No function call → enhancing compiler optimizations
- Enables optimizations such as vectorization

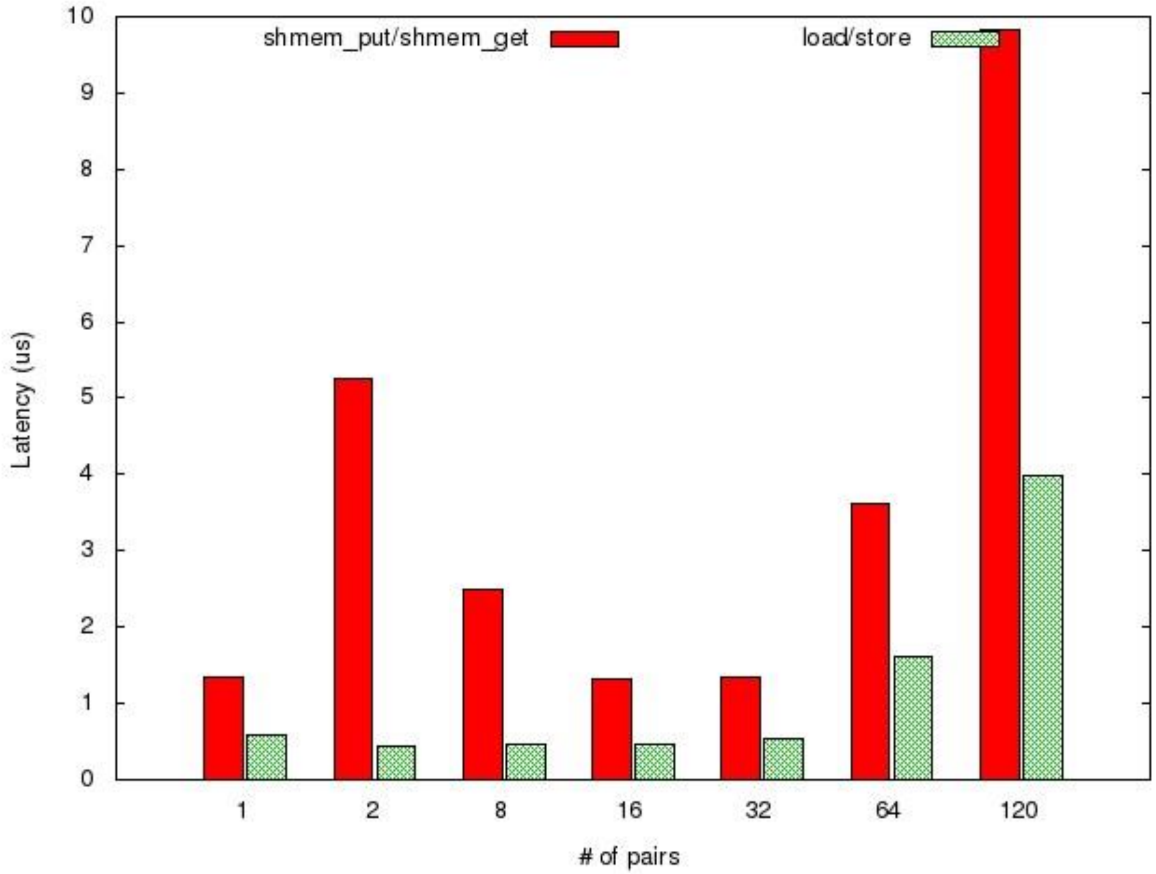
```
shmem_int_put(target,source,i,pe);
```

↓

```
int *ptr = (int *)shmem_ptr(target,pe);
for (m = 0; m < i; m+=1)
  ptr[m] = source[m];
```

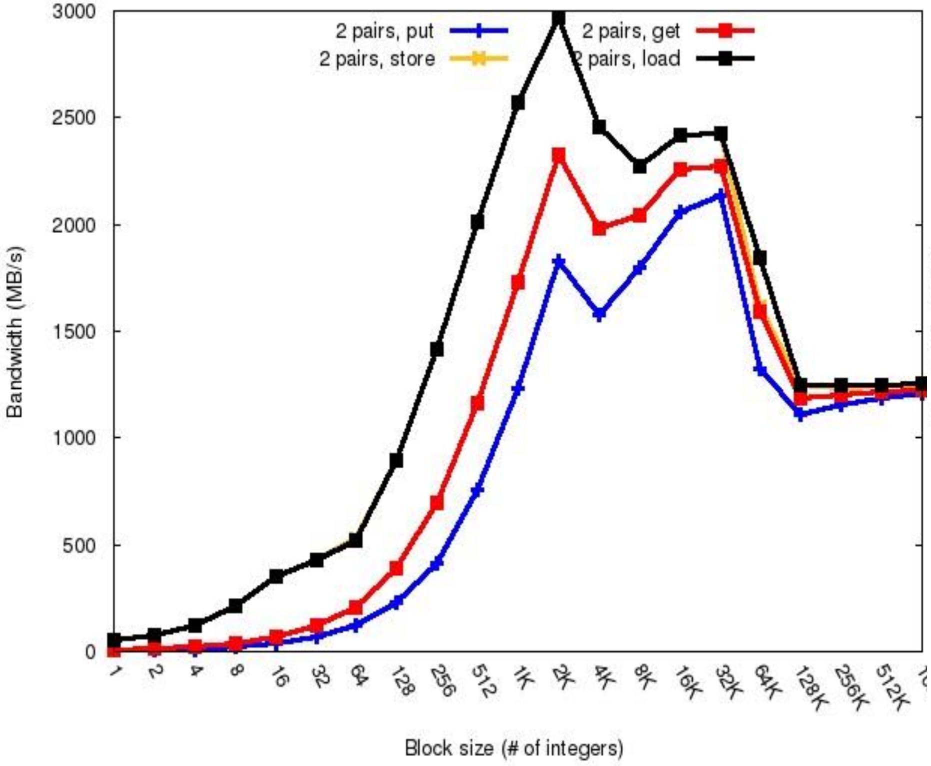
put/get → load/store - Latency

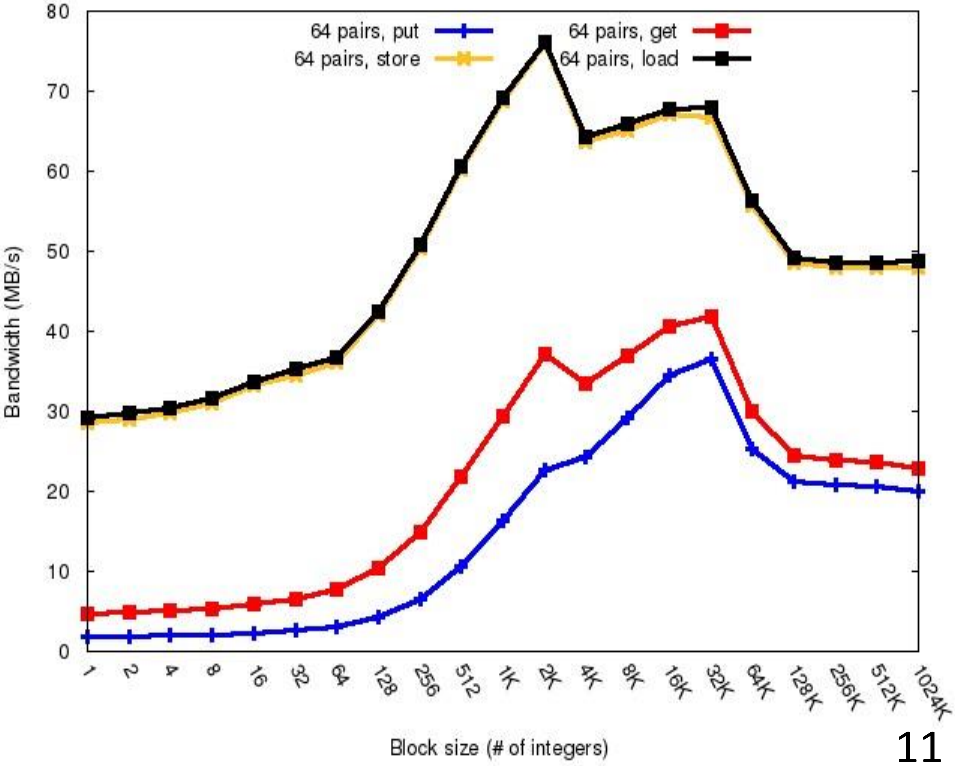
- Latency comparison
- *shmem_put/shmem_get* vs. load/store
- PGAS-Microbenchmarks from University of Houston



put/get → load/store - Bandwidth

- Bandwidth comparison
- *Blocksize* is represented in number of integers
- Results for 2 pairs and 64 pairs of PEs





Contents

- Xeon Phi Architecture
- Why OpenSHMEM?
- RMA put/get vs. Local load/store
- **Optimizing Collectives: Reduction as a use case**
- Experimental Results: NAS parallel benchmarks

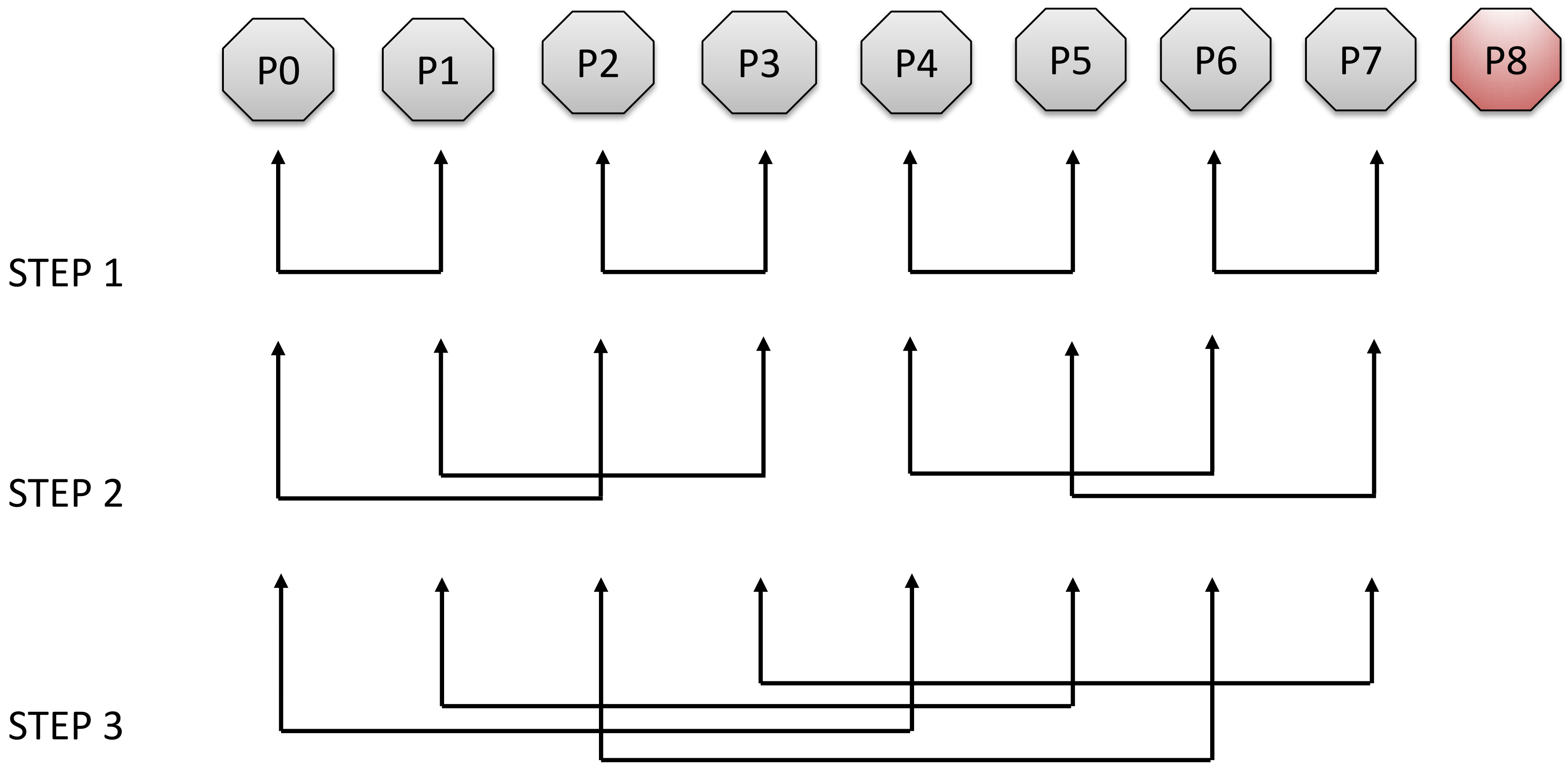
Optimizing SHMEM Collectives

- Using load/store operations to optimize SHMEM collectives: Barrier, Broadcast, Collect and Reduce
- Implementation of different reduction algorithms:
  - Flat Tree (FT)
  - Recursive Doubling (RD)
  - Bruck (BR)
  - Rabenseifner: Reduce Scatter All Gather (RSAG)*
- Performance comparison with Intel MPI and MVAPICH
  - PGAS-Microbenchmarks from University of Houston^

* Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of Collective Communication Operations in MPICH, 2005
^ https://github.com/uhhpctools/pgas-microbench

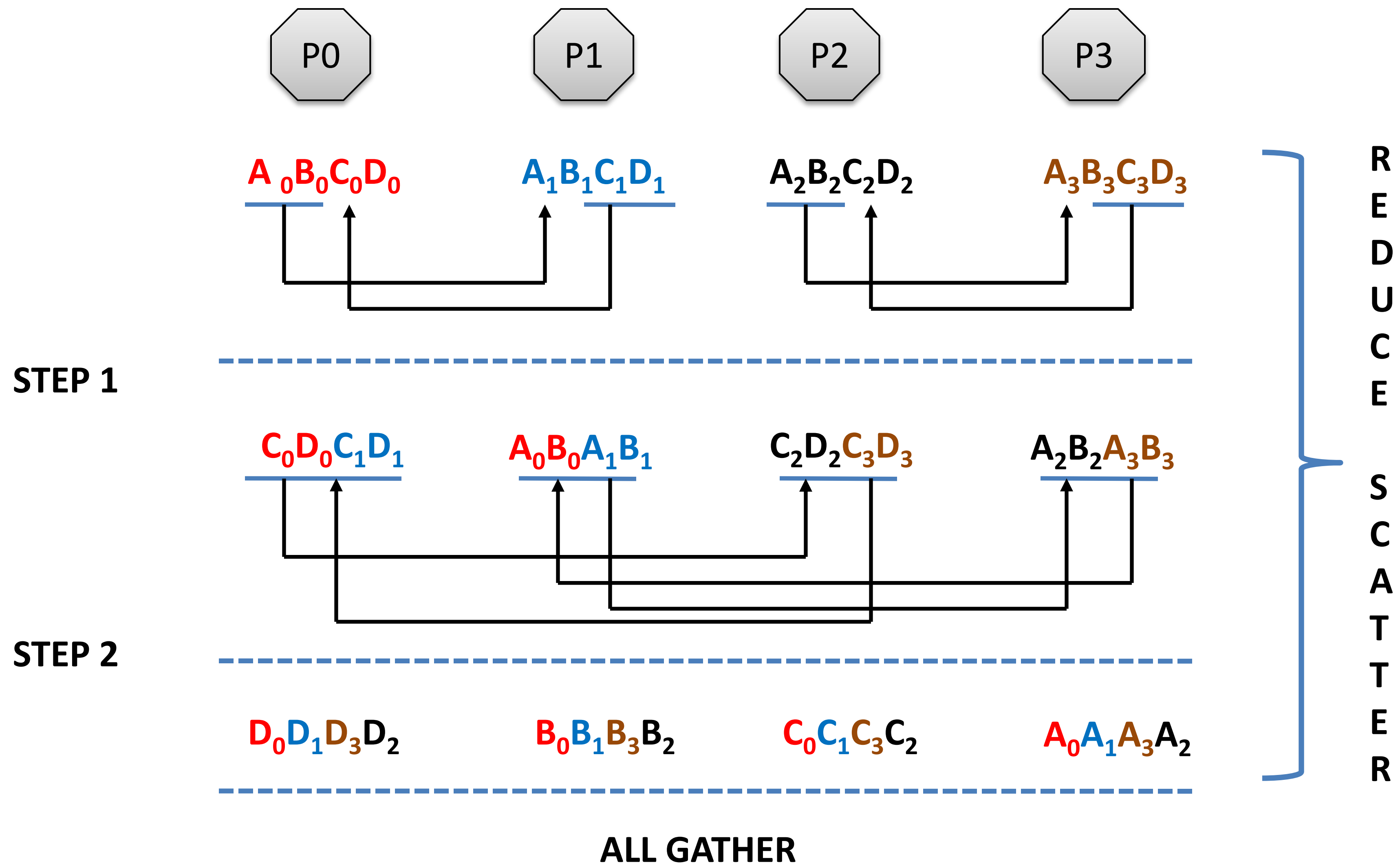
Recursive Doubling Algorithm

Array = { 1, 2 ……… 100 }



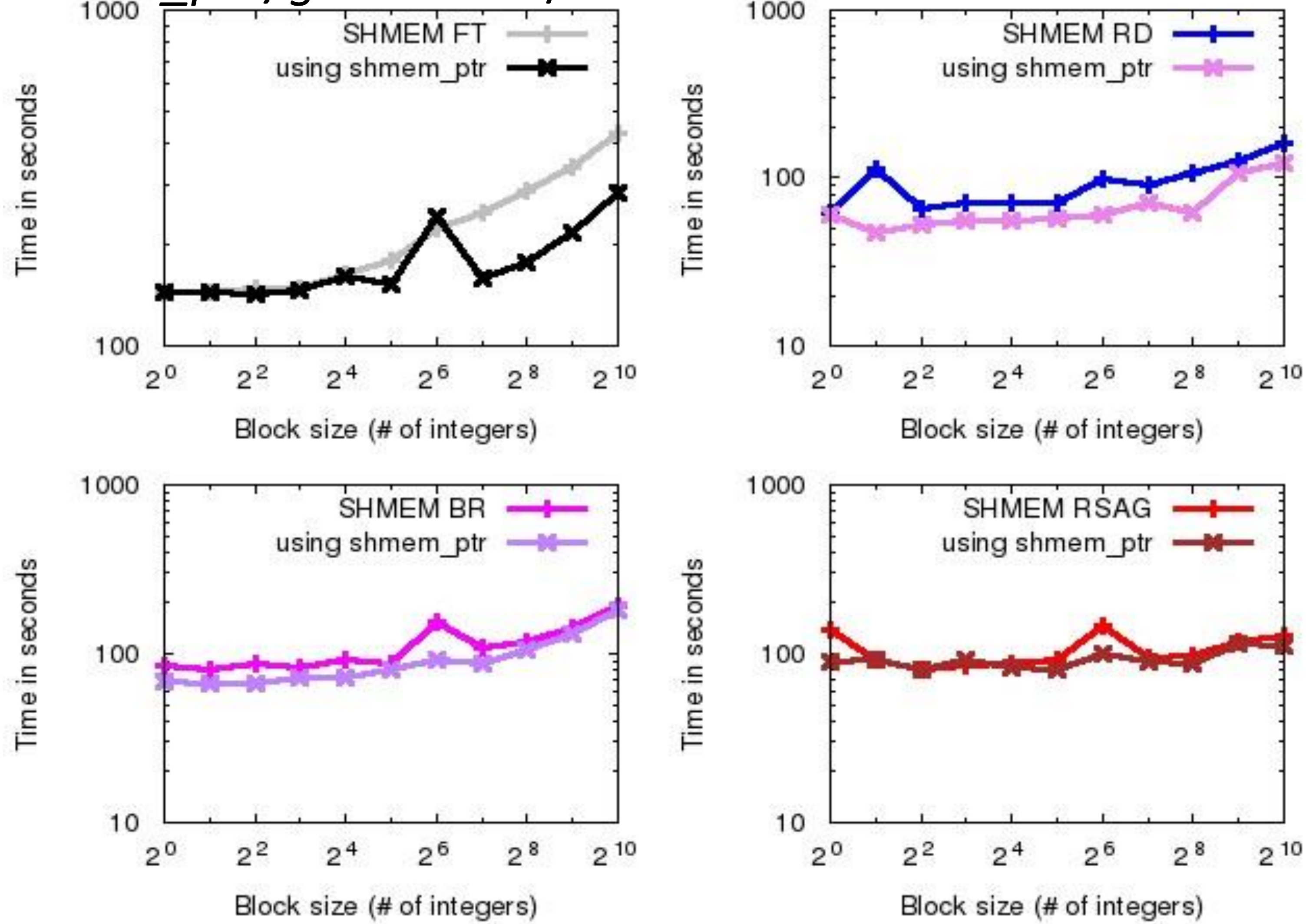
Reduce Scatter All Gather Algorithm

Array = { 1, 2 ……… 100 } ⟶ A = { 1, .. 25 } , B = { 26, .. 50 }, C = { 51, .. 75 } , D = { 76, .. 100 }



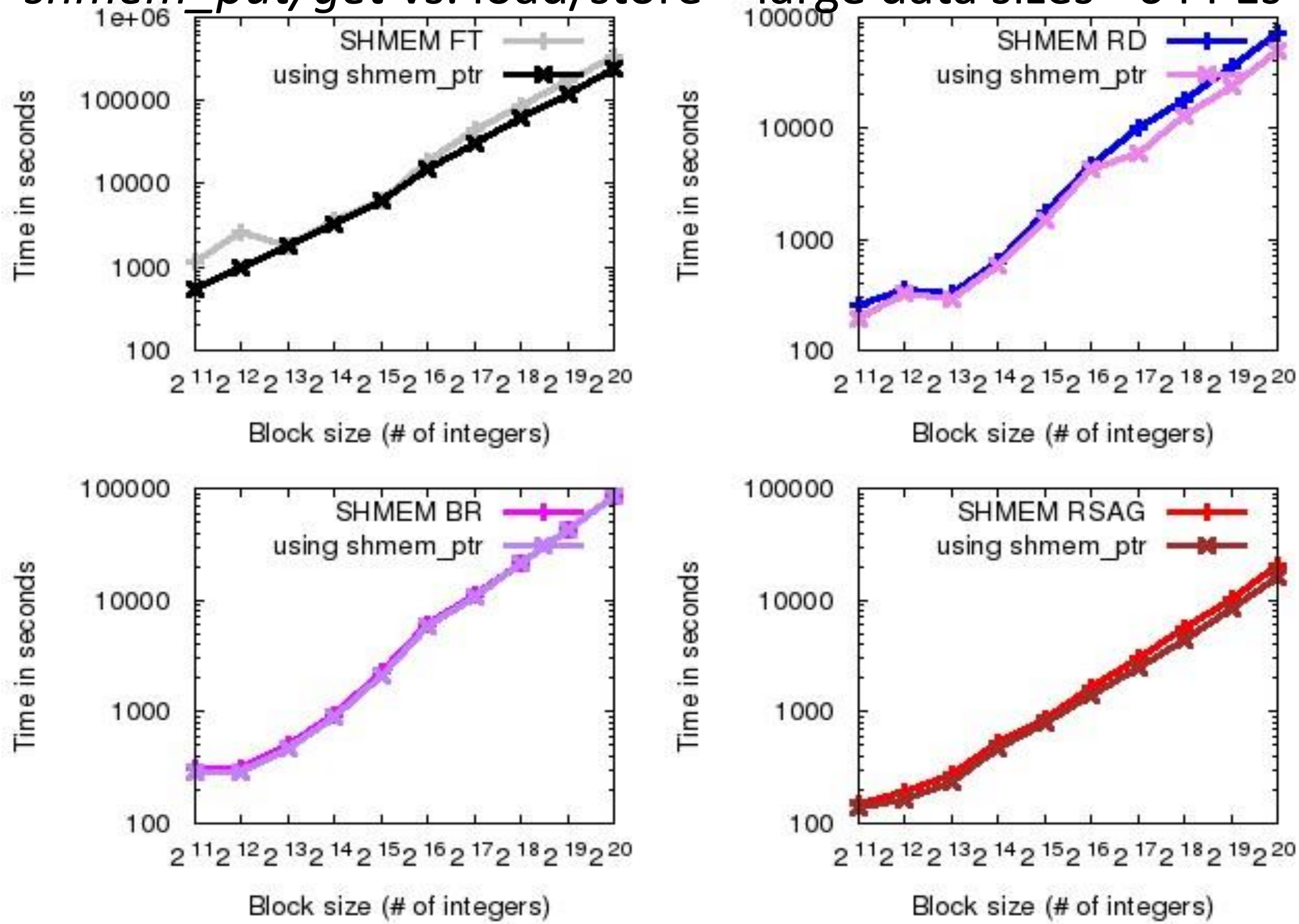
Reduction Algorithms Comparisons (1)

- *shmem_put/get* vs. load/store – small data sizes - 64 PEs



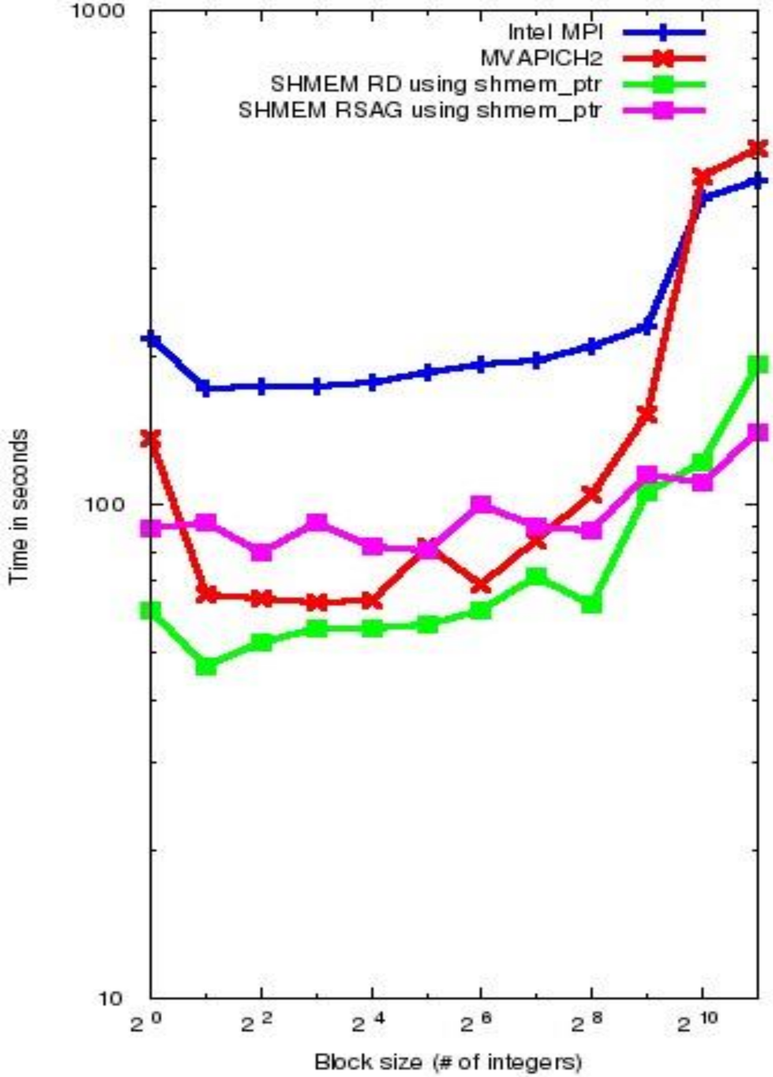
Reduction Algorithms Comparisons (2)

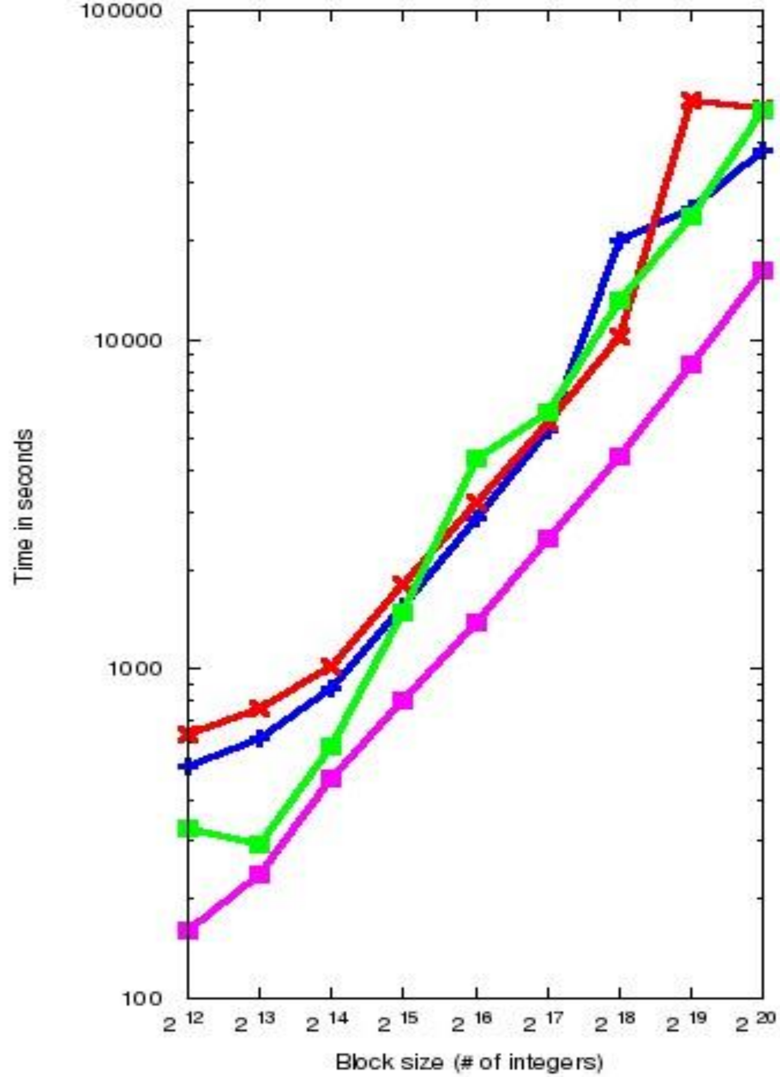
- *shmem_put/get* vs. load/store – large data sizes - 64 PEs



Reduction Algorithms Comparisons (3)

Various algorithms using load/store vs. Intel MPI and MVAPICH2 - 64 PEs





Contents

- Xeon Phi Architecture
- Why OpenSHMEM?
- RMA put/get vs. Local load/store
- Optimizing Collectives: Reduction as a use case
- **Experimental Results: NAS parallel benchmarks**

NAS Parallel Benchmarks

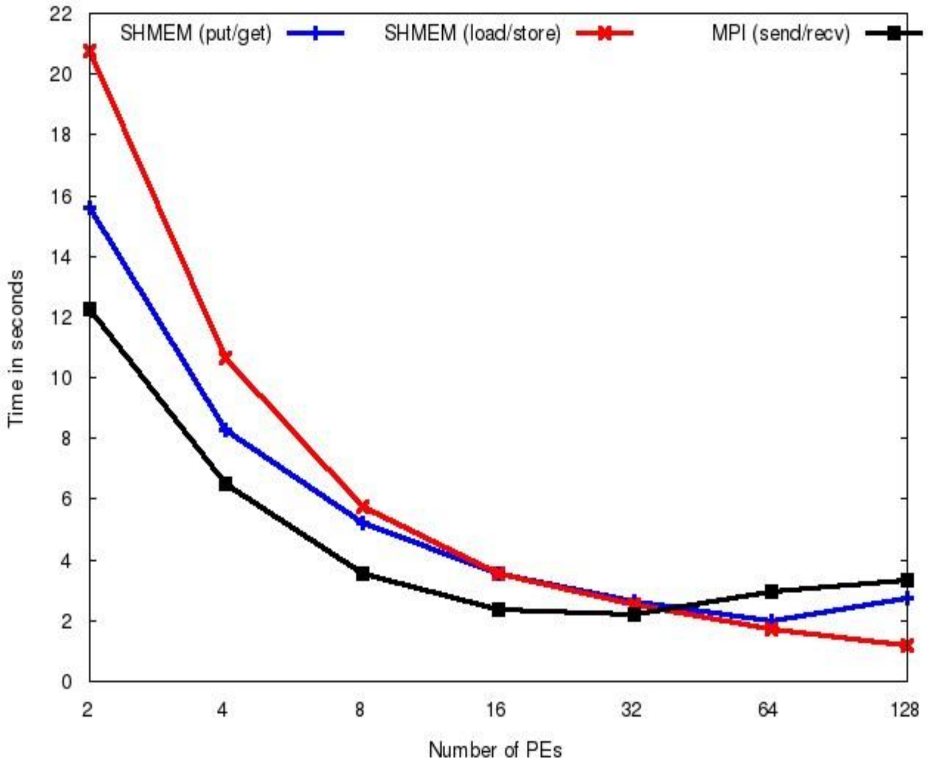
- MPI, CLASS C
- NAS SHMEM benchmarks*

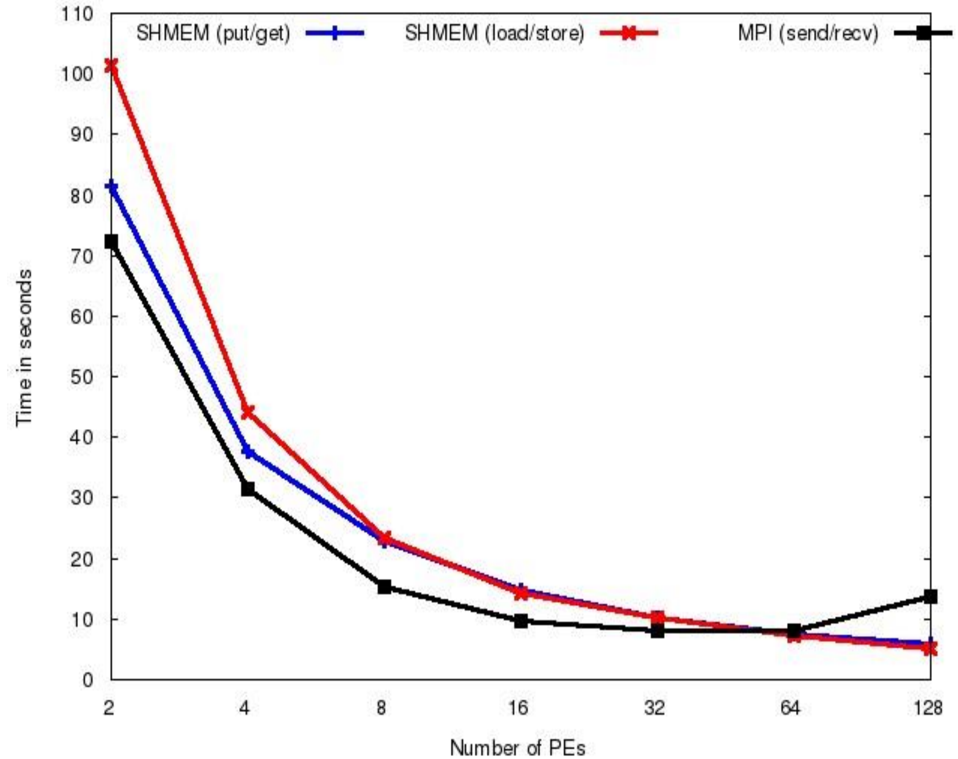
Benchmark	Reduction (%)	Remote Access (%)	SHMEM* Ver. Available
MG	0.1	19.6	YES
BT	0	15.5	YES
EP	1.6	0	YES
SP	**0**	**44.1**	**YES**
IS	**12.4**	**11.7**	**YES**
CG	0	33.2	NO
FT	0.8	31.2	NO
DT	0	10	NO
LU	0.1	14.8	NO

* https://github.com/openshmem-org/openshmem-npbs

Experimenting on NAS Benchmarks (1)

IS NAS Benchmark – *shmem_put/get* vs. load/store

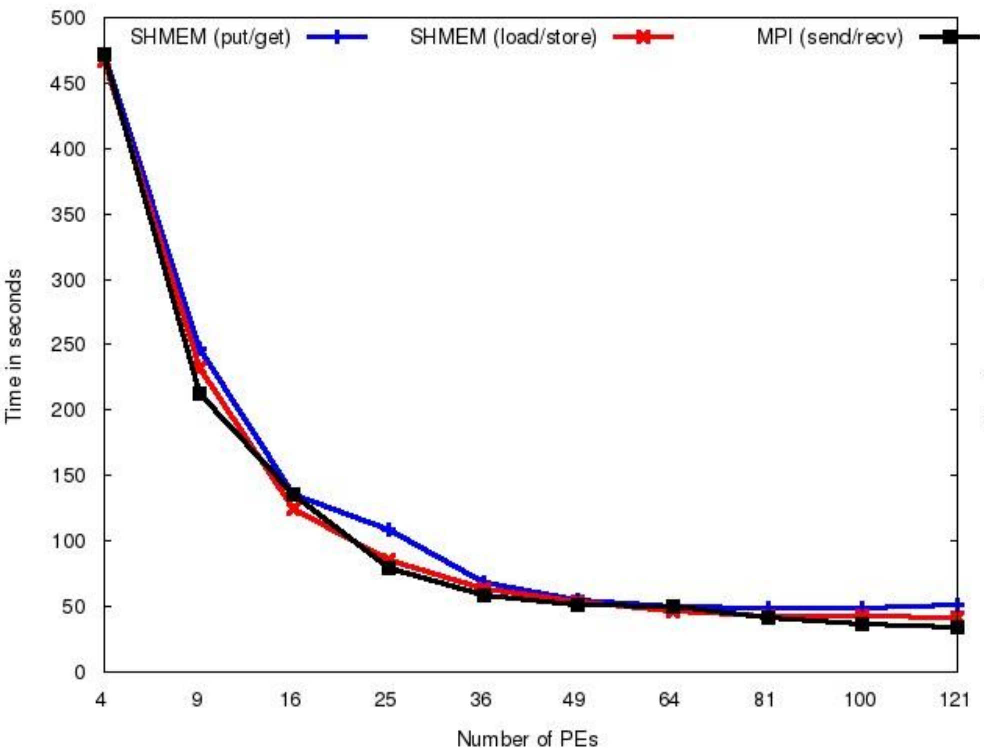




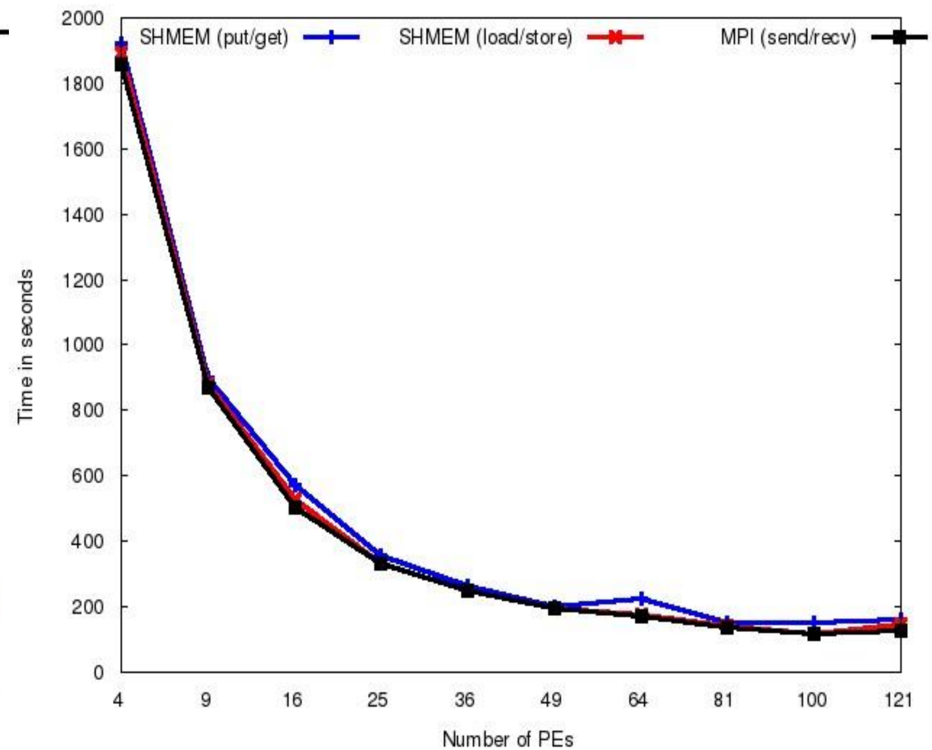
- Class B
- Class C

Experimenting on NAS Benchmarks (2)

SP NAS Benchmark – *shmem_put/get* vs. load/store



- Class B

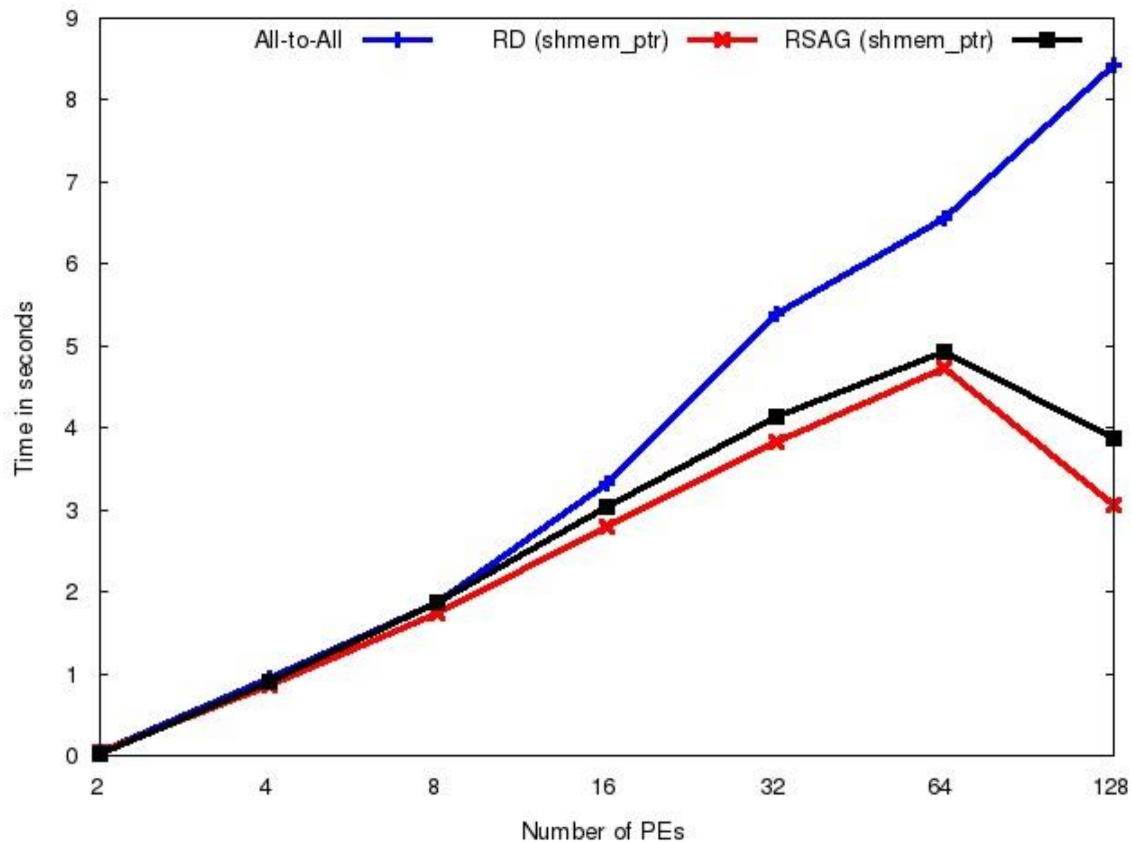


- Class C

Experimenting on NAS Benchmarks (3)

- IS NAS Benchmark, CLASS C
- load/store with various reduction algorithms
- All-to-All: Default reduction in

OpenSHMEM reference implementation



Conclusion

- Improved reduction algorithms: up to 22% compared to MVAPICH and 60% compared to IMPI
- Improved communications: decrease in latency by up to 60% and increase in bandwidth by up to 12x
- Use RD for small message sizes and RSAG for large message sizes.

Future Work

- Extending the reduction optimizations to other collectives such as Barriers
- Automation of translating RMA calls into load/store using OpenUH for shared memory systems
- PGAS Language-based on MIC, such as Fortran Coarray

Acknowledgments

- TOTAL
- TACC, Texas Advanced Computing Center

Native Mode-Based Optimizations of Remote Memory Accesses in OpenSHMEM for Intel Xeon Phi

Naveen Namashivayam, Sayan Ghosh, Dounia Khaldi, Deepak Eachempati, Barbara Chapman

Department of Computer Science

University of Houston