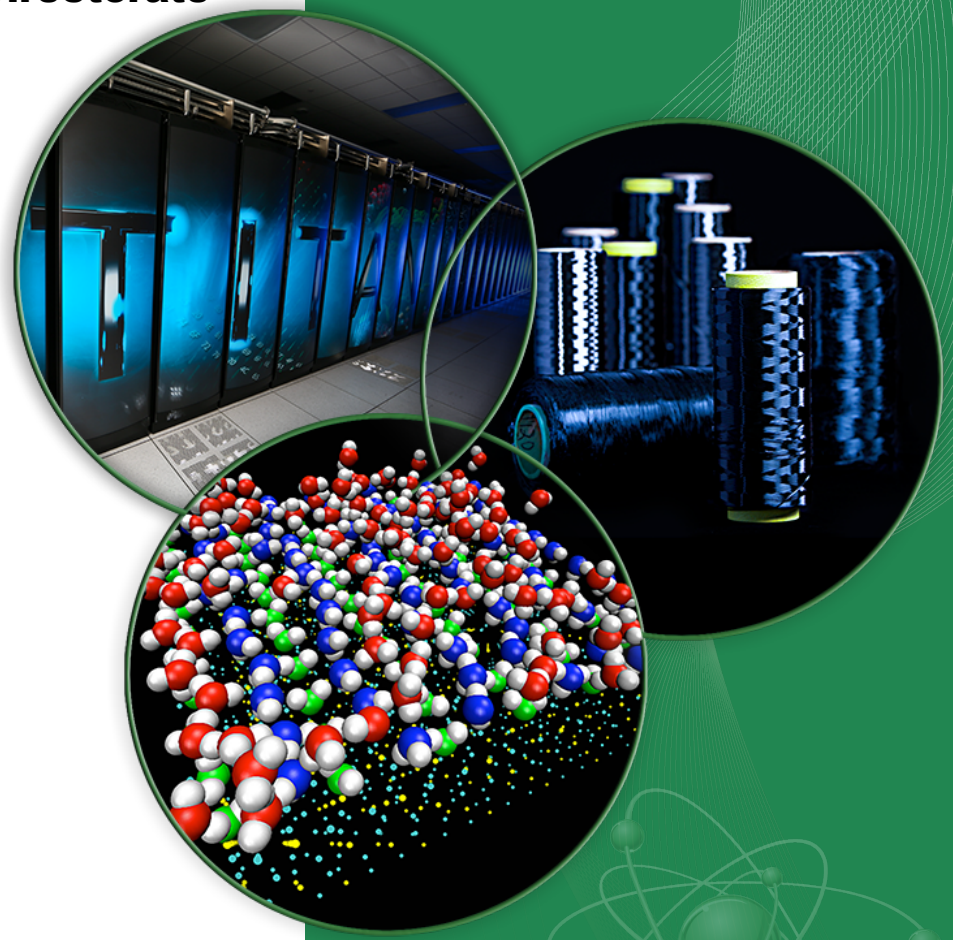
Oak Ridge National Laboratory

Computing and Computational Sciences Directorate



An Evaluation of OpenSHMEM Interfaces for the Variable-length Alltoallv() Collective Operation

OpenSHMEM Workshop 2015

M. Graham Lopez, Pavel Shamis,
Manjunath Gorentla Venkata

August 5, 2015

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

Overview

- Thanks to David Knaak (Cray)
- Understanding alltoall and alltoallv
- alltoallv in use
- Examining the alltoallv interface for usability
- Sanity checks
- Further considerations

Introduction – Alltoall()

- Each of N PEs sends and receives the **same amount** of data with all other N Pes
  - $O(N^2)$ exchanges
  - Total amount of data transferred

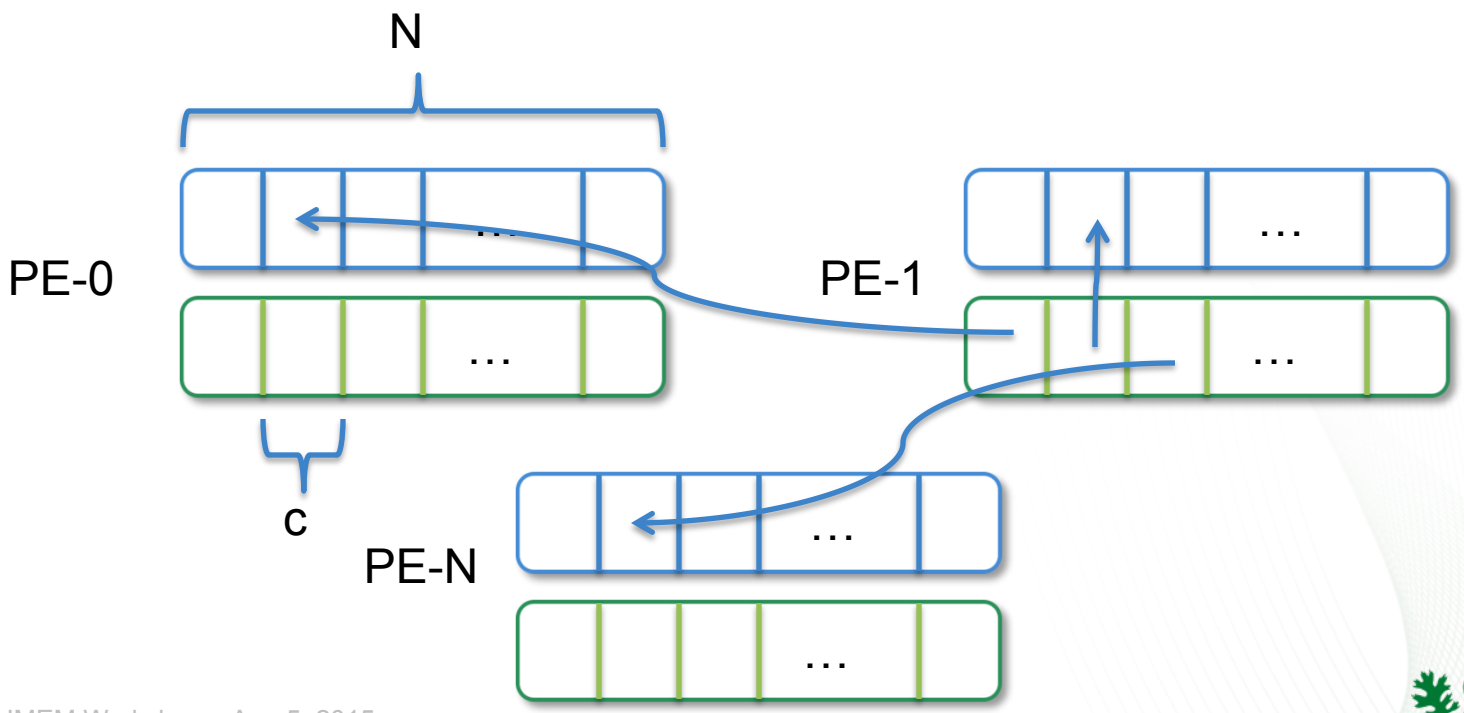
$$T_t = c * N * (N - 1)$$

- Useful for numerical algorithms
  - FFT
  - Matrix transpose
  - General integral transforms
  - Adaptive mesh algorithms

Introduction – Alltoall()

- Each of N PEs sends and receives the **same amount** of data with all other N Pes
  - $O(N^2)$ exchanges
  - Total amount of data transferred

$$T_t = c * N * (N - 1)$$



Introduction – Alltoallv()

- Generalization of Alltoall()
- Each of N PEs sends and receives **any amount** of data with all other N Pes
  - $O(N^2)$ exchanges
  - Total amount of data transferred

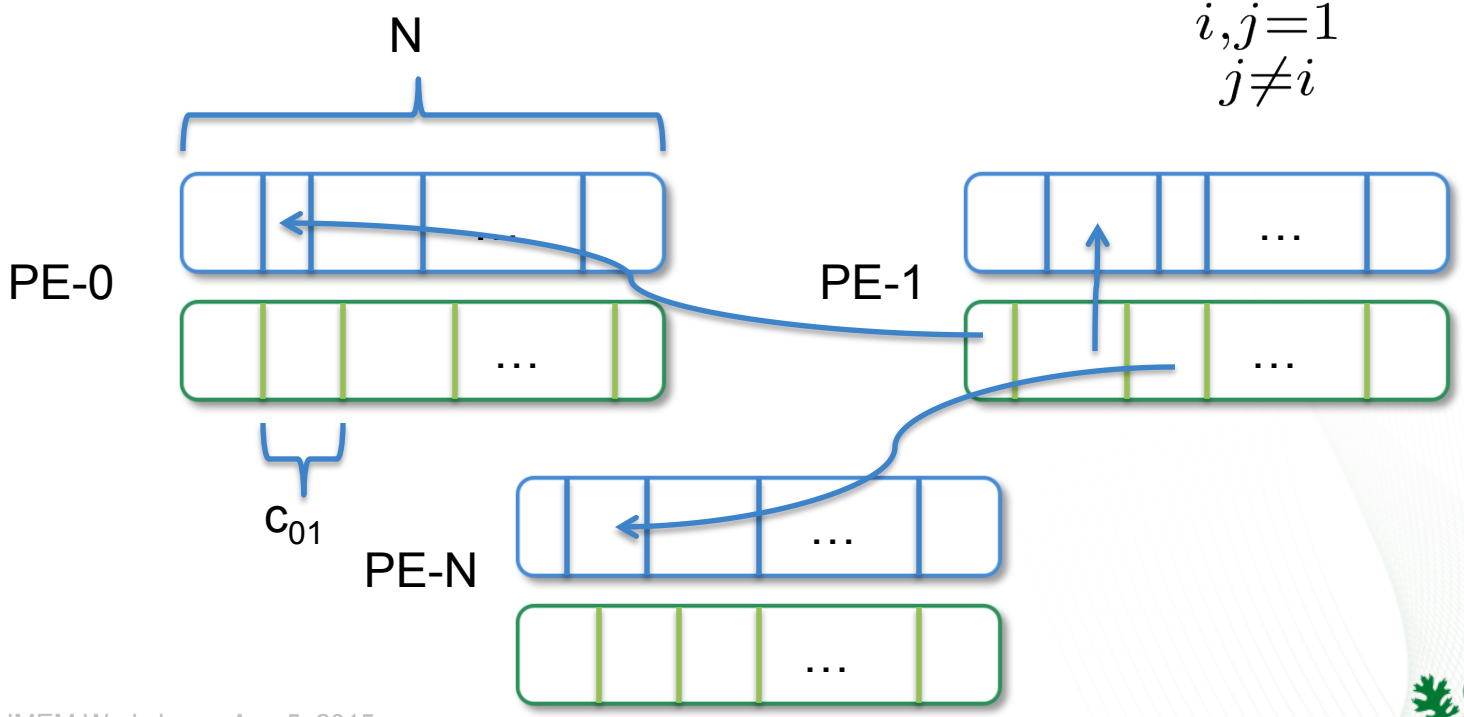
$$T_t = \sum_{\substack{i,j=1 \\ j \neq i}}^{N} c_{ij}$$

- Useful for similar algorithms…
  - FFT
  - Matrix transpose
  - General integral transforms
  - Adaptive mesh algorithms
- But with unequal loads
  - Non-uniform resources
  - Load balancing

Introduction – Alltoallv()

- Generalization of Alltoall()
- Each of N PEs sends and receives **any amount** of data with all other N Pes
  - $O(N^2)$ exchanges
  - Total amount of data transferred

$$T_t = \sum_{\substack{i,j=1 \\ j \neq i}}^{N} c_{ij}$$



Introduction – Alltoallv()

- Generalization of Alltoall()
- Each of N PEs sends and receives **any amount** of data with all other N Pes
  - $O(N^2)$ exchanges
  - Total amount of data transferred

$$T_t = \sum_{\substack{i,j=1 \\ j \neq i}}^{N} c_{ij}$$

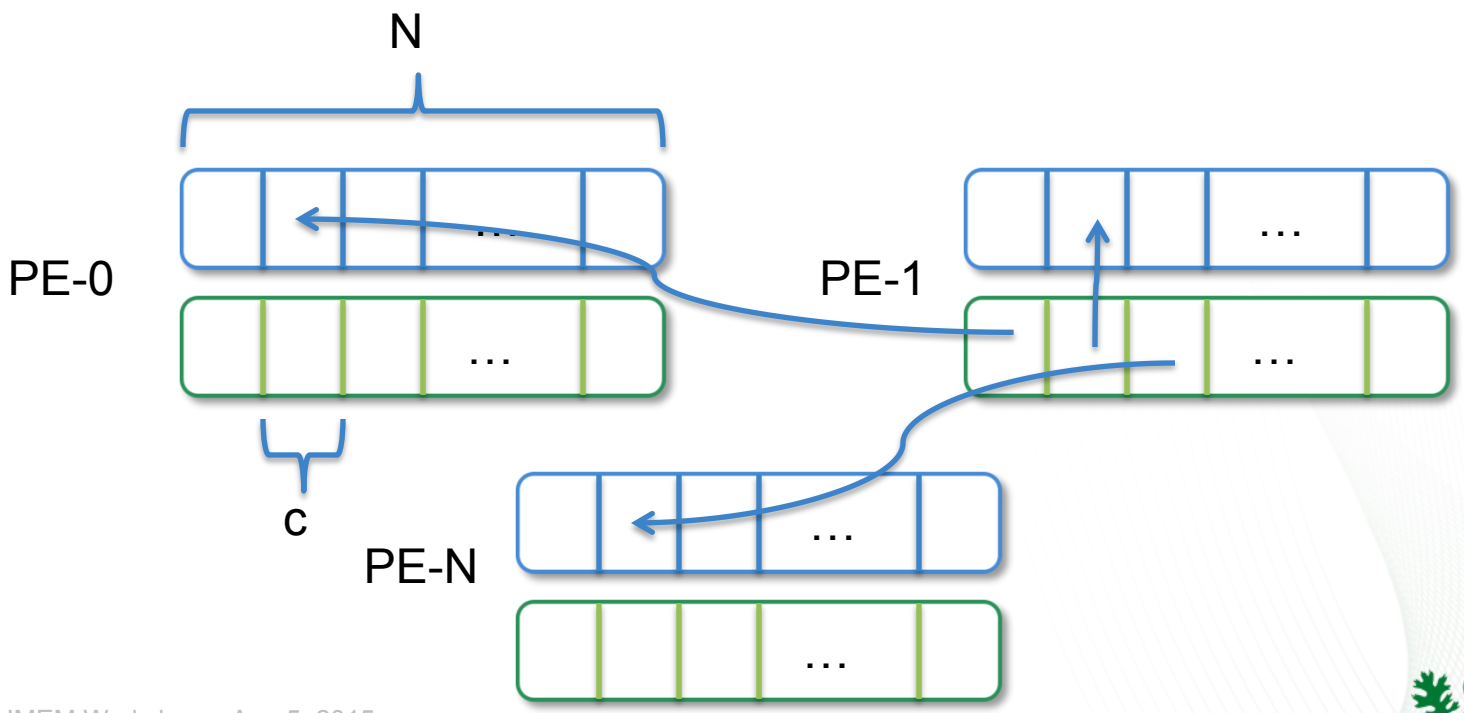
Let $c = c_{ij}$

$$T_t = c * N * (N - 1)$$

Introduction – Alltoall()

- Each of N PEs sends and receives the **same amount** of data with all other N Pes
  - $O(N^2)$ exchanges
  - Total amount of data transferred

$$T_t = c * N * (N - 1)$$



Alltoallv() in use

- NAS Parallel Benchmark suite
  - Integer Sort (IS) benchmark
- Syntactic convenience for combined ‘get’ and ‘put’ in a for loop.
  - Simplify code (hopefully)
  - Possibly allows for more sophisticated algorithms and/or hardware optimizations

Alltoallv() in use

```
/* This is the redistribution section: first find out how many keys
each processor will send to every other processor */

   for( j1=0; j1 < comm_size; j1++ ) {
       shmem_int_put(&t_sizes[my_rank], &s_sizes[j1], 1, j1);
   }
   shmem_barrier_all();

/* Determine the receive array displacement for the buckets */

   t_offsets[0] = 0;
   for( i=1; i < comm_size; i++ ) {
       t_offsets[i] = t_offsets[i-1] + t_sizes[i-1];
   }
   shmem_barrier_all();

/* Now send the keys to respective processors */

   for( j1=0; j1 < comm_size; j1++ ) {
      shmem_int_get(&k2, &t_offsets[my_rank], 1, j1);
      shmem_int_put(&key_buff2[k2/sizeof(INT_TYPE),
                    &key_buff1[s_offsets[j1]/sizeof(INT_TYPE),
                    s_sizes[j1]/sizeof(INT_TYPE), j1);
   }
   shmem_barrier_all();
```

Alltoallv() - MPI/Cray Interface

```
extern void
shmem_alltoallv(void *target,
                size_t *t_offsets,
                size_t *t_sizes,
                const void *source,
                size_t *s_offsets,
                size_t *s_sizes,
                int PE_start,
                int logPE_stride,
                int PE_size,
                long *psync);
```

Alltoallv() – MPI/Cray Interface

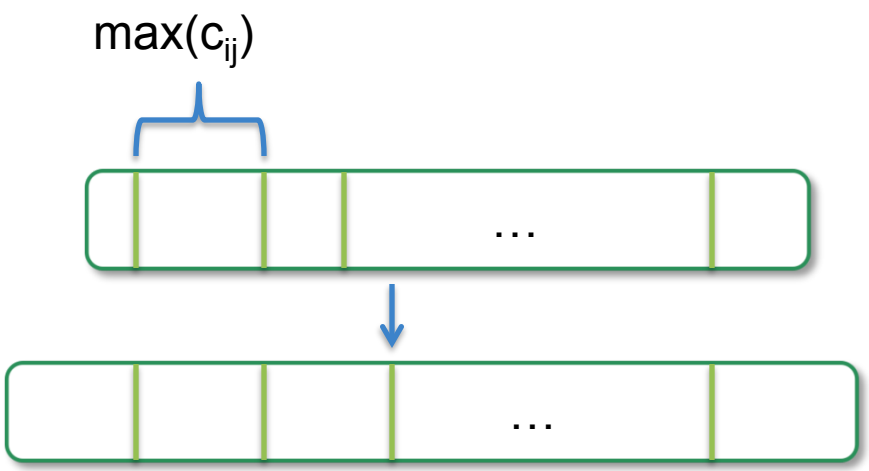
```
/* This is the redistribution section: first find out how many keys
each processor will send to every other processor */

   shmem_alltoall(&t_sizes, &s_sizes, sizeof(size_t),
                  0, 0, shmem_n_pes(), pSync);
   shmem_barrier_all();

/* Determine the receive array displacement for the buckets */

   t_offsets[0] = 0;
   for( i=1; i < comm_size; i++ ) {
       t_offsets[i] = t_offsets[i-1] + t_sizes[i-1];
   }
   shmem_barrier_all();

/* Now send the keys to respective processors */

   shmem_alltoallv(key_buff2, t_offsets, t_sizes,
                   key_buff1, s_offsets, s_sizes,
                   0, 0, comm_size, pSync);
   shmem_barrier_all();
```

Attempt to simplify Alltoallv()

- Remove extra collective by just using alltoall()
  - Set $c = c_{max} = \max(c_{ij})$

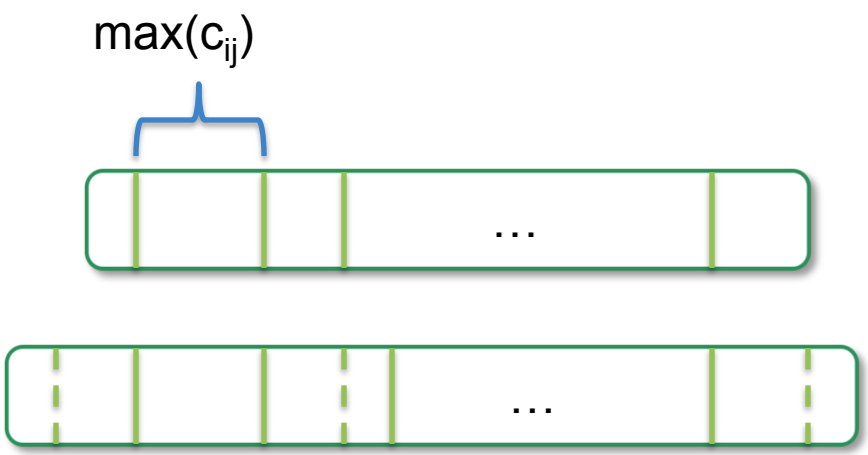
Attempt to simplify Alltoallv()

- Remove extra collective by just using alltoall()
  - Set $c = c_{max} = \max(c_{ij})$



Attempt to simplify Alltoallv()

- Remove extra collective by just using alltoall()
  - Set $c = c_{max} = \max(c_{ij})$
- At least two disadvantages
  - Extra data transferred
  - How to get data out cleanly?



$$T_{extra} = \sum_{\substack{i,j=1 \\ j \neq i}}^{N} (c_{max} - c_{ij})$$

Alltoallv_max() – Variant

```
extern void
shmem_alltoallv_max(void *target,
                    size_t max_offset,
                    size_t *t_sizes,
                    const void *source,
                    size_t *s_offsets,
                    size_t *s_sizes,
                    int PE_start,
                    int logPE_stride,
                    int PE_size,
                    long *psync);
```

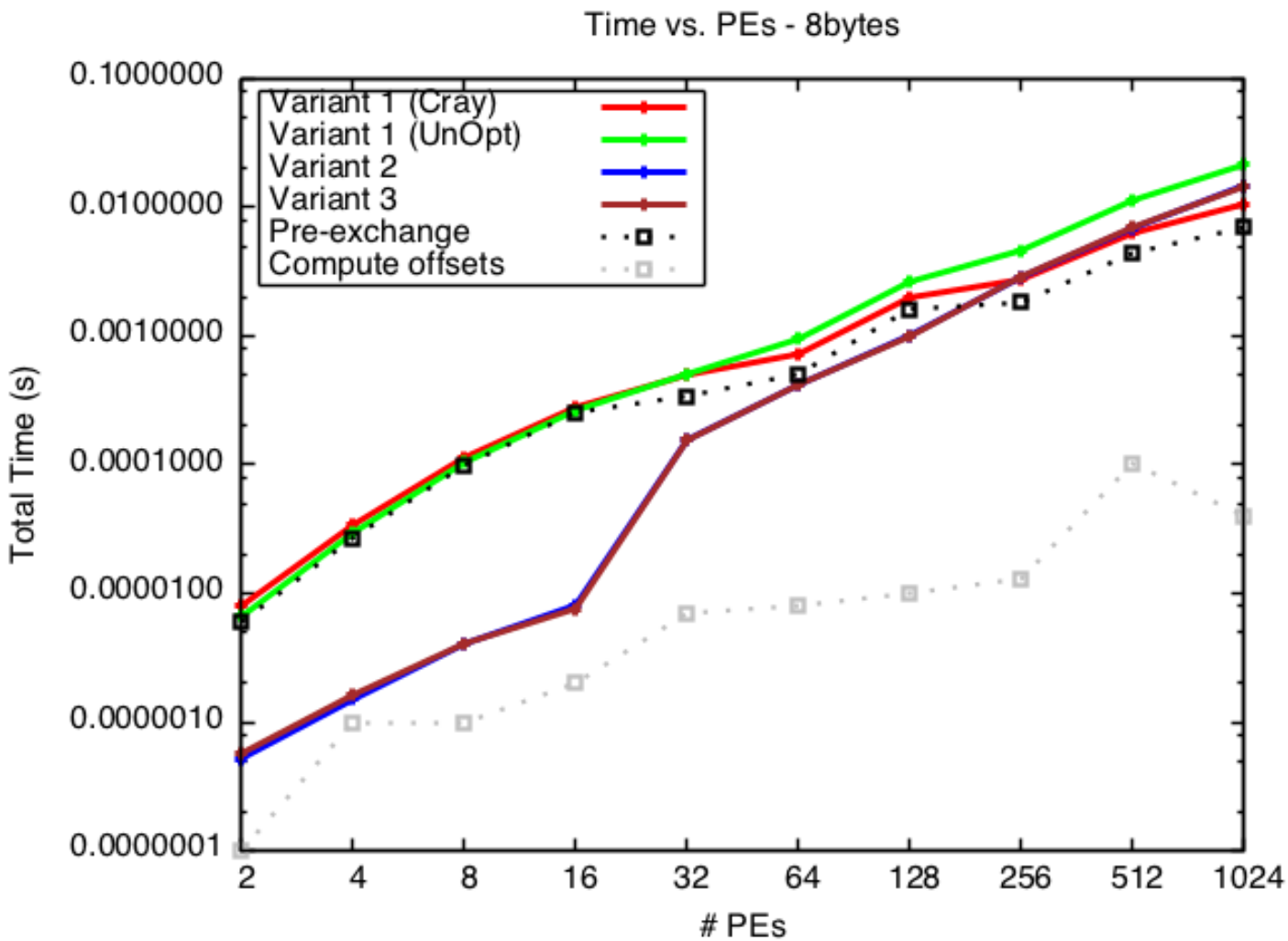
Alltoallv() – MPI/Cray Interface

```
/* This is the redistribution section: first find out how many keys
each processor will send to every other processor */

   shmem_alltoall(&t_sizes, &s_sizes, sizeof(size_t),
                  0, 0, shmem_n_pes(), pSync);
   shmem_barrier_all();

/* Determine the receive array displacement for the buckets */

   t_offsets[0] = 0;
   for( i=1; i < comm_size; i++ ) {
       t_offsets[i] = t_offsets[i-1] + t_sizes[i-1];
   }
   shmem_barrier_all();

/* Now send the keys to respective processors */

   shmem_alltoallv(key_buff2, t_offsets, t_sizes,
                   key_buff1, s_offsets, s_sizes,
                   0, 0, comm_size, pSync);
   shmem_barrier_all();
```

Alltoallv() – Max Variant Interface

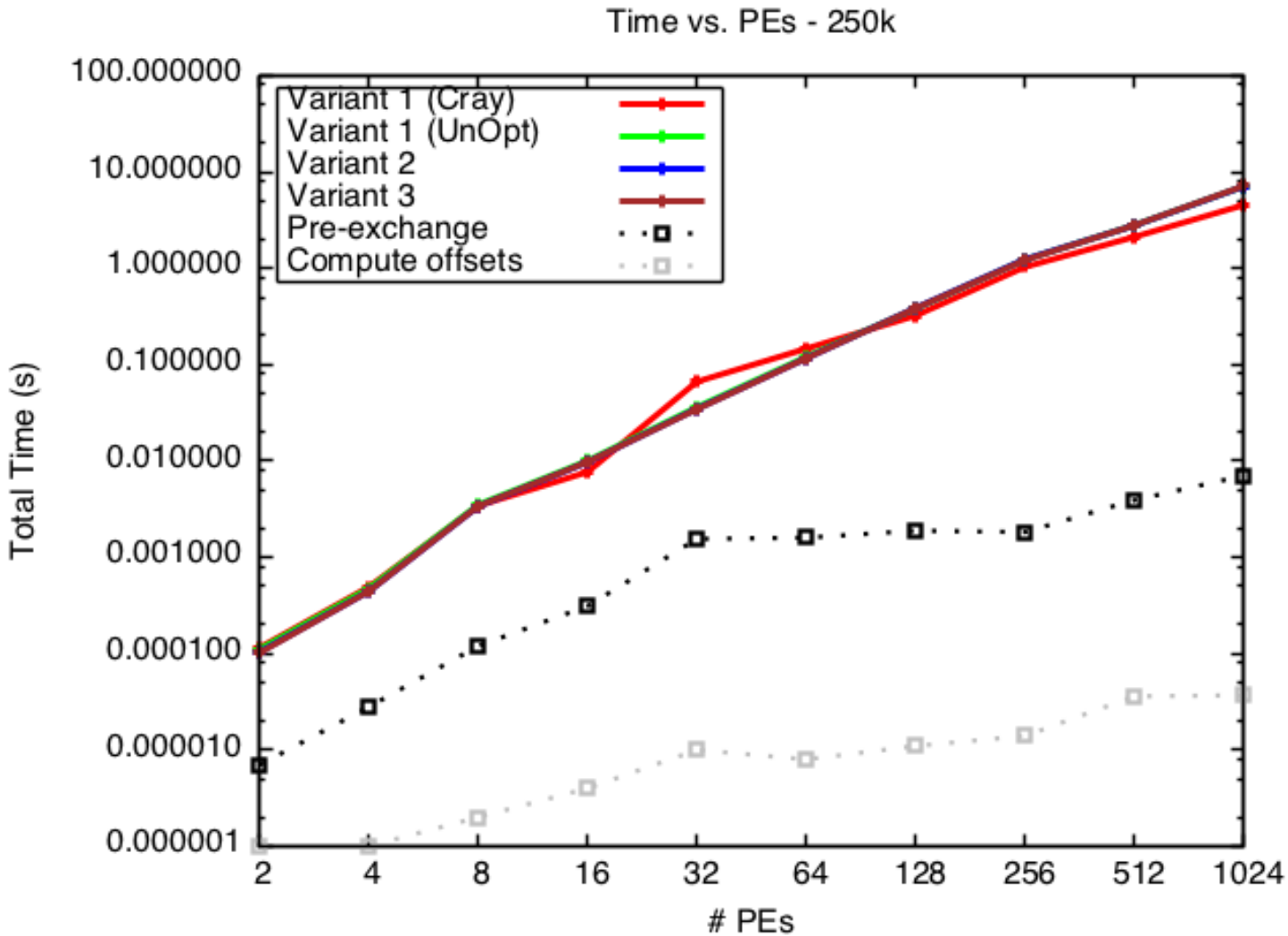
```
/* Now send the keys to respective processors */

   shmem_alltoallv_max(key_buff2, max_offset, t_sizes,
                       key_buff1, s_offsets, s_sizes,
                       0, 0, comm_size, pSync);
   shmem_barrier_all();
```

Evaluations



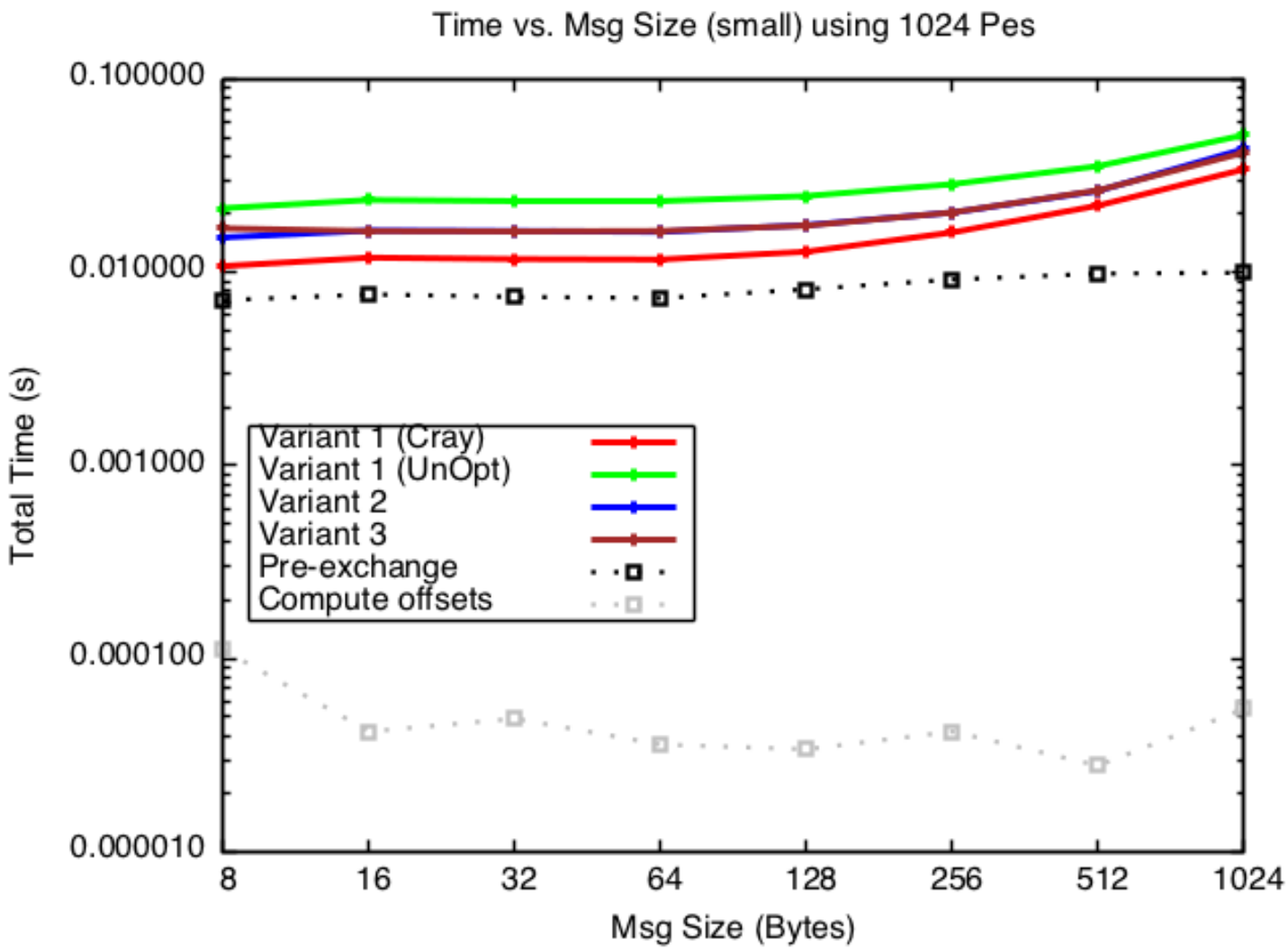
OAK RIDGE National Laboratory

Evaluations



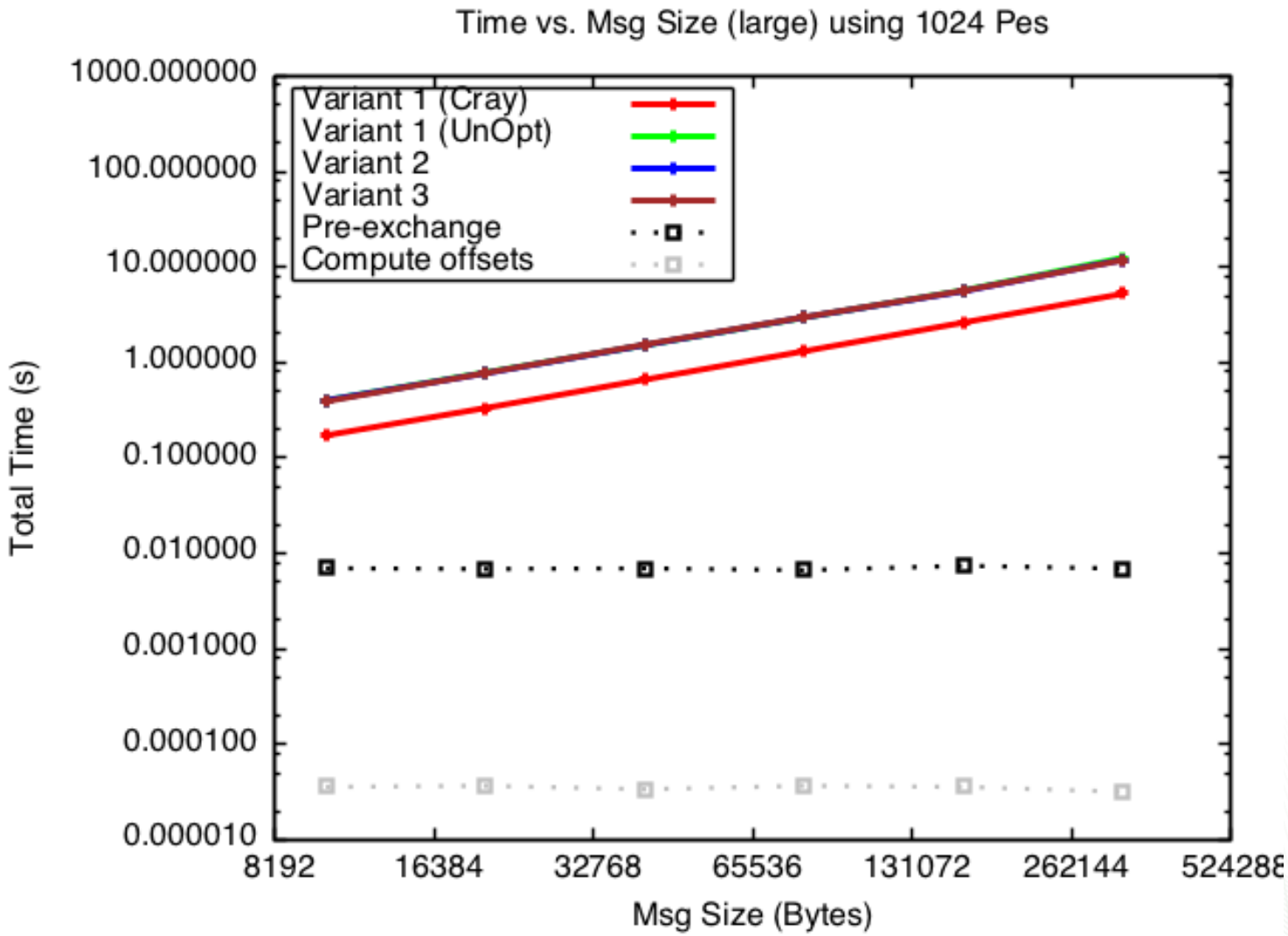
OAK RIDGE National Laboratory

Evaluations



OAK RIDGE National Laboratory

Evaluations

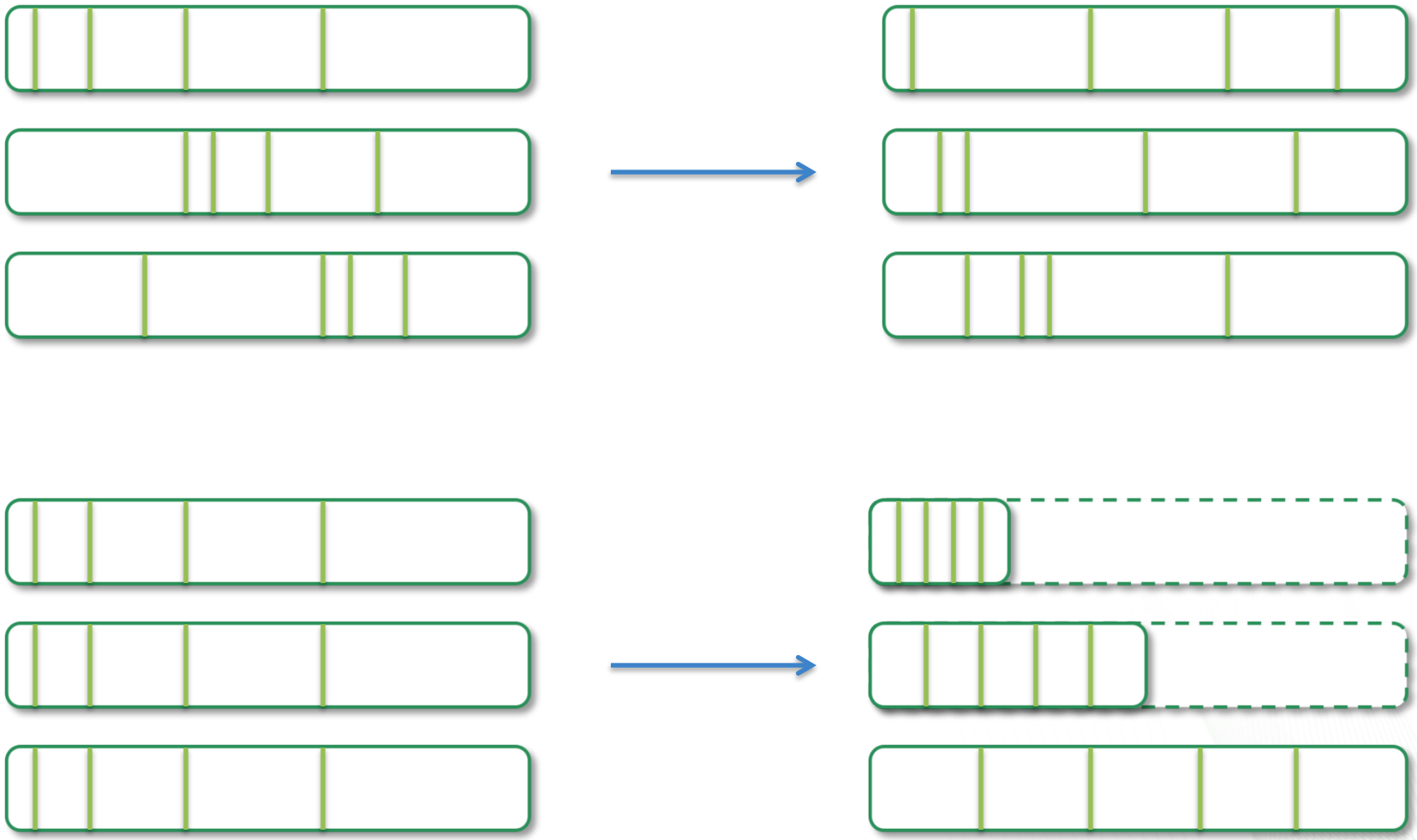


OAK RIDGE National Laboratory

Further considerations

- Opportunities for optimization
  - Hardware-supported
    - Information about data transferred to reduce overhead
  - In-software (algorithms)
    - Apply traditional alltoall() or alltoallv() algorithms, depending on the interface
- Alternate semantics and possible implications
  - Memory-bound applications

Memory-bound apps



Memory issues

- Alternate semantics and possible implications
  - Memory-bound applications

- At least two obvious ‘solutions’
  - Allow asymmetric symmetric allocations
  - Allow writing to private memory
    - (doesn’t need to be exposed directly to the user)

Summary

MPI/Cray interface

- Most general
- People are used to it
- SHMEM semantics may not allow all memory-saving advantages to be realized

max_offsets

- Can avoid an extra collective
  - More convenient
  - Better performance
- No performance or optimization negatives found yet

Summary

MPI/Cray interface

- Most general
- People are used to it
- SHMEM semantics may not allow all memory-saving advantages to be realized

max_offsets

- Can avoid an extra collective
  - More convenient
  - Better performance
- No performance or optimization negatives found yet

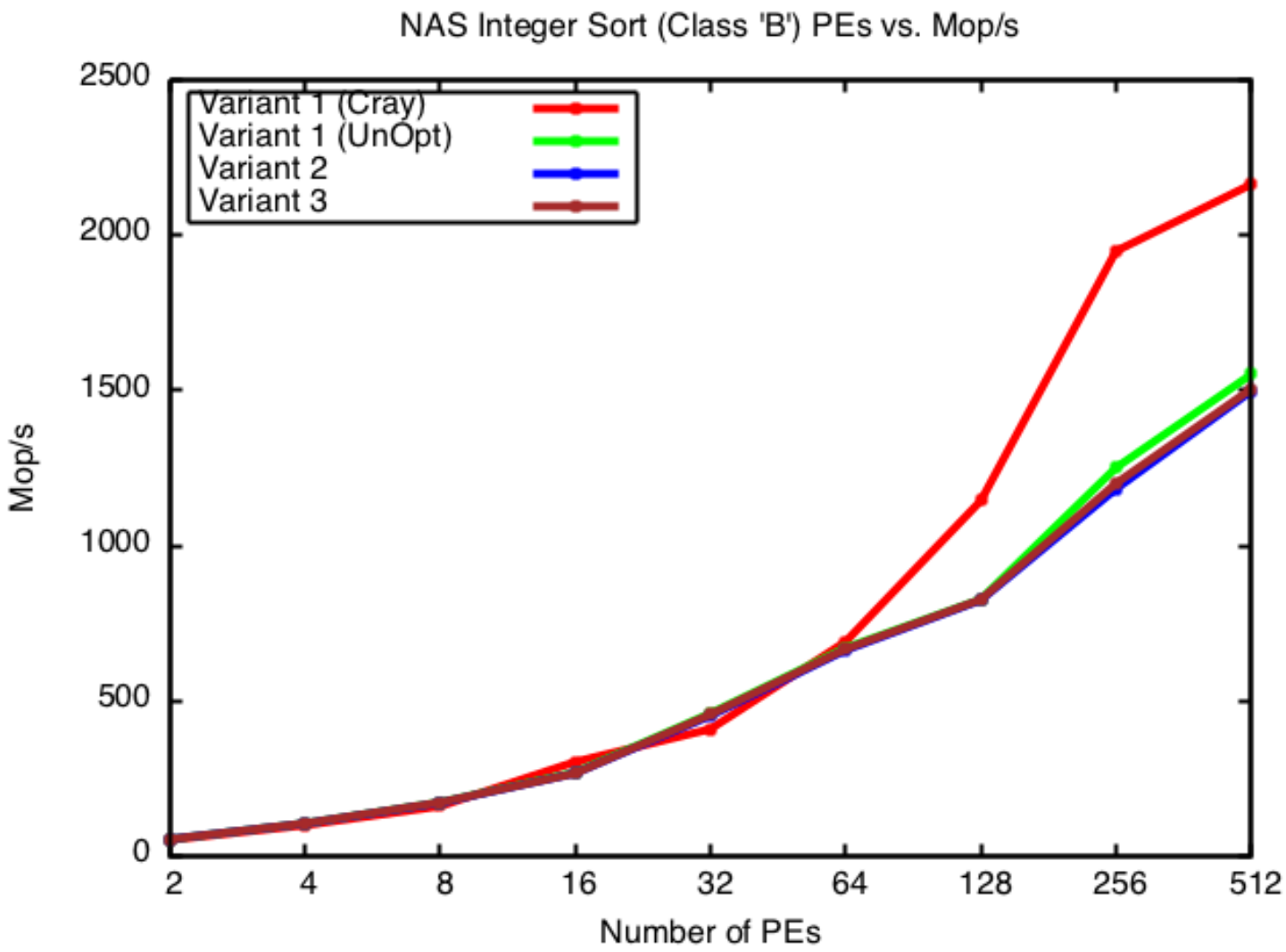
How can alltoallv() be reconsidered to better fit SHMEM ?

Acknowledgements

This work was supported by the United States Department of Defense (DoD) and used resources of the DoD-HPC Program at Oak Ridge National Laboratory.

Questions?

Evaluations



OAK RIDGE National Laboratory