

Lightweight Instrumentation and Analysis using OpenSHMEM Performance Counters

Md. Wasi-ur- Rahman, David Ozog, and James Dinan

Legal Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit <http://www.intel.com/benchmarks>.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/benchmarks>.

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system.

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at <http://www.intel.com/go/turbo>.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

© 2018 Intel Corporation.

Intel, the Intel logo, and Intel Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as property of others.

Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- Experimental Analysis
- Conclusion and Future Work

Introduction and Motivation

- Effective performance profiling and analysis tools for PGAS applications have been challenging
 - One-sided high-throughput usage model
 - Scale of parallel applications
 - Rate and volume of communication operations generated
- Tracing is the most common approach
 - Captures a log of each operation for offline analysis
 - Instrumentation introduce overhead and impact dynamic behavior of applications
- Can an alternative lightweight instrumentation approach be devised that skip library interposition, yet achieve detailed profiling for communication performance?

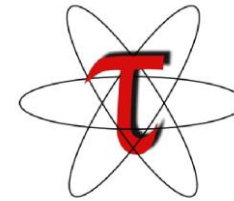
Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- Experimental Analysis
- Conclusion and Future Work

Existing Approaches

- A number of tools provide communication tracing and analysis for OpenSHMEM applications
 - Collect detailed information
 - Plug-in/out capabilities
 - User-friendly interfaces
- Can generate per-operation overhead
- Requires library interposition
- Using hardware performance counters, e.g. PAPI is challenging for process-level application performance analysis

CrayPAT



HPCToolkit

KOJAK



PAPI



scalasca

Other names and brands may be claimed as property of others

Our Approach

- Performance profiling using network and library counters through well-defined SHMEM APIs
- Associate performance information to process level as well as contexts within a process
- Simplest design of collector that impose low overhead to the application runtime
- Profiling analysis and optimization strategies proposed in this work can be applicable for other PGAS models

Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- Experimental Analysis
- Conclusion and Future Work

OpenSHMEM Performance Counters

- Unsigned 64-bit integers
- Follow C language rules for unsigned integer arithmetic
- Monotonically increasing over the duration of program execution
- Incremented 0 or more times by SHMEM operations
 - One single large put operation can be fragmented to several smaller writes
 - Operation performed through shared memory rather than fabric

Implementation in Sandia OpenSHMEM

- Different design choices available for APIs based on operation type, input arguments, and return values
- Two class API for per-context counters
 - Operations reading data from a symmetric object (get, fetch AMO)
 - Operations writing data to a symmetric object (put, non-fetch AMO)
- Each context utilizes
 - Middleware level counters for issued operations
 - Fabric level event counters for completed operations
- Tracks number of fabric operations that have completed in the local memory
 - Associated with local process instead of a particular context

Proposed APIs

```
/* Retrieve write operation counters */  
int shmemx_pcntr_get_issued_write(shmem_ctx_t ctx, uint64_t *cntr_value);  
int shmemx_pcntr_get_completed_write(shmem_ctx_t ctx, uint64_t *cntr_value);  
  
/* Retrieve read operation counters */  
int shmemx_pcntr_get_issued_read(shmem_ctx_t ctx, uint64_t *cntr_value);  
int shmemx_pcntr_get_completed_read(shmem_ctx_t ctx, uint64_t *cntr_value);  
  
/* Retrieve target operation counters */  
int shmemx_pcntr_get_completed_target(uint64_t *cntr_value);  
  
/* Retrieve all operation counters */  
int shmemx_pcntr_get_all(shmem_ctx_t ctx, shmemx_pcntr_t *pcntr);
```

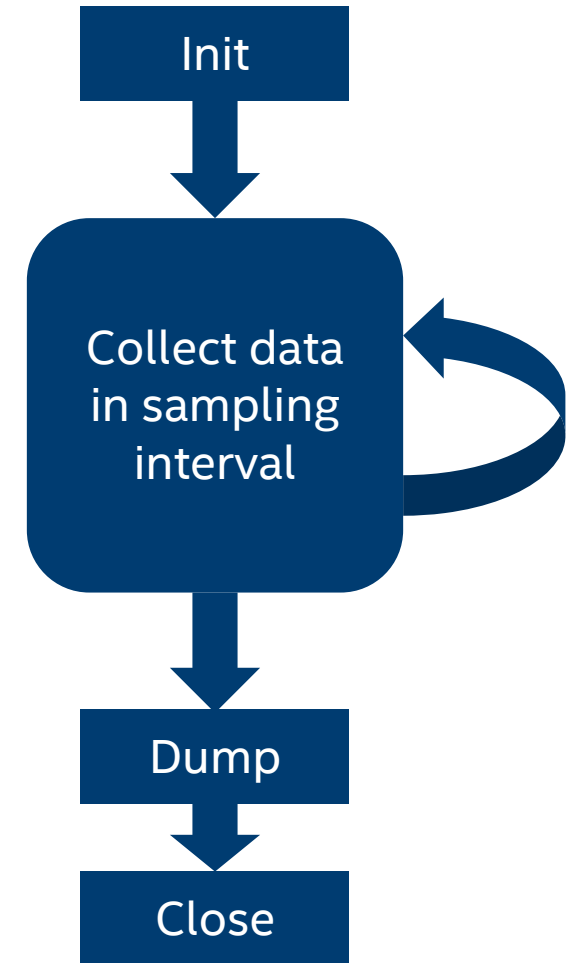
Object to hold all counter values

Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- Experimental Analysis
- Conclusion and Future Work

Design of a Collector

- Simple design choices to collect the performance counter data
- Initiated as a thread to run alongside with the calling PE
- Sampling Interval defines the rate at which the data is collected; controlled by a runtime parameter
- Samples are timestamped and stored in memory
- Samples are discarded when there has been no change in the counter values since the last collection
- By default, collects the data for `SHMEM_CTX_DEFAULT`
- Additional contexts can be added and removed during runtime; maximum number of allowable contexts can be controlled via a runtime parameter
- Stored samples are dumped in a simple CSV format



Example program utilizing the Collector

```
shmem_init();  
...  
shmem_ctx_create(&ctx,  
for (rem_pe = 0; rem_pe < n_pes; rem_pe++) {  
    shmem_ctx_put(ctx, dest,  
src, nelems, rem_pe);  
}  
...  
shmem_finalize();
```

Contexts must not be
created with
SHMEM_CTX_PRIVATE/
SHMEM_CTX_SERIALIZED



```
shmem_init_thread(SHMEM_THREAD_  
MULTIPLE, &tl);  
start_collect();  
shmem_ctx_create(&ctx);  
register_context(ctx);  
for (rem_pe = 0; rem_pe < n_pes;  
rem_pe++) {  
    shmem_ctx_put (ctx, dest,  
src,          nelems, rem_pe);  
}  
remove_context(ctx);  
stop_collect();  
shmem_finalize();
```

Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- **Experimental Analysis**
- **Conclusion and Future Work**

Experimental Setup

- Cluster with 14 compute nodes
 - Intel® Xeon® E5-2699 V3, 36 cores/node @ 2.3 GHz
 - 64 GB DDR4 memory
 - Intel® Omni-Path Fabric
- Performance counter APIs are implemented on top of Sandia OpenSHMEM (SOS), git #908682ee
- Applications
 - Integer Sort (ISx)
 - Stencil from Parallel Research Kernel (PRK)

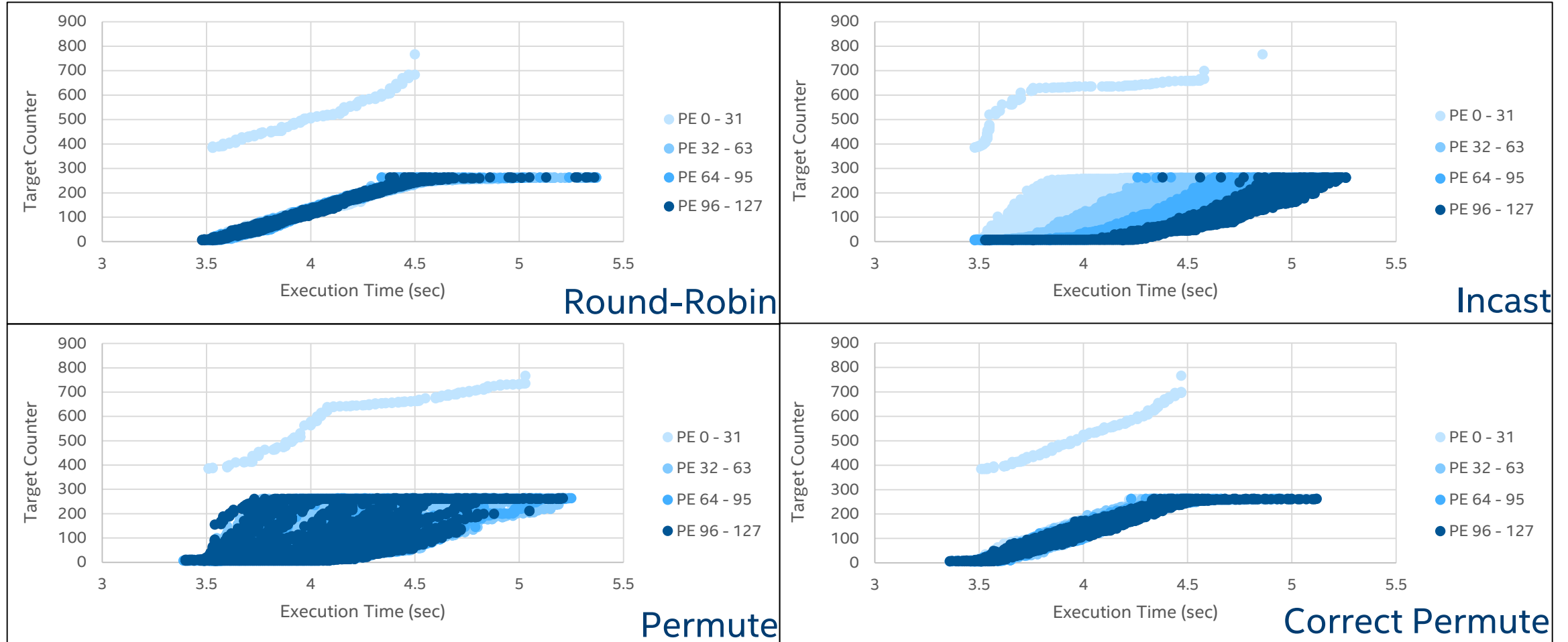
Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- **Experimental Analysis**
 - **Communication Schedule**
 - Overlap
 - Load Balance
 - Weak Scale Analysis
 - Overhead
- **Conclusion and Future Work**

Communication Schedule

- Defines the next target PE in an all-to-all key exchange for each PE
- ISx implements three different communication scheduling pattern
 - Round-Robin (default): Chooses the next PE based on the given PE's rank and loops over a circular array of PEs
 - Incast: Iterates over an array of PEs from $0, 1, 2, \dots, n$
 - Permute: Iterates over a random array of PEs
- Target counter progression follows different trend for different communication schedule
- Divide the total number of PEs into four groups to highlight the differences in progression

Communication Schedule



Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

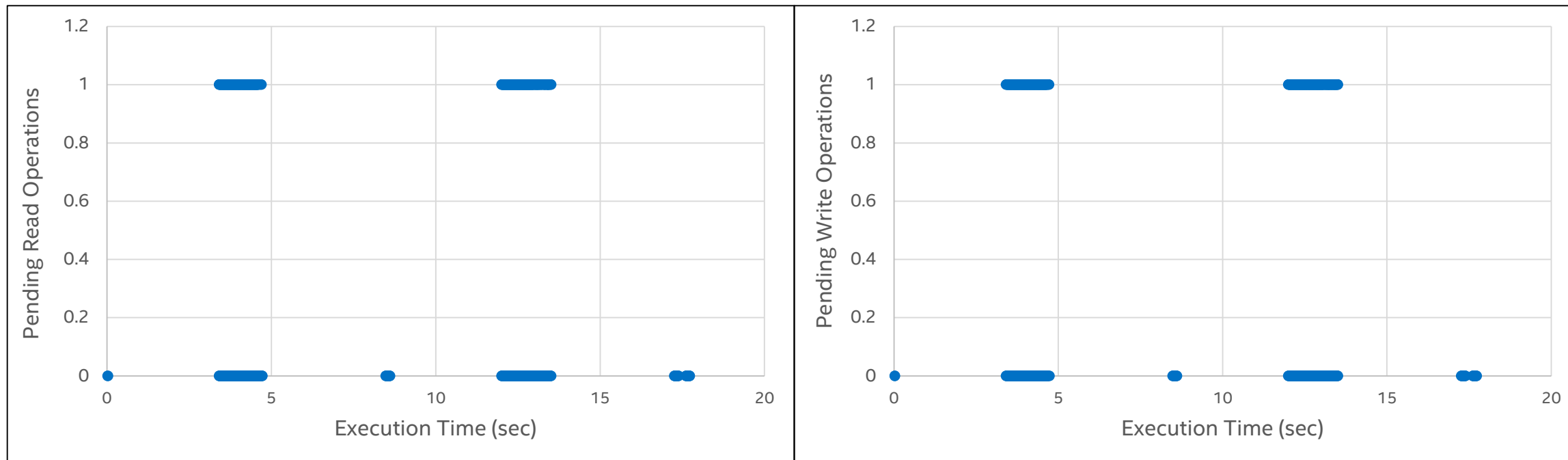
Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- **Experimental Analysis**
 - Communication Schedule
 - **Overlap**
 - Load Balance
 - Weak Scale Analysis
 - Overhead
- **Conclusion and Future Work**

Overlap

- Observe the dynamic differences between the posted and completed operation counters
- Analyze the opportunities to introduce communication overlap
- Use ISx for this analysis and apply different optimization strategies based on the counter values
- Focus on the counter changes in the key exchange routine through
 - Pending read/write operations
 - Issued write operations w.r.t. completed read operations

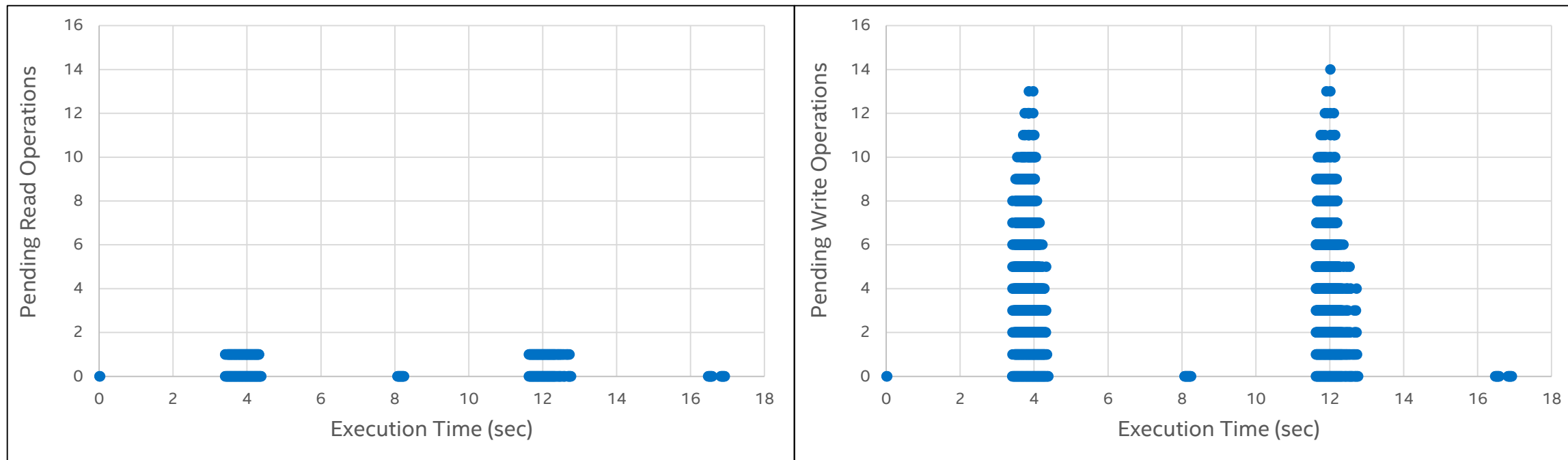
Pending Operations with default ISx



- Pending operation counters (difference between the issued and completed counter values) over execution time
- Both read (left) and write (right) counter values reveal only one pending operation at any given time – presenting the opportunity for the usage of non-blocking APIs

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Pending Operations with Non-blocking Put



- Replace Blocking Put with Non-blocking API
- Pending read operations are unchanged as no overlap is introduced; Pending write operations increase to at most 14 during the key exchange execution
- Further overlap is possible through non-blocking read (non-blocking `fadd AMO`)

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Non-blocking AMO in ISx key exchange

```
for (int i=0; i<shmem_n_pes(); i++) {  
    int dest_pe = peers_iter(i);  
    long long dest_offset = shmem_longlong_atomic_fetch_add(  
        &bucket_offset, bucket_sizes[dest_pe], dest_pe);  
    shmem_int_put_nb(&bucket_keys[dest_offset], ... , dest_pe);  
}
```

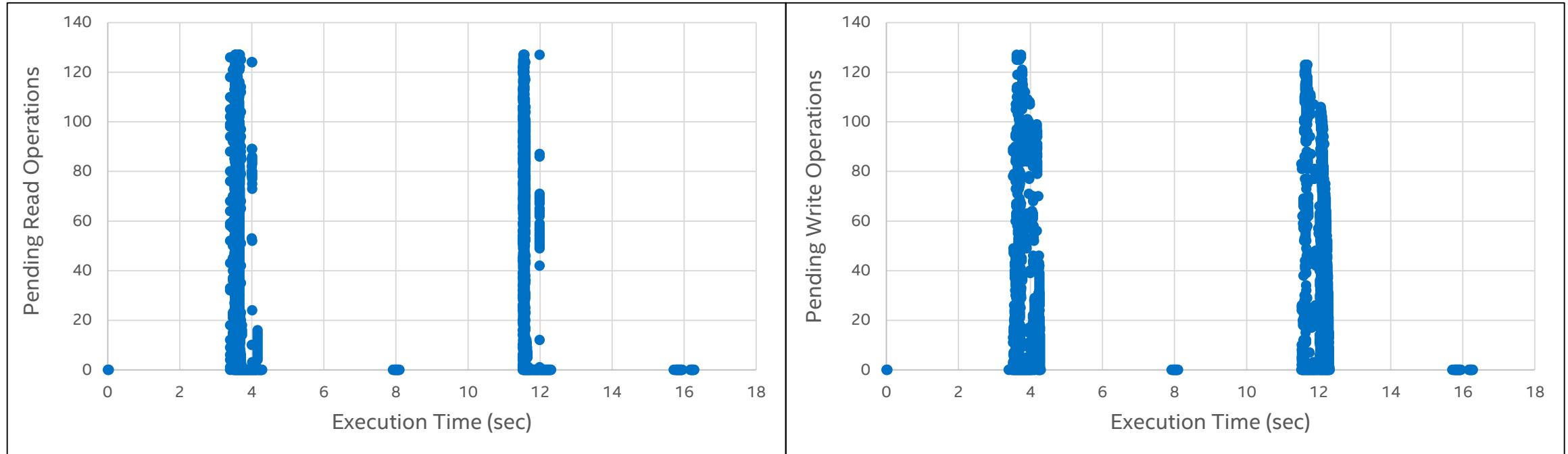


Loop fission

```
for (int i=0; i<shmem_n_pes(); i++) {  
    int dest_pe = peers_iter(i);  
    shmem_longlong_atomic_fetch_add_nbi(&bucket_offset,  
        bucket_sizes[dest_pe], &dest_offsets[dest_pe], dest_pe);  
}  
shmem_quiet();  
for (int i=0; i<shmem_n_pes(); i++) {  
    int dest_pe = peers_iter(i);  
    shmem_int_put_nb(&bucket_keys[dest_offsets[dest_pe]], ..., dest_pe);  
}
```

shmem_quiet()
ensures the
completion of
all fetches

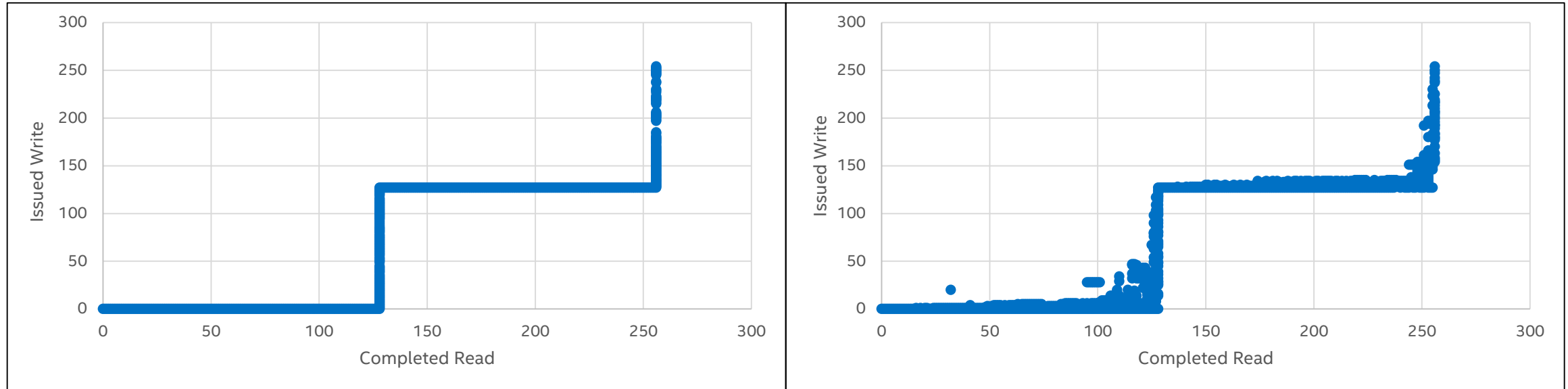
Pending Operations with Non-blocking AMO



- Both pending read and write operations increase to almost 128 (total number of PE)
- Loop fission ensures read and write operations overlap within themselves
- `shmem_quiet` ensures the completion of all fetches, but prevents any overlap between read and write

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Overlap between Read and Write



- Plot issued write v/s. completed read to present any overlap between read and write
- In both iterations, write and read progress independently and thus, no overlap
- Replace `shmem_quiet` with individual `wait_until` to wait for each non-blocking `fadd` to complete before invoking the corresponding `shmem_put`
- Both the iterations exhibit overlap between read and write with `wait_until` at the end of the loop execution

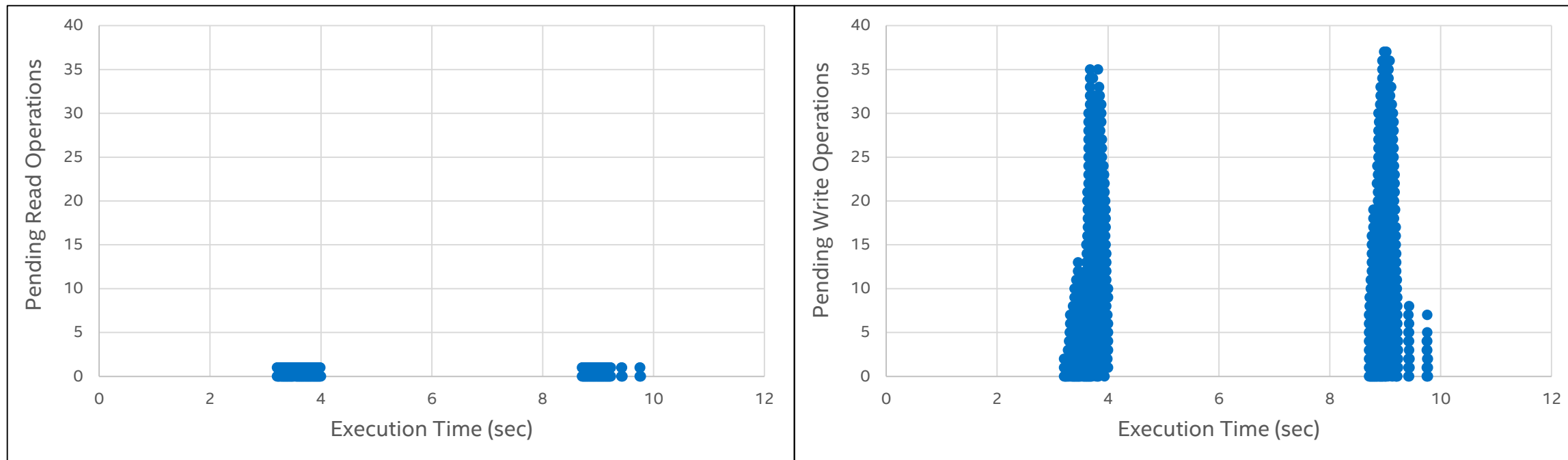
Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Overlap with Threads

- Alternative approach to pipeline the all-to-all exchange
 - Distribute loop iterations to multiple threads
 - Launch the threads in parallel
- Use OpenMP threads to the key-exchange routine
 - Create a pool of contexts to be used by different threads
 - Each thread utilizes it's own context to invoke SHMEM APIs
- Apply OpenMP threads on both implementations of ISx (with and without loop fission)

```
#pragma omp parallel num_threads(T) {  
    int thread_id = omp_get_thread_num();  
    int PEs_per_thread = shmem_n_pes()/T;  
    for (int i = thread_id * PEs_per_thread; i < (thread_id + 1) * PEs_per_thread; i++) {  
        int dest_pe = peers_iter(i);  
        long long dest_offset = shmem_longlong_atomic_fetch_add(ctx_pool[thread_id],  
            &bucket_offset, bucket_sizes[dest_pe], dest_pe);  
        shmem_int_put_nb(ctx_pool[thread_id], &bucket_keys[dest_offset], ... , dest_pe);  
    }  
}
```

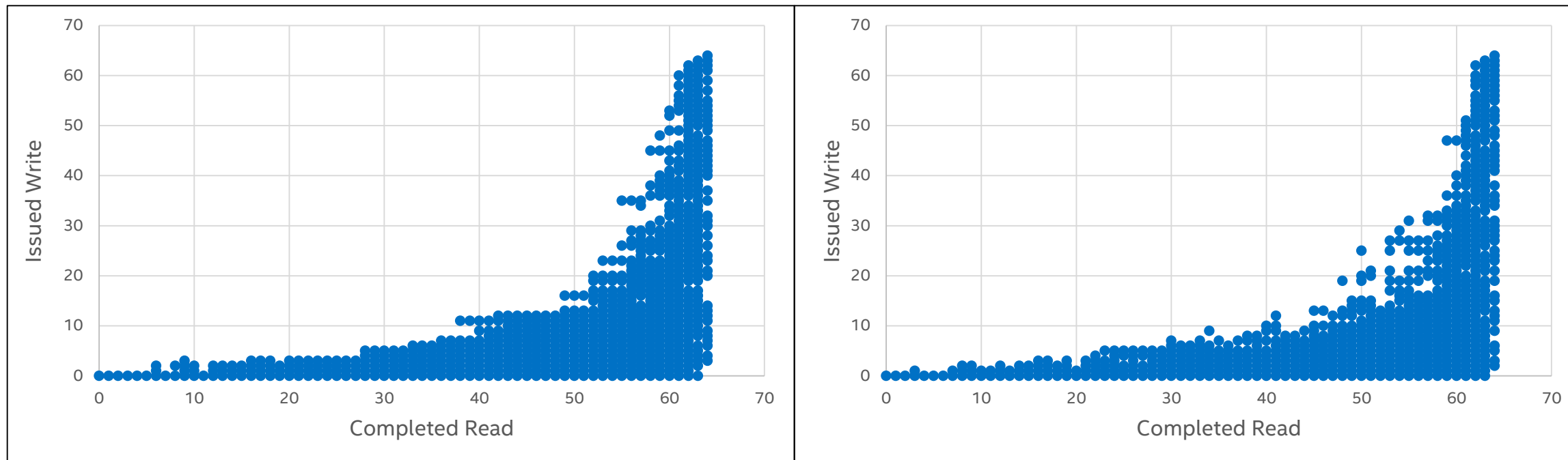
Pending Operations with Non-blocking Put and Two Threads



- Pending read operations are similar to the non-threaded implementation as they use the blocking API; per thread, it does not increase more than once
- Pending write operations increase more than that of non-threaded implementation; with multiple threads, overlapping among different write operations can be increased

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Overlap with Non-blocking Put, AMO and Two Threads



- Apply OpenMP threads on the two distributed loops of key exchange with wait-until
- Both warm-up and trial iterations exhibit more overlap with multiple threads
- Increased pipelining between read and write operations compared to the non-threaded implementation

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

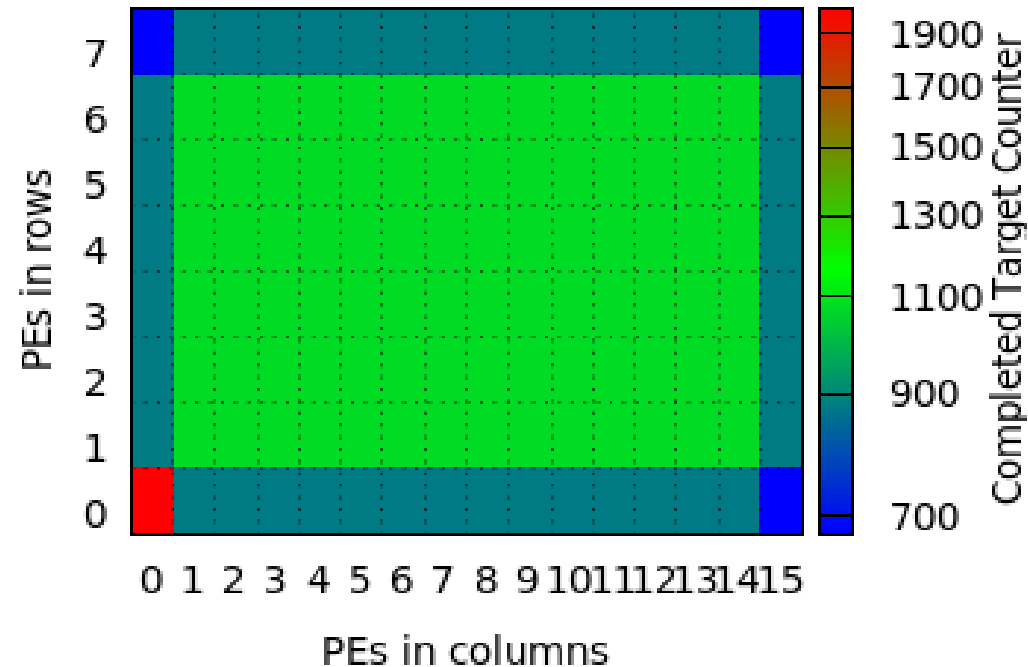
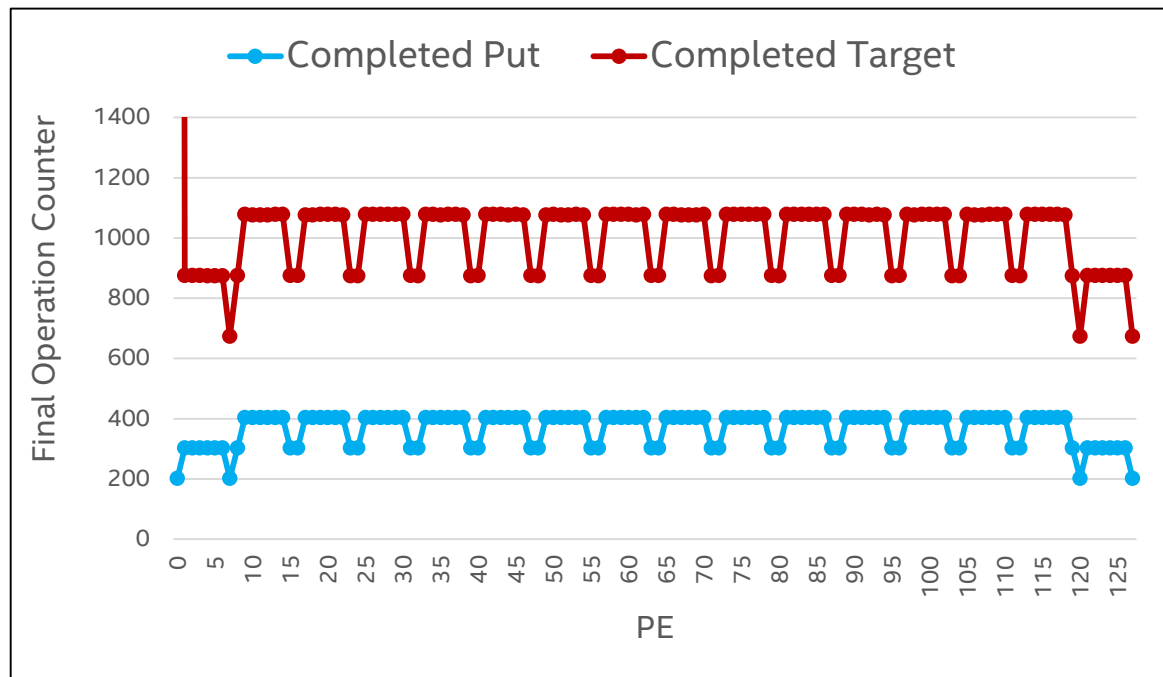
Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- **Experimental Analysis**
 - Communication Schedule
 - Overlap
 - **Load Balance**
 - Weak Scale Analysis
 - Overhead
- **Conclusion and Future Work**

Load Balance

- Utilize performance counters to detect load balance across all PEs
- Focus on the final operation counter value
- Use Stencil kernel (128 PEs with grid size of 1000 and 100 iterations)
- Observe put and get counters as well as target counter

Load Balance



- Grid of PEs with 8 rows and 16 columns; PEs with less neighbors (edge) have less load compared to the PEs with more neighbors (inner)
- Both Put and Target counter exhibit the load imbalance for stencil
- PE0 has a high target counter value because of collective and synchronize operations

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

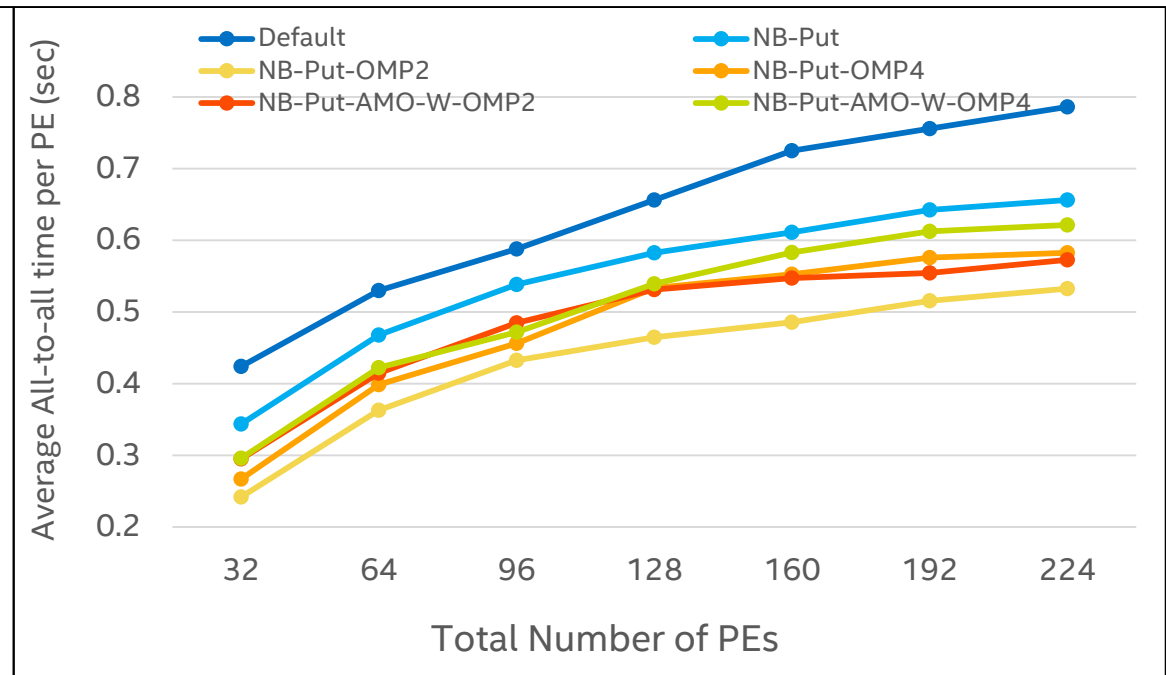
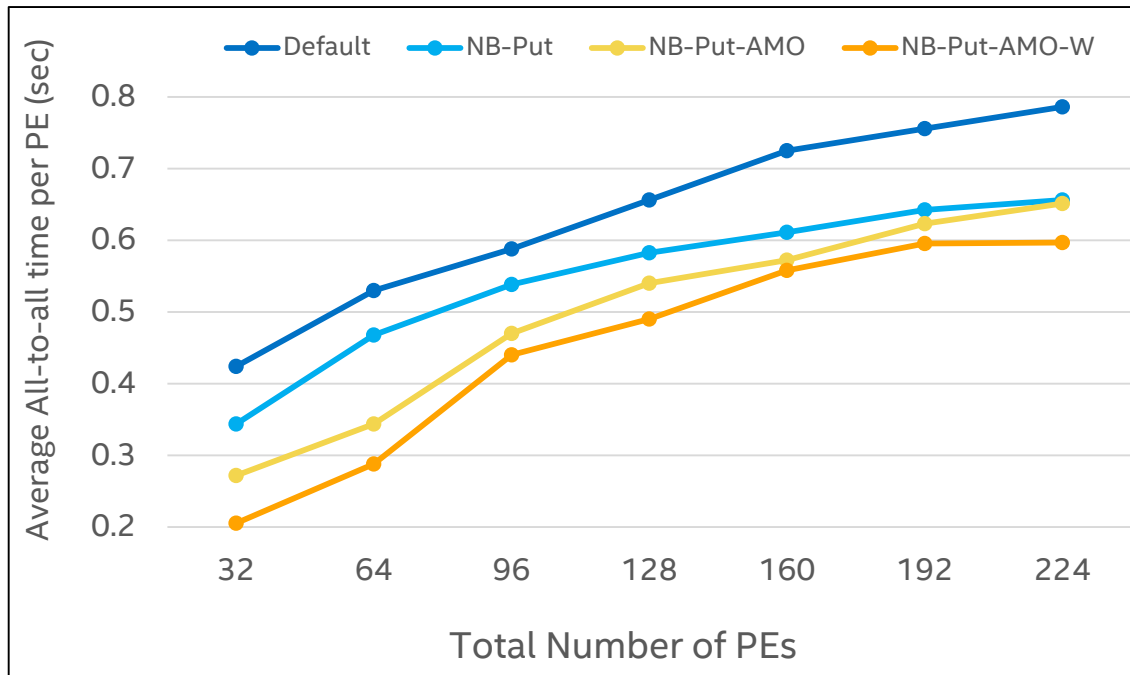
Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- **Experimental Analysis**
 - Communication Schedule
 - Overlap
 - Load Balance
 - **Weak Scale Analysis**
 - Overhead
- **Conclusion and Future Work**

Different ISx Implementations

	ISx Implementation	Description
Non-threaded	Default	Default implementation with blocking Put and AMO
	NB-Put	Implementation with non-blocking Put
	NB-Put-AMO	Implementation with non-blocking Put and AMO in two distributed loops using <code>shmem_quiet</code> in between
	NB-Put-AMO-W	Implementation with non-blocking Put and AMO in two distributed loops using <code>shmem_wait_until</code>
Threaded	NB-Put-OMP2	Implementation with non-blocking Put and 2 OpenMP threads
	NB-Put-OMP4	Implementation with non-blocking Put and 4 OpenMP threads
	NB-Put-AMO-W-OMP2	Implementation with non-blocking Put and AMO in two distributed loops using <code>shmem_wait_until</code> and 2 OMP threads
	NB-Put-AMO-W-OMP4	Implementation with non-blocking Put and AMO in two distributed loops using <code>shmem_wait_until</code> and 4 OMP threads

Weak Scale Analysis



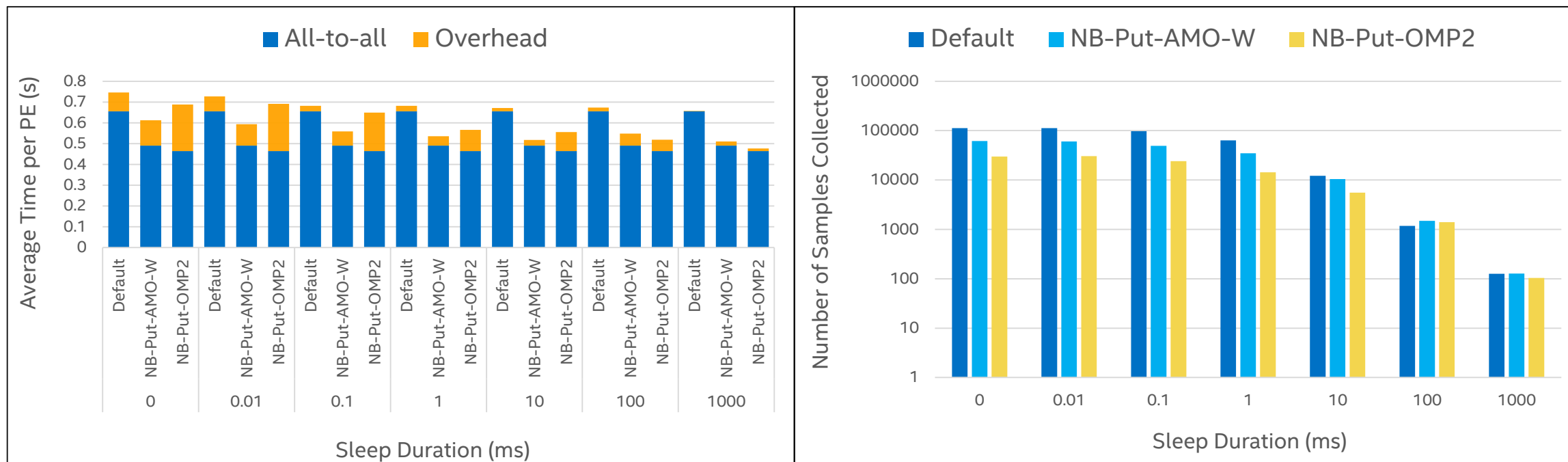
- Comparison of non-threaded implementations in 2 to 14 nodes (16 PEs/node)
- NB-Put achieves **16.5%** benefit compared to the Default; NB-Put-AMO-W out-performs NB-Put-AMO by **8.3%**
- Comparison of threaded implementations in 2 to 14 nodes (16 PEs/node) with 2, 4 threads
- NB-Put-OMP2 achieves **10%** benefit compared to the single-threaded NB-Put-AMO-W; Additional threads degrade performance

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- **Experimental Analysis**
 - Communication Schedule
 - Overlap
 - Load Balance
 - Weak Scale Analysis
 - **Overhead**
- **Conclusion and Future Work**

Collector Overhead



- Analysis on three different implementations based on different optimization choices
- Observe 20-100 ms overhead in average all-to-all time per PE for NB-Put-AMO-W
- Additional overheads for threaded implementation, NB-Put-OMP2
- Can collect reasonable number of samples with a sleep duration of 0.1 ms

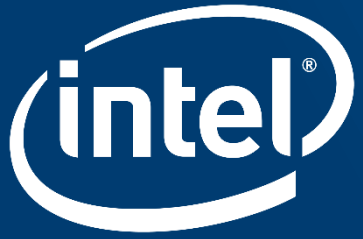
Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>. Copyright © 2018, Intel Corporation. *Other names and brands may be claimed as the property of others.

Outline

- Introduction and Motivation
- Existing Approaches
- Performance Counter APIs
- Design and Implementation of a Collector
- Experimental Analysis
- **Conclusion and Future Work**

Conclusion and Future Work

- Proposed a performance counter API extension to OpenSHMEM specification
- Implemented the APIs in Sandia OpenSHMEM library
- Designed and implemented a low-overhead collector to use these APIs
- Analyzed applications with the performance counters to
 - Reveal and fix implementation bug in communication scheduling
 - Characterize load balance
 - Identify opportunities to improve pipelining and overlapping deficiencies
- Proposed approaches improve the average all-to-all time for ISx by 30%
- Investigation on automated methods for analyzing the collected data
- Use performance counter APIs to aid developers of recent and proposed API extensions
- Identify system-level performance optimization opportunities



Questions?