One-Sided Append: A New Communication Paradigm For PGAS Models

Jim Dinan and Mario Flajslik

OpenSHMEM User Group ‘14

Legal Disclaimer & Optimization Notice

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.

Optimization Notice

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.

Notice revision #20110804

NAS Integer Sort (IS) Key Exchange

```
int dst_buf[DST_BUF_SIZE]; /* Symmetric buffer for RX of keys */
int counter = 0;           /* Symmetric integer offset counter */
...
for (i = 0; i < num_pes; i++) {
  int dst_off;

  dst_off = shmem_int_fadd(&counter, cnt[i], i);
  shmem_int_put(dst_buf+dsf_off, src_buf+src_off[i], cnt[i], i);
}
```

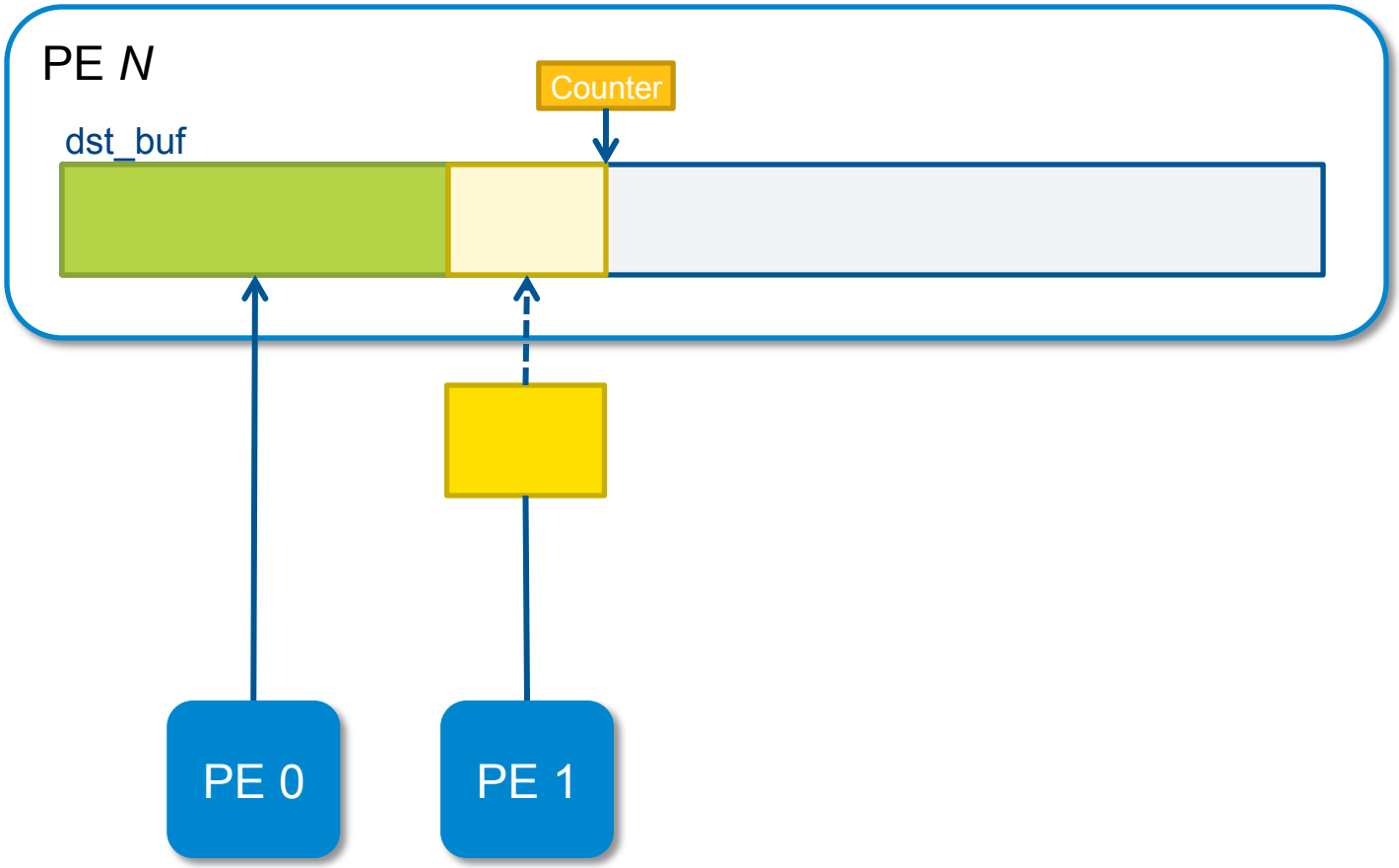
Key-exchange portion of integer bucket sort algorithm

Receiver's buffer management is done by the sender

- Requires additional fetch-add message from the sender
- Alternative is to pre-arrange buffer space

Sidebar: Iterations can be overlapped with nonblocking ops

One-Sided Append in Key Exchange



SHMEM_PUSH Interface

```
/* SHMEM Offset Counter (OCT) object management (collective) */
void shmem_oct_create(shmem_oct_t *oct, void *buffer, size_t len);
void shmem_oct_destroy(shmem_oct_t *oct);

/* Appending put, a.k.a. "push", one-sided communication */
void shmem_push(shmem_oct_t oct, const void *src_buffer,
                size_t len, int pe);

/* Offset counter update/query routines (local) */
void shmem_oct_reset(shmem_oct_t oct);
size_t shmem_oct_get(shmem_oct_t oct);
```

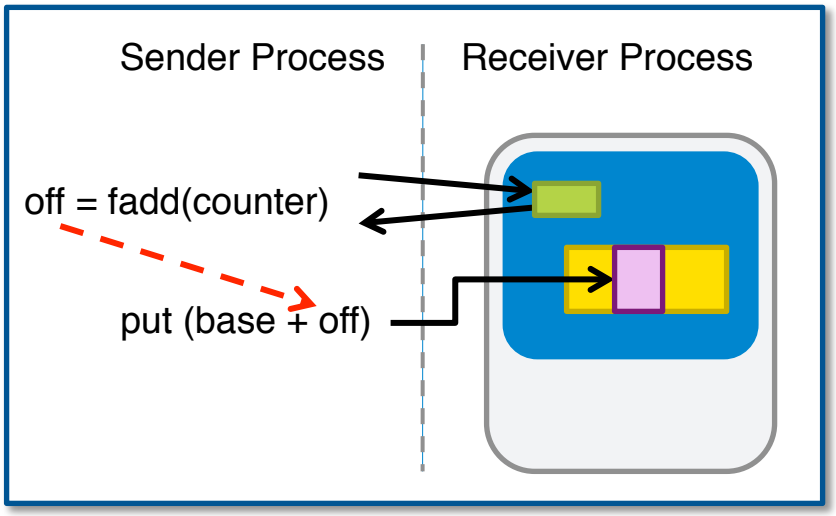
Push: One-sided operation appends sender's data to receiver's buffer

Introduces "offset counter", which is atomically updated during push

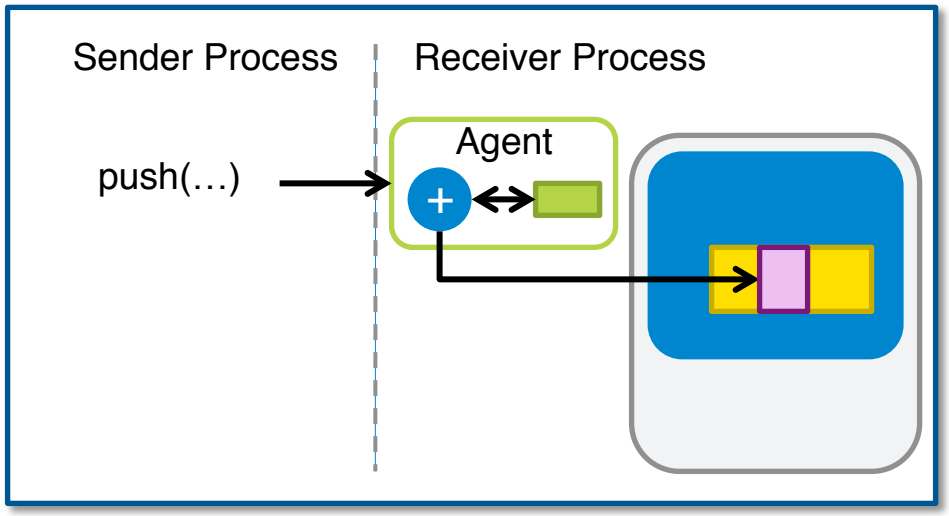
Can be used in conjunction with other proposed extensions:

- Contexts, counting puts, nonblocking communication, …

Sender vs. Receiver-Managed Approach



(a) Implementation on OpenSHMEM 1.1



(b) Receiver-Managed Implementation

Fetch-add (a) implementation has a data dependency at the sender

- Takes a network round-trip to resolve

Receiver-managed implementation is “fire-and-forget”

- Utilized “agent” (hardware/software) at receiver side to calculate dest. location

NAS IS Key Exchange (Push)

```
int dst_buf[DST_BUF_SIZE]; /* Symmetric buffer for RX of keys */
shmem_oct_create(&keys_oct, dst_buf, DST_BUF_SIZE);
...
for (i = 0; i < num_pes; i++) {
  shmem_int_push(keys_oct, src_buf+src_off[i], cnt[i], i);
}
...
shmem_oct_free(&keys_oct);
```

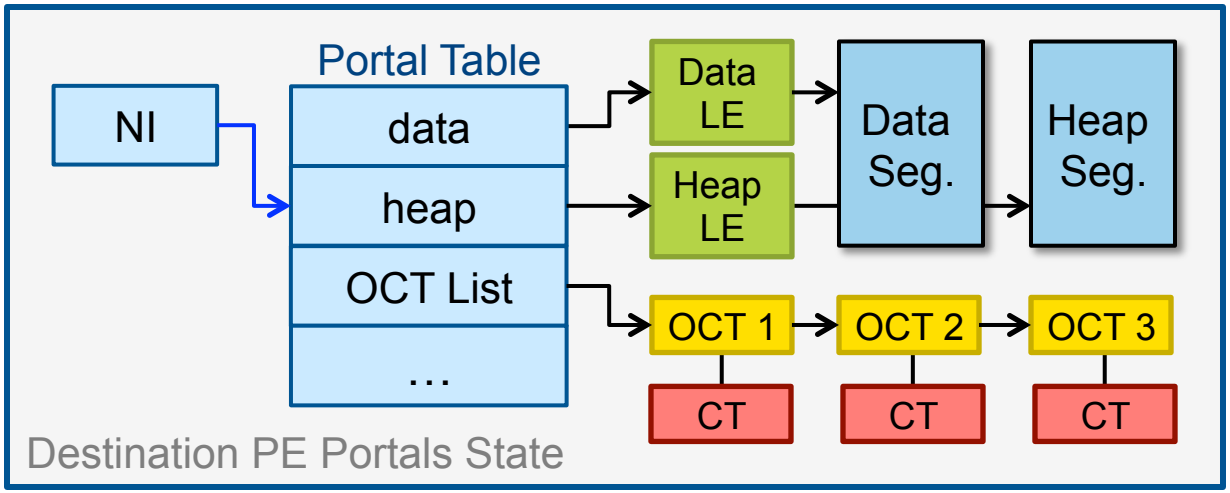
Key-exchange portion of integer bucket sort algorithm

Destination buffer addressed using the offset counter (OCT)

- Allows receiver-managed implementation
- Receiver can calculate effective target address

Sidebar: Iterations can be overlapped with nonblocking ops

Implementation of SHMEM_PUSH



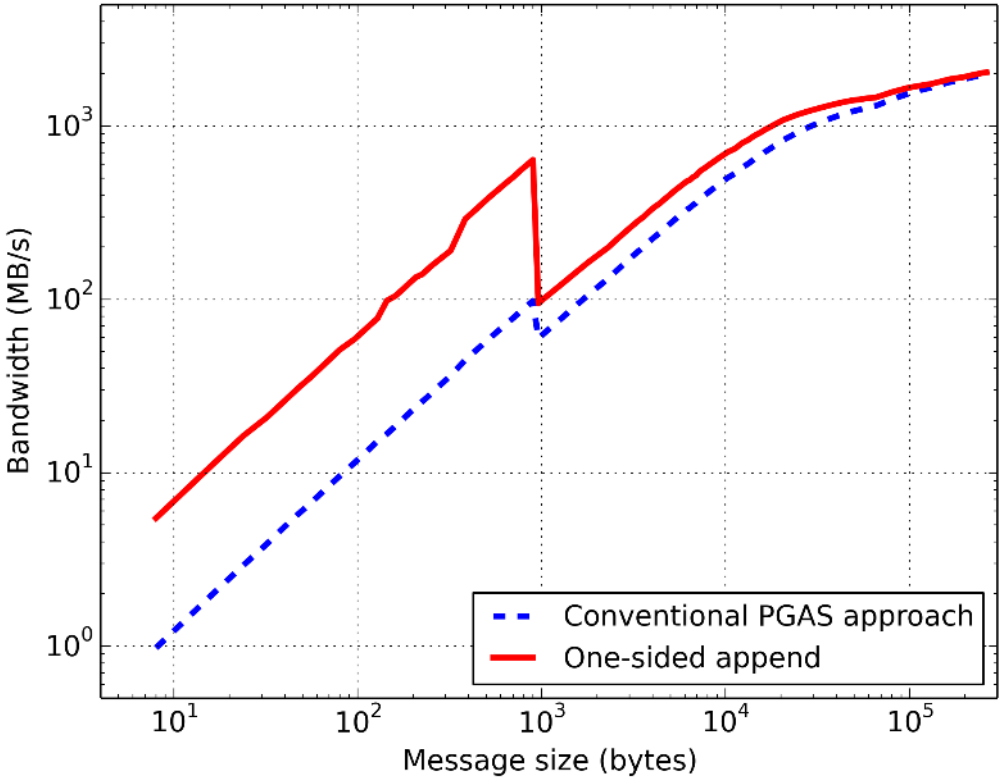
Can be implemented on top of OpenSHMEM 1.1

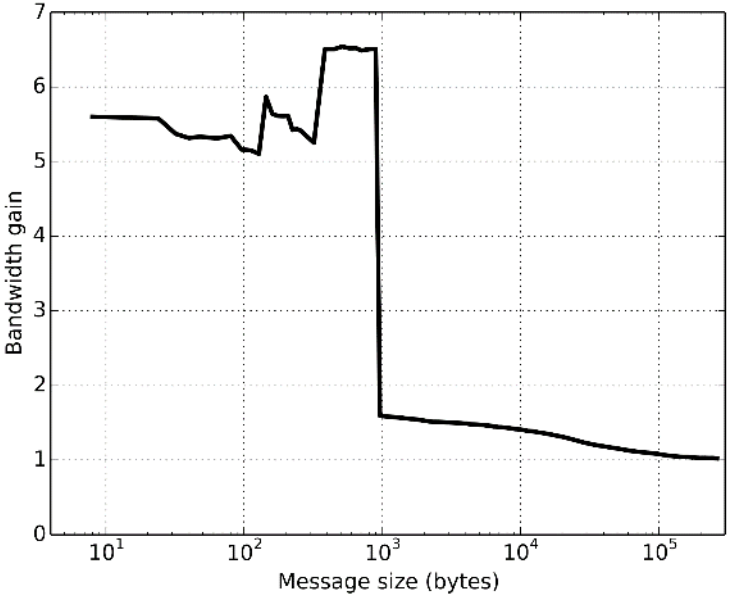
- Symmetric integer used as offset counter (OCT)
- Fetch-add of OCT followed by put

Implementation on Portals 4

1. Append buffer as new match entry (ME) on OCT match list
2. Match bits are selected to globally and uniquely identify the append buffer
3. Set locally managed offset flag on ME (PTL_ME_MANAGE LOCAL)
4. Portals counter attached to ME to expose Portals offset counter

Early Perf. Evaluation of Key Exchange





InfiniBand® Cluster: 16 nodes, 16 PEs; OpenSHMEM over Portals 4 over IB

Drop-off at Portals *max_volatile_size* (buffering converts blocking to NB)

Concluding Discussion

Proposed SHMEM_PUSH extension

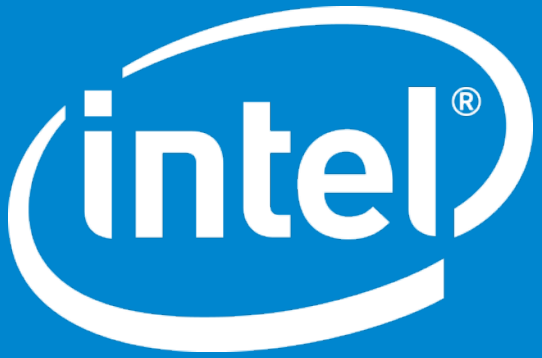
- Can accelerate appending data to a remote buffer
- Enables runtime designers to optimize comm. pattern
  - E.g. through a receiver-managed implementation

Discuss synchronization: How do users consume this data?

- Offset counter tells us next free location
  - Does not guarantee that data has arrived
- Can combine with counting puts variation that counts bytes
  - Waiting for offset and received-bytes to match guarantees completion
- Currently, programmers would use barrier or flag variables

Discuss: Can this be done efficiently in MPI?

- Probe followed by receive or fetch-add and put


intel®