Reducing Synchronization Overhead Through Bundled Communication

James Dinan, Clement Cole, Gabriele Jost, Stan Smith, Keith Underwood, and Robert W. Wisniewski

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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

SHMEM Synchronization Gap

Consistent Barrier

```
for (pe = 0; pe < NPES; pe++)
  shmem_putmem(data, PE);

shmem_barrier_all();
```

Point-to-Point Flags

```
for (pe = 0; pe < NPES; pe++)
  shmem_putmem(data, PE);

shmem_fence();

for (pe = 0; pe < NPES; pe++)
  shmem_int_add(flag, -1, PE);

shmem_int_wait_until(flag, EQ, 0);
```

Strong ordering semantic

- Remote completion of all operations

Globally synchronizes PEs

- Ensure next phase can reuse buffers
- Can be expensive
- Load Imbalance, noise, etc.

Point-to-point ordering

- Weaker than remote completion

Point-to-point synchronization

- O(P) messages, vs O(log P) for barrier

Comm/sync are split up

- Limits optimizations in the runtime

Bundling Communication and Synchronization

```
shmem_ct_create(&ct);

for (pe = 0; pe < NPES; pe++)
  shmem_put_ct(ct, data, …, PE);

shmem_ct_wait(ct, NPES);
```

Bundle comm. and synchronization together in a single operation

- Counter is incremented at the target after the operation has completed
- Weak counting semantic: the receiver can do the increment in get/wait

Bundling enables implementation optimizations

- Leverage hardware capabilities (ordering, bundling, events, …)

Enables a receiver-managed implementation

- Can significantly reduce communication involved in synchronization

Counting Puts Interface

```
void shmem_ct_create(shmem_ct_t *ct);
void shmem_ct_free(shmem_ct_t *ct);

long shmem_ct_get(shmem_ct_t ct);
void shmem_ct_set(shmem_ct_t ct, long value);
void shmem_ct_wait(shmem_ct_t ct, long greater_or_equal_val);

void shmem_putmem_ct(shmem_ct_t ct, void *target, …, int pe);
...
```

Creation / free is collective

- Every PE needs a handle to refer to the counter
- Individual counter created on each PE

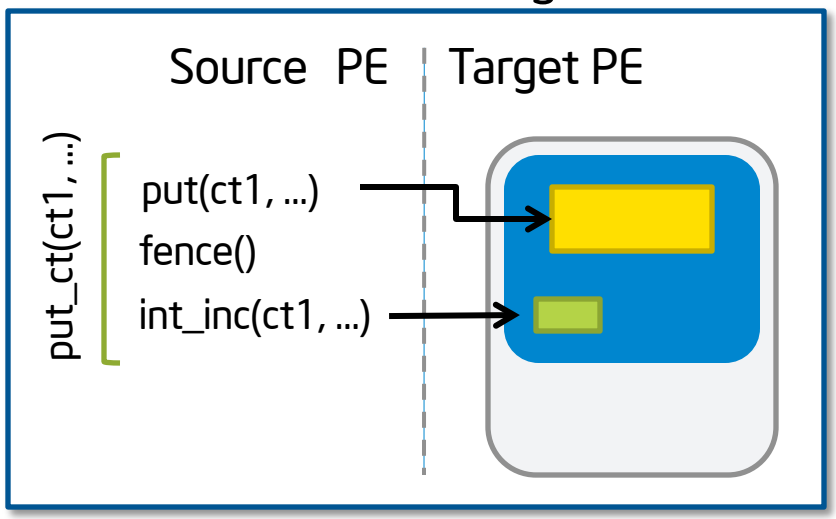
Query functions are local

- Read, write, or wait on a counter
- Counter updates can be delayed until query

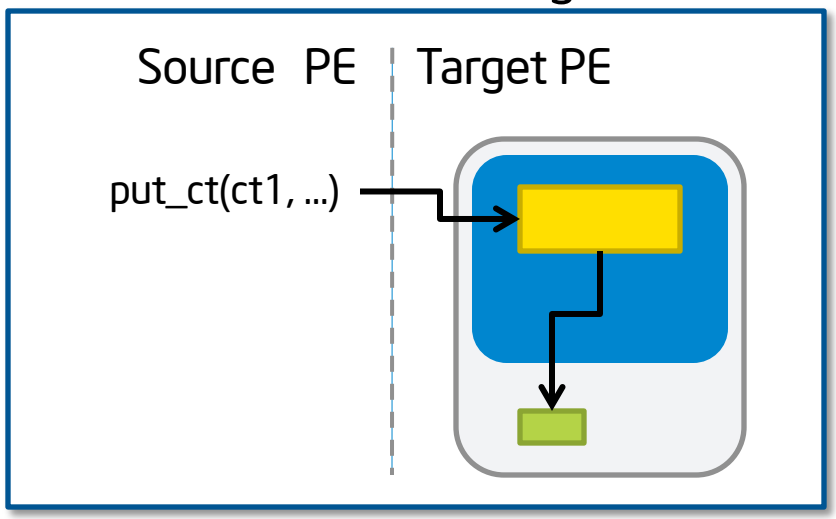
Add “counting” versions of one-sided communication operations

Counting Puts Implementation

Sender-Managed



Receiver-Managed



Implementation on top of SHMEM

- Put, fence, increment flag

Shared memory implementation

- Similar to implementation on SHMEM
- Copy, membar, atomic increment flag

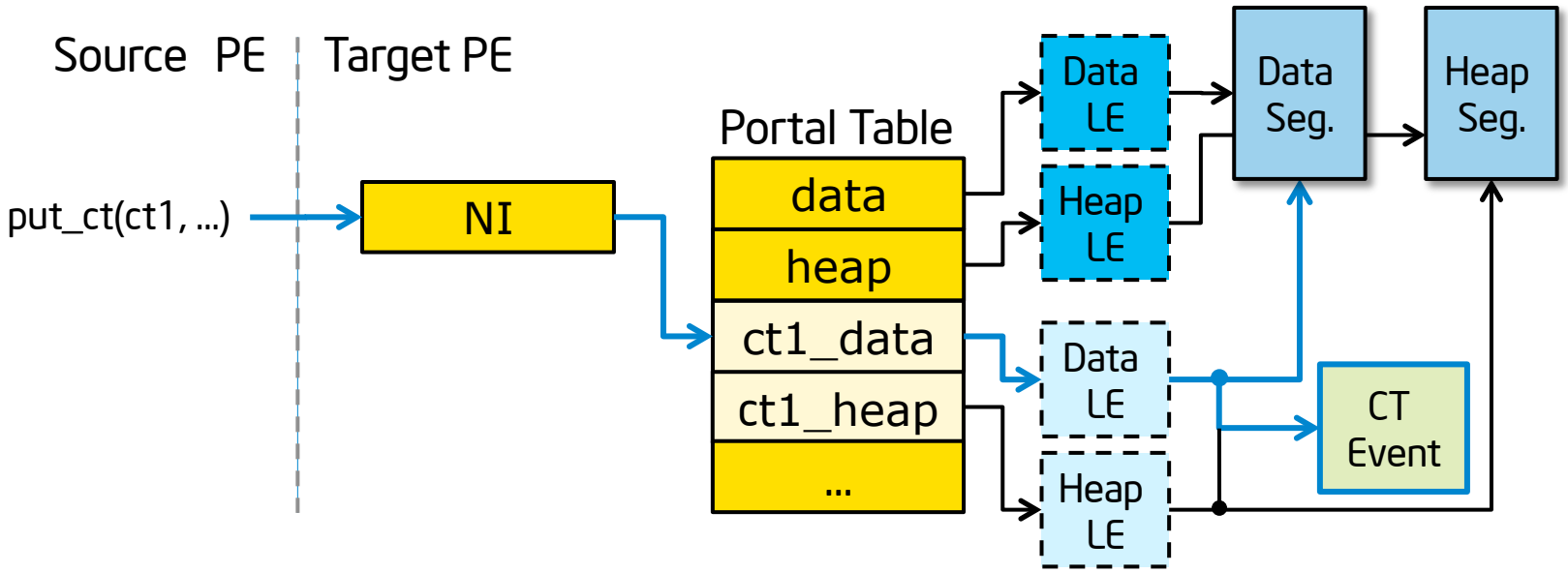
Use communication events

- Portals network – counting events, counter triggered by communication
- Other networks – completion events, update counter in query function

Piggyback info in message header

- UDP, PSM, etc...

Signaling Puts in Portals SHMEM



Create separate portal table entries for each counter

- Non-matching entries, list entry (LE) points to memory segment
- Isolates counters from each other

Portals counting event is attached to LE on counting PTE

- Automatically incremented when the operation completes

Direct counting puts to corresponding CT PTE, others to generic PTE

Empirical Evaluation

Implemented and available in Portals-SHMEM

- http://code.google.com/p/portals-shmem/

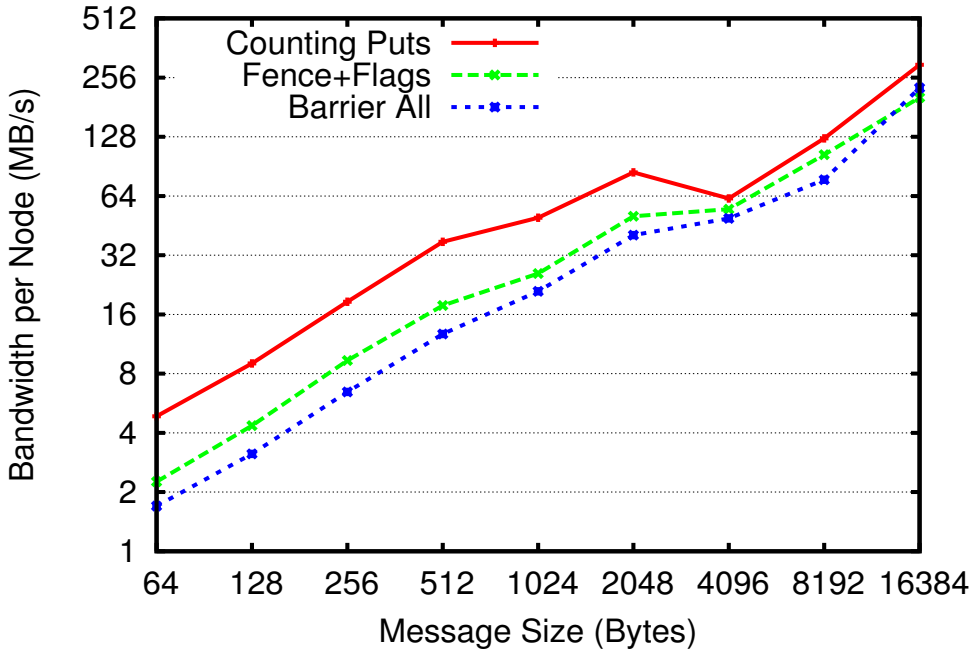
Evaluation system:

- Mellanox QDR InfiniBand, single switch
- Intel® Xeon® X5680 x 2, 24 GB memory
- 15 nodes, 12 cores per node = 180 PEs
- Open source Portals-IB, Portals-SHMEM

Benchmarks and highlights:

- All-to-all (bandwidth)
  - 2x bandwidth for small messages
- Ping-pong (latency)
  - ½ latency for small messages
- Pipelined parallel stencil kernel (overlap)
  - More than 2x improvement

All-to-All Bandwidth (180 PEs)



Measure bandwidth achieved in all-to-all

- Bandwidth shown is aggregated per-node / physical network endpoint

Bandwidth improvement of >2x for small messages

- Fence + flags approach sends O(P) additional messages
- Barrier synchronizes all PEs, only as fast as the slowest PE

Large messages amortize cost of sync messages (Amdahl)

Ping-Pong Benchmark

Fence + Flags

Sender

```
shmem_putmem(rcv, snd, msg_len, pe);
shmem_fence();
shmem_int_inc(&flag, target);
```

Receiver

```
shmem_int_wait(&flag, 0);
flag = 0;
```

Counting Puts

Sender

```
shmem_putmem_ct(ct, rcv, snd, msg_len, pe);
```

Receiver

```
shmem_ct_wait(ct, 1);
shmem_ct_set(ct, 0);
```

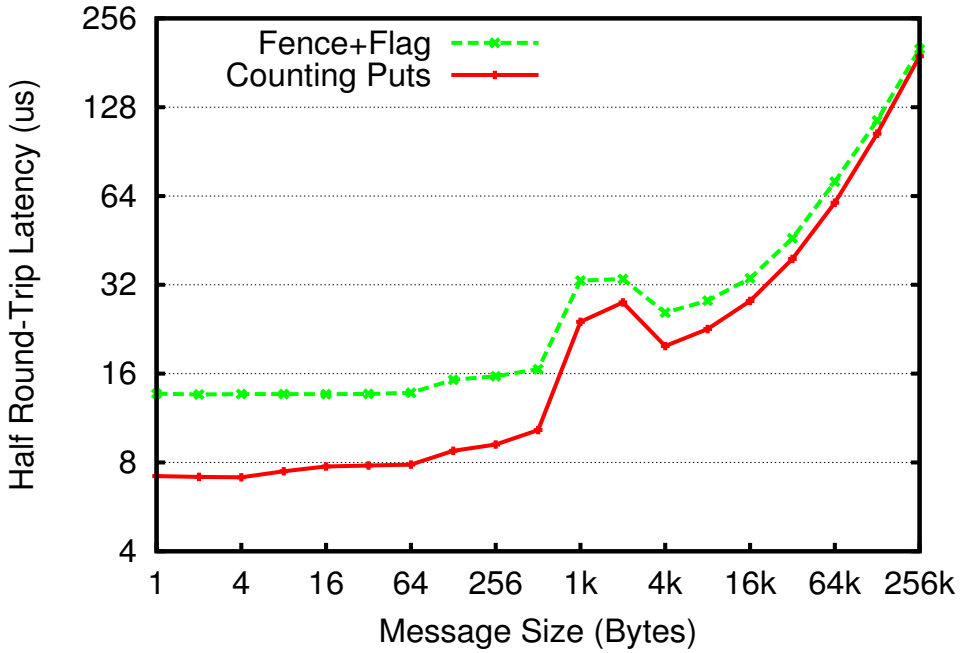
Bundle sender's operations

- Put, fence, and atomic increment are combined
- Weaker than fence, only ensures that this put is visible to receiver

Experiment:

- PEs switch sender/receiver roles every iteration
- Sweep message length parameter

Half Round-Trip Latency (2 PEs)



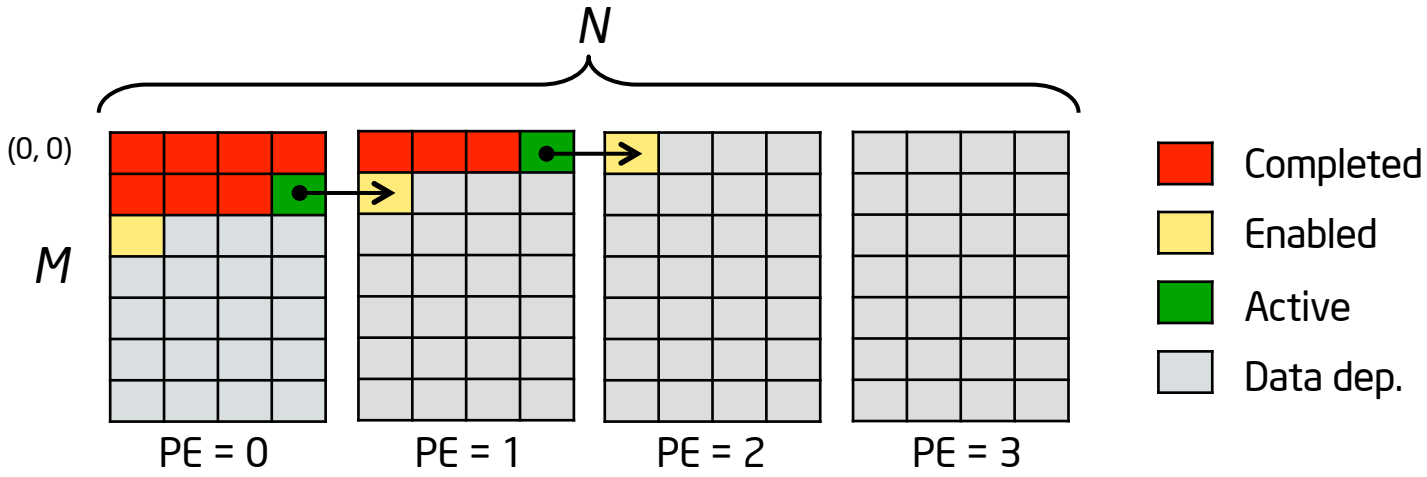
Counting puts eliminate atomic increment message

- Halves latency for small message sizes
- Large messages amortize sync. cost (Amdahl)

Explicit fence is also eliminated

- Removes waiting for completion at the sender
- We can fire and forget the counting put, through a bounce buffer

Pipelined Parallel Stencil Benchmark



Stencil update rule

- A(i, j) = A(i-1, j) + A(i, j-1) - A(i-1, j-1)

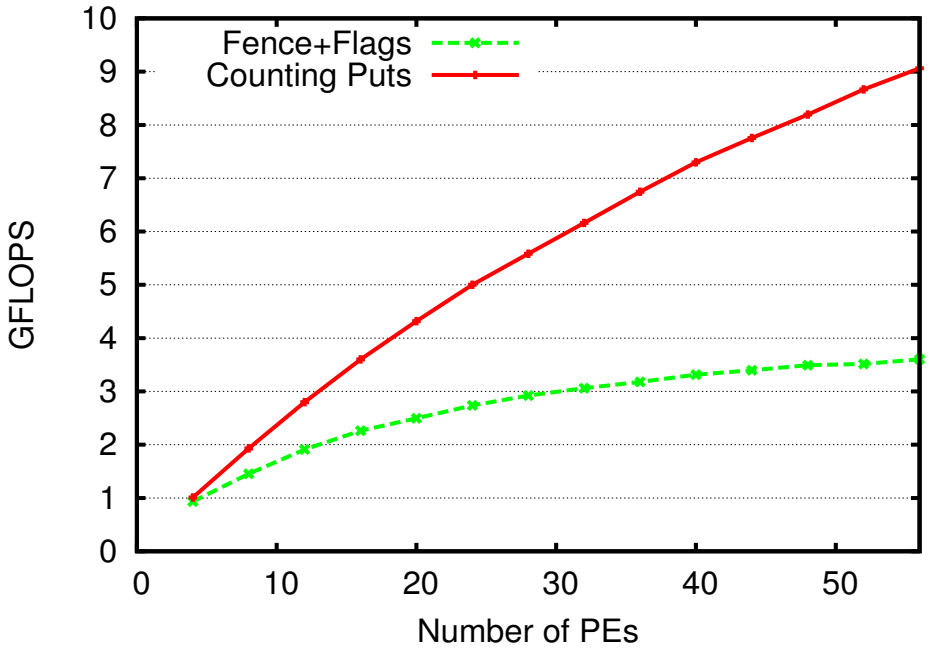
Domain decomposition along j-dimension

- Data dependence – West, North, Northwest
- Pipelined parallelism
- Put data to neighbor's ghost cell and notify them

Intel Parallel Research Kernels, Synch_p2p benchmark

- Tim Mattson, Rob van der Wijngaart (http://github.com/ParRes)

Sync_P2P Strong Scaling



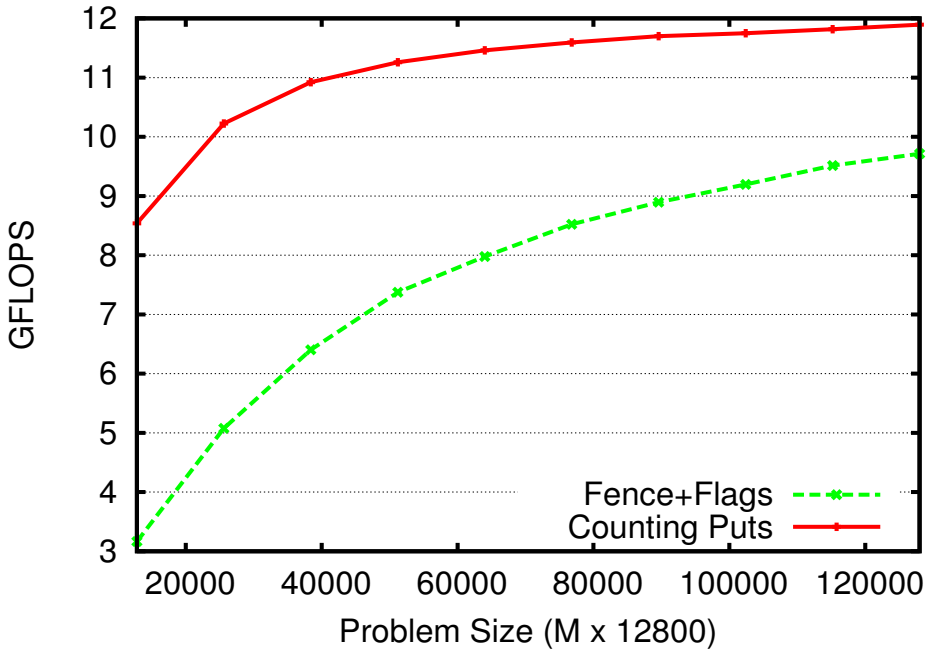
Dataset is *A[M, N]* = 12800 x 1280

- Executed using 4 PEs per node
- GFLOPS = Total FLOPs at all PEs / exec. time (notably comm. time)

Small messages (single element) result in high comm. overheads

- Flags double the number of small messages that are sent
- Benchmark is communication bound on small messages

Synch_P2P Weak Scaling (48 PEs)



Dataset is *A[M, N]* = *M* x 12800

- Run on 4 PEs per node, 48 PEs total

Shows impact of pipelining startup, shutdown latency

- Large values of M amortize this cost
- More efficient point-to-point sync. reduces pipelining overhead

Related Work

UPC semaphores proposal (Bonachea, et al.)

- Similar idea in UPC
- Could use receiver-managed implementations

Split-C signaling store

- Wait for a specific number of bytes to arrive
- One counter per process

MPI point-to-point communication

- Send/recv, data movement and sync. are conjoined

Full/empty bits

- Tera MTA, Cray XMT, Chapel, …

Put-with-flag

- ARMCI_Put_flag(), GASPI write-and-notify

Aggregate synchronization from multiple operations

Synchronize individual operations

SHMEM Synchronization Discussion

Bundling can improve efficiency of point-to-point synchronization

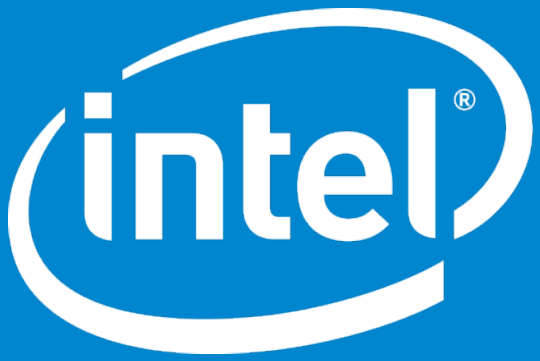
- Enables implementation optimizations
- Leverage hardware capabilities, e.g. communication events

Variety of interfaces for bundled comm. and synchronization

- Opaque flag object
  - Pro: Better enabling of receiver-managed implementations
  - Con: Requires additional API to query remotely
- Integer flag
  - Pro: Fits into existing API, can be queried remotely
  - Con: Restricts implementation options, e.g. hard to use counting events
- Update rule: increment vs. set
  - Increment: aggregates synchronizations to a single object
  - Set: fine-grain, can require many flags, e.g. scalability challenge for all-to-all

Additional synchronization operations for investigation

- Control-only barrier – counting puts already synchronized data
- Split-phase barrier and non-blocking synchronizations


intel®