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.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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
Intel and PGAS

We’ve been busy:

- Looking at PGAS and engaging with this community
- Internalizing PGAS usage models and requirements
- Participating in the development of OpenSHMEM

We aim to:

1. Ensure PGAS models can access to full capabilities of Intel HPC offerings
2. Advocate for needs of PGAS users within Intel
3. Participate in community effort to improve parallel programming models

In this talk:

- Skip the typical boring product spiel
- Give an overview of some recent PGAS work at Intel
Extensions to OpenSHMEM and issues they address:

1. *Contexts*: comm./computation overlap and multithreading
2. *Counting puts*: point-to-point synchronization
3. *One-sided append*: receiver buffer management
4. *Federations*: topology-aware mapping

Open Fabrics Interface:

- New industry standard low-level communication API
- Aligned with PGAS features and performance targets
- Intel is a key partner in industry consortium
Communication Contexts

Enables programmer to express multiple independent streams of communication

Key programming model challenges addressed:
- Improve support for communication/computation overlap
- Improve efficiency of communication from multithreaded PEs

Simple, thread-safety solutions are inefficient
- SHMEM has a single communication stream for each PE, as a result
- Communication overlap across threads is not possible; fence/quiet affects all threads
- Threads share resources within the runtime system, resulting in synchronization overheads

More information:
- Formal proposal and API: https://github.com/jdinan/openshmem-contexts
- Redmine #177: http://bongo.cs.uh.edu/redmine/issues/177
Communication in OpenSHMEM 1.x

Not possible to choose which operations are impacted by the quiet
Not possible for subset of operations to overlap with the quiet
Communication Contexts and PEs

Contexts can be used to isolate streams of communication ops from a PE
Enable communication/computation overlap and pipelining
Simple, Thread-Safe Communication

Quiet from one thread interacts with other thread’s communication
Communication Contexts and Threads

Contexts can be used to isolate streams of communication ops from threads
Threads can further isolate streams by using multiple contexts
Multithreaded Prog. with Contexts

1. OpenSHMEM resource domain: `shmem_domain_t`
   - User creates a local resource domain to represent each thread
   - Single-threaded codes can use existing `SHMEM_DOMAIN_DEFAULT`
   - Allows implementation to assign resources to thread
   - Used to eliminate resource sharing and synchronization

2. OpenSHMEM communication context: `shmem_ctx_t`
   - Threads create contexts on their resource domain
   - Each context is a separate unit of overlap

3. Context object passed to one-sided communication operations
   - Context object rather than thread-local storage (TLS)
   - Eliminates lookup overhead and avoids threading package dependence
Parallel bucket sort on a set of small integers

- Single-threaded benchmark, show impact of contexts on overlap control
- Contexts allow blocking gets to overtake nonblocking puts

Key exchange is the main contributor to communication
NAS IS Strong Scaling

Significant speedup from added overlap control via contexts

Speedup decreases when number of PE/Node increases

- All-to-all communication volume in the fabric increases as square of num. PEs
- Loss of speedup caused by sharing the underlying Portals/InfiniBand network
OpenSHMEM 1.x Pt-to-Pt Synchronization

Consistent Barrier

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

Point-to-Point Flags

```c
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
- Expensive

Point-to-point synchronization
- $O(P)$ messages for pt-to-pt
- $O(\log P)$ for barrier

Sync. not visible to runtime system
- Limits ability to optimize
Bundling Communication and Sync.

```c
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
- The receiver can do the increment in ct_get/wait, using completion events

Bundling enables implementation optimizations
- Leverage hardware capabilities (ordering, bundling, events, ...)

Enables a receiver-managed implementation
- Can eliminate flag update communication
Measure bandwidth achieved in all-to-all

- 15 Node QDR InfiniBand cluster
- 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
One-Sided Append

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, ...
NAS Integer Sort (IS) Key Exchange

```c
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+dst_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

PE $N$

Pe 0  Pe 1

Counter

dst_buf
NAS IS Key Exchange (Push)

```c
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
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)
Federations – a System-Centric View

Proposal of a teams interface that can be used to expose detailed system topology information to the application.

- Supports conventional communication groups
- Exposes evolving network topology and heterogeneous node architecture

Integration with system management and runtime

- E.g. integration with PMIx, hwloc, netloc; (see open-mpi.org)

Facilitates Topology-aware parallel decomposition

Enables definition of communication patterns aligned with topologies
Federations

Federations created by split operations performed on **SHMEM_TEAM_WORLD**

A federation contains set of all teams created during split

System topology derived federations via split keys, similarly to those used in the **MPI_Comm_split_type** operation

Introducing a new **shmem_federation_t** type

Examples:

- Leaf switch nodes (fat tree)
- All2all connected nodes (dragonfly)
PE-Centric Viewpoint

Cray proposed SHMEM Teams extension* to manage PE subsets
Takes a PE-centric view, supporting intra-team communication
Created with collective `shmem_team_split`, grouping PEs into new teams based on the value supplied in color argument

Limitations:
- Can't provide global view
- No support for topology based colors
- Application's responsibility to set color value and to track it
- Limited query functions only about the teams a given PE is member of
- Two baseline teams: `SHMEM_TEAM_WORLD` and `SHMEM_TEAM_NODE`

Federations proposal builds upon the Cray proposal

* ten Bruggencate, M.: Cray SHMEM update. Presented to OpenSHMEM Workshop (March 2014)
Discover teams (system and application initiated teams)

Query team membership (for my_PE and for any other PE)

Select code path based on team membership (e.g. support 2 cyclic red/black iterations)

Operate on, map and translate teams and team members

Intra team communications: particularly collectives

Inter team communications (similar to MPI neighborhood collectives)

Provide a common framework through UL-Federation library and relieve application from explicit team management
Federations: Conclusions

Propose a two-stage approach to extending OpenSHMEM

- Foundational team APIs defined for the OpenSHMEM standard
- Reference user-level library to demonstrate a proposed, rich set of APIs exposing greater degree of information via federations interface

Provide rich set of Federations/Teams APIs to application, while keeping OpenSHMEM standard nimble

Create evolutionary path for teams in OpenSHMEM standard

Separate system specifics from OpenSHMEM standard

New capability for application to see a complete picture of how the given job maps to the system topology

- Ability to adjust data, computation, and communication patterns to optimize for the given system
Open Fabrics Alliance

Enabling High-Performance Networking
Defined a set of SHMEM requirements
Influences various vendors fabric interface
Created OFIWG
  - Developed OFI “Open Fabric Interface”

Goal:
Redefining Fabric Mapping:
  Top Down Approach
  “OFI”
Open Fabric Interface

OFI: a high performance open source network interface

- Open Fabrics Alliance networking standard
- Goals: scalability, implementation independence, decreased software overhead

Papers

- PGAS 2014, HOTI 2015

OFI WG is open participation

- Contact the ofiwg mail list ofiwg@lists.openfabrics.org
- Source code available through github.com/ofiwg
OFI Framework

PGAS Programming Languages
(OpenSHMEM, CAF, ..)

OFI Framework

Interface Control
- Domains
- Memory Regions
- Event Queues
- Endpoints
- Address Vectors
- Counters

Communication Operations
- Tagged Messaging
- RMA
- Triggered Operations
- Message Queues
- Atomic Operations
- ...

Provider Implementations

Fabric Hardware
SHMEM-OFI: Components

Endpoint
- Full address space, single universal key
- Remote completion enabled

Counters
- Write: outgoing put/atomics
- Read: outgoing reads/atomics
- Remote: incoming messages

MR: Incoming messaging
EQ: enable “non-blocking” writes
SHMEM-OFI and IB Verbs Comparison

Methodology

- Semantic match measured in instructions
- Mapping overhead: instructions in fabric communication middleware
- Single Operation: put/sync/fence
- 2 nodes
- Lower instruction count = less software overhead

The evaluation is done on two dual socket nodes containing Intel® Xeon® X5570 Quad-core Nehalem CPUs, running at 2.93GHz with 12GB of host memory. The nodes are connected by a Mellanox QDR/10GigE ConnectX InfiniBand HCA (MT26428). The nodes are running Red Hat Linux 6.5, with kernel version 2.6.32-431.el6.x86 64. The Intel® C++ Composer XE 2013 SP1 (Intel® C++ compiler 14.0) is used with "-O3 -ipo" flags. GASNet 1.22.4 and MVAPICH2-X 2.0 are utilized for the evaluation. We use a variation of the Intel® Pin for generating instruction traces for accurate measurement.
Results SHMEM-OFI and IB Verbs

SHMEM-OFI
- 95%~ software Overhead reduction
- Improved Memory registration/usage
- Put: Inject feature
- Quiet: counters

SHMEM-GASNet
- segmented address space lookup

MVAPICH2-X
- Registration key lookup

Right abstraction can yield more efficient mapping from programming model to device
Scalability Improvements Examples

Comparison of OFI to IB Verbs (from HotI 2015 paper)

Addressing: 77% memory footprint reduction
  - 36 bytes metadata libibverbs vs. 8 bytes OFI
  - OFI AV: address data best handled by fabric
    – OFI encodes subnet path in unused addressing bits

Transmit code Path: 52% memory footprint reduction
  - Arg size for a write: 84 bytes libibverbs vs. 40 OFI
  - OFI transmit interface definition is simpler
    – Endpoint customization -> fewer branches
    – Multi-entry points for different messaging types (RMA vs. MSG vs. Tagged)
OFI for PGAS – Functionality Highlights

- Scalable endpoint enumeration: 0(1) memory usage and no lookups, can enumerate PE’s with integers
- Scalable memory usage/registration: enables upfront full VA registration, remote VA
- Remote completion for RMA/atomics no extra sync communication: lightweight fence/quiet
- Exposes counters: lightweight completion mechanism
- Rich atomic set: full SHMEM coverage, enables optimizations and synchronization algorithms
- Instruction count minimization: lightweight path with direct provider usage
OpenSHMEM Contexts Extension Mapping in OFI

- `shmex_ctx_int_get(ctx[0],...)`
- `shmex_ctx_int_put_nb(ctx[1],...)`
- `shmex_impl_ctx_int_get()`
- `shmex_impl_ctx_int_put_nb()`

**Diagram:**
- **Read Counter**
- **Write Counter**
- **Cntr_E P[0]**
- **Cntr_E P[1]**
- **Shared Transmit Command Queue (STx)**
- **Maps to command queues**
- **Scalable Fabric Endpoints**

**Optimization Notice:**
- Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.*
PGAS – OFI implementations

GASNet-OFI conduit

- Enables UPC-GASNet-OFI
- *OFI enables*: lightweight polling overhead and memory management
- Submitted to Lawrence Berkeley Lab. for review

SHMEM-OFI

- *OFI enables*: lightweight memory registration and counter completion, full atomic breadth
- Internal prototyping and testing extensions: context
References


Participants in the PGAS community:

- Ensure PGAS models can access to full capabilities of Intel HPC offerings
- Advocate for needs of PGAS users within Intel
- Participate in efforts to improve and define parallel programming models

Areas of engagement:

- Tackling key programming model challenges to OpenSHMEM
  - Multithreading and communication/computation overlap
  - Point-to-point synchronization and data dependence
  - Coordination and buffer management
- Open Fabrics Interface
  - Community effort to define new low-level communication API
  - General purpose, optimizable, and aligned with parallel programming models