OPENSHMEM TUTORIAL

Presenters: Aaron Welch, Swaroop Pophale, Tony Curtis
Material: Aaron Welch, Pengfei Hao, Deepak Eachempati, Swaroop Pophale, Tony Curtis
University of Houston, Texas

http://www.uh.edu/
OpenSHMEM UH Team

Dr. Barbara Chapman
Swaroop Pophale

Tony Curtis

Pengfei Hao

Aaron Welch
Tutorial Outline

- OpenSHMEM
  - Background
  - PGAS
    - Languages vs. Libraries
  - OpenSHMEM History
  - OpenSHMEM Effort
  - OpenSHMEM Library API
  - OpenSHMEM and Hardware
  - OpenSHMEM Implementations
    - Reference Implementation overview
    - Developing OpenSHMEM Applications
- UCCS
We assume ...

- Knowledge of C
- Familiarity with parallel computing
- Linux/UNIX command-line
Background (1)

- Concurrent
  - Multiple things logically happen at once
  - May be emulated
    - E.g. time slicing on shared machine

- Parallel
  - = Concurrent +
  - Things really happen independently
  - On separate processors

- Work is partitioned in some way across resources
Large applications require lots of compute power

Various approaches to providing this
- Mainframe
- SMP
- Cluster

All involve
- Multiple things happening at once
- ...Which needs...
- Programming methods to
  - Express this
  - Take advantage of systems
Background (3)

- **2 main software paradigms**
  - Threaded
    - OpenMP
  - Message-passing
    - MPI

- **2 main hardware paradigms**
  - Single-image multiprocessor (SMP)
  - Distributed
    - Multiple machines with separate OS
    - Connected together

- Programming environments provide abstraction
Address Spaces

Global vs. distributed

OpenMP has global (shared) space

MPI has partitioned space
  - Private data exchanged via messages

OpenSHMEM is “partitioned global address space” library
  - PGAS
Background (5)

- **SPMD**
  - Program launches many processes
  - Each starts with same code (SP)
  - And then typically operates on some specific part of the data (MD)
  - Processes may then communicate with each other
    - Share common data
    - Broadcast work
    - Collect results

- **The PGAS family**
  - Libraries include…
    - GASNet, ARMCI / Global Arrays, UCCS, CCI, GASPI/GPI, OpenSHMEM
  - Languages include…
    - Chapel, Titanium, X10, UPC, CAF

---

A language or library can be used on many machine types, their implementation hides differences & leverages features.
PGAS Languages vs Libraries

- Program written in given language
- Compiler for Language
- exec

- Program written using library API
- Some Compiler
- exec
- library implementation
## PGAS Languages vs Libraries

<table>
<thead>
<tr>
<th>Languages</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Often more concise</td>
<td>More information redundancy in program</td>
</tr>
<tr>
<td>Requires compiler support</td>
<td>Generally not dependent on a particular compiler</td>
</tr>
<tr>
<td>More compiler optimization opportunities</td>
<td>Library calls are a &quot;black box&quot; to compiler, typically inhibiting optimization</td>
</tr>
<tr>
<td>User may have less control over performance</td>
<td>Often usable from many different languages through bindings</td>
</tr>
<tr>
<td><strong>Examples:</strong> UPC, CAF, Titanium, Chapel, X10</td>
<td><strong>Examples:</strong> OpenSHMEM, Global Arrays, MPI-3</td>
</tr>
</tbody>
</table>
A number of threads working independently in a SPMD fashion

- Number of threads specified at compile-time or run-time; program variable `THREADS`
- `MYTHREAD` specifies thread index (0..`THREADS-1`)
- `upc_barrier` is a global synchronization: all wait
- `upc_forall` is the work sharing construct

There are two compilation modes

- Static Threads mode:
  - `THREADS` is specified at compile time by the user
  - The program may use `THREADS` as a compile-time constant
- Dynamic threads mode:
  - Compiled code may be run with varying numbers of threads
Any legal C program is also a legal UPC program
If you compile and run it as UPC with P threads, it will run P copies of the program.
Example of a parallel hello world using UPC:

```c
#include <upc.h>    /* needed for UPC extensions */
#include <stdio.h>

main() {
    printf("Thread %d of %d: hello UPC world\n",
            MYTHREAD, THREADS);
}
```
PGAS Language
Coarray Fortran (CAF)

- multiple executing images
- explicit data decomposition and movement across images achieved by declaring and accessing coarrays
- image control statements
  - subdivide program into execution segments
  - determine partial ordering of segments among images
  - define scope for compiler optimization
- part of Fortran 2008 standard
- other languages enhancements (teams, expanded collectives and atomics, semaphore synchronization, resilience) are being considered for next revision
An SPMD parallel programming library
- Library of functions similar in feel to MPI (e.g. `shmem_get()`)

Available for C / Fortran

Used for programs that
- perform computations in separate address spaces and
- explicitly pass data to and from different processes in the program.

The processes participating in shared memory applications are referred to as processing elements (PEs).

SHMEM routines supply remote data transfer, work-shared broadcast and reduction, barrier synchronization, and atomic memory operations.
OpenSHMEM

- An OpenSHMEM "Hello World"

```c
#include <stdio.h>
#include <mpp/shmem.h>

int main (int argc, char **argv){
    int me, npes;
    start_pes (0); /*Library Initialization*/
    me = _my_pe ();
    npes = _num_pes ();
    printf ("Hello World from node %4d of %4d\n", me, npes);
    return 0;
}
```
OpenSHMEM History

- **Cray**
  - SHMEM first introduced by Cray Research Inc. in 1993 for Cray T3D
  - Platforms: Cray T3D, T3E, PVP, XT series

- **SGI**
  - Owns the “rights” for SHMEM
  - Baseline for OpenSHMEM development (Altix)

- **Quadrics** (company out of business)
  - Optimized API for QsNet
  - Platform: Linux cluster with QsNet interconnect

- **Others**
  - HP SHMEM, IBM SHMEM
  - GPSHMEM (cluster with ARMCI & MPI support, old)

Note: SHMEM was not defined by any one standard.
Symmetric Variables

Arrays or variables that exist with the same size, type, and relative address on all PEs.

The following kinds of data objects are symmetric:

- Fortran data objects in common blocks or with the SAVE attribute.
- Non-stack C and C++ variables.
- Fortran arrays allocated with shpalloc
- C and C++ data allocated by shmalloc
Dynamic allocation of Symmetric Data

```c
int main (void)
{
    int *x;
    ...
    start_pes(0);
    ...
    x = (int*) shmalloc(sizeof(x));
    ...  
    ...  
    shmem_barrier_all();
    ...  
    shfree(x);
    return 0;
}
```
Many forms of initialization
- Include header shmem.h to access the library
  - E.g. `#include <shmem.h>` , `#include <mpp/shmem.h>`
- `start_pes`, `shmem_init`: Initializes the calling PE
- `my_pe`: Get the PE ID of local processor
- `num_pes`: Get the total number of PEs in the system

<table>
<thead>
<tr>
<th></th>
<th>SGI</th>
<th>Quadrics</th>
<th>Cray</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fortran</td>
<td>C/C++</td>
<td>C/C++</td>
</tr>
<tr>
<td><code>start_pes</code></td>
<td><code>start_pes(0)</code></td>
<td><code>shmem_init</code></td>
<td><code>start_pes</code></td>
</tr>
<tr>
<td><code>shmem_my_pe</code></td>
<td><code>shmem_my_pe</code></td>
<td></td>
<td><code>shmem_my_pe</code></td>
</tr>
<tr>
<td><code>shmem_n_pes</code></td>
<td><code>shmem_n_pes</code></td>
<td></td>
<td><code>shmem_n_pes</code></td>
</tr>
<tr>
<td><code>NUM_PES</code></td>
<td><code>num_pes</code></td>
<td><code>num_pes</code></td>
<td><code>NUM_PES</code></td>
</tr>
<tr>
<td><code>MY_PE</code></td>
<td><code>my_pe</code></td>
<td><code>my_pe</code></td>
<td></td>
</tr>
</tbody>
</table>
OpenSHMEM Effort
Divergent Implementations (2)

Hello World (SGI on Altix)

```c
#include <stdio.h>
#include <mpp/shmem.h>

int main(void)
{
    int me, npes;
    start_pes(0);
    npes = _num_pes();
    me = _my_pe();
    printf("Hello from %d of %d\n", me, npes);
    return 0;
}
```

Hello World (Cray)

```c
#include <stdio.h>
#include <shmem.h>

int main(void)
{
    int me, npes;
    shmem_init();
    npes = num_pes();
    me = my_pe();
    printf("Hello from %d of %d\n", me, npes);
    return 0;
}
```
OpenSHMEM Effort
The Project

- http://www.openshmem.org/
- Standardized specification
- OpenSHMEM Library Reference Implementation
- Validation and Verification Suite
- Tutorials & other educational material
- Vendor products & information
- Community involvement, talk to each other!
- Tool-chain ecosystem
OpenSHMEM Routines

- Initialization and Program Query
- Symmetric Data Management
- Data transfers
- Synchronization mechanisms
- Collective communication
- Atomic Memory Operations
- Data Cache control

Not supported by all OpenSHMEM implementations
void start_pes(int n)
- Initialize the OpenSHMEM program
- “n” means “number of PEs” but now ignored, set to 0
- Number of PEs taken from invoking environment
  - E.g. from MPI or job scheduler
- PEs numbered 0 .. (N – 1) in flat space

int _num_pes(void)
int shmem_n Pes(void)
  - return number of PEs in this program
int _my_pe(void)
int shmem_my_pe(void)
  - return “rank” of calling PE
OpenSHMEM
Memory Management Operation (1)

- `void *shmalloc(size_t size);`
  - Allocate symmetric memory on all PEs.

- `void *shfree(void *ptr);`
  - Deallocate symmetric memory.

- `void *shrealloc(void *ptr, size_t size);`
  - Resize the symmetric memory

- `void *shmempalign(size_t alignment, size_t size);`
  - Allocate symmetric memory with alignment
/* shmalloc() & shfree() */
#include <stdio.h>
#include <shmem.h>
int main (int argc, char **argv)
{
    int *v;
    start_pes (0);
    v=(int*)shmalloc(sizeof(int));
    shfree(v);
    return 0;
}
OpenSHMEM
Data Transfer (1)

- **Put**
  - Single variable
    - `void shmem_TYPE_p(TYPE *target, TYPE value, int pe)`
      - `TYPE = double, float, int, long, short`
  - Contiguous object
    - `void shmem_TYPE_put(TYPE *target, const TYPE *source, size_t nelems, int pe)`
      - For C: `TYPE = double, float, int, long, longdouble, longlong, short`
      - For Fortran: `TYPE=complex, integer, real, character, logical`
    - `void shmem_putSS(void *target, const void *source, size_t nelems, int pe)`
      - Storage Size (SS, bits) = 32, 64, 128, mem (any size)
  - Target must be symmetric
Example: Cyclic communication via puts

```c
/*Initializations*/
int src;
int *dest;
....
start_pes(0);
....
src = me;
target = (int *) shmalloc (sizeof (*target));
nextpe = (me + 1) % npes; /*wrap around*/

shmem_int_put (target, &src, 1, nextpe);
...
shmem_barrier_all();
x = dest * 0.995 + 45 * y;
...
```

Points To Remember

- ‘Destination’ has to be symmetric
- Consecutive puts are not guaranteed to finish in order
- Put returns after the data has been copied out of the source
- Completion guaranteed only after synchronization
OpenSHMEM
Data Transfer (3)

Excuse me while I overwrite your target with my copy of source

Output
- target on PE 0 is 3
- target on PE 1 is 0
- target on PE 2 is 1
- target on PE 3 is 2
- target on PE 4 is 3
**Get**

- **Single variable**
  - `TYPE shmem_TYPE_g(TYPE *target, TYPE value, int pe)`
  - For C: `TYPE` = `double`, `float`, `int`, `long`, `longdouble`, `longlong`, `short`
  - For Fortran: `TYPE`=`complex`, `integer`, `real`, `character`, `logical`

- **Contiguous object**
  - `void shmem_TYPE_get(TYPE *target, const TYPE *source, size_t nelems, int pe)`
  - For C: `TYPE` = `double`, `float`, `int`, `long`, `longdouble`, `longlong`, `short`
  - For Fortran: `TYPE`=`complex`, `integer`, `real`, `character`, `logical`

- `void shmem_getSS(void *target, const void *source, size_t nelems, int pe)`
  - Storage Size (SS, bits) = 32, 64, 128, mem (any size)

- Source must be symmetric
Example: Summation at PE 0

```c
/*Initializations*/
int *src;
int target, sum;
.....
start_pes(0);
.....
src = (int *) shmalloc(sizeof (*src));
src = me;
sum = me;
if (me == 0){
    for (int i = 1, i < num_pes(); i++){
        shmem_int_get(&target, src, 1, i)
        sum = sum + target;
    }
}
```

**Points To Remember**
- ‘Source’ has to be remotely accessible
- Consecutive gets finish in order
- The routines return after the data has been delivered to the ‘target’ on the local PE
Strided put/get

```c
void shmem_TYPE_iput(TYPE *target, const TYPE *source, 
    ptrdiff_t tst, ptrdiff_t sst, 
    size_t nelems, int pe)
```

- For C: TYPE = double, float, int, long, longdouble, longlong, short
- For Fortran: TYPE=complex, integer, real, character, logical
- tst and sst indicate stride between accesses of target and source resp.

And the sized variants as for put/get
#include <stdio.h>
#include <shmem.h>

int main()
{
    static short source[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    short target[10];
    int i, me;
    for (i = 0; i < 10; i += 1){
        target[i] = 666;
    }
    start_pes (0);
    me = _my_pe ();
    if (me == 1){
        shmem_short_iget (target, source, 2, 1, 4, 0); /* source[0,1,2,3] -> target[0,2,4,6] */
    }
    shmem_barrier_all (); /* sync sender and receiver */
    if (me == 1){
        for (i = 0; i < 10; i += 1){
            printf ("PE %d: target[%d] = %hd, source[%d] = %hd\n",
                      me, i, target[i], i, source[i]);
        }
    }
    shmem_barrier_all (); /* sync before exiting */
    return 0;
}
Put vs. Get

- Put call completes when data is “being sent”
- Get call completes when data is “stored locally”

Cannot assume put has written until later synchronization
- Data still in transit
- Partially written at target
- Put order changed by e.g. network

Puts allow overlap
- Communicate
- Compute
- Synchronize
Active Sets
- Way to specify a subset of PEs
- A triple:
  - Start PE
  - Stride ($\log_2$)
  - Size of set

Limitations
- Stride must be powers of 2
- Only define ‘regular’ PE sub-groups
Quick look at Active Sets

Example 1
- \( PE_{\text{start}} = 0, \ logPE_{\text{stride}} = 0, \ PE_{\text{size}} = 4 \)
- **ACTIVE SET?** PE 0, PE 1, PE 2, PE 3

Example 2
- \( PE_{\text{start}} = 0, \ logPE_{\text{stride}} = 1, \ PE_{\text{size}} = 4 \)
- **ACTIVE SET?** PE 0, PE 2, PE 4, PE 6

Example 3
- \( PE_{\text{start}} = 2, \ logPE_{\text{stride}} = 2, \ PE_{\text{size}} = 3 \)
- **ACTIVE SET?** PE 2, PE 6, PE 10
OpenSHMEM Synchronization (3)

- **Barrier (Group synchronization)**
  - `void shmem_barrier_all()`
    - Suspend PE execution until all PEs call this function
  - `void shmem_barrier(int PE_start, int PE_stride, int PE_size, long *pSync)`
    - Barrier operation on subset of PEs

- `pSync` is a symmetric work array that allows different barriers to operate simultaneously
OpenSHMEM
Synchronization (4)

shmem_barrier_all() synchronizes all executing PEs

Ensures completion of all
• local memory stores
• remote memory updates
OpenSHMEM
Synchronization (5)

#include <stdio.h>
#include <shmem.h>
#include <stdlib.h>
define GREEN 1
define RED 0

int light=RED;
int main(int argc, char **argv)
{
  int me;
  start_pes(0);
  me= _my_pe();
  if(me==0){
    printf("me:%d. Stop on Red Light\n", me);
    shmem_int_wait(&light, RED); /* Is the light still red? */
    printf("me:%d. Now I may proceed\n", me);
  }
  if(me==1){
    sleep(1);
    light=GREEN;
    printf("me:%d. I've turn light to green.\n", me);
    shmem_int_put(&light, &light, 1, 0);
  }
  return 0;
}

Output:
me:0. Stop on Red Light
me:1. I've turned light to green
me:0. Now I may proceed
Conditional wait (P2P synchronization)
- Suspend until local symmetric variable NOT equal to the value specified
  - `void shmem_wait(long *var, long value)`
  - `void shmem_TYPE_wait(TYPE *var, TYPE value)`
  - For C: `TYPE = int, long, longdouble, longlong, short`
  - For Fortran: `TYPE = complex, integer, real, character, logical`

Specific conditional wait
- Similar to the generic wait except the comparison can now be
  - `>=, >, =, !=, <, <=`
  - `void shmem_wait_until(long *var, int cond, long value)`
  - `void shmem_TYPE_wait_until(TYPE *var, int cond, TYPE value)`
  - `TYPE = int, long, longlong, short`
OpenSHMEM
Synchronization (7)

Fence
- Ensures ordering of outgoing write (put) operations on a per-PE basis.
- `void shmem_fence()`

Quiet
- Waits for completion of all outstanding remote writes and stores to symmetric data objects initiated from the calling PE.
- `void shmem_quiet()`
if (_my_pe() == 0) {
    shmem_long_put(target, source, 3, 1); /*put1*/
    shmem_long_put(target, source, 3, 2); /*put2*/
    shmem_fence();
    shmem_int_put(&targ, &src, 1, 1); /*put3*/
    shmem_int_put(&targ, &src, 1, 2); /*put4*/
}
...

put1 will be ordered to be delivered before put3
put2 will be ordered to be delivered before put4

shmem_long_put(target, source, 3, 1); /*put1*/
shmem_int_put(&targ, &src, 1, 2); /*put4*/
shmem_quiet();
shmem_long_get(target, source, 3, 1);
shmem_int_get(&targ, &src, 1, 2);
printf("target:{%d,%d,%d}

",target[0],target[1],target[2]);
printf("targ: %d
", targ); /*targ: 90*/
shmem_int_put(&targ, &src, 1, 1); /*put3*/
shmem_int_put(&targ, &src, 1, 2); /*put4*/
...

put1 & put2 will be delivered when quiet returns
OpenSHMEM
Collective Communication (1)

- **Broadcast**
  - One-to-all symmetric communication
  - No update on root

- `void shmem_broadcastSS(void *target, void *source, size_t nelems, int PE_root, int PE_start, int PE_stride, int PE_size, long *pSync)`

Storage Size (SS, bits) = 32, 64
OpenSHMEM
Collective Communication (2)

PE 0
PE 1
PE 2
PE 3

Shared Address Space
Private Address Space
...
...
int *target, *source;
target= (int *) shmalloc( sizeof(int) );
source= (int *) shmalloc( sizeof(int) );
*target= 0;
if (me == 0) {
    *source = 222;
}
else
    *source= 101;
shmem_barrier_all();
shmem_broadcast32(target, source, 1, 0, 0, 0, 4, pSync);

    printf("target on PE %d is %d\n", _my_pe(), *target);
...

Code snippet showing working of shmem_broadcast

Output

  target on PE 0 is 0
  target on PE 1 is 222
  target on PE 2 is 222
  target on PE 3 is 222
Collection

- Concatenates blocks of symmetric data from multiple PEs to an array in every PE
- Each PE can contribute different amounts
  
  ```c
  void shmem_collectSS(void *target, void *source, size_t nelems, int PE_start, int PE_stride, int PE_size, long *pSync)
  ```

  Concatenation written on all participating PEs

shmem_fcollect variant

- When all PEs contribute exactly same amount of data
- PEs know exactly where to write data, so no offset lookup overhead

Storage Size (SS, bits) = 32, 64
OpenSHMEM
Collective Communication (5)

PE 0

S
T

Shared Address Space

PE 1

PE 2

PE 3

S
T

T

Private Address Space
```c
#include <stdio.h>
#include <shmem.h>
#include <assert.h>

int sum;
int me, npe;

int main(int argc, char **argv)
{
    int i;
    long *pSync;
    int *pWrk;
    int pWrk_size;
    start_pes (0);
    me = _my_pe();
    npe = _num_pes();
    pWrk = (int *) shmalloc (npe);
    pSync = (long *) shmalloc (SHMEM_REDUCE_SYNC_SIZE);
    for (i = 0; i < SHMEM_REDUCE_SYNC_SIZE; i += 1){
        pSync[i] = _SHMEM_SYNC_VALUE;
    }
    shmem_barrier_all();
    shmem_int_sum_to_all(&sum, &me, 1, 0, 0, npe, pWrk, pSync);
    shmem_barrier_all();
    printf("me:%d. Total sum of 'me' is %d\n", me, sum);
    return 0;
}

Output:
me:1. Total sum of 'me' is 45
me:2. Total sum of 'me' is 45
me:3. Total sum of 'me' is 45
me:4. Total sum of 'me' is 45
me:5. Total sum of 'me' is 45
me:6. Total sum of 'me' is 45
me:7. Total sum of 'me' is 45
me:8. Total sum of 'me' is 45
me:9. Total sum of 'me' is 45
me:0. Total sum of 'me' is 45
```

OpenSHMEM
Collective Communication (6)
Reductions

- Perform commutative operation across symmetric data set

```c
void shmem_TYPE_OP_to_all(TYPE *target, TYPE *source, int nreduce, int PE_start, int PE_stride, int PE_size, TYPE *pWrk, long *pSync)
```

- Logical OP = and, or, xor
- Extrema OP = max, min
- Arithmetic OP = prod(uct), sum
- TYPE = int, long, longlong, longdouble, short, complex

- Reduction performed and stored on all participating PEs
- pWrk and pSync allow interleaving

- E.g. compute arithmetic mean across set of PEs
  ```c
  sum_to_all / PE_size
  ```
OpenSHMEM
Collective Communication (8)

Shared Address Space
Private Address Space
What does “atomic” mean anyway?
- Indivisible operation on symmetric variable
- No other operation can interpose during update

But “no other operation” actually means…?
- No other atomic operation
- Can’t do anything about other mechanisms interfering
  - E.g. thread outside of OpenSHMEM program
  - Non-atomic OpenSHMEM operation

Why this restriction?
- Implementation in hardware
Atomic Swap

- Unconditional
  - `long shmem_swap(long *target, long value, int pe)`
  - `TYPE shmem_TYPE_swap(TYPE *target, TYPE value, int pe)`
    - TYPE = double, float, int, long, longlong
    - Return old value from symmetric target

- Conditional
  - `TYPE shmem_TYPE_cswap(TYPE *target, TYPE cond, TYPE value, int pe)`
    - TYPE = int, long, longlong
    - Only if “cond” matches value on target
ARITHMETIC

- increment (= add 1) & add value

```
void shmem_TYPE_inc(TYPE *target, int pe)
void shmem_TYPE_add(TYPE *target, TYPE value, int pe)
```

- TYPE = int, long, longlong

- Fetch-and-increment & fetch-and-add value

```
TYPE shmem_TYPE_finc(TYPE *target, int pe)
TYPE shmem_TYPE_fadd(TYPE *target, TYPE value, int pe)
```

- TYPE = int, long, longlong

- Return previous value at target on PE
long *dest;
dest = (long *) shmalloc( sizeof(*dest) );
*dest= me;
shmemp_barrier_all();

...
new_val = me;
if (me== 1) {
    swapped_val = shmem_long_swap(target, new_val, 0);
    printf("%d: target = %d, swapped = %d\n", me, *target, swapped_val);
}
shmemp_barrier_all();
...
Locks

- Symmetric variables
- Acquired and released to define mutual-exclusion execution regions

- Only 1 PE can enter at a time

- `void shmem_set_lock(long *lock)`
- `void shmem_clear_lock(long *lock)`
- `int shmem_test_lock(long *lock)`

- Acquire lock if possible, return whether or not acquired
- But don’t block…

- Initialize lock to 0. After that managed by above API
- Can be used for updating distributed data structures
OpenSHMEM
Accessibility

- int shmem_pe_accessible(int pe)
  - Can this PE talk to the given PE?

- int shmem_addr_accessible(void *addr, int pe)
  - Can this PE address the named memory location on the given PE?

- In SGI SHMEM used for mixed-mode MPI/SHMEM programs
  - In “pure” OpenSHMEM, could just return “1”

- Could in future be adapted for fault-tolerance
Cache control

- `shmem_clear_cache_inv` - Disables automatic cache coherency mode
- `shmem_set_cache_inv` - Enables automatic cache coherency mode
- `shmem_set_cache_line_inv` - Enables automatic line cache coherency mode
- `shmem_udcflush` - Makes the entire user data cache coherent
- `shmem_udcflush_line` - Makes coherent a cache line
OpenSHMEM Hardware (1)

- Where is OpenSHMEM used?
  - Mainly clusters these days
  - Infiniband and similar networks
  - Why?

- Remote direct memory access (RDMA)
  - Network hardware writes directly into registered region of process memory
  - Without interrupting remote process(or)
  - Put symmetric memory areas here
Offload

- Infiniband HCAs can do
  - Atomics
  - Collectives
  - Memory pinning

- Meaning CPU free to do other things
- Reduced software footprint (QPs)
- OpenSHMEM library issues offload instructions rather than doing atomics etc.
OpenSHMEM
Summary

- SPMD Library for C and Fortran programs
- Point-to-point data transfer
- Broadcast/collective transfer operations
- Synchronization
- Atomic operations
OpenSHMEM Implementations

- Reference Library: University of Houston
  - On top of GASNet for portability
  - [http://www.openshmem.org/](http://www.openshmem.org/)

- ScalableSHMEM: Mellanox
  - For Mellanox Infiniband solutions

- Portals-SHMEM: open-source
  - For Portals clusters
Future Work

- Library side
  - Extended API
  - Fault tolerance
  - Larger ecosystem of tools

- Compiler side
  - OpenSHMEM-aware compilers
  - Tools to analyze source code
  - Suggest e.g.
    - code-motion to provide better communication/computation overlaps, transfer coalescing…
Developing OpenSHMEM Applications
OpenSHMEM
Looking for Overlaps (1)

- How to identify overlap opportunities
  - Put is not an indivisible operation
    - Send local, reuse local, on-wire, stored
    - Can do useful work on other data in between
OpenSHMEM
Looking for Overlaps (2)

- **How to identify overlap opportunities**

  - **General principle:**
    - Identify independent tasks/data
    - Initiate action as early as possible
      - Put/barrier/collective
    - Interpose independent work
    - Synchronize as late as possible
OpenSHMEM
Looking for Overlaps (3)

- How to identify overlap opportunities
  - How could we change OpenSHMEM to get even more overlap?
    - Divide application into distinct communication and computation phases to minimize synchronization points
    - Use of point-to-point synchronization as opposed to collective synchronization
OpenSHMEM
Looking for Overlaps (4)

- How to identify overlap opportunities
  - Shmalloc
    - Size check, allocate, barrier_all

  - Opportunities to do other work after local allocation
  - Then wait in barrier later
  - Return handle for synch.
OpenSHMEM
Looking for Overlaps (5)

- How to identify overlap opportunities
  - \texttt{\_nb} put/get calls
    - Local data not free for reuse on return
  - Return handle for later synch.
UCCS Overview

- Abstract communication and networking implementation details
- Support multiple transport layers/programming models
- Provide single consistent interface
- Communication decoupled from run-time environment
Motivation

- Efficient communication expected to become increasingly important
- What underlying technologies will prevail is unclear
- Provide general but low-level interface capable of supporting current or future models
Usability Goals

- Increase code reuse, decrease complexity of network backend
- Support for multiple communication libraries
- Tight integration to foster support for languages and tools
UCCS Design

• Designed for low overhead, scalability, and resiliency

• Maintain minimal footprint with emphasis on reducing the critical path by operating very close to the hardware

• Three different sizes for puts and gets (short, medium, and large) using different methods for handling network requests

• Emphasis on non-blocking calls using request handles

• Support for atomic operations and low-level collectives
UCCS Design

- Capabilities interface allows for querying of hardware support details
- Connectivity maps allow for heterogeneous network systems
- Dynamic memory registration for multiple remotely accessible regions
- Support for active messages
UCCS Design

Diagram showing the integration of various APIs and libraries, including UPC CAF, RDMA PUT/GET API, Atomic API, Collectives API, and RTE. The diagram illustrates the flow of data and functions supported by each API, with examples of functions listed for Contiguous and Non-Contiguous data.
UCCS Concepts

- Contexts provide communication scope and isolation
- Resources represent available communication channels for a network
- Endpoints are the destinations reachable over a particular resource
RTE Design

- Handles bootstrapping and other dynamic aspects of the run-time environment
- Provides out-of-band support
- Abstracted under a single interface to support multiple RTE backends including ORTE, STCI, SLURM, and ALPS
Modular Component Architecture

- Allows for multiple components to be plugged in or swapped out
- Interface allows for easy creation of custom components
- Licensing support extends to development of proprietary components
Transport Layers

- Multiple options including InfiniBand, uGNI, and support for intra-node communication
- Can be dynamically selected, mixed, and matched
- Allows support for hybrid network systems, multi-rail
- Integrates with capabilities for transport priority when multiple routes available
What UCCS allows

- Library implementers can easily support a wide array of network technologies and configurations
- Consistent interface across multiple interconnects and communication libraries
- Hybrid development
- Heterogeneous systems
User Environment

- Communication Library
- UCCS
- libRTE
- RTE backend (ORTE, STCI, etc)
Example: shmem_int_add()

```c
shmem_int_add(void *target, void *value, size_t nbytes, int pe) {
    uccs_request_handle_t desc;
    uccs_status_t status;
    resource = select_highest_priority_resource(pe);
    dest = translate_target_to_remote_address(target);
    find_memory_registration_for_dest_on_pe(dest, pe, &rem_reg);
    uccs_atomic_add_int64_nb(resource, endpoint, dest, &rem_reg,
                             value, 255, &desc);
    uccs_wait(desc, &uccs_status);
}
```
Example: shmem_int_p()

```c
void shmem_int_p(int *target, int value, int pe) {
  uccs_request_handle_t desc;
  dest = translate_target_to_remote_address(target);
  uccs_put_contiguous_short_nb(resource, endpoint, dest,
    &value, &remote_reg, sizeof(int), 0, &desc);
  uccs_wait(desc, &status);
}
```
Example: shmem_int_put()

```c
void shmem_int_put(int *target, const int *source, size_t len, int pe) {
    uccs_request_handle_t desc;
    dest = translate_target_to_remote_address(target);
    if (len <= comms[pe].short_put_size)
        uccs_put_contiguous_short_nb(resource, endpoint, dest,
                                       source, remote_reg, len, 0, desc);
    else if (len <= comms[pe].medium_put_size)
        uccs_put_contiguous_medium_nb(resource, endpoint, dest, source,
                                        remote_reg, NULL, len, 0, desc);
    else
        uccs_put_contiguous_large_nb(resource, endpoint, dest, source,
                                       remote_reg, NULL, len, 0, desc);
    uccs_wait(desc, &status);
}
```
Testing Environment

- OpenSHMEM reference implementation 1.0e
- GASNet v1.20.2
- Pre-production version of UCCS based on v0.2 of UCCS specification
- SGI Altix XE1300 system with 12 nodes with two Intel Xeon X5660 CPUs
- InfiniBand interconnect using Mellanox ConnectX-2 QDR HCA
- SGI MPT v2.03
Testing Environment

- Results obtained from “Designing a High Performance OpenSHMEM Implementation Using Universal Common Communication Substrate as a Communication Middleware”, First OpenSHMEM Workshop
Put Latency

![Graph showing latency vs. message size for different benchmarks.](image)
Get Latency

![Graph showing latency against message size for different configurations.](image)
Long Long Fetch-and-Add Latency
Development Status

- Supported interconnects: IB, uGNI/Cray
- [http://uccs.github.io/uccs/](http://uccs.github.io/uccs/)
- mailto:uccs-info@ornl.gov
Acknowledgements

This work was supported by the United States Department of Defense & used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.