Latest in OpenSHMEM: Specification, API, and Programming

Presenters:
Graham Lopez, Dounia Khaldi, Pavel Shamis, Manjunath Gorentla Venkata
Tutorial Outline

- The OpenSHMEM API
- Solving SAT with OpenSHMEM
- OpenSHMEM Serial to Parallel Code
  Distributed Hash Table
- OpenSHMEM implementation: from the hardware layer to the user layer
The OpenSHMEM API

Graham Lopez
ORNL

Material: Swaroop Pophale
Outline

• Background
• Introduction to OpenSHMEM
• History of SHMEM
• OpenSHMEM Effort
• OpenSHMEM Concepts
• OpenSHMEM API
We assume …

• Knowledge of C
• Familiarity with parallel computing
• Linux/UNIX command-line
Background

- **Global vs. distributed Address Spaces**
  - OpenMP has global (shared) space
  - MPI has partitioned space; private data exchanged via messages
  - OpenSHMEM uses “partitioned global address space” (PGAS)
    - Implemented as a library
Background

- SPMD – single program, multiple data
  - Program launches many processes
  - Each starts with the same code (SP)
  - But 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
    - Synchronization
Background

• 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
## Background

<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>
</tbody>
</table>

**Examples:** UPC, CAF, Titanium, Chapel, X10

**Examples:** OpenSHMEM, Global Arrays, MPI-3

**PGAS Languages vs Libraries**
Background

PGAS Language - UPC

• A number of threads working independently in an 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 and Dynamic threads mode
Background

Hello World in UPC

- Any legal C program is also a legal UPC program
- If you compile and run it as UPC with N threads, it will run N 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);
}
```
Background

• 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
Introduction to OpenSHMEM

• 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).
• OpenSHMEM routines supply remote data transfer, work-shared broadcast and reduction, barrier synchronization, and atomic memory operations.
Introduction to OpenSHMEM

- An OpenSHMEM “Hello World”

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

int main (int argc, char **argv) {
    int me, npes;
    shmem_init ();  /*Library Initialization*/
    me = shmem_my_pe ();
    npes = shmem_n_pes ();
    printf ("Hello World from PE %4d of %4d\n", me, npes);
    return 0;
}
```
History of SHMEM

- 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.
Divergent Implementations

• Many forms of initialization
  • Include header shmem.h to access the library
    • #include <shmem.h>
    • #include <mpp/shmem.h>
  • start_pes, shmem_init: Initializes the shmem portion of the program
  • my_pe: Get the PE ID of local processor
  • num_pes: Get the total number of PEs in the system
## Divergent Implementations

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

---

18
Divergent Implementations

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 Concepts

• Symmetric Variables
  • Arrays or variables that exist with the same name, 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
int main (void)
{
    int *x;
    ...
    shmem_init();
    ...
    x = (int*) shmalloc(sizeof(*x));
    ...
    ...
    shmem_barrier_all();
    ...
    shfree(x);
    return 0;
}
OpenSHMEM API

• Initialization, Query and Exit Routines
• Memory Management Routines
• Data transfers
• Synchronization mechanisms
• Collective communication
• Atomic Memory Operations
OpenSHMEM Initialization, Query and Exit

void shmem_init(void)
• Same functionality as deprecated void start_pes(int n)
• Number of PEs taken from invoking environment
  • E.g. from MPI or job scheduler
  • PEs numbered 0 .. (N – 1) in flat space

int shmem_n_pes(void)
• return number of PEs in this program

int shmem_my_pe(void)
• return “rank” of calling PE

void shmem_finalize(void)
• collective operation to exit the OpenSHMEM environment

void shmem_global_exit(int status)
• one PE may force all PEs to exit the OpenSHMEM environment
OpenSHMEM Memory Management Routines

void *shmem_malloc(size_t size)
  • Allocate symmetric memory on all PEs.

void *shmem_free(void *ptr)
  • Deallocate symmetric memory.

void *shmem_realloc(void *ptr, size_t size)
  • Resize the symmetric memory

void *shmem_align(size_t alignment, size_t size)
  • Allocate symmetric memory with alignment
OpenSHMEM Memory Management Routines

/* shmem_alloc() & shmem_free() */
#include <stdio.h>
#include <shmem.h>
int main (int argc, char **argv)
{
    int *v;
    shmem_init();
    v=(int *)shmem_malloc(sizeof(int));
    ...
    ...
    shmem_free(v);
    return 0;
}
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
OpenSHMEM Data Transfer

• 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_putN(void *target, const void *source, size_t nelems, int pe)`
    • Storage Size (N bits) = 32, 64, 128, mem (any size)

Target must be symmetric
OpenSHMEM Data Transfer

• Example: Cyclic communication via puts

/*Initializations*/
int src;
int *dest;
....
shmem_init();
...
src = me;
dest = (int *) shmem_malloc (sizeof (*dest));
nextpe = (me + 1) % npes; /*wrap around */

shmem_int_put (dest, &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

• 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_getN(void *target, const void *source, size_t nelems, int pe)
    • Storage Size (N bits) = 32, 64, 128, mem (any size)

• Source must be symmetric data object
OpenSHMEM Data Transfer

• Example: Summation at PE 0

/*Initializations*/
int *src, dest, sum;
...
shmem_init();
...
src = (int *) shmem_malloc(sizeof (*src));
src = me; sum = me;
if(me == 0){
    for(int i = 1; i < num_pes(); i++){
        shmem_int_get(&dest, src, 1, i)
        sum = sum + dest;
    }
}
...

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 ‘dest’ on the local PE
OpenSHMEM Data Transfer

• Strided put/get

  • `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.
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;
    }
    shmem_init();
    me = _my_pe();
    if (me == 1){
        /* source[0,1,2,3] -> target[0,2,4,6] */
        shmem_short_iget (target, source, 2, 1, 4, 0);
    }
    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;
}
OpenSHMEM Data Transfer

• 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
OpenSHMEM Synchronization

• Active Sets
  • Way to specify a subset of PEs
  • A triplet:
    • Start PE
    • Stride (log2)
    • Size of set

• Limitations
  • Stride must be powers of 2
  • Only define ‘regular’ PE sub-groups
OpenSHMEM Synchronization

• 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

• 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

```c
#define GREEN 1
#define RED 0

int light=RED;
int main(int argc, char **argv)
{
    int me;
    shmem_init();
    me = shmem_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
OpenSHMEM Synchronization

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

- `void shmem_quiet()`
  - Waits for completion of all outstanding remote writes and stores to symmetric data objects initiated from the calling PE.
OpenSHMEM Synchronization

Example Fence

... 
if (shmem_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

Example Quiet

... 
shmem_long_put(target, source, 3, 1); /*put1*/
shmem_int_put(&targ, &src, 1, 2);     /*put2*/
        shmem_quiet();
shmem_long_get(target, source, 3, 1);
shmem_int_get(&targ, &src, 1, 2);
printf("target:{%d,%d,%d}\n",
        target[0],target[1],target[2]);
printf("targ: %d\n", 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

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

  • void shmem_broadcastN(void *target, void *source,
    size_t nelems, int PE_root,
    int PE_start, int PE_stride,
    int PE_size, long *pSync)

Storage Size (N bits) = 32, 64
...
OpenSHMEM Collective Communication

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

  **Storage Size (N bits) = 32, 64**

  • 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
int sum;
int me, npe;
int main(int argc, char **argv)
{
    int i;
    long *pSync;
    int *pWrk, pWrk_size;
    shmem_inti();
    me = shmem_my_pe();
    npe = shmem_n_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;
}
OpenSHMEM Collective Communication

- **Reductions**
  - Perform commutative operation across symmetric data set
    - `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
  - `sum_to_all / PE_size`
OpenSHMEM Atomic Operations

- 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
OpenSHMEM Atomic Operations

• 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
OpenSHMEM Atomic Operations

• 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
OpenSHMEM Atomic Operations

... long *dest; dest = (long *) shmalloc( sizeof(*dest) ); *dest = me; shmem_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);
}
shmem_barrier_all(); ...


OpenSHMEM Atomic Operations

- 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
Solving SAT with OpenSHMEM

Pavel Shamis
ORNL
Background

The examples are based on http://ubcsat.dtompkins.com
• Code https://github.com/dtompkins/ubcsat
• OpenSHMEM based code
  https://github.com/shamisp/ubcsat/tree/shmem

Parallel solver algorithm is based on: “Massively Parallel Local Search for SAT”, Alejandro Arbelaez, Philippe Codognet
• Full paper:
  http://ieeexplore.ieee.org/xpl/login.jsp?
  tp=&arnumber=6495029&url=http%3A%2F%2Fieeexplore.ieee.org
  %2Fiel7%2F6493540%2F6495011%2F06495029.pdf%3Farnumber
  %3D6495029
What is the SAT problem?

\[ F = (v_{11} \lor v_{12} \lor v_{13}) \land (v_{21} \lor v_{22} \lor v_{23}) \land \cdots \land (v_{n1} \lor v_{n2} \lor v_{n3}) \]

- Finding an assignment for all the variables such that all clauses are satisfied and \( F \) is true.
- NP-Complete.
How do we solve it

Local search!

Sparrow local search solver
• Winner of 2011 SAT competition
• Part of UBCAST framework
Local Search for SAT

0: for try = 1 to MaxTries {
1: A = variable-initialization(F)
2: for Iteration = 1 to MaxFlips {
3: if A satisfies F then
4: return A
5: x = select-variable(A)
6: A = A with x flipped
7: }
8: }
9: return “No solution”
Local Search for SAT

```plaintext
for try = 1 to MaxTries {
    A = variable-initialization(F)
    for Iteration = 1 to MaxFlips {
        if A satisfies F then
            return A
        x = select-variable(A)
        A = A with x flipped
    }
}
return "No solution"
```

Starting point for the search

The “secret sauce”
Parallel Local Search

1 Core

A lot of cores
Portfolio-based Parallel Local Search

Each process uses unique SEED

0: Init-Seed(Process_ID)
0: for try = 1 to MaxTries {
1: A = variable-initialization(F)
2: for Iteration = 1 to MaxFlips {
3: if A satisfies F then
4: eureka-return(A)
5: x = select-variable(A)
6: A = A with x flipped
7: }
8: }
9: return “No solution”
0: Init-Seed(Process_ID)
0: for try = 1 to MaxTries {
1: A = variable-initialization(F)
2: for Iteration = 1 to MaxFlips {
3: if A satisfies F then
4: eureka-return(A)
5: x = select-variable(A)
6: A = A with x flipped
7: }
8: }
9: return “No solution”

start_pe(0);
seed = atoi(argv[2]) * (1+shmem_my_pe());

/* OpenSHMEM 1.2 */
shmem_global_exit(0);

/* OpenSHMEM 1.2 */
shmem_finalize();
Portfolio-based Parallel Local Search Information Sharing

\[ \text{ProbNormalizedW}(i=\text{True}) \] where 
\[ i \in [1,n] \]

\[ n \text{-number of variables} \]
\[ c \text{-number unsatisfied clauses} \]

\[ \text{ProbNormalizedW}(i = \text{true}) = \frac{\sum_{x \in [1,p]} K_{xi} \times \text{NormW}_x}{\sum_{x \in [1,p]} \text{NormW}_x} \]

\[ \text{NormW}_x = \frac{|C| - C_x}{|C|} \]

\[ x \in \text{PE}, i \text{-variable} \]
Information Sharing

0: **Init-Seed**(*Process_ID*)

0: for try = 1 to MaxTries {
1: \[ \text{A = variable-initialization-based-on-shared-information(F)} \]
2: for Iteration = 1 to Flips-Threshold {
3: if A satisfies F
4: then eureka-return(A)
5: \[ \text{x = select-variable(A)} \]
6: \[ \text{A = A with x flipped} \]
7: broadcast-information-to-all(F)
8: }
9: return "No solution"

start_pe(0);
seed = atoi(argv[2]) * (1 + shmem_my_pe());
/* Vertor of values */
rem_aVarValue = shmalloc(iNumVars * sizeof(BOOL) * shmem_n_pes());
if (rem_aVarValue == NULL) {
    fprintf(stderr, "Failed to allocate memory for rem_aVarValue\n");
    shmem_global_exit(1);
}
memset(rem_aVarValue, 0, sizeof(BOOL) * shmem_n_pes());
/* Cost of the current solution */
rem_iNumFalse = shmalloc(sizeof(UINT32) * shmem_n_pes());
if (rem_iNumFalse == NULL) {
    fprintf(stderr, "Failed to allocate memory for rem_iNumFalse\n");
    shmem_global_exit(1);
}
memset(rem_iNumFalse, 0, sizeof(UINT32) * shmem_n_pes());
/* Signal that data is there */
rem_counter = shmalloc(sizeof(long long) * shmem_n_pes());
if (rem_counter == NULL) {
    fprintf(stderr, "Failed to allocate memory for rem_counter\n");
    shmem_global_exit(0);
}
memset(rem_counter, 0, sizeof(long long) * shmem_n_pes());
shmem_barrier_all();

 ProbNormlizedW (i = true) = \( \sum_{x \in \text{VarValue}} \frac{|C| - \text{NormW}_{x}}{|C|} \)

8: }
9: return x - PE , i - variable
What Algorithm Performed the Best?
Test-bed

- SGI Altix
- 12 nodes each one with 24 cores (288 cores in total)
- Processing Element (PE) per core
- InfiniBand interconnect
- 50 instances from SAT2011 (random)
- 3-SAT (x20), 5-SAT(x20), 7-SAT(x10)
- 3 iterations for each instance / median reported
Performance – No Information Sharing

![Graph showing performance of different Processing Elements (PEs) over time. The x-axis represents time in seconds, ranging from 0 to 300. The y-axis represents the number of solved instances, ranging from 0 to 35. Different lines correspond to different PEs: PEs-1, PEs-48, PEs-96, PEs-192, PEs-24, PEs-96, and PEs-288. The graph indicates that PEs-48 and PEs-96 perform similarly initially but diverge later, with PEs-96 solving more instances than PEs-48 by the end of the 300 seconds.](image-url)
Sharing VS No-sharing

![Graph 1: Number of solved instances over time for Processing Elements (PEs) with sharing and no-sharing for 24 PEs.](image1)

![Graph 2: Number of solved instances over time for Processing Elements (PEs) with sharing and no-sharing for 48 PEs.](image2)

Legend:
- Green solid line: No Sharing PEs-24
- Green dashed line: Sharing PEs-24
- Red solid line: No Sharing PEs-48
- Red dashed line: Sharing PEs-48

Time (seconds)
Sharing VS No-sharing

**Graphs:**

- **Processing Elements (PEs):**
  - No Sharing PEs-96
  - Sharing PEs-96

- **No Sharing PEs-192**
- **Sharing PEs-192**

**Axes:**
- **Y-axis:** Number of solved instances
- **X-axis:** Time (seconds)

**Observation:**
- The graphs compare the number of solved instances over time for both sharing and no-sharing scenarios.
- The sharing scenario (dashed line) shows a faster initial rate of solving instances compared to the no-sharing scenario (solid line).
- Both scenarios approach a plateau around 300 seconds, indicating stabilization of the number of solved instances.

**Summary:**
- Sharing PEs generally lead to a quicker resolution of instances, particularly in the initial phase, compared to no-sharing cases.
Sharing VS No-sharing

![Graph showing the comparison between Sharing and No-sharing in terms of the number of solved instances over time. The graph indicates that Sharing PEs-288 solves more instances than No Sharing PEs-288.]
Thanks!

Questions?
OpenSHMEM Serial to Parallel Code Conversion

Distributed Hash Table (DHT)

Dounia Khaldi
University of Houston
The Orbit Calculation Problem

- Application benchmark:
  - Group theory
  - Calculation of the orbits of a group $G$ acting on a set $M$
  - Graph traversal problem, where each graph vertex is a group element
- For each graph vertex, compute some of its properties
- Which other graph vertices is a vertex connected to?

Notion of Orbit

Input: $m_0 \in M$, Group $G = \{g_i, 1 \leq i \leq r\}$

Output: A list $L$ containing the elements of $m_0G$ ($G$-orbit of $m_0$)

Example:
- $M$ is a set of vectors
- $G$ is a group of four $90^\circ$ rotations
- Composition of rotations corresponds to matrix multiplication
- Orbit: $mG$

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$G = \{R(0^\circ), R(90^\circ), R(180^\circ), R(270^\circ) \}$
The Orbit Basic Algorithm

Input: $m_0 \in M$, Group $G = \{g_i, 1 \leq i \leq r\}$

Output: A list $L$ containing the elements of $m_0G$ ($G$-orbit of $m_0$)

$L := [m_0]$

for $i$ from 1 to $r$ do

$x := m_0g_i$  // element of orbit of $m_0$

if $x$ is not in $L$ then

append $x$ to $L$

end if

end for

return $L$

Orbit Calculation: Serial Hash Table

- Linear lookup of points in lists (sequential comparison) expensive for large orbits
- Hash table to keep track of whether a vertex has been visited
- Lookups using hash tables almost independent of the orbit length
- Limited to certain size of groups by the amount of memory available in serial implementation
- Parallel (distributed) implementation to allow larger orbits
DHT: Open Addressing Strategy

**Implementation**
- **insert ()** puts new entry in the hash table if not present
- **lookup ()** searches for a key
- **delete ()** frees entry to remove from the hash table

**Advantages**
- Lookup is fast
- Insertion is fast
- Unlike chaining, it cannot have more elements than table slots

**Disadvantage**
- Deletion should take into account collisions
DHT: Sequential Code

1. Compute one element $o$ of the orbit of $m$
2. Compute $h(o)$
3. Lookup in the hash table
   - Insertion
   - Repetition
   - Collision

hashinsert($o$)
DHT: Data Structures

- `hashtab`: the memory where to store the orbit
- `hashcount`: number of repetitions
- `hashlen`: max size of the orbit \( (r) \)
- `collisions`: number of collisions
- `dest_pos`: the index of an element of the orbit in `hashtab`
- `local_val`: the value of an element of `hashtab`
DHT: Sequential Code (hashinsert(o))

- Size of G (r: number of vertices): hashlen = r;

```c
dest_pos = h(o);
while (1) {
    local_val = hashtab[dest_pos];
    if (local_val == 0) {
        hashtab[dest_pos] = o;
        hashcount[dest_pos] = 1;
        return;
    } else if (local_val == o) {
        hashcount[dest_pos]++;
        return;
    } else {
        collisions++;
        dest_pos++;
        if (dest_pos == hashlen)
            dest_pos = 0;
    }
}
```

- insert in the empty entry
- check to see if it is not a collision
- it is a repetition
- it is a collision
DHT: Sequential Code (hashinsert(o))

def dest_pos = h(o);
while (1) {
    local_val = hashtab[dest_pos];
    if (local_val == 0) {
        hashtab[dest_pos] = o;
        hashcount[dest_pos] = 1;
        return;
    } else if (local_val == o) {
        hashcount[dest_pos]++;
        return;
    } else {
        collisions++;
        dest_pos++;
        if (dest_pos == hashlen)
            dest_pos = 0;
    }
}

RMA (get)
RMA (put)
Exclusive accesses to hashtab and hashcount
Distributed Hash Table (DHT)

• The sender would know the identity of the receiver via hash, but not vice versa
• Larger hash tables created in parallel than can be done serially → much larger orbits
• Suitable for OpenSHMEM → requires RMA
DHT: OpenSHMEM Code Initialization

```c
void inithash()
{
    npes = shmem_n_pes();
    mype = shmem_my_pe();
    hashlen = r;
    localhashlen = hashlen/npes;
    hashtab = shmem_malloc( localhashlen * sizeof *hashtab );
    hashcount = shmem_malloc( localhashlen * sizeof *hashcount );
    pe_lock = shmem_malloc( npes * sizeof *pe_lock );
    memset( pe_lock, 0, npes * sizeof *pe_lock );
}
```
DHT: OpenSHMEM Code

#include <shmem.h>
//declaration of hashtab, hashcount,...
void orbit(m_0,G)
{
    shmem_init();
    inithash();
    shmem_barrier_all();
    for (i = mype; i < hashlen; i+=npes) {
        hashinsert(apply(m_0, g[i]));
    }
    shmem_barrier_all();
    finhash();
    shmem_finalize();
    return;
}
DHT: OpenSHMEM Hash Finish

```c
void finhash()
{
    shfree(hashtab);
    shfree(hashcount);
    shfree(pe_lock);
}
```
hash = h(o);
while (1) {
    dest_pe = ceil(hash / localhashlen);
    dest_pos = hash - (dest_pe)*localhashlen;
    /* lock the data */
    shmem_set_lock(&pe_lock[dest_pe]);
    shmem_int_get(&local_val, &hashtab[dest_pos], 1, dest_pe);
DHT: OpenSHMEM Code Using Locks

/* check to see if o already exists */
if (local_val == 0) {
    /* insert the entry */
    int one = 1;
    shm_int_put(&hashtab[dest_pos], &o, 1, dest_pe);
    shm_int_put(&hashcount[dest_pos], &one, 1, dest_pe);
    /* unlock before return */
    shm_clear_lock(&pe_lock[dest_pe]);
    return;
}
else... // repetition or collision
/* check to see if it is not a collision */
else if (local_val == o) {
    /* it is a repetition */
    shmem_int_inc(&hashcount[dest_pos], dest_pe);
    shmem_clear_lock(&pe_lock[dest_pe]);
    return;
} else { /* it is a collision */
    collisions++;
    hash++;
    if (hash == hashlen) {
        hash = 0;
    }
    shmem_clear_lock(&pe_lock[dest_pe]);
}
DHT: OpenSHMEM Code Using Locks

```
hash = h(o);
while (1) {
    dest_pe = ceil(hash / localhashlen);
    dest_pos = hash - (dest_pe)*localhashlen;
    shmem_set_lock(&pe_lock[dest_pe]);
    shmem_int_get(&local_val, &hashtab[dest_pos], 1, dest_pe);
    if (local_val == 0) {
        int one = 1;
        shmem_int_put(&hashtab[dest_pos], &o, 1, dest_pe);
        shmem_int_put(&hashcount[dest_pos], &one, 1, dest_pe);
        /* unlock before return */
        shmem_clear_lock(&pe_lock[dest_pe]);
        return;
    }
    else...    //repetition or collision;
```

DHT: OpenSHMEM Code Using Locks

/* check to see if it is a collision */
else if (local_val == o) {
    /* its a repetition */
    shmem_int_inc(&hashcount[dest_pos], dest_pe);
    shmem_clear_lock( &pe_lock[dest_pe] );
    return;
} else { /* its a collision */
    collisions++;
    hash++;
    if (hash == hashlen) {
        hash = 0;
    }
    shmem_clear_lock( &pe_lock[dest_pe] );
}
hash = h(o);
while (1) {
    dest_pe = ceil(hash / localhashlen);
    dest_pos = hash - (dest_pe)*localhashlen;
    local_val = shmem_int_cswap(&hashtab[dest_pos],0,o,dest_pe);
    if (local_val == 0 || local_val == o) {
        /* update successful, so increment count and return */
        shmem_int_inc(&hashcount[dest_pos], dest_pe);
        return;
    }
    else
        /* it’s a collision */
else { /* its a collision */
    collisions++;
    hash++;
    if (hash == hashlen) {
        hash = 0;
    }
}
DHT: Performance on Stampede using MVAPICH2-X

Interconnect:
InfiniBand Mellanox Switches/HCAs

Cores/Node: 16
• We used 16 PEs/nodes

Version: mvapich2-x/2.0b
ICC 13.1.0
CFLAGS= -O2

![Graph showing performance](image)
DHT: Conclusion

- DHT: look up and store large orbits
- DHT: illustrative example of using OpenSHMEM (RMA, locks, atomics)
- Easy transition from sequential to parallel
- Use atomics if implementation is adapted
OpenSHMEM Implementation Components and Characteristics

Manjunath Gorentla Venkata
ORNL
OpenSHMEM Components and Characteristics

Goal:
Overview of components and its characteristics in a typical OpenSHMEM implementation

Expert level:
Introductory
Components in a Typical OpenSHMEM Implementation

- Runtime
  - Language Bindings
  - Interface Layer
    - Protocol Layer
    - Network Layer
    - Collectives
  - Messaging Layer
    - Driver
    - HAL
Runtime Layer

Resides on the head node from which processes are launched, and typically a daemon resides in each node of the job

Functions

• Launches the job, leveraging job launchers such as rsh, ssh, slurm, pbs
• Provides out-of-band communication
• Fault detection and recovery
• Provides flexible process binding functionality
Examples

Launchers
- ALPS, SSH, Hydra

Runtime Implementations
- STCI - Runtime that supports fault-tolerance
- ORTE – Runtime for Open MPI
- PMI – Runtime for MPICH based implementations
  - SLURM PMI, BG/L PMI, Simple PMI

Interfaces for abstracting various runtime layers
- Librte – Library interface for runtimes
- Process Manager Interface (PMI)
Case Study: STCI

Components

• Agents
  • Controller, Root, Session, Tool
• Topology
  • Connects various agent components
  • Currently supports tree, meshes, BMGs
Case Study: STCI

- Communication Substrate
  - Bootstrap communication substrate:
    - Self-bootstrapping
    - Reliable and ordered
  - Active message communication substrate:
    - Provides communication between agents after bootstrapping
- Fault tolerance: Enables communication even in the event of failed agents
  - Fault detection
  - Fault tolerant communication
Messaging Layer

Protocol Layer

• Implements semantics of the programming model
• Provides protocols to support the programming model semantics and achieve performance
• Provides message fragmentation and coalescing abilities
• RDMA Protocols, Active Messages, Atomics

Network layer

• Transfers data between processing elements
• Typically hardware and network dependent, and this component requires to re-implementing for porting the message layer
Example

Programming model specific
- Mellanox SHMEM
- Open MPI /MPICH
- OpenSHMEM reference implementation

Independent of Programming model
- UCCS/UCX
- PAMI
- MXM
- Libfabrics
- GASNet
Case Study: UCX

Yossi will provide a detailed talk on UCX
What Collective Operations?

- Collective operations are global communication and synchronization operations in a parallel job.
- Important component of a parallel system software stack.
- Simulations are sensitive to collectives performance characteristics.
- Simulations spend significant amount of time in collectives.
Examples

Part of messaging layer
• OpenMPI – Tuned, Cheetah
• PAMI
• OpenSHMEM (reference implementation)

Standalone Libraries
• LibNBC
• HCOL (derived from Cheetah)
• FCA
Case Study: Cheetah

Objectives

• Develop a high-performing and highly scalable collective operations library for multicore systems
• Develop collective offload mechanism for InfiniBand HCA
• Designed to support blocking and nonblocking semantics, and achieve high scalability and performance on modern multicore systems

http://www.csm.ornl.gov/cheetah
Our Approach: Hierarchical Collectives

• A collective operation is a combination of multiple (layered) collective primitives.

• Group processes into multiple hierarchies to leverage architecture capabilities

• Build a collective by combining these basic collective primitives

• Progress independent collective primitives concurrently

![Diagram of hierarchical collectives](image)
Definitions: Reduction Operations

• Allreduce Operation: Combines the data from all participants with an operation, and distributes the results of the operation to all participants

• Reduce Operation: Combines the data from all participants with an operation, and the result is available only at the root
Hierarchical Allreduce Collective Operation

• A hierarchical allreduce is implemented as a combination of the reduce, allreduce, and broadcast primitives

• n level Allreduce is a combination of three primitives
  - Reduce (first n-1 levels), Allreduce(nth level), Broadcast (first n-1 levels)

• Example: Interaction of the collective primitives in a 8-process allreduce collective operation
Recursive K’ing Algorithm for Implementing Allreduce Primitive

Communication Pattern
Step 1

Data after Step 1

Communication Pattern
Step 2

Data after Step 2
Cheetah: A Framework for implementing hierarchical collectives
Driver

Provides low level messaging functionality

- For RDMA networks it provides – RDMA Read, Write, Atomic interfaces
- Provide thin layer of active messages on RDMA network

The functionality is accessible through well-defined interface

- The interfaces are well-defined, but might change more rapidly

The lowest interface to the network interface

- It is not typically portable across the same hardware or different hardware
Examples

Verbs
• Driver for InfiniBand

uGNI
• Driver for Cray’s Gemini and Aries NICs
• Primarily used for implementing MPI

DMAPP
• Driver for Cray’s Gemini and Aries NICs
• Primarily used for implementing PGAS models
Case Study: Verbs

What is Verbs?
- A low level abstraction of the network device, and is close to bare metal for many HCAs
- Provides RDMA functionality for the upper-layer protocols

Why use Verbs?
- Low-level interface - for portability, performance and scalability
- Provides kernel bypass
Case Study: Verbs (Continued)

Constructs
• **Objects**: QPs, SQ, RQ, CQ, Memory Region
• **Functions**:
  Control Functions: Create, Destroy, Modify, Query, Work with events
  Data Functions: Post Send, Recv, Poll CQ, Request for completion event
Example of Message Transfer using Verbs on InfiniBand
References

- [https://zcopy.wordpress.com/2010/10/08/quick-concepts-part-1––introduction-to-rdma/](https://zcopy.wordpress.com/2010/10/08/quick-concepts-part-1––introduction-to-rdma/)
Stack Trace: shmem_put call (First Packet)

#0 0x00007f2694bcc160 in ibv_create_qp ()
#1 0x00007f2694e0040c in qp_create_one ()
#2 0x00007f2694e001dd in qp_create_all ()
#3 0x00007f2694dff7af in oob_module_start_connect ()
#4 0x00007f2694df43fa in check_endpoint_state ()
#5 0x00007f2694df41ef in mca_tl_openib_put_short_nb ()
#6 0x00000000000403352 in uccs_put_contiguous_short_nb ()
#7 0x00000000000403169 in uccs_put ()
#8 0x00000000000403049 in __shmem_comms_put ()
#9 0x00000000000403a2 in shmem_int_put ()
#10 0x00000000000403f7c in shmem_int_p ()
#11 0x00000000000402054 in main ()
Stack Trace: shmem_put call (Subsequent Packets)

#0  ibv_post_send ()
#1  0x00007fe21fe3d32e in mca_tl_openib_put_short_nb ()
#2  0x00000000000403366 in uccs_put_contiguous_short_nb ()
#3  0x0000000000040317d in uccs_put ()
#4  0x0000000000040305d in __shmem_comms_put ()
#5  0x000000000004034b6 in shmem_int_put ()
#6  0x00000000000403f90 in shmem_int_p ()
#7  0x00000000000402068 in main ()
Acknowledgments

This work was supported by the United States Department of Defense & used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.
Latest in OpenSHMEM: Specification, API, and Programming

Presenters:
Graham Lopez, Dounia Khaldi, Pavel Shamis, Manjunath Gorentla Venkata