SHMEM/PGAS Developer Community Feedback for OFI Working Group

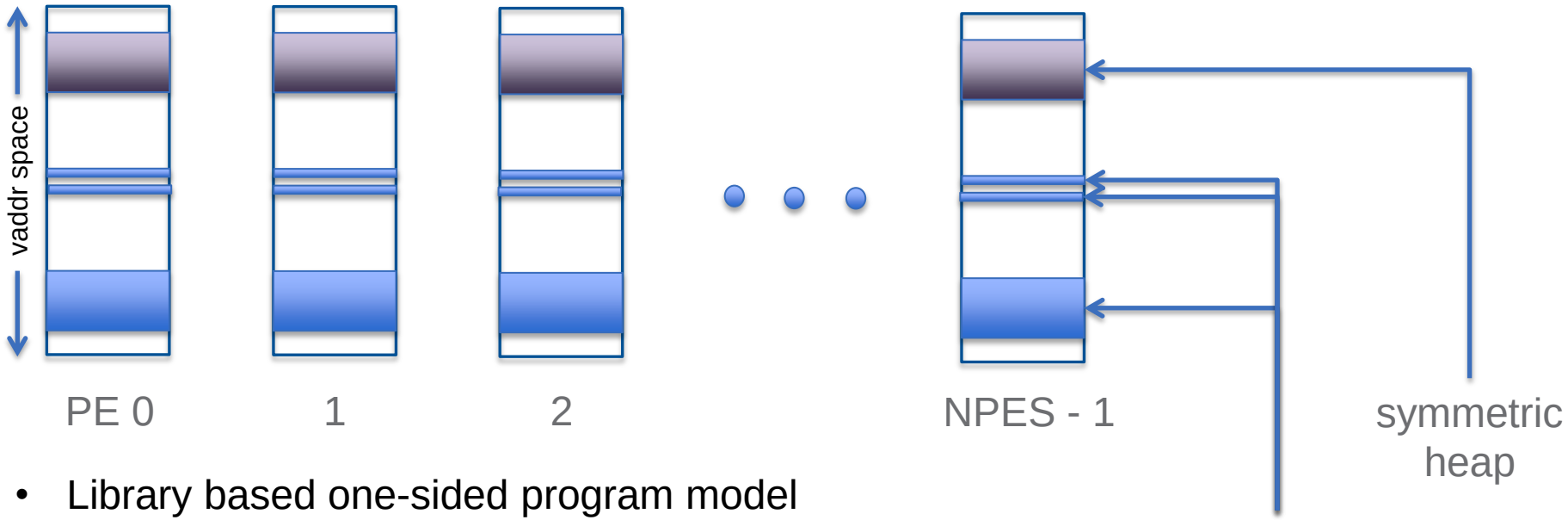
#OFADevWorkshop

Howard Pritchard, Cray Inc.

Outline

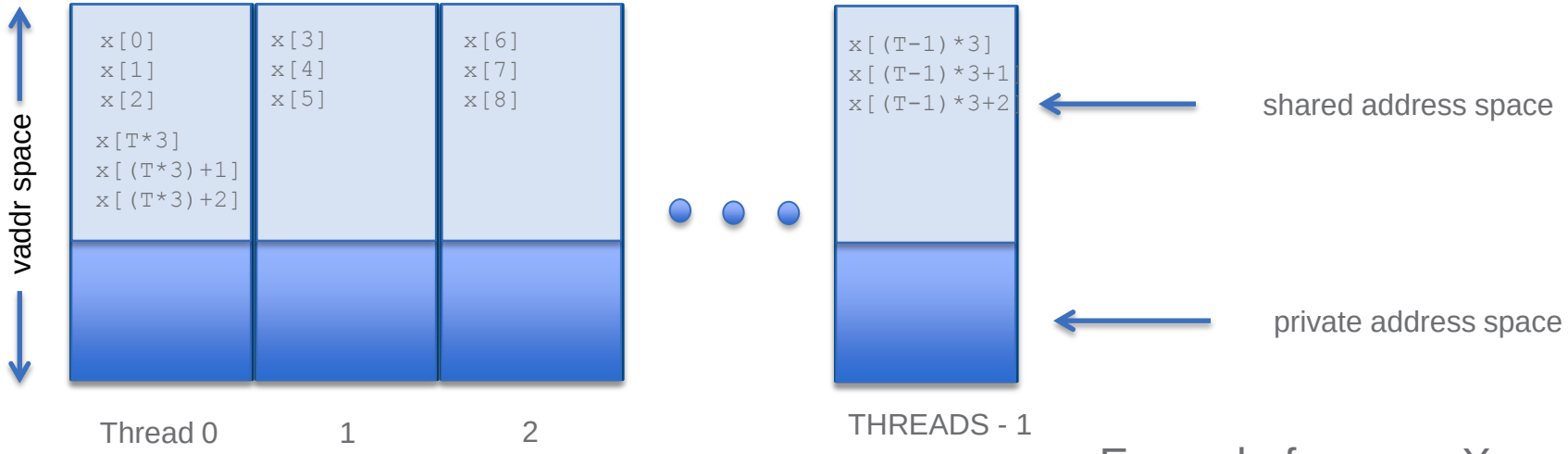
- Brief SHMEM and PGAS intro
- Feedback from developers concerning API requirements
  - Endpoint considerations
  - Memory registration
  - Remote memory references
  - Collectives
  - Active messages
- On going work

(Open)SHMEM



- Library based one-sided program model
- All ranks (PEs) in the job run the same program (SPMD)
- Only objects in *symmetric regions* – data segment(s), symmetric heap – are guaranteed to be remotely accessible
- Various vendor specific variations/extensions
- Somewhat archaic interface (think Cray T3D), being modernized as part of OpenSHMEM effort

UPC



- Compiler based program model – ‘c’ with extensions
- Each thread has affinity to a certain chunk of shared memory
- Objects declared as *shared* are allocated out of shared memory
- Objects can be distributed across the chunks of shared memory in various ways – blocked, round robin, etc.
- Collective operations, locks, etc.
- Like MPI, evolving over time
- Thread *may* map to SHMEM PE rank enumeration

Example for array X as declared below:

```
shared [3] int x[(T*3)+3]
where
T = THREADS
```

Fortran 2008 (CoArrays or CAF)

- Also compiler based one-sided program model
- CoArray construct part of Fortran 2008 standard
- Like SHMEM, currently supports only SPMD model
- Address space model closer to SHMEM than UPC
- But there is a significant complication with Fortran 2008 (and likely in future versions of UPC). Basically with Fortran 2008 not a clean separation between "shared" and "private" address spaces.

Community Feedback

SHMEM/PGAS API Requirements – caveats and disclaimers

- Assume significant degree of overlap with needs of MPI community, e.g.
  - Similar needs with respect to supporting *fork*
  - Similar needs concerning *munmap* if memory registration/deregistration needs to be managed explicitly by the SHMEM/PGAS runtime
- What are covered here are API requirements more particular to SHMEM/PGAS and similar one-sided program models
- As with MPI there are differences of opinion how best to implement runtimes to support these program models – some preferring active message style models, possibly with some RDMA offload – while others prefer a more direct-on-top-of-rdma primitives approach

API Endpoint Considerations

- Endpoint memory usage needs to be scalable
- Low overhead mechanism for enumeration of endpoints, i.e. "ranks" rather than lids, etc.
- It would be great to have connectionless (yet reliable)-style endpoints
- If connected-style endpoints are the only choice, methods to do both on-demand and full-wire up efficiently would be great
- API that supports "thread hot" thread safety model for endpoints.

Memory Registration API Requirements (1)

- Necessary evil?
  - Some say yes if needed for good performance
- Scalability of memory registration is a very important property
  - API should be designed to allow for cases where only local memory registration info is required to access remote memory
  - Does this lead to security issues? Have to have some protection mechanism.
- An idea - let application supply “r_key” values to use
- Growable (both down and up) registrations would also be useful (mmap example)

Memory Registration API Requirements (2)

- On-demand paging option requirements
  - Want flexibility to do on-demand paging when requested, but may also want “pinned” pages method, specified by application
  - PGAS compiler has to have control of memory allocation to avoid needing this – but often compiler does have control of heap allocator, etc.
  - Fortran 2008, possible future versions of UPC may still find this useful
  - Maybe also be helpful for library based one-sided models, esp. if specifications relax current restrictions on what memory is remotely accessible become more relaxed

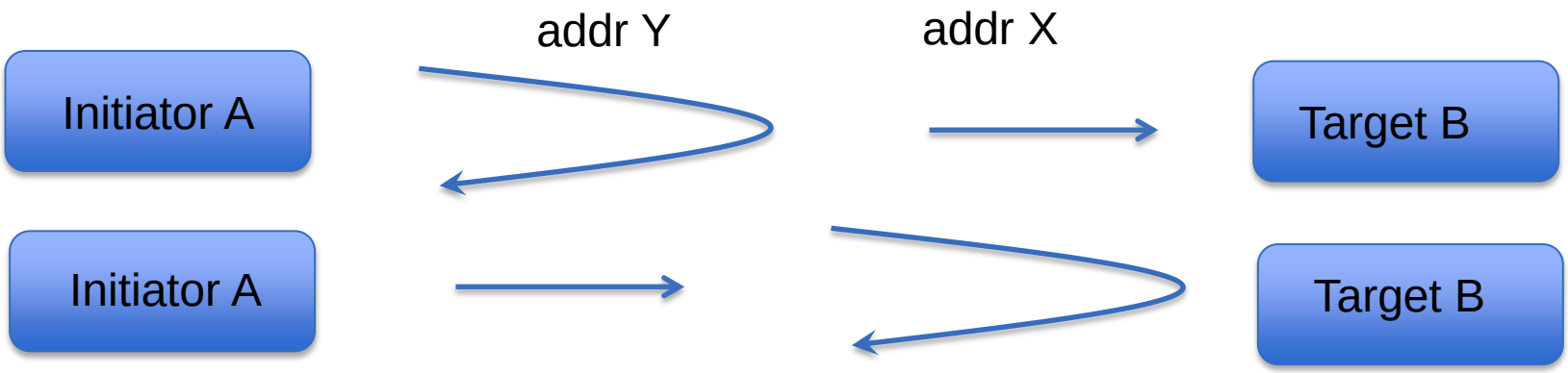
- Fortran 2008 (CoArray) in particular could benefit from ability to register large amounts of virtual memory that may be only sparsely populated

Small remote memory reference API requirements – perf and ops

- PGAS compilers implemented directly on top of native RDMA functionality need an API that can deliver high performance for small non-blocking PUTs (stores) and GETs (loads). Typical remote memory accesses much smaller than for MPI programs and many more of them
- Atomic memory ops are important
- Put with various completion notification mechanisms – more on this in a later slide
- Small *partially* blocking put requirement (till safe to reuse local buffer)
  - Shmem on top of portals4/non-blocking puts semantics example – cost of implementing partially blocking on top of non-blocking?

Small remote memory reference API requirements - ordering

PGAS compilers in particular have a special ordering requirement: Need to be able to correctly handle WAW, WAR, and RAW from a given initiator to a given target address:



Problem is one getting correctness while maintaining performance. If X != Y, then no need to order operations, only if X == Y is ordering necessary. For Cray compiler, its been found for most PGAS apps X != Y is the far more common case.

Small remote memory reference API requirements – Atomic memory ops

- Rich set of AMOs also useful – need more than FADD and CSWAP.
- Multi-element AMOs for active message support, etc.
- Nice to have AMOs that can support MCS lock algorithm at scale – implies possibly needing 128 bit AMOs
- At least two kinds of AMO performance characteristics:
  - Low latency but reliable (either fail or succeed, result being reported back to initiator). This allows use of locks, queues, etc. without giving up on resilience to transient network errors.
  - "At memory" computation, but only need a good enough answer. Throughput more important than reliability. Example is GUPS.
- 32/16 bit granularity ops would be useful in addition to 64 bit and possibly 128 bit.
- AMO cache (on NIC) coherency issues - may need functionality in any API for this.

Remote memory reference API requirements – completion notification

- Lightweight completion notification is very desirable, especially for PUTs.
  - Put with flag (delivered at target after payload)
  - Counter-like completion mechanism at target
  - At initiator side want notification of local completion (safe to reuse buffer), and global completion (safe to tell another process it can access the data at the target node).
- Ideally allow for batching of groups of PUT/GET requests with a single completion notification at the initiator.
- Get completion information at target of the get operation:
  - Data in the get buffer has been read and heading over the wire, i.e. target can reuse the buffer
  - Data has arrived in initiator's memory

Other Requests

- Option for fences between RDMA transactions – already there?
- Per transfer network ordering options would be great
- For large RDMA writes - piggyback message data (more than 32 bits of imm data) coming along with bulk data – Gasnet request – handler invocation

Collectives

- Would be nice to not have to reinvent the wheel for multiple, often concurrently used program models, e.g. app using SHMEM and MPI
  - A lower level common interface for frequently used collective operations – barrier, reductions, coalesce (allgather), alltoall
  - Flat would be better, not have to do special on-node (within a cache coherent domain) operations within the SHMEM/PGAS implementation

Active Message Support

- Two general types of uses
  - Lower performance need for RPC-like capabilities implied by some types of operations, e.g. upc_global_alloc
  - Higher performance needed for PGAS languages implemented using an Active Message paradigm, as well as other active message based program models like Charm++
- API should support sending of requests to pre-initialized queues, for which the target has registered callback functions to process the data sent from initiator. Payload can be restricted to small size ~256 bytes or less.
- Initiator of message should get response back that message has arrived, optionally that message has been consumed by callback function
- Implementation needs to be able to handle transient network errors, message is delivered once and only once to target
- Some applications may require ordering of messages from a given initiator to given target, would be good to be able to specify this at queue initialization.
- Flow control is important.

On Going Work

- Working with DOE Office of Science Co-design teams to collect additional, future requirements particularly for new program models like Legion.

- Soliciting focused input from Charm++, PAMI end users, etc.

References

- http://openshmem.org/
- **BM Parallel Environment Runtime Edition Version 1 Release 2: PAMI Programming Guide (SA23-2273-03)**
- **Using the GNI and DMAPP APIs**
- http://www.cs.sandia.gov/Portals/portals4-spec.html
- http://www.openfabrics.org/downloads/OFWG/

Thank You

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

This material was assembled with help of the following organizations/people

Los Alamos National Lab
- Latchesar Ionkov
- Ginger Young

Oak Ridge National Lab
- Steve Poole
- Pavel Shamis

Sandia National Lab
- Brian Barrett

Intel
- David Addison
- Charles Archer
- Sayantan Sur

Mellanox
- Liran Liss

Cray
- Monika ten Bruggencate
- Howard Pritchard (scribe)

Input was also obtained from Paul Hargrove (LBL) and Jeff Hammond (ANL), and others.

UPC/Fortran 2008(2)

```
   .
#include <upc.h>
   .

long *shared sh_ptr[THREADS];
long *my_mallocd_array = NULL;
long *thread_zeros_array = NULL;

if (MYTHREAD == 0) {
   my_mallocd_array = (long *)malloc(1000 * sizeof(long));
   sh_ptr[0] = my_mallocd_array;
}
upc_barrier(100);
if (MYTHREAD == 1) {
   thread_zeros_array = sh_ptr[0];
   for (int i=0;i<10;i++) thread_zeros_array[i] = 0xdeadbeef;
}
upc_barrier(101);
if (MYTHREAD == 0) {
    for (int i=0;i<10;i++) assert(my_mallocd_array[i] == 0xdeadbeef);
}
```

Array of shared pointers to private long (likely preregistered with NIC)

private pointers to private long

Compiler translates this into an underlying RDMA get(load), including retrieval of mem reg info, etc.

Compiler translates into RDMA put

This is not currently a legal UPC code example, but Fortran 2008 equivalent is. Just did not want to use Fortran for example.

Different Views of PGAS – implementing and using

Active Message Based Implementations

implementer viewpoints

Pure (almost) one-sided implementations

Productivity more important than performance

user viewpoints

Want to use SHMEM, etc. inside MPI app for performanc reasons

Expect SHMEM, etc. to beat MPI on performance

Backup Material

(Open)SHMEM - example

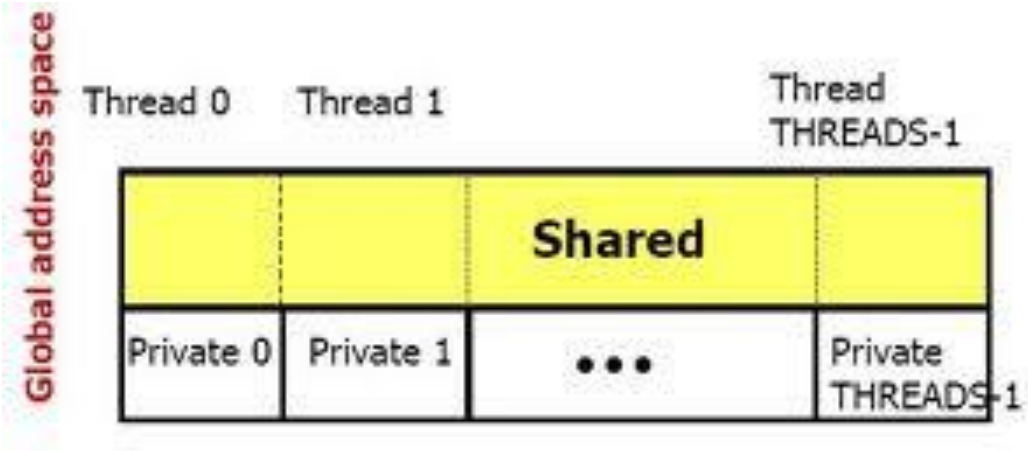
```
        .
        .
        .
#include <mpp/shmem.h>

long target_array[1000];

int main(int argc,char **argv)
{
       int i, my_pe, n_pes, r_neighbor, l_neighbor;
       int n_elems;
       long *source_array;

       shmem_init();

       n_elems = sizeof(target_array)/sizeof(long);
       my_pe = shmem_my_pe();
       n_pes = shmem_n_pes();
       r_neighbor = (my_pe + 1) % n_pes;
       l_neighbor = (my_pe + n_pes - 1) % n_pes;

       source_array = (long *)malloc(sizeof(long) * n_elems);
       for (i=0;i<n_elems;i++) source_array[i] = (long) my_pe;

       shmem_barrier_all();
       shmem_put64(target_array,source_array,1000,r_neighbor);
       shmem_barrier_all();

       for (i=0;i<n_elems;i++)
            if(target_array[i] != l_neighbor) printf("something's wrong\n");

       shmem_finalize();
       return(0);
}
```

vendor specific

symmetric address – array in data segment

Barrier – process and memory synchronization

UPC



- Compiler based program model – ‘c’ with extensions
- Each thread (PE in SHMEM model) has affinity to a certain chunk of shared memory
- Objects declared as *shared* are allocated out of shared memory
- Objects can be distributed across the chunks of shared memory in various ways – blocked, round robin, etc.
- Collective operations, locks, etc.
- Like MPI, evolving over time

Small remote memory reference API requirements – AMO survey

Operation	IBM BG/Q	Cray XC	Quadrics SHMEM	IB
swap	x	x	x	-
comp & swap	x	x	x	x
masked swap	x	x (AFAX)	x	-
add	x	x	x	x
bitwise or	x	x	-	-
bitwise and	x	x	-	-
comp & or	x	-	-	-
comp & add	x	-	-	-
comp & or	x	-	-	-
comp & and	x	-	-	-
comp & xor	x	-	-	-
min	-	x	-	-
max	-	x	-	-

Small remote memory reference API requirements - ordering

PGAS compilers in particular have a special ordering requirement:

Many PGAS compilers can benefit greatly from hardware which provides protection against WAW, WAR, and RAW hazards for remote memory references from a given initiator to a given target and address in the target's address space.

app knows g_idx has no overlap between different threads, but possible repeat of indices for one thread, so no need for locks, etc. in update loop

```
    .
#include <upc_relaxed.h>
    .

void update_shared_array(shared long *g_array, long *local_data, int *g_idx, int nupdates)
{
    int i;

    for (i=0;i<nupdates;i++) {
         g_array[g_idx[i]] += local_data[i];
    }

    upc_barrier(0);

    if(MYTHREAD == 0) {
        printf("Done with work\n");
    }
}
```

Problem for compiler is here. What if for some n,m it is the case that g_idx[n] == g_idx[m]. See notes.