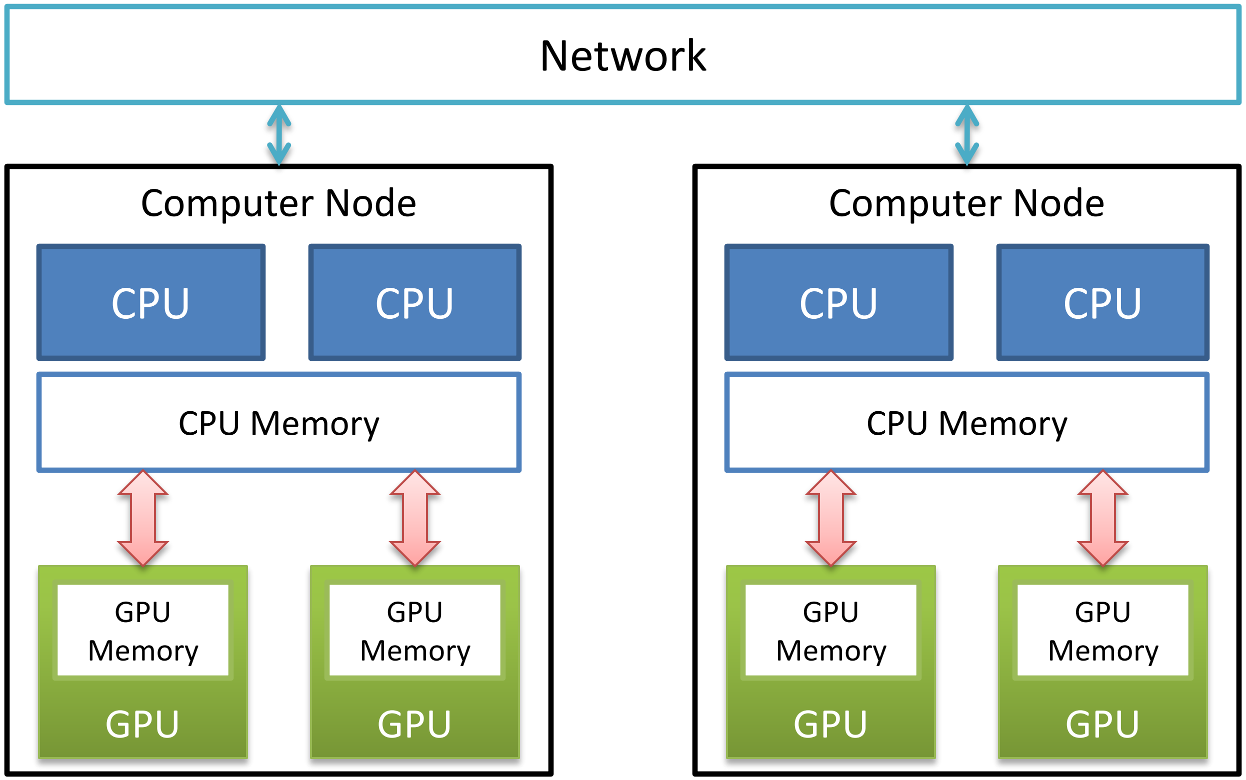
Extending Unified Parallel C for GPU Computing

Yili Zheng, Costin Iancu, Paul Hargrove,
Seung-Jai Min, Katherine Yelick

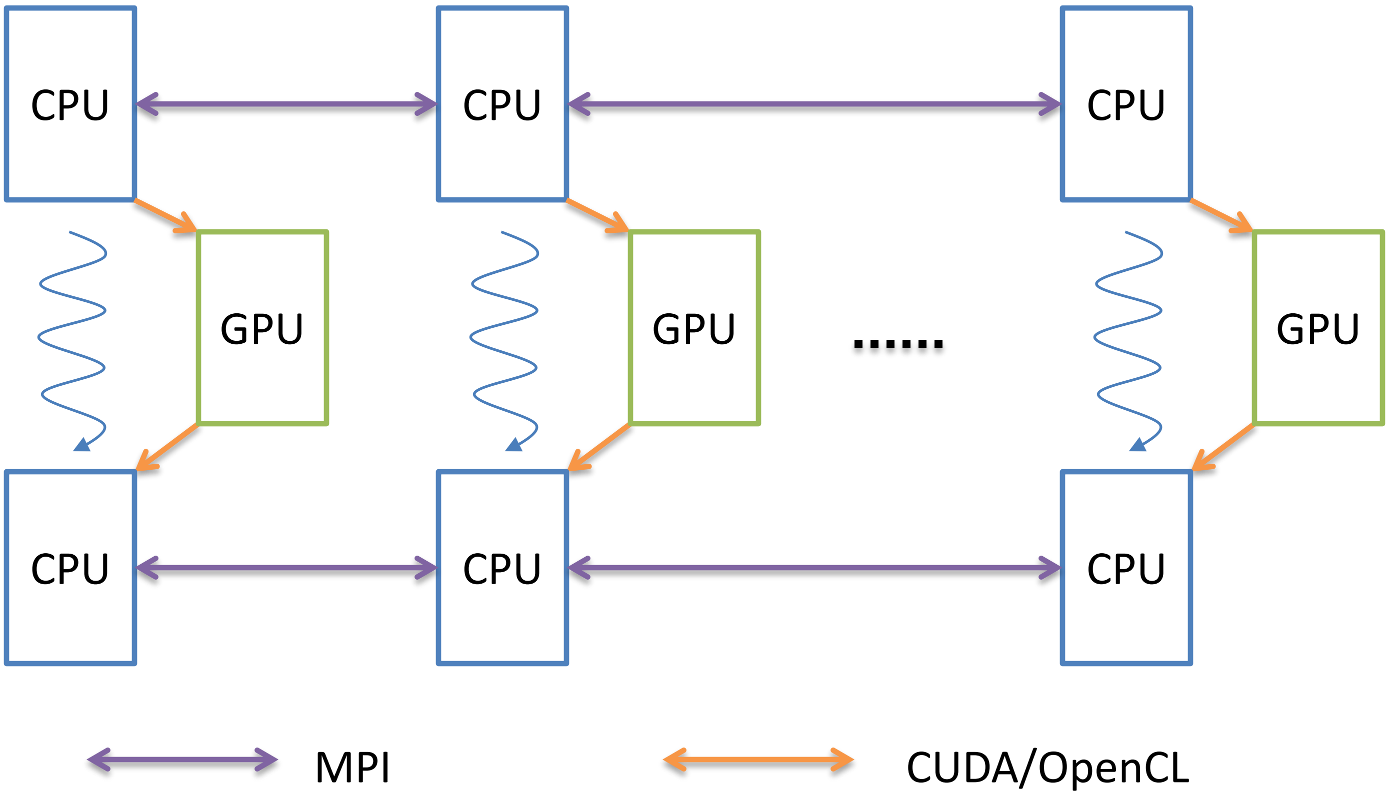
Lawrence Berkeley National Lab

GPU Cluster with Hybrid Memory

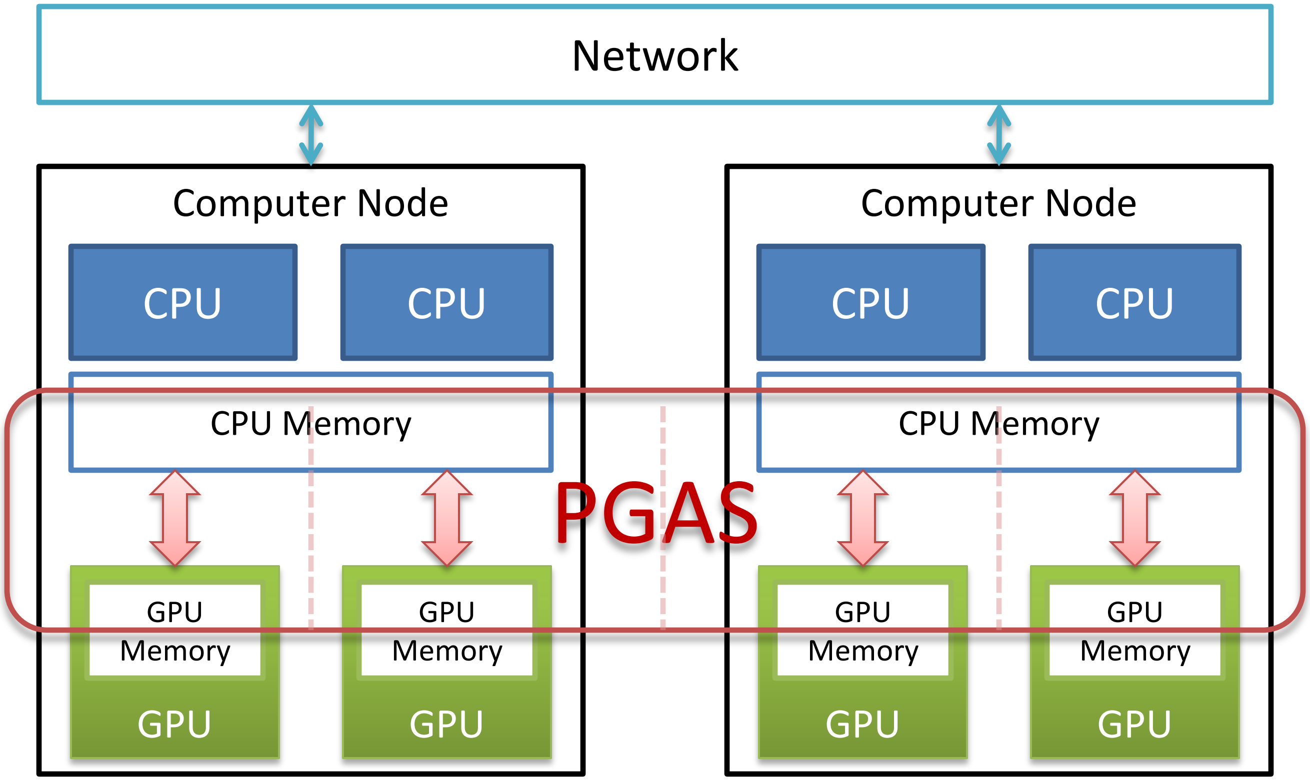


Current Programming Model for GPU Clusters

MPI + CUDA/OpenCL



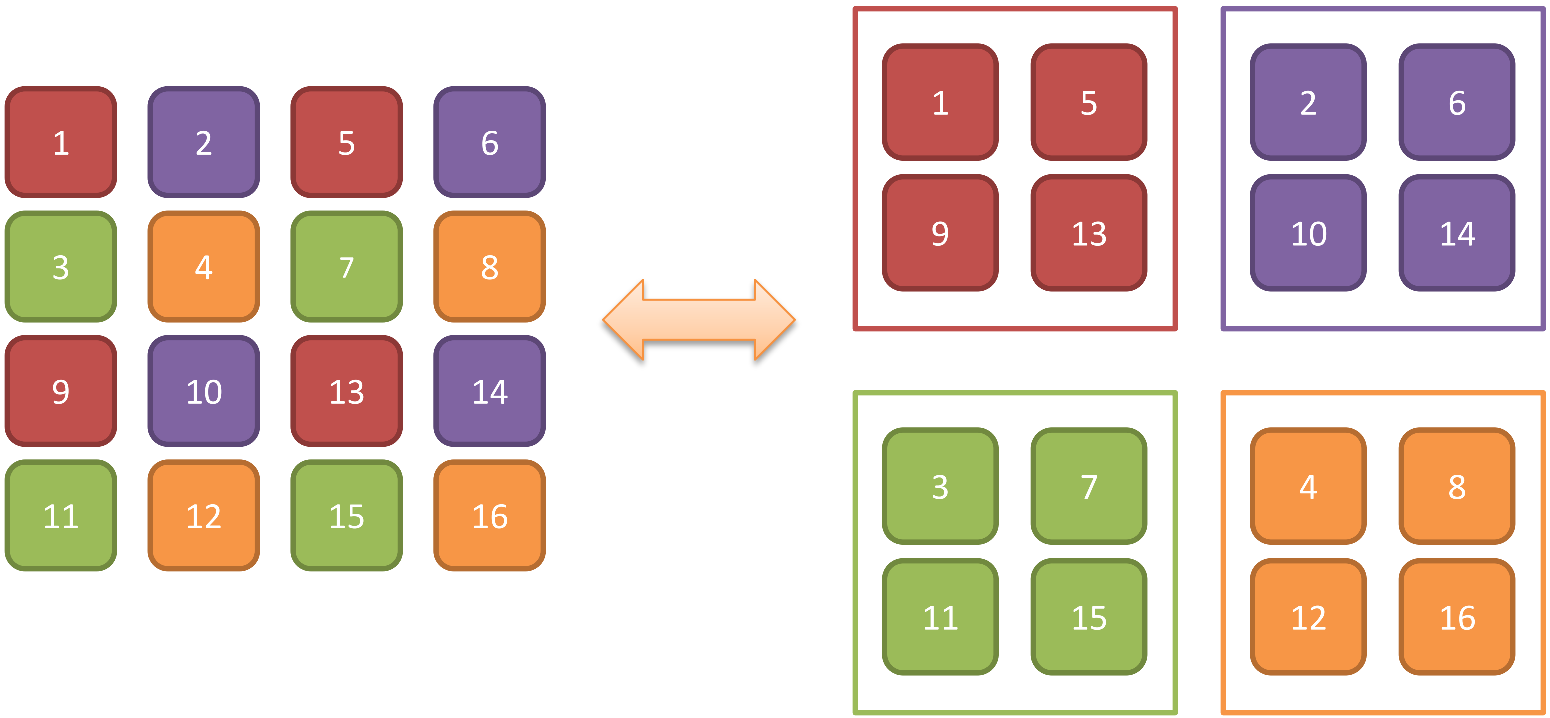
PGAS Programming Model for Hybrid Multi-Core Systems



PGAS Example: Global Matrix Distribution

Global Matrix View

Distributed Matrix Storage

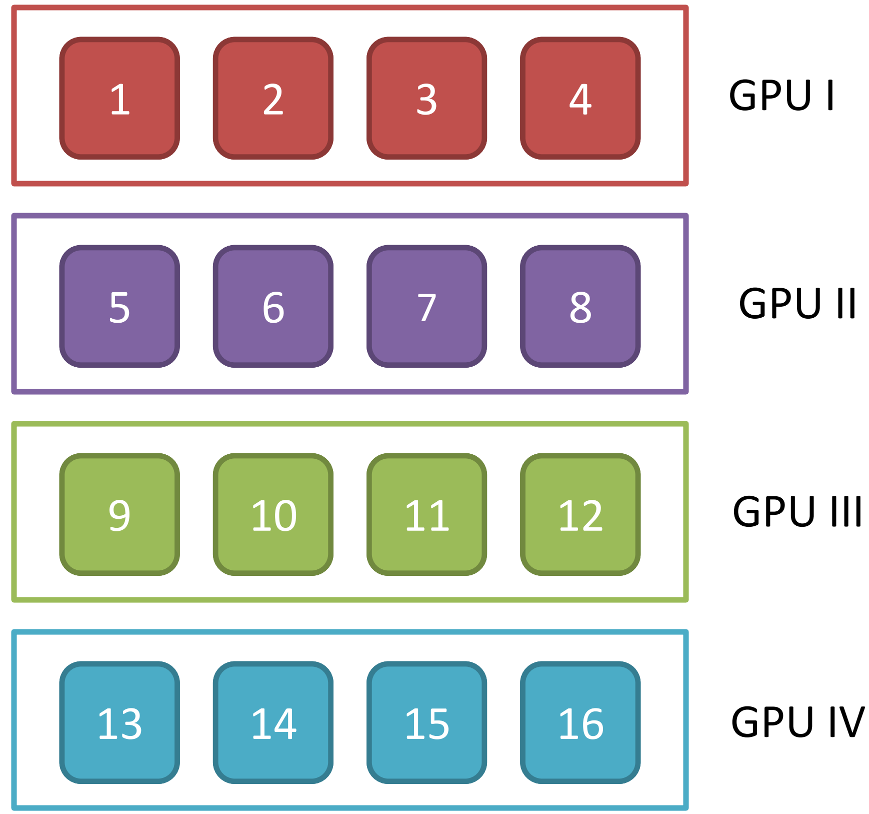


PGAS Example: Fast Fourier Transform

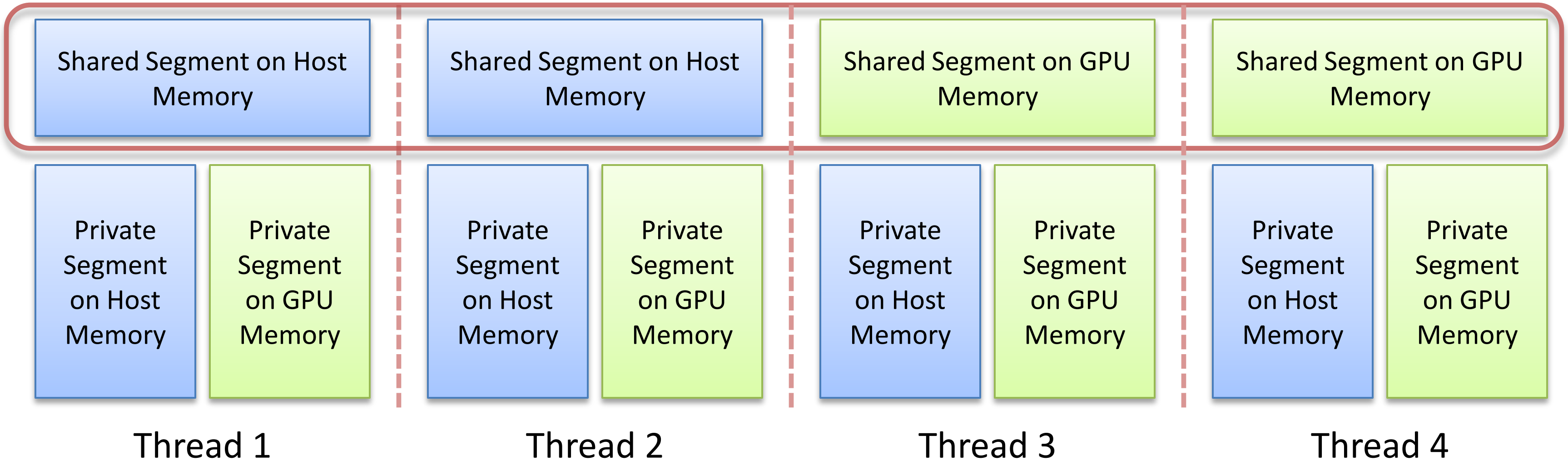
Global Matrix View


```
Shared [4] A[4][4];
1D_FFT(A); // row fft
A'=A^T;
1D_FFT(A'); // col fft
A= A'^T
```

Distributed Matrix Storage



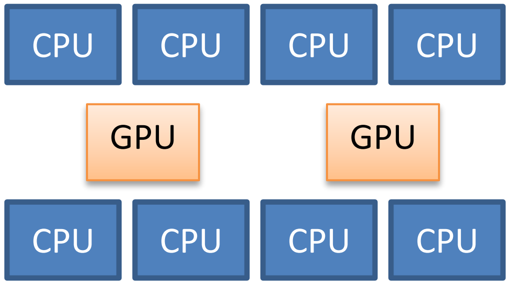
Hybrid Partitioned Global Address Space



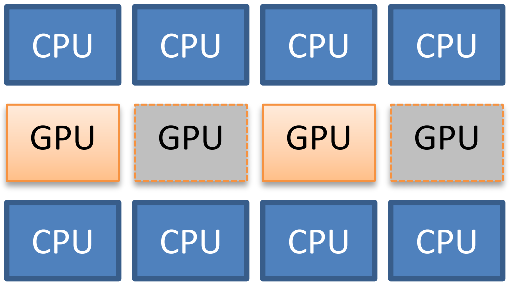
- Each thread has only one shared segment, which can be either in host memory or in GPU memory, but not both.
- Decouple the memory model from execution models; therefore it supports various execution models.
- Backward compatible with current UPC and CUDA/OpenCL programs.

Execution Models

- Synchronous model



- Virtual GPU model

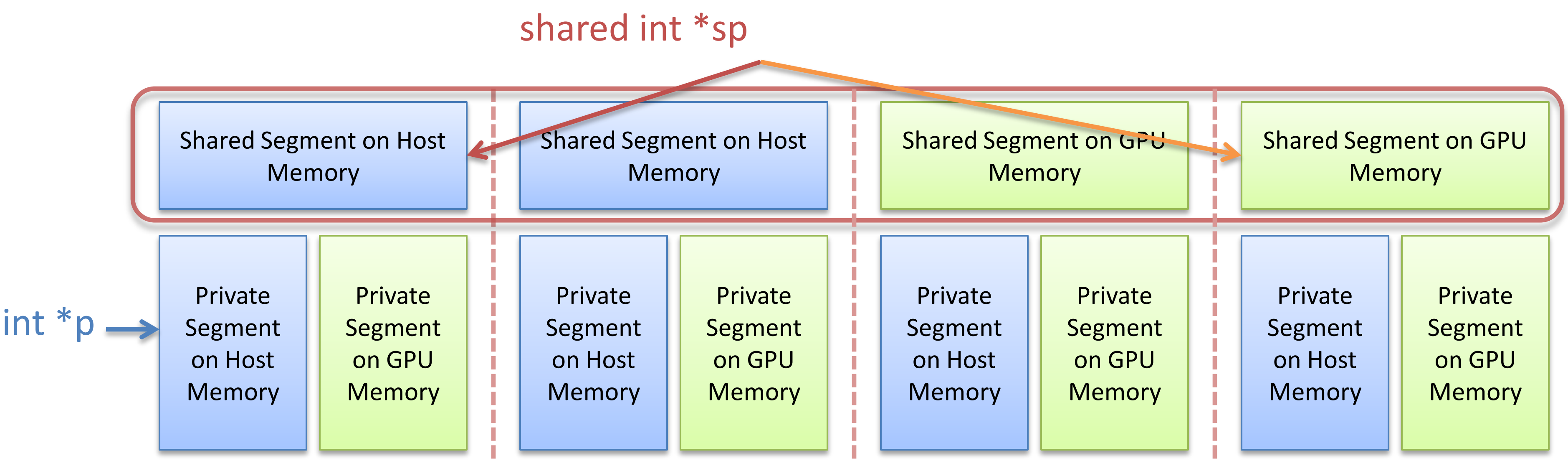


- Hybrid model

UPC Overview

- PGAS dialect of ISO C99
- Distributed shared arrays
- Dynamic shared-memory allocation
- One-sided shared-memory communication
- Synchronization: barriers, locks, memory fences
- Collective communication library
- Parallel I/O library

Hybrid PGAS Example



	Thread 1	Thread 2	Thread 3	Thread 4
Standard C	p = malloc(4)	*p STOP	*p STOP	*p STOP
UPC	sp = upc_alloc(4)	*sp ✓	*sp ✓	*sp ✓
UPC with GPU shared heap	*sp ✓	*sp ✓	*sp ✓	sp = upc_alloc(4)

Data Transfer Example

- MPI+CUDA copy *nbytes* data from *src* on GPU 1 to *dst* on GPU 2

```
Process 1:                                Process 2:
send_buffer = malloc(nbytes);             recv_buffer = malloc(nbytes);
cudaMemcpy(send_buffer, src);
MPI_Send(send_buffer); <- - - - - - - ->  MPI_Recv(recv_buffer);
free(send_buffer);                        cudaMemcpy(dst, recv_buffer);
                                          free(recv_buffer);
```

- UPC with GPU extensions

```
Thread 1:                                 Thread 2:
upc_memcpy(dst, src, nbytes);             // no operation required
```

Advantages of PGAS on GPU clusters

- Don’t need explicit buffer management by the user.
- Facilitate end-to-end optimizations such as data transfer pipelining.
- One-sided communications map well to DMA transfers for GPU devices.
- Concise code

PGAS GPU Code Example

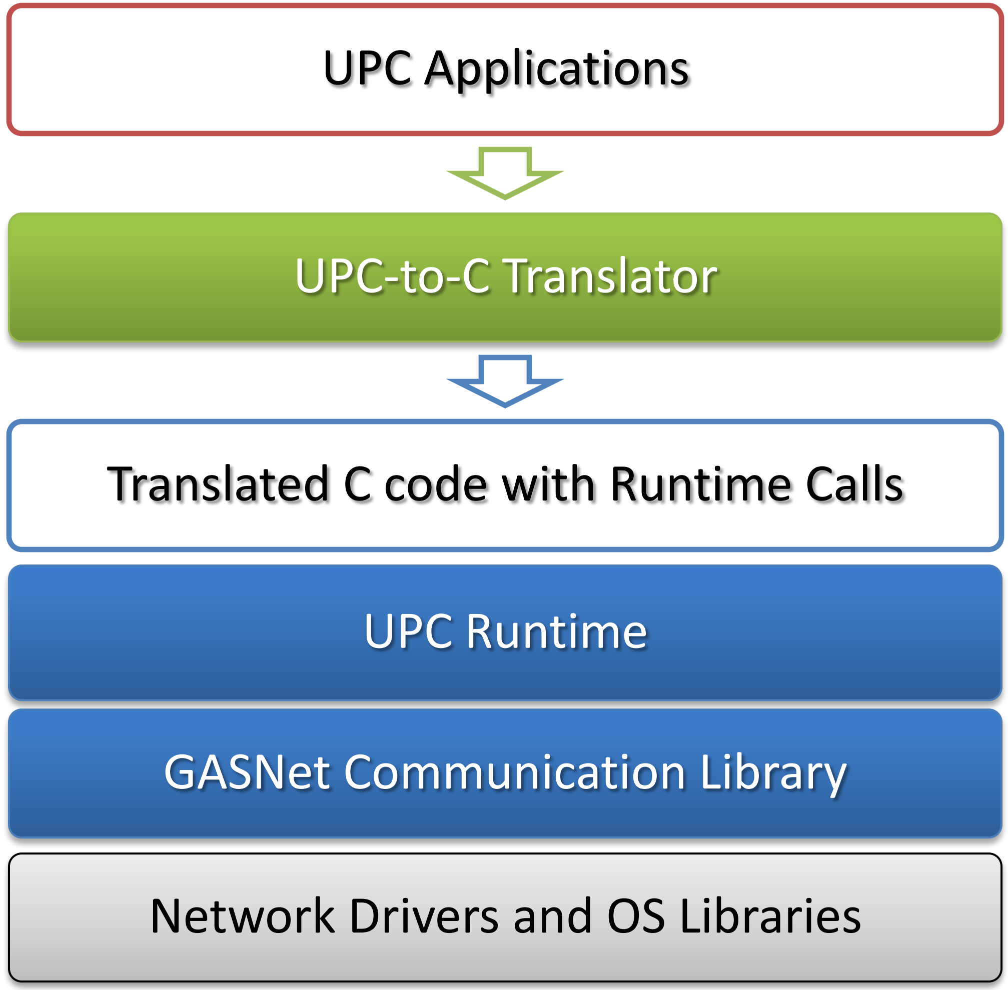
- Thread 1

```
// use GPU memory for shared segment
bupc_attach_gpu(gpu_id);

shared [] int * shared  sp;

// shared memory is allocatd on GPU
sp = upc_alloc(sizeof(int));
*sp = 4; // write to GPU memory
upc_barrier;   <------------------------------>
```

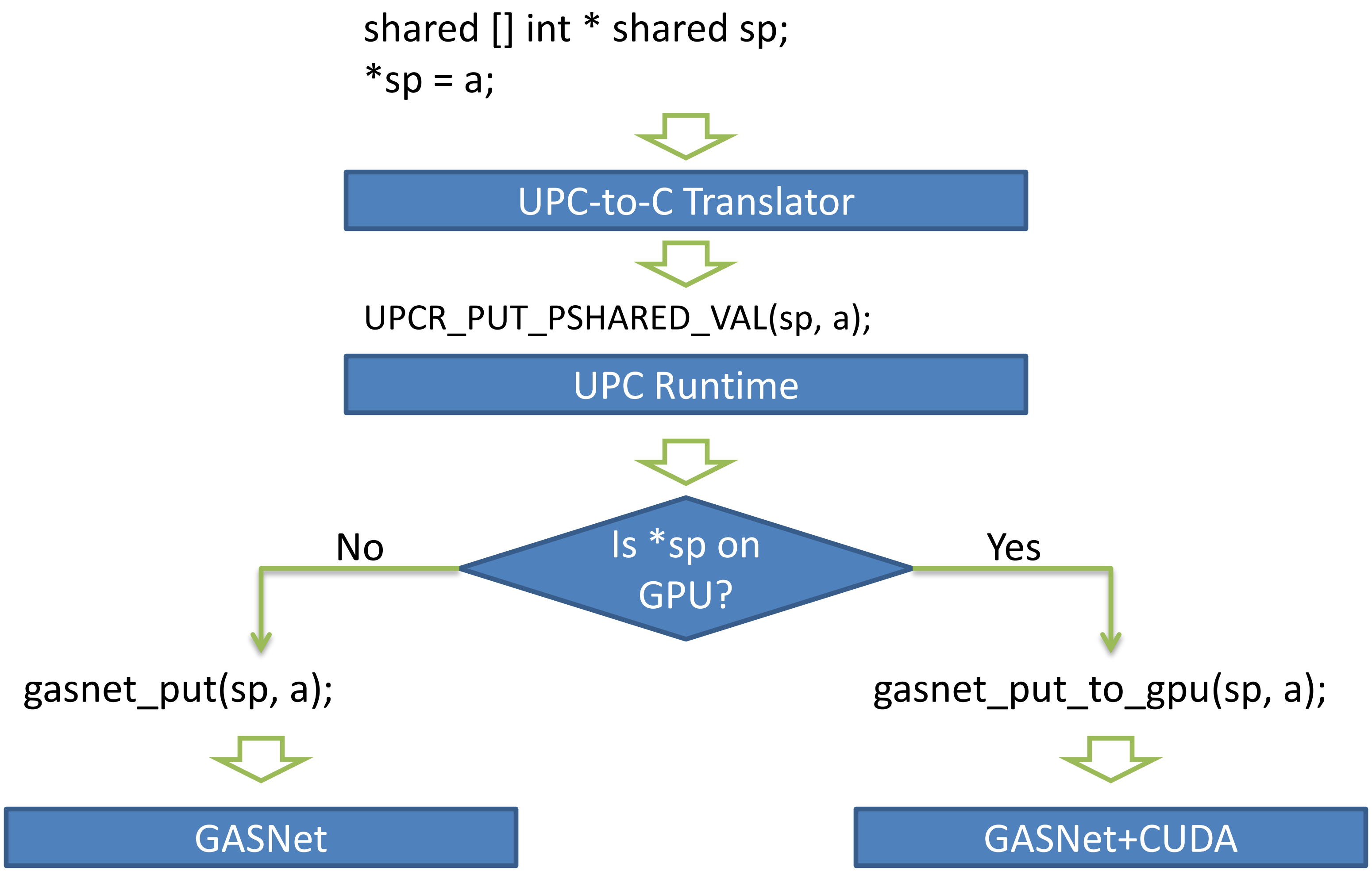
- Thread 2

```
shared [] int * shared sp;

upc_barrier;
// read from remote GPU memory
printf(“%d”, *sp);
```

Berkeley UPC Software Stack

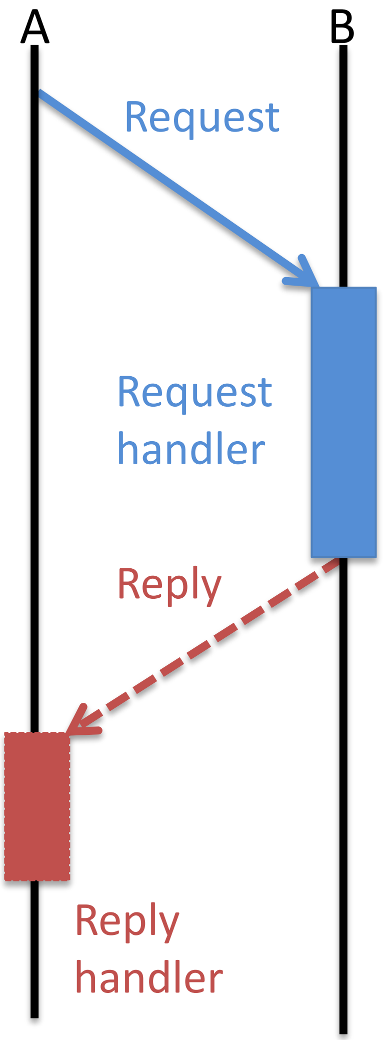


Translation and Call Graph Example

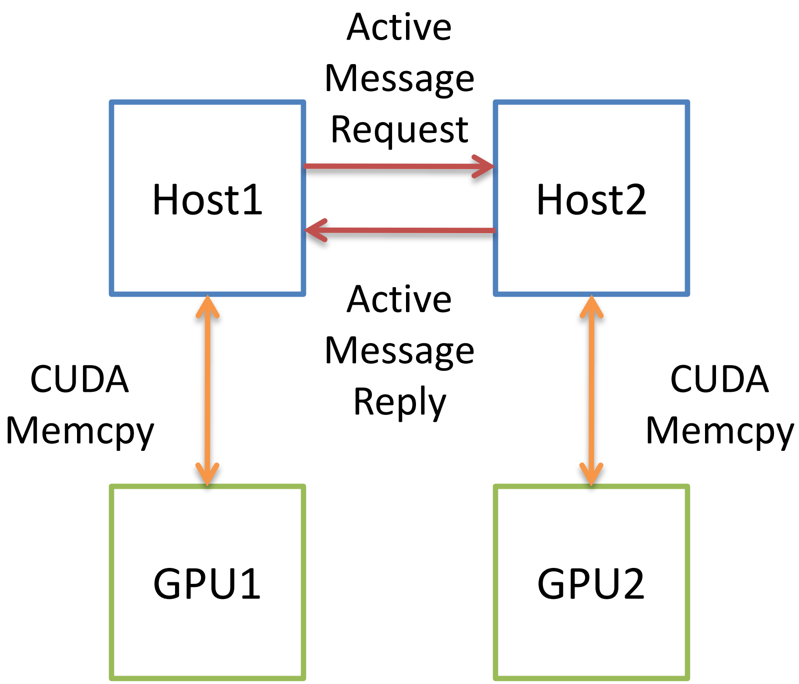


Active Messages

- Active messages = Data + Action
- Key enabling technology for both one-sided and two-sided communications
  - Software implementation of Put/Get
  - Eager and Rendezvous protocols
- Remote Procedural Calls
  - Facilitate “owner-computes”
  - Execute asynchronous tasks



GASNet Extensions for GPU One-sided Communication



- How to transfer to/from remote GPU device memory?
  - Active Messages (AM)
  - Need to execute CUDA operations outside of AM handler context because they may block
  - Solution: asynchronous GPU task queue
- How to know when the data transfer is done?
  - Send an ACK message after the GPU op is done on the GPU device
  - Solution: GPU task queue polling and callback support

Implementation

- UPC-to-C translator
  - No change because the UPC runtime API is intact.
  - Compile UPC code and CUDA code separately and then link the object files with libs together.
- UPC runtime extensions
  - Shared-heap management for GPU device memory
  - Accesses to shared data on GPU (via pointer-to-shared)
  - Interoperability of UPC and GPU (CUDA)
- GASNet extensions
  - Put and Get operations for GPU
  - Asynchronous GPU task queue for running GPU operations outside of AM handler context

Summary

- Runtime extensions for enabling PGAS on GPU clusters
  - Unified API for data management and communication
  - High-level expressions of data movement enabling end-to-end optimizations
  - Compatible with different execution models and existing GPU applications
- Reusable modular components in the implementation
  - Task queue for asynchronous task execution
  - Communication protocols for heterogeneous processors
  - Portable to other GPU SDK, e.g., OpenCL. Platform (CUDA) specific codes are limited and encapsulated.
- Work in progress

Thank You!

- MS 31

  UPC at Scale

  1:20 PM - 3:20 PM

  Room: Leonesa II

- MS 52

  Getting Multicore Performance with UPC

  1:20 PM - 3:20 PM

  Room: Eliza Anderson Amphitheater