DMAPP

An API for one-sided program models on Baker systems

Monika ten Bruggencate

Cray Inc,

Harvey Richardson

Cray Exascale Research Initiative

Overview

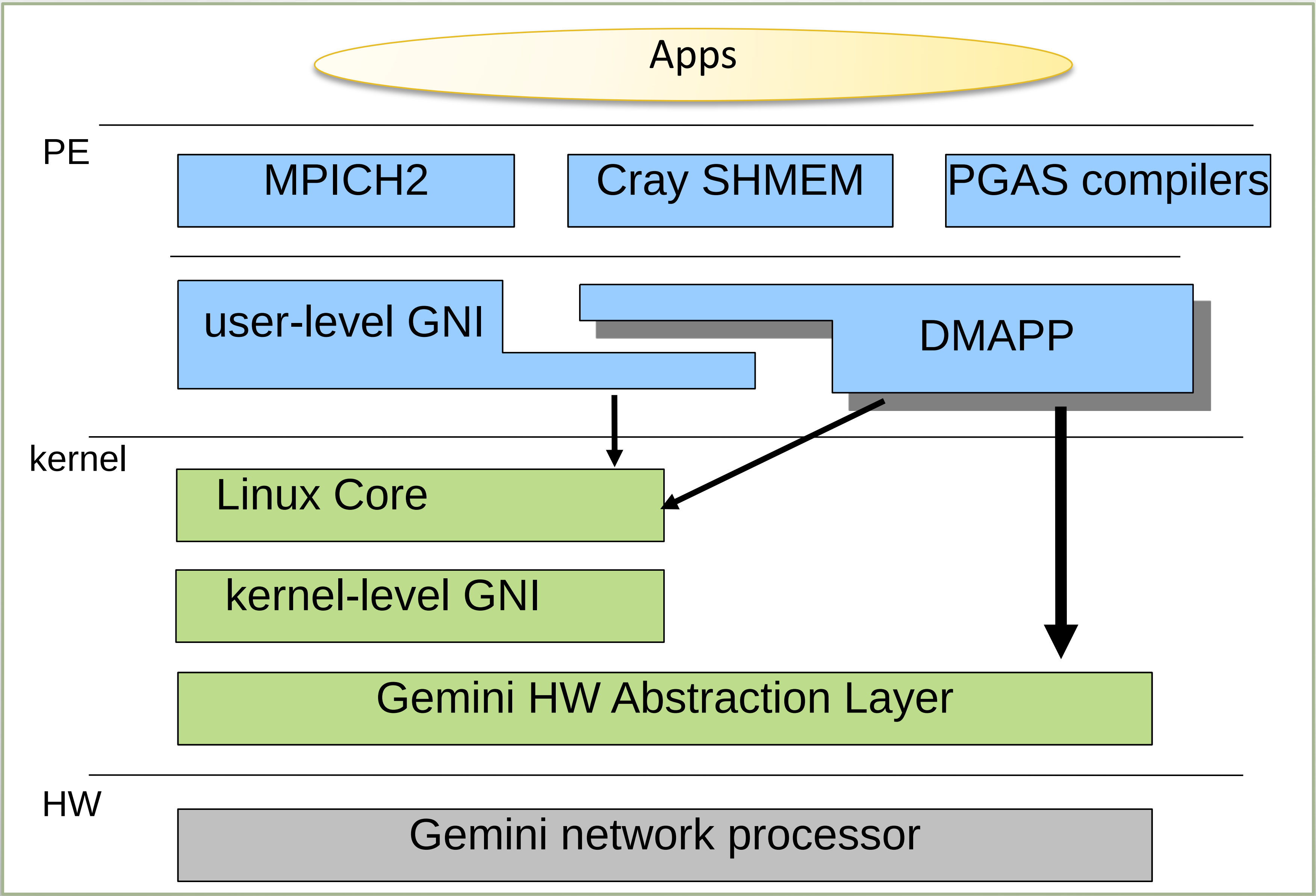
- DMAPP in context
- Basic features of the API
- Memory allocation and sample API calls
- Preliminary Gemini performance measurements

What is DMAPP?

The Distributed Memory Application (DMAPP) API

- Supports features of the Gemini Network Interface
- Used by higher levels of the software stack:
  - PGAS compiler runtime
  - SHMEM library
- Balance between portability and hardware intimacy
- Intended to be used by system software developers
  - Application developers should use SHMEM

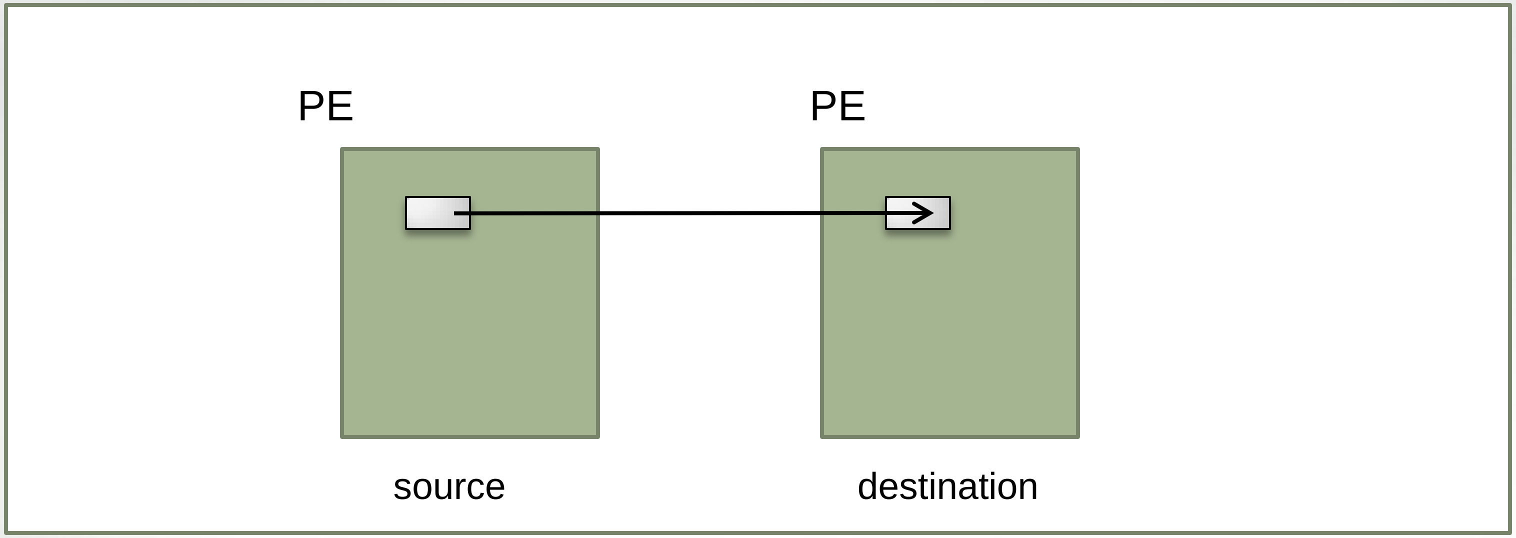
DMAPP in context



DMAPP Programming Model

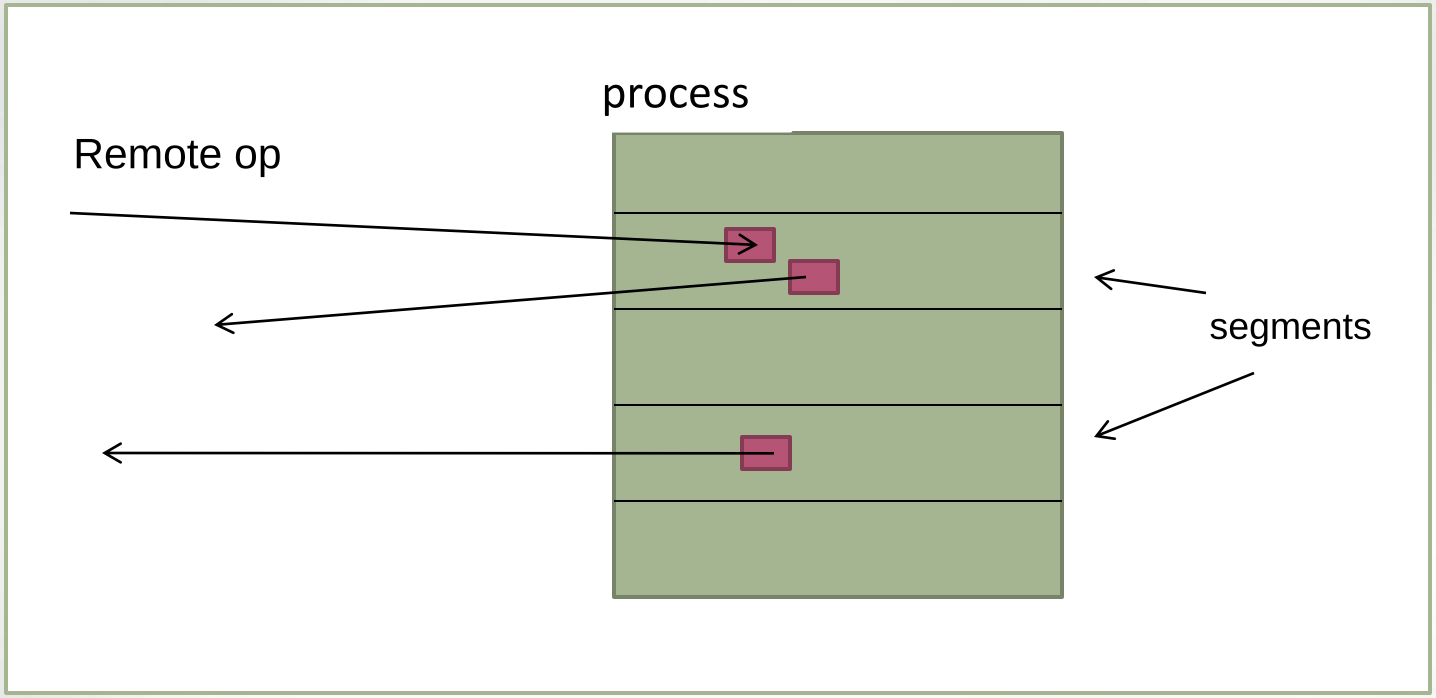
- Distributed memory model
- One-sided model for participating (SPMD) processes launched by Alps aprun command
- Each PE has local memory but has one-sided access (PUT/GET) to remote memory
- Remote memory has to be in an accessible memory segment

Hardware Operations



- Network supports direct remote get/put from user process to user process.
- Mechanisms:
  - Block Transfer (BTE)
  - Fast Memory Access (FMA) including Atomic Memory Operations (AMOs)

Hardware Operations



- Remote source or destination in either data or symmetric-heap segments
- Symmetry means we can use local address information in remote context

DMAPP Initialization and setup

- **`dmapp_init`**
  - Sets up access to data and symmetric heap (exports memory)
  - barrier
  - you can set or read available resource limits
- **`dmapp_get_jobinfo`**
  - Returns a structure with useful information:
    - Number of PEs
    - Index of this PE
    - Pointers to data and symmetric heap segments required in other calls

API Example: dmapp_put()

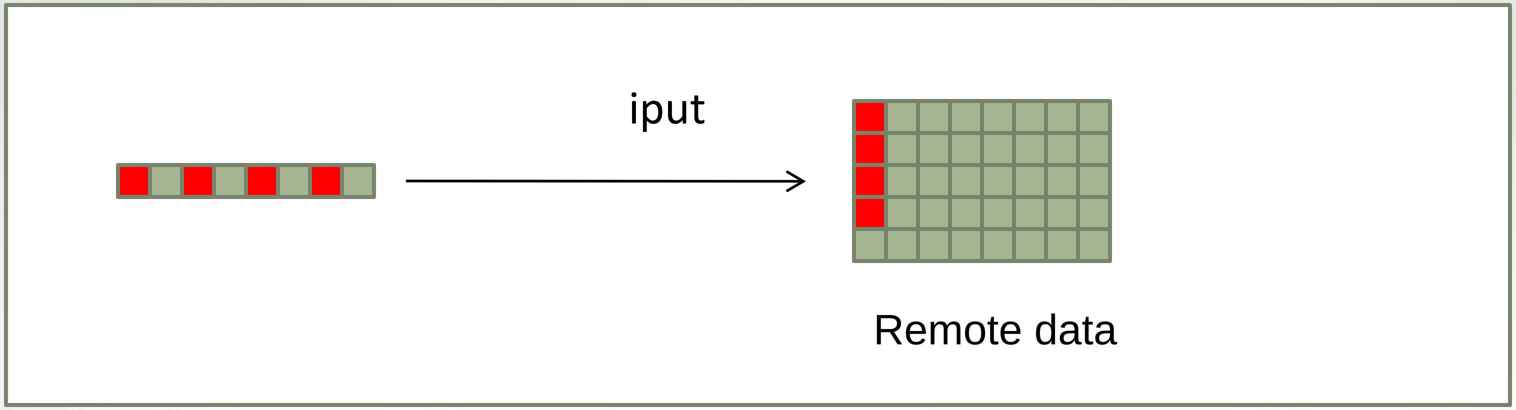
```
dmapp_put(*target_addr, *target_seg, target_pe,
                source_addr, nelems, type)
```

- Remote locations defined by:
  address, segment, pe
- This is a blocking operation
- `type` can be DMAPP_{BYTE,DW,QW,DQW) for 1, 4, 8 and 16 bytes.
- Analogous **get** call

Variants of DMAPP one-sided RMA calls

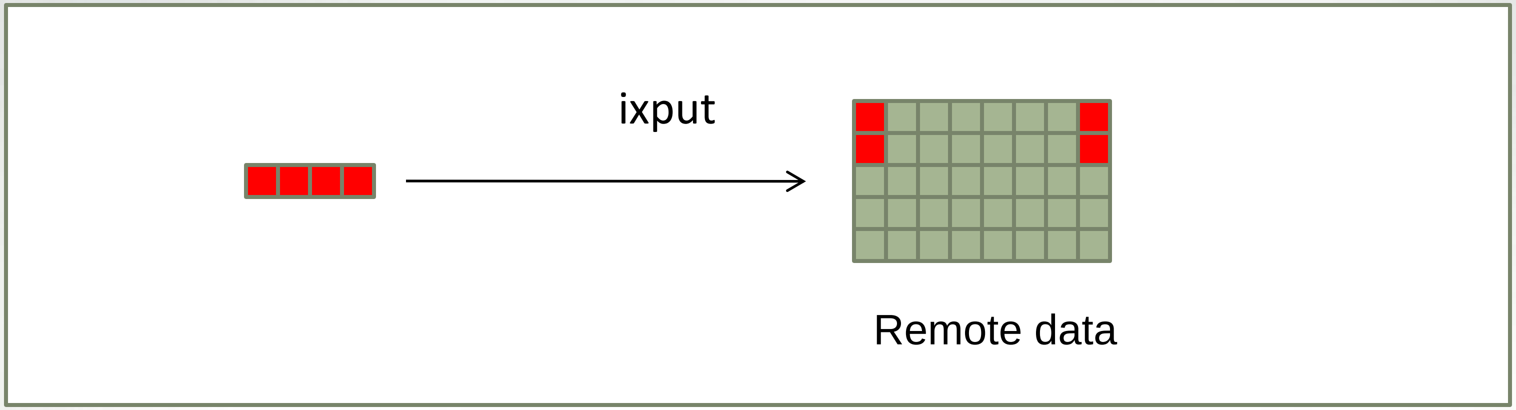
- Blocking (no suffix)
  `dmapp_put`, **`dmapp_get`**
- Non-blocking *explicit* (_nb suffix)
  `dmapp_put_nb(…, syncid)`
- Non-blocking *implicit* (_nbi suffix)
  - No handle to test for completion
- Synchronization (memory completion/visibility)
  - Can wait on specific **`syncid`**
  - Can wait for all implicit operations to complete

Variants of DMAPP one-sided RMA calls…



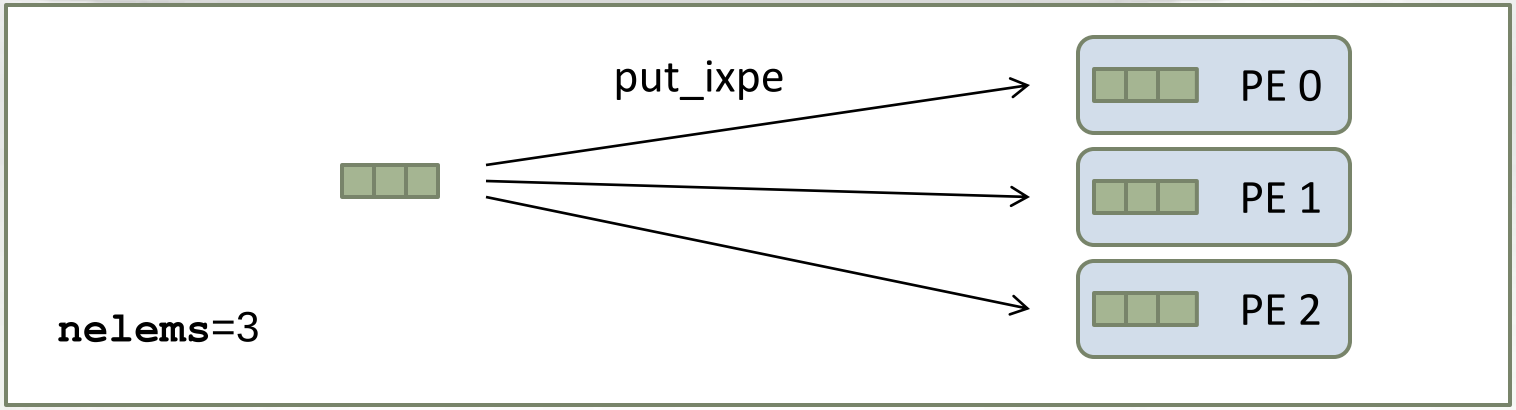
- Strided calls
  `dmapp_iput`…, **`dmapp_iget`**…
- Additional arguments define source and destination stride in terms of elements

Variants of DMAPP one-sided RMA calls…



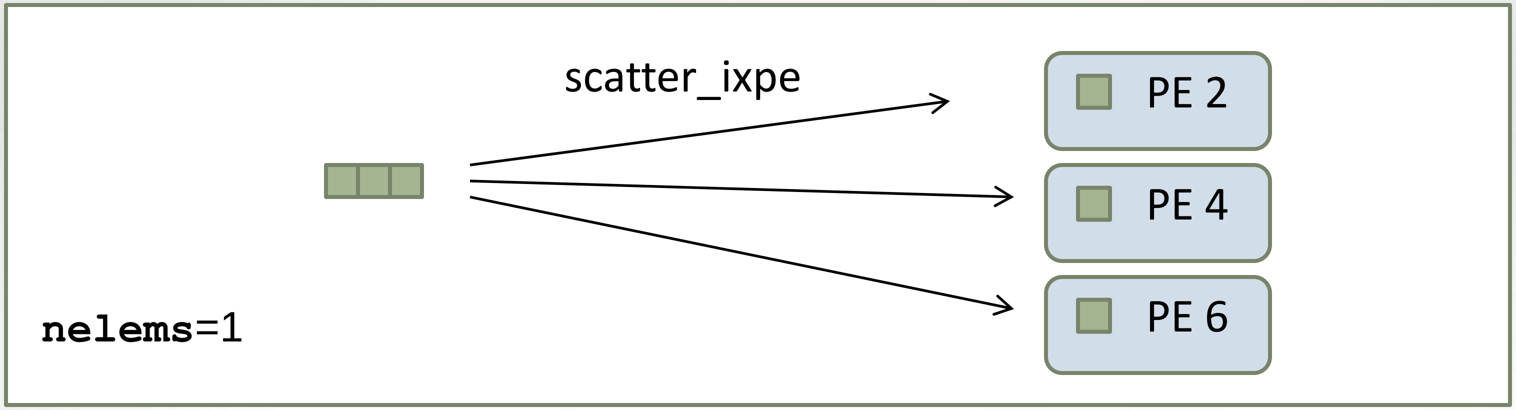
- Scatter/Gather calls
  `dmapp_ixput`…, **`dmapp_ixget`**…
- Local data is contiguous
- Remote data is distributed as defined by an array of offsets

Variants of DMAPP one-sided RMA calls…



- Put with indexed PE-stride calls **`dmapp_put_ixpe`**…, **`dmapp_get_ixpe`**…
- Local data is contiguous
- Remote data is distributed (as defined by an array of PE-offsets) to the same address on each PE
- Use for small amounts of data
- *These are not collective operations*

Variants of DMAPP one-sided RMA calls…



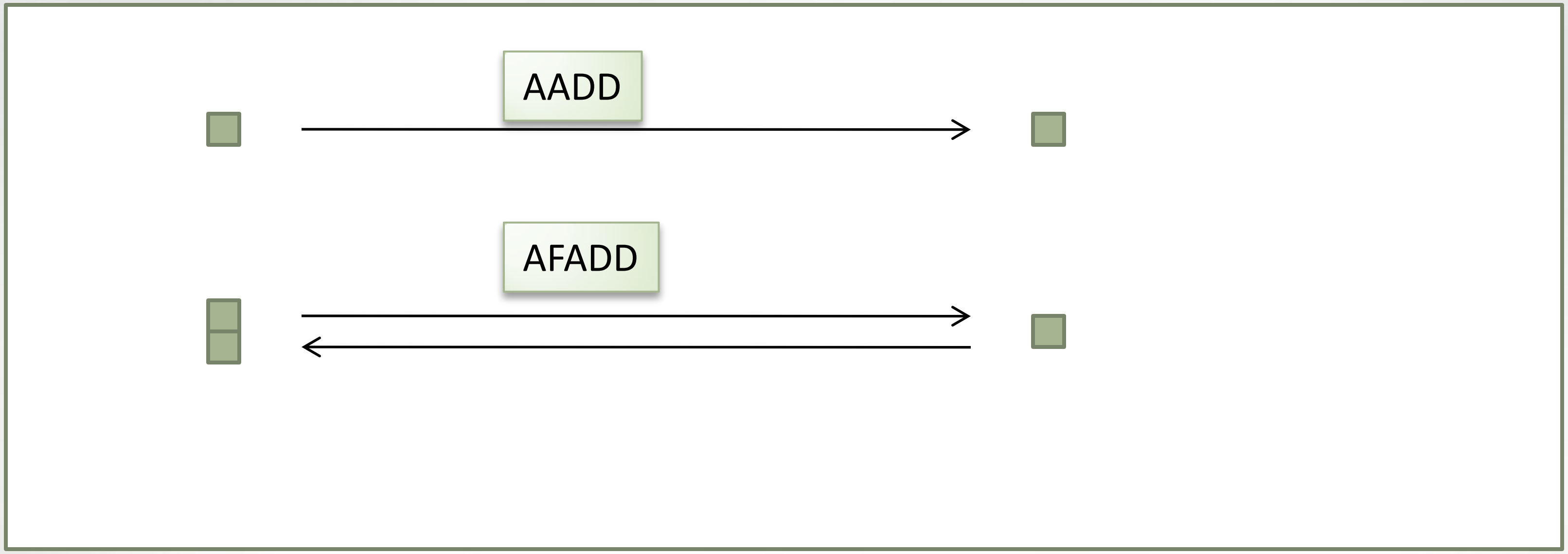
- Scatter/Gather with indexed PE-stride calls **`dmapp_scatter_ixpe`**, **`dmapp_gather_ixpe`**
- Local data is contiguous
- Source is scattered to (or gathered from) PEs nelems elements at a time.

Atomic Memory Operations

Atomic operations to 8-byte (QW) remote data

Command	Operation
AADD	Atomic ADD
AAND	Atomic AND
AOR	Atomic OR
AXOR	Atomic EXCLUSIVE OR
AFADD	Atomic fetch and ADD
AFAND	Atomic fetch and AND
AFOR	Atomic fetch and OR
AFXOR	Atomic fetch and XOR
AFAX	Atomic fetch AND-EXCLUSIVE OR
ACSWAP	Compare and SWAP

Atomic Memory Operations



- Direct support in NIC
- Be careful to only read values via DMAPP API

Synchronization

- Some calls return syncid (_nb)
- Can test or wait on completion
  - **`dmapp_syncid_wait(*syncid)`**
  - **`dmapp_syncid_test(*syncid,*flag)`**
- For implicit non-blocking (_nbi)
  - **`dmapp_gsync_wait()`**
  - **`Dmapp_gsync_test(*flag)`**
  - Use for many small messages

Symmetric Heap

- DMAPP applications can allocate memory in symmetric heap

```
double *a;
 a=(double*)
   dmapp_sheap_malloc(N*sizeof(double));
```

- Associated **`realloc`** and **`free`** calls.
- Application is responsible for maintaining symmetry of allocations

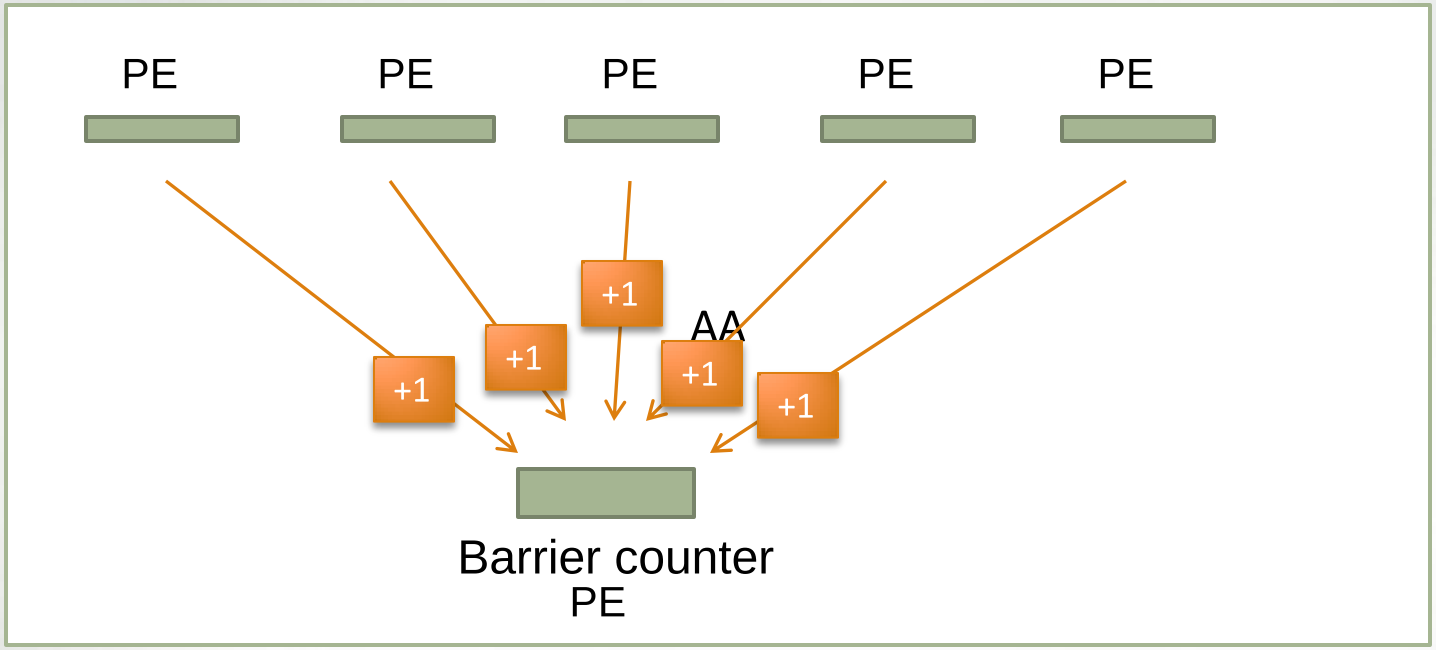
Memory useable for remote operations

DMAPP exports data and symmetric heap for you

This means:

- For C
  - File scope and static inside function
  - Allocated in symmetric Heap
- For Fortran (no API but if there was)
  - SAVEd data
  - Data in COMMON

Example: Barrier check-in



- Atomic add for master counter (FADD for testing)
- Master compares (with n-1) and swaps with 0
- … master releases other PEs

Barrier check-in code

THE SUPERCOMPUTER COMPANY

```
static uint64_t barrier_counter, bc;

  if (mype==master){
   do{
    // wait until counter is npes-1, swap with 0
    dmapp_acswap_qw(&bc,(void *)&barrier_counter,
                    seg_data,mype,npes-1,0);
    } while ( bc!=(npes-1));

   } else {

    dmapp_aadd_qw((void*)&barrier_counter,seg_data,
                  master,1);
   }
// now release barrier…
```

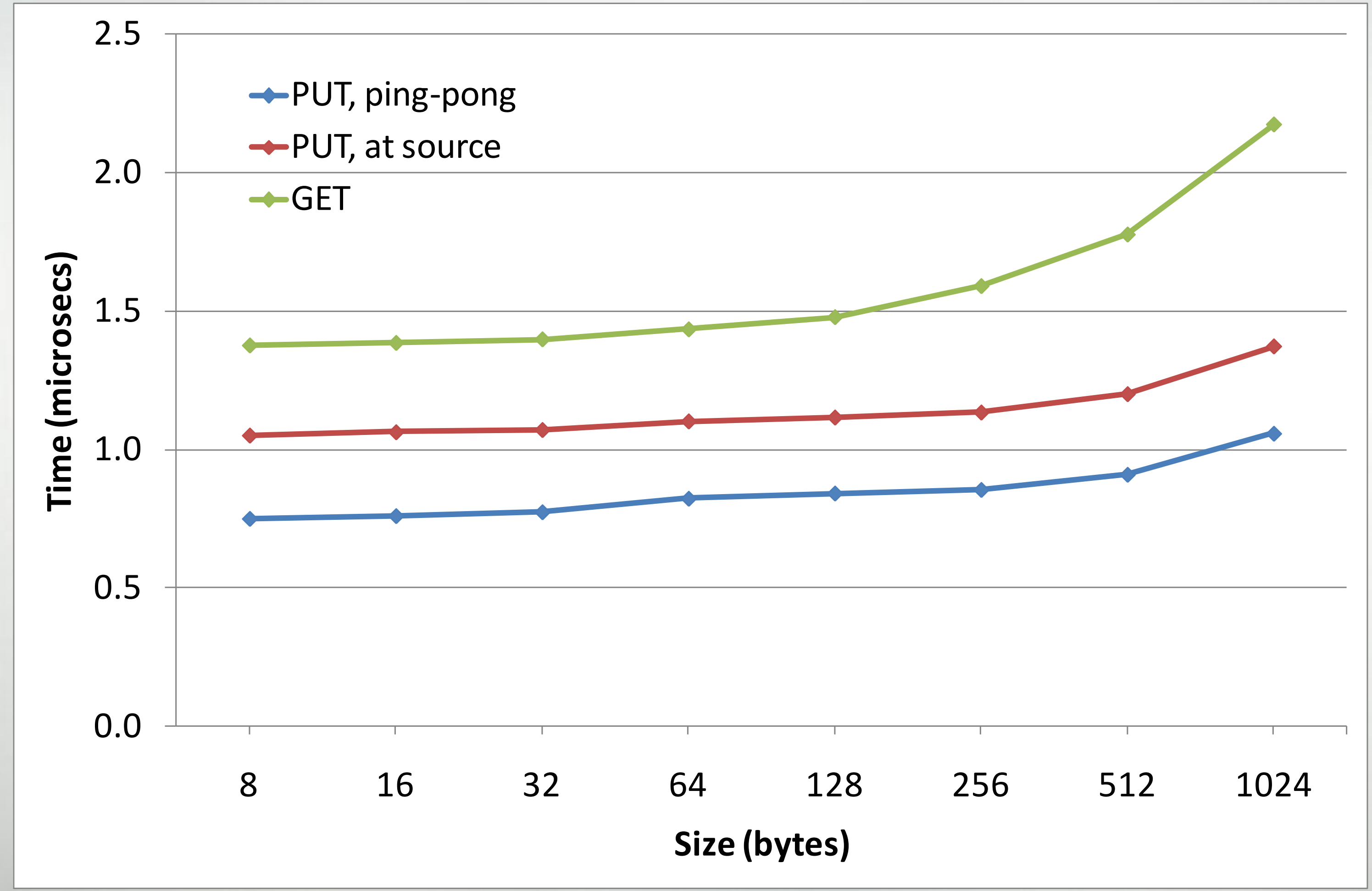
Comparison with SHMEM

- SHMEM
  - Has same SPMD model
  - Requires use of symmetric memory
  - Original interface is blocking
  - Non-standard extensions for non-blocking put/get
  - Varying-sized data items with typed API
  - Get/put with strided and gather/scatter variants
  - Barrier and collective operations on sets of PEs
  - Has the same atomic memory operations
- SHMEM is implemented using DMAPP for Gemini systems

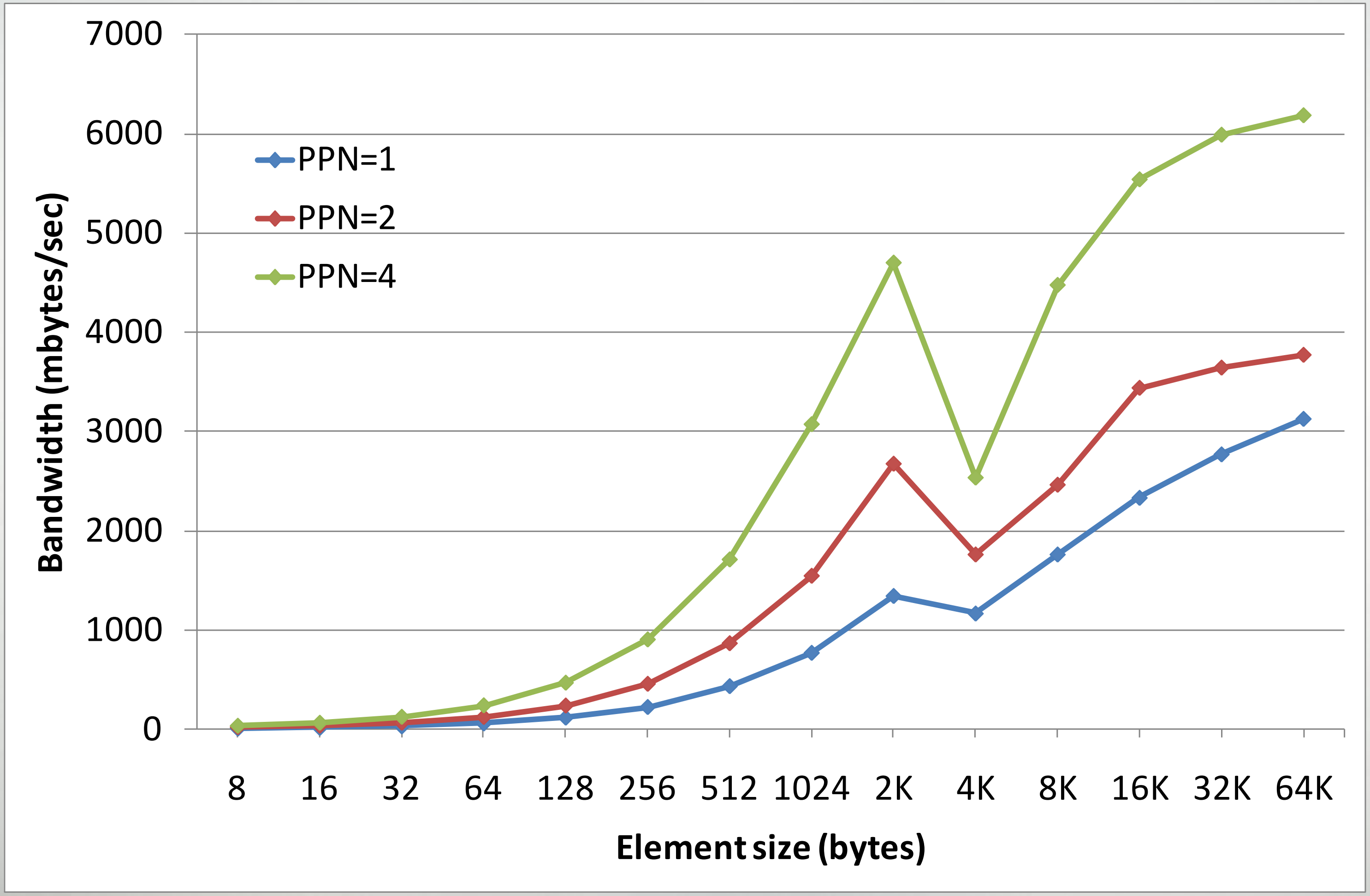
Preliminary Gemini Performance Data

- Data measured on prototype system during Q1 2010
- 2100MHz Opteron processors
- 2400MHz HyperTransport interface
- Dual node tests run between PEs on neighbouring Gemini routers

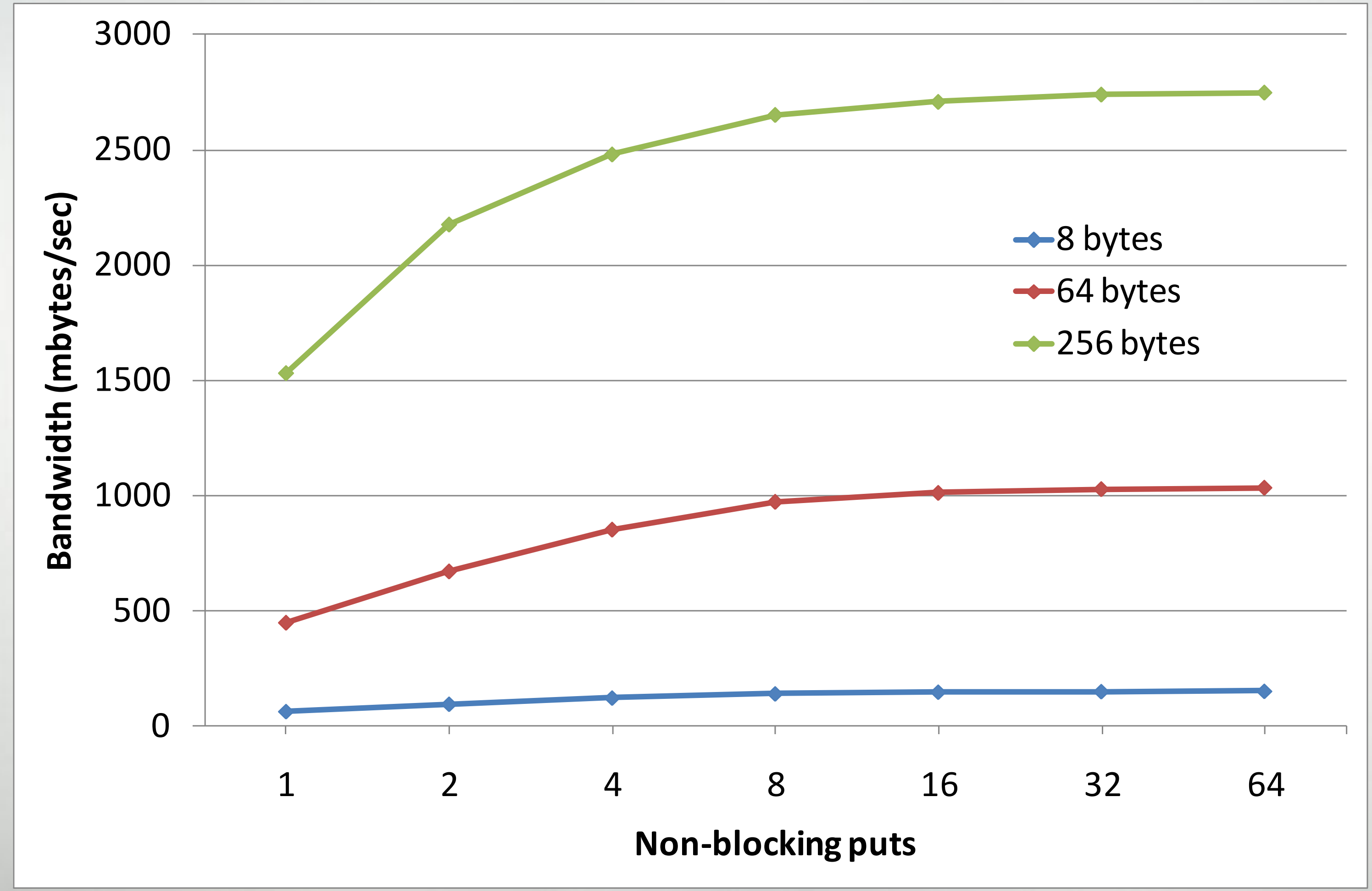
Gemini DMAPP put and get latencies



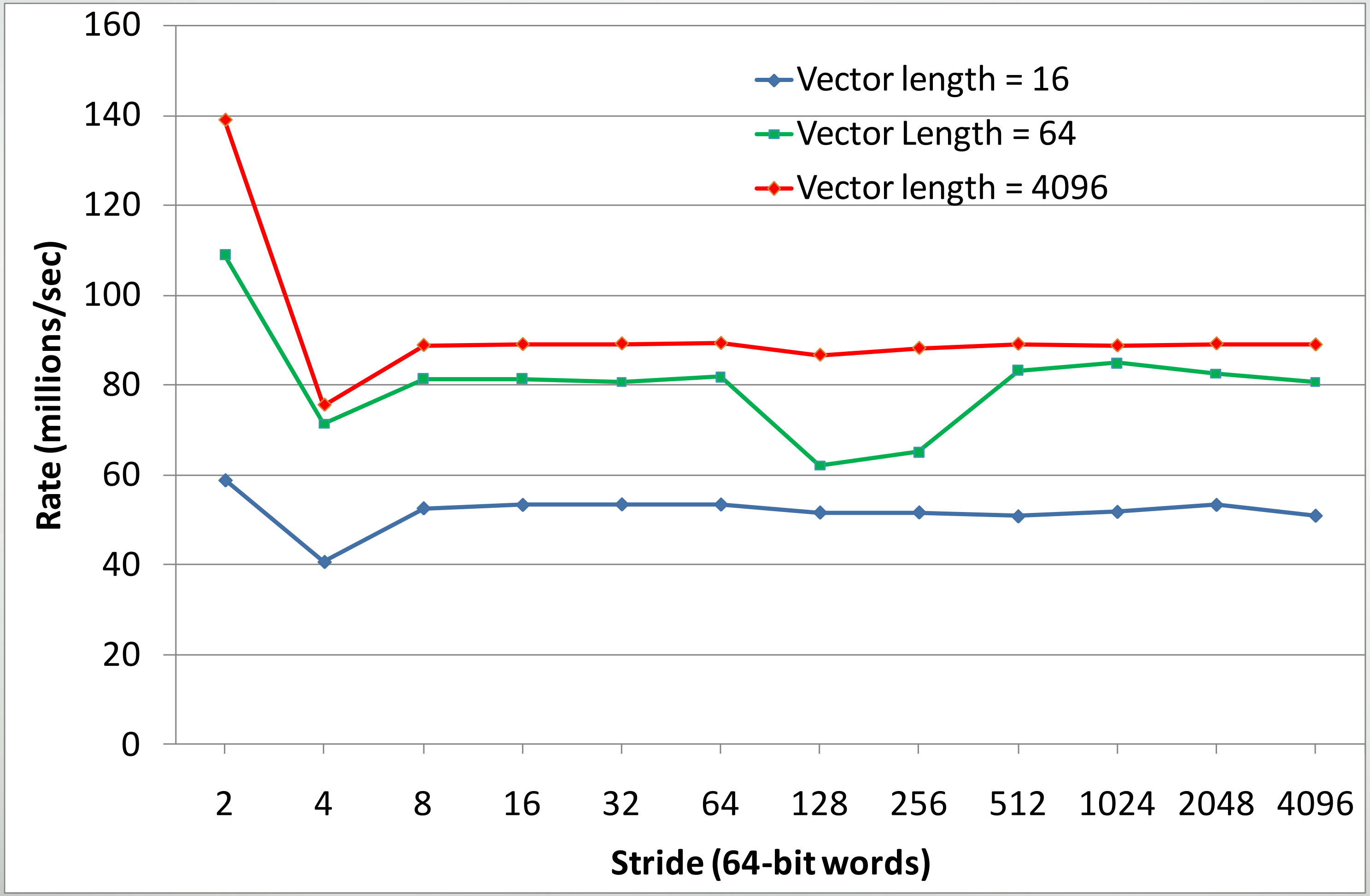
Put bandwidth as a function of transfer size



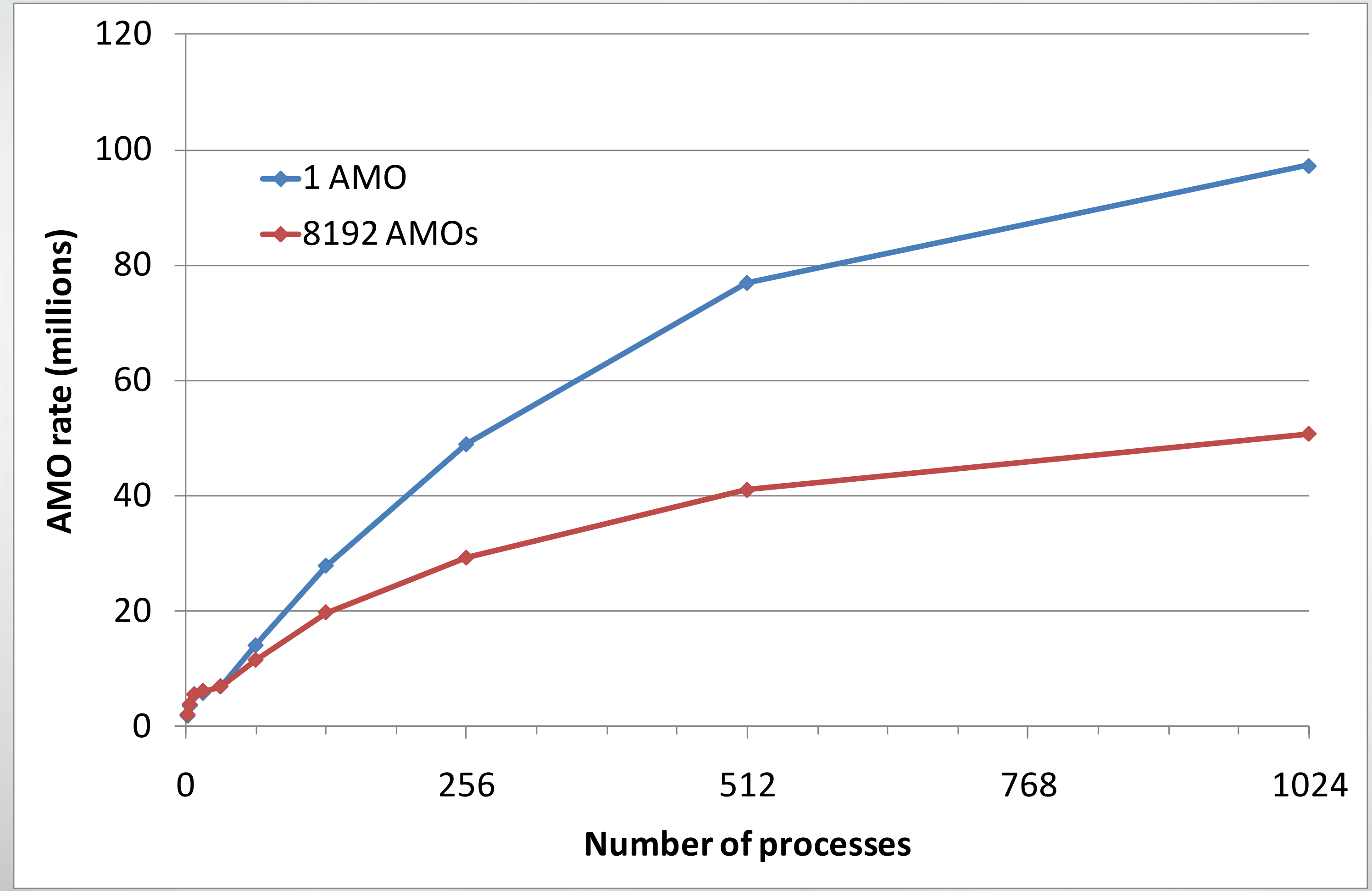
Bandwidth on small non-blocking puts



8-byte strided put rate



Many-to-one AMO rate



Observations

- Latency (~1μs) far better than SeaStar
- Good aggregate bandwidths on small transfers
- High AMO rates, especially when multiple processes target the same variables
- Strided puts are an important case for CAF

- Ongoing optimization effort (for example reduce number of FMA descriptor updates)

Recap

- What is DMAPP and where does it fit?
- Basic features of the API
- Memory allocation and sample API calls
- Preliminary Gemini performance data

CRAY

THE SUPERCOMPUTER COMPANY