



---

# UPC Library Extensions for Explicitly Non-blocking and Non-contiguous Memcpy

**Dan Bonachea**

**Christian Bell, Wei Chen , Jason Duell  
Paul Hargrove, Parry Husbands , Costin Iancu  
Mike Welcome, Katherine Yelick**

<http://upc.lbl.gov>



# General Motivation



- Reminder of current `upc_memcpy` library functions:

```
void upc_memcpy(shared void *dst, shared const void *src, size_t nbytes);
```

```
void upc_memget(void *dst, shared const void *src, size_t nbytes);
```

```
void upc_mempu(shared void *dst, const void *src, size_t nbytes);
```

- Very semantically limited: these functions move a single, contiguous chunk of data, possibly across a network, as a fully blocking operation
  - In practice, implementations cannot do much better than naïve translation without expensive runtime checking and/or hefty and conservative compiler analyses
- Four orthogonal proposed extensions to the `upc_memcpy` bulk data movement library (sent to UPC community list on Feb 10)
    - Explicitly non-blocking `upc_memcpy`
    - Non-contiguous (Vector, Indexed and Strided) `upc_memcpy`
  - All are now implemented as prototypes in Berkeley UPC 2.0



# Explicitly Non-blocking Memcpy Motivation



- Distributed-memory NIC hardware is naturally non-blocking
  - can provide significant speedups by overlapping communication with computation or other communication
  - Want to exploit these capabilities
- Allow programmer to directly express the lack of data dependencies using explicitly non-blocking bulk transfer operations
  - The programmer often knows that given bulk data movements are completely independent, but it's often very difficult for a compiler to infer this info
    - due to buffer aliasing, data dependent operations, unexpressed restrictions on the set of legal inputs, etc
  - Proposed interface is easy to understand and use
    - Also trivial to implement on most network hardware



# Explicitly Non-blocking Memcpy Interface



- Very simple extension to existing library:
  - New "flavors" of `upc_mem{put,get,cpy}` with "`_async`" suffix:

```
upc_handle_t bupc_memcpy_async(shared void *dst, shared const void *src,
                               size_t nbytes);
```

```
upc_handle_t bupc_memget_async(void *dst, shared const void *src,
                               size_t nbytes);
```

```
upc_handle_t bupc_mempu_async(shared void *dst, const void *src,
                              size_t nbytes);
```
  - Same args and semantics as blocking variants, returns a `upc_handle_t`
    - an opaque handle representing the initiated asynchronous operation
    - analogous to `MPI_Request` object for `MPI_Isend/MPI_Irecv`
  - Synchronized using one of two new functions:
    - Block for completion:

```
void bupc_waitsync(upc_handle_t handle);
```
    - Non-blocking test for completion:

```
int bupc_trysync(upc_handle_t handle);
```



## Explicitly Non-blocking Memcpy Example



Example of a nearest-neighbor data fetch on a regular 1-D blocked decomposition using `upc_memget_async`:

```
#define BLKSZ 100
shared [BLKSZ] double A[BLKSZ*THREADS];
double leftdata[BLKSZ], rightdata[BLKSZ]; /* local temporary buffers */
upc_handle_t leftfetch_handle = UPC_COMPLETE_HANDLE; /* handles */
upc_handle_t rightfetch_handle = UPC_COMPLETE_HANDLE;

if (MYTHREAD > 0) /* initiate fetch of data from left neighbor */
    leftfetch_handle = bupc_memget_async(leftdata, &(A[BLKSZ*(MYTHREAD-1)]),
                                         BLKSZ*sizeof(double));
if (MYTHREAD < THREADS-1) /* initiate fetch of data from right neighbor */
    rightfetch_handle = bupc_memget_async(rightdata, &(A[BLKSZ*(MYTHREAD+1)]),
                                           BLKSZ*sizeof(double));

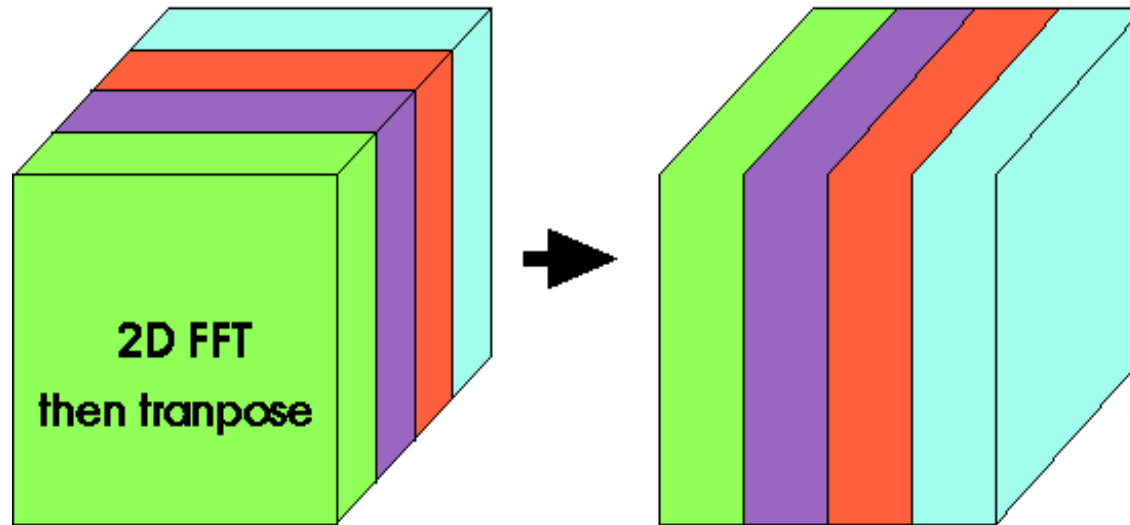
/* perform some independent overlapped computations here */

bupc_waitsync(leftfetch_handle); /* block for completion, if necessary */
bupc_waitsync(rightfetch_handle);

/* now safe to operate on leftdata and rightdata */
```



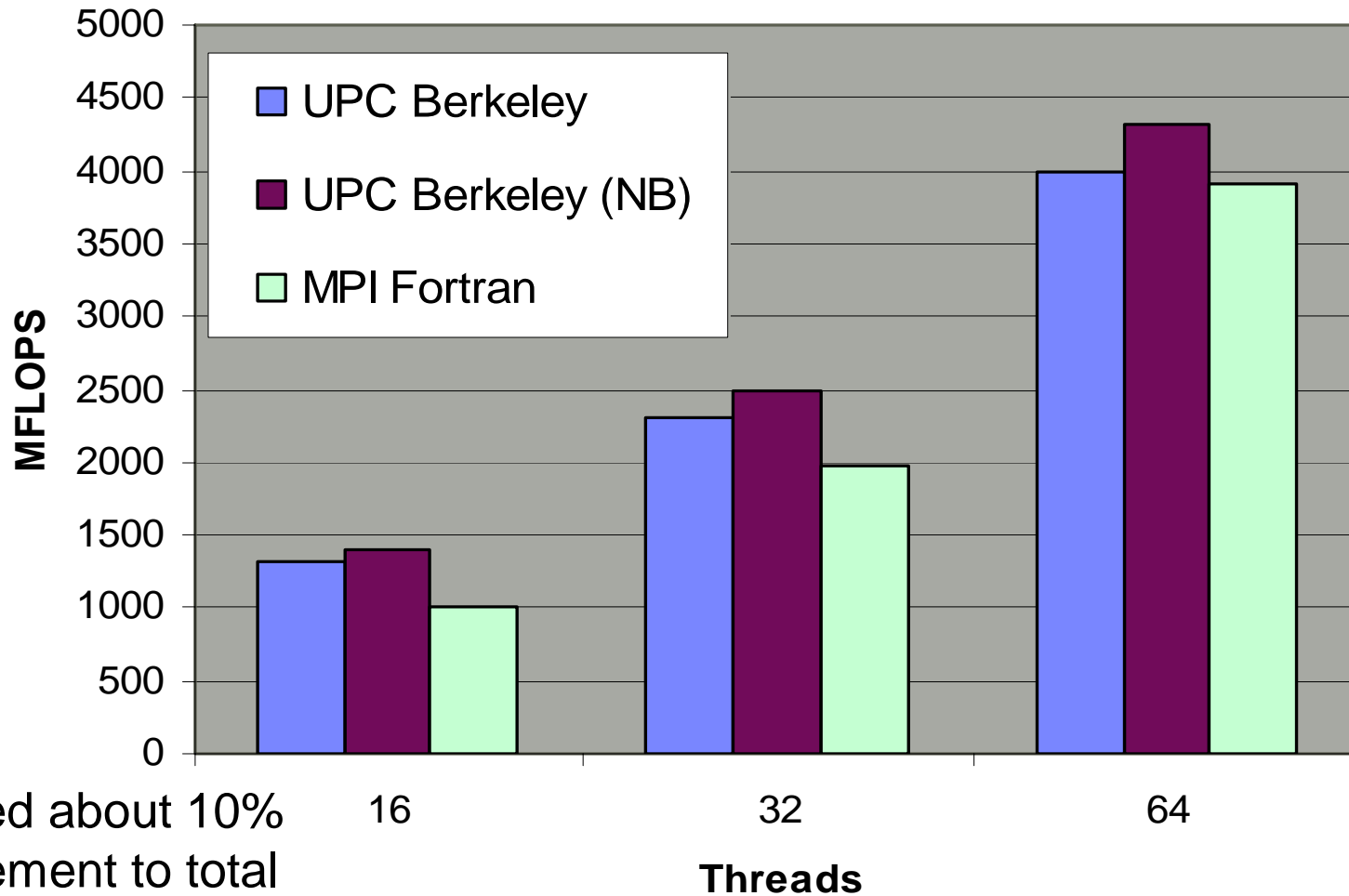
# NAS FT Benchmark - 3D FFT



- Work by Christian Bell & Wei Chen at UC Berkeley, using Berkeley UPC
- 2D FFT Computations partitioned across each thread
- The transpose of 2D planes can interleave communication of plane  $i$  and computation of plane  $i+1$
- Non-blocking extensions allow this interleaving to occur concurrently, which effectively breaks up and hides some of the all-to-all transpose cost



## NAS FT 2.3 Class B - NERSC Alvarez Cluster



Achieved about 10% improvement to total application runtime by using non-blocking memcpy to overlap communication

Machine: NERSC Alvarez cluster  
80 Dual PIII-866MHz 1GB Nodes running Berkeley UPC  
(gm-conduit /Myrinet 2K, 33Mhz-64Bit bus)

Berkeley UPC at LBNL/UCB



# Other motivations for non-blocking upc\_memcpy

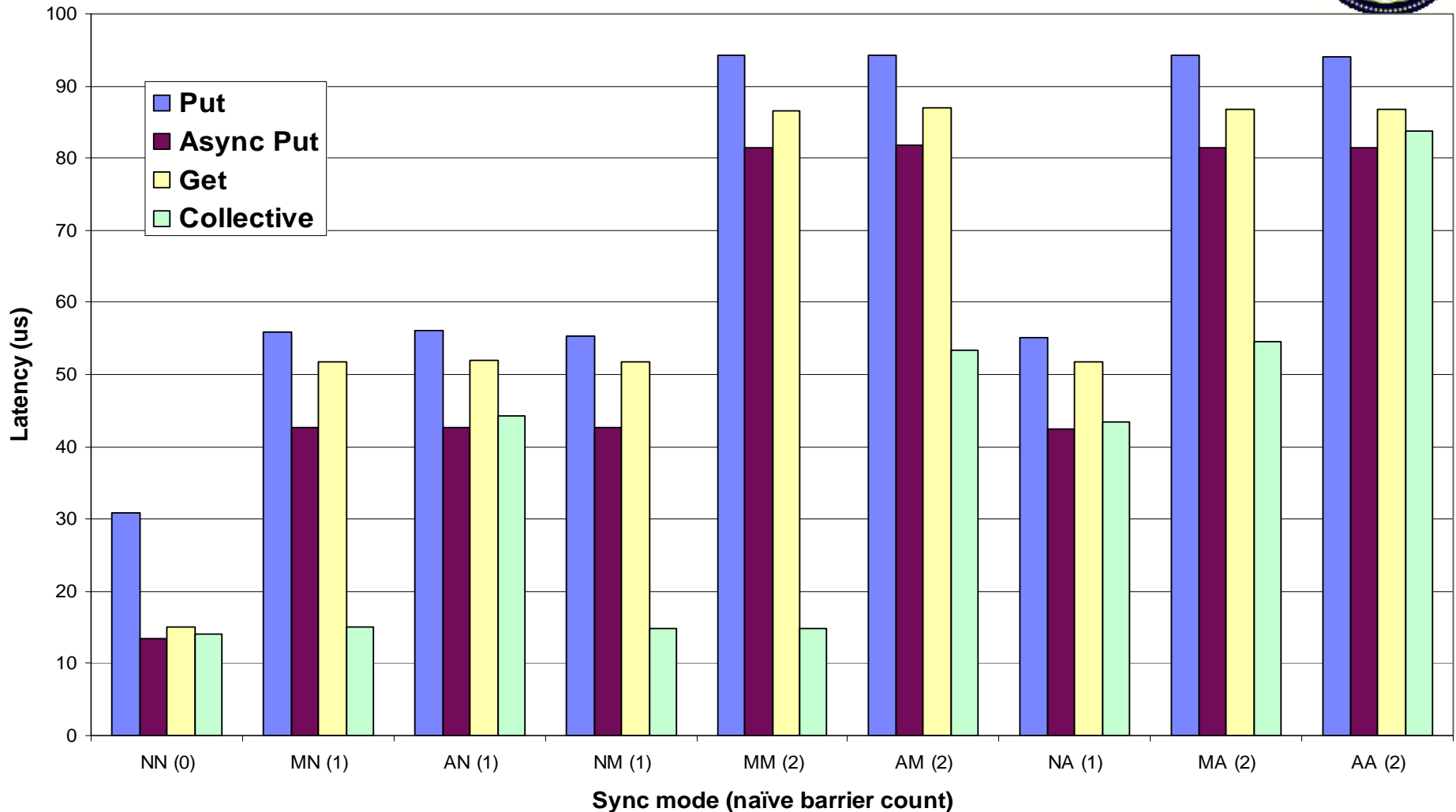
---



- Has already proven itself useful to UPC library writers
  - UPC *Reference* Collectives implementation
    - written by Steve Seidel from MTU
    - push/pull based collectives written purely in UPC
    - modified by us to use the non-blocking memcpy extensions
    - can often trivially replace blocking upc\_memput/get ops with non-blocking bupc\_mem{put,get}\_async ops
    - Note: Berkeley UPC 2.0 ships a different collective library, based on GASNet collectives utilizing finer-grained synch.
  - Prototype UPC-IO library
    - bupc\_mem{put,get}\_async useful for gathering contiguous shared data to/from IO nodes
    - Non-contiguous memcpy extensions also useful for gathering distributed shared arrays to/from IO nodes



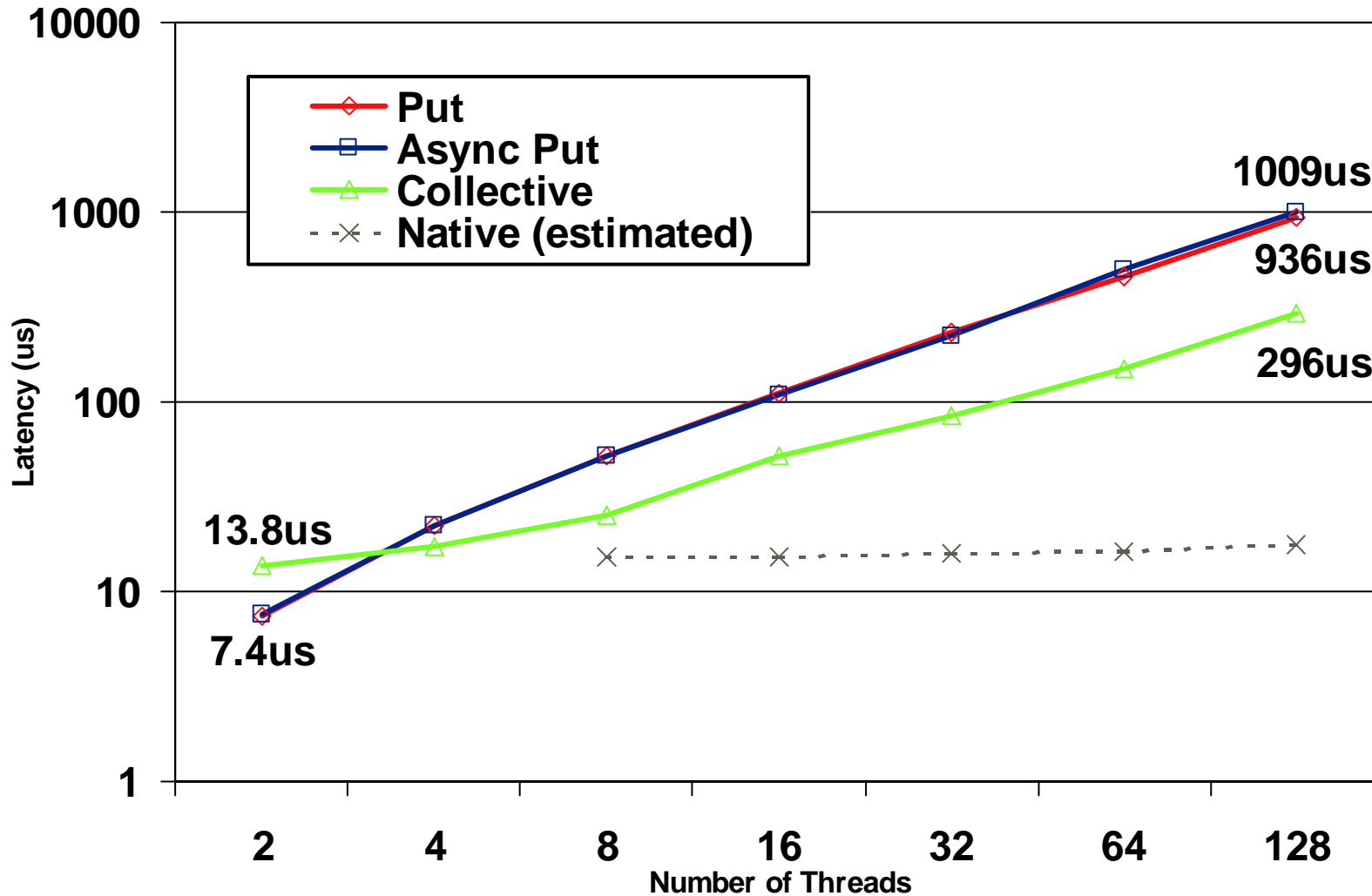
# Performance of 8-byte upc\_all\_broadcast



4-nodes of P4 Xeon Mellanox Infiniband & all the latest drivers  
Berkeley UPC 2.0 w/ vapi-conduit



# Performance of 8-byte upc\_all\_broadcast (SYNC=NO/NO)



Quadrics elan3, Tru64 AlphaServer (lemieux, PSC)



---

# UPC Library Extensions for Non-Contiguous Memcpy (preliminary prototype)

Vector (variable-length scatter/gather),  
Indexed (fixed-length scatter/gather) &  
**Strided** (regular non-contiguous)  
a.k.a. "VIS"



# Application Motivation for VIS



- Many applications have non-collective, non-contiguous (ie sparse) remote access patterns
  - irregular cases: SPMV, distributed graph data structures
  - regular cases: remote sub-array access (ghost value exchange)
- Most natural way to write these algorithms leads to a fine-grained comm.
  - naïve translation to individual remote accesses performs poorly on modern networks
- Want communication aggregation optimizations
  - Save by aggregating small messages into larger ones (ie pack/unpack), possibly with help from hardware
  - Allow sophisticated users to directly express aggregate non-contiguous communication
  - Provide compilation target so optimizer can express automated aggregation (ie message coalescing)



# UPC Support for Non-Contiguous Remote Access



- Propose extending `upc_mem{put,get,cpy}` library with non-contiguous variants
  - All proposed functions include both blocking and non-blocking variants
  - See full proposal on the Berkeley UPC publications page
- Vector
  - `src` and `dst` are each a list of variable-sized contiguous regions
- Indexed
  - `src` and `dst` are each a list of fixed sized contiguous regions
- Strided
  - `src/dst` are each a set of regularly sized and spaced regions
  - sufficient for expressing arbitrary rectangular sections over dense N-d arrays (for any dimension N)



# Network Hardware Support for VIS



Hardware support for non-contiguous RMA varies widely:

Hardware	Vector	Indexed	Strided	RDMA
Cray X-1		YES		N/A
Quadrics Elan		YES		YES
Infiniband	local-side	local-side		YES
Myrinet (MX)	YES			YES
IBM LAPI	YES	YES	YES	NO

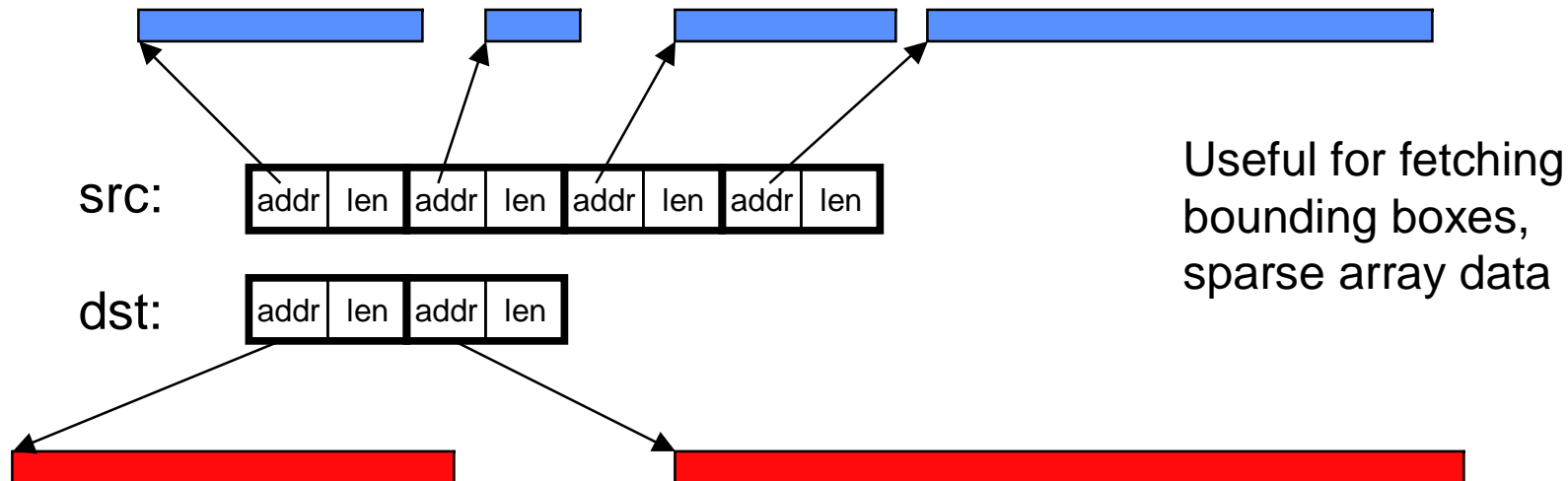
want the library/compiler VIS features to exploit the hardware support where available, without rewriting the compiler for each platform



# Proposed UPC Interfaces for Non-Contiguous Remote Access



- **Vector** - src/dst are list of variable length contiguous regions:



- Source/Destination region counts and sizes may differ
  - only total data sz must match
- Most general and flexible option - least hardware support
- Blocking and non-blocking explicit handle variants



# UPC Vector Interface



```
typedef struct {  
    void *addr;  
    size_t len;  
} bupc_pmemvec_t;
```

```
typedef struct {  
    shared void *addr; ← treated as a (shared [] char *) - ie. no wrapping  
    size_t len;  
} bupc_smemvec_t;
```

```
void bupc_memcpy_list (size_t dstcount, bupc_smemvec_t const dstlist[],  
                       size_t srccount, bupc_smemvec_t const srclist[]);
```

```
void bupc_memput_list (size_t dstcount, bupc_smemvec_t const dstlist[],  
                       size_t srccount, bupc_pmemvec_t const srclist[]);
```

```
void bupc_memget_list (size_t dstcount, bupc_pmemvec_t const dstlist[],  
                       size_t srccount, bupc_smemvec_t const srclist[]);
```

*(analogous async variants not shown)*



# UPC Vector Example



```
#define BLKSZ 100
shared [BLKSZ] double A[BLKSZ*THREADS]; /* assume THREADS >= 3 */
bupc_smemvec_t srclist[] = {
    { &(A[14]), sizeof(double) }, /* element 14 (from thread 0) */
    { &(A[20]), sizeof(double) }, /* element 20 (from thread 0) */
    { &(A[100]), 50*sizeof(double) }, /* elements 100..149 (from thread 1) */
    { &(A[2*BLKSZ]), BLKSZ*sizeof(double) } /* entire block (from thread 2) */
};
double mybuf[52+BLKSZ];
bupc_pmemvec_t dstlist[] = { { mybuf, sizeof(mybuf) } };

bupc_memget_list ( 1, dstlist, 4, srclist );

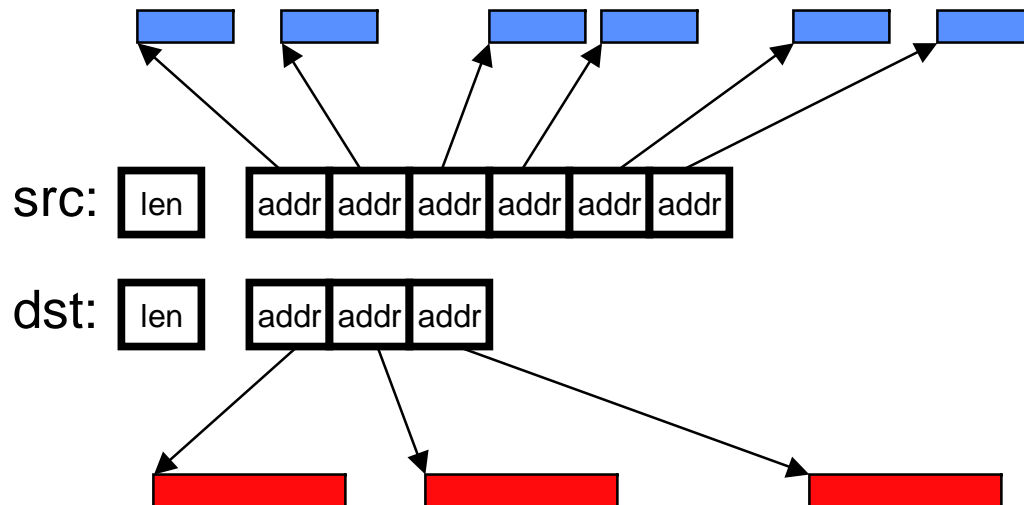
/* compute on contents of mybuf */
```



# Proposed UPC Interfaces for Non-Contiguous Remote Access



- **Indexed** - list of fixed-length contiguous regions:



Useful for fetching irregular set of array elements, inspector/executor

- More restrictive than vector interface
  - Less metadata due to fixed size
  - Closer to most available network hardware support
- Also have non-blocking explicit handle variants



# UPC Indexed Interface

---



void **bupc\_memcpy\_list** (size\_t dstcount, shared void \* const dstlist[], size\_t dstlen,  
size\_t srccount, shared const void \* const srclist[], size\_t srclen);

void **bupc\_memput\_list** (size\_t dstcount, shared void \* const dstlist[], size\_t dstlen,  
size\_t srccount, const void \* const srclist[], size\_t srclen);

void **bupc\_memget\_list** (size\_t dstcount, void \* const dstlist[], size\_t dstlen,  
size\_t srccount, shared const void \* const srclist[], size\_t srclen);

*(analogous async variants not shown)*



# UPC Indexed Example



```
void bupc_memget_list (size_t dstcount, void * const dstlist[], size_t dstlen,  
    size_t srccount, shared const void * const srclist[], size_t srclen);
```

```
#define BLKSZ 100
```

```
shared [BLKSZ] double A[BLKSZ*THREADS]; /* assume THREADS >= 2 */
```

```
shared void * srclist[] = {
```

```
    &(A[14]), &(A[15]), &(A[16]), /* element 14..16 (from thread 0) */
```

```
    &(A[100]), &(A[110]) /* element 100 and 110 (from thread 1) */
```

```
};
```

```
double mybuf[5];
```

```
void * dstlist[] = { &mybuf };
```

```
bupc_memget_list (1, dstlist, 5*sizeof(double), 5, srclist, sizeof(double));
```

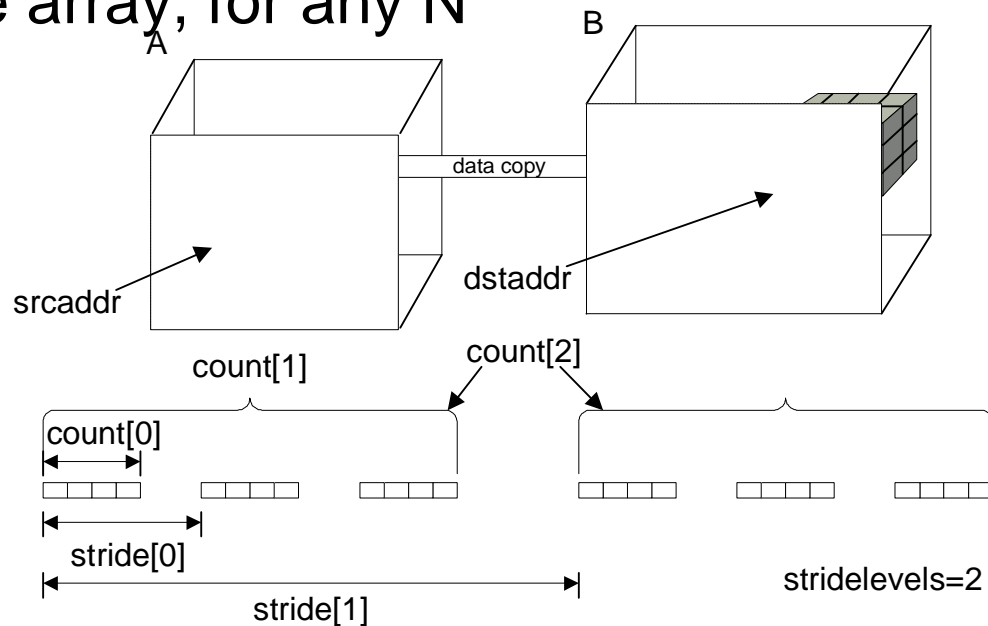
```
/* compute on contents of mybuf */
```



# Proposed UPC Interfaces for Non-Contiguous Remote Access



- **Strided:** regularly spaced/sized accesses
- src/dst specify an arbitrary rectangular section on an N-d dense array, for any N



Useful for fetching any non-contiguous section of a dense array

- Most restrictive access pattern - least metadata
  - metadata size is linear in dimensionality (N)
- Also have non-blocking explicit handle variants



# UPC Strided Interface



void **bupc\_memcpy\_strided** (shared void \*dstaddr, const size\_t dststrides[],  
shared const void \*srcaddr, const size\_t srcstrides[],  
const size\_t count[], size\_t stridelevels);

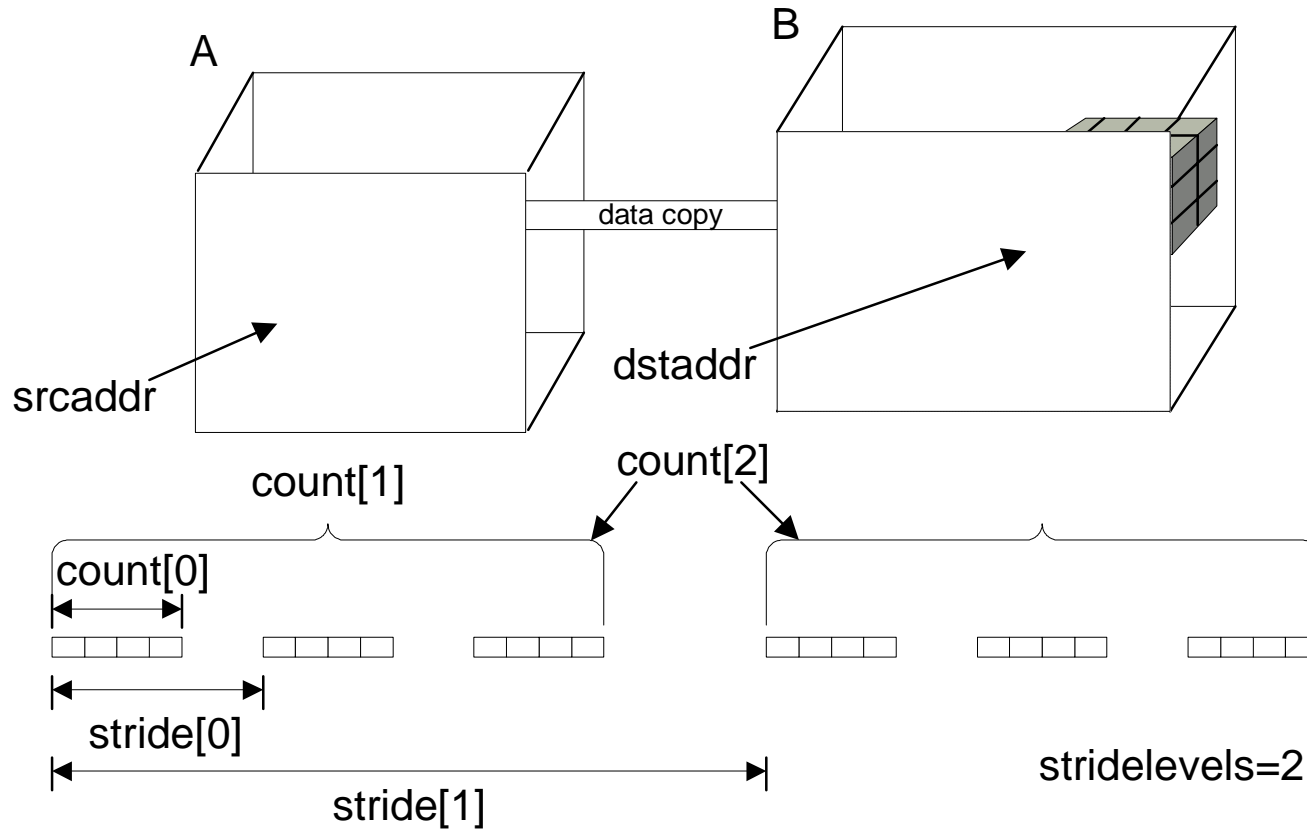
void **bupc\_memput\_strided** (shared void \*dstaddr, const size\_t dststrides[],  
const void \*srcaddr, const size\_t srcstrides[],  
const size\_t count[], size\_t stridelevels);

void **bupc\_memget\_strided** (void \*dstaddr, const size\_t dststrides[],  
shared const void \*srcaddr, const size\_t srcstrides[],  
const size\_t count[], size\_t stridelevels);

*(analogous async variants not shown)*



# UPC Strided Example





# UPC Strided Example



*Example: To put a 3-d block of data, shaped 2x3x4,  
starting at location (5, 6, 7) in A to B in location (8, 9, 10):*

```
double A[11][12][13]; /* local array */
```

```
shared [] double B[14][15][16]; /* remote array */
```

```
srcaddr = &(A[5][6][7]);
```

```
srcstrides[0] = 13 * sizeof(double); /* stride in bytes for the rightmost dimension */
```

```
srcstrides[1] = 12 * 13 * sizeof(double); /* stride in bytes for the middle dimension */
```

```
dstaddr = &(B[8][9][10]);
```

```
dststrides[0] = 16 * sizeof(double); /* stride in bytes for the rightmost dimension */
```

```
dststrides[1] = 15 * 16 * sizeof(double); /* stride in bytes for the middle dimension */
```

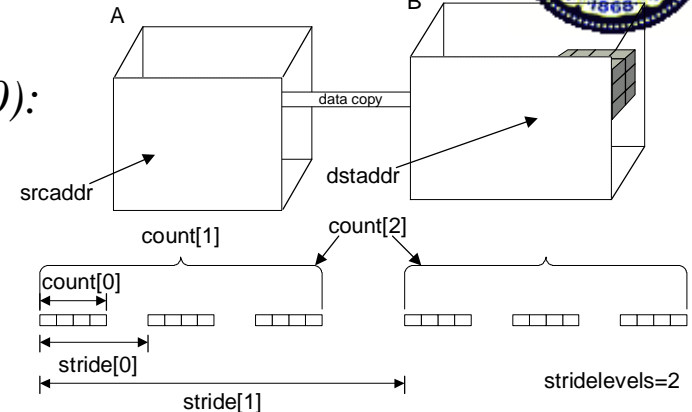
```
count[0] = 4 * sizeof(double); /* bytes of contig data (width in rightmost dimension) */
```

```
count[1] = 3; /* width in middle dimension */
```

```
count[2] = 2; /* width in leftmost dimension */
```

```
stridelevels = 2;
```

```
bupc_memput_strided ( srcaddr, dststrides, dstaddr, srcstrides, count, stridelevels );
```





# UPC-VIS Implementation Status

---



- Reference implementation complete, refinement underway
  - In terms of existing put/get (RDMA) - done
  - In terms of each other (eg strided over indexed) - done
  - In terms of GASNet Active Messages - in progress
  - Internally maintain many different algorithmic options to allow experimentation and tuning
  - Select algorithm based on hardware characteristics, transfer parameters (size, sparsity, etc) and current network status
- Berkeley UPC implements them by utilizing analogous extensions to the GASNet communication system interface
  - Translator can generate VIS calls - message coalescing
  - Berkeley UPC users can already call them as a library
  - Pushing for lang. acceptance of these library extensions



# UPC-VIS Future Work



- Network-specific implementations and hardware exploitation next
  - Use the reference implementation as a starting point
  - Directly leverage available hardware support to tune
  - Starting with Quadrics/Elan4 (FY04)
  - Move on to VIS support over Myrinet/MX, Infiniband/VAPI (local-side only), IBM/LAPI (software), Cray X1 (vector load/store)
- Investigate compiler-generated VIS calls
  - message coalescing, inspector/executor
- Performance evaluation & tuning of VIS - coming soon...
  - Microbenchmarks
  - Application-level benchmarks
    - Programmer-inserted calls to VIS functionality
    - Compiler-generated VIS calls
- Consider adding adopting these into the UPC standard library
  - based on performance results and user experience