UPC Collective Operations Specifications
V1.0

A publication of the UPC Consortium

December 12, 2003

Acknowledgements

Many have contributed to the ideas and concepts behind these specifications. Elizabeth Wiebel and David Greenberg are the authors of the first draft of this document. Steve Seidel organized the effort to refine that document into its current form. Thanks go to many in the UPC community for their interest and helpful comments, particularly Dan Bonachea, Bill Carlson, Jason Duell and Brian Wibecan. Tarek El-Ghazawi's efforts in organizing workshops provided an important forum for discussion of these specifications. Thanks also go to Lauren Smith for her efforts to support the development of this specification.

Members of the UPC consortium may be contacted via the world wide web at `http://www.upcworld.org` and `http://upc.gwu.edu` where an archived mailing list may be joined. Comments on these specifications are always welcome.

Contents

Introduction

1. The earliest version of this specification was authored by Elizabeth Wiebel and David Greenberg and appeared as CCS-TR-02-159 in March, 2002[2]. UPC collectives have been discussed at workshops at SC2001, SC2002, SC2003, and at workshops held at George Washington University in March, 2002, and May, 2003.

1 Scope

1. This document describes UPC functions that supplement UPC. All UPC specifications as per V1.1.1 [1] are considered part of this specification, and therefore will not be addressed in this document.

2. Small parts of UPC V1.1.1 may be repeated for self-containment and clarity of the functions defined here.

2 Definitions

1. **collective**:
   a requirement placed on some language operations which constrains invocations of such operations to be matched[1] across all threads. The behavior of collective operations is undefined unless all threads execute the same sequence of collective operations.

2. **single-valued**:
   is an operand to a collective operation, which has the same value on every thread. The behavior of the operation is otherwise undefined.

3. All but one of the functions defined in this specification include a figure that roughly illustrates how blocks of data are copied from one thread to another. Four threads labeled T_0, T_1, T_2, and T_3 are shown in each figure along with a suitable number of blocks of data labeled D_i. These figures are not intended to represent the full generality of the associated functions. The figures do not correspond to the example code segments unless otherwise noted. The figures and the example code segments should not be viewed as formal parts of this specification.

[1] A collective operation need not provide any actual synchronization between threads, unless otherwise noted. The collective requirement simply states a relative ordering property of calls to collective operations that must be maintained in the parallel execution trace for all executions of any legal program. Some implementations may include unspecified synchronization between threads within collective operations, but programs must not rely upon such unspecified synchronization for correctness.

3 Common requirements

1. The following requirements apply to all of the functions defined in this document whose names begin `upc_all_`... .

2. All of the functions are collective.

3. All collective function arguments are single-valued.

4. Collective functions may not be called between `upc_notify` and the corresponding `upc_wait`.

5. The last argument of each collective function is the variable `sync_mode` of type `upc_flag_t`. Values of `sync_mode` are formed by or-ing together a constant of the form `UPC_IN_`*X*`SYNC` and a constant of the form `UPC_OUT_`*Y*`SYNC`, where *X* and *Y* may be `NO`, `MY`, or `ALL`.

   If `sync_mode` has the value (`UPC_IN_`*X*`SYNC` | `UPC_OUT_`*Y*`SYNC`), then if *X* is

   `NO` the collective function may begin to read or write data when the first thread has entered the collective function call,

   `MY` the collective function may begin to read or write only data which has affinity to threads that have entered the collective function call, and

   `ALL` the collective function may begin to read or write data only after all threads have entered the collective function call[2]

   and if *Y* is

   `NO` the collective function may read and write data until the last thread has returned from the collective function call,

   `MY` the collective function call may return in a thread only after all reads and writes of data with affinity to the thread are complete[3], and

   `ALL` the collective function call may return only after all reads and writes of data are complete.[4]

[2] `UPC_IN_ALLSYNC` requires the collective function to guarantee that after all threads have entered the collective function call all threads will read the same values of the input data.

[3] `UPC_OUT_MYSYNC` requires the collective function to guarantee that after a thread returns from the collective function call the thread will not read any earlier values of the output data with affinity to that thread.

[4] `UPC_OUT_ALLSYNC` requires the collective function to guarantee that after a thread returns from the collective function call the thread will not read any earlier values of the output data. `UPC_OUT_ALLSYNC` is not required to provide an "implied" barrier. For example, if the entire collective operation has been completed by a certain thread before some other threads have reached their corresponding function calls, then that thread may exit its call.

UPC_IN_*X*SYNC alone is equivalent to (UPC_IN_*X*SYNC | UPC_OUT_ALLSYNC),
UPC_OUT_*X*SYNC alone is equivalent to (UPC_IN_ALLSYNC | UPC_OUT_*X*SYNC),
and 0 is equivalent to (UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC), where *X* is NO, MY, or ALL.

Forward reference: upc_flag_t (4.1.2).

4 Collectives library

4.1 Standard header

1. The standard header is

   <upc_collective.h>

2. upc_collective.h contains the definitions of the types upc_flag_t and upc_op_t. upc_flag_t and upc_op_t are of integral type.

   Forward reference: upc_op_t (4.3.1).

4.2 Relocalization Operations

4.2.1 The upc_all_broadcast function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_broadcast (shared void *dst, shared const void *src,
                           size_t nbytes, upc_flag_t sync_mode);
   nbytes:  the number of bytes in a block
   ```

Description

1. The upc_all_broadcast function copies a block of memory with affinity to a single thread to a block of shared memory on each thread. The number of bytes in each block is nbytes. If copying takes place between objects that overlap, the behavior is undefined.
2. nbytes must be strictly greater than 0.
3. The upc_all_broadcast function treats the src pointer as if it pointed to a shared memory area with the type:
   ```

`shared [] char[nbytes]`

The effect is equivalent to copying the entire array pointed to by `src` to each block of `nbytes` bytes of a shared array `dst` with the type:

`shared [nbytes] char[nbytes*THREADS]`

4. The target of the `dst` pointer must have affinity to thread 0.

5. The `dst` pointer is treated as if it has phase 0.

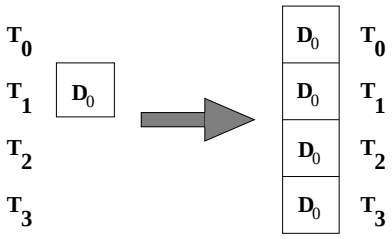

Figure 1. `upc_all_broadcast`.

Examples

1. This example corresponds to Figure 1.

```
shared int A[THREADS];
shared int B[THREADS];
// Initialize A.
upc_barrier;
upc_all_broadcast( B, &A[1], sizeof(int),
                   UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

2. Example of `upc_all_broadcast`.

```
#define NELEMS 10
shared [] int A[NELEMS];
shared [NELEMS] int B[NELEMS*THREADS];
// Initialize A.
upc_all_broadcast( B, A, sizeof(int)*NELEMS,
                   UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC );
```

3. In this example, (A[3],A[4]) is broadcast to (B[0],B[1]), (B[10],B[11]), (B[20],B[21]), ..., (B[NELEMS*(THREADS-1)],B[NELEMS*(THREADS-1)+1]).

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS];
shared [NELEMS] int B[NELEMS*THREADS];
// Initialize A.
upc_barrier;
upc_all_broadcast( B, &A[3], sizeof(int)*2,
                   UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

4.2.2 The upc_all_scatter function

Synopsis

1.
```
#include <upc.h>
#include <upc_collective.h>
void upc_all_scatter (shared void *dst, shared const void *src,
                      size_t nbytes, upc_flag_t sync_mode);
nbytes:  the number of bytes in a block
```

Description

1. The upc_all_scatter function copies the *i*th block of an area of shared memory with affinity to a single thread to a block of shared memory with affinity to the *i*th thread. The number of bytes in each block is nbytes. If copying takes place between objects that overlap, the behavior is undefined.

2. nbytes must be strictly greater than 0.

3. The upc_all_scatter function treats the src pointer as if it pointed to a shared memory area with the type:

   shared [] char[nbytes∗THREADS]

   and it treats the dst pointer as if it pointed to a shared memory area with the type:

   shared [nbytes] char[nbytes∗THREADS]

4. The target of the dst pointer must have affinity to thread 0.

5. The `dst` pointer is treated as if it has phase 0.

6. For each thread *i*, the effect is equivalent to copying the *i*th block of `nbytes` bytes pointed to by `src` to the block of `nbytes` bytes pointed to by `dst` that has affinity to thread *i*.

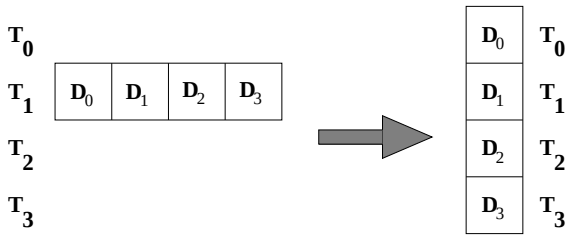

Figure 2. `upc_all_scatter`.

Examples

1. Example of `upc_all_scatter` for the dynamic threads compilation environment. This example corresponds to Figure 2.

```
#define NUMELEMS 10
#define SRC_THREAD 1
shared int *A;
shared [] int *myA, *srcA;
shared [NUMELEMS] int B[NUMELEMS*THREADS];

// allocate and initialize an array distributed across all threads
A = upc_all_alloc(THREADS, THREADS*NUMELEMS*sizeof(int));
myA = (shared [] int *) &A[MYTHREAD];
for (i=0; i<NUMELEMS*THREADS; i++)
    myA[i] = i + NUMELEMS*THREADS*MYTHREAD;   // (for example)
// scatter the SRC_THREAD's row of the array
srcA = (shared [] int *) &A[SRC_THREAD];
upc_barrier;
upc_all_scatter( B, srcA, sizeof(int)*NUMELEMS,
                 UPC_IN_NOSYNC | UPC_OUT_NOSYNC);
upc_barrier;
```

2. Example of upc_all_scatter for the static threads compilation environment.

```
#define NELEMS 10
shared [] int A[NELEMS*THREADS];
shared [NELEMS] int B[NELEMS*THREADS];
// Initialize A.
upc_all_scatter( B, A, sizeof(int)*NELEMS,
                 UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC );
```

4.2.3 The upc_all_gather function

Synopsis

1.
```
#include <upc.h>
#include <upc_collective.h>
void upc_all_gather (shared void *dst, shared const void *src,
                     size_t nbytes, upc_flag_t sync_mode);
nbytes:  the number of bytes in a block
```

Description

1. The upc_all_gather function copies a block of shared memory that has affinity to the *i*th thread to the *i*th block of a shared memory area that has affinity to a single thread. The number of bytes in each block is nbytes. If copying takes place between objects that overlap, the behavior is undefined.
2. nbytes must be strictly greater than 0.
3. The upc_all_gather function treats the src pointer as if it pointed to a shared memory area of nbytes bytes on each thread and therefore had type:

   `shared [nbytes] char[nbytes*THREADS]`

   and it treats the dst pointer as if it pointed to a shared memory area with the type:

   `shared [] char[nbytes*THREADS]`
4. The target of the src pointer must have affinity to thread 0.
5. The src pointer is treated as if it has phase 0.
6. For each thread *i*, the effect is equivalent to copying the block of nbytes bytes pointed to by src that has affinity to thread *i* to the *i*th block of nbytes bytes pointed to by dst.

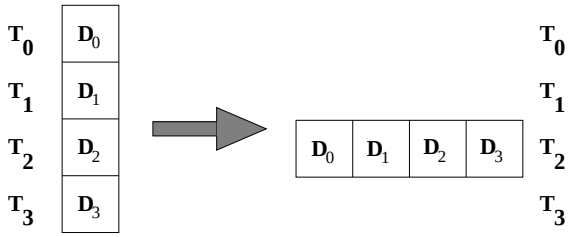

Figure 3. upc_all_gather.

Examples

1. Example of upc_all_gather for the static threads compilation environment.

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS];
shared [] int B[NELEMS*THREADS];
// Initialize A.
upc_all_gather( B, A, sizeof(int)*NELEMS,
                UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC );
```

2. Example of upc_all_gather for the dynamic threads compilation environment.

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS];
shared [] int *B;
B = (shared [] int *) upc_all_alloc(1,NELEMS*THREADS*sizeof(int));
// Initialize A.
upc_barrier;
upc_all_gather( B, A, sizeof(int)*NELEMS,
                UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

4.2.4 The `upc_all_gather_all` function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_gather_all (shared void *dst, shared const void *src,
                            size_t nbytes, upc_flag_t sync_mode);
   nbytes:  the number of bytes in a block
   ```

Description

1. The `upc_all_gather_all` function copies a block of memory from one shared memory area with affinity to the *i*th thread to the *i*th block of a shared memory area on each thread. The number of bytes in each block is `nbytes`. If copying takes place between objects that overlap, the behavior is undefined.
2. `nbytes` must be strictly greater than 0.
3. The `upc_all_gather_all` function treats the `src` pointer as if it pointed to a shared memory area of `nbytes` bytes on each thread and therefore had type:

   `shared [nbytes] char[nbytes*THREADS]`

   and it treats the `dst` pointer as if it pointed to a shared memory area with the type:

   `shared [nbytes*THREADS] char[nbytes*THREADS*THREADS]`
4. The targets of the `src` and `dst` pointers must have affinity to thread 0.
5. The `src` and `dst` pointers are treated as if they have phase 0.
6. The effect is equivalent to copying the *i*th block of `nbytes` bytes pointed to by `src` to the *i*th block of `nbytes` bytes pointed to by `dst` that has affinity to each thread.
   ```

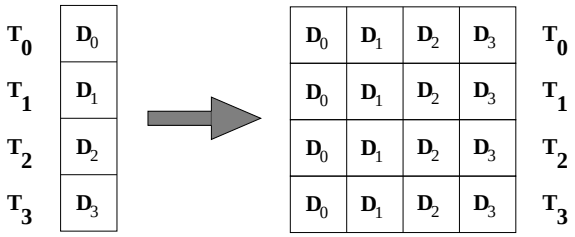

Figure 4. upc_all_gather_all.

Examples

1. Example of upc_all_gather_all for the static threads compilation environment.

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS];
shared [NELEMS*THREADS] int B[THREADS][NELEMS*THREADS];
// Initialize A.
upc_barrier;
upc_all_gather_all( B, A, sizeof(int)*NELEMS,
                    UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

2. Example of upc_all_gather_all for the dynamic threads compilation environment.

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS];
shared int *Bdata;
shared [] int *myB;

Bdata = upc_all_alloc(THREADS*THREADS, NELEMS*sizeof(int));
myB = (shared [] int *)&Bdata[MYTHREAD];

// Bdata contains THREADS*THREADS*NELEMS elements.
// myB is MYTHREAD's row of Bdata.
// Initialize A.
upc_all_gather_all( Bdata, A, NELEMS*sizeof(int),
                    UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC );
```

4.2.5 The `upc_all_exchange` function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_exchange (shared void *dst, shared const void *src,
                          size_t nbytes, upc_flag_t sync_mode);
   nbytes:  the number of bytes in a block
   ```

Description

1. The `upc_all_exchange` function copies the *i*th block of memory from a shared memory area that has affinity to thread *j* to the *j*th block of a shared memory area that has affinity to thread *i*. The number of bytes in each block is `nbytes`. If copying takes place between objects that overlap, the behavior is undefined.
2. `nbytes` must be strictly greater than 0.
3. The `upc_all_exchange` function treats the `src` pointer and the `dst` pointer as if each pointed to a shared memory area of `nbytes*THREADS` bytes on each thread and therefore had type:

   ```
   shared [nbytes*THREADS] char[nbytes*THREADS*THREADS]
   ```
4. The targets of the `src` and `dst` pointers must have affinity to thread 0.
5. The `src` and `dst` pointers are treated as if they have phase 0.
6. For each pair of threads *i* and *j*, the effect is equivalent to copying the *i*th block of `nbytes` bytes that has affinity to thread *j* pointed to by `src` to the *j*th block of `nbytes` bytes that has affinity to thread *i* pointed to by `dst`.
   ```

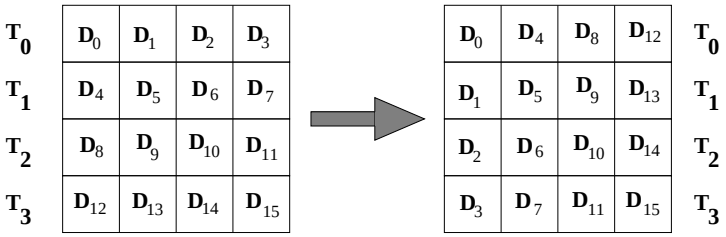

Figure 5. upc_all_exchange.

Examples

1. Example of upc_all_exchange for the static threads compilation environment.

```
#define NELEMS 10
shared [NELEMS*THREADS] int A[THREADS][NELEMS*THREADS];
shared [NELEMS*THREADS] int B[THREADS][NELEMS*THREADS];
// Initialize A.
upc_barrier;
upc_all_exchange( B, A, NELEMS*sizeof(int),
                  UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

2. Example of upc_all_exchange for the dynamic threads compilation environment.

```
#define NELEMS 10
shared int *Adata, *Bdata;
shared [] int *myA, *myB;
int i;

Adata = upc_all_alloc(THREADS*THREADS, NELEMS*sizeof(int));
myA = (shared [] int *)&Adata[MYTHREAD];
Bdata = upc_all_alloc(THREADS*THREADS, NELEMS*sizeof(int));
myB = (shared [] int *)&Bdata[MYTHREAD];

// Adata and Bdata contain THREADS*THREADS*NELEMS elements.
// myA and myB are MYTHREAD's rows of Adata and Bdata, resp.

// Initialize MYTHREAD's row of A.  For example,
for (i=0; i<NELEMS*THREADS; i++)
    myA[i] = MYTHREAD*10 + i;

upc_all_exchange( Bdata, Adata, NELEMS*sizeof(int),
                  UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC );
```

4.2.6 The upc_all_permute function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_permute (shared void *dst, shared const void *src,
                         shared const int *perm, size_t nbytes,
                         upc_flag_t sync_mode);
   nbytes:  the number of bytes in a block
   ```

Description

1. The upc_all_permute function copies a block of memory from a shared memory area that has affinity to the *i*th thread to a block of a shared memory that has affinity to thread
   ```

perm[i]. The number of bytes in each block is nbytes. If copying takes place between objects that overlap, the behavior is undefined.

2. nbytes must be strictly greater than 0.
3. perm[0..THREADS−1] must contain THREADS distinct values: 0, 1, ..., THREADS −1.
4. The upc_all_permute function treats the src pointer and the dst pointer as if each pointed to a shared memory area of nbytes bytes on each thread and therefore had type:
   shared [nbytes] char[nbytes∗THREADS]
5. The targets of the src, perm, and dst pointers must have affinity to thread 0.
6. The src and dst pointers are treated as if they have phase 0.
7. The effect is equivalent to copying the block of nbytes bytes that has affinity to thread *i* pointed to by src to the block of nbytes bytes that has affinity to thread perm[*i*] pointed to by dst.

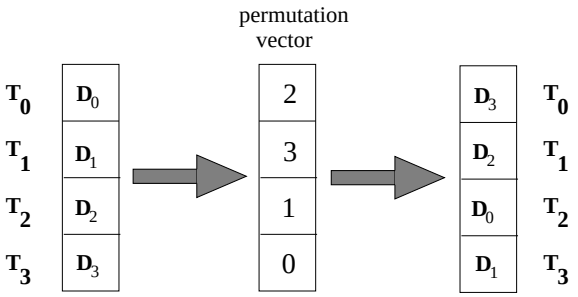

Figure 6. upc_all_permute**.**

Example

1. Example of upc_all_permute.

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS], B[NELEMS*THREADS];
shared int P[THREADS];
// Initialize A and P.
upc_barrier;
upc_all_permute( B, A, P, sizeof(int)*NELEMS,
                 UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

4.3 Computational Operations

4.3.1 Computational operations

1. A variable of type `upc_op_t` can have the following values:

   `UPC_ADD` Addition.

   `UPC_MULT` Multiplication.

   `UPC_AND` Bitwise `AND` for integer and character variables. Results are undefined for floating point numbers.

   `UPC_OR` Bitwise `OR` for integer and character variables. Results are undefined for floating point numbers.

   `UPC_XOR` Bitwise `XOR` for integer and character variables. Results are undefined for floating point numbers.

   `UPC_LOGAND` Logical `AND` for all variable types.

   `UPC_LOGOR` Logical `OR` for all variable types.

   `UPC_MIN` For all data types, find the minimum value.

   `UPC_MAX` For all data types, find the maximum value.

   `UPC_FUNC` Use the specified commutative function `func` to operate on the data in the `src` array at each step.

   `UPC_NONCOMM_FUNC` Use the specified noncommutative function `func` to operate on the data in the `src` array at each step.

2. The operations represented by a variable of type `upc_op_t` (including user-provided operators) are assumed to be associative. A reduction or a prefix reduction whose result is dependent on the order of operator evaluation will have undefined results.[5]

   Forward references: reduction (4.3.2), prefix reduction (4.3.3).

3. The operations represented by a variable of type `upc_op_t` (except those provided using `UPC_NONCOMM_FUNC`) are assumed to be commutative. A reduction or a prefix reduction (using operators other than `UPC_NONCOMM_FUNC`) whose result is dependent on the order of the operands will have undefined results.

   Forward references: reduction (4.3.2), prefix reduction (4.3.3).

[5] Implementations are not obligated to prevent failures that might arise because of a lack of associativity of built-in functions due to floating-point roundoff or overflow.

4.3.2 The `upc_all_reduce` function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_reduceT(shared void *dst, shared const void *src,
                        upc_op_t op, size_t nelems, size_t blk_size,
                        TYPE (*func)(TYPE, TYPE), upc_flag_t sync_mode);
   nelems:  the number of elements
   blk_size:  the number of elements in a block
   ```

Description

1. The function prototype above represents the 11 variations of the `upc_all_reduceT` function where *T* and *TYPE* have the following correspondences:

T	*TYPE*	*T*	*TYPE*
C	signed char	L	signed long
UC	unsigned char	UL	unsigned long
S	signed short	F	float
US	unsigned short	D	double
I	signed int	LD	long double
UI	unsigned int		

   For example, if *T* is `C`, then *TYPE* must be `signed char`.

2. If the value of `blk_size` passed to `upc_all_reduceT` is greater than 0 then `upc_all_reduceT` treats the `src` pointer as if it pointed to a shared memory area of `nelems` elements of type *TYPE* and blocking factor `blk_size`, and therefore had type:

   `shared [blk_size] TYPE[nelems]`

3. If the value of `blk_size` passed to `upc_all_reduceT` is 0 then `upc_all_reduceT` treats the `src` pointer as if it pointed to a shared memory area of `nelems` elements of type *TYPE* with an indefinite layout qualifier, and therefore had type[6]:

   `shared [] TYPE[nelems]`

[6] Note that `upc_blocksize(src) == 0` if `src` has this type, so the argument value 0 has a natural connection to the block size of `src`.
   ```

4. The phase of the `src` pointer is respected when referencing array elements, as specified in items 2 and 3 above.

5. `upc_all_reduce`*T* treats the `dst` pointer as having type:

   `shared` *TYPE* `*`

   and at function exit the value of the *TYPE* shared object referenced by `dst` is

   `src[0]` $\oplus$ `src[1]` $\oplus \cdots \oplus$ `src[nelems-1]`

   where "$\oplus$" is the operator specified by the variable `op`.

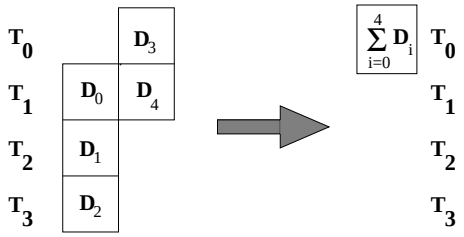

Figure 7. `upc_all_reduce` **with addition operator.**

Example

1. Example of `upc_all_reduce` of type `long UPC_ADD`.

```
#define BLK_SIZE 3
#define NELEMS 10
shared [BLK_SIZE] long A[NELEMS*THREADS];
shared long *B;
long result;
// Initialize A.  The result below is defined only on thread 0.
upc_barrier;
upc_all_reduceL( B, A, UPC_ADD, NELEMS*THREADS, BLK_SIZE,
                 NULL, UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

4.3.3 The `upc_all_prefix_reduce` function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_prefix_reduceT(shared void *dst, shared const void *src
                               upc_op_t op, size_t nelems, size_t blk_size,
                               TYPE (*func)(TYPE, TYPE),
                               upc_flag_t sync_mode);
   nelems:  the number of elements
   blk_size:  the number of elements in a block
   ```

Description

1. The function prototype above represents the 11 variations of the `upc_all_reduceT` function where *T* and *TYPE* have the following correspondences:

T	*TYPE*	*T*	*TYPE*
C	signed char	L	signed long
UC	unsigned char	UL	unsigned long
S	signed short	F	float
US	unsigned short	D	double
I	signed int	LD	long double
UI	unsigned int		

   For example, if *T* is `C`, then *TYPE* must be `signed char`.

2. If the value of `blk_size` passed to `upc_all_prefix_reduceT` is greater than 0 then `upc_all_prefix_reduceT` treats the `src` pointer and the `dst` pointer as if each pointed to a shared memory area of `nelems` elements of type *TYPE* and blocking factor `blk_size`, and therefore had type:

   `shared [blk_size] TYPE[nelems]`

3. If the value of `blk_size` passed to `upc_all_prefix_reduceT` is 0 then `upc_all_prefix_reduceT` treats the `src` pointer and the `dst` pointer as if each pointed to a shared memory area of `nelems` elements of type *TYPE* with an indefinite layout qualifier, and therefore had type[7]:

   `shared [] TYPE[nelems]`

[7] Note that `upc_blocksize(src) == 0` if `src` has this type, so the argument value 0 has a natural connection to the block size of `src`.
   ```

4. The phases of the `src` and `dst` pointers are respected when referencing array elements, as specified in items 2 and 3 above.

5. If the memory areas pointed to by `src` and `dst` overlap, the behavior of this function is undefined.

6. At function exit

   `dst[i]` = `src[0]` $\oplus$ `src[1]` $\oplus \cdots \oplus$ `src[i]`

   for $0 \leq$ `i` $\leq$ `nelems-1` and where "$\oplus$" is the operator specified by the variable `op`.

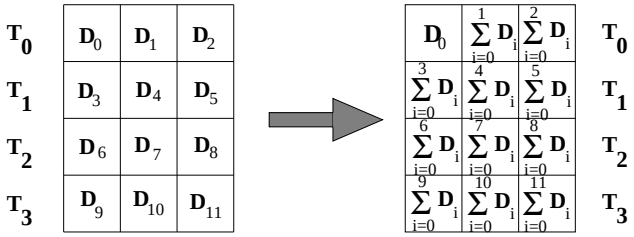

Figure 8. `upc_all_prefix_reduce` **with addition operator. The D_i's are scalars of type** *TYPE*.

Example

1. Example of `upc_all_prefix_reduce` of type `long UPC_ADD`.

```
#define BLK_SIZE 3
#define NELEMS 10
shared [BLK_SIZE] long A[NELEMS*THREADS];
shared [BLK_SIZE] long B[NELEMS*THREADS];
// Initialize A.
upc_all_prefix_reduceL( B, A, UPC_ADD, NELEMS*THREADS, BLK_SIZE,
                        NULL, UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC );
```

4.3.4 The `upc_all_sort` function

Synopsis

1. ```
   #include <upc.h>
   #include <upc_collective.h>
   void upc_all_sort (shared void *A, size_t elem_size, size_t nelems,
            size_t blk_size, int (*func)(shared void *, shared void *),
            upc_flag_t sync_mode);
   elem_size:  the size of each element
   nelems:  the number of elements
   blk_size:  the number of elements in a block
   ```

Description

1. The function `upc_all_sort` takes a shared array `A` of `nelems` elements of size `elem_size` bytes each and sorts them in place in ascending order using the function `func` to compare elements.

2. If the value of `blk_size` passed to `upc_all_sort` is greater than 0 then `upc_all_sort` treats the array `A` as if it had blocking factor `blk_size`.

3. If the value of `blk_size` passed to `upc_all_sort` is 0 then `upc_all_sort` treats the array `A` as if it had an indefinite layout qualifier.

4. The phase of the `A` is respected when referencing array elements, as specified in items 2 and 3 above.

5. The function `func` shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

6. The function `upc_all_sort` performs a *stable* sort, that is, elements which compare equal are not reordered.
   ```

Example

1. Example of upc_all_sort.

```
#define NELEMS 10
shared [NELEMS] int A[NELEMS*THREADS];

int lt_int( shared void *x, shared void *y )
{
   int x_val = *(shared int *)x, y_val = *(shared int *)y;
   return x_val > y_val ? -1 : x_val < y_val ? 1 : 0;
}

// Initialize A.
upc_barrier;
upc_all_sort( A, sizeof(int), NELEMS*THREADS, NELEMS,
              lt_int, UPC_IN_NOSYNC | UPC_OUT_NOSYNC );
upc_barrier;
```

References

[1] T. El-Ghazawi, W. Carlson, and J. Draper. UPC language specifications V1.1.1. Technical report, George Washington University and IDA Center for Computing Sciences, October 7 2003. http://www.gwu.edu/~upc/documentation.html.

[2] E. Wiebel and D. Greenberg. Collective operations in UPC. Technical Report CCS TR-02-159, IDA Center for Computing Sciences, March 2002.