UPC Collective Operations Specification pre4V1.0

Elizabeth Wiebel, David Greenberg
Center for Computing Sciences
Bowie, Maryland

Steven Seidel
Department of Computer Science
Michigan Technological University
Houghton, Michigan

April 2, 2003
Acknowledgements

Thanks go to many in the UPC community for their interest and helpful comments, particularly Brian Wibecan, Dan Bonachea, and Bill Carlson. Tarek El-Ghazawi’s efforts in organizing workshops provided an important forum for discussion of these specifications. UPC collectives have been discussed at workshops at SC2001, SC2002, and in March 2002. Thanks also go to Lauren Smith for her efforts to support the development of this specification.
Contents

Introduction 4

1 Scope 4

2 Normative references 4

3 Definitions 4

4 Common requirements 5

5 Collectives library 5

5.1 Standard header ................................................. 5

5.2 Initialization ....................................................... 6

5.2.1 The upc_all_init function .................................... 6

5.3 Relocalization Operations ...................................... 6

5.3.1 The upc_all_broadcast function ............................. 6

5.3.2 The upc_all_scatter function ................................ 8

5.3.3 The upc_all_gather function ................................ 10

5.3.4 The upc_all_gather_all function ......................... 11

5.3.5 The upc_all_exchange function ............................. 13

5.3.6 The upc_all_permute function ............................. 15

5.4 Computational Operations ............................... 17

5.4.1 The upc_all_reduce function ............................. 17

5.4.2 The upc_all_prefix_reduce function .................... 19

5.4.3 The upc_all_sort function ............................. 22

References 23
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]. That version was discussed at the UPC workshop held in Washington, DC, in March, 2002.

2. V0.2 of this specification was discussed at the SC2002 UPC workshop at Baltimore, Maryland, in November, 2002.

1 Scope

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

2. Small parts of UPC V1.1 may be repeated for self-containment and clarity of the functions defined here.

2 Normative references

1. The section numbering of this document has no correspondence to that of UPC specification V1.1.

3 Definitions

1. collective:
   a requirement placed on some language operations which constrains invocations of such operations to be matched\textsuperscript{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.

\textsuperscript{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. 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.

4 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 function arguments are single-valued.
4. This document explicitly states which function arguments must have affinity to thread 0. All other function arguments may have affinity to any thread.
5. This document does not place any restrictions on the phase of function arguments.
6. Any thread executing a collective function may modify the memory areas associated with its arguments during its execution, unless it is prohibited in this specification. Therefore, all references to those memory areas made by other threads that are not currently executing the same collective function are undefined.
7. If a thread that is not executing a collective function reads or writes to a memory area associated with the arguments of a call to a collective function after any thread has begun execution of that function or before any thread has completed execution of that function, then the result of that read or write is undefined and the results of the collective function are undefined.

5 Collectives library

5.1 Standard header

1. The standard header is

   `<upc_collective.h>`

5
5.2 Initialization

5.2.1 The upc_all_init function

Synopsis

1. #include <upc.h>
   #include <upc_collective.h>
   void upc_all_init ();

Description

1. This is a collective function.

2. In a program which calls any of the collective functions described in this document, each thread must call this function exactly once.

3. Each thread’s call to this function must precede all of its calls to other collective functions described in this document.

5.3 Relocalization Operations

5.3.1 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);
   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. 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]

3. The dst array must have affinity to thread 0.

![Diagram](image)

**Figure 1.** upc_all_broadcast.

**Examples**

1. This example corresponds to Figure 1.

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

2. Example of upc_all_broadcast.

```c
#define NELEMS 10
shared [ ] int A[NELEMS];
shared [NELEMS] int B[NELEMS*THREADS];
// Initialize A.
upc_barrier;
upc_all_broadcast( B, A, sizeof(int)*NELEMS );
upc_barrier;
```
3. In this example, \((A[3], A[4])\) is broadcast to array destinations \((B[0], B[1]), (B[10], B[11]), (B[20], B[21]), \ldots, (B[\text{NELEMS} \times (\text{THREADS} - 1)], B[\text{NELEMS} \times (\text{THREADS} - 1) + 1])\).

```c
#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_barrier;
```

5.3.2 The `upc_all_scatter` function

**Synopsis**

1. `#include <upc.h>`
   `#include <upc_collective.h>`
   ```c
   void upc_all_scatter (shared void *dst,
                      shared const void *src, size_t nbytes);
   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. The `upc_all_scatter` function treats the `src` pointer as if it pointed to a shared memory area with the type:
   ```c
   shared [] char[nbytes*THREADS]
   ```
   and it treats the `dst` pointer as if it pointed to a shared memory area with the type:
   ```c
   shared [nbytes] char[nbytes*THREADS]
   ```

3. The `dst` array must have affinity to thread 0.

4. 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 \(i\).
Examples

1. Example of `upc_all_scatter` for the static threads compilation environment.

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

2. Example of `upc_all_scatter` for the dynamic threads compilation environment.

   ```c
   #define NELEMS 10
   shared [] int *A;
   shared [NELEMS] int B[NELEMS*THREADS];
   A = (shared [] int *) upc_all_alloc(1,NELEMS*THREADS*sizeof(int));
   // Initialize A.
   upc_barrier;
   upc_all_scatter( B, A, sizeof(int)*NELEMS );
   upc_barrier;
   ```
5.3.3 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);
   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. 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]

3. The src array must have affinity to thread 0.

4. 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.

![Diagram](image)

**Figure 3.** upc_all_gather.
Examples

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

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

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

```c
#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_barrier;
```

5.3.4 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);

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. 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]

3. The src and dst arrays must have affinity to thread 0.

4. 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 that has affinity to thread $i$ pointed to by dst.

![Diagram](image)

**Figure 4.** upc_all_gather_all.

**Examples**

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

```c
#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_barrier;
```
2. Example of `upc_all_gather_all` for the dynamic threads compilation environment.

```c
#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_barrier;
upc_all_gather_all( Bdata, A, NELEMS*sizeof(int) );
upc_barrier;
```

### 5.3.5 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);
   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. 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:
3. The src and dst arrays must have affinity to thread 0.

4. For each pair of threads $i$ and $j$, the effect is equivalent to copying the $i$th block of $n$ bytes that has affinity to thread $j$ pointed to by src to the $j$th block of $n$ bytes that has affinity to thread $i$ pointed to by dst.

![Figure 5. upc_all_exchange.](image)

Examples

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

```c
#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_barrier;
```
2. Example of upc_all_exchange for the dynamic threads compilation environment.

```c
#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_barrier;
upc_all_exchange( Bdata, Adata, NELEMS*sizeof(int) );
upc_barrier;
```

5.3.6 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);  

    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 ith 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. perm[0..THREADS−1] must contain THREADS distinct values: 0, 1, ..., THREADS −1.

3. 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]

4. The src, perm, and dst arrays must have affinity to thread 0.

5. 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.](image)

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_barrier;
   ```
5.4 Computational Operations

5.4.1 The upc_all_reduce function

Synopsis

1. #include <upc.h>
   #include <upc_collective.h>
   TYPE upc_all_reduceT(shared const void *src, UPC_OP op, size_t nelems,
   size_t blk_size, TYPE (*func)(TYPE, TYPE));

   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:

<table>
<thead>
<tr>
<th>T</th>
<th>TYPE</th>
<th>T</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>{signed} char</td>
<td>L</td>
<td>{signed} long</td>
</tr>
<tr>
<td>UC</td>
<td>unsigned char</td>
<td>UL</td>
<td>unsigned long</td>
</tr>
<tr>
<td>S</td>
<td>{signed} short</td>
<td>F</td>
<td>float</td>
</tr>
<tr>
<td>US</td>
<td>unsigned short</td>
<td>D</td>
<td>double</td>
</tr>
<tr>
<td>I</td>
<td>{signed} int</td>
<td>LD</td>
<td>long double</td>
</tr>
<tr>
<td>UI</td>
<td>unsigned int</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   For example, if T is C, then TYPE must be signed char or 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^2:

   shared [] TYPE[nelems]

^2Note 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 src array must have affinity to thread 0.

5. The variable op can take one of the following values:

   **UPC_ADD** For signed character, integer or floating point variables, regular addition. For unsigned character variables, addition of the characters’ ASCII values.

   **UPC_MULT** For signed character, integer or floating point variables, regular multiplication. For unsigned character variables, multiplication of the characters’ ASCII values.

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

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

   **UPC_XOR** Bitwise XOR for integer and character variables. Results 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. Unsigned characters are compared using ASCII values.

   **UPC_MAX** For all data types, find the maximum value. Unsigned characters are compared using ASCII values.

   **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. Providing for a user-defined noncommutative function allows operations such as subtraction and division that do not give reduction answers that are well-defined.

6. At function exit the value returned to thread 0 is

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

   where “\(\oplus\)” is the operator specified by the variable op. The value returned to all other threads is undefined.
Example

1. Example of upc_all_reduce of type long UPC_ADD.

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

5.4.2 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 op, size_t nelems, size_t blk_size,
                            TYPE (*func)(TYPE, TYPE));

   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 or 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\(^3\):
   shared [] TYPE[nelems]

4. The src and dst arrays must have affinity to thread 0.

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

6. The variable op can take one of the following values:

   UPC_ADD For signed character, integer or floating point variables, regular addition. For unsigned character variables, addition of the characters’ ASCII values.

   UPC_MULT For signed character, integer or floating point variables, regular multiplication. For unsigned character variables, multiplication of the characters’ ASCII values.

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

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

\(^3\)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.
**UPC\_XOR**  Bitwise XOR for integer and character variables. Results 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. Unsigned characters are compared using ASCII values.

**UPC\_MAX**  For all data types, find the maximum value. Unsigned characters are compared using ASCII values.

**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. Providing for a user-defined noncommutative function allows operations such as subtraction and division that do not give reduction answers that are well-defined.

7. At function exit

\[
\text{dst}[i] = \text{src}[0] \oplus \text{src}[1] \oplus \cdots \oplus \text{src}[i]
\]

for \(0 \leq i \leq \text{nelems-}1\) 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`.

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

5.4.3 The `upc_all_sort` function

Synopsis

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

   ```c
   void upc_all_sort (shared void *A, size_t elem_size, size_t nelems,
                      size_t blk_size, int (*func)(shared void *, shared void *));
   ```

   *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 function `func(x, y)` returns -1, 0, or 1 depending on whether `x > y`, `x == y`, or `x < y`, respectively.
5. 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`.

```c
#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_barrier;
```

**References**
