UPC Collective Operations Specification pre3V1.0

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

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

February 14, 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.
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: an operation which all threads execute “together”. The behavior of collective operations is undefined unless all threads execute the same sequence of collective operations. For operations which are function calls, the behavior is undefined unless all threads pass identical arguments.

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 intended to be supplemental only. They do not represent the full generality of the associated functions. They should not be viewed as a formal part 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 arguments to the functions are single-valued.

4. All arguments named src and dst have affinity to thread 0.

5. 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 executing the same collective function are undefined.

6. 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, 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>`

5.2 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. 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 \texttt{upc\_all\_broadcast} function

Synopsis

1. \texttt{#include <upc.h>}
   \texttt{#include <upc\_collective.h>}
   \texttt{void upc\_all\_broadcast (shared void \*dst,}
   \texttt{ shared const void \*src, size\_t blk);}

Description

1. The \texttt{upc\_all\_broadcast} function copies a block of memory with affinity to thread 0 to a block of shared memory on each thread. The number of bytes in each block is \texttt{blk}. If copying takes place between objects that overlap, the behavior is undefined.

2. The \texttt{upc\_all\_broadcast} function treats the \texttt{src} pointer as if it pointed to a shared memory area with affinity to thread 0 and therefore had type:
   \texttt{shared [] char[blk]}

   The effect is equivalent to copying the entire array pointed to by \texttt{src} to each block of \texttt{blk} bytes of a shared array \texttt{dst} with the type:
   \texttt{shared [blk] char[blk*THREADS]}
Example

1. Example of upc_all_broadcast.

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

5.3.2 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 blk);

Description

1. The upc_all_scatter function copies the $i$th block of an area of shared memory with affinity to thread 0 to a block of shared memory with affinity to the $i$th thread. The number of bytes in each block is $blk$. 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 affinity to thread 0 and therefore had type:
shared [] char[blk*THREADS]
and it treats the dst pointer as if it pointed to a shared memory area with the type:
shared [blk] char[blk*THREADS]
For each thread $i$, the effect is equivalent to copying the $i$th block of blk bytes pointed to by src to the block of blk 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 static threads compilation environment.

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

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

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

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 thread 0. The number of bytes in each block is \(blk\). 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 \(blk\) bytes on each thread and therefore had type:
   
   \[
   \text{shared} \ [blk] \ \text{char}[blk \times \text{THREADS}]
   \]

   and it treats the dst pointer as if it pointed to a shared memory area with affinity to thread 0 and therefore had type:

   \[
   \text{shared} \ [\] \ \text{char}[blk \times \text{THREADS}]
   \]

   For each thread \(i\), the effect is equivalent to copying the block of \(blk\) bytes pointed to by src that has affinity to thread \(i\) to the \(i\)th block of \(blk\) bytes pointed to by dst.

![Figure 3. upc_all_gather.](image)
Examples

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

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

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

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

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 \texttt{blk}. If copying takes place between objects that overlap, the behavior is undefined.

2. The \texttt{upc\_all\_gather\_all} function treats the \texttt{src} pointer as if it pointed to a shared memory area of \texttt{blk bytes} on each thread and therefore had type:
   
   \begin{verbatim}
   shared [blk] char[blk\*THREADS]
   \end{verbatim}

   and it treats the \texttt{dst} pointer as if it pointed to a shared memory area with the type:
   
   \begin{verbatim}
   shared [blk\*THREADS] char[blk\*THREADS\*THREADS]
   \end{verbatim}

   For each thread \(i\), the effect is equivalent to copying the block of \texttt{blk} bytes pointed to by \texttt{src} that has affinity to thread \(i\) to the \(i\)th block of \texttt{blk} bytes that has affinity to thread \(i\) pointed to by \texttt{dst}.

   \begin{figure}
   \centering
   \includegraphics[width=\textwidth]{figure4.png}
   \caption{\texttt{upc\_all\_gather\_all}.}
   \end{figure}
Examples

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

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

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

```c
#define numelems 10
shared [numelems] int A[numelems*THREADS];
shared int *Bdata;
shared [] int *myB;
Bdata = upc_all_alloc(THREADS*THREADS, numelems*sizeof(int));
myB = (shared [] int *)&Bdata[MYTHREAD];

// Bdata contains THREADS*THREADS*numelems elements.
// myB is MYTHREAD’s row of Bdata.
// Initialize A.
upc_barrier;
upc_all_gather_all( Bdata, A, numelems*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 blk);`
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 \( \text{blk} \). If copying takes place between objects that overlap, the behavior is undefined.

2. The upc_all_exchange function treats the \text{src} pointer and the \text{dst} pointer as if each pointed to a shared memory area of \( \text{blk} \times \text{THREADS} \) bytes on each thread and therefore had type:

   \[
   \text{shared} \left[ \text{blk} \times \text{THREADS} \right] \text{ char} \left[ \text{blk} \times \text{THREADS} \times \text{THREADS} \right]
   \]

   For each pair of threads \( i \) and \( j \), the effect is equivalent to copying the \( i \)th block of \( \text{blk} \) bytes that has affinity to thread \( j \) pointed to by \text{src} to the \( j \)th block of \( \text{blk} \) bytes that has affinity to thread \( i \) pointed to by \text{dst}.

   \[
   \begin{array}{cccc}
   T_0 & D_0 & D_1 & D_2 & D_3 \\
   T_1 & D_4 & D_5 & D_6 & D_7 \\
   T_2 & D_8 & D_9 & D_{10} & D_{11} \\
   T_3 & D_{12} & D_{13} & D_{14} & D_{15} \\
   \end{array}
   \Rightarrow
   \begin{array}{cccc}
   T_0 & D_0 & D_4 & D_8 & D_{12} \\
   T_1 & D_5 & D_9 & D_{13} & \\
   T_2 & D_6 & D_{10} & D_{14} & \\
   T_3 & D_{11} & D_{15} & \\
   \end{array}
   \]

   \textbf{Figure 5. upc\_all\_exchange.}
Examples

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

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

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

```c
#define numelems 10
shared int *Adata, *Bdata;
shared [] int *myA, *myB;
int i;
Adata = upc_all_alloc(THREADS*THREADS, numelems*sizeof(int));
myA = (shared [] int *)&Adata[MYTHREAD];
Bdata = upc_all_alloc(THREADS*THREADS, numelems*sizeof(int));
myB = (shared [] int *)&Bdata[MYTHREAD];

// Adata and Bdata contain THREADS*THREADS*numelems 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 < numelems*THREADS; i++)
    myA[i] = MYTHREAD*10 + i;
upc_barrier;
upc_all_exchange( Bdata, Adata, numelems*sizeof(int) );
upc_barrier;
```
5.3.6 The \texttt{upc\_all\_permute} function

Synopsis

1. \texttt{#include <upc.h>}
   \texttt{#include <upc\_collective.h>}
   \texttt{void upc\_all\_permute (shared void *dst,}
   
   \texttt{shared const void *src,}
   
   \texttt{shared const int *perm, size\_t blk);} 

Description

1. The \texttt{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 $\text{perm}[i]$. The number of bytes in each block is $\text{blk}$. If copying takes place between objects that overlap, the behavior is undefined.

2. $\text{perm}[0..\text{THREADS}-1]$ must contain \texttt{THREADS} distinct values: 0, 1, ..., \texttt{THREADS} – 1.

3. The \texttt{upc\_all\_permute} function treats the \texttt{src} pointer and the \texttt{dst} pointer as if each pointed to a shared memory area of $\text{blk}$ bytes on each thread and therefore had type:
   \begin{verbatim}
   shared [blk] char[blk*\text{THREADS}]
   \end{verbatim}
   The effect is equivalent to copying the block of $\text{blk}$ bytes that has affinity to thread $i$ pointed to by \texttt{src} to the block of $\text{blk}$ bytes that has affinity to thread $\text{perm}[i]$ pointed to by \texttt{dst}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{permute.png}
\caption{\texttt{upc\_all\_permute}.}
\end{figure}
Example

1. Example of upc_all_permute.

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

5.4 Computational Operations

5.4.1 The upc_all_reduce function

Synopsis

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

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 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 n elements of type TYPE and blocking factor blk, and therefore had type:
shared [blk] TYPE[n]

3. If the value of blk 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 n elements of type TYPE with an indefinite layout qualifier, and therefore had type:
   shared [] TYPE[n]

4. 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 function func to operate on the data in the src array at each
   step. Providing for a user-defined function allows operations such as subtraction
   and division that are not commutative and therefore do not give reduction answers
   that are well-defined.

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

   src[0] ⊕ src[1] ⊕ ... ⊕ src[n-1]

   where “⊕” 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 numelems 10
shared [blk] long A[numelems*THREADS];
long result;
// Initialize A. The result below is defined only on thread 0.
upc_barrier;
result = upc_all_reduceL( A, UPC_ADD, numelems*THREADS, blk, 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 n, size_t blk,
   TYPE (*func)(TYPE, TYPE));

Description

1. The function prototype above represents the 11 variations of the `upc_all_reduceT` function where `T` and `TYPE` have the following correspondences:
For example, if T is C, then TYPE must be signed char or char.

2. If the value of blk 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 n elements of type TYPE and blocking factor blk, and therefore
   had type:
   
   shared [blk] TYPE[n]

3. If the value of blk 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 n elements of type TYPE with an indefinite layout qualifier,
   and therefore had type:
   
   shared [] TYPE[n]

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

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 function `func` to operate on the data in the `src` array at each step. Providing for a user-defined function allows operations such as subtraction and division that are not commutative and therefore do not give reduction answers that are well-defined.

6. At function exit

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

for \(0 \leq i \leq n-1\) and where “\(\oplus\)” is the operator specified by the variable `op`.

<table>
<thead>
<tr>
<th>(T_0)</th>
<th>(D_0)</th>
<th>(D_1)</th>
<th>(D_2)</th>
<th>(D_0 + D_1)</th>
<th>(D_0 + D_1 + D_2)</th>
<th>(T_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_1)</td>
<td>(D_3)</td>
<td>(D_4)</td>
<td>(D_5)</td>
<td>(D_0 + D_3)</td>
<td>(D_0 + D_1 + D_3)</td>
<td>(T_1)</td>
</tr>
<tr>
<td>(T_2)</td>
<td>(D_6)</td>
<td>(D_7)</td>
<td>(D_8)</td>
<td>(D_0 + D_6)</td>
<td>(D_0 + D_1 + D_6)</td>
<td>(T_2)</td>
</tr>
<tr>
<td>(T_3)</td>
<td>(D_9)</td>
<td>(D_{10})</td>
<td>(D_{11})</td>
<td>(D_0 + D_9)</td>
<td>(D_0 + D_1 + D_9)</td>
<td>(T_3)</td>
</tr>
</tbody>
</table>

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

---

20
Example

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

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

5.4.3 The `upc_all_sort` function

Synopsis

1. #include <upc.h>
   #include <upc_collective.h>
   void upc_all_sort (shared void *A, size_t size, size_t n, size_t blk,
                    int (*func)(shared void *, shared void *));

Description

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

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

3. If the value of `blk` 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 numelems 10
shared [numelems] int A[numelems*THREADS];

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

// Initialize A.
upc_barrier;
upc_all_sort( A, sizeof(int), numelems*THREADS, numelems, lt_int);
upc_barrier;
```

References