UPC Collective Operations Specification pre V1.0

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

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

January 17, 2003
Acknowledgements

Thanks go to many in the UPC community for their interest and helpful comments, particularly Brian Wibecan, Dan Bonachea, and Bill Carlson. Terek 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 Collectives Library 5

4.1 Relocalization Operations 5

4.1.1 The `upc_all_broadcast` function 5

4.1.2 The `upc_all.scatter` function 6

4.1.3 The `upc_all.gather` function 7

4.1.4 The `upc_all.gather_all` function 8

4.1.5 The `upc_all.to_all` function 9

4.1.6 The `upc_all.permute` function 10

4.2 Computational Operations 11

4.2.1 The `upc_all.reduce` function 11

4.2.2 The `upc_all.prefix_reduce` function 13

4.2.3 The `upc_all.sort` function 15

**References** 15
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 UPC workshop held at SC2002 held in 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. All functions defined in this specification are 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. 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 Collectives Library

4.1 Relocalization Operations

4.1.1 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 blk);
```

Description

1. The upc_all_broadcast function copies a block of memory from one shared memory area to another shared memory area on each 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_broadcast function treats the src pointer as if it pointed to a shared memory space on a single thread and therefore had type:

   shared [] char[blk]

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

   shared [blk] char[blk*THREADS]

   ![Figure 1. upc_all_broadcast.](image)
4.1.2 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 memory of a shared memory area on one thread to a shared memory area on 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 space on a single thread and therefore had type:
   
   ```
   shared [] char[blk*THREADS]
   ```
   
   and it treats the dst pointer as if it pointed to a shared memory space 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 $i$.

```
\begin{figure}
\centering
\begin{tabular}{cccc}
T_0 & D_0 & D_1 & D_2 & D_3 \\
T_1 &       &       &       &       \\
T_2 &       &       &       &       \\
T_3 &       &       &       &       \\
\end{tabular}
\qquad\qquad
\begin{tabular}{cccc}
D_0 & T_0 &       &       \\
D_1 &       & T_1 &       \\
D_2 &       &       & T_2 \\
D_3 &       &       &       & T_3 \\
\end{tabular}
\caption{upc_all_scatter.}
\end{figure}
```
4.1.3 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 memory from a shared memory area on the i\textsuperscript{th} thread to the i\textsuperscript{th} block of a shared memory area on one 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_gather function treats the src pointer as if it pointed to a shared memory space of blk bytes on each thread and therefore had type:
   shared [blk] char[blk*THREADS]
   and it treats the dst pointer as if it pointed to a shared memory space with the type:
   shared [] char[blk*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\textsuperscript{th} block of blk bytes pointed to by dst.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{upc_all_gather.png}
\caption{upc_all_gather.}
\end{figure}
4.1.4 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 on the ith thread to the ith block of a shared memory area on each 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_gather_all function treats the src pointer as if it pointed to a shared memory space of blk bytes on each thread and therefore had type:
   shared [blk] char[blk*THREADS]

   and it treats the dst pointer as if it pointed to a shared memory space with the type:
   shared [blk*THREADS] char[blk*THREADS*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 ith block of blk bytes on thread i pointed to by dst.

   \begin{figure}
   \begin{center}
   \begin{tabular}{|c|c|c|c|c|}
   \hline
   T_0 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   T_1 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   T_2 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   T_3 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   \end{tabular}
   \hspace{1cm}
   \begin{tabular}{|c|c|c|c|c|}
   \hline
   T_0 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   T_1 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   T_2 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   T_3 & D_0 & D_1 & D_2 & D_3 \\
   \hline
   \end{tabular}
   \end{center}
   \caption{upc_all_gather_all.}
   \end{figure}
4.1.5 The upc_all_to_all function

Synopsis

1. #include <upc.h>
   #include <upc_collective.h>
   void upc_all_to_all (shared void *dst,
                        shared const void *src, size_t blk);

Description

1. The upc_all_to_all function copies the $i$th block of memory from one shared memory area on thread $j$ to the $j$th block of a shared memory area on thread $i$. 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_to_all function treats the src pointer and the dst pointer as if each pointed to a shared memory space of $blk \times THREADS$ bytes on each thread and therefore had type:

   shared [blk*THREADS] char[blk*THREADS*THREADS]

For each pair of threads $i$ and $j$, the effect is equivalent to copying the $i$th block of $blk$ bytes on thread $j$ pointed to by src to the $j$th block of $blk$ bytes on thread $i$ pointed to by dst.

![Figure 5. upc_all_to_all.](image)
4.1.6 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 blk);

Description

1. The upc_all_permute function copies a block of memory from a shared memory area on the \(i\)th thread to a block of a shared memory on 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 \(\text{THREADS}\) distinct values: 0, 1, ..., \(\text{THREADS} - 1\).

3. The upc_all_permute function treats the \(\text{src}\) pointer and the \(\text{dst}\) pointer as if each pointed to a shared memory space of \(\text{blk}\) bytes on each thread and therefore had type:
   shared \([\text{blk}]\) char[\(\text{blk}*\text{THREADS}\)]

   The effect is equivalent to copying the block of \(\text{blk}\) bytes on thread \(i\) pointed to by \(\text{src}\) to the block of \(\text{blk}\) bytes on thread \(\text{perm}[i]\) pointed to by \(\text{dst}\).

\[
\begin{array}{c|c|c|c}
\hline
\text{T}_0 & \text{D}_0 & 2 & \text{D}_2 \\
\text{T}_1 & \text{D}_1 & 3 & \text{T}_0 \\
\text{T}_2 & \text{D}_2 & 0 & \text{T}_1 \\
\text{T}_3 & \text{D}_3 & 1 & \text{T}_2 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\hline
\text{permutation} & \text{vector} & \text{D}_0 & \text{D}_1 \\
\hline
\text{T}_0 & \text{T}_1 & \text{D}_2 & \text{T}_2 \\
\text{T}_1 & \text{T}_2 & \text{D}_3 & \text{T}_3 \\
\text{T}_2 & \text{T}_3 & \text{T}_0 & \text{T}_1 \\
\text{T}_3 & \text{T}_0 & \text{T}_1 & \text{T}_2 \\
\hline
\end{array}
\]

\textbf{Figure 6.} upc_all_permute.
4.2 Computational Operations

4.2.1 The `upc_all_reduce` function

Synopsis

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

Description

1. The function prototype above represents the 6 variations of the `upc_all_reduceT` function where T is L, D, C, I, S, or F, and TYPE is long, double, unsigned char, int, short, or float, respectively.

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 space of type TYPE, length n, and had 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 space of type TYPE, length n, and a layout qualifier of the form [*] and therefore had type:
   shared [*] TYPE[n]

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

   UPC_ADD For integer or floating point variables, regular addition. For unsigned character variables, addition of the characters’ ASCII values.
   UPC_MULT For integer or floating point variables, regular multiplication. For unsigned character variables, multiplication of the characters’ ASCII values.
   UPC_AND Bitwise AND for integer and unsigned character variables. Results undefined for floating point numbers.
   UPC_OR Bitwise OR for integer and unsigned character variables. Results undefined for floating point numbers.
UPC\_XOR  Bitwise XOR for integer and unsigned 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 \texttt{func} to operate on the data in the \texttt{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

\[ \texttt{src[0]} \oplus \texttt{src[1]} \oplus \cdots \oplus \texttt{src[n-1]} \]

where “\(\oplus\)" is the operator specified by the variable \texttt{op}. The value returned to all other threads is undefined.

![Figure 7. upc\_all\_reduce with addition operator.](image)
4.2.2 The upc_all_prefix_reduce function

Synopsis

1. #include <upc.h>
#ifndef UPC_CC
#include <upc_collective.h>
#endif

void upc_all_prefix_reduceT(shared TYPE *dst, shared const TYPE *src
UPC_OP op, size_t n, size_t blk,
TYPE (*func)(TYPE, TYPE));

Description

1. The function prototype above represents the 6 variations of the upc_all_prefix_reduceT
function where T is L, D, C, I, S, or F, and TYPE is long, double, unsigned char, int,
short, or float, respectively.

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 as if it pointed to a shared memory
space of type TYPE, length n, and had 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 as if it pointed to a shared memory
space of type TYPE, length n, and a layout qualifier of the form [*] and therefore had
type:

shared [*] TYPE[n]

4. The pointer-to-shared dst has the same type as src and it points to a shared memory
space of at least n elements.

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

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

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

UPC_AND Bitwise AND for integer and unsigned character variables. Results undefined
for floating point numbers.
UPC\_OR Bitwise OR for integer and unsigned character variables. Results undefined for floating point numbers.

UPC\_XOR Bitwise XOR for integer and unsigned 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

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

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

\[
\begin{array}{ccc}
T_0 & D_0 & D_1 & D_2 \\
T_1 & D_3 & D_4 & D_5 \\
T_2 & D_6 & D_7 & D_8 \\
T_3 & D_9 & D_{10} & D_{11}
\end{array}
\]

\[
\begin{array}{ccc}
D_0 & D_0 + D_1 & D_0 + D_1 + D_2 \\
D_3 + \cdots + D_5 & D_3 + \cdots + D_5 & D_3 + \cdots + D_5 \\
D_6 + \cdots + D_8 & D_6 + \cdots + D_8 & D_6 + \cdots + D_8 \\
D_9 + \cdots + D_{11} & D_9 + \cdots + D_{10} & D_9 + \cdots + D_{11}
\end{array}
\]

Figure 8. upc\_all\_prefix\_reduce with addition operator. The \(D_i\)'s are scalars of type TYPE.
4.2.3 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 a layout qualifier of the form $[*]$.

4. The function $func(x, y)$ returns -1, 0, or 1 depending on whether $x > y$, $x == y$, or $x < y$, respectively.

References