UPC Required Library Specifications
Version 1.3

A publication of the UPC Consortium

November 16, 2013

Contents

7 Library

1 This section provides UPC parallel extensions of [ISO/IEC00 Sec 7.1.2]. Also see the UPC Optional Library Specifications.

2 The libraries specified in this document are required and shall be provided by all conforming implementations of the UPC language.

7.4 UPC Collective Utilities <upc_collective.h>

1 Implementations that support this interface shall predefine the feature macro `__UPC_COLLECTIVE__` to the value 1.

2 The following requirements apply to all of the functions defined in Section 7.4.

3 All of the functions are collective. [1]

4 All collective function arguments are single-valued.

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

7.4.1 Standard headers

1 The standard header is

`<upc_collective.h>`

2 Unless otherwise noted, all of the functions, types and macros specified in Section 7.4 are declared by the header `<upc_collective.h>`.

3 Every inclusion of `<upc_collective.h>` has the effect of including `<upc_types.h>`.

[1] Note that collective does not necessarily imply barrier synchronization. The synchronization behavior of the library functions is explicitly controlled by using the `upc_flag_t flags` argument. See UPC Language Specification, Section 7.3.3 for details.

7.4.2 Relocalization Operations

7.4.2.1 The `upc_all_broadcast` function

Synopsis

1
```
#include <upc_collective.h>
void upc_all_broadcast(shared void * restrict dst,
     shared const void * restrict src, size_t nbytes,
     upc_flag_t flags);
```

Description

2 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`.

3 `nbytes` must be strictly greater than 0.

4 The `upc_all_broadcast` function treats the `src` pointer as if it pointed to a shared memory area with the type:

```
shared [] char[nbytes]
```

5 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]
```

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

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

8 If copying takes place between objects that overlap, the behavior is undefined.

9 EXAMPLE 1 shows `upc_all_broadcast`

```
#include <upc_collective.h>
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;
```

EXAMPLE 2:

```
#include <upc_collective.h>
#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 );
```

EXAMPLE 3 shows `(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])`.

```
#include <upc_collective.h>
#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;
```

7.4.2.2 The `upc_all_scatter` function

Synopsis

```
#include <upc_collective.h>
void upc_all_scatter(shared void * restrict dst,
     shared const void * restrict src, size_t nbytes,
     upc_flag_t flags);
```

Description

The `upc_all_scatter` function copies the ith block of an area of shared memory with affinity to a single thread to a block of shared memory with affinity to the ith thread. The number of bytes in each block is `nbytes`.

`nbytes` must be strictly greater than 0.

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]
```

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

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

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

If copying takes place between objects that overlap, the behavior is undefined.

EXAMPLE 1 `upc_all_scatter` for the dynamic THREADS translation environment.

```
#include <upc_collective.h>
#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;
```

EXAMPLE 2 `upc_all_scatter` for the *static THREADS* translation envi-

ronment.

```
#include <upc_collective.h>
#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 );
```

7.4.2.3 The `upc_all_gather` function

Synopsis

1
```
#include <upc_collective.h>
void upc_all_gather(shared void * restrict dst,
     shared const void * restrict src, size_t nbytes,
     upc_flag_t flags);
```

Description

2 The `upc_all_gather` function copies a block of shared memory that has affinity to the ith thread to the ith block of a shared memory area that has affinity to a single thread. The number of bytes in each block is `nbytes`.

3 `nbytes` must be strictly greater than 0.

4 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]
```

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

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

6 The target of the `src` pointer must have affinity to thread 0.

7 The `src` pointer is treated as if it has phase 0.

8 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 ith block of `nbytes` bytes pointed to by `dst`.

9 If copying takes place between objects that overlap, the behavior is unde-

fined.

10 EXAMPLE 1 `upc_all_gather` for the *static THREADS* translation environment.

```
#include <upc_collective.h>
#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 );
```

11 EXAMPLE 2 `upc_all_gather` for the *dynamic THREADS* translation environment.

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

7.4.2.4 The `upc_all_gather_all` function

Synopsis

1
```
#include <upc_collective.h>
void upc_all_gather_all(shared void * restrict dst,
     shared const void * restrict src, size_t nbytes,
     upc_flag_t flags);
```

Description

2 The `upc_all_gather_all` function copies a block of memory from one shared memory area with affinity to the ith thread to the ith block of a shared memory area on each thread. The number of bytes in each block is `nbytes`.

3 `nbytes` must be strictly greater than 0.

4 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]
```

5 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]
```

6 The targets of the `src` and `dst` pointers must have affinity to thread 0.

7 The `src` and `dst` pointers are treated as if they have phase 0.

8 The effect is equivalent to copying the ith block of `nbytes` bytes pointed to by `src` to the ith block of `nbytes` bytes pointed to by `dst` that has affinity to each thread.

9 If copying takes place between objects that overlap, the behavior is undefined.

10 EXAMPLE 1 `upc_all_gather_all` for the *static THREADS* translation environment.

```
#include <upc_collective.h>
#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;
```

11 EXAMPLE 2 `upc_all_gather_all` for the *dynamic THREADS* translation environment.

```
#include <upc.h>
#include <upc_collective.h>
#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 );
```

7.4.2.5 The `upc_all_exchange` function

Synopsis

1
```
#include <upc_collective.h>
void upc_all_exchange(shared void * restrict dst,
     shared const void * restrict src, size_t nbytes,
     upc_flag_t flags);
```

Description

2 The `upc_all_exchange` function copies the ith block of memory from a shared memory area that has affinity to thread j to the jth block of a shared memory area that has affinity to thread i. The number of bytes in each block is `nbytes`.

3 `nbytes` must be strictly greater than 0.

4 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]
```

5 The targets of the `src` and `dst` pointers must have affinity to thread 0.

6 The `src` and `dst` pointers are treated as if they have phase 0.

7 For each pair of threads i and j, the effect is equivalent to copying the ith block of `nbytes` bytes that has affinity to thread j pointed to by `src` to the jth block of `nbytes` bytes that has affinity to thread i pointed to by `dst`.

8 If copying takes place between objects that overlap, the behavior is undefined.

9 EXAMPLE 1 `upc_all_exchange` for the *static THREADS* translation environment.

```
#include <upc_collective.h>
#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;
```

10 EXAMPLE 2 `upc_all_exchange` for the *dynamic THREADS* translation environment.

```
#include <upc.h>
#include <upc_collective.h>
#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 );
```

7.4.2.6 The `upc_all_permute` function

Synopsis

1
```
#include <upc_collective.h>
void upc_all_permute(shared void * restrict dst,
    shared const void * restrict src,
    shared const int * restrict perm,
    size_t nbytes, upc_flag_t flags);
```

Description

2 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`.

3 `nbytes` must be strictly greater than 0.

4 `perm[0..THREADS-1]` must contain `THREADS` distinct values: `0, 1, ..., THREADS-1`.

5 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]
```

6 The targets of the `src`, `perm`, and `dst` pointers must have affinity to thread 0.

7 The `src` and `dst` pointers are treated as if they have phase 0.

8 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`.

9 If any of the elements referenced by `dst` overlap any of the elements referenced by `src` or `perm`, the behavior is undefined.

10 EXAMPLE 1 `upc_all_permute`.

```
#include <upc_collective.h>
#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;
```

7.4.3 Computational Operations

1 Computational operations are specified using a value of type `upc_op_t`, which is specified in UPC Language Specification, Section 7.3.1. All of the operations defined in that section are supported for computational collectives.

In addition, the following `upc_op_t` value macros are defined in `<upc_collective.h>`:

`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 non-commutative function `func` to operate on the data in the `src` array at each step.

2 Bitwise operations shall not be specified for collective operations on floating-point types.

3 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.[2]

4 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, prefix reduction (7.4.3.1).

[2] 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.

7.4.3.1 The `upc_all_reduce` and `upc_all_prefix_reduce` functions

Synopsis

1

```
#include <upc_collective.h>
void upc_all_reduce<<T>>(
        shared void * restrict dst,
        shared const void * restrict src,
        upc_op_t op,
        size_t nelems,
        size_t blk_size,
        <<TYPE>>(*func)(<<TYPE>>, <<TYPE>>),
        upc_flag_t flags);
void upc_all_prefix_reduce<<T>>(
        shared void * restrict dst,
        shared const void * restrict src,
        upc_op_t op,
        size_t nelems,
        size_t blk_size,
        <<TYPE>>(*func)(<<TYPE>>, <<TYPE>>),
        upc_flag_t flags);
```

Description

2 The function prototypes above represents the 22 variations of the `upc_all_reduce`*T* and `upc_all_prefix_reduce`*T* functions where *T* and *TYPE* have the following correspondences: [3]

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		

3 On completion of the `upc_all_reduce` variants, the value of the *TYPE* shared object referenced by `dst` is `src[0]` $\oplus$ `src[1]` $\oplus \cdots \oplus$ `src[nelems-1]` where

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

"⊕" is the operator specified by the variable `op`.

4 On completion of the `upc_all_prefix_reduce` variants, the value of the *TYPE* shared object referenced by `dst[i]` is `src[0]` ⊕ `src[1]` ⊕ ··· ⊕ `src[i]` for 0 ≤ `i` ≤ `nelems-1` and where "⊕" is the operator specified by the variable `op`.

5 If a floating-point variant of either function encounters an operand with a *NaN* value (as defined in [ISO/IEC00 Sec 5.2.4.2.2]), behavior is implementation-defined.

6 If the value of `blk_size` passed to these functions is greater than 0 then they treat 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]
```

7 If the value of `blk_size` passed to these functions is 0 then they treat 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[4]:

```
shared []  TYPE[nelems]
```

8 The phase of the `src` pointer is respected when referencing array elements, as specified above.

9 `upc_all_prefix_reduceT` treats the `dst` pointer equivalently to the `src` pointer as described in the past 3 paragraphs.

10 `upc_all_prefix_reduceT` requires the affinity and phase of the `src` and `dst` pointers to match – ie. `upc_threadof(src) == upc_threadof(dst) && upc_phaseof(src) == upc_phaseof(dst)`.

11 `upc_all_reduceT` treats the `dst` pointer as having type:

```
shared TYPE *
```

12 If any of the elements referenced by `src` and `dst` overlap, the behavior is undefined.

13 EXAMPLE 1 `upc_all_reduce` of type `long UPC_ADD`.

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

[4]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`.

```
#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;
```

EXAMPLE 2 `upc_all_prefix_reduce` of type `long UPC_ADD`.

```
#include <upc_collective.h>
#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 );
```

7.5 High-Performance Wall-Clock Timers <upc_tick.h>

1 This subsection provides extensions of [ISO/IEC00 Sec 7.23]. All the characteristics of library functions described in [ISO/IEC00 Sec 7.1.4] apply to these as well. Implementations that support this interface shall predefine the feature macro `__UPC_TICK__` to the value 1.

Rationale

2 The `upc_tick_t` type and associated functions provide convenient and portable support for querying high-precision system timers for obtaining high-precision wall-clock timings of sections of code. Many hardware implementations offer access to high-performance timers with a handful of instructions, providing timer precision and overhead that can be several orders of magnitude better than can be obtained through the use of existing interfaces in [ISO/IEC00] or POSIX (e.g. the `gettimeofday()` system call).

7.5.1 Standard header

1 The standard header is

```
<upc_tick.h>
```

2 Unless otherwise noted, all of the functions, types and macros specified in Section 7.5 are declared by the header `<upc_tick.h>`.

7.5.1.1 upc_tick_t Type

1 The following type is defined in `upc_tick.h`:

```
upc_tick_t
```

2 `upc_tick_t` is an unsigned integral type representing a quantity of abstract timer ticks, whose ratio to wall-clock time is implementation-dependent and thread-dependent.

3 `upc_tick_t` values are thread-specific quantities with a thread-specific interpretation (e.g. they might represent a hardware cycle count on a particular processor, starting at some arbitrary time in the past). More specifically, `upc_tick_t` values do *not* provide a globally-synchronized timer (i.e. the simultaneous absolute tick values may differ across threads), and furthermore

the tick-to-wall-clock conversion ratio might also differ across UPC threads (e.g. on a system with heterogenerous processor clock rates, the tick values may advance at different rates for different UPC threads).

4 As a rule of thumb, `upc_tick_t` values and intervals obtained by *different* threads should never be directly compared or arithmetically combined, without first converting the relevant tick intervals to wall time intervals (using `upc_ticks_to_ns`).

7.5.1.2 `UPC_TICK_MAX` and `UPC_TICK_MIN`

1 The following macro values are defined in `upc_tick.h`:

```
UPC_TICK_MAX
UPC_TICK_MIN
```

2 `UPC_TICK_MAX` and `UPC_TICK_MIN` are constants of type `upc_tick_t`. They respectively provide the minimal and maximal values representable in a variable of type `upc_tick_t`.

7.5.2 upc_tick_t functions

7.5.2.1 The `upc_ticks_now` function

Synopsis

1
```
#include <upc_tick.h>

upc_tick_t upc_ticks_now(void);
```

Description

2 `upc_ticks_now` returns the current value of the tick timer for the calling thread, as measured from an arbitrary, thread-specific point of time in the past (which is fixed during any given program execution).

3 The function always succeeds.

7.5.2.2 The `upc_ticks_to_ns` function

Synopsis

1
```
#include <upc_tick.h>

uint64_t upc_ticks_to_ns(upc_tick_t ticks);
```

Description

2 `upc_ticks_to_ns` converts a quantity of ticks obtained by the calling thread into wall-clock nanoseconds.

3 The function always succeeds.

4 EXAMPLE 1: an example of the upc_tick interface in use:

```
#include <upc_tick.h>
#include <stdio.h>

upc_tick_t start = upc_ticks_now();
  compute_foo(); /* do something that needs to be timed */
upc_tick_t end = upc_ticks_now();

printf("Time was: %f seconds\n", upc_ticks_to_ns(end-start)/1.0E-9);
```

Index