UPC Collective Conformance Suite

Lisa Begum Charles Wallace
Michigan Technological University
{kbegum,wallace}@mtu.edu

Michigan Technological University
Computer Science Technical Report
CS-TR-06-01
January 16, 2006

UPC Collective Conformance Suite

Lisa Begum Charles Wallace
Michigan Technological University
{kbegum,wallace}@mtu.edu

January 16, 2006

1 Overview

The UPC collective conformance suite is a collection of tests that help determine how closely a given implementation of the UPC collective operations conform to the specifications. The test suite is based on Version 1.1 of the UPC specification [1] and Version 1.0 of the UPC collectives specifications [2]. It has been used on the MuPC and HP UPC platforms. These tests exercise all of the collective operations, with the exception of the deprecated `upc_all_sort` operation, and notify the tester of behavior that departs from the specifications. This document describes the tests in the suite and explains why they were chosen.

It is quite easy to construct pathological instances of UPC collectives use, which run contrary to the intended use of these operations. For such cases, the specifications state that the resulting behavior is "undefined": in other words, the specifications place no constraints on program behavior. Hence from the viewpoint of conformance, such cases are irrelevant. Of course, it may be of interest to build in robust responses to such cases, in which the UPC program fails gracefully or provides debugging information to the programmer. A suite of tests for the robustness of a UPC implementation would be useful; however, they are outside the scope of this conformance suite.

Each collective operation implies a choice for each of the following variables:

- Compilation environment: static threads, dynamic threads
- `sync_mode`: 0, `UPC_IN_`X`SYNC`, `UPC_OUT_`Y`SYNC`, `UPC_IN_`X`SYNC`| `UPC_OUT_`Y`SYNC` (X, Y in `NO,MY,ALL`)
- `nbytes`: 1. . . `MAX_BLOCK_SIZE`[1]

Each individual test is identifed by a string of the form $A.B.C.D.\ldots$, where A through D are defined below and the rest will be defined in the section of each collective operation.

A identifies the collective operation being tested:

[1]`MAX_BLOCK_SIZE` is fixed at 1024 based on the maximum block size allowed in the HP UPC implementation.

A	operation
0	`upc_all_broadcast`
1	`upc_all_scatter`
2	`upc_all_gather`
3	`upc_all_gather_all`
4	`upc_all_exchange`
5	`upc_all_permute`
6	`upc_all_reduce`
7	`upc_all_prefix_reduce`

B identifies the compilation environment:

B	compilation environment
0	static
1	dynamic

C identifies the value of `sync_mode`. The following choices cover all possible combinations of `IN` and `OUT` synchronization:

C	`sync_mode`
0	`0` (`UPC_IN_ALLSYNC`|`UPC_OUT_ALLSYNC`)
1	`UPC_IN_NOSYNC` (|`UPC_OUT_ALLSYNC`)
2	`UPC_IN_MYSYNC` (|`UPC_OUT_ALLSYNC`)
3	`UPC_OUT_NOSYNC` (|`UPC_IN_ALLSYNC`)
4	`UPC_OUT_MYSYNC` (|`UPC_IN_ALLSYNC`)
5	`UPC_IN_NOSYNC`|`UPC_OUT_NOSYNC`
6	`UPC_IN_NOSYNC`|`UPC_OUT_MYSYNC`
7	`UPC_IN_MYSYNC`|`UPC_OUT_NOSYNC`
8	`UPC_IN_MYSYNC`|`UPC_OUT_MYSYNC`

D identifies the affinity of `*src`. The boundary cases of 0 and `THREADS-1` are tested:

D	`*src` affinity
0	0
1	`THREADS`/2
2	`THREADS-1`

Compliance with the selected synchronization modes is tested through the following actions:

1. initialization of the `*src` and `*dst` blocks;
2. `upc_barrier`;
3. remote updates of the `*src` block (*i.e.,* updates by threads to which the updated memory locations do not have affinity);
4. execution of the collective operation;
5. reading of the resulting values.

The updates before the collective operation are done remotely in order to test the performance of the necessary synchronization in the `MY_SYNC` and `ALL_SYNC` cases.

Note that a race condition has been observed in the tests for `upc_all_reduce`, however, it has been irreproducible.

2 upc_all_broadcast

upc_all_broadcast is called in different test cases and checked for correctness. The function upc_all_broadcast[2] is:

```
#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
```

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.

Each test of the form $0.B.C.D.E.F$ performs a broadcast from a block specified by *src and then checks that exactly nbytes bytes of data were broadcast to the block specified by *dst. The block *dst must have affinity to thread 0. Each test involves a choice for each of the following variables:

- affinity of *src: 0 . . . THREADS-1
- *src block size: nbytes. . . MAX_BLOCK_SIZE
- phase of src within block: 0. . . MAX_BLOCK_SIZE-nbytes
- *dst block size: nbytes. . . MAX_BLOCK_SIZE

E identifies the value of nbytes. The boundary cases of 1 and MAX_BLOCK_SIZE are chosen:

E	nbytes
0	1
1	MAX_BLOCK_SIZE

F identifies the block sizes and phase of src. The boundary cases are tested through the following choices:

F	block sizes and phase of src
0	*src block size = *dst block size = nbytes; phase = 0
1	*src block size = MAX_BLOCK_SIZE, *dst block size = nbytes; phase = MAX_BLOCK_SIZE-nbytes

There are no test cases with $E = 1$ and $F = 1$ since they would be identical to the cases with $E = 1$ and $F = 0$.

[2]As described in the UPC Collective Operations Specifications [2].

3 `upc_all_scatter`

`upc_all_scatter` is called in different test cases and checked for correctness. The function `upc_all_scatter` is:

```
#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
```

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`.

Each test of the form $1.B.C.D.E.F$ performs a scatter from a block specified by `*src` and then checks that exactly `nbytes*THREADS` bytes of data were scattered to the block specified by `*dst`. The block `*dst` must have affinity to thread 0. Each test involves a choice for each of the following variables:

- affinity of `*src`: 0 . . . `THREADS-1`
- `*src` block size: `nbytes*THREADS`. . . `MAX_BLOCK_SIZE`
- *dst block size: `nbytes`. . . `MAX_BLOCK_SIZE / THREADS`
- phase of `src` within block: 0. . . `MAX_BLOCK_SIZE-(nbytes*THREADS)`

E identifies the value of `nbytes`. The boundary cases of 1 and `MAX_BLOCK_SIZE / THREADS` are chosen:

E	`nbytes`
0	1
1	`MAX_BLOCK_SIZE / THREADS`

F indicates the block sizes and phase of `src`. The boundary cases are tested through the following choices:

F	block sizes and phase
0	`*src` block size = `nbytes*THREADS`, `*dst` block size = `nbytes` ; phase = 0
1	`*src` block size = `MAX_BLOCK_SIZE*THREADS`, `*dst` block size = `nbytes`; phase = `(MAX_BLOCK_SIZE-nbytes) * THREADS`

There are no test cases with $E = 1$ and $F = 1$ since they would be identical to the cases with $E = 1$ and $F = 0$.

4 upc_all_gather

upc_all_gather is called in different test cases and checked for correctness. The function upc_all_gather is:

```
#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
```

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.

Each test of the form 2.$B.C.D.E.F$ performs a gather from a block specified by *src and then checks that exactly nbytes bytes of data were gathered to the block specified by *dst. Each test involves a choice for each of the following variables:

- affinity of *src: 0 . . . THREADS-1
- *src block size: nbytes. . . MAX_BLOCK_SIZE
- *dst block size: nbytes*THREADS. . . MAX_BLOCK_SIZE
- phase of src within block: 0. . . MAX_BLOCK_SIZE-nbytes

Since the block *src must have affinity to thread, there are no test cases for D=1,2.

E identifies the value of nbytes. The boundary cases of 1 and MAX_BLOCK_SIZE / THREADS are chosen:

E	nbytes
0	1
1	MAX_BLOCK_SIZE / THREADS

F indicates the block sizes and phase of src. The boundary cases are tested through the following choices:

F	block sizes and phase
0	*src block size = nbytes, *dst block size = nbytes * THREADS; phase = 0
1	*src block size = MAX_BLOCK_SIZE, *dst block size = nbytes * THREADS; phase = MAX_BLOCK_SIZE-nbytes

There are no test cases with $E = 1$ and $F = 1$ since they would be identical to the cases with $E = 1$ and $F = 0$.

G indicates the affinity of *dst. The boundary cases of 0 and THREADS-1 are tested:

G	*dst affinity
0	0
1	THREADS/2
2	THREADS-1

5 upc_all_gather_all

upc_all_gather_all is called in different test cases and checked for correctness. The function upc_all_gather_all is:

```
#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
```

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.

Each test of the form $3.B.C.D.E.F$ performs an all-all gather from a block specified by *src and then checks that exactly nbytes * THREADS bytes of data was all-all gathered to the block specified by *dst. The blocks *src and *dst must have affinity to thread 0. Each test involves a choice for each of the following variables:

- *src block size: nbytes. . . MAX_BLOCK_SIZE
- *dst block size: nbytes*THREADS. . . MAX_BLOCK_SIZE
- phase of src within block: 0. . . MAX_BLOCK_SIZE-nbytes

Since the block *src must have affinity to thread 0, there are no test cases for D=1,2.

E identifies the value of nbytes. The boundary cases of 1 and MAX_BLOCK_SIZE / THREADS are chosen:

E	nbytes
0	1
1	MAX_BLOCK_SIZE / THREADS

F indicates the block sizes and phase of *src. The boundary cases are tested through the following choices:

F	block sizes and phase
0	*src block size = nbytes; *dst block size = nbytes * THREADS; phase =0
1	*src block size = MAX_BLOCK_SIZE*dst block size = nbytes * THREADS; phase = MAX_BLOCK_SIZE-nbytes

There are no test cases with $E = 1$ and $F = 1$ since they would be identical to the cases with $E = 1$ and $F = 0$.

6 upc_all_exchange

upc_all_exchange is called in different test cases and checked for correctness. The function upc_all_exchange is:

```
#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
```

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.

Each test of the form $4.B.C.D.E.F$ performs an exchange from a block specified by *src and then checks that exactly nbytes * THREADS * THREADS bytes of data were exchanged to the block specified by *dst. The blocks *src and *dst must have affinity to thread 0. Each test involves a choice for each of the following variables:

- *src block size: nbytes*THREADS...MAX_BLOCK_SIZE
- *dst block size: nbytes*THREADS...MAX_BLOCK_SIZE
- phase of src within block: 0...MAX_BLOCK_SIZE-nbytes

Since the block *src must have affinity to thread 0, there are no test cases for D=1,2.

E identifies the value of nbytes. The boundary cases of 1 and MAX_BLOCK_SIZE / THREADS are chosen:

E	nbytes
0	1
1	MAX_BLOCK_SIZE / THREADS

F indicates the block sizes of *src. The boundary cases are tested through the following choices:

F	block sizes and phase
0	*src block size = *dst block size = nbytes * THREADS; phase =0
1	*src block size = MAX_BLOCK_SIZE; *dst block size = nbytes * THREADS; phase = MAX_BLOCK_SIZE-nbytes

There are no test cases with $E = 1$ and $F = 1$ since they would be identical to the cases with $E = 1$ and $F = 0$.

7 upc_all_permute

upc_all_permute is called in different test cases and checked for correctness. The function upc_all_permute is:

```
#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
```

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.

Each test of the form $5.B.C.D.E.F$ performs a permute from a block specified by *src and then checks that exactly nbytes bytes of data were permuted to the block specified by *dst. The blocks *src, *dst, and *perm must have affinity to thread 0. Each test involves a choice for each of the following variables:

- *src block size: nbytes. . . MAX_BLOCK_SIZE
- *dst block size: nbytes. . . MAX_BLOCK_SIZE
- phase of src within block: 0. . . MAX_BLOCK_SIZE-nbytes

Since the block *src must have affinity to thread 0, there are no test cases for D=1,2.

E identifies the value of nbytes. The boundary cases of 1 and MAX_BLOCK_SIZE are chosen:

E	nbytes
0	1
1	MAX_BLOCK_SIZE

F indicates the block sizes. The boundary cases are tested through the following choices:

F	block sizes
0	*src block size = *dst block size = nbytes
1	*src block size = MAX_BLOCK_SIZE, *dst block size = nbytes

There are no test cases with $E = 1$ and $F = 1$ since they would be identical to the cases with $E = 1$ and $F = 0$.

Since the block *dst must have affinity to thread 0, there are no test cases for G=1,2.

H indicates the contents of the perm array.

H	contents of perm
0	0, 1, . . .
1	THREADS-1, THREADS-2, . . .
2	0, THREADS-1, 1, THREADS-2, . . .

8 upc_all_reduce

`upc_all_reduce` is called in different test cases and checked for correctness. The function `upc_all_reduce` is:

```
#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,
                     upc_flag_t sync_mode);
nelems: the number of elements
blk_size: the number of elements in a block
```

Each test of the form $6.B.C.D.E.F.\ldots L$ performs a reduce from a block specified by `*src` and then checks that exactly `nelems` elements of the specified data type were reduced to the block specified by `*dst`.

Each test involves a choice for each of the following variables:

- affinity of `*src`: 0. . . `THREADS-1`
- `blk_size`: 0. . . `MAX_BLOCK_SIZE`
- phase of `src` within block: 0. . . `MAX_BLOCK_SIZE-1`
- `op`: `UPC_ADD`, `UPC_MULT`, `UPC_AND`, `UPC_OR`, `UPC_XOR`, `UPC_LOGAND`, `UPC_LOGOR`, `UPC_MIN`, `UPC_MAX`, `UPC_FUNC`, `UPC_NONCOMM_FUNC`
- `nelems`: 1 . . . `THREADS * MAX_BLOCK_SIZE`

I indicates the type of the elements involved. All possible types are tested through the following choices:

I	Type of elements involved
0	`C (signed char)`
1	`UC (unsigned char)`
2	`S (signed short)`
3	`US (unsigned short)`
4	`I (signed int)`
5	`UI (unsigned int)`
6	`L (signed long)`
7	`UL (unsigned long)`
8	`F (float)`
9	`D (signed double)`
10	`UD (unsigned double)`

J indicates the value of `op`. All possible values are tested through the following choices:

J	op
0	UPC_ADD
1	UPC_MULT
2	UPC_AND
3	UPC_OR
4	UPC_XOR
5	UPC_LOGAND
6	UPC_LOGOR
7	UPC_MIN
8	UPC_MAX
9	UPC_FUNC
10	UPC_NONCOMM_FUNC

There are no test cases with $I \in \{8, 9, 10\}$ and $J \in \{2, 3, 4\}$ since results are undefined for bitwise operations on floating point values. The test case with $J = 9$ is performed with the function $f(x, y) = (x^2 + y^2)$. There is no test case for $J = 10$ since the behavior is undefined.

K indicates the value of `blk_size` and the phase of `src`. The boundary cases are tested through the following choices:

K	`blk_size` and phase of `src`
0	`blk_size` = 0; phase = 0
1	`blk_size` = 1; phase = 0
2	`blk_size` = `MAX_BLOCK_SIZE`; phase = `MAX_BLOCK_SIZE-1`

L indicates the number of elements involved. The boundary cases of 1 and `THREADS * MAX_BLOCK_SIZE` are tested:

L	nelems
0	1
1	THREADS * MAX_BLOCK_SIZE

There is no test case for $K = 1, L = 1$ since the number of elements in `*src` is less than `MAX_BLOCK_SIZE`. There is no test case for $K = 2, L = 1$ since it imposes that the `blk_size` of the source array to be bigger than `MAX_BLOCK_SIZE`.

9 upc_all_prefix_reduce

upc_all_prefix_reduce is called in different test cases and checked for correctness. The function upc_all_prefix_reduce is:

```
#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,
                            upc_flag_t sync_mode);
nelems: the number of elements
blk_size: the number of elements in a block
```

Each test of the form $7.B.C.D.E.F \ldots L$ performs a prefix reduce from a block specified by `*src` and then checks that exactly `nelems` elements of the specified data type were prefix-reduced to the block specified by `*dst`.

Each test involves a choice for each of the following variables:

- affinity of `*src`: 0...`THREADS-1`
- `blk_size`: 0...`MAX_BLOCK_SIZE`
- phase of `src` within block: 0...`MAX_BLOCK_SIZE-1`
- op: `UPC_ADD`, `UPC_MULT`, `UPC_AND`, `UPC_OR`, `UPC_XOR`, `UPC_LOGAND`, `UPC_LOGOR`, `UPC_MIN`, `UPC_MAX`, `UPC_FUNC`, `UPC_NONCOMM_FUNC`
- `nelems`: 1 ...`THREADS * MAX_BLOCK_SIZE`

I indicates the type of the elements involved. All possible types are tested through the following choices:

I	Type of elements involved
0	`C (signed char)`
1	`UC (unsigned char)`
2	`S (signed short)`
3	`US (unsigned short)`
4	`I (signed int)`
5	`UI (unsigned int)`
6	`L (signed long)`
7	`UL (unsigned long)`
8	`F (float)`
9	`D (signed double)`
10	`UD (unsigned double)`

J indicates the value of `op`. All possible values are tested through the following choices:

J	op
0	UPC_ADD
1	UPC_MULT
2	UPC_AND
3	UPC_OR
4	UPC_XOR
5	UPC_LOGAND
6	UPC_LOGOR
7	UPC_MIN
8	UPC_MAX
9	UPC_FUNC
10	UPC_NONCOMM_FUNC

There are no test cases with $I \in \{8, 9, 10\}$ and $J \in \{2, 3, 4\}$ since results are undefined for bitwise operations on floating point values. The test case with $J = 9$ is performed with the function $f(x, y) = (x^2 + y^2)$. There is no test case for $J = 10$ since the behavior is undefined.

K indicates the value of `blk_size` and the phase of `src`. The boundary cases are tested through the following choices:

K	`blk_size` and phase of `src`
0	`blk_size` = 0; phase = 0
1	`blk_size` = 1; phase = 0
2	`blk_size` = `MAX_BLOCK_SIZE`; phase = `MAX_BLOCK_SIZE-1`

L indicates the number of elements involved. The boundary cases of 1 and `THREADS * MAX_BLOCK_SIZE` are tested:

L	nelems
0	1
1	THREADS * MAX_BLOCK_SIZE

There is no test case for $K = 1, L = 1$ since the number of elements in `*src` is less than `MAX_BLOCK_SIZE`. There is no test case for $K = 2, L = 1$ since it requires the `blk_size` of the source array to be bigger than `MAX_BLOCK_SIZE`.

References

[1] T. A. El-Ghazawi, W. Carlson and J. Draper, UPC Language Specification V1.1.1, Technical Report, George Washington University and IDA Center for Computing Sciences, October 7, 2003, <http://www.gwu.edu/~upc/docs/upc_spec_1.1.1.pdf>, March 16, 2005.

[2] E. Wiebel, D. Greenberg, and S. Seidel, UPC Collective Operations Specifications V1.0, Technical Report, George Washington University and IDA Center for Computing Sciences, December 12, 2003, <http://www.gwu.edu/~upc/docs/UPC_Coll_Spec_V1.0.pdf>, March 16, 2005.