Errata proposal - GASPI_ERROR_END_OF_QUEUE

Christian Simmendinger

December 14, 2016

1 GASPI_QUEUE_FULL

In order to resolve problems in portability and in order to provide improved handling of race conditions, we suggest that the queue related function calls gaspi_notify, gaspi_write, gaspi_write_notify, gaspi_write_list_notify, gaspi_read, gaspi_read_notify and gaspi_read_list_notify return a predefined return value when reaching the end of the respective queue, namely GASPI_QUEUE_FULL.

2 Needed Resources

- none.

3 Additional (necessary) Changes to the Standard

3.1 Return values

GASPI procedures have four general return values:

GASPI_SUCCESS implies that the procedure has completed successfully.

GASPI_TIMEOUT implies that the procedure could not complete in the given period of time. This does not necessitate an error. The procedure has to be invoked subsequently in order to fully complete the operation.

GASPI_ERROR implies that the procedure has terminated due to an error. There are no predefined error values specifying the detailed cause of an error. The function gaspi_error_message translates the error code into a human readable format.

GASPI_QUEUE_FULL implies that one of the function calls gaspi_notify, gaspi_write, gaspi_write_notify, gaspi_write_list_notify, gaspi_read, gaspi_read_notify and gaspi_read_list_notify has reached the end of the

queue and that the corresponding communication request could not be issued. If GASPI_QUEUE_FULL is returned, users should either switch to another queue or wait (see gaspi_wait) and subsequently re-issue the communication request.

3.2 Constants

```
GASPI_QUEUE_FULL
```

GASPI_QUEUE_FULL is returned if the end of the used queue has been reached. ⌟

3.3 Boilerplate

If the queue to which the communication request has been posted is full, i. e. if the number of posted communication requests has already reached the queue size of a given queue, the communication request has not been issued and the procedure returns with return value GASPI_QUEUE_FULL. In this case users should either switch to another queue or wait (see gaspi_wait) and subsequently re-issue the communication request.

3.4 BoilerplateWriteList

If the queue to which the communication request has been posted is full, i. e. if the number of posted communication requests has already reached the queue size of a given queue, the communication request has not been issued and the procedure returns with return value GASPI_QUEUE_FULL. In this case users should either switch to another queue or wait (see gaspi_wait) and subsequently re-issue the communication request.

The user should be aware that a subsequent gaspi_notify only guarantees non-overtaking conditions for the same queue to which previous gaspi_write calls have been posted.

3.5 Functions

```
GASPI_WRITE ( segment_id_local
            , offset_local
            , rank
            , segment_id_remote
            , offset_remote
            , size
            , queue
            , timeout )
```

Parameter:
(in) segment_id_local: the local segment ID to read from
(in) offset_local: the local offset in bytes to read from

(in) rank: the remote rank to write to
(in) segment_id_remote: the remote segment to write to
(in) offset_remote: the remote offset to write to
(in) size: the size of the data to write
(in) queue: the queue to use
(in) timeout: the timeout

```
gaspi_return_t
gaspi_write ( gaspi_segment_id_t segment_id_local
            , gaspi_offset_t offset_local
            , gaspi_rank_t rank
            , gaspi_segment_id_t segment_id_remote
            , gaspi_offset_t offset_remote
            , gaspi_size_t size
            , gaspi_queue_id_t queue
            , gaspi_timeout_t timeout )
```

```
function gaspi_write(segment_id_local,offset_local,&
&         rank, segment_id_remote,offset_remote,size,&
&         queue,timeout_ms) &
&         result( res ) bind(C, name="gaspi_write")
  integer(gaspi_segment_id_t), value :: segment_id_local
  integer(gaspi_offset_t), value :: offset_local
  integer(gaspi_rank_t), value :: rank
  integer(gaspi_segment_id_t), value :: segment_id_remote
  integer(gaspi_offset_t), value :: offset_remote
  integer(gaspi_size_t), value :: size
  integer(gaspi_queue_id_t), value :: queue
  integer(gaspi_timeout_t), value :: timeout_ms
  integer(gaspi_return_t) :: res
end function gaspi_write
```

Execution phase:
Working

Return values:
`GASPI_SUCCESS`: operation has returned successfully
`GASPI_TIMEOUT`: operation has run into a timeout
`GASPI_ERROR`: operation has finished with an error
`GASPI_QUEUE_FULL`: operation could not be posted due to a full queue ⌟

The listings for alltoall_write and alltoall_read should change accordingly:

Listing 1: wait_if_queue_full.h

```
1 #ifndef _WAIT_IF_QUEUE_FULL_H
2 #define _WAIT_IF_QUEUE_FULL_H 1
```

```
3
4  #include <GASPI.h>
5
6  #define WAIT_IF_QUEUE_FULL(f, queue)                    \
7    {                                                     \
8      gaspi_return_t ret;                                 \
9      while ((ret = (f)) == GASPI_QUEUE_FULL)             \
10        {                                                \
11           ASSERT (gaspi_wait ((queue), GASPI_BLOCK));   \
12        }                                                \
13     ASSERT (ret == GASPI_SUCCESS);                      \
14   }
15 #endif
```

Listing 2: GASPI all-to-all communication (matrix transpose) implemented with `gaspi_write`

```
1  #include <stdlib.h>
2  #include <GASPI.h>
3  #include <success_or_die.h>
4  #include <wait_if_queue_full.h>
5
6  extern void dump (int *arr, int nProc);
7
8  int
9  main (int argc, char *argv[])
10 {
11   ASSERT (gaspi_proc_init (GASPI_BLOCK));
12
13   gaspi_rank_t iProc;
14   gaspi_rank_t nProc;
15
16   ASSERT (gaspi_proc_rank (&iProc));
17   ASSERT (gaspi_proc_num (&nProc));
18
19   gaspi_notification_id_t notification_max;
20   ASSERT (gaspi_notification_num(&notification_max));
21
22   if (notification_max < (gaspi_notification_id_t)nProc)
23     {
24       exit (EXIT_FAILURE);
25     }
26
27   ASSERT (gaspi_group_commit (GASPI_GROUP_ALL, GASPI_BLOCK));
28
29   const gaspi_segment_id_t segment_id_src = 0;
```

```
30    const gaspi_segment_id_t segment_id_dst = 1;
31
32    const gaspi_size_t segment_size = nProc * sizeof(int);
33
34    ASSERT (gaspi_segment_create ( segment_id_src, segment_size
35                                 , GASPI_GROUP_ALL, GASPI_BLOCK
36                                 , GASPI_ALLOC_DEFAULT
37                                 )
38          );
39    ASSERT (gaspi_segment_create ( segment_id_dst, segment_size
40                                 , GASPI_GROUP_ALL, GASPI_BLOCK
41                                 , GASPI_ALLOC_DEFAULT
42                                 )
43          );
44
45    int *src = NULL;
46    int *dst = NULL;
47
48    ASSERT (gaspi_segment_ptr (segment_id_src, &src));
49    ASSERT (gaspi_segment_ptr (segment_id_dst, &dst));
50
51    const gaspi_queue_id_t queue_id = 0;
52
53  #pragma omp parallel
54    for (gaspi_rank_t rank = 0; rank < nProc; ++rank)
55      {
56        src[rank] = iProc * nProc + rank;
57
58        const gaspi_offset_t offset_src = rank * sizeof (int);
59        const gaspi_offset_t offset_dst = iProc * sizeof (int);
60        const gaspi_notification_id_t notify_ID = rank;
61        const gaspi_notification_t notify_val = 1;
62
63        WAIT_IF_QUEUE_FULL
64          (gaspi_write( segment_id_src, offset_src
65                      , rank, segment_id_dst, offset_dst
66                      , sizeof (int), notify_ID, notify_val
67                      , queue_id, GASPI_BLOCK
68                      )
69           , queue_id
70           );
71      }
72
73    gaspi_notification_id_t notify_cnt = nProc;
74    gaspi_notification_id_t first_notify_id;
75
```

```
76    while (notify_cnt > 0)
77    {
78      ASSERT (gaspi_notify_waitsome ( segment_id_dst, 0, nProc,
79                                      , &first_notify_id, GASPI_BLOCK));
80
81      gaspi_notification_id_t notify_val = 0;
82
83      ASSERT (gaspi_notify_reset (segment_id_dst, first_notify_id
84                                  , &notify_val));
85
86      if (notify_val != 0)
87         {
88           --notify_cnt;
89         }
90    }
91
92    dump (dst, nProc);
93
94    ASSERT (gaspi_wait (queue_id, GASPI_BLOCK));
95
96    ASSERT (gaspi_barrier (GASPI_GROUP_ALL, GASPI_BLOCK));
97
98    ASSERT (gaspi_proc_term (GASPI_BLOCK));
99
100   return EXIT_SUCCESS;
101 }
```

Listing 3: GASPI all-to-all communication (matrix transpose) implemented with gaspi_read

```
1  #include <stdlib.h>
2  #include <GASPI.h>
3  #include <success_or_die.h>
4  #include <wait_if_queue_full.h>
5
6  extern void dump (int *arr, int nProc);
7
8  int
9  main (int argc, char *argv[])
10 {
11   ASSERT (gaspi_proc_init (GASPI_BLOCK));
12
13   gaspi_rank_t iProc;
14   gaspi_rank_t nProc;
15
16   ASSERT (gaspi_proc_rank (&iProc));
```

```
17  ASSERT (gaspi_proc_num (&nProc));
18
19  ASSERT (gaspi_group_commit (GASPI_GROUP_ALL, GASPI_BLOCK));
20
21  const gaspi_segment_id_t segment_id_src = 0;
22  const gaspi_segment_id_t segment_id_dst = 1;
23
24  const gaspi_size_t segment_size = nProc * sizeof(int);
25
26  ASSERT (gaspi_segment_create ( segment_id_src, segment_size
27                               , GASPI_GROUP_ALL, GASPI_BLOCK
28                               , GASPI_ALLOC_DEFAULT
29                               )
30        );
31  ASSERT (gaspi_segment_create ( segment_id_dst, segment_size
32                               , GASPI_GROUP_ALL, GASPI_BLOCK
33                               , GASPI_ALLOC_DEFAULT
34                               )
35        );
36
37  int *src = NULL;
38  int *dst = NULL;
39
40  ASSERT (gaspi_segment_ptr (segment_id_src, &src));
41  ASSERT (gaspi_segment_ptr (segment_id_dst, &dst));
42
43  const gaspi_queue_id_t queue_id = 0;
44
45  for (gaspi_rank_t rank = 0; rank < nProc; ++rank)
46    {
47      src[rank] = iProc * nProc + rank;
48    }
49
50  ASSERT (gaspi_barrier (GASPI_GROUP_ALL, GASPI_BLOCK));
51
52  for (gaspi_rank_t rank = 0; rank < nProc; ++rank)
53    {
54      const gaspi_offset_t offset_src = iProc * sizeof (int);
55      const gaspi_offset_t offset_dst = rank * sizeof (int);
56
57      WAIT_IF_QUEUE_FULL
58        (gaspi_read ( segment_id_dst, offset_dst
59                    , rank, segment_id_src, offset_src
60                    , sizeof (int), queue_id, GASPI_BLOCK
61                    )
62         , queue_id
```

```
63            );
64      }
65
66    ASSERT (gaspi_wait (queue_id, GASPI_BLOCK));
67
68    dump (dst, nProc);
69
70    ASSERT (gaspi_barrier (GASPI_GROUP_ALL, GASPI_BLOCK));
71
72    ASSERT (gaspi_proc_term (GASPI_BLOCK));
73
74    return EXIT_SUCCESS;
75  }
```