GASPI Proposal: Memory provided by applications

Rui Machado
CC-HPC
Fraunhofer ITWM
Kaiserslautern, Germany

Mirko Rahn
CC-HPC
Fraunhofer ITWM
Kaiserslautern, Germany

Daniel Grünewald
CC-HPC
Fraunhofer ITWM
Kaiserslautern, Germany

Valeria Bartsch
CC-HPC
Fraunhofer ITWM
Kaiserslautern, Germany

July 1, 2015

Abstract

This proposal targets at version 1.01 from November 14th, 2013 of the GASPI specification. It proposes to extend the interface by two functions, to add a new type and to clarify at two places.

The proposed extensions allow applications to provide memory to the GASPI environment and use it for further communication.

1 Motivation and Use-case(s)

We have identified a missing feature: The possibility for the user to provide an already existing memory buffer as the memory space of a GASPI segment. Currently the creation of segments requires the user to provide an identifier and size while the GASPI implementation allocates that space. This proposal would allow the user to provide an already allocated buffer and register it as a GASPI segment to be globally accessible for communication. Such

functionality not only is generally more flexible but it also allows an even better support of different kinds of memory (e. g. NVRAM and accellerators).

Also there is a concrete use case the would get improved performance: Data exchange between two GASPI applications A and B on a common subset of hosts: At the moment the segments of A and B can not overlap (as they are created by the independent GASPI runtime's of A and B), so at least one copy of the data is required, using for example shared memory. That copy (or copies) can be eliminated.

2 Proposed interface

In section 4.1 add:

BEGIN ADDITION

`gaspi_memory_description_t`

The GASPI *memory description type used to describe properties of user provided memory.* ⌟

> *Implementor advice:* The intention of `gaspi_memory_description_t` is to describe properties of memory that is provided by the application, e.g. `MEMORY_GPU` or `MEMORY_HOST` might be relevant to an implementation. ⌟

END ADDITION

In section 7.2 add the two functions `gaspi_segment_bind` and `gaspi_segment_use`:

BEGIN ADDITION

[section number] `gaspi_segment_bind`

The *synchronous local blocking* procedure `gaspi_segment_bind` binds a segment id to user provided memory.

```
GASPI_SEGMENT_BIND ( segment_id
                   , memory_description
                   , pointer
                   , size
                   )
```

Parameter:
(in) segment_id: Unique segment ID to bind.
(in) memory_description: The description of the memory provided.
(in) pointer: The begin of the memory provided by the user.
(in) size: The size of the memory provided by *pointer* in bytes.

```
gaspi_return_t
gaspi_segment_bind
  ( gaspi_segment_id_t const segment_id
  , gaspi_memory_description_t const memory_description
  , gaspi_pointer_t const pointer
  , gaspi_size_t const size
  )
```

```
TODO: FORTRAN INTERFACE
```

Execution phase:
Working

Return values:
`GASPI_SUCCESS`: operation has returned successfully
`GASPI_TIMEOUT`: operation has run into timeout
`GASPI_ERROR`: operation has finished with an error ⌟

`gaspi_segment_bind` binds the segment identified by the identifier *segment_id* to the user provided memory of size *size* located at the address *pointer*. To provide less than *size* bytes results in undefined behavior. The identifier *segment_id* must be unique in the local GASPI process. Bind to a segment with an existing segment ID (regardless of bind or allocated) results in undefined behavior. Note that the total number of segments is restricted by the underlying hardware capabilities. The maximum number of supported segments can be retrieved by invoking `gaspi_segment_max`.

To bind successfully the user provided memory must satisfy implementation specific constraints, e. g. alignment constraints.

After successful procedure completion, i. e. return value `GASPI_SUCCESS`, the segment can be accessed locally and has the same capabilities like a segment that was allocated by a successful call to `gaspi_segment_alloc`.

If the procedure returns with `GASPI_ERROR`, the bind has failed and the segment can not be used.

> *User advice:* A GASPI implementation may allocate additional memory for internal management. Depending on the implementation it might be required that the management memory must reside on the same device as the provided memory. ⌟

END ADDITION

BEGIN ADDITION

[**section number**] `gaspi_segment_use`

The *synchronous collective time-based blocking* procedure `gaspi_segment_use` is semantically equivalent to a collective aggregation of `gaspi_segment_bind`, `gaspi_segment_register` and `gaspi_gaspi_barrier` involving all members of a given group. If the communication infrastructure was not established for all group members beforehand, `gaspi_segment_use` will accomplish this as well.

```
GASPI_SEGMENT_USE ( segment_id
                  , memory_description
                  , pointer
                  , size
                  , group
                  , timeout
                  )
```

Parameter:
(in) segment_id: Unique segment ID to bind.

(in) memory_description: The description of the memory provided.
(in) pointer: The begin of the memory provided by the user.
(in) size: The size of the memory provided by *pointer* in bytes.
(in) group: The group which should create the segment.
(in) timeout: The timeout for the operation.

```
gaspi_return_t
gaspi_segment_use
  ( gaspi_segment_id_t const segment_id
  , gaspi_memory_description_t const memory_description
  , gaspi_pointer_t const pointer
  , gaspi_size_t const size
  , gaspi_group_t const group
  , gaspi_timeout_t const timeout
  )
```

```
TODO: FORTRAN INTERFACE
```

Execution phase:
Working

Return values:
`GASPI_SUCCESS`: operation has returned successfully
`GASPI_TIMEOUT`: operation has run into timeout
`GASPI_ERROR`: operation has finished with an error ⌟

`gaspi_segment_use` can be formulated in pseudo code as

```
GASPI_SEGMENT_USE (id, memory, pointer, size, group, timeout)
{
  GASPI_SEGMENT_BIND (id, memory, pointer, size);

  foreach (rank : group)
  {
    timeout -= GASPI_CONNECT (id, rank, timeout);
    timeout -= GASPI_SEGMENT_REGISTER (id, rank, timeout);
  }

  GASPI_BARRIER (group, timeout);
}
```

where the call gets executed on all members of *group*.

END ADDITION

In section 7.2.1 change the user advice to

BEGIN CHANGE

User advice: A GASPI implementation may allocate additional memory for internal management. Depending on the implementation it might be required that the management memory must reside on the same device as the allocated memory. ⌟

END CHANGE

In section 7.3.1 change the first sentence of the description to:

BEGIN CHANGE

User advice: `gaspi_segment_delete` releases the resources that were acquired by GASPI of the segment referenced by the *segment_id* identifier. ⌟

END CHANGE

3 Influence on Implementation

- additional indirection:

  Introduces additional indirection to find area with meta-data.

  Reason: It is impossible to put the memory required for management (e.g. for notifications) directly before or after the memory any longer. Therefore that memory must be allocated somewhere else and later on its address looked up.

  Costs: Low, the indirection is a table from segment_id → pointer of management area. Code that now says something like

```
manage_at (pointer[segment_id] - MANAGEMENT_MEMORY_SIZE)
```

would change to

```
manage_at (management[segment_id])
```

- additional error codes:

  User provides memory that does not fit the restrictions. Preferable one error code for each restriction, e.g. `MEMORY_NOT_PROPERLY_ALIGNED`, `MEMORY_SIZE_NOT_A_MULTIPLE_OF_PAGESIZE`, ...

4 Influence on Applications

4.1 Influence on Existing Applications

No influence.

Existing applications that want to use the new library must be re-linked but not re-compiled.

4.2 Influence on Future Applications

The new function will allow future applications to communicate data from memory that is not allocated by the GASPI runtime system but provided to it. Chains of GASPI applications can work on the same data without copying it.

5 Influence on Performance

We do not foresee any major impact on performance, neither from an implementation point of view nor from the application point of view.

(In high pressure notification situations it might be possible that the additional indirection kicks in. On the other hand, in such situations the lookup table is kept in processor cache and after all the process still wait for the completion of a remote operation.)

6 Influence on Current Specification

Clarifications in user advices in 7.2.1 and 7.3.1.