Callback-based Completion Notification in MPI

Joachim Protze (protze@itc.rwth-aachen.de) &
Joseph Schuchart (schuchart@[hlrs.de|icl.utk.edu])

MPI HACC-WG, January 21, 2021

Asynchronous Programming Models and MPI

Asynchronous Programming Models

Dispatching work to a scheduler for eventual execution: OmpSs, OpenMP tasks, TBB, CUDA streams

MPI

- ≈ dependencies not exposed to underlying scheduler
- `MPI_THREAD_MULTIPLE` and nonblocking operations
- Requires active polling/waiting for completion

```
#pragma omp task depend(in: sendbuf)
{
  MPI_Request req;
  MPI_Isend(sendbuf, myrank, ..., &req);
  while(!complete(req))
    #pragma omp taskyield
}

#pragma omp task depend(out: recvbuf)
{
  MPI_Recv(recvbuf, myrank, ...);
}
```

What do users of APMs need from MPI?

- Potentially large number of concurrently active messages
- Eventual completion notification (of subsets) of operations
- Avoiding application-layer request management
- Thread-safety
- Portability

What do users of APMs need from MPI?

- Potentially large number of concurrently active messages
- Eventual completion notification (of subsets) of operations
- Avoiding application-layer request management
- Thread-safety
- Portability

Previous (specialized) solutions

- TAMPI [OmpSs-2]
- ULT-integration in MPI [not portable]
- Using (abusing?) `MPI_T` interface [request-tracking]

Usage Example: OpenMP Detached Tasks

```
omp_event_handle_t event;

/* task to receive data */
#pragma omp task depend(out: recvbuf) detach(event)
{
  int flag;
  MPI_Request request;
  MPI_Irecv(recvbuf, ..., &request);
  MPI_CallbackOnCompletion(
                request,
                &omp_fulfill_event, /* callback to invoke */
                event               /* argument to pass */);
}

/* task to process received data */
#pragma omp task depend(in: recvbuf)
{
  process_received_data(recvbuf);
}

/* wait for all tasks to complete */
#pragma omp taskwait
```

Note: concrete MPI interface will be discussed below

2 Independent Proposals @EuroMPI'20

MPI Detach

J. Protze, M.-A.Hermanns, A. Demiralp, M. S. Müller, and T. Kuhlen. 2020. *MPI Detach - Asynchronous Local Completion*. DOI: `https://doi.org/10.1145/3416315.3416323`

MPI Continuations

J. Schuchart, C. Niethammer, and J. Garcia. 2020. *Fibers are not (P)Threads: The Case for Loose Coupling of Asynchronous Programming Models and MPI Through Continuations*. DOI: `https://doi.org/10.1145/3416315.3416320`

MPI Detach

- *Detaching* an MPI Request:
  - Callback will be invoked once request(s) found complete

```
typedef void MPIX_Detach_function(void *data);
typedef void MPIX_Detach_status_function(
  void *data, MPI_Status status);
typedef void MPIX_Detach_all_statuses_function(
  void *data, int count, MPI_Status *statuses);

int MPIX_Detach(
  MPI_Request *request,
  MPIX_Detach_function *callback, // callback to execute
  void *data);                    // data to pass

int MPIX_Detach_status(
  MPI_Request *request,
  MPIX_Detach_status_function *callback,
  void *data);
```

```
int MPIX_Detach_all(
  int count,
  MPI_Request array_of_requests[],// array of requests
  MPIX_Detach_function *callback, // callback to execute
  void *data);                    // data to pass
```

```
int MPIX_Progress(void*); // Explicit progress function
```

MPI Detach

- *Detaching* an MPI Request:
  - Callback will be invoked once request(s) found complete
- Status objects:
  - Allocated and provided by API
  - Passed to callback if `*_status` functions are used

```
typedef void MPIX_Detach_function(void *data);
typedef void MPIX_Detach_status_function(
  void *data, MPI_Status status);
typedef void MPIX_Detach_all_statuses_function(
  void *data, int count, MPI_Status *statuses);

int MPIX_Detach(
  MPI_Request *request,
  MPIX_Detach_function *callback, // callback to execute
  void *data);                    // data to pass

int MPIX_Detach_status(
  MPI_Request *request,
  MPIX_Detach_status_function *callback,
  void *data);
```

```
int MPIX_Detach_all(
  int count,
  MPI_Request array_of_requests[],// array of requests
  MPIX_Detach_function *callback, // callback to execute
  void *data);                    // data to pass
```

```
int MPIX_Progress(void*); // Explicit progress function
```

MPI Detach

- *Detaching* an MPI Request:
  - Callback will be invoked once request(s) found complete
- Status objects:
  - Allocated and provided by API
  - Passed to callback if `*_status` functions are used
- Progress: `MPIX_Progress` or internal progress thread

```
typedef void MPIX_Detach_function(void *data);
typedef void MPIX_Detach_status_function(
  void *data, MPI_Status status);
typedef void MPIX_Detach_all_statuses_function(
  void *data, int count, MPI_Status *statuses);

int MPIX_Detach(
  MPI_Request *request,
  MPIX_Detach_function *callback, // callback to execute
  void *data);                    // data to pass

int MPIX_Detach_status(
  MPI_Request *request,
  MPIX_Detach_status_function *callback,
  void *data);
```

```
int MPIX_Detach_all(
  int count,
  MPI_Request array_of_requests[],// array of requests
  MPIX_Detach_function *callback, // callback to execute
  void *data);                    // data to pass
```

```
int MPIX_Progress(void*); // Explicit progress function
```

MPI Continuations

- `MPIX_Continue[all]`:
  - Return immediately
  - Take ownership of non-persistent requests
  - May signal immediate completion (`flag`)
  - Never invoke any callbacks!

```
typedef void (MPIX_Continue_cb_function)(
  MPI_Status * statuses , void* cb_data);

int MPIX_Continue(
  MPI_Request* op_request,
  int * flag,          // true if complete immediately
  MPIX_Continue_cb_function *cb, // callback to invoke
  void * cb_data,          // data to pass
  MPI_Status* status,      // array of statuses
  MPI_Request cont_req     // Continuation Request
);
```

```
int MPIX_Continueall (
  int count,
  MPI_Request op_requests [],
  int* flag,          // true if complete immediately
  MPIX_Continue_cb_function *cb, // callback to invoke
  void* cb_data,           // data to pass
  MPI_Status statuses [], // array of statuses
  MPI_Request cont_req     // Continuation Request
);
```

```
int MPIX_Continue_init(
  MPI_Request * cont_req, MPI_Info info);
```

MPI Continuations

- `MPIX_Continue[all]`:
  - Return immediately
  - Take ownership of non-persistent requests
  - May signal immediate completion (`flag`)
  - Never invoke any callbacks!
- Statuses:
  - User-provided status object(s)
  - Set before returning (immediate completion) or before invoking callback
  - May be `MPI_STATUS[ES]_IGNORE`

```
typedef void (MPIX_Continue_cb_function)(
  MPI_Status * statuses , void* cb_data);

int MPIX_Continue(
  MPI_Request* op_request,
  int * flag,          // true if complete immediately
  MPIX_Continue_cb_function *cb, // callback to invoke
  void * cb_data,          // data to pass
  MPI_Status* status,      // array of statuses
  MPI_Request cont_req     // Continuation Request
);
```

```
int MPIX_Continueall (
  int count,
  MPI_Request op_requests [],
  int* flag,          // true if complete immediately
  MPIX_Continue_cb_function *cb, // callback to invoke
  void* cb_data,           // data to pass
  MPI_Status statuses [], // array of statuses
  MPI_Request cont_req     // Continuation Request
);
```

```
int MPIX_Continue_init(
  MPI_Request * cont_req, MPI_Info info);
```

MPI Continuations

- `MPIX_Continue[all]`:
  - Return immediately
  - Take ownership of non-persistent requests
  - May signal immediate completion (`flag`)
  - Never invoke any callbacks!
- Statuses:
  - User-provided status object(s)
  - Set before returning (immediate completion) or before invoking callback
  - May be `MPI_STATUS[ES]_IGNORE`
- Continuation Requests:
  - Accumulate continuations
  - Complete once last continuation executed
  - Persistent request implicitly started upon first registration (after init or completion)
  - Provide progress facility
  - May itself have continuation attached

```
typedef void (MPIX_Continue_cb_function)(
  MPI_Status * statuses , void* cb_data);

int MPIX_Continue(
  MPI_Request* op_request,
  int * flag,          // true if complete immediately
  MPIX_Continue_cb_function *cb, // callback to invoke
  void * cb_data,          // data to pass
  MPI_Status* status,      // array of statuses
  MPI_Request cont_req     // Continuation Request
);
```

```
int MPIX_Continueall (
  int count,
  MPI_Request op_requests [],
  int* flag,          // true if complete immediately
  MPIX_Continue_cb_function *cb, // callback to invoke
  void* cb_data,           // data to pass
  MPI_Status statuses [], // array of statuses
  MPI_Request cont_req     // Continuation Request
);
```

```
int MPIX_Continue_init(
  MPI_Request * cont_req, MPI_Info info);
```

Example: MPI Detach

```
omp_event_handle_t event;



/* task to receive data */
#pragma omp task depend(out: recvbuf) detach(event)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  MPIX_Detach(opreq,
              &omp_fulfill_event, /* callback to invoke */
              event,              /* argument to pass */
  );

}

/* task to process received data */
#pragma omp task depend(in: recvbuf)
  process_received_data(recvbuf);

/* wait for all tasks to complete */
#pragma omp taskwait
```

Example: MPI Detach

```
omp_event_handle_t event;



/* task to receive data */
#pragma omp task depend(out: recvbuf) detach(event)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  MPIX_Detach(opreq,
              &omp_fulfill_event, /* callback to invoke */
              event,              /* argument to pass */
  );

}

/* task to process received data */
#pragma omp task depend(in: recvbuf)
  process_received_data(recvbuf);

/* wait for all tasks to complete */
#pragma omp taskwait
```

Progress Function

```
void mpi_progress()
{
    MPIX_Progress(NULL);
}
```

⇝ Progress thread, recurring task, or service

Example: MPI Continuations

```
omp_event_handle_t event;
/* set up continuation request */
MPI_Request contreq;
MPIX_Continue_init(&contreq, MPI_INFO_NULL);

/* task to receive data */
#pragma omp task depend(out: recvbuf) detach(event)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  if (flag) omp_fulfill_event(event);
}

/* task to process received data */
#pragma omp task depend(in: recvbuf)
  process_received_data(recvbuf);

/* wait for all tasks to complete */
#pragma omp taskwait

MPI_Request_free(&contreq);
```

Example: MPI Continuations

```
omp_event_handle_t event;
/* set up continuation request */
MPI_Request contreq;
MPIX_Continue_init(&contreq, MPI_INFO_NULL);

/* task to receive data */
#pragma omp task depend(out: recvbuf) detach(event)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  if (flag) omp_fulfill_event(event);
}

/* task to process received data */
#pragma omp task depend(in: recvbuf)
  process_received_data(recvbuf);

/* wait for all tasks to complete */
#pragma omp taskwait

MPI_Request_free(&contreq);
```

Continuation Callback

```
void complete_event(
  MPI_Status *status,
  void       *cb_data)
{
  omp_event_handle_t event;
  event = (omp_event_handle_t) cb_data;
  /* release dependencies */
  omp_fulfill_event(event);
}
```

Example: MPI Continuations

```c
omp_event_handle_t event;
/* set up continuation request */
MPI_Request contreq;
MPIX_Continue_init(&contreq, MPI_INFO_NULL);

/* task to receive data */
#pragma omp task depend(out: recvbuf) detach(event)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  if (flag) omp_fulfill_event(event);
}

/* task to process received data */
#pragma omp task depend(in: recvbuf)
  process_received_data(recvbuf);

/* wait for all tasks to complete */
#pragma omp taskwait

MPI_Request_free(&contreq);
```

Continuation Callback

```c
void complete_event(
  MPI_Status *status,
  void       *cb_data)
{
  omp_event_handle_t event;
  event = (omp_event_handle_t) cb_data;
  /* release dependencies */
  omp_fulfill_event(event);
}
```

Progress Function

```c
void mpi_progress()
{
  int flag; // ignored
  MPI_Test(&contreq, &flag,
           MPI_STATUS_IGNORE);
}
```

⇝ Progress thread, recurring task, or service

Summary

MPI Detach

- `MPIX_Detach[_all|_each]`, `MPIX_Detach_status[_all|_each]`
- Explicit progress call (`MPIX_Progress`)
- Status objects allocated by MPI
- Currently implemented as library
- Execution by progress thread or `MPIX_Progress`

J. Protze, M.-A.Hermanns, A. Demiralp, M. S. Müller, and T. Kuhlen. 2020. *MPI Detach - Asynchronous Local Completion*. DOI: `https://doi.org/10.1145/3416315.3416323`

Summary

MPI Detach

- `MPIX_Detach[_all|_each]`, `MPIX_Detach_status[_all|_each]`
- Explicit progress call (`MPIX_Progress`)
- Status objects allocated by MPI
- Currently implemented as library
- Execution by progress thread or `MPIX_Progress`

J. Protze, M.-A.Hermanns, A. Demiralp, M. S. Müller, and T. Kuhlen. 2020. *MPI Detach - Asynchronous Local Completion*. DOI: `https://doi.org/10.1145/3416315.3416323`

MPI Continuations

- `MPIX_Continue[all]`
- Progress through *Continuation Requests*
- Immediate completion signal
- Execution from within MPI routines*
- Implemented inside Open MPI

J. Schuchart, C. Niethammer, and J. Garcia. 2020. *Fibers are not (P)Threads: The Case for Loose Coupling of Asynchronous Programming Models and MPI Through Continuations.* DOI: `https://doi.org/10.1145/3416315.3416320`

Continuations: Configuration Options

Current implemented via `MPI_Info`

- `continue_poll_only`: only execute continuations when polling a Continuation Request*
  ⇝ Useful for heavy callbacks to not disturb other threads
- `continue_enqueue_complete`: always return `flag=0` and enqueue for later execution*
  ⇝ Useful if no time for immediate completion handling
- `continue_max_poll`: max number of continuations to execute when testing `cont_req`
  ⇝ Useful to ensure quick turnaround with large numbers of complete requests

* *Better as parameters to `MPIX_Continue`?*

Handling of Canceled Requests

```c
void request_cb(MPI_Status *status, void *data)
{
  int is_canceled = 0;
  MPI_Test_cancelled(status, &is_canceled);
  if (is_canceled) return;
  ...
}

void function_that_cancels()
{
  MPI_Status *status = malloc(sizeof(status));
  MPI_Request req;
  MPI_Irecv_init(..., &req);
  MPI_Start(&req);
  MPIX_Continue(&req, ..., status, ..., cont_req);
  MPI_Cancel(&req);
}
```

⇝ No specific callback required for canceled requests :)

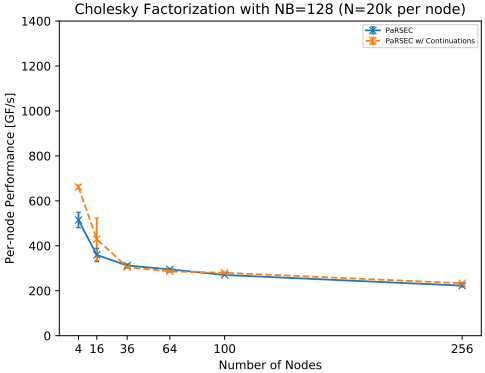
Why MPI_Continue may not execute callbacks

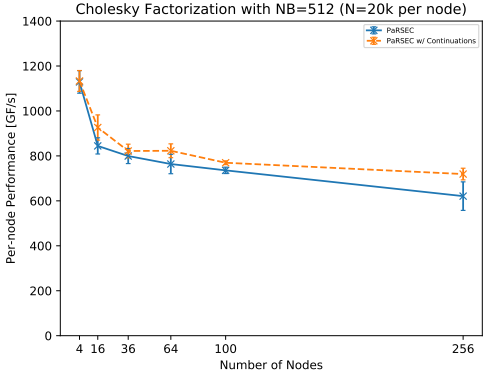
```c
void block_on_req(fiber_state_t *fs, MPI_Request *req) {
  int flag;
  ABT_mutex_lock(fs->mtx);
  /* execution of complete_cb would cause deadlock */
  MPIX_Continue(&req, &flag, &complete_cb, fs, MPI_STATUS_IGNORE, cont_req);
  /* block waiting for completion */
  if (!flag) ABT_cond_wait (fs->cond, fs->mtx);
  ABT_mutex_unlock(fs->mtx);
}

int complete_cb(MPI_Status * status, void *data) {
  fiber_state_t * ts = (fiber_state_t *) data;
  /* take mutex to avoid losing signals */
  ABT_mutex_lock(fs->mtx);
  ABT_cond_signal(fs->cond);
  ABT_mutex_unlock(fs->mtx);
  return MPI_SUCCESS;
}
```

Use Cases: System Configuration

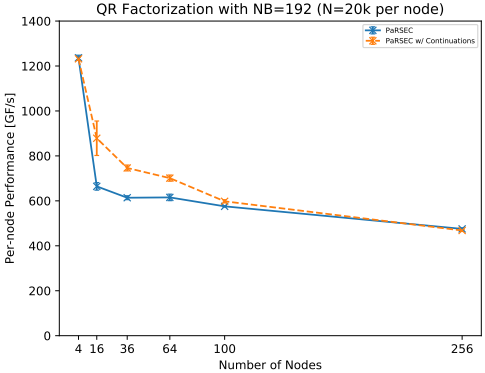
- HPE Apollo *Hawk* installed at HLRS
- 5,632 nodes, 2x AMD Epyc 7742 (Rome)
- InfiniBand HDR200
- GCC 10.2.0
- Clang 12 (development)
- Open MPI 4.0.x

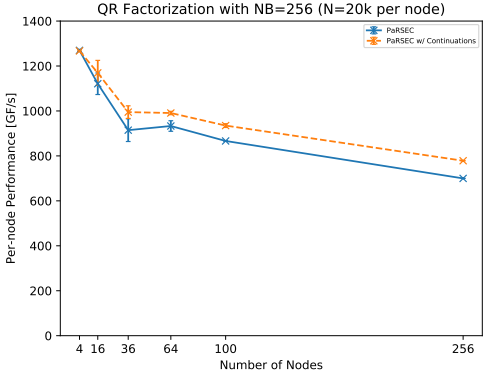
Use Case: PaRSEC Cholesky



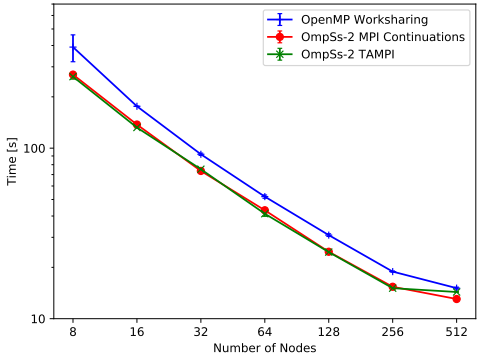


Use Case: PaRSEC QR Factorization





Use Case: NPB BT-MZ



Summary & Open Questions

- Restrictions on execution context? (e.g., signal handler, internal progress thread)
- Allow for multiple continuations to be attached to an operation request?
  - Or allow for querying attached continuations?
- Who should allocate `MPI_Status` objects?
- How to ensure progress?

Use-cases to explore:

- C++ Interface?
- CUDA streams?

Discussion