# Completion Notification using MPI Continuations

Joseph Schuchart

February 25, 2021

# MPI & Task-based Programming Models

**MPI *provides***

- Non-blocking two-sided/collective communication, RMA & IO
- Requests represent operations
- Completion notification through *polling*

# MPI & Task-based Programming Models

**MPI *provides***

- Non-blocking two-sided/collective communication, RMA & IO
- Requests represent operations
- Completion notification through *polling*

**Task-based Applications *have to***

- Handle hundreds of requests in application-space
- Poll for completion
- *React to state changes*

# MPI in OpenMP Tasks

- MPI ≈ dependencies not exposed to the scheduler
- Using MPI in OpenMP is next to impossible[1]

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

[1] J. Schuchart, K. Tsugane, J. Gracia, M. Sato. "The Impact of Taskyield on the Design of Tasks Communicating Through MPI." In: Evolving OpenMP for Evolving Architectures (Proceedings of IWOMP'18). 2018. Awarded Best Paper.

# MPI in OpenMP Tasks

- MPI ≈ dependencies not exposed to the scheduler
- Using MPI in OpenMP is next to impossible[1]

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

[1] J. Schuchart, K. Tsugane, J. Gracia, M. Sato. "The Impact of Taskyield on the Design of Tasks Communicating Through MPI." In: Evolving OpenMP for Evolving Architectures (Proceedings of IWOMP'18). 2018. Awarded Best Paper.

# MPI in OpenMP Tasks

- MPI ≈ dependencies not exposed to the scheduler
- Using MPI in OpenMP is next to impossible[1]

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


[1] J. Schuchart, K. Tsugane, J. Gracia, M. Sato. "The Impact of Taskyield on the Design of Tasks Communicating Through MPI." In: Evolving OpenMP for Evolving Architectures (Proceedings of IWOMP'18). 2018. Awarded Best Paper.

# MPI in OpenMP Tasks

- MPI ≈ dependencies not exposed to the scheduler
- Using MPI in OpenMP is next to impossible[1]

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


**Previous approaches:** TAMPI, Argobots/Qthreads integration in MPI, . . .
⇝ **Not portable!**

[1] J. Schuchart, K. Tsugane, J. Gracia, M. Sato. "The Impact of Taskyield on the Design of Tasks Communicating Through MPI." In: Evolving OpenMP for Evolving Architectures (Proceedings of IWOMP'18). 2018. Awarded Best Paper.


WHAT IF MPI COULD TELL ME
WHEN MY RECV IS COMPLETE?
imgflip.com

# MPI Continuations: An Example

```
/* task to receive data */
#pragma omp task depend(out: recvbuf)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  do {
    MPI_Test(&opreq, &flag,
             MPI_STATUS_IGNORE);
    if (flag) break;
    /* May or may not work! */
    #pragma omp taskyield
  } while (1);
}

/* 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 Continuations: An Example

```
/* task to receive data */
#pragma omp task depend(out: recvbuf)
{
  int flag;
  MPI_Request opreq;
  MPI_Irecv(recvbuf, ..., &opreq);
  do {
    MPI_Test(&opreq, &flag,
             MPI_STATUS_IGNORE);
    if (flag) break;
    /* May or may not work! */
    #pragma omp taskyield
  } while (1);
}

/* 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 Continuations: An Example

```
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);
  /* register a Continuation */
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  /* release dependency if completed immediately */
  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 Continuations: An Example

```
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);
  /* register a Continuation */
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  /* release dependency if completed immediately */
  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);
```

# MPI Continuations: An Example

```
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);
  /* register a Continuation */
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  /* release dependency if completed immediately */
  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);
}
```

# MPI Continuations: An Example

```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);
  /* register a Continuation */
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  /* release dependency if completed immediately */
  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);
}
```

# MPI Continuations: An Example

```
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);
  /* register a Continuation */
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  /* release dependency if completed immediately */
  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);
}
```

## Progress Function

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

⇝ Progress thread, recurring task, or service

# MPI Continuations: An Example

```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);
  /* register a Continuation */
  MPIX_Continue(&opreq, &flag,
                &complete_event, /* callback to invoke */
                event,           /* argument to pass */
                MPI_STATUS_IGNORE, contreq);
  /* release dependency if completed immediately */
  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

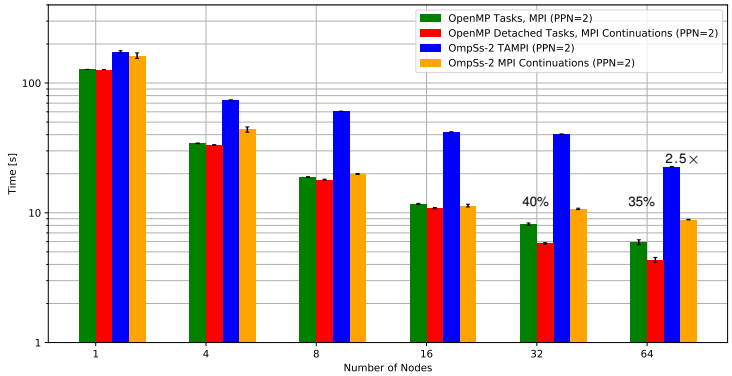
```c
void complete_event(
  MPI_Status *status,
  void       *cb_data)
{
  omp_event_handle_t event;
  event = (omp_event_handle_t) cb_data;
  /* release dependencies */
  omp_fulfill_event(event);
}
```

## Progress Function

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

⇝ Progress thread, recurring task, or service

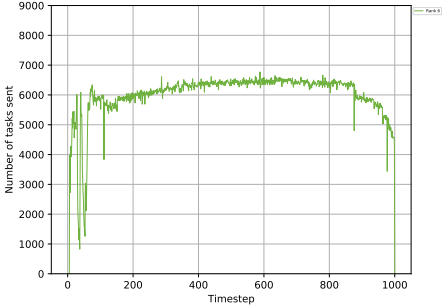
# Early Results: NPB BT-MZ



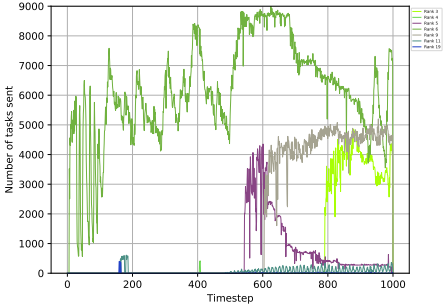

Class D @ Hawk (2× AMD Epyc 7742 64C, 128 GB)

# Early Results: ExaHyPE

Reference




Using Continuations




Order 7, $81^3$ cell cloud simulation, 12 nodes @ Hawk (2× AMD Epyc 7742 64C, 128 GB)

# Conclusions

**MPI Continuations**[2]

- Move MPI request management into MPI
- Let applications focus on application concern
- Fine-grain control over behavior using info keys
- Demonstrated use with:
  - Argobots
  - OpenMP detached tasks
  - OmpSs-2
  - PaRSEC
  - Dynamic load balancing in ExaHyPE
- Looking for *feedback* & *use-cases*

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

# Conclusions

**MPI Continuations**[2]

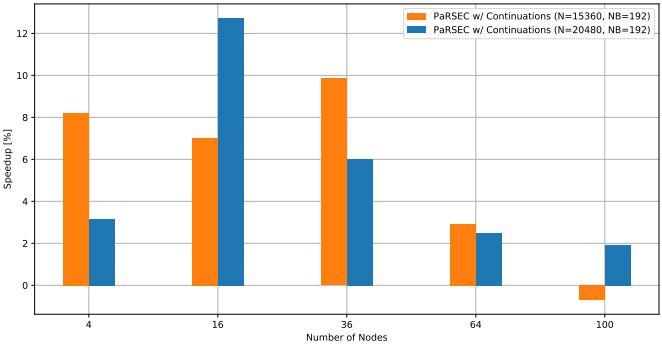
- Move MPI request management into MPI
- Let applications focus on application concern
- Fine-grain control over behavior using info keys
- Demonstrated use with:
  - Argobots
  - OpenMP detached tasks
  - OmpSs-2
  - PaRSEC
  - Dynamic load balancing in ExaHyPE
- Looking for *feedback* & *use-cases*

**Thank you!**

schuchart@icl.utk.edu

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

# Preliminary Results: PaRSEC




DGEQRF @ Hawk (2× AMD Epyc 7742 64C, 128 GB)

# MPI Continuations: API[3]

- `MPIX_Continue[all]`:
  - Returns immediately
  - Takes ownership of non-persistent requests
  - May signal immediate completion (`flag = 1`)
  - Never invokes 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);
```

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

# MPI Continuations: API[3]

- `MPIX_Continue[all]`:
  - Returns immediately
  - Takes ownership of non-persistent requests
  - May signal immediate completion (`flag = 1`)
  - Never invokes 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);
```

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

# MPI Continuations: API[3]

- `MPIX_Continue[all]`:
  - Returns immediately
  - Takes ownership of non-persistent requests
  - May signal immediate completion (`flag = 1`)
  - Never invokes 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
  - 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);
```

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