Tasks, PGAS, and MPI Continuations

Joseph Schuchart
HLRS, Stuttgart University | ICL, UTK

WSSP31, March 19, 2021

Introduction

- Recently finished PhD thesis at HLRS / Stuttgart University
  - Title: *Global Task Data Dependencies in PGAS Applications*
  - Supervisor: Prof. Michael Resch
  - Second Assessor: Prof. Barbara Chapman

Today's talk:

- Global Task Data-Dependencies in PGAS
- Task-based programming using MPI Continuations

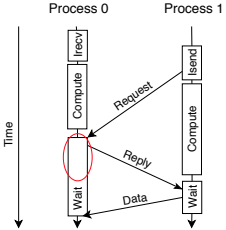
Background: Challenges of MPI+OpenMP

MPI: Message Passing Interface
OpenMP: Thread-parallel work-sharing

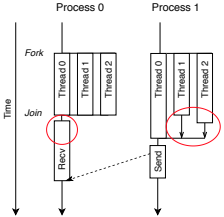
MPI *two-sided communication*:

- Couples communication and synchronization
- Limited latency hiding
- Message Matching

Traditional MPI+OpenMP fork/join model involves various *synchronization points*



MPI Synchronization



MPI+OpenMP Synchronization

Background: Challenges of MPI+OpenMP

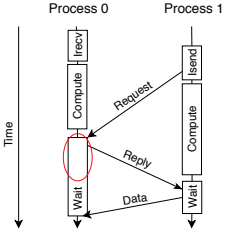
MPI: Message Passing Interface
OpenMP: Thread-parallel work-sharing

MPI *two-sided communication*:

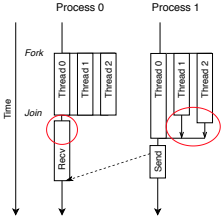
- Couples communication and synchronization
- Limited latency hiding
- Message Matching

Traditional MPI+OpenMP fork/join model involves various *synchronization points*

Goal: Dynamic programming model with *minimal synchronization*



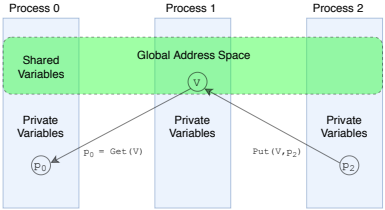
MPI Synchronization



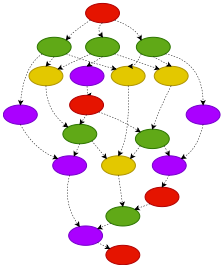
MPI+OpenMP Synchronization

New Models Gaining Traction

Partitioned Global Address Space (PGAS)

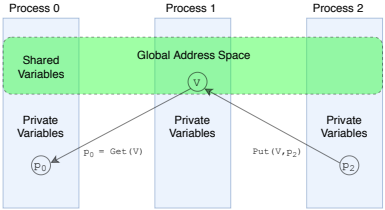


Task-based programming



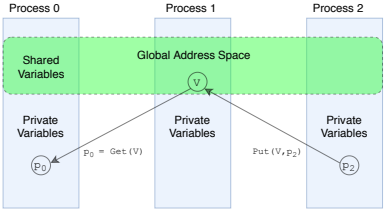
Background: Partitioned Global Address Space (PGAS)

- *Shared memory* across physical node boundaries
- Direct access to low-level hardware features
  - No message matching
  - No synchronization between sender and receiver
- Decoupled synchronization and data transfer
- Low-level interfaces:
  - MPI Remote Memory Access (MPI RMA)
  - OpenSHMEM
  - GASnet
  - GASPI



Background: Partitioned Global Address Space (PGAS)

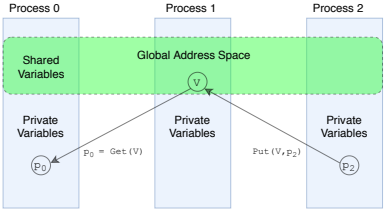
- *Shared memory* across physical node boundaries
- Direct access to low-level hardware features
  - No message matching
  - No synchronization between sender and receiver
- Decoupled synchronization and data transfer
- Low-level interfaces:
  - MPI Remote Memory Access (MPI RMA)
  - OpenSHMEM
  - GASnet
  - GASPI



Problem: Decoupled synchronization and data transfer!

Background: Partitioned Global Address Space (PGAS)

- *Shared memory* across physical node boundaries
- Direct access to low-level hardware features
  - No message matching
  - No synchronization between sender and receiver
- Decoupled synchronization and data transfer
- Low-level interfaces:
  - MPI Remote Memory Access (MPI RMA)
  - OpenSHMEM
  - GASnet
  - GASPI



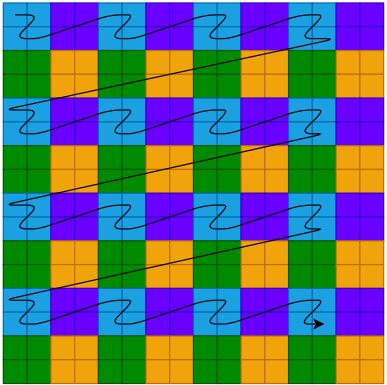
Problem: Decoupled synchronization and data transfer!

Goal: Synchronize data accesses in Global Address Space

The DASH PGAS Library[1]

```
// select 2D tiled distribution
using PatternT = typename dash::TilePattern<2>;
// type of the 2D tiled matrix to use
using TiledMatrix = dash::NArray<double, 2, dash::default_index_t, PatternT>;

// specify 2D distribution of processes
dash::TeamSpec<2> ts(dash::size(), 1);
ts.balance_extents();

// Allocate the NxM matrix with tiles
// of size NBxMB
TiledMatrix A(N, M, dash::TILE(2),
                    dash::TILE(2), ts);

// local iteration over all elements
for (value_t* it = A.lbegin(); it != A.lend(); ++it) {
...
}

/* Wait for everyone to complete */
dash::barrier();

/* Copy the first row */
dash::copy(A.begin(), A.begin() + M, local_buffer);
```

16×16 tiled matrix on 4 procs

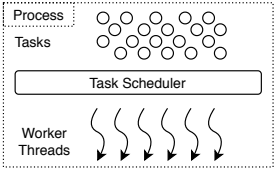


[1] K. Fuerlinger, T. Fuchs, R. Kowalewski. *DASH: A C++ PGAS Library for Distributed Data Structures and Parallel Algorithms*, In: 18th IEEE International Conference on High Performance Computing and Communications. 2016.

Background: Task-based Programming

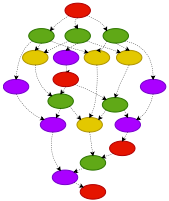
Application expressed as set of:

- **Work packages** with well-defined inputs and outputs
- Expressed **concurrency**, scheduler-managed **parallelism**
- **Oversubscription** reduces idle time



Task Synchronization

- **Constraints** on the execution order of tasks
- Correct order of memory accesses



OpenMP tasks hard to couple with MPI[2]

OpenMP task-dependencies strictly local

[2] 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.

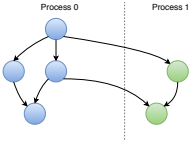
PhD Contribution

Data Dependencies on Global Memory

- Data-centric
- No explicit communication channels

Local Task Discovery

- No scalability bottlenecks during discovery
- No memory access restrictions



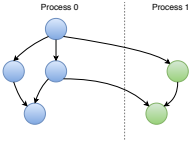
PhD Contribution

Data Dependencies on Global Memory

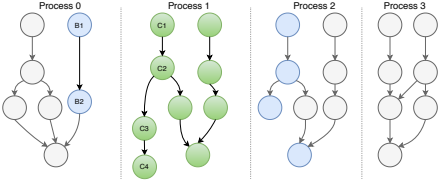
- Data-centric
- No explicit communication channels

Local Task Discovery

- No scalability bottlenecks during discovery
- No memory access restrictions



Distributed data-dependency discovery?



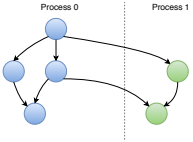
PhD Contribution

Data Dependencies on Global Memory

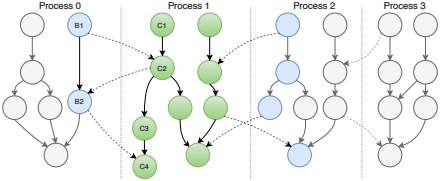
- Data-centric
- No explicit communication channels

Local Task Discovery

- No scalability bottlenecks during discovery
- No memory access restrictions



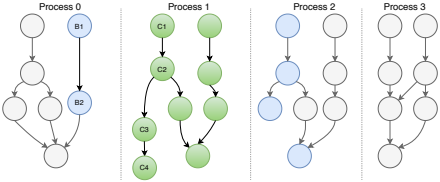
Distributed data-dependency discovery?



Local Dependency Matching

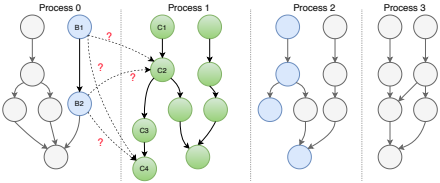
Dependencies in same process matched similar to OpenMP (formal model established)

- Strictly *backward matching*
  $\rightsquigarrow$ Graph dictated by *order of discovery*



Global Dependency Matching

- Order of discovery insufficient for distributed dependencies
- Additional information required

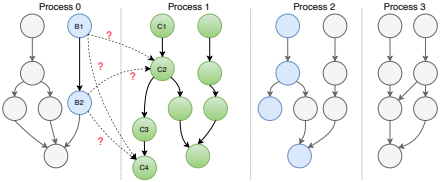


Global Dependency Matching

- Order of discovery insufficient for distributed dependencies
- Additional information required

Enter: Phases

- Horizontal task graph partitioning
- *Backward matching*
  ⇝ Remote dependencies match *previous phases*
  ⇝ Proof on deadlock-freedom in thesis

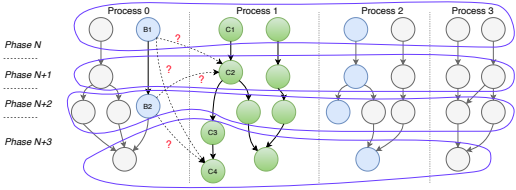


Global Dependency Matching

- Order of discovery insufficient for distributed dependencies
- Additional information required

Enter: Phases

- Horizontal task graph partitioning
- *Backward matching*
  ⇝ Remote dependencies match *previous phases*
  ⇝ Proof on deadlock-freedom in thesis

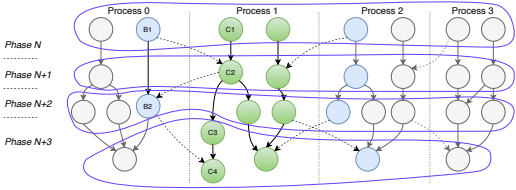


Global Dependency Matching

- Order of discovery insufficient for distributed dependencies
- Additional information required

Enter: Phases

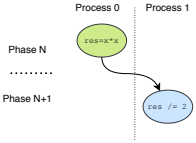
- Horizontal task graph partitioning
- *Backward matching*
  - ⇝ Remote dependencies match *previous phases*
  - ⇝ Proof on deadlock-freedom in thesis



Global Task Data Dependencies: Implementation

- **Context**: DASH C++ PGAS library
  - Distributed static and dynamic containers
  - Communication through MPI RMA
  - Applicable to all PGAS abstractions providing *global pointers*

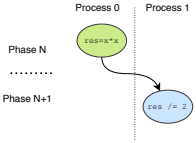
Global Task Data Dependencies: Implementation



- **Context**: DASH C++ PGAS library
  - Distributed static and dynamic containers
  - Communication through MPI RMA
  - Applicable to all PGAS abstractions providing *global pointers*
- Combine `std::async` and data dependencies
- Phases expressed through `async_fence`

```
/* Value in Global Address Space */
Shared<int> res{};
if (res.is_local()) {
  int x = input();
  async([&, x](){
      res.set(x*x);
    }, in(x), out(res)
  );
}

async_fence();

if (!res.is_local()) {
{
  async([&](){
      res.set(res.get() / 2);
    }, inout(res)
  );
}
complete();
```

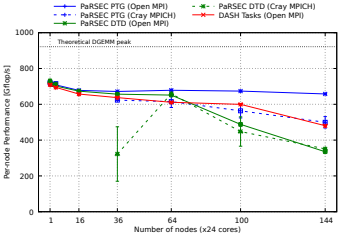
Global Task Data Dependencies: Implementation



- **Context**: DASH C++ PGAS library
  - Distributed static and dynamic containers
  - Communication through MPI RMA
  - Applicable to all PGAS abstractions providing *global pointers*
- Combine `std::async` and data dependencies
- Phases expressed through `async_fence`
- Cancellation, taskloops, reductions, tool integration, . . .

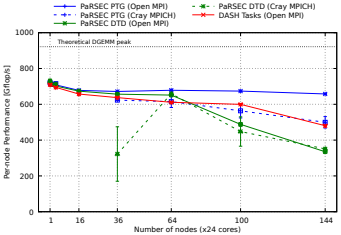
```
/* Value in Global Address Space */
Shared<int> res{};
if (res.is_local()) {
  int x = input();
  async([&, x](){
      res.set(x*x);
    }, in(x), out(res)
  );
}

async_fence();

if (!res.is_local()) {
{
  async([&](){
      res.set(res.get() / 2);
    }, inout(res)
  );
}
complete();
```

Cholesky Factorization: Cray XC40

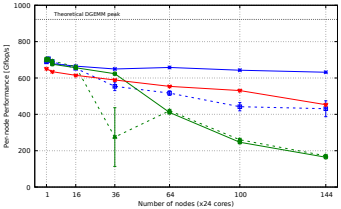


N=25k^2/node, NB=320^2

Cholesky Factorization: Cray XC40

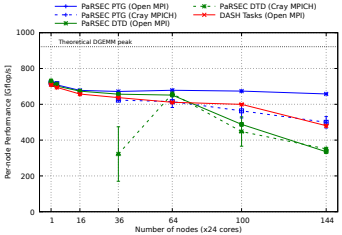


N=25k^2/node, NB=320^2

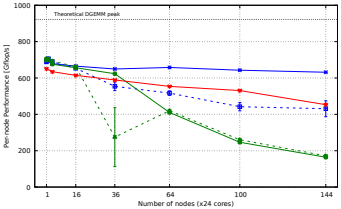


N=24k^2/node, NB=256^2

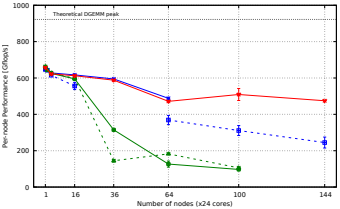
Cholesky Factorization: Cray XC40



N=25k^2/node, NB=320^2



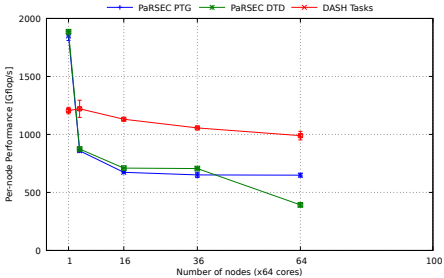
N=24k^2/node, NB=256^2



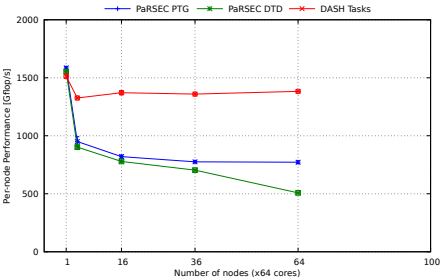
N=24k^2/node, NB=192^2

Per-node performance (higher is better)

Cholesky Factorization: Hawk

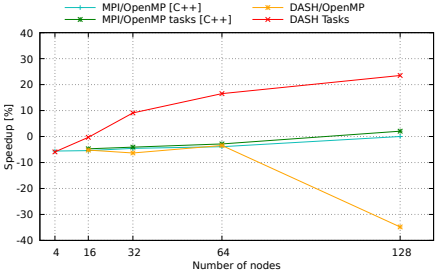


N=25k^2/node, NB=320^2



N=35k^2/node, NB=320^2

NPB BT-MZ: Taurus



Strong scaling speedup over Fortran reference implementation, Class D, Taurus

Summary

- PGAS promise low communication overhead
- Synchronization is crucial in PGAS
- Proposal: lightweight global task dependencies

Summary

- PGAS promise low communication overhead
- Synchronization is crucial in PGAS
- Proposal: lightweight global task dependencies

Outlook

- Explore use in ML (Stonybrook / Brookhaven)
- Transfer techniques to PaRSEC

MPI & Asynchronous Programming Models

- MPI $\approx$ dependencies not exposed to scheduler
- Coupling with OpenMP is difficult
- Request management is tedious
- Potentially *thousands* of active requests
- Test-yield cycles costly
- Previous proposals: TAMPI, Argobots/Qthread integration, . . .
  $\rightsquigarrow$ **None of that is portable!**


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

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


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

MPI Continuations: 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;
  } while (1);
  /* May or may not work! */
  # pragma omp taskyield
}

/* 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: 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;
  } while (1);
  /* May or may not work! */
  # pragma omp taskyield
}

/* 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: 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: 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: 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: 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: 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: 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
- 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

MPI Continuations

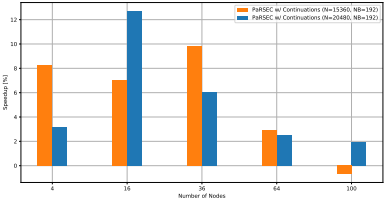
- Move MPI request management back into MPI
- Let applications focus on application-level issues
- Demonstrated use with:
  - Argobots
  - OpenMP detached tasks
  - OmpSs-2
  - PaRSEC / DPLASMA[4]
  - Dynamic load balancing in ExaHyPE[4]
- Under consideration in MPI WG
- Looking for interesting *use cases*!
- Concurrent (similar) proposal: MPI Detach[5]

[4] J. Schuchart, P. Samfass, C. Niethammer, J. Gracia, G. Bosilca. 2021. *Callback-based Completion Notification using MPI Continuations*. Submitted for review to *Parallel Computing*.

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

MPI Continuations

- Move MPI request management back into MPI
- Let applications focus on application-level issues
- Demonstrated use with:
  - Argobots
  - OpenMP detached tasks
  - OmpSs-2
  - PaRSEC / DPLASMA[4]
  - Dynamic load balancing in ExaHyPE[4]
- Under consideration in MPI WG
- Looking for interesting *use cases*!
- Concurrent (similar) proposal: MPI Detach[5]

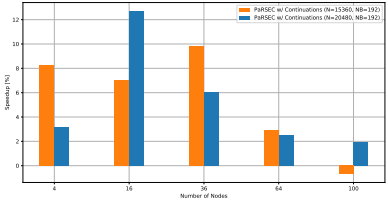


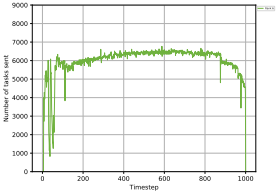
[4] J. Schuchart, P. Samfass, C. Niethammer, J. Gracia, G. Bosilca. 2021. *Callback-based Completion Notification using MPI Continuations*. Submitted for review to *Parallel Computing*.

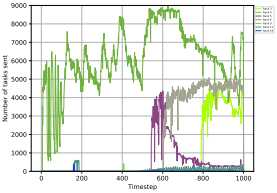
[5] J. Protze, M.-A. Hermanns, A. Demiralp, M. S. Müller, and T. Kuhlen. 2020. *MPI Detach – Asynchronous Local Completion*. `https://doi.org/10.1145/3416315.3416323`

MPI Continuations

- Move MPI request management back into MPI
- Let applications focus on application-level issues
- Demonstrated use with:
  - Argobots
  - OpenMP detached tasks
  - OmpSs-2
  - PaRSEC / DPLASMA[4]
  - Dynamic load balancing in ExaHyPE[4]
- Under consideration in MPI WG
- Looking for interesting *use cases*!
- Concurrent (similar) proposal: MPI Detach[5]





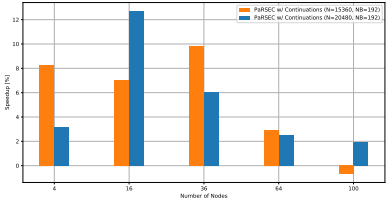


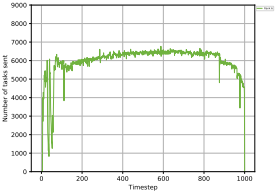
[4] J. Schuchart, P. Samfass, C. Niethammer, J. Gracia, G. Bosilca. 2021. *Callback-based Completion Notification using MPI Continuations*. Submitted for review to *Parallel Computing*.

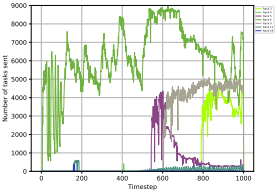
[5] J. Protze, M.-A. Hermanns, A. Demiralp, M. S. Müller, and T. Kuhlen. 2020. *MPI Detach – Asynchronous Local Completion*. `https://doi.org/10.1145/3416315.3416323`

MPI Continuations

- Move MPI request management back into MPI
- Let applications focus on application-level issues
- Demonstrated use with:
  - Argobots
  - OpenMP detached tasks
  - OmpSs-2
  - PaRSEC / DPLASMA[4]
  - Dynamic load balancing in ExaHyPE[4]
- Under consideration in MPI WG
- Looking for interesting *use cases*!
- Concurrent (similar) proposal: MPI Detach[5]







Looking for applications and feedback :)

[4] J. Schuchart, P. Samfass, C. Niethammer, J. Gracia, G. Bosilca. 2021. *Callback-based Completion Notification using MPI Continuations*. Submitted for review to *Parallel Computing*.

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

Questions?
Contact me at `schuchart@icl.utk.edu` **:)**