MPI Detach - Asynchronous Local Completion

Joachim Protze (protze@itc.rwth-aachen.de), Marc-André Hermanns,
Ali Demiralp, Matthias S. Müller, Torsten Kuhlen

Performance Optimisation and Productivity
A Centre of Excellence in HPC

Visual Computing Institute | High Performance Computing | RWTH AACHEN UNIVERSITY

Disclaimers

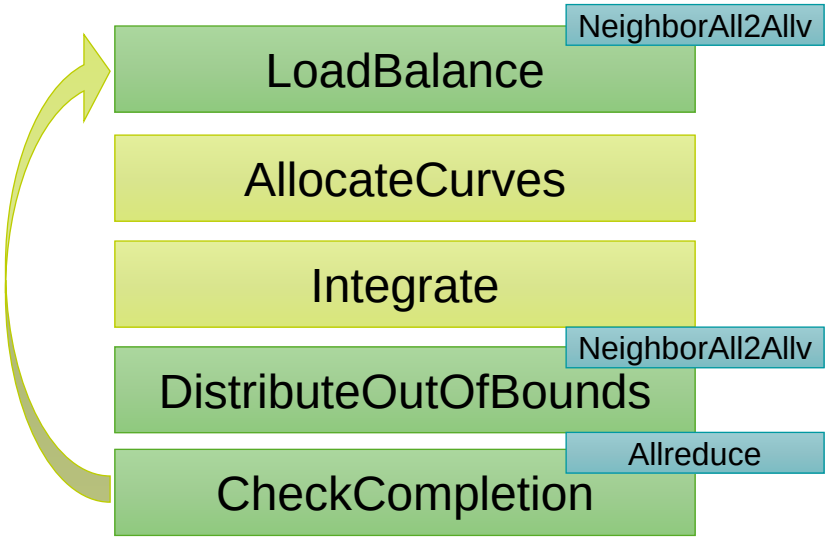
Names are hollow words!

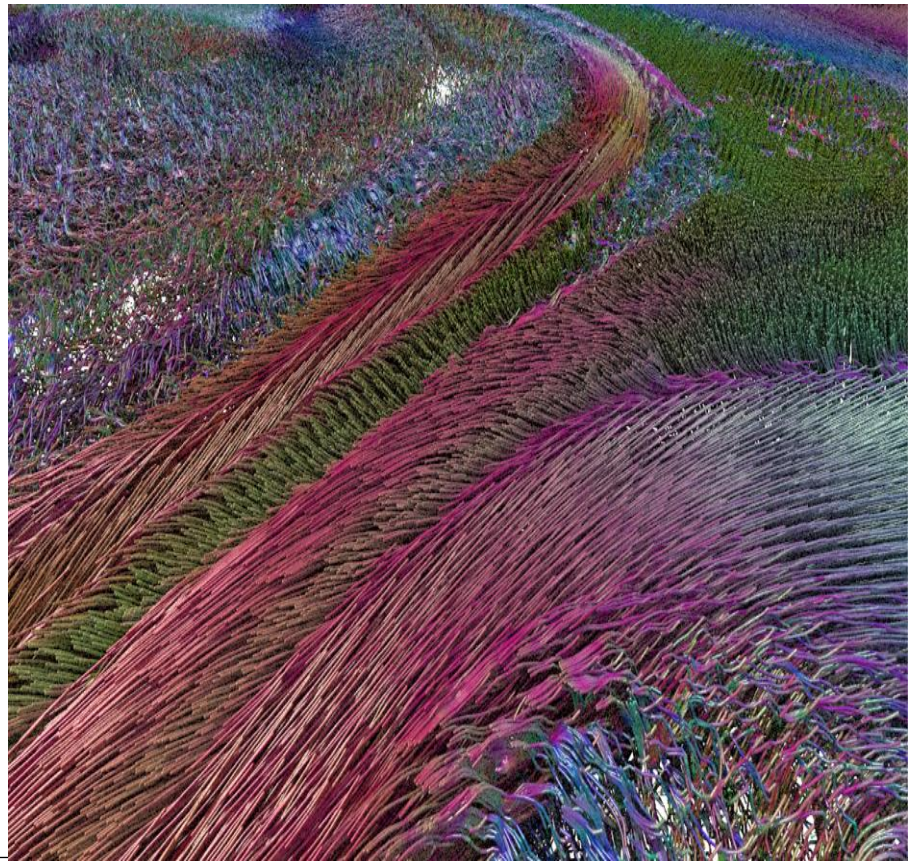
Ignore the names, question the concepts ☺

All functions and other entities appearing in this work are fictitious. Any resemblance to real functions, dead or alive, or other real-life entities, past or present, is purely coincidental.

Distributed particle advection

- DPA: Particle advector built on Boost MPI and Intel TBB.
  - Static data partitioning across nodes.
  - Dynamic diffusive load balancing of particles/workload.
- Potential for improving performance:
  - Overlap compute/communication
  - Enable better termination checking.



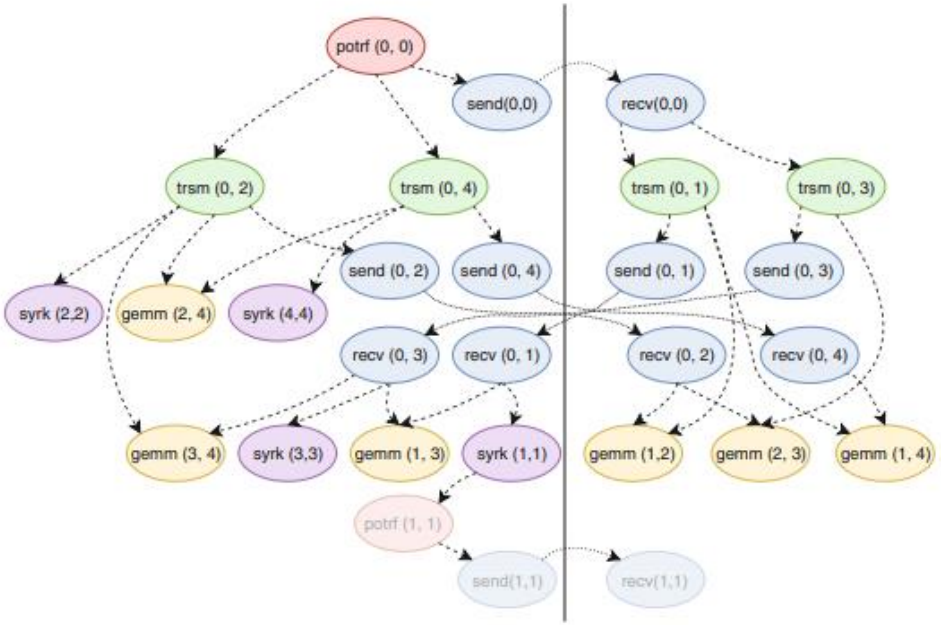


Communication in OpenMP tasking programms

- Work initially presented by Schuchart et al. at IWOMP'18
- Calculation tasks generate data
- Sender tasks depend on the data to communicate
- Receiver tasks have out dependencies for the received data

➢ Current MPI + OpenMP provide no scalable solution

➢ TAMPI + OmpSs-2 show a feasible implementation



Wishlist

From the MPI forum issue tracker (288)

```
auto buffer = some_t{num_ranks};
auto future = gather(comm, root(comm), my_offsets, buffer)
              .then([&](){
                root_only([&](){ root_does_something_with(buffer); });
                return broadcast(comm, root(comm), buffer);
              }).then([&]() {
                return all_do_something_with(buffer);
              }).then([&](auto result) {
                return all_reduce(comm, root(comm), result,
                          [](auto acc, auto v) { return acc && v; });
              }).then([&](auto result) {
                if (result) { return; /* we are done */ }
                else {
                  root_only([](){ write_some_error_log(); });
                  throw some_exception;
                }
              });
/* Here nothing has happened yet! */
/* ... lots and lots of unrelated code that can execute concurrently
       and overlaps with communication ... */

future.get();
```

OpenMP tasks using detach clause (OpenMP 5.0)

```
#pragma omp task depend(out:recvbuf) detach(req)
{
  MPI_Request req;
  MPI_Irecv(recvbuf, …, &req);
} // the task finishes execution and returns, but completion
  // is coupled with completion of the receive

#pragma omp task depend(in:sendbuf)
  MPI_Send(sendbuf, …);


#pragma omp task depend(in:recvbuf)
  do_something_with_buf(recvbuf);
```

Our contribution

Proposed MPI detach functions

- Hand back request to MPI library and register for a completion callback.
- Compare to `int MPI_Wait(MPI_Request *request, MPI_Status *status)`:

```
typedef void MPIX_Detach_status_function(void *data, MPI_Status status);

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

- If we are not interested in the status, we cannot pass `MPI_STATUS_IGNORE`:

```
typedef void MPIX_Detach_function(void *data);

int MPIX_Detach(
  MPI_Request *request,
  MPIX_Detach_function *callback,
  void *data);
```

Proposed MPI detach functions

- Compare to `int MPI_Waitall(int count, MPI_Request requests[], MPI_Status statuses[])`:

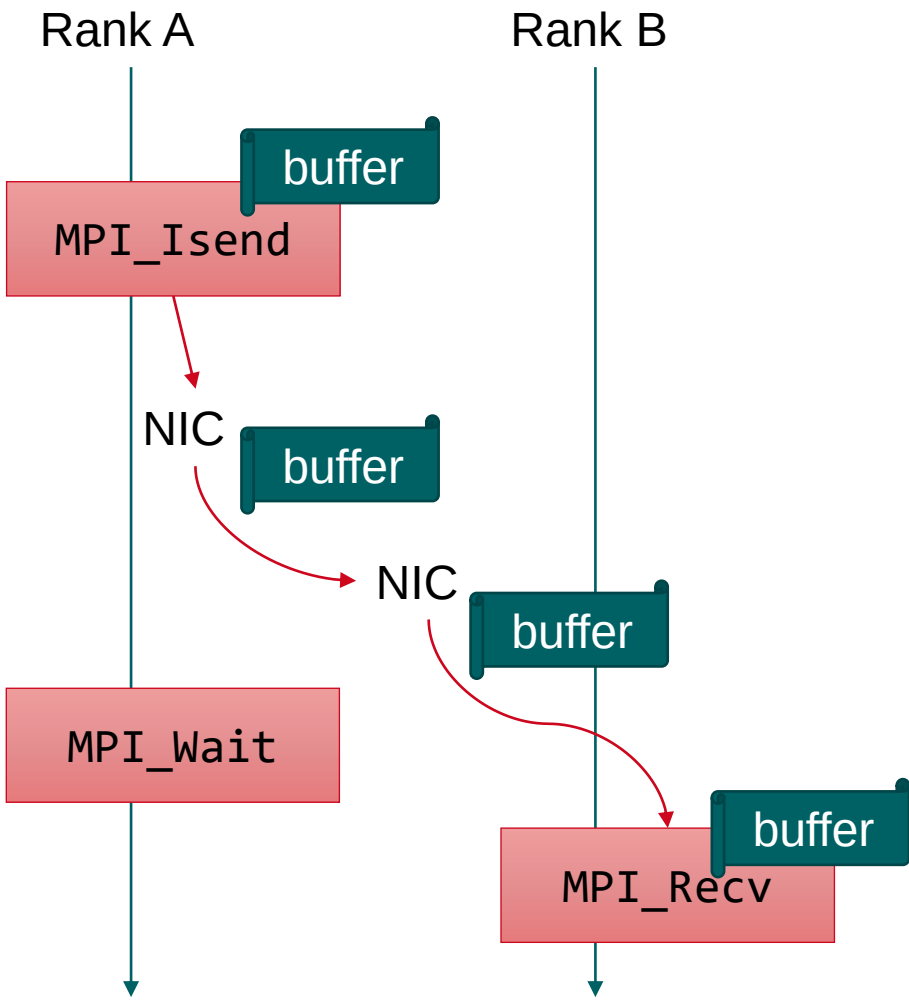
```
typedef void MPIX_Detach_statuses_function(void *data, int count, MPI_Status
statuses[]);

int MPIX_Detach_all_status(
  int count,
  MPI_Request requests[],
  MPIX_Detach_status_function *callback,
  void *data);
```

- Again, we cannot pass `MPI_STATUSES_IGNORE`:

```
int MPIX_Detach_all(
  MPI_Request *request,
  MPIX_Detach_function *callback,
  void *data);
```

Progress in MPI communication

- MPI guarantees that once a matching pair of send and receive is initiated, one of them will (eventually) complete.
- Choices:
  - Is there a progress thread, supervising communication progress?
  - Is progress only driven during API calls?



Driving progress

- When should MPI make progress after calling `MPIX_Detach`?

- The new function `MPIX_Progress` can be registered with a polling service:

```
int MPIX_Progress(void *data);
```

- The polling service calls the registered function when ever it is convenient.
- The function must be non-blocking.

- Calling this function regularly will finally allow the completion callback to be called.

Usage

Wrapping MPI detach into C++ future

```
std::future<mpi::status> detach_as_future(mpi::request& r)
{
  auto* promise = new std::promise<mpi::status>();
  auto future = promise->get_future();
  mpix::detach(&r, [promise] (const mpi::status& s)
  {
    promise->set_value(s);
    delete promise;
  });
  return std::move(future);
}
```

From the MPI forum issue tracker (288)

```
auto buffer = some_t{num_ranks};
auto future = gather(comm, root(comm), my_offsets, buffer)
              .then([&](){
                root_only([&](){ root_does_something_with(buffer); });
                return broadcast(comm, root(comm), buffer);
              }).then([&]() {
                return all_do_something_with(buffer);
              }).then([&](auto result) {
                return all_reduce(comm, root(comm), result,
                          [](auto acc, auto v) { return acc && v; });
              }).then([&](auto result) {
                if (result) { return; /* we are done */ }
                else {
                  root_only([](){ write_some_error_log(); });
                  throw some_exception;
                }
              });
/* Here nothing has happened yet! */
/* ... lots and lots of unrelated code that can execute concurrently
       and overlaps with communication ... */

future.get();
```

OpenMP tasks using detach clause (OpenMP 5.0)

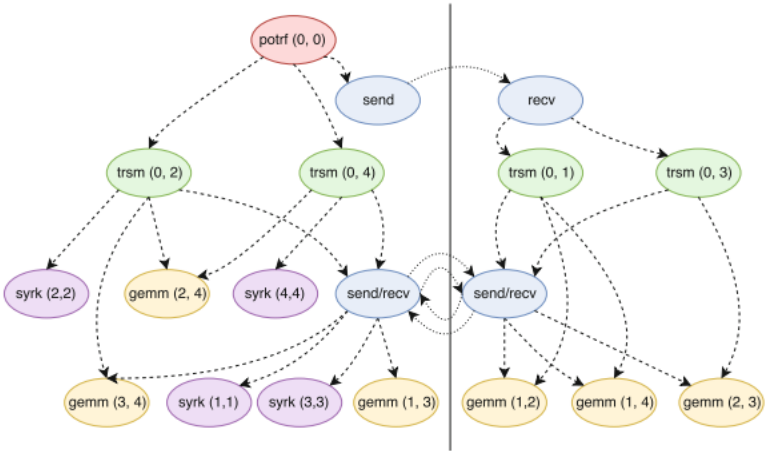
```
omp_event_handle_t ev_handle;
#pragma omp task detach(ev_handle) depend(out: recvbuf)
{
  MPI_Request req;
  MPI_Irecv(recvbuf, …, &req);
  MPIX_Detach(&req, (MPI_Detach_callback *) omp_fulfill_event,
                    (void*) ev_handle);
}


#pragma omp task depend(in: recvbuf)
  do_something_with_buf(recvbuf);
```

Evaluation

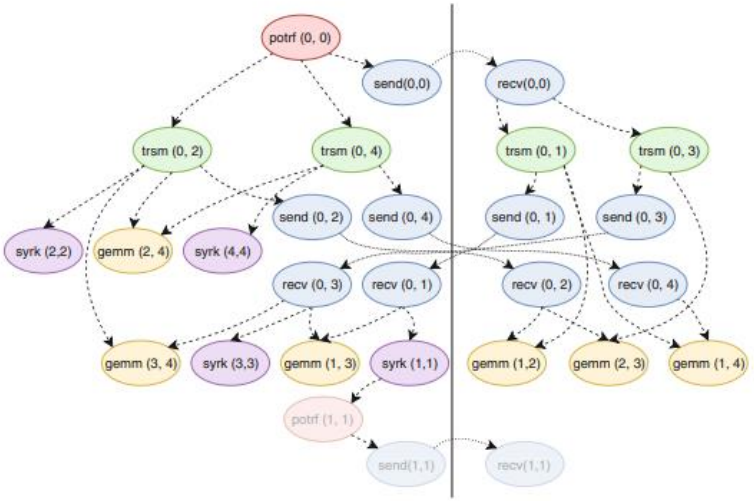
Extending distributed Blocked Cholesky Factorization with MPI detach

- Presented by Schuchart et al. at IWOMP'18:
https://github.com/devreal/cholesky_omptasks

- Coarse-grain communication (***singlecom***) using a single communication task per level



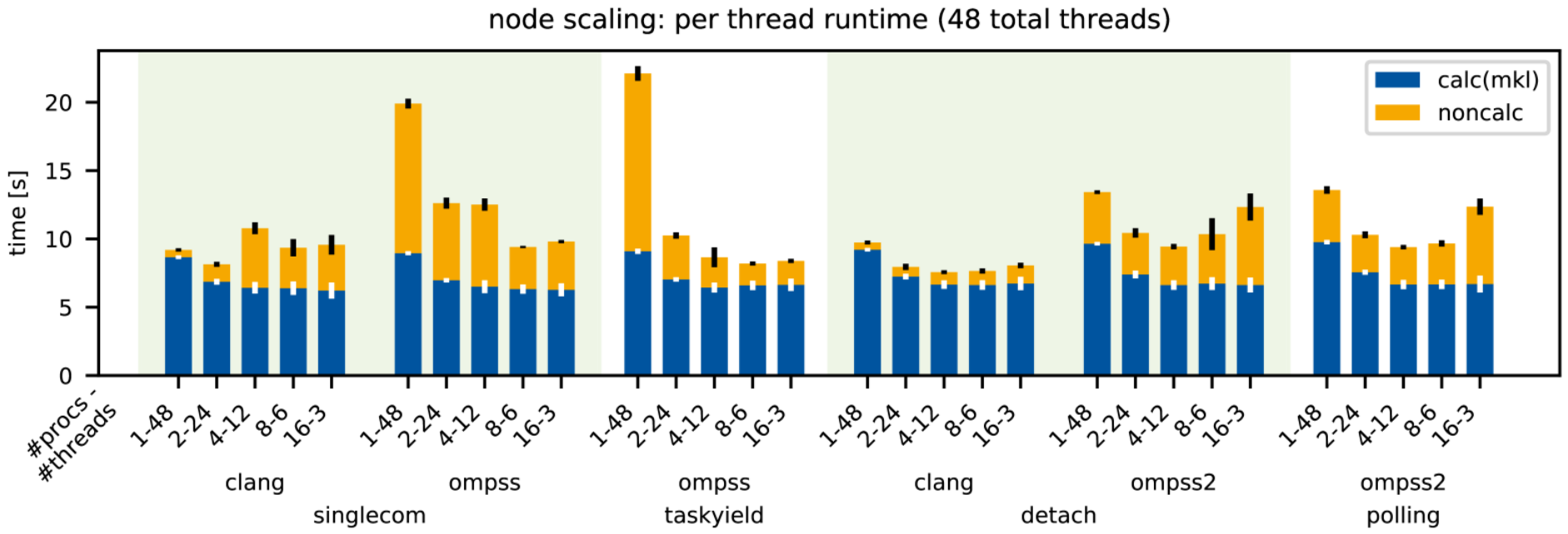
- Fine-grain communication using a communication task per block
  - ***taskyield***: test-loop with taskyield
  - ***detach***: MPIX-detach replaces test-loop
  - ***polling***: MPIX-progress replaces progress thread



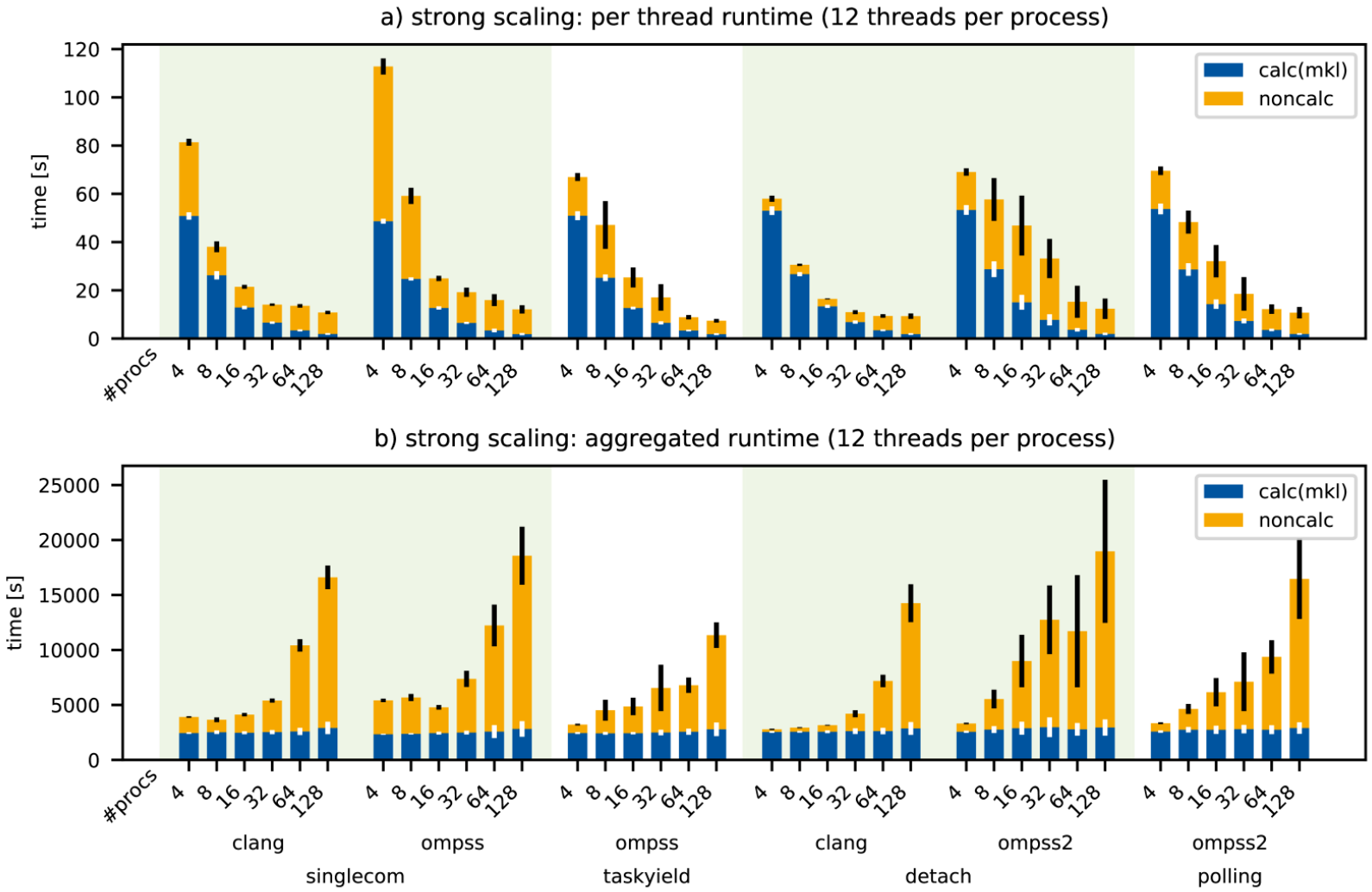
Graphs from: Schuchart et al., *The Impact of Taskyield on the Design of Tasks Communicating Through MPI*, IWOMP'18

Node scaling experiment: $(32k)^2$ matrix, 256^2 blocks

- Filling 48 cores of a node with MPI processes/OpenMP threads



Strong scaling experiment: (64k)² matrix, 256² blocks



Conclusion

Conclusion

- Slim extension of MPI (7 new functions) to provide asynchronous local completion
- Usage presented for OpenMP tasks and C++ futures
  - Can also be used in other tasking models
- Prototype implemented as PMPI wrapper:
  - https://github.com/RWTH-HPC/mpi-detach (thread-safe and just 347 sloc)
- Performance study based on distributed blocked cholesky factorization:
  - https://github.com/RWTH-HPC/cholesky_omptasks
- Next step: Add a polling service interface to OpenMP