NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Introduction to Parallel Programming with MPI

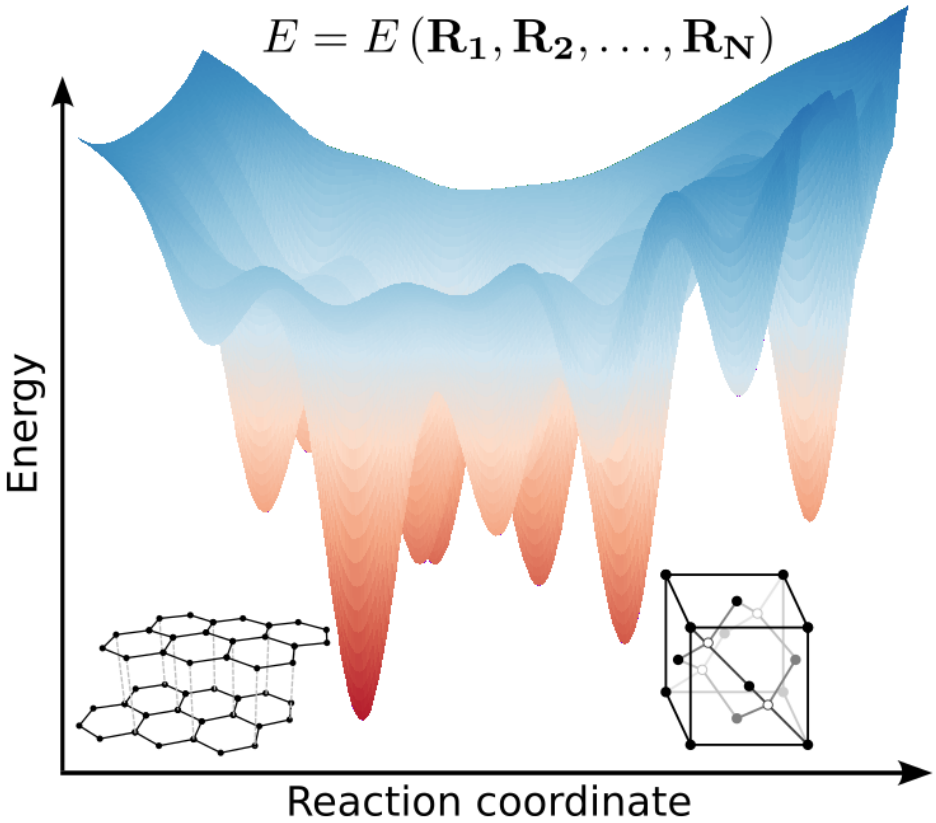
Dr. Alireza Ghasemi, Dr. Georg Hager

Erlangen National High Performance Computing Center

MPI Point-to-Point Communication: Nonblocking

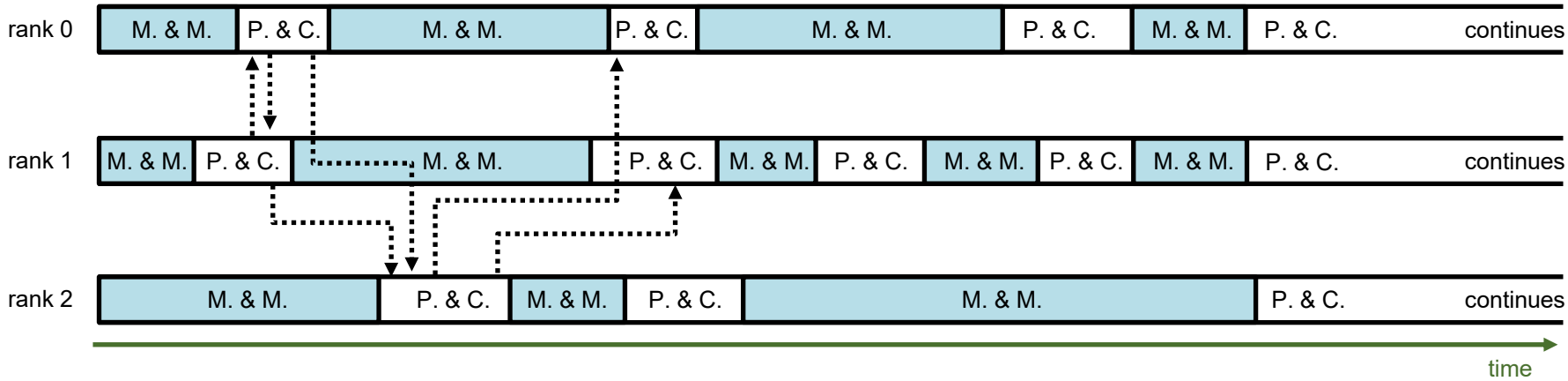
Crystalline and Molecular Structures

- Challenges in structure search: chemistry and material science
  - Local minimization
    - Computational cost: variable
  - Many local minima
    - Exponential increase with respect to system size
  - Global optimization methods
    - Deterministic and stochastic walker
- Parallel computers:
  - Each MPI process taking care of a walker



Work load imbalance

- Walkers:
  - Move and Minimization
  - Processing and Communication

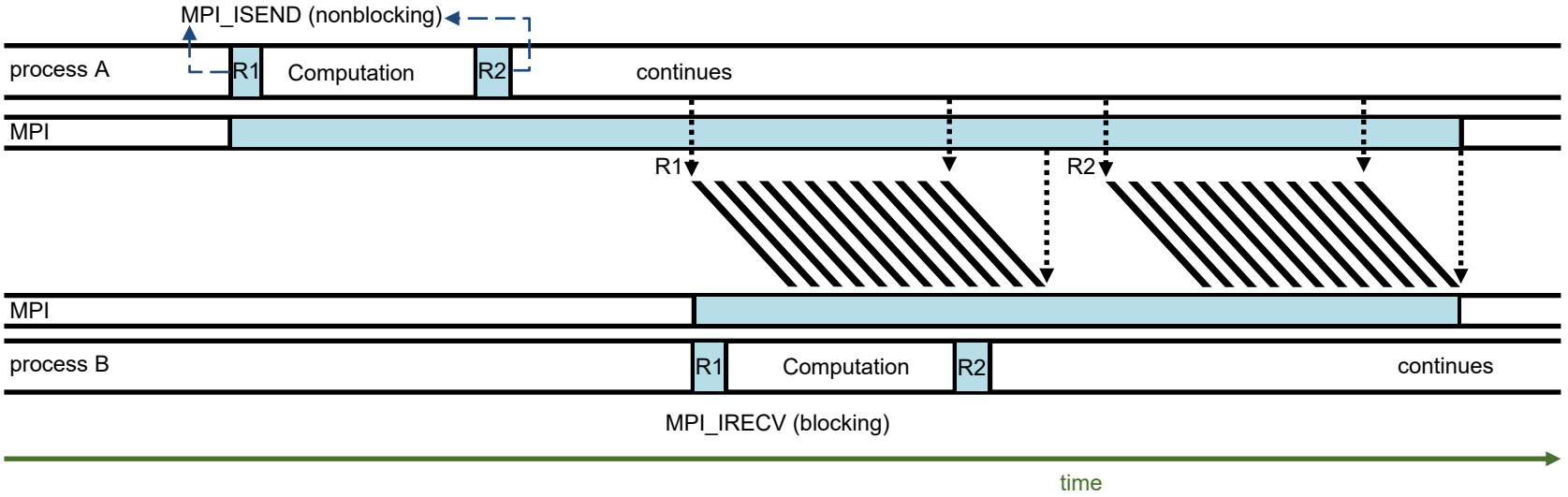


- If blocking PtP communication mode would be used:
  - will result in significant loss of resources: idle time and synchronization
  - The timelines of walkers would include long idle time of many processes

This is a prime example that blocking PtP communication is inappropriate!

Nonblocking point-to-point communication

- Call to a nonblocking send/recv procedure returns straight away. It avoids synchronization so that the following opportunities can be exploited:
  - Avoiding certain deadlocks
  - Truly bidirectional commun.
  - Avoid idle time:
    - Overlapping commun. and comput.



Standard nonblocking send/receive

- ```
  MPI_Isend(sendbuf, count, datatype, dest,   tag,
            comm, MPI_Request * request);

  MPI_Irecv(recvbuf, count, datatype, source, tag,
            comm, MPI_Request * request);
  ```

  `request`: pointer to variable of type **`MPI_Request`**,
  will be associated with the corresponding operation
  ```

Standard nonblocking send/receive

- `MPI_Isend(sendbuf, count, datatype, dest, tag, comm, MPI_Request * request);`

  `MPI_Irecv(recvbuf, count, datatype, source, tag, comm, MPI_Request * request);`

  `request`: pointer to variable of type **`MPI_Request`**, will be associated with the corresponding operation
- **Do not reuse `sendbuf/recvbuf` before `MPI_Isend/MPI_Irecv` has been completed!!!**
  - Return of call does not imply completion
- **`MPI_Irecv`** has no status argument
  - obtained later during completion via **`MPI_Wait*/MPI_Test*`**

Nonblocking send and receive variants

- Completion
  - Return of **`MPI_I*`** call does not imply completion
  - Check for completion via **`MPI_Wait* / MPI_Test*`**
  - Semantics identical to blocking call after successful completion

nonblocking MPI function	blocking MPI function	type	completes when
`MPI_Isend`	**`MPI_Send`**	synchronous or buffered	depends on type
`MPI_Ibsend`	`MPI_Bsend`	buffered	buffer has been copied
`MPI_Issend`	`MPI_Ssend`	synchronous	remote starts receive
`MPI_Irecv`	**`MPI_Recv`**	--	message was received

Test for completion

Two test modes:

Test for completion

Two test modes:

- Blocking
  - `MPI_Wait*`: Wait until the communication has been completed and buffer can safely be reused

Test for completion

Two test modes:

- Blocking
  - `MPI_Wait*`: Wait until the communication has been completed and buffer can safely be reused
- Nonblocking
  - `MPI_Test*`: Return true (false) if the communication has (not) completed

Test for completion

Two test modes:

- Blocking
  - **`MPI_Wait*`**: Wait until the communication has been completed and buffer can safely be reused
- Nonblocking
  - **`MPI_Test*`**: Return true (false) if the communication has (not) completed

Despite the naming, the modes both pertain to nonblocking point-to-point communication!

Test for completion – single request

- Test one communication handle for completion:

```
MPI_Wait(MPI_Request * request,
         MPI_Status * status);
MPI_Test(MPI_Request * request, int * flag,
         MPI_Status * status);
```

request: request handle of type **MPI_Request**

status: status object of type **MPI_Status** (cf. **MPI_Recv**)

flag: variable of type **int** to test for success

Use of wait/test

MPI_Wait

```
MPI_Request request;
MPI_Status status;


MPI_Isend(
  send_buffer, count, MPI_CHAR,
  dst, 0, MPI_COMM_WORLD, &request);


// do some work…
// do not use send_buffer

MPI_Wait(&request, &status);

// use send_buffer
```

Use of wait/test

MPI_Wait

```
MPI_Request request;
MPI_Status status;


MPI_Isend(
  send_buffer, count, MPI_CHAR,
  dst, 0, MPI_COMM_WORLD, &request);


// do some work…
// do not use send_buffer

MPI_Wait(&request, &status);

// use send_buffer
```

MPI_Test

```
MPI_Request request;
MPI_Status status;
int flag;

MPI_Isend(
  send_buffer, count, MPI_CHAR,
  dst, 0, MPI_COMM_WORLD, &request);

do {
    // do some work…
    // do not use send_buffer
    MPI_Test(&request, &flag, &status);
} while (!flag);

// use send_buffer
```

Wait for completion – all requests in a list

- MPI can handle multiple communication requests
- Wait/Test for completion of multiple requests:

```
MPI_Waitall(int count, MPI_Request requests[],
                       MPI_Status statuses[]);

MPI_Testall(int count, MPI_Request requests[],
            int *flag, MPI_Status statuses[]);
```

- Waits for/Tests if all provided requests have been completed

Use of `MPI_Waitall`

```
MPI_Request requests[2];
MPI_Status  statuses[2];

MPI_Isend(send_buffer, …, &(requests[0]));
MPI_Irecv(recv_buffer, …, &(requests[1]));

// do some work…

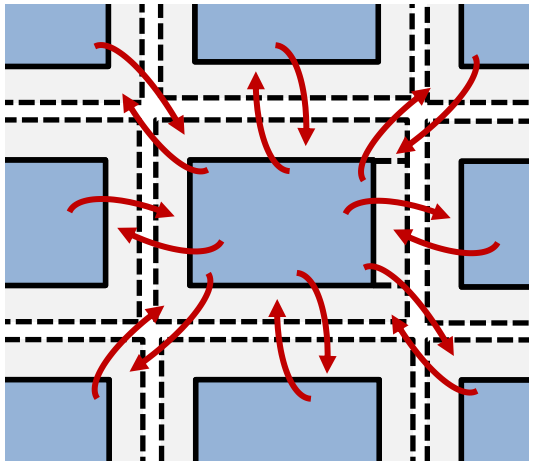
MPI_Waitall(2, requests, statuses)
// Isend & Irecv have been completed
```

Arrays of requests and statuses

number of elements in the arrays

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once

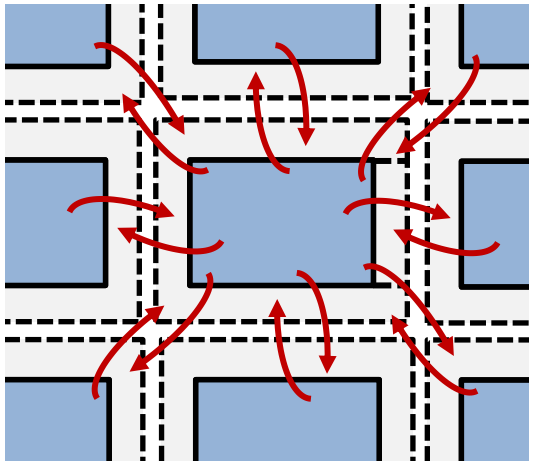


Possible implementation:

1. Update cells that need the halo

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once

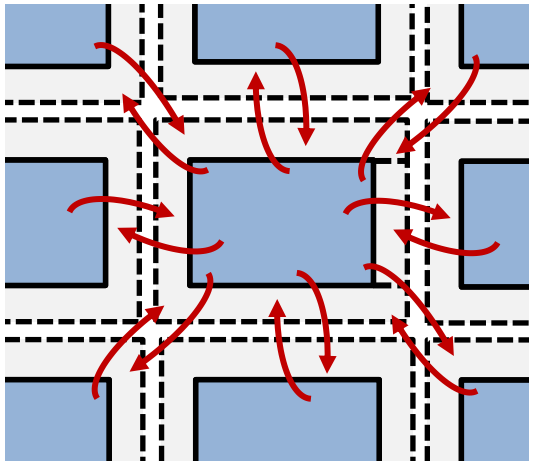


Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once

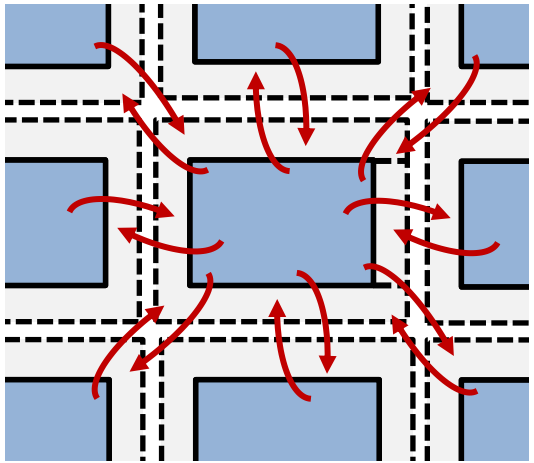


Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers
3. Start nonblocking receives/sends from/to corresponding neighbors

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once

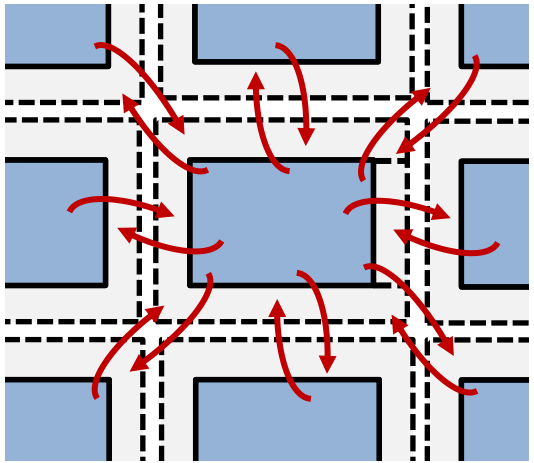


Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers
3. Start nonblocking receives/sends from/to corresponding neighbors
4. Update local cells that do not need halo cells for boundary conditions (“bulk update”)

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once

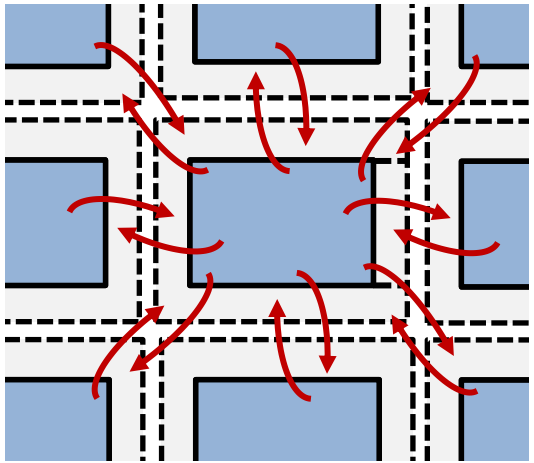


Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers
3. Start nonblocking receives/sends from/to corresponding neighbors
4. Update local cells that do not need halo cells for boundary conditions (“bulk update”)
5. Wait with `MPI_Waitall` for all obtained requests to complete

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once

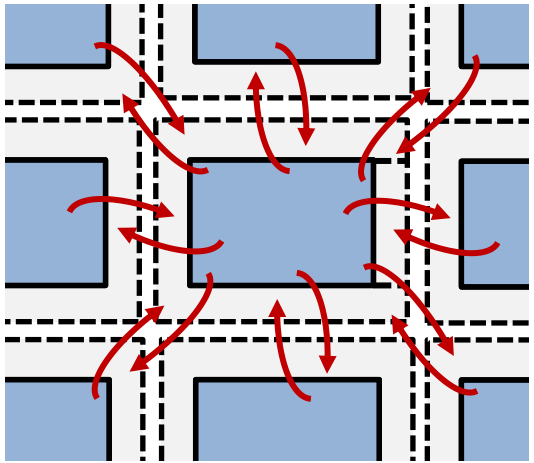


Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers
3. Start nonblocking receives/sends from/to corresponding neighbors
4. Update local cells that do not need halo cells for boundary conditions (“bulk update”)
5. Wait with `MPI_Waitall` for all obtained requests to complete
6. Copy received halo data into ghost cells

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once



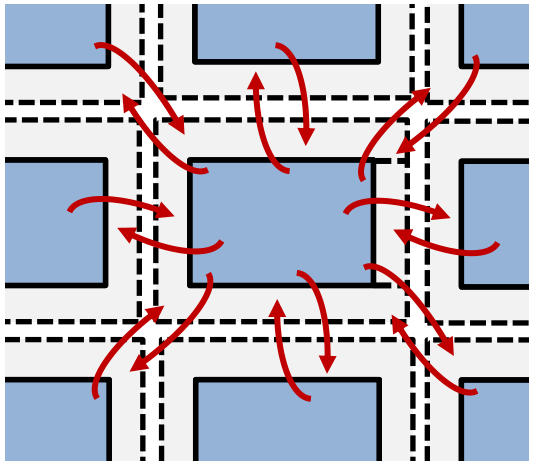
Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers
3. Start nonblocking receives/sends from/to corresponding neighbors
4. Update local cells that do not need halo cells for boundary conditions (“bulk update”)
5. Wait with `MPI_Waitall` for all obtained requests to complete
6. Copy received halo data into ghost cells

Opportunity to overlap communication (steps 3-5) with bulk update

Ghost cell exchange with nonblocking MPI

Ghost cell exchange with nonblocking send/recv with all neighbors at once



Possible implementation:

1. Update cells that need the halo
2. Copy new data into contiguous send buffers
3. Start nonblocking receives/sends from/to corresponding neighbors
4. Update local cells that do not need halo cells for boundary conditions (“bulk update”)
5. Wait with `MPI_Waitall` for all obtained requests to complete
6. Copy received halo data into ghost cells

Opportunity to overlap communication (steps 3-5) with bulk update (MPI implementation permitting)

Wait for completion – one or several requests out of a list

Wait for/Test if exactly one request among many has been completed

- ```
  MPI_Waitany(int count, MPI_Request requests[],
              int * idx, MPI_Status * status);

  MPI_Testany(int count, MPI_Request requests[],
              int * idx, int * flag,
              MPI_Status * status);
  ```
  ```

Wait for completion – one or several requests out of a list

Wait for/Test if exactly one request among many has been completed

- ```
  MPI_Waitany(int count, MPI_Request requests[],
              int * idx, MPI_Status * status);

  MPI_Testany(int count, MPI_Request requests[],
              int * idx, int * flag,
              MPI_Status * status);
  ```

Wait for/Test if at least one request among many has been completed

- ```
  MPI_Waitsome(int incount, MPI_Request requests[], int * outcount,
               int indices[], MPI_Status statuses[]);

  MPI_Testsome(int incount, MPI_Request requests[], int * outcount,
               int indices[], MPI_Status statuses[]);
  ```

Use of `MPI_Testany`

```
MPI_Request requests[2];
MPI_Status  status;
int finished = 0;

MPI_Isend(send_buffer, …, &(requests[0]));
MPI_Irecv(recv_buffer, …, &(requests[1]));
do {
  // do some work…
  MPI_Testany(2, requests, &idx, &flag, &status);
  if (flag) { ++finished; }
} while (finished < 2);
```

- completed requests are automatically set to `MPI_REQUEST_NULL`
- completed requests: `requests[idx]`

Pitfalls with nonblocking MPI and compiler optimizations

- Fortran:

```
MPI_IRECV(recvbuf, ..., request, ierror)
MPI_WAIT(request, status, ierror)
write (*,*) recvbuf
```

- may be compiled as

```
MPI_IRECV(recvbuf, ..., request, ierror)
registerA = recvbuf
MPI_WAIT(request, status, ierror)
write (*,*) registerA
```

- i.e., old data is written instead of received data!
- Workarounds:
  - **recvbuf** may be allocated in a common block, or
  - calling **MPI_GET_ADDRESS(recvbuf, iaddr_dummy, ierror)** after **MPI_WAIT**
  - **asynchronous attribute**

Pitfalls with nonblocking MPI and compiler optimizations

- Fortran:

```
MPI_IRECV(recvbuf, ..., request, ierror)
MPI_WAIT(request, status, ierror)
write (*,*) recvbuf
```

- may be compiled as

```
MPI_IRECV(recvbuf, ..., request, ierror)
registerA = recvbuf
MPI_WAIT(request, status, ierror)
write (*,*) registerA
```

MPI might modify recvbuf after MPI_IRECV returns, but the compiler has no idea about this

- i.e., old data is written instead of received data!
- Workarounds:
  - **recvbuf** may be allocated in a common block, or
  - calling **MPI_GET_ADDRESS(recvbuf, iaddr_dummy, ierror)** after **MPI_WAIT**
  - **asynchronous attribute**

Nonblocking point-to-point communication

- Standard nonblocking send/recv **`MPI_Isend()`**/**`MPI_Irecv()`**
  - Return of call does not imply completion of operation
  - Use **`MPI_Wait*()`** / **`MPI_Test*()`** to check for completion using request handles
- All outstanding requests must be completed!
- Potentials
  - Overlapping of communication with work (not guaranteed by MPI standard)
  - Overlapping send and receive
  - Avoiding synchronization and reducing idle times

- Caveat: Compiler does not know about asynchronous modification of data

Quiz:

1. Every nonblocking send or receive requires a subsequent MPI_Wait* or MPI_Test* call?
   a. Correct
   b. Incorrect

Quiz:

1. Every nonblocking send or receive requires a subsequent MPI_Wait* or MPI_Test* call?

   a. Correct

   b. Incorrect

Answer: a.

Quiz:

1. Every nonblocking send or receive requires a subsequent MPI_Wait* or MPI_Test* call?

   a. Correct b. Incorrect

Answer: a.

2. Can MPI_Isend be matched with blocking receive (MPI_Recv)?

   a. Yes b. No

Quiz:

1. Every nonblocking send or receive requires a subsequent MPI_Wait* or MPI_Test* call?

   a. Correct b. Incorrect

Answer: a.

2. Can MPI_Isend be matched with blocking receive (MPI_Recv)?

   a. Yes b. No

Answer: a.

Quiz:

1. Every nonblocking send or receive requires a subsequent MPI_Wait* or MPI_Test* call?
   a. Correct b. Incorrect

Answer: a.

2. Can MPI_Isend be matched with blocking receive (MPI_Recv)?
   a. Yes b. No

Answer: a.

3. Which one is not a certain benefit of using nonblocking MPI point-to-point calls?
   a. Overlapping send and receive
   b. Avoiding idle times
   c. Overlapping of communication with work

Quiz:

1. Every nonblocking send or receive requires a subsequent MPI_Wait* or MPI_Test* call?
   a. Correct b. Incorrect

Answer: a.

2. Can MPI_Isend be matched with blocking receive (MPI_Recv)?
   a. Yes b. No

Answer: a.

3. Which one is not a certain benefit of using nonblocking MPI point-to-point calls?
   a. Overlapping send and receive
   b. Avoiding idle times
   c. Overlapping of communication with work

Answer: c.