ulm university universität ulm

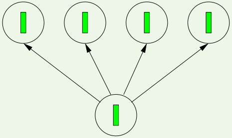
Scientific Computing

Parallele Algorithmen

Prof. Dr. Stefan Funken, Prof. Dr. Alexander Keller,
Prof. Dr. Karsten Urban | 11. Januar 2007

Communication with MPI

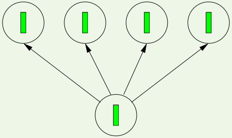
Collective Communication



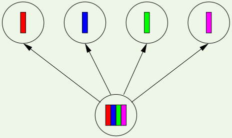
broadcast

Communication with MPI

Collective Communication



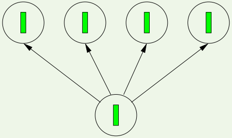
broadcast



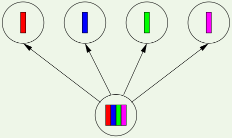
scatter

Communication with MPI

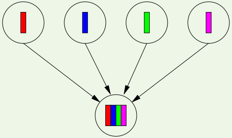
Collective Communication



broadcast



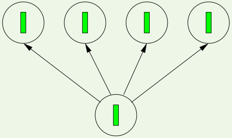
scatter



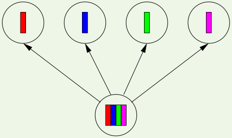
gather

Communication with MPI

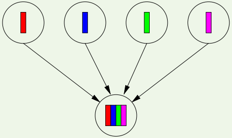
Collective Communication



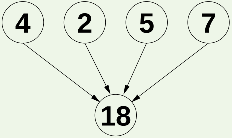
broadcast



scatter



gather



reduction

Communication with MPI

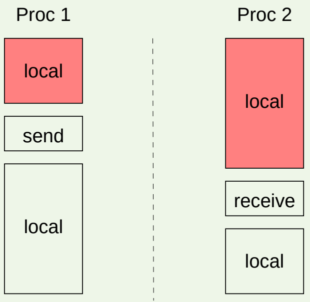
Point-to-Point Communication

Example 1: Hello world

```
char    msg[20];
int     myrank;
int     tag = 99;
MPI_Status status;

MPI_Comm_rank( MPI_COMM_WORLD, &myrank);

if (myrank == 0) {
    strcpy( msg, "Hello world!");
    MPI_Send( msg, strlen( msg) + 1, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
}
else if (myrank == 1) {
    MPI_Recv( msg, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
    printf( "%s\n", msg);

}
```

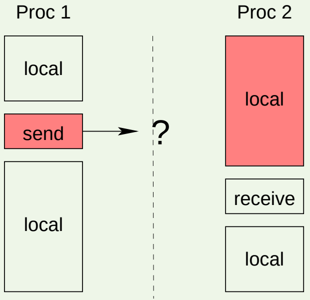
Communication with MPI

Nonbufferred Communication



Communication with MPI

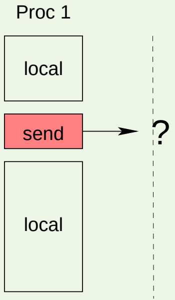
Nonbufferred Communication



1. P1 has to wait till P2 is ready, if there is no/not enough buffer.

Communication with MPI

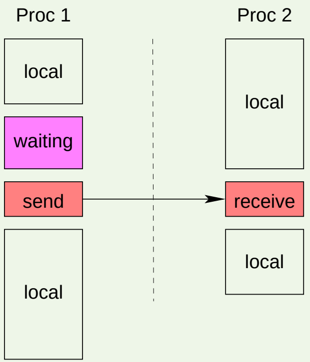
Nonbufferred Communication



1. P1 has to wait till P2 is ready, if there is no/not enough buffer.
2. P1 will not continue, P1 is blocked.

Communication with MPI

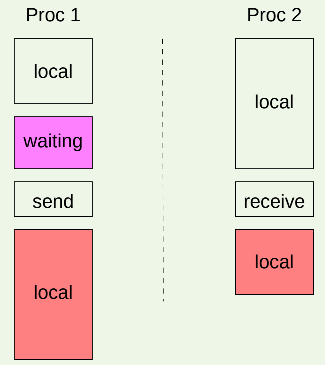
Nonbufferred Communication



1. P1 has to wait till P2 is ready, if there is no/not enough buffer.
2. P1 will not continue, P1 is blocked.

Communication with MPI

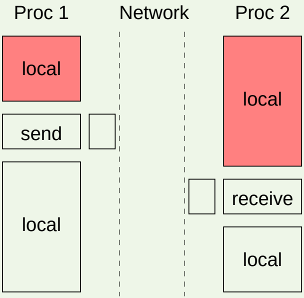
Nonbufferred Communication



1. P1 has to wait till P2 is ready, if there is no/not enough buffer.
2. P1 will not continue, P1 is blocked.

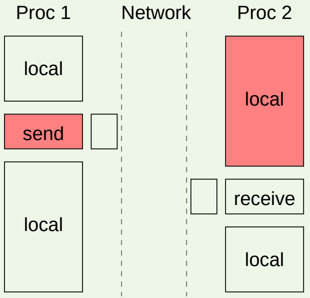
Communication with MPI

Bufferred Communication



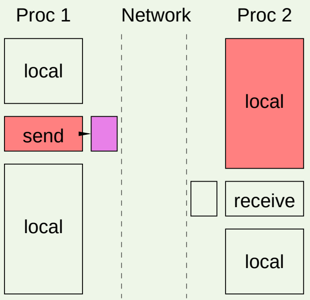
Communication with MPI

Bufferred Communication



Communication with MPI

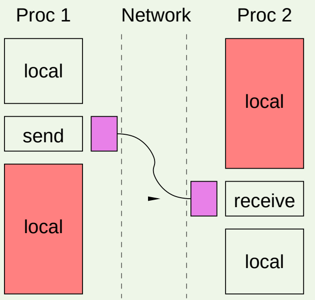
Bufferred Communication



1. P1 copies data to buffer.

Communication with MPI

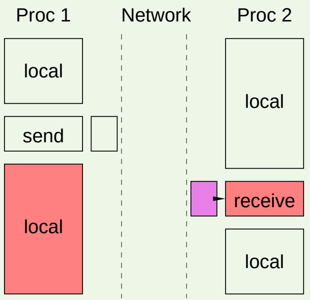
Bufferred Communication



1. P1 copies data to buffer.
2. P1 continues.

Communication with MPI

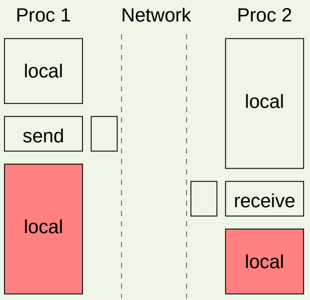
Bufferred Communication



1. P1 copies data to buffer.
2. P1 continues.
3. P2 will continue work after receiving data.

Communication with MPI

Bufferred Communication



1. P1 copies data to buffer.
2. P1 continues.
3. P2 will continue work after receiving data.

Why do messages need to be buffered ?

- MPI Send normally does not wait for corresponding `MPI_Recv`.

Why do messages need to be buffered ?

- MPI Send normally does not wait for corresponding `MPI_Recv`.
- Still, user buffer may be reclaimed, modified, etc.
  $\Rightarrow$ Message needs to be stored somewhere in the 'system'.

Why do messages need to be buffered ?

- MPI Send normally does not wait for corresponding `MPI_Recv`.
- Still, user buffer may be reclaimed, modified, etc.
  $\Rightarrow$ Message needs to be stored somewhere in the 'system'.

Problem: How large is the system buffer ?

Why do messages need to be buffered ?

- MPI Send normally does not wait for corresponding `MPI_Recv`.
- Still, user buffer may be reclaimed, modified, etc.
  $\Rightarrow$ Message needs to be stored somewhere in the 'system'.

Problem: How large is the system buffer ?

- Programs may fail due to system buffer exhaustion.

Why do messages need to be buffered ?

- MPI Send normally does not wait for corresponding `MPI_Recv`.
- Still, user buffer may be reclaimed, modified, etc.
  $\Rightarrow$ Message needs to be stored somewhere in the 'system'.

Problem: How large is the system buffer ?

- Programs may fail due to system buffer exhaustion.
- Too bad: system buffer size depends on
  - architecture,
  - operating system,
  - MPI implementation.

Why do messages need to be buffered ?

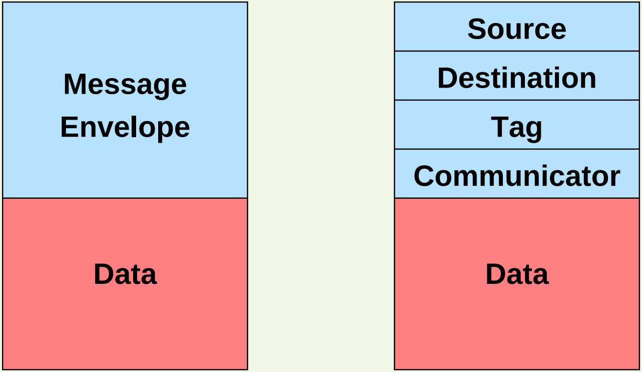
- MPI Send normally does not wait for corresponding `MPI_Recv`.
- Still, user buffer may be reclaimed, modified, etc.
  $\Rightarrow$ Message needs to be stored somewhere in the 'system'.

Problem: How large is the system buffer ?

- Programs may fail due to system buffer exhaustion.
- Too bad: system buffer size depends on
  - architecture,
  - operating system,
  - MPI implementation.
- Portability of code restricted.

Communication with MPI

What makes a Message?



Communication with MPI

MPI_Send

```
int MPI_Send( void *buffer,              /* address of send buffer      */
              int count,                 /* number of entries in buffer */
              MPI_Datatype datatype,     /* datatype of entry           */
              int destination            /* rank of destination         */
              int tag,                   /* message tag                 */
              MPI_Comm communicator      /* communicator                */
)
```

- Standard blocking send operation.
- Assembles message envelope.
- Sends message to destination.
- May return as soon as message is handed over to 'system' (buffered communication).
- May wait for corresponding receive operation (unbuffered communication).
- Buffering behaviour is implementation-dependent.
- No synchronization with receiver (guaranteed).

Communication with MPI

MPI_Recv

```
int MPI_Recv(void *buffer,             /* OUT : address of receive buffer */
             int count,                /* IN  : maximum number of entries */
             MPI_Datatype datatype,    /* IN  : datatype of entry         */
             int source                /* IN  : rank of source            */
             int tag,                  /* IN  : message tag               */
             MPI_Comm communicator,    /* IN  : communicator              */
             MPI_Status *status        /* OUT : return status             */
)
```

- Standard blocking receive operation.
- Receives message from source with tag.
- Disassembles message envelope.
- Stores message data in buffer.
- Returns not before message is received.
- Returns additional status data structure.

Communication with MPI

- Receiving messages from any source?

  Use wildcard source specification `MPI_ANY_SOURCE`.

Communication with MPI

- Receiving messages from any source?
  Use wildcard source specification `MPI_ANY_SOURCE`.

- Receiving messages with any tag?
  Use wildcard tag specification `MPI_ANY_TAG`

Communication with MPI

- Receiving messages from any source?
  Use wildcard source specification `MPI_ANY_SOURCE`.
- Receiving messages with any tag?
  Use wildcard tag specification `MPI_ANY_TAG`
- Message buffer larger than message?
  Don't worry, superfluous buffer fields remain untouched.

Communication with MPI

- Receiving messages from any source?
  Use wildcard source specification `MPI_ANY_SOURCE`.
- Receiving messages with any tag?
  Use wildcard tag specification `MPI_ANY_TAG`
- Message buffer larger than message?
  Don't worry, superfluous buffer fields remain untouched.
- Message buffer smaller than message?
  Message is truncated, no buffer overflow.
  `MPI_Recv` returns error code `MPI_ERR_TRUNCATE`.

Communication with MPI

Deadlock I

Time	Process A	Process B
1	`MPI_Send` to B, tag = 0	local work
2	`MPI_Send` to B, tag = 1	local work
3	local work	`MPI_Recv` from A, tag = 1
4	local work	`MPI_Recv` from A, tag = 0

Communication with MPI

Deadlock I

Time	Process A	Process B
1	MPI_Send to B, tag = 0	local work
2	MPI_Send to B, tag = 1	local work
3	local work	MPI_Recv from A, tag = 1
4	local work	MPI_Recv from A, tag = 0

- The program will deadlock, if system provides no buffer.

Communication with MPI

Deadlock I

Time	Process A	Process B
1	MPI_Send to B, tag $= 0$	local work
2	MPI_Send to B, tag $= 1$	local work
3	local work	MPI_Recv from A, tag $= 1$
4	local work	MPI_Recv from A, tag $= 0$

- The program will deadlock, if system provides no buffer.
- Process A is not able to send message with tag=0.

Communication with MPI

Deadlock I

Time	Process A	Process B
1	`MPI_Send` to B, tag $= 0$	local work
2	`MPI_Send` to B, tag $= 1$	local work
3	local work	`MPI_Recv` from A, tag $= 1$
4	local work	`MPI_Recv` from A, tag $= 0$

- The program will deadlock, if system provides no buffer.
- Process A is not able to send message with tag=0.
- Process B is not able to receive message with tag=1.

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

- The program will deadlock, if system provides no buffer.

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

- The program will deadlock, if system provides no buffer.
- Process A and Process B are not able to send messages.

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

- The program will deadlock, if system provides no buffer.
- Process A and Process B are not able to send messages.
- Order communications in the right way!

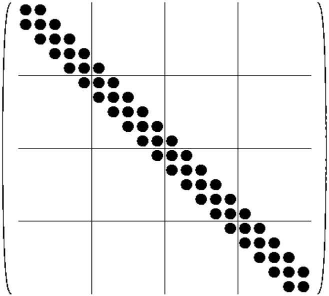
Communication with MPI

Example: Exchange of messages

```
if (myrank == 0) {
    MPI_Send( sendbuf, 20, MPI_INT, 1, tag, communicator);
    MPI_Recv( recvbuf, 20, MPI_INT, 1, tag, communicator, &status);
}
else if (myrank == 1) {
    MPI_Recv( recvbuf, 20, MPI_INT, 0, tag, communicator, &status);
    MPI_Send( sendbuf, 20, MPI_INT, 0, tag, communicator);
}
```

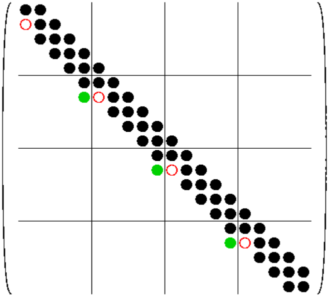
- This code succeeds even with no buffer space at all !!!
- **Important note: Code which relies on buffering is considered unsafe !!!**

How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

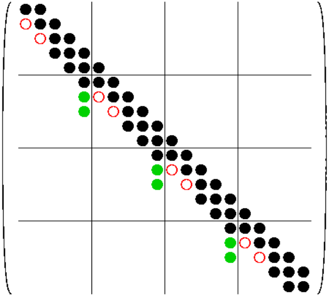
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.

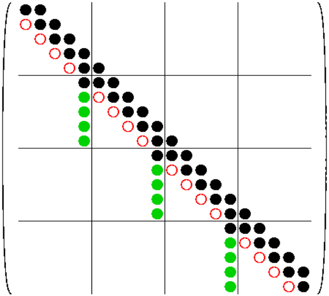
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.

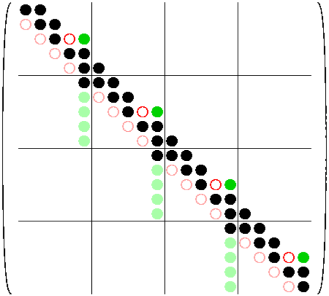
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.

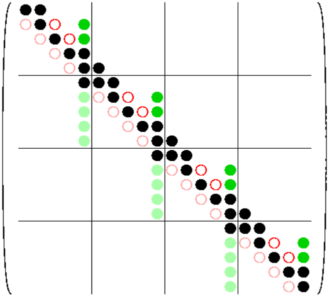
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.

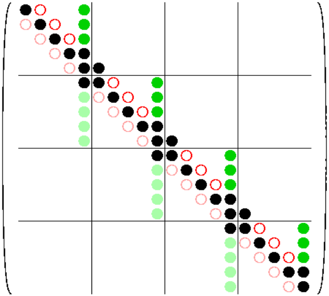
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.

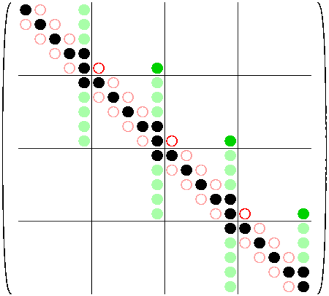
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.

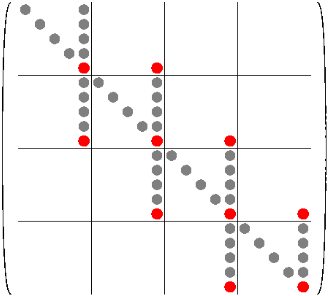
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

How to solve a tridiagonal system?

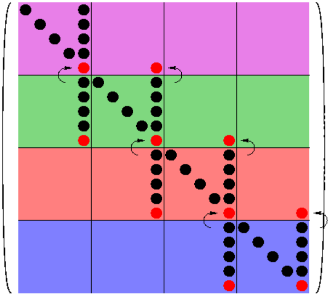


Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

Results in a tridiagonal subsystem with unknowns x_5, x_{10}, x_{15}, x_{20}.

How to solve a tridiagonal system?

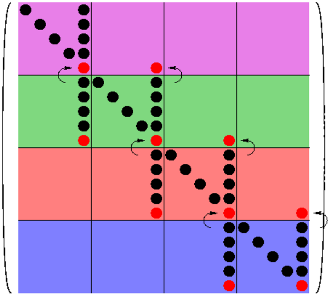


Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

Results in a tridiagonal subsystem with unknowns x_5, x_{10}, x_{15}, x_{20}.

How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

Results in a tridiagonal subsystem with unknowns x_5, x_{10}, x_{15}, x_{20}.
If data are stored rowwise only one communication to neighbouring processor neccessary.