Übung MPI

MPI Hello World in C

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
  int size;
  int rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf(“Hello, World! This is rank %d of
          %d\n",rank,size);
  MPI_Finalize();
}
// compile using mpicc…
```

Kompilieren & Starten

- Eventuell Environment setzten für MPICH
- mpicc -o pp hello_world.c
- mpirun -n 2 ./hello_world.c

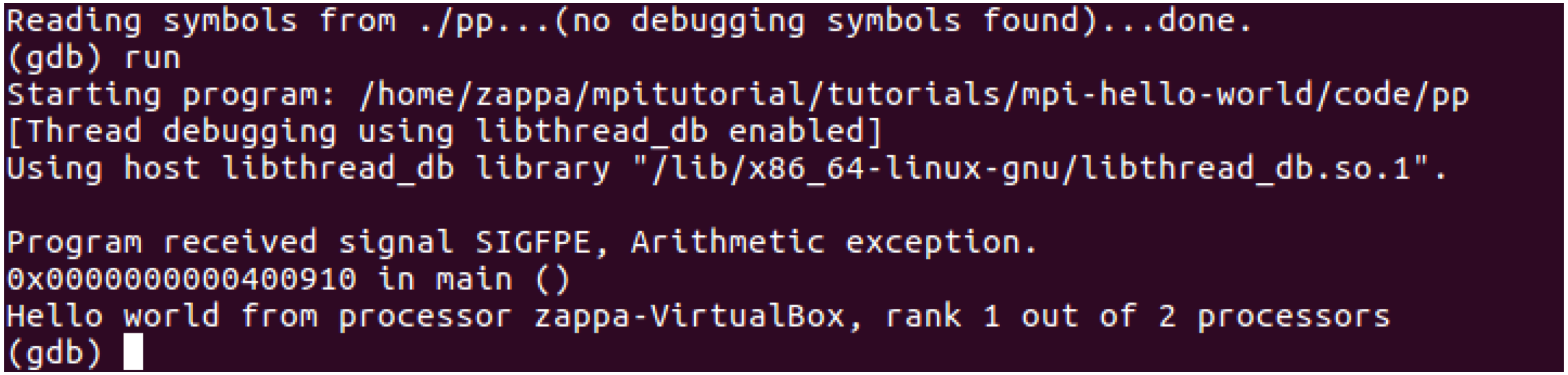
Debugging

- Mittels TotalView ... Der gdb ist im Prinzip nicht MPI-fähig
- Aber:
- Mittels mpirun ... **:** ...
- Eigentlich für Strukturen wie
- mpirun -n 1 ./master : -n 8 ./compute : -n 8 ./helper
- Zum Debuggen missbraucht als:
- mpirun –n 1 **gdb** ./hello_world : -n 1 ./hello_world

MPI Hello World in C mit Fehler

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
  int size;
  int rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf(“Hello, World! This is rank %d of
          %d\n",rank/rank,size);
  MPI_Finalize();
}
// compile using mpicc…
```

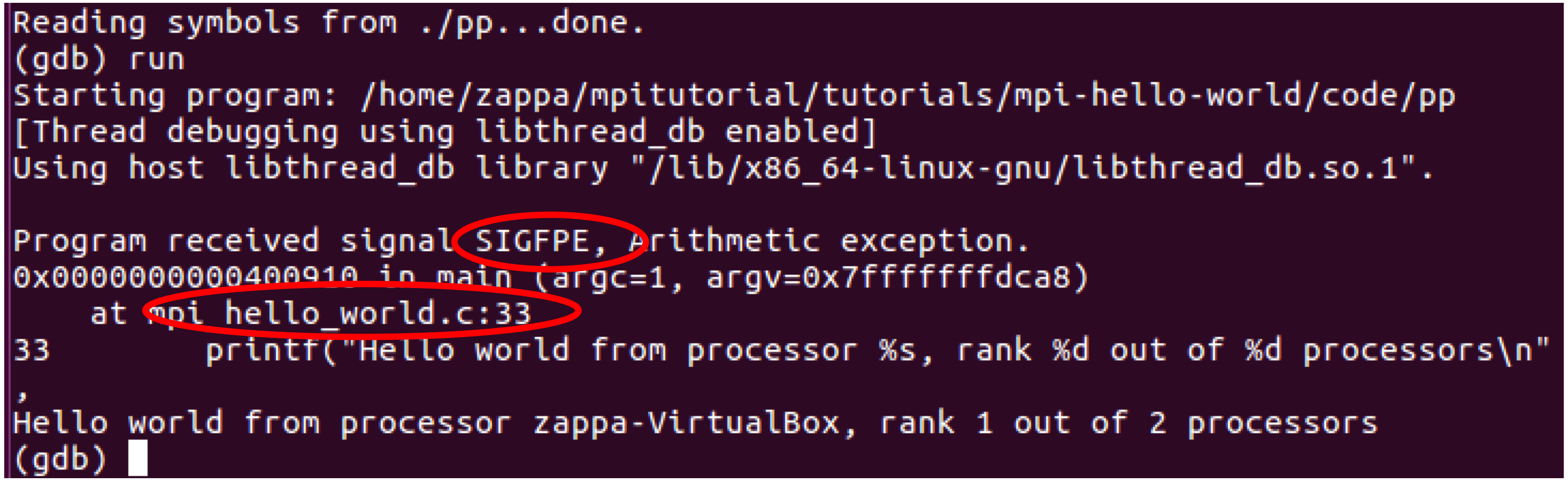
Beispiel Output



```
Reading symbols from ./pp...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/zappa/mpitutorial/tutorials/mpi-hello-world/code/pp
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGFPE, Arithmetic exception.
0x0000000000400910 in main ()
Hello world from processor zappa-VirtualBox, rank 1 out of 2 processors
(gdb) 
```

- mpicc -o hello_world hello_world.c



```
Reading symbols from ./pp...done.
(gdb) run
Starting program: /home/zappa/mpitutorial/tutorials/mpi-hello-world/code/pp
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGFPE, Arithmetic exception.
0x0000000000400910 in main (argc=1, argv=0x7fffffffdca8)
    at mpi_hello_world.c:33
33          printf("Hello world from processor %s, rank %d out of %d processors\n"
,
Hello world from processor zappa-VirtualBox, rank 1 out of 2 processors
(gdb) 
```

- mpicc **-g** -o hello_world hello_world.c

Datentypen in MPI

MPI Datatype	C Datatype
MPI_CHAR	char
MPI_SHORT	short
MPI_INT	int
MPI_LONG	long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Standard blocking send:

```
int MPI_Send(void* buffer,      // message sending buffer
                   int count,  // number of elements to send
    MPI_Datatype datatype,  // element type
           int destination,  // rank of destination process
                      int tag,  // message tag
         MPI_Comm comm);  // communicator
```

- The function is blocking
    - Buffer may be reused after return
    - That doesn‘t mean that message was recieved!
    - May block until the message is received by the destination process – implementation/message size depending

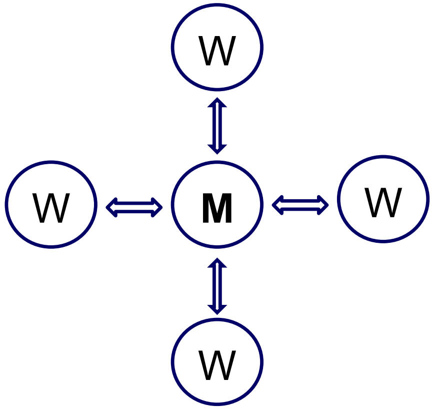
Standard Blocking Receive

```
MPI_Receive(void* buffer, // message receive buffer
                    int count, // max. number of elements to receive
  MPI_Datatype datatype, // element type
                  int source, // rank of source process
                      int tag, // message tag
        MPI_Comm comm, // communicator
       MPI_Status* status); // receive status
```

- The function is blocking
  - Message has been successfully received
- Buffer size can be larger than message size
- Number of elements in the message can be less than count

Kommunikationsbeispiel

- Classical approach to parallel programming
  - One process is a master
  - The other processes are workers
  - Master collects results from workers
- Uses only MPI_SEND and MPI_RECEIVE
- Point-to-point communication pattern



Collective Communication - Reduction

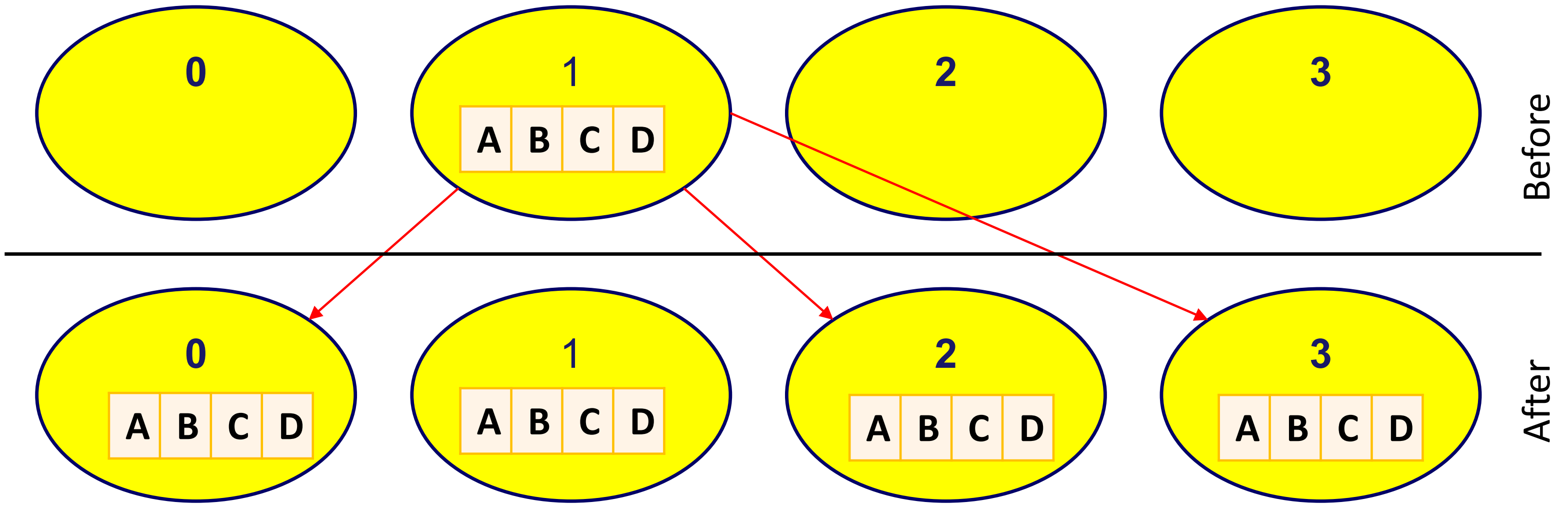
```
int MPI_Reduce(
            const void *sendbuf,
            void *recvbuf,
            int count,
            MPI_Datatype datatype,
            MPI_Op op,
            int root,
            MPI_Comm comm)
```

Operatoren

- MPI_MAX -- maxmimum
- MPI_MIN – minimum
- MPI_SUM -- sum
- MPI_PROD -- product
- MPI_LAND – logical AND
- MPI_BAND – bit-wise AND
- MPI_LOR – logical OR
- MPI_BOR – bit-wise OR
- MPI_LXOR – logical XOR
- MPI_BXOR – bit-wise XOR
- MPI_MAXLOC – maximum value and location
- MPI_MINLOC – minimum value and location

Collective Communication: Bcast

```
int MPI_Bcast(
        void *buffer,
        int count,
        MPI_Datatype datatype,
        int root,
        MPI_Comm comm);
```



MPI Hello World in C

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
  int size;
  int rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf(“Hello, World! This is rank %d of
          %d\n",rank,size);
  MPI_Finalize();
}
// compile using mpicc…
```

Summe der Quadrate mit send/recv

```
…
  if ( rank == 0 ){
    int i = 1;
    for ( i=1; i<world_size; i++){
      int dummy = 0;
      MPI_Recv(&dummy, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      sum = sum + dummy;
      printf(" received %d , new sum is %d \n",dummy,sum);
    }
  } else {
    MPI_Send( &square, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    printf(" send %d \n",square);
  }


  if ( rank == 0 ){
    sum = sum + square;
    printf("Final sum of squares from 1 to %d : %d \n", world_size, sum);
  }

  MPI_Finalize();
}
```

Summe der Quadrate mit reduce

```
…
   int square = (rank+1) * (rank+1);

  printf(" rank %d : square: %d  \n", rank, square);

  int sum = 0;

  printf("Number of prcesses: %d \n", world_size);

  MPI_Reduce(&square, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);


  if ( rank == 0){
        printf("Final sum of squares from 1 to %d : %d \n", world_size, sum);
  }

  MPI_Finalize();
}
```

Summe der Quadrate und %-Angabe mit reduce und bcast

```
…
   int square = (rank+1) * (rank+1);

  int sum = 0;

  MPI_Reduce(&square, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

  if ( rank == 0)
        printf("Final sum of squares from 1 to %d : %d \n", world_size, sum);

  printf("BEFORE Bcast rank %d : MySquare: %d Sum: %d \n", rank, square, sum);

  MPI_Bcast(&sum,1 , MPI_INT ,0 , MPI_COMM_WORLD);

  double perc = (double)square/sum;

  printf("AFTER Bcast rank %d : Percent %f \n", rank, perc);

  MPI_Finalize();
}
```