

5. SOFTWARE FÜR PARALLELE SYSTEMEN (FORTSETZUNG)

Collective Communication

Kommunikationsroutinen die alle Prozesse innerhalb eines Communicator betreffen.

z.B.: Broadcast

Alle Prozesse innerhalb des Communicators müssen die Routine mit den gleichen Parametern aufrufen.

Alle Collective Routinen blockieren

3 Kategorien von Collective Routinen

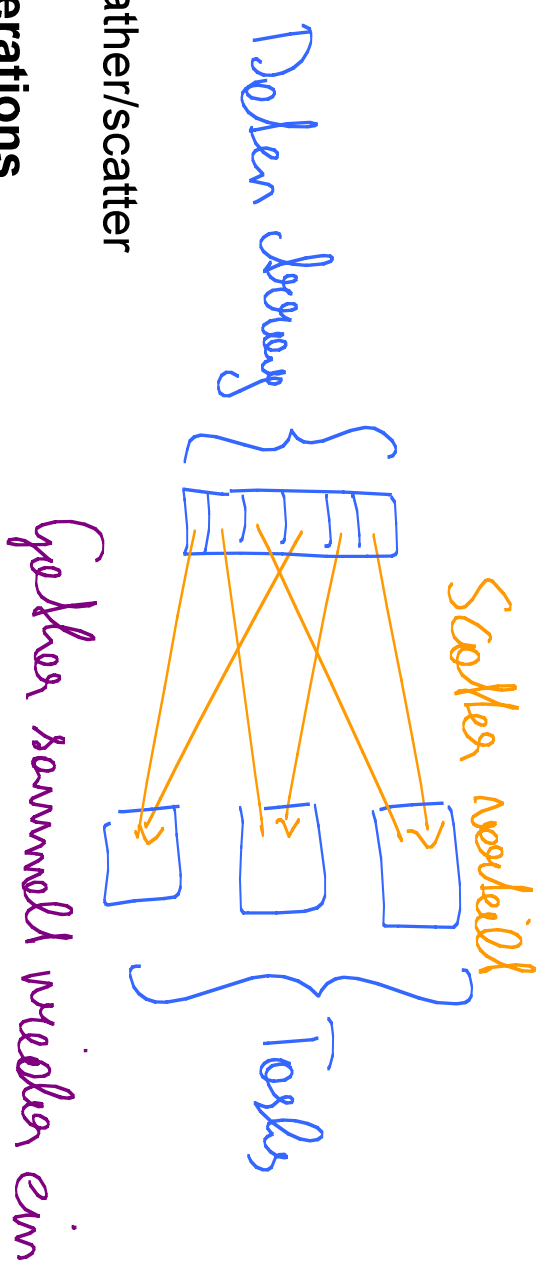
- Synchronization
- Data movement
- Global reduction operations

Synchronization

MPI_Barrier(communicator)

Data Movement

- Broadcast
- Gather
- Scatter
- Variationen von gather/scatter



Global Reduction Operations

MPI_Reduce(sendbuf,recvbuf,count,datatype,operation,root,comm)

Mögliche Operationen

- Minimum, Maximum
- Summe, Produkt
- Logische und Bitweise Verknüpfungen
- Benutzerdefiniert

Jeder Task hat ein Datenblock; über diese Operationen kann man jedem Task dieses Feld angefordert werden und

z.B.: aufsummiert werden

Beispiel: π berechnen

$$\lim_{n \rightarrow \infty} \frac{4}{n} \sum_{k=1}^n \frac{1}{1 + \left(\frac{k - \frac{1}{2}}{n}\right)^2} = \pi$$

Teilsummen parallel berechnen!

```
#include "math.h"
#include "mpi.h"

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int _tmain(int argc, _TCHAR* argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double starttime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Get_processor_name(processor_name, &namelen);
```

```

while (!done) {
    if (myid == 0) {
        fprintf(stdout, "Enter the number of intervals: (0 quits) ");
        fflush(stdout);
        if (scanf("%d", &n) != 1) {
            fprintf(stdout, "No number entered; quitting\n");
            n = 0;
        }
        starttime = MPI_Wtime();
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0)
        done = 1;
    else {
        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
            x = h * ((double) i - 0.5);
            sum += f(x);
        }
        mypi = h * sum;
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
        if (myid == 0) {
            printf("pi is approximately %.16f, Error is %.16f\n",
                pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("wall clock time = %f\n", endwtime - starttime);
            fflush(stdout);
        }
    }
}
return 0;
}
}

```

Um Programm zu starten:

```
... \MPICH2\bin\mpiexec.exe -n [#Tasks] [Executablename]
```

Jeder Rechner im MPI Verbund muss auf das Executable zugreifen koennen

zum Abbruch ctrl + c 1 x druecken, damit MPI die Proezesse auf allen Rechnern terminierne kann; bei 2 x ctrl+c bleiben Prozesse bestehen!