Parallele Programmierung mit MPI

Marc-Oliver Straub

entstanden aus:
Parallele Programmierung mit MPI - ein Praktikum

Warum Parallelprogrammierung

- *große numerische Probleme (Simulation)*
- *optische Bildverarbeitung (Post)*
- *große WWW-Server*
- *Datenbanken*

Was ist ein paralleles Programm

- *mehrere kooperierende Prozesse*
- *1 Prozess pro Prozessor*
- *Kommunikation zwischen den Prozessen über Shared Memory oder Nachrichten*

Was ist MPI

- *Bibliothek zur Parallelprogrammierung für nachrichtengekoppelte Rechner*
- *besteht aus plattformspezifischer User-Library mit plattformübergreifendem Interface*
- *vom Hersteller optimierte Bibliotheken für Cray, IBM SP, ...*

Distributionen

- *MPICH*

  http://www-unix.mcs.anl.gov/mpi/mpich/

- *LAM-MPI*

  http://www.lam-mpi.org/

- *Dokumentation*

  http://www.mpi-forum.org/

Grundlagen

- *Struktur eines MPI-Programmes:*
  - *Initialisierung: MPI_Init()*
  - *Berechnung, Kommunikation*
  - *Cleanup: MPI_Finalize()*

Ping Pong

```
#include <mpi.h>
MPI_Init();
MPI_Comm_size(&procs);
MPI_Comm_rank(&myid);
if (myid == 0)
        MPI_Send(buf, length, MPI_INT, 1, PING);
        MPI_Recv(buf, length, MPI_INT, 1, PONG, &status);
else if (myid == 1)
        MPI_Recv(buf, length, MPI_INT, 0, PING, &status);
        MPI_Send(buf, length, MPI_INT, 0, PONG);
endif

MPI_Finalize();
```

P2P-Kommunikation

- *Synchron: MPI_Ssend, MPI_Srecv*
- *Nichtblockierend: MPI_Isend, MPI_Irecv, MPI_Wait[any], MPI_Test[any]*
- *Gepuffert: MPI_Send, MPI_Recv*

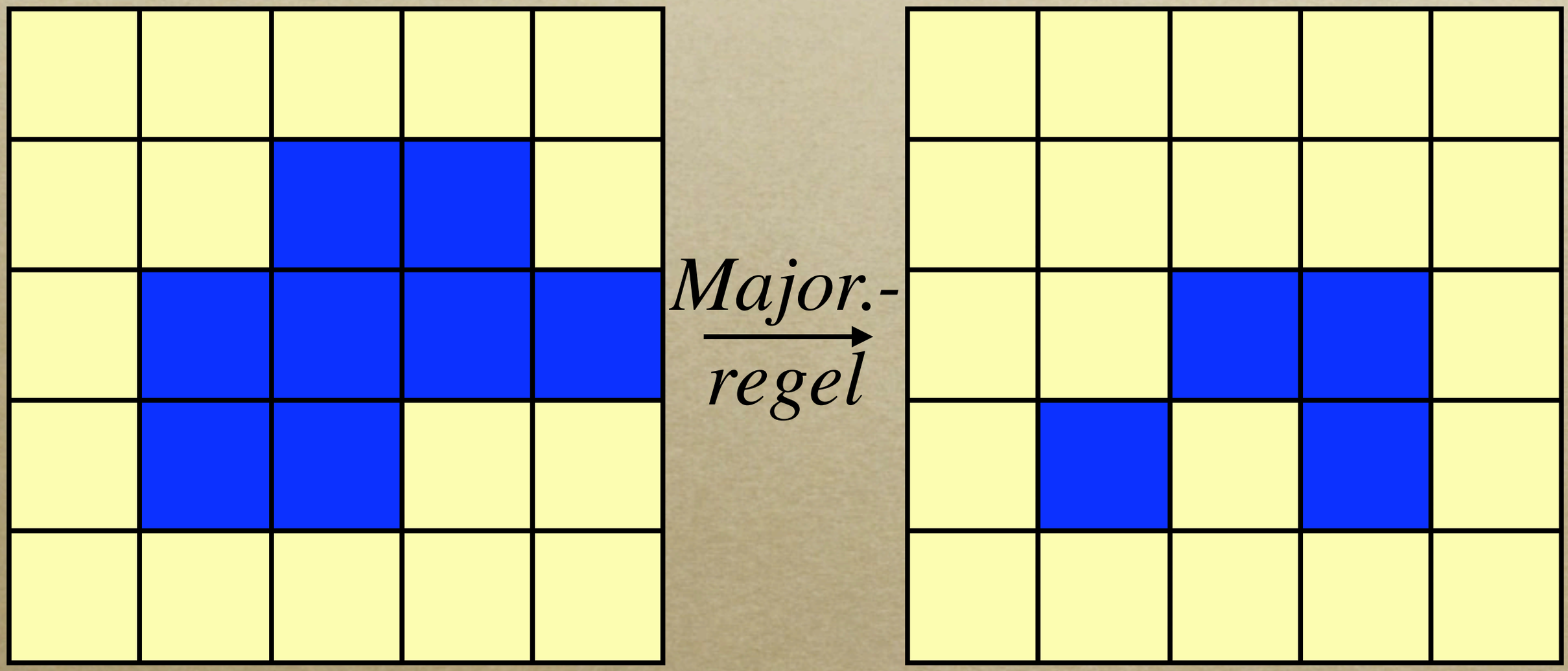
Kollektive Operationen

- *Synchronisation: MPI_Barrier*
- *Datenverteilung: MPI_Broadcast, MPI_Scatter*
- *Ergebnissammlung: MPI_Gather, MPI_Reduce*

Beispiele

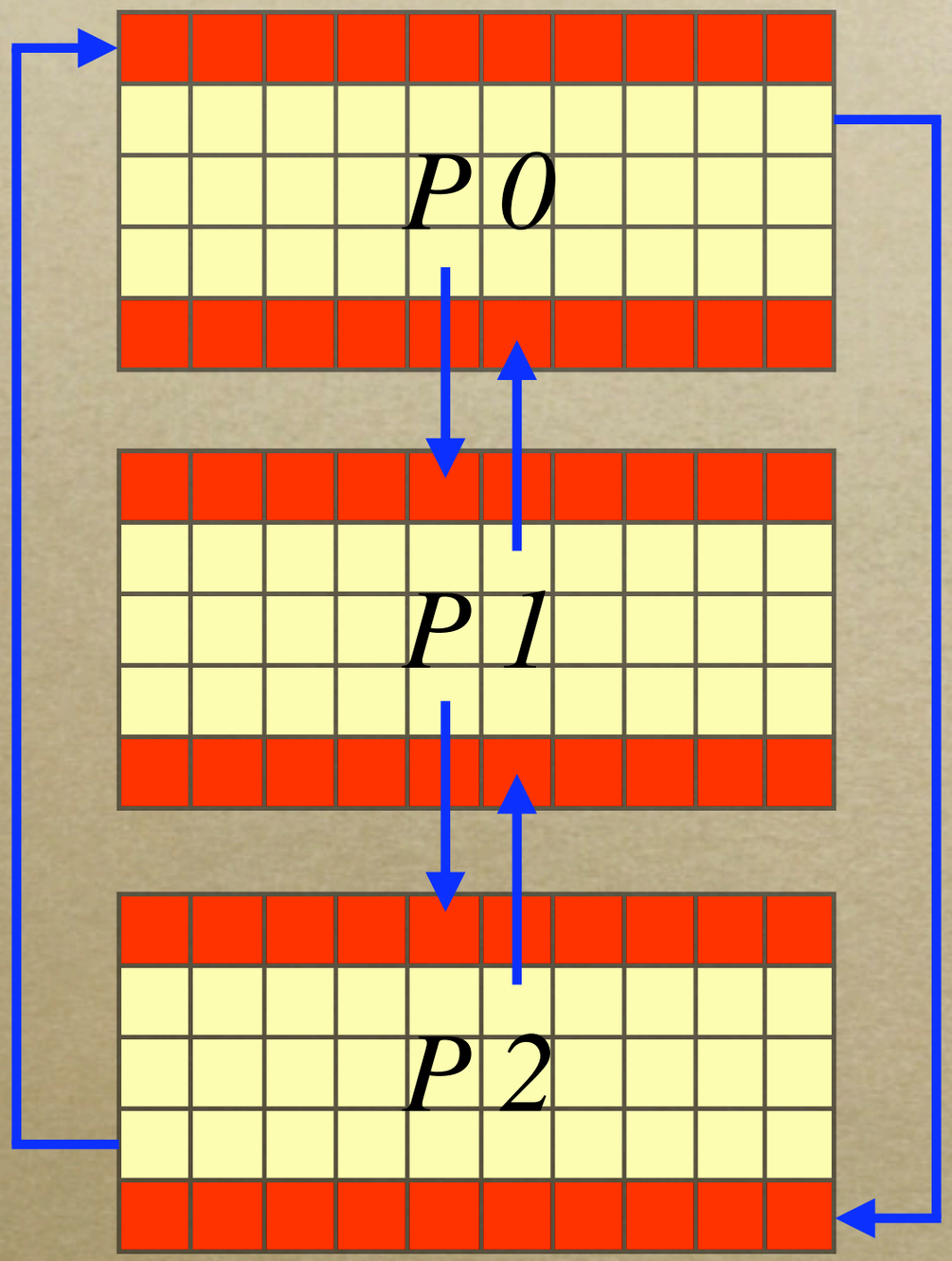
- *Simulation von Zellularautomaten*
- *Matrixmultiplikation*
- *Sortieren*
- *Master-Worker-Schema*

Simulation von ZA



n	0	1	2	3	4	5	6	7	8	9
f(n)	0	0	0	0	1	0	1	1	1	1

1D-Problemzerlegung



1D-Algorithmus

```
MPI_Comm_size(&procs);
MPI_Comm_rank(&myid);
int prev = (myid-1) % procs; int next = (myid+1) % procs;

while(...)
        MPI_Isend(buflineO+1, rowlen, MPI_INT, prev, U_ROW);
        MPI_Irecv(buflinen+1, rowlen, MPI_INT, next, U_ROW);
        MPI_Isend(buflinen, rowlen, MPI_INT, next, L_ROW);
        MPI_Recv(buflineO, rowlen, MPI_INT, prev, L_ROW);
        compute();
endwhile;
```

Verdeckung von Kommunikation

```
MPI_Comm_size(&procs);
MPI_Comm_rank(&myid);
int prev = (myid-1) % procs; int next = (myid+1) % procs;

while(...)
    compute_first_and_last_line();
    MPI_Isend(bufline0+1, rowlen, MPI_INT, prev, U_ROW);
    MPI_Isend(buflinen, rowlen, MPI_INT, next, L_ROW);
    MPI_Irecv(newlinen+1, rowlen, MPI_INT, next, U_ROW, &r1);
    MPI_Irecv(newline0, rowlen, MPI_INT, prev, L_ROW, &r2);
    compute_remaining_lines();
    MPI_Waitall(2, r1, r2); switch_buffers();
endwhile;
```

Weitere Optimierungen

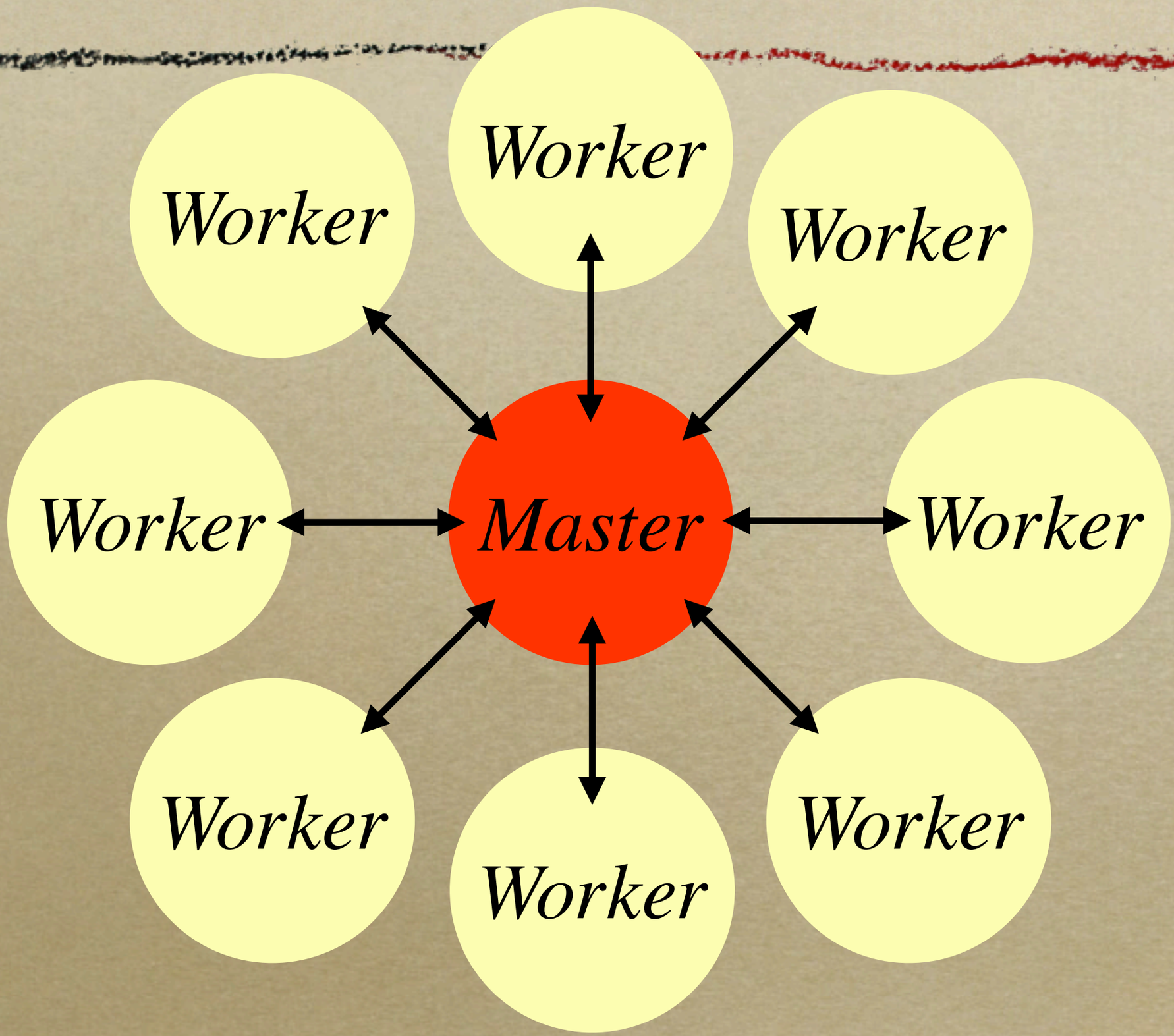
Kommunikation ist langsamer als Berechnung -> Kommunikationsaufwand verringern:

- *nicht nur eine Zeile übertragen, sondern k -> Kommunikation nur alle k Schritte*
- *2D-Quadratzerlegung -> jeder Prozessor erhält sqrt(n) Elemente*

Master-Worker Schema

- *Master - Aufgabenverteilung*
- *Worker - Rechenknechte*

Master-Worker



Worker-Code

```
req_buf = WORK_REQUEST;

while(...)
        MPI_Sendrecv(req_buf, req_len, MPI_INT, 0, recv_buf,
                                recv_len, MPI_INT, 0);
        if (recv_buf == GOT_WORK)
                process_work(recv_buf, &result_buf);
                MPI_Send(result_buf, result_len, MPI_INT, 0);
        else
                break;
        fi;
od;
```

Master-Code

```
while(some_work_left())
        MPI_Recv(req_buf, req_len, MPI_INT,
                        MPI_ANY_SOURCE, &sender);
        if (req_buf == REQUEST_WORK)
                find_some_work( &work_buf);
                MPI_Send(work_buf, len, MPI_INT, sender,
                                WORK_TAG);
        else if (req_buf == RESULT)
                extract_result(req_buf);
        fi;
od;
```

Sortieren

- *N Elemente liegen unsortiert auf den PEs*
- *jedes PE hat n=N/P viele Elemente*
- *Nach dem Sortieren soll jedes PE ≈ n Elemente in sortierter Reihenfolge besitzen*

Eingangs-daten

19	7	12
1	11	13
25	4	2

19	7	12
1	9	13
25	4	2

19	7	12
1	9	13
25	4	2

Samples

7	13	25

7	13	25

7	13	25

sortierte Samples

6	7	10	13	17	18	20	21	25

Pivotelemente

-∞	10	18	∞

Klassifi-zierung

I_0	I_1	I_2
1	11	25
4	12	19
2	13	
7		

I_0	I_1	I_2
6	17	30
10	13	27
	11	22
	16	

I_0	I_1	I_2
3	14	20
5	18	21
8	16	
	15	

Algorithmus

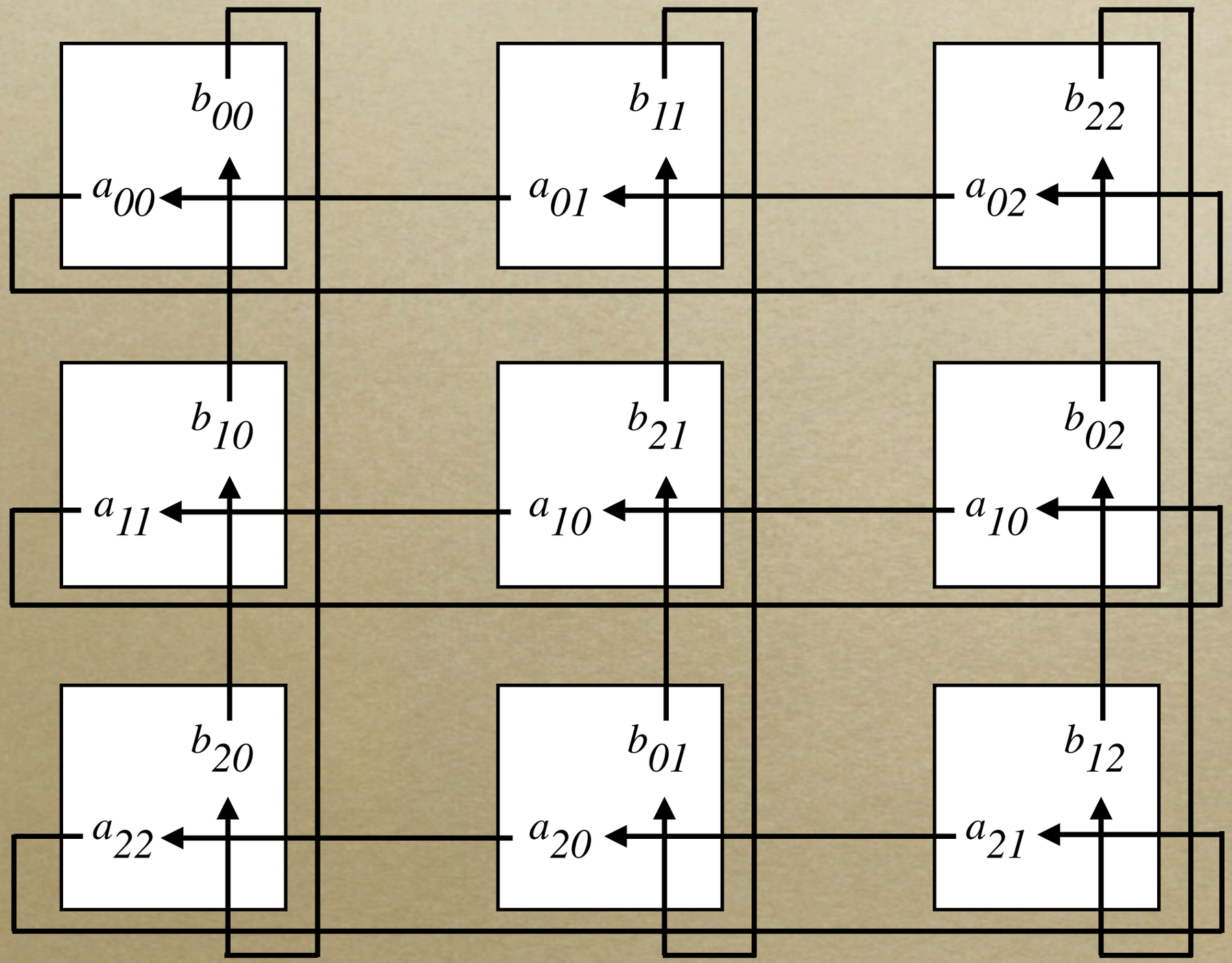
```
select_random_samples();
MPI_Gather(sample_buf, sample_count, &recv_buf, 0);
if (myid == 0)
        sort_locally(recv_buf, &sorted_buf);
        select_pivot_elements(sorted_buf, pivot_count, &pivot_buf);
fi;
MPI_Broadcast(pivot_buf, pivot_count, 0);

sort_according_to_pivots(buf, &pivot_buf, &boundary_indices);
broadcast_item_counts(boundary_indices, &recv_counts);
MPI_Allgatherv(buf, boundary_indices, &recv_buf, recv_counts,
recv_displacements);
```

Matrixmultiplikation

- *Parallele Multiplikation von Matrizen:*
- $A * B = C$
- *Anfangspermutation der Teilmatrizen, danach einfaches Verschieben*

Anfangspermutation



Algorithmus

```
permute_matrices(&A, &B);
compute_neighbours(&top, &left, &bot, &right);
while(...)
    MPI_Isend(A, n, MPI_DOUBLE, left, A_MATRIX);
    MPI_Isend(B, n, MPI_DOUBLE, top, B_MATRIX);
    MPI_Irecv(newA, n, MPI_DOUBLE, right, A_MATRIX, &r1);
    MPI_Irecv(newB, n, MPI_DOUBLE, bot, B_MATRIX, &r2);
    mat_mult(A, B, C);
    MPI_Waitall(2, r1, r2);
    A = newA; B = newB;
od;
MPI_Gather(C, n, MPI_DOUBLE, gesamtC, n, 0);
```

Zusätzliche Funktionen

- *MPI_Barrier: erst, wenn alle PEs diesen Aufruf getätigt haben, wird das Programm fortgesetzt -> Synchronisation*
- *MPI_Reduce: Einsammeln von Werten durch Ausführung einer binären Operation, z.B. Summe, Maximum, ...*

Weiterführende Konzepte

- *Typkonversion in heterogenen Netzen*
- *Benutzerdefinierte Datentypen*
- *Kommunikatoren*
- *Virtuelle Topologien*

Interessante Zahlen

- *gibts unter der Adresse:*

  http://liinwww.ira.uka.de/~skampi/

  (Vergleich von IBM-SP, Cray T3E,...)