

High Performance Computing und Paralleles Rechnen

Dr. Momin Ahmad, Marcel Weichel

HS Karlsruhe

27-11-2023



Teil I

Advanced MPI



Inhaltsverzeichnis I

1 Motivation

2 Communicator/Groups

- Groups
- Communicators

3 Virtuelle Topologien

- Aufgabe von Topologien
- Kartesische Topologie
- MPI Funktionen für Topologie



Inhaltsverzeichnis II

4 MPI-IO

- Standard IO
- Lesen und Schreiben mit MPI-IO
- Views in MPI-IO
- Kleiner Exkurs: selbstdefinierte MPI-Typen
- Kollektives MPI-IO
- Kleiner Exkurs: Aufbau parallele Dateisystem
- Hints for faster IO

5 Grenzen von MPI

6 Abschluss



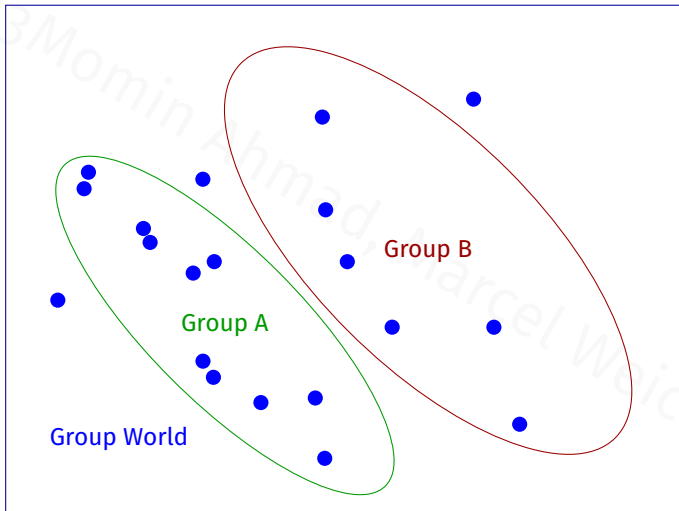
Motivation

- Verwalten von Prozessen anhand von Topologien
- Paralleles Speichern/Lesen von Daten



Communicator/Groups

Groups and Communicators





Groups, Context, Communicators

Group Gruppe von Prozessen

Context Kommunikationskontext (Eigenschaft eines Communicators)

Communicator Führt eine Gruppe und einen Context zusammen



Group

MPI Group

“A **group** is an ordered set of process identifiers (henceforth processes); processes are implementation-dependent objects. Each process in a group is associated with an integer rank. Ranks are contiguous and start from zero. Groups are represented by opaque group objects, and hence cannot be directly transferred from one process to another. A group is used within a communicator to describe the participants in a communication universe and to rank such participants.” [For12, S. 226]



Context

MPI Context

“A **context** is a property of communicators that allows partitioning of the communication space. A message sent in one context cannot be received in another context. Furthermore, where permitted, collective operations are independent of pending point-to-point operations. Contexts are not explicit MPI objects; they appear only as part of the realization of communicators.” [For12, S. 226]



Communicator

MPI Intra-Communicator

“Intra-communicators bring together the concepts of group and context. To support implementation-specific optimizations, and application topologies, communicators may also cache additional information. MPI communication operations reference communicators to determine the scope and the communication universe in which a point-to-point or collective operation is to operate. Each communicator contains a group of valid participants; this group always includes the local process. The source and destination of a message is identified by process rank within that group. For collective communication, the intra-communicator specifies the set of processes that participate in the collective operation (and their order, when significant). Thus, the communicator restricts the spatial scope of communication, and provides machine-independent process addressing through ranks. Intra-communicators are represented by opaque intra-communicator objects, and hence cannot be directly transferred from one process to another” [For12, S. 227]

Anmerkung: Inter-Kommunikatoren (für Kommunikation zwischen Gruppen) hier nicht behandelt.



Groups

MPI_Group_size

Aufgabe

Gibt die Anzahl an Prozessen in der Gruppe zurück

Signatur

```
1 int MPI_Group_size(  
2     MPI_Group group,    //[in] group  
3     int *size           //[out] number of processes in the group  
4 );
```



MPI_Group_rank

Aufgabe

Gibt die rank ID des Prozesses in der Gruppe zurück

Signatur

```
1 int MPI_Group_rank(  
2     MPI_Group group,    //[in] group  
3     int *rank           //[out] rank of the calling process in group , or MPI_UNDEFINED if the process is  
4     not a member  
5 );
```

MPI_Group_incl

Aufgabe

Erzeugt eine neue Gruppe basierend auf der bestehenden Gruppe mit den rank IDs aus der übergebenen Liste

Signatur

```
1  int MPI_Group_incl(  
2      MPI_Group group,      //[in] group  
3      int n,                //[in] number of elements in array ranks  
4      int *ranks,           //[in] ranks of processes in group to appear in newgroup  
5      MPI_Group *newgroup  //[out] new group derived from above, in the order defined by ranks  
6  );
```

MPI_Group_excl

Aufgabe

Erzeugt eine neue Gruppe basierend auf der bestehenden Gruppe ohne die rank IDs aus der übergebenen Liste

Signatur

```
1 int MPI_Group_excl(  
2     MPI_Group group,      //[in] group  
3     int n,               //[in] number of elements in array ranks  
4     int *ranks,          //[in] array of integer ranks in group not to appear in newgroup  
5     MPI_Group *newgroup //[out] new group derived from above, preserving the order defined by group  
6 );
```


MPI_Group_range_incl

Aufgabe

Erzeugt eine neue Gruppe basierend auf der bestehenden Gruppe mit den Bereichen von rank IDs aus der Liste

Signatur

```
1 int MPI_Group_range_incl(  
2     MPI_Group group,      //[in] group  
3     int n,                //[in] number of triplets in array ranges  
4     int ranges[][3],      //[in] a one - dimensional array of integer triplets , of the form ( first rank ,  
6     last rank , stride ) indicating ranks in group or processes to be included in newgroup .  
5     MPI_Group *newgroup  //[out] new group derived from above , in the order defined by ranges  
6 );
```

MPI_Group_range_excl

Aufgabe

Erzeugt eine neue Gruppe basierend auf der bestehenden Gruppe ohne die Bereiche von rank IDs aus der Liste

Signatur

```
1 int MPI_Group_range_excl(  
2     MPI_Group group,      //[in] group  
3     int n,               //[in] number of triplets in array ranges  
4     int ranges[][3],      //[in] a one - dimensional array of integer triplets of the form ( first rank ,  
                           last rank , stride ), indicating the ranks in group of processes to be excluded from the output group  
                           newgroup .  
5     MPI_Group *newgroup  //[out] new group derived from above , preserving the order in group  
6 );
```

MPI_Group_intersection

Aufgabe

Erzeugt eine neue Gruppe durch Schnitt der beiden Gruppen

Signatur

```
1 int MPI_Group_intersection(  
2     MPI_Group group1,    //[in] first  group  
3     MPI_Group group2,    //[in] second group  
4     MPI_Group *newgroup  //[out] intersection group  
5 );
```

MPI_Group_difference

Aufgabe

Erzeugt eine neue Gruppe durch Differenz der beiden Gruppen

Signatur

```
1 int MPI_Group_difference(  
2     MPI_Group group1,    //[in] first   group  
3     MPI_Group group2,    //[in] second group  
4     MPI_Group *newgroup  //[out] difference group  
5 );
```

MPI_Group_union

Aufgabe

Erzeugt eine neue Gruppe durch Vereinigung der beiden Gruppen

Signatur

```
1 int MPI_Group_union(  
2     MPI_Group group1,    //[in] first  group  
3     MPI_Group group2,    //[in] second group  
4     MPI_Group *newgroup  //[out] union  group  
5 );
```

MPI_Group_compare

Aufgabe

Vergleich von zwei Gruppen auf Gleichheit

Signatur

```
1 int MPI_Group_compare(  
2     MPI_Group group1, //[in] group1  
3     MPI_Group group2, //[in] group2  
4     int *result        //[out] integer which is MPI_IDENT if the order and members of the two groups are  
5     the same , MPI_SIMILAR if only the members are the same , and MPI_UNEQUAL otherwise  
6 );
```



MPI_Group_free

Aufgabe

Löschen einer Gruppe

Signatur

```
1 int MPI_Group_free(  
2     MPI_Group *group    //[in] group to free  
3 );
```

MPI_Group_translate_ranks

Aufgabe

Übersetzt die rank IDs von einer Gruppe in die rank IDs der anderen Gruppe

Signatur

```
1 int MPI_Group_translate_ranks(  
2     MPI_Group group1,    //[in] group1  
3     int n,               //[in] number of ranks in ranks1 and ranks2 arrays  
4     int *ranks1,         //[in] array of zero or more valid ranks in group1  
5     MPI_Group group2,    //[in] group2  
6     int *ranks2          //[out] array of corresponding ranks in group2 , MPI_UNDEFINED when no  
                           correspondence exists .  
7 );
```




Communicators



MPI_Comm_rank

Aufgabe

Gibt die rank ID des Prozesses im Communicator zurück

Signatur

```
1 int MPI_Comm_rank(  
2     MPI_Comm comm,    //[in] communicator  
3     int *rank         //[out] rank of the calling process in the group of comm  
4 );
```

MPI_Comm_size

Aufgabe

Gibt die Anzahl an Prozessen im Communicator zurück

Signatur

```
1 int MPI_Comm_size(  
2     MPI_Comm comm,    //[in]  communicator  
3     int *size         //[out] number of processes in the group of comm  
4 );
```



MPI_Comm_create

Aufgabe

Erzeugt einen neuen Communicator

Signatur

```
1 int MPI_Comm_create(  
2     MPI_Comm comm,      //[in]  communicator  
3     MPI_Group group,    //[in]  group ,  which is a subset of the group of comm  
4     MPI_Comm *newcomm  //[out] new   communicator  
5 );
```

MPI_Comm_dup

Aufgabe

Dupliziert einen Communicator

Signatur

```
1 int MPI_Comm_dup(  
2     MPI_Comm comm,      //[in]  Communicator to be duplicated  
3     MPI_Comm *newcomm  //[out] A new communicator over the same group as comm but with a new context .  
4 );
```

MPI_Comm_group

Aufgabe

Gibt die Gruppe auf dem der Communicator basiert zurück

Signatur

```
1 int MPI_Comm_group(  
2     MPI_Comm comm,    //[in]  Communicator  
3     MPI_Group *group  //[out] Group in communicator  
4 );
```

MPI_Comm_set_name

Aufgabe

Setzt den Namen für einen Communicator

Signatur

```
1 int MPI_Comm_set_name(  
2     MPI_Comm comm,    //[in] communicator to name  
3     char *comm_name  //[in] Name for communicator  
4 );
```

MPI_Comm_get_name

Aufgabe

Gibt den Namen des Communicators zurück

Signatur

```
1 int MPI_Comm_get_name(  
2     MPI_Comm comm,    //[in]  Communicator to get name of ( handle )  
3     char *comm_name,  //[out] On output , contains the name of the communicator . It must be an array of  
4     size at least MPI_MAX_OBJECT_NAME .  
5     int *resultlen    //[out] Number of characters in name  
6 );
```




MPI_Comm_free

Aufgabe

Löscht den Communicator

Signatur

```
1 int MPI_Comm_free(  
2     MPI_Comm *comm //[in] Communicator to be destroyed  
3 );
```



Vordefinierte Gruppen und Communicatoren

MPI_GROUP_EMPTY Gruppe ohne Prozesse

MPI_COMM_WORLD Communicator mit allen Prozessen

MPI_COMM_SELF Communicator mit nur dem eigenen Prozess

MPI_GROUP_NULL Wert für ungültige Gruppe

MPI_COMM_NULL Wert für ungültigen Communicator

MPI Communicator Example

```
1 <MPI Basis::init ?? >
2 int me, count, count2;
3 void *send_buf, *recv_buf, *send_buf2, *recv_buf2;
4 MPI_Group mpi_group_world, grprem;
5 MPI_Comm commrem;
6 static int ranks[] = {0};
7
8 MPI_Comm_group(MPI_COMM_WORLD, &mpi_group_world);
9 MPI_Comm_rank(MPI_COMM_WORLD, &me); /* local */
10 MPI_Group_excl(mpi_group_world, 1, ranks, &grprem); /* local */
11 MPI_Comm_create(MPI_COMM_WORLD, grprem, &commrem);
12 if (me != 0) {
13     /* compute on worker */
14     // ...
15     MPI_Reduce(send_buf, recv_buf, count, MPI_INT, MPI_SUM, 1, commrem);
16     // ...
17     MPI_Comm_free(&commrem);
18 }
19 /* zero falls through immediately to this reduce , others do later ... */
20 MPI_Reduce(send_buf2, recv_buf2, count2,
21 MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
22 MPI_Group_free(&mpi_group_world);
23 MPI_Group_free(&grprem);
24 <MPI Basis::deinit ?? >
```

Quelle: [For12, S. 251f.]

MPI Communicator Example I

```
1 <MPI Basis::init ?? >
2 int ma, mb;
3 MPI_Group mpi_group_world, group_a, group_b;
4 MPI_Comm comm_a, comm_b;
5 static int list_a[] = {0, 1};
6 #if defined(EXAMPLE_2B) || defined(EXAMPLE_2C)
7 static int list_b[] = {0, 2, 3};
8 #else /* EXAMPLE_2A */
9 static int list_b[] = {0, 2};
10 #endif
11 int size_list_a = sizeof(list_a)/sizeof(int);
12 int size_list_b = sizeof(list_b)/sizeof(int);
13
14 MPI_Comm_group(MPI_COMM_WORLD, &mpi_group_world);
15 MPI_Group_incl(mpi_group_world, size_list_a, list_a, &group_a);
16 MPI_Group_incl(mpi_group_world, size_list_b, list_b, &group_b);
17 MPI_Comm_create(MPI_COMM_WORLD, group_a, &comm_a);
18 MPI_Comm_create(MPI_COMM_WORLD, group_b, &comm_b);
19 if(comm_a != MPI_COMM_NULL)
20     MPI_Comm_rank(comm_a, &ma);
21 if(comm_b != MPI_COMM_NULL)
```

MPI Communicator Example II

```
22 MPI_Comm_rank(comm_b, &mb);
23 if(comm_a != MPI_COMM_NULL)
24 lib_call(comm_a);
25 if(comm_b != MPI_COMM_NULL) {
26     lib_call(comm_b);
27     lib_call(comm_b);
28 }
29 if(comm_a != MPI_COMM_NULL)
30     MPI_Comm_free(&comm_a);
31 if(comm_b != MPI_COMM_NULL)
32     MPI_Comm_free(&comm_b);
33 MPI_Group_free(&group_a);
34 MPI_Group_free(&group_b);
35 MPI_Group_free(&mpi_group_world);
36 <MPI Basis::deinit ?? >
```



Virtuelle Topologien



Virtual Topologies

- Prozesse werden auf eine Topologie/Struktur abgebildet
 - Gitter/Kartesische Topologie
 - Ring
 - Graph
- Prozesse könnten entsprechend dem Problem angeordnet werden
- müssen nicht/können aber mit der Topologie der Hardware/Netzwerk übereinstimmen
- Erzeugen von Unter-Topologien
- Bestimmen von Nachbarn
- Virtuelle Topologien sind Attribute des Communicators

Kartesische Topologie

10 (0,2)	11 (1,2)	12 (2,2)	13 (3,2)	14 (4,2)
5 (0,1)	6 (1,1)	7 (2,1)	8 (3,1)	9 (4,1)
0 (0,0)	1 (1,0)	2 (2,0)	3 (3,0)	4 (4,0)

MPI_Cart_create

Aufgabe

Erzeugt einen neuen Communicator basierend auf der übergebenen kartesischen Topologie

Signatur

```
1 int MPI_Cart_create(  
2     MPI_Comm comm_old,    //[in] input    communicator  
3     int ndims,            //[in] number of dimensions of cartesian grid  
4     int *dims,            //[in] integer array of size ndims specifying the number of processes in each  
                           dimension  
5     int *periods,         //[in] logical array of size ndims specifying whether the grid is periodic (  
                           true ) or not ( false ) in each dimension  
6     int reorder,          //[in] ranking may be reordered ( true ) or not ( false )  
7     MPI_Comm *comm_cart  //[out] communicator with new cartesian topology  
8 );
```

Anmerkung: Siehe auch 2D Gebietszerlegung

MPI_Cart_rank

Aufgabe

Gibt die rank ID für die übergebenen Koordinaten in der kartesischen Topologie zurück

Signatur

```
1 int MPI_Cart_rank(  
2     MPI_Comm comm, //[in] communicator with cartesian structure  
3     int *coords, //[in] integer array (of size ndims, the number of dimensions of the Cartesian  
4     topology associated with comm) specifying the cartesian coordinates of a process  
5     int *rank //[out] rank of specified process  
6 );
```

Anmerkung: Bei nicht periodischen Rändern muss "coords" im Gitter liegen

MPI_Cart_coords

Aufgabe

Gibt die Koordinaten für eine rank ID in der kartesischen Topologie zurück

Signatur

```
1 int MPI_Cart_coords(  
2     MPI_Comm comm, //[in] communicator with cartesian structure  
3     int rank, //[in] rank of a process within group of comm  
4     int maxdims, //[in] length of vector coords in the calling program  
5     int *coords //[out] integer array (of size ndims) containing the Cartesian coordinates of  
        specified process  
6 );
```

MPI_Cart_get

Aufgabe

Gibt die kartesischen Topologie-Informationen zurück

Signatur

```
1 int MPI_Cart_get(  
2     MPI_Comm comm, //[in] communicator with cartesian structure  
3     int maxdims, //[in] length of vectors dims, periods, and coords in the calling program  
4     int *dims, //[out] number of processes for each cartesian dimension  
5     int *periods, //[out] periodicity (true / false) for each cartesian dimension  
6     int *coords //[out] coordinates of calling process in cartesian structure  
7 );
```

MPI_Cart_shift

Aufgabe

Gibt die rank ID des benachbarten Prozess in der kartesischen Topologie zurück

Signatur

```
1 int MPI_Cart_shift(  
2     MPI_Comm comm, //[in] communicator with cartesian structure  
3     int direction, //[in] coordinate dimension of shift  
4     int displ, //[in] displacement (> 0: upwards shift, < 0: downwards shift)  
5     int *source, //[out] rank of source process  
6     int *dest //[out] rank of destination process  
7 );
```

Anmerkung: Source ist der Rank in positive und Destination der in negative Shift-Richtung.

MPI_Cart_sub

Aufgabe

Erzeugt einen neuen Communicator in dem die kartesischen Topologie um eine oder mehrere Dimensionen reduziert wird

Signatur

```
1 int MPI_Cart_sub(  
2     MPI_Comm comm,      //[in] communicator with cartesian structure  
3     int *remain_dims,   //[in] the ith entry of remain_dims specifies whether the ith dimension is kept  
4     MPI_Comm *newcomm  //[out] communicator containing the subgrid that includes the calling process  
5 );
```

Kartesische Untertopologien

10 (0,2)	11 (1,2)	12 (2,2)	13 (3,2)	14 (4,2)
5 (0,1)	6 (1,1)	7 (2,1)	8 (3,1)	9 (4,1)
0 (0,0)	1 (1,0)	2 (2,0)	3 (3,0)	4 (4,0)

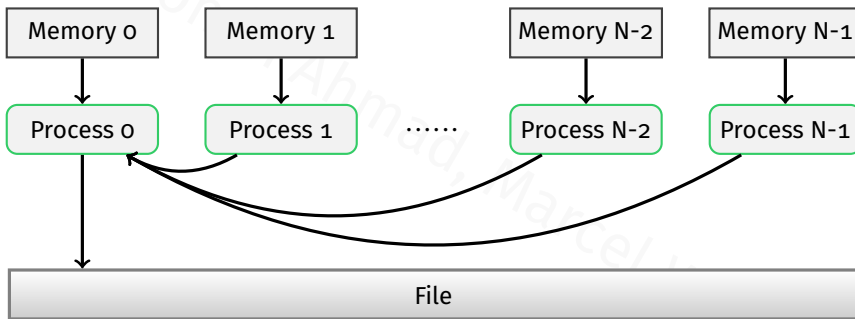


MPI-IO



Standard IO

Datei für alle Prozesse



I/O über Master-Prozess

```
1  /* example of sequential Uniz write into a common file */
2  <MPI Basis::init ?? >
3      #define BUFSIZE 100
4      int i, buf[BUFSIZE];
5      MPI_Status status;
6      FILE *myfile;
7
8      for (i=0; i<BUFSIZE; i++)
9          buf[i] = myrank * BUFSIZE + i;
10     if (myrank != 0)
11         MPI_Send(buf, BUFSIZE, MPI_INT, 0, 99, MPI_COMM_WORLD);
12     else {
13         myfile = fopen("testfile", "w");
14         fwrite(buf, sizeof(int), BUFSIZE, myfile);
15         for (i=1; i<numprocs; i++) {
16             MPI_Recv(buf, BUFSIZE, MPI_INT, i, 99, MPI_COMM_WORLD, &status);
17             fwrite(buf, sizeof(int), BUFSIZE, myfile);
18         }
19         fclose(myfile);
20     }
21 <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013

Datei für alle Prozesse

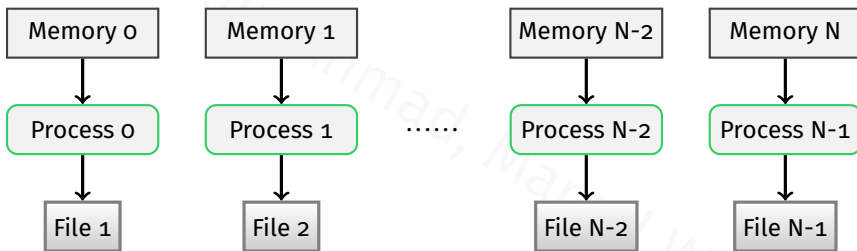
Vorteile

- Benutzung von Standard I/O Librarys
- Arbeiten mit nur einer Datei
- Schreiben/Lesen von großen chunks

Nachteile

- Skaliert nicht für viele Prozesse/große Datenmengen
- Nutzt parallele Dateisysteme nicht aus
- Setzen der Daten an entsprechende Stelle in der Datei kann aufwendig und teuer (seeks) sein

Eine Datei pro Prozess



Eine Datei pro Prozess

```
1  /* example of parallel Unix write into separate files */
2  <MPI Basis::init ?? >
3  #define BUFSIZE 100
4      int  buf[BUFSIZE];
5      char filename[128];
6      FILE *myfile;
7
8      for (i=0; i<BUFSIZE; i++)
9          buf[i] = myrank * BUFSIZE + i;
10     sprintf(filename, "testfile.%d", myrank);
11     myfile = fopen(filename, "w");
12     fwrite(buf, sizeof(int), BUFSIZE, myfile);
13     fclose(myfile);
14 <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013

Eine Datei pro Prozess

Vorteile

- Benutzung von Standard I/O Librarys
- Sehr einfach zu implementieren

Nachteile

- Viele kleine Dateien
- Müssen möglicherweise nachträglich zusammengesetzt werden
- Dateisystem bekommt bei vielen Prozessen und kleinen chunks Probleme
- Zur weiteren parallelen Verarbeitung müssen gleich viele Prozesse verwendet werden oder zusätzlicher Implementierungsaufwand nötig



Lesen und Schreiben mit MPI-IO

MPI-IO

Vorteile

- Erweiterung zum parallelen Lesen und Schreiben von Daten
- Hilfsfunktionen zum Lesen/Schreiben von kleinen, verteilten Chunks
- Kollektive Lese-/Schreib-Funktionen für schnelles I/O

Nachteile

- Benötigt ein paralleles Dateisystem für Leistungszuwachs
- Nicht mehr volle Kontrolle über Daten und Lese-/Schreib-Prozesse



MPI_File_open

Aufgabe

Öffnen einer Datei mit MPI-IO

Signatur

```
1 int MPI_File_open(  
2     MPI_Comm comm,      //[in]  communicator  
3     char *filename,     //[in]  name of file to open  
4     int amode,          //[in]  file access mode  
5     MPI_Info info,      //[in]  info object  
6     MPI_File *mpi_fh    //[out] file handle  
7 );
```



MPI_File_open

Options werden mit Oder “|” verknüpft übergeben

MPI_MODE_RDONLY Nur lesen

MPI_MODE_WRONLY Nur schreiben

MPI_MODE_RDWR Lesen und schreiben

MPI_MODE_CREATE Erzeugen wenn noch nicht vorhanden

MPI_MODE_EXCL Fehler wenn schon vorhanden

MPI_MODE_DELETE_ON_CLOSE Löschen wenn Datei geschlossen wird

MPI_MODE_UNIQUE_OPEN Datei wird nur von diesem Prozess geöffnet

MPI_MODE_SEQUENTIAL Datei wird nur sequentiell geöffnet

MPI_MODE_APPEND Alle Datei-Pointer an das Ende der Datei setzen

MPI_File_close

Aufgabe

Schließen einer Datei mit MPI-IO

Signatur

```
1 int MPI_File_close(  
2     MPI_File *mpi_fh    //[in] file handle  
3 );
```



MPI_File_delete

Aufgabe

Löschen einer Datei mit MPI-IO

Signatur

```
1 int MPI_File_delete(  
2     char *filename,    //[in] name of file to delete  
3     MPI_Info info      //[in] info object  
4 );
```

MPI_File_get_size

Aufgabe

Bestimmen der Größe einer Datei mit MPI-IO

Signatur

```
1 int MPI_File_get_size(  
2     MPI_File mpi_fh,    //[in] file handle  
3     MPI_Offset *size    //[out] size of the file in bytes  
4 );
```

MPI_File_read

Aufgabe

Lesen einer Datei mit MPI-IO

Signatur

```
1 int MPI_File_read(  
2     MPI_File mpi_fh,           //[in] file handle  
3     void *buf,                //[out] initial address of buffer  
4     int count,                //[in] number of elements in buffer  
5     MPI_Datatype datatype,    //[in] datatype of each buffer element  
6     MPI_Status *status        //[out] status object  
7 );
```

MPI_File_write

Aufgabe

Schreiben in eine Datei mit MPI-IO

Signatur

```
1 int MPI_File_write(  
2     MPI_File mpi_fh,           //[in] file handle  
3     void *buf,                 //[in] initial address of buffer  
4     int count,                 //[in] number of elements in buffer  
5     MPI_Datatype datatype,     //[in] datatype of each buffer element  
6     MPI_Status *status         //[out] status object  
7 );
```




MPI_File_sync

Aufgabe

Flush aller Daten mit MPI-IO

Signatur

```
1 int MPI_File_sync(  
2     MPI_File mpi_fh    //[in] file handle  
3 );
```

MPI_File_seek

Aufgabe

Springen an eine Position innerhalb einer Datei mit MPI-IO

Signatur

```
1 int MPI_File_seek(  
2     MPI_File mpi_fh,      //[in] file   handle  
3     MPI_Offset offset,   //[in] file   offset  
4     int whence           //[in] update mode  
5 );
```



MPI_File_seek

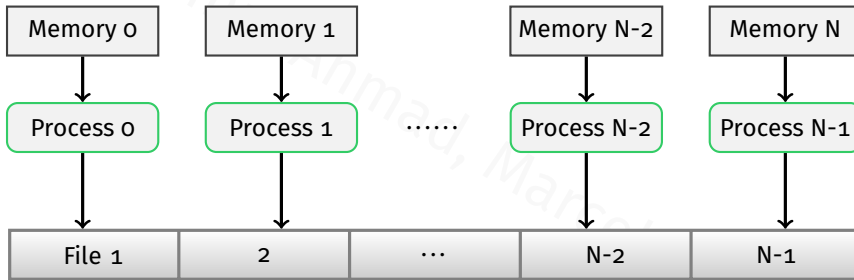
- MPI_SEEK_SET** pointer wird an Offset gesetzt
- MPI_SEEK_CUR** pointer wird um den übergeben Offset verschoben
- MPI_SEEK_END** pointer wird ans Ende plus den Offset verschoben

MPI-IO: eine Datei pro Prozess

```
1  /* example of parallel MPI write into separate files */
2  <MPI Basis::init ?? >
3  #define BUFSIZE 100
4      int i, buf[BUFSIZE];
5      char filename[128];
6      MPI_File myfile;
7
8      for (i=0; i<BUFSIZE; i++)
9          buf[i] = myrank * BUFSIZE + i;
10     sprintf(filename, "testfile.%d", myrank);
11     MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_WRONLY | MPI_MODE_CREATE,
MPI_INFO_NULL, &myfile);
12     MPI_File_write(myfile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
13     MPI_File_close(&myfile);
14 <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013

Eine Datei für alle Prozesse



MPI_File_write_at

Aufgabe

Schreiben an einer bestimmten Position in der Datei

Signatur

```
1 int MPI_File_write_at(  
2     MPI_File mpi_fh,           //[in] file   handle  
3     MPI_Offset offset,        //[in] file   offset  
4     void *buf,                //[in] initial address of buffer  
5     int count,                //[in] number of elements in buffer  
6     MPI_Datatype datatype,    //[in] datatype of each buffer element  
7     MPI_Status *status        //[out] status object  
8 );
```

MPI_File_read_at

Aufgabe

Lesen ab einer bestimmten Position in der Datei

Signatur

```
1 int MPI_File_read_at(  
2     MPI_File mpi_fh,           //[in] file   handle  
3     MPI_Offset offset,        //[in] file   offset  
4     void *buf,                //[out] initial address of buffer  
5     int count,                //[in] number of elements in buffer  
6     MPI_Datatype datatype,    //[in] datatype of each buffer element  
7     MPI_Status *status        //[out] status object  
8 );
```

MPI-IO: eine Datei für alle Prozesse

```
1  /* example of parallel MPI write into a single file */
2  <MPI Basis::init ?? >
3  #define BUFSIZE 100
4      int i, myrank, buf[BUFSIZE];
5      MPI_File thefile;
6
7      MPI_Init(&argc, &argv);
8      MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
9      for (i=0; i<BUFSIZE; i++)
10         buf[i] = myrank * BUFSIZE + i;
11      MPI_File_open(MPI_COMM_WORLD, "testfile",
12                   MPI_MODE_CREATE | MPI_MODE_WRONLY,
13                   MPI_INFO_NULL, &thefile);
14      MPI_File_write_at(thefile, myrank * BUFSIZE * sizeof(int), buf,
15                       BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
16      MPI_File_close(&thefile);
17  <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013



Views in MPI-IO

MPI_File_set_view

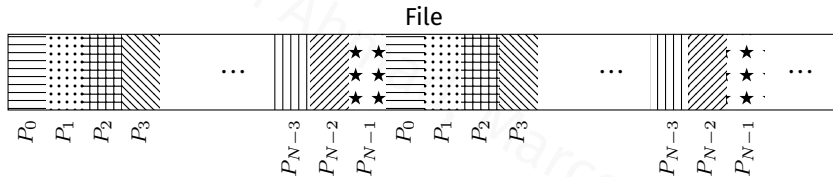
Aufgabe

Setzen eines “Fensters” auf die Datei

Signatur

```
1 int MPI_File_set_view(  
2     MPI_File mpi_fh,           //[in] file   handle  
3     MPI_Offset disp,           //[in] displacement  
4     MPI_Datatype etype,        //[in] elementary datatype  
5     MPI_Datatype filetype,     //[in] filetype  
6     char *datarep,             //[in] data   representation  
7     MPI_Info info,             //[in] info   object  
8 );
```

Eine Datei für alle Prozesse



MPI-IO: eine Datei für alle Prozesse

```
1  /* example of parallel MPI write into a single file */
2  <MPI Basis::init ?? >
3  #define BUFSIZE 100
4  int i, myrank, buf[BUFSIZE];
5  MPI_File thefile;
6
7  MPI_Init(&argc, &argv);
8  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
9  for (i=0; i<BUFSIZE; i++)
10     buf[i] = myrank * BUFSIZE + i;
11  MPI_File_open(MPI_COMM_WORLD, "testfile",
12     MPI_MODE_CREATE | MPI_MODE_WRONLY,
13     MPI_INFO_NULL, &thefile);
14  MPI_File_set_view(thefile, myrank * BUFSIZE * sizeof(int),
15     MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
16  MPI_File_write(thefile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
17  MPI_File_close(&thefile);
18  <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013



Kleiner Exkurs: selbstdefinierte MPI-Typen

MPI_Type_create_subarray

Aufgabe

Erzeugt einen neuen Datentyp der ein Untergitter in einem N-Dimensionalen regulären Raum beschreibt

Signatur

```
1 int MPI_Type_create_subarray(  
2     int ndims,                //[in] number of array dimensions  
3     int array_of_sizes[],      //[in] number of elements of type oldtype in each dimension of the full  
    array  
4     int array_of_subsizes[],  //[in] number of elements of type oldtype in each dimension of the  
    subarray  
5     int array_of_starts[],     //[in] starting coordinates of the subarray in each dimension  
6     int order,                //[in] array storage order flag  
7     MPI_Datatype oldtype,      //[in] array element datatype  
8     MPI_Datatype *newtype      //[out] new datatype  
9 );
```



MPI_Type_create_subarray

Order

MPI_ORDER_C row-major order wie bei C arrays

MPI_ORDER_FORTRAN column-major order wie bei Fortran arrays



MPI_Type_commit

Aufgabe

Neuen MPI-Typen bekannt machen zur späteren Verwendung

Signatur

```
1 int MPI_Type_commit(  
2     MPI_Datatype *datatype //[in] datatype  
3 );
```




Weitere MPI Funktionen zur Erzeugung komplexe MPI Datentypen

MPI_Type_create_hvector

MPI_Type_create_resized

MPI_Type_create_indexed_block

MPI_Type_create_hindexed

MPI_Type_create_darray

MPI_Type_contiguous

MPI_Type_create_struct

MPI-IO: Schreiben eines 2D zerlegten Gebiets I

```
1 <MPI Basis::init ?? >
2 int          gsizes[2], distribs[2], dargs[2], psizes[2], rank, size, m, n;
3 MPI_Datatype filetype;
4 int          local_array_size, num_local_rows, num_local_cols;
5 int          row_procs, col_procs, row_rank, col_rank;
6 int          dims[2], periods[2], lsizes[2], coords[2], start_indices[2];
7 MPI_Comm     comm;
8 MPI_File     fh;
9 float        *local_array;
10 MPI_Status   status;
11
12 /* This code is particular to a 2 x 3 process decomposition */
13 MPI_Comm_size( MPI_COMM_WORLD, &size );
14 if (size != 6) {
15     printf("Communicator size must be 6\n");
16     MPI_Abort( MPI_COMM_WORLD, 1 );
17 }
18
19 /* number of points in each direction */
20 m = 11; n = 15;
21
```

MPI-IO: Schreiben eines 2D zerlegten Gebiets II

```
22  /* See comments on block distribution */
23  row_procs = 2;
24  col_procs = 3;
25  num_local_rows = (m + row_procs - 1) / row_procs;
26  /* adjust for last row */
27  if (row_rank == row_procs-1)
28      num_local_rows = m - (row_procs-1) * num_local_rows;
29  num_local_cols = (n + col_procs - 1) / col_procs;
30  /* adjust for last column */
31  if (col_rank == col_procs-1)
32      num_local_cols = n - (col_procs-1) * num_local_cols;
33
34  local_array = (float *)malloc(num_local_rows * num_local_cols * sizeof(float));
35
36  /* ... set elements of local_array ... */
37
38  gsizes[0] = m; /* no. of rows in global array */
39  gsizes[1] = n; /* no. of columns in global array */
40
41  lsizes[0] = num_local_rows; /* no. of rows in local array */
42  lsizes[1] = num_local_cols; /* no. of columns in local array */
43
```

MPI-IO: Schreiben eines 2D zerlegten Gebiets III

```
44 dims[0] = 2;
45 dims[1] = 3;
46 periods[0] = periods[1] = 1;
47 MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm);
48 MPI_Comm_rank(comm, &rank);
49 MPI_Cart_coords(comm, rank, 2, coords);
50
51 /* global indices of the first element of the local array */
52 start_indices[0] = coords[0] * lsizes[0];
53 start_indices[1] = coords[1] * lsizes[1];
54
55 MPI_Type_create_subarray(2, gsizes, lsizes, start_indices,
56 MPI_ORDER_C, MPI_FLOAT, &filetype);
57 MPI_Type_commit(&filetype);
58
59 MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
60 MPI_MODE_CREATE | MPI_MODE_WRONLY,
61 MPI_INFO_NULL, &fh);
62 MPI_File_set_view(fh, 0, MPI_FLOAT, filetype, "native",
63 MPI_INFO_NULL);
64
65 local_array_size = lsizes[0] * lsizes[1];
```

MPI-IO: Schreiben eines 2D zerlegten Gebiets IV

```
66 MPI_File_write_all(fh, local_array, local_array_size,  
67   MPI_FLOAT, &status);  
68  
69 MPI_File_close(&fh);  
70  
71 <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013

MPI-IO: Schreiben eines 2D zerlegten Gebiets mit Ghost-Layer I

```
1 <MPI Basis::init ?? >
2 int      gsizes[2], distribs[2], dargs[2], psize[2], rank, size, m, n;
3 int      lsize[2], dims[2], periods[2], coords[2], start_indices[2];
4 int      memsize[2];
5 MPI_Datatype filetype, memtype;
6 MPI_Comm comm;
7 int      local_array_size, num_local_rows, num_local_cols;
8 int      row_procs, col_procs, row_rank, col_rank;
9 MPI_File fh;
10 float    *local_array;
11 MPI_Status status;
12
13 /* This code is particular to a 2 x 3 process decomposition */
14 MPI_Comm_size(MPI_COMM_WORLD, &size);
15 if (size != 6) {
16     printf("Communicator size must be 6\n");
17     MPI_Abort(MPI_COMM_WORLD, 1);
18 }
19
20 /* number of points in each direction */
21 m = 11; n = 15;
```

MPI-IO: Schreiben eines 2D zerlegten Gebiets mit Ghost-Layer II

```
22
23 /* See comments on block distribution */
24 row_procs = 2;
25 col_procs = 3;
26 num_local_rows = (m + row_procs - 1) / row_procs;
27 /* adjust for last row */
28 if (row_rank == row_procs-1)
29     num_local_rows = m - (row_procs-1) * num_local_rows;
30 num_local_cols = (n + col_procs - 1) / col_procs;
31 /* adjust for last column */
32 if (col_rank == col_procs-1)
33     num_local_cols = n - (col_procs-1) * num_local_cols;
34
35 /* no. of rows and columns in global array */
36 gsizes[0] = m;    gsizes[1] = n;
37 /* no. of processes in vertical and horizontal dimensions
38    of process grid */
39 lsizes[0] = num_local_rows; /* no. of rows in local array */
40 lsizes[1] = num_local_cols; /* no. of columns in local array */
41 dims[0] = 2;    dims[1] = 3;
42 periods[0] = periods[1] = 1;
43 MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm);
```

MPI-IO: Schreiben eines 2D zerlegten Gebiets mit Ghost-Layer III

```
44 MPI_Comm_rank(comm, &rank);
45 MPI_Cart_coords(comm, rank, 2, coords);
46 /* global indices of the first element of the local array */
47 start_indices[0] = coords[0] * lsizes[0];
48 start_indices[1] = coords[1] * lsizes[1];
49 MPI_Type_create_subarray(2, gsizes, lsizes, start_indices,
50 MPI_ORDER_C, MPI_FLOAT, &filetype);
51 MPI_Type_commit(&filetype);
52 MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
53 MPI_MODE_CREATE | MPI_MODE_WRONLY,
54 MPI_INFO_NULL, &fh);
55 MPI_File_set_view(fh, 0, MPI_FLOAT, filetype, "native",
56 MPI_INFO_NULL);
57
58 /* add 4 ghostlayers to each side and create a derived datatype that describes
59 * the layout of the local array in the memory buffer that includes the ghost area. */
60 memsizes[0] = lsizes[0] + 8; /* no. of rows in allocated array */
61 memsizes[1] = lsizes[1] + 8; /* no. of columns in allocated array */
62
63 /* ... set elements of local_array ... */
64 local_array = (float *)malloc(memsizes[0] * memsizes[1] * sizeof(float));
65
```


MPI-IO: Schreiben eines 2D zerlegten Gebiets mit Ghost-Layer IV

```
66 start_indices[0] = start_indices[1] = 4;  
67 /* indices of the first element of the local array in the  
68    allocated array */  
69 MPI_Type_create_subarray(2, memsizes, lsizes, start_indices,  
70    MPI_ORDER_C, MPI_FLOAT, &memtype);  
71 MPI_Type_commit(&memtype);  
72 MPI_File_write_all(fh, local_array, 1, memtype, &status);  
73 MPI_File_close(&fh);  
74  
75 <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013



Kollektives MPI-IO



Kollektives MPI-IO

- Kann viele kleine Chunks zu größeren zusammenfassen
- Weniger seeks bei z.B. kartesischen Topologien
- Höherer Lese-/Schreib-Durchsatz
- Alle Prozesse müssen Lese-/Schreib-Operation aufrufen (sonst Deadlock)

MPI_File_write_all

Aufgabe

Kollektives Schreiben in eine Datei

Signatur

```
1 int MPI_File_write_all(  
2     MPI_File mpi_fh,           //[in] file handle  
3     void *buf,                 //[in] initial address of buffer  
4     int count,                 //[in] number of elements in buffer  
5     MPI_Datatype datatype,     //[in] datatype of each buffer element  
6     MPI_Status *status        //[out] status object  
7 );
```

MPI_File_read_all

Aufgabe

Kollektives Lesen aus einer Datei

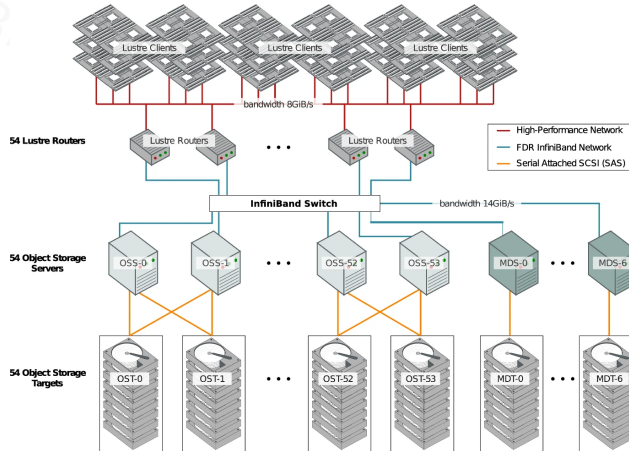
Signatur

```
1 int MPI_File_read_all(  
2     MPI_File mpi_fh,           //[in] file handle  
3     void *buf,                 //[out] initial address of buffer  
4     int count,                 //[in] number of elements in buffer  
5     MPI_Datatype datatype,     //[in] datatype of each buffer element  
6     MPI_Status *status         //[out] status object  
7 );
```



Kleiner Exkurs: Aufbau parallele Dateisystem

Aufbau parallele Dateisystem



Quelle: <https://link.springer.com/article/10.1007/s11227-021-03730-7>, besucht 30.11.2021



Hints for faster IO



Hints for faster IO

- Optimieren des Lese-/Schreib-Prozesses durch zusätzliche Informationen
- dateisystemspezifische Parameter (z.B. NFS, XFS)
- Hardware/Software Struktur (z.B. Raid XX, OTS Anzahl)



MPI_Info_create

Aufgabe

Erzeugt ein neues Info Objekt

Signatur

```
1 int MPI_Info_create(  
2     MPI_Info *info //[out] info object created  
3 );
```



MPI_Info_set

Aufgabe

Setzen eines Hints (Key/Value Paar)

Signatur

```
1 int MPI_Info_set(  
2     MPI_Info info, //[in] info object  
3     char *key,      //[in] key  
4     char *value     //[in] value  
5 );
```



MPI_Info_create

Aufgabe

Löscht ein Info Objekt

Signatur

```
1 int MPI_Info_free(  
2     MPI_Info *info //[in] info object to be freed  
3 );
```

MPI-IO: Hints für paralleles IO I

```
1 <MPI Basis::init ?? >
2 MPI_File fh;
3 MPI_Info info;
4
5 MPI_Info_create(&info);
6
7 /* FOLLOWING HINTS ARE PREDEFINED IN MPI */
8 /* no. of I/O devices across which the file should be striped */
9 MPI_Info_set(info, "striping_factor", "4");
10 /* the striping unit in bytes */
11 MPI_Info_set(info, "striping_unit", "65536");
12 /* buffer size for collective I/O */
13 MPI_Info_set(info, "cb_buffer_size", "8388608");
14 /* no. of processes that should perform disk accesses
15    during collective I/O */
16 MPI_Info_set(info, "cb_nodes", "4");
17
18 /* FOLLOWING ARE ADDITIONAL HINTS SUPPORTED BY ROMIO */
19 /* the I/O device from which to start striping the file */
20 MPI_Info_set(info, "start_iodevice", "2");
21 /* buffer size for data sieving in independent reads */
```

MPI-IO: Hints für paralleles IO II

```
22 MPI_Info_set(info, "ind_rd_buffer_size", "2097152");
23 /* buffer size for data sieving in independent writes */
24 MPI_Info_set(info, "ind_wr_buffer_size", "1048576");
25 /* use direct I/O on SGI's XFS file system
26    ( platform - specific hints ) */
27 MPI_Info_set(info, "direct_read", "true");
28 MPI_Info_set(info, "direct_write", "true");
29
30 /* NOW OPEN THE FILE WITH THIS INFO OBJECT */
31 MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
32    MPI_MODE_CREATE | MPI_MODE_RDWR, info, &fh);
33
34 MPI_Info_free(&info); /* free the info object */
35
36 /* ... access file ... */
37
38 MPI_File_close( &fh );
39
40 /* ... */
41 <MPI Basis::deinit ?? >
```

Quelle: [GLT99], <http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>, 01.12.2013



Grenzen von MPI

- Redundanz (muss selbst implementiert werden)
- Fehlertoleranz
- Skalierbarkeit bei $> 1.000.000$ CPUs (Managementaufwand)
- Debugging




Zusammenfassung


- Gruppen und Communicatoren
- Virtuelle Topologien
- MPI-IO
- Komplexe MPI Datentypen




Vielen Dank für ihre Aufmerksamkeit!






Fragen zur Vorlesung?

 GEORGE ALMÁSI, RALPH BELLOFATTO, JOSÉ BRUNHEROTO, CĂLIN CAȘCAVAL, JOSÉ G. CASTAÑOS, PAUL CRUMLEY, C. CHRISTOPHER ERWAY, DEREK LIEBER, XAVIER MARTORELL, JOSÉ E. MOREIRA, RAMENDRA SAHOO, ALDA SANOMIYA, LUIS CEZE, and KARIN STRAUSS.
An overview of the bluegene/l system software organization.
Parallel Processing Letters, 13(04):561–574, 2003.

 Gene M. Amdahl.
Validity of the single processor approach to achieving large scale computing capabilities.
In Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

 Clay Breshears.
The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications.
O'Reilly Media, 0 edition, 5 2009.

 David Culler, J.P. Singh, and Anoop Gupta.
Parallel Computer Architecture: A Hardware/Software Approach (The Morgan Kaufmann Series in Computer Architecture and Design).
Morgan Kaufmann, 1 edition, 8 1998.

-  **David Griffiths Dawn Griffiths.**
C von Kopf bis Fuß.
O'Reilly Vlg. GmbH und Co., 9 2012.
-  **William James Dally and Brian Patrick Towles.**
Principles and Practices of Interconnection Networks (The Morgan Kaufmann Series in Computer Architecture and Design).
Morgan Kaufmann, 1 edition, 1 2004.
-  **Victor Eijkhout.**
Introduction to High Performance Scientific Computing.
lulu.com, 10 2012.
-  **Message Passing Interface Forum.**
MPI: A Message-Passing Interface Standard, Version 3.0.
High Performance Computing Center Stuttgart, 2012.
-  **Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta.**
Introduction to Parallel Computing (2nd Edition).

Addison-Wesley, 2 edition, 1 2003.



William Gropp, Ewing L. Lusk, and Anthony Skjellum.

Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation).

The MIT Press, second edition edition, 11 1999.



William Gropp, Ewing L. Lusk, and Rajeev Thakur.

Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation).

The MIT Press, 1st edition, 11 1999.



John L. Gustafson.

Reevaluating amdahl's law.

Commun. ACM, 31(5):532–533, May 1988.



Georg Hager.

Introduction to high performance computing for scientists and engineers (chapman & hall/crc computational science), 9 2012.



David R. Hanson.

C Interfaces and Implementations: Techniques for Creating Reusable Software.
Addison-Wesley Professional, 1 edition, 8 1996.



J. Kim, W.J. Dally, S. Scott, and D. Abts.

Technology-driven, highly-scalable dragonfly topology.
In Computer Architecture, 2008. ISCA '08. 35th International Symposium on, pages 77–88,
June 2008.



David Padua, editor.

Encyclopedia of Parallel Computing (Springer Reference).
Springer, 2011 edition, 9 2011.



Manish Parashar, Xiaolin Li, and Sumir Chandra.

Advanced Computational Infrastructures for Parallel and Distributed Applications (Wiley Series on Parallel and Distributed Computing).
Wiley-Interscience, 1 edition, 12 2009.



Thomas Rauber.



Parallele Programmierung (eXamen.press) (German Edition).

Springer, 3. aufl. 2012 edition, 9 2012.



Thomas Rauber and Gudula Rünger.

Parallele programmierung (examen.press) (german edition), 9 2012.



Josef Schüle.

Paralleles Rechnen.

Oldenbourg Wissensch.Vlg, 9 2010.



Anthony Skjellum.

MPI - Eine Einführung.

Oldenbourg Wissensch.Vlg, 6 2007.



Peter van der Linden.

Expert C Programming: Deep C Secrets.

Prentice Hall, 1st edition, 6 1994.



Eric F. Van de Velde.

Concurrent scientific computing.



Texts in applied mathematics ; 16. Springer, New York, 1994.



G. Zumbusch.

Tuning a finite difference computation for parallel vector processors.

In *Parallel and Distributed Computing (ISPD), 2012 11th International Symposium on*,
pages 63–70, June 2012.