
	<p style="text-align: center;">Parallelrechner Übung Prof. Dr.-Ing. W. Rehm</p> <hr/> <p style="text-align: center;">Praktische Übung 2: Einführung in kollektive Operationen in MPI</p> <p style="text-align: center;">Bearbeiter: Lars Jordan, Nico Mittenzwey Letzte Änderung: René Oertel, 13.05.2015</p>	
---	---	---

1 Ziele

- Erstellen von Prozessgruppen und Kommunikatoren
- Kennenlernen globaler Kommunikationsoperationen

2 Einführung

Neben einfachen Sende- und Empfangsoperationen zwischen zwei Prozessen (Point-to-Point Kommunikation) unterstützt MPI auch sogenannte kollektive Operationen. An diesen Operationen nehmen alle Prozesse einer vorher definierten Prozessgruppe teil. Dabei kann man folgende Gruppen von kollektive Operationen unterscheiden:

One-To-All Ein Prozess schickt Daten an alle anderen Prozesse.
Beispiele: MPI_Bcast, MPI_Scatter

All-To-One Alle Prozesse tragen zur Operation bei und ein Prozess erhält das Ergebnis.
Beispiele: MPI_Gather, MPI_Reduce

All-To-All Alle Prozesse tragen zur Operation bei und alle erhalten das Ergebnis.
Beispiele: MPI_Alltoall, MPI_Allgather, MPI_Allreduce

Andere Operationen die nicht in die oben genannten Gruppen passen.
Beispiel: MPI_Barrier zur Synchronisation aller Prozesse

Gruppen Gruppen beschreiben eine abzählbare Sammlung von Prozessen und dienen der Erfassung von Mitgliedern eines Kommunikationsraumes für Punkt-zu-Punkt und kollektive Kommunikation. Jeder Prozess besitzt innerhalb seiner Gruppe einen eindeutigen numerischen Identifikator (in MPI-Terminologie: Rank). Ausgehend von bereits bestehenden Gruppen ist die Ableitung neuer Gruppen möglich.

Gruppen können als Realisierungsbasis oder innere Repräsentation von Kommunikatoren aufgefasst werden. Die Adressierung eines Kommunikationspartners auf der Basis von Gruppen ist nicht möglich, für diese Identifikation werden ausschließlich Communicator-Rank-Angaben akzeptiert. Jeder Kommunikator baut jedoch auf einer Gruppe auf.

Kommunikatoren Kommunikatoren können als eine Art Hülle betrachtet werden, in die Gruppen, Kontexte und andere MPI-Verwaltungsobjekte eingebettet sind. Damit lassen sich in MPI verschiedene, voneinander geschützte Kontexte bzw. „Kommunikationswelten“ aufbauen. Nachrichten eines Kontextes können von Mitgliedern eines anderen Kontextes nicht direkt empfangen werden.

3 Gruppen und Kommunikatoren

Ein Prozess kann gleichzeitig Mitglied in mehreren Gruppen sein. In jeder dieser Gruppen besitzt er eine eigenständige Identität (Gruppenhandle und Rank), wobei der Rank immer relativ zur Gruppe ist. Bei gleichzeitiger Mitgliedschaft in mehreren Gruppen kann demnach ein Prozess in jeder dieser Gruppen einen unterschiedlichen Rank aufweisen. Der Rank ist in MPI eine einfache Ganzzahl und liegt bei einer Gruppengröße von n im Bereich 0 bis $n - 1$. Als Anwender besitzt man keinen Einfluss darauf, wie MPI die Zuordnung von Rank zu Prozess organisiert.

Eine äquivalente vordefinierte Gruppe zu dem initialen Kommunikator `MPI_COMM_WORLD` existiert nicht. Durch Anwendung der Funktion `MPI_Comm_group()` lässt sich jedoch sehr leicht ein Handle für die entsprechende Prozessgruppe generieren. Die so („zurück-“ gewonnene Gruppe enthält alle Prozesse der gesamten MPI-Applikation. Die Größe einer Gruppe sowie der eigene Identifikator lassen sich durch `MPI_Group_size()` und `MPI_Group_rank()` ermitteln. Existieren mehrere Prozessgruppen, so kann für einen Prozess bei Kenntnis des Ranks in einer Gruppe sein Rank in anderen Gruppen abgefragt werden. Besteht zum Beispiel die Notwendigkeit, dass die Master (Rank = 0) zweier Prozessgruppen miteinander kommunizieren müssen, so ließe sich unter Nutzung von `MPI_Group_translate_ranks()` der entsprechende Rank des Partners im vordefinierten und gemeinsamen Kommunikator `MPI_COMM_WORLD` feststellen.

3.1 Gruppenkonstruktoren

MPI erlaubt die Bildung einer neuen Prozessgruppe nur als Ableitung (Unter- oder Übermenge) bereits existierender Gruppen. Zum Anlegen bzw. Ermitteln eines neuen Gruppenhandles werden folgende drei Verfahrensweisen bereitgestellt:

1. Ausführung einer Mengenoperationen (Vereinigung, Differenz, Durchschnitt) auf zwei existierende Gruppen.
2. n Mitglieder einer existierenden Gruppe werden in eine Liste aufgenommen. Auf Basis dieser Liste erfolgt die Erzeugung der neuen Gruppe. Dabei werden die n Mitglieder entweder in die neue Gruppe aufgenommen oder von der neuen Gruppe ausgeschlossen.
3. Von jedem existierenden Kommunikator lässt sich ein Handle der entsprechenden Gruppe ermitteln.

Gruppenkonstruktoren dürfen auch von Prozessen benutzt werden, die nicht der neu zu erzeugenden Gruppe angehören. Einzige Bedingung für einen zulässigen Konstruktoraufruf ist, dass alle Prozesse über alle Handles derjenigen Gruppen verfügen, von denen die neue Gruppe abgeleitet wird. Die gleiche Handhabung trifft bei der Überführung einer Gruppe in einen Kommunikator zu. Damit können einem Prozess alle verfügbaren Kommunikatoren bekannt sein, benutzen darf er allerdings nur die, in denen er selbst Mitglied ist.

Eine einfache Variante zum Test auf Gruppenmitgliedschaft bietet der Aufruf von `MPI_Group_rank()` bzw. `MPI_Comm_rank()`. Falls ein Prozess nicht in die übergebene Gruppe (bzw. Kommunikator) eingeschlossen ist, stimmt der zurückgegebene Rank mit der symbolischen Konstante `MPI_UNDEFINED` überein.

3.2 Konvertierung zwischen Gruppe und Kommunikator

Die reine Bildung einer Gruppe eröffnet noch keine Möglichkeiten, die Mitglieder dieser neuen Gruppe innerhalb kollektiver oder Punkt-zu-Punkt-Kommunikation tatsächlich anzusprechen, da jeglicher Datenaustausch in MPI immer an einen Kommunikator gebunden ist. MPI bietet deshalb einen Konstruktor, der eine gegebene Gruppe in einen äquivalenten Kommunikator überführt (`MPI_Comm_create()`). Im Gegenzug ermöglicht MPI auch die Ermittlung der entsprechenden Gruppe von einem bestehenden Kommunikator (`MPI_Comm_group()`). Der letztgenannte Weg gibt dem Nutzer beispielsweise die Möglichkeit, zum initialen Kommunikator `MPI_COMM_WORLD` die korrespondierende Gruppe zu ermitteln.

3.3 Beispiel zur Anwendung von Gruppenkonstruktoren

```
MPI_Group MPIGroupWorld, OddEvenGroup, OthersGroup;
MPI_Comm OddEvenComm, OthersComm;
int i, j, GroupSize, GroupRank;
int *RankList;

/*
 *      Zu MPI_COMM_WORLD korrespondierende Gruppe ermitteln
 */

MPI_Comm_group(MPI_COMM_WORLD, &MPIGroupWorld);
MPI_Group_size(MPIGroupWorld, &GroupSize);
MPI_Group_rank(MPIGroupWorld, &GroupRank);

/*
 * Bildung zweier neuer Gruppen - eine Gruppe für alle Tasks mit ungeraden Ranks,
 * die zweite für alle Tasks mit geraden Ranks
 */

RankList = (int *) malloc(GroupSize * sizeof(int));
for (j = 0, i = 0; i < GroupSize; i++)
    if ((GroupRank & 0x01) == (i & 0x01))
        RankList[j++] = i;

/*
 * Ein Rank mit gerader Nummer hat anschließend alle geraden Ranks
 * in RankList, ein ungerader Rank hat alle ungeraden Ranks in RankList
 */
/*
 * Zunächst die eigene Gruppe bilden - inklusive Auswertung der Rankliste
 */

MPI_Group_incl(MPIGroupWorld, j, RankList, &OddEvenGroup);
MPI_Comm_create(MPI_COMM_WORLD, OddEvenGroup, &OddEvenComm);

/*
 * Durch Mengenoperation alle Tasks außerhalb der eigenen Gruppe
 * in OthersGroup aufnehmen
 */

MPI_Group_difference(MPIGroupWorld, OddEvenGroup, &OthersGroup);
MPI_Comm_create(MPI_COMM_WORLD, OthersGroup, &OthersComm);

/*
 * Den Rank in der eigenen Gruppe feststellen und mit Hilfe von
 * MPI_Group_translate() den Rank des Masters der anderen Gruppe
 * in MPI_COMM_WORLD ermitteln.
 */

MPI_Group_rank(OddEvenGroup, &GroupRank);

if (GroupRank == MASTER) {
    MPI_Group_translate_ranks(OthersGroup, 1, &GroupRank,
                              MPIGroupWorld, &OtherMaster);

    /*
     * Kommunikation mit dem anderen Master ueber MPI_COMM_WORLD
     */
}
}
```

4 Kollektive Operationen

Es existieren kollektive Operationen zur Synchronisation aller Prozesse einer Gruppe, dem Datentransport (Verteilen, Zusammenfassen, Splitten) oder zur globalen Berechnung (z.B. globales Minimum oder eine globale Summe). Wenn möglich sollten bei der Programmierung von Programmen kollektive Operationen immer einfachen Send- und Receive-Operationen vorgezogen werden, da MPI-Implementierungen wie Open MPI kollektive Operationen an die Netzwerktopologie und andere Umgebungsparameter anpassen und damit stark beschleunigen können. Die jeweilige kollektive Operation muss von **allen** Prozessen einer Gruppe gerufen werden, andernfalls kommt es zum Deadlock, da die rufenden Prozesse auf alle anderen warten, bis diese die Operation ausführen.

4.1 Datentransportfunktionen

Broadcast Schickt Daten *buffer* von einem Prozess *root* einer Gruppe *comm* an alle anderen Prozesse dieser Gruppe.

Aufruf: `MPI_Bcast(buffer, count, datatype, root, comm)`

Scatter Verteilt die Daten *sendbuf* eines Prozesses *root* an alle anderen Prozesse der Gruppe *comm*. Dabei sollte $\text{sendcount} = n \cdot \text{recvcount}$ sein, wobei n die Anzahl der Prozesse ist. Der *root* Prozess „schickt“ somit auch sich selbst ein Datum.

Aufruf: `MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)`

Gather Wie Scatter nur in die andere Richtung. Sammelt Daten von allen Prozessen der Gruppe auf einem Prozess ein.

Aufruf: `MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)`

AllGather Wie Gather, nur dass nach der Operation alle Prozesse alle Daten besitzen.

Aufruf: `MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)`

AllToAll Kompletter Austausch aller Daten aller Mitglieder (im Bild 1 als „complete exchange“ bezeichnet). Dabei sollte $\text{sendcount} = n \cdot \text{recvcount}$ sein, wobei n die Anzahl der Prozesse ist. Jeder Prozess „schickt“ sich selbst also auch sein eigenes Datum.

Aufruf: `MPI_Alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)`

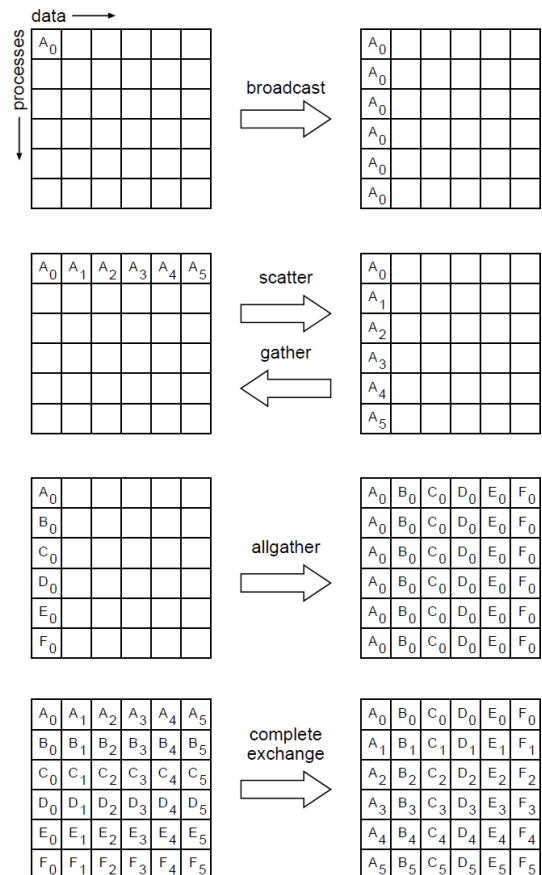


Abbildung 1: Kollektive Datentransportfunktionen dargestellt für eine Gruppe von 6 Prozessen. Ein Prozess kann bis zu 6 Dateneinheiten besitzen. Quelle: [1]

4.2 Globale Reduktionsoperationen

Für globale Reduktionen bietet MPI die beiden Basisroutinen `MPI_Reduce()` (nur der Master erhält das Resultat) und `MPI_Allreduce()` (alle erhalten das Resultat) an. Die Art der Reduktionsoperation (Summe, Minimum etc.) lässt sich vom Nutzer per Übergabeparameter spezifizieren. Neben einer Reihe vordefinierter Operatoren besteht die Möglichkeit, eigene Operationen selbst zu definieren.

Aufruf:

`MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)`

`MPI_Allreduce(sendbuf, recvbuf, count, datatype, op, comm)`

Dabei kann op eine der folgenden Konstanten sein:

MPI_MAX	-	Maximum
MPI_MIN	-	Minimum
MPI_SUM	-	Summe
MPI_PROD	-	Produkt
MPI_LAND	-	Logisches AND
MPI_BAND	-	Bit-Wise AND
MPI_LOR	-	Logisches OR
MPI_BOR	-	Bit-Wise OR
MPI_LXOR	-	Logisches XOR
MPI_BXOR	-	Bit-Wise XOR
MPI_MAXLOC	-	Maximum und Prozess, der dieses hat
MPI_MINLOC	-	Minimum und Prozess, der dieses hat

5 Anmerkungen

Eine Übersicht über viele MPI-Funktionen mit Beispielen, befindet sich unter [2]. Da jedoch nicht auszuschließen ist, dass sich in diese Übersicht - oder andere im Internet zu findende Literatur - Fehler eingeschlichen haben, sollte im Zweifelsfall immer der MPI-Standard [1] zu Rate gezogen werden.

6 Aufgaben

6.1 Kontrollfragen

1. Wer darf, wer muss einen Gruppenkonstruktor rufen?
2. Welche grundlegenden Schritte zur Eröffnung eines neuen Kommunikators müssen vollzogen werden?
3. Wie kann man feststellen, ob ein Prozess Mitglied einer bestimmten Gruppe ist?
4. Ist es möglich, eine globale Reduktionsoperation zu rufen, ohne alle Mitglieder des verwendeten Kommunikators einzubeziehen?

6.2 Praktische Aufgaben

1. Schauen Sie sich das Beispielprogramm an, kompilieren Sie es mit GCC und messen Sie die Laufzeit mit dem Unix-Befehl `time`.
2. Warum ist es nicht sinnvoll die FOR-Schleifen innerhalb der Funktion `calc_energy` zu parallelisieren?
3. Parallelisieren Sie die innere FOR-Schleife (Verschiebung in drei Richtungen) der Berechnung und bestimmen Sie das Maximum mit Hilfe einer Reduktionsoperation. Hierbei initialisiert jeder Prozess zur Vereinfachung erst einmal die Arrays selbst. Die Funktion `calc_energy` soll dabei so bleiben, wie sie ist.
4. Bilden Sie Gruppen zu je 3 Prozessen um über die Atome zu parallelisieren und bestimmen Sie das Maximum mit Hilfe einer Reduktionsoperation.
5. Lassen Sie die Initialisierung (bzw. das „Einlesen“) der Anzahl der Atome und deren Positionen nur vom Master/Root-Prozess erledigen. Dieser soll danach beides per Broadcast an alle anderen Prozesse verteilen.
6. Messen Sie jeweils die Laufzeiten mit `time`.

6.3 Hinweise

Zur Parallelisierung einer FOR-Schleife nutzt man üblicherweise die Modulo-Funktion, um die einzelnen Schleifendurchgänge auf alle verfügbaren Prozesse verteilen zu können. Also beispielsweise: `if(i % size == rank){ ... }`

Hinweise, Berichtigungen und Kritik zu den Übungsunterlagen bitte an:

- René Oertel <rene.oertel@cs.tu-chemnitz.de>

Literatur und wichtige Links

- [1] *MPI 2.2 Standard*
<http://www.mpi-forum.org/docs/docs.html>
- [2] *Übersicht über MPI-Funktionen mit Beispielen*
<http://www.tu-chemnitz.de/informatik/RA/projects/mpihelp/mpihelp.html>
- [3] *Homepage von Open MPI*
<http://www.open-mpi.org/>

13. Mai 2015