Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Schriftliche Ausarbeitung

MPI-Datentypen

vorgelegt von

Antonina Braun

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	Informatik
Matrikelnummer:	7402175

Betreuer: Hermann Lenhart

Hamburg, 2021-08-16

Contents

1 Was ist MPI?

Bevor ich beginne, die MPI Datentypen zu erläutern, erkläre ich hier kurz, was MPI eigentlich ist.
Die Abkürzung MPI bedeutet Message Passing Interface. Interfaces sind vielen bereits aus verschiedensten Programmiersprachen bekannt - sie sind Strukturvorgaben für z.B. Programme oder andere konkrete Konstrukte, geben also die Funktionalität, nicht aber die Implementation vor. Zusammen mit dem "Message Passing" ergibt sich also, dass MPI eine Strukturvorgabe für den Austausch von Nachrichten ist.
Bei den Nachrichten Austauschenden handelt es sich um Computer, die sich in einem System befinden und einen gemeinsamen Speicher verwenden, wie sie zum Beispiel in Rechenzentren vorhanden sind. Dieser Speicher ist aber unterschiedlichen Prozessoren zugeordnet, sodass ein gemeinsamer Zugriff organisiert werden muss.
MPI kann in vielen verschiedenen Sprachen implementiert werden, die meist verwendeten sind C und FORTRAN.
Eine solche Norm ist sinnvoll, weil es so nicht zu Kommunikationsfehlern zwischen den unterschiedlichen Rechnern kommen kann. Das MPI ist bezüglich Effizienz und Schnelligkeit optimiert und bietet daher viele Vorteile für die Kommunikation zwischen Computern, die ohne diese Norm in jedem System selbst programmiert werden müssten.

Der Aufbau der Befehle für das Senden und Empfangen von Daten sieht wie folgt aus:

```
1 MPI_Send (buf, count, datatype, dest, tag, comm)
```

Listing 1.1: Befehl zum Senden von Daten [1]

```
1 MPI_Recv (buf, count, datatype, source, tag, comm, status)
```

Listing 1.2: Befehl zum Empfangen von Daten [1]

Die verschiedenen Variablen bedeuten dabei Folgendes:

- buf: Zeiger auf den Datenbeginn im Speicher
- count: Zahl der Elemente, die zu senden sind
- datatype: Datentyp der Elemente

- dest: Rang des Zielprozesses
- tag: Markierung der Nachricht
- comm: Kommunikator der Prozessgruppe
- source: Rang des Quellprozesses
- status: Zeiger auf eine Statusstruktur, in der Informationen über die empfangene Nachricht abgelegt werden sollen

Die Kommunikation funktioniert also wie folgt:

Dem Send-Befehl werden im Quellprozess der Daten alle notwendigen Informationen zu den Daten als Parameter übergeben. Diese Informationen beinhalten einen Zeiger auf den Beginn der Daten im Speicher (buf), die Zahl der Speicherzellen, die insgesamt versendet werden sollen (count), den Datentyp der Daten (datatype), den Rang des Zielprozesses, an den die Daten geschickt werden sollen (dest), gegebenenfalls eine zusätzliche Markierung der Nachricht (tag) und den Kommunikator der Prozessgruppe, in der sich Quell- und Zielprozess befinden(comm).
Als Parameter bei Letzterem findet sich oft der Ausdruck COMM_WORLD bzw. MPI_-COMM_WORLD. Dieser bezeichnet die Summe aller Prozesse in dem System, in dem gesendet wird. Ein Send-Befehl nach MPI könnte zum Beispiel so aussehen:

```
1 MPI_Send(message, strlen(message)+1, MPI_CHAR, 0, tag,
2 COMM_WORLD);
```

Listing 1.3: Ein Beispiel-Send-Befehl in C [1]

Hier wird der Beginn der Daten als Variable mit dem Namen message übergeben, die Daten sind insgesamt so lang wie der String message und vom Typ char. Der Rang des Zielprozesses ist 0, es wird eine Markierung in Form einer Variable namens tag übergeben, und der Kommunikator der Prozessgruppe ist der eben erwähnte Ausdruck COMM_WORLD.

Auch dem Recieve-Befehl werden die notwendigen Informationen im Zielprozessals Parameter übergeben. Viele verhalten sich genau wie im Send-Befehl, doch einige kommen neu hinzu. Der dest-Parameter wird durch einen source-Parameter ersetzt, welcher den Rang des Quellprozesses der Daten angibt. Außerdem wird als letzter Parameter ein status übergeben - dieser ist ein Zeiger auf eine Statusstruktur im Speicher, über die Informationen über die empfangene Nachricht abgelegt und abgerufen können.

Anhand des Parameters datatype sieht man bereits, dass für die Kommunikation sogenannte MPI-Datentypen benötigt werden. Mit diesen werde ich mich nun befassen.

2 MPI-Datentypen

Für die Kommunikation sind Datentypen essenziell. Ohne ihre Verwendung würden die Daten nicht komplett mit gesendet werden müssen, was sehr Zeit- und Speicheraufwändig wäre. So kann der empfangende Rechner die Daten direkt aus dem Speicher lesen, weil ihm mit dem Datentypen übermittelt wird, wie lang die jeweiligen Daten sind, wo im Speicher sie liegen und wie viele Daten dieses Typs versendet werden sollen.

2.1 Elementare Datentypen

Es gibt viele verschiedene Arten von Datentypen im MPI. Die elementaren Datentypen sind die Basis für alle weiteren Typen. Diese Datentypen werden in vielen Programmiersprachen auf genau gleiche Art verwendet und sind daher sehr gut mit den Typen aus diesen Sprachen zu vergleichen [2] (siehe Fig. 2.1).

C type	MPI type
char	MPI_CHAR
unsigned char	MPI_UNSIGNED_CHAR
char	MPI_SIGNED_CHAR
short	MPI_SHORT
unsigned short	MPI_UNSIGNED_SHORT
int	MPI_INT
unsigned int	MPI_UNSIGNED
long int	MPI_LONG
unsigned long int	MPI_UNSIGNED_LONG
long long int	MPI_LONG_LONG_INT
float	MPI_FLOAT
double	MPI_DOUBLE
long double	MPI_LONG_DOUBLE
unsigned char	MPI_BYTE

Figure 2.1: Verleich von Datentypen in C und MPI [2]

Die Bedeutung dieser Datentypen ist wie folgt:

- char: Ein 4-bit Buchstabe
- short: Eine 16-bit Zahl

- int: Eine 32-bit Zahl
- long int: Eine 64-bit Zahl
- float: Eine 32-bit Gleitkommazahl
- double: Eine 64-bit Gleitkommazahl

In der Grafik ist jeder Datentyp auch einmal in unsigned aufgeführt. Das bedeutet, dass die Zahl nie als negativ aufgefasst wird, während sie signed[1] ab einer gewissen Größe als eine negative Zahl, nämlich ihr Zweierkomplement, aufgefasst werden würde. Dadurch können mit einem unsigned Datentyp größere Werte gespeichert werden als mit der signed Variante desselben Typs, allerdings nur, wenn im Voraus bekannt ist, dass der Wert niemals negativ wird.

2.2 Der Contiguous-Datentyp

Der Contiguous-Datentyp ist eine Datenstruktur, die mehrere Elemente des gleichen Datentyps enthält. Dieser Datentyp kann entweder elementar oder ein vorher definierter Typ sein. Diese Elemente müssen im Speicher direkt hintereinanderliegen [2] (siehe Fig. 2.2). Als Parameter werden dem Array die Anzahl der Elemente und der Typ dieser übergeben, zurückgegeben wird der Contiguous-Typ (siehe List. 2.1).

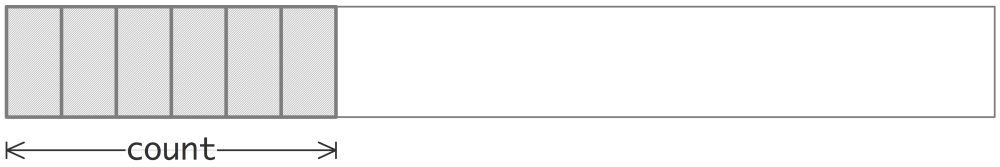

Figure 2.2: Speicherzellen, die mit dem Contiguous-Typ versendet werden [2]

Der Aufbau des Typs sieht aus wie folgt:

```
1 MPI_TYPE_CONTIGUOUS (count, oldtype, newtype)
```

Listing 2.1: Aufbau des Contiguous-Datentyps [2]

[1] Nicht unsigned

2.3 Der Vector-Datentyp

Der Vector-Datentyp ist dem eben besprochenen Contiguous-Typen sehr ähnlich. Es werden ebenfalls Daten mit demselben Typ versendet, diesmal müssen sie allerdings nicht direkt nebeneinander im Speicher liegen, müssen aber in gleich großen Datenblöcken verteilt sein, die denselben Abstand zueinander haben (siehe Fig. 2.3). Um dieses Konzept zu realisieren, werden außer den Parametern, die für den Contiguous-Typen von Nöten waren, auch die blocklength - die Anzahl der Elemente pro Datenblock - und der stride - der Abstand des Starts der Datenblöcke - eingegeben (siehe List. 2.2). Die Daten kommen beim Empfänger dann als Contiguous-Typ an [2] (siehe Fig. 2.3).

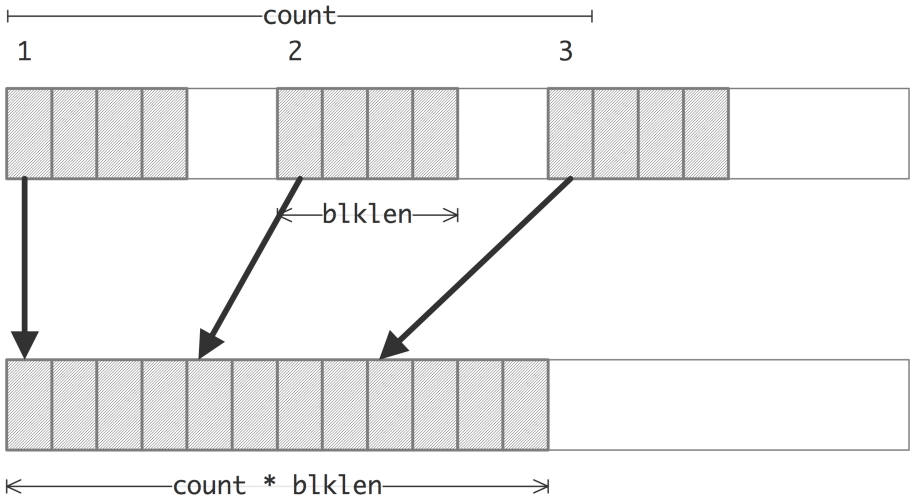

Figure 2.3: Speicherzellen, die mit dem Vector-Typ versendet werden [2]

Der Aufbau des Vector-Typs sieht aus wie folgt:

```
1 MPI_Type_Vector(count, blocklength, stride, oldtype,
2 newtype)
```

Listing 2.2: Aufbau des Vector-Datentyps [2]

2.4 Der Indexed-Datentyp

Der Indexed-Datentyp ist ebenfalls eine leicht komplexe Version des zuvor erläuterten Datentyps: Er ist dem Vector-Typ sehr ähnlich, unterscheidet sich von diesem aber durch die Tatsache, dass die Datenblöcke unterschiedlich groß sein und im Speicher unterschiedlich große Abstände haben können (siehe Fig. 2.4). Das wird realisiert, indem statt nur einem Wert für Abstand und Blocklänge jeweils ein Array an Werten eingegeben wird [2] (siehe List. 2.3).

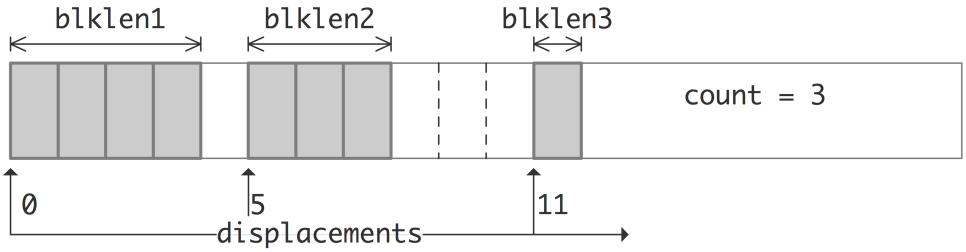

Figure 2.4: Speicherzellen, die mit dem Indexed-Typ versendet werden [2]

Der Aufbau des Typs sieht aus wie folgt:

```
1 MPI_Type_indexed(count, array_of_blocklengths,
2 array_of_displacements, oldtype, newtype)
```

Listing 2.3: Aufbau des Indexed-Datentyps [2]

2.5 Der Struct-Datentyp

Auch der Struct-Datentyp ist sehr ähnlich zum Indexed-Typ, unterscheidet sich aber in dem Punkt, dass hier nun auch Typen verschiedener Elemente zusammen verstendet werden können (siehe Fig. 2.5). Um das zu erreichen, werden nun auch die Typen als Array angegeben [2] (siehe List. 2.4).

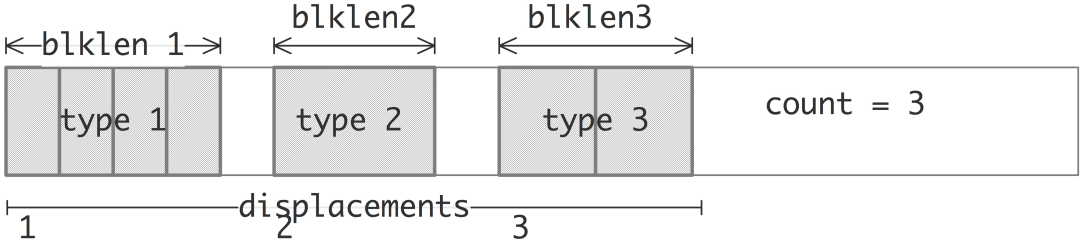

Figure 2.5: Speicherzellen, die mit dem Struct-Typ versendet werden [2]

Der Aufbau des Struct-Typs sieht aus wie folgt:

```
1 MPI_Type_struct(count, array_of_blocklengths,
2 array_of_displacements, array_of_types, newtype)
```

Listing 2.4: Aufbau des Struct-Datentyps [3]

3 Zusammenfassung

Zusammenfassend lässt sich also sagen, dass MPI ein Standard für Kommunikation zwischen Prozessen auf verschiedenen Prozessoren ist, welcher unter anderem mittels eigener Datentypen Effizienz und Performanz optimierend arbeitet.
Diese Datentypen tragen maßgebend zu den beiden eben genannten Faktoren bei, weil sie die Kommunikation vereinfachen, indem sie dem Versenden der Daten vorbeugen. Stattdessen liest der empfangende Prozess die daten selbst aus dem Speicher aus.

MPI-Datentypen sind entweder elementare Datentypen wie char, short, int, long, float oder double, oder sie sind aus ihnen oder anderen bereits erstellten Typen zusammengesetzt, wie zum Beispiel ein Contiguous-, Vector-, Intexed- oder Struct-Typ.
Um sie zu erstellen, bedarf es einer Operation, welche allerdings zunächst mit einer Programmiersprache implementiert werden muss, da das MPI nur ein Interface, also eine Strukturvorgabe ist.

4 Literatur

[1] MPI Tutorial, 2013: Shao-Ching Huang
[2] https://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-data.html
[3] https://www.mpich.org/static/docs/v3.3.x/www3/MPI_Type_struct.html