
	<p style="text-align: center;">Parallelrechner Übung Prof. Dr.-Ing. W. Rehm</p> <hr/> <p style="text-align: center;">Praktische Übung 3: Nutzerspezifische Datentypen und MPI-Profilng Bearbeiter: Lars Jordan, Nico Mittenzwey Letzte Änderung: René Oertel, 21.05.2015</p>	
---	--	---

1 Ziele

- Erstellen von nutzerspezifischen MPI Datentypen
- Kennenlernen von MPE zum Profiling von MPI Anwendungen

2 Einführung

Sinn und Zweck paralleler Programmierung ist es, ein Programm durch die Nutzung von mehreren Recheneinheiten zu beschleunigen. Eine wichtige Größe dabei ist der **Speedup**. Dieser gibt an, um wie viel ein paralleler Algorithmus gegenüber einem korrespondierenden sequenziellen Algorithmus schneller ist: $S_p = \frac{T_s}{T_p}$. Dabei ist p die Anzahl der Prozesse, T_s die Ausführungszeit des sequenziellen Algorithmus und T_p die Ausführungszeit des parallelen Algorithmus. Ist $S_p = p$, spricht man von „linearem Speedup“ oder „idealem Speedup“. In diesem Fall halbiert sich die Rechenzeit bei einer Verdopplung der Prozesse. In den meisten Fällen, wird man aufgrund von Synchronisierungs- und Datenaustauschoperationen zwischen den Prozessen keinen idealen Speedup erreichen. In einigen wenigen Fällen kann der Speedup jedoch sogar über dem idealen liegen, wenn z. B. durch eine Aufteilung der Daten diese vollständig in den Cache einer CPU passen.

Ziel dieser Übung ist es, Methoden zur Reduzierung der Kommunikationszeit und Erkennung von Flaschenhälsen kennenzulernen. Neben der Nutzung von kollektiven Operationen (siehe MPI-Versuch 2) kann auch das geschickte Verpacken von Daten zu dieser Reduzierung beitragen.

2.1 Applikationsgerechte Kommunikation

Wie bereits im MPI-Versuch 1 erwähnt, basiert jeglicher Datentransfer in MPI auf MPI-spezifischen Datentypen. Sollen Daten verschiedenen Typs ausgetauscht werden, müsste mit jedem der Basis-Datentypen ein separater Kommunikationsschritt ausgeführt werden. Bei kleinen Datenmengen ist häufig die eigentliche Übertragungszeit geringer als die Startup-Zeit. Die Startup-Zeit ist die Zeit, die benötigt wird um die Kommunikation zu initialisieren. Deshalb wäre es effektiver, möglichst viele Daten innerhalb eines Kommunikationsschrittes auszutauschen. MPI bietet für die Realisierung eines solchen Übertragungsverhaltens zwei Möglichkeiten an.

1. Definition und Registrierung eines neuen anwenderspezifischen MPI-Datentypes basierend auf Vektoren (Zusammenfassung gleichartiger Datenelemente) oder Strukturen (Zusammenfassung verschiedener Datenelemente).
2. Kommunikation mit gepackten Daten. Alle Daten werden unabhängig vom Typ in einem Puffer abgelegt, der dann in einem Kommunikationsschritt übertragen wird. Der Empfänger muss die Daten anschließend typgerecht aus dem Empfangspuffer extrahieren.

2.2 Profiling von MPI-Anwendungen

Mit Hilfe von MPE (MPI Parallel Environment) [4] lässt sich das Verhalten von MPI-Anwendungen automatisch aufzeichnen. Die erzeugten Logfiles enthalten den Zeitpunkt und die Dauer von allen MPI Ereignissen. Mit Hilfe des Visualisierungstools „Jumpshot“ lassen sich diese dann grafisch darstellen und analysieren. Bild 1 zeigt eine typische Ansicht von Jumpshot für 6 Prozesse.

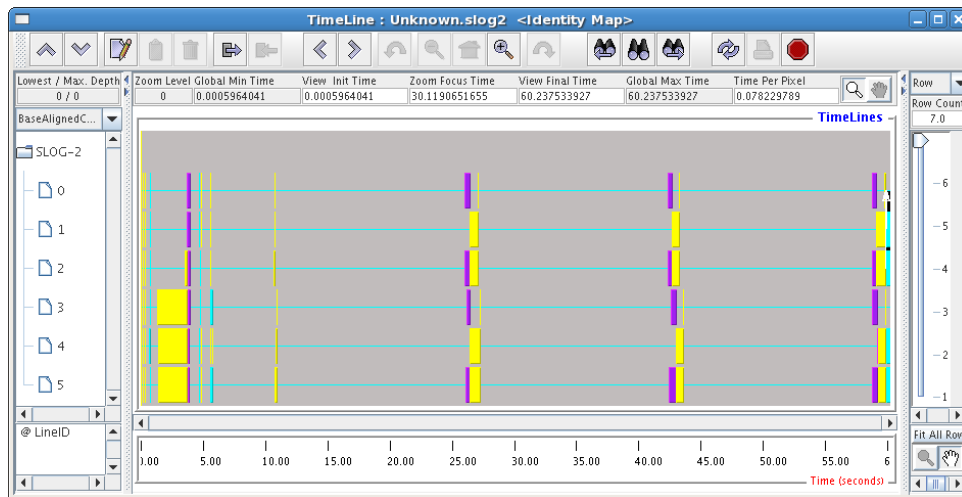


Abbildung 1: Visualisierung der MPI Kommunikation von 6 Prozessen einer Anwendung. Gelb zeigt die Dauer einer MPI_Barrier, Lila die einer MPI_Allreduce und Blau die einer MPI_Bcast Operation.

3 Applikationsgerechte Kommunikation

3.1 Anwenderspezifische Datentypen

Unabhängig von der verwendeten Routine erfordert der Aufbau eines neuen Datentypes in MPI immer zwei Schritte:

1. Definition bzw. Beschreibung des neuen Datentyps als Vektor- oder Strukturdatentyp
2. Registrierung, bzw. Bekanntgabe des neuen Datentyps, um ihn für Kommunikation nutzen zu können

3.1.1 Definition von Vektor-Datentypen

Die Beschreibung neuer Vektordatentypen beruht in MPI auf einer Replikation (Vervielfachung) bereits vorhandener MPI-Basis- oder nutzerspezifischer Datentypen. Ausgangspunkt einer solchen Beschreibung ist zunächst ein vorhandener Datentyp (Grundelement). Mehrere solcher Grundelemente werden innerhalb eines Blockes erfasst. Falls es die Anwendung erfordert, lassen sich zwischen zwei aufeinander folgenden Blöcken auch Lücken integrieren. Ist eine kontinuierliche Aufeinanderfolge gewünscht, so sind Schrittweite und Elemente je Block identisch zu setzen. Für den letztgenannten Fall bietet sich in diesem Fall auch der einfachere Vektorkonstruktor `MPI_Type_contiguous()` an, der als Eingabeparameter nur den alten Datentyp und die Anzahl der Elemente erwartet.

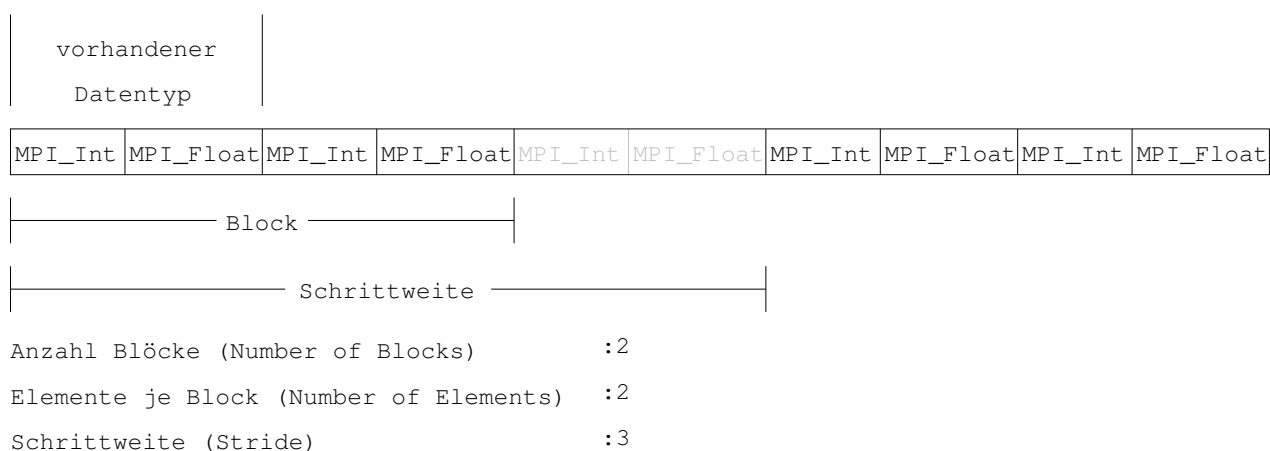


Abbildung 2: Grafische Darstellung des Vektor-Datentyps

3.1.2 Programmfragment zur Errichtung eines neuen MPI-Vektor-Datentyps

```
/* Beschreibung des Datentyps in C */
typedef struct demo_elem_ {
    int    int_l;
    float  float_l;
} demo_elem_t;

demo_elem_t demo[4] ;
/ * Hier muesste zunaechst ein MPI-Datentyp (z.B. MPI_Int_Float)
    errichtet werden, der je ein int und ein float enthaelt
    -> siehe Abschnitt Definition von Struktur-Datentypen */
int Blocks          = 2;
int Blockklen       = 2;
int Stride           = 3;
MPI_Datatype New_Vector;

/* Vector-Datentyp beschreiben */
MPI_Type_vector(Blocks, Blockklen, Stride, MPI_Int_Float, &New_Vector);

/*Vector-Datentyp registrieren */
MPI_Type_commit(&New_Vector);
```

Inhaltlich verwandte MPI-Vektor-Funktionen:

```
MPI_Type_vector(), MPI_Type_hvector(),
MPI_Type_indexed(), MPI_Type_hindexed(),
MPI_Type_size(), MPI_Type_get_extent(),
MPI_Type_commit(), MPI_Type_free()
```

3.1.3 Definition von Struktur-Datentypen

Strukturen werden oft verwendet, um Daten verschiedenen Typs zu einer Verwaltungseinheit zusammenzufassen und eine bessere logische Strukturierung des Programms zu erreichen. Neben diesem der Übersichtlichkeit dienenden verwaltungstechnischen Aspekt existieren auch praktisch bedeutsame Vorteile. Statt einer Funktion mehrere einzelne Argumente zu übergeben, ist es vom Standpunkt der Laufzeit wesentlich effektiver, nur die Anfangsadresse (Pointer) einer Struktur zu übermitteln (innerhalb eines Adressraumes). `MPI_Type_create_struct()` ist die allgemeinste und universellste Routine zur Beschreibung nahe zu beliebiger anwenderspezifischer Datentypen (Vektoren eingeschlossen). Die Definition des neuen Datentyps erfolgt über drei Arrays (Maps).

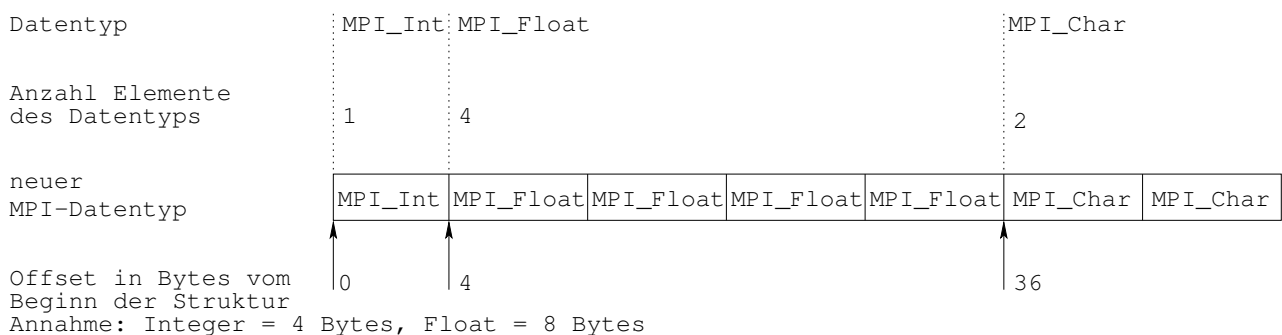


Abbildung 3: Grafische Darstellung des Struktur-Datentyps

```
Type-Map = { MPI_INT, MPI_FLOAT, MPI_CHAR }
Block-Map = { 1, 4, 2 }
Displ-Map = { 0, 4, 36 }
```

Type-Map Bekanntgabe der verwendeten MPI-Datentypen. Die Beschreibung der Nutzer-Struktur erfolgt durch die Auflistung aller enthaltenen Datentypen und muss nicht zwingend ihrer Speicherreihenfolge entsprechen. Die richtige Zuordnung von Datentyp zu Anfangsadresse ab Beginn der Struktur ergibt sich aus zwei weiteren Maps.

Blocklängen-Map Beschreibt in Übereinstimmung mit der Type-Map den Wiederholungsfaktor der entsprechenden Datentypen.

Displacement-Map Gibt die Startadresse jedes Datentyps in der Type-Map als Offset (Verschiebung) zum Beginn der Datenstruktur an. Beispiel für Beschreibung eines MPI-Strukturtyps. Als Datentyp wird hierbei MPI_Aint genutzt, der je nach verwendeter Architektur die richtige Größe für Speicheradressen hat.

3.1.4 Programmfragment zur Errichtung eines neuen MPI-Strukturdatentyps

```
typedef struct demo_struct_ {
    int    int_1;
    float  float_1, float_2, float_3, float_4;
    char   char_1, char_2;
} demo_struct_t;

demo_struct_t demo;
MPI_Datatype New_Struct;
MPI_Datatype Type[] = {MPI_INT, MPI_FLOAT, MPI_CHAR};
int          Block[] = {1, 4, 2 };
MPI_Aint     Disp[3];

/* Adressen der Blockanfänge innerhalb der Struktur ermitteln */
MPI_Address(&demo.int_1, Disp);
MPI_Address(&demo.float_1, Disp + 1);
MPI_Address(&demo.char_1, Disp + 2);

/* Adressen zu Displacements konvertieren */
Disp[2] -= Disp[0];
Disp[1] -= Disp[0];
Disp[0] = 0;

/* neuen MPI-Datentyp beschreiben */

MPI_Type_create_struct(3, Block, Disp, Type, &New_Struct);

/* neuen MPI-Datentyp registrieren */

MPI_Type_commit(&New_Struct);
```

Inhaltlich verwandte MPI-Struktur-Funktionen:

```
MPI_Type_create_struct(), MPI_Address(),
MPI_Type_size(), MPI_Type_get_extent(),
MPI_Type_commit(), MPI_Type_free()
```

3.1.5 Kommunikation mit gepackten Daten

Eine weitere Möglichkeit Daten verschiedenen Typs oder nichtkontinuierlich gespeicherte Daten in einem einzigen Kommunikationsschritt zu übertragen, ergibt sich durch die Nutzung gepackter Daten. Der Begriff gepackte Daten verdeutlicht bereits die grundsätzliche Handhabung. Um eine Nachricht zu versenden hat der Nutzer seine Daten in einem speziellen Puffer abzulegen (zu packen). Auf diese Weise lassen sich mehrere verschiedene Datentypen zu einem einzigen MPI-Datentyp (MPI_PACKED) zusammenfassen. Der Empfänger hat nach erfolgreichem Empfang die Daten richtig zu extrahieren. Dazu benötigt er allerdings Kenntnis über die Ablagereihenfolge der Daten bzw. Kenntnis zu einem bestimmten

gemeinsamen Protokoll. Diesem erhöhten Aufwand des Packens und Entpackens steht der Vorteil gegenüber, dass der Empfänger vor und während des eigentlichen Empfangs nicht informiert sein muss, wie viel Daten welchen Typs zu empfangen sind. Er muss lediglich das benutzte Protokoll kennen (z. B. Integer zur Kennzeichnung des Datentyps, Integer zur Kennzeichnung der Anzahl noch kommender Elemente des zurückerkannten Datentyps, Datenelemente selbst).

3.1.6 Programmfragment zur Kommunikation mit gepackten Daten

```
#define BUFFERSIZE 1024
#define MASTER 0
#define TAG 17

typedef struct demo_struct_ {
    int    int_1;
    float  float_1, float_2, float_3, float_4;
    char   char_1, char_2;
} demo_struct_t;
demo_struct_t demo;

static char Buffer[BUFFERSIZE] ;

int Position, Bytes;
MPI_Status Status;

/* Master packt Daten in Puffer */
if (myid==0) {
    Position = 0;
    MPI_Pack(&demo.int_1, 1, MPI_INT, Buffer, BUFFERSIZE, &Position, MPI_COMM_WORLD);
    MPI_Pack(&demo.float_1, 4, MPI_DOUBLE, Buffer, BUFFERSIZE, &Position, MPI_COMM_WORLD);
    MPI_Pack(&demo.char_1, 2, MPI_CHAR, Buffer, BUFFERSIZE, &Position, MPI_COMM_WORLD);

    /* Sende Größe der gepackten Daten an SLAVE */
    MPI_Send(&Position, 1, MPI_INT, 1, TAG, MPI_COMM_WORLD);
    /* Senden der Daten als MPI_PACKED */
    MPI_Send(Buffer, Position, MPI_PACKED, 1, TAG, MPI_COMM_WORLD);
}

/* Slave empfaengt und entpackt */
if (myid!=0) {
    Position = 0;
    /* Empfange Größe der gepackten Daten */
    MPI_Recv(&Bytes, 1, MPI_INT, MASTER, TAG, MPI_COMM_WORLD, &Status);
    /* Empfange Daten */
    MPI_Recv(Buffer, Bytes, MPI_PACKED, MASTER, TAG, MPI_COMM_WORLD, &Status);

    /* Entpacken */
    MPI_Unpack(Buffer, Bytes, &Position, &demo, MPI_INT, 1, MPI_COMM_WORLD);
    MPI_Unpack(Buffer, Bytes, &Position, &demo, MPI_DOUBLE, 4, MPI_COMM_WORLD);
    MPI_Unpack(Buffer, Bytes, &Position, &demo, MPI_CHAR, 2, MPI_COMM_WORLD);
}
```

4 MPI Profiling mit MPE

Mit Hilfe von MPE lassen sich Flaschenhälse in MPI-Applikationen entdecken. Der Quellcode lässt sich von [4] herunterladen und enthält neben den nötigen Bibliotheken auch das Visualisierungstool *Jumpshot*.

4.1 Installation

Im RA-Pool wurde MPE schon installiert. Für eine eigene Installation in einer anderen Umgebung benötigt MPE zur Kompilierung eine installierte MPI-Implementierung wie Open MPI, sowie C-Compiler (gcc) und Fortran-Compiler (gfortran). Das Visualisierungstool Jumpshot benötigt darüber hinaus ein Java Development Kit (JDK). Nach dem Download und dem Entpacken des Quellcodes, muss `configure` und `make` aufgerufen werden:

```
[user@host mpe2-1.3.0] MPI_CC=mpicc MPI_F77=mpif77 F77=gfortran ./configure --prefix=${HOME}/mpe
[user@host mpe2-1.3.0] make
[user@host mpe2-1.3.0] make install
```

4.2 Verwendung

Der folgende Befehl linkt die MPE Library an ein MPI-Programm:

```
mpicc u2.c -o u2 -llmpe -lmpe
```

Jetzt kann dieses Programm normal gestartet werden. Nach der Beendigung, wird eine Log-Datei im aktuellen Arbeitsverzeichnis abgelegt:

```
[user@host work]$ mpirun -np 3 u2
Max Energie für Atom 0: 380.000000
Max Energie für Atom 1: 382.000000
Max Energie für Atom 2: 384.000000
Max Energie für Atom 3: 386.000000
Max Energie für Atom 4: 388.000000
Max Energie für Atom 5: 390.000000
Max Energie für Atom 6: 392.000000
Max Energie für Atom 7: 394.000000
Max Energie für Atom 8: 396.000000
Max Energie für Atom 9: 398.000000
Writing logfile....
Enabling the Default clock synchronization...
Finished writing logfile u2.clog2
```

4.3 Visualisierung

Mit Hilfe von Jumpshot kann die erstellte Logdatei nun analysiert werden. Jumpshot befindet sich im `bin`-Verzeichnis bzw. im RA-Pool schon im `PATH` und kann somit einfach mit `jumpshot` gestartet werden. Nach dem Laden von „clog2“-Dateien, fragt Jumpshot, ob diese in das „SLOG-2“-Format konvertiert werden sollen. Dies kann bestätigt, durch einen Klick auf „Convert“ getan und durch einen Klick auf „OK“ abgeschlossen werden.

5 Aufgaben

5.1 Kontrollfragen

1. Beschreiben Sie mit Hilfe von `MPI_Type_create_struct()` einen normalen Vektor.
2. Welche Möglichkeiten gibt es in MPI, Daten von einem vorher nicht bestimmten Prozess zu empfangen?

5.2 Praktische Aufgaben

1. Verändern Sie das Vorgabeprogramm so, dass zwei `int`-Werte und vier `double`-Werte an alle Prozesse per Broadcast geschickt werden. Dabei soll weiterhin nur ein *einzelner* Broadcast genutzt werden.

2. Analysieren Sie das Kommunikationsverhalten des MPI-Programmes aus dem 2. Versuch mit Hilfe von MPE.
3. *Zusatz:* Verwenden Sie anstelle von MPE den Intel Trace Analyzer und Collector [5] zur Analyse.

5.3 Hinweise

Statt `MPI_Address()` kann man auch `MPI_Type_get_extent()` (siehe [1] für Details) nutzen, um die relativen Positionen der Datenelemente in Aufgabe 1 zu berechnen. Bis auf die Änderungen zur Bestimmung der Positionen, müssen keine Änderungen an MPI-Aufrufen vorgenommen werden.

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/>
- [4] *MPE (MPI Parallel Environment)*
<http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm>
- [5] *Intel Trace Analyzer and Collector*
<https://software.intel.com/en-us/intel-trace-analyzer>