

Parallele Eingabe/Ausgabe

- ▶ Einleitung, Konzepte, Definitionen
- ▶ Einfache E/A
- ▶ Nichtzusammenhängende Zugriffe
- ▶ Kollektive Aufrufe
- ▶ Nichtblockierende E/A
- ▶ Gemeinsame Dateizeiger
- ▶ Dateiformate
- ▶ Leistungsaspekte
- ▶ Die Implementierung

Parallele Eingabe/Ausgabe

Die zehn wichtigsten Fragen

- ▶ Was ist MPI-2 I/O und wozu braucht man es?
- ▶ Welche Konzepte gibt es dabei?
- ▶ Wie ist eine Datei strukturiert?
- ▶ Wie geht die einfachste E/A?
- ▶ Wie funktionieren nichtzusammenhängende Zugriffe?
- ▶ Wie funktionieren kollektive Aufrufe?
- ▶ Wie funktioniert nichtblockierende E/A?
- ▶ Wozu verwendet man gemeinsame Dateizeiger?
- ▶ Wie optimiert man die Leistung?
- ▶ Welche Implementierung gibt es?

Was ist MPI-2 I/O

- ▶ Erweiterung des MPI-Standards um parallele Eingabe/Ausgabe
- ▶ Wird definiert im MPI-2-Standard
- ▶ Semantik ist analog zum Nachrichtenaustausch von Prozessen
 - ▶ Z.B. collective, nonblocking werden auf E/A übertragen
 - ▶ E/A äquivalent zum Senden und Empfangen von Nachrichten

Wozu parallele E/A in MPI?

- ▶ Leistungsgewinn
 - ▶ Z.B. durch kollektive Aufrufe
 - ▶ Z.B. durch asynchrone E/A
 - ▶ Z.B. durch spezielle Dateisichten
- ▶ Einfacherer Zugriff durch Problemanpassung
 - ▶ Z.B. abgeleitete Datentypen bei irregulären Daten
 - ▶ Dadurch auch Portabilität in heterogenen Umgebungen

Konzepte der MPI-I/O

- ▶ File View (Dateisicht)
 - ▶ Prozessbezogene Sicht auf die Daten einer Datei
- ▶ File Pointer (Dateizeiger)
 - ▶ Individueller/gemeinsamer Dateizeiger
- ▶ Noncontiguous Access (Nichtzusammenhängender Zugriff)
- ▶ Collective Call (Kollektiver Aufruf)
- ▶ Hints (Hinweise)
 - ▶ Informationen für die Implementierungsschicht

Einige Definitionen

- ▶ **file (Datei)**
 - ▶ Eine geordnete Sammlung typisierter Daten
 - ▶ Zugriff erfolgt wahlfrei oder sequentiell
 - ▶ Kollektives Öffnen durch eine Gruppe von Prozessen
- ▶ **displacement (Versatz)**
 - ▶ Eine absolute Byte-Position relativ zum Dateianfang, an der eine Dateisicht beginnt
- ▶ **etype (elementary datatype)**
 - ▶ Die Einheit, mit der auf die Datei zugegriffen und in ihr positioniert wird

Einige Definitionen...

► filetype (Dateityp)

- Schablone, nach der eine Datei aufgebaut wird
- Besteht aus etype's und gleichgroßen Löchern

etype

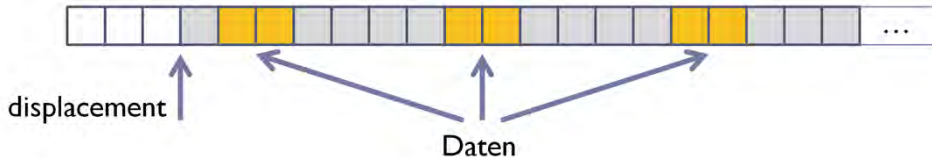


filetype



← Löcher

Aufbau einer Datei



Einige Definitionen...

▶ view (Prozeßdateisicht)

- ▶ Definiert durch displacement, etype und filetype

etype



process 0 filetype



process 1 filetype



process 2 filetype



Aufbau einer Datei



Einige Definitionen...

- ▶ offset (Versatz)
 - ▶ Position in der Datei relativ im aktuellen view ausgedrückt durch die Anzahl der etype's
- ▶ file size (Dateigröße)
 - ▶ Anzahl Bytes ab dem Anfang der Datei
- ▶ file pointer (Dateizeiger)
 - ▶ Intern von MPI verwalteter Versatz
 - ▶ individual file pointer: jeder Prozeß hat einen
 - ▶ shared file pointer: alle Prozesse teilen sich einen
- ▶ file handle (Datei-Handle ☹)
 - ▶ Wie üblich

Einfache E/A: mehrere Prozesse lesen/schreiben Datei

- ▶ Prozesse öffnen (kollektiv!) eine Datei, ...
`MPI_FILE_OPEN`
- ▶ ... jeder Prozeß positioniert mit seinem eigenen Dateizeiger ...
`MPI_FILE_SEEK`
- ▶ ... und liest aus der Datei/schreibt in die Datei ...
`MPI_FILE_READ`
`MPI_FILE_WRITE`
- ▶ ... und schließt die Datei
`MPI_FILE_CLOSE`

Einfache E/A: Prototypen (C)

```
int MPI_File_open (MPI_Comm comm,  
    char *filename, int amode, MPI_Info info,  
    MPI_File *fh)
```

```
int MPI_File_seek (MPI_File fh, MPI_Offset,  
    int whence)
```

```
int MPI_File_read (MPI_File fh, void *buf,  
    int count, MPI_Datatype datatype,  
    MPI_Status *status)
```

```
int MPI_File_write (MPI_File fh, void *buf,  
    int count, MPI_Datatype datatype,  
    MPI_Status *status)
```

```
int MPI_File_close (MPI_File *fh)
```

Datenzugriff: Positionierung

- ▶ Drei Varianten der Positionierung
 - ▶ Explicit offsets
 - ▶ Individual file pointers
 - ▶ Shared file pointers
- ▶ Können innerhalb eines Programms gemischt verwendet werden
- ▶ Syntax
 - ▶ Explicit offsets: `MPI..._AT`
 - ▶ Shared: `MPI..._SHARED`, `MPI..._ORDERED`

Nichtzusammenhängende Zugriffe und kollektive Aufrufe

- ▶ Bisher vorgestellte E/A auch durch üblich Unix-E/A bewerkstelligbar: eine Datei, zusammenhängende Daten
- ▶ Aber: parallele Programme greifen oft mit mehreren Prozessen unabhängig und auf nichtzusammenhängende Positionen einer Datei zu
- ▶ MPI-2 I/O bietet Funktionen, die mit **einem** Aufruf nichtzusammenhängende Daten lesen können und es mehreren Prozessen gestatten, gleichzeitig auf die Datei zuzugreifen

Nichtzusammenhängende Zugriffe: Dateisicht

- ▶ Durch Dateisichten sieht jeder Prozeß nur „seine“ Daten
- ▶ Dateisicht definiert durch
 - ▶ displacement, etype, filetype
etype und filetype sind Standard-Datentypen oder aus ihnen abgeleitete Datentypen!
- ▶ Dateisicht definiert durch
MPI_FILE_SET_VIEW
- ▶ Löcher müssen auch definiert werden
MPI_TYPE_CREATE_RESIZED

Nichtzusammenhängende Zugriffe: Beispiel


```
/* 2 MPI_INT zusammenhängend als derived data type */
MPI_Type_contiguous(2,MPI_INT,&contig);

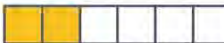
/* 4 Löcher anhängen; ergibt Größe 6 */
lower_boundary=0;
extent=6*sizeof(int);
MPI_Type_create_resized(contig,lower_boundary,extent,
    &filetype);

/* und machen den neuen Typ bekannt */
MPI_Type_commit(&filetype);

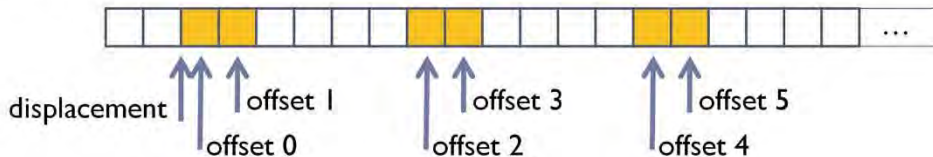
/* und jetzt die Dateisicht */
MPI_File_set_view(filehandle,displacement,etype,filetype,
    "native",MPI_INFO_NULL);
```


Nichtzusammenhängende Zugriffe: Beispiel

etype = MPI_INT 

filetype = 2*MPI_INT resized zur Größe 6 

Aufbau einer Datei



Kollektive Aufrufe

- ▶ Zur weiteren Optimierung können alle Prozesse gleichzeitig in der Datei zugreifen
- ▶ Definition einer Sicht wie zuvor, zusätzlich aber spezielle Funktionen
 - ▶ Erlaubt es der MPI-Implementierung, Zugriffe mehrerer Prozesse zu optimieren
- ▶ Selbst wenn jeder Prozeß nur kleine, unzusammenhängende Stücke liest, kann die MPI-Implementierung (womöglich) einen großen, zusammenhängenden Zugriff daraus erstellen
- ▶ **`MPI_FILE_READ_ALL`, `MPI_FILE_WRITE_ALL`**

Nichtblockierende E/A

- ▶ Wird verwendet, um E/A mit Kommunikation und/oder Berechnung zu überlappen
- ▶ Alle nicht-kollektiven(!) Lese- und Schreibfunktionen haben nichtblockierende Entsprechungen
 - ▶ Zur Überprüfung der Beendigung kommt die Standard-MPI-Test-Funktion zum Einsatz
- ▶ Namenskonvention: **MPI_FILE_I...**
Also z.B. **MPI_FILE_IREAD**

Gemeinsamer Dateizeiger

- ▶ Bisher nur individuelle Zeiger und Versatz
- ▶ Ebenso möglich: gemeinsamer Zeiger
 - ▶ Von allen Prozessen gemeinsam genutzt
 - ▶ Jeder Zugriff irgendeines Prozesses verändert die Position
 - ▶ Nächster zugreifender Prozess sieht neue Position
- ▶ Funktionen
 - `MPI_FILE_SEEK_SHARED`
 - `MPI_FILE_READ_SHARED`
 - `MPI_FILE_WRITE_SHARED`

Gemeinsamer Dateizeiger...

- ▶ Bei kollektiven Aufrufen kann gemäß dem Rang der Prozesse serialisiert werden

`MPI_FILE_READ_ORDERED`

`MPI_FILE_READ_ORDERED_BEGIN`

- ▶ Typischer Anwendungsfall
 - ▶ Gemeinsame Protokolldateien

Hinweise (hints)

- ▶ Hinweise geben dem Nutzer die Möglichkeit, Informationen an die MPI-Implementierung durchzureichen
- ▶ Beispiele für Hinweise sind hier
 - ▶ Anzahl der Festplatten, über die eine Datei verteilt werden soll (striping)
 - ▶ Breite der Streifen (stripsize)
- ▶ Hinweise sind immer optional, der Benutzer muss sie nicht angeben
 - ▶ Gleichzeitig darf eine Implementierung Hinweise beliebig ignorieren

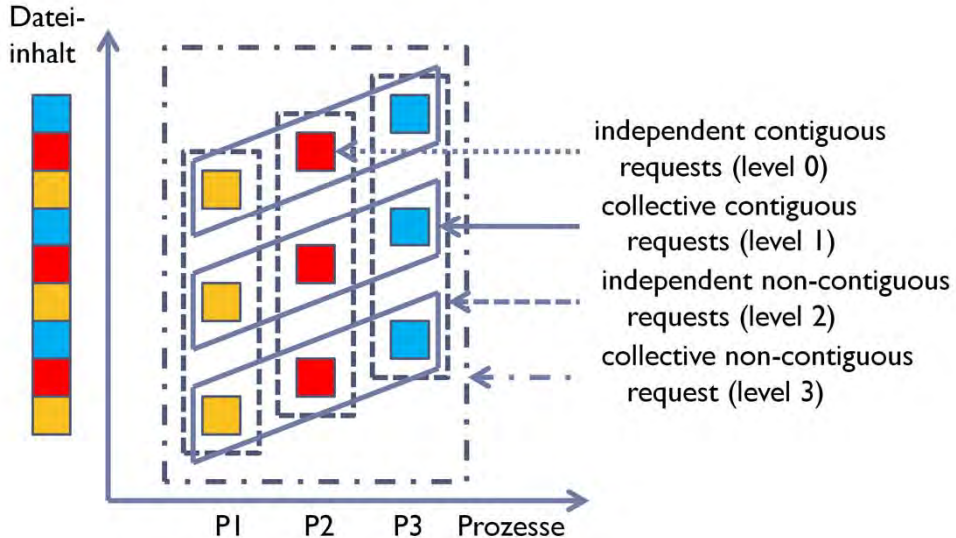
Dateiformate

- ▶ Dateien werden als Folge von Bytes gesehen
Die konkrete Abspeicherung ist Sache der Implementierung
- ▶ MPI definiert drei Daten-Repräsentationen, die unterschiedliche Portabilität erlauben
 - ▶ „native“: keine Wandlung (=Speicherabbild)
schnell und nichtportabel
 - ▶ „internal“: portabel zwischen den Plattformen, die diese MPI-Implementierung unterstützt
 - ▶ „external32“: 32-bit big endian; portabel zu jeder MPI-Implementierung auf jeder Architektur; langsam

Leistungsaspekte

- ▶ Die Wahl der geeignetsten E/A-Methode bestimmt die erzielbare E/A-Bandbreite
 - ▶ Zusammenhängend / nichtzusammenhängend
 - ▶ Kollektiv / nichtkollektiv
- ▶ Beispiel
 - ▶ Datei einer 3x3-Matrix komplexer Datentypen

Leistungsaspekte...



Implementierung ROMIO

- ▶ ROMIO ist eine/(die) Implementierung von MPI-2 I/O
 - ▶ Gehört zu MPICH, kann aber separat verwendet werden, insbesondere in anderen MPI-Implementierungen
- ▶ ROMIO unterstützt eine Reihe von Hardware-Architekturen und Dateisystemen
- ▶ ROMIO unterstützt alle(?) Merkmale von MPI-2 I/O

Parallele Eingabe/Ausgabe

Zusammenfassung

- ▶ Parallele E/A analog zur Kommunikation definiert
- ▶ Verwendet auch abgeleitete Datentypen
- ▶ Dateien sind eine Sequenz elementarer Datentypen
- ▶ Jeder Prozess hat seine eigene Dateisicht
- ▶ Wir positionieren explizit, mit individuellen Dateizeigern oder einem gemeinsamen
- ▶ Nichtzusammenhängende Zugriffe erhöhen die Effizienz
- ▶ Kollektive Aufrufe erhöhen die Effizienz
- ▶ ROMIO ist die Standard-Implementierung