Tutorium 11: Parallelprogrammierung mit MPI

Paul Brinkmeier

13. Januar 2020

Tutorium Programmierparadigmen am KIT

EULENFEST 23.01.2020

Infobau 50.34 ab 19 Uhr







Helfer werden! https://www.redseat.de/eulenfest20/

Heutiges Programm

Parallelprogrammierung

ProPa-Stoff zu Parallelprogrammierung:

- Grundlegende Begriffe
- Message Passing, wurde in OS kurz behandelt ("message queues")
- Shared Memory + Synchronisierung, wie in SWT1, OS, etc.
 - In Java, mit ein paar Details zur JVM

Begriffe

Flynns Taxonomie

- SISD: Single Instruction, Single Data
 Ein Datum wird von einer Ausführungsarbeit bearbeitet
- SIMD: Single Instruction, Multiple Data
 Eine Ausführungseinheit bearbeitet mehrere Daten gleichzeitig
- MIMD: Multiple Instruction, Multiple Data
 ≈ Mehrere Ausführungseinheiten arbeiten gleichzeitig
- MISD: Multiple Instruction, Single Data
 ≈ Mehrere Ausführungseinheiten arbeiten gleichzeitig an einem Datum

Flynns Taxonomie

- SISD: Single Instruction, Single Data
 Ein Datum wird von einer Ausführungsarbeit bearbeitet
- SIMD: Single Instruction, Multiple Data
 Eine Ausführungseinheit bearbeitet mehrere Daten gleichzeitig
- MIMD: Multiple Instruction, Multiple Data
 ≈ Mehrere Ausführungseinheiten arbeiten gleichzeitig
- MISD: Multiple Instruction, Single Data
 ≈ Mehrere Ausführungseinheiten arbeiten gleichzeitig an einem Datum

Beispiele?

Daten- und Taskparallelismus

Parallele Probleme sind üblicherweise entweder

- "datenparallel": Problem kann auf identische Ausführungseinheiten verteilt werden Beispiel: map primeFactors [1432793, 651433, ...]
- "taskparallel": Problembestandteile sind nicht homogen Beispiel: Videospiel mit Render-, Netzwerk- und Logikprozessen

Datenparallele Probleme sind i.d.R. einfacher zu behandeln (auch: "embarrassingly parallel"). Bei manchen Problemen verschwimmt die Grenze auch (bspw. Webserver).

MPI-Basics

MPI

MPI ("Message Passing Interface") ist ein Standard für Parallelprogrammierung. Es existieren verschiedene Implementierungen für verschiedene Sprachen. Die VL verwendet Open MPI, eine Open-Source-Implementierung.

- MPI-,, Prozesse" beziehen sich i.d.R. auf Prozessorkerne
- Man verwendet Message Passing statt Shared Memory
 - Daten werden explizit über Send und Recv geteilt
- MPI-Prozesse werden in sog. Communicators eingeteilt. Wir verwenden immer den Communicator, der alle Prozesse enthält (MPI_COMM_WORLD))

Installation von MPI

MPI-Beispiel gehen von Linux-Systemen aus, verwendet unter Windows bitte WSL.

- apt install openmpi-bin (Ubuntu)
- pacman -S openmpi (Arch Linux)
- dnf install openmpi (Fedora)

Dann mpicc --version zum Testen der Installation.

Bauen und Ausführen von MPI-Programmen

MPI-Programme werden mit mpicc (Wrapper um gcc) kompiliert:

```
cd demos/mpi/hello
mpicc -o hello hello.c # oder make
```

Um ein Programm auszuführen, wird mpirun verwendet:

```
mpirun [--oversubscribe] -np [N] ./hello
```

- N ist die Zahl der Prozesse, die ausgeführt werden sollen
- --oversubscribe braucht ihr, falls N größer als die Zahl eurer Prozessorkerne ist
- Vergleicht die Ausgabe der Demos hello und sendrecv

Send/Recv

Per Send und Recv werden Daten zwischen Prozessen ausgetauscht.

Die Aufrufe sind unabhängig vom Medium (IPC, Sockets, ...).

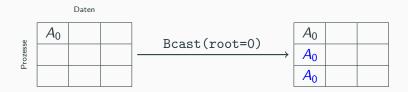


- int MPI_Send(buf, count, datatype, dest, tag, comm)
- int MPI_Recv(buf, count, datatype, source, tag, comm, status)

Kollektive Operationen

Bcast

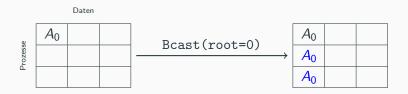
Bcast verteilt ein Datum auf alle Prozesse.



- int MPI_Bcast(buf, count, datatype, root, comm)
- Daten befinden sich ursprünglich auf root
 - → Fallunterscheidung in Bcast:
 - if rank == root then forall others: send() else recv()

Bcast

Bcast verteilt ein Datum auf alle Prozesse.



- int MPI_Bcast(buf, count, datatype, root, comm)
- Daten befinden sich ursprünglich auf root
 - → Fallunterscheidung in Bcast:
 - if rank == root then forall others: send() else recv()

Implementiert custom_Bcast in demos/mpi/custom_broadcast!

Scatter

Scatter verteilt eine Liste von Daten auf mehrere Prozesse.



- int MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
- sendcount, recvcount: Zahl der Elemente, die an einen Prozess verteilt werden
- i.d.R.: sendcount == recvcount

Gather

Gather sammelt Daten von allen Prozessen in einer Liste.



- int MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
- sendcount, recvcount: Zahl der Elemente, die an einen Prozess verteilt werden
- i.d.R.: sendcount == recvcount

Scatter und Gather

Scatter und Gather sind "invers":

```
int nums[4];
int local:
if (rank == 0) \{ nums = \{0, 1, 2, 3\}; \}
MPI_Scatter(nums, 1, MPI_INT, &local, 1, MPI_INT,
    O, MPI_COMM_WORLD);
// in P_i gilt: local = i
MPI_Gather(&local, 1, MPI_INT, nums, 1, MPI_INT,
    O, MPI_COMM_WORLD);
```

Dieser Code hat keine Effekte außer Seiteneffekte.

Aufgabe zu Scatter und Gather

Implementiert folgendes Programm mit MPI:

- n: Prozessoranzahl (MPI_Comm_size), x: 10
- P_0 legt long-Liste mit Elementen $[1, 2, ..., N \cdot x]$ an
- P_i summiert einen x-Ausschnitt der Liste mit $i \in [0; N)$
- P₀ summiert die einzelnen Summen

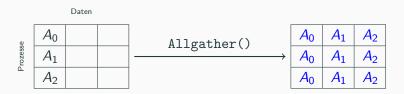
Verwendet dafür:

- MPI_Comm_size, MPI_Comm_rank
- MPI_Scatter
- MPI_Gather

Dokumentation für MPI-Funktionen bekommt ihr mit man <f>

Allgather

Allgather ist die Verknüpfung von Gather und Bcast.



- int MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
- Im Gegensatz zu Gather gibt es keinen Parameter root

Alltoall

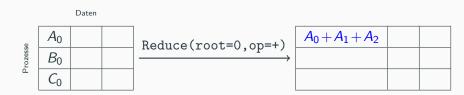
Alltoall stückelt Daten von jedem Prozess und verteilt sie.



- int MPI_Alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
- Es führt sozusagen jeder Prozess einmal Scatter aus

Reduce

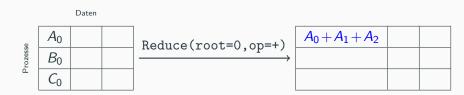
Reduce wendet eine assoziative Operation auf verteilte Daten an.



- int MPI_Reduce(sendbuf, recvbuf, count, type, op, root, comm)
- *Ungefähr* dasselbe wie ein Fold!

Reduce

Reduce wendet eine assoziative Operation auf verteilte Daten an.



- int MPI_Reduce(sendbuf, recvbuf, count, type, op, root, comm)
- Ungefähr dasselbe wie ein Fold!
- Ersetzt den letzten Teil der Summenaufgabe durch einen Aufruf zu Reduce

Ende

Ende

- Im Campus-System kann man sich bis zum 17.03. für die ProPa-Klausur anmelden
- Ab Mittwoch kann man sich Rückmelden bis zum 15.02.
- Eulenfest am 23.01.!