Übungsblatt 6 zur Vorlesung Parallele Programmierung

Abgabe: 14.12.2024, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik • Otto-von-Guericke-Universität Magdeburg
https://parcio.ovgu.de

Dieses Übungsblatt soll Ihnen die Möglichkeit geben, Ihre ersten Schritte in der Programmierung mit MPI zu machen. Die erworbenen Fertigkeiten werden auf späteren Übungsblätter für komplexere Aufgaben benötigt.

Binden Sie den Header `mpi.h` ein und benutzen Sie den Compiler `mpicc`. Um auf dem Cluster Zugriff auf den MPI-Compiler zu erhalten, müssen Sie die Softwareumgebung mit folgendem Befehl laden (beachten Sie das Leerzeichen zwischen `.` und `/`).

```
$ . /opt/spack/pp-202425/env.sh
```

Ein Tutorial zur MPI-Programmierung finden Sie unter:

`https://hpc-tutorials.llnl.gov/mpi/`

1. Das erste MPI-Programm (180 Punkte)

Erstellen sie ein MPI-Programm `timempi` in C, welches folgende Ausgabe erzeugt:

`HOSTNAME: TIMESTAMP`

`HOSTNAME:` Kurzer Hostname des Rechners, auf dem das Programm ausgeführt wird.

`TIMESTAMP:` Zeitstempel zur Zeit der Ausführung des Programms in einem mindestens auf die Mikrosekunde genauen Format.

Dabei sind folgende Vorgaben zu beachten:

- Die Prozesse mit den Rängen 1 bis n sollen den String `HOSTNAME: TIMESTAMP` bei sich erzeugen und als String per MPI an den Prozess mit Rang 0 senden, welcher die komplette Ausgabe übernimmt. Rang 0 selbst soll dabei keinen eigenen String erzeugen und ausgeben (aber einen Timestamp produzieren).
  (Tipp: Sehen Sie sich die Funktionen `gethostname()` und `gettimeofday()` an. In den Materialien finden Sie `timestamp.c` mit einem Beispiel für die Erzeugung des Zeitstempels.)
- Die Ausgabe soll nach Rang der Prozesse geordnet erfolgen.
- Direkt nach der Ausgabe der empfangenen Strings soll der Prozess mit Rang 0 noch den kleinsten Mikrosekunden-Anteil aller Prozesse (inklusive Rang 0) ausgeben.
  (Tipp: Hierfür können Sie `MPI_Reduce()` verwenden.)

- Die Prozesse sollen alle erst beenden, wenn die Ausgabe komplett erfolgt ist. Das Programm ist falsch, wenn ein Prozess zu früh beenden könnte!
- Direkt vor dem Beenden soll jeder Prozess einen Text ausgeben: „Rang X beendet jetzt!“ (Tipp: Dazu können Sie `MPI_Barrier()` verwenden.)
- Das Programm muss mit beliebig vielen Prozessen lauffähig sein.

Die Ausgabe könnte wie folgt aussehen:

```
ant13: 2024-11-14 13:15:57.968558
ant14: 2024-11-14 13:15:57.968557
968557
Rang 2 beendet jetzt!
Rang 0 beendet jetzt!
Rang 1 beendet jetzt!
```

2. Ringkommunikation mit MPI (120 Punkte)

In den Materialien finden Sie das serielle Programm `circle.c`; modifizieren Sie es mit Hilfe von MPI so, dass sich die parallele Variante wie folgt verhält:

Ein eindimensionales Array der Länge `N` wird auf `nprocs` Prozesse möglichst gleichmäßig aufgeteilt und mit zufälligen Werten (im Bereich 0–24) initialisiert. `N` wird dem Programm dabei als erstes und einziges Kommandozeilenargument übergeben. Die Werte sollen ihrer Reihenfolge im Array nach ausgegeben werden. Die Prozesse sollen einen Ring bilden, wobei jeder Prozess mit seinem Vorgänger und dem Nachfolger kommuniziert (also Rang n mit den Rängen $n-1$ und $n+1$); der letzte Prozess kommuniziert dabei mit Rang 0 und umgekehrt. In der Funktion `circle` tauschen die Prozesse ihre Daten aus. In jedem Schleifendurchlauf versendet jeder Prozess seine Daten an seinen Nachfolger und empfängt die Daten seines Vorgängers. Dies passiert solange bis der letzte Prozess am Anfang seines Arrays den Wert des ersten Arrayelementes des ersten Prozesses vor Iterationsbeginn hat (siehe Abbildung 1). Dann wird der Speicher in der Reihenfolge der Prozesse ausgegeben und das Programm beendet.

Beachten Sie dabei folgende Anforderungen:

- Das Programm soll mit einer beliebigen Anzahl an Prozessen und einer beliebigen Arraylänge umgehen können. Die Eingabe muss auf mögliche Fehler geprüft werden und das Programm soll angemessen reagieren.
- Zu keinem Zeitpunkt darf ein Prozess das gesamte Array im Speicher halten.
- Der Abbruch soll nicht nach Anzahl der Iterationen erfolgen, sondern nach Eintreten der oben beschriebenen Bedingung. Beachten Sie, dass dieser Fall auch nach weniger als `nprocs - 1` Iterationen auftreten kann, da derselbe Wert mehrfach vorkommen kann (siehe Abbildung 1).
- Achten Sie auf die richtige Reihenfolge der Ausgabe.

- Ersetzen Sie Aufrufe von `MPI_Send` durch `MPI_Ssend`, um sicherzustellen, dass das Programm mit beliebigen Nachrichtengrößen lauffähig ist. Andere Varianten wie z. B. `MPI_Isend` oder `MPI_Sendrecv` sind selbstverständlich ebenso zulässig.

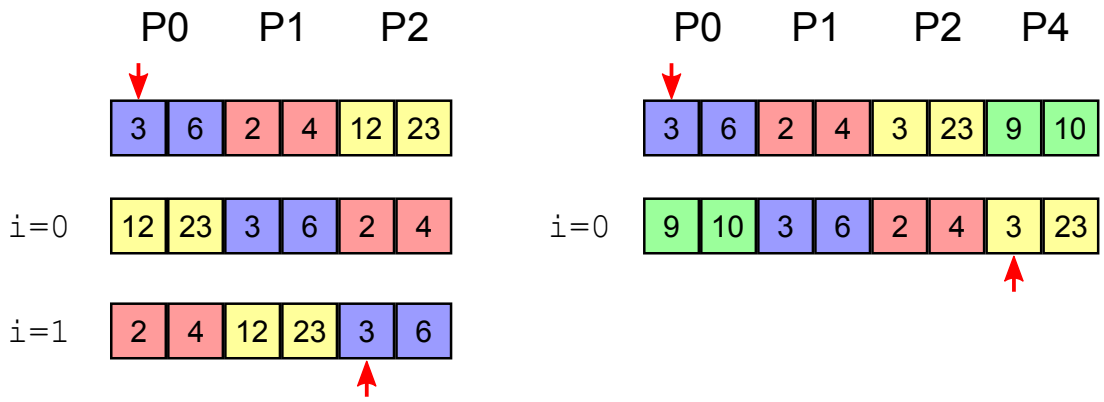

Abbildung 1: Beispiele möglicher Programmabläufe von Initialisierung bis Abbruch (PX bezeichnet Prozess X, i=n die n. Iteration)

Abgabe

Als Abgabe werten wir den letzten Commit vor der Abgabefrist in Ihrem Git-Repository. Im Hauptverzeichnis des Repositories wird ein Verzeichnis `PP-2024-Uebung-06-Materials` mit folgendem Inhalt erwartet:

- Eine Datei `gruppe.md` mit den Gruppenmitgliedern (eines je Zeile) im folgenden Format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- Der Code des `timempi`-Programms im Unterverzeichnis `timempi` (Aufgabe 1)
  - Alle Quellen, aus denen Ihr Programm besteht (`timempi.c`); gut dokumentiert
  - Ein `Makefile` derart, dass `make timempi`, `make clean` und `make` erwartungsgemäße Binärdateien erzeugen bzw. löschen.
- Der überarbeitete Code des `circle`-Programms im Unterverzeichnis `circle` (Aufgabe 2)
  - Alle Quellen, aus denen Ihr Programm besteht (`circle.c`); gut dokumentiert
  - Ein `Makefile` derart, dass `make circle`, `make clean` und `make` erwartungsgemäße Binärdateien erzeugen bzw. löschen.