Übungen zur Vorlesung

Paralleles Höchstleistungsrechnen

Dr. S. Lang, D. Popović

Abgabe: 24. November 2009 in der Übung

Übung 11 Semaphoren Deadlock (3 Punkte)

Betrachten Sie unten stehendes Pseudo-Programm. Erklären Sie, wie es in der Situation zu einem Deadlock kommen kann. Die Argumente des Konmstruktors initialisieren die Semaphore.

```
Semaphore s(1), t(0);

Process Π0 {                    Process Π1 {
  while(true) {                   while(true) {
    t.wait();                       s.wait();
    s.wait();                       t.wait();
    ...;                            ...;
    s.signal();                     t.signal();
    t.signal();                     s.signal();
  }                               }
}                               }
```

Übung 12 Resourcen-Verwaltung (5 Punkte)

In einem System gebe es zwei identische, belegbare Ressourcen und N Bewerber für die Belegung einer der beiden Resourcen. Implementieren Sie mit den `Threadtools` ein Programm auf Basis von Semaphoren, das sicherstellt, dass höchstens zwei Bewerber gleichzeitig die Resourcen belegen können. Die Semaphoren sollen also jedem Bewerber eine Resource zuweisen. Eine mögliche Lösung ist:

- Ein mit `false` initialisiertes (globales) `boolean`-Feld `R[2]` kennzeichnet die Resourcen-Belegung,
- Bewerber, die Zugang erhalten, setzen eine der beiden Resourcen auf `true` (= `belegt`).
- Eine binäre Semaphore sichert den Zugriff auf die Resourcen ab.
- Eine Semaphore, deren Wert $0, 1$ oder 2 sein kann, realisiert den Zugriff auf die Resourcen.

Sie müssen eine Klasse `Consumer` vom `BasicThread` ableiten, welche die Methode `run` überlädt. Ihr Programm könnte in etwa folgendes Layout haben (nicht vollständig):

```
boolean   R[2];   // Resourcen
Semaphore s_res;  // Semaphore der Resourcen
Semaphore s_zgr;  // bin. Semaphore sichert Zugriff auf Resourcen ab
...
class Consumer : public BasicThread {
  void run {
    Warte auf Resourcen;
    Sichere Zugriff auf Resource ab;
    Waehle freie Resource i;
    Entsichere Zugriff;
    Arbeite auf Resource i;
    Gib Resource i frei;
  }
};
```

Testen Sie Ihr Programm, z. B. mit $N = 22$. Ein Beispiel aus dem Leben wären 22 Fussballer die sich nach dem Spiel 2 Duschen teilen müssen ☺.

Übung 13 MPI Ring-Test (5 Punkte)

Mit dieser Aufgabe wollen wir die ersten Schritte mit MPI wagen. Implementieren Sie daher einen Ring, in dem Nachrichten versendet werden. Um Verklemmungen zu vermeiden, sollen in einem Schritt zunächst alle Prozesse mit geradem Rang und im folgenden Schritt alle mit ungeradem Rang an den nachfolgenden Nachbarn senden. Auf der Homepage finden Sie ein in `C` geschriebenes Framework namens `mpi_ring.c`, das Sie vervollständigen sollen. Darin initialisiert jeder Prozess einen Sendepuffer namens `receive` (hier ein Integer) mit beliebigen Daten (hier der Rang). In der Loop wird der `receive` -Puffer, der die gesendete Information in der letzen Iteration erhalten haben sollte, in den Sendepuffer kopiert, um die Information an den nächsten Prozess zu senden. Diese Kommunikation soll nun von Ihnen in der Loop mit `MPI_Ssend` and `MPI_Recv` implementiert werden. Testen Sie Ihr Programm im Pool oder einem anderen Rechner und überprüfen Sie, ob alle Nachrichten im Ring in der richtigen Reihenfolge gesendet wurden (Bitte Output mit abgeben).

Genauere Informationen zur Verwendung von MPI im Pool finden auf der Homepage. Wertvoll sind ausserdem die Man-Pages zu MPI (z.B. `man MPI_Comm_rank`).

Übung 14 Paralleles Berechnen von π mit MPI (7 Punkte)

Aus der Identität $\pi = 4(\arctan 1)$ erhält man durch Ausnutzung der Ableitung des arctan, $(\arctan x)' = 1/(1 + x^2)$, eine Vorschrift zur Berechnung von π:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

Durch Teilen des Intervalls in n äquidistante Teilstücke kann das Integral mit der Mittelpunktsregel ausgewertet werden. Das sequentielle Programm finden auf der Homepage `piseq.c`. Wir wollen es mit MPI parallelisieren. Die Strategie ist:

- Prozess 0 liest die Anzahl der Teilintervalle ein und teilt sie allen anderen Prozessen mit,
- Die `for`-Schleife über die Teilintervalle wird parallelisiert, jeder Prozess berechnet eine lokale Teilsumme. Die Ergebnisse werden von Prozess 0 mit `MPI_Reduce` eingesammelt und die Teilsummen addiert.

Implementieren Sie die parallele Version des Programms. Vergleichen Sie die Genauigkeit in den Rechnungen (Nachkommastellen) mit der sequentiellen Lösung und dem genauen Wert. Bestimmen Sie ausserdem die Konvergenzordnung der Mittelpunktsregel.