Übungen zur Vorlesung

Wissenschaftliches Rechnen – Paralleles Höchstleistungsrechnen

Prof. Dr. C. Engwer, S. Westerheide
http://wwwmath.uni-muenster.de/num/Vorlesungen/WissenschaftlichesRechnen_SS11/

Abgabe 23. 5. 2011. Abg. der Programmieraufgaben per Email an `sebastian.westerheide@uni-muenster.de`, schriftliche Abgabe Dienstags in der Vorlesung.

- Alle Programmierübungen müssen per Email und in ausgedruckter Form abgegeben werden.
- Achten sie darauf, ihr Programm ordentlich zu formatieren und gut zu kommentieren.
  In Zukunft wird die Form mit in die Bewertung eingehen.

Auf diesem Aufgabenblatt sollen Sie sich mit dem in der Vorlesung besprochenen Konzept der Semaphore (Kapitel 4.3) und dessen Anwendung vertraut machen.

Übung 1 Barriere mit Semaphore

Folgende Barriere mit Semaphore ist aus der Vorlesung bekannt (Kapitel 4.3.2).

parallel barrier-2-semaphore
{
  Semaphore $b1 = 0$, $b2 = 0$;
  process Π_1
  {
    while (true) {
      Berechnung;
      V($b1$);
      P($b2$);
    }
  }
  process Π_2
  {
    while (true) {
      Berechnung;
      V($b2$);
      P($b1$);
    }
  }
}

Erweitern Sie diese mittels rekursiver Verdopplung auf 2^d Prozessoren. Erklären Sie, was dafür getan werden muss und formulieren Sie ihre Lösung anschließend in der Pseudosprache aus der Vorlesung.

4 Punkte

Übung 2 Implementierung einer Semaphore

Überlegen Sie sich, wie der abstrakte Datentyp Semaphore mit Hilfe des Konzepts des Mutex realisiert werden kann, insbesondere welche Daten jede Instanz einer Semaphore in diesem Fall halten muss. Implementieren Sie darauf aufbauend einen Datentyp `Semaphore` in `C++` unter Verwendung der Bibliothek `Pthreads`.

- Erstellen Sie dazu einen zusammengesetzten Datentyp

```
struct Semaphore
{
  ...  // Komponenten zur Datenhaltung
};
```

  welcher die Semaphore repräsentiert, sowie eine Menge von Funktionen für die Initialisierung der Instanz einer Semaphore, die Durchführung ihrer `P`- bzw. `V`-Operation und die Freigabe der Instanz einer Semaphore, wenn diese nicht mehr benötigt wird. Diese Funktionen könnten beispielsweise folgende Signaturen besitzen:

```
void init (Semaphore& s, int value)
void P (Semaphore& s)
void V (Semaphore& s)
void destroy (Semaphore& s)
```

- Testen Sie ihre Implementierung anhand des Erzeuger-Verbraucher-Problems (Kapitel 4.3.3), welches Sie in der letzten Übung ohne die Verwendung einer Semaphore implementieren sollten.

Hinweise: Informieren Sie sich gegebenenfalls vor der Bearbeitung der Aufgabe über den Umgang mit zusammengesetzten Datentypen in `C++`. Eine geeignete Quelle hierfür ist Kapitel 9.3 des Informatik I Skriptes, welches auf der Vorlesungsseite verlinkt ist. Falls Sie bereits mit objektorientierter Programmierung in `C++` vertraut sind, dürfen Sie `Semaphore` alternativ auch gerne als Klasse realisieren.

8 Punkte