Prof. Dr. R. Loogen, M. Dieterle
Fachbereich Mathematik und Informatik
Hans-Meerwein-Straße
D-35032 Marburg

17.Oktober 2008

Philipps Universität Marburg

Übungen zu „Parallele Programmierung“, WS 2008/09

Nr. 1, Abgabe der Aufgaben: 24.Oktober 2008 in der Übung

Hinweise:

- Die Übungsblätter erscheinen zu den Übungen und sind auch dort wieder abzugeben.
- Programmieraufgaben senden Sie bitte **zusätzlich** per E-Mail an Ihren Tutor.
- Es werden nur fehlerfrei compilierbare Programme bepunktet.

Benutzung von Linux, Netz des Fachbereichs

In den praktischen Übungen zur Vorlesung werden Programmiersprachen und Systeme benutzt, die zum Teil nur unter Linux verfügbar sind. Auch wenn das HOME-Verzeichnis unter Windows als Laufwerk *U:* sichtbar ist, müssen Sie mit Hilfe einer Konsole (Shell) Programme übersetzen und ausführen. Machen Sie Sich möglichst bald mit der Bedienung einer Linux-Konsole und den gängigsten Kommandos vertraut.

Windows: Aufruf des *SSH-Client* (Programme-Network) und Angabe eines Rechnernamens. Dies öffnet eine Konsole auf dem angegebenen Rechner. Sie können mit Windows-Programmen in Ihrem HOME arbeiten und die Konsole zum Übersetzen und Ausführen verwenden.

Linux: Sie können einen der Linux-Rechner in den Räumen D4424 oder D5437 benutzen. Dort arbeiten Sie unter einer grafischen Benutzeroberfläche und können eine Konsole öffnen (Systemwerkzeuge-Terminal).

Benutzbare Rechner (in den PC-Arbeitsräumen) sind die folgenden:
`tanga, annaba, kananga, yola, luena, boende, tamale`

Programmiersprache MPD

Im ersten Teil der Vorlesung wird die Sprache MPD (Multithreaded, Parallel, and Distributed Programming) benutzt. Die Sprache ist am Fachbereich unter Linux installiert. Zur Benutzung fügen Sie */app/lang/parallel/mpd/bin* zu Ihrem Suchpfad hinzu. Sie können den Pfad permanent erweitern, je nach Shell

bash: in der Datei *~/.bashrc* durch die Zeile: `export PATH=$PATH:/app/lang/parallel/mpd/bin`

tcsh: in der Datei *~/.tcshrc* durch die Zeile: `setenv PATH ${PATH}:/app/lang/parallel/mpd/bin`

Die Syntax der Sprache MPD ist ähnlich wie die von C (die Vorgängersprache SR war dagegen syntaktisch an Pascal angelehnt).

- Anweisungen und Deklarationen werden durch Semikolon oder Zeilenwechsel getrennt.
- Blöcke von Anweisungen und Deklarationen werden in geschweifte Klammern gesetzt. Parameter für Prozeduren (und andere Konstrukte) stehen in runden Klammern, Indices für Arrayzugriffe und Schleifenvariablen in eckigen.
- Variablen müssen (irgendwo!) vor der ersten Benutzung deklariert werden,

Bitte wenden!

- Einzeilige Kommentare werden mit # eingeleitet, mehrzeilige Kommentare in `/* ...*/` eingeschlossen.
- Die zentralen Konzepte in MPD sind virtuelle Machinen (`vm`), Ressourcen (`ressource`) und globale Objekte (`global`).

  Eine *virtuelle Machine* ist ein Adressbereich (und entspricht i.W. einer "realen" Maschine im Netz). Implizit wird stets eine VM für das Hauptprogramm angelegt.

  Eine *Ressource* ist eine Schablone ähnlich wie eine Klassendefinition: sie definiert eine Einheit von lokalen Variablen, Prozeduren und Prozessen und kann mehrfach instanziert werden. Die im Sourcecode *als letztes* definierte Resource ist das Hauptprogramm und wird beim Programmstart automatisch einmal instanziert.

  Ressourcen können andere Ressourcen importieren und ihre Operationen aufrufen. Ebenso werden auch *globale Objekte* importiert; sie sind aber in einem gesamten Adressbereich (VM) für alle importierenden Ressourcen-Instanzen die gleichen.

Auf der Vorlesungsseite finden Sie ein Dokument mit einer Zusammenfassung der wichtigsten vordefinierten Operatoren und Funktionen von MPD . Das Dokument bezieht sich auf den Vorgänger SR, gilt allerdings bis auf die Zuweisung (`=` statt `:=`) und die Vergleichsoperation (`==` statt `=`) auch für MPD .
Die erste Übung enthält Aufgaben, in denen Sie das Prozedurkonzept von MPD kennen lernen. Eine Prozedur wird mit `op` deklariert und mit `proc` definiert. Alternativ kann man Prozeduren auch mit `procedure` definieren, falls keine externe Deklaration nötig ist. Prozeduren können mit Hilfe von `co ...oc` konkurrierend gestartet werden, wobei dieses Konstrukt auch automatisch für eine Synchronisation am Ende sorgt.

Aufgaben

1.1 Binomialkoeffizienten — 7 Punkte

Bekanntlich lassen sich die Binomialkoeffizienten neben der geschlossenen Formel auch über die Rekursionsformel $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ bestimmen. Das Berechnungsschema ist bekannt als das *Pascalsche Dreieck*.

(a) Schreiben Sie ein MPD -Programm, das Binomialkoeffizienten mit Hilfe einer rekursiven Prozedur berechnet. Benutzen sie `co ... oc` zur Parallelisierung Ihres Programms. / 3

(b) Schreiben Sie eine zweite Version Ihres Programms, die ein *global definiertes*, zweidimensionales Array benutzt. Das Array soll am Ende das Pascalsche Dreieck bis zum gewünschten Wert enthalten und schrittweise berechnet werden. / 3

Wie viele Prozesse benutzt Ihr Programm, wie viele Arbeitsschritte führen die Prozesse aus? Welche Probleme können bei Ihrer Parallelisierung auftreten? / 1

1.2 Speedup-Schätzungen — 5 Punkte

(a) Ein paralleler Algorithmus erfordere bei Problemgröße n $14n$ Schritte sowie eine sequenzielle Vorverarbeitung der Eingabedaten, welche $(\log_2 n)^3$ Schritte benötigt. Wie viele Prozessoren müssen jeweils (mindestens) eingesetzt werden, um für ein Problem der Größe $n = 1024$ einen Speedup von 10 zu erzielen? / 2

(b) Zeigen Sie, dass sich Amdahls Gesetz in der Form $S_p(n) \leq \frac{1}{f+\frac{1-f}{p}}$ in das von Gustafson & Barsis überführen lässt. / 3

Hinweis: Drücken Sie dazu den bei Gustafson & Barsis zugrunde liegenden sequenziellen Anteil s durch den in Amdahls Gesetz verwendeten Faktor f aus.