Technische Universität München
Institut für Informatik
Dr. rer. nat. Tobias Weinzierl
Matous Sedlacek

März 2012

Kompaktkurs Einführung in die Programmierung

Übungsblatt 2: Ausgabe

Lernziele

- Implementieren von Funktionen.
- Umgang mit Schleifen.
- Realisieren einfacher mathematischer Ausdrücke.

1) Funktionsergänzung (P)

Auf der WWW-Seite der Vorlesung finden Sie ein Programm `blatt2.cpp`, das gemäß der folgenden Aufgaben modifiziert werden soll (in der Funktion `main` werden die betreffenden Funktionen verwendet, so dass Sie schnell sehen können, ob Ihre Funktionen das tun, was sie sollen).

i) Ergänzen Sie die vorgegebene Funktion

```
int fakultaet(int n) {
    int produkt;  // hier ggf. weitere Variablen
    produkt = 1;
    // Hier die Berechnung von n!
    return produkt;
}
```

so, dass sie wirklich die Fakultät $n!$ berechnet.

ii) In dem Programm finden Sie auch die beiden Funktionen `int teilersumme(int n)` – Summe der echten Teiler von n – und `bool vollkommen(int n)` – prüfen, ob n eine vollkommene Zahl ist.

Ergänzen Sie die Funktion

```
void vollkommenListe(int n) {
    std::cout << "Vollkommene Zahlen <= " << n << "\n";
}
```

so, dass Sie genau die vollkommenen Zahlen kleiner oder gleich n ausdrucken.

2) Dreiecksmuster (P)

An der gleichen Stelle finden Sie das Programm `drucken.cpp`.

i) Finden Sie heraus, was das Programm tut (dazu könnten Sie es z.B. ausführen). Probieren Sie auch andere Parameterwerte im Funktionsaufruf `drucke2(3)` (in der Funktion `main`) aus!

ii) Ändern Sie das Programm so, dass es statt der ganzen Texte ein Dreiecksmuster ausgibt, z.B. für `drucke2(4)`:

```
x
xx
xxx
xxxx
```

Die Struktur des Programms (Funktionen `drucke1` und `drucke2`) soll dabei erhalten bleiben, nur die Anweisungen zum Ausdrucken sollen passend geändert (ergänzt, entfernt,...) werden.

3) Einstieg in OpenMP (★)

Wozu Parallelisierung: Eine CPU immer schneller machen, gelingt heute nicht mehr so einfach, stattdessen werden Mehrkern (Dualcore, Quadcore,...) CPUs in die Rechner eingebaut.

- Mögliche Parallelisierungart:
  - Distributed-memory – mehrere einzelne Rechner in einem Cluster. Verwendung des Message Passing Interfaces (MPI).
  - Shared-memory – mehrere CPUs in einem Rechner, die auf einen gemeinsamen Speicher zugreifen können. Verwendung der OpenMP API.
- Erweiterung für existierende Programmiersprachen (C/C++/Fortran), hauptsächlich über Compiler Direktiven, einige wenige Library-Funktionen, Schwerpunkt: Parallelisierung von Loops.
- Inkrementelle Parallelisierung (Parallelisierung eines existierenden seriellen Programms).
- Um ein shared-memory OpenMP Programm zu entwerfen müssen Sie Ihren sequentiellen/seriellen code mit OpenMP pragmas ergänzen und anschließend bei der Compilierung eine spezielle Option verwenden: beim `gcc` ist das `-fopenmp`. Die Option ist Compiler-abhängig, sollte aber in der Dokumentation zu ihrem Compiler aufgelistet sein.
- Tutorials zu OpenMP z.B. unter `http://openmp.org/wp/resources/#Tutorials` oder `https://computing.llnl.gov/tutorials/openMP/` zu finden.
- Siehe auch OpenMP Syntax Quick Reference Card `OpenMP3.1-CCard.pdf` (auf der website zum Kurs).

Ein erstes shared-memory OpenMP Programm

i) Machen Sie sich mit OpenMP und der shared-memory Parallelisierung vertraut (z.B. über obige Links).

ii) Versuchen Sie nun nachstehendes Programm `helloOpenMP.cpp` zu kompilieren. Unter Verwendung des `gcc` Compilers werden Sie zur Übersetzung das flag `-fopenmp` brauchen, das Programm also mittels `g++ -fopenmp -o helloOpenMP ./helloOpenMP.cpp` übersetzen.
helloOpenMP:

```
#include <omp.h>
#include <iostream>
int main() {
  #pragma omp parallel
        std::cout << "Hello, world! This is thread "
                  << omp_get_thread_num() << " of "
                  << omp_get_num_threads() << std::endl;
  return 0;
}
```

iii) In dieser Aufgabe soll die approximative Berechnung von π mittels OpenMP parallelisiert werden. Versuchen Sie den Quellcode des Programms `pi_seriell.cpp` (auf der website) nachzuvollziehen. Die Approximation an π wird hier über numerische Integration durchgeführt:

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

Erweitern Sie das serielle Programm mittels OpenMP Direktiven, so dass die Approximation an π parallel ausgeführt wird. Hinweis: die Direktive `reduce(...)` könnte hilfreich sein. Sie können die Anzahl an zu benutzenden Threads p über `omp_set_num_threads(p);` im Programm hardcodieren.