Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Mohammad Norouzi, Marcus Ritter

Systemnahe und parallele Programmierung (WS 20/21)

Praktikum: C und OpenMP

Die Lösungen müssen bis zum 12. Januar 2021, 13:30 Uhr, in Moodle submittiert werden. Anschließend müssen Sie ihre Lösung einem Tutor vorführen. Das Praktikum wird benotet. Im Folgenden einige allgemeine Bemerkungen, die für alle Aufgaben auf diesem Blatt gelten:

- Es gibt ein vorgegebenes Makefile welches alle Programme kompiliert, die Sie entwickeln sollen.
- Dynamisch allokierter Speicher muss freigegeben werden, bevor das Programm endet.
- Es folgt eine Liste von Funktionen der C-Standardbibliothek, die bei der Lösung hilfreich sein könnten. Bitte informieren Sie sich im Internet über die genaue Funktion und Verwendung dieser Funktionen.
  - `srand()`
  - `rand()`
  - `atoi()`
- Die Programme müssen auf dem Lichtenberg Cluster kompilierbar und ausführbar sein. Alle Zeitmessungen müssen auf dem Lichtenberg Cluster ausgeführt werden. Es kann ein Beispieljobscript von Moodle heruntergeladen werden. Es enthält auch Hinweise, wie Sie es für Ihre Bedürfnisse anpassen können.
- Vorgegebener C/C++ Code in den Dateivorlagen darf nicht geändert werden und muss wie gegeben verwendet werden.
- Die Zeit kann mit Hilfe der Funktion `omp_get_wtime()` bestimmt werden.
- Alle Lösungen, die keinen Programmcode erfordern, bitte zusammen in einer PDF Datei einreichen.
- Laden Sie alle Dateien in einem .tar, .zip Archiv in Moodle hoch.

Aufgabe 1

(4 Punkte) Schreiben Sie ein Programm in der Datei `hello-openmp.c`, das zunächst die maximale Anzahl Threads ausgibt, die in einem *parallel construct* genutzt werden können. Setzen Sie danach diese maximale Anzahl auf einen anderen Wert. Schließlich soll ein Team von Threads erstellt werden, von dem jeder Thread `Hello` gefolgt von seinem Identifikator ausgibt.

Die OpenMP Referenz könnte dabei hilfreich sein:
`https://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf`

Aufgabe 2

(10 Punkte) Gegeben sei das folgende Programm:

```
1  int g1, g2;
2
3  int foo(int p)
4  {
5      return p + g1 + g2;
6  }
7
8  int main()
9  {
10     int i, j;
11
12 #pragma omp parallel for private(g1)
13     for (i=0; i<10; i++)
14     {
15         j += g1 + g2 + i;
16         j += foo( g2 );
17     }
18     return 0;
19 }
```

a) Geben Sie an ob die folgenden Variablen *private* oder *shared* sind im Konstrukt `omp parallel for`:

1. i:
2. j:
3. g1:
4. g2:

b) Geben Sie an ob die folgenden Variablen *private* oder *shared* sind in der Funktion `foo`.

1. p:
2. g1:
3. g2:

Aufgabe 3

(15 Punkte) Das Skalarprodukt zweier Vektoren $\vec{x}$ und $\vec{y}$ der Länge n berechnet sich aus

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i \cdot y_i \tag{1}$$

In dieser Aufgabe entwickeln Sie eine sequenzielle und eine parallele Version zur Berechnung des Skalarprodukts. Nutzen Sie zur Lösung die Dateivorlage `dotproduct.c`.

a) **(5 Punkte)** Die Funktion `test01` soll das Skalarprodukt sequenziell berechnen.

b) **(5 Punkte)** Die Funktion `test02` soll das Skalarprodukt parallel mit OpenMP unter Nutzung des *loop constructs* und der *reduction clause* berechnen.

c) **(5 Punkte)** Messen Sie die sequenzielle Ausführungszeit (`test01`) und die Zeit der parallelen Berechnung (`test02`) mit $1, 2, 4, 8$ und 16 Threads für Vektoren der Länge 10.000.000. Stellen Sie die Ergebnisse grafisch dar.

Aufgabe 4

(13 Punkte) Gegeben ist ein sequenzielles C++ Programm in der Datei `heated-plate-parallel.cpp`, dass die Wärmeleitungsgleichung für ein 2-dimensionales Gebiet löst.

a) **(9 Punkte)** Parallelisieren Sie das vorgegebene Programm mit OpenMP *loop constructs* in der Datei `heated-plate-parallel.cpp`. Es genügt bei verschachtelten Schleifen nur die Äußere zu parallelisieren.

b) **(4 Punkte)** Messen Sie die Laufzeit von Ihrem parallelen Algorithmus für 1, 2, 4, 8, 16 und 32 Threads. Stellen Sie die Messergebnisse grafisch dar.

Aufgabe 5

(7 Punkte) Was gibt `printf()` im folgenden Codeausschnitt aus? Begründen Sie Ihre Antwort.

```
1 int a = 0;
2 #pragma omp parallel private(a)
3 {
4     a++;
5     printf("%d\n", a);
6 }
```

Aufgabe 6

(26 Punkte)

Schreiben sie ein Programm, das das Produkt zweier Matrizen berechnet. **Beispiel:** Gegeben sind die Matrizen A[m][n], B[m][p], und C[p][n]. Berechnen sie deren Multiplikation in der folgenden Form:

A=B*C

Das Ergebnis der Multiplikation ist A.

a) **(5 Punkte)** Schreiben sie ein sequentielles C Programm, dass die Multiplikation der Matrizen ausführt. Initialisieren sie alle 3 Arrays einzeln.

b) **(11 Punkte)** Schreiben sie ein Programm mit Hilfe von OpenMP, dass den selben Zweck erfüllt, aber nun parallel mithilfe des *worksharing-loop* Konstruktes. Nutzen sie dazu die *collapse* und *schedule* Klauseln um die Leistung des Programms zu verbessern. Klassifizieren sie zusätzlich die Variablen.

c) **(2 Punkte)** Bestimmen sie die Laufzeit des seriellen **(1 Punkt)** und des parallelen Programms **(1 Punkt)**. Die Punkte gibt es für den Code zur Messung der Laufzeit.

d) **(8 Punkte)** Führen sie das parallele Programm mit 1, 2, 4, 8, 16 und 32 Threads aus und erstellen sie ein Diagramm, dass die Skalierbarkeit des Programms zeigt (Laufzeit).