Grundlagen der Informatik III

Wintersemester 2010/2011

Michael Koch, Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

5. Praktikum

Ausgabe: 19.01.2011; Abgabe: 13.02.2011, 23:59 Uhr

Abgabe der Programme: Die Abgabe der Programme erfolgt über den SVN-Server https://ics.ra.informatik.tu-darmstadt.de/svn/. Die Beantwortung der Theoriefragen erfolgt in einem eigenen Dokument (PDF).

Hinweis: Die Bearbeitung der Aufgabe 1 ergibt 1 Punkt. Durch Bearbeitung der restlichen Aufgaben können bis zu 3 weitere Punkte erreicht werden.

Wenn Sie keinen geeigneten Rechner besitzen um die Laufzeitmessungen durchzuführen können Sie im Rahmen des Praktikum einen Zugang auf einem geeigneten Rechner erhalten. Für die Einrichtung eines Accounts schreiben Sie Wolfgang Heenes (heenes@ra.informatik.tu-darmstadt.de) eine Email.

Aufgabe 1: Berechnung und Darstellung der Mandelbrotmenge (1 Punkt)

Dieses Praktikum behandelt die nach Benoît B. Mandelbrot benannte Mandelbrotmenge. Die Mandelbrotmenge ist eine fraktale Menge, die auch als Apfelmännchen bekannt ist. „Der von Mandelbrot geprägte Begriff fraktale Menge oder Fraktal bezeichnet alle »nichtglatten« Mengen." [Mand87, S. 7] [1] Mandelbrot beschrieb diese Menge 1980 in seiner Arbeit [Mand80]. [2]

Die Menge ist wie folgt definiert:

$M = \{c \in \mathbb{C} | \lim_{n\to\infty} Z_n \neq \infty\}$ mit $Z_0 = c$ und $Z_{n+1} = Z_n^2 + c$

Die Menge M enthält also diejenigen komplexen Zahlen für welche die rekursive Funktion Z_n mit unendlich vielen Iterationen auf Sie angewendet einen Wert ungleich Unendlich liefert.

Eine Einführung in die komplexen Zahlen finden Sie unter anderem in [BSMM2001, S. 34].[3]

Tipp: Es lässt sich leicht beweisen dass sobald $|Z_n|$ einmal größer 2 wird es nie wieder kleiner als 2 wird und sehr schnell gegen Unendlich geht.

a) Schreiben Sie ein C++ Programm welches die von Mandelbrot beschriebene Menge berechnet. Nutzen Sie hierzu die Datei **mandelbrot_vorgabe.cpp** als Vorlage. Achten Sie darauf dass Ihr Klassendesign eine einfache Erweiterung hinsichtlich der nachfolgenden Aufgaben ermöglicht. Verwenden Sie bei Ihren Berechnungen für alle Zwischenergebnisse double als Datentyp. Das Ergebnis Ihrer Berechnungen soll in der Variable **asciimap** abgelegt werden (1= in der Mandelbrotmenge, 0= nicht in der Mandelbrotmenge). Zur Ausgabe soll die Funktion **printASCIIMandelbrot** verwendet werden. Ihr Programm soll folgende Parameter übergeben bekommen: Anzahl der durchzuführenden Iterationen, Ausgabemodus [0=keine;1=ASCII-Art].
Beispiel (10.000 Iterationen, keine Ausgabe):
```
$ ./mandelbrot 10000 0
```

[1] [Mand87] Mandelbrot, Benoît B.: Die fraktale Geometrie der Natur- Basel; Boston: Birkhäuser, 1987.

[2] [Mand80] Mandelbrot, Benoît B.: FRACTAL ASPECTS OF THE ITERATION OF $z \to \lambda z(1-z)$ FOR COMPLEX λ and z- Mathematics Department Harvard University Cambridge, Massachusetts 02138, 1980.

[3] [BSMM2001] Bronštein, Semendjaev, Musiol, Mühlig: Taschenbuch der Mathematik- Thun; Frankfurt am Main: Verlag Harri Deutsch, 5. Auflage 2001.

Als Referenz der Programmiersprache C++ ist das Buch [Stro2009][4] des Entwicklers Bjarne Stroustrup zu empfehlen.

b) Testen Sie Ihre Implementierung indem Sie die Ausgabe mit Abbildung 1.1 vergleichen.

Abbildung 1.1: Die Mandelbrotmenge als ASCII-Art

Hinweis: Die Darstellung als ASCII-Art ist auf Grund von Rundungsfehlern bei zu geringer Auflösung nicht symmetrisch.

[4] [Stro2009] Stroustrup, Bjarne: The C++ programming language - Boston: Addison-Wesley, 2009.

Aufgabe 2: Grafische Darstellung (1 Punkt)

Erweitern Sie das Programm um die Möglichkeit die Mandelbrotmenge grafisch darzustellen. Achten Sie in Ihrem Klassendesign auf eine strikte Trennung der Berechnung und der Ausgabe. Ihre grafische Darstellung soll mit einer Auflösung von 1024x768 Pixeln arbeiten. Punkte die sich in der Mandelbrotmenge befinden sollen weiß, alle anderen sollen schwarz dargestellt werden. Die grafische Ausgabe soll aktiviert werden wenn Ihr Programm als zweiten Parameter eine 2 übergeben bekommt.
Beispiel (10.000 Iterationen, grafische Ausgabe):

```
$ ./mandelbrot 10000 2
```

Tipp: Verwenden Sie für die grafische Ausgabe die Bibliothek Simple DirectMedia Layer[5], sie ist plattformunabhängig.

Aufgabe 3: Berechnung mit OpenMP und POSIX Threads (1 Punkt)

a) Implementieren Sie die Berechnung der Menge mit OpenMP. Ihr Programm soll nun beim Start einen weiteren Parameter übergeben bekommen welcher die Anzahl der Threads bestimmt (Ist der Parameter 1 soll OpenMP deaktiviert bleiben).
Beispiel (10.000 Iterationen, grafische Ausgabe, OpenMP mit 4 Threads):

```
$ ./mandelbrot 10000 2 4
```

b) Lassen Sie den Compiler Ihren Code optimieren indem Sie den Parameter -O1 übergeben. Bestimmen Sie nun die Laufzeit der Berechnung ohne OpenMP und mit 2^n Threads für $n \in [1,5]$. Wählen Sie für die Laufzeitmessungen eine Auflösung von 1024x768 Pixeln und eine geeignete Anzahl an Iterationen. Nutzen Sie zur Laufzeitmessung eine Maschine mit mindestens 4 Cores. Stellen Sie Ihre Ergebnisse als Tabelle sowie als Graph dar.
Bestimmen Sie außerdem den Speedup für die einzelnen Berechnungen. Der Speedup berechnet sich wie folgt: $S_p = \frac{T_1}{T_p}$ wobei p die Anzahl der Threads und T_p die Laufzeit mit p Threads ist.

Tipp: Deaktivieren Sie für die Laufzeitmessungen die grafische Ausgabe (zweiter Parameter = 0)

c) Sie sollen nun die Menge mit Hilfe von POSIX-Threads berechnen. Als einführendes Beispiel in POSIX-Threads betrachten wir folgendes Programm, welches zwei Arrays a und b der Länge 10^6 mit Zufallszahlen füllt, diese elementweise aufaddiert und in einem dritten Array c ablegt. Um eine vergleichbare Laufzeit zu erzielen wird dieser Vorgang in einer Schleife 10^5 mal wiederholt. Das Programm bekommt einen Parameter übergeben welcher die Anzahl der Threads bestimmt. Die zu addierenden Werte werden gleichmäßig auf die erstellten Threads verteilt.

Um das Programm zu compilieren nutzen Sie folgenden Compiler-Aufruf:

```
$ g++ posix_threads.cpp -pthread -o posix_threads
```

```
1  /*
2   * posix_threads.cpp
3   *
4   *  Created on: 10.01.2011
5   *      Author: mk
6   */
7
8  #include <pthread.h> // POSIX-threads
9  #include <time.h> // time operations (to calculate the runtime)
10 #include <stdio.h> // standard IO operations (to print to console)
11 #include <stdlib.h> // general purpose standard library (to generate random numbers)
12 #define NUM_ELEMENTS 1000000 // number of array-elements to add
13
14 pthread_t* threads; // stores the posix-thread-objects
15 int* threadNums; // stores the thread-numbers
16 time_t beginCalc,endCalc; // used to calculate the runtime of the program
17 unsigned long a[NUM_ELEMENTS],b[NUM_ELEMENTS],c[NUM_ELEMENTS]; // arrays to add (c=a+b)
```

[5] http://www.libsdl.org/

```
18 int numThreads; // number of threads to create
19
20 /*
21  * fills the arrays with random numbers between 0 and 9
22  */
23 void fillArrays ()
24 {
25   for(unsigned i=0;i<NUM_ELEMENTS;i++)
26   {
27     a[i] = rand() % 10;
28     b[i] = rand() % 10;
29   }
30 }
31
32 /*
33  * this function is called after a thread has been created by pthread_create
34  */
35 void* addArrays(void* pThreadNum)
36 {
37     int* threadNum = (int*)pThreadNum;
38     printf("Thread %i started\n",*threadNum);
39     unsigned begin,end; // stores the first and last array index which this thread calculates
40     if(*threadNum == -1) // only one thread, so no POSIX-threads to create
41     {
42       begin = 0;
43       end = NUM_ELEMENTS;
44     }
45     else // calculate the first and last array index this thread has to calculate
46     {
47       begin = (NUM_ELEMENTS/numThreads)**threadNum;
48       end = (NUM_ELEMENTS/numThreads)*(*threadNum+1);
49     }
50     for (unsigned j=0;j<10000;j++) // run 10.000 times to get a runtime long enough to compare
51     for (unsigned i=begin;i<end;i++)
52       c[i] = a[i] + b[i]; // add array elements
53 }
54
55 int main(int argc, char* argv[])
56 {
57   if(argc!=2)
58   {
59     printf("error! first parameter should be: number of threads\n");
60     return -1;
61   }
62   numThreads = atoi(argv[1]); // get number of threads from the first parameter
63    fillArrays (); // fill the arrays with random numbers
64   time(&beginCalc); // store the time at the beginning of the calculation
65   if(numThreads==1) // no POSIX-threads need to be created
66   {
67     int noThreads = -1;
68     addArrays((void*)&noThreads);
69   }
70   else if(numThreads>1)
71   {
72     threads = (pthread_t*)malloc(numThreads*sizeof(pthread_t)); // reserve memory for the threads
73     threadNums = (int*)malloc(numThreads*sizeof(int)); // reserve memory for the thread-numbers
74     for (unsigned i=0;i<numThreads;i++)
75     {
76       threadNums[i] = i;
77       pthread_create(&threads[i],NULL,addArrays,(void *)&threadNums[i]); // create thread
78     }
79      // wait for termination of all threads
80     for(unsigned i=0;i<numThreads;i++)
81       pthread_join(threads[i],NULL);
82     // free memory
83     free(threads);
84     free(threadNums);
85   }
86   else
87   {
```

```
88      printf("error! number of threads must be positive\n");
89      return -1;
90    }
91    time(&endCalc); // store the time at the end of the calculation
92    double timePassed = difftime(endCalc,beginCalc); // calculate the passed time between beginning and end of the calculation
93    printf("%f seconds passed\n",timePassed);
94    return 0;
95  }
```

Implementieren Sie nun die Berechnung der Menge mit POSIX-Threads. Ihr Programm soll nun beim Start einen weiteren Parameter übergeben bekommen welcher angibt ob mit OpenMP oder mit POSIX-Threads gearbeitet werden soll (0 = POSIX-Threads; 1 = OpenMP).
Beispiel (10.000 Iterationen, grafische Ausgabe, 4 POSIX-Threads):

```
$ ./mandelbrot 10000 2 4 0
```

Tipp: Alle notwendigen Funktionen befinden sich in der Datei pthread.h. Eine ausführliche Einführung in POSIX-Threads Programmierung finden Sie unter anderem auf der Website von Blaise Barney, Lawrence Livermore National Laboratory.[6]

d) Lassen Sie den Compiler Ihren Code optimieren indem Sie den Parameter -O1 übergeben. Bestimmen Sie nun die Laufzeit der Berechnung mit 2^n Threads für $n \in [1,5]$. Bestimmen Sie außerdem den Speedup für die einzelnen Berechnungen. Wählen Sie für die Laufzeitmessungen eine Auflösung von 1024x768 Pixeln. Nutzen Sie die selbe Anzahl an Iterationen und die selbe Maschine wie in der vorhergehenden Teilaufgabe. Ergänzen Sie die in der vorhergehenden Teilaufgabe erstellte Tabelle und stellen Sie Ihr Ergebnis als Graph dar.

Aufgabe 4: Berechnung mit SSE (1 Punkt)

a) Implementieren Sie die Berechnung der Menge mit SSE. SSE soll aktiviert werden wenn Ihr Programm als fünften Parameter eine 1 übergeben bekommt.
Beispiel (10.000 Iterationen, grafische Ausgabe, 4 POSIX-Threads, SSE):

```
$ ./mandelbrot 10000 2 4 0 1
```

Tipp: Alle notwendigen Funktionen befinden sich in der Datei emmintrin.h. Weitergehende Informationen hierzu finden Sie auch in der Intel® 64 and IA-32 Architectures Optimization Reference Manual 248966-023 (Stand Januar 2011 in Kapitel 4.3).[7]

b) Bestimmen Sie die Laufzeit sowie den Speedup der Berechnung mit SSE. Ergänzen Sie die in Aufgabe 4 erstellte Tabelle und stellen Sie Ihr Ergebnis als Graph dar.

Lassen Sie den Compiler Ihren Code optimieren indem Sie den Parameter -O1 übergeben. Bestimmen Sie nun die Laufzeit der Berechnung mit SSE ohne Threads sowie mit SSE und 2^n Threads für $n \in [1,5]$. Bestimmen Sie außerdem den Speedup für die einzelnen Berechnungen. Wählen Sie für die Laufzeitmessungen eine Auflösung von 1024x768 Pixeln. Nutzen Sie die selbe Anzahl an Iterationen und die selbe Maschine wie in den vorhergehenden Teilaufgaben. Ergänzen Sie die in den vorhergehenden Teilaufgaben erstellte Tabelle und stellen Sie Ihr Ergebnis als Graph dar.

[6] https://computing.llnl.gov/tutorials/pthreads/

[7] http://www.intel.com/Assets/PDF/manual/248966.pdf?wapkw=%28architecture+optimization+reference+manual%29

Aufgabe 5: Zoom in die Mandelbrotmenge

a) Ergänzen Sie Ihre grafische Darstellung um eine Möglichkeit einen Teilbereich der Mandelbrotmenge zu selektieren welcher dann vergrößert dargestellt werden soll. Dieser Vorgang soll sich beliebig oft ausführen lassen. Außerdem soll es möglich sein wieder zur Ausgangsmenge zurück zu kehren.

b) Passen Sie die Anzahl der Iterationen je nach Zoomtiefe automatisch an. Pro Zoomvorgang soll die Anzahl der Iterationen verdoppelt werden.

c) Testen Sie verschiedene Möglichkeiten die Mandelbrotmenge farblich darzustellen und dokumentieren Sie Ihre Ergebnisse mit Screenshots. Erweitern Sie Ihr Programm so, dass sich die verschiedenen farblichen Darstellungen über einen Parameter aktivieren lassen. Ist der Parameter 0 soll die Mandelbrotmenge wie zuvor in Schwarz-Weiß ausgegeben werden. Jeder neuen farblichen Darstellung soll ein Wert für den Parameter größer Null zugeordnet werden.
Beispiel 1 (10.000 Iterationen, grafische Ausgabe, 4 POSIX-Threads, SSE, schwarz/weiß):
```
$ ./mandelbrot 10000 2 4 0 1 0
```
Beispiel 2 (10.000 Iterationen, grafische Ausgabe, 4 POSIX-Threads, SSE, blau-grün-grau):
```
$ ./mandelbrot 10000 2 4 0 1 1
```