Otto-von-Guericke-Universität Magdeburg
Max-Planck-Institut für Dynamik komplexer technischer Systeme
Computational Methods in Systems and Control Theory

Dr. Jens Saak, Dipl.-Math. Martin Köhler
Webseite: http://www.mpi-magdeburg.mpg.de/mpcsc/lehre/2013_SS_SC/

Sommer Semester 2013

Wissenschaftliches Rechnen 2
Hausaufgabenblatt 2

Ausgabe: 23. April 2013 **Abgabe:** 30. April 2013

Hinweis. Alle Abgaben sind in gedruckter Form und per E-Mail einzureichen. Die Abgabe umfasst den Quelltext **und** die Ausgabe der Programme (mindestens ein Testlauf). Wenn Sie die virtuelle Maschine von der Webseite nutzen können eventuell keine verwertbaren Zeiten gemessen werden bzw. spiegeln diese ein "falsches" Bild wieder.

Aufgabe 1: (10 Punkte)

Wir betrachten das Matrix-Vektor-Produkt $x = Ay$ ($x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ und $A \in \mathbb{R}^{m \times n}$). Die Matrix A ist dicht-besetzt und besitzt keine weitere ausnutzbare Struktur.

a.) Schreiben Sie eine C-Funktion `void mvp(struct my_matrix_st *A, double *y, double *x)`, welche dieses Produkt berechnet. Die Matrix A ist dabei im Fortran-typischen Format gespeichert und mit den notwendigen Informationen in folgender Struktur verpackt (siehe *Wissenschaftliches Rechnen I*):

```
struct my_matrix_st {
        int cols;
        int rows;
        int LD;
        double * values;
        char structure;
};
```

b.) Parallelisieren Sie die Funktion aus Teilaufgabe a.) mit Hilfe von PThreads wie folgt: Die Funktion `mvp` erhält einen zusätzlichen Parameter um die Anzahl der Threads T festzulegen. Die Berechnungen sollen dabei so verteilt werden, dass jeder Thread $\approx \frac{m}{T}$ Einträge des Ergebnisvektors x berechnet. Die Funktion soll folgenden Aufruf genügen:
`void mvp_threaded(int T, struct my_matrix_st *A, double *y, double *x).`

c.) Schreiben Sie eine weitere parallele Implementierung von Teilaufgabe a.), indem Sie für jeden Eintrag im Ergebnisvektor x einen separaten Thread erstellen. Die Funktion soll folgenden Aufruf genügen:
`void mvp_threaded2(int T, struct my_matrix_st *A, double *y, double *x).`

Vergleichen Sie die Ergebnisse, die Laufzeiten und die CPU-Zeiten. Womit lassen sich Unterschiede erklären. Testen Sie unterschiedlich große Matrizen, z.B. $n = 100$, $n = 1000$, $n = 5000$.

Ein Grundgerüst steht unter http://www.mpi-magdeburg.mpg.de/mpcsc/lehre/2013_SS_SC/Data/mvp_skeleton.tar.gz zur Verfügung.

Aufgabe 2: (10 Punkte)

Wir betrachten die Simpson-Regel zur Berechnung eines Integrals:

$$s = \int_a^b f(x)\,\mathrm{d}x.$$

Dabei wird der Integrationsbereich $[a, b]$ durch $n + 1$ äquidistante Punkte

$$x_i = a + ih, \quad i = 0, \ldots, n$$

mit $h = \frac{(b-a)}{n}$ in n Intervalle $[x_i, x_{i+1}]$ mit zugehörigen Mittelpunkten

$$x_{i+\frac{1}{2}} = \frac{x_i + x_{i+1}}{2}$$

unterteilt. Das Integral kann dann durch die Summe

$$s = \frac{h}{6} \sum_{i=0}^{n-1} \left(f(x_i) + 4f(x_{i+\frac{1}{2}}) + f(x_{i+1}) \right)$$

approximiert werden.
Erstellen Sie eine parallele Implementierung, welche mit Hilfe von T Threads das Integral berechnet. Dabei sollen die Teilintervalle gleichmäßig auf die Anzahl der Threads verteilt werden. Die einzelnen Teilergebnisse sollen mit Hilfe von `pthread_exit` an das Hauptprogramm zurückgegeben werden und dort final aufaddiert werden.
Nutzen Sie als Testfunktion

$$f(x) = e^{x^2} - 1.$$

auf dem Intervall $[0, 2]$. Füllen Sie folgende Laufzeittabelle auf:

Teilintervalle	$T = 1$	$T = 2$	$T = 4$
$n = 512$			
$n = 8\ 192$			
$n = 65\ 536$			

Aufgabe 3: (10 Punkte)

Um eine korrekte Interaktion zwischen einzelnen Threads sicher zu stellen bietet die PThreads Bibliothek Mutexe und Signale an. Realisieren Sie folgenden Ablauf mit Hilfe dieser beiden Konstrukte:

- Erstellen Sie 4 Threads.
- Im Hauptprogramm soll eine Zahl eingelesen werden. Diese wird über eine globale Variable an den ersten Thread weitergereicht.
- Der erste Thread verdoppelt die Zahl und reicht sie an den zweiten Thread auf gleiche Art und Weise weiter.
- Die Threads 2 und 3 nehmen jeweils die Zahl von ihrem Vorgänger entgegen, verdoppeln diese und reichen sie an ihren Nachfolger weiter.

- Thread 4 erhält die Zahl von Thread 3 und quadriert diese. Das Ergebnis wird weiter an das Hauptprogramm geschickt.
- Das Hauptprogramm wartet in der Zwischenzeit auf das Ergebnis von Thread 4 und gibt dieses aus.

Jeder Thread soll dabei auf dem Bildschirm kurze Statusmeldungen ausgeben, was er gerade macht. Da die Ausgabe auf die Kommandozeile gepuffert ist, kann es hierbei zu unerwünschten Ausgabefehlern kommen. Um diesem Problem ein wenig vorzubeugen, ist es hilfreich nach jeder Ausgabe den Ausgabebuffer durch einen Aufruf von `fflush(stdout);` zu leeren.

Gesamtpunktzahl: 30