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 3

Ausgabe: 7. Mai 2013 **Abgabe:** 14. 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: (8 Punkte)

Wir betrachten nochmals 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 symmetrisch. Die Matrix A ist dabei im Fortran-typischen Format gespeichert, jedoch sind nur die Werte im unteren, linken, Dreieck besetzt und zu verwenden.

a.) Implementieren Sie das Matrix-Vektor-Produkt $x = Ay$ als C-Funktion
`void mvp_openmp(struct my_matrix_st *A, double *y, double *x).`
Nutzen Sie OpenMP zur Parallelisierung. Beachten Sie eventuell auftretende Race-Conditions und sichern Sie dies mit Hilfe von OpenMP-Kontrukten ab.

b.) Überlegen Sie sich eine Möglichkeit, $x = Ay$ parallel mit Hilfe von OpenMP zu berechnen, ohne das eine Synchronisation innerhalb einer Schleife notwendig ist. Eine solche Lösung kann extra Speicher für Zwischenergebnisse erfordern. Implementieren Sie dazu die Funktion:
`void mvp_openmp2(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. $n = 100$, $n = 1000$, $n = 5000$. Das Grundgerüst[1] kann analog wiederverwendet werden. Die Annahme der Symmetrie soll sichergestellt werden, in dem nur auf das untere, linke, Dreieck der Matrix A zugegriffen wird.

Aufgabe 2: (8 Punkte)

Wenn große Datenmengen effizient zusammengefasst werden müssen, ist die sogenannte Tree-Reduction ein Mittel dies zu parallelisieren. Dazu betrachten wir die Summe

$$S := \sum_{i=0}^{n} x_i$$

über einen Vektor $x \in \mathbb{R}^n$. Implementieren Sie eine Funktion
`double treesum (int n, double *x),`
welche einen gegebenen Vektor x aufsummiert. Die Implementierung soll dabei folgenden Kriterien genügen:

[1] http://www.mpi-magdeburg.mpg.de/mpcsc/lehre/2013_SS_SC/Data/mvp_skeleton.tar.gz

- Die Funktion soll mit Hilfe von OpenMP parallelisiert sein.
- An die Länge n des Vektors werden keine besonderen Anforderungen gestellt.
- Der Vektor x darf überschrieben werden und muss am Ende der Funktion **nicht** wieder rekonstruiert werden.
- In jedem Level sollen immer 2 Elemente aufaddiert werden. Das heißt je Level halbiert sich die Anzahl der notwendigen Operationen und der noch zu verarbeitenden Elemente. Es ist Ihnen dabei freigestellt, ob Sie benachbarte Elemente aufaddieren oder ein anderes Zugriffsmuster wählen.

Aufgabe 3: **(5 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

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

approximiert werden.
Erstellen Sie eine parallele Implementierung, welche mit Hilfe von OpenMP. Nutzen Sie als Testfunktion

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

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

Teilintervalle	$T = 1$	$T = 2$	$T = 4$
$n = 512$			
$n = 8\ 192$			
$n = 65\ 536$			
$n = 1\ 000\ 000$			

Vergleichen Sie diese Ergebnisse mit denen von Hausaufgabenblatt 2, Aufgabe 2. Ist der Mehraufwand der PThreads Implementierung zu rechtfertigen?

Aufgabe 4: **(9 Punkte)**

Implementieren Sie die “Block Outer Product” LU Zerlegung ohne Pivotisierung in C. Die Input-Matrix A soll dabei mit ihren Faktoren L und U wie aus der Vorlesung bekannt überschrieben werden. Das innere Rang-k-Update soll dabei mit Hilfe von OpenMP parallelisiert werden. Die Funktion soll folgendem Aufruf genügen:

```
void LU(struct my_matrix_st *A, int k)
```

wobei k die Größe des inneren Rank-k-Updates und somit die Blockgröße ist.
Wenden Sie die Funktion auf Matrizen der Größen 1000, 2000, 3000, 4000 und 5000 an und bestimmen Sie den Speed-Up und die Effizienz für 1, 2 und 4 Threads für verschiedene Werte des Blockgrößen-Parameters r (z.B.: 2, 8, 16, 32, 64).

Gesamtpunktzahl: 30