Übung 6 !
Anwendung höherer Programmiersprachen
im Maschinenbau

11.07.2019

Aufgabe 1

Ziel: Parallelisierungskonzept OpenMP anwenden

Im Archiv „Vorlesung_openmp.tgz“ befinden sich Beispiele aus den Vorlesungsfolien. Machen Sie sich durch die Beispiele die Funktionsweise der verschiedenen OpenMP Direktiven klar.

Bearbeitungsreihenfolge:

- `omp_parallel.f90`
- `omp_atomic.f90`
- `omp_critical.f90`
- `omp_private.f90`
- `omp_firstprivate.f90`
- `omp_reduction.f90`
- `omp_do.f90`

Aufgabenstellung: (nicht zu viel Zeit verwenden….)

- Übersetzen Sie jeweils das Programm (Compileroption `-fopenmp` benötigt)
- Lassen Sie das Programm laufen
- Verändern Sie die Anzahl der Threads: hierzu müssen Sie in der Konsole die Umgebungsvariable `OMP_NUM_THREADS` setzen
  Beispiel: es sollen zwei Threads verwendet werden

  ```
  export OMP_NUM_THREADS=2
  ```

  (Hinweis: Information zur Anzahl der verfügbaren Kerne können Sie z.B. in der Datei /proc/cpuinfo erhalten: `more /proc/cpuinfo`)

  Zeitmessung:

  - Die Zeitmessung können Sie durch Aufruf der zum OpenMP Modul gehörenden Funktion `omp_get_wtime()` durchführen: diese gibt die aktuelle „Wallclockzeit“ zurück. Durch einen Aufruf dieser Funktion direkt vor und nach dem durch OpenMP parallelisierten Block kann die Zeitdifferenz (=Laufzeit) berechnet werden.

    ```
    t1=omp_get_wtime()
         Block
    t1=omp_get_wtime()-t1
    ```

Übung 6 !
Anwendung höherer Programmiersprachen
im Maschinenbau

11.07.2019

Aufgabe 2: Anwendung Matrix/Matrix , Matrix Vektor und Vektor /Vektor Operationen

Ziel: Parallelisieren von Aufgaben, die genügend Rechenaufwand aufweisen

Verwenden Sie Variablen mit `allocatable` Attribut zum Anlegen der Matrizen und Vektoren.

Verwenden Sie „große“ Matrizen Vektoren (>500 Elemente/Zeilen/Spalten)

1. Schreiben Sie eine serielle Matrix Vektormultiplikation, um eine Vergleichsmessung der Rechenzeit zu haben.
   - Verwendung von matmul()
   - zwei do-Schleifen; beide Reihenfolgen (i,j) und (j,i) implementieren.
   - Spezialfall symmetrische Matrix: was können Sie dann ausnutzen!

2. Kopieren Sie den obigen Codeblock und parallelisieren Sie die Matrix/Vektor Multiplikation indem Sie
   - die äußere do-Schleife parallelisieren
   - die innere do-Schleife parallelisieren
   - beide Schleifen parallelisieren

Welche Variante ist am schnellsten?
Wie skaliert die Rechenzeit mit der Anzahl der Threads?

(Hinweis: Hier ist es erforderlich mehrere Rechnungen durchzuführen um Schwankungen in der Rechenzeit zu mitteln)

Um für die Zeitmessung genügend Rechenaufwand zu erzeugen, verändern Sie die Größe der Matrizen bzw. des Vektors. Testen Sie z.B. auch Matrizen mit 1023,1024,1025 Zeilen und Spalten.

Aufgabe 3: Matrix/Matrix Multiplikation --- Reihenfolge der do-Schleifen

Für diese Aufgabe befindet sich ein Source-Code auf ILIAS: `main_do_parallel.f90`

Es handelt sich um die Matrix*Matrix Multiplikation. Betrachten Sie zunächst den seriellen Code zur Berechnung der Multiplikation. Es sind hierzu drei do-Schleifen notwendig.

```
do i=1,nmax
    do j=1,nmax
        do k=1,nmax
            mat3(i,j) = mat3(i,j)+ mat1(i,k)*mat2(k,j)
        end do
    end do
end do
```

Das Ergebnis der obigen Rechnung ist unabhängig von der Reihenfolge der Schleifen.

Verändert sich die Rechenzeit, wenn Sie die Reihenfolge der Schleifen vertauschen?
z.B. Reihenfolge (`i,j,k`) oder (`j,k,i`) oder ….

Messen Sie die Rechenzeit der seriellen Variante(n).

Erklären Sie die Beobachtung.

Übersetzen Sie das Programm mit verschiedenen Compileroptimierungen:

Optimierungsoptionen: `-O2` oder `-O3` oder `Ofast`
Für OpenMP Teil immer Option `-fopenmp` hinzufügen

OpenMP Teil: Entfernen Sie zunächst die `stop` Anweisung.

Parallelisieren Sie die Matrix Matrix Multiplikation analog zum Vorgehen in Aufgabe 2: Welche Parallelisierung ist am effizientesten? Welche Reihenfolge der do-Schleifen führt zur schnellsten Berechnung?

Verwenden Sie wieder verschiedene Optimierungsoptionen.

(Falls Sie den Intel Compiler `ifort` haben: vergleichen Sie die Rechenzeit von `gfortran` mit `ifort`)
(Intel-Compiler: Optimierung `-fast` ausprobieren; was beobachten Sie?)

(Warum werden die Matrizen in eine Datei geschrieben?)

Übung 6 !
Anwendung höherer Programmiersprachen
im Maschinenbau

11.07.2019

Aufgabe 3: CG-Löser

Schreiben Sie einen CG Löser: Siehe Vorlesungsfolien.

Parallelisieren Sie den CG Löser:

- parallelisieren Sie die verschiedenen Operationen des Lösers (Skalarprodukt, Matrix Vektorprodukt, siehe vorherige Aufgaben)
- Testen Sie den Löser:
  - welche Anforderung ist an die Matrix gestellt
  - geeignetes Konvergenzkriterium

Aufgabe 4: Andere Anwendung – Mandelbrotmenge

Mandelbrotmenge: all komplexen Zahlen c für welche die Iterationsvorschrift

$$z_{n+1} := z_n^2 + c$$

beschränkt bleibt (<2), wobei $z_0 = 0$ **gilt (n: Iterationsschritt).**

1. Laden Sie sich das Beispielprogramm zur fraktalen Mandelbrotmenge herunter und kompilieren es: (Datei `mb.f90`)
2. Das Programm schreib die Funktionswerte (als Integer) für alle Punkte eines regelmäßigen Gitters in der komplexen Ebene in die Datei `mb.dat`
   Ausgabeformat einer Zeile : x,y, n_iter
   Nach jeder Zeile: (d.h. wenn x wieder bei 0 anfängt, wird eine Leerzeile eingefügt werden, damit gnuplot dies als Gitterdaten versteht)
3. Gnuplot: folgende Befehle zur Darstellung der Daten verwenden
   In Gnuplot Kommandozeile:
   ```
   set pm3d map
   set size ratio 1
   spl 'Dateiname_Ergebnis'
   ```
4. Parallelisierung des Programms:
   1. Welche Teile können parallelisiert werden? Führen Sie die Parallelisierung durch.
   2. Überprüfen Sie die Ergebnisse, schauen Sie sich die Ausgabe zum Testen für kleine Systeme und im Vergleich mit der nicht-parallelen Version an!!