PS1

Zürcher Hochschule für Angewandte Wissenschaften

Arbeitsblatt: PSP

Name: Austin Albrecht **Kurznamen:** albreaus

Programmierung mit FORTRAN

1. Allgemeine Fragen

Kreuzen Sie die wahren Aussagen an

☒ Assembler war die erste Anwendung von CASE.
☒ Aktuelle Programmiersprachen unterstützen meist mehrere Paradigmen.
☒ Während einer gewissen Zeit, waren mehr als die Hälfte der Programme in FORTRAN geschrieben.
☒ FORTRAN eignet sich fürs HPC wegen seiner einfachen/linearen Datenstrukturen.
☐ Python eignet sich für rechenintensiven Aufgaben wie z.B. das Trainieren von ANN und ist deshalb in der KI so beliebt.

Installation und Austesten der Umgebung

Installieren Sie sich zuerst einen FORTRAN Compiler, z.B. gfortran
gnu-Fortran: http://gcc.gnu.org/wiki/GFortranBinaries
Oder einfacher können Sie auch entsprechend der *Anleitung auf der INF1* Web Seite das ZIP File entpacken und die Pfade von Hand setzen.

Aufgabe

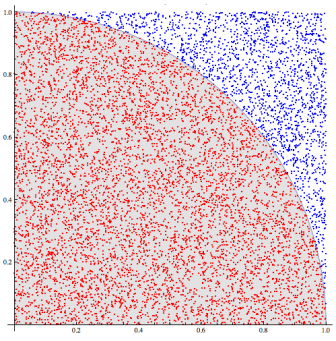
Übersetzen Sie das Hello.f95 Programm und führen Sie dieses aus.

Hinweise:

- http://gcc.gnu.org/wiki/GFortranGettingStarted
- Aufruf: `gfortran hello.f95 -ohello.exe`

2. Berechnung von π mittels Montecarlo Verfahren

Die Zahl π lässt sich mittels einer Monte Carlo Simulation bestimmt. Ein einfaches, aber nicht sehr genaues Verfahren funktioniert folgendermassen. Es werden beliebige Punkte innerhalb des Einheitsquadrates zufällig bestimmt. Ist der Abstand zum Ursprung kleiner als 1, dann zählt man ihn zur roten Menge. Die Anzahl der roten Punkte dividiert durch die Gesamtzahl der Versuche ergibt eine Näherung für π/4.



Aufgabe:

Sie haben ein Python Programm vorgegeben. Schreiben Sie ein FORTRAN Programm, das π mittels dem obigen Verfahren bestimmt, indem Sie die Funktion `calcpi` implementieren

Messen Sie die Laufzeit für 100'000'000 Schleifen Durchläufe.

a) Von welcher Ordnung ist der Algorithmus O(n)

b) Laufzeit des Python Programms für 100'000'000 Durchläufe. 35194 ms

c) Laufzeit des FORTRAN Programms für 100'000'000 Durchläufe 1380 ms

d) Welchen Faktor ist das FORTRAN Programm schneller als Python 24.5

Hinweise:

- Verwenden Sie das vorgegebene Gerüst des Pi Programms
- Die Standardfunktion rand(0) liefert eine Folge von Zufallszahl zwischen [0..1[

Abgabe:

Praktikum: PS1.1

Filename: pi.f95

3. Erhöhen Sie die Performance mittels Open MP Bibliothek

Mittels der OMP Bibliothek lässt sich die Performance verbessern. Parallelisieren Sie den Algorithmus in der Funktion `calcpi_omp1`

Hinweise:

- Aufruf: `gfortran -fopenmp helloomp.f95 -o hello.exe`

a) Welche Beschleunigung erwarten Sie für die π Berechnung bei "echt" (ohne Hyperthreading) 8 Kernen für die Berechnung (Amdahl's Law) 8

b) Welche Zeit messen Sie tatsächlich für 100'000'000 Schleifen Durchläufe?
1373 ms

c) Was stellen Sie fest und haben Sie eine Erklärung dafür?

Das program ist nicht schneller. Threads müssen auf einander warten was zu "Load Imbalance" führt.

Abgabe:

Praktikum: PS1.2

Filename: pi.f95

4. Alternativer Zufallszahlengenerator

Um das Programm weiter zu beschleunigen, kann der Zufallszahlengenerator im Anhang verwendet werden, wobei jedoch der seed (für die Übergabe in einer Variablen gespeichert!) in den einzelnen Threads unterschiedlich initialisiert sein muss (z.B. mit der ThreadID; siehe HelloOMP.f95). Parallelisieren Sie den Algorithmus mit dem neuen Zufallszahlengenerator in der Funktion `calcpi_omp2`

Hinweis:

- FORTRAN verwendet einen sog. *one pass* Compiler. Falls die Funktion nach dem Hauptprogramm steht, muss in der Deklaration des Hauptprogramms noch `real*8 :: ran0` stehen (Vorwärtsdeklaration)
- Mittels der (-Ofast) Compiler Option können Sie die Compiler Optimierungen aktivieren
- Um mit C Programm zu linken, die -fno-underscoring Compiler Option verwenden

Abgabe:

Praktikum: PS1.3

Filename: pi.f95

5. Weitere Optimierungen und Wettbewerb

Die Zufallszahl Berechnung dominiert klar den Rechenzeit Bedarf. Eine mögliche Verbessung wäre der Einsatz von Intel Spezial Instruktionen (bringt leider nicht so viel).

http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

https://software.intel.com/content/dam/develop/external/us/en/documents/drng-software-implementation-guide-2-1-185467.pdf

Eine weitere mögliche Verbessung wäre der Einsatz der GPU/CUDA; es werden Verbesserungen bis zu einer Grössenordnung erwartet:

https://www.math.univ-paris13.fr/~cuvelier/docs/Informatique/CUDA/docs/5.0/CURAND_Library.pdf

Das Beispiel der PI Berechnung mittels CUDA finden Sie hier

https://github.com/phrb/intro-cuda/tree/master/src/cuda-samples/7_CUDALibraries/MC_EstimatePiP

Zu schlagen gilt es übrigens 78 ms für 100'000'000 Iterationen (auf dem i9-9980HK Laptop Ihres Dozenten). Bringen Sie die Laufzeit unter diesen Wert?

Tragen Sie Ihre Zeit unter den obigen Wert in das PDF Dokument ein und geben Sie es ab.

Das Siegerteam bekommt eine Führung im Supercomputing Rechenzentrum in Manno und darf ev. sein Programm auf dem Supercomputer ausführen lassen. https://www.cscs.ch/

Welche Zeit messen Sie für 100'000'000 Durchläufe bei maximaler Parallelisierung: 397 ms und somit 104 mal schneller als Python

Abgabe

Praktikum: PS1

Filename: PS1.pdf

```
function  ran0(seed)
integer*4  seed,ia,im,iq,ir,mask,k
real*8  ran0,am
parameter  (ia=16807,im=2147483647,am=1./im, iq=127773,ir=2836,mask=123459876)
seed=ieor(seed,mask)
k=seed/iq
seed=ia*(seed-k*iq)-ir*k
if  (seed.lt.0)  seed=seed+im
ran0=am*seed
seed=ieor(seed,mask)
return
end
```

Notizpapier ;-)