

## Arbeitsblatt: PSP

Name:

Kurznamen:

## Programmierung mit FORTRAN, OpenMP und CUDA

### 1. Allgemeine Fragen

Kreuzen Sie die wahren Aussagen an

CASE = Computer-Aided Software

- ☐ Assembler war die erste Anwendung von CASE. Engineering
- ☐ 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

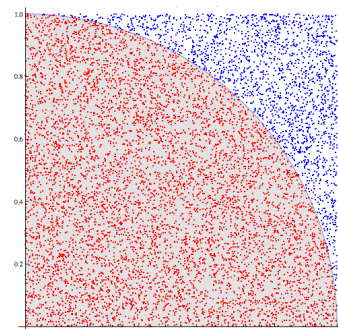
Übersetzen Sie das Hello.f95 Programm und führen Sie dieses aus.

#### Hinweise:

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

### 2. Berechnung von $\pi$ mittels Montecarlo Verfahren

Die Zahl  $\pi$  lässt sich mittels einer Monte Carlo Simulation bestimmen. Ein einfaches, aber nicht sehr genaues Verfahren funktioniert folgendermassen. Es werden beliebige Punkte innerhalb des Einheitsquadrates zufällig gewählt. 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  $\pi/4$ .



### Aufgabe:

Sie haben ein Python Programm vorgegeben. Schreiben Sie ein FORTRAN Programm, das  $\pi$  mittels dem obigen Verfahren bestimmt, indem Sie die Funktion `calcp_i` implementieren

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

- a) Von welcher Ordnung ist der Algorithmus
- b) Laufzeit des Python Programms für 100'000'000 Durchläufe.  ms
- c) Laufzeit des FORTRAN Programms für 100'000'000 Durchläufe  ms
- d) Welchen Faktor ist das FORTRAN Programm schneller als Python

### 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 `calcp_i_omp1`

### Hinweise:

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


a) Welche Beschleunigung erwarten Sie für die  $\pi$  Berechnung bei "echt" (ohne

Hyperthreading) 8 Kernen für die Berechnung (Hinweis: Amdahl's Law)

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

ms

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

 Zum Berechnen hier 8 Threads, jedoch Anzahl Threads aus Code = 2

### Abgabe:

Praktikum: PS1.2

Filename: pi.f95

## 4. Alternativer Zufallszahlengenerator

Um das Programm weiter zu beschleunigen, kann ein alternativer Zufallszahlengenerator 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`<sup>1</sup> 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 Verbesserung wäre der Einsatz von Intel Spezial Instruktionen<sup>2</sup>. Dies bringt leider für diese Anwendung nicht sehr viel, wie Tests gezeigt haben.

Zu schlagen gilt es übrigens **78 ms für 100'000'000 Iterationen** auf dem i9-9980HK Dell 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.

Welche Zeit messen Sie für 100'000'000 Durchläufe bei maximaler Parallelisierung mittels OMP:  ms und somit  mal schneller als Python. Die schnellste Zeit wird übrigens automatisch auf der PSPP Seite angezeigt.

### Abgabe

Praktikum: PS1

Filename: PS1.pdf (dieses Arbeitsblatt)

<sup>1</sup> <http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>


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

## Alternativer Zufallszahlengenerator


```
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
```

## 6. Parallele Verarbeitung mit CUDA

Eine weitere mögliche Laufzeitverbesserung bekommt man durch Einsatz von spezifischer Hardware. CUDA (früher auch Compute Unified Device Architecture genannt) ist eine von Nvidia entwickelte Programmierschnittstelle (API), mit der Programmteile durch den Grafikprozessor (GPU) abgearbeitet werden können. In Form der GPU wird zusätzliche Rechenkapazität bereitgestellt, wobei die GPU im Allgemeinen bei hochgradig parallelisierbaren Programmläufen (hohe Datenparallelität) signifikant schneller arbeitet als die CPU. CUDA wird vor allem bei wissenschaftlichen und technischen Berechnungen eingesetzt. © Wikipedia

Die ZHAW verfügt über einen Tesla T4 Cluster. Der Linux Server hat die IP  160.85.253.187. Er ist direkt über SSH und SFTP zugreifbar. Sie können sich mit Ihrem ZHAW Benutzernamen und Passwort anmelden. Um die CUDA Befehle via Kommandozeile aufzurufen, muss noch folgender Befehl ausgeführt werden.

```
export PATH=/usr/local/cuda-12.2/bin${PATH:+:${PATH}}
```

Dieser wird vorzugsweise Ihrem .bashrc File hinzugefügt, welche automatisch beim Login ausgeführt wird. Für die Übersetzung ist ein makefile vorbereitet. Falls Sie auf einer andern Infrastruktur übersetzen wollen, braucht es ev. andere Compiler  Optionen<sup>3</sup>. Das Programm Pi.cu<sup>4</sup> hat schon eine ganz gute Laufzeiten, welche jedoch noch verbessert werden kann und es finden sich weitere Implementierungen<sup>5</sup>.

Aufgabe: Profilieren Sie den Code. Welche Operation ist die teuerste? Man kann diese Operation einfach von der Zeitmessung ausschliessen (Begründung) und bekommt dann eine Laufzeit im einstelligen Millisekundenbereich. Was wäre der nächste Schritt die Laufzeit weiter zu verbessern?

### Abgabe:

Praktikum: PS1.4

Filename: Pi.cu

<sup>3</sup> <https://arnon.dk/matching-sm-architectures-arch-and-gencode-for-various-nvidia-cards/>

<sup>4</sup> [https://radar.zhaw.ch/~rege/psp\\_hs23/Cuda/Pi.cu](https://radar.zhaw.ch/~rege/psp_hs23/Cuda/Pi.cu)

<sup>5</sup> [https://github.com/phrb/intro-cuda/tree/master/src/cuda-samples/7\\_CUDAlibraries/MC\\_EstimatePiP](https://github.com/phrb/intro-cuda/tree/master/src/cuda-samples/7_CUDAlibraries/MC_EstimatePiP)

## Notizpapier ;-)

The image shows a blank sheet of notepad paper. It has a light green background with a white central writing area. The paper is framed by a decorative border of small grey dots. On the left side, there is a vertical margin with horizontal lines. The top and bottom edges of the writing area are bordered by a solid light green strip.