Dr. Andreas Borchert
Prof. Dr. Stefan Funken
Prof. Dr. Karsten Urban
Markus Bantle, Kristina Steih

Institut für Numerische Mathematik
Universität Ulm
Sommersemester 2013

High Performance Computing – Blatt 5

(Präsenzübung 27. Mai 2013)

Aufgaben für daheim / Diskussion

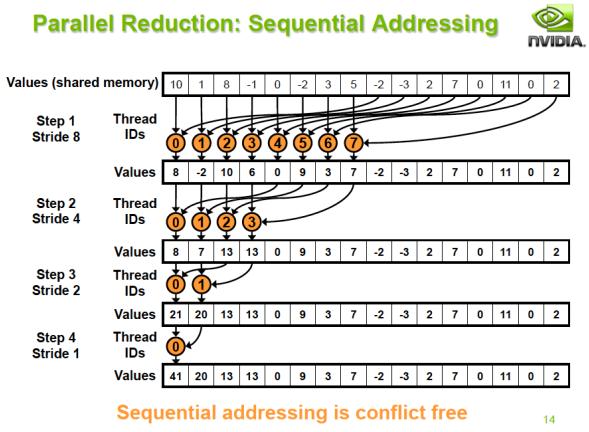
(1) Vervollständigen Sie das Game of Life (Blatt 4 Aufgabe 2).

(2) Auf der Homepage finden Sie die folgenden zwei Code-Beispiele. Beide Beispiele sind vollständig programmiert und lauffähig. Lesen Sie beide Codes sorgfältig durch und beantworten Sie die folgenden Fragen.

`add_one.cu`:
Gegeben ist ein integer Array der Länge `n`. Zu jedem Element des Arrays soll 1 addiert werden.
Die Funktion, die man für die CPU schreiben würde, heisst `inc_CPU` (Zeile 16). Die Funktion, die auf der GPU läuft ist `inc_GPU`. Vergleichen Sie die beiden Funktionen. Was passiert in Zeile 7? Was sind `blockIdx`, `blockDim` und `threadIdx`?
Betrachten Sie die `main` Funktion. Was passiert in den Zeilen 37 und 40? In den Zeilen 45-49 wird der Kernel (d.h. die Funktion `inc_GPU`) konfiguriert und gestartet. Erklären Sie genau was passiert. Was macht Zeile 52?

`norm.cu`:
Gegeben sind `Nv` Vektoren der Länge `Ne`, von denen jeweils die Norm berechnet werden soll. Die Vektoren sind spaltenweise in einer Matrix abgelegt. Um die Norm aller Vektoren auf der CPU zu berechnen kann die Funktion `normCPU` verwendet werden (Zeile 17). Für die Berechnung auf der GPU wird der Kernel `normGPU`, der in der Datei `kernel.cu` implementiert ist, verwendet.

Lesen und verstehen Sie zunächst die Funktion `normCPU`. Betrachten Sie nun die Funktion `normGPU`. Versuchen Sie genau zu verstehen, was in der Funktion passiert. Stellen Sie dazu an Hand von "Bildchen" dar, welcher Thread welche Aufgabe übernimmt. Z.B. können die Zeilen 20-24 in `kernel.cu` durch die Grafik rechts beschrieben werden (Quelle: NVidia). Was bedeutet `__shared__`? Lesen Sie das Hauptprogramm (`main`) und erklären Sie was passiert.



- Die Programme sollen nun auf Olympia übersetzt und gestartet werden. Gehen Sie dabei wie folgt vor (Wir gehen davon aus, das Sie an einem iMac im Mac-Pool sitzen):
  1. Loggen Sie sich im Mac-Pool ein und erstellen Sie im Heimatverzeichnis eine Datei namens ".bashrc" (Terminal öffnen, `cd` tippen und dann `vim .bashrc` (oder einen anderen Editor anstatt vim nehmen)) Fügen Sie dann die beiden Zeilen

     ```
     export PATH=/usr/local/cuda/bin:/usr/local/cuda/lib64:$PATH
     export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
     ```

     ein. Falls in der Datei .bashrc schon etwas steht, die beiden Zeilen einfach hinten anhängen.
  2. Loggen Sie sich per SSH auf Olympia ein. (ssh username@olympia.mathematik.uni-ulm.de). username ist hierbei der login für den Mac-Pool. Wenn Sie `echo $PATH` tippen, sollte in dem Text der erscheint irgendwo `/usr/local/cuda/bin` auftauchen.
  3. Wechseln Sie mit `cd` in das Verzeichnis, in dem die Programe liegen (ist das gleiche Verzeichnis wie das, in dem die Dateien im Mac-Pool abgelegt wurden).
  4. Übersetzen Sie die Programme mit nvcc, Z.B.
     `nvcc -o add_one -arch compute_20 add_one.cu`, und führen Sie die Programme auch. Im Programm `norm_with_timing.cu` werden die Rechenzeiten für GPU und CPU gemessen. Die Ergebnisse sind interessant :-).
- (Freiwilige) Zusatzaufgabe: Schreiben Sie ein Programm, in dem zwei Vektoren auf der Grafik-Karte addiert werden. Die Vektoren sollen auf dem host initialisiert und dann an die Grafik-Karte geschickt werden. Das Ergebnis soll von der Grafik-Karte zurück geschickt und vom host ausgegeben werden.

Hinweis: Hilfreiche Ressourcen zu CUDA gibt es auf der Website
https://developer.nvidia.com/get-started-cuda-cc
Hier sind besonders die Folien zu "Introduction to GPU Computing" empfehlenswert. Auch diese Präsentation bietet eine gute CUDA Einführung
http://www.nvidia.com/content/GTC-2010/pdfs/2131_GTC2010.pdf

Aufgabe für die Übung am 27. Mai

In dieser Übung programmieren wir die Potenzmethode zur Berechnung des maximalen Eigenwerts einer Matrix $A \in \mathbb{R}^{N \times N}$. Die Potenzmethode ist in folgendem Algorithmus beschieben. Hierbei konvergieren die Werte von $\rho^{(k)}$ gegen den maximalen Eigenwert von A.

Algorithm 1 Potenzmethode zur Berechnung des maximalen Eigenwartes von $A \in \mathbb{R}^{N \times N}$

Input: Matrix $A \in \mathbb{R}^{N \times N}$. Startvektor $x^{(0)} \in \mathbb{R}^N$ mit $\|x^{(0)}\| = 1$.

1: **for** $k = 0, 1, \ldots$ **do**
2: $\quad a^{(k)} = Ax^{(k)}$
3: $\quad \rho^{(k)} = x^{(k)\,T} a^{(k)}$
4: $\quad x^{(k+1)} = a^{(k)} / \|a^{(k)}\|$
5: **end for**

Um diesen Algorithmus zu realisieren benötigen wir also Funktionen für

- die Matrix-Vektor Multiplikation,
- das Skalarprodukt,
- die Norm und
- die Skalierung eines Vektors.

Einfache Funktionen für die CPU sind in der Datei `simple_blas.h` implementiert.

(1) Vervollständigen Sie die Funktionen für die GPU in der Datei `kernels.h` und testen Sie jede Funktion einzeln. Hierfür steht ihnen das Programm `test.cu` zur Verfügung. Folgen Sie den Anweisungen im Code.

(2) In der Datei `potMethod.cu` finden Sie die Implementierung für die Potenzmethode auf der CPU und eine (noch nicht vollständige) Implementierung der Potenzmethode für die GPU. Vervollständigen Sie die Funktion `potMethodGPU` und folgen Sie dabei den Anweisungen im Code.

(3) Testen Sie das Programm für verschiedene Matrix Größen und Iterationsschritte.

(4) Zusatzaufgabe: Ersetzen Sie Ihre Funktion für die MAtrix-Vektor Multiplikation durch die cublas Funktion `cublasSgemv`. Achten Sie auf die Laufzeiten!