CUDA und Python

Christian Wilms

Integriertes Seminar
Projekt Bildverarbeitung

Universität Hamburg
WiSe 2013/14

12. Dezember 2013

Gliederung

Übersicht

Wie mache ich mein Programm schneller?

Bisherige Möglichkeiten

- Interpreter wechseln
- tlw. Programmiersprache wechseln

Aber was ist mit den ganz dicken Brettern?

$\Rightarrow$ Wir brauchen schweres Gerät!

Weitere Möglichkeit: Wechsel der Architektur

- Multi-Core CPU nutzen
- GPU
- Multi-Core CPU + GPU

Welche Architekturen gibt es?

Abbildung 1: SISD

Welche Architekturen gibt es?

Abbildung 2: MISD

Welche Architekturen gibt es?

Abbildung 3: MIMD

Welche Architekturen gibt es?

Abbildung 4: SIMD

Übersicht

Warum ist die GPU so schnell?

CPU

- wenige Kerne
- schnelle Kerne
- muss alles können
- minimale Latenz je Thread
- großer Cache
- ausgefallene Logik

GPU

- viele Kerne
- langsamere Kerne
- Datenstrom durch Stufen
- spezielle Hardwarefunktionen
- kaum Cache
- maximaler Durchsatz

$\Rightarrow$ Eine handelsübliche Grafikkarte hat deutlich mehr Kerne als ein entsprechender Multi-Core-PC (bis zu 3000).

Warum ist die GPU so schnell?

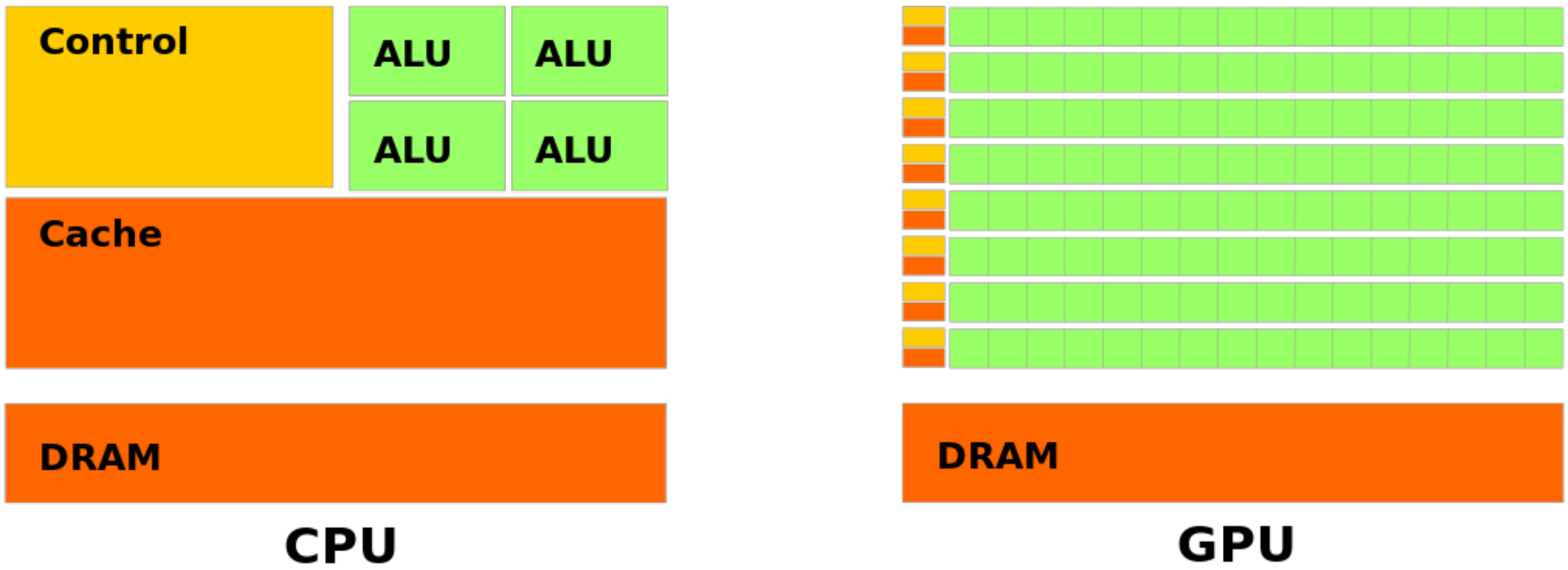

Abbildung 5: CPU vs. GPU [5]

GPGPU als Stein der Weisen?

Stärken

- sehr viele Threads können gleichzeitig arbeiten $\Rightarrow$ MPP möglich
- recht preiswerte Rechenleistung

Schwächen

- alle Threads arbeiten den selben Code ab
- vereinfachte Logik
- Konditionale und Schleifen tlw. sehr aufwendig
- Kommunikation zwischen Threads aufwendig
- Datenstrukturen sollten zur GPU passen

Wie kann ich die GPU einbinden?

Früher

- starke Orientierung an der Grafikpipeline
- sehr limitierte Möglichkeiten
- kaum Kontrollfluss außer Sequenz
- basierend auf Grafik-APIs

Heute

- Architektur ist flexibler
- CUDA (NVIDIA), Stream (ATI), ...
- weiterhin Geschwindigkeitseinbußen durch bestimmte Strukturen

$\Rightarrow$ Die Mittel müssen zur Karte passen, der Code auch!

Übersicht

Was ist CUDA?

Plattform zur Entwicklung von Programmteilen (ausschließlich) für NVIDIA-Karten.

- unterstützt übliche Programmstrukturen
- Nutzung als GPGPU wird ermöglicht
- Nutzung der Special Function Units
- Programmierung in CUDA C/C++ oder CUDA Fortran
- Bindings für viele Sprachen vorhanden, auch Python (PyCUDA, NumbaPro,...)
- weitere Bibliotheken und Tools vorhanden

Meist werden nur (geeignete) Teile des Programms in CUDA ausgelagert.

Wie arbeitet CUDA?

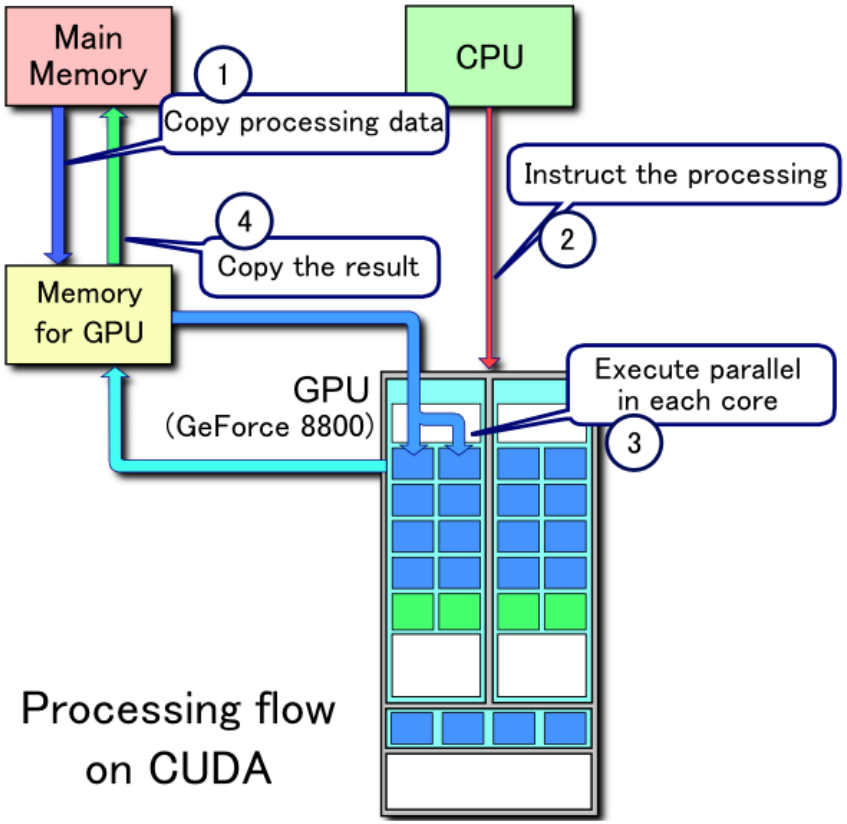

Abbildung 6: [6]

- Daten auf die Karte laden
- Kernel auf die Karte laden
- ein Stapel von Threads wird starteten
- alle Threads führen den selben Kernel aus
- die Daten von der Karte laden

Wie arbeitet CUDA?

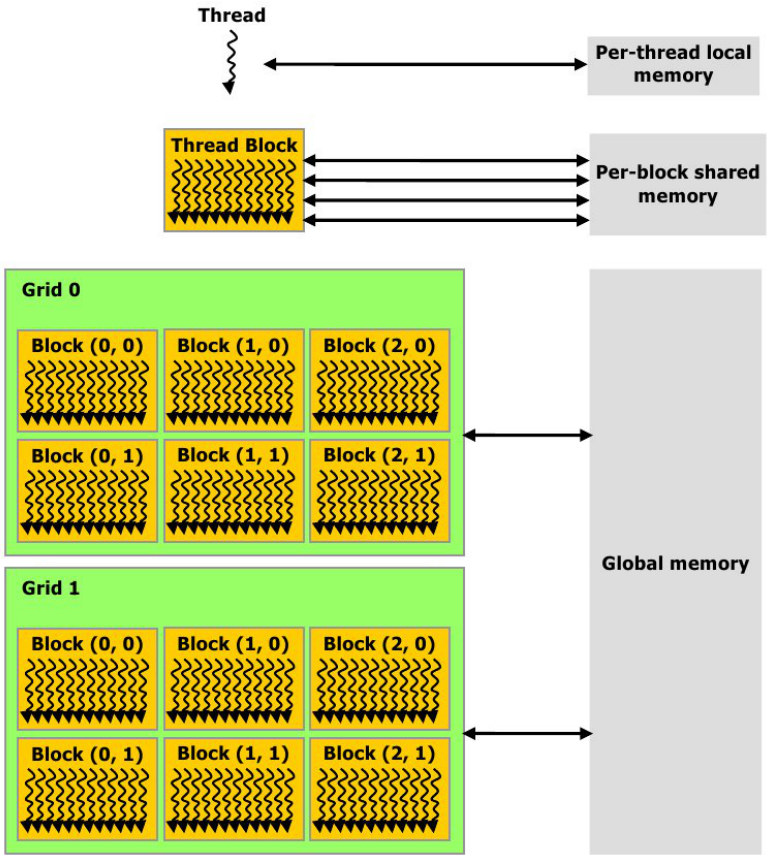

Abbildung 7: [5]

- jeder Thread hat eigenen Speicher
- Threads treten immer in Blöcken auf
- jeder Block hat einen gemeinsamen Speicher
- Blocks sind in Grid angeordnet
- keine Kommunikation über Blockgrenzen
- alle Grids haben einen Kontextspeicher (von außen zugreifbar)

Ein PyCUDA-Beispiel I [3]

```
1  import pycuda.driver as cuda
2  import pycuda.autoinit
3  from pycuda.compiler import SourceModule
4
5  a = numpy.random.randn(4,4)
6  a = a.astype(numpy.float32)
7  a_gpu = cuda.mem_alloc(a.nbytes)
8  cuda.memcpy_htod(a_gpu, a)
9
10 func = mod.get_function("doublify") # Kernel
11 func(a_gpu, block=(4,4,1))
12
13 a_doubled = numpy.empty_like(a)
14 cuda.memcpy_dtoh(a_doubled, a_gpu)
```

Ein PyCUDA-Beispiel II [3]

```
1  mod = SourceModule("""
2      __global__ void doublify(float *a)
3      {
4        int idx = threadIdx.x + threadIdx.y*4;
5        a[idx] *= 2;
6      }
7      """)
```

Aufruf:

```
10 func = mod.get_function("doublify") # Kernel
11 func(a_gpu, block=(4,4,1))
```

Das Wichtigste in der Übersicht I

Kontext erzeugen, die Karte als Device registrieren und CUDA vorbereiten.

```
2 import pycuda.autoinit
```

Speicher allokieren und Daten auf die Karte schreiben (Host to Device - htod).

```
7 a_gpu = cuda.mem_alloc(a.nbytes)
8 cuda.memcpy_htod(a_gpu, a)
```

Das Wichtigste in der Übersicht II

Definierte Kernelfunktion holen und mit Parametern sowie Thread-Blockgröße aufrufen. Auch Grid-Größe u.a. kann spezifiziert werden.

```
10 func = mod.get_function("doublify") # Kernel
11 func(a_gpu, block=(4,4,1))
```

Daten von der Karte kopieren (Device to Host - dtoh).

```
14 cuda.memcpy_dtoh(a_doubled, a_gpu)
```

Alternative:

```
11 func(cuda.InOut(a), block=(4, 4, 1))
```

Das Beispiel in ultrakompakter Schreibweise mit GPUArray

```
1 import pycuda.gpuarray as gpuarray
2 import pycuda.driver as cuda
3 import pycuda.autoinit
4 import numpy
5
6 a_gpu = gpuarray.to_gpu(numpy.random.randn
      (4,4).astype(numpy.float32))
7 a_doubled = (2*a_gpu).get()
```

Viele weitere klassische Operationen verfügbar, tlw. auch in anderen Paketen wie pycuda.cumath.

Ein komplexerer Kernel [4]

```
kernel = SourceModule("""
  __global__ void threshold(float *x, int len)
  {
  int idx = blockIdx.x * blockDim.x +
      threadIdx.x;
  int numThreads = blockDim.x * gridDim.x;
  for ( int i = idx ;i < len ;i+=numThreads)
   x[i] = -1 < x[i] && x[i] < +1 ? 1.0 : 0.0;
  } """)
```

Aufruf:

```
threshold = kernel.get_function('threshold')
threshold(x_gpu, len(x), block=(256,1,1), grid
    =(16,1))
```

Was heißt schneller? [7]

Bildgröße	CPU(ms)	GPU(ms)	Speedup
5120x5120	1746.40	37.29	46.8
1600x1200	128.51	3.25	39.5
1024x816	54.60	1.79	30.5
512x512	17.53	0.93	18.8
320x408	8.83	0.97	9.1

Tabelle 1: Histogram-Equalization

Bildgröße	CPU (ms)	GPU (ms)	Speedup
4096x4096	31925.99	402.01	79.4
700x525	1745.02	21.98	79.7

Tabelle 2: Wolkenentfernung

Was heißt schneller? [1]

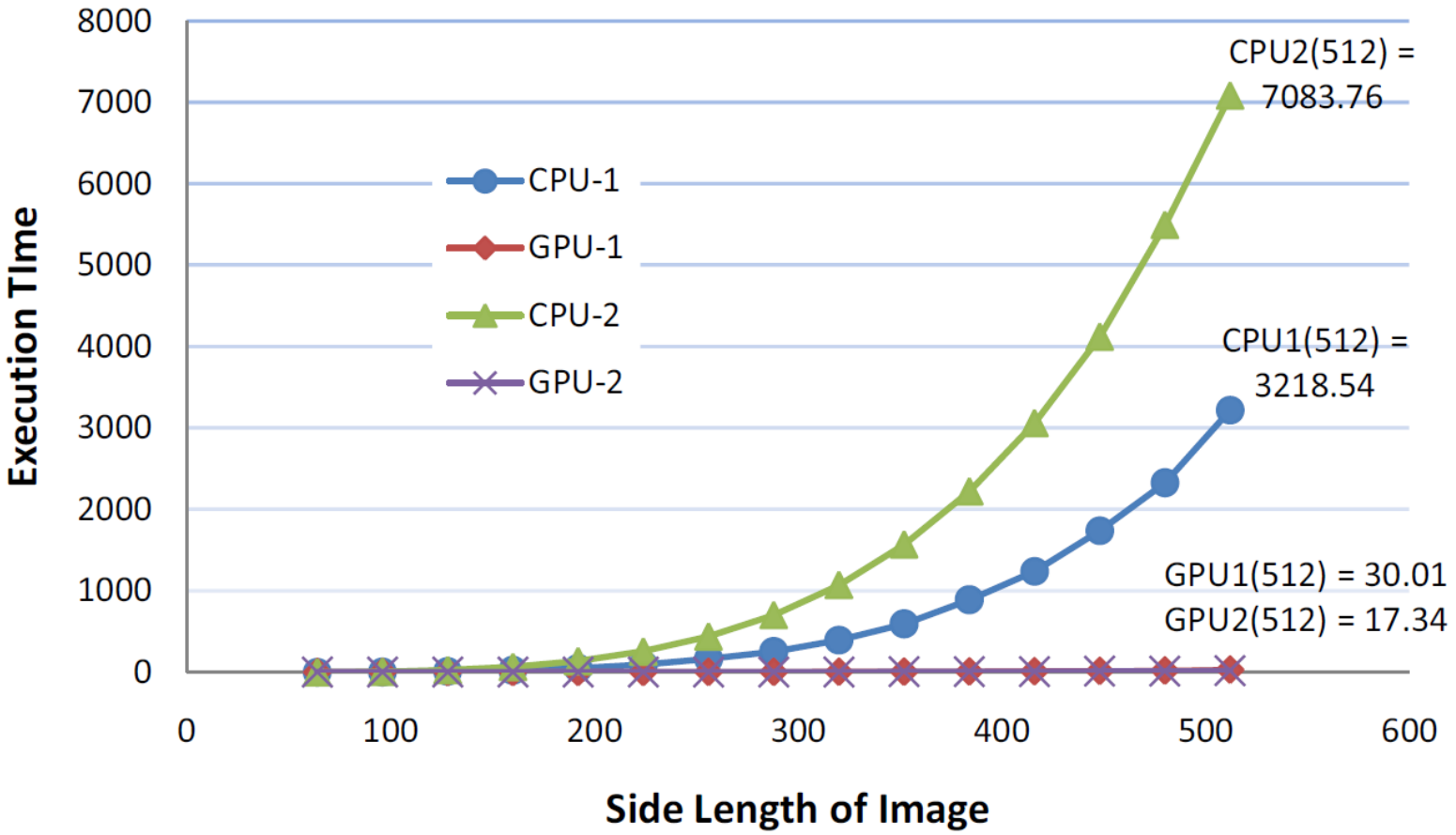

Abbildung 8: Hough-Transformation [1]

Übersicht

Was soll ich mitnehmen?

- SIMD-Architektur als Grundlage der GPU
- Schnelligkeit durch viele spezialisierte Prozessoren und kleines Aufgabenspektrum
- Probleme mit komplexeren Kontrollstrukturen und großer Kommunikation
- CUDA/PyCUDA ermöglicht einfache Programmierung und erweitere Möglichkeiten
- eigene Kernels in CUDA C bieten große Möglichkeiten
- Abstraktion durch GPUArrays in PyCuda

$\Rightarrow$ Vieles lässt sich mit der GPU stark beschleunigen, aber nicht alles!

Quellen I

S. Chen, H. Jiang

Accelerating the Hough Transform with CUDA on Graphics Proecessing Units

International Conference on Parallel and Distributed Processung Techniques and Applications, 2011

M. Garrigues

Image Processing on GPU with CUDA and C++

ENSTA-ParisTech, 2011

A. Kloeckner

Tutorial - PyCUDA

PyCUDA Documentation, 2013

Quellen II

M. Lazaro-Gredilla
An introduction to CUDA using Python
Universidad Carlos III de Madrid, 2013

T. O'Neil
CUDA Lecture
University of Akron, 2011

Tosaka
CUDA processing flow
Wikipedia

Z. Yang, Y. Zhu, Y. Pu.
Parallel Image Processing Based on CUDA
International Conference on Computer Science and Software Engineereing, 2008

Quellen III

C. Zeller
Tutorial CUDA
NVIDIA Developer Technology, 2008