JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

DEZSYS

GPGPU

Systemtechnik DEZSYS
5BHITT 2015/16

Burkhard Hampl, Simon Wortha

Version 1

Note:

Betreuer: Borko/Micheler

Begonnen am 8. Jänner 2016

Beendet am 24. Oktober 2016

Inhaltsverzeichnis

1 Einführung

1.1 Ziele

Die Aufgabe beinhaltet eine Recherche über grundsätzliche Einsatzmöglichkeiten für GPGPU. Dabei soll die Sinnhaftigkeit der Technologie unterstrichen werden. Die Fragestellungen sollen entsprechend mit Argumenten untermauert werden.
Im zweiten Teil der Arbeit soll der praktische Einsatz von OpenCL trainiert werden. Diese können anhand von bestehenden Codeexamples durchgeführt werden. Dabei wird auf eine sprechende Gegenüberstellung (Benchmark) Wert gelegt.
Die Aufgabenstellung soll in einer Zweiergruppe bearbeitet werden.

1.2 Aufgabenstellung

Informieren Sie sich über die Möglichkeiten der Nutzung von GPUs in normalen Desktop-Anwendungen. Zeigen Sie dazu im Gegensatz den Vorteil der GPUs in rechenintensiven Implementierungen auf [1Pkt].

Gibt es Entwicklungsumgebungen und in welchen Programmiersprachen kann man diese nutzen [1Pkt]?

Können bestehende Programme (C/C++ und Java) auf GPUs genutzt werden und was sind dabei die Grundvoraussetzungen dafür [1Pkt]? Gibt es transcompiler und wie kommen diese zum Einsatz [1Pkt]?

Präsentieren Sie an einem praktischen Beispiel den Nutzen dieser Technologie. Wählen Sie zwei rechenintensive Algorithmen (z.B. Faktorisierung) und zeigen Sie in einem aussagekräftigen Benchmark welche Vorteile der Einsatz der vorhandenen GPU Hardware gegenüber dem Ausführen auf einer CPU bringt (OpenCL). Punkteschlüssel:

Auswahl und Argumentation der zwei rechenintensiven Algorithmen (Speicher, Zugriff, Rechenoperationen) [0..4Pkt]

Sinnvolle Gegenüberstellung von CPU und GPU im Benchmark [0..2Pkt]

Anzahl der Durchläufe [0..2Pkt]

Informationen bei Benchmark [0..2Pkt]

Beschreibung und Bereitstellung des Beispiels (Ausführbarkeit) [0..2Pkt]

2 Ergebnisse

2.1 Desktopanwendungen

Eine mögliche Desktopanwendung wäre mehrere gleichzeitige Berechnungen in Excel zu optimieren. Für viele kleine Rechenschritte, die gleichzeitig ausgeführt werden, ist eine GPU um einiges schneller als eine CPU, da eine GPU viel mehr Kerne hat. Der Nachteil ist, dass alles Gleichzeitig ausgeführt werden muss und deshalb die GPU nicht für alle Zwecke geeignet (bzw. nicht immer schneller) ist.

Vorteile

„Grafikkarten sind auf massive Parallelität ausgelegt. Es können je nach Anzahl an programmierbaren Shadereinheiten über 1000 Berechnungen zugleich in einem Takt ausgeführt werden. Applikationen die parallelisierbare Algorithmen (Kapitel Arlgorithmen) verwenden können mit der Verwendung der GPU intensive Berechnungen auslagern und beschleunigen." [1]

Nachteile

„Da eine Grafikkarte parallel arbeitet stellt Sie die Entwickler vor neue Herausforderungen. Algorithmen müssen für den parallelen Einsatz optimiert oder entwickelt werden.
Ein vernünftiges GPGPU Cluster zu betreiben ist sehr teuer. Um auf kurze Sicht Kosten zu sparen können zum Beispiel bei Amazon Webservices GPU Instanzen gemietet werden (siehe Kapitel XaaS)" [1]

2.2 Entwicklungsumgebungen und Programmiersprachen

Entwicklungsumgebungen

Von NVIDIA gibt es das Nsight Plug-in für Visual Studio und eine eigene Eclipse Version für CUDA-c Applikationen.

Visual Studio:

„NVIDIA® Nsight™ Development Platform, Visual Studio Edition brings GPU Computing into Microsoft Visual Studio (including VS2015 Community Edition). Build, Debug, Profile and Trace heterogeneous compute and graphics applications using CUDA C/C++, OpenCL, DirectCompute, Direct3D, and OpenGL." [2]

Eclipse:

„NVIDIA® Nsight™ Eclipse Edition is a full-featured IDE powered by the Eclipse platform that provides an all-in-one integrated environment to edit, build, debug and profile CUDA-C applications. Nsight Eclipse Edition supports a rich set of commercial and free plugins." [3]

Programmiersprachen

Der Host-Code kann in ziemlich jeder Sprache geschrieben werden (Dieses Code konfiguriert die Grafikkarte). Der Teil der direkt auf der GPU ausgeführt wird, kann zum Beispiel in OpenCl – c gescchrieben werden (einer c ähnlich Sprache speziell für diesen Zweck). [1]

Weiters gibt es CUDA-c von NVIDIA, welches ähnlich wie OpenCl ist. (aber halt nicht open ist)

2.3 Bestehende Programme

Es gibt Tools die das ermöglichen. Es ist aber nicht bei jedem Programm zu empfehlen, da dies bei manchen Anwendungen zu einer Steigerung der Zeit führen kann. Wir haben dafür zwei konkrete Tools gefunden, welche in 2.4 Transcompiler genauer beschrieben sind

2.4 Transcompiler

Hier gibt es das Tool Aparapi. Dieses ermöglicht Java Code auf der GPU auszuführen. Aparapi konvertiert den Java-Bytecode zu OpenCl zur Laufzeit. Der OpenCl-Code wird dann auf der GPU ausgeführt. Sollte die Übersetzung aus irgendeinen Grund nicht möglich sein, wird der Code von Aparapi in einen Java-thread-pool ausgeführt. [4]

Weiters gibt es die Java Bibliothek „Rootbeer" die es ermöglicht Java-Code auf der GPU auszuführen. „Rootbeer" verwendet dazu CUDA von NVIDIA. [8]

„The Rootbeer GPU Compiler lets you use GPUs from within Java. It allows you to use almost anything from Java on the GPU:

- *Composite objects with methods and fields*
- *Static and instance methods and fields*
- *Arrays of primitive and reference types of any dimension.*" [8]

2.5 Auswahl und Argumentation der Algorithmen

PI

Wir berechnen Pi auf 32 Millionen Nachkommastellen mithilfe von GPUPI. Dies ist ein Tool welches uns die Programmierarbeit abnimmt und sowohl für GPU als auch CPU geeignet ist. Ausfolgenden Grund haben wir uns für diesen Algorithmus entschieden:

„GPUPI calculates the mathematical constant Pi in parallel by using the BPP formula and optimizing it for OpenCL capable devices like graphics cards and main processors. It's implemented with C++, STL and pure Win32 to avoid unnecessary dependencies. The result of the benchmark are exactly nine digits of pi in hexadecimal. So if you're calculating pi in 1B (1 billion) it will not output all digits like available serial calculations of Pi, but display only the nine digits after the billionth hexadecimal digit. This limitation is due to the nature of parallel implementations and the used Pi formula." [5]

Die Berechnung funktioniert so:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

Arrays

Hier werden mit vielen Operationen die CPU bzw. GPU belastet. Dabei werden ca. 8 Millionen data points verwendet, was ca. 32 MB an Daten bedeutet.

```
data_points = 2**23 # ~8 million data points, ~32 MB data
workers = 2**8 # 256 workers, play with this to see performance differences
                        # eg: 2**0 => 1 worker will be non-parallel execution on
gpu
                        # data points must be a multiple of workers
a = numpy.random.rand(data_points).astype(numpy.float32)
b = numpy.random.rand(data_points).astype(numpy.float32)
c_result = numpy.empty_like(a)
# Speed in normal CPU usage
time1 = time()
```

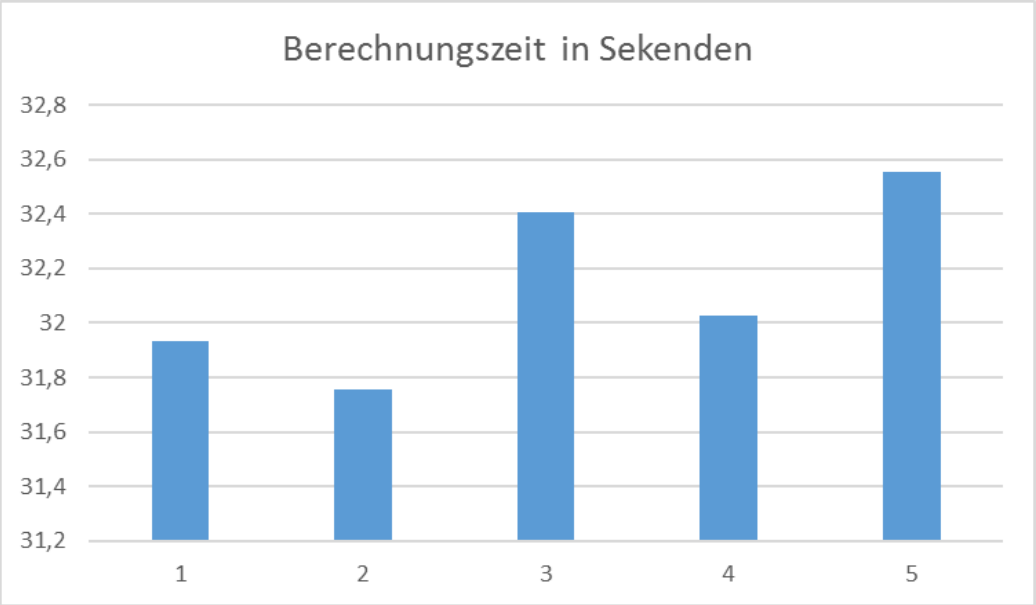
```
c_temp = (a+b) # adds each element in a to its corresponding element in b
c_temp = c_temp * c_temp # element-wise multiplication
c_result = c_temp * (a/2.0) # element-wise half a and multiply
time2 = time()
```

Dieses Beispiel kommt von github. [7]

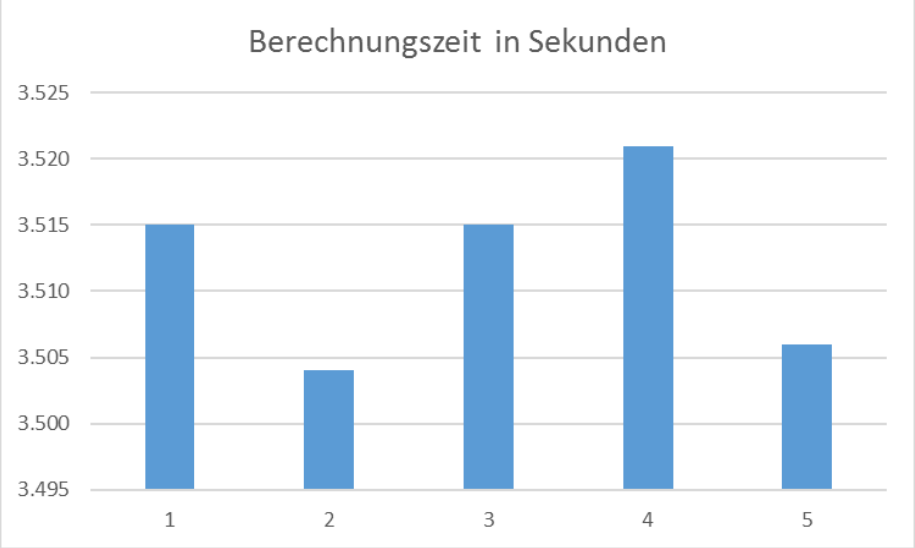
2.6 Gegenüberstellung

Pi

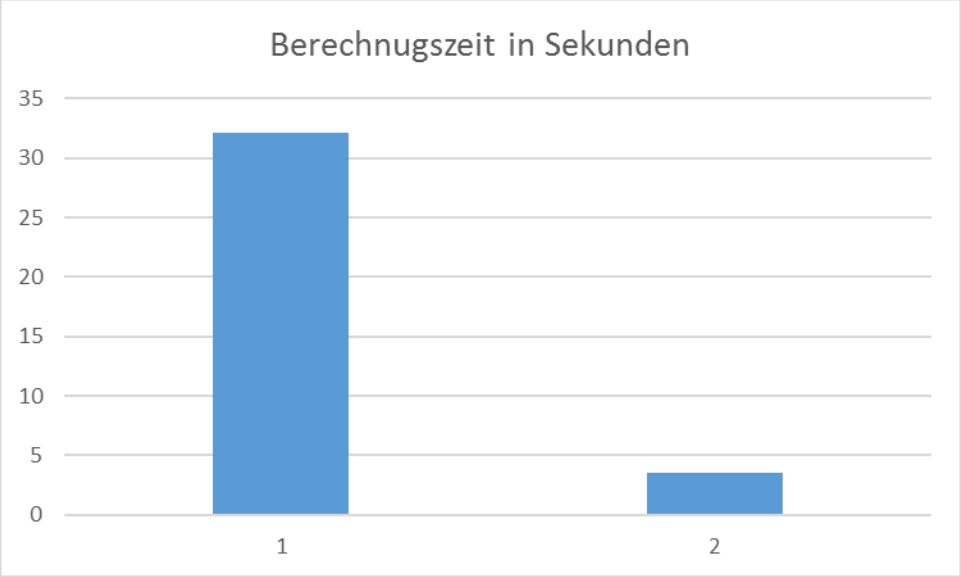
Hier die 5 Durchläufe auf der CPU:



Und hier die Durchläufe auf der GPU:



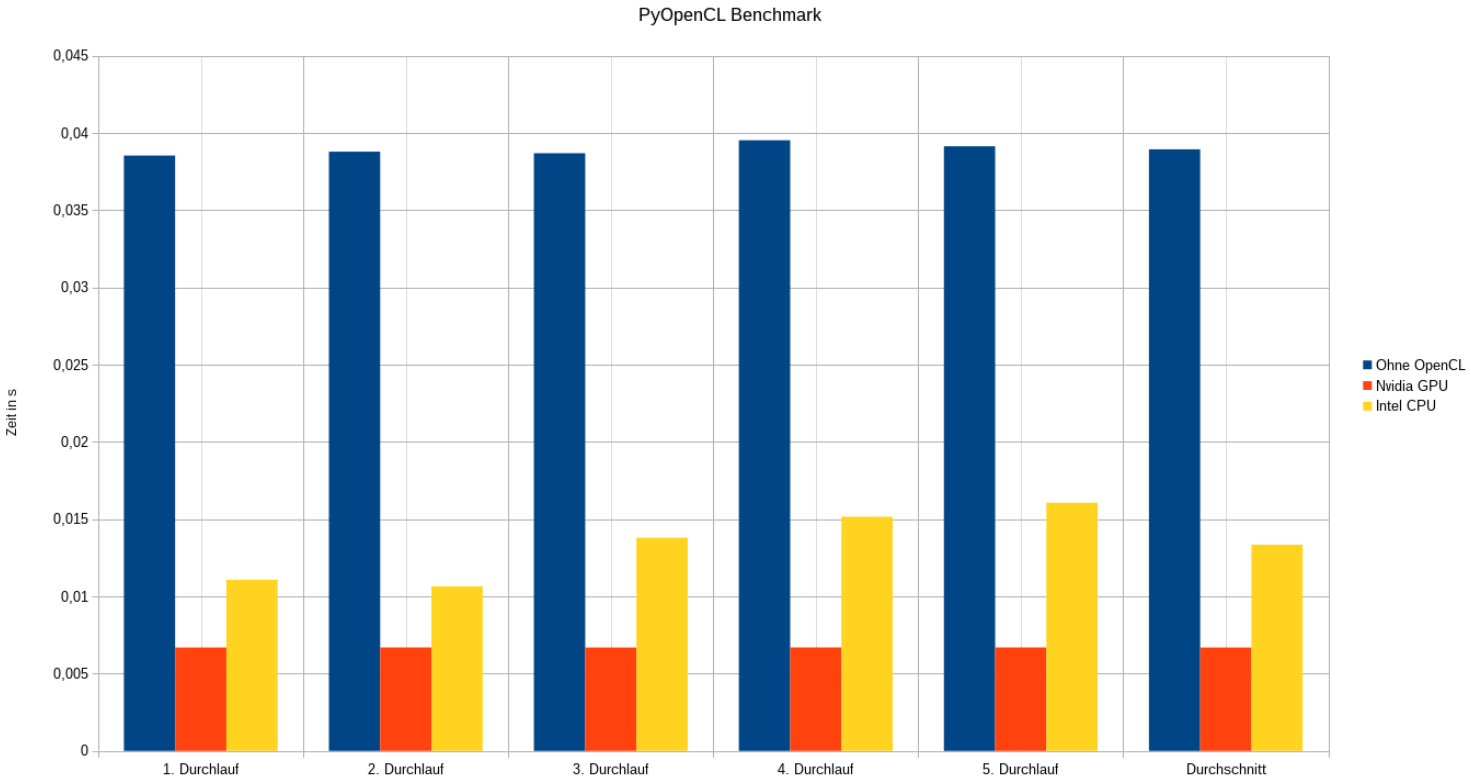
Der Durchschnitt der CPU- und GPU-Durchläufe gegenübergestellt:
(Wobei 1 die CPU ist und 2 die GPU)



Für genauere Information über Hardware, Zeiten usw. siehe txt Dateien im Ordner „Pi-Auswertung".

Array

Hier die Auswertung des Array-Benchmarks:



Für genauere Informationen über die Hardware siehe python-benchmark.txt.

Für diese Benchmark wurde auch der Code mitangegeben und zwar in der Datei benchmark.py.

Feedback

2.7 Probleme

Probleme:
LWJGL:
Nach dem ich ein funktionierendes LWJGL 3 Beispiel gefunden habe, haben wir versucht dies auszuführen, doch dabei ist uns auf Hampls PC (Arch Linux) eine komische Exception begegnet:

```
CL created
Exception in thread "main" java.lang.IllegalArgumentException: Negative capacity: -2147483584
        at java.nio.Buffer.<init>(Buffer.java:199)
        at java.nio.ByteBuffer.<init>(ByteBuffer.java:281)
        at java.nio.ByteBuffer.<init>(ByteBuffer.java:289)
        at java.nio.MappedByteBuffer.<init>(MappedByteBuffer.java:89)
        at java.nio.DirectByteBuffer.<init>(DirectByteBuffer.java:119)
        at java.nio.ByteBuffer.allocateDirect(ByteBuffer.java:311)
        at org.lwjgl.BufferUtils.createAlignedByteBuffer(BufferUtils.java:221)
        at org.lwjgl.BufferUtils.createAlignedByteBufferCacheLine(BufferUtils.java:252)
        at org.lwjgl.system.APIBuffer.ensureCapacity(APIBuffer.java:101)
        at org.lwjgl.system.APIBuffer.param(APIBuffer.java:119)
        at org.lwjgl.system.APIBuffer.bufferParam(APIBuffer.java:148)
        at org.lwjgl.opencl.CLPlatform.createCapabilities(CLPlatform.java:70)
        at org.lwjgl.opencl.CLPlatform.<init>(CLPlatform.java:34)
        at org.lwjgl.opencl.CLPlatform$1.create(CLPlatform.java:133)
        at org.lwjgl.opencl.CLPlatform$1.create(CLPlatform.java:130)
        at org.lwjgl.opencl.CLPlatform.filterObjects(CLPlatform.java:190)
        at org.lwjgl.opencl.CLPlatform.getPlatforms(CLPlatform.java:130)
        at org.lwjgl.opencl.CLPlatform.getPlatforms(CLPlatform.java:100)
        at at.ac.tgm.hit.syt.dezsys.hamplwortha.OpenCLSumY.main(OpenCLSumY.java:47)
```

Zu dieser Exception findet man nichts im Internet und dieser Fehler tritt nur auf Hampls PC auf. Darauf hin haben wir die Entwickler gefragt, von den haben wir aber keine Antwort bekommen.
http://echelog.com/logs/browse/lwjgl/1453158000

2.8 Zeitaufzeichnung

Hampl		**Wortha**	
Datum	**Zeit**	**Datum**	**Zeit**
08.01.2016	2 Stunden	08.01.2016	2 Stunden
15.01.2016	2 Stunden	15.01.2016	2 Stunden
21.01.2016	1 Stunden	21.01.2016	1 Stunden
22.01.2016	4 Stunden	22.01.2016	4 Stunden
24.01.2016	3 Stunden	24.01.2016	4 Stunden
Gesamt:	**12 Stunden**	**Gesamt:**	**12 Stunden**

3 Quellen

[1]: Paul Kalauner, Rene Hollander; GPGPU;
Online: https://elearning.tgm.ac.at/mod/resource/view.php?id=45929
zuletzt abgerufen: 21.01.2016

[2]: NVIDEA; NVIDIA Nsight Visual Studio Edition;
Online: https://developer.nvidia.com/nvidia-nsight-visual-studio-edition
zuletzt abgerufen: 21.1.2016

[3]: NVIDEA; Nsight Eclipse Edition;
Online: https://developer.nvidia.com/nsight-eclipse-edition
zuletzt abgerufen: 21.1.2016

[4]: Aparapi;
Online: https://code.google.com/p/aparapi/
zuletzt abgerufen: 22.01.2016

[5]: GPUPI;
Online: https://www.overclockers.at/news/gpupi-international-support-thread#attachmentid-198371
zuletzt abgerufen: 24.01.2016

[7]: pyopencl;

online:
https://github.com/pyopencl/pyopencl/blob/v2015.2.3/examples/benchmark.py
zuletzt abgrufen: 24.01.2016

[8]: Rootbeer;
online: https://github.com/pcpratts/rootbeer1
zuletzt abgerufen: 22.01.2015