

**DezSys-Protokoll**

**DezSysLabor-08 "GPGPU"**

**Dezentrale Systeme**

**5BHITT 2015/16**

**Manuel Reiländer, Thomas Stedronsky**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 15. Jänner 2016** |
| **Betreuer: Th. Micheler** | **Beendet 24.01.2016** |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc441398955)

[1.1 Ziele 3](#_Toc441398956)

[1.2 Aufgabenstellung 3](#_Toc441398957)

[2 Ergebnisse 4](#_Toc441398958)

[2.1 Theorie 4](#_Toc441398959)

[3 Quellen 6](#_Toc441398960)

# Einführung

Die Übung soll die Grundlagen einer GPU mit Hilfe eines praktischen Beispiels vertiefen.

## 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.

## 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]

# Ergebnisse

## Theorie

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.

* **GPUPI**

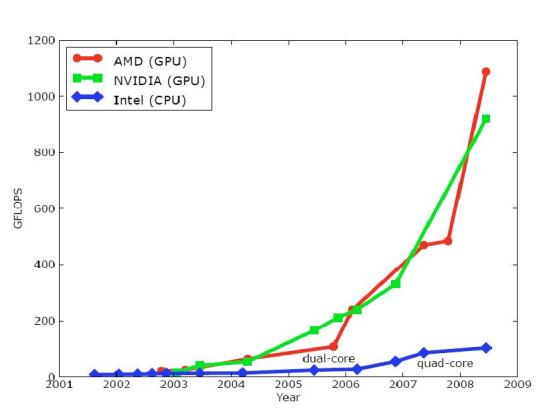
Ist eine Desktopanwendung die mithilfe der GPU sich die Nachkommastellen von PI ausrechnet. [1]

* **Mathematica**

Ist ein Rechenprogramm zur symbolischen Verarbeitung von Gleichungen, numerische Lösung oder Auswertung von Gleichungen, Visualisierung von 2D und 3D Graphiken, … [2]

* **MATLAB**

Ist ein Rechen –und Visualisierungsprogramm für zum Beispiel Mathematische Funktionen für lineare Algebra, Statistik, Fourier-Analyse, Filterung, Optimierung, numerische Integration und Lösung gewöhnlicher Differenzialgleichungen. [3]

Also ja prinzipiell ist es möglich die GPU in Desktopanwendungen zu verwenden. Allerdings muss immer eine Abschätzung erfolgen ob dies auch sinnvoll ist.  
[4]

Bei Rechenintensiven Anwendungen gibt es ganz klar einen Vorteil der GPUs wie man in der Grafik erkennen kann. Allerdings gibt es kaum Nutzungsgebiete bei normalen Desktopanwendungen. Meist erfordern normale Desktopanwendungen keine große Parallelität deswegen reichen dort meist CPUs aus. GPUs müssen allerdings nicht zwingend schneller sein, GPUs entfalten ihr großes Potential erst bei Anwendungen die enorm viel Parallelität in Anspruch nehmen.

Gibt es Entwicklungsumgebungen und in welchen Programmiersprachen kann man diese nutzen?

* Nsight Eclipse Edition[5] ist eine IDE die auf Eclipse basiert und sich auf die Implementierung von CUDA spezialisiert.
* APP SDK[6] ist eine von AMD entwickelte IDE für die Implementierung von GPU Anwendungen in Programmiersprachen wie OPENCL und C/C++.
* Für C/C++ gibt es außerdem noch Microsoft Lösung mit Visual Studio.
* Außerdem können für C/C++ Anwendungen noch die von JetBrains entwickelte IDE CLion[7] verwendet werden.
* …

Können bestehende Programme (C/C++ und Java) auf GPUs genutzt werden und was sind dabei die Grundvoraussetzungen dafür?

Es ist möglich bereits bestehende Java oder C/C++ Programme ausführen zu lassen. Was allerdings nicht bedeuten muss, dass dieses Programm dann schneller ist.

Für Java gibt es einen GPU Compiler, der möglich macht jeden beliebigen Java Code auf der GPU ausführen zu lassen. Dieser Code wird dann automatisch auf die parallele Architektur der GPU angepasst. Mit Rootbeer[8](so heißt dieser Compiler) muss auch kein eigener GPU-Kernel für CUDA oder OpenCL geschrieben werden. Bis auf dynamische Methodenaufrufe, Reflections und native Methoden sind alle Eigenschaften von Java unterstützt.

Für C/C++ gibt es ein eigenes CUDA Toolkit[9], mit diesen man Problemlos C/C++ Code auf der GPU ausführen kann. Es müssen keine weiteren Anpassungen vorgenommen werden.

Gibt es transcompiler und wie kommen diese zum Einsatz?

Allgemein ist ein transcompiler ein Compiler der einen Source Code einer Programmiersprache in eine andere Programmiersprache übersetzt.

Ja es gibt transcompiler bei GPUs zum Beispiel den gcompiler der von CUDA in OpenCL übersetzen kann.

Diese kommen zum Einsatz um Programme auf mehrere Programmiersprachen auszuführen um eventuelle Geschwindigkeitsunterschiede zu ermitteln. Außerdem kommt es zum Einsatz wenn man zum Beispiel eine NVIDIA Grafikkarte hat und diese mit einer anderen vergleichen will. Hierbei kann derselbe Code dann in CUDA, OpenCL, getestet werden.

## Praxis

Ein gutes Beispiel zum Testen ist die Berechnung von . Hierzu nimmt man folgende Folge

Will man nun PI berechnen, bildet man z.B. die Summe der Folge für und erhält ein bestimmtes Ergebnis. Damit man nun auf kommt muss man dieses Ergebnis mit multiplizieren und danach die Wurzel daraus ziehen. Je höher , desto genauer kommt man an .

Nun stellt sich die Frage, wie sich diese Berechnungen parallelisieren lassen, doch das ist im Prinzip ganz einfach. Will man diese Berechnung z.B. auf 2 Threads aufteilen, so lässt man den einen Thread die Summe der Folge für n von 1 bis 1000 ausrechnen und den anderen Thread für n von 1001 bis 10000. Zählt man nun beide Ergebnisse der Threads zusammen, kommt man auf ein Gesamtergebnis, welches dann mit 6 multipliziert und daraus die Wurzel gezogen werden muss.

## OpenCL code

\_\_kernel void calculate\_pi(\_\_global const int\* start, \_\_global const int\* end, \_\_global float\* out)  
{  
 int i;  
 float erg=0;  
 for(i=start;i < end;++i) {  
 erg += (float)(1)/(i\*i);  
 }  
 out[0]=erg;  
}

# Quellen

[1] GPUPI: International support thread, <https://www.overclockers.at/news/gpupi-international-support-thread>, zuletzt besucht am 24.01.2016

[2] [Wolfram Mathematica](https://www.wolfram.com/mathematica/), <https://www.wolfram.com/mathematica/>, zuletzt besucht am 24.01.2016

[3] Matlab, <http://de.mathworks.com/products/matlab/?requestedDomain=de.mathworks.com>, zuletzt besucht am 24.01.2016

[4] <https://intranet.einfochips.com/media/kunena/attachments/565/cpugpu.jpg>, zuletzt besucht am 23.01.2016

[5] Nsight Eclipse Edition, <https://developer.nvidia.com/nsight-eclipse-edition>, zuletzt besucht am 24.01.2016

[6] APP SDK – A Complete Development Platform, <http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/>, zuletzt besucht am 24.01.2016

[7] A cross-platform IDE for C and C++ JetBrains CLion, <https://www.jetbrains.com/clion/>, zuletzt besucht am 24.01.2016

[8] Java für die GPU, <http://www.golem.de/news/rootbeer-java-fuer-die-gpu-1208-93795.html>, zuletzt besucht am 24.01.2016

[9] GPU Accelerated Computing with C and C++, <https://developer.nvidia.com/how-to-cuda-c-cpp>, zuletzt besucht am 24.01.2016

Applications running GPGPU

<http://www.nvidia.com/content/gpu-applications/PDF/gpu-applications-catalog.pdf>

<http://www.nvidia.com/object/gpu-applications.html>

GPGPU Languages Support

<http://www.nvidia.com/content/GTC-2010/pdfs/2004_GTC2010.pdf>

Video rendern auf CPU oder GPU

https://handbrake.fr/