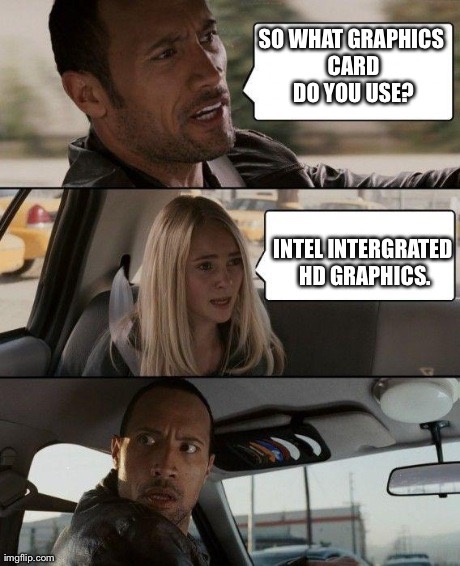
DEZSY09 GPGPU

**Ayvazyan Ari, Brunner Helmuth**

TGM Wien

21.04.2015

Inhaltsverzeichnis

[Aufgabenstellung 3](#_Toc417586170)

[Vorteile von GPGPU 4](#_Toc417586171)

[Aufgabenstellung 5](#_Toc417586172)

[Algorithmus 1 - GGT finden 5](#_Toc417586173)

[Implementierung 5](#_Toc417586174)

[Rahmenbedingungen: 5](#_Toc417586175)

[Benchmark Ergebnisse: 6](#_Toc417586176)

[Algorithmus 2 – fibonacci 12](#_Toc417586177)

[Implementierung: 12](#_Toc417586178)

[Rahmenbedinungen 12](#_Toc417586179)

[Benchmark Ergebnisse 13](#_Toc417586180)

[Starten des Programms: 16](#_Toc417586181)

# Aufgabenstellung

GPU Computing oder GPGPU(= General Purpose Computing on GPUs) bezeichnet die Verwendung eines Grafikprozessors (engl. Graphics Processing Unit oder GPU) für allgemeine Berechnungen im wissenschaftlich-technischen Bereich. Übersetzt bedeutet GPGPU in etwa Allgemeine Berechnung auf Grafikprozessoren.

Informieren Sie sich über die Möglichkeiten der Nutzung von GPUs in normalen 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 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 Benchmark welche Vorteile der Einsatz der vorhandenen GPU Hardware bringt [12Pkt]! Um auch einen Vergleich auf verschiedenen Platformen zu gewährleisten, bietet sich die Verwendung von OpenCL an.

Diese Aufgabe ist als Gruppenarbeit (2) zu lösen. Zusätzliche Abgaben erhöhen die Gesamtpunkte und können somit zur Notenverbesserung dienen.

Quellen

<http://www.nvidia.de/page/gpu_computing.html>

<http://developer.nvidia.com/cuda-gpus>

<http://people.maths.ox.ac.uk/gilesm/cuda/>

<http://www.khronos.org/opencl/>

# Vorteile von GPGPU

Zeigen Sie dazu im Gegensatz den Vorteil der GPUs in rechenintensiven Implementierungen auf [1Pkt].

GPGPU ist eine Programmierschnittstelle um Source-Code auf einer GPU ausführen zu können. Weiters kann durch das auslagern von Berechnungen auf die GPU, die vorhandene Hardware, optimal ausgenützt werden wenn diese gerade nicht gebraucht wird.

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

Ja es gibt Entwicklungsumgebungen die viele dieser werden direkt vom Hersteller zur Verfügung gestellt.

Nsight-Eclipse-Edition:

Nsight-Eclipse-Edition ist eine Entwicklungsumgebung für Nividias CUDA Implementation. Es werden die zwei Programmiersprachen CUDA C und C++ unterstützt.

<https://developer.nvidia.com/nsight-eclipse-edition>

SDAccel Development Environment

SDAccel Development Environment, ist eine Entwicklungsumgebung für OpenCL, C und C++. Diese wird über eine Eclipse integriertes Plugin in die Eclipse die eingebunden.

<http://www.xilinx.com/products/design-tools/sdx.htm.html>

Können bestehende Programme (C und Java) auf GPUs genutzt werden und was sind dabei die Grundvoraussetzungen dafür [1Pkt]?

Im Grunde kann jedes Programm egal ob C oder Java mit ein bisschen Arbeit auf einer GPU genutzt werden. Es besteh aber ein Irrglaube das gleich alle Programme die auf der GPU laufen schneller sind als die die auf einer CPU ausgeführt werden. GPUs sind spezielle für grafische Berechnungen spezialisiert und somit verarbeiten diese auch speziell geschrieben Code besser. So wie es FPUs gibt die schnell FlotingPoint-Operations verarbeiten können GPUs graphische Berechnungen schnell durchführen.

# Aufgabenstellung

Präsentieren Sie an einem praktischen Beispiel den Nutzen dieser Technologie. Wählen Sie zwei rechenintensive Algorithmen (z.B. Faktorisierung) und zeigen Sie in einem Benchmark welche Vorteile der Einsatz der vorhandenen GPU Hardware bringt [12Pkt]! Um auch einen Vergleich auf verschiedenen Platformen zu gewährleisten, bietet sich die Verwendung von OpenCL an.

## Algorithmus 1 - GGT finden

### Implementierung

Das Programm wurde auf Basis von OpenCL mittels LWJGL 2.8.4 umgesetzt

int Euklid(int a, int b);  
  
kernel void calc(global const float \*a, global const float \*b, global float \*answer) {  
 unsigned int xid = get\_global\_id(0);  
 answer[xid] = Euklid(a[xid],b[xid]);  
}  
  
int Euklid(int a, int b)  
{  
 if (a == 0) /\*\*Wenn a=0 ist b der größte gemeinsame Teiler laut Definition\*\*/  
 {  
 return b;  
 }  
 while(b != 0) /\*\*So lange wiederholen, wie b nicht 0 ist.\*\*/  
 {  
 if (a > b)  
 {  
 a = a - b; /\*\*Wenn a größer als b, subtrahiere b von a.\*\*/  
 }  
 else  
 {  
 b = b - a; /\*\*In jedem anderen Fall subtrahiere a von b.\*\*/  
 }  
 }  
 return a; /\*\*In a steht jetzt der größte gemeinsame Teiler von a und b.\*\*/  
}

### Rahmenbedingungen:

1,000 Benchmarks

100,000 (\*2) Werte pro Benchmark, zu denen der GGT berechnet wird

### Benchmark Ergebnisse:

#### OSX

Device #0(CPU):Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz

Compute Units: 8 @ 2600 mghtz

Local memory: 32 KB

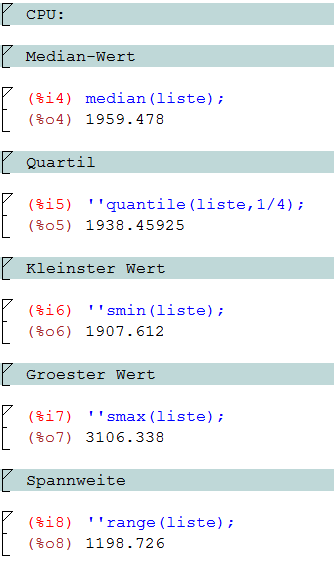
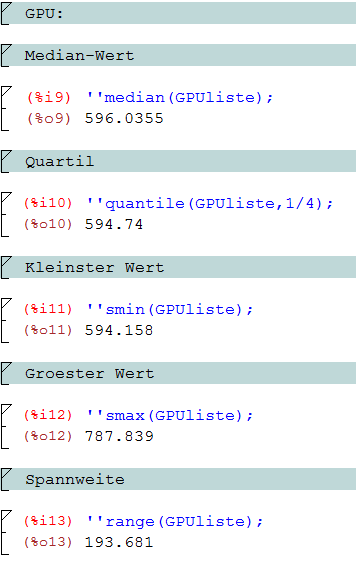
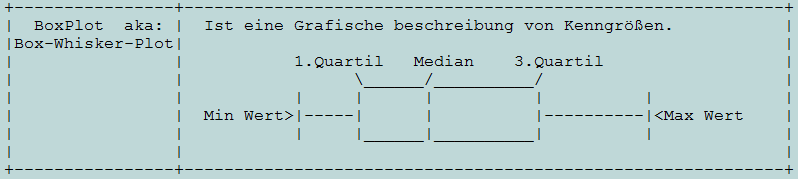
Global memory: 16 GB

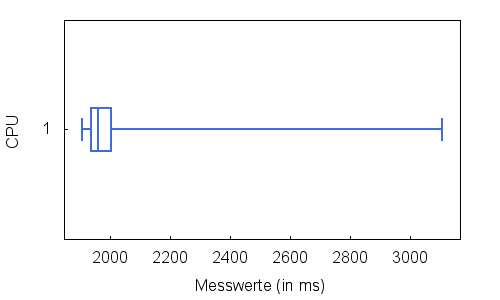
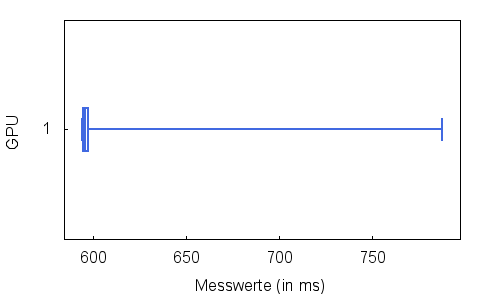
Device #1(GPU):HD Graphics 4000

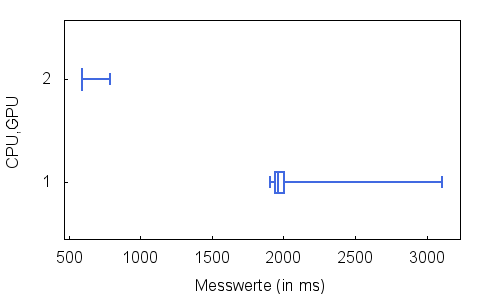
Compute Units: 16 @ 1250 mghtz

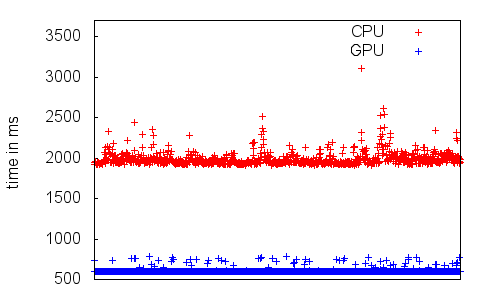
Local memory: 64 KB

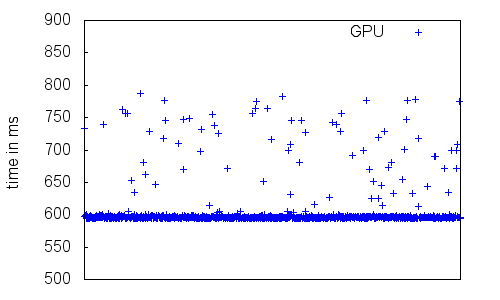
Global memory: 1 GB

Wertangaben in millisekunden 







#### Windows 8.1 pro

Platform #0:NVIDIA CUDA

Device #0(GPU):GeForce GTX 560 Ti (SLI) Device #1(GPU):GeForce GTX 560 Ti (SLI)

Compute Units: 8 @ 1760 mghtz Compute Units: 8 @ 1760 mghtz

Local memory: 48 KB Local memory: 48 KB

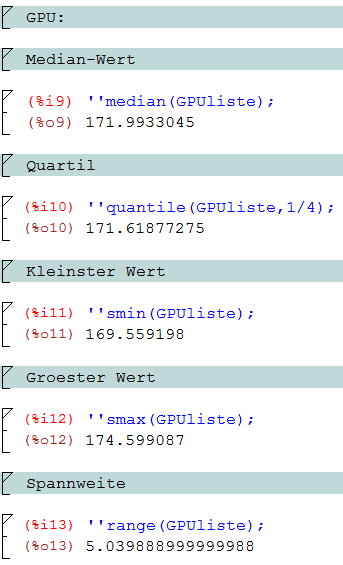
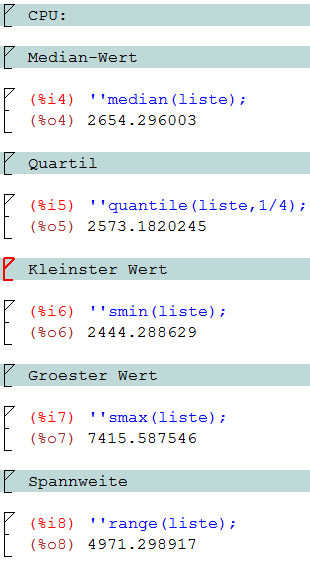
Global memory: 1 GB Global memory: 1 GB

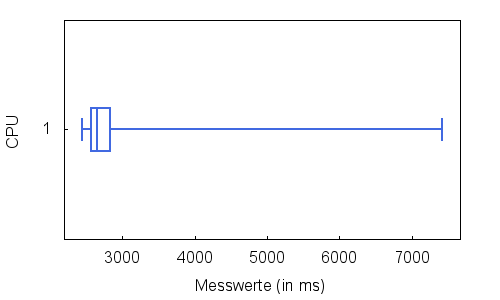
Platform #1:Intel(R) OpenCL

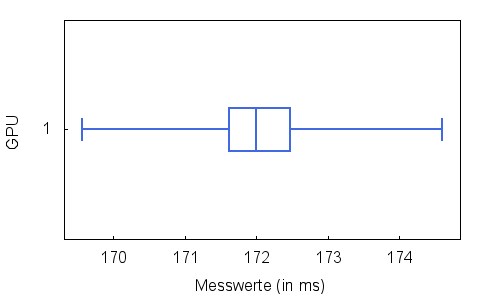
Device #0(CPU): Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz

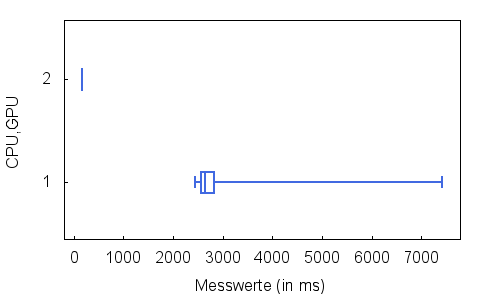
Compute Units: 4 @ 3300 mghtz

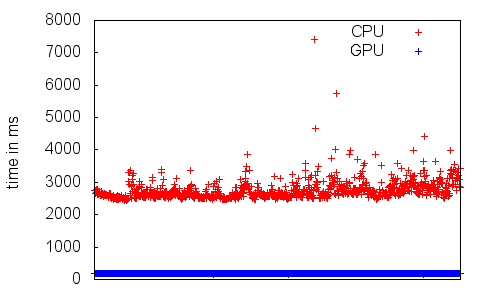
Local memory: 32 KB

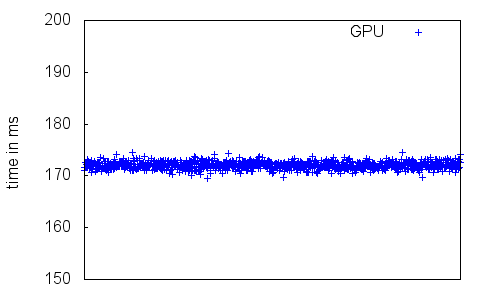
 Global memory: 8 GB











## Algorithmus 2 – fibonacci

### Implementierung:

**float** Fibonacci(**float** n);  
  
kernel **void** calc(global **const float** \*a, global **const float** \*b, global **float** \*answer) {  
 **unsigned int** xid = get\_global\_id(0);  
 answer[xid] = Fibonacci(a[xid]);  
}  
  
**float** Fibonacci(**float** n) {  
 **if**(n <= 0) **return** 0;  
 **if**(n > 0 && n < 3) **return** 1;  
  
 **float** result = 0;  
 **float** preOldResult = 1;  
 **float** oldResult = 1;  
  
 **for** (**int** i=2;i<n;i++) {  
 result = preOldResult + oldResult;  
 preOldResult = oldResult;  
 oldResult = result;  
 }  
  
 **return** result;  
}

### Rahmenbedinungen

100 Benchmarks

100,000 Werte pro Benchmark, zu denen die fibonacci Zahl berechnet wird

### Benchmark Ergebnisse

#### Windows 8.1 pro

Benchmarks: 100; pause zwischen den benchmarks: ~100ms

Platform #0:NVIDIA CUDA

Device #0(GPU):GeForce GTX 560 Ti (SLI) Device #1(GPU):GeForce GTX 560 Ti (SLI)

Compute Units: 8 @ 1760 mghtz Compute Units: 8 @ 1760 mghtz

Local memory: 48 KB Local memory: 48 KB

Global memory: 1 GB Global memory: 1 GB

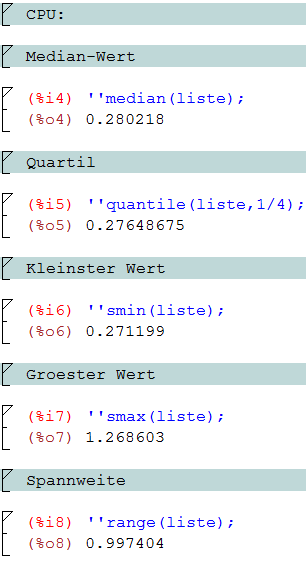
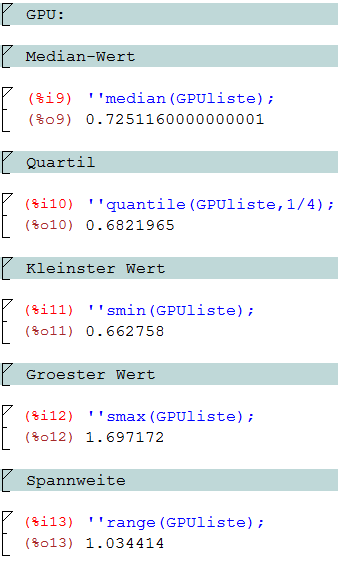
Platform #1:Intel(R) OpenCL

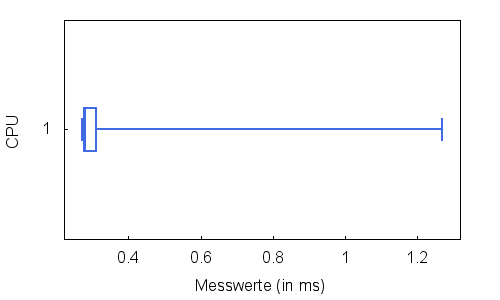
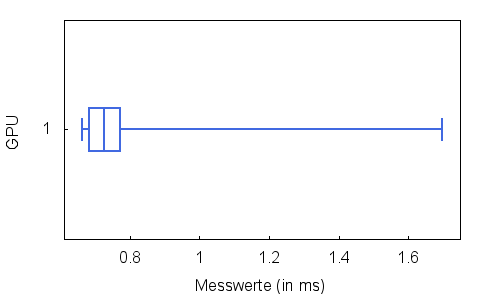
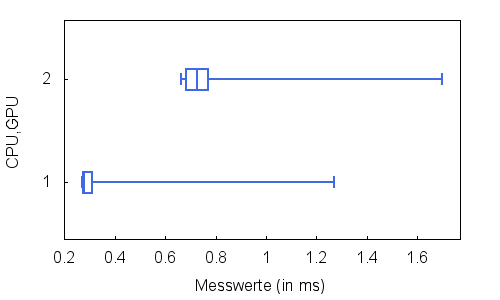
Device #0(CPU): Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz

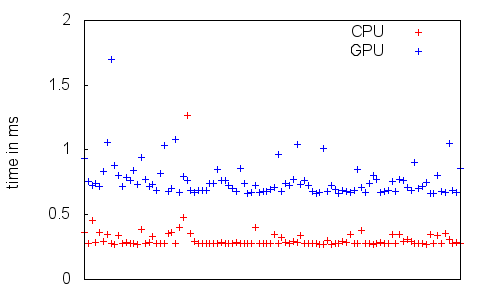
Compute Units: 4 @ 3300 mghtz

Local memory: 32 KB

Global memory: 8 GB





# Starten des Programms:

Windows: gradlew run

Unix: ./gradle run

Starten von algorithmus 1: **“calc“** als Parameter

Straten von algorithmus 2: keine parameter