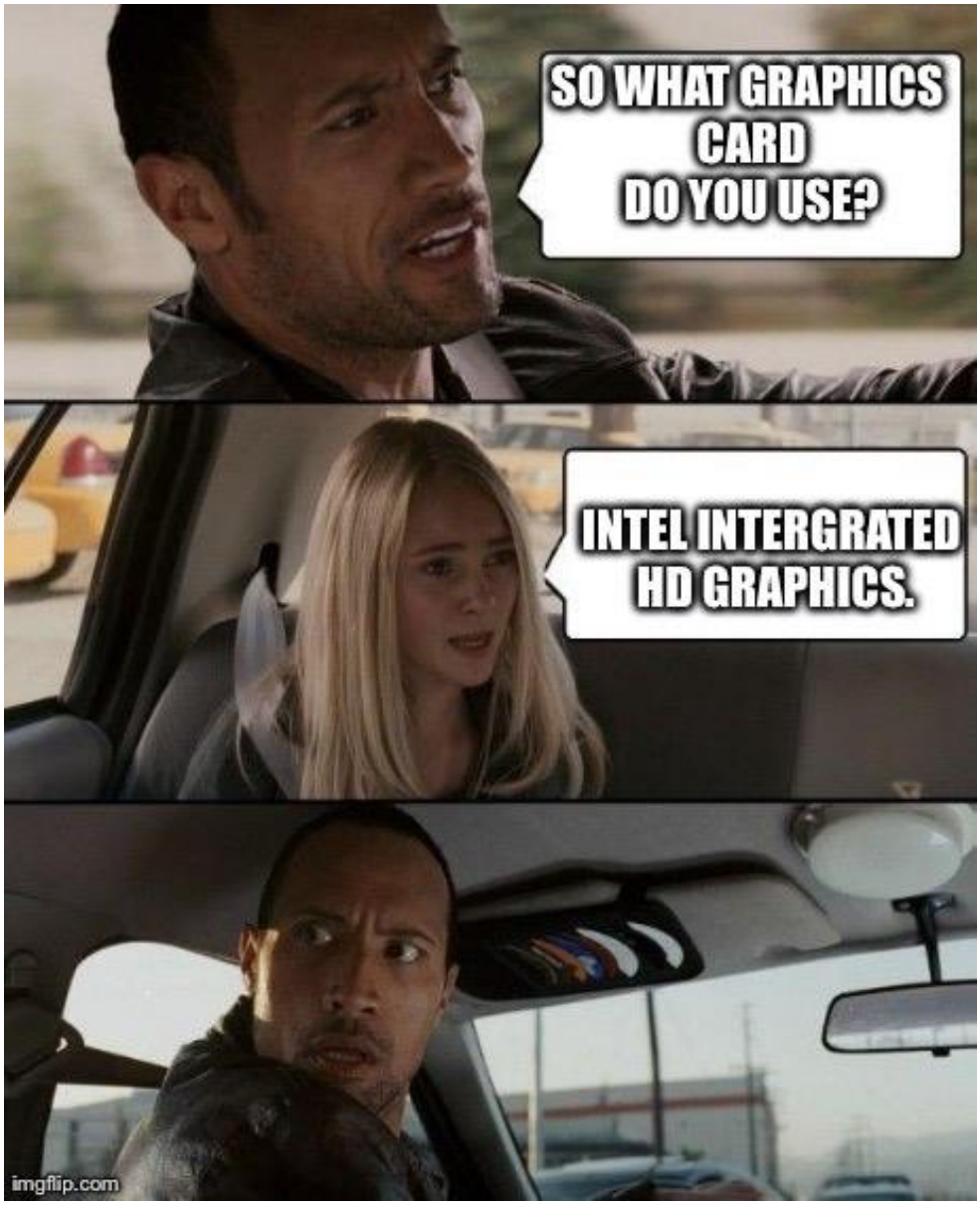
DEZSY09 GPGPU

Ayvazyan Ari, Brunner Helmuth



TGM Wien
21.04.2015

Inhaltsverzeichnis

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

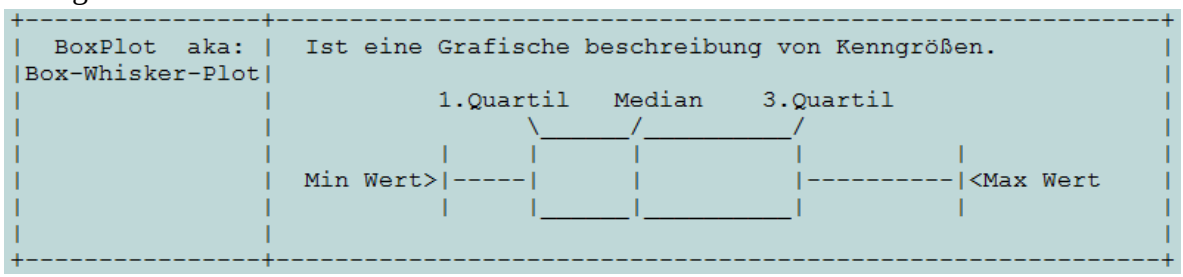
Benchmark Ergebnisse:

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



CPU:

Median-Wert

```
(%i4) median(liste);
(%o4) 1959.478
```

Quartil

```
(%i5) ''quantile(liste,1/4);
(%o5) 1938.45925
```

Kleinster Wert

```
(%i6) ''smin(liste);
(%o6) 1907.612
```

Groester Wert

```
(%i7) ''smax(liste);
(%o7) 3106.338
```

Spannweite

```
(%i8) ''range(liste);
(%o8) 1198.726
```

GPU:

Median-Wert

```
(%i9) ''median(GPUliste);
(%o9) 596.0355
```

Quartil

```
(%i10) ''quantile(GPUliste,1/4);
(%o10) 594.74
```

Kleinster Wert

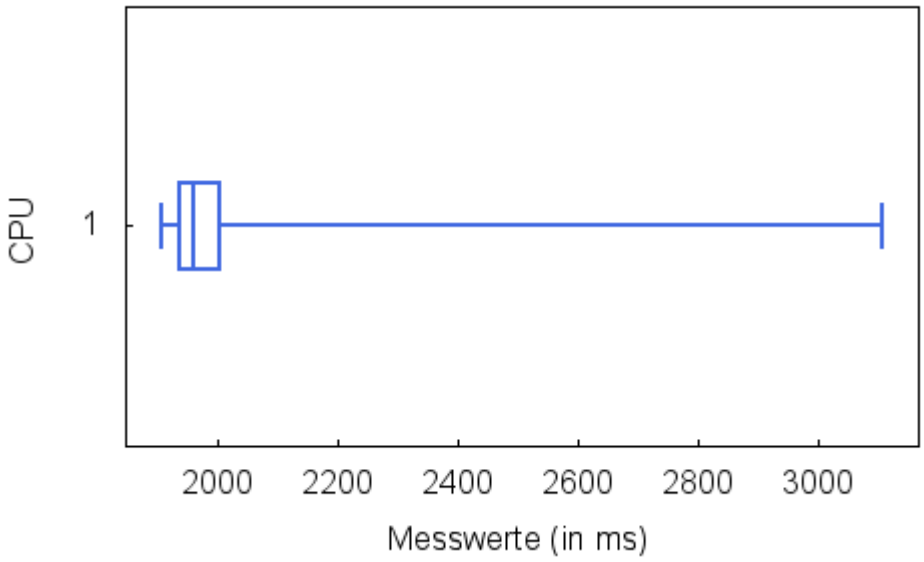
```
(%i11) ''smin(GPUliste);
(%o11) 594.158
```

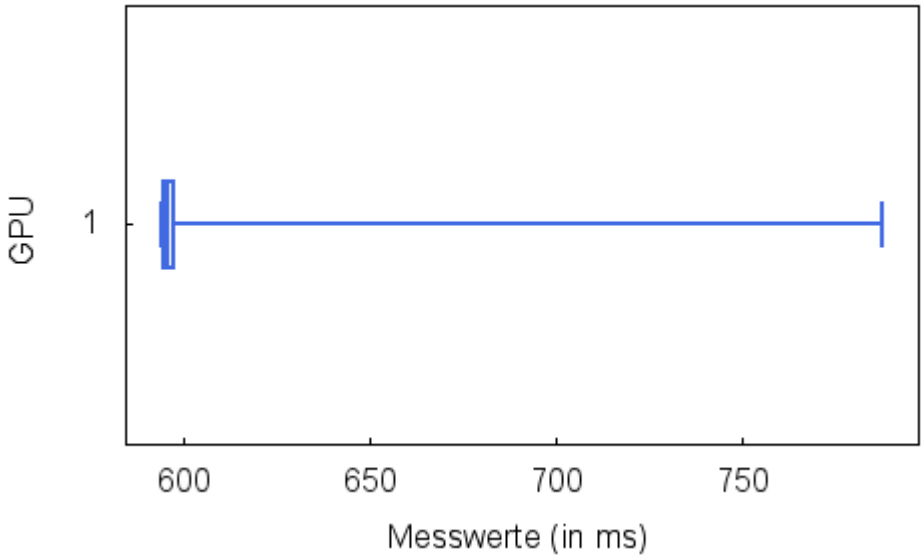
Groester Wert

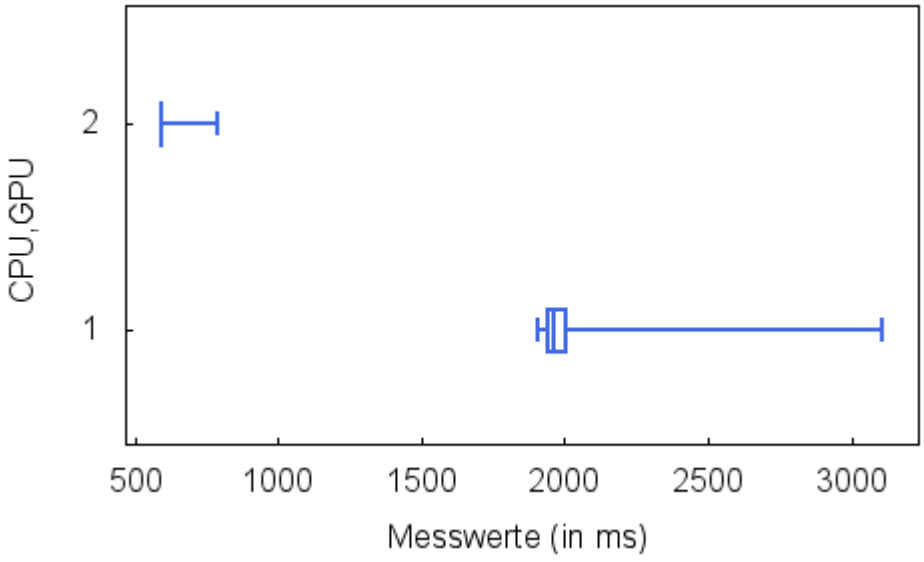
```
(%i12) ''smax(GPUliste);
(%o12) 787.839
```

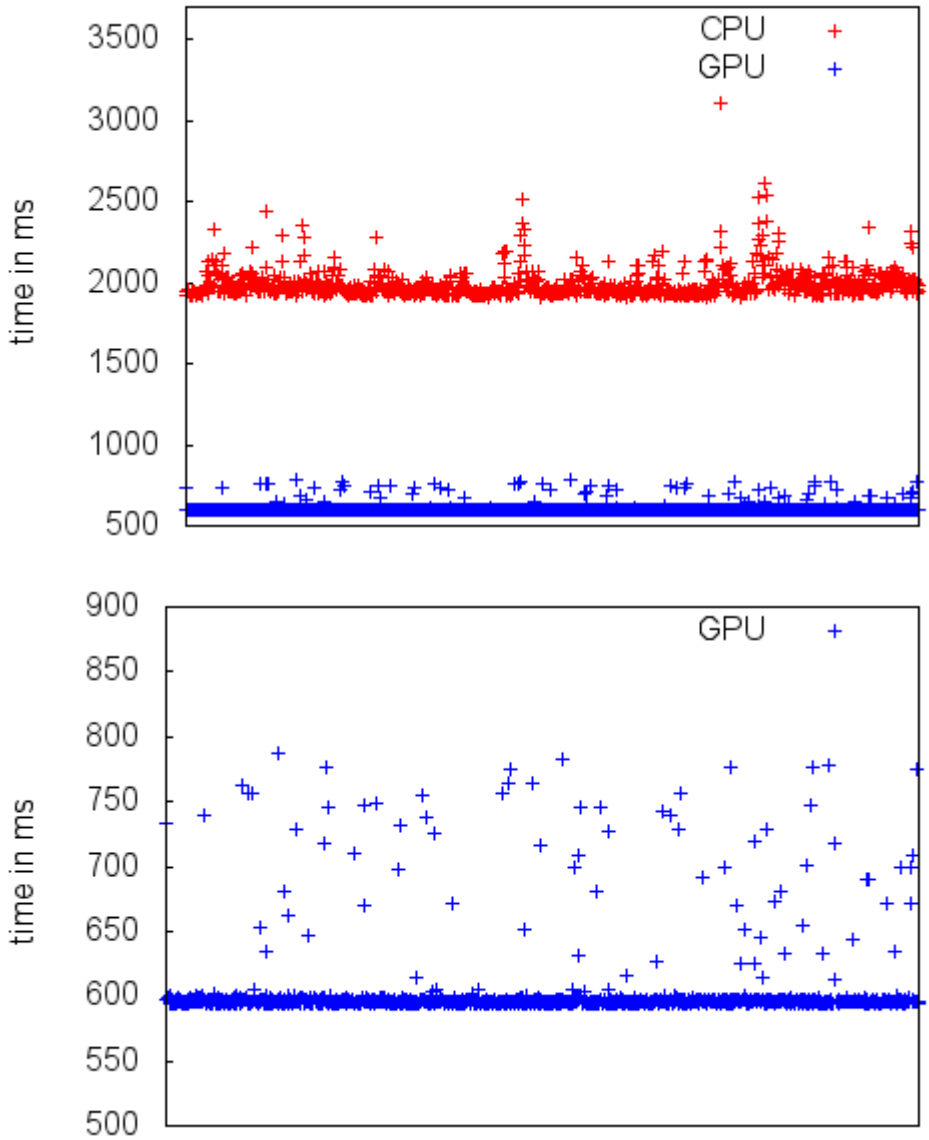
Spannweite

```
(%i13) ''range(GPUliste);
(%o13) 193.681
```


CPU
1
2000
2200
2400
2600
2800
3000
Messwerte (in ms)


GPU
1
600
650
700
750
Messwerte (in ms)


CPU,GPU
2
1
500
1000
1500
2000
2500
3000
Messwerte (in ms)


CPU
GPU
time in ms
3500
3000
2500
2000
1500
1000
500
GPU
time in ms
900
850
800
750
700
650
600
550
500

Windows 8.1 pro

Platform #0:NVIDIA CUDA
Device #0(GPU):GeForce GTX 560 Ti (SLI)
Compute Units: 8 @ 1760 mghtz
Local memory: 48 KB
Global memory: 1 GB

Device #1(GPU):GeForce GTX 560 Ti (SLI)
Compute Units: 8 @ 1760 mghtz
Local memory: 48 KB
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

CPU:

Median-Wert

```
(%i4) ''median(liste);
(%o4) 2654.296003
```

Quartil

```
(%i5) ''quantile(liste,1/4);
(%o5) 2573.1820245
```

Kleinster Wert

```
(%i6) ''smin(liste);
(%o6) 2444.288629
```

Groester Wert

```
(%i7) ''smax(liste);
(%o7) 7415.587546
```

Spannweite

```
(%i8) ''range(liste);
(%o8) 4971.298917
```

GPU:

Median-Wert

```
(%i9) ''median(GPUliste);
(%o9) 171.9933045
```

Quartil

```
(%i10) ''quantile(GPUliste,1/4);
(%o10) 171.61877275
```

Kleinster Wert

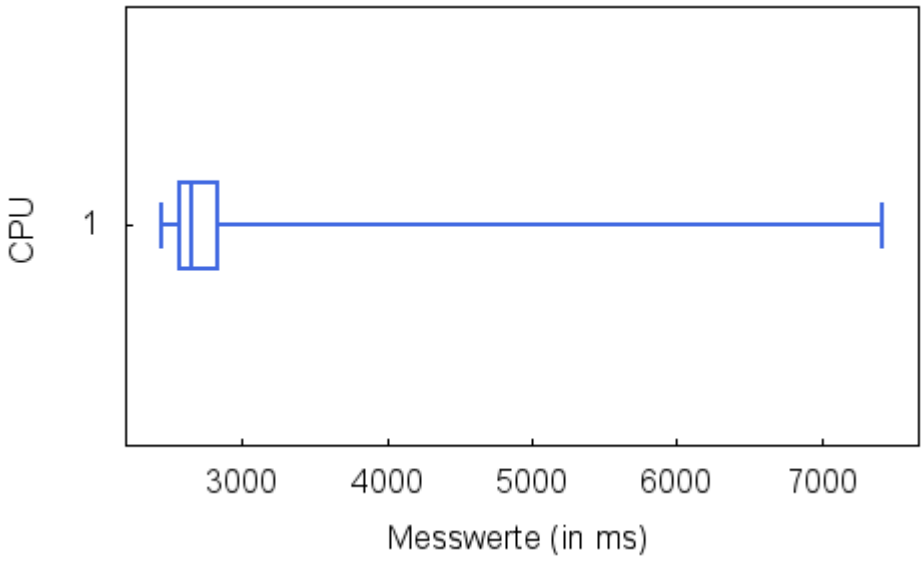
```
(%i11) ''smin(GPUliste);
(%o11) 169.559198
```

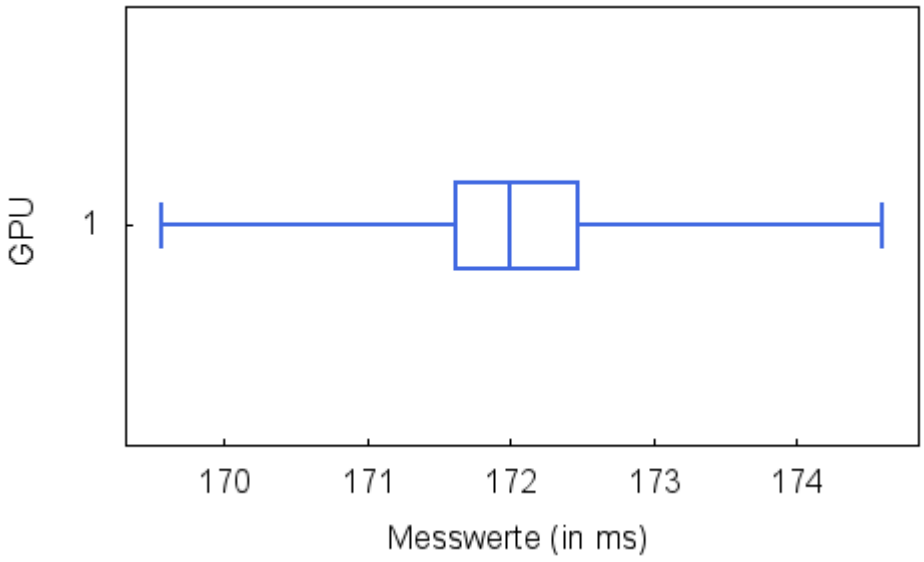
Groester Wert

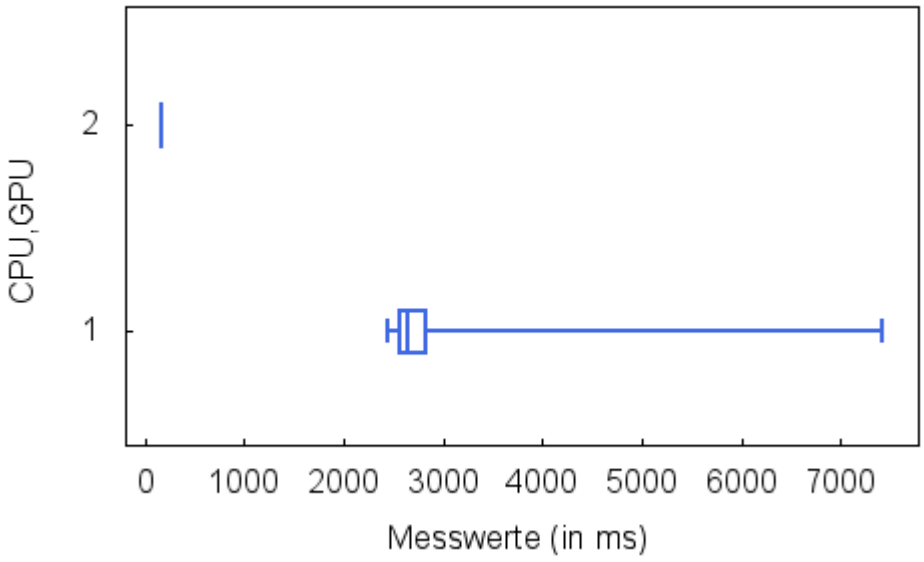
```
(%i12) ''smax(GPUliste);
(%o12) 174.599087
```

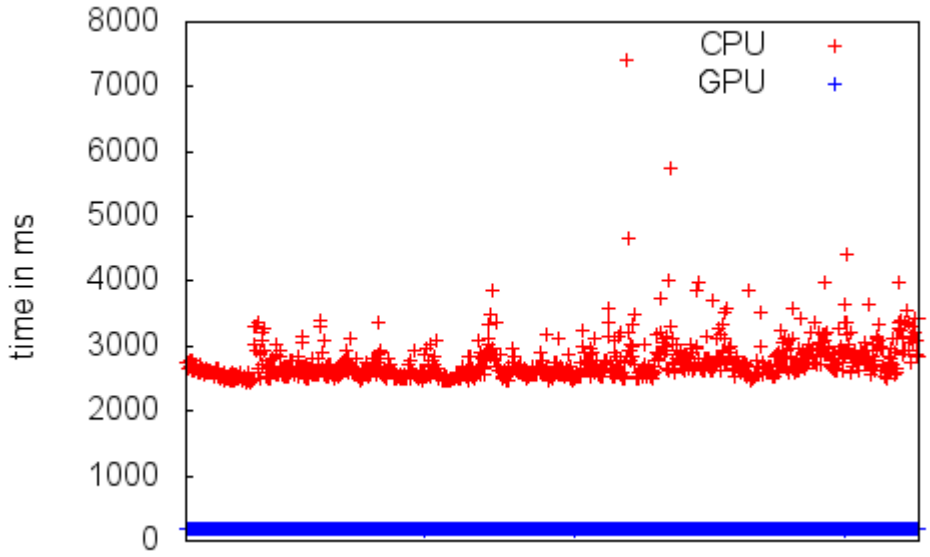
Spannweite

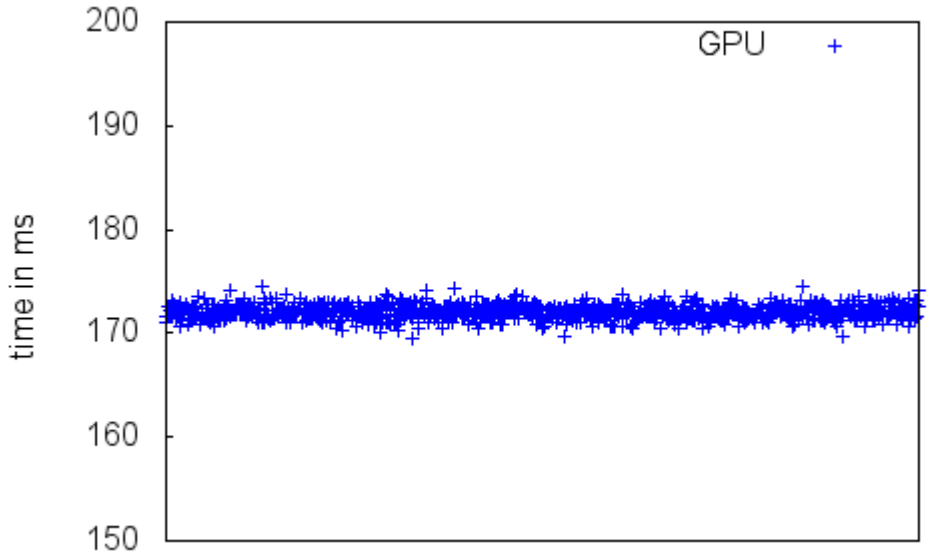
```
(%i13) ''range(GPUliste);
(%o13) 5.039888999999988
```


CPU
1
3000
4000
5000
6000
7000
Messwerte (in ms)


GPU
1
170
171
172
173
174
Messwerte (in ms)


CPU,GPU
2
1
0
1000
2000
3000
4000
5000
6000
7000
Messwerte (in ms)


8000
7000
6000
5000
4000
3000
2000
1000
0
time in ms
CPU
GPU


200
190
180
170
160
150
time in ms
GPU

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)
- Compute Units: 8 @ 1760 mghtz
- Local memory: 48 KB
- Global memory: 1 GB

Device #1(GPU):GeForce GTX 560 Ti (SLI)
- Compute Units: 8 @ 1760 mghtz
- Local memory: 48 KB
- 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

CPU:

Median-Wert

```
(%i4) ''median(liste);
(%o4) 0.280218
```

Quartil

```
(%i5) ''quantile(liste,1/4);
(%o5) 0.27648675
```

Kleinster Wert

```
(%i6) ''smin(liste);
(%o6) 0.271199
```

Groester Wert

```
(%i7) ''smax(liste);
(%o7) 1.268603
```

Spannweite

```
(%i8) ''range(liste);
(%o8) 0.997404
```

GPU:

Median-Wert

```
(%i9) ''median(GPUliste);
(%o9) 0.7251160000000001
```

Quartil

```
(%i10) ''quantile(GPUliste,1/4);
(%o10) 0.6821965
```

Kleinster Wert

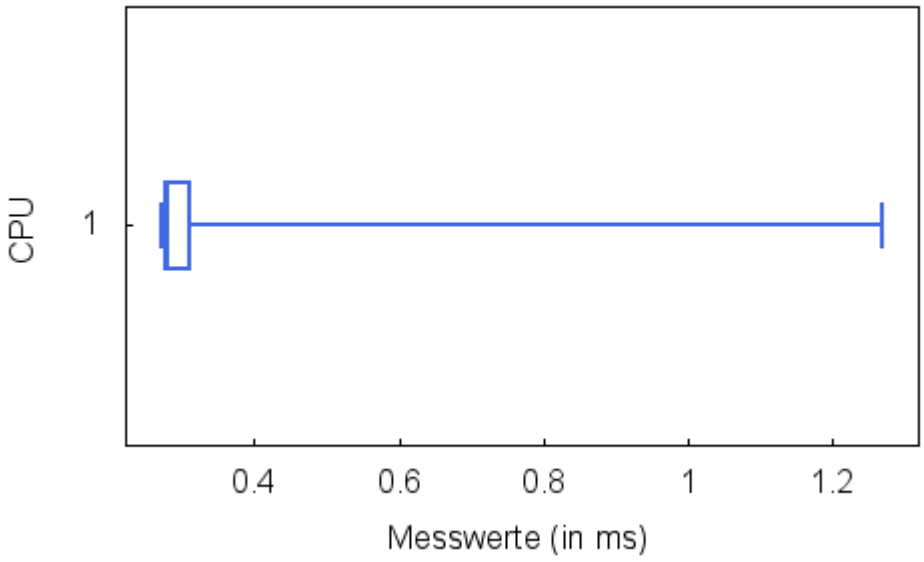
```
(%i11) ''smin(GPUliste);
(%o11) 0.662758
```

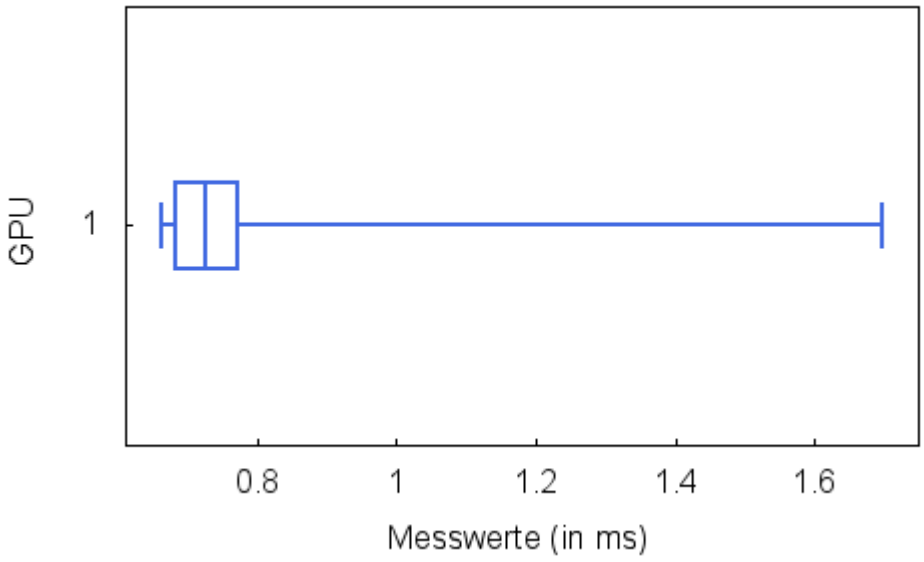
Groester Wert

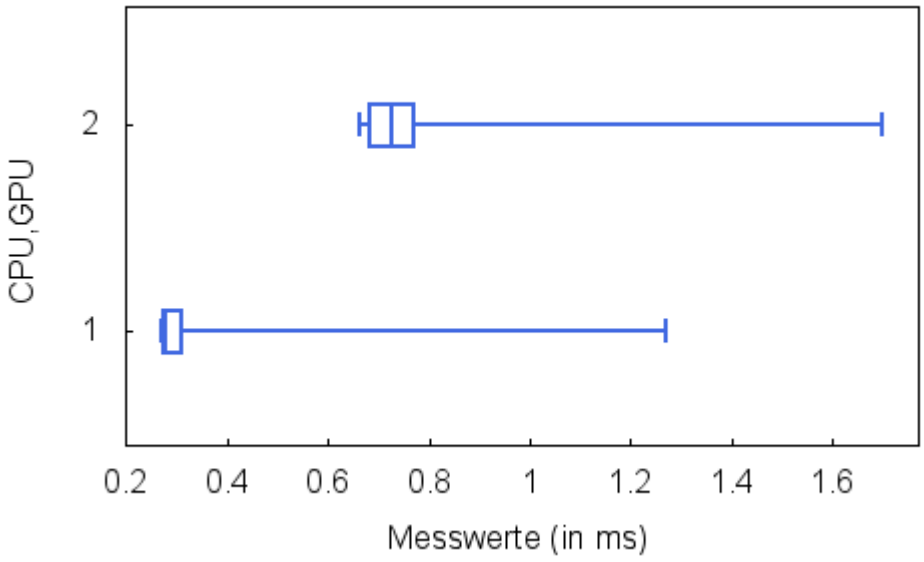
```
(%i12) ''smax(GPUliste);
(%o12) 1.697172
```

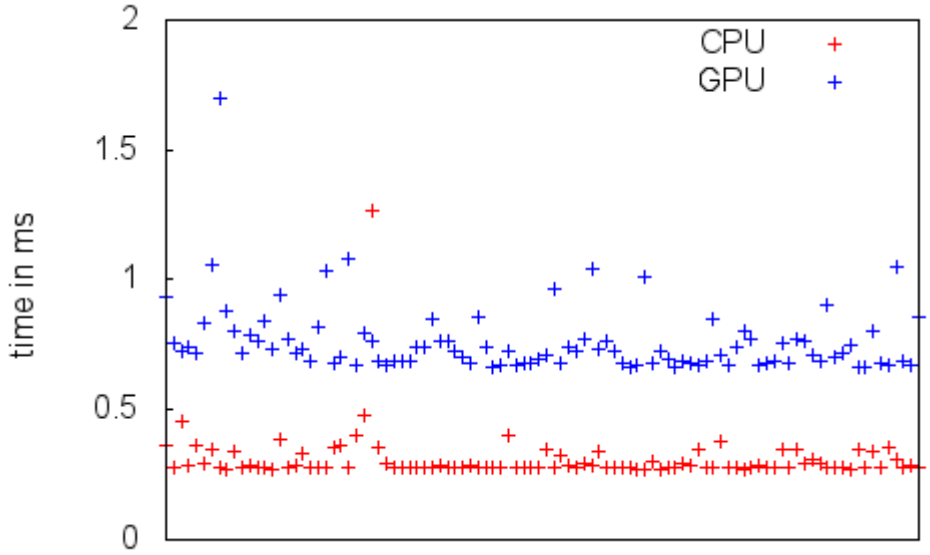
Spannweite

```
(%i13) ''range(GPUliste);
(%o13) 1.034414
```


CPU
1
0.4
0.6
0.8
1
1.2
Messwerte (in ms)


GPU
1
0.8
1
1.2
1.4
1.6
Messwerte (in ms)


CPU,GPU
2
1
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Messwerte (in ms)


CPU
GPU
time in ms
2
1.5
1
0.5
0

Starten des Programms:

Windows: gradlew run
Unix: ./gradle run

Starten von algorithmus 1: `"calc"` als Parameter
Straten von algorithmus 2: keine parameter