Evaluierung und Vorhersage von Ausführungszeiten für OpenCL-basierte Berechnungen auf GPGPU-Systemen

Abschlussvortrag

Alexander Pöppl

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Fakultät für Informatik
Technische Universität München

5. September 2014

Agenda

Motivation

Motivation

Motivation

- Bemerkenswerte Entwicklungen im Bereich der GPUs
- Vorteile von GPUs:
  - Hochgradig Parallel
  - Schnelle Speicherzugriffe
  - Spezialisiert auf Berechnungen auf großen Datensätzen

Motivation

- Bemerkenswerte Entwicklungen im Bereich der GPUs
- Vorteile von GPUs:
  - Hochgradig Parallel
  - Schnelle Speicherzugriffe
  - Spezialisiert auf Berechnungen auf großen Datensätzen
- *General Purpose GPU Programming*
- *Signifikante Performancegewinne möglich.*

Motivation

- Programmierung für die GPU komplexer als für die CPU
- Stark an die Hardwarestruktur angelehnt

Motivation

- Programmierung für die GPU komplexer als für die CPU
- Stark an die Hardwarestruktur angelehnt
- Berechnung auf der GPU nicht zwangsläufig schneller
- Overhead für Transfer von Daten zur GPU

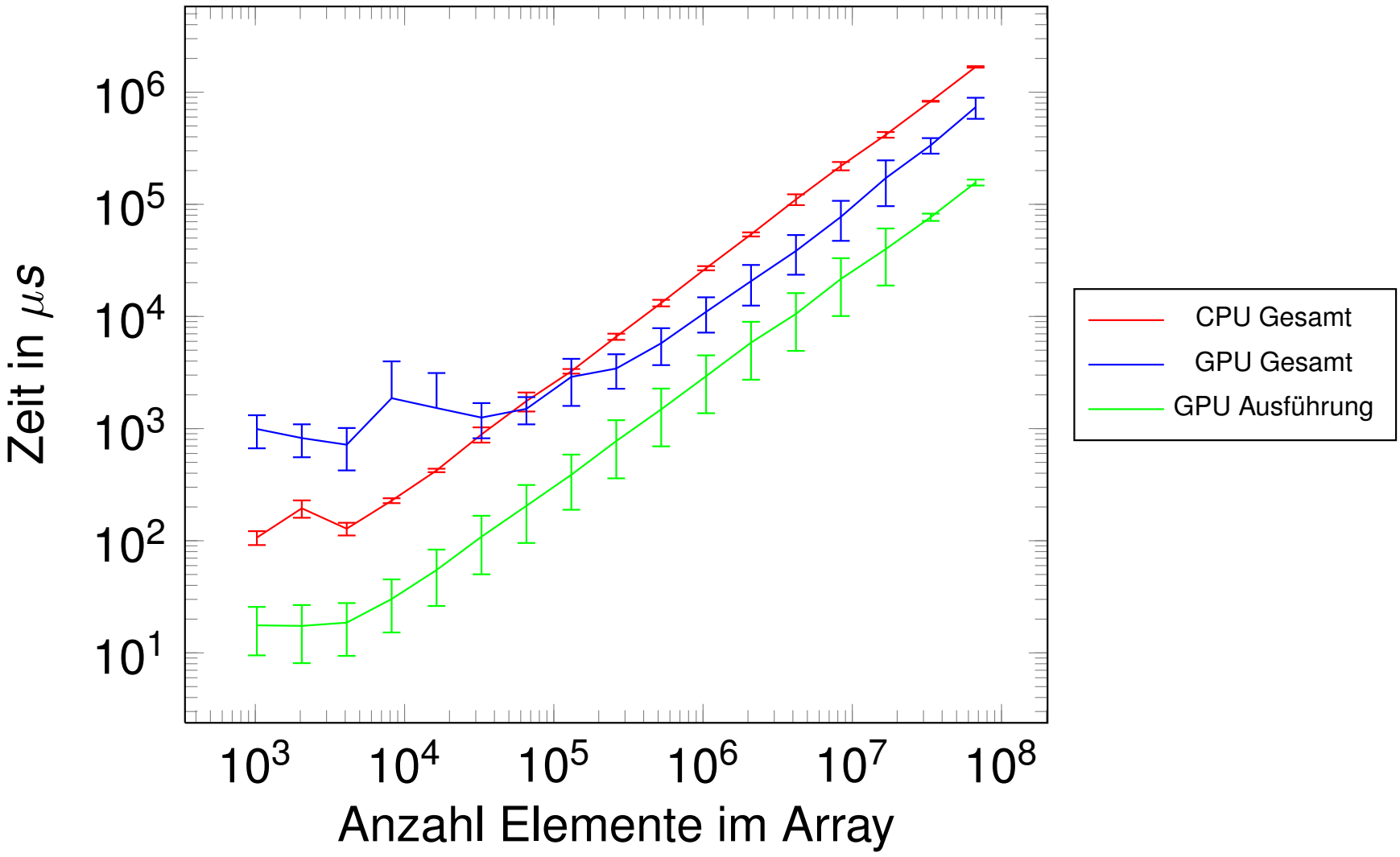
Motivation

- Programmierung für die GPU komplexer als für die CPU
- Stark an die Hardwarestruktur angelehnt
- Berechnung auf der GPU nicht zwangsläufig schneller
- Overhead für Transfer von Daten zur GPU
- Compiler soll entscheiden, wo Berechnung ausgeführt wird
- Entscheidung basierend auf erwarteten Ausführungszeiten der Berechnungen auf GPU und CPU

Motivation

- Programmierung für die GPU komplexer als für die CPU
- Stark an die Hardwarestruktur angelehnt
- Berechnung auf der GPU nicht zwangsläufig schneller
- Overhead für Transfer von Daten zur GPU
- Compiler soll entscheiden, wo Berechnung ausgeführt wird
- Entscheidung basierend auf erwarteten Ausführungszeiten der Berechnungen auf GPU und CPU
- *Modellierung der Ausführungszeiten auf der GPU*

Motivation


Zeit in μs
10^6
10^5
10^4
10^3
10^2
10^1
10^3
10^4
10^5
10^6
10^7
10^8
Anzahl Elemente im Array
CPU Gesamt
GPU Gesamt
GPU Ausführung

Agenda

Theorie

OpenCL

- Offener Standard zur Durchführung von parallelen Berechnungen auf heterogenen Systemen
- Unterstützt GPUs, CPUs, Accelerators
- Implementierungen u.a. von Intel, AMD, NVidia, Apple
- Unterstützt Taskparallelität, Datenparallelität

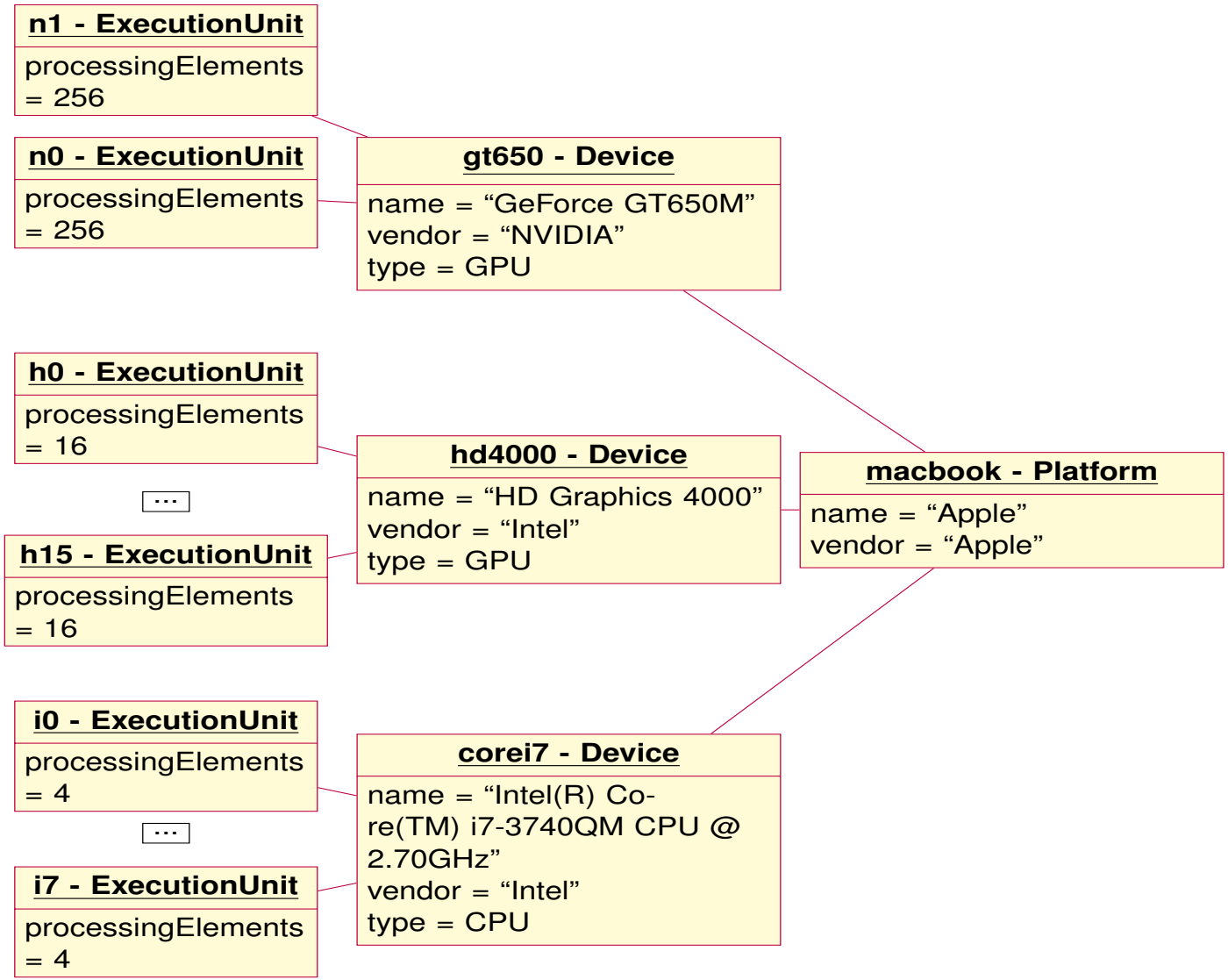
Theorie

OpenCL

- **Platform:** “Konventionelle” Ausführungsumgebung. Kann Devices nutzen, um Berechnungen durchzuführen.
- **Device:** Gerät, auf dem OpenCL-Code ausgeführt wird
- **Kernel:** Die Funktion, die auf der GPU ausgeführt wird. Muss kompiliert werden.
- **Memory:** Die Daten, die der Kernel nutzt. Müssen transferiert werden.

Theorie

OpenCL-Plattformmodell



Theorie

OpenCL-Ausführungsmodell

Begriffe

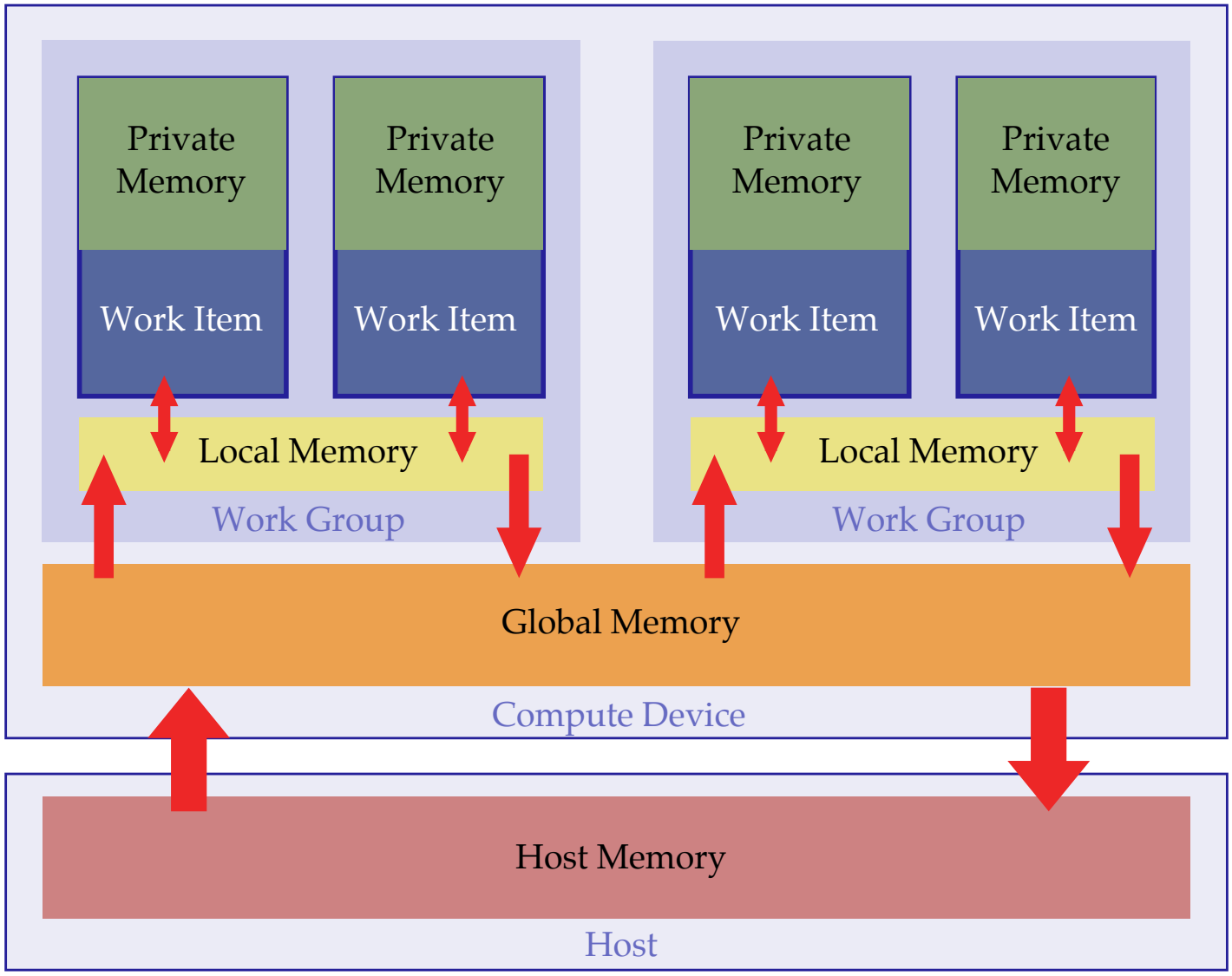
- **Work-Item:** Einzelner Thread, ein Durchlauf des Kernels
- **Work-Group:** Gruppe von Work-Items, die nebenläufig ausgeführt wird

Einschränkungen

- Es werden immer nur Work-Items einer Work-Group gleichzeitig ausgeführt
- Vor Beginn der Ausführung einer neuen Work-Group muss die vorhergehende abgeschlossen sein
- $n_{Work-Items} \bmod s_{Work-Group} = 0$

Theorie

OpenCL-Speichermodell



Theorie

funkyIMP

- Sprachfeature von Interesse: *Domain Iterations*

Theorie

funkyIMP

- Sprachfeature von Interesse: *Domain Iterations*
- Iteration über Mehrdimensionale Arrays

```
domain one_d{x} = {(a) | a < x}
domain two_d{x, y} : one_d{x*y} (o) = {(a,b) | a<x & b<y}

float M[two_d{3,4}] = new float[two_d{3,4}];
//... Init values ...
//init code

M'v2 = M.\(x,y) { M[x,y] * 2 }
```

Agenda

Erweiterung des funkyIMP Compilers

- Generierung des Codes aufseiten des Hosts
  - Sammeln der Variablen
  - Code zum Kompilieren des Kernels
  - Transfer der Daten zur GPU
  - Ausführung des Kernels
  - Rücktransfer des Ergebnisses

Erweiterung des funkyIMP Compilers

- Generierung des Codes aufseiten des Hosts
  - Sammeln der Variablen
  - Code zum Kompilieren des Kernels
  - Transfer der Daten zur GPU
  - Ausführung des Kernels
  - Rücktransfer des Ergebnisses
- Übersetzung der Domain Iteration in einen OpenCL-Kernel
  - Kernel Header
  - Expression innerhalb der Domain Iteration

Erweiterung des funkyIMP Compilers

Beispiel

```
return ma.\(a,b,c) {ma[c,a,b] + g(a)};
```

Erweiterung des funkyIMP Compilers

Beispiel - Code auf Hostseite

```
return [&]() ->funky::LinearArray< int >::Version* {
  if (1) {
    return
      ([&]()->funky::LinearArray< int >::Version*
      {
        funky::LinearArray< int >::Version*__VAL_TMP0=ma;
        funky::LinearArray< int >::Version*__EXP_TMP0
          = __VAL_TMP0->getNewVersion();

        // Compile the kernel
        ocl_kernel f_0(&device,"tmp/f_0.cl");
```

Erweiterung des funkyIMP Compilers

Beispiel - Code auf Hostseite (Fortsetzung)

```
// Translate ma into a format
// that is understandable by OpenCL
int *__ma_0=ma->toNative();
ocl_mem __ma_GPU_0
  = device.malloc(sizeof(int)*16777216,
                  CL_MEM_READ_ONLY);
int __ma0_dim_0=256;
int __ma0_dim_1=256;
int __ma0_dim_2=256;

// Create the return value
int *__return_val_0=new int[16777216];
ocl_mem __return_val_GPU_0
  = device.malloc(sizeof(int)*16777216,
                  CL_MEM_WRITE_ONLY);
// Copy ma to the GPU
__ma_GPU_0.copyFrom(__ma_0);
```

Erweiterung des funkyIMP Compilers

Beispiel - Code auf Hostseite (Fortsetzung)

```
// Set the Kernel Arguments
f_0.setArgs(__ma_GPU_0.mem(),&__ma0_dim_0,
            &__ma0_dim_1,&__ma0_dim_2,
            __return_val_GPU_0.mem());

//Run the kernel and wait for it to be done.
int id = f_0.timedRun(1024, 16777216);
device.finish();
```

Erweiterung des funkyIMP Compilers

Beispiel - Code auf Hostseite (Fortsetzung)

```
    // Translate the return value back to the
    // funkyIMP format.
    __return_val_GPU_0.copyTo(__return_val_0);
    funky::LinearArray<int> *__return_LINARR0
      = new funky::LinearArray<int>(16777216,
                                    __return_val_0);
    funky::LinearArray< int >::Version* __return_0
      = new funky::LinearArray<int>
              ::Version(__return_LINARR0,
                3,256,256,256);
        return __return_0;
    })();
} else {
    // Generated CPU iteration
}
```

Erweiterung des funkyIMP Compilers

Beispiel - Kernel Code

```
//FUNCTION HEADER DECLS
int   __s_int_g_int(int   __x);

//FUNCTION BODY DECLS
int   __s_int_g_int(int   __x)
{
    return 2*(__x);
}
```

Erweiterung des funkyIMP Compilers

Beispiel - Kernel Code (Fortsetzung)

```
//KERNEL CODE
__kernel void f_0(__global int *__ma_0, int __ma0_dim_0,
        int __ma0_dim_1, int __ma0_dim_2,
        __global int *__return_0)
{
    size_t __CUR_POS_0=get_global_id(0);
    int __a_0=(__CUR_POS_0/65536)%256;
    int __b_0=(__CUR_POS_0/256)%256;
    int __c_0=(__CUR_POS_0/1)%256;
    __return_0[__CUR_POS_0]=((__ma_0)[(__b_0) + (__a_0)
        * 256L + (__c_0) * 65536L])
        + (__s_int_g_int(__a_0));
}
```

Agenda

Benchmark Suite

- Suite zur Durchführung von Benchmarks zur experimentellen Bestimmung der Ausführungszeiten
- Führt *n* Benchmarks auf einem Device aus, für verschiedene Werte (z.B Größe der Work-Group, Größe des Speichers, Anzahl Operationen)
- Mehrmalige Durchführung, solange bis Standardfehler kleiner als angegeben

Agenda

Ausführungszeitmodell

Datentransfer

- GPU typischerweise über PCI Express angebunden
- Einfluss von Bandbreite von PCI Express BUS
- Einfluss von Latenz von PCI Express BUS, Speicheranbindung, Speicher

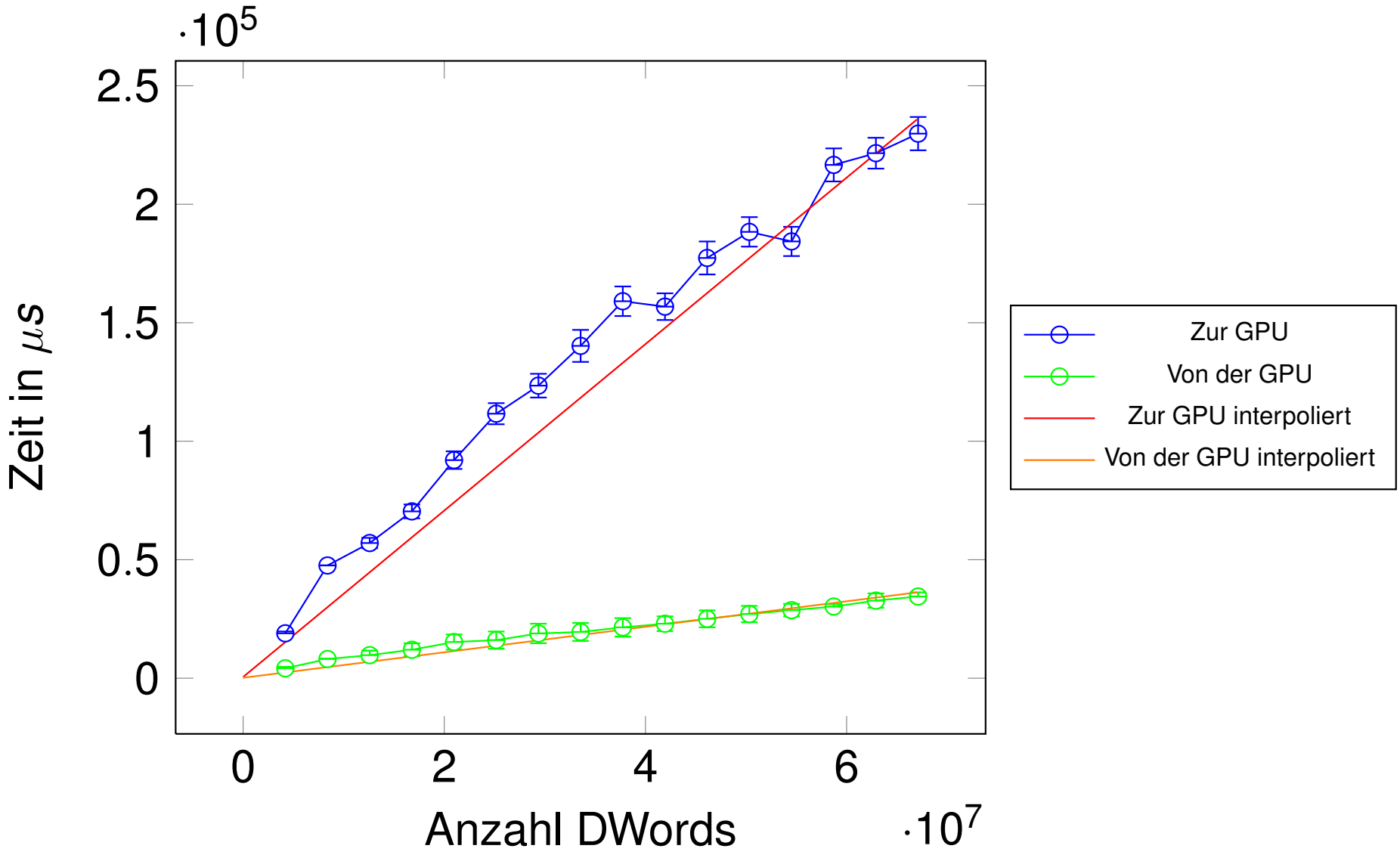
Ausführungszeitmodell

Datentransfer - Experiment

- Transfer von sukzessiv größer werdenden Speicherblöcken
- Messung der Zeit für Transfer zu und von der GPU
- $128kB \rightarrow 16MB$, in Schritten zu $128kB$

Ausführungszeitmodell

Datentransfer - Experiment



Ausführungszeitmodell

Datentransfer

Ergebnis

- Linear abhängig von der Speichergröße
- $T_{trans}(x) = b^{-1} * x + l_{prop}$

MacBook Pro

$$T_{\rightarrow GPU_{rMBP}} = 0.0035800\mu s * x + 516.41\mu s$$
$$T_{\leftarrow GPU_{rMBP}} = 0.00053337\mu s * x + 173,63\mu s$$

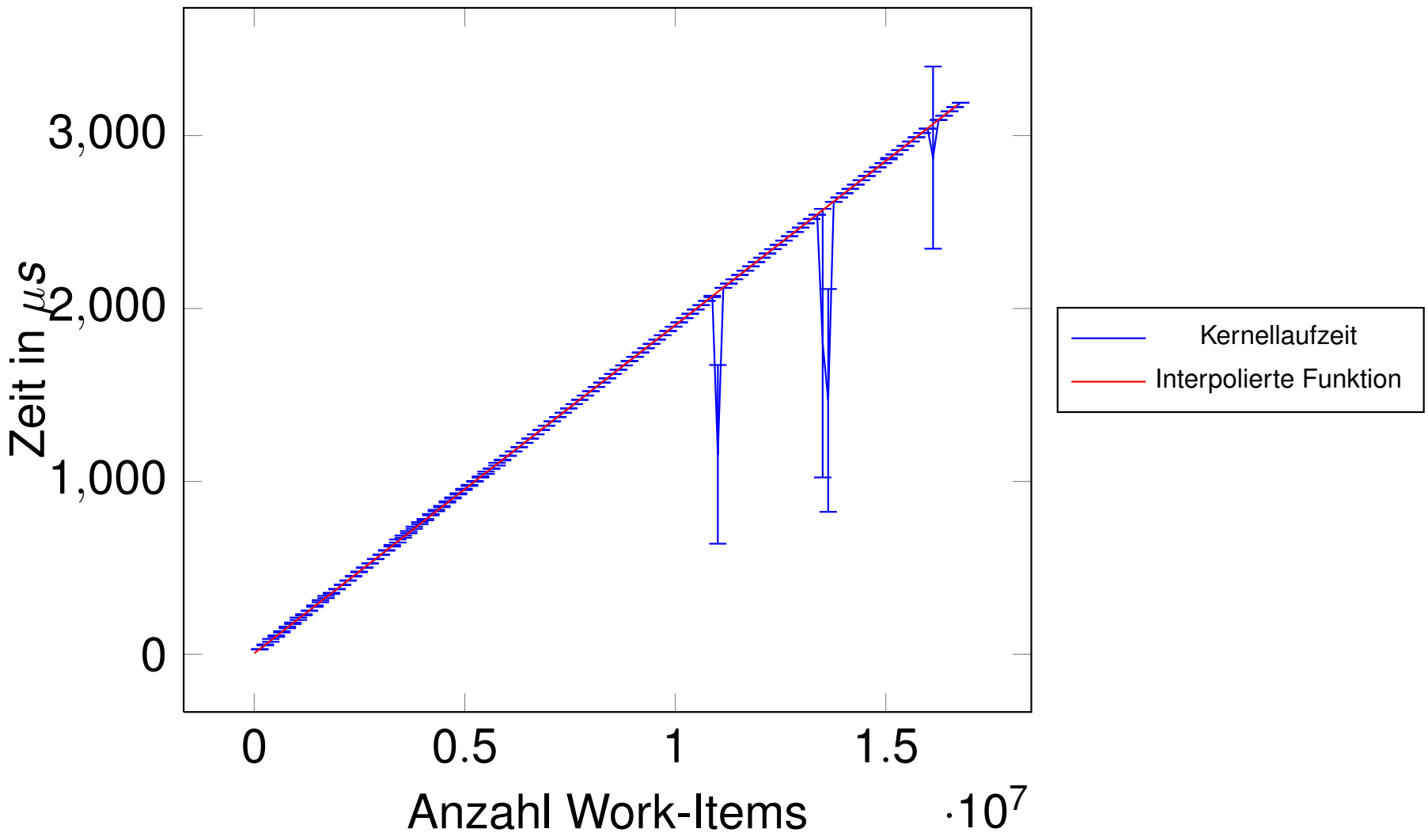
Ausführungszeitmodell

Leere Kernel

- Basiskosten für Kernelausführungen
- Ausführung von leeren Kernel auf sukzessiv wachsenden Speichersegmenten
- Messung der Ausführungszeiten
- $128kB \rightarrow 16MB$, in Schritten zu $128kB$

Ausführungszeitmodell

Leere Kernel - Experiment



Ausführungszeitmodell

Leere Kernel

Ergebnis

- Linear abhängig von der Anzahl Work-Items
- $T_{Base}(x) = t_{Element} * x + c$

MacBook Pro

$$T_{Base_{rMBP}} = 0.18989ns * x + 5.7265\mu s$$

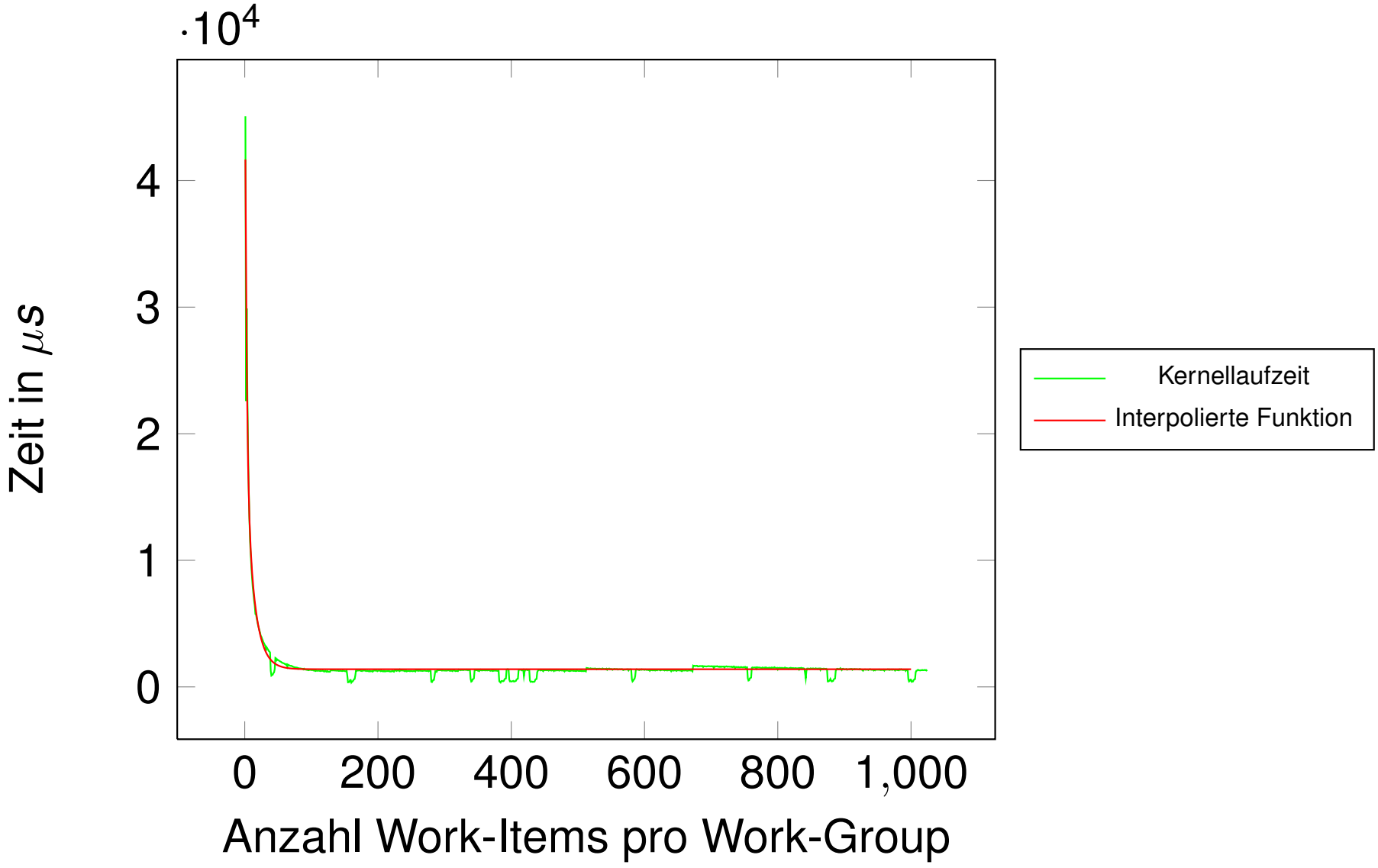
Ausführungszeitmodell

Größe der Work-Group

- Langsamere Ausführungszeit für manche Kernel beobachtet
- Experiment: Ausführung eines Kernels auf einer fixen Anzahl Elemente mit sukzessiv wachsender Größe der Work-Group
- Messung der Ausführungszeiten
- $1 \rightarrow s_{Work-Group_{max}}$

Ausführungszeitmodell

Größe der Work-Group - Experiment



Ausführungszeitmodell

Größe der Work-Group

Ergebnis

- Invers exponentiell abhängig von der Größe der Work-Group
- $M_{WG}(x) = \underbrace{B_1 * e^{\frac{-x}{t_1}}}_{\text{A}} + \underbrace{B_2 * e^{\frac{-x}{t_2}}}_{\text{B}}$

MacBook Pro

$$M_{WG_{rMBP}}(x) = 13.04 * e^{\frac{-x}{12.21124}} + 26.10 * e^{\frac{-x}{2.28329}}$$

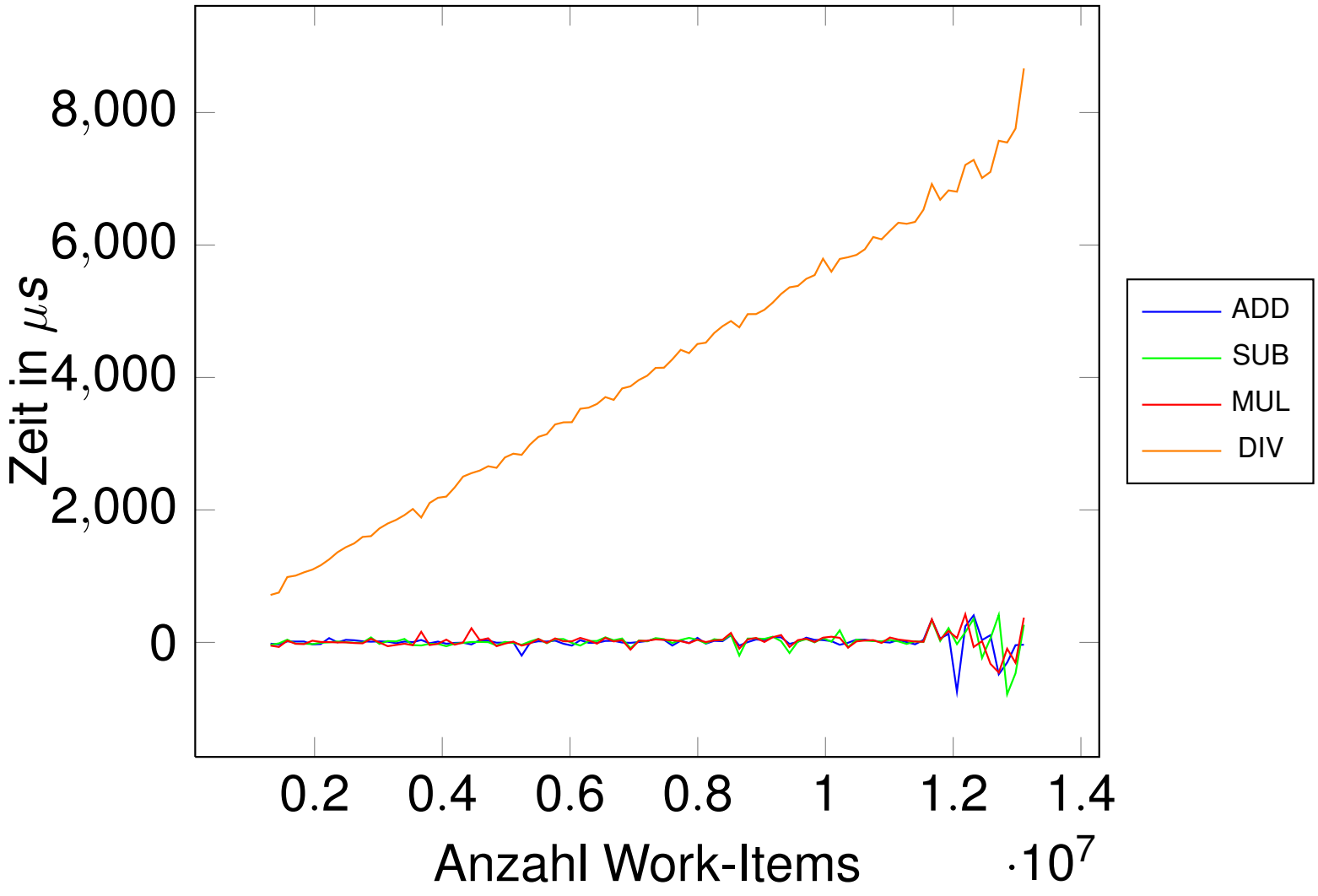
Ausführungszeitmodell

Rechenoperationen

- Betrachtet werden die Operatoren $+$, $-$, $*$, $/$
- Auf den Typen `int` und `float`
- Eine Operation pro Kernel, sukzessiv steigende Anzahl Elemente
- $128kB \rightarrow 16MB$, in Schritten zu $128kB$

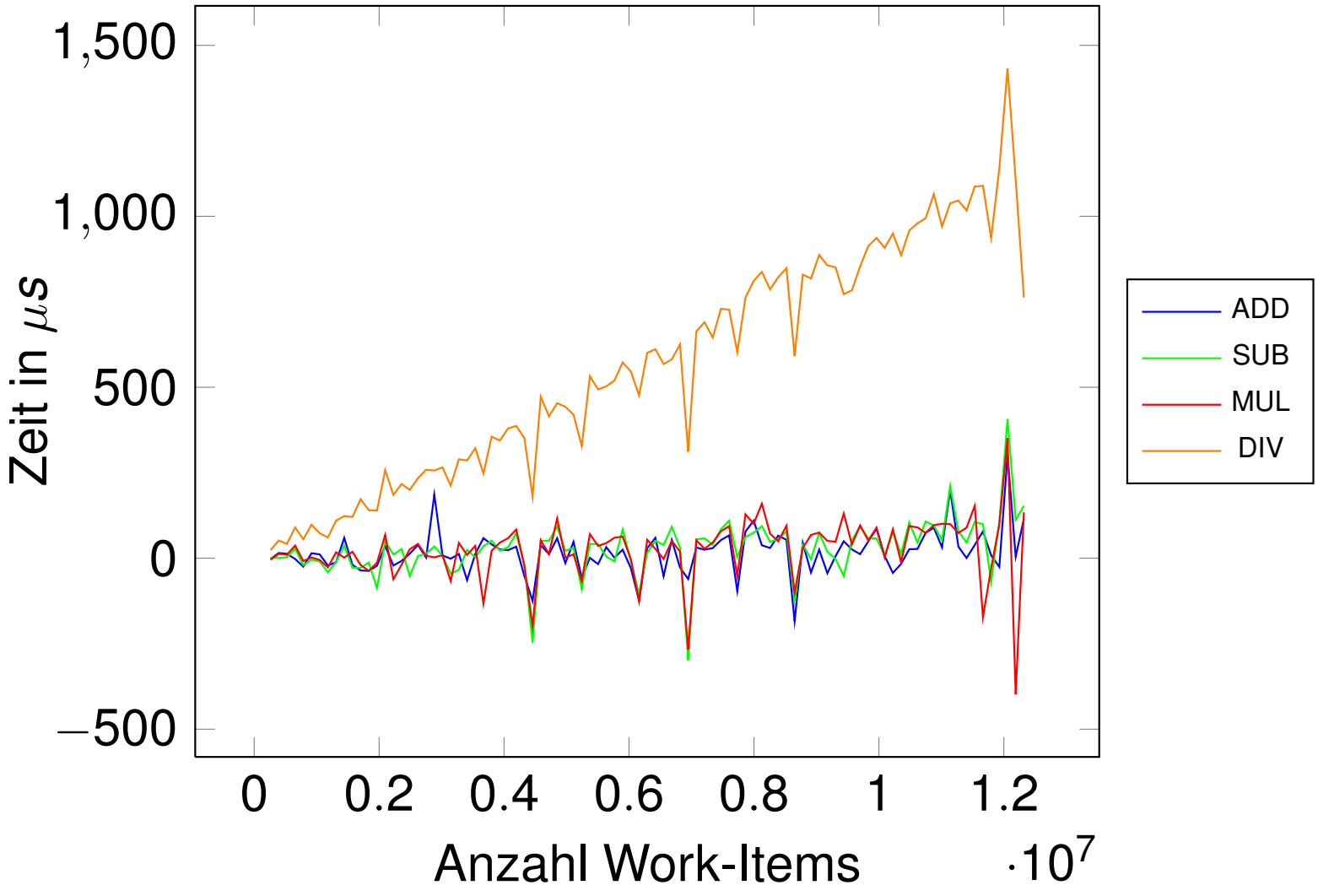
Ausführungszeitmodell

Rechenoperationen - Experiment Fließpunktarithmetik



Ausführungszeitmodell

Rechenoperationen - Experiment Ganzzahlarithmetik



Ausführungszeitmodell

Rechenoperationen

Ergebnis

- Linear abhängig von der Anzahl Work-Items
- $T^{Op}_{type}(x) = t_{Element} * x + c$
- Division in beiden Fällen teuerste Operation

MacBook Pro

$$T^{+}_{float}(x) = 3.2679ps * x$$
$$T^{-}_{float}(x) = 4.4204ps * x$$
$$T^{*}_{float}(x) = 4.4434ps * x$$
$$T^{/}_{float}(x) = 0.00058651\mu s * x + 2.2847\mu s$$

Ausführungszeitmodell

Rechenoperationen

Ergebnis

- Linear abhängig von der Anzahl Work-Items
- $T^{Op}_{type}(x) = t_{Element} * x + c$
- Division in beiden Fällen teuerste Operation

MacBook Pro

$$T^{+}_{int}(x) = 3.2859ps * x$$
$$T^{-}_{int}(x) = 6.1655ps * x$$
$$T^{*}_{int}(x) = 4.8305ps * x$$
$$T^{/}_{int}(x) = 8.9781 * 10^{-2}ns * x + 3.0253\mu s$$

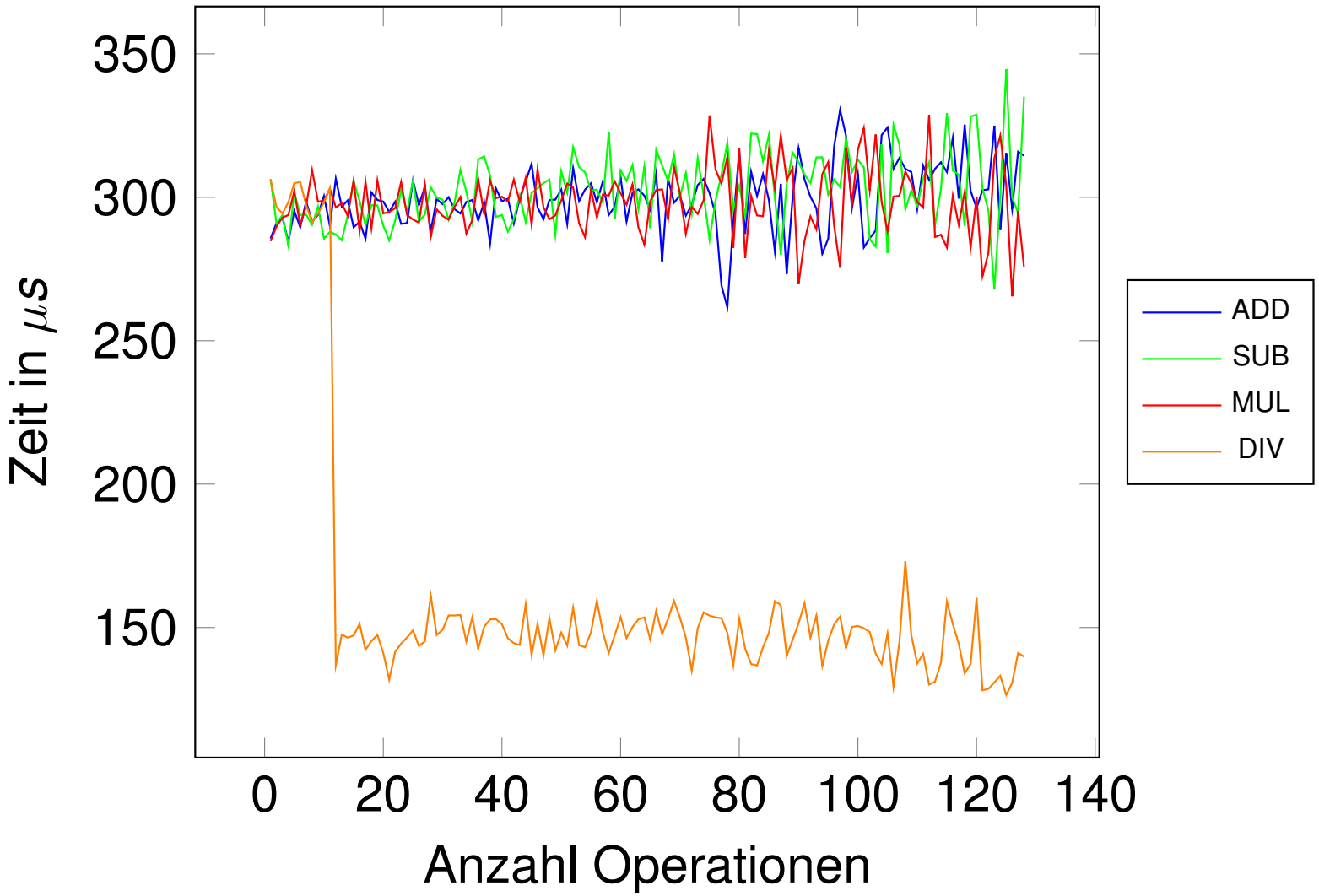
Ausführungszeitmodell

Rechenoperationen - Mehrere Operationen pro Kernel

- Betrachtet werden die Operatoren $+$, $-$, $*$, $/$
- Auf den Typen `int` und `float`
- Sukzessiv steigende Anzahl Operationen in Kernel, konstante Anzahl Work-Items
- $1 \rightarrow 128$ Elemente

Ausführungszeitmodell

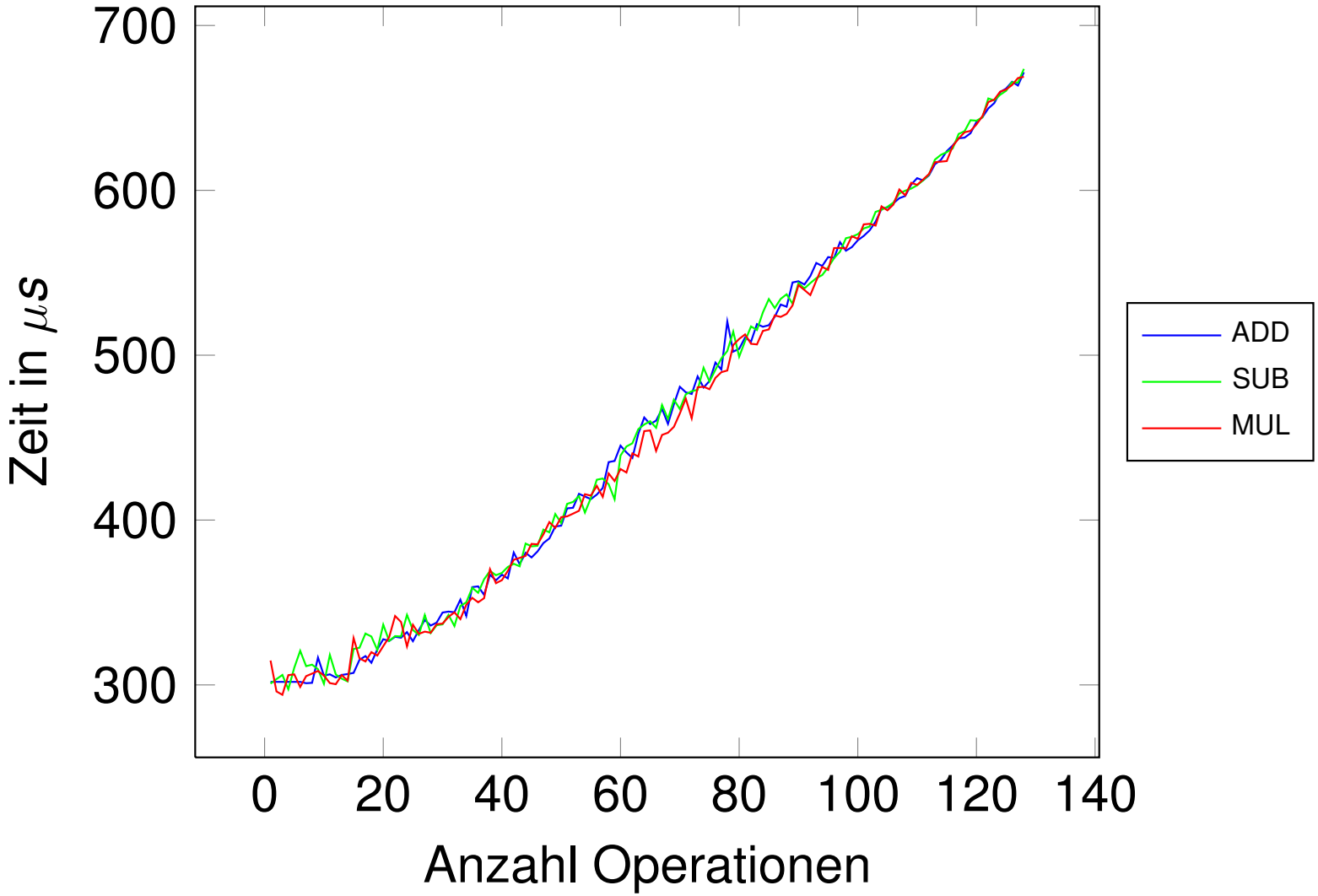
Rechenoperationen - Mehrere Operationen pro Kernel - Experiment Ganzzahlarithmetik



Ausführungszeitmodell

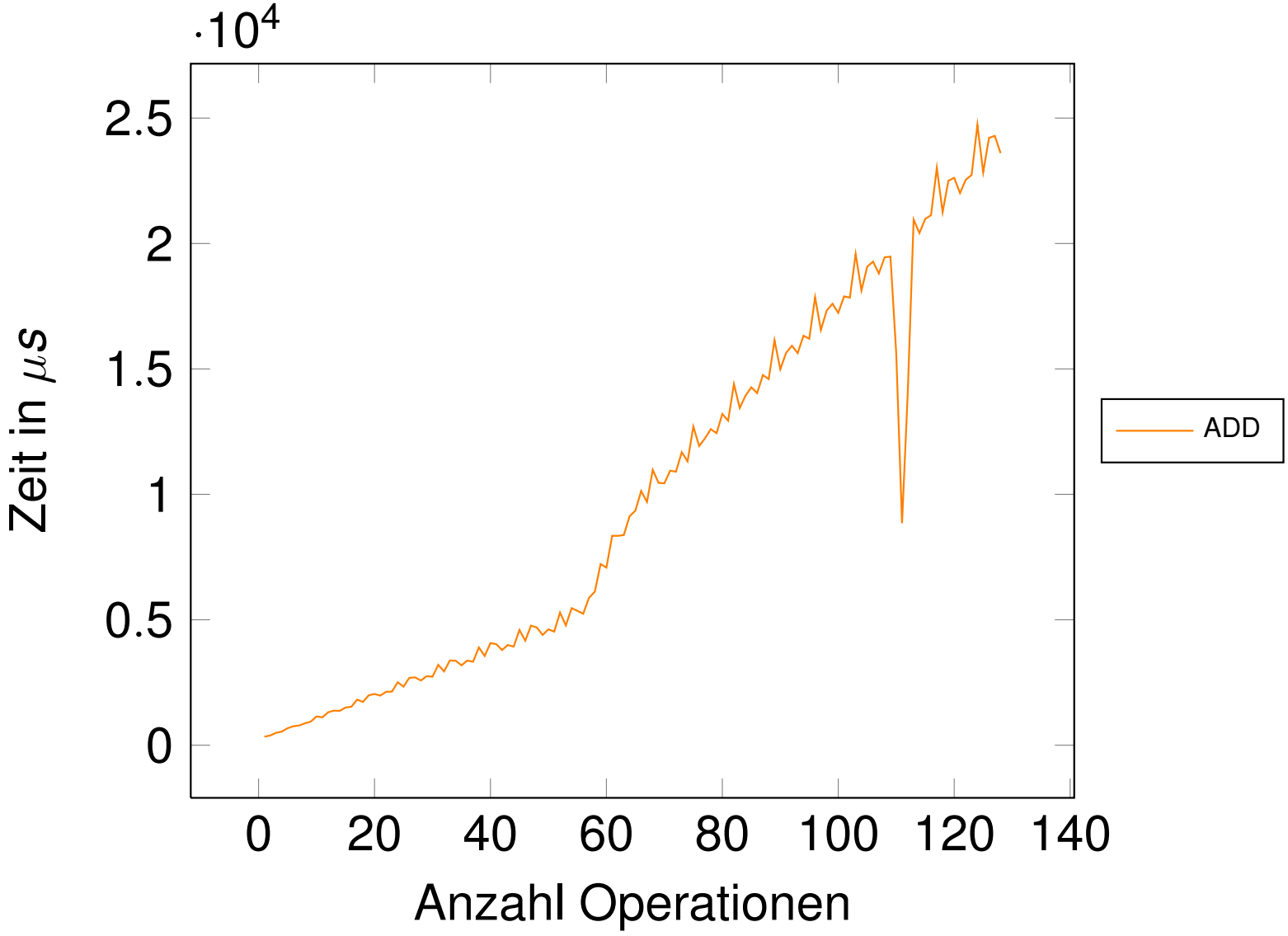
Rechenoperationen - Mehrere Operationen pro Kernel - Experiment

Fließpunktarithmetik



Ausführungszeitmodell

Rechenoperationen - Mehrere Operationen pro Kernel - Experiment Fließpunktarithmetik



Ausführungszeitmodell

Rechenoperationen - Mehrere Operationen pro Kernel

- Bei Ganzzahloperationen Anzahl Operationen nicht relevant
- Bei Fließpunktoperationen $+$, $-$ und $*$ gewisse Anzahl ohne Einfluss, danach lineare Abhängigkeit

$$M(x) = \begin{cases} \frac{1}{x} & (x \leq x_{sat}) \\ \frac{x_{sat}-x}{x} & (x < x_{sat}) \end{cases}$$

- Fließpunktoperation $/$ zeigt irreguläres Verhalten

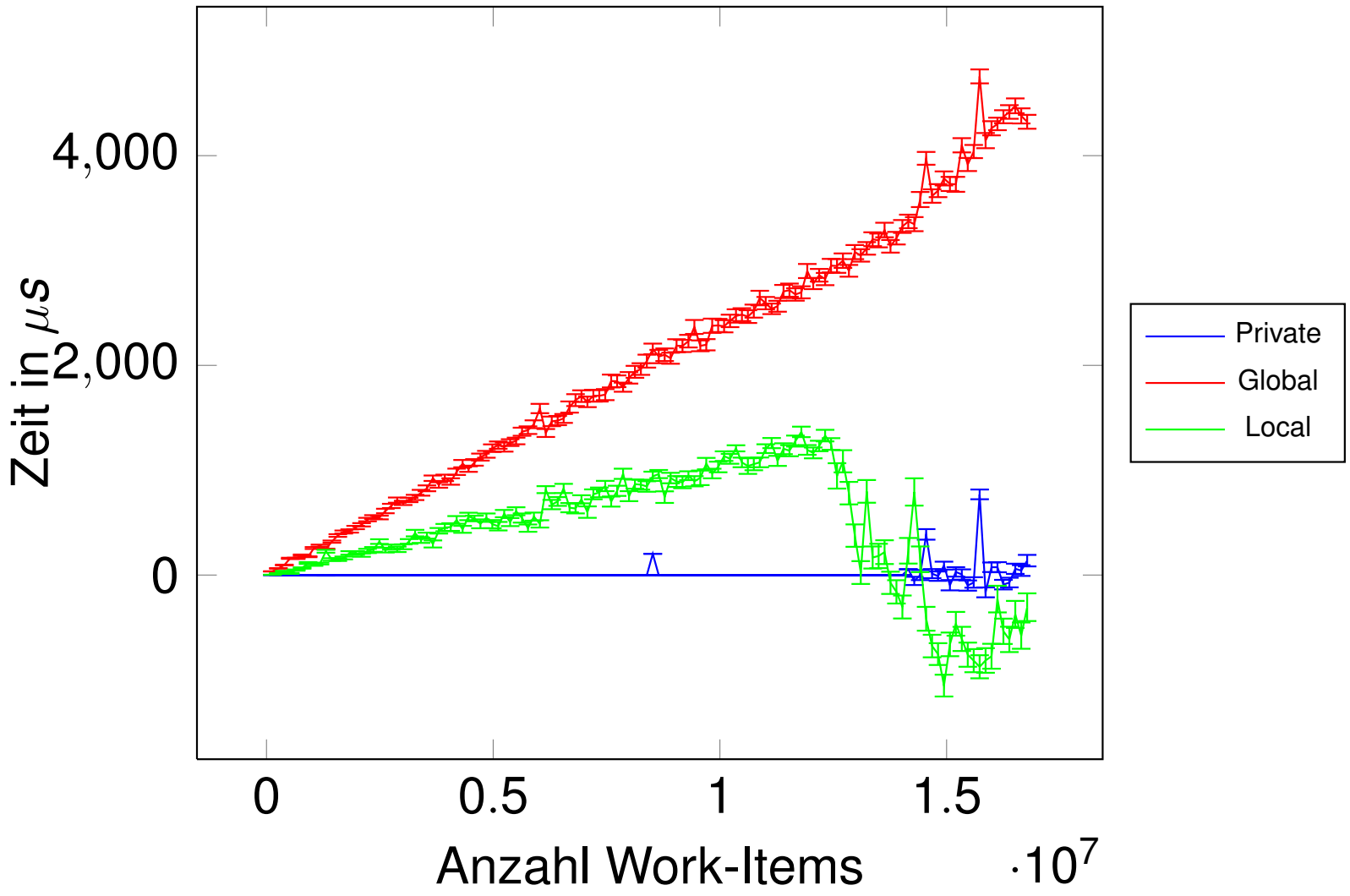
Ausführungszeitmodell

Speicherzugriffe

- 3 Arten Speicherzugriffe: `global`, `local` und `private`
- Eine Operation pro Kernel, sukzessiv steigende Anzahl Elemente
- $128kB \rightarrow 16MB$, in Schritten zu $128kB$

Ausführungszeitmodell

Speicherzugriffe - Experiment



Ausführungszeitmodell

Speicherzugriffe

Ergebnis

- Linear abhängig von der Anzahl Work-Items
- $T_{access}(x) = t_{Element} * x$
- Kosten für `private` vernachlässigbar

MacBook Pro

$$T_{private}(x) = 0s$$
$$T_{global}(x) = 0.247444074967131ns * x$$
$$T_{local}(x) = 0.1038783725ns * x$$

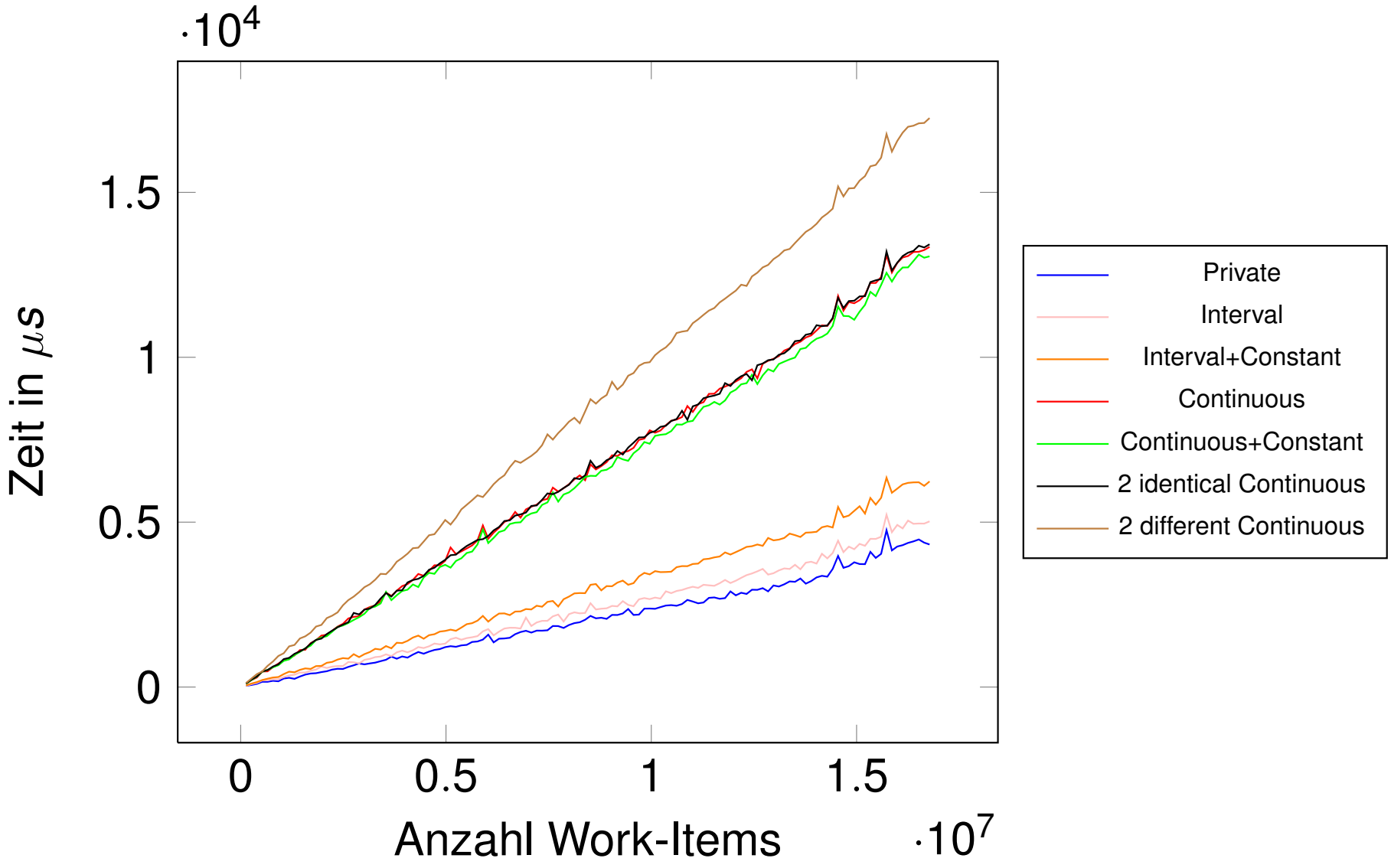
Ausführungszeitmodell

Speicherzugriffe - Zugriffsklassen

- Modell zu unpräzise für globale Zugriffe
- *GPUs haben möglicherweise Caches*
- Kategorisierung nach Zugriffsklassen
  - Constant Access
  - Interval Access
  - Continuous Access
  - Complex Access

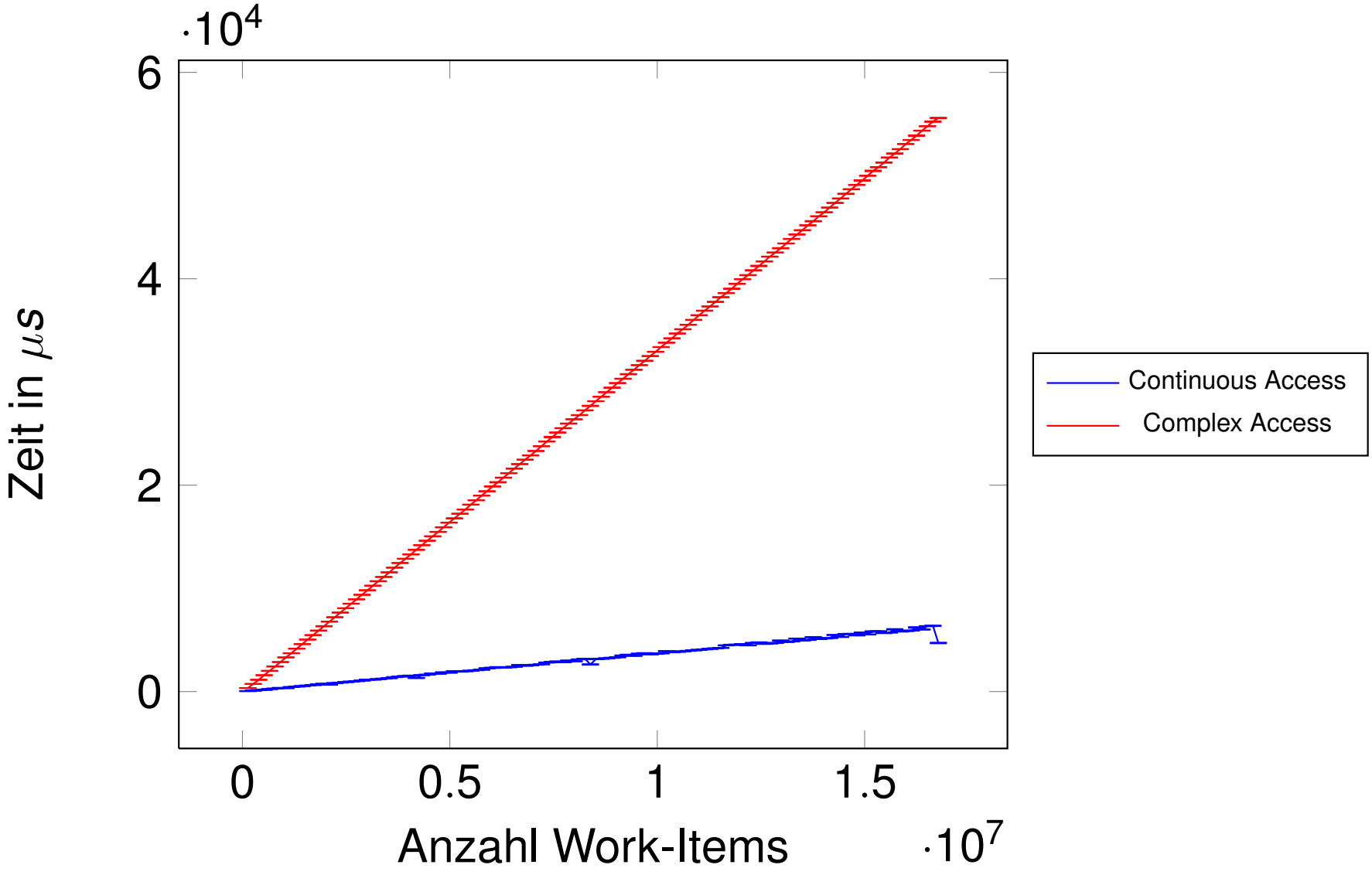
Ausführungszeitmodell

Speicherzugriffe - Zugriffsklassen - Experiment



Ausführungszeitmodell

Speicherzugriffe - Zugriffsklassen - Experiment



Ausführungszeitmodell

Speicherzugriffe - Zugriffsklassen

Ergebnis

- Linear abhängig von der Anzahl Work-Items
- Cacheeffekte
  - Zwei Zugriffe auf selbe Adresse $\rightarrow$ Zweiter kostenfrei
  - Zugriffe mit begrenztem Adressbereich deutlich günstiger
  - Zugriffe mit komplexem Zugriffsmuster deutlich teurer

MacBook Pro

$$T_{const}(x) = 45.9504ps * x$$
$$T_{ivl}(x) = 45.9504ps * x$$
$$T_{cont}(x) = 0.521932ns * x + 3\mu s$$
$$T_{complex}(x) = 3.5202282ns * x + 3\mu s$$

Agenda

Ergebnisse

Analyse zur Abschätzung von Laufzeiten

- Analyse iteriert über Syntaxbaum der Domain Iteration
- Sammelt Anzahl verschiedener Operationen, Anzahl Speicherzugriffe
- Weist Kosten zu
- Summe ist erwartete Laufzeit

Ergebnisse

Analyse zur Abschätzung von Laufzeiten - Beispiel

```
matrix.\(x,y) { x + matrix[x,y] }
```

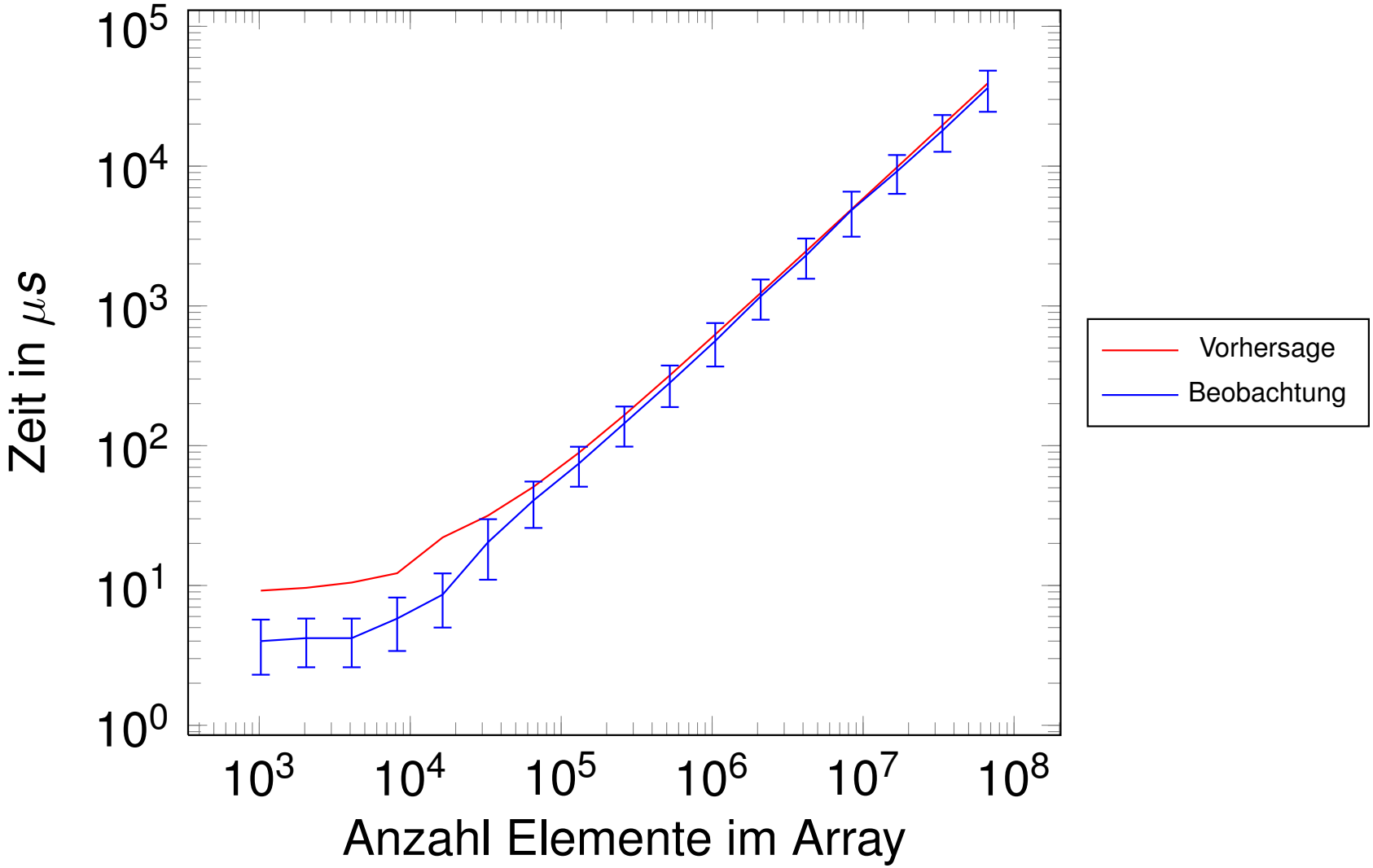
Ergebnisse

Analyse zur Abschätzung von Laufzeiten - Beispiel

Cost Type	# in Kernel	Time
FLOAT_ADD	1	54.82778805217169
FLOAT_SUB	0	0.0
FLOAT_MUL	0	0.0
FLOAT_DIV	0	0.0
INT_ADD	2	55.128781833866825
INT_SUB	0	0.0
INT_MUL	3	81.04237583646253
INT_DIV	4	1509.3169877866271
LOCAL_ACCESS	0	0.0
PRIVATE_ACCESS	1	0.0
GLOBAL_WRITE	0	0.0
CONSTANT_GLOBAL_READ	0	0.0
CACHED_GLOBAL_READ	0	0.0
GLOBAL_READ	1	4981.5752014161835
COMPLEX_GLOBAL_READ	0	0.0
BASE_COST	1	3191.479259200592
TOTAL_COST	**X**	**9873.370394125905**

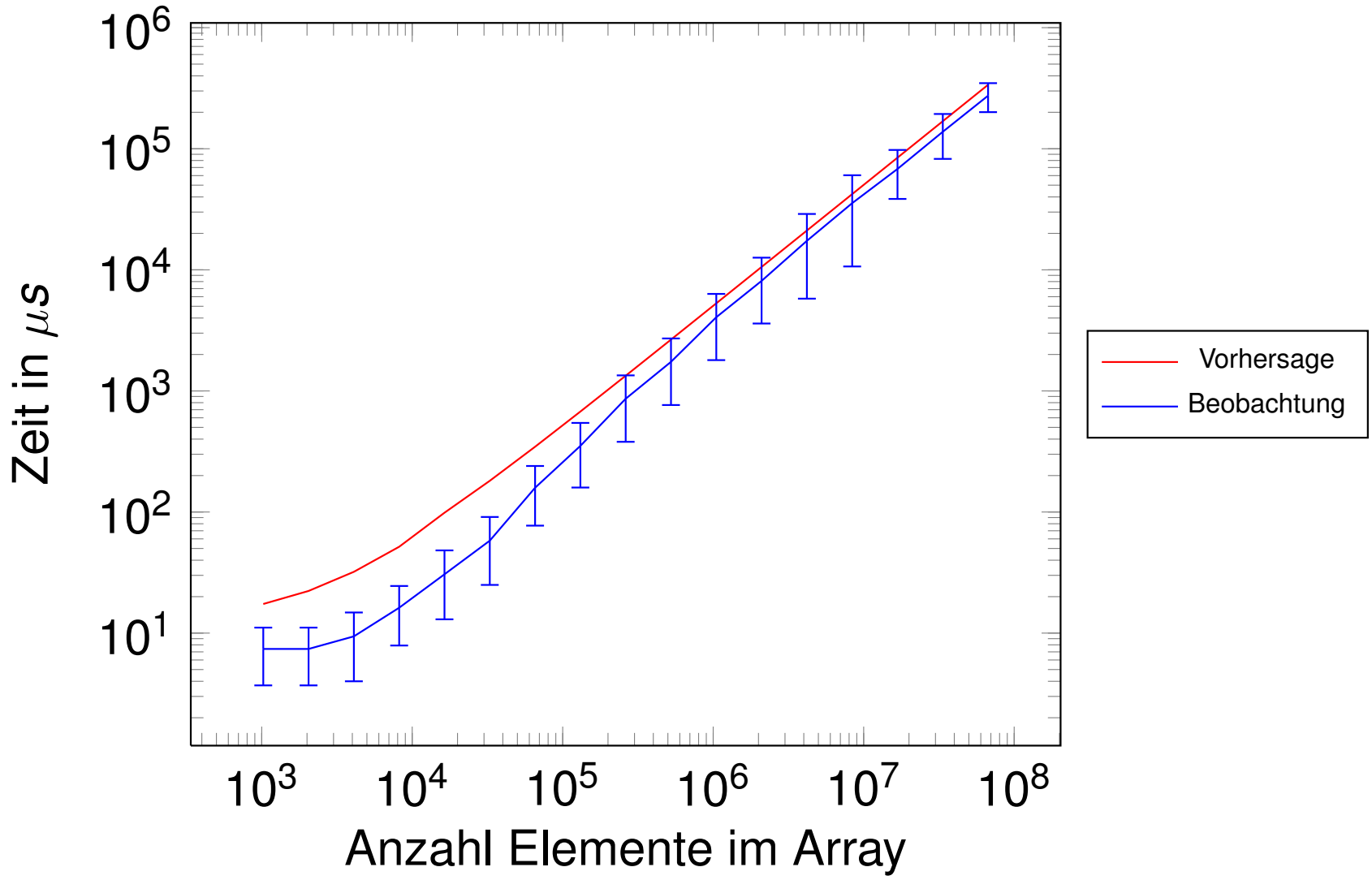
Ergebnisse

Vorhersagen - `matrix[x,y] * (matrix[x,y] * matrix[x,y])`



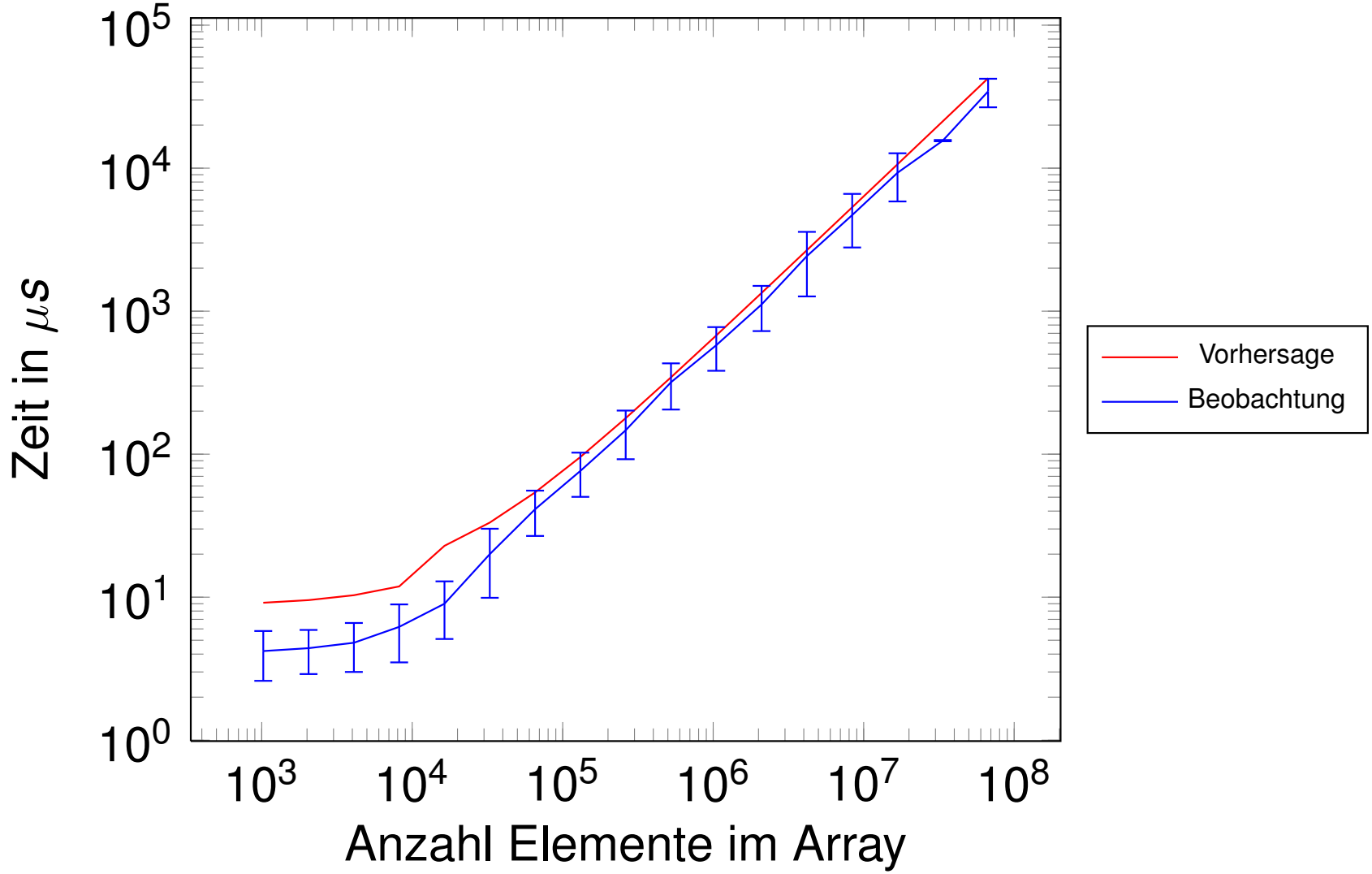
Ergebnisse

Vorhersagen - `428.3741f + ((matrix[1 % HEIGHT, 1 % WIDTH] + matrix[x,y]) + matrix[y % HEIGHT, x % WIDTH])`



Ergebnisse

Vorhersagen - `matrix[x,y] + matrix[1 % HEIGHT, 1 % WIDTH]`



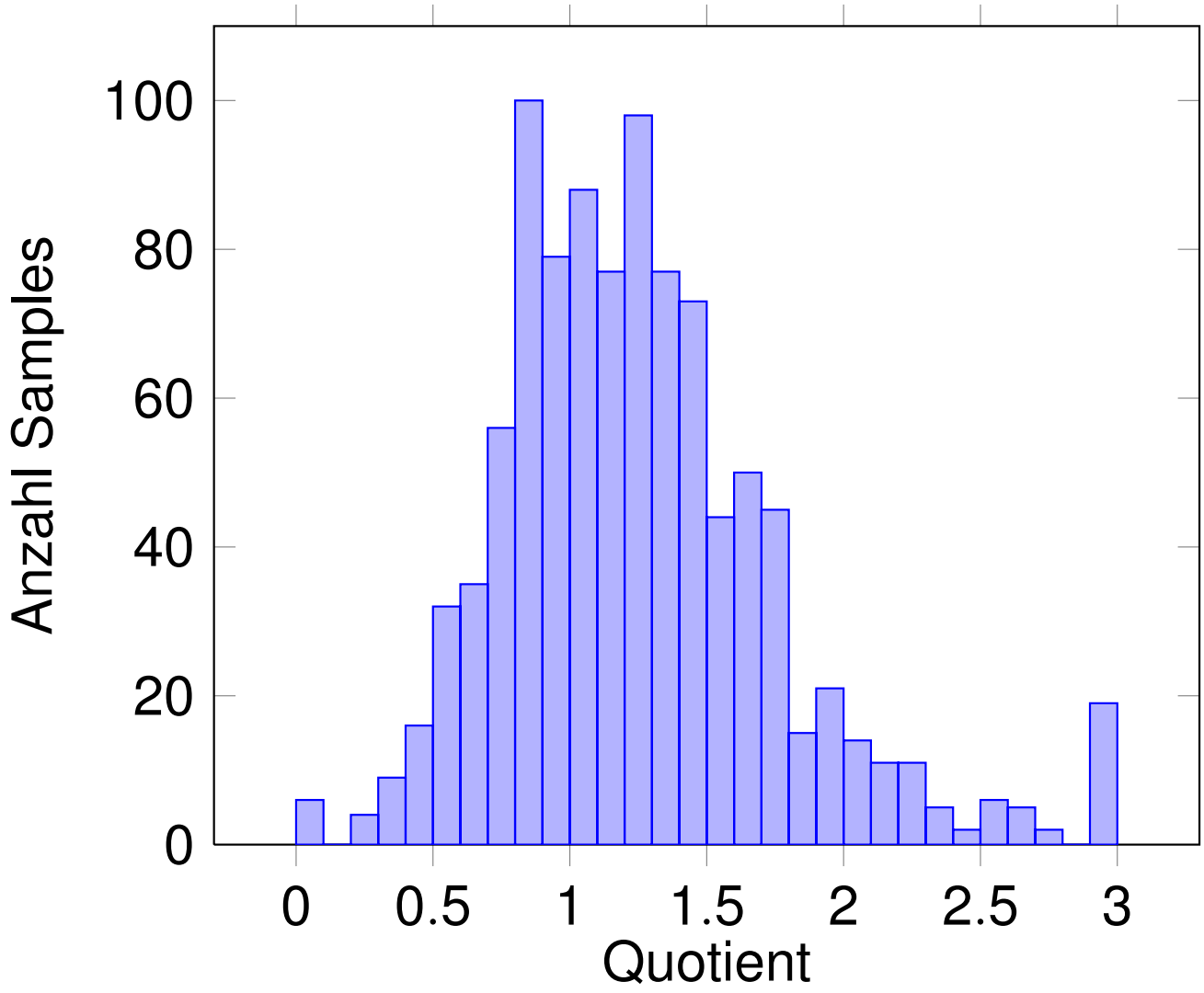
Ergebnisse

Vorhersagen - Evaluation

- Zufällig generierte Domain Iterations
- Zwei Tests, jeweils 1000 Samples
  - Zufällige Iterationen
  - “Realistische” Iterationen, keine Division
- Zur Evaluation: $q = \frac{t_{prediction}}{t_{result}}$

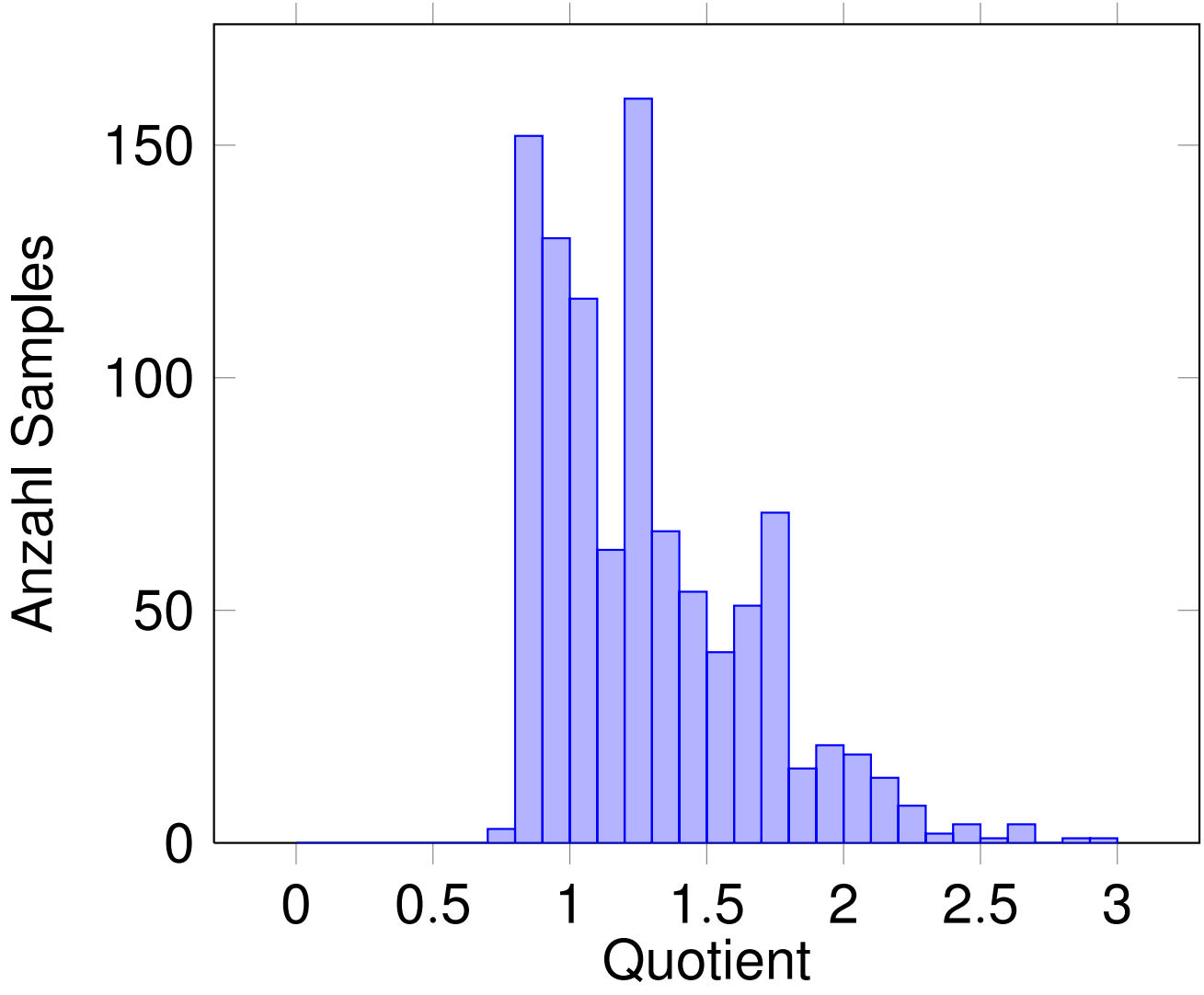
Ergebnisse

Vorhersagen - Evaluation - Zufällige Samples



Ergebnisse

Vorhersagen - Evaluation - “Realistische” Samples



Ergebnisse

Vorhersagen - Evaluation

- Leichte Überapproximation der Laufzeit
- Mögliche Gründe
  - Hardwareoptimierungen
  - Störfaktoren
  - Division
  - Falsch kategorisierte Speicherzugriffe
  - Ungenaue Erfassung lokaler Speicherzugriffe

Agenda

Ansatzpunkte

- Verbesserung des Modells
  - Lokaler Speicher
  - Klassifizierung von Speicherzugriffen
  - Division
- Andere GPU-Architekturen
  - Getestet auf GT-650M, Quadro K4000, AMD Radeon 5770 und Intel HD4000
  - Modell mit kleinen Änderungen portierbar
  - Evaluation ausstehend
- Implementierung von Sprachfeatures
  - Verschachtelte Iterationen
  - Objektorientierung

Vielen Dank

Fragen?