

-Gliederung

Gliederung



CUDA-Hardware

Einführung CUDA-Entwicklung

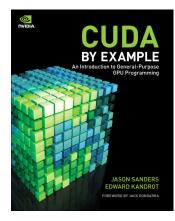
Programmier-Modell

-Literaturhinweis



Literaturhinweis





CUDA BY EXAMPLE

An Introduction to General-Purpose GPU Programming

Jason Sanders Edward Kandrot

ISBN: 0131387685

CUDA-Hardwa

CUDA-Entwicklur

rogrammier-Modell

-CUDA

CUDA

CUDA: Computer Unified Device Architectur Architektur aus Soft- und Hardware

Bereich (z.B. seti@home)
Programmiersprachen: C, C++, Java, Python, Matlab



- ► CUDA: Computer Unified Device Architectur
- Architektur aus Soft- und Hardware
- massiv paralleles Rechnen
- Anwendung im wissenschaftlichen und technischen Bereich (z.B. seti@home)
- ▶ Programmiersprachen: C, C++, Java, Python, Matlab

CUDA-Hardware

CUDA-Entwicklung

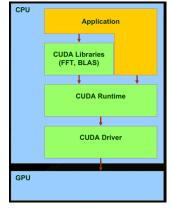
rogrammier-Modell

CUDA-Architektur



- 1. CUDA: Computer Unified Device Architectur
- 2. Architektur aus Soft- und Hardware

CUDA-Architektur



CUDA-Architektur *

HOCHSCHULE MITTWEIDA UNIVERSITY OF

CUDA-Hardwar

Einführung CUDA-Entwicklu

Programmier-Mode

^{*}http://people.maths.ox.ac.uk/~gilesm/hpc/NVIDIA/NVIDIA_CUDA_Tutorial_No_NDA_Apr08.pdf



CUDA-Hardware unterstützte Hardware Fermi-Architektur



CUDA-Hardware unterstützte Hardware

Fermi-Architektur

Einführung

Programmier Modell



```
unterstützte Hardware

settertützte CPU:

s C EX

s C T XX

s C T
```

unterstützte Hardware



unterstützte GPUs:

- ► G 8X
- ► GT 2XX
- ► GF 1XX

unterstütze Kartenfamilien:

- ► GeForce / GeForce Mobile
- ► Quadro / Quadro Mobile

unterstütze Architekturen:

- Tesla
- ► Fermi

CUDA-Hardwa

Einführung CUDA-Entwicklun

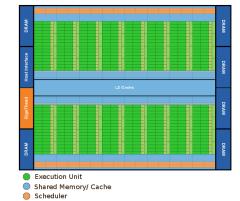
Programmier-Model





Fermi-Architektur





Fermi Architektur *

Einführung CUDA-Entwicklu

Programmier-Mode

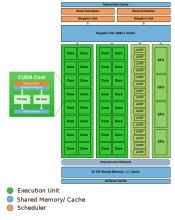
^{*}http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf





Fermi Streaming Multiprozessor





Fermi Streaming Multiprozessor *

^{*}http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf

2011-03-06

führung CUDA-Entwicklung Vorraussetzungen CUDA C Entwicklungswerkzeuge

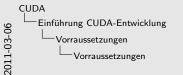


Einführung CUDA-Entwicklung

Vorraussetzungen CUDA C Entwicklungswerkzeuge

UDA-Entwickl

Ouellen



CUDA-fähiger Grafütprozensor (www.módia.com/cada)
 NVDIA Grafüterber
 CUDA Ethnicksupungsbung (nocc, CUDA-Böhichsken)
 Standard Cernpler
 Widoon Vinad Jadea 2003-2008

Mac OS X (ab Version 10.5.7): gcc

Vorraussetzungen



- ► CUDA-fähiger Grafikprozessor (www.nvidia.com/cuda)
- ► NVIDIA Gerätetreiber
- ► CUDA Entwicklungsumgebung (nvcc, CUDA-Bibliotheken)
- ► Standard C Compiler
 - Windows: Visual Studio 2005/2008
 - Linux: gcc
 - Mac OS X (ab Version 10.5.7): gcc

CUDA-Hardware

Einführung

Programmier Mede



```
Begriffe

1000 Pasiers der de mais Melbode ausführt.
(d.R. P.C.)

1000 Godfspessene (P.P.I.) und Contingenheit

1001 Fasier des Auf dem Derien ausgeliet

1001 Fasier des Auf dem Derien ausgeliet

1001 Auf Derien auf gener dem die Pasier

1001 Fasier des Aufgenfer des Information auf deriebweit

1002 Fasier des Aufgenfer des Information auf deriebweit

1003 Fasier des Aufgenfer des Information der mobile der aufgehöre des
```

Begriffe



Host: Rechner der die main-Methode ausführt. (i.d.R. PC)

Device: Grafikprozessor (GPU) und Grafikspeicher

(DRAM)

Kernel: Funktion die auf dem Device ausgeführt

und vom Host aufgerufen wird. Parallele Ausführung dieser Funktion auf mehreren

Prozessoren.

Devicefunktion: Funktion auf dem Device die nicht vom

Host aus aufrufbar ist.

CUDA-Hardwar

Einführung CUDA-Entwicklun

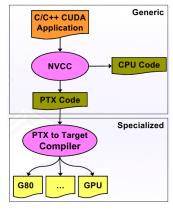
Programmier-Mode





ptx: Parallel Thread Execution

Übersetzen



Übersetzen *



CUDA-Hardw

Einführung

Programmier-Mode

^{*}http://people.maths.ox.ac.uk/~gilesm/hpc/NVIDIA/NVIDIA_CUDA_Tutorial_No_NDA_Apr08.pdf

```
CUDA

Einführung CUDA-Entwicklung

CUDA C

CUDA C

C-Erweiterungen
```

```
Envelopment

Datasetimen für Funktionen 

mittelling und Romelmacht, // / Romelmatische auf Denis 

Datasetimen für Romelmacht, // / Romelmatische auf Denis 

Denis (Denis (Den
```

C-Erweiterungen



► Deklarationen für Funktionen

```
__global__ void KernelFunc(...); // Kernelfunktion auf Device __device__ void Device(...); // Devicefunktion
```

► Deklarationen für Variablen

```
__device__ int GlobalVar; // Variable in Device-Memory __shared__ int SharedVar; // Variable in Shared Memory
```

► Erweiterte Aufrufsyntax für Kernelfunktionen
KernelFunc<<<500,128>>>(...); // 500 Blöcke a 128 Threads

```
Einführung
UDA-Entwicklung
```

Programmier-Modell

```
CUDA

Einführung CUDA-Entwicklung

CUDA C

CUDA C

Minimalbeispiel
```

Minimalbeispiel

```
HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES
```

```
__global__ void kernel( void ) {
}
int main( void ) {
   kernel <<<1,1>>>();
   printf( "Hello, World!\n" );
   return 0;
}
```

CODA-EIIIWICKI

ogrammici-ivic



chränkungen für Kernel- und cefunktionen

- ine Rekumionen
- keine statischen Variablendeklarationen
- ine variable Argumentenanzahl
- Kernelfunktionen haben keinen Rückgabewert

Einschränkungen für Kernel- und Devicefunktionen



- keine Rekursionen
- ▶ keine statischen Variablendeklarationen
- ▶ keine variable Argumentenanzahl
- ▶ automatische Variablen befinden sich in Registern
- ► Kernelfunktionen haben keinen Rückgabewert

CUDA-Hardwar

Einführung CUDA-Entwicklu

Programmier-Model

```
CUDA

Einführung CUDA-Entwicklung

CUDA C

CUDA C

Fehlerbehandlung
```

Fehlerbehandlung



Die meisten CUDA-API-Funktionen geben einen Fehlercode vom Typ cudaError_t zurück.

```
Behandlung
```

```
err = cudaMalloc((void **)&dev_c, sizeof(int));

if(err != cudaSuccess) {
  printf("%s\n", cudaGetErrorString(err));
  exit(EXIT_FAILURE);
}
```

CUDA-Hardwa

Einführung

Programmier-Model

```
CUDA

Einführung CUDA-Entwicklung

CUDA C

CUDA C

Grafikspeichernutzung
```

ikspeichernutzung

- Speicher reservieren
 .cudaNyvu_t cudaMalloc(void **devPtv_size_t si
- Speicher kopieren

 coakbreur_t coabbeopy(vod -dat, coast vod -auc,
 ain-t coast, euse coabbeopy(xad kind)
 euser coabbboopy(xad kind)
 euser coabbboopy(xad kind)
 coabbboopy(xad talout Heat Heat
 caabbboopy(xad talout Heat Device
 caabbboopy(xad talout Christon Heat
 caabbboopy(xad talout Christon
 Chris
- Speicher freigeben
 cudalivur_t cudalivee (void *desPtr)

Grafikspeichernutzung



- Speicher reservieren
 - cudaError_t cudaMalloc(void **devPtr,size_t size)
- Speicher kopieren
 - cudaError_t cudaMemcpy(void *dst,const void *src, size_t count,enum cudaMemcpyKind kind)
 - enum cudaMemcpyKind
 - ullet cudaMemcpyHostToHost Host o Host
 - ullet cudaMemcpyHostToDevice Host o Device
 - ullet cudaMemcpyDeviceToHost Device o Host
 - ullet cudaMemcpyDeviceToDevice Device
 ightarrow Device
- ► Speicher freigeben
 - o cudaError t cudaFree (void *devPtr)

CUDA-Hardwa

Einführung

Programmier-Mode

```
CUDA

Einführung CUDA-Entwicklung

CUDA C

Parameter und Rückgabewerte
```

Parameter und Rückgabewerte

```
HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES
```

```
__global__ void add(int a,int b,int *c) {
   *c = a + b;
int main( void ) {
    int c;
    int *dev_c;
    cudaMalloc((void**)&dev_c, sizeof(int));
    add < <<1,1>>>( 31, 11, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int),
        cudaMemcpyDeviceToHost);
    printf( "31 + 11 = %d\n", c );
    cudaFree(dev_c);
    return 0:
```

Einführung

CUDA-Entwicklun

rogrammier-iviodeii



ggen

CUDA-gdb
Device Emulate Mode (nvcc -deviceemu)
Island Standaran oder Treiber notwerdig
Device Threads werden auf dem Host emuliert
Verwendung von Hostalskrinnen (priest) in Kemelfunktionen m\(\tilde{\text{pit}}\) (b)
Lugriff auf Devicespapitische Dazen von Host m\(\tilde{\text{git}}\) (c)

Debuggen



- ► CUDA-gdb
- ► Device Emulate Mode (nvcc -deviceemu)
 - keine Hardware oder Treiber notwendig
 - Device Threads werden auf dem Host emuliert
 - Verwendung von Hostfunktionen (printf) in Kernelfunktionen möglich
 - Zugriff auf Devicespezifische Daten vom Host möglich und vice versa

CUDA-Hardwa

Einführung CUDA-Entwicklung

Programmier-Modell



```
Debuggen

- COUAge

- COUAge

- Denies Enales Made (over-decreases)

- Denies Enales Made (over-decreases)

- Denies Enales Made (over-decreases)

- Denies Tendat varies and fast fine resident
- decreases and over-decreases (over-decreases)

- La grade and the consequention Dates som their midglich
- decreases over-decreases Made
- Denies Made (over-decreases Made (over-decreases))

- Denies Tendam (over-decreases)
- Denies Tendam (over-decreas
```

Debuggen



- ► CUDA-gdb
- ► Device Emulate Mode (nvcc -deviceemu)
 - keine Hardware oder Treiber notwendig
 - Device Threads werden auf dem Host emuliert
 - Verwendung von Hostfunktionen (printf) in Kernelfunktionen möglich
 - Zugriff auf Devicespezifische Daten vom Host möglich und vice versa

Fallstricke des Device Emulate Mode

- ► Devicethreads werden sequentiell ausgeführt
- Ergebnisse von Gleitpunktoperationen können verschieden sein
 - verschiedene Compileroutputs
 - verschiedene Befehlssätze
 - andere Präzisionen für Zwischenergebnisse

CUDA-Hardwa

Einführung UDA-Entwicklung

Programmier-Modell

Programmier-Modell
Grids, Threads and Blocks
Speicherarten
Wichtige Funktionen und Variablen



Programmier-Modell

Grids, Threads and Blocks Speicherarten Wichtige Funktionen und Variablen

IDA-Entwicklun

rogrammier-Model



Grids, Threads and Blocks



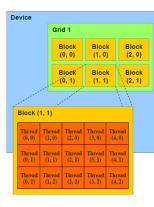
Grids

Kernelfunktion wird als Gitter von Blöcken ausgeführt

Blocks

Menge von Threads, die miteinander kooperieren können:

- Synchronisierung
- ► Shared Memory



(Quelle: NVIDIA 2006)

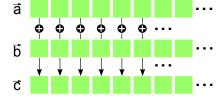
Einführung CUDA-Entwicklur Programmier-Mod





Beispiel: Vektoraddition





- CUDA-Hardwa
- Einführung CUDA-Entwicklu
- Programmier-Model
 - Queller

- $\vec{c} = \vec{a} + \vec{b}$
- ► Größe jedes Vektors: WIDTH
- ightharpoonup jeder Block bearbeitet ein Element von \vec{c}

```
CUDA
Programmier-Modell
Grids, Threads and Blocks
Konfiguration der Grids/Blocks
```

Konfiguration der Grids/Blocks



wird im Kernelaufruf festgelegt:

```
__global__ void vecadd(int* a,int* b,int* c)
    { ... }

int main() {
    ...
    // Kernelaufruf <<<dimGrid, dimBlock>>>
    vecadd<<<<WIDTH, 1>>>(a, b, c);
    ...
}
```

▶ max. 512 Threads pro Block

CUDA-Hard

Einführung

Programmier-Model

```
CUDA
Programmier-Modell
Grids, Threads and Blocks
Block- und Thread-IDs
```

```
Slock- and Thread-IDs

(1) shorters, (1) sho
```

Block- und Thread-IDs

```
HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES
```

```
__global___ void vecadd(int* a,int* b,int* c) {
    int i;
    i = blockIdx.x;

    c[i] = a[i] + b[i];
}

int main() {
    ...
    // Kernelaufruf <<<dimGrid, dimBlock>>>
    vecadd<<<N, 1>>>(a, b, c);
    ...
}
```

CUDA-Hardwa

Einführung CUDA-Entwicklun

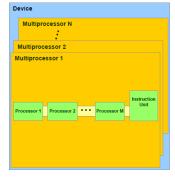
Programmier-Model

CUDA 90-60 -Trogrammier-Modell Grids, Threads and Blocks Multiprozessoren: SPMD & SIMD



Multiprozessoren: SPMD & SIMD





Einführung CUDA-Entwicklun

Programmier-Mode

Queller

(Quelle: NVIDIA 2006)

SPMD: Single Program Multiple Data

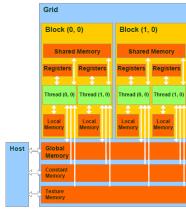
SIMD: Single Instruction Multiple Data







- ► Global Memory
 - Read/Write für alle Threads
 - langsam
- Constant Memory
- ► Texture Memory
- Shared Memory
- Registers
- ► Local Memory



(Quelle: NVIDIA 2008)

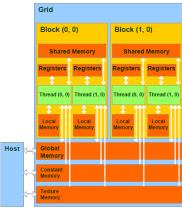
. Januar 201







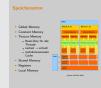
- ► Global Memory
- ► Constant Memory
 - Read-Only für alle Threads
 - ullet cached o schnell
- ► Texture Memory
- ► Shared Memory
- Registers
- ► Local Memory



(Quelle: NVIDIA 2008)

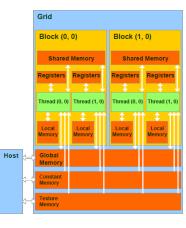
grammier-N





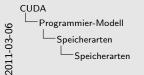


- ► Global Memory
- ► Constant Memory
- ► Texture Memory
 - Read-Only f
 ür alle Threads
 - ullet cached o schnell
 - mehrdimensionaler
 Cache
- ► Shared Memory
- Registers
- ► Local Memory



(Quelle: NVIDIA 2008)

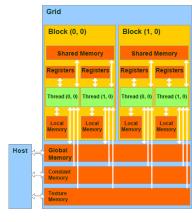
OA-Entwickli rammier-Mo







- Global Memory
- Constant Memory
- ► Texture Memory
- ► Shared Memory
 - Read/Write für alle Threads im Block
 - schnell
- Registers
- ► Local Memory



(Quelle: NVIDIA 2008)

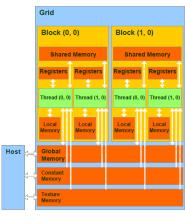
DA-Entwick







- ► Global Memory
- Constant Memory
- ► Texture Memory
- Shared Memory
- Registers
 - Read/Write für einzelnen Thread
 - schnell
- ► Local Memory



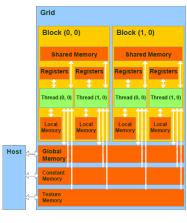
(Quelle: NVIDIA 2008)







- ► Global Memory
- Constant Memory
- ► Texture Memory
- Shared Memory
- Registers
- ► Local Memory
 - Read/Write für einzelnen Thread
 - langsam

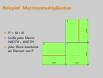


(Quelle: NVIDIA 2008)

DA-Entwick

rogrammier-Mod

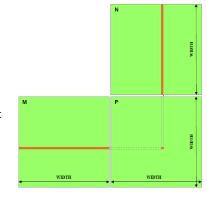




Beispiel: Matrizenmultiplikation



- P = M * N
- ► Größe jeder Matrix: WIDTH * WIDTH
- jeder Block bearbeitet ein Element von P



CUDA-Hardwa

Einführung CUDA-Entwicklun

Programmier-Model



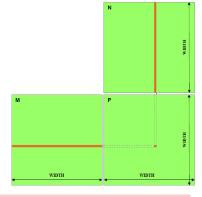


Beispiel: Matrizenmultiplikation





- ► Größe jeder Matrix: WIDTH * WIDTH
- jeder Block bearbeitet ein Element von P



Finführung

CUDA-Entwicklun

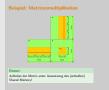
Programmier-Mode

Quellen

Problem:

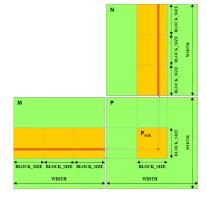
Viele (langsame) Zugriffe auf dieselben Speicherbereiche!





Beispiel: Matrizenmultiplikation





Einführung CUDA-Entwicklur

Programmier-Mode

Queller

Besser:

Aufteilen der Matrix unter Ausnutzung des (schnellen) Shared Memory!



Wichtige Funktionen und Variable

- blockIdx.x und blockIdx.y identifizieren Block
 threadIdx.x und threadIdx.y identifizieren Thread
- void __ayacthreads(void) synchronisiert alle Threads eines Blocks

Wichtige Funktionen und Variablen



- blockIdx.x und blockIdx.y identifizieren Block
- threadIdx.x und threadIdx.y identifizieren Thread innerhalb seines Blocks
- ▶ void __syncthreads(void) synchronisiert alle Threads eines Blocks

CUDA-Hardware

CUDA-Entwicklung

Programmier-Modell



en

- NVIDIA CUDA Library Documentation
- cuds/2_3/toolkit/docs/online/index.html CUDA: A New Architecture for Computing on the GPU von NVIDIA (2006) http://people.maths.ox.ac.uk/-gilesm/hpc/
- NYIDIA/Cuda2.pdf

 NVIDIA's Next Generation CUDA Compute A chitecture: Fermi von NVIDIA (2006)
- NVIDIAFermiComputeArchitectureWhitepaper.p.

 * Tutorial CUDA von Cyrl Zeller (2008)
 http://people.maths.ox.ac.uk/~gilesm/hpc/
 NVIDIA/NVIDIA_CUDA_Tutorial_No_NDA_Apr06.
 pdf

Quellen



- ► NVIDIA CUDA Library Documentation
 http://developer.download.nvidia.com/compute/
 cuda/2_3/toolkit/docs/online/index.html
- ► CUDA: A New Architecture for Computing on the GPU von NVIDIA (2006) http://people.maths.ox.ac.uk/~gilesm/hpc/ NVIDIA/Cuda2.pdf
- ► NVIDIA's Next Generation CUDA Compute Architecture: Fermi von NVIDIA (2006)
 http://www.nvidia.com/content/PDF/
 fermi_white_papers/
 NVIDIAFermiComputeArchitectureWhitepaper.pdf
- ► Tutorial CUDA von Cyril Zeller (2008) http://people.maths.ox.ac.uk/~gilesm/hpc/ NVIDIA/NVIDIA_CUDA_Tutorial_No_NDA_Apr08. pdf

CUDA-Hardwa

Einführung

Programmier-Mode

Vielen Dank für die Aufmerksamkeit.

Fragen?



Vielen Dank für die Aufmerksamkeit.

Fragen?