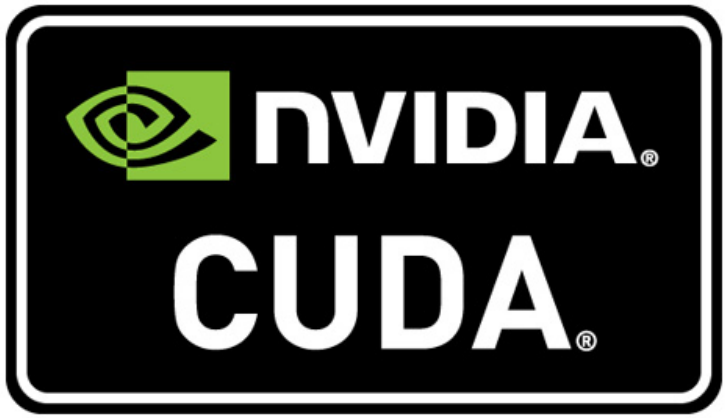
Compute Unified Device Architecture

CUDA

Thomas Koller

06. Februar 2012



Gliederung

Einleitung

CUDA: **C**ompute **U**nified **D**evice **A**rchitecture

- entwickelt von Nvidia Corporation
- spezifiziert Software- und Hardwareeigenschaften
- Ziel: Grafikkarte als allgemeiner Parallelrechner nutzen
  - engl. GPGPU: **g**eneral-**p**urpose computation on **g**raphics **p**rocessor **u**nits

Gründe für diese Entwicklung

- wesentlich größere theoretische Rechenleistung als CPU
- Grafikkarten bis zu 512 Kerne (Nvidia), 2048 Kerne (AMD)
- aktuelle CPUs 2-8 Kerne
- wesentlich größere Speicherbandbreite (über 250GB/s)

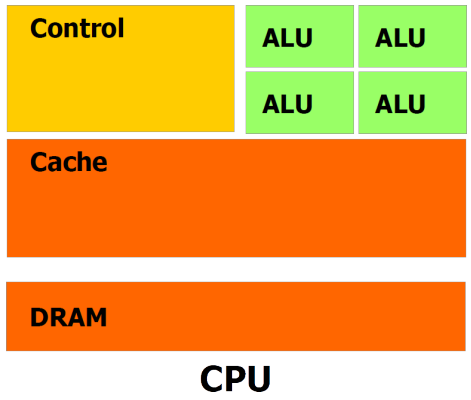
Probleme

- unterschiedliche Programmiermodelle
- komplexer als bei CPU-Programmierung

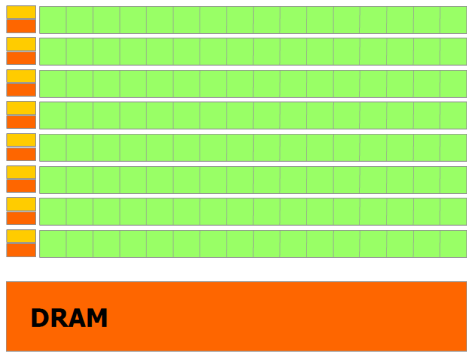
Die Grafikkarte

Unterschiede zum Hauptprozessor

- unterschiedliche Aufgabengebiete
- CPU
  - sequenzielle Berechnungen
  - anfangs ein Kern, nun bis zu acht Kernen
  - Leistungssteigerung hauptsächlich durch Taktsteigerungen
  - große Kontrolllogik und Caches für höhere sequenzielle Geschwindigkeit
- GPU
  - anfangs nur für Bildberechung
  - Manycore-Konzept
  - Bildberechungen lassen sich gut parallelisieren
  - Leistungssteigerung durch mehr, dafür einfachere Kerne
  - SIMD-Prinzip: durch Parallelisierung wenig Kontrolllogik und Caches notwendig



CPU



GPU

CUDA

Aufbau und Konzept

- Sprache: CUDA C
  - C mit CUDA-spezifischen Schlüsselwörtern
- Nvidia C-Compiler: nvcc
  - Trennung zwischen Device-Code (GPU) und Host-Code (CPU)
  - Device-Code: nvcc
  - Host-Code: normaler Compiler, z.B. GCC
- auch zur Laufzeit Unterscheidung zwischen Host- und Device-Code
  - asynchrone Ausführung möglich

CUDA

Programmablauf

- CPU und GPU besitzen jeweils eigenen Speicher
  - Programmierer muss Speicher selbstständig verwalten
  - Daten werden nicht automatisch kopiert
  - direkter Zugriff auf den Hauptspeicher ist nicht möglich
- Programmablauf:
  1. Programmlogik weiterhin auf Host
  2. GPU-Speicher allokieren
  3. Daten vom Host zum Device kopieren
  4. Berechnungen auf GPU
  5. Ergebnis vom Device zum Host kopieren
  6. allokierten Speicher wieder freigeben

Programmiermodell

Ausführungselemente

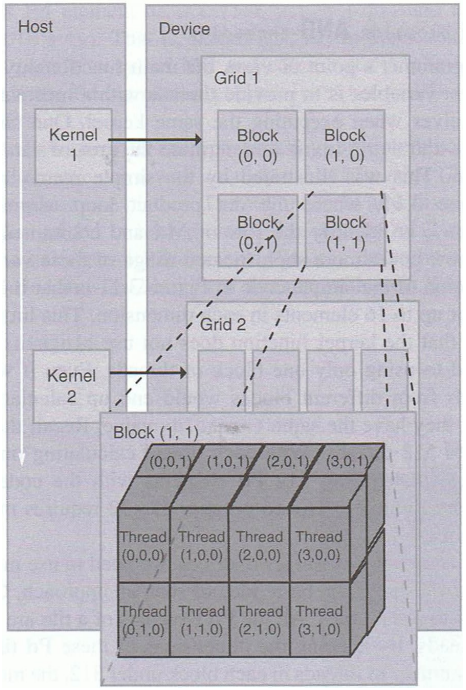
- Funktionen, die auf GPU ausgeführt werden: „Kernel“
- Aufruf: `KernelFunction<<<dimGrid, dimBlock>>>(...);`
- Datenparallelität: N Threads in einem Kernel
- Schlüsselwort: `__global__`
- wichtig: `void`
  - Kernel können keine Werte zurückliefern
  - Rückgabe erfolgt über Zurückkopieren des Speicherbereichs

```
1  __global__ void VecAdd(float* A, float* B,
       float* C) {
2    int i = threadIdx.x;
3    C[i] = A[i] + B[i];
4  }
5
6  int main() {
7    ...
8    VecAdd<<<1, N>>>(A, B, C);
9    ...
10 }
```

Programmiermodell

Datenparallelität

- Datenparallelität:
  - Daten aufgeteilen
  - auf jedem dieser Datenteile gleiche Berechung durchführen
- CUDA unterstützt Strukturierung in Blöcke und Threads
- Kernel wird als Grid ausgeführt
- Grid besteht aus 2D-Blöcken
- Blöcke bestehen aus 3D-Threads
- Kernelaufruf: `KernelFunction<<< dimGrid, dimBlock>>>(...);`
- `dimGrid`, `dimBlock` vom Typ `dim3`




```
1 dim3 dimGrid(2, 2, 1);
2 dim3 dimBlock(4, 2, 2);
3 KernelFunction<<<dimGrid,
      dimBlock>>>(...);
```

Programmiermodell

Speicherverwaltung und Synchronisation

Speicherverwaltung

- Grafikkarte besitzt eigenen lokalen Speicher
  - schneller Zugriff
  - hohe Bandbreite (über 250GB/s)
  - manuelle Verwaltung notwendig
- CUDA-Befehle:
  - `cudaMalloc(void **devPtr, size_t size);`
  - `cudaMemcpy(void *dst, const void *src, size_t count, enum cudaMemcpyKind kind);`
    - `cudaMemcpyHostToDevice`, `cudaMemcpyDeviceToHost`, `cudaMemcpyHostToHost`, `cudaMemcpyDeviceToDevice`
  - `cudaFree(void *devPtr);`

Synchronisation

- nur Barrieren-Synchronisation
- `__syncthreads();`

Programmiermodell

Compute Capabilities

- Spezifikation des Funktionsumfangs der Grafikchips
- Unterteilung in „major revision“ und „minor revision“
- major revision: Architektur des Chips
  - Geforce 8800 GTX (G80) bis Geforce GTX 285 (GT200b): 1.x
  - seit Geforce GTX 480 (GF100) „Fermi-Architektur“: 2.x
- minor revision: kleinere Architekturverbesserungen
  - 1024 anstatt 512 Threads pro Block
  - Operationen auf Gleitkommazahl mit doppelter Genauigkeit
  - atomare Funktionen im Speicher

	Compute Capability				
Feature Support (Unlisted features are supported for all compute capabilities)	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**
Atomic functions operating on 32-bit integer values in global memory (Section B.11)	No	Yes			
atomicExch() operating on 32-bit floating point values in global memory (Section B.11.1.3)					
Atomic functions operating on 32-bit integer values in shared memory (Section B.11)	No		Yes		
atomicExch() operating on 32-bit floating point values in shared memory (Section B.11.1.3)					
Atomic functions operating on 64-bit integer values in global memory (Section B.11)					
Warp vote functions (Section B.12)					
Double-precision floating-point numbers	No			Yes	
Atomic functions operating on 64-bit					

Werkzeuge

Entwicklungswerkzeuge und Bibliotheken

Entwicklungswerkzeuge

- CUDA Toolkit
  - aktuell Version 4.1
  - unterstützt Linux, Mac und Windows
  - enthält nvcc, cuda-dgb, cuda-memcheck, Visual Profiler und Bibliotheken für häufig verwendete Funktionen
- Nvidia Parallel Nsight
  - Integration in Microsoft Visual Studio 2008 oder 2010
  - debugging, profiling und tracing

Bibliotheken

- große Anzahl an unterschiedlichen Bibliotheken
- cuFFT: numerische Algorithmen
- cuBLAS: lineare Algebra
- Thrust:
  - Erstellen und Kopieren von einfachen Datenstrukturen
  - Algorithmen: z.B. Sortierung, Transformation, Reduktion

Zusammenfassung

- Ziel von CUDA: Grafikkarte als allgemeiner Parallelrechner
- Nutzung der potentiell wesentlich größeren Rechenleistung
- Problem: CPU und GPU haben unterschiedliche Programmiermodelle
  - manuelle Speicherverwaltung
  - eingeschränkte Synchronisation
- Kernel, Datenparallelität
  - Grid, Block, Threads
- Compute Capabilities spezifizieren die Hardwareeigenschaften
- unterschiedliche Entwicklungswerkzeuge und Bibliotheken

Vielen Dank
für
Ihre Aufmerksamkeit!