# Einführung

## Motivation

## Zielsetzung

## Kapitelübersicht

# Grundlagen

## Paralleles Programmieren

### Daten Parallelität

Was ist das

Given the difficulty of finding enough subsystem tasks to assign to dozens of cores—the only elements of which there are a comparable number are data elements—the data-parallel approach is simply to assign an individual data element to a separate logical core for processing. Instead of breaking code down by subsystems, we look for fine-grained inner loops within each subsystem and parallelize those. For some tasks, there may be thousands to millions of data elements, enabling assignment to thousands of cores. (Boyd, 2008)

Compute bound and memory bound

The most important characteristic of the GPU memory subsystem is the cache architecture. Unlike a CPU, the GPU has hardly any read/write cache. It is assumed that so much data will be streaming through the processor that it will overflow just about any cache. As a result, the only caches present are separate read-through and write-through buffers that smooth out the data flow. Therefore, it is critical to select algorithms that do not rely on reuse of data at scales larger than the few local registers available.

### Parallel STL

#### Standard Algorithms

#### Execution Policies

#### Typische anwendung von Parallelem Code

Stencil operations, Bildverarbeitung,

## (GPU und Nvidia HPC)(Heterogene Systeme)

### GPU

#### Hardwarearchitektur

Eine CPU ist so entworfen, dass sie eine Sequenz von Operationen so schnell wie möglich ausführen kann. Dahingegen ist die GPU darauf ausgelegt tausende von Operationen gleichzeitig auszuführen. Deshalb unterscheidet sich die Hardwarearchitektur der GPU von der einer CPU.(Programming Guide : CUDA Toolkit Documentation, 2022)

Die Grundarchitektur einer GPU unterscheidet von der einer CPU in mehreren Aspekten. Während eine CPU aus einer Handvoll von komplexen Kernen besteht, ist eine GPU mit hunderten einfacheren Kernen ausgestattet. Diese Kerne besitzen jeweils tausende parallel laufende Hardware Threads.(Brodtkorb et al., 2013, 2013) Abbildung 1 veranschaulicht den Aufbauunterschied.

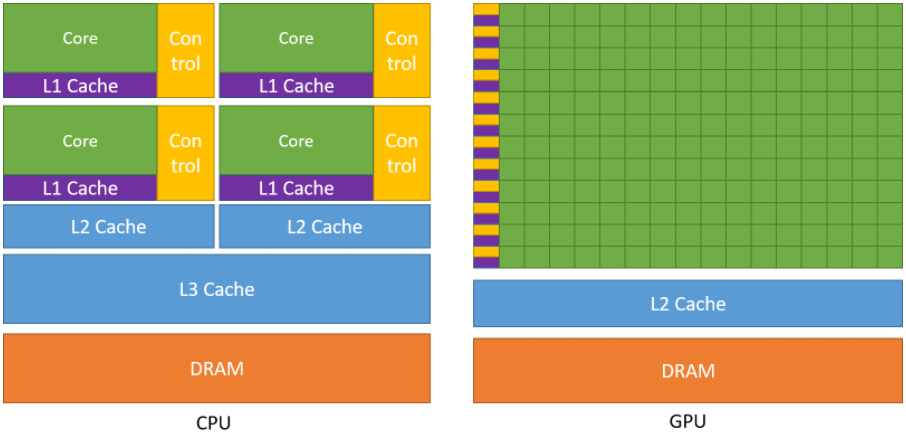


Abbildung . Die GPU dediziert mehr Kerne für die Datenverarbeitung. (Programming Guide : CUDA Toolkit Documentation, 2022)

#### Speicherverwaltung / Heterogene Systeme

Zusammenarbeit zwischen CPU und GPU in bezug auf Speicherverwaltung.

### CUDA

CUDA ist eine Programmierschnittstelle entwickelt von Nvidia. Diese ermöglicht es dem Programmierer Anwendungssoftware zu schreiben, die ihre Parallelität skaliert um die steigende Anzahl an Prozessorkernen vollends auszunutzen. CUDA stellt drei Abstraktionen bereit, welche es ermöglichen ein Problem in Teilprobleme aufzuteilen. Abstrahiert wird unteranderem die zu Verfügung stehenden Threads. Threads werden zu gleichgroßen Blöcken zusammengefasst. Teilprobleme können dann unabhängig voneinander parallel gelöst werden, indem jedem Teilproblem ein Block mit Threads zugeteilt wird. Die Anzahl von Threads in einem Block ist limitiert durch die Anzahl der Threads die sich auf einem Prozessorkern befinden. Auf aktuellen GPUs kann ein Threadblock bis zu 1024 Threads beinhalten. Threadblöcke sind Teile eines sogenannten Gitters, welches die Threadblöcke organisiert. (Programming Guide : CUDA Toolkit Documentation, 2022) Abbildung 2 zeigt die logische Unterteilung von Threads in Blöcke und Gitter.

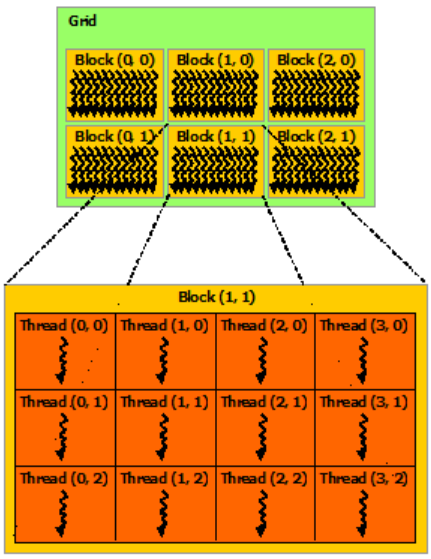


Abbildung . Ein Gitter (engl. Grid) ist unterteilt in Blöcke (engl. Blocks) von Threads. (Programming Guide : CUDA Toolkit Documentation, 2022)

CUDA erlaubt es dem Programmierer bestimmte Funktionen zu definieren, welche dann parallel von CUDA Threads ausgeführt werden. Diese Funktionen werden Kernel genannt. Einem Kernel wird definiert wie viele CUDA Threads diesen ausführen. Hierbei ist die Anzahl der Threads nicht limitiert, da ein Kernel von mehreren Threadblöcken ausgeführt werden kann. Bei der Definition eines Kernels wird die Anzahl der Threads pro Block und die Anzahl von Blöcken pro Gitter angegeben.(Programming Guide : CUDA Toolkit Documentation, 2022)

Nvidias GPU Architektur ist um eine Anordnung von Streaming Multiprozessoren (SMs) konzipiert. Jeder Streaming Multiprozessor kann einen Threadblock nach dem anderen ausführen. Die Anwendungssoftware wird auf die entsprechende GPU Hardwarearchitektur skaliert, indem die auszuführenden Threadblöcke auf die Anzahl der verfügbaren Streaming Multiprozessoren gleichmäßig verteilt werden. So kann eine GPU mit vier Streaming Multiprozessoren vier Threadblöcke gleichzeitig ausführen, während eine GPU mit zwei Streaming Multiprozessoren in der gleichen Zeitspanne nur zwei Threadblöcke ausführen kann. (Programming Guide : CUDA Toolkit Documentation, 2022)

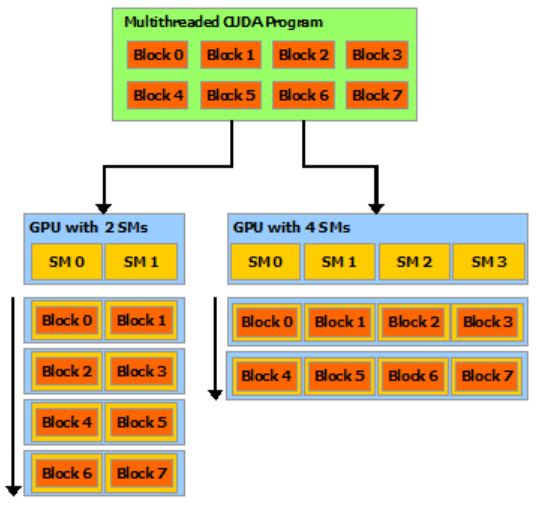


Abbildung 3. Automatische Skalierung durch Aufteilung der Blöcke auf die Verschiedenen Streaming Multiprozessoren. (Programming Guide : CUDA Toolkit Documentation, 2022)

## **(Algorithmus**)

# Lösungsansatz / Versuchsaufbau

# Ergebnisse

# Zusammenfassung

References

Boyd, C. Data-parallel computing. In *ACM SIGGRAPH 2008 classes on - SIGGRAPH '08*; Unknown, Ed.; ACM Press: New York, New York, USA, 2008; p 1. DOI: 10.1145/1401132.1401150.

Brodtkorb, A. R.; Hagen, T. R.; Sætra, M. L. Graphics processing unit (GPU) programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing* [Online] **2013,** *73* (1), 4–13. https://​www.researchgate.net​/​publication/​257252061\_Graphics\_processing\_unit\_GPU\_programming\_strategies\_and\_trends\_in\_GPU\_computing.

Programming Guide : CUDA Toolkit Documentation. https://​docs.nvidia.com​/​cuda/​cuda-c-programming-guide/​index.html (accessed February 1, 2022).