GPU Computing

Christian Mösl Burhan Karababa

10. Jänner 2020

Department of Computer Sciences
University of Salzburg

Graphics Processing Unit - Computing

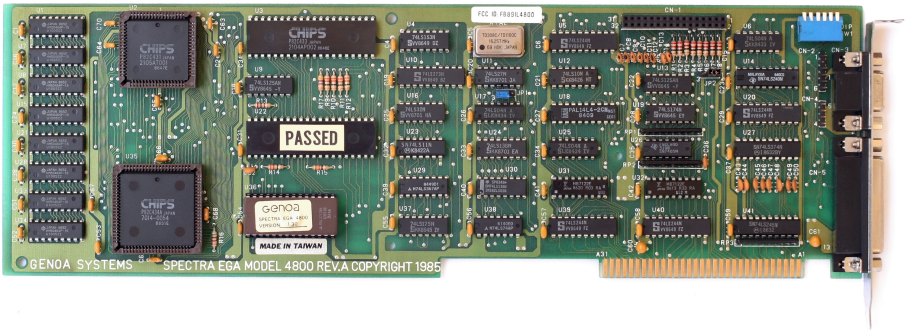
Aufgabe der Grafikkarte

3 x 3 Pixel Bildschirm:

```
framebuffer[3][3] = {
  0x000102, 0x00000, 0xFFFFFF
  0x0000FF, 0x00000, 0xAAAAAA
  0xBBBBBB, 0x00000, 0x000000
}
```

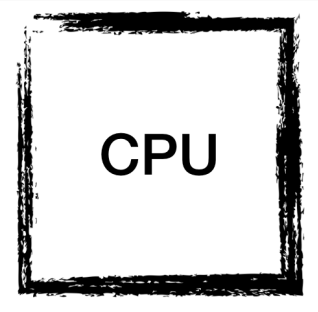
http://www.digitron.cz/galerie/original/a_PMD60-1b.jpg

Grafikkarte aus 1985



https://de.wikipedia.org/wiki/Grafikkarte

2D Beschleuniger



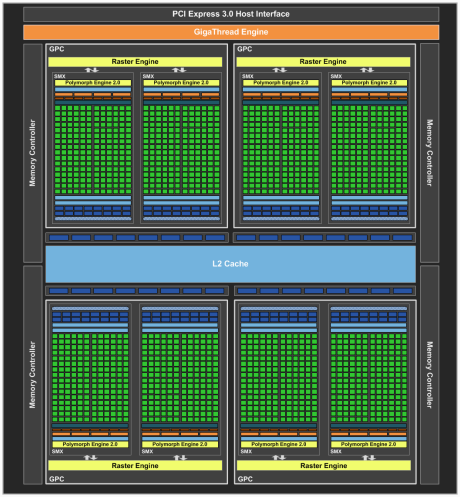
$$drawLine(x, y, length, angle) \Rightarrow$$

3D Beschleuniger: Spiele

Rendering Problem:

- Geometrie
- Texturen
- Licht
- Schattierung
- Perspektive

Parallelität in Hardware

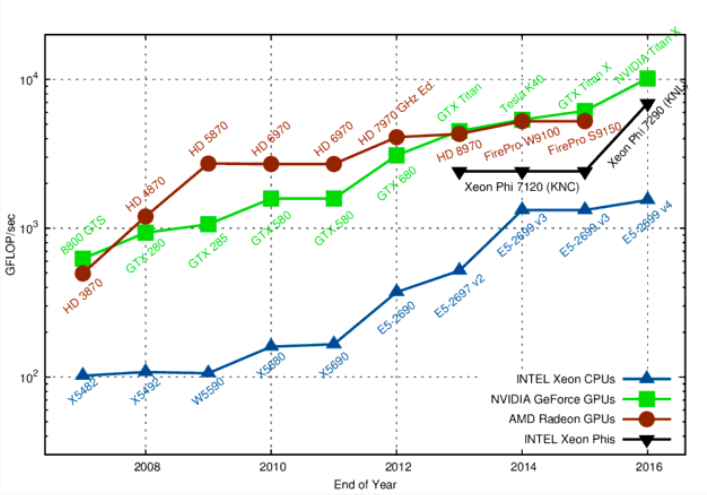


https://images.anandtech.com/doci/5699/GeForce_GTX_680_Block_Diagram_FINAL.png

FLOPS - eine Metrik

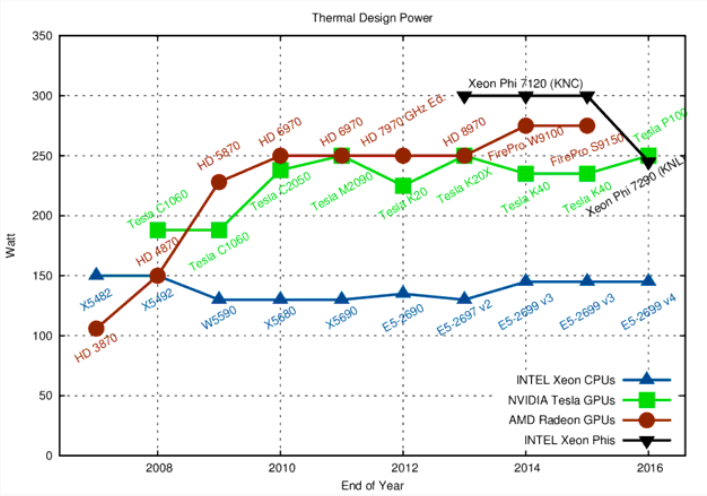
- **Fl**oating Point **Op**eration **P**er **S**econd
- arithmetische Operationen: +, -,...
- nicht überbewerten

CPU vs. GPU



https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

CPU vs. GPU



https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

Mythbusters Demo GPU versus CPU

KLICK

GPU Computing: Mathematische Probleme

- Lösung von Algorithmen/mathematischen Problemen
- Bewertung des Algorithmus aufgrund der Komplexität/Parallelisierungsgrad
- Arithmetische Berechnungen in der GPU/CPU?
- Embarrassingly-Parallel-problems und Inherently-serial-problems

Embarrassingly-parallel-problems

- keine Abhängigkeiten zwischen einzelnen Schritten
- Verarbeitungsschritte parallel durchführen
- Unterteilung in mehrerere kleinere Probleme
- kein Kommunikationsaufwand zwischen Prozessen notwendig
- benötigt mögichlicherweise Zusammenführung der Ergebnisse
- Beispiele: Matrizenrechnung, Password cracking

Inherently-serial-problems

- keine Parallelisierbarkeit aufgrund von starken Abhängigkeiten
- benötigen Zwischenergebnisse um effizient weiterzuarbeiten
  $\rightarrow$ z.B. Three-body-problem

GPGPU (General Purpose Computation on Graphics Processing Unit)

- Programmierschnittstelle: allgemeine Berechnungen von GPU auf Grafikkarte
- Rechenleistung konsequent ausnutzen
- Privatgebrauch: praktisch keine Anwendung für GPGPU (Ausnahme: Programme mit Videoverarbeitung)
- Einsatzgebiet: Berechnung von wissenschaftlichen arithmetischen Problemen
- 2007 Meilenstein: NVidia-Toolkit für die Programmierung von GPU's

CUDA - Toolkit

- CUDA (Compute Unified Device Architecture): Wegbereiter GPU Computing
- von Nvidia entwickelte Programmier-Technik $\rightarrow$ Bereitstellung Rechenkapazität
- entwickelt für wissenschaftliche Programmierungen
- nur für Einsatz auf NVidia-Karten $\rightarrow$ herstellerabhängig
- Erweiterung der Sprache C/C++ $\rightarrow$ geringe Einarbeitungszeit, keine Grafikkenntnisse notwendig

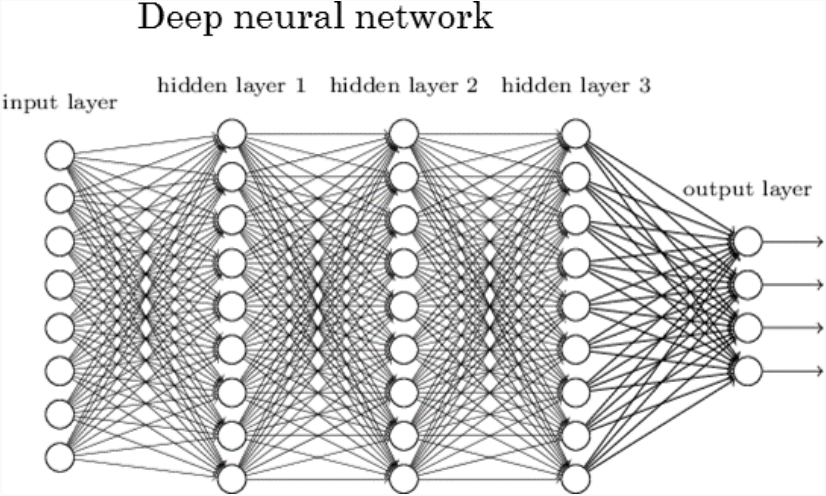
OpenCL - Toolkit

- OpenCL (Open Computing Language): offener Standard für Implementierung aller Typen von GPU's
- 2008 zum ersten Mal als Standard eingereicht von AMD, IBM, Intel
- Erweiterung der Sprache C/C++ (wie CUDA)
- Installation von Treiber und Bibliotheken sehr komplex
- Schwierigkeiten bei Installation Linux $\rightarrow$ mangelnde Integration der Installer

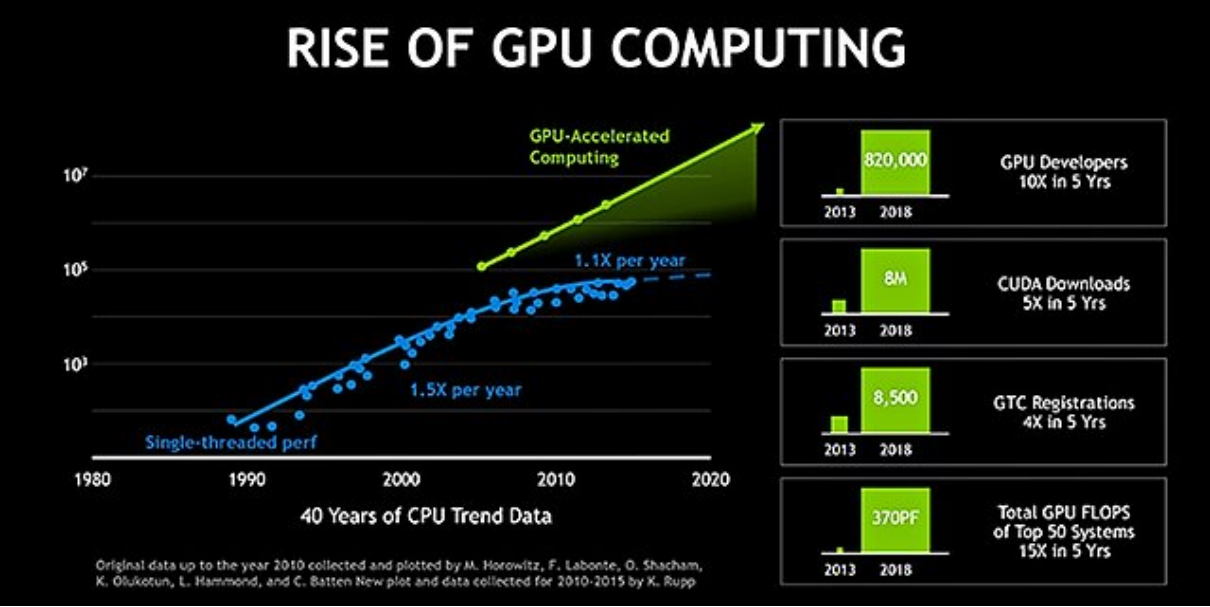
Anwendung: Folding@Home

- Erforschung von Krankheiten (Alzheimer, Krebs) - Universität Stanford
- besseres Verständnis für Faltung von Proteinen $\rightarrow$ Mechanismen erkennen
- eines der ersten Projekte die auch GPU's miteinbeziehen
- erreichte am 20. Mai 2016 eine Rechenleistung von über 100 PetaFLOPS
- CUDA-Technik und OpenCL
- seit Projektbeginn insgesamt 205 Publikationen (Stand 31. Dezember 2018)

Anwendung: Deep Learning



https://datawarrior.wordpress.com/2017/10/31/interpretability-of-neural-networks/



https://www.nvidia.com/de-de/about-nvidia/ai-computing/

Danke für eure Aufmerksamkeit!