Vorlesung 12

Aufgabe 1

Select three slides from the lecture, research more about the topics, and report on them.

1. „General-Purpose Computing on Graphics Processing Units“

Die Idee ist, dass man GPU nicht für Graphics nutzt, sondern für rechenintensive Anwendungen. Wieso?

1) GPUs haben deutlich mehr Kerne (bzw. Streaming-Prozessoren/ Shader-Einheiten) als CPUs, deswegen sind sie so nützlich wenns darum geht die Aufgaben zu lösen, die man in viele kleine unabhängige parallele Arbeitsschritte aufteilen kann.
2) GPUs sind auf arithmetische Rechenoperationen spezialisiert (insbesondere Gleitkommaoperationen die bei vielen wissenschaftlichen und KI-Anwendungen wichtig sind)

Damit man überhaupt seine “normalen” Aufgaben (die nichts direkt mit Grafikrendering zu tun haben) auf einer GPU rechnen lassen kann, gibt es verschiedene APIs und Bibliotheken, die auf der Folie aufgelistet sind.

- CUDA = entwickelt von NVIDIA, läuft nur auf NVIDIA-GPUs, in Machine Learning verbreitet, ermöglicht GPU-spezifischen Code in C-ähnlichen Sprache zu schreiben und GPU direkt zu steuern
- OpenCL = offener Standard, funktioniert auf GPUs verschiedener Hersteller
- Metal = von Apple für Apple-Hardware entwickelt
- OpenACC = Standard für paralleles Programmieren, der auf Direktiven basiert (ähnlich wie OpenMP), Compiler übernimmt die Parallelisierung auf GPU

2. Message Passing Interface (MPI)

MPI lässt die Nodes in großen Clustern kommunizieren um deren Ergebnisse zusammenzuführen

Die Idee ist, dass die Nodes in distributed-memory-systems folgendes haben:

- Eine oder mehrere CPUs (oder CPU-Kerne) die Berechnungen machen
- Einen eigenen RAM, der nur von diesem Node genutzt wird
- Eine Verbindung zu einem Netzwerk

Der Speicher ist also verteilt. Die Kommunikation läuft über ein Interconnection Network, wobei dieser Austausch minimiert werden soll. Egal wie schnell Netzwerk sit, ist lokale Berechnung immer schneller. Und MPI ist nicht nur für Senden/empfangen da, sondern auch für Kollektivoperationen (wie Synchronisation von Prozessen)

3. NUMA

Im Punkt davor ging’s um distributed-memory-systems, bei NUMA geht es um shared-memory-systems.

Bei UMA ist der RAM an einem zentralen Ort angebunden und alle Kerne haben denselben Zugriff darauf – die Latenz und Bandbreite sind für alle gleich. Das Problem ist auch ein einzelner Speicherbus und viele Zugriffe überlasten inh. Und NUMA verteilt den Speicher, jede CPU hat eigenen lokalen RAM - das ist dann ein Node in NUMA. Die Nodes in diesem shared-memory-system sind durch Interconnect-Links verbunden, das bedeutet CPU kann auch auf den RAM eines anderen Sockets zugreifen. OS und Hardware abstrahieren so, dass jeder Prozessor den Gesamtspeicher sieht, auch wenn er physisch verteilt ist