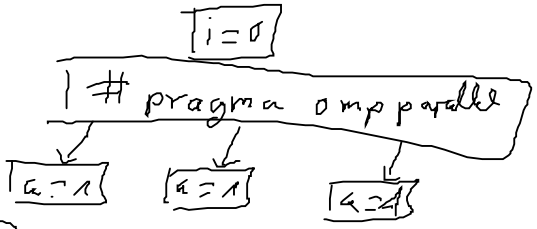
Realisierung von parallel laufenden Programmen

multithreading (shared memory)

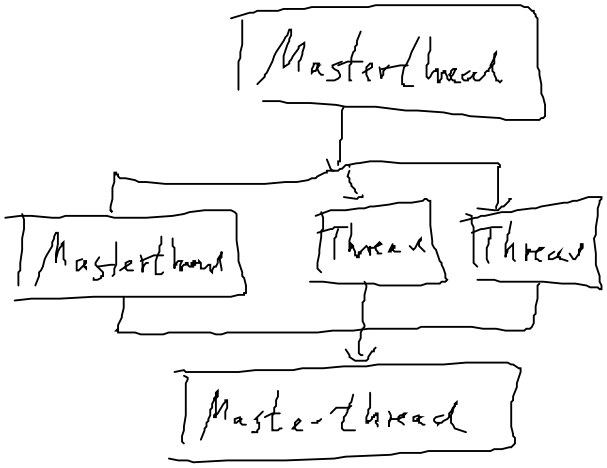
- per Hand (Linux, z.B. pThread Library), nicht zu empfehlen für wiss. Anwendung, benötigt viel Handarbeit und viel Erfahrung.
- OpenMP, Compilerdirektiven, die Schleifen automatisiert parallelisieren (im Prinzip für Dummy geeignet, Performance nicht so gut ohne zusätzliche Überlegungen)

Sehr leicht zu Programmieren

Fork Join Modell



Bei komplexen Problemen Synchronisation.



Auch tw. SIMD im Hintergrund (complex)

- OpenCL: Standard der die Maschine abstrahiert
  => Prozessoren und Grafikkarten auf die gleiche Weise programmieren

Nachteil programmiert sich wie OpenGL

Sehr ungewohnter Zugang für viele

Einstiegshürde

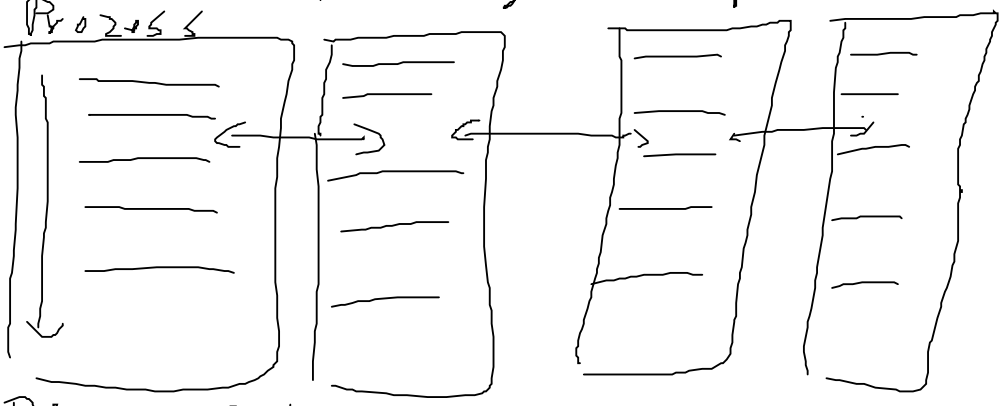
- Bibliotheken, die im Hintergrund parallel sind
  z. B. ViennaCL, --

Wichtige Bemerkung: Bibliothek müssen thread safe sein, damit diese verwendet werden können.

Mehrere Prozesse

- parallel schreiben, mit Interprozessen (z. B. Pipes) meist zu wenig portabel.
- MPI (Industriestandard), Standard um Nachrichten/Daten zu transferieren (z. B. Netzwerk)

Konzept zum programmieren:
Prozesse laufen im Gleichschritt parallel:



Definierte Punkte an denen etwas gesendet oder empfangen wird.

Wichtig zw. Senden und Empfangen etwas sinnvolles tun.

- Library, die MPI verwendet

Generell: Ausgereifte numerische Bibliotheken,
die flexibel verwendet werden können

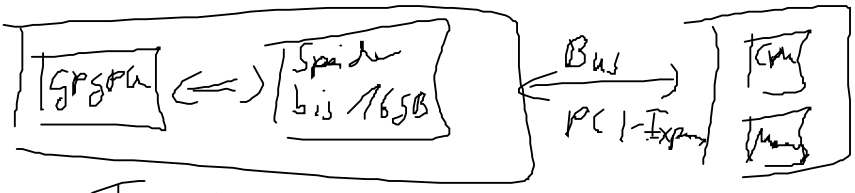
GPU; Spezielle Hardware

Plattform: Wichtig insbesondere Spezialhardware
für GPU Rechner für numerische Anwendungen.
oder Xeon Phi

Hauptunterschied zur CPU

sehr viele Kerne, spezialisiert auf bestimmte Operationen
geringer Takt!

Beispiel: Pascal Architektur von Nvidia
Chip für Supercomputer (GP100)
56 x SM (multiprozessor): 32 double precission
und 64 single precission
(Bei Grafikkarten 128 single precission
und 4 double precision)
Eine SM teilt sich ein L1 Cache für Textur (Array)
und 64 KB Shared Memory
Gemeinsam L2 Cache 4096 KB



Transfer zur CPU langsam
Rechnung auf GPU dürfen nur am Anfang und
Ende (oder selten) mit CPU interagieren (sonst langsam)

Wichtig: Genauigkeit der Fließkommaoperationen geringer als CPU (Situation bei GPGPU Chips besser)

(Generell andere Datentyp, etw. kein Integer)

Wegen massiver Parallelität ⇒ Programmierung und Befehle anders.

Viele Recheneinheiten führen die selben Operationen aus auf großen Datenblöcken. (Art der Art von Befehlen)

Damit die Shader nicht aus dem Takt kommen gibt es kein Sprünge (z.B. CPU)

CPU:

```
if (Bedingung) {
    Code mit vielen Zeilen
}
```

Falls Bedingung erfüllt ist Sprung

GPU:

```
if (Bedingung) {
    Code mit vielen Zeilen
}
```

⇐ Falls Bedingung erfüllt ist, wird Code trotzdem ausgeführt, aber es ändert nichts!

↑ Vorteil: Synchronisierung nicht so schwierig.

Auch hier gibt es Möglichkeit zur Synchronisierung, aber auch gibt vermeiden.

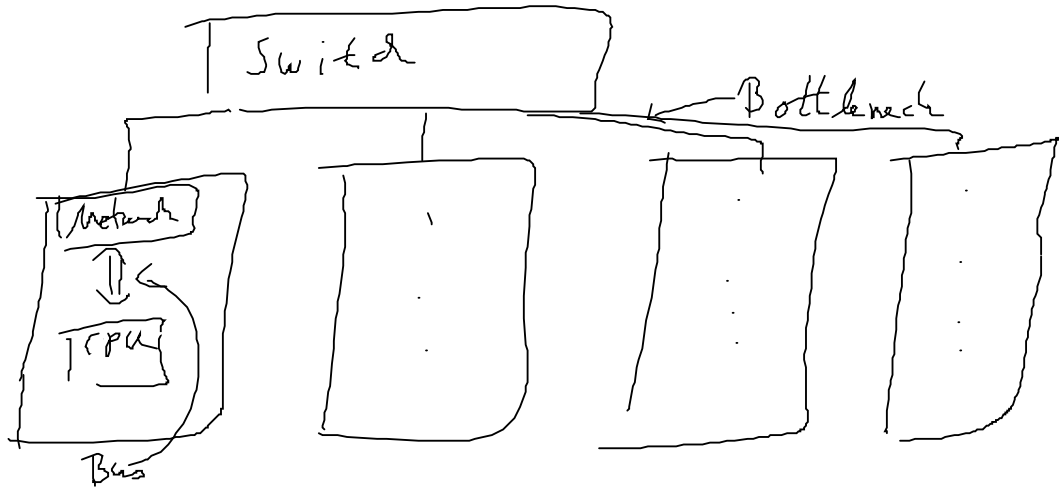
Programmierung:

- Open GL (Eigentlich für Grafik, nicht wirklich zu empfehlen)

- CUDA (nur nvidia support, ist der weitverbreitet)
- Open CL (Hersteller unabhängig)

oder Library verwenden.

Mehrere Rechner im Cluster



Bei größeren Anwendungen Kopplung verschiedener Rechner.

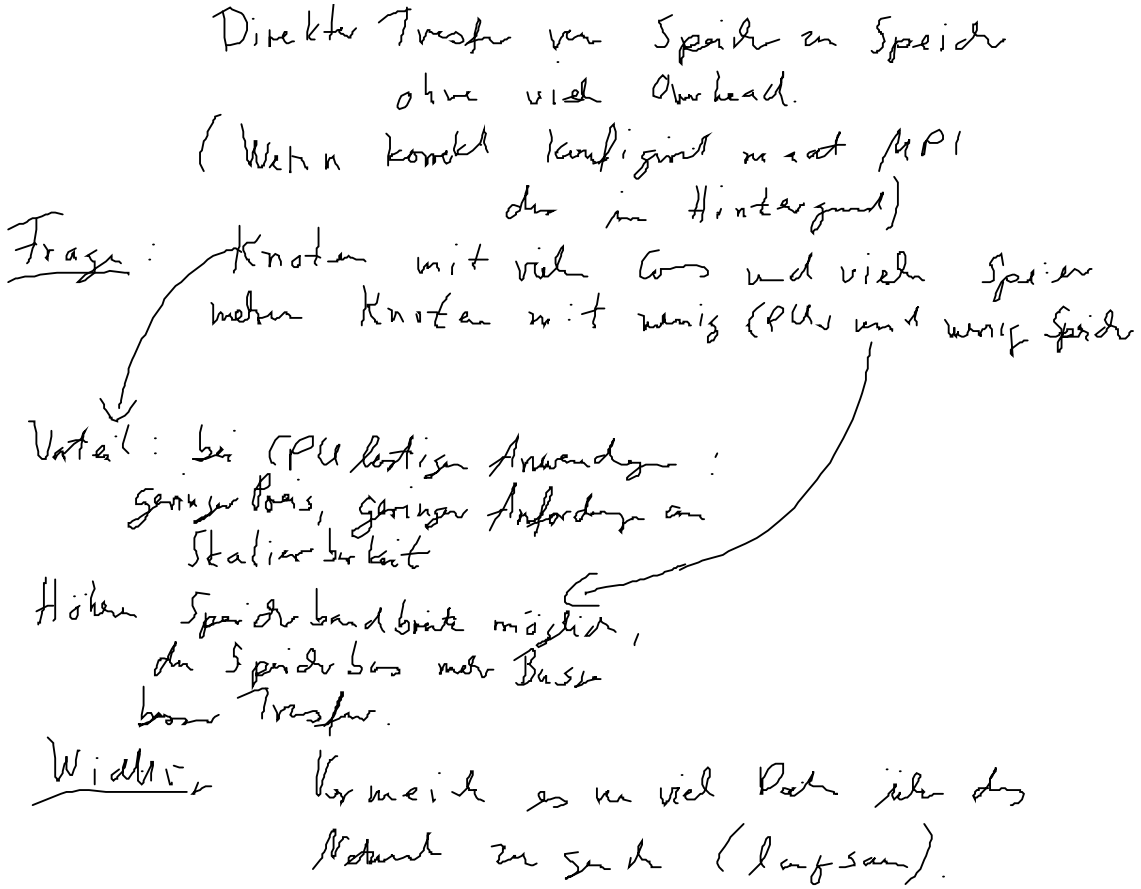
mehrere Möglichkeiten
- 1 Prozess je Rechner mit mehreren Threads (Open MP)
- mehrere Prozesse je Rechner mit single Threads

Kommunikation in der Regel mit MPI Library (verschiedene Implementierungen, Open MPI, mpich2)

Vernetzung: Herkömmlich Gigabit Ethernet IP. (Hardware zu langsam, Ethernet, IP Protokoll zu hoher Overhead)

richtige Cluster => oft: Infiniband (z.B. 20Gbit und aufwärts)

RDMA Protokolle (Remote Direct Memory Access)


Direkter Transfer von Speicher zu Speicher
ohne viel Overhead.
(Wenn korrekt konfiguriert meist MPI
das im Hintergrund)
Frage: Knoten mit vielen Cores und vielen Speicher
mehrere Knoten mit wenig CPUs und wenig Speicher
Vorteil: bei CPU lastigen Anwendungen:
Genauer Basis, geringer Anforderungen an
Skalierbarkeit
Höhere Speicherbandbreite möglich,
da Speicherbus mehr Busse
besser Transfer.
Widrig: Vermeidet es zu viel Daten über das
Netzwerk zu senden (langsam).