Fortsetzung Rechnerarchitekturen

GPU: Spezielle Hardware

Wichtig inzwischen spezial GPU um zu Rechnen,
oder XeonPhi

Hauptunterschied zu CPU

sehr viele Kerne, spezialisiert auf bestimmte Operation;
geringer Takt: Beispiel Pascal Architekten von Nvidia
chip für Supercomputer (GP100)

56 x SM (Multiprozessor): 32 double precision
und 64 single precision
shader

(Bei Grafikkarte 128 singleprecision
4 double precision)

= 1792 double precision und 3584 singleprecision.

Ein SM teilt sich L1 Cache für Texter (Arrays)
und 64 kB Shared Memory

Gemeinsam L2 Cache 4096 kB

Transfer zu CPU langsam

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

Widtis: Präzision von Fließkommazahl geringer als
bei CPU (Situation besser bei GPGPU chips)
(Generell an den Datentypen, z.B Zeichtegen)

Wegen massiver Parallität => Programmierung und Befehle
anders

Viele Recheneinheiten führen dieselben Operation
durch aber an anderen Daten parallel. (Andere Arten
von Befehlen)

Neuer Trend: Auch Rechenkerne speziell optimiert für
Tensorberechnung auf der Grafikkarte
(Nvidia oder als externe Beschleunigung (google
in den Cloud Rechenzentren)

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

CPU

```
if (Bedingung) {   <= Falls die Bedingung unwahr ist Sprung
    Code mit vielen Befehlen
}
```

GPU:

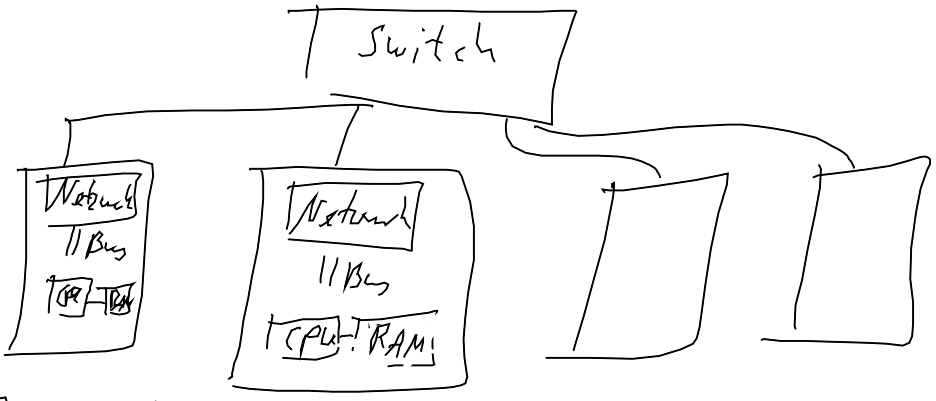
```
if (Bedingung) {   Falls Bedingung unwahr
    Code mit vielen Befehlen   <= Code wird trotzdem ausgeführt
}
```

Ein Flag verhindert das Daten verändert werden.

Vorteil: Synchronisierung nicht so schwierig

Auch hier gibt es Möglichkeit zur Synchronisierung
aber auch gilt es diese zu vermeiden.

Mehrere Rechner im Cluster



Bei größeren Anwendungen Kopplung verschiedener Rechner:
mehrere Möglichkeiten
- 1 Prozess je Rechner mit vielen Threads (OMP)
- mehrere Prozesse je Rechner mit single Thread

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

Vernetzung:
- Herkömmlich Gigabit Ethernet
(Übertragung zu langsam, Ethernet, IP Protokoll zu hoher Overhead)

richtige Cluster => oft: Infiniband (z.B. 20 Gbit und mehr)
RDMA Protokoll
(Remote Direct Memory Access)
Direkter Transfer von Speicher zu Speicher ohne viel Overhead
(Wenn korrekt konfiguriert macht MPI den Hintergrund)

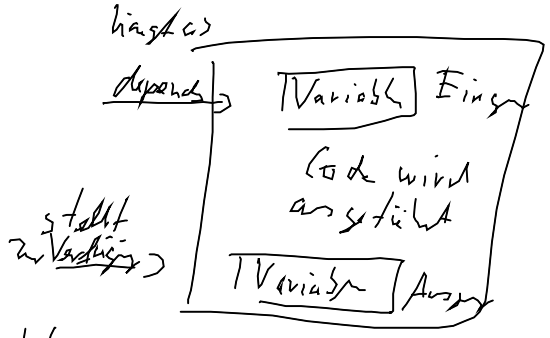
Frage:
Knoten mit viel CPUs und viel Speicher
versus mehr Knoten mit wenig CPUs und wenig Speicher)

Vorteil bei CPU lastigen Anwendungen:
geringer Preis, geringer Anforderung
an Skalierbarkeit

Höhere Speicherbandbreite möglich, da Speicherbus Flaschenhals
mehr Bus besser Transfer

Wichtig: Vermeide es zu viel Daten über das Netzwerk
zu schicken langsam.

Neue Programmiermöglichkeit Rechenoperation in einzelnen
Aufgaben zerlegen in tasks:



Problem in einzelne Task zerlegen.
Anwender sowohl CPU wie auch GPU oder Transferops.

Frameworks
- OpenMP kann das
- Tensorframework z.B. tensorflow
für Tensor spezialchip, Grafikkarte
etc.

I.3 (Nicht-) Lineare Algebra für parallel programming

Vektorisierung der Problems

Warum Vektorisierung? Viele Hardwarearchitekturen,
Software-Bibliotheken etc. stehen zur Verfügung,
um optimiert Matrixmultiplikation und
Vektoroperation durchzuführen.

Beispiel für typisch lineare Probleme (formuliert):

Lösung linearer Gleichungen:

$$\underline{\underline{A}} \cdot \underline{x} = \underline{b}$$

$\underline{x}$: Vektor, $\underline{\underline{A}}$: Matrix

Lösung von zeitabhängigen Problemen

$$\partial_t \underline{v} = \underline{\underline{M}} \cdot \underline{v}$$

$\underline{\underline{M}}$: Matrix, $\underline{v}$: Vektor

Eigenwertprobleme

$$\underline{\underline{A}} \cdot \underline{x} = E\,\underline{x}$$

$\underline{\underline{A}}$: Matrix, E: Zahl, $\underline{x}$: Vektor

(und weitere Probleme spectral transform, singular value decomposition)

Beispiel für physikalische Probleme

Schrödinger-Gl.

$$H|\psi\rangle = E|\psi\rangle \Rightarrow$$

H: Matrix, $|\psi\rangle$: Vektor, E: Zahl

Wichtig ist es eine geeignete Basis zu finden (s. Finite Differenzen und Quantenchemie)

$$|\psi\rangle = \sum_i c_i |i\rangle$$

$|i\rangle$: Spaltenvektor

Liouville von Neumann Gleichung

$$\partial_t \rho = -\frac{i}{\hbar}[H, \rho]_-$$

(Hier ist die Dichtematrix ein Vektor (z.B. $\langle i|\rho|j\rangle = v[i + Nxj]$ und $-\frac{i}{\hbar}[H, \cdot]$ wird auf die Matrix abgebildet.

Helmholtz Gleichung

$$\Delta \underline{A}(\underline{r},\omega) + \frac{\omega^2}{c^2}\varepsilon(\omega)\underline{A}(\underline{r},\omega) = 0 \quad + \text{Randbed.}$$

Matrix — Vektor

Besondere Form des Eigenwertproblems (nicht hermitesch)

z. B. Wichtig für quantisierung Lichtfeld.

Entwicklung $\underline{A}(\underline{r},\omega)$ in Sätze oder Finite Elemente

Beispiel für Vektor und Matrixformate

Sequentielle Formate

Vektorformat: Vektor ist ein array von Fließkommazahlen:

Speicher | z[0] | z[1] | z[2] | z[3] | ..... | z[4] |

Abbildung Speicher auf single [32 bit] oder double [64 bit] precision Fließkommazahlen.

z. B. IEEE 754

$$X = S \; m \; 2^e \quad \leftarrow \text{Exponent}$$

Vorzeichen 1 bit — Mantisse (23 bit single, 52 bit double) — (8 bit single $-126 \le e \le 127$, 11 bit double $-1022 \le e \le 1023$)

Wichtig Genauigkeit beachten (Multiplikation kennt Addition sehr großer und sehr kleiner Zahlen schwierig)

Bei komplexen Zahlen Speicherung z[i] als zwei Werten. real und imaginär Teil (Ist ein Glück schon fertig implementiert ...)

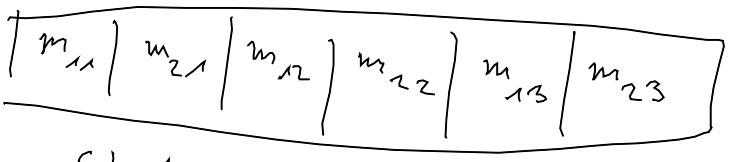
Matrixformate

Lineare Möglichkeit (wie BLAS)

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{pmatrix}$$

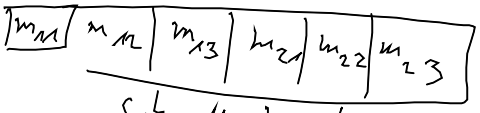
Zwei Möglichkeiten das in Speicher abzulegen:

1) Spaltenzuerst (Column major) (z.B. Fortran native) BLAS



Schnell bei Matrix vektor multiplikation $\underline{\underline{M}}^T \cdot \vec{x}$, wenn schnelle Spaltenzugriff (langsam)

2) Reihe zuerst (Row major) (z.B. Standard bei C, C++)



Schnell bei transponiert Matrix Vektor $M^T x$

(Bei den meisten Anwendungen/Algorithmen wird meist nur die Anwendung der Matrix oder der transponierten Matrix gebraucht)

Wichtig, die innere Schleife muss immer die am meisten variierende Größe ansprechen sonst langsam!

Low-level Routine für die Matrix format finden sich im BLAS

An die Architektur optimierte lineare Algebra Pakete, bei Supercomputer von Hersteller...

Dieses Format ideal (gute Speicherausnutzung) für dünn besetzte Matrizen.

$$\begin{pmatrix} 2 & 2 & 4 \\ 1 & 0 & 7 \\ 9 & 5 & 3 \end{pmatrix}$$ ← Matrix mit wenig 0