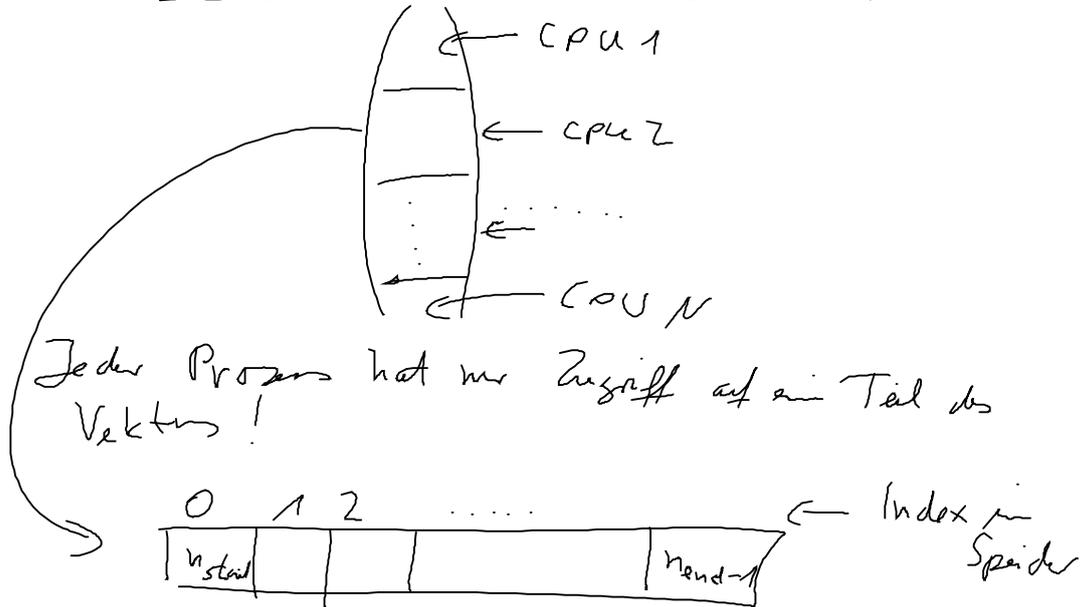


Fortsetzung Matrix/Vektoroperationen ...
Parallelisierte Berechnung von Matrizen

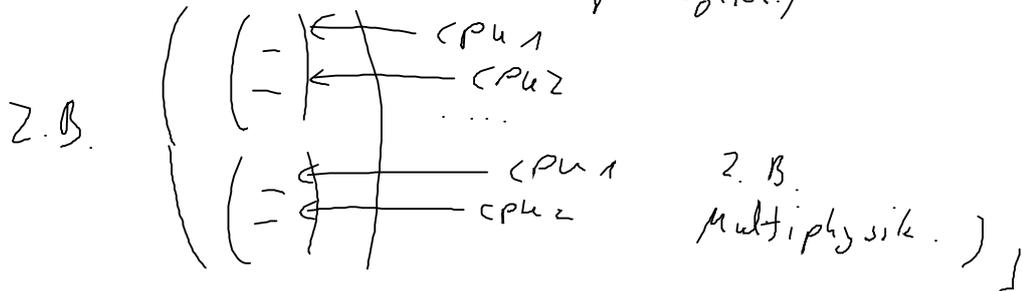
Vektoren für MPI Anwendung (Beispiel)



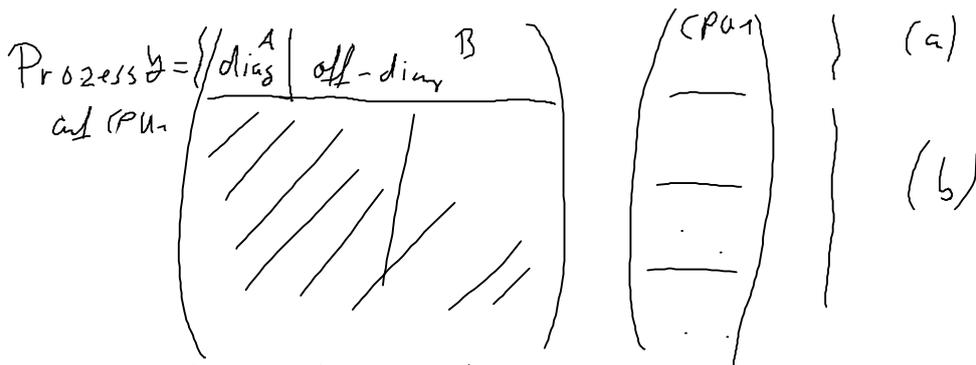
Möchte die CPU auf ein Teil des Vektors zugreifen, der ihr nicht gehört, so muss dies geschehen.
 (z.B. Speicher, Netzwerk)

Bemerkung: Vektor müssen oft während des Zugriffs
 gesichert (z. B. Petsc bei Sparse Matrix Vektoren).

(Beispiel, auch andere Verteilung möglich.)



Wie funktioniert die Matrixmultiplikation?



2.0 Petsc \hookrightarrow Vorgehen bei Matrixmultiplikation. (CPU 1)

1.) Schicke eigenen Vektor (w) an andere CPUs
(aber nur die Einträge, die diese auch brauchen (Spaltenplatz))

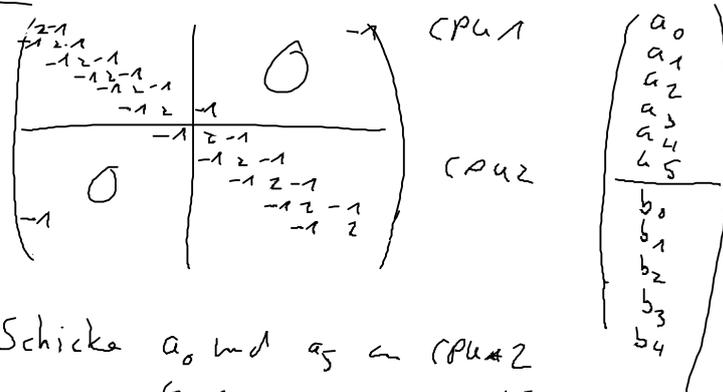
2.) Währen dessen: Berechne $\underline{x} = \underline{A} \cdot \underline{a}$ \swarrow diag \neq

3.) Empfangen des Vektoranteils von anderen CPUs \underline{b}

4.) Berechne: $\underline{y} = \underline{B} \cdot \underline{b} + \underline{x}$
 \swarrow off-diag \uparrow Nullspalten mit fort.

Die anderen CPUs machen dies entsprechend.

Beispiel



1.) Schicke a_0 und a_5 an CPU #2

2.) Berechne $\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \underline{x}$

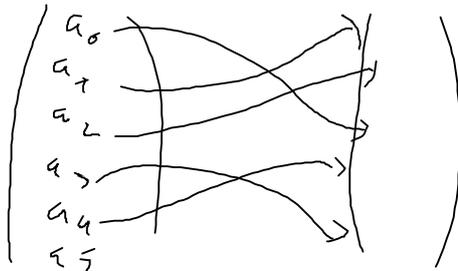
3.) Empfangen $\begin{pmatrix} b_0 \\ b_4 \end{pmatrix}$

4) Berechne $\begin{pmatrix} 0 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -1 & \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} + \underline{x} = y$

Tipps: U_b bedingt Matrix, so aufbauen, dass der off diagonale Anteil keine besteht ist:

- Besondere Anordnung von Sitten im Vektor (Nächste VL)
- Es existieren für bel. Matrizen (DGL) oder unstrukturierte Grid Bibliotheken, die die Einträge im den Vektor, so umtauschen, dass es wenig off diagonale Einträge gibt.

z.B. pernetis



Matrix free Matrixmultiplikation
(Matrix shell)

Das heißt in der Fall wird nur ein Funktion programmiert.

MatMult (vecin, vecout)

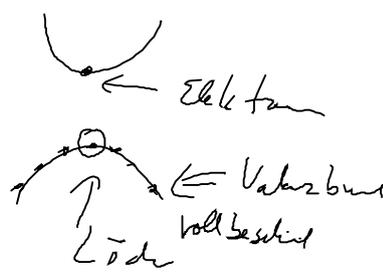
die bedeutet $\underline{v}_{out} = \underline{M} \cdot \underline{v}_{in}$

Vorteil: kein weiterer Speicher verbraucht.
• schon implementiert

Nachteil : Parallelisierte Implementierung aufwendiger.

II. 4 Lösung der Schrödinger über finite Differenz
z.B. Exziton und Trion im Halbleiter

Nur ein Beispiel wie man partielle Differentialgl. lösen kann



Elektron-Lochpaar wird z.B. optisch erzeugt.



Coulomb Wechselwirkung, führt zu gebundenen Exzitonen (Zustand wie in H-Atom)

Effektiv lässt sich ein Exziton beschreiben mit:

$$\psi(\underline{r}, \underline{R}) = \psi_{rel}(\underline{r}) \chi(\underline{R})$$

\underline{r} ist relative Koordinate
 und \underline{R} ist Schwerpunktskoordinate.

$$\underline{r} = \underline{r}_e - \underline{r}_h$$

$$\underline{R} = \frac{m_e \underline{r}_e + m_h \underline{r}_h}{M}$$

Analog zur Schrödingergl. sieht es die Wanniergl. für die Wellenfunktion der relativen Bewegung.

$$\frac{1}{m_r} = \frac{1}{m_e} + \frac{1}{m_h}$$

$$\left(-\frac{\hbar^2}{2m_r} \Delta_r + V_{eh}(|r|) \right) \psi(r) = E \psi(r)$$

Frei bewegte
de Teile

modifiziert
Coulomb Potential
zwischen den Teilen

Wenn das System dotiert ist, sieht es aus Trion...



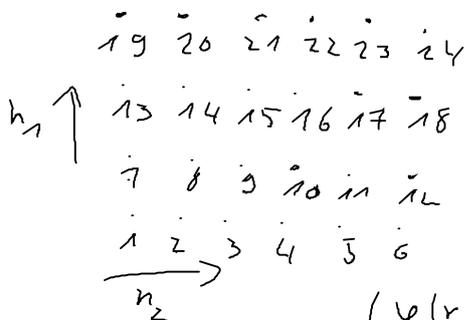
$$\left(-\frac{\hbar^2}{2m_{r1}} \Delta_{r_1} - \frac{\hbar^2}{2m_{r2}} \Delta_{r_2} - \frac{\hbar^2}{2m_h} \nabla_{r_1} \cdot \nabla_{r_2} + V_{eh}(r_1) + V_{eh}(r_2) + V_{ee}(|r_1 - r_2|) \right) \psi(r_1, r_2) = E \psi(r_1, r_2)$$

Nehmen wir an wir wollten sie die beiden Gleichungen lösen.
Nur wie löst man sie part. DGL?

- 1) Heute sehr häufig verwendetes Verfahren Finite Elemente
(sehr effektiv, da mit dem Problem angepasste
Gitter gearbeitet)
- 2) Finite Differenzen, sehr einfach zu umsetzen
(nicht unbedingt effektiv)

Hier Beispielstrategie für Finite Differenzen!

Auf einem Gitter z.B. 2 Dimension für ein Teilchen:



Wir brauchen eine Abbildung um
an ein index i in ein Array
ein Gitterpunkt zuzuordnen.
 $r[i]$, am besten in 2D
 $r[n_1, n_2]$ oder $i[n_1, n_2]$

$$|\psi\rangle_{11} = \begin{pmatrix} \psi(r[0]) \\ \psi(r[1]) \\ \vdots \end{pmatrix} =: \psi$$

$$\frac{d^n f}{dx^n} \Big|_{x=x_0} \approx \sum_{j=0}^h \delta_{n,j} f(x_j),$$

$x=x_0$ Auf der Stelle
 $\sum_{j=0}^h$ Grad Approximat
 $\delta_{n,j} f(x_j)$ Bestimmen

Wie wählen wir den Grad der Approximierten?

Große Faustregel: Höherer Grad, größere Diskretisierung, möglich \Rightarrow weniger Speicher für den Vektor M_2

Aber mehr Speicher für die Matrizen $M_2 \cdot n$ (für jede Dimension)

Wertebereiche sind Genauigkeiten im Problem (Fließkommazahl!)

Je höher Grad der Approximierten (Positive, negative Beiträge lösbar sind)