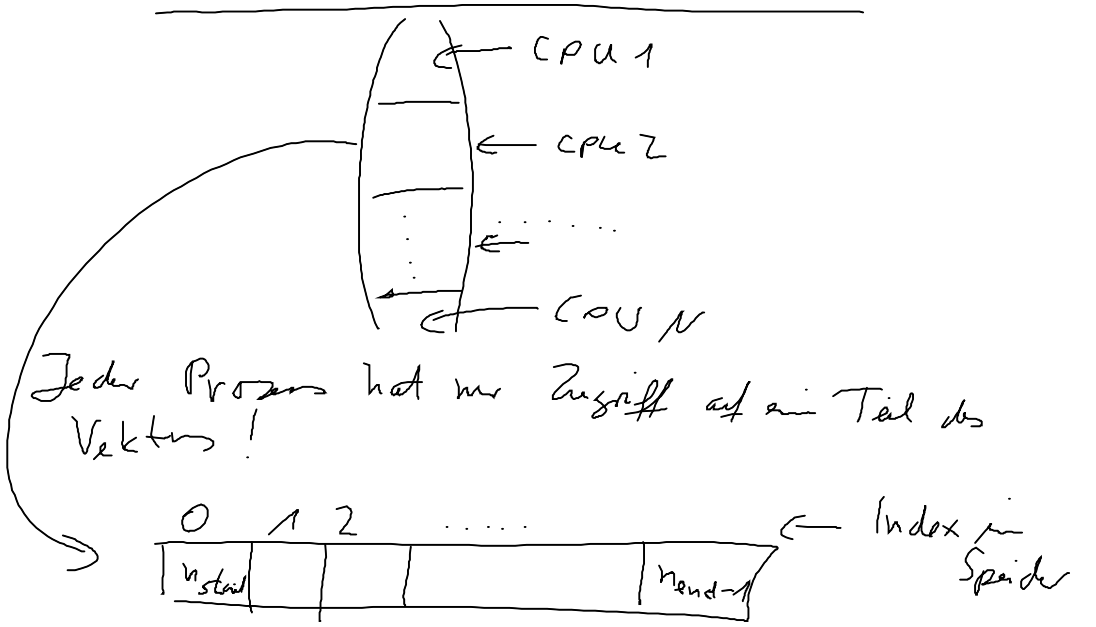
Fortsetzung Matrix/Vektoroperationen ..

Parallelisierte Berechnung von Matrizen

Vektoren für MPI Anwendung (Beispiel)



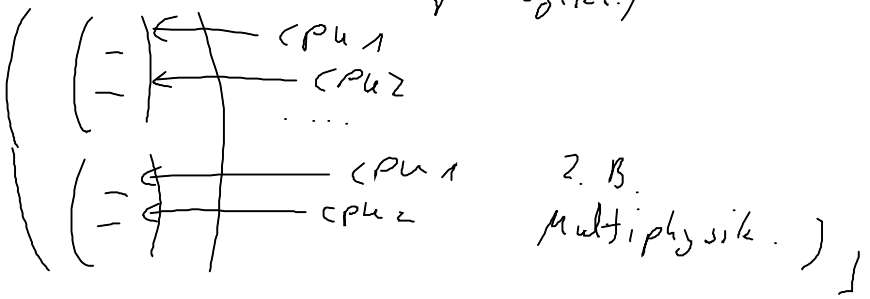
Jeder Prozess hat nur Zugriff auf ein Teil des Vektors!

Möchte die CPU auf ein Teil des Vektors zugreifen, der ihr nicht gehört, so muss dieser gescattert. (tw. Spicker, Netzwerk)

Bemerkung: Vektor müssen oft während des Zugriffs gelockt (z.B. Petsc bei Grafikkartvektoren).

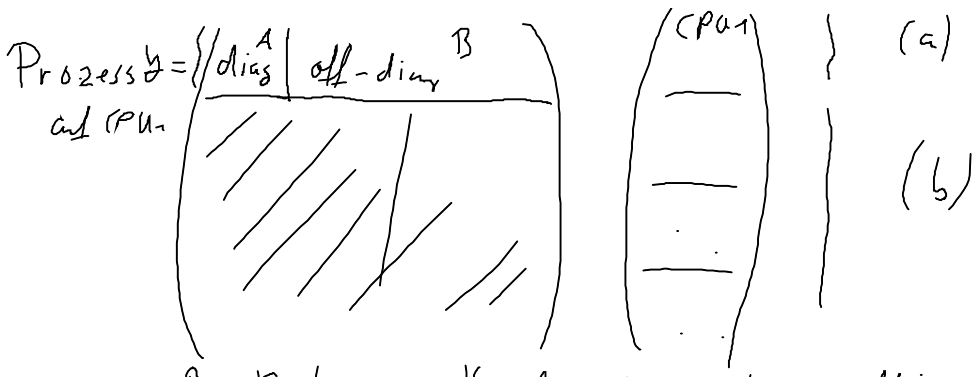
(Beispiel, auch andere Verteilung möglich.)

z.B.



z.B. Multiphysik.)

Wie funktioniert die Matrixmultiplikation?



z.B. Petsc Vorgehen bei Matrix multiplikation.

1) Schicke eigenen Vektor (a) an andere CPUs (CPU1)
(aber nur die Einträge, die diese auch brauchen (Speicherplatz))

2.) Während dessen: Berechne $\underline{x} = \underline{\underline{A}}_{\text{diag}} \cdot \underline{a}$

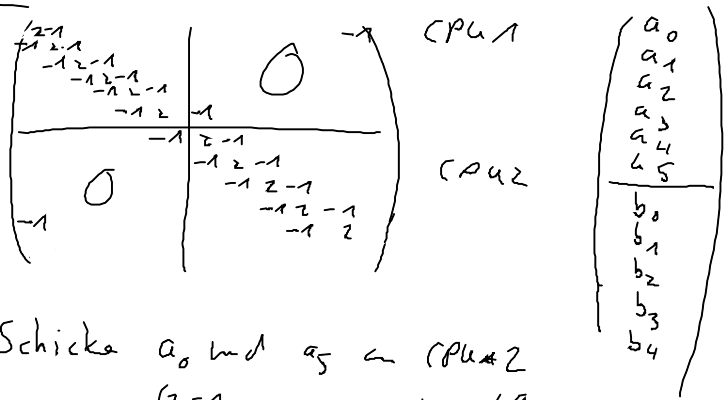
3.) Empfange den Vektoranteil von anderen CPUs $\underline{b}$

4.) Berechne: $\underline{y} = \underline{\underline{B}}_{\text{offdiag}} \cdot \underline{b} + \underline{x}$

↑ Nullspalten entfernt.

Die anderen CPUs machen das entsprechend.

Beispiel



1) Schicke a_0 und a_5 an CPU#2

2) Berechne $\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & 0 & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & 0 & & & -1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \underline{x}$

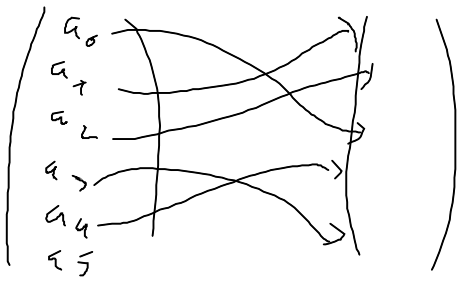
3) Empfange $\begin{pmatrix} b_0 \\ b_4 \end{pmatrix}$

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

Tipps: Um bedingt Matrix, so aufbauen, dass der Off diagonal Anteil kann besetzt ist.

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

z.B. parmetis



Matrix freie Matrixmultiplikation (Matrix shell)

Das heißt in dem Fall wird nur ein Funktion programmiert.

MatMult (vec in, vec out)

die berechnet $\underline{v}_{out} = \underline{\underline{A}} \cdot \underline{v}_{in}$

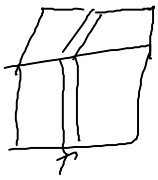
Vorteil: - kann weniger Speicher verbrauchen.
- schon implementiert

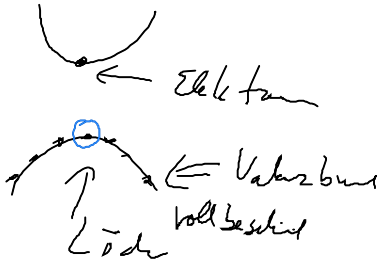
Nachteil: Parallelisierte Implementierung aufwendig.

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

Nur ein Beispiel wie man partielle Differentialgl. lösen kann

Halbleiter z.B Quantenwelle





Elektron-Loch-Paar wird z.B optisch erzeugt.

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

Effektiv läßt sich ein Exziton beschreiben mit

$$\Psi(\underline{r},\underline{R}) = \varphi_{rel}(\underline{r})\,\Sigma(\underline{R})$$

$\underline{r}$ ist relativ 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. gilt es die Wanniergl. für die Wellenfkt der relativen Bewegung.

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

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

Freie Bewegung der Teilchen

modifizierte Coulomb Potential zwischen den Teilchen

Wenn das System dotiert ist, sieht es aus wie Trionen.

$$\left(-\frac{\hbar^2}{2m_{r_1}}\Delta_{r_1} - \frac{\hbar^2}{2m_{r_2}}\Delta_{r_2} - \frac{\hbar^2}{2m_h}\nabla_{r_1}\cdot\nabla_{r_2} + V_{eh}(r_1) + V_{eh}(r_2) + V_{ee}(\underline{r}_1 - \underline{r}_2)\right)\psi(\underline{r}_1, \underline{r}_2) = E\psi(\underline{r}_1, \underline{r}_2)$$

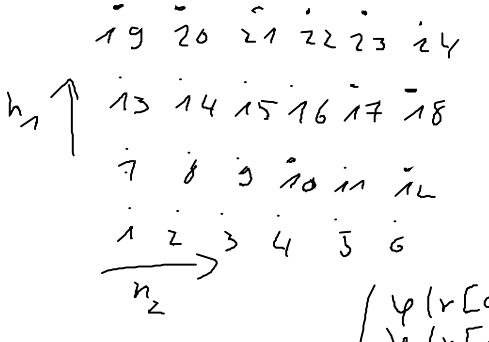
Nehmen wir an wir wollten nur die beiden Gleichungen lösen. Nun wie löst man eine part. DGL?

1) Heute sehr häufig verwendetes Verfahren Finite Elemente (sehr effektiv, da mit dem Problem angepasstem Gitter gearbeitet)

2) Finite Differenzen, sehr einfach zu programmieren. (nicht unbedingt effektiv)

Hier Beispielstrategie für Finite Differenzen!

Auf ein Gitter z.B. 2 Dimensionen für ein Teilchen:



Wir brauchen eine Abbildung um aus einem Index i in einem Array einen Gitterpunkt zu erhalten.

$r[i]$, am besten in 2D

$r[n_1, n_2]$ oder $i[n_1, n_2]$

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

$\begin{pmatrix} \vdots \\ \varphi(r[n]) \end{pmatrix}$

Gesucht Matrix H_m für unser Problem

$\underline{\underline{H}}_m \cdot \underline{\varphi} = E \underline{\varphi}$

Funktion in Hamiltonoperator

z.B

$V_{eh}(\underline{r}) \varphi(\underline{r}) \longrightarrow$ $\begin{pmatrix} V_{eh}(r[0]) & & 0 \\ & \ddots & \\ 0 & & V_{eh}(r[n]) \end{pmatrix} \cdot \underline{\varphi}$

Differentialoperator

Was passiert z.B mit

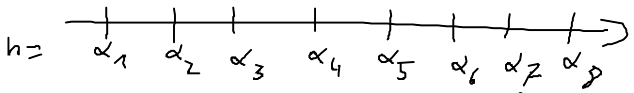
$-\frac{\hbar^2}{2m_e} \Delta_r$ oder $-\frac{\hbar^2}{2m_h} \nabla_{r_1} \cdot \nabla_{r_2}$

auf dem Gitter.

$\Rightarrow$ Finite Differenzen (guter übersichtsartikel: Acta Numerica, 203 - 207 (1995) Fornberg, Sloan)

Was ist die Idee?

Nehmen wir haben ein Gitter



Das sind die Stellen der Gitterpunkte, diese müssen nicht equidistant sein!

Wir wollen die n-te Ableitung der Funktion auf dem Gitter, mit Hilfe der Funktionswerte benachbarter Gitterpunkte berechnen.

$$\left.\frac{d^n f}{dx^n}\right|_{x=x_0} \approx \sum_{\nu=0}^{n} \delta_{n,\nu}^{m} f(x_\nu),$$

$x=x_0$: Auf der Stelle; n: Grad Approximant; $\delta_{n,\nu}^{m}$: Bestimmen.

Wie wählen wir den Grad der Approximation?

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

Aber mehr Speicher für die Matrix $N_e \cdot n$ (für jede Dimension).

Wertvoll sind Genauigkeiten in Prozent (Fließkommazahlen!)

Bei höheren Graden der Approximation (Positive, negative Beträge löschen sich aus)