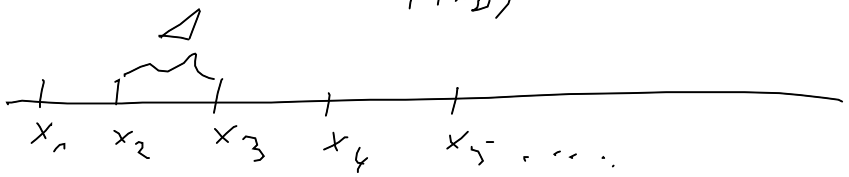
Letzte Vorlesung:
Format für dicht besetzte Matrizen.
Jetzt: Format für dünn besetzte Matrizen

*Problem: (Sehr) oft sind es dünnbesetzte Matrizen

Ein Beispiel

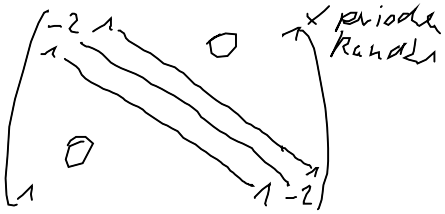
$\partial_x^2 \psi(x)$

Wir diskretisieren $\psi(x_n)$



$\partial_x^2 \psi(x) \Longrightarrow \psi(x_{n-1}) - 2\psi(x_n) + \psi(x_{n+1})$

Die zugehörige Matrix hat die Form



Vektor der Länge N

Also von N^2 Einträgen sind nur $3N$ besetzt! $\frac{3N}{N^2} = \frac{3}{N} \leftarrow$ Bei großen Gitter beliebig schlecht!

Das naive Matrixformat ist sehr schlecht bei großen Gitter!

Anderes Beispiel $V(x_n)\psi(x_n) \longrightarrow \begin{pmatrix} V(x_1) & & 0 \\ & \ddots & \\ 0 & & V(x_n) \end{pmatrix}$

(Literatur linalg book netlib.org)

Dünn besetzte Matrizen

Naivste Format

Ein Array mit Fließkommazahlen $v[\,]$

Arrays mit ganzen Zahlen $i[\,]$ $j[\,]$

Beispiel: $i = 0\ 1\ 2\ 3$

$$j = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{pmatrix} 2 & 6 & \\ & & 8 \\ & 7 & \end{pmatrix}$$

$v = \{2, 6, 8, 7\}$
$i = \{0, 2, 3, 1\}$
$j = \{0, 0, 2, 3\}$

funktioniert zwar, ist aber ineffektiv!

Compressed Row Storage Format aka Yale sparse format, general sparse AIJ format

Array val[] speichert die Elemente die nicht null sind

Array col_ind[] enthält die Spalten indizes

Array row_ptr[] enthält den Index, in den beiden anderen Arrays, bei dem die Reihe beginnt.

Beispiel (von netlib.org):

$$M = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val = {10, -2, 3, 9, 3, 7, 8, 7, 3 ..., 9, 13, 4, 2, -1}
col_ind = {0, 4, 0, 1, 5, 1, 2, 3, 0 ... 4, 5, 1, 4, 5}
row_ptr = {0, 2, 5, 8, 12, 16, 19}

Vorteil Speicherplatzersparnis, Nachteil: schlechtes memory access pattern Vektor

Compressed Colum Storage:

das gleiche mit vertauschter Reihe und Spalte (z.B. Transponierte zu bauen)

Weitere Formate für spezielle Anwendungen existieren.

Für die typischen Formate ex. opt. Fassung für Matrixmultiplikation.

Tipps: Zusammenbau der sparse Matrix aufwendig.

In der Praxis muss man vor der Speicherallozierung der Struktur die Elemente abzählen. Da die Anpassung der Arrays größen ineffektiv ist.

Blockmatrizen: (z.B. als multiphysik Problem)

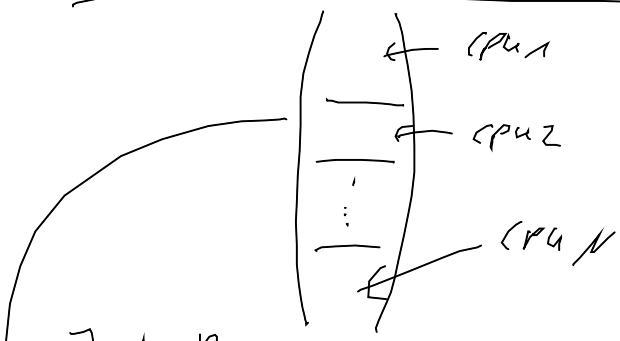
$$\begin{pmatrix} A_{aa} & A_{ab} & A_{ac} \\ A_{ba} & A_{bb} & A_{bc} \\ A_{ca} & A_{cb} & A_{cc} \end{pmatrix}$$

Zur Möglichkeit
1) monolithische Matrix
oder
2) verschaltete Matrizen

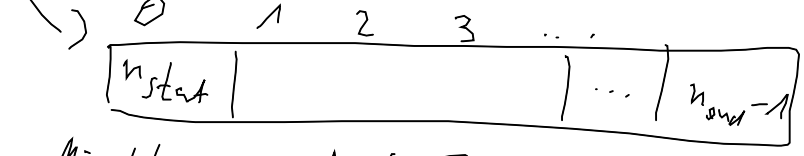
Es kann organisiertes Vorteil haben (verschiedene Gruppen)
oder aber auch Vorteil in Algorithmen,
daher ist die andere Speicherung möglich.

Parallele Berechnung der Matrizen

Zunächst Vektoren für MPI (Beispiel)



Jeder Prozess hat nur Zugriff auf den Vektorteil der ihm gehört! (Unterschied zu Open MP)

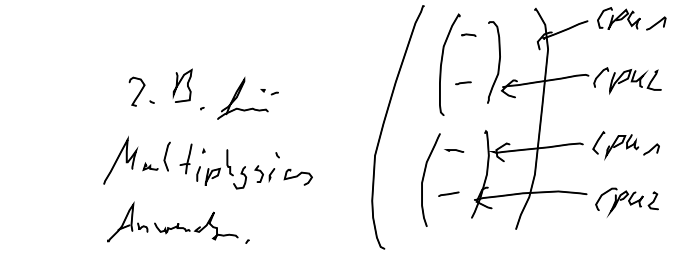


Möchte man auf ein Teil des Vektors zugreifen, der mir nicht gehört, so muss dieser gescattert werden (Stichwort Gather, Scatter) übertragen werden.

Bemerkung: Vektor müssen während der Zugriffs gelockt werden (bei Petsc z.B. ist es auf der Festplatte, muss der Vektor übertragen werden) (über Speicher, Netzwerk etc.)

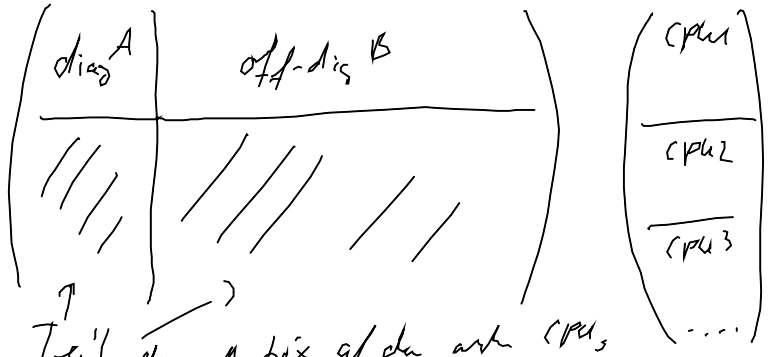
Alternativ für geblockte Anwendung:

z.B. bei Multiplyssics Anwendg.



Wie funktioniert die Matrixmultiplikation bei Parallelisierung?

Prozess auf CPU1



Teil der Matrix auf der anderen CPUs

z.B. Vorgehen von Petsc bei Matrixmultiplikation (CPU1)

1) Schicke eigene Vektoren an anderen CPUs (aber nur die Einträge, die gebraucht werden, welche Spalten sind von Nullen verschieden um den Transfer zu minimieren!)

2.) Während dessen: Berechne $\underline{x} = \underline{\underline{A}} \cdot \underline{b}$

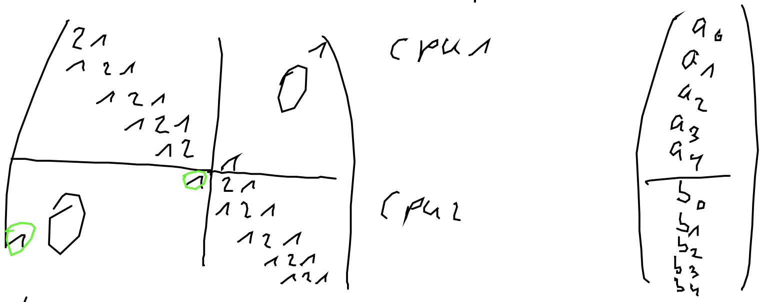
3) Empfange die Vektorteile von den anderen CPUs $\underline{b}$

4) Berechne $\underline{y} = \underline{\underline{B}} \cdot \underline{b} + \underline{x}$

Ergebnis, Nullen Spalten entfernt.

Die anderen CPUs machen das entsprechend.

Beispiel:



1) Schicke a_0 und a_4 an CPU 2

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

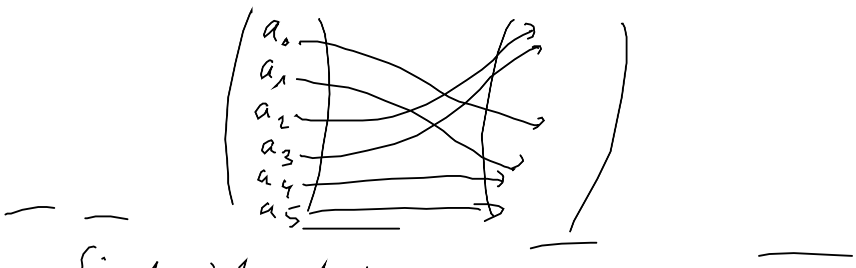
3) Empfange $\begin{pmatrix} b_0 \\ b_4 \end{pmatrix}$

4) Berechne $\begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_4 \end{pmatrix} + \underline{x} = \ldots$

Tipps: Unbedingt Matrix so aufbauen, dass der offdiagonale Part kaum besetzt ist.

- Besondere Anordnung von Gittern im Speicher (Nächste VL)
- Es ex. für bel. Matrizen (DGL) oder unstrukturierte Gitter, Bibliotheken, die die Einträge im Vektor so umsortieren, dass es nur kleine offdiagonal Einträge gibt

z.B. parmetis (auch im Petsc eingebaut)



Sind viele Matrixelemente gleich oder lässt sich die Matrix schnell berechnen (z.B. bestimmte Finite Elemente Methode)

Das heisst man implementiert eine Funktion mit

Matrix Shell Mat Mult (vecin, vecout)

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

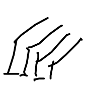
Dies ist entweder schneller als die Verwendung einer Matrix, schon implementiert, braucht weniger Speicher

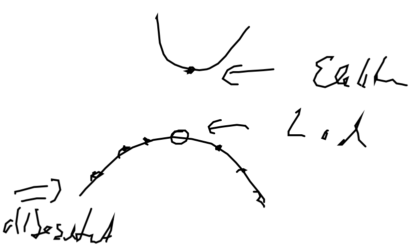
Nachteil: Parallelisierte Implementierung aufwendiger

II.4 Lösung der Schrödingergleichung über finite Differenzen

z.B. für Exziton und Trion, Biexziton

Nun ein Beispiel aus der Physik wie man partielle Differentialgl. anwenden kann:

Halbleiter z.B. Quantenwell 



Elektron-Loch paar wird z.B optisch erzeugt

Coulombwechselwirkung führt zu einem Exziton, ein gebundener Zustand wie beim Wasserstoffproblem.