

Parallele Algorithmen für \mathcal{H} -Matrizen

Ronald Kriemann

Max-Planck-Institut für Mathematik in den Naturwissenschaften
Leipzig

15.4.2005



Überblick

1 Einführung



Überblick

1 Einführung

2 \mathcal{H} -Matrizen



Überblick

- 1 Einführung
- 2 \mathcal{H} -Matrizen
- 3 Parallele \mathcal{H} -Algebra



Überblick

- 1 Einführung
- 2 \mathcal{H} -Matrizen
- 3 Parallele \mathcal{H} -Algebra
- 4 Bemerkungen



Überblick

- 1 Einführung
- 2 \mathcal{H} -Matrizen
- 3 Parallele \mathcal{H} -Algebra
- 4 Bemerkungen



Problemstellung

Die numerische Behandlung von **Differential-** und **Integralgleichungen** führt häufig zu linearen Gleichungssystemen

$$A \cdot x = y$$

mit zu bestimmendem Vektor x der Dimension n .



Problemstellung

Die numerische Behandlung von **Differential-** und **Integralgleichungen** führt häufig zu linearen Gleichungssystemen

$$A \cdot x = y$$

mit zu bestimmendem Vektor x der Dimension n .

Lösungswege

Gauß-Elimination oder LU-Zerlegung:

- Aufwand $\mathcal{O}(n^3)$,
- geeignet für kleine Systeme.

Iterationsverfahren:

- Beispiele: Mehrgitterverfahren, Methode der konjugierten Gradienten,
- Aufwand bestimmt durch Kosten für $A \cdot x$.



Approximation

Vollbesetzte Matrizen

Anzahl der Einträge und somit Aufwand für $A \cdot x$ ist $\mathcal{O}(n^2)$.



Approximation

Vollbesetzte Matrizen

Anzahl der Einträge und somit Aufwand für $A \cdot x$ ist $\mathcal{O}(n^2)$.

Approximative Darstellung

Durch numerische Behandlung entsteht typischerweise ein **Diskretisierungsfehler** $\varepsilon \geq 0$.

Deshalb genügt approximative Matrixdarstellung \tilde{A} mit

$$\|A - \tilde{A}\| \leq \varepsilon_1$$

Zu lösen ist damit:

$$\tilde{A}\tilde{x} = b \quad \text{mit} \quad \|x - \tilde{x}\| \leq \varepsilon_2$$



Approximation

Vollbesetzte Matrizen

Anzahl der Einträge und somit Aufwand für $A \cdot x$ ist $\mathcal{O}(n^2)$.

Approximative Darstellung

Durch numerische Behandlung entsteht typischerweise ein **Diskretisierungsfehler** $\varepsilon \geq 0$.

Deshalb genügt approximative Matrixdarstellung \tilde{A} mit

$$\|A - \tilde{A}\| \leq \varepsilon_1$$

Zu lösen ist damit:

$$\tilde{A}\tilde{x} = b \quad \text{mit} \quad \|x - \tilde{x}\| \leq \varepsilon_2$$

Approximationsverfahren

	Speicheraufwand	Algebra
Multipol-Methode:	$\mathcal{O}(n \log^\alpha n)$	MVM
Panel-Clustering:	$\mathcal{O}(n \log^\beta n)$	MVM
\mathcal{H}-Matrizen:	$\mathcal{O}(n \log^\gamma n)$	MVM, Add., Mult., Inv.



Numerik mit \mathcal{H} -Matrizen

Lösung von Gleichungssystemen

- 1 Matrix \tilde{A} wird mittels \mathcal{H} -Matrix-Algebra invertiert.
- 2 Berechnung von $\tilde{x} = \tilde{A}^{-1}b$ in **einem** Schritt.

Aufwand für schwach- und vollbesetzte Matrizen: $\mathcal{O}(n \log^2 n)$.



Numerik mit \mathcal{H} -Matrizen

Lösung von Gleichungssystemen

- 1 Matrix \tilde{A} wird mittels \mathcal{H} -Matrix-Algebra invertiert.
- 2 Berechnung von $\tilde{x} = \tilde{A}^{-1}b$ in **einem** Schritt.

Aufwand für schwach- und vollbesetzte Matrizen: $\mathcal{O}(n \log^2 n)$.

Problem

Komplexität der \mathcal{H} -Matrix-Algebra mit großen Konstanten verbunden.



Numerik mit \mathcal{H} -Matrizen

Lösung von Gleichungssystemen

- 1 Matrix \tilde{A} wird mittels \mathcal{H} -Matrix-Algebra invertiert.
- 2 Berechnung von $\tilde{x} = \tilde{A}^{-1}b$ in **einem** Schritt.

Aufwand für schwach- und vollbesetzte Matrizen: $\mathcal{O}(n \log^2 n)$.

Problem

Komplexität der \mathcal{H} -Matrix-Algebra mit großen Konstanten verbunden.

Lösung

- effizientere Algorithmen, z.B. \mathcal{H} -LU-Zerlegung anstelle \mathcal{H} -Inversion,
- effizientere Approximation, z.B. mittels \mathcal{H}^2 -Matrizen,
- **Parallelisierung** der \mathcal{H} -Matrix-Algebra



Überblick

- 1 Einführung
- 2 \mathcal{H} -Matrizen**
- 3 Parallele \mathcal{H} -Algebra
- 4 Bemerkungen



Clusterbäume

Seien I eine Indexmenge mit $|I| = n$ und $A \in \mathbb{R}^{I \times I}$. Für die Näherung \tilde{A} werden Blöcke $\tau \times \sigma \subset I \times I$ gesucht, in denen A mit niedrigem Rang approximierbar ist.

Eine Suche in $\mathcal{P}(I \times I)$ ist zu aufwändig. Deshalb erfolgt eine hierarchische Zerlegung der Indexmenge I : der **Clusterbaum**.

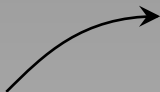


Clusterbäume

Seien I eine Indexmenge mit $|I| = n$ und $A \in \mathbb{R}^{I \times I}$. Für die Näherung \tilde{A} werden Blöcke $\tau \times \sigma \subset I \times I$ gesucht, in denen A mit niedrigem Rang approximierbar ist.

Eine Suche in $\mathcal{P}(I \times I)$ ist zu aufwändig. Deshalb erfolgt eine hierarchische Zerlegung der Indexmenge I : der **Clusterbaum**.

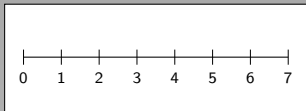
Beispiel (binäre Raumteilung)



Clusterbaum

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

Gebiet Ω



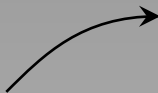
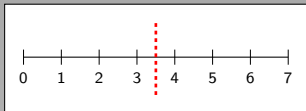
Clusterbäume

Seien I eine Indexmenge mit $|I| = n$ und $A \in \mathbb{R}^{I \times I}$. Für die Näherung \tilde{A} werden Blöcke $\tau \times \sigma \subset I \times I$ gesucht, in denen A mit niedrigem Rang approximierbar ist.

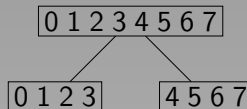
Eine Suche in $\mathcal{P}(I \times I)$ ist zu aufwändig. Deshalb erfolgt eine hierarchische Zerlegung der Indexmenge I : der **Clusterbaum**.

Beispiel (binäre Raumteilung)

Gebiet Ω



Clusterbaum



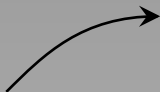
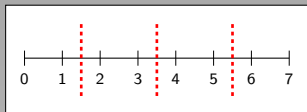
Clusterbäume

Seien I eine Indexmenge mit $|I| = n$ und $A \in \mathbb{R}^{I \times I}$. Für die Näherung \tilde{A} werden Blöcke $\tau \times \sigma \subset I \times I$ gesucht, in denen A mit niedrigem Rang approximierbar ist.

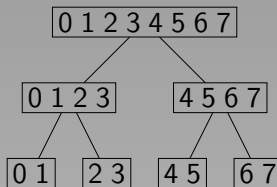
Eine Suche in $\mathcal{P}(I \times I)$ ist zu aufwändig. Deshalb erfolgt eine hierarchische Zerlegung der Indexmenge I : der **Clusterbaum**.

Beispiel (binäre Raumteilung)

Gebiet Ω



Clusterbaum



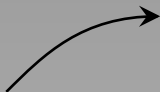
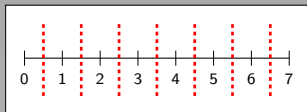
Clusterbäume

Seien I eine Indexmenge mit $|I| = n$ und $A \in \mathbb{R}^{I \times I}$. Für die Näherung \tilde{A} werden Blöcke $\tau \times \sigma \subset I \times I$ gesucht, in denen A mit niedrigem Rang approximierbar ist.

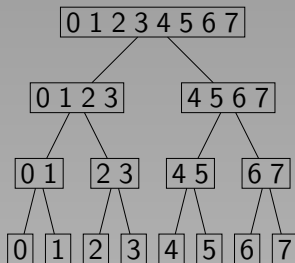
Eine Suche in $\mathcal{P}(I \times I)$ ist zu aufwändig. Deshalb erfolgt eine hierarchische Zerlegung der Indexmenge I : der **Clusterbaum**.

Beispiel (binäre Raumteilung)

Gebiet Ω



Clusterbaum



Blockclusterbaum

Durch Multiplikation zweier Clusterbäume entsteht der **Blockclusterbaum** über $I \times I$.

Anstelle des vollständigen Produktes werden **zulässige** Paare (τ, σ) , mit approximierbarem Matrixblock $A \in \mathbb{R}^{\tau \times \sigma}$, zu Blättern.

Standardzulässigkeit:

$$\min\{\text{diam}(\tau), \text{diam}(\sigma)\} \leq \eta \text{dist}(\tau, \sigma)$$



Blockclusterbaum

Durch Multiplikation zweier Clusterbäume entsteht der **Blockclusterbaum** über $I \times I$.

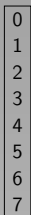
Anstelle des vollständigen Produktes werden **zulässige** Paare (τ, σ) , mit approximierbarem Matrixblock $A \in \mathbb{R}^{\tau \times \sigma}$, zu Blättern.

Standardzulässigkeit:

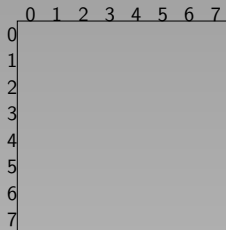
$$\min\{\text{diam}(\tau), \text{diam}(\sigma)\} \leq \eta \text{dist}(\tau, \sigma)$$

Beispiel

Clusterbaum



Blockclusterbaum



$I \times I$



Blockclusterbaum

Durch Multiplikation zweier Clusterbäume entsteht der **Blockclusterbaum** über $I \times I$.

Anstelle des vollständigen Produktes werden **zulässige** Paare (τ, σ) , mit approximierbarem Matrixblock $A \in \mathbb{R}^{\tau \times \sigma}$, zu Blättern.

Standardzulässigkeit:

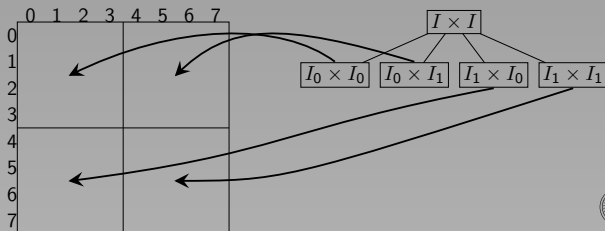
$$\min\{\text{diam}(\tau), \text{diam}(\sigma)\} \leq \eta \text{dist}(\tau, \sigma)$$

Beispiel

Clusterbaum



Blockclusterbaum



Blockclusterbaum

Durch Multiplikation zweier Clusterbäume entsteht der **Blockclusterbaum** über $I \times I$.

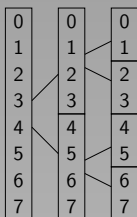
Anstelle des vollständigen Produktes werden **zulässige** Paare (τ, σ) , mit approximierbarem Matrixblock $A \in \mathbb{R}^{\tau \times \sigma}$, zu Blättern.

Standardzulässigkeit:

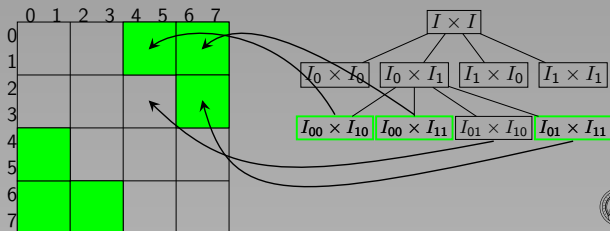
$$\min\{\text{diam}(\tau), \text{diam}(\sigma)\} \leq \eta \text{dist}(\tau, \sigma)$$

Beispiel

Clusterbaum



Blockclusterbaum



Blockclusterbaum

Durch Multiplikation zweier Clusterbäume entsteht der **Blockclusterbaum** über $I \times I$.

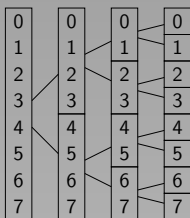
Anstelle des vollständigen Produktes werden **zulässige** Paare (τ, σ) , mit approximierbarem Matrixblock $A \in \mathbb{R}^{\tau \times \sigma}$, zu Blättern.

Standardzulässigkeit:

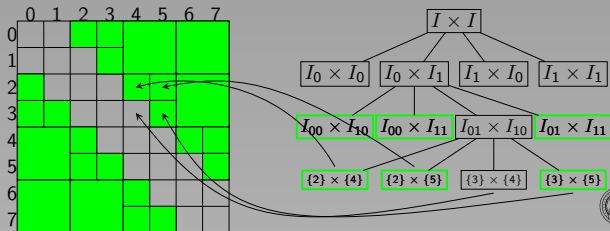
$$\min\{\text{diam}(\tau), \text{diam}(\sigma)\} \leq \eta \text{dist}(\tau, \sigma)$$

Beispiel

Clusterbaum

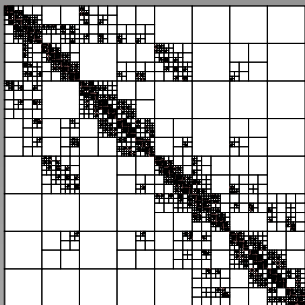


Blockclusterbaum

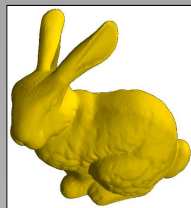
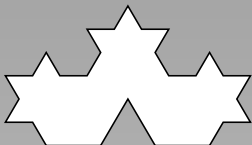
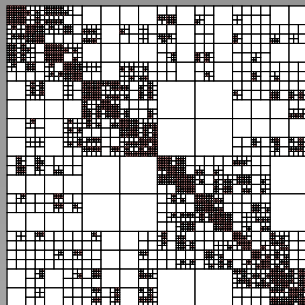


Beispiele

Partielle Differentialgl. im \mathbb{R}^2



Randintegralgleichung im \mathbb{R}^3



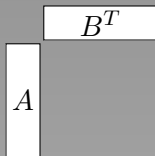
Rank- k -Matrizen

Rang- k -Darstellung

Niedrigrangmatrizen $M \in \mathbb{R}^{n \times m}$ mit Rang k werden repräsentiert durch

$$A \cdot B^T \quad \text{mit } A \in \mathbb{R}^{n \times k} \text{ und } B \in \mathbb{R}^{m \times k}.$$

Reduktion des Speicheraufwands von $n \cdot m$ auf $k(n + m)$.



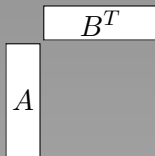
Rang- k -Matrizen

Rang- k -Darstellung

Niedrigrangmatrizen $M \in \mathbb{R}^{n \times m}$ mit Rang k werden repräsentiert durch

$$A \cdot B^T \quad \text{mit } A \in \mathbb{R}^{n \times k} \text{ und } B \in \mathbb{R}^{m \times k}.$$

Reduktion des Speicheraufwands von $n \cdot m$ auf $k(n + m)$.



Rang- k -Arithmetik

Produkt $AB^T \cdot M$ mit Matrix M besitzt Rang k .

Summe $A_1B_1^T + A_2B_2^T$ besitzt im allgemeinen Rang $2k$.

Daher Approximation der Summe mittels **Singulärwertzerlegung**.



\mathcal{H} -Matrix-Algebra

Algorithmen

\mathcal{H} -Algebra verwendet (rekursive) Algorithmen für Blockmatrizen.

Auftretende Additionen zwischen Rang- k -Matrizen werden approximativ berechnet.

Komplexität

Speicher	$\mathcal{O}(n \log n)$
Matrix-Vektor-Multiplikation	$\mathcal{O}(n \log n)$
Matrix-Addition	$\mathcal{O}(n \log n)$
Matrix-Multiplikation	$\mathcal{O}(n \log^2 n)$
Matrix-Inversion	$\mathcal{O}(n \log^2 n)$



Überblick

- 1 Einführung
- 2 \mathcal{H} -Matrizen
- 3 Parallele \mathcal{H} -Algebra**
- 4 Bemerkungen



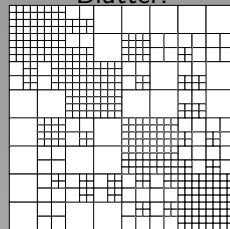
Matrix-Aufbau

Parallelisierung interessant z.B. bei Integralgleichungen, da häufig aufwändige Quadraturen erforderlich sind.

Idee

Verteilung der Blätter im Blockclusterbaum da hier die Arbeit erfolgt.

Blätter:



Matrix-Aufbau

Parallelisierung interessant z.B. bei Integralgleichungen, da häufig aufwändige Quadraturen erforderlich sind.

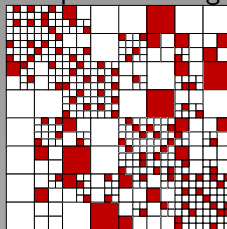
Idee

Verteilung der Blätter im Blockclusterbaum da hier die Arbeit erfolgt.

Lastbalancierung

Mittels **LPT** bei Güte $\left(\frac{4}{3} - \frac{1}{3 \cdot p}\right)$.

Beispielverteilung:



Matrix-Aufbau

Parallelisierung interessant z.B. bei Integralgleichungen, da häufig aufwändige Quadraturen erforderlich sind.

Idee

Verteilung der Blätter im Blockclusterbaum da hier die Arbeit erfolgt.

Lastbalancierung

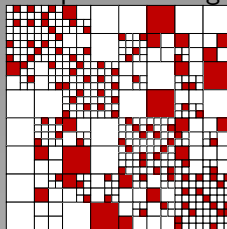
Mittels **LPT** bei Güte $\left(\frac{4}{3} - \frac{1}{3 \cdot p}\right)$.

Resultat

Aufwand ohne Hierarchie: $\mathcal{O}\left(\frac{n \log n}{p}\right)$

Aufwand mit Hierarchie: $\mathcal{O}\left(\frac{n \log n}{p} + n\right)$

Beispielverteilung:



Matrix-Aufbau

Parallelisierung interessant z.B. bei Integralgleichungen, da häufig aufwändige Quadraturen erforderlich sind.

Idee

Verteilung der Blätter im Blockclusterbaum da hier die Arbeit erfolgt.

Lastbalancierung

Mittels **LPT** bei Güte $\left(\frac{4}{3} - \frac{1}{3 \cdot p}\right)$.

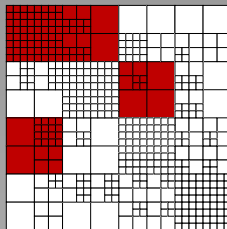
Resultat

Aufwand ohne Hierarchie: $\mathcal{O}\left(\frac{n \log n}{p}\right)$

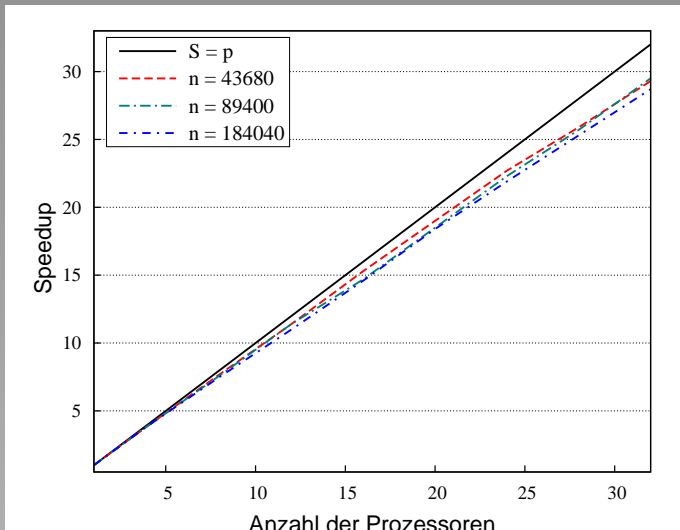
Aufwand mit Hierarchie: $\mathcal{O}\left(\frac{n \log n}{p} + n\right)$

Stufenweise Verteilung reduziert Komplexität und erhöht Lokalität.

Stufenweise Verteilung:



Matrix-Aufbau für Einfachschichtpotential



Matrix-Vektor-Multiplikation

Ausgangssituation

Mit \mathcal{H} -Matrix A und $x, y \in \mathbb{R}^n$ sei zu berechnen:

$$A \cdot x = y$$

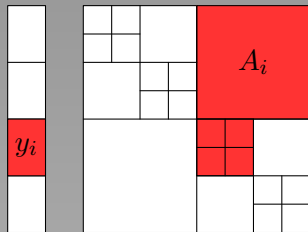


Datenverteilung

Vektoren x_i, y_i enthalten n/p

Elemente von x bzw. y .

A_i enthalte Matrixblöcke von A auf
Prozessor i .



Matrix-Vektor-Multiplikation

Ausgangssituation

Mit \mathcal{H} -Matrix A und $x, y \in \mathbb{R}^n$ sei zu berechnen:

$$A \cdot x = y$$

Datenverteilung

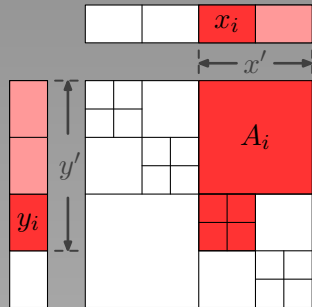
Vektoren x_i, y_i enthalten n/p

Elemente von x bzw. y .

A_i enthalte Matrixblöcke von A auf
Prozessor i .

Algorithmus

- 1 Sende x_i an alle Prozessoren.
- 2 Berechne lokal $A_i x' = y'$.
- 3 Sende y' und summiere y_i .



$$\mathcal{O}(d_{sh} \cdot n/p)$$

$$\mathcal{O}((n \log n)/p)$$

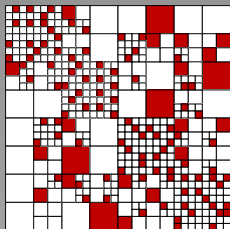
$$\mathcal{O}(|y'| + d_{sh} \cdot n/p)$$

Teilungsgrad d_{sh} : Anzahl von Prozessoren pro Index.

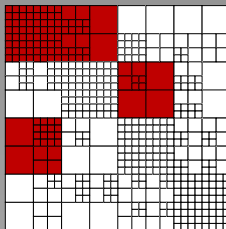


Verteilung der Matrix

Beispielverteilungen



LPT



Stufenweises LPT

Problem

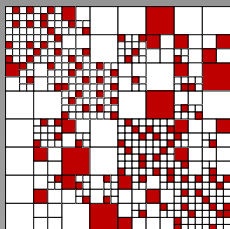
Bei ungünstiger Verteilung von A ist $d_{\text{sh}} \sim \mathcal{O}(p)$ und somit der Aufwand

$$\mathcal{O}\left(\frac{n \log n}{p} + n\right)$$

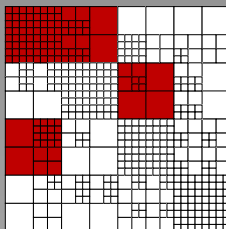


Verteilung der Matrix

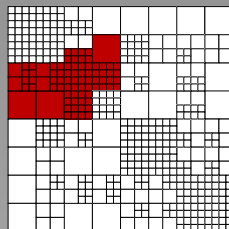
Beispielverteilungen



LPT



Stufenweises LPT



Problem

Bei ungünstiger Verteilung von A ist $d_{\text{sh}} \sim \mathcal{O}(p)$ und somit der Aufwand

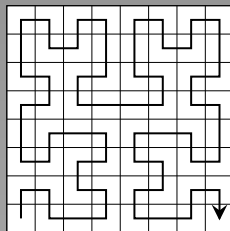
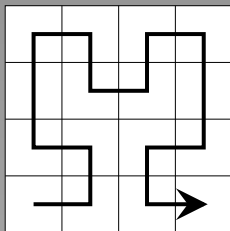
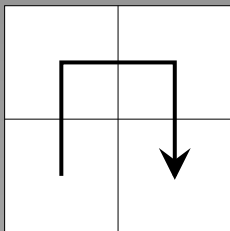
$$\mathcal{O}\left(\frac{n \log n}{p} + n\right)$$

Gesucht ist **kompakte** Verteilung von A , so dass $d_{\text{sh}} \sim \mathcal{O}(1)$.



Lastbalancierung mit raumfüllenden Kurven

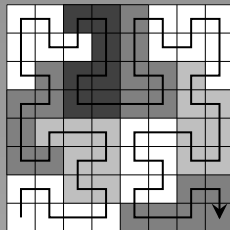
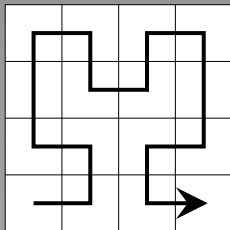
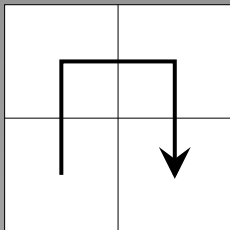
Hilbertkurve im \mathbb{R}^2



Rekursive Konstruktion entspricht Blockclusterbaum über binärem Clusterbaum.

Lastbalancierung mit raumfüllenden Kurven

Hilbertkurve im \mathbb{R}^2



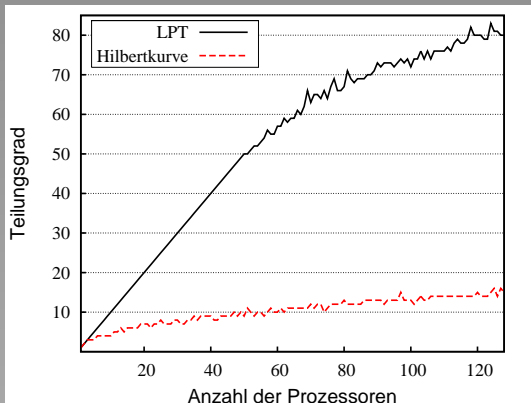
Rekursive Konstruktion entspricht Blockclusterbaum über binärem Clusterbaum.

Lastbalancierung

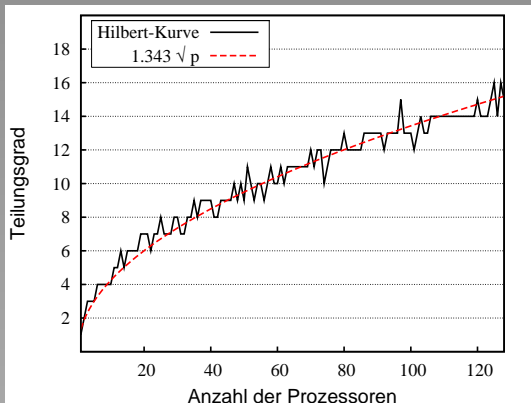
- 1 Anordnung der Matrixblöcke mittels raumfüllender Kurve.
- 2 **Sequenzpartitionierung** der so definierten Liste.



Matrix-Vektor-Multiplikation mit raumfüllenden Kurven



Matrix-Vektor-Multiplikation mit raumfüllenden Kurven

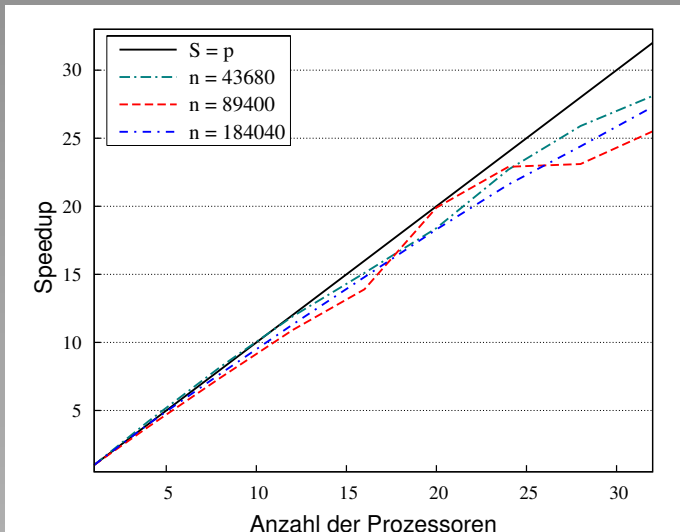


Bei Verwendung von raumfüllenden Kurven ergibt sich Aufwand

$$\mathcal{O}\left(\frac{n \log n}{p} + \frac{n}{\sqrt{p}} + n\right)$$



Matrix-Vektor-Multiplikation für Einfachschichtpotential



Matrix-Multiplikation

Situation

Berechnung von

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \cdot \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}$$

auf Rechnersystem mit gemeinsamem Speicher

Erster Ansatz

Lastbalancierung auf Menge der Einzelmultiplikationen

$$\{C_{00} = A_{00}B_{00}, C_{00} = A_{01}B_{10}, C_{01} = A_{00}B_{01}, \dots, C_{11} = A_{11}B_{11}\}$$



Matrix-Multiplikation

Situation

Berechnung von

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \cdot \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}$$

auf Rechnersystem mit gemeinsamem Speicher

Erster Ansatz

Lastbalancierung auf Menge der Einzelmultiplikationen

$$\{C_{00} = A_{00}B_{00}, C_{00} = A_{01}B_{10}, C_{01} = A_{00}B_{01}, \dots, C_{11} = A_{11}B_{11}\}$$

Problem

Mehrere Multiplikationen haben gleiche Zielmatrix.

Dadurch Gefahr von Kollisionen.

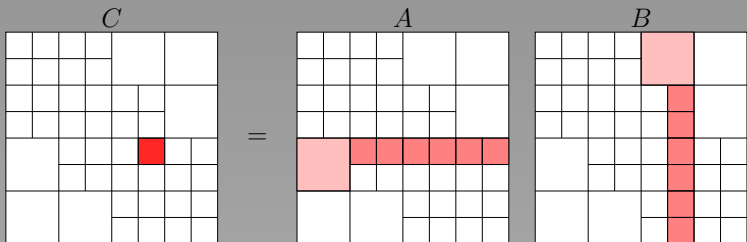
Verwendung von **Semaphoren** reduziert aber parallele Effizienz.



Matrix-Multiplikation

Reduktion der Kollisionen

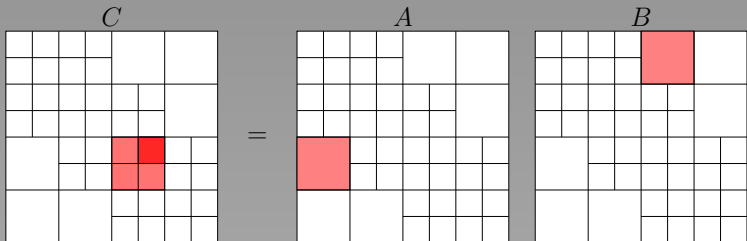
Identifikation von Multiplikationen mit gleicher Zielmatrix



Matrix-Multiplikation

Reduktion der Kollisionen

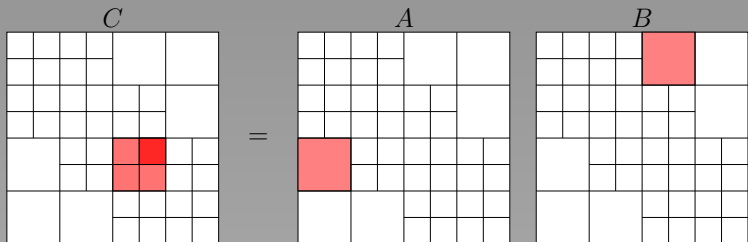
Identifikation von Multiplikationen mit gleicher Zielmatrix



Matrix-Multiplikation

Reduktion der Kollisionen

Identifikation von Multiplikationen mit gleicher Zielmatrix



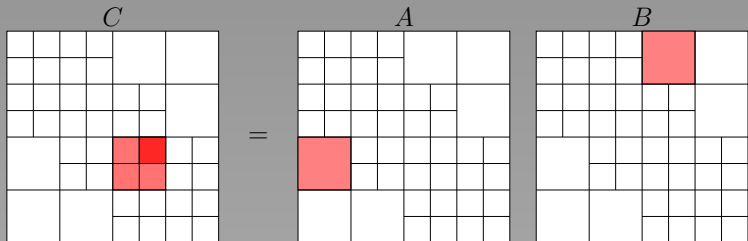
Lastbalancierung auf Menge der Zielmatrixblöcke.



Matrix-Multiplikation

Reduktion der Kollisionen

Identifikation von Multiplikationen mit gleicher Zielmatrix



Lastbalancierung auf Menge der Zielmatrixblöcke.

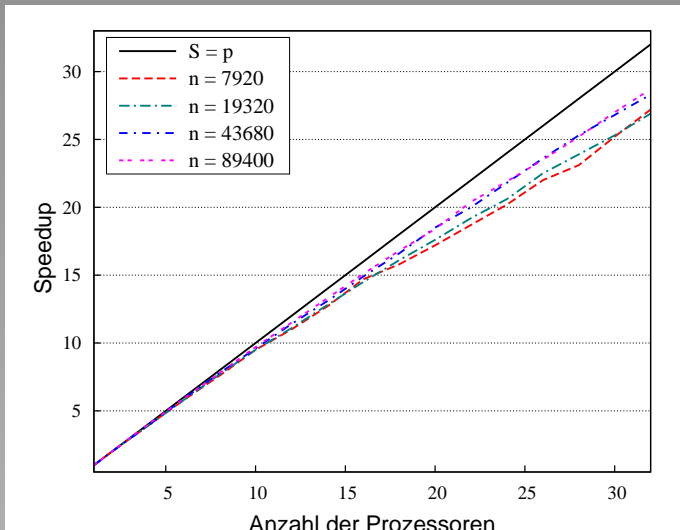
Schedulingvarianten

Online-Scheduling: schnelles Verfahren aber nur Güte $2 - \frac{1}{p}$.

Offline-Scheduling: Kostenbestimmung **sehr aufwändig**



Matrix-Multiplikation für Einfachschichtpotential



Matrix-Inversion

Algorithmus

Sequentielle Gauß-Elimination für

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \text{ führt zu } \begin{pmatrix} A_{00}^{-1} + A_{00}^{-1} A_{01} S^{-1} A_{10} A_{00}^{-1} & -A_{00}^{-1} A_{01} S^{-1} \\ -S^{-1} A_{10} A_{00}^{-1} & S^{-1} \end{pmatrix},$$

$$S = A_{11} - A_{10} A_{00}^{-1} A_{01},$$

mit rekursivem Aufruf für A_{00} und A_{11} bzw. S und Matrix-Multiplikationen.



Matrix-Inversion

Algorithmus

Sequentielle Gauß-Elimination für

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \text{ führt zu } \begin{pmatrix} A_{00}^{-1} + A_{00}^{-1} A_{01} S^{-1} A_{10} A_{00}^{-1} & -A_{00}^{-1} A_{01} S^{-1} \\ -S^{-1} A_{10} A_{00}^{-1} & S^{-1} \end{pmatrix},$$

$$S = A_{11} - A_{10} A_{00}^{-1} A_{01},$$

mit rekursivem Aufruf für A_{00} und A_{11} bzw. S und Matrix-Multiplikationen.

Parallelisierungsansatz

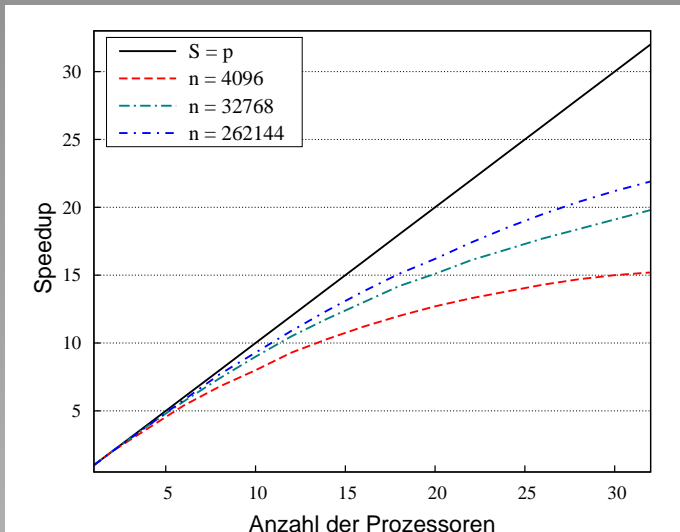
Benutze parallele Multiplikation (Online- oder Offline-Scheduling).

Diagonale wird strikt sequentiell behandelt.

Komplexität der Inversion hierdurch:

$$\mathcal{O}\left(\frac{n \log^2 n}{p} + n\right)$$



Matrix-Inversion für Laplace-Gleichung im \mathbb{R}^3 

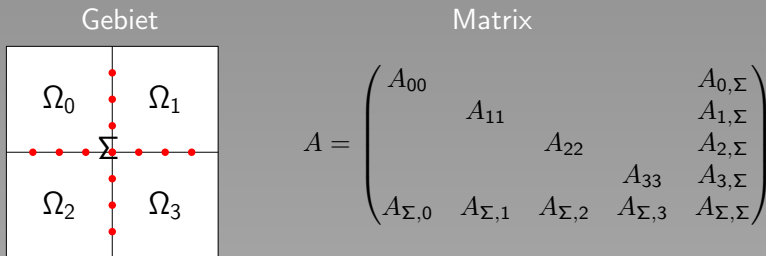
Überblick

- 1 Einführung
- 2 \mathcal{H} -Matrizen
- 3 Parallele \mathcal{H} -Algebra
- 4** Bemerkungen



Gebietszerlegung

Für partielle Differentialgleichungen bietet sich die Technik der **Gebietszerlegung** an:



Bei Inversion lassen sich A_{ii} , $A_{\Sigma,i}$ und $A_{i,\Sigma}$ **parallel** behandeln. Dabei tritt **Schurkomplement** $A_{\Sigma,\Sigma} - \sum_{i=0}^{p-1} A_{\Sigma,i} A_{ii}^{-1} A_{i,\Sigma}$ auf.

Mit \mathcal{H} -Matrix-Arithmetik ergibt sich im \mathbb{R}^d Aufwand:

$$\mathcal{O}\left(\frac{n}{p} \log^2 \frac{n}{p} + p^{\frac{1}{d}} n^{1-\frac{1}{d}} \log p \log n\right)$$



Einfluss der Speicherverwaltung

Parallele Laufzeit für \mathcal{H} -Multiplikation (10 GB Speicher)

p	1	4	8	16
SUNmalloc	15305.2 s	4210.3 s	3201.4 s	3536.1 s
MTmalloc	34244.0 s	3635.6 s	1858.4 s	967.2 s
PTmalloc	185079.4 s	9360.6 s	3404.9 s	1689.2 s

Sequentielle Laufzeit für \mathcal{H} -Inversion (5 GB Speicher)

n	4096	16384	65536	262144
SUNmalloc	13.0 s	115.8 s	818.0 s	5257.6 s
MTmalloc	13.0 s	117.8 s	924.6 s	8488.3 s
PTmalloc	12.9 s	122.4 s	1268.9 s	30556.7 s



Einfluss der Speicherverwaltung

Parallele Laufzeit für \mathcal{H} -Multiplikation (10 GB Speicher)

p	1	4	8	16
SUNmalloc	15305.2 s	4210.3 s	3201.4 s	3536.1 s
MTmalloc	34244.0 s	3635.6 s	1858.4 s	967.2 s
PTmalloc	185079.4 s	9360.6 s	3404.9 s	1689.2 s
Rmalloc	15111.9 s	3649.5 s	1853.9 s	954.7 s

Sequentielle Laufzeit für \mathcal{H} -Inversion (5 GB Speicher)

n	4096	16384	65536	262144
SUNmalloc	13.0 s	115.8 s	818.0 s	5257.6 s
MTmalloc	13.0 s	117.8 s	924.6 s	8488.3 s
PTmalloc	12.9 s	122.4 s	1268.9 s	30556.7 s
Rmalloc	12.9 s	114.8 s	800.7 s	5003.5 s



PHI – Parallele \mathcal{H} -Matrix Implementierung

Implementierung der parallelen \mathcal{H} -Matrix-
Algorithmen für Systeme mit gemeinsamem
und verteiltem Speicher.

Verwendung von **POSIX-Threads** bzw. **BSP-
Bibliothek** zur Parallelisierung.

Unterstützung verschiedener Rechnerarchitekturen:

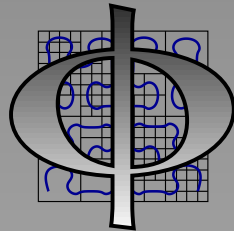
x86, UltraSparc, Alpha, POWER, Itanium, PA-RISC, MIPS

und Betriebssysteme:

Linux, Solaris, AIX, HP-UX, IRIX, Darwin, Tru64.

Verfügbar unter

<http://www.mis.mpg.de/scicomp/>



Zusatz

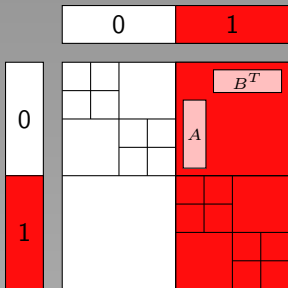
- 1 Rang- k -Teilung
- 2 Rmalloc



Matrix-Vektor-Multiplikation mit Rang- k -Teilung

Problem

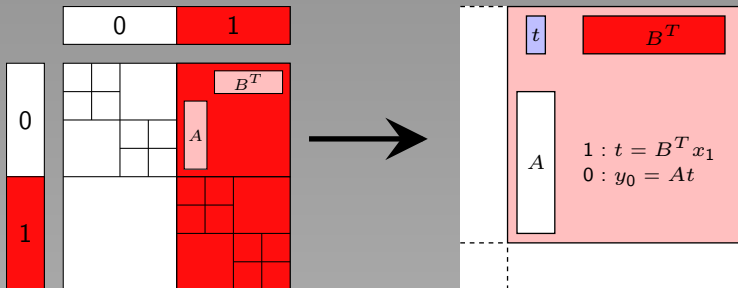
Kommunikation dominiert durch **große** Blöcke der Dimension $\mathcal{O}(n)$ da Matrixblöcke atomare Einheiten der Lastbalancierung bilden.



Matrix-Vektor-Multiplikation mit Rang- k -Teilung

Problem

Kommunikation dominiert durch **große** Blöcke der Dimension $\mathcal{O}(n)$ da Matrixblöcke atomare Einheiten der Lastbalancierung bilden.



Alternative

Aufteilung der Multiplikation $y = AB^T x$ bei Niedrigrangmatrizen in $t = B^T x$ und $y = At$.

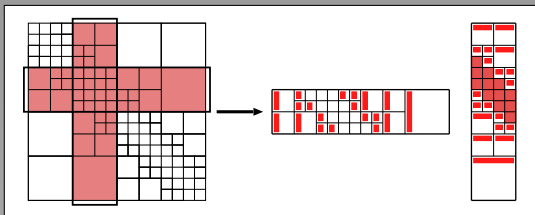
Kommunikation somit anstelle von $\mathcal{O}(n)$ nur noch $\mathcal{O}(k)$.



Matrix-Vektor-Multiplikation mit Rang- k -Teilung

Aufteilung der \mathcal{H} -Matrix

Zerlegung in **Blockspalten** und **-zeilen** jeweils der Dimension $\mathcal{O}(n/p)$. Für Matrizen $R = AB^T$ Zuweisung der Matrix B zu Blockspalte bzw. A zu Blockzeile:



Algorithmus

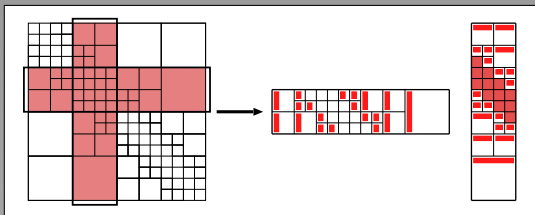
- 1 Multiplikation von $t_i = B^T x_i$ für alle Rang- k -Matrizen.
- 2 Senden von t_i an entsprechende Prozessoren.
- 3 Multiplikation von $y_i = A t_i$.



Matrix-Vektor-Multiplikation mit Rang- k -Teilung

Aufteilung der \mathcal{H} -Matrix

Zerlegung in **Blockspalten** und **-zeilen** jeweils der Dimension $\mathcal{O}(n/p)$. Für Matrizen $R = AB^T$ Zuweisung der Matrix B zu Blockspalte bzw. A zu Blockzeile:



Algorithmus

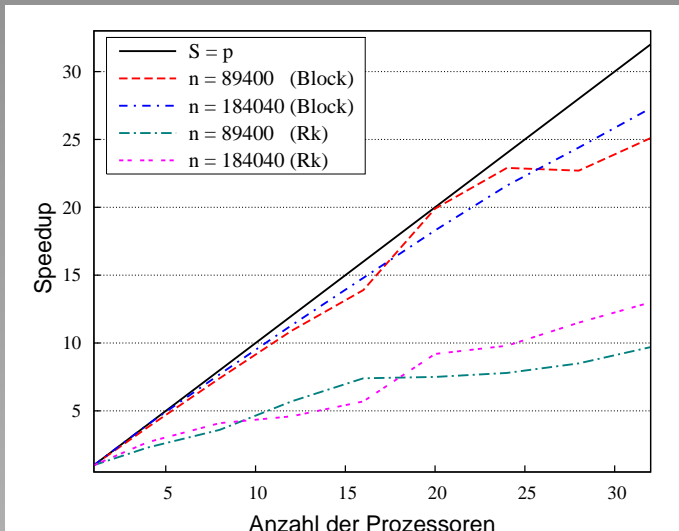
- 1 Multiplikation von $t_i = B^T x_i$ für alle Rang- k -Matrizen.
- 2 Senden von t_i an entsprechende Prozessoren.
- 3 Multiplikation von $y_i = A t_i$.

Komplexität

$$\mathcal{O}\left(\frac{n \log n}{p} + \frac{n}{p}\right)$$



Matrix-Vektor-Multiplikation für Einfachschichtpotential



Rmalloc

Eigenschaften

- 1 Jeder Prozessor besitzt eigene Speicherverwaltung:
 - größtes Mass an Parallelität,
 - Vermeidung von **False-Sharing**.



Rmalloc

Eigenschaften

- 1 Jeder Prozessor besitzt eigene Speicherverwaltung:
 - größtes Mass an Parallelität,
 - Vermeidung von **False-Sharing**.
- 2 Jeder Speicherblock ist an einen Prozessor gekoppelt:
 - Vermeidung von unbeschränktem Speicherverbrauch.



Rmalloc

Eigenschaften

- 1 Jeder Prozessor besitzt eigene Speicherverwaltung:
 - größtes Mass an Parallelität,
 - Vermeidung von **False-Sharing**.
- 2 Jeder Speicherblock ist an einen Prozessor gekoppelt:
 - Vermeidung von unbeschränktem Speicherverbrauch.
- 3 Klassifizierung von Speichergrößen durch exponentielle Folge:
 - schnelle Suche nach freien Speicherblöcken



Rmalloc

Eigenschaften

- 1 Jeder Prozessor besitzt eigene Speicherverwaltung:
 - größtes Mass an Parallelität,
 - Vermeidung von **False-Sharing**.
- 2 Jeder Speicherblock ist an einen Prozessor gekoppelt:
 - Vermeidung von unbeschränktem Speicherverbrauch.
- 3 Klassifizierung von Speichergrößen durch exponentielle Folge:
 - schnelle Suche nach freien Speicherblöcken
- 4 Spezielle Verwaltung kleiner Blöcke mittels **Kontainertechnik**:
 - Zugriff in $\mathcal{O}(1)$,
 - Reduktion des Speicheroverheads.



Rmalloc

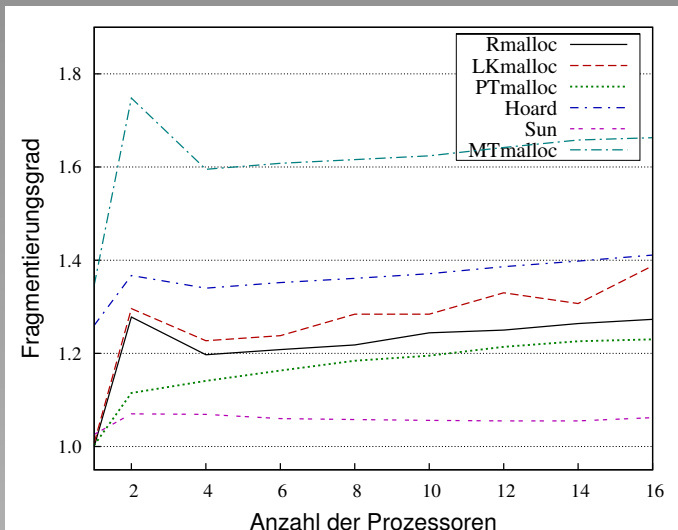
Eigenschaften

- 1 Jeder Prozessor besitzt eigene Speicherverwaltung:
 - größtes Mass an Parallelität,
 - Vermeidung von **False-Sharing**.
- 2 Jeder Speicherblock ist an einen Prozessor gekoppelt:
 - Vermeidung von unbeschränktem Speicherverbrauch.
- 3 Klassifizierung von Speichergrößen durch exponentielle Folge:
 - schnelle Suche nach freien Speicherblöcken
- 4 Spezielle Verwaltung kleiner Blöcke mittels **Kontainertechnik**:
 - Zugriff in $\mathcal{O}(1)$,
 - Reduktion des Speicheroverheads.
- 5 Dynamische **Präallokation** von Speicher:
 - Reduktion von teuren Systemaufrufen.



Fragmentierung

\mathcal{H} -Multiplikation



Fragmentierung

\mathcal{H} -Inversion

