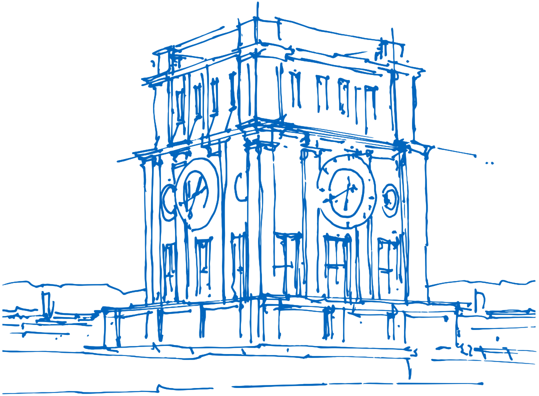
Praktikum GRA – Projektaufgabe A314

Multiplikation dünnbesetzter Matrizen: ELLPACK (row-major)

Simon Gerz, Florian Kainz und Pierre Klein

School of Computation, Information and Technology
Technische Universität München

23. August 2024



Inhalt

ELLPACK Format

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0.5 & 7 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

```
4,4,2                <no of rows>,<no of colums>,<max values per row>
5,*,6,*,0.5,7,3,*    <values>
0,*,1,*,0,1,3,*      <columns>
```

Problem: Multiplikation dünnbesetzter Matrizen

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0.5 & 7 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} \times \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0.5 & 7 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 25 & 0 & 0 & 0 \\ 0 & 36 & 0 & 0 \\ 2.5 & 42 & 0 & 0 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

Maximale Dimension: $(2^{64} - 1) \times (2^{64} - 1)$

Parallelisierbarkeit: Werte suchen (Indexe)

Inhalt

Implementierung V1
Naive Multiplikation

Suche für jeden Wert in der linken Matrix nach einem zugehörigen in der Spalte der rechten Martix

Vorteile:
- leicht nachvollziehbar
- keine Umwandlungen

Nachteile:
- suchen zu multiplizierender Werte aufwändig

Implementierung V0
Gustavson Algorithmus

Vorteile:

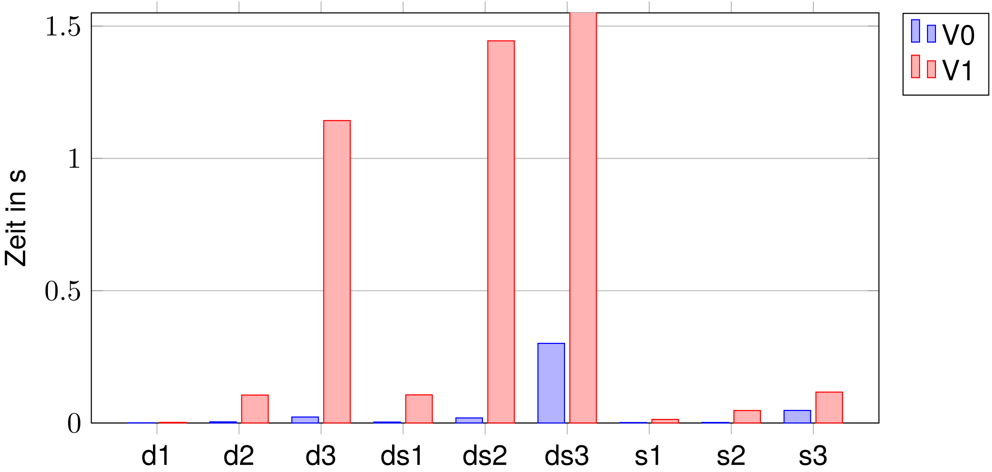
- Geschwindigkeit durch bessere Cache-Nutzung

Nachteile:

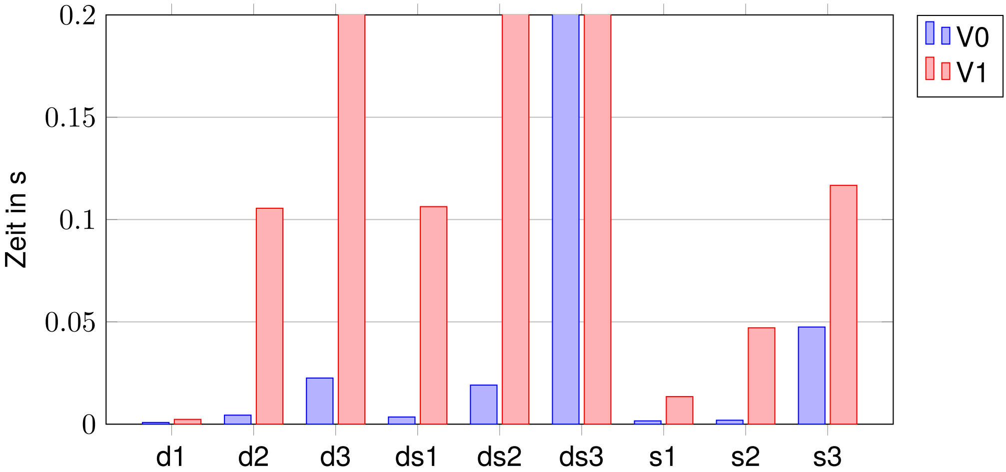
- Zwischenspeichern von Werten (Zusätzlicher Speicherverbrauch)

Besonderheit: Optimierte Suche nach rechten Werten

Benchmarking 0-1



Benchmarking 0-1



Implementierung V2
Multiplikation mit transponierter rechter Matrix

Vorteile:

- bessere Cachefreundlichkeit
- suchen zu multiplizierender Werte vereinfacht

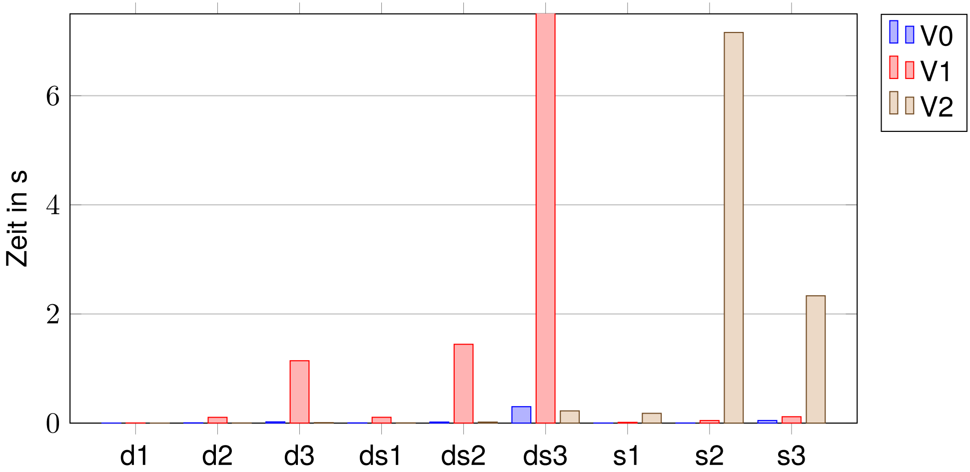
Nachteile:

- rechte Matrix muss transponiert werden

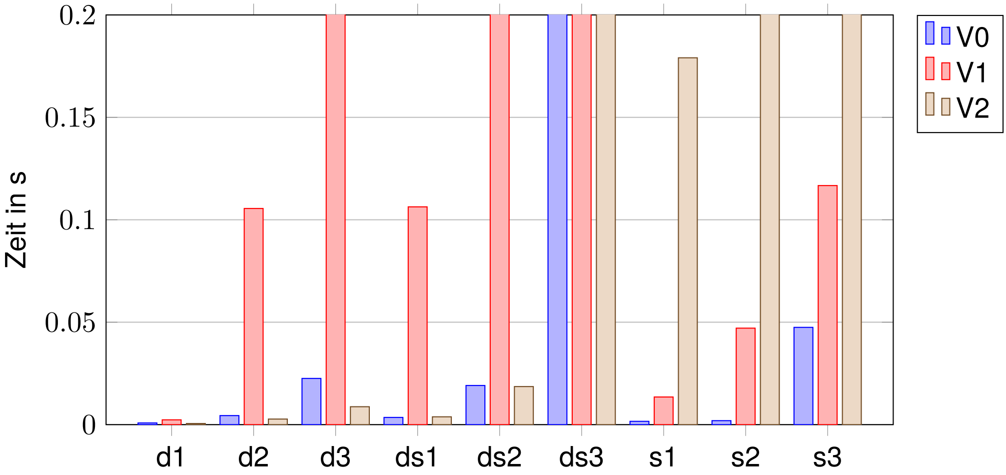
Besonderheit: Transponieren verändert maxNoNonZero von rechts

$$\begin{array}{l} 4,4,2 \\ 5,*,\boxed{6},*,0.5,\boxed{7},3,* \\ 0,*,\boxed{1},*,0,\boxed{1},3,* \end{array} \rightarrow \begin{array}{l} 4,4,2 \\ 5,0.5,\boxed{6},\boxed{7},*,*,3,* \\ 0,2,\boxed{1},\boxed{2},*,*,3,* \end{array}$$

Benchmarking 2



Benchmarking 2



Implementierung V3

'Standard' Matrixmultiplikation auf dichten Matrizen

Vorteile:

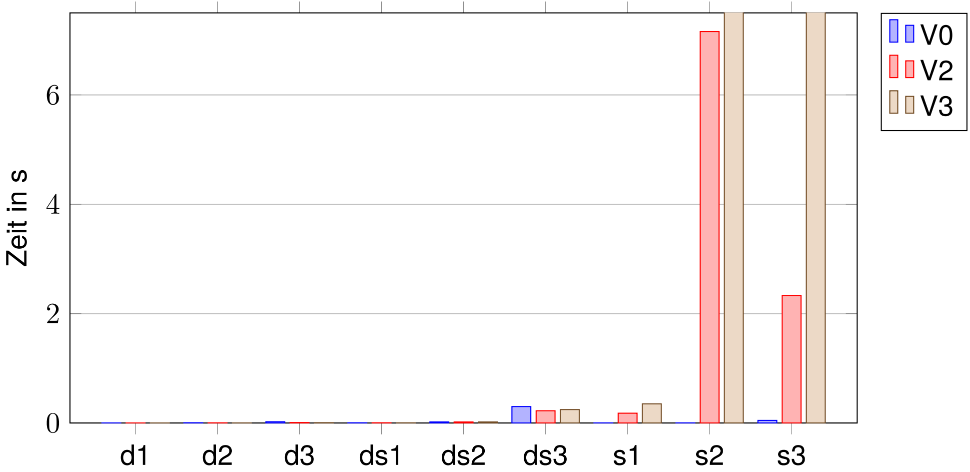
- speichert keine Zeilenindizes
- simpel, keine Suche zu multiplizierender Werte

Nachteile:

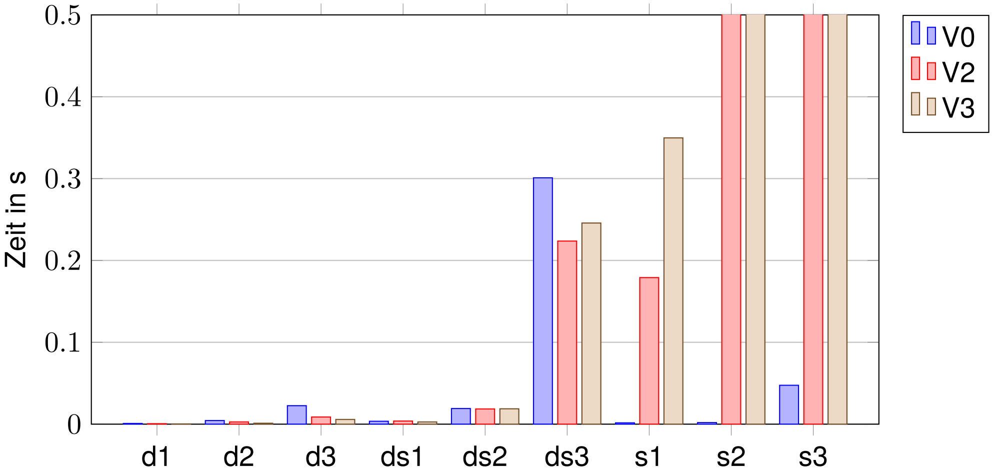
- nutzt Vorteil dünnbesetzter Matrizen nicht aus: speichert und multipliziert Nullwerte

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0.5 & 7 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} \times \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0.5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 25 & 0 & 0 & 0 \\ 0 & 36 & 0 & 0 \\ 2.5 & 42 & 0 & 0 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

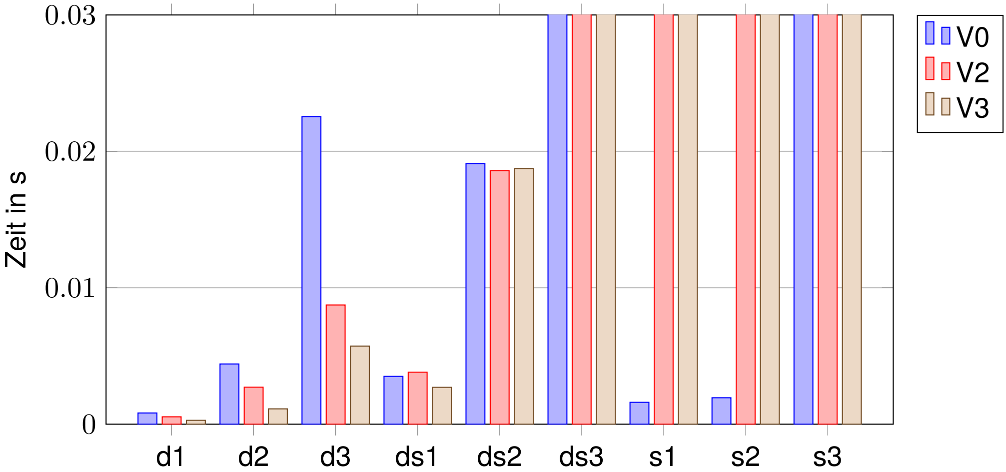
Benchmarking 3



Benchmarking 3



Benchmarking 3



Implementierung V4

Parallele Multiplikation dichter Matrizen, rechts transponiert

Vorteile:

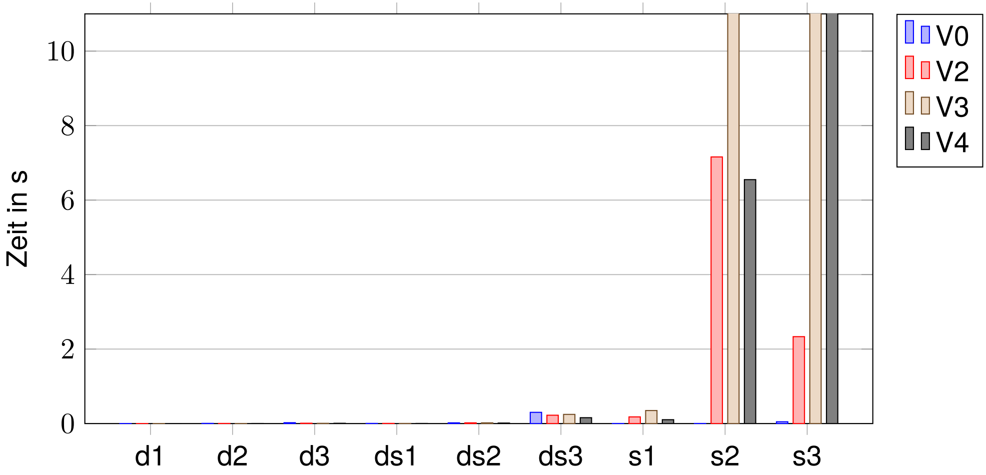
- schnelle Multiplikation durch Parallelisierung
- erhöhte Cachefreundlichkeit
- berechnet genauere Ergebnisse

Nachteile:

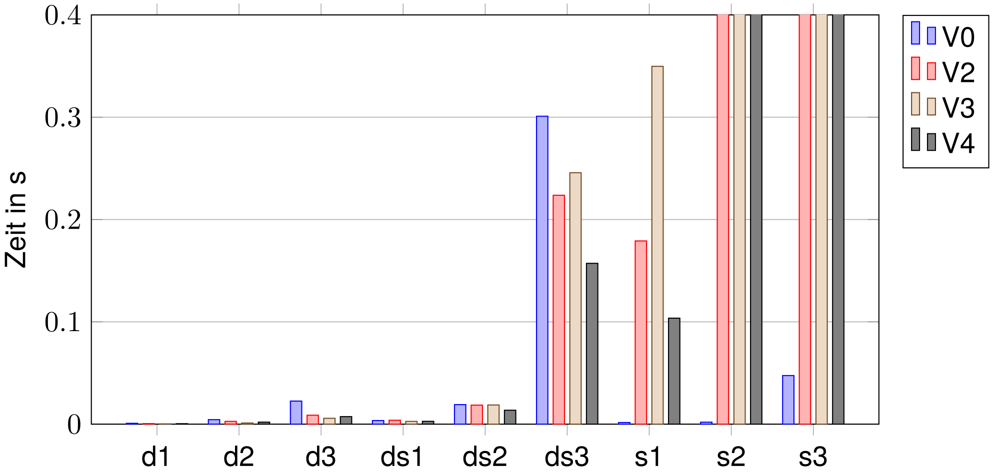
- nutzt Vorteil dünnbesetzter Matrizen nicht aus
- 3 Umwandlungen: transponieren, zu dichter Matrix, zu XMM

$$\begin{array}{r} a \\ b \\ \rightarrow c \\ d \end{array} \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0.5 & 7 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} \times \begin{array}{r} a \\ \rightarrow b \\ c \\ d \end{array} \begin{pmatrix} 5 & 0 & 0.5 & 0 \\ 0 & 6 & 7 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 25 & 0 & 0 & 0 \\ 0 & 36 & 0 & 0 \\ 2.5 & 42 & 0 & 0 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

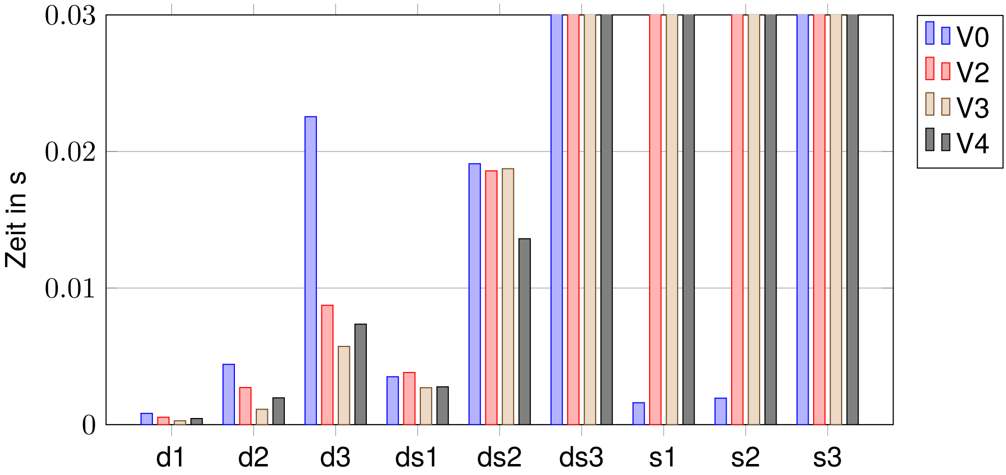
Benchmarking 4



Benchmarking 4



Benchmarking 4



Implementierung V5

Naive Multiplikation mit verbesserter Suche zu multiplizierender Werte

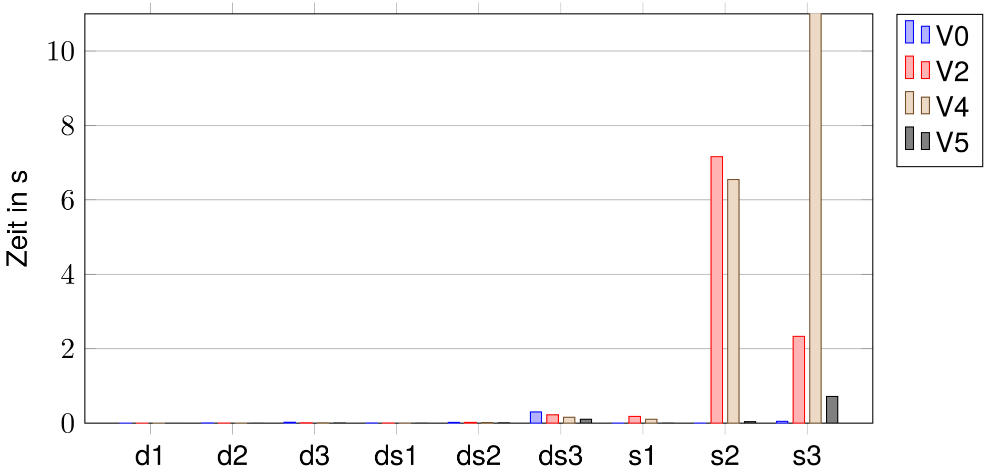
Baut Ergebnismatrix Spaltenweise statt Zeilenweise auf

Speichert aktuelle Spaltenpositionen ab $\rightarrow$ nicht jedes mal neu suchen

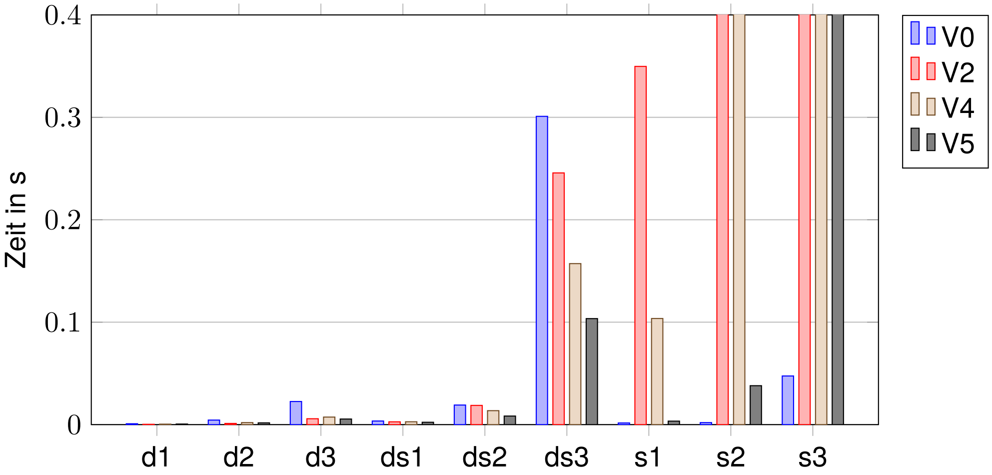
Vorteile:

- schnelle Suche ohne Umwandlungen

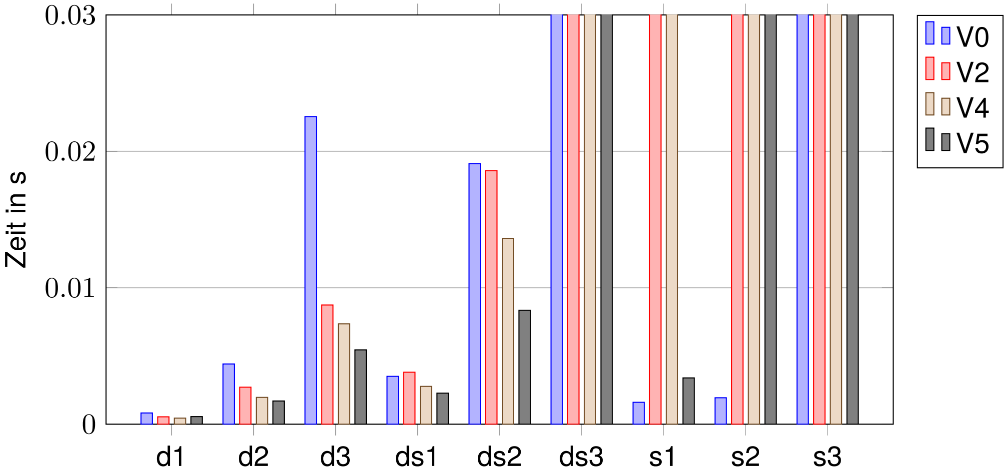
Benchmarking 5



Benchmarking 5



Benchmarking 5



Inhalt

Beweis zur maximalen Anzahl an nicht-Null Einträgen

Lemma. *$\mathcal{M}$ endliches Mengensystem endlicher Mengen, $\exists c \in \mathbb{N} : \forall M \in \mathcal{M} : |M| \leq c$.*

$$\implies \left| \bigcup_{M \in \mathcal{M}} M \right| \leq c|\mathcal{M}|$$

Beweis. trivial □

Definition. *für $A \in K^{n \times m}$ sei*

$$P(A) := \max_{1 \leq i \leq n} |\{(j, a_{ij}) : 1 \leq j \leq m, a_{ij} \neq 0\}|$$

Definition. *für $n \in \mathbb{N}_{>0}$ sei $[n] := \{1, \ldots, n\}$*

Beweis (fort.)

Lemma. $A \in K^{n \times k}, B \in K^{k \times m}, C = AB, P(A) \leq p, P(B) \leq q \implies P(C) \leq pq$

Beweis. Sei für $l \in [k], i \in [n]$:

$$\begin{aligned} N_i &:= \{M_l \mid a_{il} \neq 0, l \in [k]\}, && \implies |N_i| \leq p \\ M_l &:= \{j \mid b_{lj} \neq 0, j \in [m]\}, && \implies |M_l| \leq q \end{aligned}$$

Sei $i \in [n], X := \bigcup_{M \in N_i} M$.

$$\begin{aligned} &\implies \forall j \in [m] : (c_{ij} \neq 0 \implies (\exists M \in N_i : j \in M) \iff j \in X) \\ &\implies \{j \in [m] : c_{ij} \neq 0\} \subset X, \quad |X| \leq pq \\ &\implies P(C) \leq pq \end{aligned}$$

□

Inhalt

Korrektheit

Korrektheit bestätigt durch:

- Ausführliche Tests: Vergleiche zwischen Versionen und mit externen Berechnungen
- Test von Randfällen: extreme Dichten, extreme Dimensionen
- Test von verschiedensten invaliden Eingaben
- Testen mit Werkzeugen wie Valgrind oder Sanitizern

Zusammenfassung

V0 meistens beste Version

ebenfalls gut: V5

weitere Verbesserungen:

- dynamisch beste Version aufrufen
- Version, welche mit Doubles arbeitet für höhere Genauigkeit