

PARA'04

# A Method to Derive the Cache Performance of Irregular Applications on Machines with Direct Mapped Caches

CARSTEN K. SCHOLTES

20. June 2004

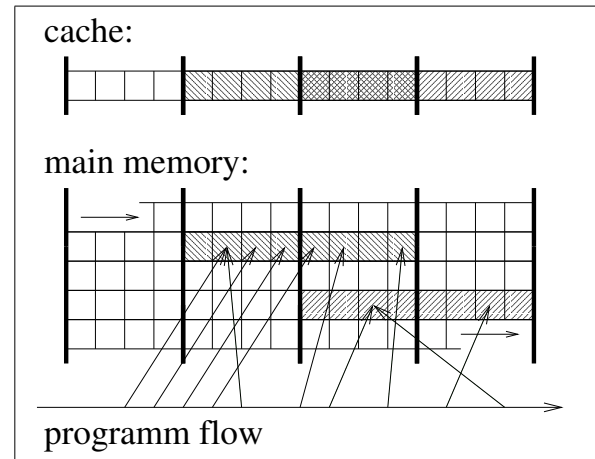
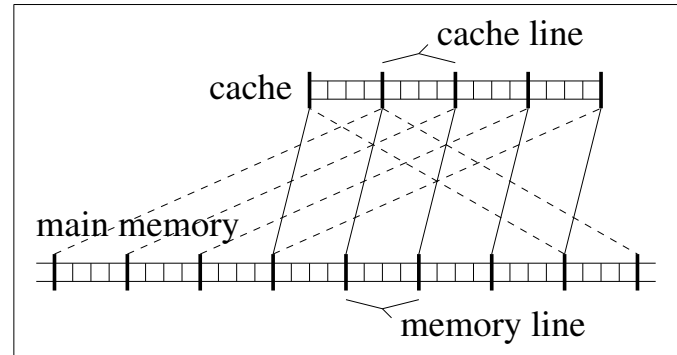
- 1 Introduction
- 2 Method
- 3 Matrix Multiplication
- 4 Cholesky Factorization
- 5 Conclusions und Outlook

# 1. Introduction

## 1.1. Cache

- small, fast, expensive memory
- cache lines and memory lines
- associativity (direct mapped)
- responsibility (data cache)
- Cache hierarchy (one level)
- Example: Intel Pentium 4
  - line size: 64 Bytes
  - memory access (1,5 GHz / 100 MHz):  
102 - 192 CPU cycles

Caches	level 1	level 2
size	64 KB	256 KB / 512 KB
assoc.	4-way	8-way
cycles	2 - 9	7



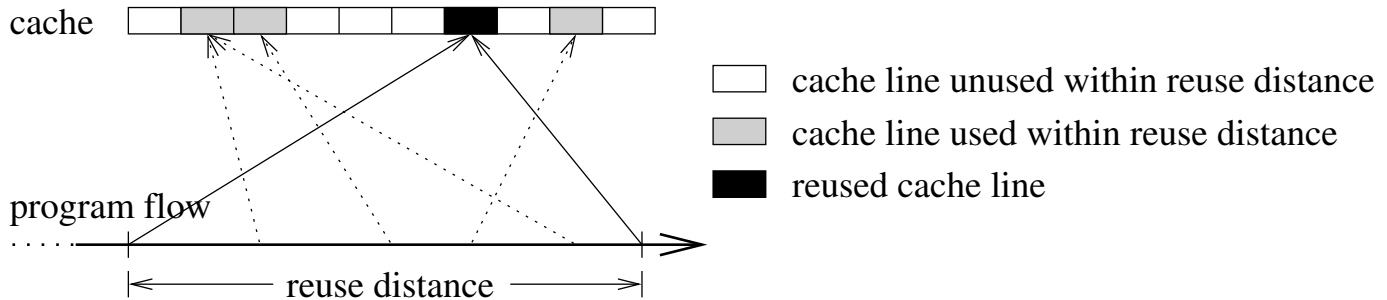
## 1.2. Objectives

- Development of a method to determine the number of misses:
  - based on analysis of the source code
  - for one level, direct mapped data cache
  - result: in constant time evaluable set of calculations
    - \* architecture parameters ( $cs, ds$ )
    - \* input parameters ( $n, \alpha$ )
- Application of the method to sparse matrix operations:
  - matrix multiplication
  - Cholesky factorization
- Approches for Automation

## 2. Method

### 2.1. Basic assumption

The miss probability of a reuse  
 equals the relative proportion of the cache,  
 that has been accessed within the reuse distance.



## 2.2. Procedure

```

R = set of all references ;
for_all references  $r \in R$  {
  determination of reuse types of  $r$  ;
  classification of the accesses through  $r$  ;  $C$  = set of classes of  $r$  ;
  for_all classes  $c \in C$  {
     $n[c] = |c|$  ;
     $d[c]$  = common reuse distance in  $c$  ;
     $I$  = set of interference sets for distance  $d[c]$  ;
    for_all interference sets  $i \in I$  {
       $p_c[i] = \text{affected proportion of the cache}(i, d[c])$  ;
    }
     $p_r[c] = \bigcup_{i \in I} p_c[i]$  ;
  }
   $f[r] = \sum_{c \in C} p_r[c] \cdot n[c]$  ;
}
estimation of total number of misses =  $\sum_{r \in R} f[r]$  ;

```

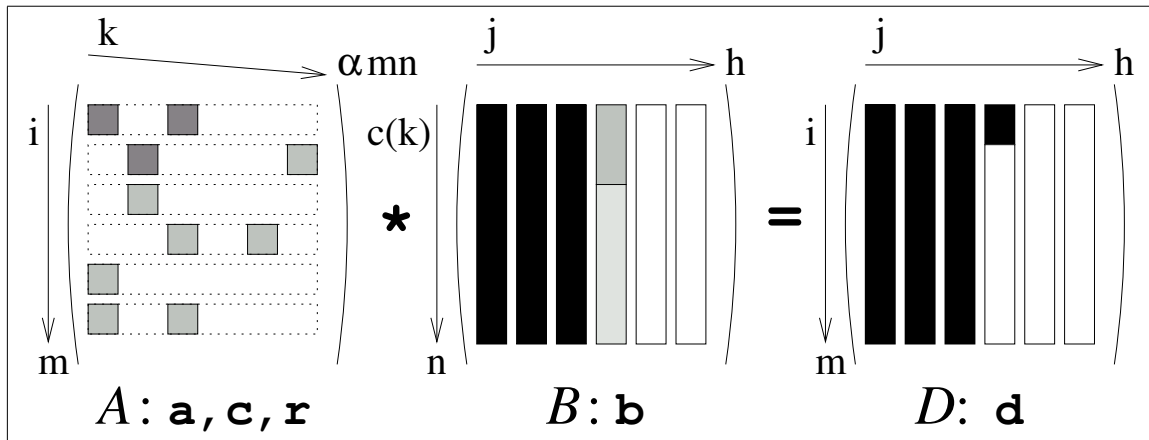
### 3. Matrix multiplication

#### 3.1. Problem

- Multiplication of a sparse matrix  $A$  with a dense matrix  $B$ :

$$A \cdot B = D$$

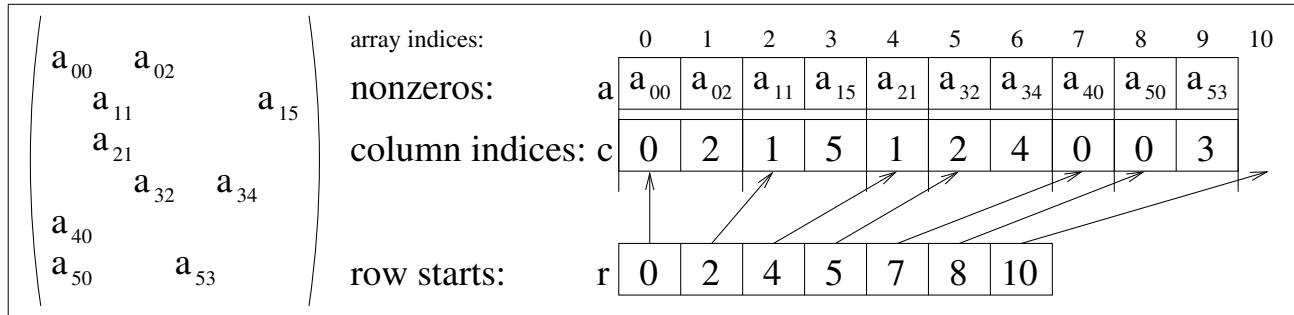
- Visualization :



- Parameters for the analysis: side lengths  $(m, n, h)$  and sparsity factor  $\alpha$

### 3.2. Data structure

- dense matrices: column by column
- sparse matrix: compressed, row by row (crs: compressed row storage)



### 3.3. Tests

- Measurement of misses to compare with analysis:

Tools	Tracing data accesses	misses
WARTS	qpt2	dinerolV
own	instrumented source code	refsim

Trace for dinerolV:

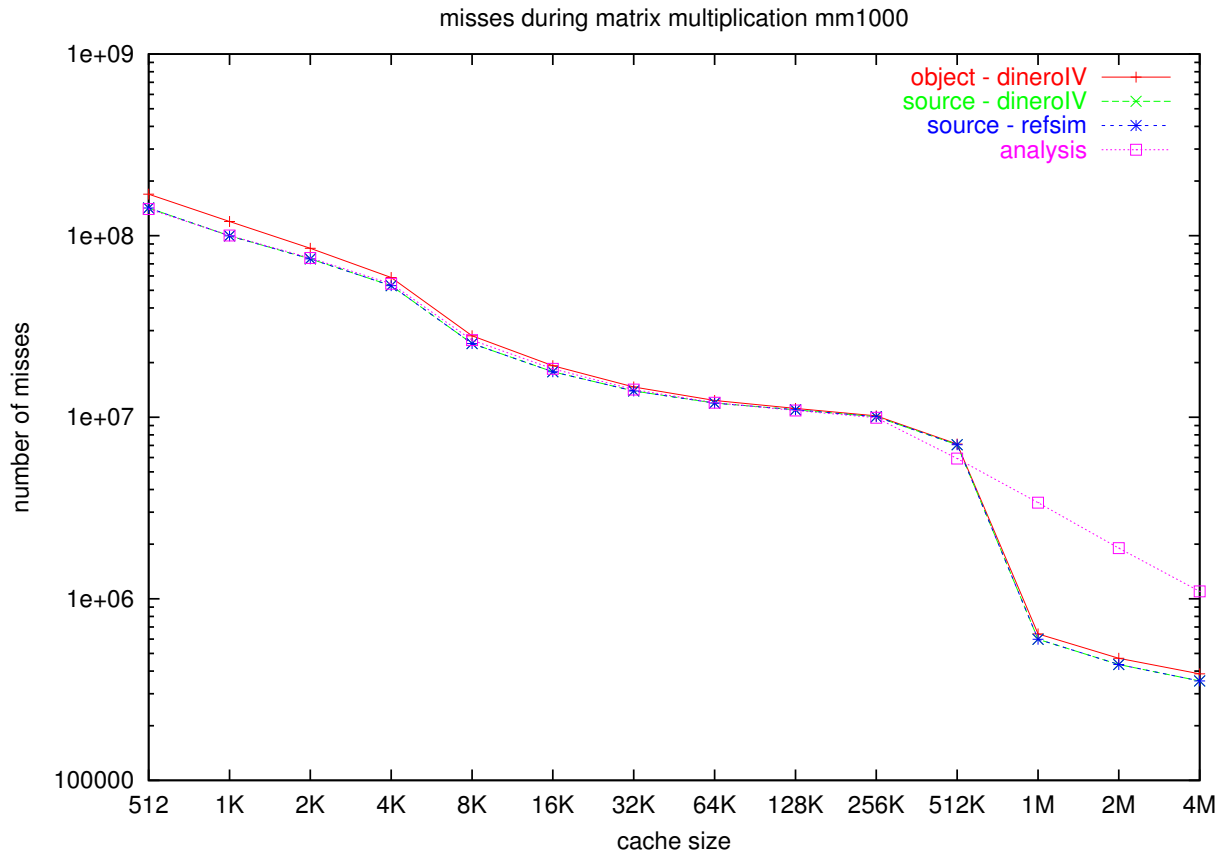
1	ffbeea88
2	11368
0	ffbeea88
2	1136c

Trace for refsim:

0	ffbeea9c	4	23
0	ffbeea8c	4	44
1	ffbeea60	8	51
0	ffbeea9c	4	26

- Testing environment:
  - SUN Ultra-60 with 2 UltraSPARC II processors under Solaris 8.0
  - GNU gcc 2.8.1 without optimization
- Example problem: side length = 1000, sparsity factor  $\alpha = 0.05$ , Nonzeros randomly distributed

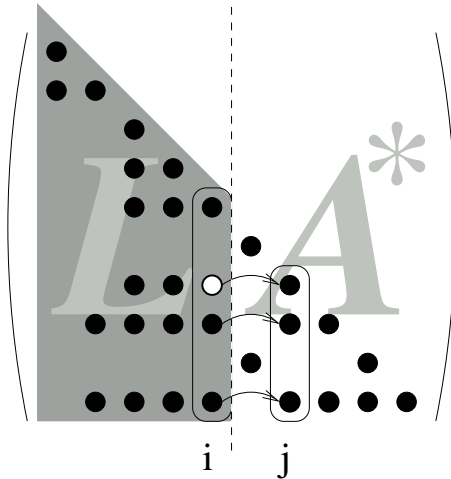
### 3.4. Misses during matrix multiplication



## 4. Cholesky factorization

### 4.1. Problem

- Cholesky factorization of sparse, positive definite, symmetric matrices  $A$ , such that  $A = L \cdot L^T$
- Visualization:



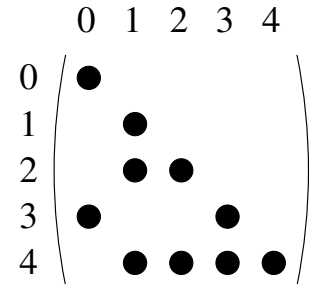
```

complete(j) {
    ljj = sqrt(ajj);
    for(i = j+1; i < n; i++) {
        lij = aij/ljj;
    }
}
modify(j, k) {
    for(i = j; i < n; i++) {
        aij = ljk*lik;
    }
}
cholesky() {
    for(i = 0; i < n; i++) {
        complete(i);
        for(j = i+1; j < n, j++) {
            modify(j, i);
        }
    }
}
    
```

## 4.2. Sparse Cholesky factorization

- Storage: ccs (compressed column storage) with additionally compressed row indices

Index	0	1	2	3	4	5	6	7	8	9
nz	$l_{00}$	$l_{30}$	$l_{11}$	$l_{21}$	$l_{41}$	$l_{22}$	$l_{42}$	$l_{33}$	$l_{43}$	$l_{44}$
row	0*	3	1*	2*	4	3*	4*			
firstnz	0	2	5	7	9	10				
startrow	0	2	3	5	6					



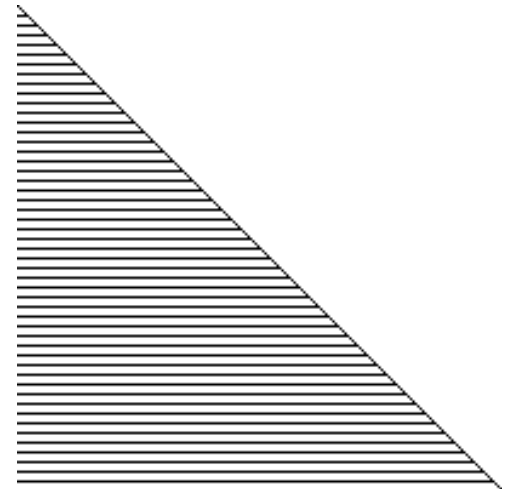
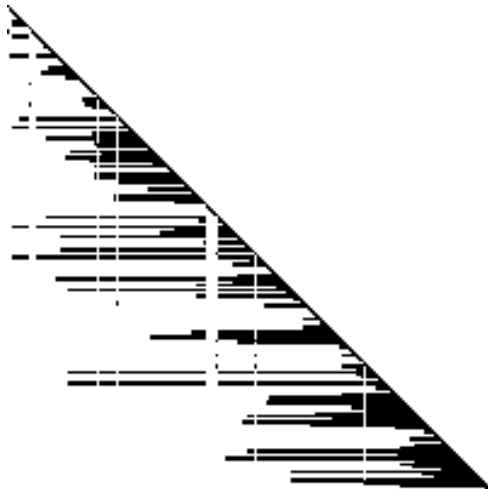
- In modify and complete operations only nonzeros of  $L$  have to be considered
- `modify(j, k)` is pointless, if  $l_{jk} = 0$  ( $\Rightarrow$  exactly one modification per subdiagonal nonzero)
- Parameters for analysis:  
 side length  $n$ , sparsity factor  $\alpha = \frac{nz-n}{0,5 \cdot n \cdot (n-1)}$ , row compression factor  $\beta = \frac{rws}{nz}$

### 4.3. Tested input matrices

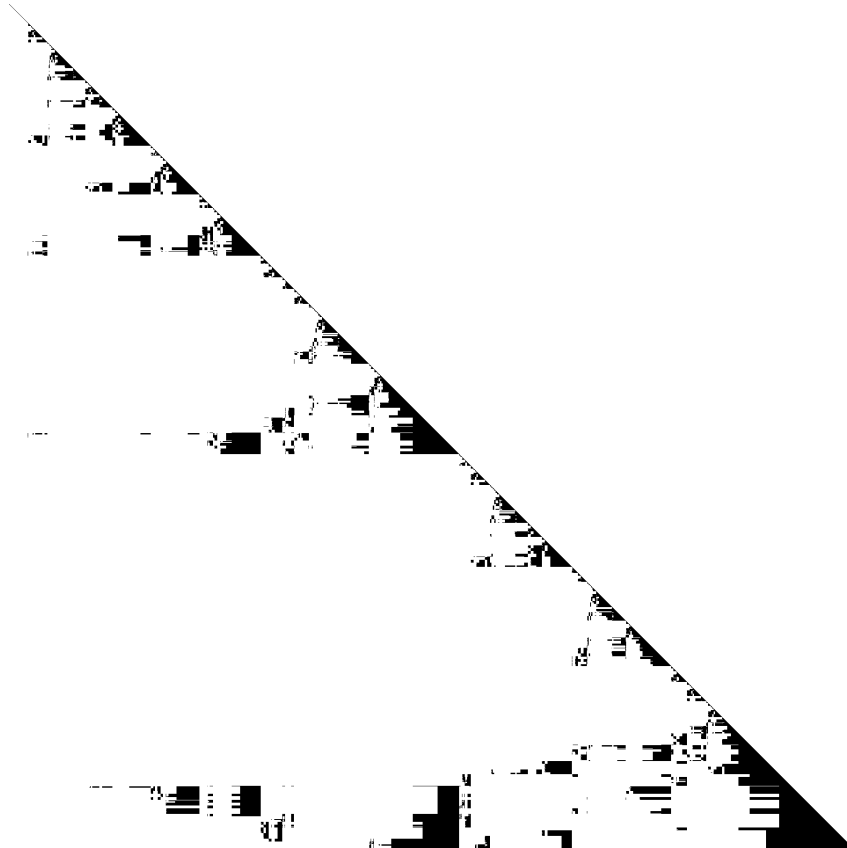
- Overview:

name	side-length: $n$	nonzeros in $A$ : $m$	column indices	sparsity factor: $\alpha$	compression factor: $\beta$
zfall200	200	4605	2432	0,2214	0,5282
konst200	200	5100	3875	0,2462	0,7598
bcsstk14	1806	112267	17756	0,0678	0,1582

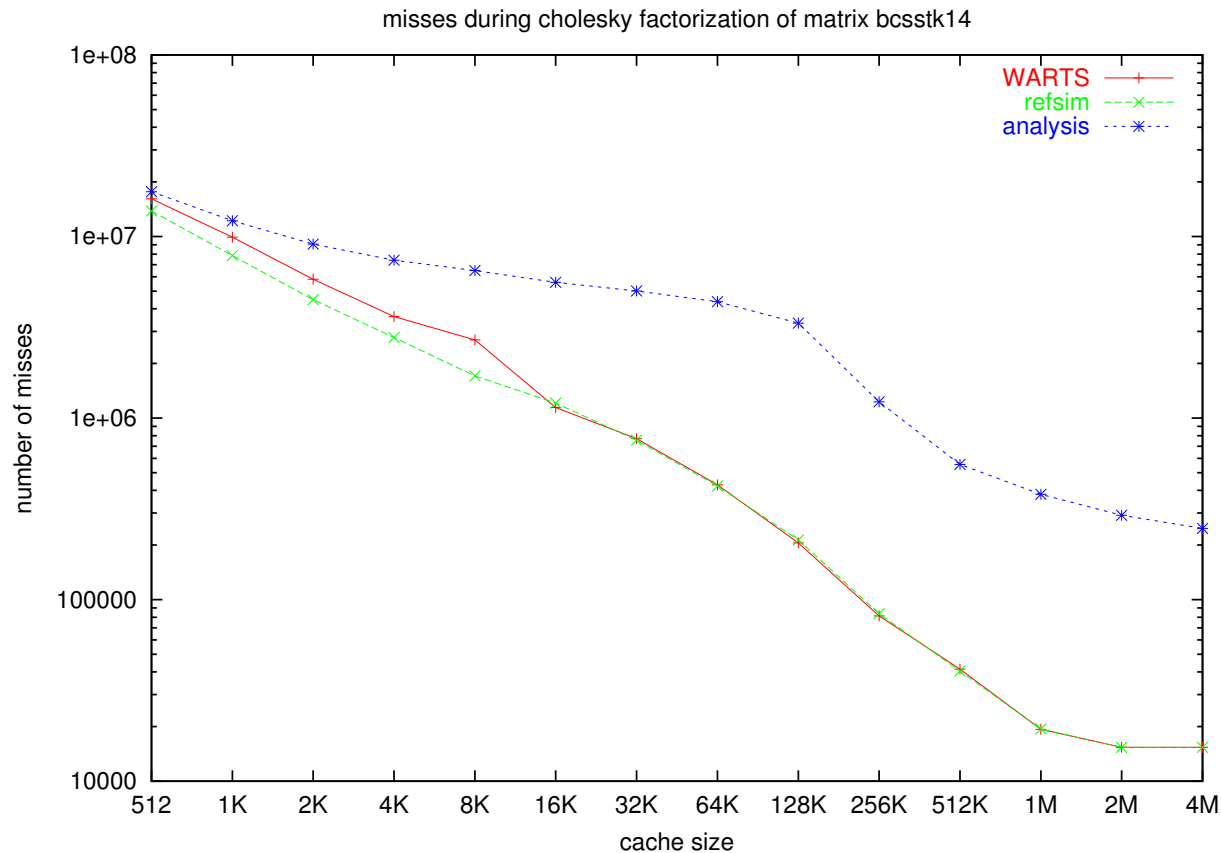
- Structures of matrices  $L$  for the inputs zfall200 and konst200:



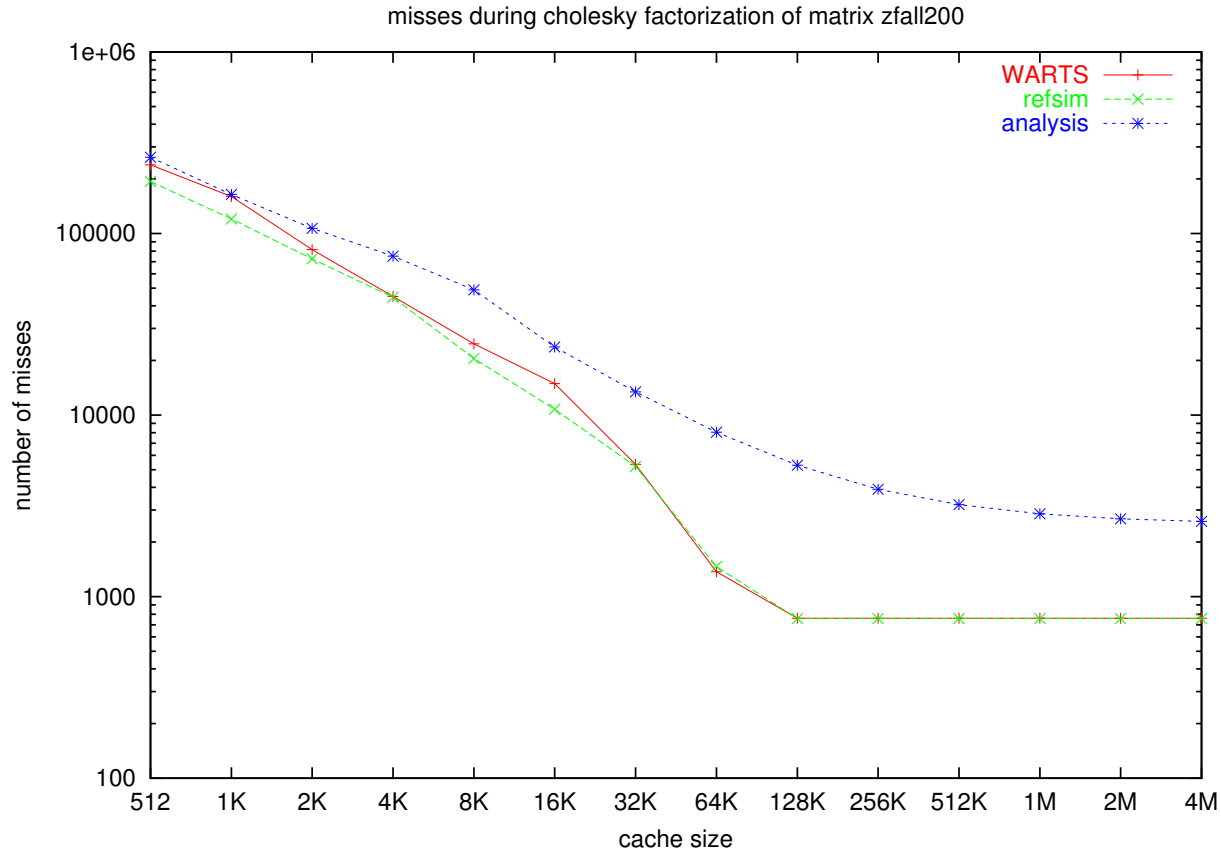
- Structure of matrix  $L$  for the input `bcsstk14`:



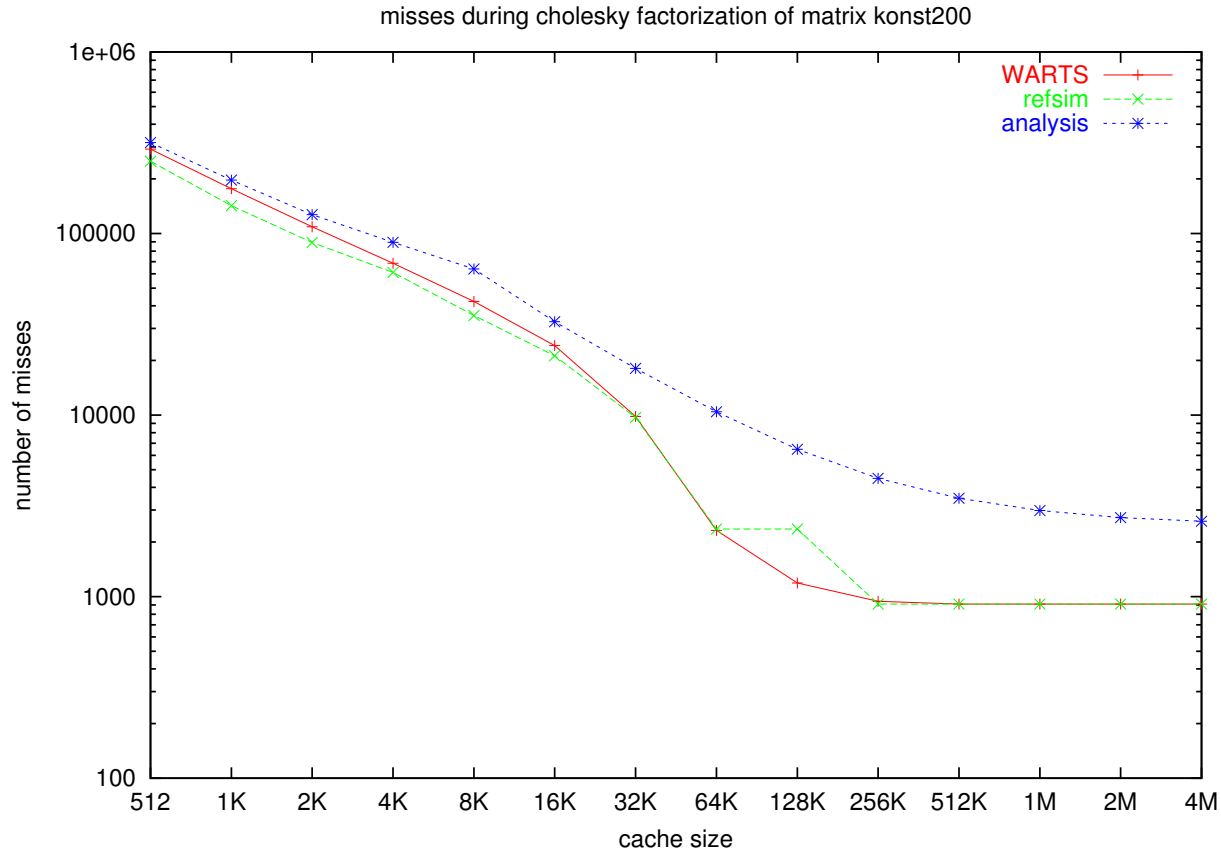
## 4.4. Total misses for bcsstk14



## 4.5. Total misses for zfall200



## 4.6. Total misses for konst200



## 4.7. Evaluation of total misses

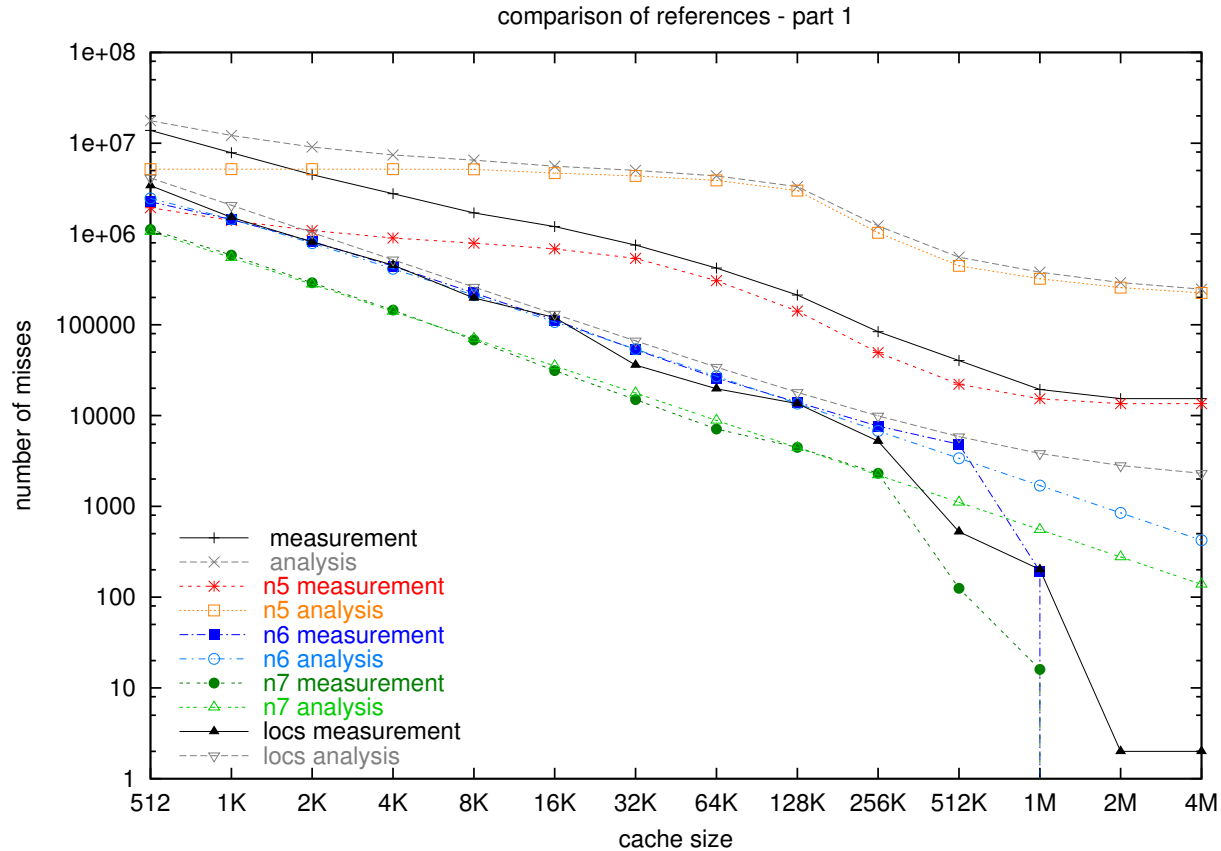
- Deviations with large caches:
  - basic assumption does not hold, if, for example, all data fits into the cache

– size of data:

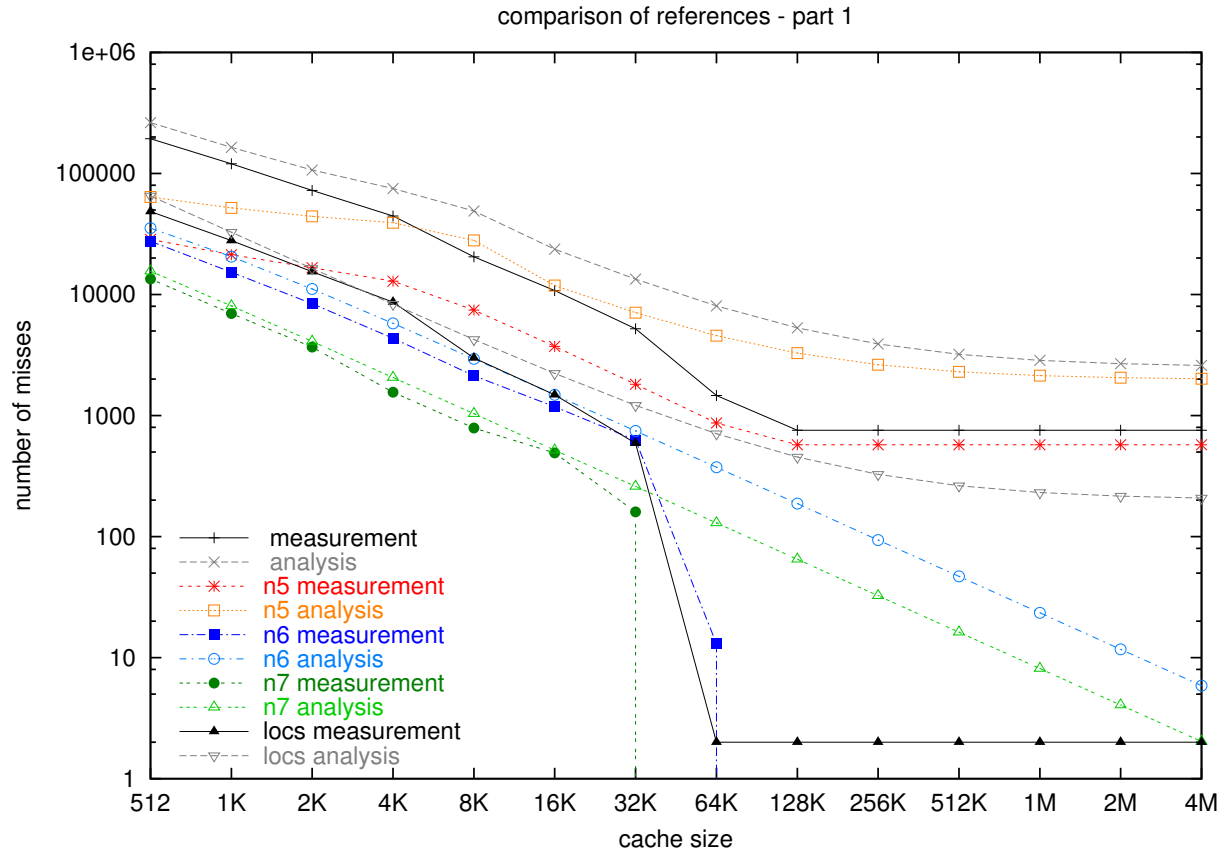
name	bytes	lines
bcsstk14	983692	15371
zfall200	48252	754
konst200	57984	906

- cache size of measured sudden decrease in the number of misses is concordant with size of data ( $\leftarrow$  data stored consecutively)
- Large differences between measurements and analysis:
  - small  $\alpha$  display large over estimations (bcsstk14  $\leftrightarrow$  zfall200)  $\rightarrow$  4.3
    - \* uneven distribution of nonzeros (fill-in)
    - \* smaller deviations with uniform distribution in konst200
  - deviation grows with matrix size ( $\leftrightarrow$  large reuse distances)
  - deviation grows with cache size ( $\leftrightarrow$  large reuse distances)

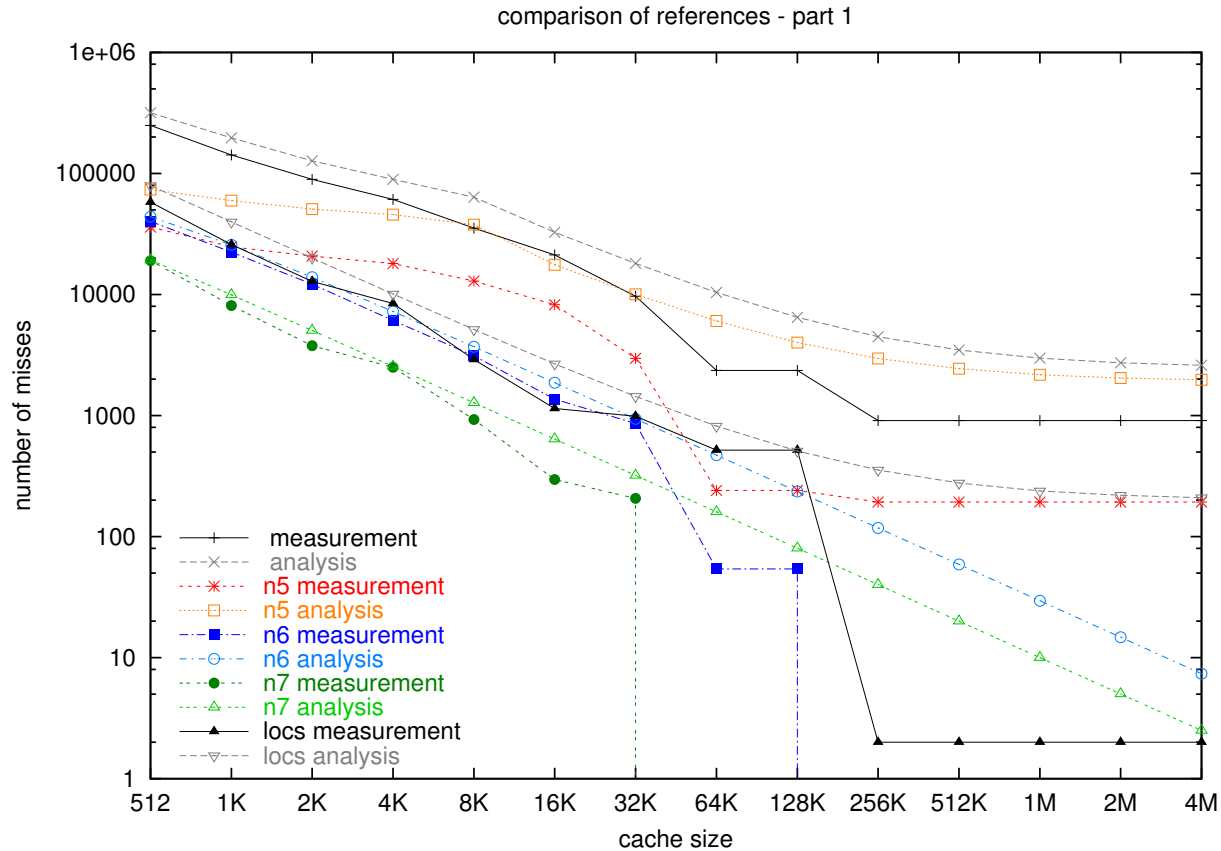
## 4.8. Comparison of references for bcsstk14



## 4.9. Comparison of references for zfall200



## 4.10. Comparison of references for konst200



## 5. Conclusions and Outlook

### 5.1. Conclusions

- Method: developed, applied, constant time
- matrix multiplication: very good estimations
- Cholesky factorization: good estimations, problematic references
- automation: references, reuse types, classification, interference patterns

### 5.2. Outlook

- Cholesky factorization in more detail: large reuse distances, nonzero distribution
- Automation: partial, integration in compiler
- Cache architectures: associativity, hierarchy, instruction / trace cache

# Inhalt

<b>1 Introduction</b>	<b>2</b>	<b>6 Einleitung</b>	<b>24</b>	8.4.6 Einzelvergleich für Matrixgröße 40 - Teil 2	58
1.1 Cache	2	6.1 Cache	25	8.4.7 Einzelvergleich für Matrixgröße 200 - Teil 1	59
1.2 Objectives	3	6.2 Terminologie	26	8.4.8 Einzelvergleich für Matrixgröße 200 - Teil 2	60
<b>2 Method</b>	<b>4</b>	6.3 Ziele	27	8.4.9 Einzelvergleich für Matrixgröße 1000 - Teil 1	61
2.1 Basic assumption	4	<b>7 Methode</b>	<b>28</b>	8.4.10 Einzelvergleich für Matrixgröße 1000 - Teil 2	62
2.2 Procedure	5	7.1 Grundannahme	29	8.4.11 Auswertung Einzelvergleich	63
<b>3 Matrix multiplication</b>	<b>6</b>	7.2 Wahrscheinlichkeit	30	<b>9 Choleskyzerlegung</b>	<b>64</b>
3.1 Problem	6	7.3 Analyse am Beispiel	31	9.1 Problemstellung	65
3.2 Data structure	7	7.4 Wiederverwendungstypen	33	9.2 Right-looking Strategie	66
3.3 Tests	8	7.4.1 Beispiel Wiederverwendungstypen	33	9.3 Dünn besetzte Choleskyzerlegung	67
3.4 Misses during matrix multiplication	9	7.4.2 Algorithmus Wiederverwendungstypen	34	9.4 Programm	68
<b>4 Cholesky factorization</b>	<b>10</b>	7.5 Zugriffsklassen	35	9.5 Tests	69
4.1 Problem	10	7.6 Wiederverwendungsdistanzen	36	9.5.1 Beispielprobleme	70
4.2 Sparse Cholesky factorization	11	7.7 Interferenzen	37	9.5.2 Gesamtzahlen für bcsstk14	72
4.3 Tested input matrices	12	7.8 Interferenzmuster	38	9.5.3 Gesamtzahlen für zfall200	73
4.4 Total misses for bcsstk14	14	7.8.1 Sequentielle Interferenz	39	9.5.4 Gesamtzahlen für konst200	74
4.5 Total misses for zfall200	15	7.8.2 Partielle Interferenz	40	9.5.5 Auswertung Gesamtzahlen	75
4.6 Total misses for konst200	16	7.8.3 Selbstinterferenz	41	9.5.6 Einzelvergleich für bcsstk14 Teil 1	76
4.7 Evaluation of total misses	17	7.8.4 Begrenzte Interferenz	42	9.5.7 Einzelvergleich für bcsstk14 Teil 2	77
4.8 Comparison of references for bcsstk14	18	7.8.5 Interpolation	43	9.5.8 Einzelvergleich für bcsstk14 Teil 3	78
4.9 Comparison of references for zfall200	19	7.8.6 Gleichverteilte begrenzte Interferenz	44	9.5.9 Einzelvergleich für bcsstk14 Teil 4	79
4.10 Comparison of references for konst200	20	7.8.7 Gewichtete begrenzte Interferenz	46	9.5.10 Einzelvergleich für zfall1200 Teil 1	80
<b>5 Conclusions and Outlook</b>	<b>21</b>	7.9 Vorgehensweise	47	9.5.11 Einzelvergleich für zfall1200 Teil 2	81
5.1 Conclusions	21	<b>8 Matrixmultiplication</b>	<b>48</b>	9.5.12 Einzelvergleich für zfall1200 Teil 3	82
5.2 Outlook	21	8.1 Problemstellung	49	9.5.13 Einzelvergleich für zfall1200 Teil 4	83
		8.2 Datenstruktur	50	9.5.14 Einzelvergleich für konst200 Teil 1	84
		8.3 Programm	51	9.5.15 Einzelvergleich für konst200 Teil 2	85
		8.4 Tests	52	9.5.16 Einzelvergleich für konst200 Teil 3	86
		8.4.1 Gesamtzahlen für Matrixgröße 40	53	9.5.17 Einzelvergleich für konst200 Teil 4	87
		8.4.2 Gesamtzahlen für Matrixgröße 200	54	9.5.18 Einzelvergleich	88
		8.4.3 Gesamtzahlen für Matrixgröße 1000	55	<b>10 Zusammenfassung</b>	<b>89</b>
		8.4.4 Auswertung Gesamtzahlen	56	<b>11 Ausblick</b>	<b>90</b>
		8.4.5 Einzelvergleich für Matrixgröße 40 - Teil 1	57		

## Übersicht

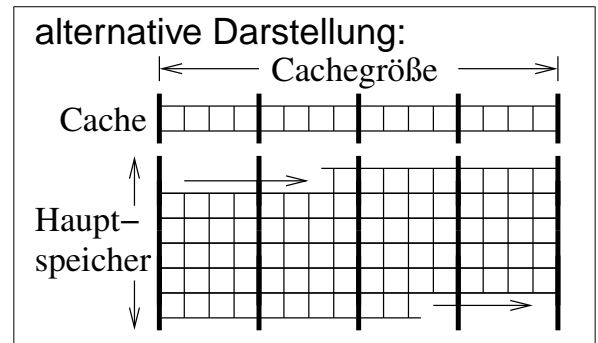
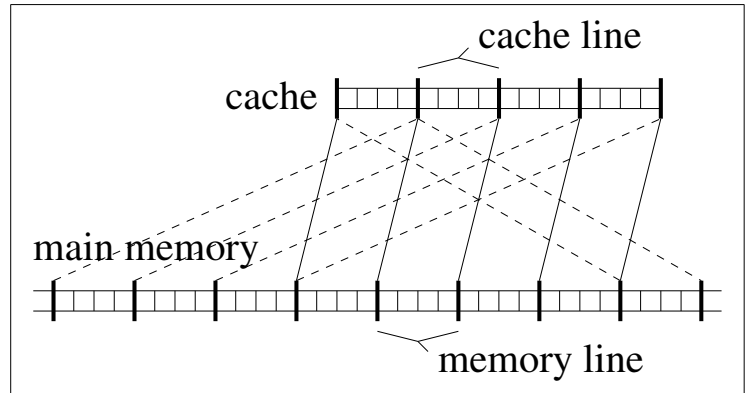
- 6 Einleitung
- 7 Methode
- 8 Matrixmultiplication
- 9 Choleskyzerlegung
- 10 Zusammenfassung
- 11 Ausblick

## 6. Einleitung

- 6.1 Cache
- 6.2 Terminologie
- 6.3 Ziele

## 6.1. Cache

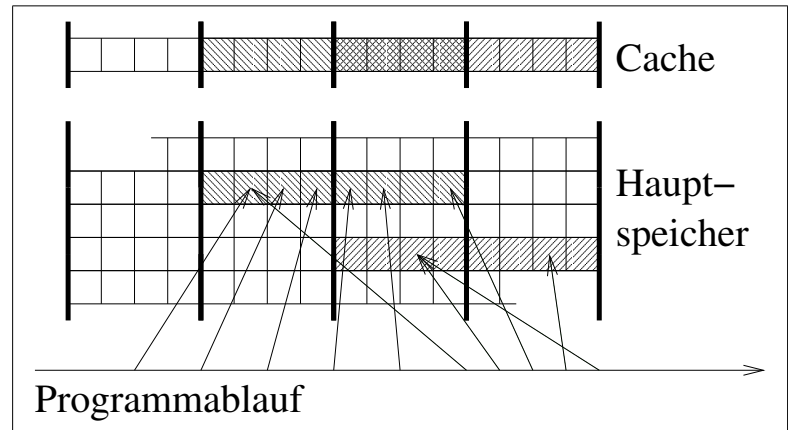
- schneller, teurer, kleiner Speicher
- Assoziativität (direct mapped)
- Rückschreibestrategie
- Zuständigkeit (Daten)
- Cache-Hierarchie
- Beispiel: Intel Pentium 4
  - Zeilengröße: 64 Bytes
  - Speicherzugriff (1,5 GHz / 100 MHz):  
102 - 192 CPU-Zyklen



Caches	Stufe 1	Stufe 2
Größe	64 KB	256 KB / 512 KB
Assoz.	4-fach	8-fach
Zyklen	2 - 9	7

## 6.2. Terminologie

- Cache-Treffer  $\leftrightarrow$  Cache-Fehlzugriff
- Zwanghafte und Ersetzungs- (Konflikt- / Kapazitäts-) Fehlzugriffe
- Wiederverwendung, Quellzugriff, Wiederverwendungsdistanz
- Zeitliche und räumliche (sequentielle) Wiederverwendung
- Referenz, Gruppen- und Selbst-Wiederverwendung
- Lokalität und Interferenz



## 6.3. Ziele

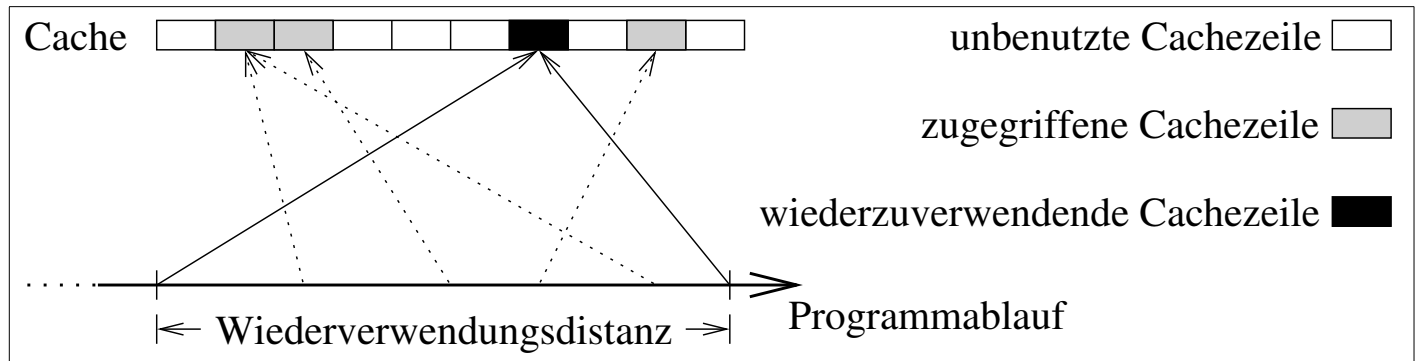
- Entwicklung einer Methode zur Bestimmung der Fehlzugriffe:
  - basierend auf Analyse des Quellprogramms
  - für einstufigen, direct mapped Cache
  - Ergebnis: in konstanter Zeit auswertbare Berechnungsvorschrift
    - \* Parameter zur Charakterisierung der Architektur
    - \* Parameter zur Charakterisierung der Eingabe
- Anwendung der Methode bei dünn besetzten Matrixoperationen:
  - Matrixmultiplication
  - Choleskyzerlegung
- Ansätze zur Automatisierung

## 7. Methode

- 7.1 Grundannahme
- 7.4 Wiederverwendungstypen
- 7.5 Zugriffsklassen
- 7.6 Wiederverwendungsdistanzen
- 7.7 Interferenzen
- 7.8 Interferenzmuster
- 7.9 Vorgehensweise

## 7.1. Grundannahme

Die Wahrscheinlichkeit für einen Fehlzugriff entspricht dem relativen Anteil des Caches, auf den seit dem letzten Zugriff auf die wiederzuverwendende Speicherzeile zugegriffen wurde.



## 7.2. Wahrscheinlichkeit

- $\sigma$ -Algebra  $\mathcal{A}: \mathcal{A} \subset 2^\Omega$ :
  - A1)  $\Omega \in \mathcal{A}$
  - A2)  $A \in \mathcal{A} \Rightarrow \bar{A} \in \mathcal{A}$
  - A3)  $A_i \in \mathcal{A}, i \in \mathbb{N} \Rightarrow \bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$
- Wahrscheinlichkeitsmaß  $P : \mathcal{A} \rightarrow [0, 1]$ 
  - W1)  $P(\Omega) = 1$
  - W2)  $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$ , wenn  $\forall i \neq j \in \mathbb{N} : A_i \cap A_j = \emptyset$
- Messraum:  $(\Omega, \mathcal{A})$ ,
- Wahrscheinlichkeitsraum:  $(\Omega, \mathcal{A}, P)$
- Laplace-Annahme: gleiche Wahrscheinlichkeit für jedes Elementarereignis  $\{a\} \in \Omega$ 
  - $P(A) = \frac{|A|}{|\Omega|}$
  - $\Omega$ : Menge der Elementarereignisse
  - $A \subset \Omega$ : Ereignis
  - $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Unabhängigkeit:
  - $A_1, \dots, A_n$  unabhängig  $\Leftrightarrow \forall I \subset \{1, \dots, n\} : P(\bigcap_{i \in I} A_i) = \prod_{i \in I} P(A_i)$
  - $A, B$  unabhängig  $\Leftrightarrow P(A \cap B) = P(A) \cdot P(B)$
- Bedingte Wahrscheinlichkeit:
  - $P(A|B) = \frac{P(A \cap B)}{P(B)}$

### 7.3. Analyse am Beispiel

- Beispielprogramm:
- Referenzen:  $b_1, a_1, b_2, \dots$
- Wiederverwendungstypen für Referenz  $b_2$ :
  - a) zeitlich Gruppe mit  $b_1$
  - b) sequentiell selbst bezüglich  $j$
  - c) zeitlich selbst bezüglich  $k$

```

for(i = 0; i < n; i++)
    block[i]1 = i * i - 1;
for(j = 0; j < n; j++)
    for(k = 0; k < n; k++)
        antwort[k]1 = block[j]2 + k;
    
```

- Klassifikation für  $b_2$ : Start mit  $(n^2, \{0, a, b, c\}, ) \rightarrow (n^2, \{b, c\}, a)$ :
  - 1  $(n, \{b\}, a)$ : 1. Iteration von  $k$  ( $k=0$ )
    - 1.1  $(1, \emptyset, a)$ : 1. Iteration von  $j$  ( $j=0$ )
    - 1.2  $(n-1, \{b\}, a)$ : Folge-Iteration von  $j$  ( $j>0$ )
      - 1.2.1  $(P1 \cdot (n-1), \emptyset, b)$ : selbe Zeile
      - 1.2.2  $((1-P1) \cdot (n-1), \emptyset, a)$ : neue Zeile
  - 2  $(n^2 - n, \emptyset, c)$ : Folge-Iteration von  $k$  ( $k>0$ )

$$P1 = \frac{\frac{z}{t} - 1}{\frac{z}{t}} = \frac{z-t}{z}, \text{ mit}$$

$z = \text{Zeilengröße}$  und  
 $t = \text{Größe von block}[j]$

## Analyse am Beispiel - Fortsetzung

- Interferenzen:  $\left\{ \begin{array}{l} 1.1 \text{ (Typ a)} : b_1, b_2, i_*, n_*, j_*, k_* \\ 1.2.1 \text{ (Typ b)}: a_1, k_*, n_*, j_* \\ 1.2.2 \text{ (Typ a)}: a_1, b_1, b_2, i_*, n_*, j_*, k_* \\ 2 \text{ (Typ c)} : a_1, k_*, n_x, j_y \end{array} \right.$
- Interferenzwahrscheinlichkeiten für Klasse 1.2.2:

-  $a_1: A_S(n \cdot ts)$

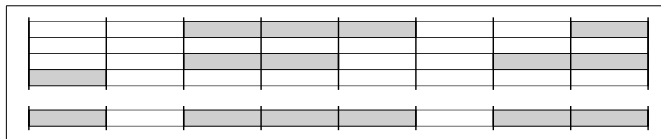
-  $b_1, b_2: A_S(C(n \cdot ts) \cdot cs)$

-  $i, n, j, k: A_{locs}(4 \cdot is)$

```
for(i = 0; i < n; i++)
    block[i]_1 = i * i - 1;
for(j = 0; j < n; j++)
    for(k = 0; k < n; k++)
        antwort[k]_1 = block[j]_2 + k;
```

- Kumulation:  $p \cup q := p + q - p \cdot q$

Bsp.:  $P(1.2.2) = A_S(n \cdot ts) \cup A_S(C(n \cdot ts) \cdot cs) \cup A_{locs}(4 \cdot is)$



## 7.4. Wiederverwendungstypen

### 7.4.1. Beispiel zur Bestimmung der Wiederverwendungstypen

- Beispielprogramm:

```
for(i = 0; i < n; i++) {  
    block[i]1 = i * i - 1;  
}  
for(j = 0; j < n; j++) {  
    for(k = 0; k < n; k++) {  
        antwort[k]1 = block[j]2 + k;  
    }  
}
```

- Referenzen:  $b_1, a_1, b_2, \dots$
- Wiederverwendungstypen für Referenz  $b_2$ :
  - a) zeitlich Gruppe mit  $b_1$
  - b) sequentiell selbst bezüglich  $j$
  - c) zeitlich selbst bezüglich  $k$

## 7.4.2. Algorithmus zur Bestimmung der Wiederverwendungstypen

```
Menge Wiederverwendungstypen(Referenz  $r$ ) {
   $W := \emptyset$ ;
  Für_alle( $s \in R$ ) {
    /* Wiederverwendungen mit Quellzugriff über Referenz  $s$ : */
     $Q := \{(z, q) \mid z \in Z(r), q \in Z(s) \text{ mit } q \text{ greift vor } z$ 
           $\text{ auf dieselbe Speicherzeile zu}\}$ ;
    /* Wiederverwendungstypen erkennen: */
     $V := \{A \subseteq Q \mid A \text{ enthält alle Paare } (z, q) \in Q,$ 
           $\text{ die eine ähnliche Distanz aufweisen.}\}$ ;
    Für_alle( $A \in V$ ) {
       $D :=$  Beschreibung der Distanzen zwischen den Paaren  $(z, q) \in A$ ;
       $W := W \cup \{T(r, s, D)\}$ ;
    }
  }
  return  $W$ ;
}
```

## 7.5. Zugriffsklassen

- Zuriffen eindeutige Kombination von Wiederverwendungstypen zuordnen
- Wiederverwendungstyp 0 für Zugriffe ohne Wiederverwendung
- Fortlaufende Unterteilung der Zugriffe in Klassen
- Heuristiken: kurze Distanzen zuerst; erste Iterationen getrennt
- Klassification für  $b_2$  im Beispiel aus Abschnitt 7.4.1

**Start:**  $(n^2, \{0, a, b, c\}, ) \rightarrow (n^2, \{b, c\}, a)$

1  $(n, \{b\}, a)$ : 1. Iteration von k

1.1  $((1 - P1) \cdot n, \emptyset, a)$ : neue Zeile

1.2  $(P1 \cdot n, \emptyset, b)$ : in Zeile

2  $(n^2 - n, \emptyset, c)$ : Folge-It. von k

$$P1 = \frac{\frac{z}{i} - 1}{\frac{z}{i}} = \frac{z-t}{z}, \text{ mit}$$

$z$  = Zeilengröße und

$t$  = Größe des Typs von `block[j]`

```
for(i = 0; i < n; i++) {
    block[i]_1 = i * i - 1;
}
for(j = 0; j < n; j++) {
    for(k = 0; k < n; k++) {
        antwort[k]_1 = block[j]_2 + k;
    }
}
```

- a) zeitlich Gruppe mit  $b_1$
- b) sequentiell selbst bezüglich j
- c) zeitlich selbst bezüglich k

## 7.6. Wiederverwendungsdistanzen

- Gegeben: alle  $k$  Zugriffsklassen einer bestimmten Referenz  $r$ 
  - mit  $n_i =$  Anzahl der Zugriffe in Klasse  $i$  für  $i \in \{1, \dots, k\}$  (aus Klassifikation)
  - Gesucht:  $p_i =$  Wahrscheinlichkeit für Fehlzugriff in Klasse  $i$
  - Zur Abschätzung der Anzahl  $f$  der Fehlzugriffe über Referenz  $r$ :  $f = \sum_{i=1}^k p_i \cdot n_i$
- Bestimmung der  $p_i$  über Wiederverwendungsdistanz in Klasse  $i$ :
  - konstante Distanz: Wahrscheinlichkeit gemäß Grundannahme (7.1)
  - regelmäßig variable Distanz: Mittel der Wahrscheinlichkeiten, alternativ: Wahrscheinlichkeit für mittlere Distanz
  - unregelmäßig variable Distanz: feinere Klassifizierung, alternativ: gröbere Abschätzung
- Beispiel: Wiederverwendungstyp a in Klasse 1 . 1 (Abschnitt 7.5):  
Mittlere Distanz (in Iterationen der Schleife über  $j$ ):  $\frac{1}{n} \cdot \sum_{j=0}^{n-1} (n-1-j) = \frac{n-1}{2}$

## 7.7. Interferenzen

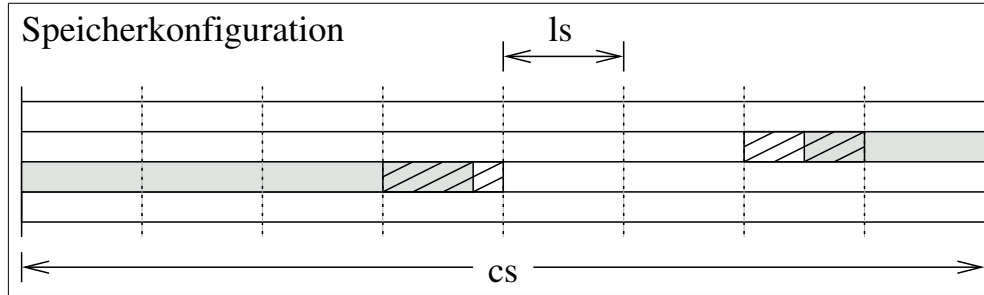
- Interferenz: innerhalb Wiederverwendungsdistanz verwendete Referenz
- Interferenzmengen: Menge von Interferenzen, über die auf gleichen Speicherbereich zugegriffen wird. (Bsp.:  $x = r[i] + r[i + 1]$ )
- Interferenzwahrscheinlichkeit: über Interferenzmenge zugegriffener relativer Anteil des Caches ( $\rightarrow$  Zugriffe innerhalb Wiederverwendungsdistanz)
- $M_d$ : Menge der Interferenzwahrscheinlichkeiten für alle innerhalb der Wiederverwendungsdistanz auftretenden Interferenzmengen
- Annahme: Interferenzwahrscheinlichkeiten in  $M_d$  voneinander unabhängig
- Kumulation zweier Interferenzwahrscheinlichkeiten:  $A_1 \cup A_2 = A_1 + A_2 - A_1 \cdot A_2$
- Wahrscheinlichkeit für Fehlzugriff nach Wiederverwendungsdistanz:  $p_d = \bigcup_{A \in M_d} A$

## 7.8. Interferenzmuster

- 7.8.1 Sequentielle Interferenz
- 7.8.2 Partielle Interferenz
- 7.8.3 Selbstinterferenz
- 7.8.4 Begrenzte Interferenz
- 7.8.5 Interpolation
- 7.8.6 Gleichverteilte begrenzte Interferenz
- 7.8.7 Gewichtete begrenzte Interferenz

## 7.8.1. Sequentielle Interferenz

- Zugriff auf alle Speicherzeilen eines zusammenhängenden Bereichs von  $n$  Bytes

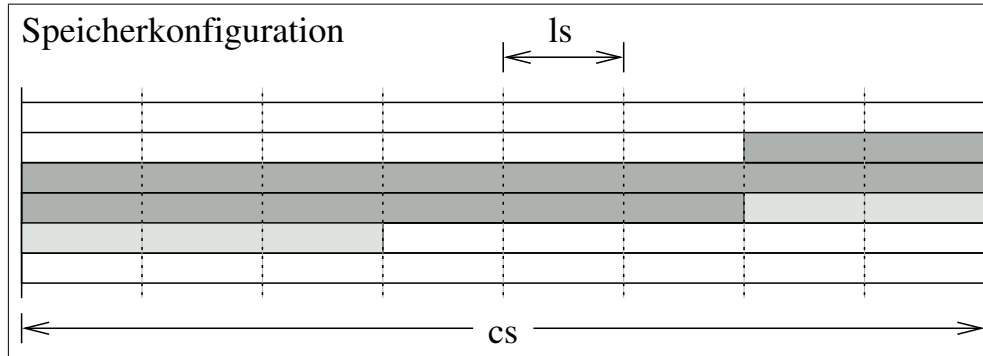


- Interferenzwahrscheinlichkeit:

$$A_s(n) = \max \left\{ 1, \frac{n + ls - 1}{cs} \right\}$$

## 7.8.2. Partielle Interferenz

- Zugriff auf einen zusammenhängenden Bereich von  $n$  Bytes, wobei jede Speicherzeile des Bereichs mit der Wahrscheinlichkeit  $p$  getroffen wird.

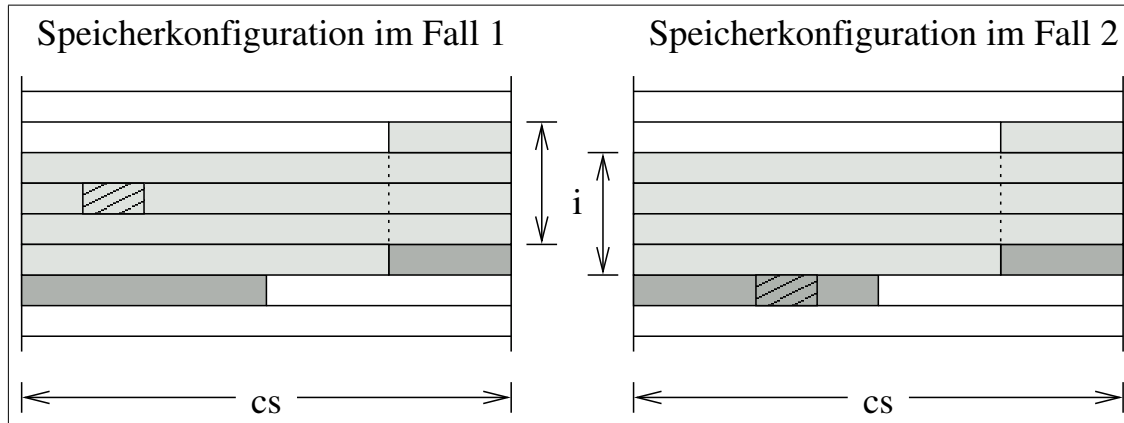


- Interferenzwahrscheinlichkeit:

$$A_p(n, p) = 1 - (1 - p)^{\lfloor \frac{n}{cs} \rfloor} \cdot \left( 1 - \left( \frac{n}{cs} - \lfloor \frac{n}{cs} \rfloor \right) \cdot p \right)$$

### 7.8.3. Selbstinterferenz

- geringere Interferenz, wenn wiederzuverwendende Speicherzeile innerhalb des interferierenden Speicherbereichs von  $n$  Bytes liegt:



- Größe des interferierenden Bereichs nur  $C(n) \cdot cs$  statt  $n$ , wobei mit  $i = \lfloor \frac{n}{cs} \rfloor$  gilt:

$$C(n) = \frac{\left(\frac{i \cdot cs}{ls}\right)}{\left(\frac{n}{ls}\right)} \cdot \left(i - 1 + \frac{(n - i \cdot cs)}{cs}\right) + \frac{\left(\frac{n - i \cdot cs}{ls}\right)}{\left(\frac{n}{ls}\right)} \cdot i = \frac{\lfloor \frac{n}{cs} \rfloor \cdot \left(2 \cdot \frac{n}{cs} - \lfloor \frac{n}{cs} \rfloor - 1\right)}{\frac{n}{cs}}$$

## 7.8.4. Begrenzte Interferenz

- Wiederzuverwendende Speicherzeile wird innerhalb Distanz  $d_{max}$  mindestens einmal verwendet:
  - mögliche Wiederverwendungsdistanzen  $d \in \{1, \dots, d_{max}\}$
  - Zugriff vor Distanz  $d$  mit Wahrscheinlichkeit  $P$
  - Interferenzwahrscheinlichkeit  $A(d)$  für bestimmtes  $d$  bekannt
- Beispiel:  $t_2$  weist zeitliche Gruppen-Wiederverwendung mit  $t_1$  auf:

```
for(i = 0; i < n; i++) {  
    if(zufall() < P) t1 = i;  
}  
ergebnis = t2 + 1;
```

- Interferenzwahrscheinlichkeit:

$$A_b(A(d), d_{max}, P) = \frac{1}{1 - (1 - P)^{d_{max}}} \cdot \sum_{d=1}^{d_{max}} (1 - P)^{d-1} \cdot P \cdot A(d)$$

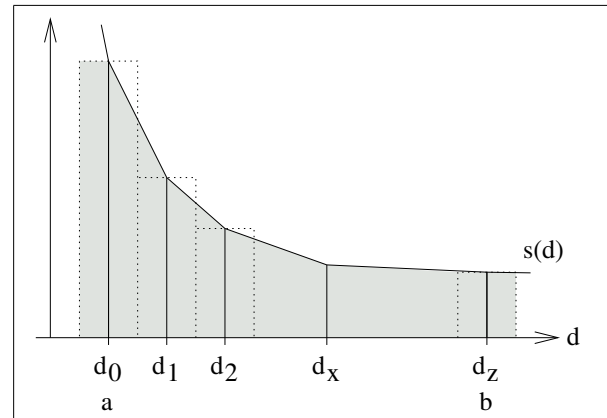
## 7.8.5. Interpolation

- Interpolation zur Abschätzung von Summen  $\sum_{d=a}^b s(d)$  in konstanter Zeit, wobei  $a < b$  und  $s(d)$  links-gekrümmt, monoton-fallend
- exponentiell wachsende Stützstellen  $d_x$  mit Mindestabstand 1 für  $x \in \{0, \dots, z\}$ :

$$d_x = \max \left( a + x, a - 1 + e^{\frac{x}{z} \cdot \ln(b-a+1)} \right)$$

- Interpolation (für z.B.  $z = 10$  konstant):

$$\sum_{d=a}^b s(d) \approx \frac{s(a) + s(b)}{2} + \sum_{x=0}^{z-1} \frac{s(d_x) + s(d_{x+1})}{2} \cdot (d_{x+1} - d_x)$$



## 7.8.6. Gleichverteilte begrenzte Interferenz

- Mehrfache begrenzte Interferenz mit folgenden Eigenschaften:
  - mögliche Distanz  $d \in \{1, \dots, d_{max}\}$ , pro Zugriff variierendes  $d_{max} \in \{1, \dots, n\}$
  - gleiche Wahrscheinlichkeit für alle  $d_{max} \in \{1, \dots, n\}$
  - Zugriff vor Distanz  $d$  mit Wahrscheinlichkeit  $P$
  - Interferenzwahrscheinlichkeit  $A(d)$  für bestimmtes  $d$  bekannt
- Beispiel:  $t_1$  weist wiederholte zeitliche Selbst-Wiederverwendung auf:

```
for(i = 0; i < n; i++)
    if(zufall() < P) t1 = i;
ergebnis = t2 + i;
```

- Interferenzwahrscheinlichkeit (arithmetisches Mittel):

$$A_g(A(d), n, P) = \frac{1}{n} \cdot \sum_{i=1}^n A_b(A(d), i, P) = \frac{P}{n} \cdot \sum_{d=1}^n (1-P)^{d-1} \cdot A(d) \cdot \sum_{i=d}^n \frac{1}{1-(1-P)^i}$$

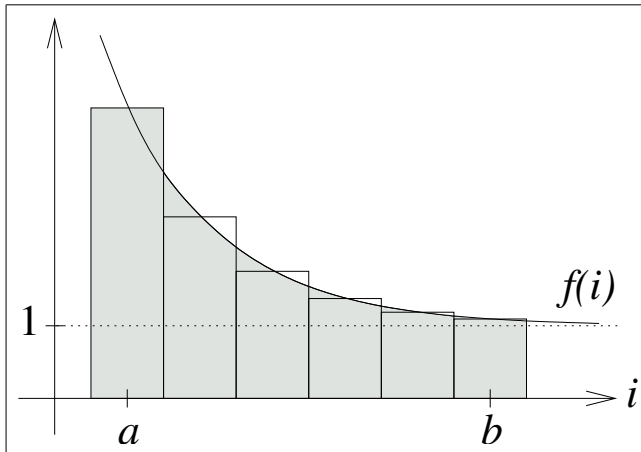
- Problem: Auswertung der beiden Summen in konstanter Zeit!

- innere Summe der Form  $\sum_{i=a}^b \frac{1}{1-q^i}$  mittels Integration nähern:

- Summand  $f(i) = \frac{1}{1-q^i} \Leftrightarrow$  Stammfunktion  $F(i) = i - \frac{1}{\ln(q)} \cdot \ln(1 - q^i)$

- Näherung:

$$\sum_{i=a}^b \frac{1}{1-q^i} \approx f(a) + \int_{a+0,5}^{b+0,5} f(i) di \approx \frac{1}{1-q^a} + b - a - \frac{1}{\ln q} \ln \left( \frac{1 - q^{b+0,5}}{1 - q^{a+0,5}} \right)$$



- äußere Summe mittels Interpolation nähern

## 7.8.7. Gewichtete begrenzte Interferenz

- Mehrfache begrenzte Interferenz mit folgenden Eigenschaften:
  - mögliche Distanz  $d \in \{1, \dots, d_{max}\}$ , pro Zugriff variierendes  $d_{max} \in \{1, \dots, n\}$
  - Häufigkeitsmaß für bestimmtes  $d_{max} \in \{1, \dots, n\}$  gemäß  $g(d_{max})$
  - Zugriff vor Distanz  $d$  mit Wahrscheinlichkeit  $P$
  - Interferenzwahrscheinlichkeit  $A(d)$  für bestimmtes  $d$  bekannt
- Gewichte:  $\forall i \in \{1, \dots, n\} : w(i) = \frac{g(i)}{\sum_{k=1}^n g(k)}$

- Interferenzwahrscheinlichkeit (gewichtetes arithmetisches Mittel):

$$\begin{aligned}
 A_w(A(d), n, P, g(i)) &= \sum_{i=1}^n w(i) \cdot A_b(A(d), i, P) \\
 &= \frac{P}{\sum_{k=1}^n g(k)} \cdot \sum_{d=1}^n (1-P)^{d-1} \cdot A(d) \cdot \sum_{i=d}^n \frac{g(i)}{1 - (1-P)^i}
 \end{aligned}$$

- Abschätzung in konstanter Zeit durch zweifach verschachtelte Interpolation

## 7.9. Vorgehensweise

$R =$  Menge aller Referenzen des untersuchten Programmabschnitts ;

Für\_alle Referenzen  $r \in R$  {

Wiederverwendungstypen von  $r$  bestimmen ;

Klassifizierung der Zugriffe über  $r$ ;  $C =$  Menge der Zugriffsklassen von  $r$  ;

Für\_alle Klassen  $c \in C$  {

$$n[c] = |c|;$$

$$d[c] = \text{mittlere Wiederverwendungsdistanz in } c;$$

Bilden der Interferenzmengen bei Wiederverwendungsdistanz  $d[c]$  ;

$I =$  Menge der Interferenzmengen bei Wiederverwendungsdistanz  $d[c]$  ;

Für\_alle Interferenzmengen  $i \in I$  {

$$p_c[i] = \text{zugegriffener Cacheanteil}(i, d[c]);$$

}

$$p_r[c] = \bigcup_{i \in I} p_c[i];$$

}

$$f[r] = \sum_{c \in C} p_r[c] \cdot n[c];$$

}

$$\text{Anzahl Fehlzugriffe} = \sum_{r \in R} f[r];$$

## 8. Matrixmultiplication

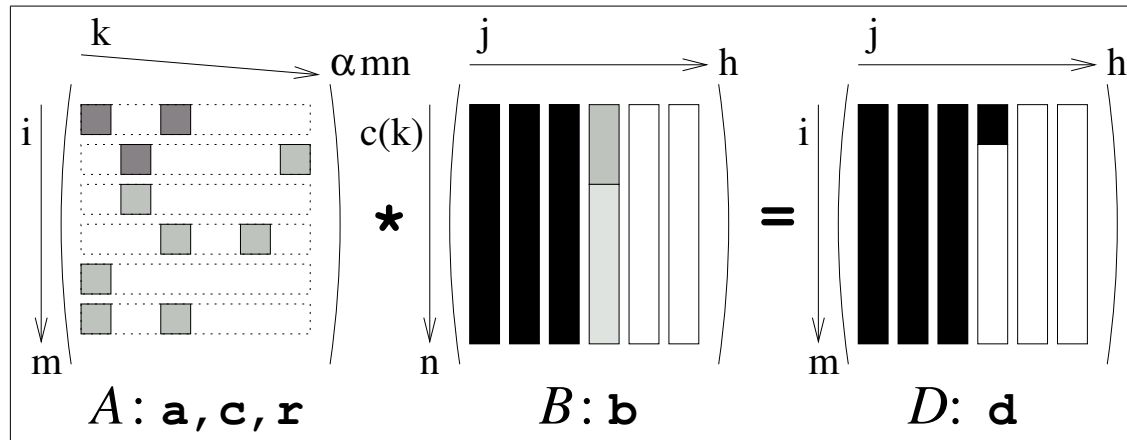
- 8.1 Problemstellung
- 8.2 Datenstruktur
- 8.3 Programm
- 8.4 Tests
  - 8.4.1 Gesamtzahlen
  - 8.4.5 Einzelvergleiche

## 8.1. Problemstellung

- Multiplication einer dünn besetzten Matrix  $A$  mit einer voll besetzten Matrix  $B$ :

$$A \cdot B = D$$

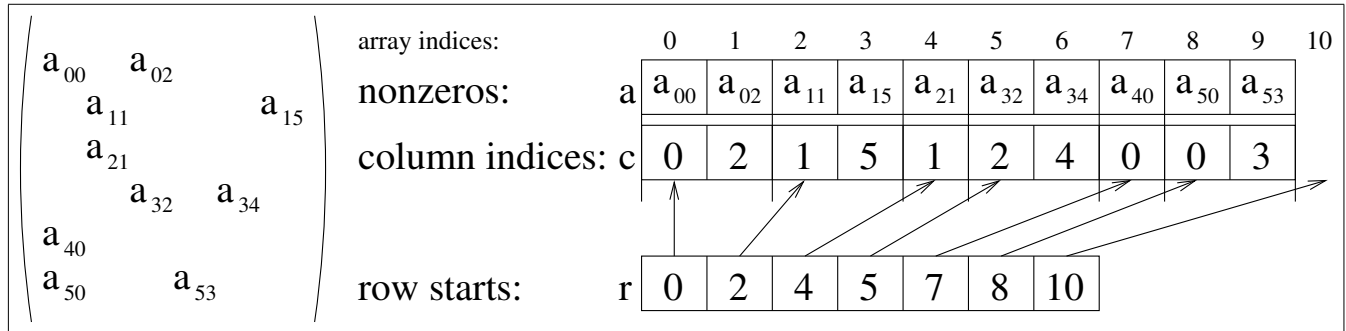
- Veranschaulichung:



- Eingabeparameter für Analyse:  $m, n, h$  und  $\alpha$

## 8.2. Datenstruktur

- voll besetzte Matrizen: spaltenweise hintereinander
- dünn besetzte Matrix: komprimiert, zeilenweise



### 8.3. Programm

```
for(j1 = 0; j2 < h1; j4 = j3 + 1) {  
    for(i1 = 0; i2 < m1; i4 = i3 + 1) {  
        reg1 = 0;  
        for(k1 = r1[i5]; k2 < r2[i6 + 1]; k4 = k3 + 1) {  
            reg3 = reg2 + a1[k5] * b1[j5 * n1 + c1[k6]];  
        }  
        d1[j6 * m2 + i7] = reg4;  
    }  
}
```

## 8.4. Tests

- Messung der Fehlzugriffe zum Vergleich mit Analyse:

Werkzeuge	Datentrace	Fehlzugriffe
WARTS	qpt2	dinerolV
Eigene	instrumentierter Quelltext	refsim

Trace für dinerolV:

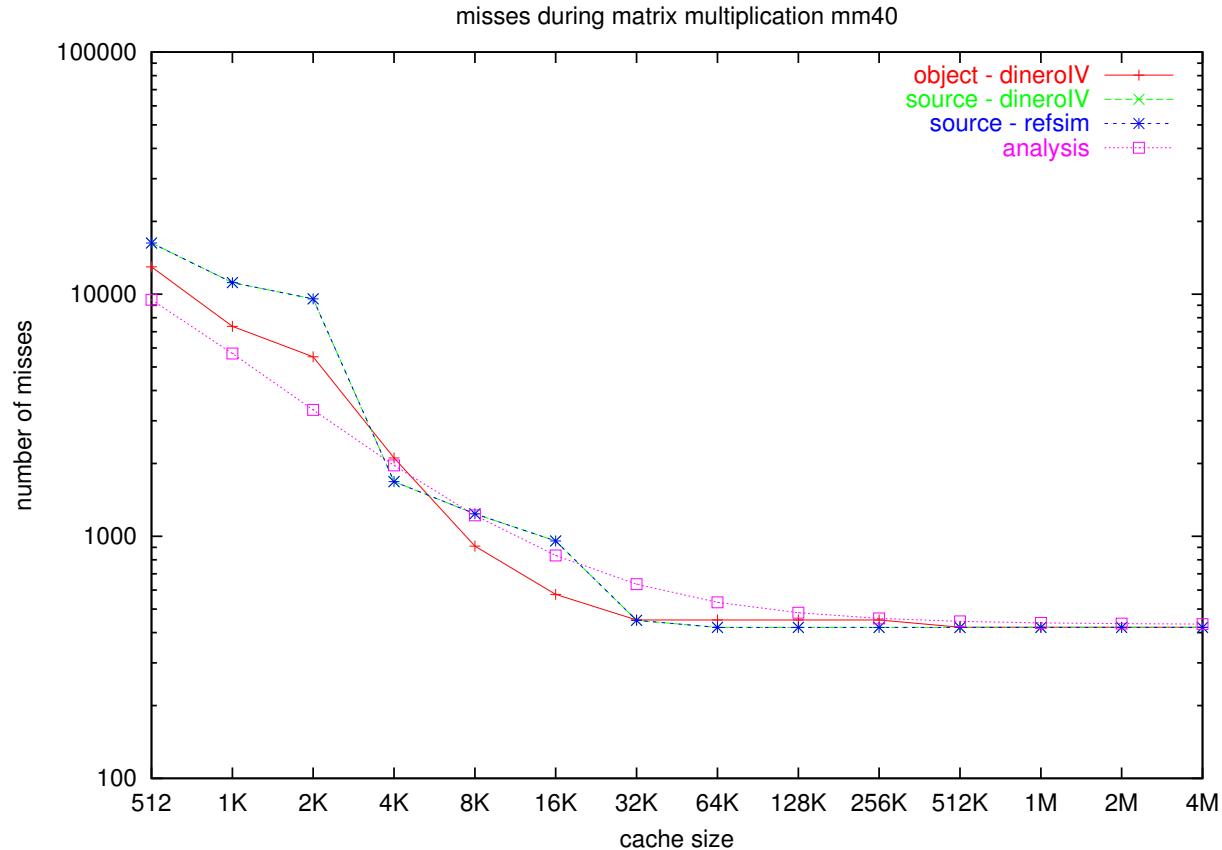
1	ffbeea88
2	11368
0	ffbeea88
2	1136c

Trace für refsim:

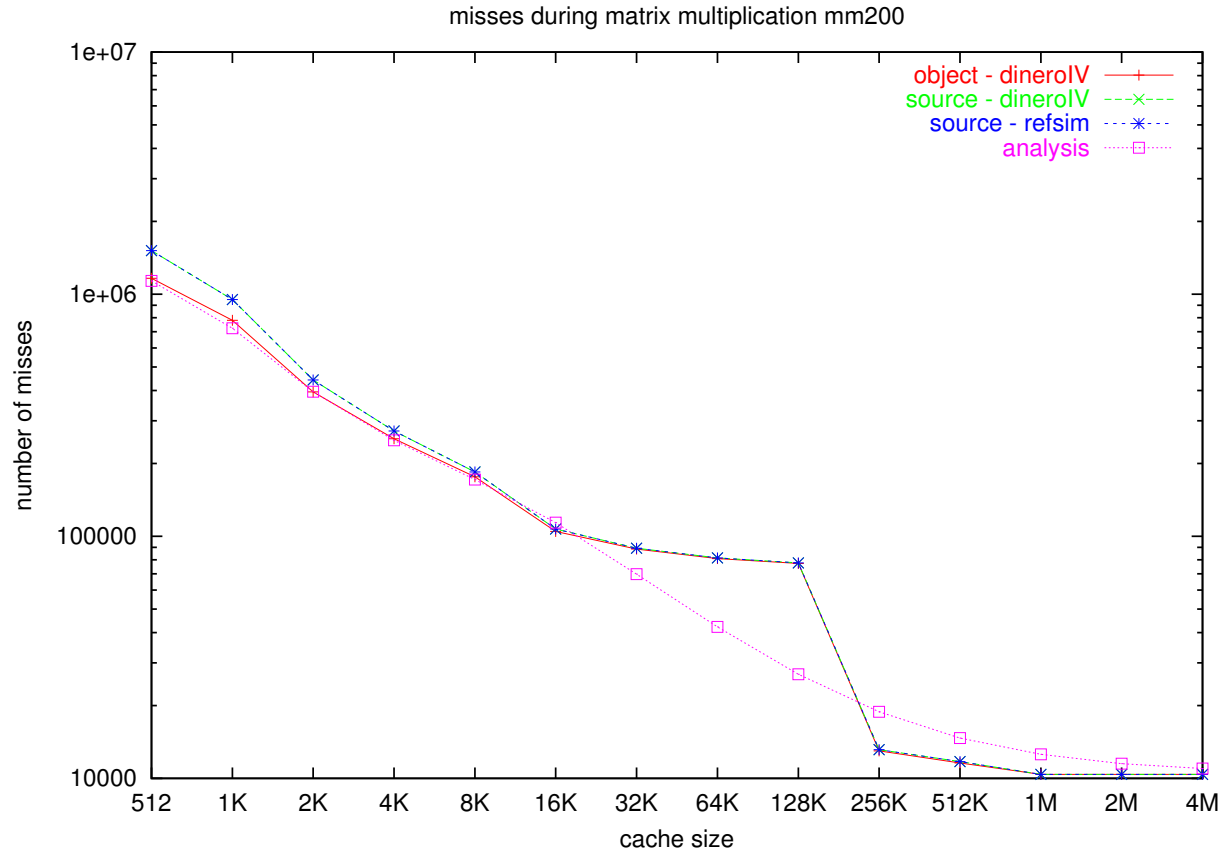
0	ffbeea9c	4	23
0	ffbeea8c	4	44
1	ffbeea60	8	51
0	ffbeea9c	4	26

- Testumgebung:
  - SUN Ultra-60 mit 2 UltraSPARC II Prozessoren unter Solaris 8.0
  - GNU gcc 2.8.1 ohne Optimierung
- Beispielprobleme:  $n = m = h \in \{40, 200, 1000\}$ ,  $\alpha = 0.05$ , zufällig belegt

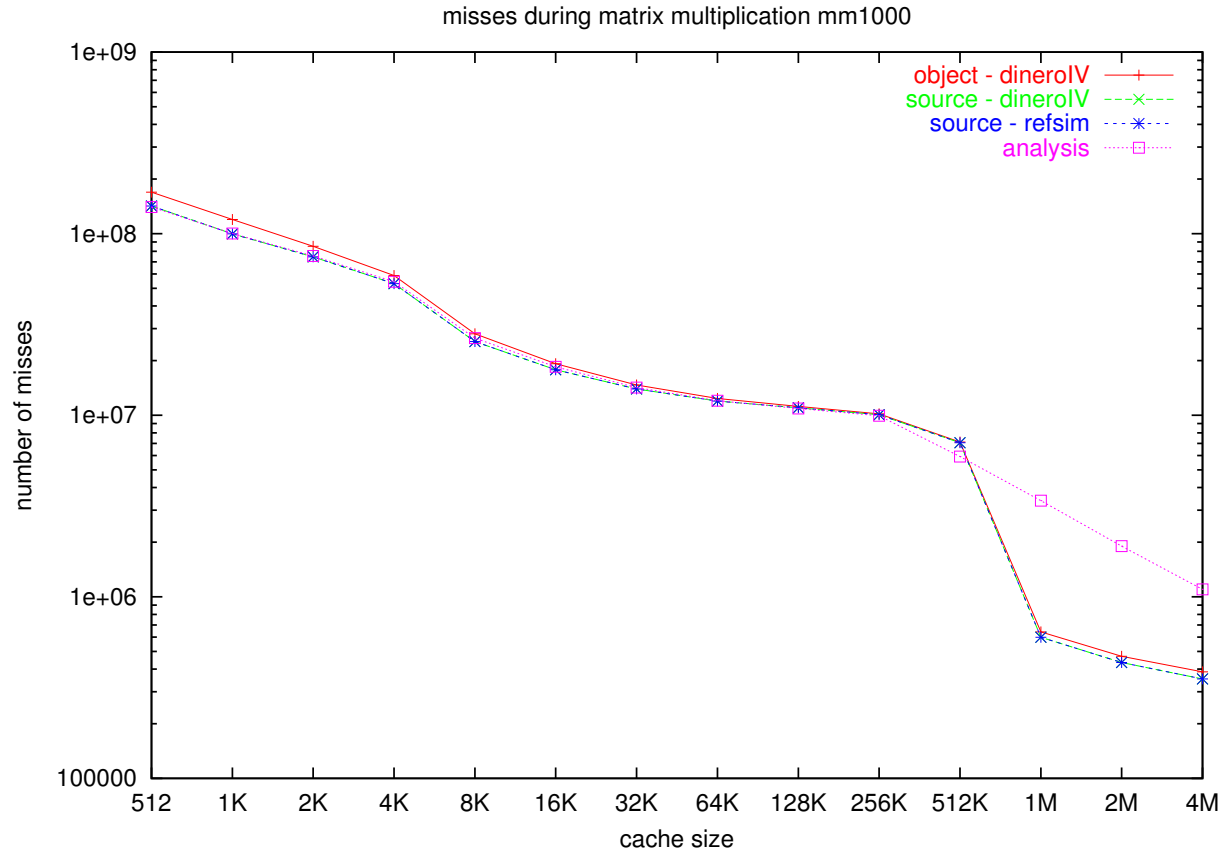
## 8.4.1. Gesamtzahlen für Matrixgröße 40



## 8.4.2. Gesamtzahlen für Matrixgröße 200



### 8.4.3. Gesamtzahlen für Matrixgröße 1000



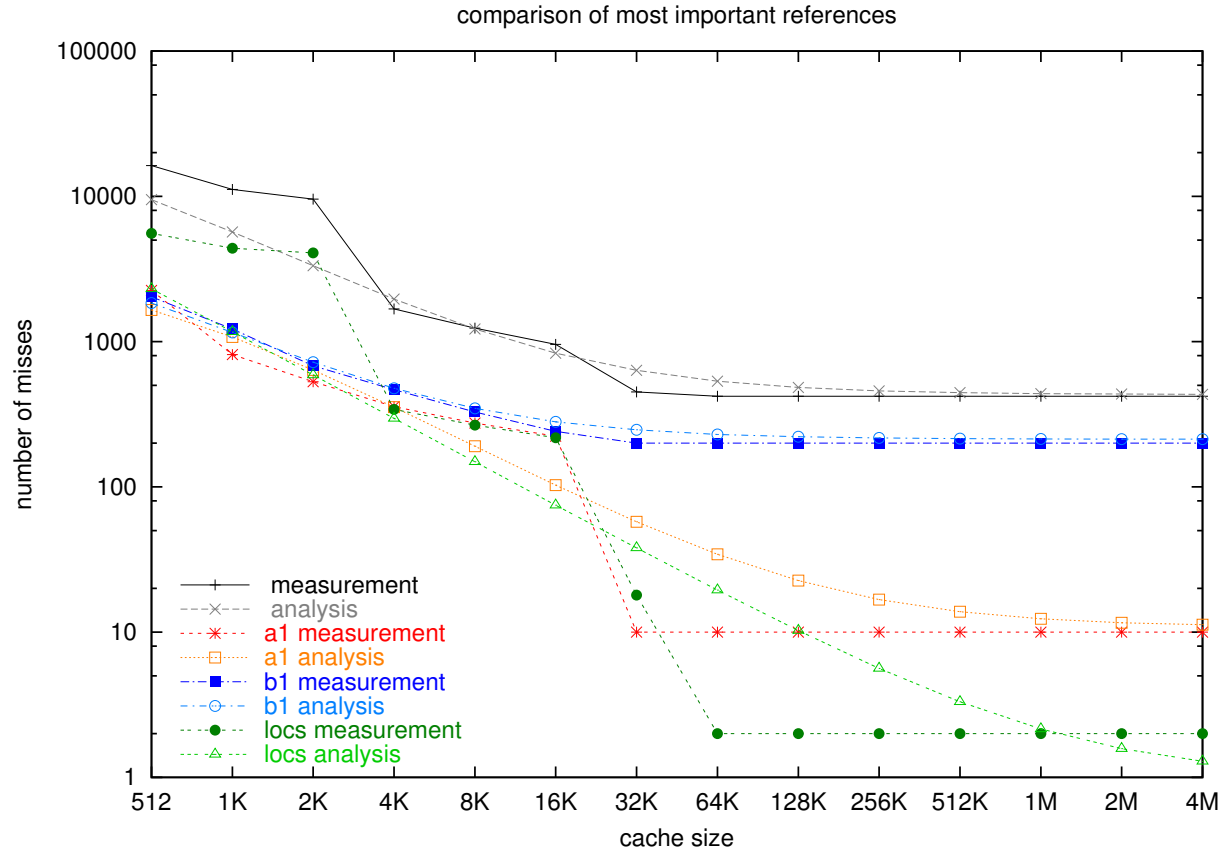
### 8.4.4. Auswertung Gesamtzahlen

- Unterschiede geringer bei größeren Systemen:
  - zufällige Unterschiede
  - qpt2 erfasst mehr Zugriffe (z.B. temporäre Variablen)
- Unterschätzung bei kleinen Caches:
  - qpt2 erfasst mehr Zugriffe (z.B. temporäre Variablen)
- Abweichungen bei großen Caches:
  - Grundannahme greift nicht, wenn z.B. alle Daten nebeneinander im Cache

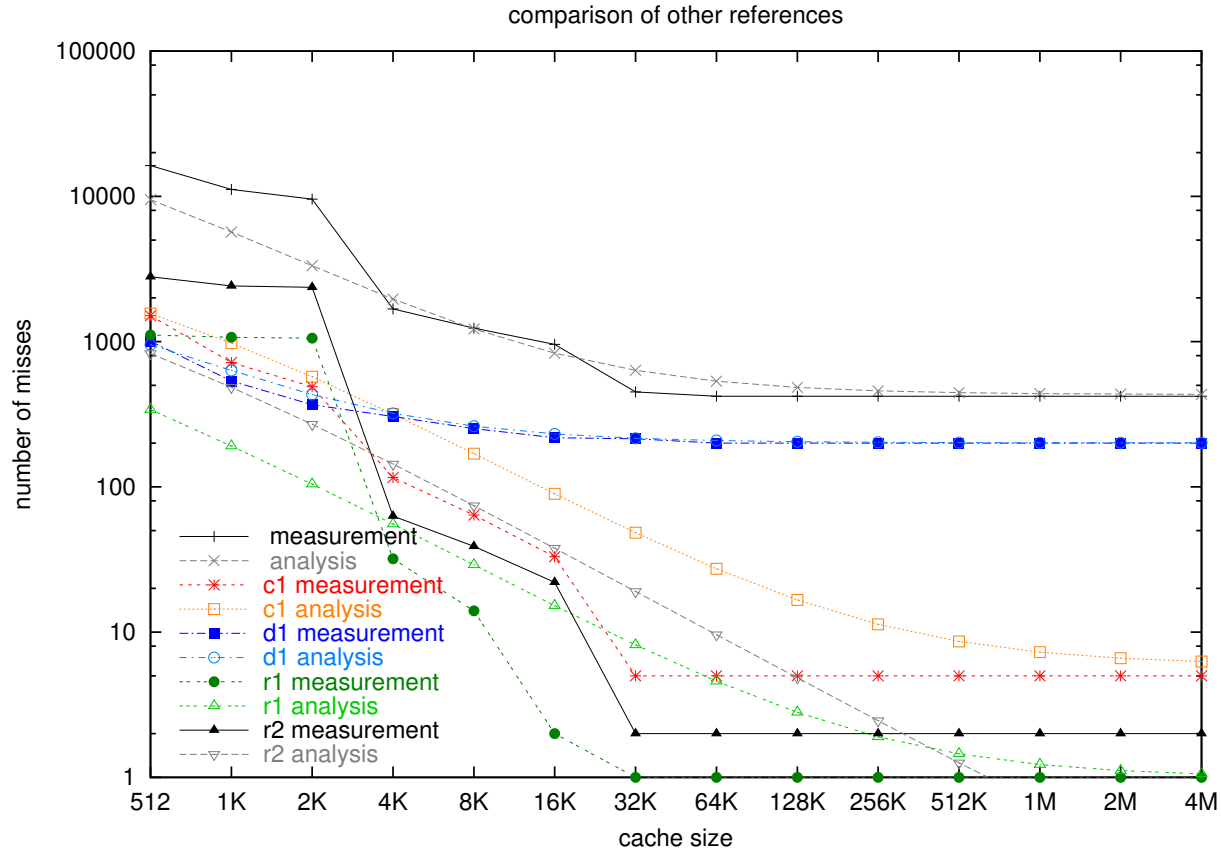
– Datengrößen:

Problemgröße	40	200	1000
<i>a</i>	640	16000	400000
<i>c</i>	320	8000	200000
<i>r</i>	164	804	4004
Matrix <i>A</i>	1124	24804	604004
Matrix <i>B</i> bzw. <i>D</i>	12800	320000	8000000
Spalte von <i>B</i>	320	1600	8000
alle Daten	26752	664832	16604032

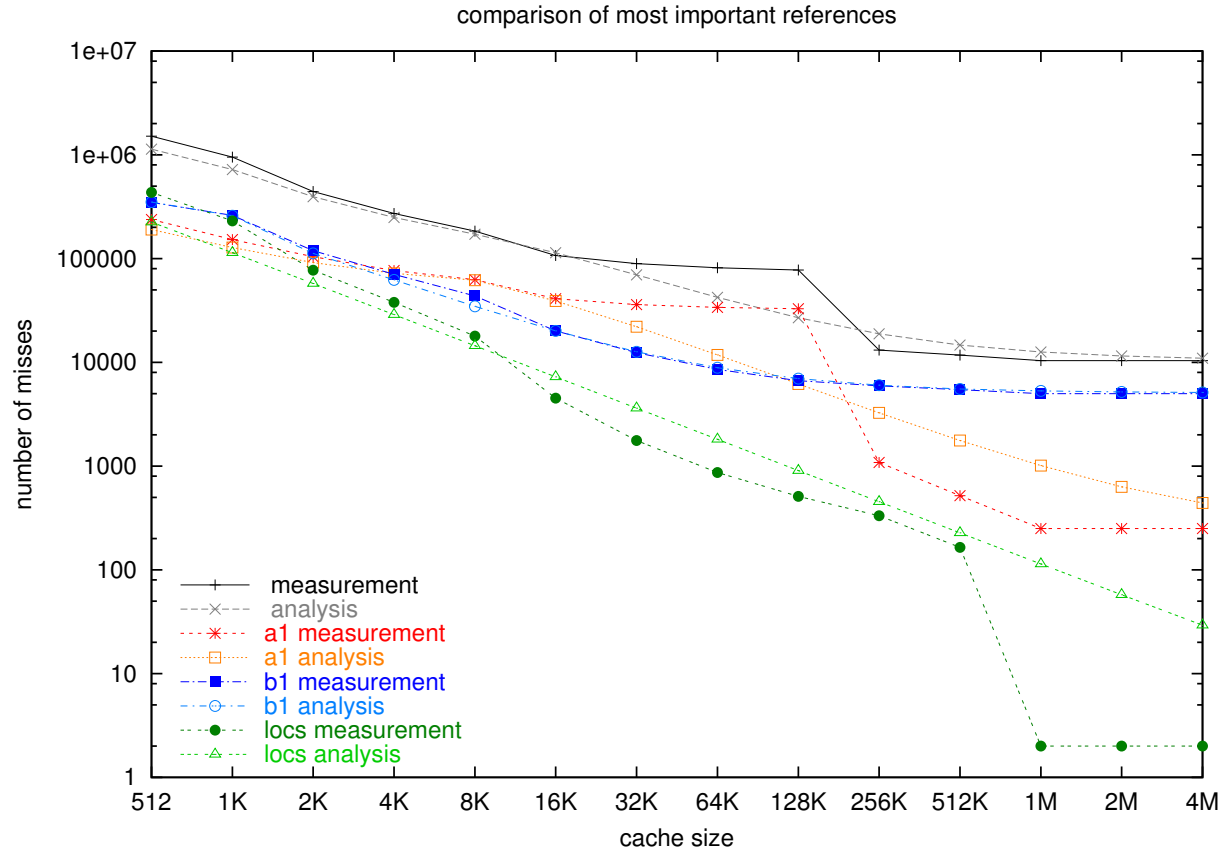
### 8.4.5. Einzelvergleich für Matrixgröße 40 - Teil 1



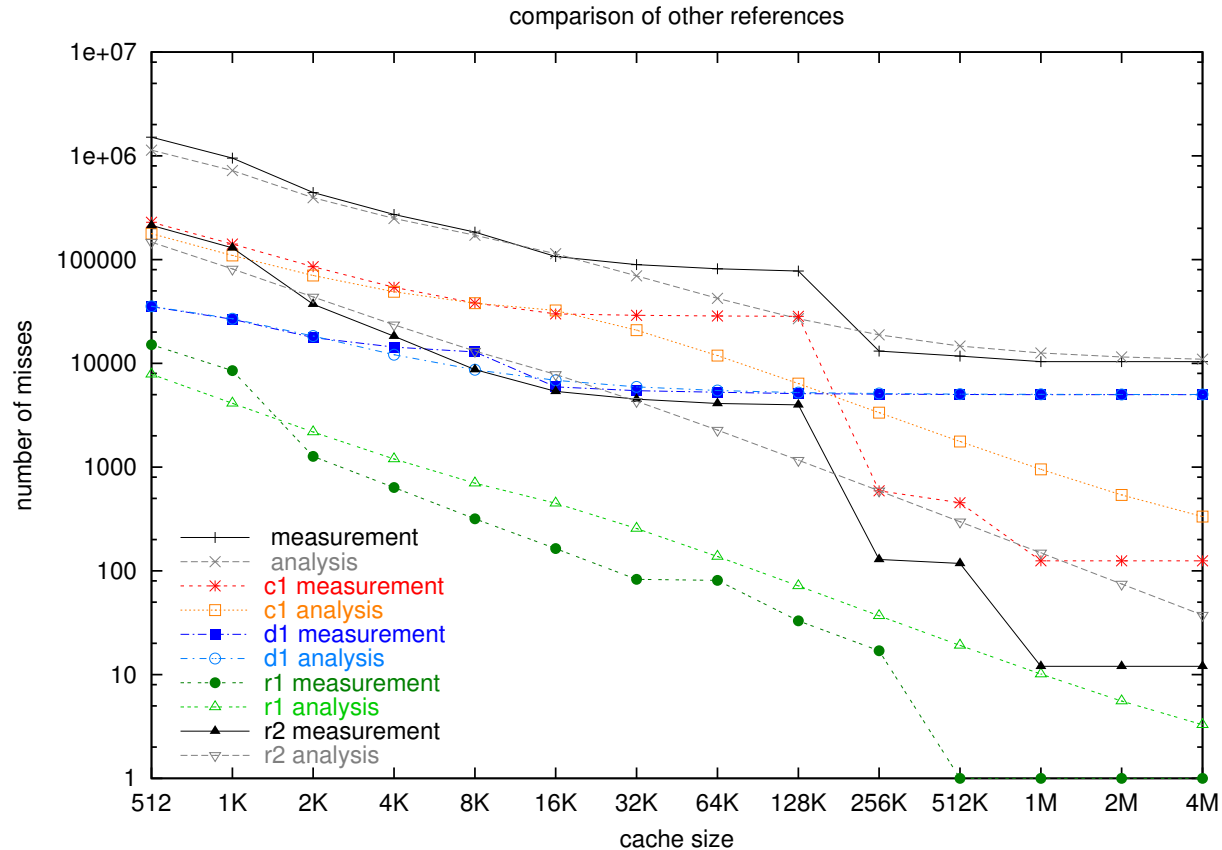
## 8.4.6. Einzelvergleich für Matrixgröße 40 - Teil 2



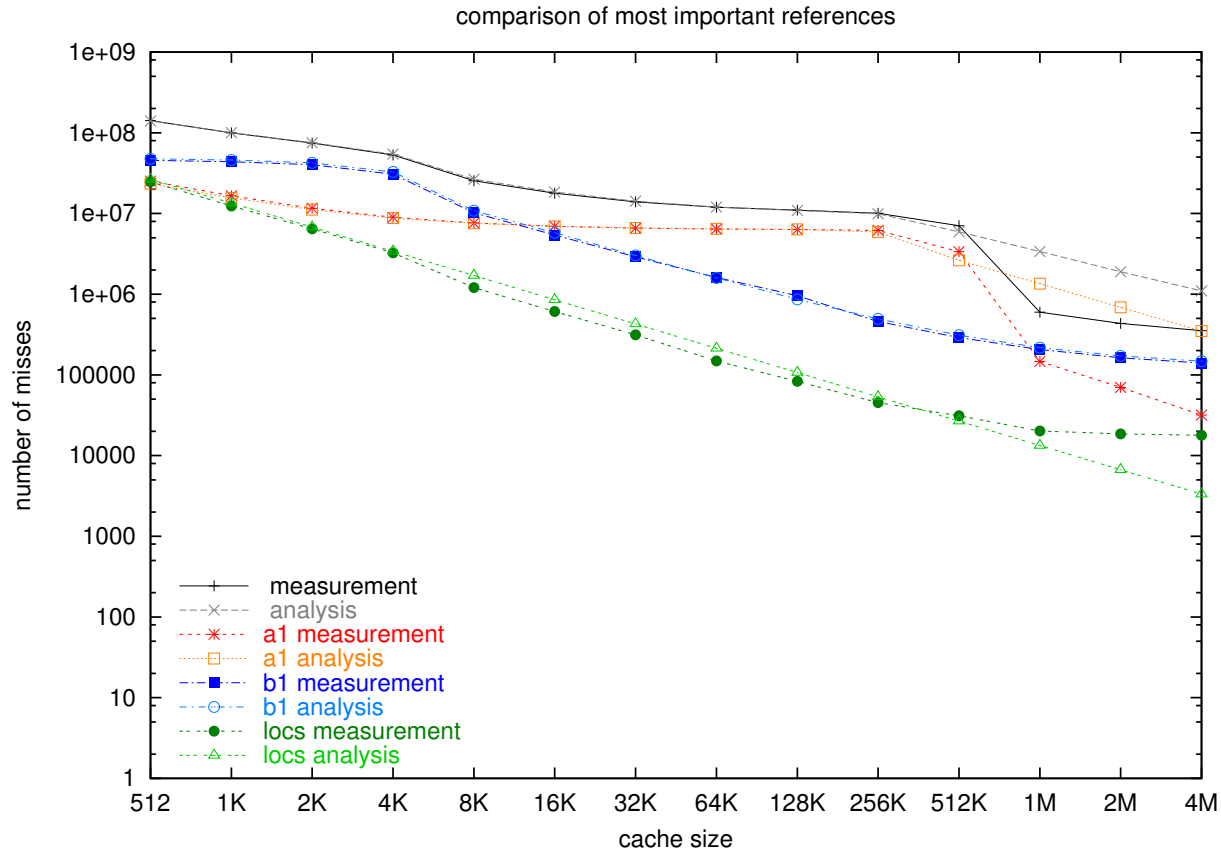
### 8.4.7. Einzelvergleich für Matrixgröße 200 - Teil 1



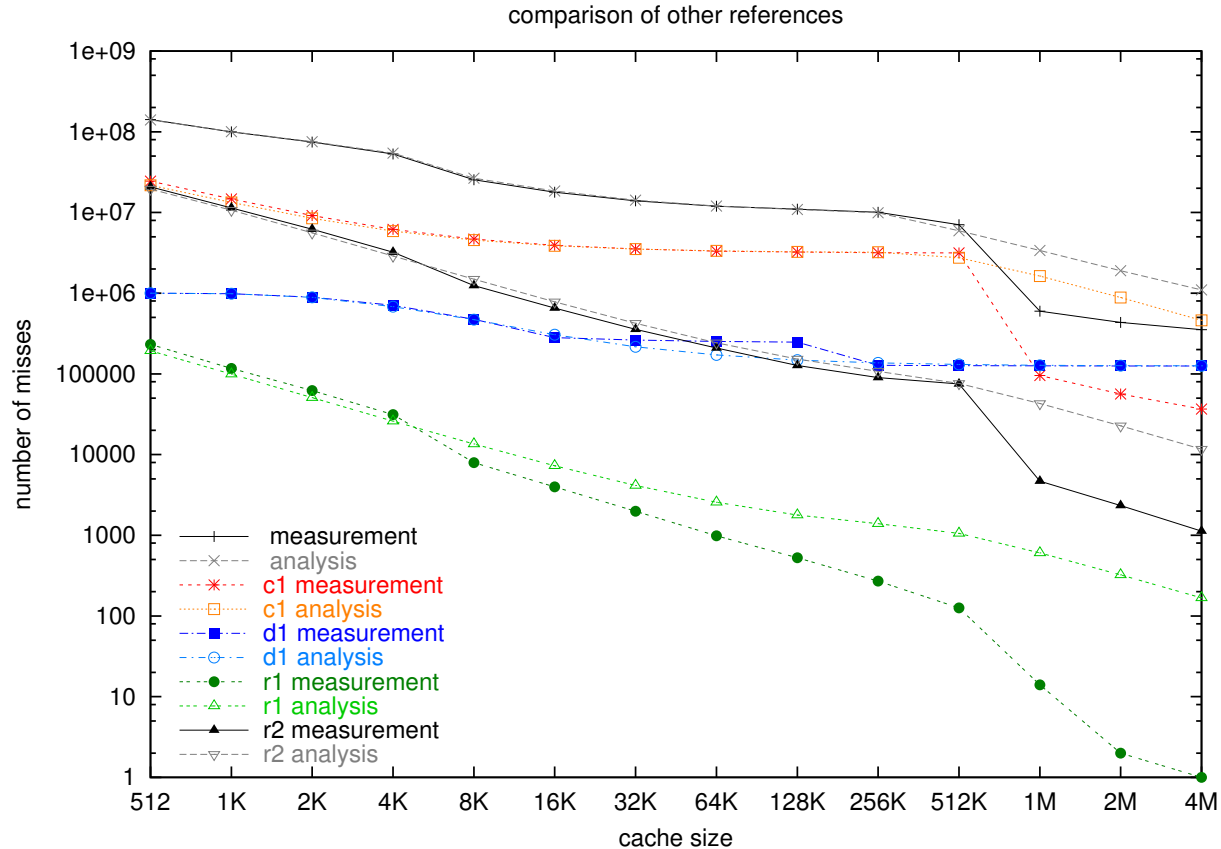
## 8.4.8. Einzelvergleich für Matrixgröße 200 - Teil 2



## 8.4.9. Einzelvergleich für Matrixgröße 1000 - Teil 1



## 8.4.10. Einzelvergleich für Matrixgröße 1000 - Teil 2



## 8.4.11. Auswertung Einzelvergleich

- auch einzeln sehr gute Abschätzungen
- bessere Ergebnisse bei größeren Problemen
- Fehlzugriffe auf lokale Variablen bei kleinen Problemen unterschätzt
- qpt2-Abweichungen dort noch größer (temporäre Variablen)

## 9. Choleskyzerlegung

- 9.1 Problemstellung
- 9.2 Right Looking Strategie
- 9.3 Dünn besetzte Choleskyzerlegung
- 9.4 Programm
- 9.5 Tests
  - 9.5.2 Gesamtzahlen
  - 9.5.6 Einzelvergleiche

## 9.1. Problemstellung

- Choleskyzerlegung dünn besetzter, positiv-definiten, symmetrischer Matrizen  $A$ :

$$A = L \cdot L^T$$

- $\Rightarrow a_{ij} = \sum_{k=0}^{n-1} l_{ik} \cdot l_{kj}^T = \sum_{k=0}^{n-1} l_{ik} \cdot l_{jk}$

- $\Rightarrow$  spaltenweise:  $A_j = \sum_{k=0}^{n-1} l_{jk} \cdot L_k = \left( \sum_{k=0}^{j-1} l_{jk} \cdot L_k \right) + l_{jj} \cdot L_j$

- $\Rightarrow L_j = \frac{1}{l_{jj}} \left( A_j - \sum_{k=0}^{j-1} l_{jk} \cdot L_k \right)$  und:  $l_{jj} = \frac{1}{l_{jj}} \left( a_{jj} - \sum_{k=0}^{j-1} l_{jk} \cdot l_{jk} \right) = \sqrt{a_{jj} - \sum_{k=0}^{j-1} l_{jk}^2}$

- Sei  $L'_j := A_j - \sum_{k=0}^{j-1} l_{jk} \cdot L_k$  (Modifikationen)

- $\Rightarrow L_j = \frac{1}{l_{jj}} L'_j$  und  $l_{jj} = \sqrt{l'_{jj}}$  (Vervollständigung)

## 9.2. Right-looking Strategie

```

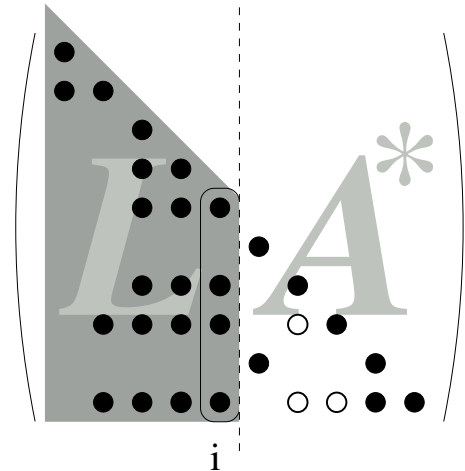
modify(j, k) {
    for(i=j; i<n; i++) {
        aij = ljk*lik;
    }
}
    
```

```

complete(j) {
    ljj = sqrt(ajj);
    for(i=j+1; i<n; i++) {
        lij = aij/ljj;
    }
}
    
```

```

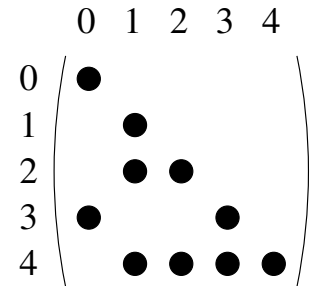
cholesky() {
    for(i=0; i<n; i++) {
        complete(i);
        for(j=i+1; j<n, j++) {
            modify(j, i);
        }
    }
}
    
```



### 9.3. Dünn besetzte Choleskyzerlegung

- `modify(j, k)` nutzlos wenn  $l_{jk} = 0$  ( $\Rightarrow$  genau eine Modification pro Nonzero)
- Bei Modification und Vervollständigung nur Nonzeros von  $L$  berücksichtigen
- komprimierte Spaltenspeicherung:

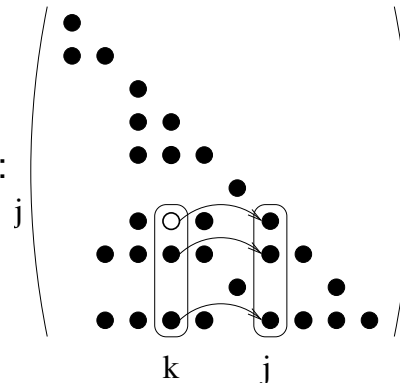
Index	0	1	2	3	4	5	6	7	8	9
nz	$l_{00}$	$l_{30}$	$l_{11}$	$l_{21}$	$l_{41}$	$l_{22}$	$l_{42}$	$l_{33}$	$l_{43}$	$l_{44}$
row	$0^*$	3	$1^*$	$2^*$	4	$3^*$	$4^*$			
firstnz	0	2	5	7	9	10				
startrow	0	2	3	5	6					



- Eingabeparameter für Analyse:

$$n, \alpha = \frac{nz-n}{0,5 \cdot n \cdot (n-1)}, \beta = \frac{rws}{nz}$$

- Beispiel für dünn besetzte Modification:



## 9.4. Programm

```

n1 = L1->n1;
i1 = 0;
while(i2 < n2) {
    /* Spalte i vervollstaendigen: */
    p_nz_s1 = L2->nz1 + L3->firstnz1[i3]1;
    p_nz_e1 = L4->nz2 + L5->firstnz2[i4 + 1]2;
    f1 = 1.0 / sqrt(*1p_nz_s2);
    p_nz_i1 = p_nz_s3;
    while(p_nz_i2 < p_nz_e2) {
        *3p_nz_i4 = *2p_nz_i3 * f2;
        p_nz_i6 = p_nz_i5 + 1;
    }
    /* mit Spalte i modifizieren: */
    p_nz_s5 = p_nz_s4 + 1;
    p_rw_s1 = L6->row1 + L7->startrow1[i5]1 + 1;
    while(p_nz_s6 < p_nz_e3) {
        /* Spalte *p_rw_s mit Spalte i modifizieren: */
        p_nz_i7 = p_nz_s7;
        p_rw_i1 = p_rw_s2;
        j1 = *1p_rw_s3;
        p_nz_j1 = L8->nz3 + L9->firstnz3[j2]3;
        p_rw_j1 = L10->row2 + L11->startrow2[j3]2;
        f3 = *4p_nz_s8;

```

```

        while(p_nz_i8 < p_nz_e4) {
            /* Nonzero in Spalte j finden: */
            t1 = *2p_rw_i2;
            while(*3p_rw_j2 < t2) {
                p_rw_j4 = p_rw_j3 + 1;
                p_nz_j3 = p_nz_j2 + 1;
            }

            *7p_nz_j5 = *5p_nz_j4 - f4 * *6p_nz_i9;

            p_nz_i11 = p_nz_i10 + 1;
            p_rw_i4 = p_rw_i3 + 1;
            p_nz_j7 = p_nz_j6 + 1;
            p_rw_j6 = p_rw_j5 + 1;
        }
        p_nz_s10 = p_nz_s9 + 1;
        p_rw_s5 = p_rw_s4 + 1;
    }
    i7 = i6 + 1;
}

```

## 9.5. Tests

- Messung der Fehlzugriffe zum Vergleich mit Analyse:

Werkzeuge	Datentrace	Fehlzugriffe
WARTS	qpt2	dinerolV
Eigene	instrumentierter Quelltext	refsim

Trace für dinerolV:

```
1 ffbeea88
2 11368
0 ffbeea88
2 1136c
```

Trace für refsim:

```
0 ffbeea9c 4 23
0 ffbeea8c 4 44
1 ffbeea60 8 51
0 ffbeea9c 4 26
```

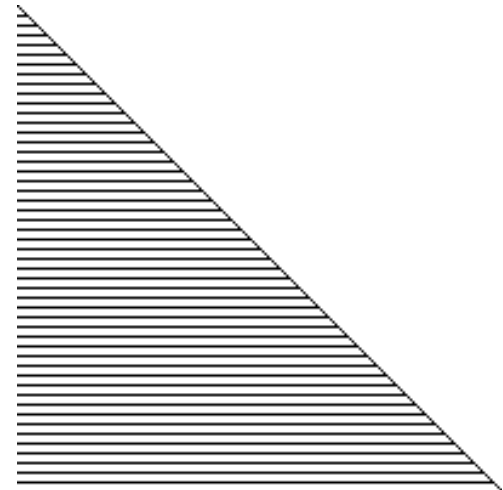
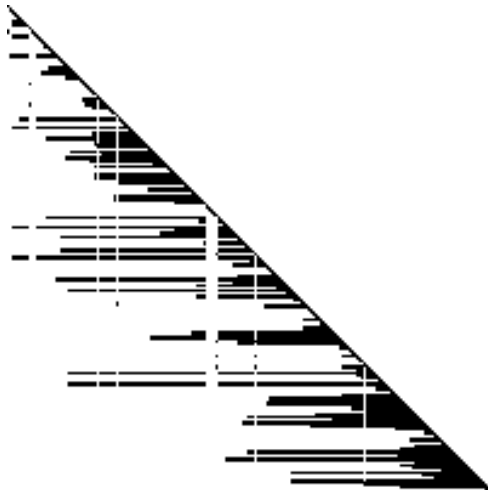
- Testumgebung:
  - SUN Ultra-60 mit 2 UltraSPARC II Prozessoren unter Solaris 8.0
  - GNU gcc 2.8.1 ohne Optimierung

## 9.5.1. Beispielprobleme

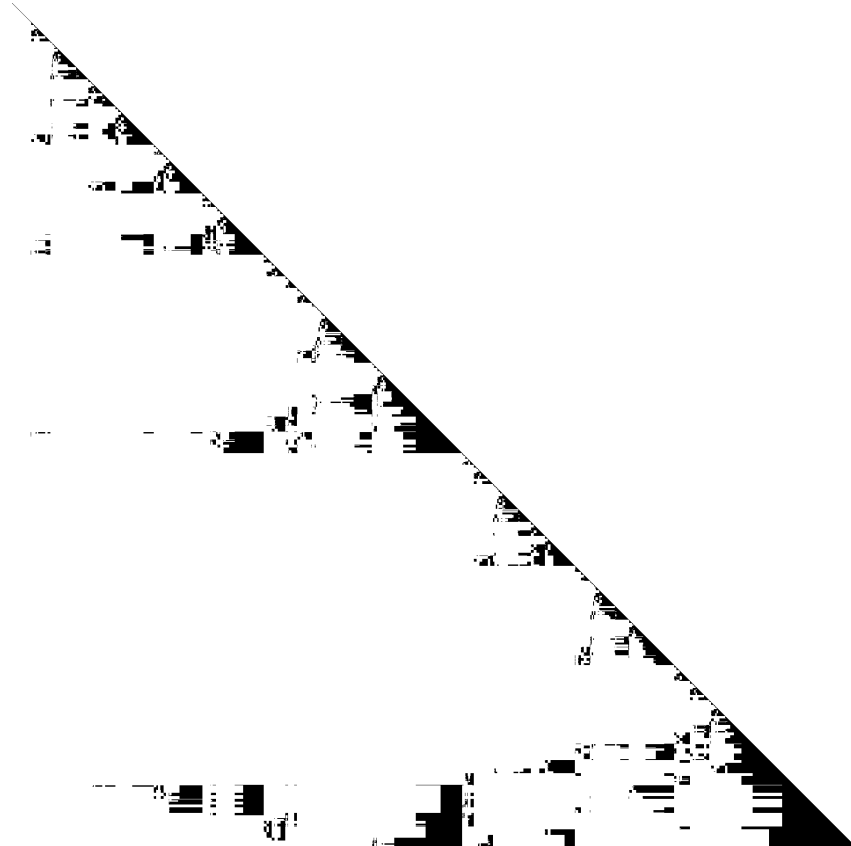
- Übersicht:

Name	Seitenlänge: $n$	Nonzeros in $A$ : $m$	Zeilenindizes	Belegungs-faktor: $\alpha$	Kompressions-faktor: $\beta$
zfall200	200	4605	2432	0,2214	0,5282
konst200	200	5100	3875	0,2462	0,7598
bcsstk14	1806	112267	17756	0,0678	0,1582

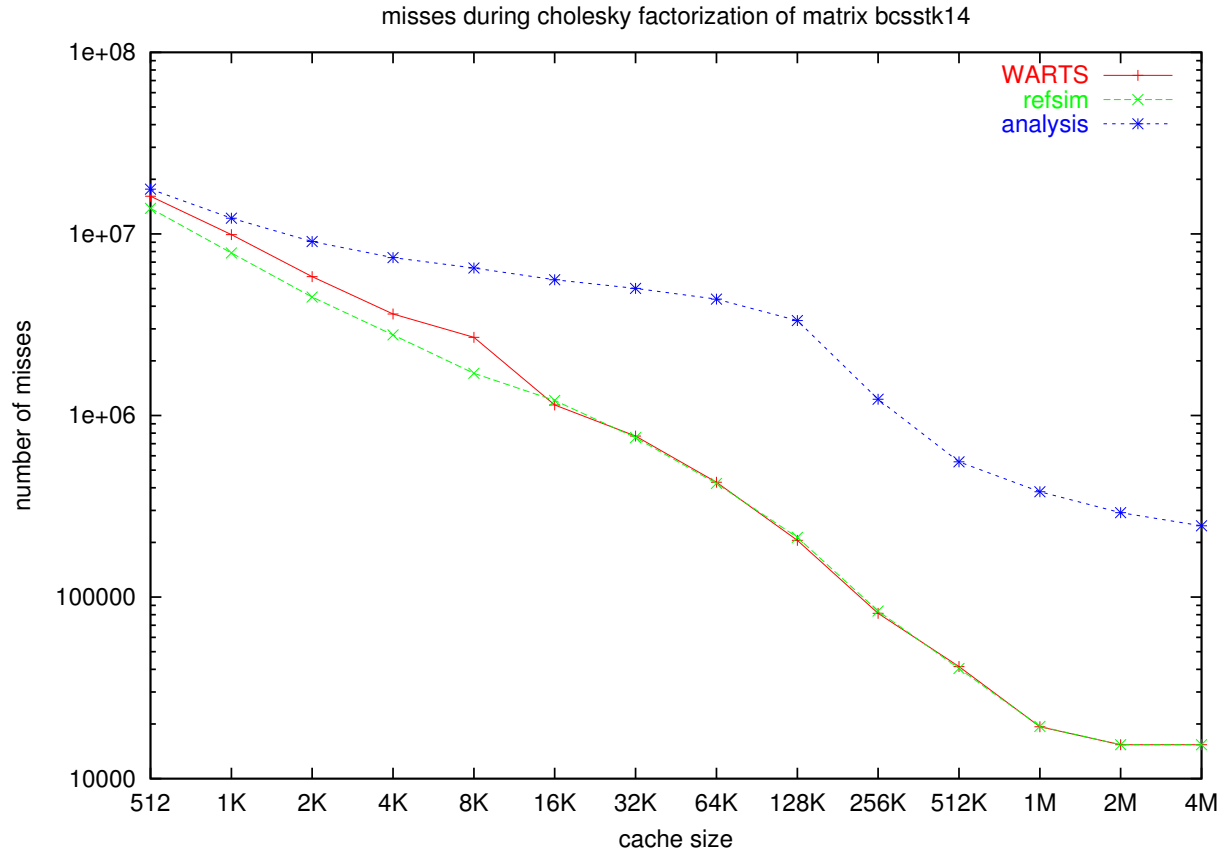
- Strukturen der Matrizen  $L$  für die Probleme zfall200 und konst200:



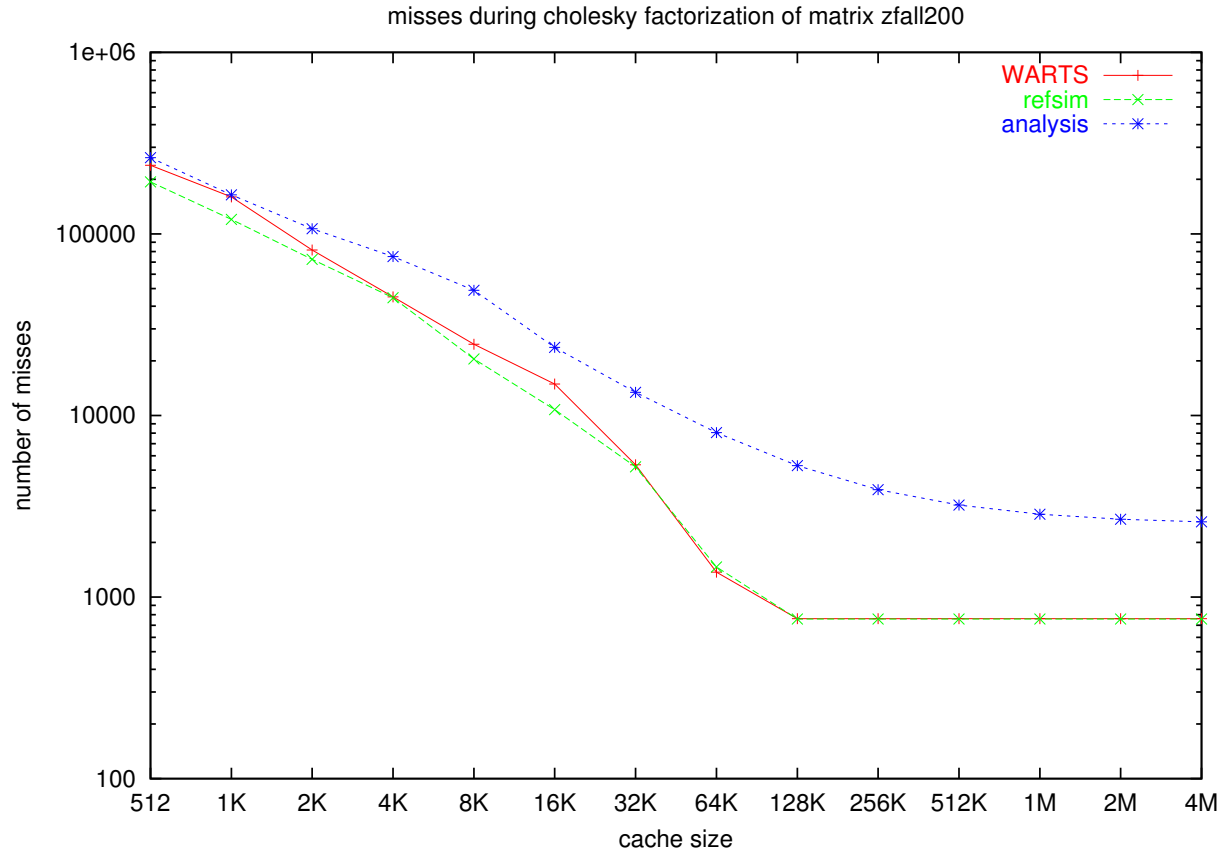
- Struktur der Matrix  $L$  für das Problem `bcsstk14`:



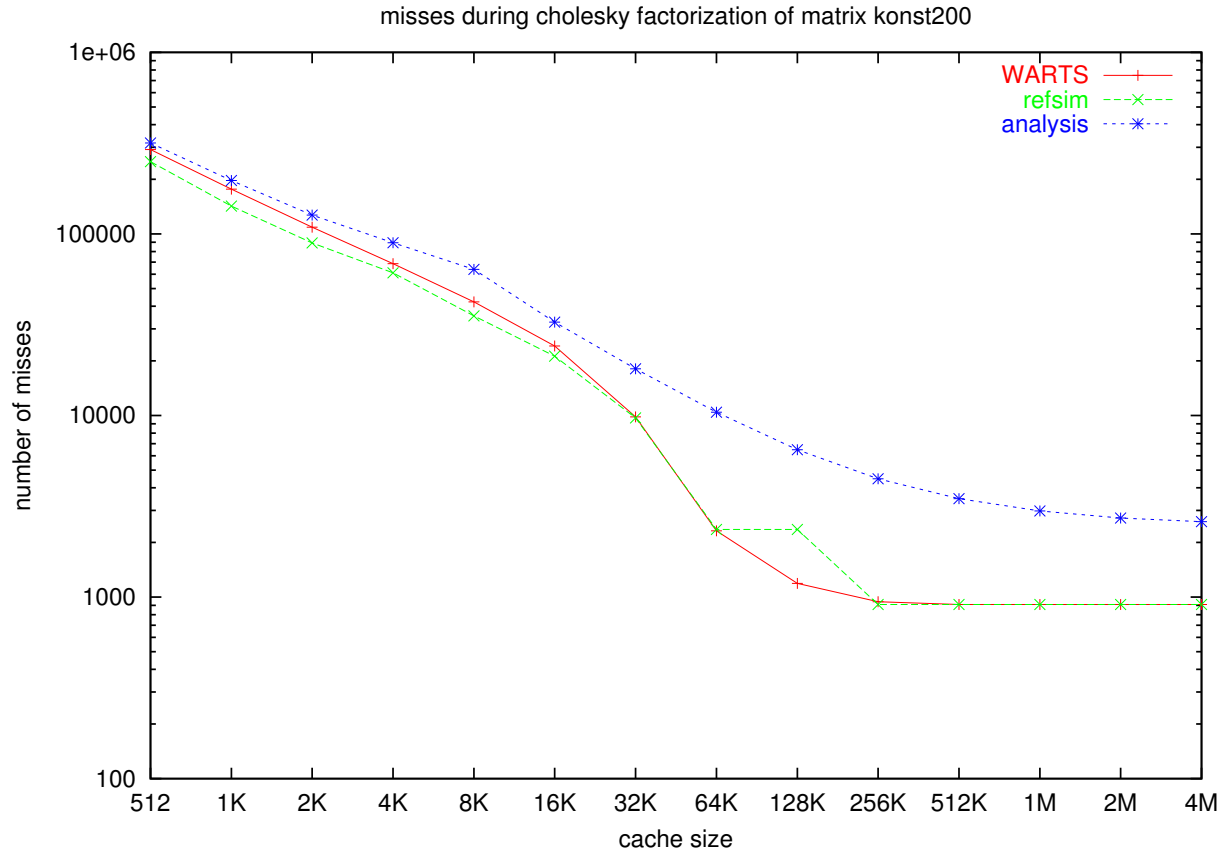
## 9.5.2. Gesamtzahlen für bcsstk14



### 9.5.3. Gesamtzahlen für zfall200



### 9.5.4. Gesamtzahlen für konst200



## 9.5.5. Auswertung Gesamtzahlen

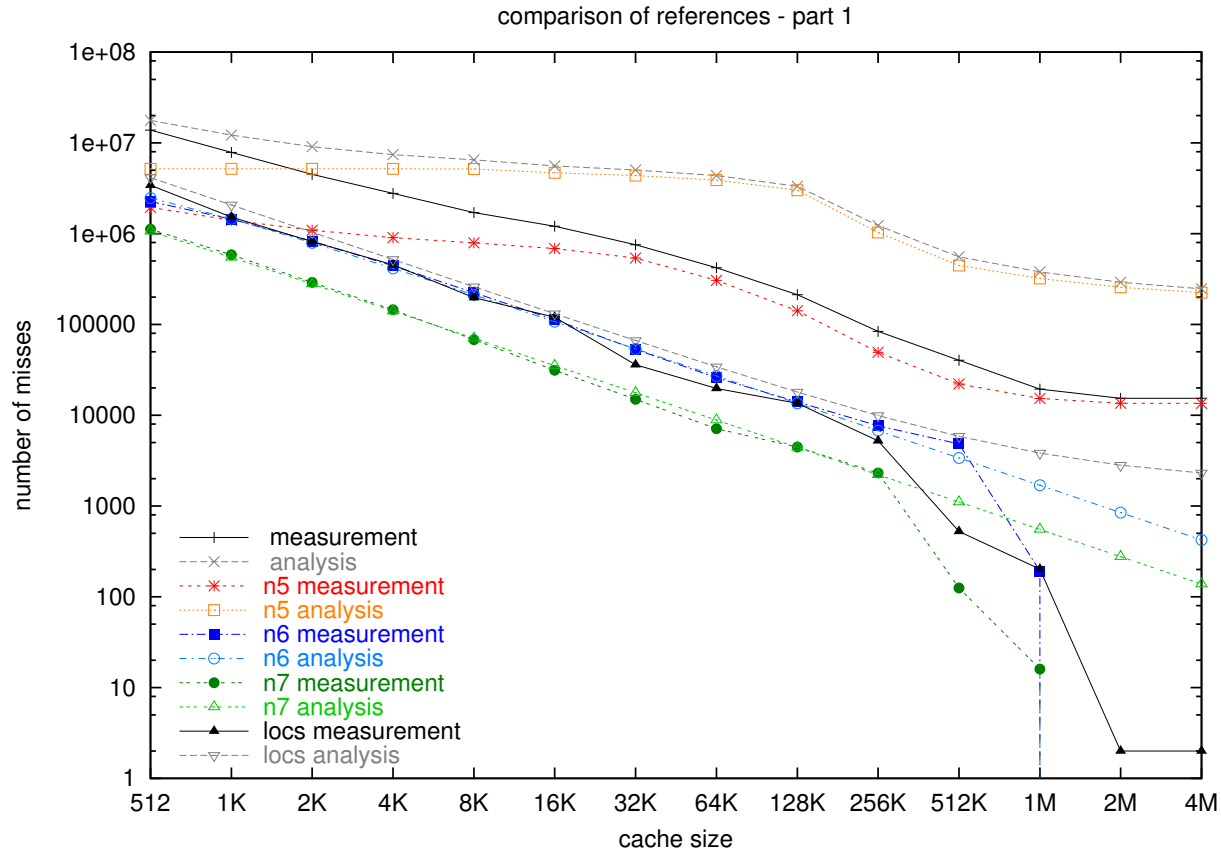
- sehr große Unterschiede zwischen Messung und Analyse
  - Unterschiede geringer bei kleineren Caches ( $\leftrightarrow$  große Interferenzen)
  - Unterschiede bei großer Matrix stärker ( $\leftrightarrow$  große Interferenzen)
- kleine  $\alpha$  mit großer Überschätzung (`bcsstk14`  $\leftrightarrow$  `zfall200`)  $\rightarrow$  9.5.1
  - Verteilung der Nonzeros im Allgemeinen nicht gleichmäßig (Fill-In)
  - Verteilung der Nonzeros gleichmäßig bei `konst200`
- Abweichungen bei großen Caches:
  - Grundannahme greift nicht, wenn z.B. alle Daten nebeneinander im Cache

– Datengrößen:

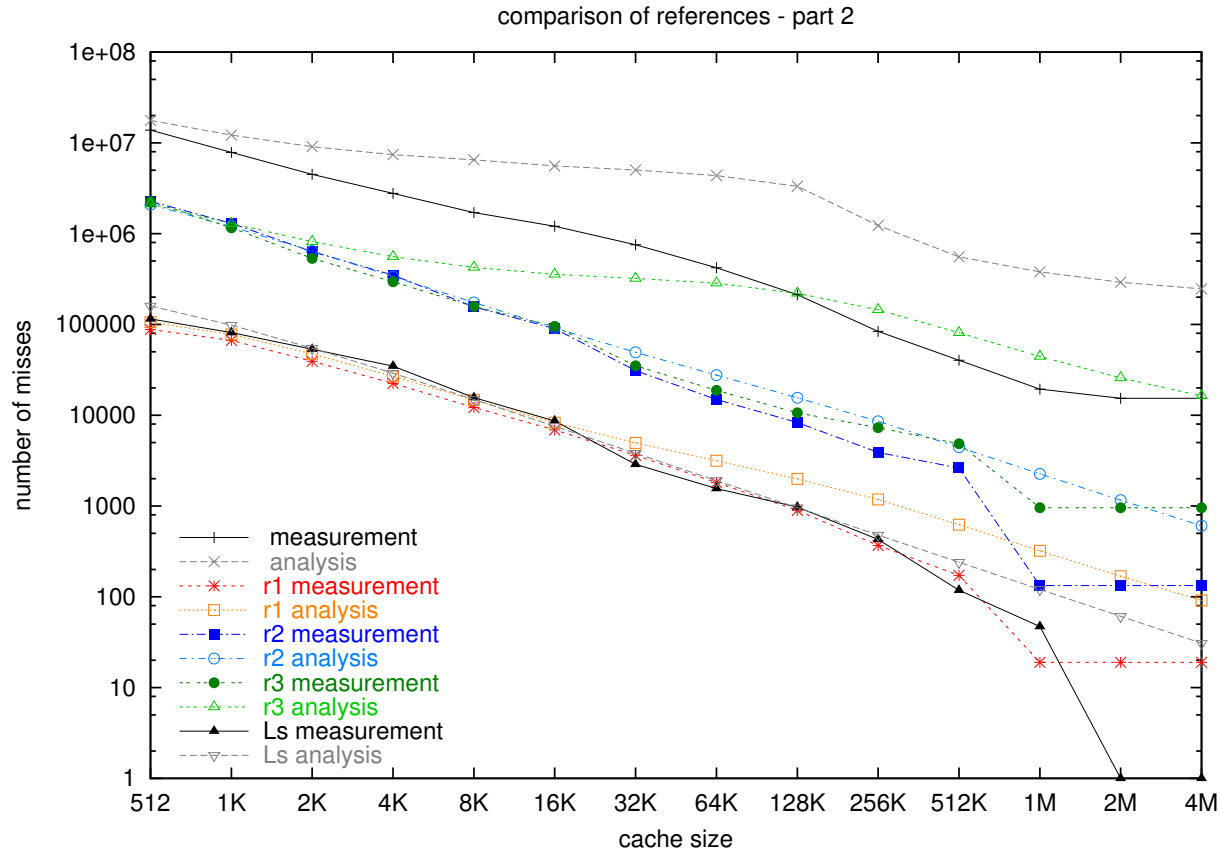
Name	Speicherbedarf	Speicherzeilen
<code>bcsstk14</code>	983692	15371
<code>zfall200</code>	48252	754
<code>konst200</code>	57984	906

- gemessener plötzlicher Abfall stimmt mit Speicherbedarf überein ( $\leftarrow$  konsequente Abspeicherung)

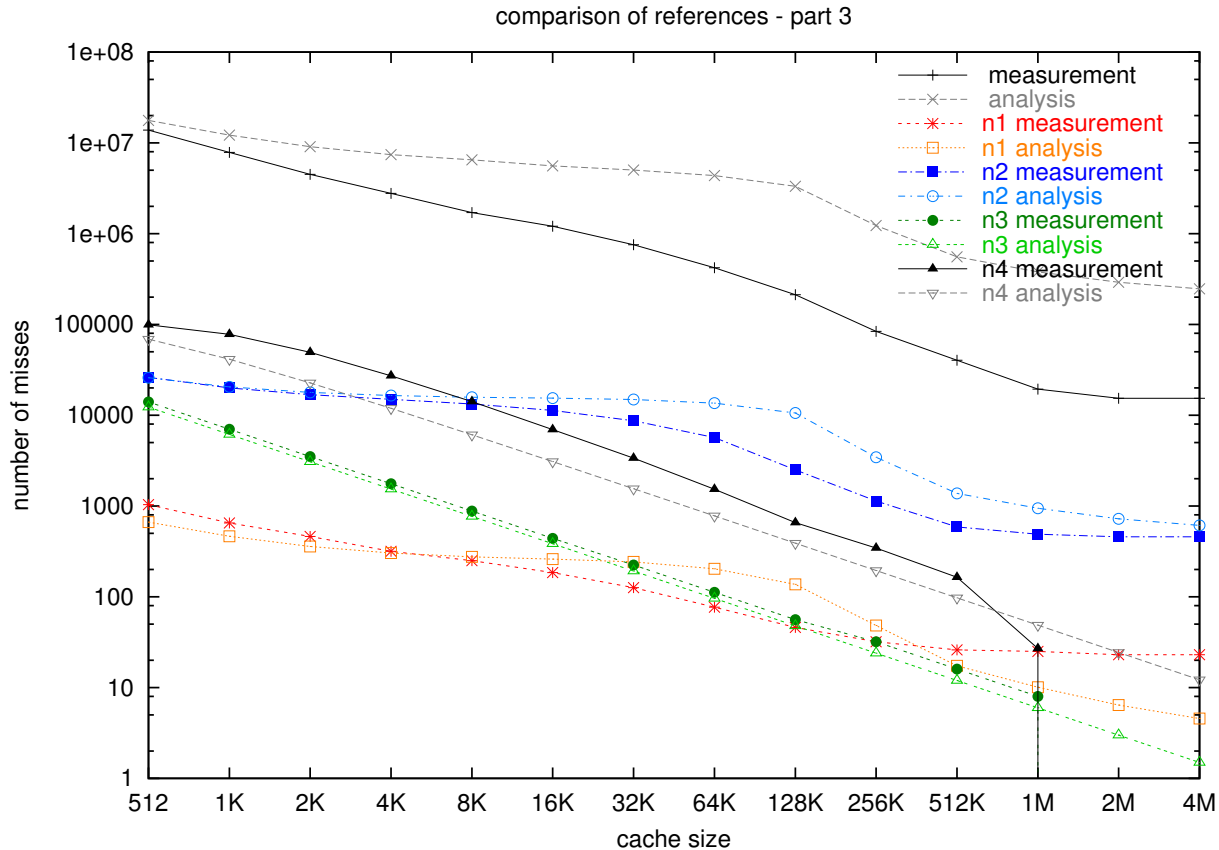
## 9.5.6. Einzelvergleich für bcsstk14 Teil 1



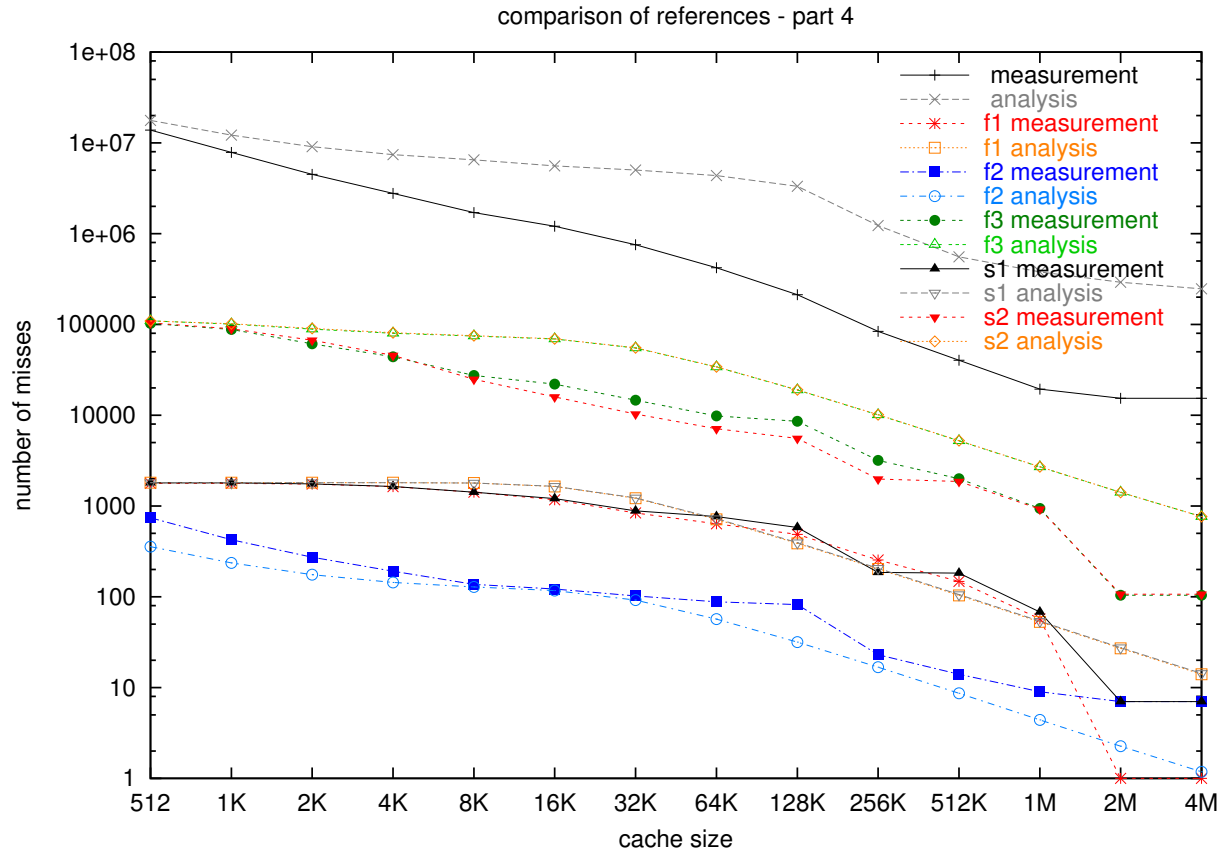
## 9.5.7. Einzelvergleich für bcsstk14 Teil 2



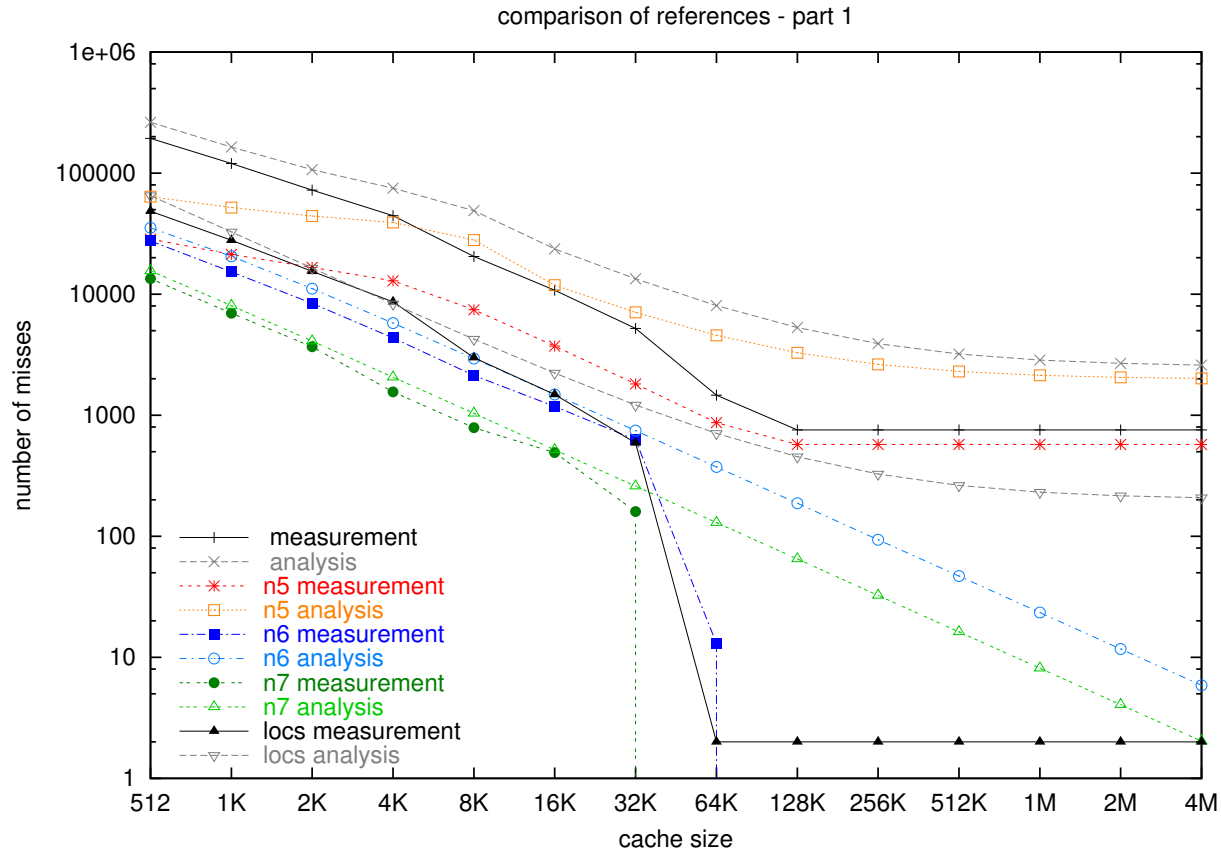
### 9.5.8. Einzelvergleich für bcsstk14 Teil 3



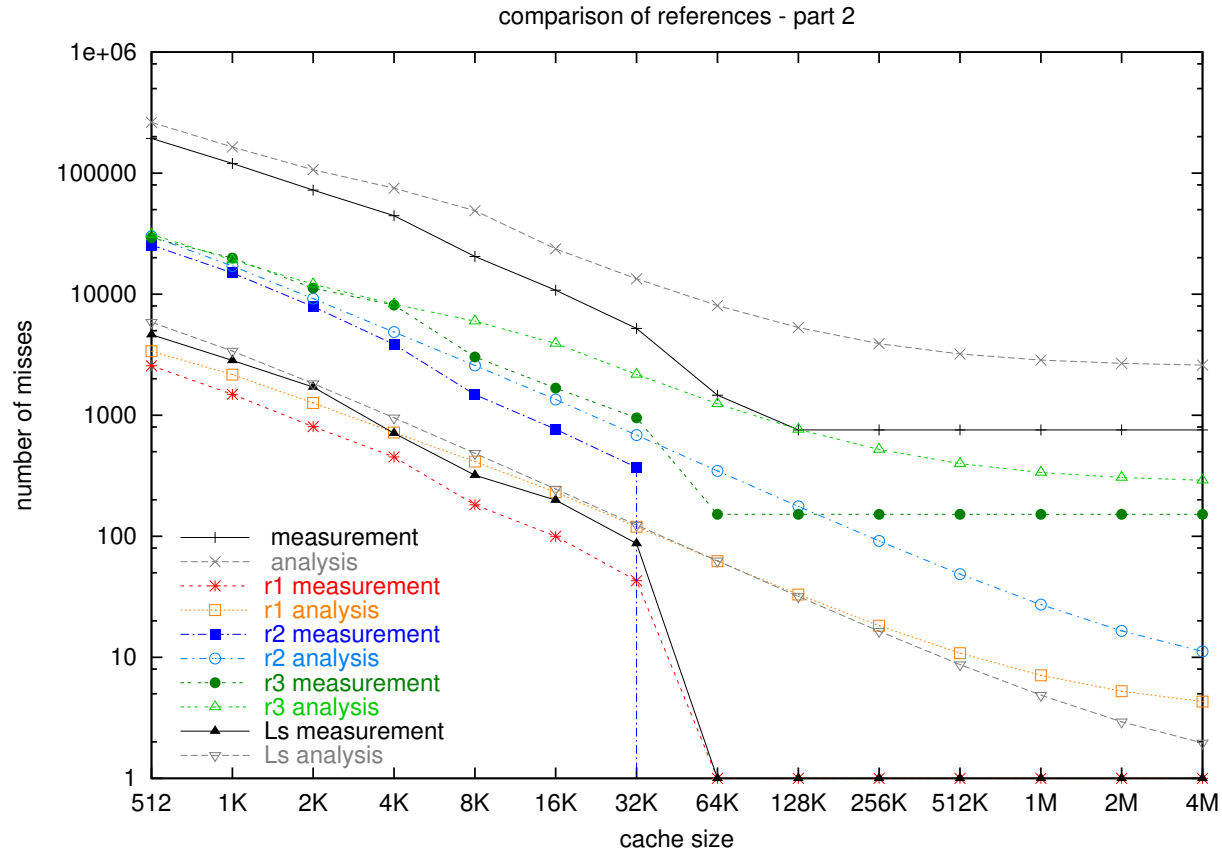
## 9.5.9. Einzelvergleich für bcsstk14 Teil 4



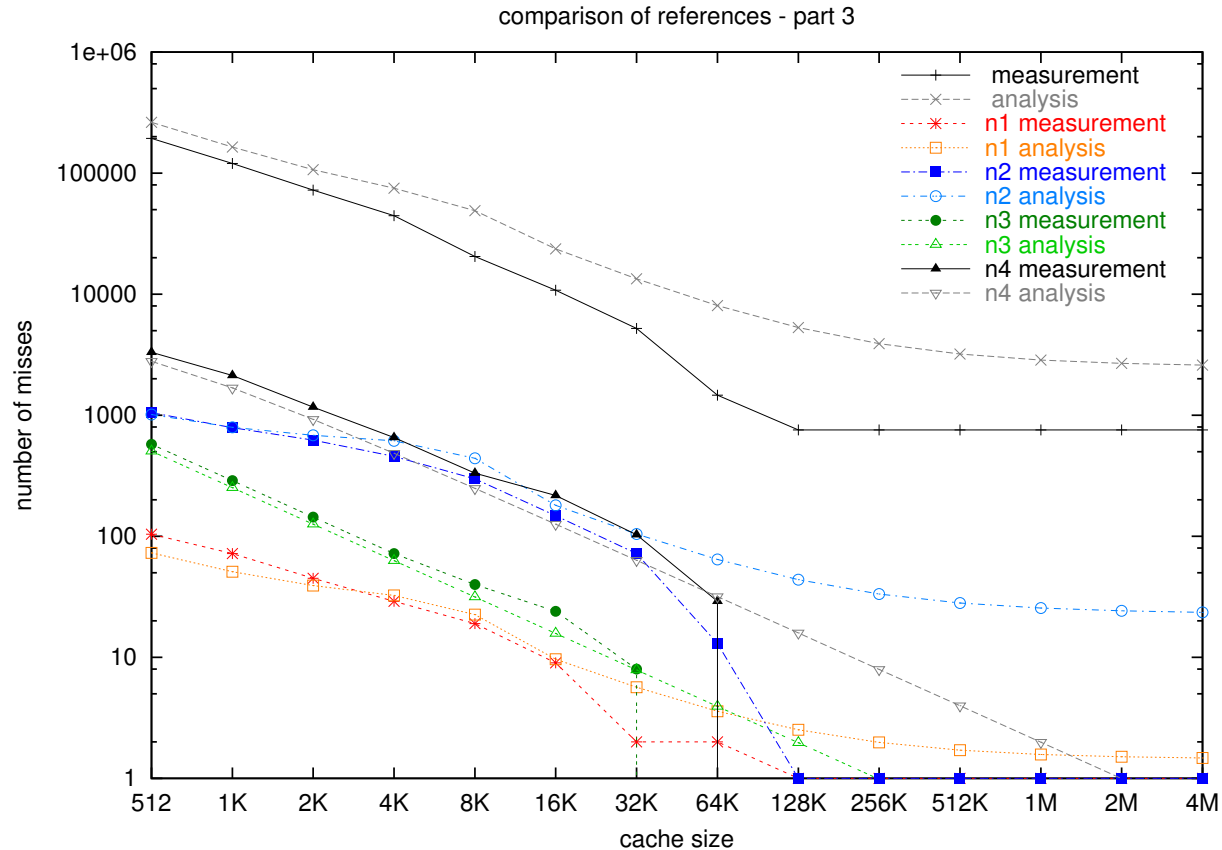
## 9.5.10. Einzelvergleich für zfall200 Teil 1



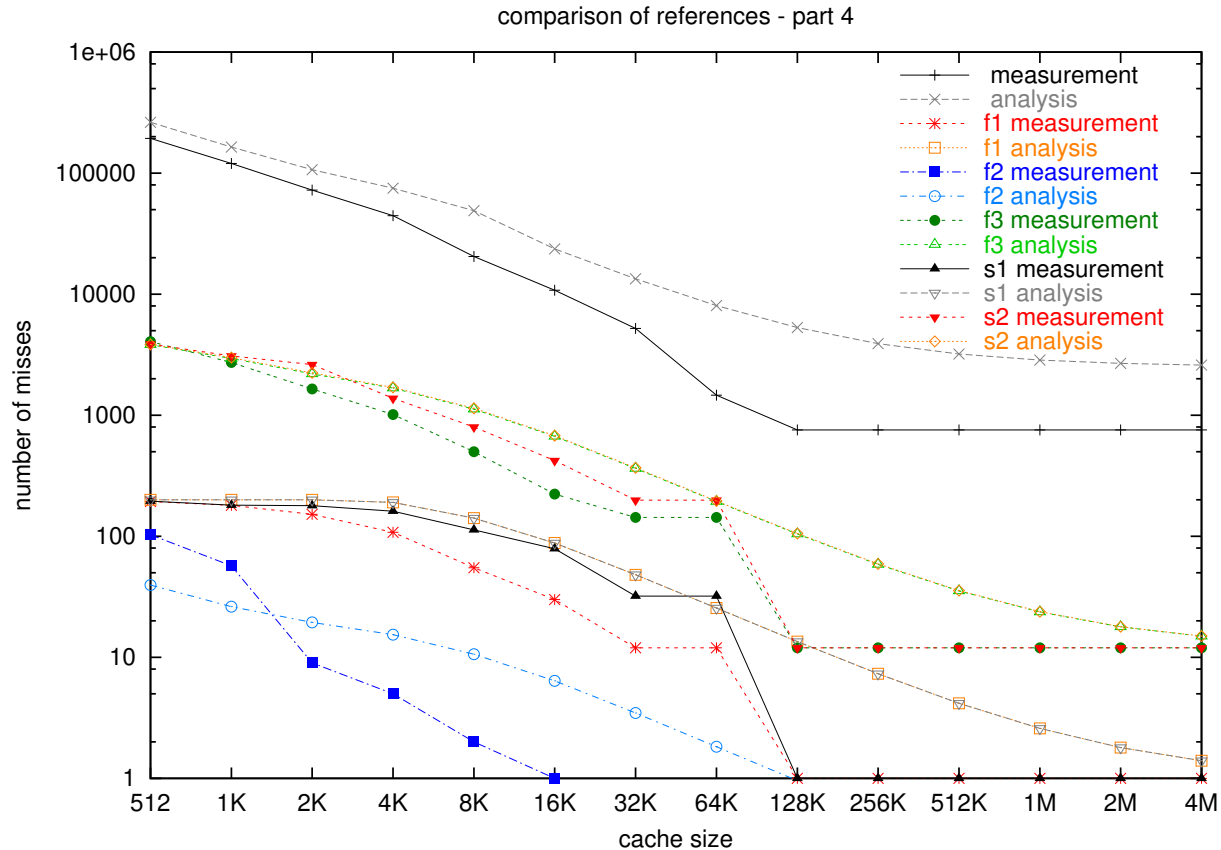
## 9.5.11. Einzelvergleich für zfall200 Teil 2



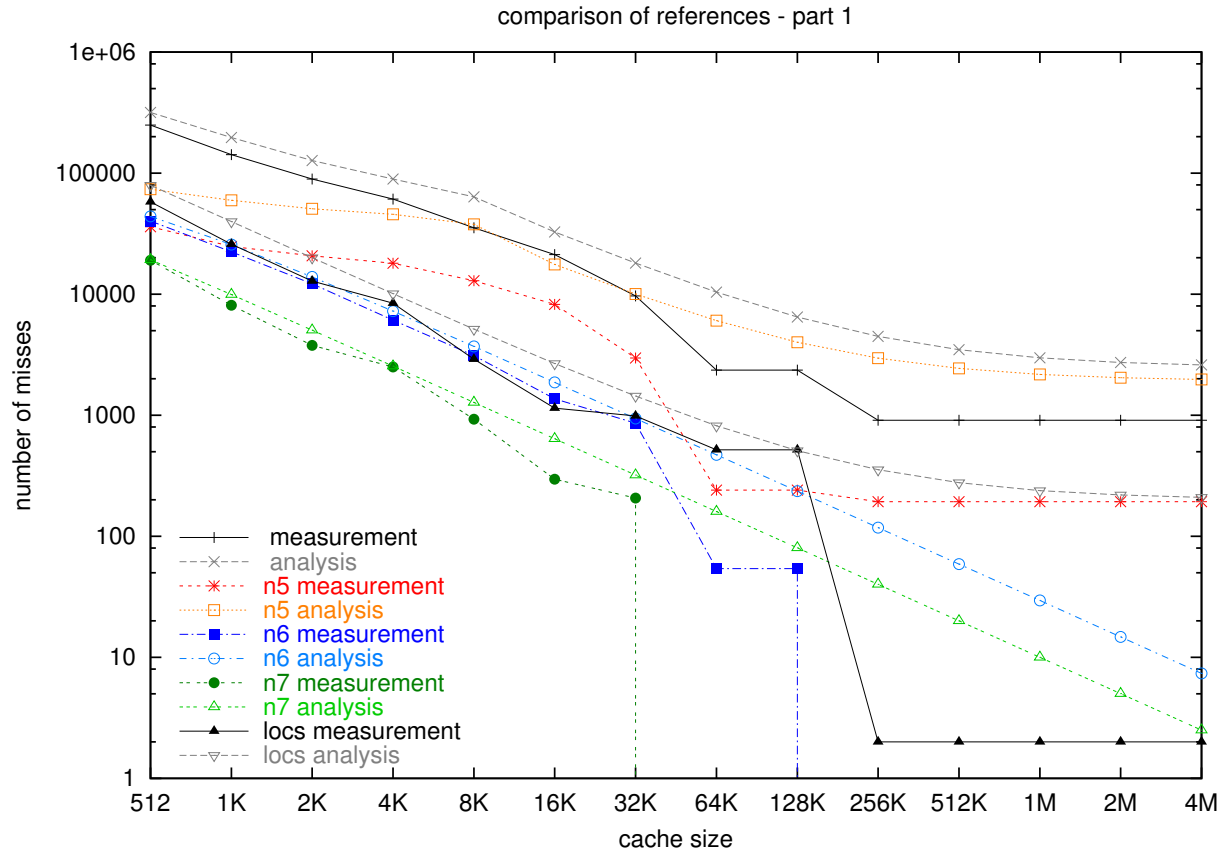
## 9.5.12. Einzelvergleich für zfall200 Teil 3



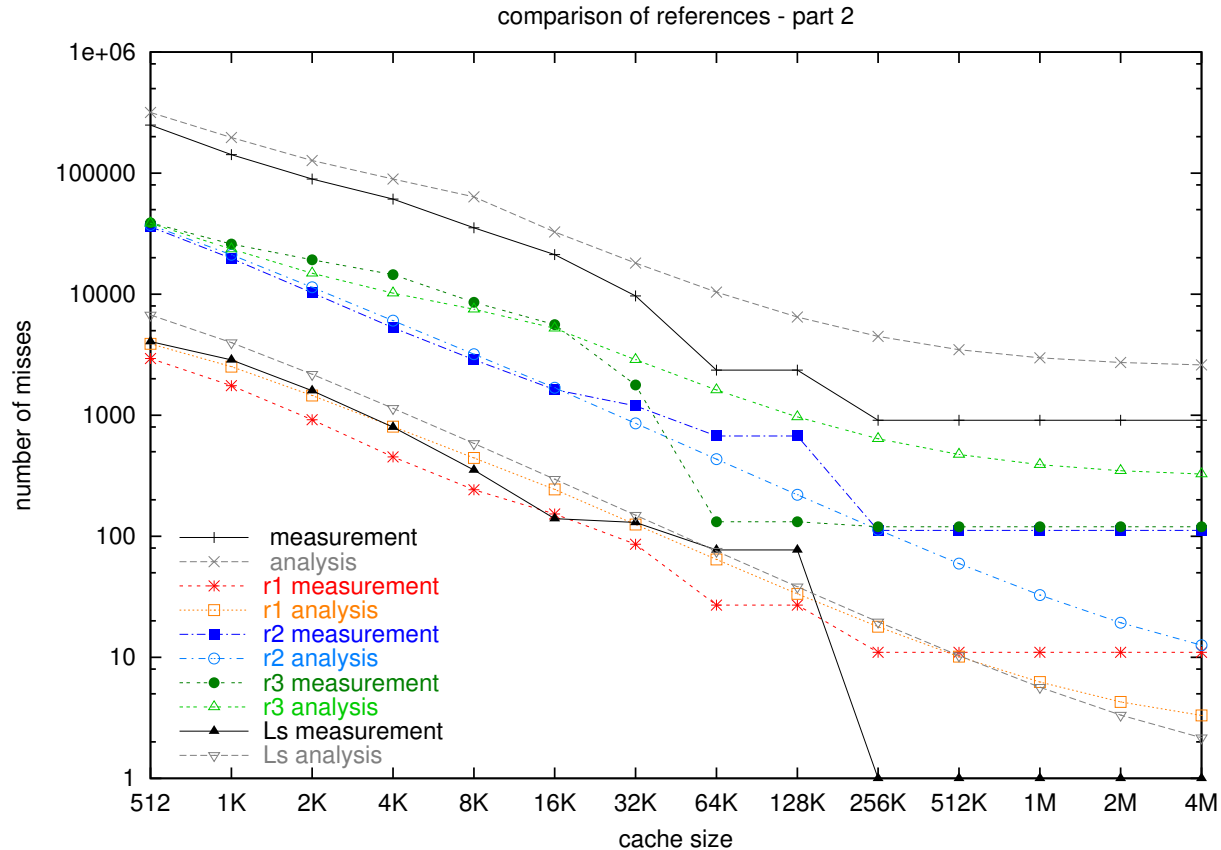
## 9.5.13. Einzelvergleich für zfall200 Teil 4



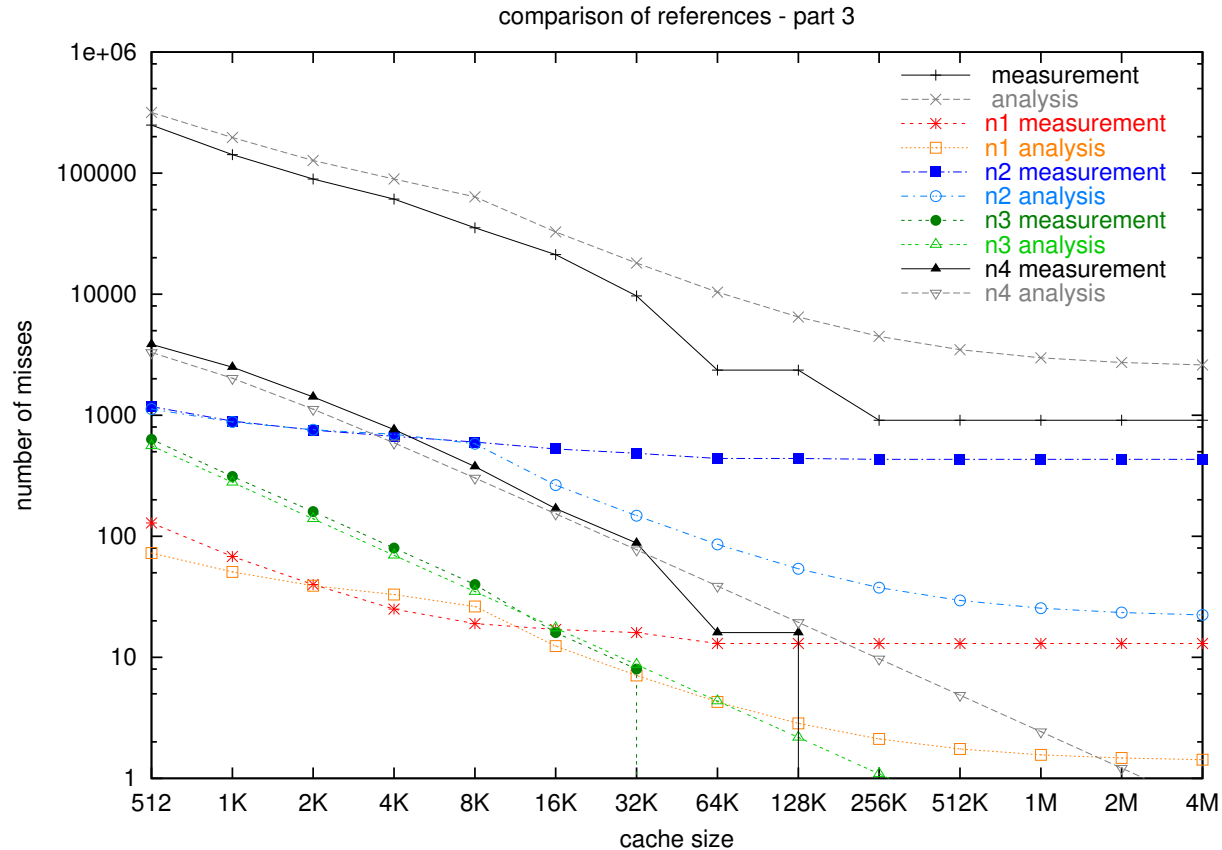
## 9.5.14. Einzelvergleich für konst200 Teil 1



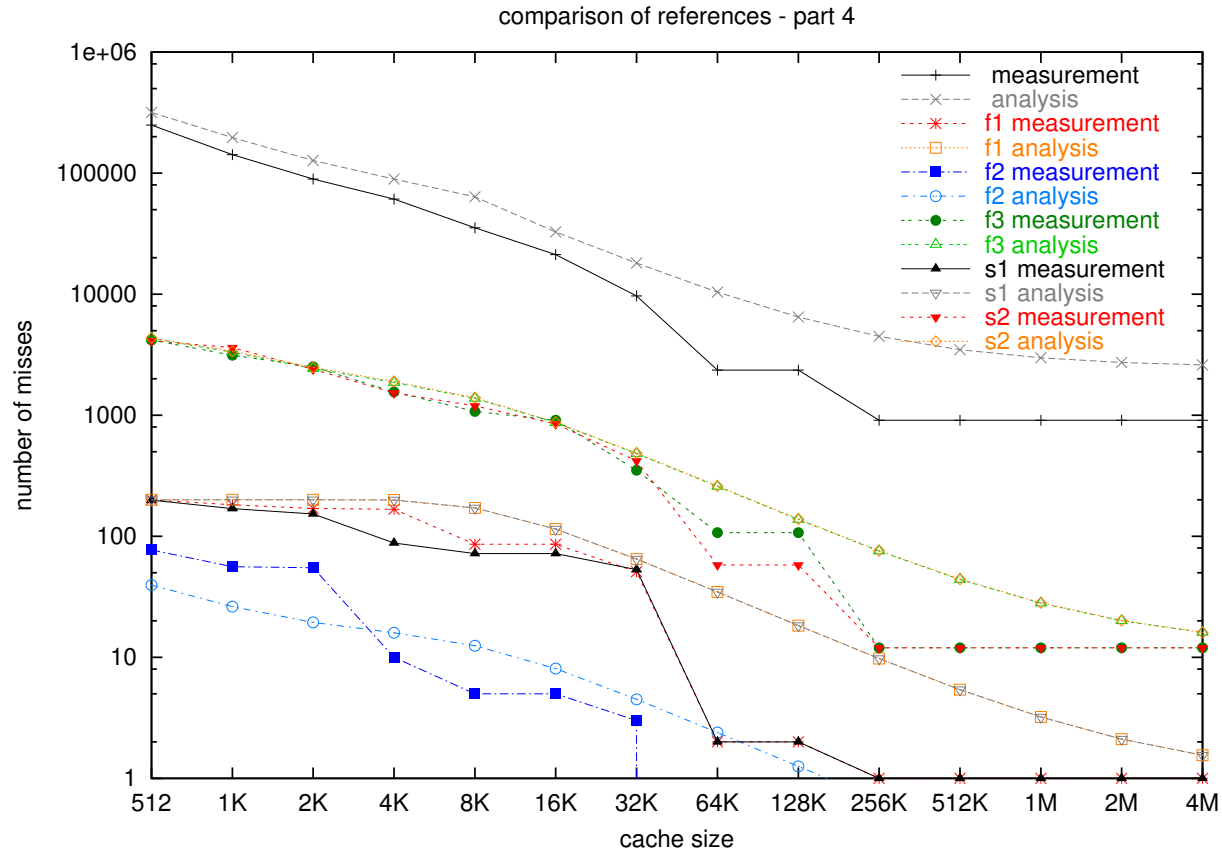
## 9.5.15. Einzelvergleich für konst200 Teil 2



## 9.5.16. Einzelvergleich für konst200 Teil 3



## 9.5.17. Einzelvergleich für konst200 Teil 4



## 9.5.18. Einzelvergleich

- geringe Unterschiede bei meisten Referenzen (auch bei lokalen Variablen)
- $n_5$  (und  $r_3$ ) mit größeren Abweichungen
  - $n_5$  begründet bereits schlechte Gesamtab schätzung
  - in innerer Schleife  $\Rightarrow$  großer Einfluß
  - Lokalität oft nur über mehrere Iterationen der äußersten Schleife möglich,  
 $\leftrightarrow$  ungleichmäßige Verteilung der Nonzeros

## 10. Zusammenfassung

- Methode entwickelt und angewendet
  - in konstanter Zeit auswertbare Programme
  - Prinzipbedingte Schwäche der Grundannahme bei relativ großen Caches
- Matrixmultiplication: sehr gute Abschätzungen
- Choleskyzerlegung: gute Abschätzungen mit zwei problematischen Referenzen
- Automatisierung:
  - Referenzen automatisch aus Objektcode  $\leftrightarrow$  Verwendung
  - Wiederverwendungstypen systematisch bestimmen
  - Klassifizierungsheuristiken (kurz zuerst, erste Iterationen)
  - Interferenzmuster

## 11. Ausblick

- Choleskyzerlegung genauer untersuchen:
  - große Interferenzen
  - Nonzero Verteilung
- Automatisierung:
  - Teilaspekte automatisch unterstützen
  - Integration in Compiler
- Erweiterung auf andere Cachearchitekturen:
  - assoziativ
  - mehrstufig
  - Instruktions- / Trace-Cache