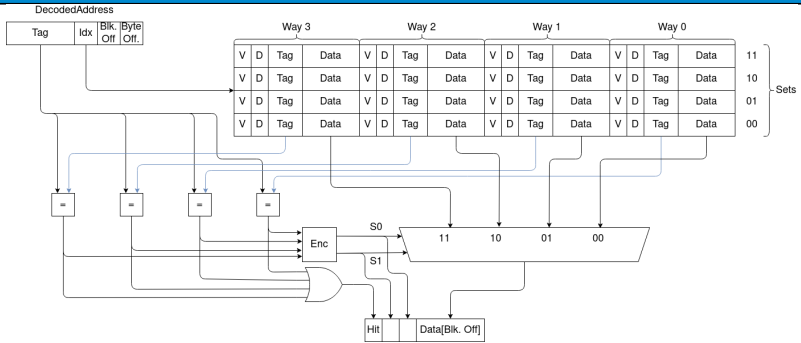
Architekturen und Entwurf von Rechnersystemen
Besprechung Übung 5 + Theorieblatt 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 2022/2023

Yannick Lavan
Fachgebiet Eingebettete Systeme und ihre Anwendungen



Übung 5: Caches

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Cache Theorie und Simulation

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulator für Rechensysteme

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulator für Rechensysteme
- Keine konkreten HDL Implementierungen nötig

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulator für Rechensysteme
- Keine konkreten HDL Implementierungen nötig
- Schnelle DSE mit vielen Systemparametern möglich

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulator für Rechensysteme
- Keine konkreten HDL Implementierungen nötig
- Schnelle DSE mit vielen Systemparametern möglich
- Auch Mikroarchitektur simulierbar

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulator für Rechensysteme
- Keine konkreten HDL Implementierungen nötig
- Schnelle DSE mit vielen Systemparametern möglich
- Auch Mikroarchitektur simulierbar
  - Pipelinestufen visualisierbar

gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulator für Rechensysteme
- Keine konkreten HDL Implementierungen nötig
- Schnelle DSE mit vielen Systemparametern möglich
- Auch Mikroarchitektur simulierbar
  - Pipelinestufen visualisierbar
  - Visualisierungsformat auch in HDL-Simulationen nutzbar

Aufgabe 5.1a - Grundbegriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein direct-mapped Cache ist _____ set-associative und ein vollassoziativer Cache mit *m* Cache-Blöcken ist _____ set-associative.

Aufgabe 5.1a - Grundbegriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein direct-mapped Cache ist 1-way set-associative und ein vollassoziativer Cache mit *m* Cache-Blöcken ist _____ set-associative.

Aufgabe 5.1a - Grundbegriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein direct-mapped Cache ist 1-way set-associative und ein vollassoziativer Cache mit *m* Cache-Blöcken ist *m*-way set-associative.

Aufgabe 5.1b - Cache-Konfiguration

TECHNISCHE UNIVERSITÄT DARMSTADT

Eckdaten:

- Speicher 1 kB Byte-adressierbar
- Cache Capacity 32 B, 4-way set-associative, Blockgröße 4 B
- Bestimmung Bitbreiten von Tag, Index, Offset

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Breite Offset:

$$w_{offset} = log_2(\text{Blockgröße}) = log_2(4) = 2$$

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Breite Offset:

$$w_{offset} = log_2(\text{Blockgröße}) = log_2(4) = 2$$

Breite Index:

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Breite Offset:

$$w_{offset} = log_2(\text{Blockgröße}) = log_2(4) = 2$$

Breite Index:

$$n_{blocks} = \frac{\text{Capacity}}{\text{Blockgröße}} = \frac{32}{4} = 8$$

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Breite Offset:

$$w_{offset} = log_2(\text{Blockgröße}) = log_2(4) = 2$$

Breite Index:

$$n_{blocks} = \frac{\text{Capacity}}{\text{Blockgröße}} = \frac{32}{4} = 8$$

$$n_{sets} = \frac{n_{blocks}}{\text{Associativity}} = \frac{8}{4} = 2$$

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Breite Offset:

$$w_{offset} = log_2(\text{Blockgröße}) = log_2(4) = 2$$

Breite Index:

$$n_{blocks} = \frac{\text{Capacity}}{\text{Blockgröße}} = \frac{32}{4} = 8$$

$$n_{sets} = \frac{n_{blocks}}{\text{Associativity}} = \frac{8}{4} = 2$$

$$w_{index} = log_2(n_{sets}) = log_2(2) = 1$$

Aufgabe 5.1b - Bitbreiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adressbreite gesamt:

$$w_{address} = log_2(1024) = 10$$

Breite Offset:

$$w_{offset} = log_2(\text{Blockgröße}) = log_2(4) = 2$$

Breite Index:

$$n_{blocks} = \frac{\text{Capacity}}{\text{Blockgröße}} = \frac{32}{4} = 8$$

$$n_{sets} = \frac{n_{blocks}}{\text{Associativity}} = \frac{8}{4} = 2$$

$$w_{index} = log_2(n_{sets}) = log_2(2) = 1$$

Breite Tag:

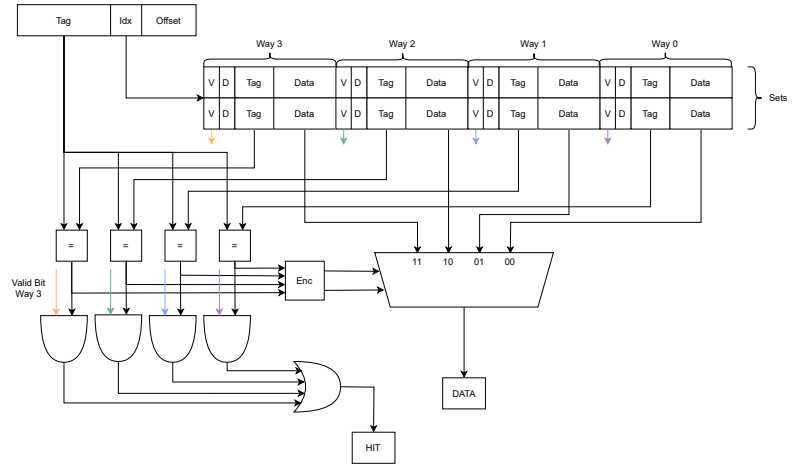
$$w_{tag} = w_{address} - (w_{index} + w_{offset}) = 10 - (1 + 2) = 7$$

Aufgabe 5.1b - Cache Skizze

TECHNISCHE UNIVERSITÄT DARMSTADT

Aufgabe 5.1b - Cache Skizze

TECHNISCHE UNIVERSITÄT DARMSTADT



Aufgabe 5.1c - Hit/Miss Bestimmung

TECHNISCHE UNIVERSITÄT DARMSTADT

Aufgabe 5.1c - Hit/Miss Bestimmung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zugriffe:
  0, 64, 4, 80, 8, 96, 12, 112, 128, 0,
  68, 4, 84, 8, 100, 12, 116, 132

Aufgabe 5.1c - Hit/Miss Bestimmung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zugriffe:
  0, 64, 4, 80, 8, 96, 12, 112, 128, 0, 68, 4, 84, 8, 100, 12, 116, 132
- Zugriffe in Hex:
  0x0, 0x40, 0x4, 0x50, 0x8, 0x60, 0xc, 0x70, 0x80, 0x0, 0x44, 0x4, 0x54, 0x8, 0x64, 0xc, 0x74, 0x84

Block 3	Block 2	Block 1	Block 0	Set
				1
				0

Aufgabe 5.1c - Hit/Miss Bestimmung

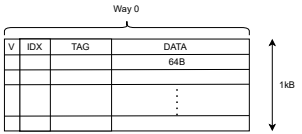
TECHNISCHE UNIVERSITÄT DARMSTADT

- Zugriffe:
  0, 64, 4, 80, 8, 96, 12, 112, 128, 0, 68, 4, 84, 8, 100, 12, 116, 132
- Zugriffe in Hex:
  0x0, 0x40, 0x4, 0x50, 0x8, 0x60, 0xc, 0x70, 0x80, 0x0, 0x44, 0x4, 0x54, 0x8, 0x64, 0xc, 0x74, 0x84
- Zugriff 10 Conflict Miss, Zugriffe 14 & 16 Capacity Misses

Block 3	Block 2	Block 1	Block 0	Set
$132_{18}, 84_{13}$	$12_{16}, 68_{11}$	$100_{15}, 12_{7}$	$116_{17}, 4_{3,12}$	1
$0_{10}, 8_{5}$	$128_{9}, 80_{4}$	$112_{8}, 64_{2}$	$8_{14}, 96_{6}, 0_{1}$	0

Aufgabe 5.2a - Matrixmultiplikation in gem5

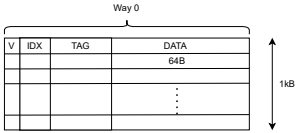
TECHNISCHE UNIVERSITÄT DARMSTADT

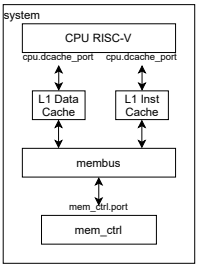


Aufgabe 5.2a - Matrixmultiplikation in gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ziel: Verstehen wie Software Performanz beeinflusst

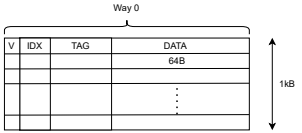


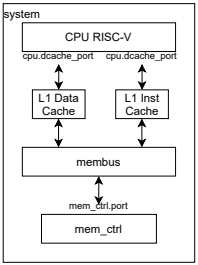


Aufgabe 5.2a - Matrixmultiplikation in gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ziel: Verstehen wie Software Performanz beeinflusst
- Erste Verwendung von gem5

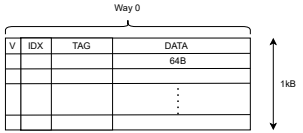


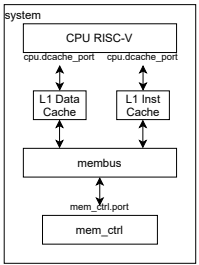


Aufgabe 5.2a - Matrixmultiplikation in gem5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ziel: Verstehen wie Software Performanz beeinflusst
- Erste Verwendung von gem5
- Software: Multiplikation zweier Matrizen ($[50 \times 10] \cdot [10 \times 50]$)





Aufgabe 5.2a - Matrixmultiplikation Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

Aufgabe 5.2a - Matrixmultiplikation Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- **`result[row_a][col_b]`**:
  $2 \cdot rows_b \cdot cols_b \cdot rows_a = 2 \cdot 10 \cdot 50 \cdot 50$

Aufgabe 5.2a - Matrixmultiplikation Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- **`result[row_a][col_b]`**:
  $2 \cdot rows_b \cdot cols_b \cdot rows_a = 2 \cdot 10 \cdot 50 \cdot 50$
- Ähnlich für **`a[row_a][k]`** und **`b[k][col_b]`**

Aufgabe 5.2a - Matrixmultiplikation Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- **`result[row_a][col_b]`**:
  $2 \cdot rows_b \cdot cols_b \cdot rows_a = 2 \cdot 10 \cdot 50 \cdot 50$
- Ähnlich für **`a[row_a][k]`** und **`b[k][col_b]`**
- $\Rightarrow 4 \cdot 10 \cdot 50 \cdot 50$

Aufgabe 5.2a - Matrixmultiplikation Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- **`result[row_a][col_b]`**:
  $2 \cdot rows_b \cdot cols_b \cdot rows_a = 2 \cdot 10 \cdot 50 \cdot 50$
- Ähnlich für **`a[row_a][k]`** und **`b[k][col_b]`**
- $\Rightarrow 4 \cdot 10 \cdot 50 \cdot 50$
- Initialisierung **`result`**: $50 \cdot 50$

Aufgabe 5.2a - Matrixmultiplikation Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- **`result[row_a][col_b]`**:
  $2 \cdot rows_b \cdot cols_b \cdot rows_a = 2 \cdot 10 \cdot 50 \cdot 50$
- Ähnlich für **`a[row_a][k]`** und **`b[k][col_b]`**
- $\Rightarrow 4 \cdot 10 \cdot 50 \cdot 50$
- Initialisierung **`result`**: $50 \cdot 50$

$\Rightarrow 50 \cdot 50 + 4 \cdot 10 \cdot 50 \cdot 50 = 77500$

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe
- 359115 Hits

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe
- 359115 Hits
- 36143 Misses

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe
- 359115 Hits
- 36143 Misses
- Hitrate $\approx 90.8\%$

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe
- 359115 Hits
- 36143 Misses
- Hitrate $\approx 90.8\%$
- 395258 > 77500

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe
- 359115 Hits
- 36143 Misses
- Hitrate $\approx 90.8\%$
- 395258 > 77500
  - Wie entsteht dieser Unterschied?

Aufgabe 5.2a - Tatsächliche Speicherzugriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Insgesamt 395258 Speicherzugriffe
- 359115 Hits
- 36143 Misses
- Hitrate $\approx 90.8\%$
- 395258 > 77500
  - Wie entsteht dieser Unterschied?
  - Lokale Variablen! Nicht immer Platz in Register File $\Rightarrow$ Stack

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- `A[3][4]` = 0x6f088
- `A[2][4]` = 0x6f060
- `A[2][5]` = 0x6f064

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- `A[3][4]` = 0x6f088
- `A[2][4]` = 0x6f060
- `A[2][5]` = 0x6f064
- Welche Variante terminiert nach kürzerer Zeit?

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- `A[3][4]` = 0x6f088
- `A[2][4]` = 0x6f060
- `A[2][5]` = 0x6f064
- Welche Variante terminiert nach kürzerer Zeit?
  - Cache: 16 Sets mit 64B pro Set ⇒ betrachte letzte 10 Bit (4 Index, 6 Offset)

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- **A[3][4]** = 0x6f088
- **A[2][4]** = 0x6f060
- **A[2][5]** = 0x6f064
- Welche Variante terminiert nach kürzerer Zeit?
  - Cache: 16 Sets mit 64B pro Set ⇒ betrachte letzte 10 Bit (4 Index, 6 Offset)
  - **A[3][4]** liegt in Set 2, **A[2][4]** und **A[2][5]** in Set 1.

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- **A[3][4]** = 0x6f088
- **A[2][4]** = 0x6f060
- **A[2][5]** = 0x6f064
- Welche Variante terminiert nach kürzerer Zeit?
  - Cache: 16 Sets mit 64B pro Set ⇒ betrachte letzte 10 Bit (4 Index, 6 Offset)
  - **A[3][4]** liegt in Set 2, **A[2][4]** und **A[2][5]** in Set 1.
  - **A[2][4]** wird in den Cache geladen ⇒ **A[2][5]** wird in Cache geladen

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- **A[3][4]** = 0x6f088
- **A[2][4]** = 0x6f060
- **A[2][5]** = 0x6f064
- Welche Variante terminiert nach kürzerer Zeit?
  - Cache: 16 Sets mit 64B pro Set ⇒ betrachte letzte 10 Bit (4 Index, 6 Offset)
  - **A[3][4]** liegt in Set 2, **A[2][4]** und **A[2][5]** in Set 1.
  - **A[2][4]** wird in den Cache geladen ⇒ **A[2][5]** wird in Cache geladen

  ⇒ greife zeitnah auf **A[2][5]** zu

Aufgabe 5.2b - Loop Interchange 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Variante A:
for(j=0;j<10;j++)
  for(i=0;i<100;i++)
    A[i][j] = 2*A[i][j] ;

// Variante B:
int i,j;
for(i=0;i<100;i++)
  for(j=0;j<10;j++)
    A[i][j] = 2*A[i][j] ;
```

- **A[3][4]** = 0x6f088
- **A[2][4]** = 0x6f060
- **A[2][5]** = 0x6f064
- Welche Variante terminiert nach kürzerer Zeit?
  - Cache: 16 Sets mit 64B pro Set ⇒ betrachte letzte 10 Bit (4 Index, 6 Offset)
  - **A[3][4]** liegt in Set 2, **A[2][4]** und **A[2][5]** in Set 1.
  - **A[2][4]** wird in den Cache geladen ⇒ **A[2][5]** wird in Cache geladen

  ⇒ greife zeitnah auf **A[2][5]** zu

  ⇒ Variante B schneller, da weniger Misses

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

$\Rightarrow$ Wird ein Element geladen werden auch andere Elemente in der gleichen Reihe geladen

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

⇒ Wird ein Element geladen werden auch andere Elemente in der gleichen Reihe geladen

⇒ Konsekutive Iterationen über Spalten

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

⇒ Wird ein Element geladen werden auch andere Elemente in der gleichen Reihe geladen

⇒ Konsekutive Iterationen über Spalten

⇒ Iterationsvariablen von außen nach innen:

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

⇒ Wird ein Element geladen werden auch andere Elemente in der gleichen Reihe geladen

⇒ Konsekutive Iterationen über Spalten

⇒ Iterationsvariablen von außen nach innen:

  - **`row_a`**, indiziert immer Reihe

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

⇒ Wird ein Element geladen werden auch andere Elemente in der gleichen Reihe geladen

⇒ Konsekutive Iterationen über Spalten

⇒ Iterationsvariablen von außen nach innen:

- **`row_a`**, indiziert immer Reihe
- **`k`**, indiziert Spalte während **`row_a`** Reihe angibt. Indiziert Reihe während **`col_b`** Spalte angibt

Aufgabe 5.2b - Loop Interchange 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (row_a = 0; row_a < rows_a; row_a++)
{
    for (col_b = 0; col_b < cols_b; col_b++)
    {
        result[row_a][col_b] = 0;
        for (k = 0; k < rows_b; k++)
        {
            result[row_a][col_b] += a[row_a][k] * b[k][col_b];
        }
    }
}
```

- Zur Erinnerung: Spalten einer Reihe liegen im Speicher hintereinander

⇒ Wird ein Element geladen werden auch andere Elemente in der gleichen Reihe geladen

⇒ Konsekutive Iterationen über Spalten

⇒ Iterationsvariablen von außen nach innen:

- **`row_a`**, indiziert immer Reihe
- **`k`**, indiziert Spalte während **`row_a`** Reihe angibt. Indiziert Reihe während **`col_b`** Spalte angibt
- **`col_b`**, indiziert immer Spalte

Aufgabe 5.2b - Loop Interchange 3

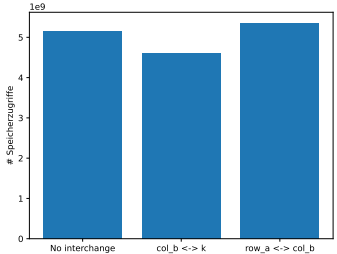


Abbildung: Anzahl der Speicherzugriffe

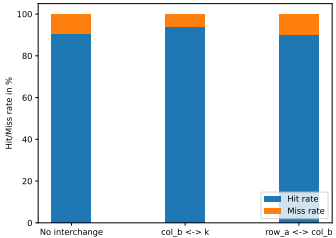


Abbildung: Hit und Miss Rates im Vergleich

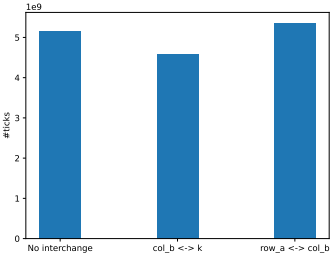


Abbildung: Simulationticks im Vergleich

Aufgabe 5.3 - Untersuchung verschiedener Cache Architekturen

TECHNISCHE UNIVERSITÄT DARMSTADT

Aufgabe 5.3 - Untersuchung verschiedener Cache Architekturen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Cache Konfiguration wirkt sich auf Anzahl und Art der Misses aus

Aufgabe 5.3 - Untersuchung verschiedener Cache Architekturen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Cache Konfiguration wirkt sich auf Anzahl und Art der Misses aus
- Schrittweise Evaluation verschiedener Cache-Parameter bei gleichbleibendem Programm

Aufgabe 5.3 - Untersuchung verschiedener Cache Architekturen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Cache Konfiguration wirkt sich auf Anzahl und Art der Misses aus
- Schrittweise Evaluation verschiedener Cache-Parameter bei gleichbleibendem Programm

⇒ Gleichbleibende Datengröße und Anzahl Speicherzugriffe

Aufgabe 5.3a - Untersuchung verschiedener Cache Größen

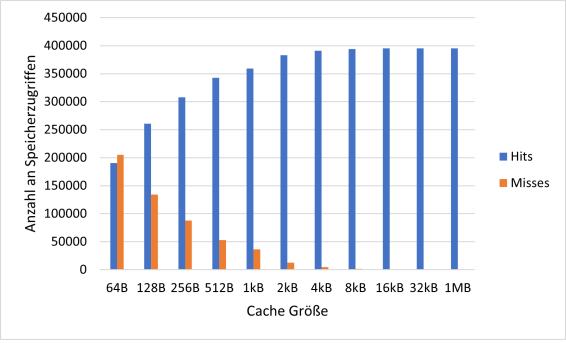


Abbildung: Cache Hits und Misses für verschiedene Cache Größen

Aufgabe 5.3a - Untersuchung verschiedener Cache Größen

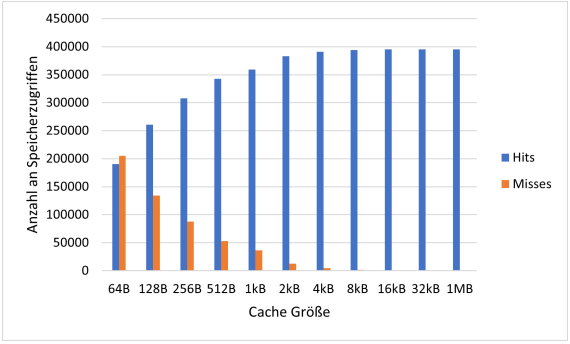


Abbildung: Cache Hits und Misses für verschiedene Cache Größen

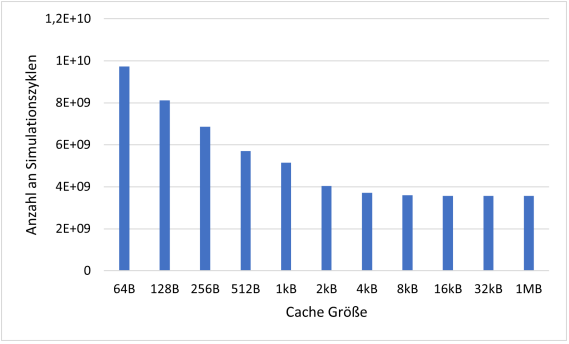


Abbildung: Ausführungszeiten für verschiedene Cache Größen

Aufgabe 5.3b - Untersuchung der Assoziativität

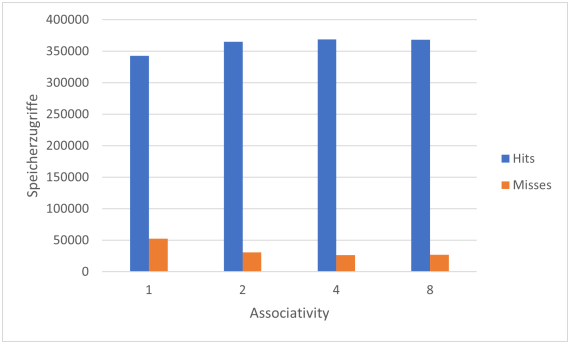
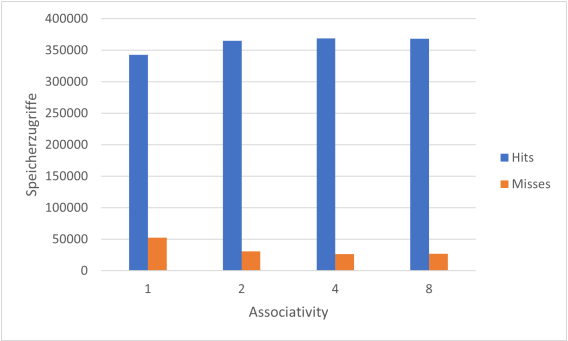


Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

Aufgabe 5.3b - Untersuchung der Assoziativität



- Miss Rate sinkt mit steigender Associativity

Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

Aufgabe 5.3b - Untersuchung der Assoziativität

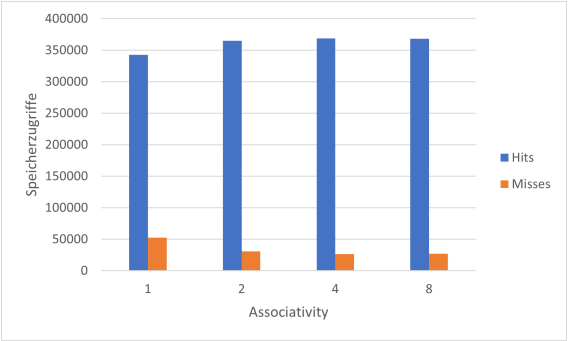


Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

- Miss Rate sinkt mit steigender Associativity
  - Ausnahme: 8-way set-associative Cache. Warum?

Aufgabe 5.3b - Untersuchung der Assoziativität

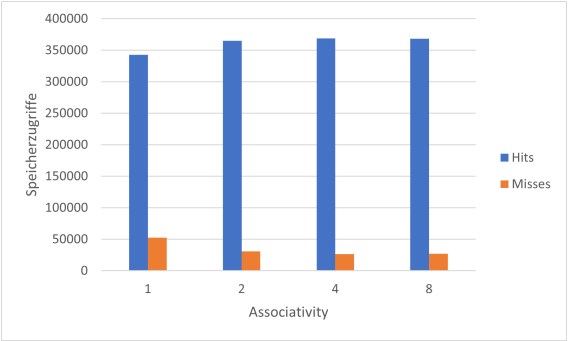


Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

- Miss Rate sinkt mit steigender Associativity
  - Ausnahme: 8-way set-associative Cache. Warum?
  - ⇒ 8-way set-associative Cache = vollassoziativer Cache

Aufgabe 5.3b - Untersuchung der Assoziativität

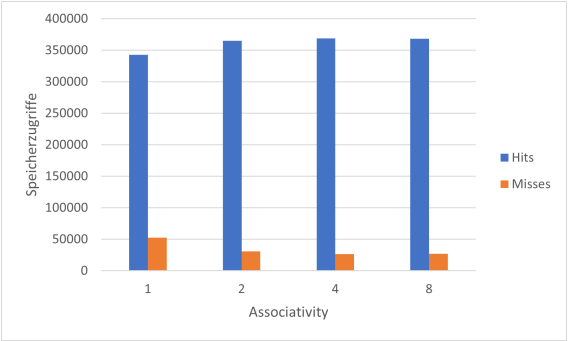


Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

- Miss Rate sinkt mit steigender Associativity
  - Ausnahme: 8-way set-associative Cache. Warum?
  - ⇒ 8-way set-associative Cache = vollassoziativer Cache
  - ⇒ Einfluss LRU-Strategie und ausgeführte Anwendung

Aufgabe 5.3b - Untersuchung der Assoziativität

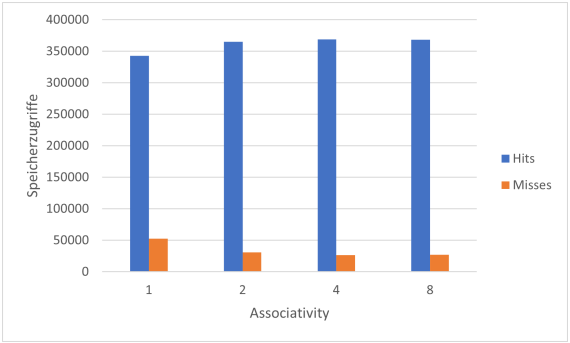


Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

- Miss Rate sinkt mit steigender Associativity
  - Ausnahme: 8-way set-associative Cache. Warum?
  - ⇒ 8-way set-associative Cache = vollassoziativer Cache
  - ⇒ Einfluss LRU-Strategie und ausgeführte Anwendung
  - ⇒ Ältester Eintrag wird entfernt, obwohl potenziell bald benötigt

Aufgabe 5.3b - Untersuchung der Assoziativität

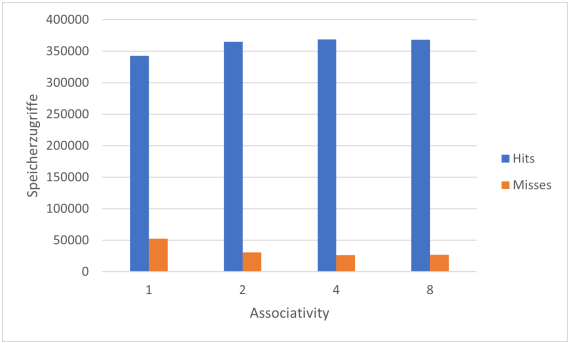


Abbildung: Cache Hits und Misses für verschiedene Assoziativitäten

- Miss Rate sinkt mit steigender Associativity
  - Ausnahme: 8-way set-associative Cache. Warum?
  - ⇒ 8-way set-associative Cache = vollassoziativer Cache
  - ⇒ Einfluss LRU-Strategie und ausgeführte Anwendung
  - ⇒ Ältester Eintrag wird entfernt, obwohl potenziell bald benötigt
  - Mehrere Sets reduzieren Wahrscheinlichkeit für diesen Fall

Aufgabe 5.3c - Untersuchung der Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

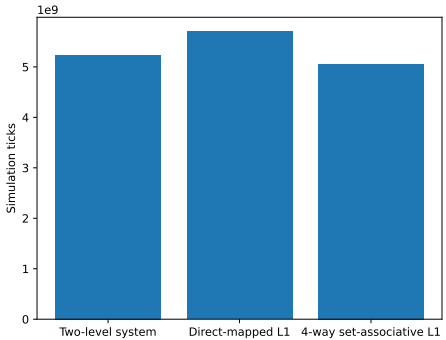
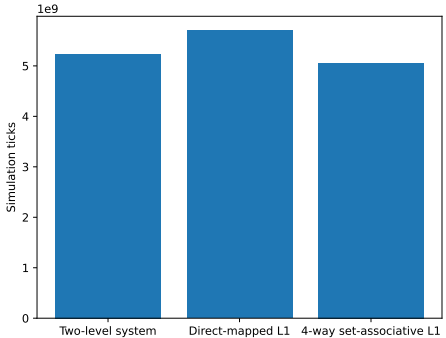


Abbildung: Ausführungszeit für verschiedene Hierarchiekonfigurationen

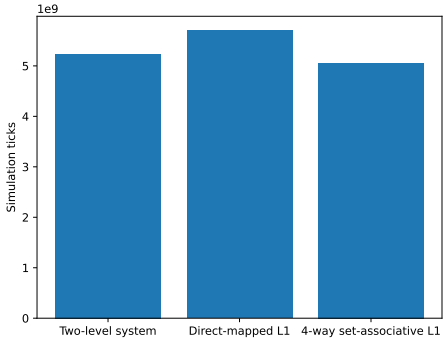
Aufgabe 5.3c - Untersuchung der Hierarchie



- System mit zwei Cachestufen schneller als einzelner direct-mapped Cache

Abbildung: Ausführungszeit für verschiedene Hierarchiekonfigurationen

Aufgabe 5.3c - Untersuchung der Hierarchie

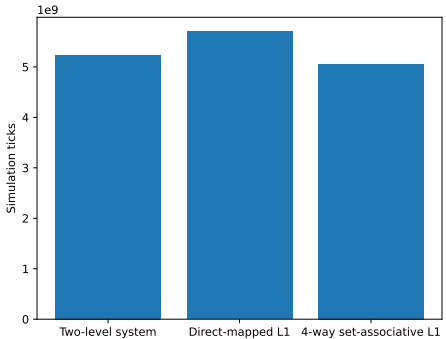


- System mit zwei Cachestufen schneller als einzelner direct-mapped Cache

⇒ Niedrigere Miss-Penalty

Abbildung: Ausführungszeit für verschiedene Hierarchiekonfigurationen

Aufgabe 5.3c - Untersuchung der Hierarchie



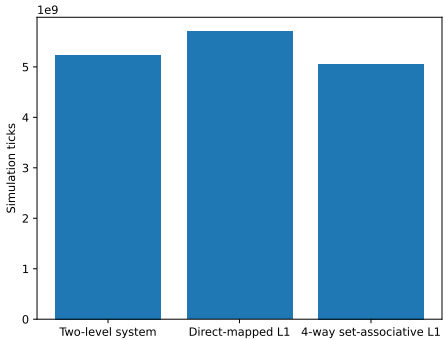
- System mit zwei Cachestufen schneller als einzelner direct-mapped Cache

⇒ Niedrigere Miss-Penalty

- Aber: Zweistufiges System langsamer als 4-way set-associative L1 Cache

Abbildung: Ausführungszeit für verschiedene Hierarchiekonfigurationen

Aufgabe 5.3c - Untersuchung der Hierarchie



- System mit zwei Cachestufen schneller als einzelner direct-mapped Cache

⇒ Niedrigere Miss-Penalty

- Aber: Zweistufiges System langsamer als 4-way set-associative L1 Cache

⇒ Weniger Conflict Misses ⇒ keine Miss-Penalty

Abbildung: Ausführungszeit für verschiedene Hierarchiekonfigurationen

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Performanz steigt mit wachsender I$-Größe

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

- Performanz steigt mit wachsender I$-Größe
- Stagniert ab 512B Größe

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

- Performanz steigt mit wachsender I$-Größe
- Stagniert ab 512B Größe
- Warum?

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

- Performanz steigt mit wachsender I$-Größe
- Stagniert ab 512B Größe
- Warum?
  1. Performanzsteigerung: Programm hauptsächlich Schleifen $\Rightarrow$ weniger Capacity Misses durch höhere Capacity

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

- Performanz steigt mit wachsender I$-Größe
- Stagniert ab 512B Größe
- Warum?
  1. Performanzsteigerung: Programm hauptsächlich Schleifen $\Rightarrow$ weniger Capacity Misses durch höhere Capacity
  2. Stagnation: Programm gerade einmal 576B groß

Aufgabe 5.3d - Untersuchung des Instruktionscaches

TECHNISCHE UNIVERSITÄT DARMSTADT

I$-Größe	Ticks
128B	12771783000
256B	8516973000
512B	5147377000
1kB	5147377000
64kB	5147377000

- Performanz steigt mit wachsender I$-Größe
- Stagniert ab 512B Größe
- Warum?
  1. Performanzsteigerung: Programm hauptsächlich Schleifen $\Rightarrow$ weniger Capacity Misses durch höhere Capacity
  2. Stagnation: Programm gerade einmal 576B groß

  $\Rightarrow$ Performanzrelevanter Code passt vollständig in Cache

Aufgabe 5.3e - Performanz ohne Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

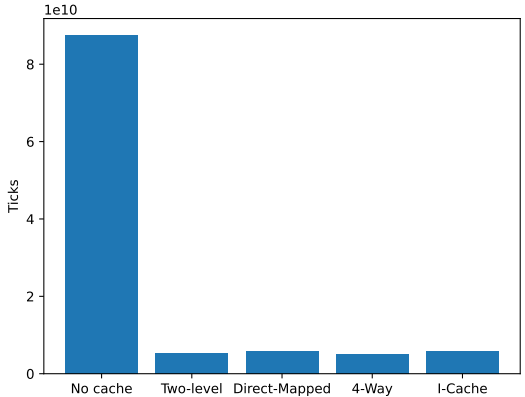


Abbildung: Simulationsticks für verschiedene Cache-Konfigurationen aus dieser Übung

Cache Hardware

Cache Hardware - Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vertiefung der in den vorherigen Aufgaben erlangten Kenntnisse
- Vertiefung BSV-Kenntnisse der vorherigen Übungen
  - Vector
  - Structs/Tagged-Unions
  - for-Schleifen
  - FSMs
  - Subinterfaces
- Ähnliche Konzepte in Betriebssystemen (Paging)

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:
  - Cache verwaltet nur Daten

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:
  - Cache verwaltet nur Daten
  - Controller steuert Cache (Anfragen, Befüllung, Ersetzungsstrategie)

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:
  - Cache verwaltet nur Daten
  - Controller steuert Cache (Anfragen, Befüllung, Ersetzungsstrategie)
- Cache Interface:

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:
  - Cache verwaltet nur Daten
  - Controller steuert Cache (Anfragen, Befüllung, Ersetzungsstrategie)
- Cache Interface:
  - **`ls_server`**-Interface - Server Interface, das Tagged Union akzeptiert und ein **`Maybe`** mit Tagged Union zurückliefert

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:
  - Cache verwaltet nur Daten
  - Controller steuert Cache (Anfragen, Befüllung, Ersetzungsstrategie)
- Cache Interface:
  - **`ls_server`**-Interface - Server Interface, das Tagged Union akzeptiert und ein **`Maybe`** mit Tagged Union zurückliefert
  - **`updateBlock`** - Ersetze ganzen Cache-Block

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- N-way set-associative Cache
- Write-back Strategie
- Ansatz:
  - Cache verwaltet nur Daten
  - Controller steuert Cache (Anfragen, Befüllung, Ersetzungsstrategie)
- Cache Interface:
  - **`ls_server`**-Interface - Server Interface, das Tagged Union akzeptiert und ein **`Maybe`** mit Tagged Union zurückliefert
  - **`updateBlock`** - Ersetze ganzen Cache-Block
  - **`getBlock`** - Liefere ganzen Cache-Block (falls valide) oder **`tagged Invalid`**

Cache Hardware - BSV Typen

TECHNISCHE UNIVERSITÄT DARMSTADT

Structs und Typedefs für bessere Lesbarkeit des Codes

```
typedef Bit#(ADDR_WIDTH)          Address;
typedef Bit#(WORD_WIDTH)          Data;
typedef Bit#(TAG_WIDTH)           Tag;
typedef Bit#(BLOCK_OFFSET_WIDTH)
    ↪   BlockOffset;
typedef Bit#(BYTE_OFFSET_WIDTH)
    ↪   ByteOffset;
typedef Bit#(INDEX_WIDTH)         Index;
typedef Bit#(WAY_WIDTH)           Way;

typedef struct {
  Tag tag;
  Index index;
  BlockOffset block_offset;
  ByteOffset byte_offset;
} DecodedAddress deriving (Bits, FShow);
```

```
typedef struct {
  Bool dirty;
  Tag tag;
  Vector#(WORDS_PER_BLOCK, Data)
  ↪   data_block;
} CacheEntry deriving (Bits, FShow);

instance DefaultValue#(CacheEntry);
  defaultValue = CacheEntry {
    dirty: False,
    tag: 0,
    data_block: defaultValue
  };
endinstance
```

Cache Hardware - Cache Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkCache(Cache);
2    FIFO#(CacheReq) req_fifo <- mkFIFO;
3    FIFO#(Maybe#(CacheRsp)) rsp_fifo <- mkFIFO;
4    Vector#(N_SETS, Vector#(ASSOCIATIVITY, Reg#(Maybe#(CacheEntry)))) data_field <-
     ↪   replicateM(replicateM(mkReg(tagged Invalid)));
5
6    // Internal logic on next slides...
7
8    method Action updateBlock(CacheEntry new_entry, Index set, Way position);
9      data_field[set][position] <= tagged Valid new_entry;
10   endmethod
11
12   method Maybe#(CacheEntry) getBlock(Index set, Way position);
13     Maybe#(CacheEntry) block = data_field[set][position];
14     return block;
15   endmethod
16
17   interface Server ls_server;
18     interface Put request = toPut(req_fifo);
19     interface Get response = toGet(rsp_fifo);
20   endinterface
21 endmodule
```

Cache Hardware - Blocksuche

TECHNISCHE UNIVERSITÄT DARMSTADT

Ziel: Finde Weg in Set mit passendem Tag

```
1  function Tuple2#(Bool, Way) findWay(Vector#(ASSOCIATIVITY, Reg#(Maybe#(CacheEntry))) set, Tag tag);
2    Way pos = 0;
3    Bool hit = False;
4
5    for(Integer i = 0; i < valueOf(ASSOCIATIVITY); i = i + 1) begin
6      let entry_i = set[i];
7      if(entry_i matches tagged Valid .e) begin
8        if(e.tag == tag) begin
9          pos = fromInteger(i);
10         hit = True;
11       end
12     end
13   end
14   return tuple2(hit, pos);
15 endfunction
```

Cache Hardware - Cache Requests

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  rule handle_req;
2    let req = req_fifo.first();
3    req_fifo.deq();
4    DecodedAddress addr;
5    Data data = 0;
6    if(req matches tagged Load .load)
7        addr = load; // LoadReq was just a typedef to DecodedAddress
8    else begin
9        addr = req.Store.addr;
10       data = req.Store.data;
11   end
12
13   let set = addr.index;
14   let row = data_field[set];
15
16   match {.hit, .pos} = findWay(row, addr.tag);
17   CacheRsp rsp = unpack(0);
```

Cache Hardware - Cache Responses

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  if(hit) begin
2    let entry_data = fromMaybe(defaultValue, row[pos]).data_block;
3    if(req matches tagged Load .load) begin
4      data = entry_data[addr.block_offset];
5      rsp = tagged LoadResponse LoadRsp{way: pos, data: data};
6    end
7    else if(req matches tagged Store .store) begin
8      entry_data[addr.block_offset] = data;
9
10     CacheEntry upd = CacheEntry{
11         dirty: True,
12         tag: addr.tag,
13         data_block: entry_data
14     };
15     data_field[set][pos] <= tagged Valid upd;
16     rsp = tagged StoreResponse pos;
17   end
18   rsp_fifo.enq(tagged Valid rsp);
19 end
20 else
21   rsp_fifo.enq(tagged Invalid);
22 endrule
```

Cache Hardware - Cache Implementierung Ende

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vertiefte Konzepte:
  - Structs
  - Pattern Matching
  - Maybe
  - Tuples
  - Schleifen
- Fragen zu `mkCache`?

TECHNISCHE UNIVERSITÄT DARMSTADT

Fragen zu Übung 5?