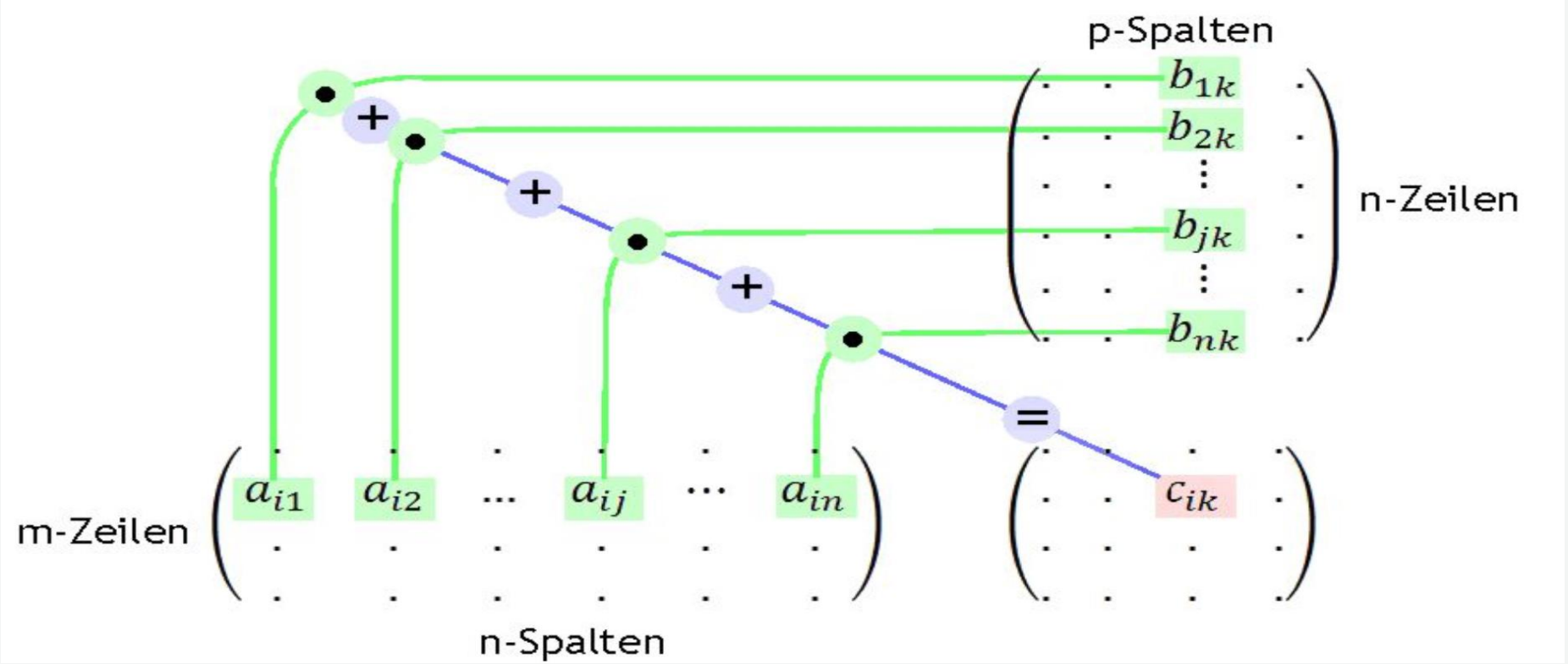
Matrizenmultiplikation

- Charakteristik
- Speicherung: Stack oder Heap
- Datentypen: Int , Float, Double
- Verbesserung der Lokalität

Matrizenmultiplikation



Vorname: Nouralrahman
Nachname: Hussain

Charakteristik

- Das Programm allokiert Platz in Heap oder Stack für 3 Matrizen
- Jede Matrix hat (1000*1000*4byte)=4 Megabyte Speichergröße. Bei manchen Versuchen ist die Zahl kleiner, sodass es mit dem Speichergröße anpasst.
- Installierung Erfolg durch random-numbers.
- Es werden 2 Matrizen miteinander multipliziert und das Ergebnis wird in einem 3ten Matrix gespeichert.

Stack Heap

Heap

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("NS elapsed: %llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k, sum;
    clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allokation der Matrizen
    start_alloc = clock();
    int **matrix_a = (int **)malloc(N * sizeof(int *));
    int **matrix_b = (int **)malloc(N * sizeof(int *));
    int **matrix_c = (int **)malloc(N * sizeof(int *));
    for (i = 0; i < N; i++) {
        matrix_a[i] = (int *)malloc(N * sizeof(int));
        matrix_b[i] = (int *)malloc(N * sizeof(int));
        matrix_c[i] = (int *)malloc(N * sizeof(int));
    }
    end_alloc = clock();

    // Initialisierung der Matrizen
    start_init = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i][j] = rand() % 100;
            matrix_b[i][j] = rand() % 100;
            matrix_c[i][j] = 0;
        }
    }
    end_init = clock();

    // Matrixmultiplikation
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum;
        }
    }
    end_mult = clock();

    // Berechnung der Zeiten
   // printf("Allocieren\n");
    compute_timediff(start_alloc, end_alloc, 1);
    //printf("Initialisieren\n");
    compute_timediff(start_init, end_init, 1);
    //printf("Multiplizieren\n");
    compute_timediff(start_mult, end_mult, 1);
    //printf("wobei die Laenge der Matrizen = 1000\n");

    // Freigabe des Speichers
    for (i = 0; i < N; i++) {
        free(matrix_a[i]);
        free(matrix_b[i]);
        free(matrix_c[i]);
    }
    free(matrix_a);
    free(matrix_b);
    free(matrix_c);

    return 0;
}

```

Stack

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("NS elapsed: %llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k, sum;
    clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allokation der Matrizen
    start_alloc = clock();
    int matrix_a[N][N];
    int matrix_b[N][N];
    int matrix_c[N][N];
    end_alloc = clock();

    // Initialisierung der Matrizen
    start_init = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i][j] = rand() % 100;
            matrix_b[i][j] = rand() % 100;
            matrix_c[i][j] = 0;
        }
    }
    end_init = clock();

    // Matrixmultiplikation
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum;
        }
    }
    end_mult = clock();

    // Berechnung der Zeiten
    //printf("Allocieren\n");
    compute_timediff(start_alloc, end_alloc, 1);
    //printf("Initialisieren\n");
    compute_timediff(start_init, end_init, 1);
    //printf("Multiplizieren\n");
    compute_timediff(start_mult, end_mult, 1);
    //printf("wobei die Laenge der Matrizen = 1000\n");

    return 0;
}

```

Stack

Vorteile

- Schnellere Allokation: "Push" und "Pop„
- Deterministische Speicherverwaltung: reproduzierbar. Minimierung von Fehler wegen Automatische Bearbeitung

Nachteile

- Begrenzte Größe: Stack Overflow
- Statische Speicherzuweisung: Zur Kompilierzeit bekannt

Heap

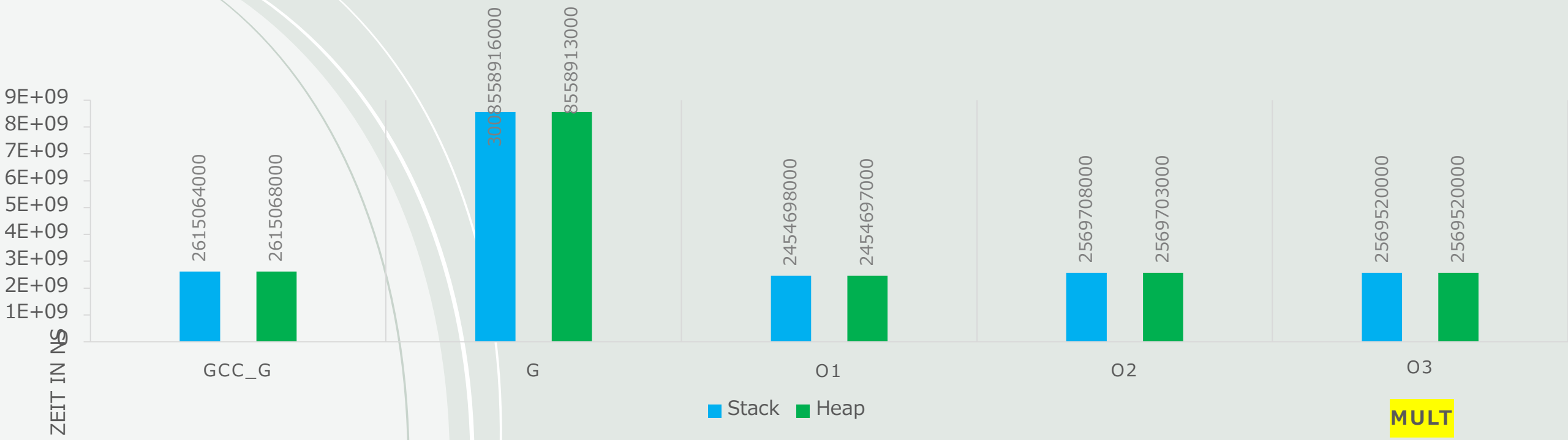
Vorteile

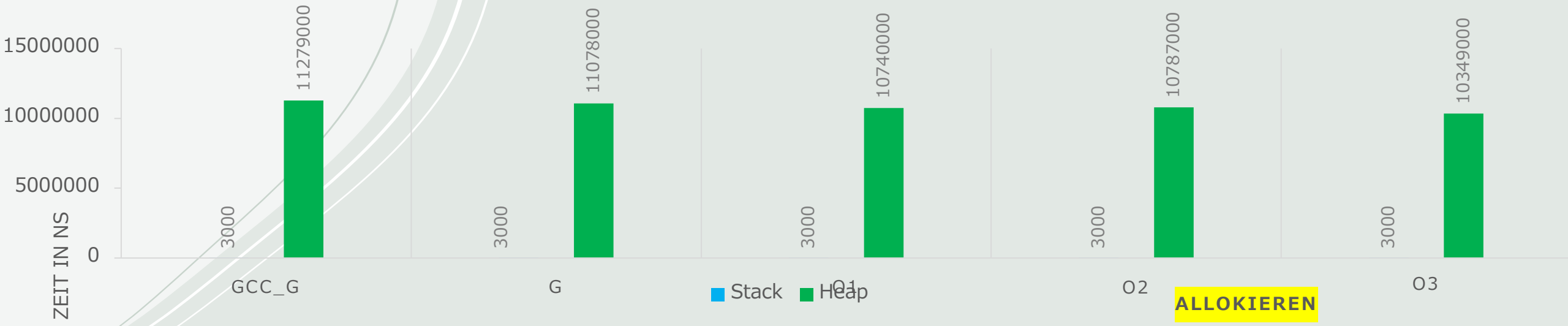
- Dynamische Speicherallokation: flexibler verwalten
- Größere Speicherkapazität: Im Vergleich zum Stack kann der Heap normalerweise eine größere Menge an Speicher zur Verfügung

Nachteile

- Langsamere Allokation: Suche geeigneten Speicherblock Aktualisierung des Heap-Managers
- Manuelle Speicherverwaltung: "Memory Leak"

In den meisten Fällen soll nicht zu sehr auf die Geschwindigkeit der Speicherallokation konzentrieren, da moderne Compiler und Laufzeitsysteme optimierte Mechanismen für die Speicherverwaltung verwenden, um die Leistung zu maximieren.


9E+09
8E+09
7E+09
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
ZEIT IN NS
2615064000
2615068000
3008558916000
8558913000
2454698000
2454697000
2569708000
2569703000
2569520000
2569520000
GCC_G
G
O1
O2
O3
Stack
Heap
MULT


15000000
10000000
5000000
0
ZEIT IN NS
3000
11279000
3000
11078000
3000
10740000
3000
10787000
3000
10349000
GCC_G
G
O1
O2
O3
Stack
Heap
ALLOKIEREN

Datentypen

int

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("NS elapsed: %llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k, sum;
    clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allokation der Matrizen
    start_alloc = clock();
    int **matrix_a = (int **)malloc(N * sizeof(int *));
    int **matrix_b = (int **)malloc(N * sizeof(int *));
    int **matrix_c = (int **)malloc(N * sizeof(int *));
    for (i = 0; i < N; i++) {
        matrix_a[i] = (int *)malloc(N * sizeof(int));
        matrix_b[i] = (int *)malloc(N * sizeof(int));
        matrix_c[i] = (int *)malloc(N * sizeof(int));
    }
    end_alloc = clock();

    // Initialisierung der Matrizen
    start_init = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i][j] = rand() % 100;
            matrix_b[i][j] = rand() % 100;
            matrix_c[i][j] = 0;
        }
    }
    end_init = clock();

    // Matrixmultiplikation
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum;
        }
    }
    end_mult = clock();

    // Berechnung der Zeiten
    // printf("Allocieren\n");
    compute_timediff(start_alloc, end_alloc, 1);
    //printf("Initialisieren\n");
    compute_timediff(start_init, end_init, 1);
    //printf("Multiplizieren\n");
    compute_timediff(start_mult, end_mult, 1);
    //printf("wobei die Laenge der Matrizen = 1000\n");

    // Freigabe des Speichers
    for (i = 0; i < N; i++) {
        free(matrix_a[i]);
        free(matrix_b[i]);
        free(matrix_c[i]);
    }
    free(matrix_a);
    free(matrix_b);
    free(matrix_c);

    return 0;
}
```

float

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("%llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k;
    float sum;
    clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allocation of matrices
    start_alloc = clock();
    float **matrix_a = (float **)malloc(N * sizeof(float *));
    float **matrix_b = (float **)malloc(N * sizeof(float *));
    float **matrix_c = (float **)malloc(N * sizeof(float *));
    for (i = 0; i < N; i++) {
        matrix_a[i] = (float *)malloc(N * sizeof(float));
        matrix_b[i] = (float *)malloc(N * sizeof(float));
        matrix_c[i] = (float *)malloc(N * sizeof(float));
    }
    end_alloc = clock();

    // Initialization of matrices
    start_init = clock();
    srand(time(NULL));
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i][j] = ((float)rand() / RAND_MAX) * 100.0f;
            matrix_b[i][j] = ((float)rand() / RAND_MAX) * 100.0f;
            matrix_c[i][j] = 0.0f;
        }
    }
    end_init = clock();

    // Matrix multiplication
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0.0f;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum;
        }
    }
    end_mult = clock();

    // Calculation of times
    // printf("Allocation\n");
    compute_timediff(start_alloc, end_alloc, 1);
    //printf("Initialization\n");
    compute_timediff(start_init, end_init, 1);
    //printf("Multiplication\n");
    compute_timediff(start_mult, end_mult, 1);
    //printf("where the length of matrices = 1000\n");

    // Freeing memory
    for (i = 0; i < N; i++) {
        free(matrix_a[i]);
        free(matrix_b[i]);
        free(matrix_c[i]);
    }
    free(matrix_a);
    free(matrix_b);
    free(matrix_c);

    return 0;
}
```

double

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("NS elapsed: %llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k;
    double sum;
    clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allocation of matrices
    start_alloc = clock();
    double **matrix_a = (double **)malloc(N * sizeof(double *));
    double **matrix_b = (double **)malloc(N * sizeof(double *));
    double **matrix_c = (double **)malloc(N * sizeof(double *));
    for (i = 0; i < N; i++) {
        matrix_a[i] = (double *)malloc(N * sizeof(double));
        matrix_b[i] = (double *)malloc(N * sizeof(double));
        matrix_c[i] = (double *)malloc(N * sizeof(double));
    }
    end_alloc = clock();

    // Initialization of matrices
    start_init = clock();
    srand(time(NULL));
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i][j] = ((double)rand() / RAND_MAX) * 100.0;
            matrix_b[i][j] = ((double)rand() / RAND_MAX) * 100.0;
            matrix_c[i][j] = 0.0;
        }
    }
    end_init = clock();

    // Matrix multiplication
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0.0;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum;
        }
    }
    end_mult = clock();

    // Calculation of times
    //printf("Allocation\n");
    compute_timediff(start_alloc, end_alloc, 1);
    // printf("Initialization\n");
    compute_timediff(start_init, end_init, 1);
    // printf("Multiplication\n");
    compute_timediff(start_mult, end_mult, 1);
    // printf("where the length of matrices = 1000\n");

    // Freeing memory
    for (i = 0; i < N; i++) {
        free(matrix_a[i]);
        free(matrix_b[i]);
        free(matrix_c[i]);
    }
    free(matrix_a);
    free(matrix_b);
    free(matrix_c);

    return 0;
}
```

Datentyp

Int

- 4byte 32bit
- ohne Genauigkeitsverlust
- weniger Speicherplatz
- schnellere Berechnungen
- Eingeschränkte Darstellung von Dezimalzahlen

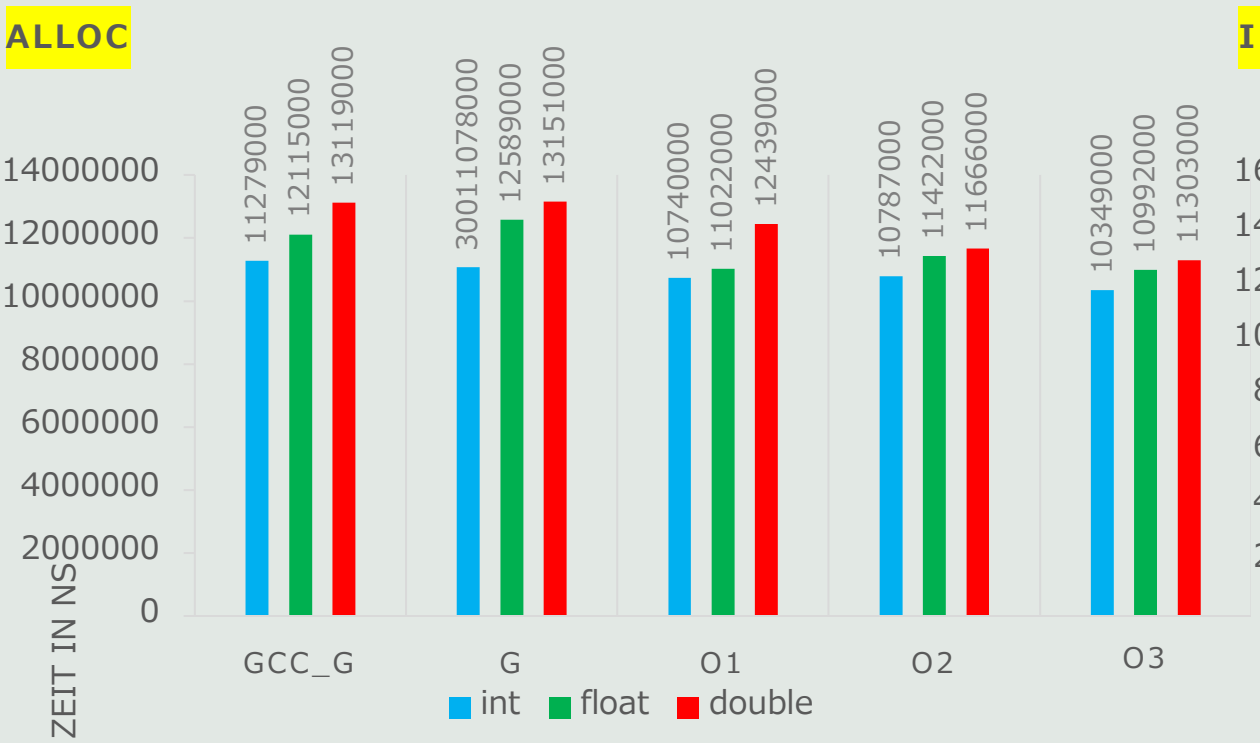
Float

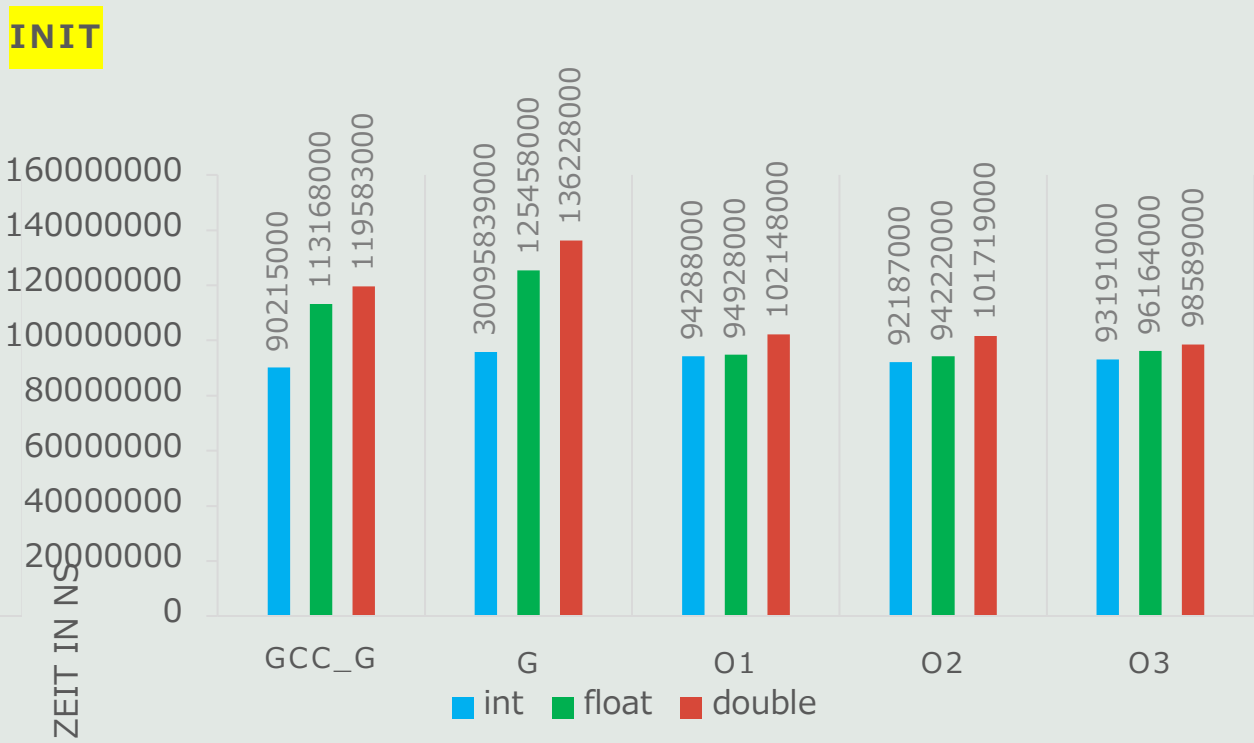
- 4byte 32bit
- Genauigkeitsverlust
- Speicherplatz mehr als int in der Regel
- Langsame Berechnungen
- Unterstützung für mathematische Funktionen
- Eingeschränkter Wertebereich

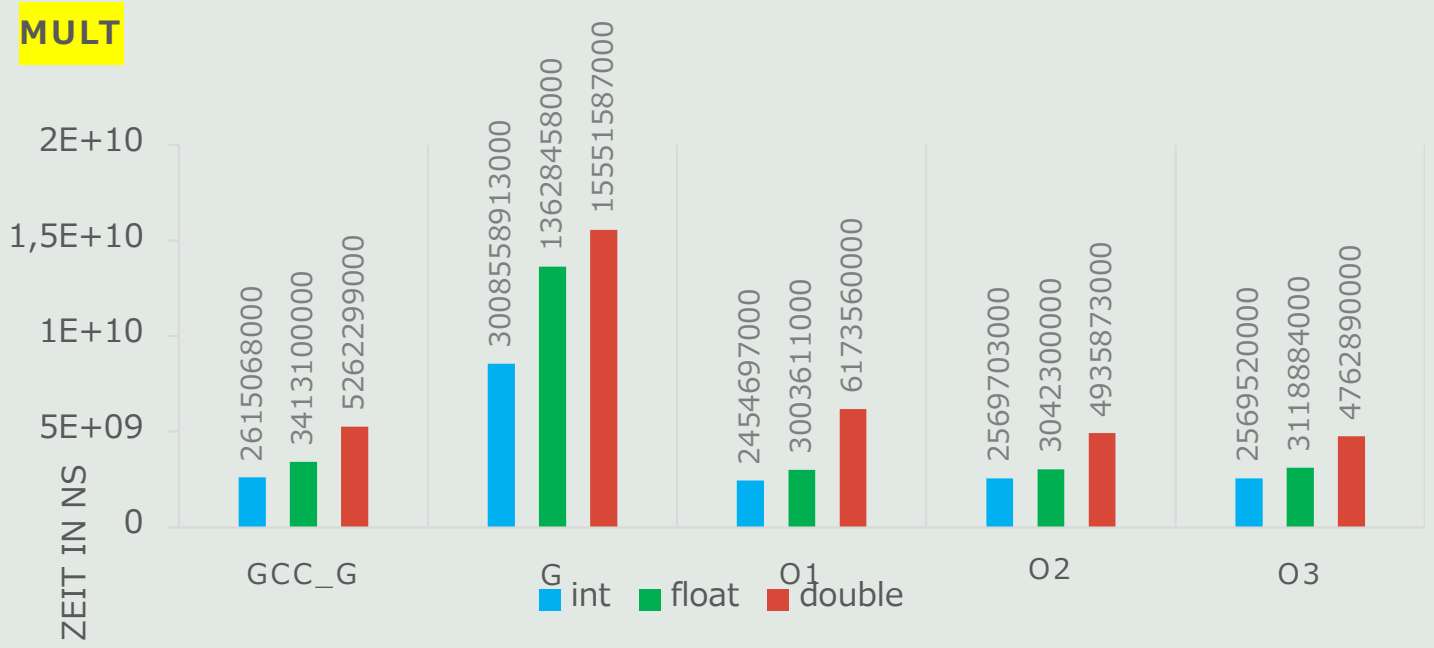
Double

- 8byte 64bit
- Hohe Genauigkeit
- Größer Speicherbedarf
- Langsame Berechnungen
- Unterstützung für mathematische Funktionen

Die beste Entscheidung bezüglich der Geschwindigkeit wäre Int.


ALLOC
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
ZEIT IN NS
11279000
12115000
13119000
30011078000
12589000
13151000
10740000
11022000
12439000
10787000
11422000
11666000
10349000
10992000
11303000
GCC_G
G
O1
O2
O3
int
float
double


INIT
160000000
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
ZEIT IN NS
90215000
113168000
119583000
30095839000
125458000
136228000
94288000
94928000
102148000
92187000
94222000
101719000
93191000
96164000
98589000
GCC_G
G
O1
O2
O3
int
float
double


MULT
2E+10
1,5E+10
1E+10
5E+09
0
ZEIT IN NS
2615068000
3413100000
5262299000
300855891300
0
13628458000
15551587000
2454697000
3003611000
6173560000
2569703000
3042300000
4935873000
2569520000
3118884000
4762890000
GCC_G
G
O1
O2
O3
int
float
double

Cache Preformance durch Speicherung

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("NS elapsed: %llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k, sum;
    clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allokation der Matrizen
    start_alloc = clock();
    int **matrix_a = (int **)malloc(N * sizeof(int *));
    int **matrix_b = (int **)malloc(N * sizeof(int *));
    int **matrix_c = (int **)malloc(N * sizeof(int *));
    for (i = 0; i < N; i++) {
        matrix_a[i] = (int *)malloc(N * sizeof(int));
        matrix_b[i] = (int *)malloc(N * sizeof(int));
        matrix_c[i] = (int *)malloc(N * sizeof(int));
    }
    end_alloc = clock();

    // Initialisierung der Matrizen
    start_init = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i][j] = rand() % 100;
            matrix_b[i][j] = rand() % 100;
            matrix_c[i][j] = 0;
        }
    }
    end_init = clock();

    // Matrixmultiplikation
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum;
        }
    }
    end_mult = clock();

    // Berechnung der Zeiten
   // printf("Allocieren\n");
    compute_timediff(start_alloc, end_alloc, 1);
    //printf("Initialisieren\n");
    compute_timediff(start_init, end_init, 1);
    //printf("Multiplizieren\n");
    compute_timediff(start_mult, end_mult, 1);
    //printf("wobei die Laenge der Matrizen = 1000\n");

    // Freigabe des Speichers
    for (i = 0; i < N; i++) {
        free(matrix_a[i]);
        free(matrix_b[i]);
        free(matrix_c[i]);
    }
    free(matrix_a);
    free(matrix_b);
    free(matrix_c);

    return 0;
}
```

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000 // Matrix size

uint64_t compute_timediff(clock_t start, clock_t end, uint8_t print) {
    uint64_t ticks = end - start;
    uint64_t nanosec = (ticks * 1000000000) / CLOCKS_PER_SEC;
    if (print)
        printf("%llu ns\n", (unsigned long long)nanosec);
    return nanosec;
}

int main() {
    int i, j, k, sum;
     clock_t start_alloc, end_alloc, start_init, end_init, start_mult, end_mult;

    // Allokation und Initialisierung der Matrizen
    start_alloc = clock();
    int *matrix_a = (int *)malloc(N * N * sizeof(int));
    int *matrix_b = (int *)malloc(N * N * sizeof(int));
    int *matrix_c = (int *)calloc(N * N, sizeof(int));
    end_alloc = clock();

    srand(time(NULL));

    // Initialisierung der Matrizen
    start_init = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            matrix_a[i * N + j] = rand() % 100;
            matrix_b[i * N + j] = rand() % 100;
        }
    }
    end_init = clock();

    // Matrixmultiplikation
    start_mult = clock();
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += matrix_a[i * N + k] * matrix_b[k * N + j];
            }
            matrix_c[i * N + j] = sum;
        }
    }
    end_mult = clock();

    // Berechnung der Zeiten
   // printf("Allocieren\n");
    compute_timediff(start_alloc, end_alloc, 1);
//printf("Initialisieren\n");
    compute_timediff(start_init, end_init, 1);
  //  printf("Multiplizieren\n");
    compute_timediff(start_mult, end_mult, 1);
    //printf("wobei die Laenge der Matrizen = 1000\n");

    // Freigabe des Speichers
    free(matrix_a);
    free(matrix_b);
    free(matrix_c);

    return 0;
}
```

Original Programm

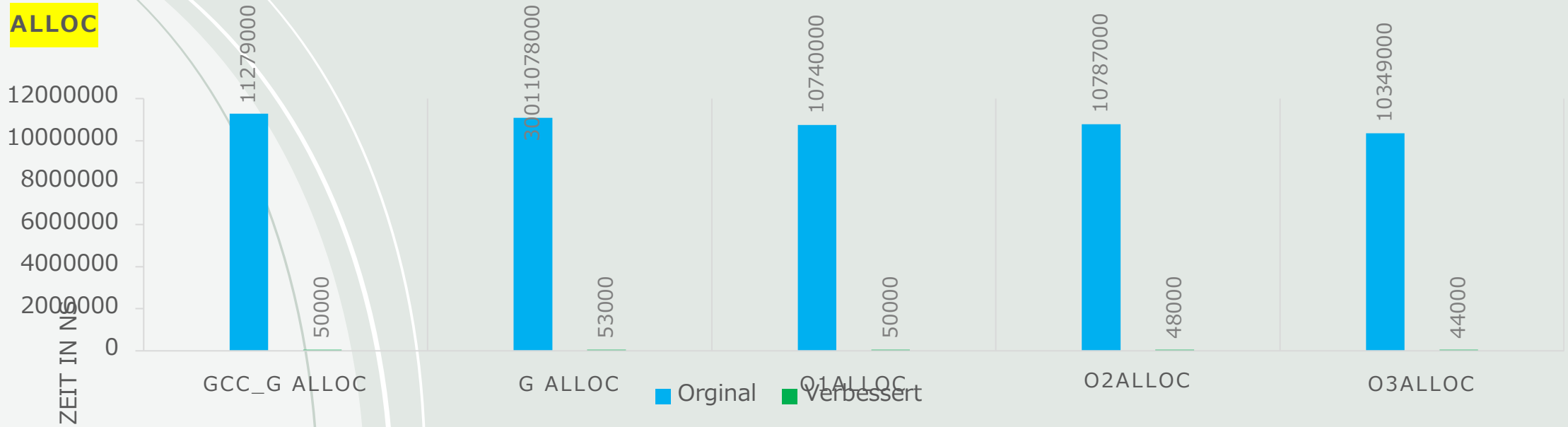
Hier werden die Matrizen als Arrays von Zeigern auf Zeilen allokiert. Das führt zu einer schlechteren räumlichen Lokalität. Der Zugriff auf die Elemente in der innersten Schleife erfolgt über verschiedene Speicherbereiche, da die Zeiger auf unterschiedliche Zeilen verweisen. Dies kann zu einer höheren Wahrscheinlichkeit von Cache-Misses führen, da die benachbarten Elemente möglicherweise nicht im Cache gehalten werden.

Verbessertees Programm

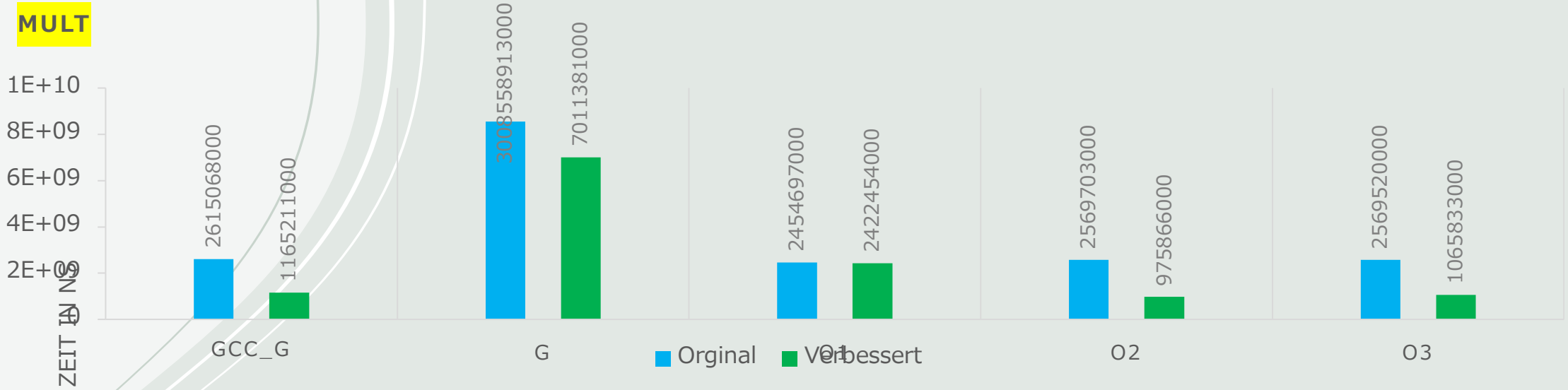
Hier erfolgt der Zugriff auf die Matrizenelemente in einem kontinuierlichen Speicherbereich. Dies kann dazu führen, dass die Elemente, die im Cache gehalten werden, zusammenhängend sind und die Wahrscheinlichkeit von Cache-Misses verringert wird. Dies liegt daran, dass beim Zugriff auf ein Element im eindimensionalen Array benachbarte Elemente ebenfalls im Cache vorhanden sein können.

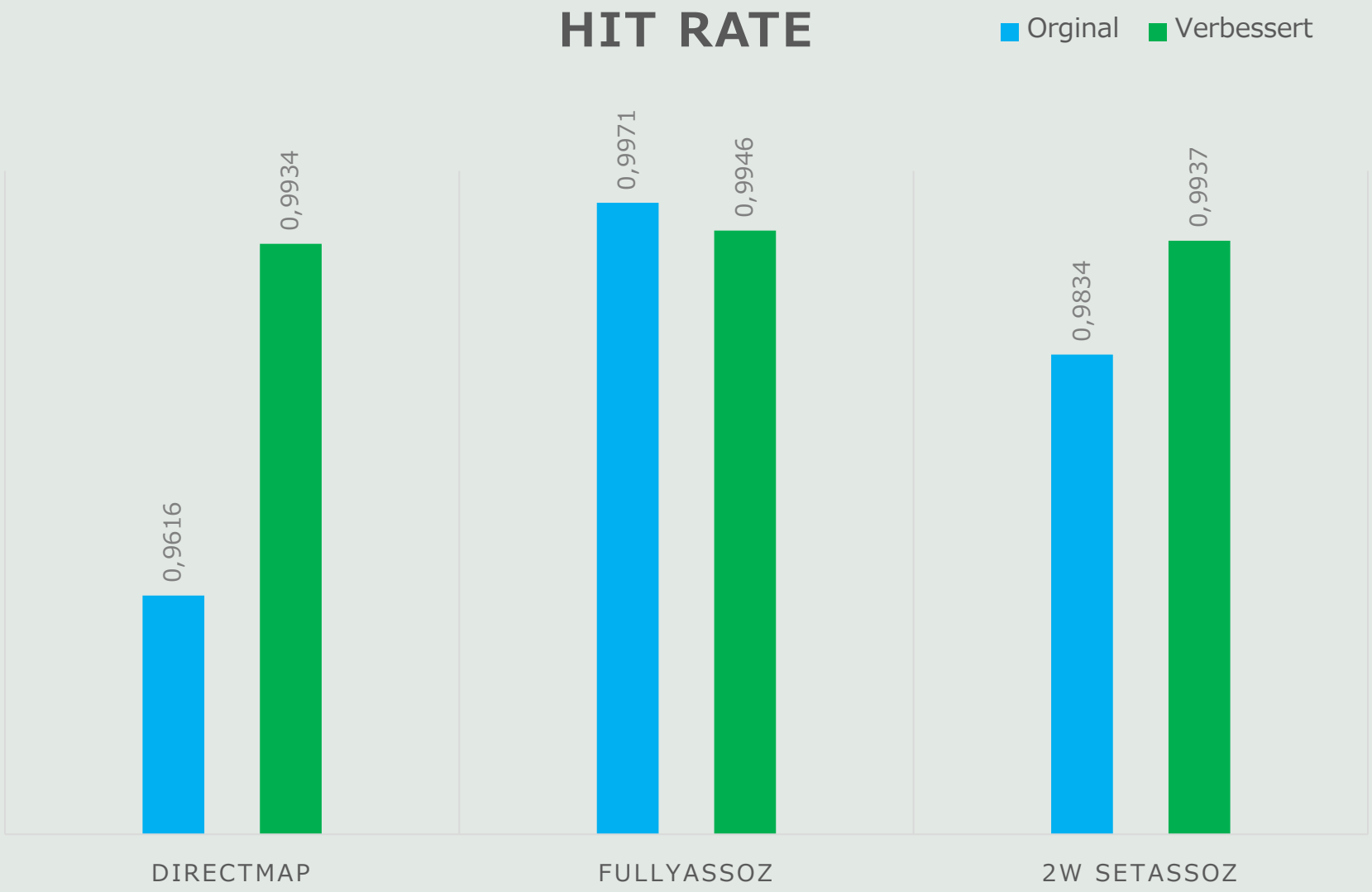
Zusammenfassend lässt sich sagen, dass das verbesserte Programm in Bezug auf die Speichernutzung und das Cache-Verhalten effizienter ist.

ALLOC


12000000
10000000
8000000
6000000
4000000
2000000
0
ZEIT IN NS
11279000
50000
30011078000
53000
10740000
50000
10787000
48000
10349000
44000
GCC_G ALLOC
G ALLOC
O1ALLOC
O2ALLOC
O3ALLOC
Orginal
Verbessert

MULT


1E+10
8E+09
6E+09
4E+09
2E+09
0
ZEIT IN NS
2615068000
1165211000
300855891300
7011381000
2454697000
2422454000
2569703000
975866000
2569520000
1065833000
GCC_G
G
O1
O2
O3
Orginal
Verbessert


HIT RATE
Orginal
Verbessert
0,9616
0,9934
0,9971
0,9946
0,9834
0,9937
DIRECTMAP
FULLYASSOZ
2W SETASSOZ

OrginalDM

Verbessert DM

Ripes

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word direct-mapped

2^N Lines: 5 | Repl. policy: LRU

2^N Ways: 0 | Wr. hit: Write-back

2^N Words/Line: 2 | Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

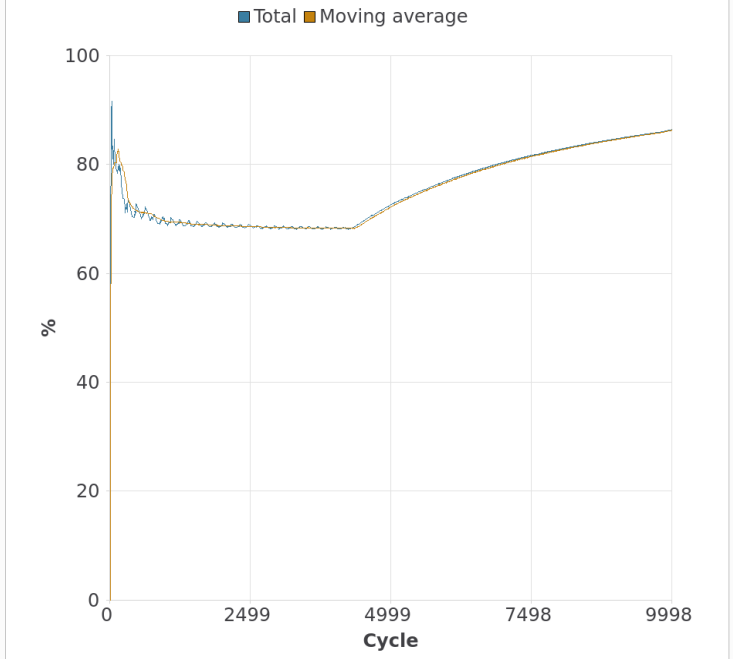
Moving avg. 50 cyc.

Statistics:

Size (bits): 4896

Hit rate: .9616 Writebacks: 175

Hits: 11438 Misses: 457



0 9998

1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00001a00
1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x00000000	0x00000000	0x000e0000	0x00000000	0x00000000
1	1	0x00000000	0x00000000	0x00012f00	0x00000000	0x25000000
1	1	0x00000000	0x00000000	0x00000000	0x31000000	0x00000000
1	1	0x00000094	0x00000000	0x00000000	0x0001288c	0x00000000
1	1	0x00000094	0x00000000	0x00010490	0x00000000	0x00000000
0	0					
0	0					
0	0					
0	0					
0	0					
0	0					
0	0					
0	0					
1	0	0x00000093	0x00010074	0x00010120	0x000100d0	0x00000000
1	1	0x00000093	0x0000000c	0x00012a2c	0x00012a94	0x00012afc
0	0					
1	0	0x00000095	0x00000000	0x00000000	0x00000000	0x00000000
1	0	0x00000093	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x003fffff	0x00000000	0x00000000	0x00012b68	0x000108d8
1	1	0x00000097	0x00000000	0x00000000	0x00000000	0x000
1	1	0x003fffff	0x00000000	0x00012890	0x00000000	0x000
1	1	0x003fffff	0x00000000	0x0000001c	0x00000000	0x000
1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00010474
1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x000100cc
1	0	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000

Addre
> Dirt

Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word direct-mapped

2^N Lines: 5 | Repl. policy: LRU

2^N Ways: 0 | Wr. hit: Write-back

2^N Words/Line: 2 | Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

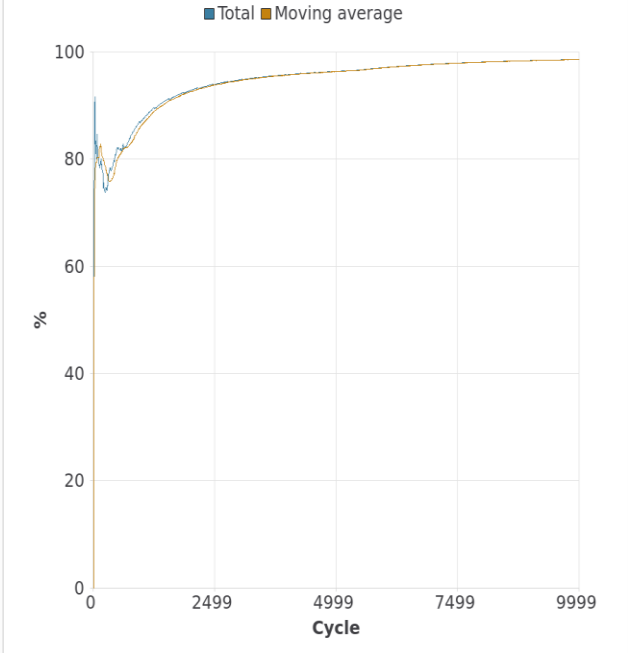
Moving avg. 50 cyc.

Statistics:

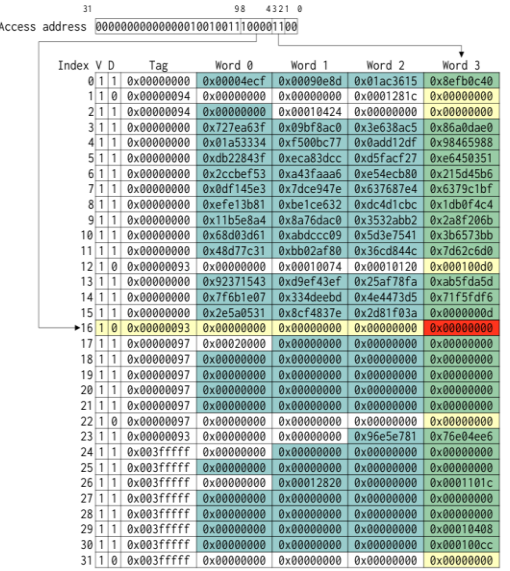
Size (bits): 4896

Hit rate: .9934 Writebacks: 10

Hits: 8977 Misses: 60



0 9999



Warum ist das verbesserte Programm besser bei Direct Mapping Strategie?

Beim Multiplizieren der Matrizen im verbesserten Programm erfolgt der Zugriff auf die Elemente in aufeinanderfolgenden Speicherbereichen. Da jeder Zeilenwert der Matrix einem bestimmten Line im Cache zugeordnet ist, werden auch die benachbarten Elemente auf demselben Line gespeichert.
Dadurch bleiben die benachbarten Elemente im Cache zusammen und können schneller abgerufen werden. Dies erhöht die Wahrscheinlichkeit, dass die benötigten Daten im Cache verfügbar sind, und somit steigt die Cache-Hit-Rate.
Darüber hinaus kann die lineare Allokation der Matrix im verbesserten Programm zu einer besseren räumlichen Lokalität beitragen, da benachbarte Elemente im Cache im selben Cache-Block gespeichert werden. Dies hilft, Cache-Miss-Situationen zu reduzieren.

Original FA

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word fully associative

2^N Lines: 0 | Repl. policy: LRU

2^N Ways: 5 | Wr. hit: Write-back

2^N Words/Line: 2 | Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

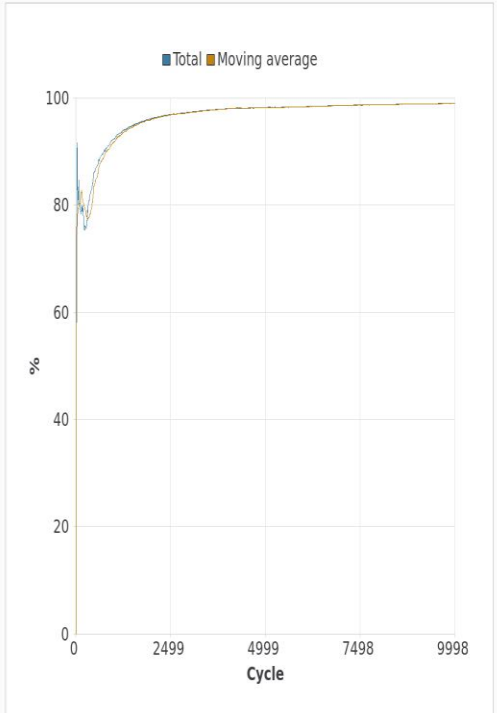
Moving avg. 50 cyc.

Statistics:

Size (bits): 5216

Hit rate:).9971 Writebacks: 3

Hits: 11860 Misses: 35



0 9998

0000000(0010011101111100

V	D	LRU	Tag	Word 0	Word 1	Word 2	Word 3
1	1	28	0x000012f8	0x00020000	0x00000000	0x00000000	0x00000000
1	1	8	0x000012f9	0x00000000	0x00000000	0x00000000	0x00000000
1	1	19	0x00000007	0x00000000	0x00000000	0x31000000	0x00000000
1	0	10	0x000012a1	0x00000000	0x00000000	0x00000000	0x00000000
1	0	0	0x00001277	0x00000000	0x00000000	0x00000000	0x00000000
1	1	25	0x000012fd	0x00000000	0x00000000	0x00000000	0x00000000
1	0	1	0x000012f7	0x00012740	0x00000000	0x00012740	0xffffffff
1	1	5	0x00001288	0x00000000	0x00000000	0x0001288c	0x00000000
1	1	6	0x00001289	0x00000000	0x00010490	0x00000000	0x00000000
1	1	11	0x07fffffe	0x00000000	0x00000000	0x00000000	0x000100cc
1	0	9	0x00001273	0x00010074	0x00010120	0x000100d0	0x00000000
1	0	31	0x07ffffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	2	0x07fffffb	0x00000000	0x0000001c	0x00000000	0x00000000
1	1	7	0x07fffffa	0x00000000	0x00012890	0x00000000	0x00011088
1	1	21	0x07fffff9	0x00000000	0x00000000	0x00000000	0x00000000
1	0	30	0x000012b9	0x00012b88	0x00012b88	0x00012b90	0x00012b90
1	0	29	0x000012ba	0x00012b98	0x00012b98	0x00012ba0	0x00012ba0
1	0	23	0x000012b7	0x00012b68	0x00012b68	0x00012b70	0x00012b70
1	0	22	0x000012b6	0x00000000	0x00000000	0x00000000	0x00000000
1	1	24	0x07fffff8	0x00000000	0x00000000	0x00012b68	0x000108d8
1	1	27	0x07fffff7	0x00000000	0x00000000	0x00000000	0x00010edc
1	1	26	0x00001274	0x0000000c	0x00012a2c	0x00012a94	0x00012afc
1	1	3	0x07fffffc	0x00000000	0x00000000	0x00000000	0x00000000
1	1	4	0x07fffffd	0x00000000	0x00000000	0x00000000	0x00010474
1	1	13	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	1	12	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000
1	1	16	0x0000127e	0x00000000	0x00000000	0x96e5e781	0x76e04ee6
1	1	20	0x00000002	0x00000000	0x00000000	0x00000000	0x00001a00
1	1	15	0x00000003	0x00000000	0x00000000	0x00000000	0x00000000
1	1	14	0x00000004	0x00000000	0x00000000	0x00000000	0x00000000
1	1	17	0x00000005	0x00000000	0x000e0000	0x00000000	0x00000000
1	1	18	0x00000006	0x00000000	0x00012f00	0x00000000	0x25000000

Verbessert FA

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word fully associative

2^N Lines: 0 | Repl. policy: LRU

2^N Ways: 5 | Wr. hit: Write-back

2^N Words/Line: 2 | Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

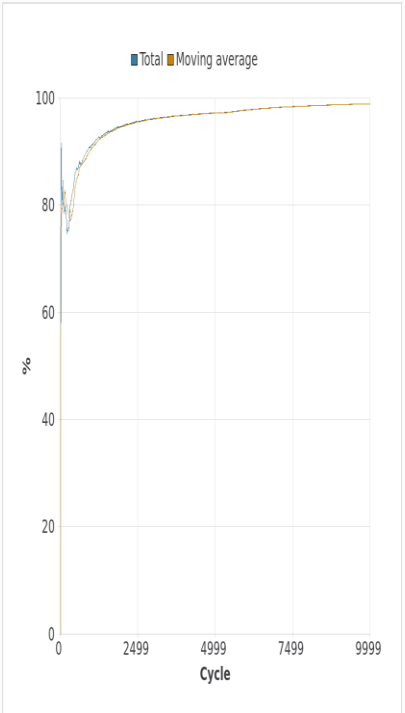
Moving avg. 50 cyc.

Statistics:

Size (bits): 5216

Hit rate:).9946 Writebacks: 12

Hits: 8988 Misses: 49



0 9999

Access address 00000000 0011100001100

Index	V	D	LRU	Tag	Word 0	Word 1	Word 2	Word 3
	1	1	11	0x07fffffe	0x00000000	0x00000000	0x00000000	0x000100cc
	1	0	5	0x00001281	0x00000000	0x00000000	0x0001281c	0x00000000
	1	1	24	0x00000006	0x2ccbef53	0xa43faaa6	0xe54ecb80	0x215d45b6
	1	1	16	0x00000007	0x0df145e3	0x7dce947e	0x637687e4	0x6379c1bf
	1	1	23	0x00000008	0xefe13b81	0xbe1ce632	0xdc4d1cbc	0x1db0f4c4
	1	0	9	0x0000126c	0x00000000	0x00010074	0x00010120	0x000100d0
	1	0	1	0x000012f0	0x000126d0	0x00000000	0x000126d0	0xffffffff
	1	1	15	0x00000009	0x11b5e8a4	0x8a76dac0	0x3532abb2	0x2a8f206b
	1	1	22	0x0000000a	0x68d03d61	0xabdccc09	0x5d3e7541	0x3b6573bb
	1	1	13	0x0000000d	0x92371543	0xd9ef43ef	0x25af78fa	0xab5fda5d
	1	1	14	0x0000000b	0x48d77c31	0xbb02af80	0x36cd844c	0x7d62c6d0
	1	1	21	0x0000000c	0xdd2f22d3	0xc4e3db17	0x5cfda276	0xb8f6b4ce
	1	1	2	0x07fffffb	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	7	0x07fffffa	0x00000000	0x00012820	0x00000000	0x0001101c
	1	1	30	0x07fffff9	0x00000000	0x00000000	0x00000000	0x00000000
0	1	1	17	0x0000000e	0x7f6b1e07	0x334deebd	0x4e4473d5	0x71f5fdf6
	1	1	12	0x0000000f	0x2e5a0531	0x8cf4837e	0x2d81f03a	0x0000000d
	1	0	0	0x00001270	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	31	0x000012af	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	29	0x07fffff8	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	8	0x000012f2	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	10	0x0000129a	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	3	0x07fffffc	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	6	0x00001282	0x00000000	0x00010424	0x00000000	0x00000000
	1	1	4	0x07fffffd	0x00000000	0x00000000	0x00000000	0x00010408
	1	1	28	0x00001277	0x00000000	0x00000000	0x96e5e781	0x76e04ee6
	1	1	27	0x00000000	0x00004ecf	0x00090e8d	0x01ac3615	0x8efb0c40
	1	1	20	0x00000001	0x8801ccd3	0x550282db	0x5c3129ea	0x2b540a96
	1	1	26	0x00000002	0x0014ad1a	0x42c4ed9f	0x07d5695b	0x701c65ec
	1	1	19	0x00000003	0x727ea63f	0x09bf8ac0	0x3e638ac5	0x86a0dae0
	1	1	25	0x00000004	0x01a53334	0xf500bc77	0x0add12df	0x98465988
	1	1	18	0x00000005	0xdb22843f	0xeca83dcc	0xd5facf27	0xe6450351

Warum ist das verbesserte Programm schlechter bei fully assoziative- Strategie?

In einem voll assoziativen Cache kann jede Adresse im Speicher an beliebiger Stelle im Cache abgelegt werden. Dies bedeutet, dass keine feste Zuordnung der Speicheradressen im Cache erfolgt. Dadurch kann es zu Kollisionen kommen, wenn aufeinanderfolgende Adressen im Speicher an verschiedenen Positionen im Cache auftreten, was zu einer niedrigeren Hit-Rate führt.

Im verbesserten Programm können aufgrund des linearen Speicherlayouts und des Zugriffsmusters eindeutige Adressen benachbart im Speicher auftreten. Dies erhöht die Wahrscheinlichkeit von Kollisionen in einem voll assoziativen Cache und führt zu einer niedrigeren Hit-Rate.

Also das verbesserte Programm ist nicht für diese Cache-Variante geeignet.

Orginal2Way

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word 2-way set associative

2^N Lines: 4 — Repl. policy: LRU

2^N Ways: 1 — Wr. hit: Write-back

2^N Words/Line: 2 — Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

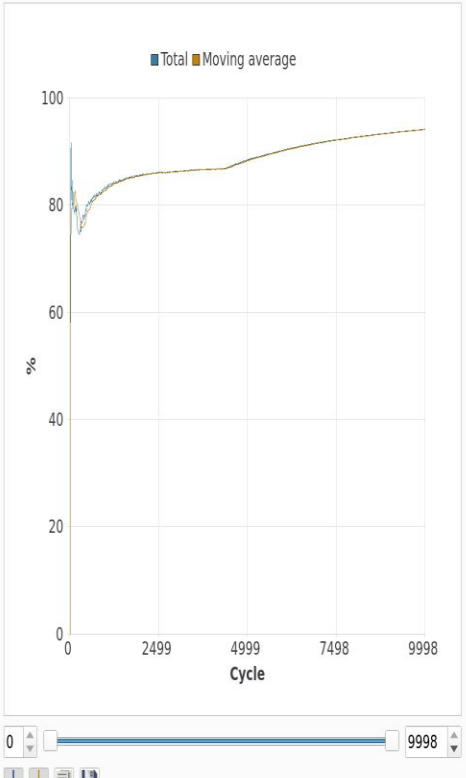
Moving avg. 50 cyc.

Statistics:

Size (bits): 4960

Hit rate: .9834 Writebacks: 87

Hits: 11697 Misses: 198



0 9998

31 87 4321 0

ss 00000000000000010010011101111100

dex	V	D	LRU	Tag	Word 0	Word 1	Word 2	Word 3
0	1	1	0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
	0	0	1					
1	1	1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x0000012a	0x00000000	0x00000000	0x00000000	0x00000000
2	1	1	0	0x00000000	0x00000000	0x00000000	0x00000000	0x00001a00
	0	0	1					
3	1	0	0	0x00000127	0x00010074	0x00010120	0x000100d0	0x00000000
	1	1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
4	1	1	1	0x00000127	0x0000000c	0x00012a2c	0x00012a94	0x00012afc
	1	1	0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
5	1	1	0	0x00000000	0x00000000	0x000e0000	0x00000000	0x00000000
	0	0	1					
6	1	0	1	0x0000012b	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	0	0x00000000	0x00000000	0x00012f00	0x00000000	0x25000000
→ 7	1	0	0	0x00000127	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	1	0x0000012f	0x00012740	0x00000000	0x00012740	0xffffffff
8	1	0	0	0x00000128	0x00000000	0x00000000	0x0001288c	0x00000000
	1	1	1	0x007fffff	0x00000000	0x00000000	0x00012b68	0x000108d8
9	1	1	0	0x00000128	0x00000000	0x00010490	0x00000000	0x00000000
	1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0	1	0x0000012b	0x00012b98	0x00012b98	0x00012ba0	0x00012ba0
	1	1	0	0x007fffff	0x00000000	0x00012890	0x00000000	0x00011088
11	1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	0	0x007fffff	0x00000000	0x0000001c	0x00000000	0x00000000
12	1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x00000000
13	1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
	1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x00010474
14	1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x000100cc
	1	1	1	0x00000127	0x00000000	0x00000000	0x96e5e781	0x76e04ee6
15	1	0	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x00000000
	0	0	1					

Verbessert 2Way

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word 2-way set associative

2^N Lines: 4 — Repl. policy: LRU

2^N Ways: 1 — Wr. hit: Write-back

2^N Words/Line: 2 — Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

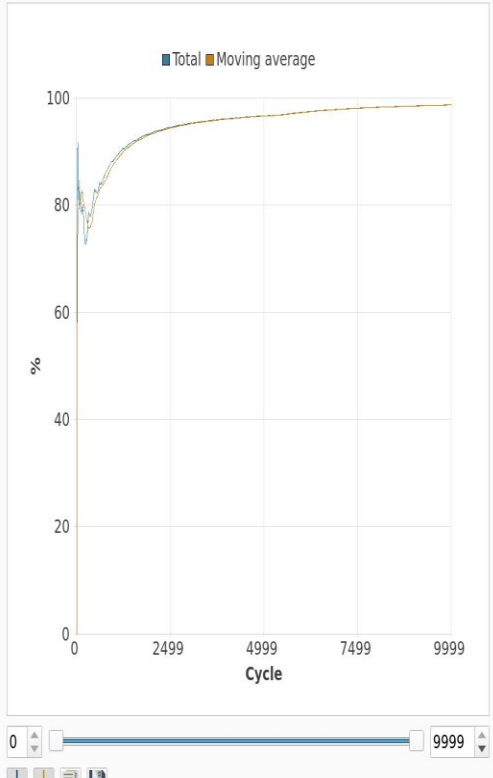
Moving avg. 50 cyc.

Statistics:

Size (bits): 4960

Hit rate: .9937 Writebacks: 12

Hits: 8980 Misses: 57



0 9999

0 1100001100

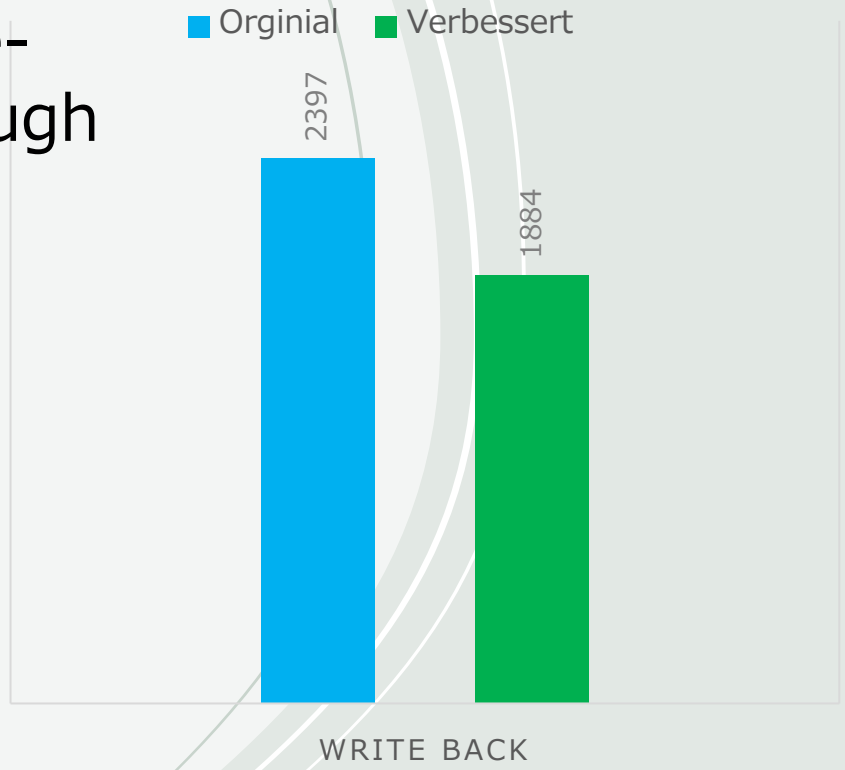
V	D	LRU	Tag	Word 0	Word 1	Word 2	Word 3
1	0	0	0x00000127	0x00000000	0x00000000	0x00000000	0x00000000
1	0	1	0x0000012f	0x000126d0	0x00000000	0x000126d0	0xffffffff
1	1	1	0x00000000	0x8801ccd3	0x550282db	0x5c3129ea	0x2b540a96
1	0	0	0x00000128	0x00000000	0x00000000	0x0001281c	0x00000000
1	1	0	0x00000128	0x00000000	0x00010424	0x00000000	0x00000000
1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x00000000	0x727ea63f	0x09bf8ac0	0x3e638ac5	0x86a0dae0
1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x00000000	0x01a53334	0xf500bc77	0x0add12df	0x98465988
1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x00000000	0xdb22843f	0xeca83dcc	0xd5facf27	0xe6450351
1	1	1	0x0000012f	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x00000000	0x2ccbef53	0xa43faaa6	0xe54ecb80	0x215d45b6
1	1	0	0x00000000	0x0df145e3	0x7dce947e	0x637687e4	0x6379c1bf
1	1	1	0x00000127	0x00000000	0x00000000	0x96e5e781	0x76e04ee6
1	1	1	0x007fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x00000000	0xefe13b81	0xbe1ce632	0xdc4d1cbc	0x1db0f4c4
1	1	1	0x007fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x00000000	0x11b5e8a4	0x8a76dac0	0x3532abb2	0x2a8f206b
1	0	1	0x00000129	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x007fffff	0x00000000	0x00012820	0x00000000	0x0001101c
1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	1	0x00000000	0x48d77c31	0xbb02af80	0x36cd844c	0x7d62c6d0
1	0	1	0x00000126	0x00000000	0x00010074	0x00010120	0x000100d0
1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	1	0x00000000	0x92371543	0xd9ef43ef	0x25af78fa	0xab5fda5d
1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x00010408
1	1	0	0x007fffff	0x00000000	0x00000000	0x00000000	0x000100cc
1	1	1	0x00000000	0x7f6b1e07	0x334deebd	0x4e4473d5	0x71f5fdf6
1	1	0	0x00000000	0x2e5a0531	0x8cf4837e	0x2d81f03a	0x0000000d
1	0	1	0x0000012a	0x00000000	0x00000000	0x00000000	0x00000000

Warum ist das verbesserte Programm besser bei 2way set assoziative?

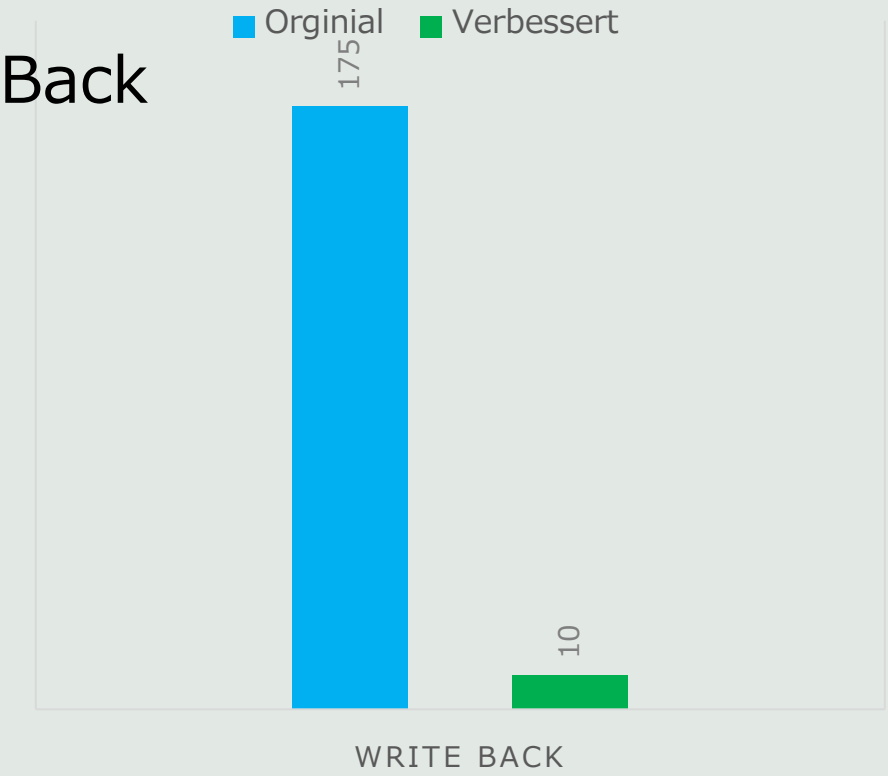
In Fall des verbesserten Programms, wo die Matrizen kontinuierlich abgelegt werden, gibt es eine höhere Wahrscheinlichkeit, dass kürzlich verwendete Adressen im Cache angesammelt sind. Daher besteht eine höhere Wahrscheinlichkeit für eine Verbesserung der Cache-Hit-Rate, da mehr Cache-Hits auftreten und weniger Cache-misses.
Im Gegensatz dazu kann bei dem orginalen Programm, wo die Matrizen verstreut abgelegt werden, die Anzahl der Cache-misses zunehmen, da die nicht zusammenhängenden Daten im Hauptspeicher möglicherweise nicht gut zum Cache-Layout passen. Somit besteht eine höhere Wahrscheinlichkeit dafür, dass die benötigten Daten nicht im Cache vorhanden sind, was zu einer Verringerung der Cache-Hit-Rate führt.
Im Allgemeinen bietet das verbesserte Programm eine bessere Anordnung der Daten im Speicher, die besser mit dem 2-Wege-assoziativen Cache übereinstimmt. Daher wird erwartet, dass die Cache-Hit-Rate im verbesserten Programm im Vergleich zu dem orginalem besser ist.

Write-Back und Write-Through

Write-Through



Write-Back



Die Vorteile der Write-Back-Strategie

- Reduzierte Hauptspeicherzugriffe
- Bessere Ausnutzung des Cache-Speichers
- Geringere Latenzzeit für Schreiboperationen

OrginalDM

Ripes

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word direct-mapped

2^N Lines: 5 — Repl. policy: LRU

2^N Ways: 0 — Wr. hit: Write-back

2^N Words/Line: 2 — Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

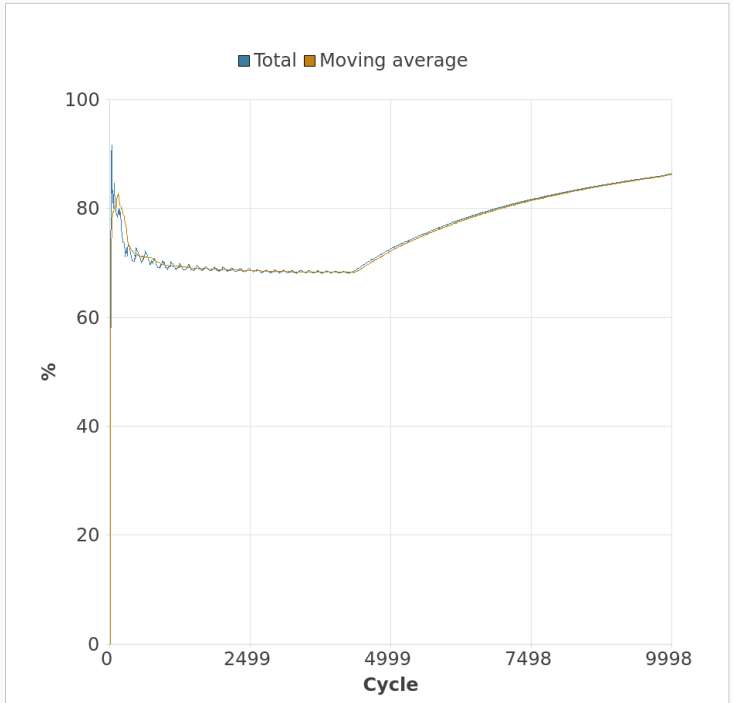
Moving avg. 50 cyc.

Statistics:

Size (bits): 4896

Hit rate:).9616 Writebacks: 175

Hits: 11438 Misses: 457



0 9998

		0000	0x00000000	0x00000000	0x00000000	0x00000000
		0095	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00001a00
1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x00000000	0x00000000	0x000e0000	0x00000000	0x00000000
1	1	0x00000000	0x00000000	0x00012f00	0x00000000	0x25000000
1	1	0x00000000	0x00000000	0x00000000	0x31000000	0x00000000
1	1	0x00000094	0x00000000	0x00000000	0x0001288c	0x00000000
1	1	0x00000094	0x00000000	0x00010490	0x00000000	0x00000000
0	0					
0	0					
0	0					
0	0					
0	0					
0	0					
0	0					
0	0					
1	0	0x00000093	0x00010074	0x00010120	0x000100d0	0x00000000
1	1	0x00000093	0x0000000c	0x00012a2c	0x00012a94	0x00012afc
0	0					
1	0	0x00000095	0x00000000	0x00000000	0x00000000	0x00000000
1	0	0x00000093	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x003fffff	0x00000000	0x00000000	0x00012b68	0x000108d8
1	1	0x00000097	0x00000000	0x00000000	0x00000000	0x000
1	1	0x003fffff	0x00000000	0x00012890	0x00000000	0x000
1	1	0x003fffff	0x00000000	0x0000001c	0x00000000	0x000
1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00010474
1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x000100cc
1	0	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000

Addre
> Dirt

Verbessert DM

Ripes

Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset: 32-entry 4-word direct-mapped

2^N Lines: 5 — Repl. policy: LRU

2^N Ways: 0 — Wr. hit: Write-back

2^N Words/Line: 2 — Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

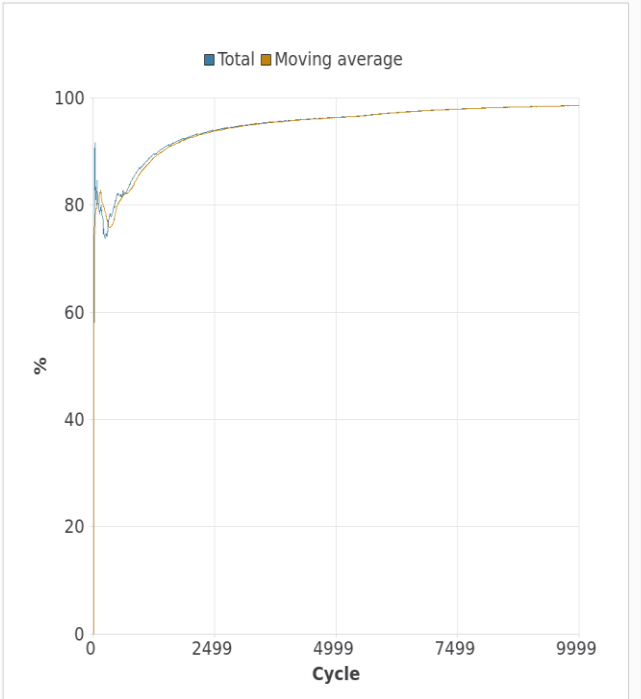
Moving avg. 50 cyc.

Statistics:

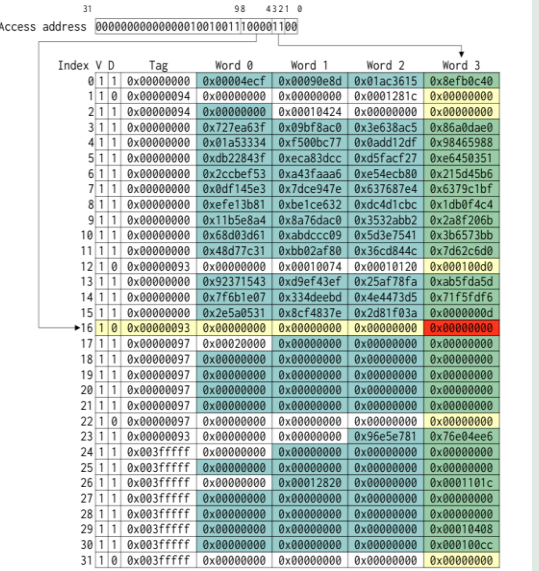
Size (bits): 4896

Hit rate:).9934 Writebacks: 10

Hits: 8977 Misses: 60



0 9999



Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	1	1	0x00000000	0x00004ecf	0x00090e8d	0x01ac3615	0x8efb0c40
1	1	0	0x00000094	0x00000000	0x00000000	0x0001281c	0x00000000
2	1	1	0x00000094	0x00000000	0x00010424	0x00000000	0x00000000
3	1	1	0x00000000	0x727ea63f	0x09bf8ac0	0x3e638ac5	0x86a0dae0
4	1	1	0x00000000	0x01a53334	0xf500bc77	0x0add12df	0x98465988
5	1	1	0x00000000	0xdb22843f	0xeca83dcc	0xd5facf27	0xe6450351
6	1	1	0x00000000	0x2ccbef53	0xa43faaa6	0xe54ecb80	0x215d45b6
7	1	1	0x00000000	0x0df145e3	0x7dce947e	0x637687e4	0x6379c1bf
8	1	1	0x00000000	0xefe13b81	0xbe1ce632	0xdc4d1cbc	0x1db0f4c4
9	1	1	0x00000000	0x11b5e8a4	0x8a76dac0	0x3532abb2	0x2a8f206b
10	1	1	0x00000000	0x68d03d61	0xabdccc09	0x5d3e7541	0x3b6573bb
11	1	1	0x00000000	0x48d77c31	0xbb02af80	0x36cd844c	0x7d62c6d0
12	1	0	0x00000093	0x00000000	0x00010074	0x00010120	0x000100d0
13	1	1	0x00000000	0x92371543	0xd9ef43ef	0x25af78fa	0xab5fda5d
14	1	1	0x00000000	0x7f6b1e07	0x334deebd	0x4e4473d5	0x71f5fdf6
15	1	1	0x00000000	0x2e5a0531	0x8cf4837e	0x2d81f03a	0x0000000d
16	1	0	0x00000093	0x00000000	0x00000000	0x00000000	0x00000000
17	1	1	0x00000097	0x00020000	0x00000000	0x00000000	0x00000000
18	1	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
19	1	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
20	1	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
21	1	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
23	1	1	0x00000093	0x00000000	0x00000000	0x96e5e781	0x76e04ee6
24	1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
25	1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
26	1	1	0x003fffff	0x00000000	0x00012820	0x00000000	0x0001101c
27	1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
29	1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00010408
30	1	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x000100cc
31	1	0	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000

Ripes

Original WT

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset:

2^N Lines: 5 — Repl. policy: LRU

2^N Ways: 0 — Wr. hit: Write-through

2^N Words/Line: 2 — Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

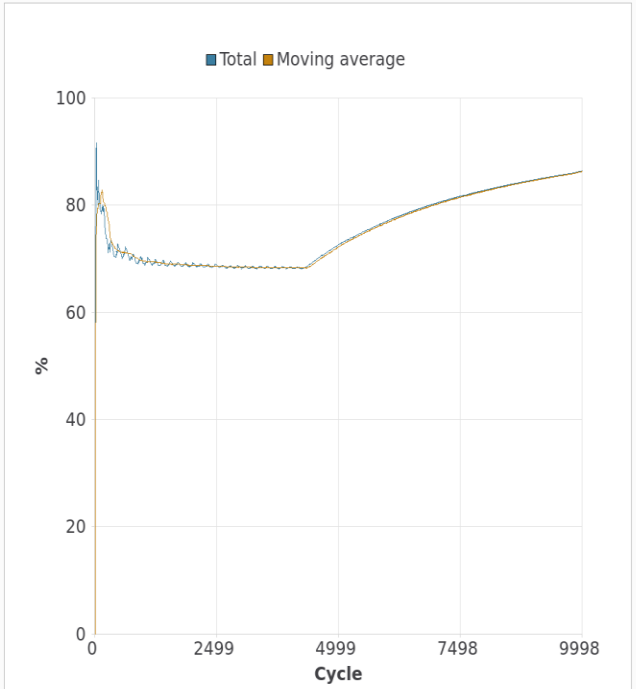
Moving avg. 50 cyc.

Statistics:

Size (bits): 4864

Hit rate: .9616 Writebacks: 2397

Hits: 11438 Misses: 457



0 9998

98 4321 0

00000000000001001001110111 1100

	Tag	Word 0	Word 1	Word 2	Word 3
1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	0x00000095	0x00000000	0x00000000	0x00000000	0x00000000
1	0x00000000	0x00000000	0x00000000	0x00000000	0x00001a00
1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	0x00000000	0x00000000	0x000e0000	0x00000000	0x00000000
1	0x00000000	0x00000000	0x00012f00	0x00000000	0x25000000
1	0x00000000	0x00000000	0x00000000	0x31000000	0x00000000
1	0x00000094	0x00000000	0x00000000	0x0001288c	0x00000000
1	0x00000094	0x00000000	0x00010490	0x00000000	0x00000000
0					
0					
0					
0					
0					
0					
0					
0					
0					
1	0x00000093	0x00010074	0x00010120	0x000100d0	0x00000000
1	0x00000093	0x0000000c	0x00012a2c	0x00012a94	0x00012afc
0					
1	0x00000095	0x00000000	0x00000000	0x00000000	0x00000000
1	0x00000093	0x00000000	0x00000000	0x00000000	0x00000000
1	0x003fffff	0x00000000	0x00000000	0x00012b68	0x000108d8
1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
1	0x003fffff	0x00000000	0x00012890	0x00000000	0x00011088
1	0x003fffff	0x00000000	0x0000001c	0x00000000	0x00000000
1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00010474
1	0x003fffff	0x00000000	0x00000000	0x00000000	0x000100cc
1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000

Verbessert WT

File Edit View Help

1 ms

L1 Data Cache | L1 Instr. Cache

Cache configuration:

Preset:

2^N Lines: 5 — Repl. policy: LRU

2^N Ways: 0 — Wr. hit: Write-through

2^N Words/Line: 2 — Wr. miss: Write allocate

Plot configuration:

Numerator: Hits

Denominator: Access count

Ratio

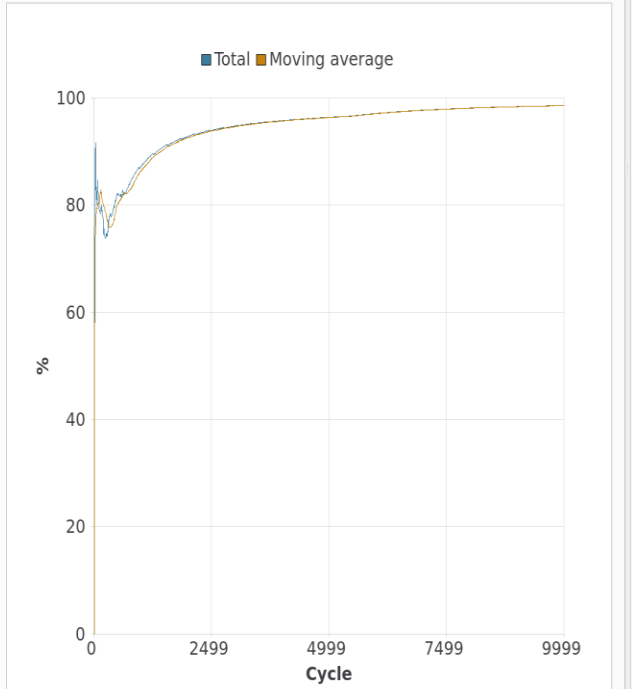
Moving avg. 50 cyc.

Statistics:

Size (bits): 4864

Hit rate: .9934 Writebacks: 1884

Hits: 8977 Misses: 60



0 9999

31 98 4321 0

Access address 00000000000000010010011100001100

Index	V	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0x00000000	0x00004ecf	0x00090e8d	0x01ac3615	0x8efb0c40
1	1	0x00000094	0x00000000	0x00000000	0x0001281c	0x00000000
2	1	0x00000094	0x00000000	0x00010424	0x00000000	0x00000000
3	1	0x00000000	0x727ea63f	0x09bf8ac0	0x3e638ac5	0x86a0dae0
4	1	0x00000000	0x01a53334	0xf500bc77	0x0add12df	0x98465988
5	1	0x00000000	0xdb22843f	0xeca83dcc	0xd5facf27	0xe6450351
6	1	0x00000000	0x2ccbef53	0xa43faaa6	0xe54ecb80	0x215d45b6
7	1	0x00000000	0x0df145e3	0x7dce947e	0x637687e4	0x6379c1bf
8	1	0x00000000	0xefe13b81	0xbe1ce632	0xdc4d1cbc	0x1db0f4c4
9	1	0x00000000	0x11b5e8a4	0x8a76dac0	0x3532abb2	0x2a8f206b
10	1	0x00000000	0x68d03d61	0xabdccc09	0x5d3e7541	0x3b6573bb
11	1	0x00000000	0x48d77c31	0xbb02af80	0x36cd844c	0x7d62c6d0
12	1	0x00000093	0x00000000	0x00010074	0x00010120	0x000100d0
13	1	0x00000000	0x92371543	0xd9ef43ef	0x25af78fa	0xab5fda5d
14	1	0x00000000	0x7f6b1e07	0x334deebd	0x4e4473d5	0x71f5fdf6
15	1	0x00000000	0x2e5a0531	0x8cf4837e	0x2d81f03a	0x0000000d
16	1	0x00000093	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0x00000097	0x00020000	0x00000000	0x00000000	0x00000000
18	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0x00000097	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0x00000093	0x00000000	0x00000000	0x96e5e781	0x76e04ee6
24	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0x003fffff	0x00000000	0x00012820	0x00000000	0x0001101c
27	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000
29	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00010408
30	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x000100cc
31	1	0x003fffff	0x00000000	0x00000000	0x00000000	0x00000000