Rechnerarchitekturen
Laborübung
Beispiel 3

Alexander Ölzant
9301547

SS 1998

1 Aufgabe 1

C-Programm zur Erstellung des Traces

```
int i, ai, bi, ci, di, ei;
int a[1024], b[1024], c[1024], d[1024], e[1024];
void compute()
{

        ai=0;
        bi=0;
        ci=0;
        di=0;
        ei=0;
         printf("; compute:\n");
        printf("2 1000         ;move $9,0\n");
        printf("2 1004         ;la $8,a\n");
        printf("2 1008         ;la $7,b\n");
        printf("2 100c         ;la $6,c\n");
        printf("2 1010         ;la $5,d\n");
        printf("2 1014         ;la $4,e\n");

        for (i=0; i<1000; i++) {

                 printf("; $L11:\n");
                printf("2 1018         ;lw $3,0($6)\n");
                printf("0 %x         \n",0x4000+ci*4);
                printf("2 101c         ;addu $6, $6, (4*n)=28\n");
                printf("2 1020         ;lw $2,0($7)\n");
                printf("0 %x \n",0x3000+bi*4);
                printf("2 1024         ;addu $7, $7, 4\n");
                printf("2 1028         ;addu $2, $2, $3\n");
                printf("2 102c         ;sw $2,0($8)\n");
                printf("1 %x \n",0x2000+ai*4);
                printf("2 1030         ;addu $9, $9, 1\n");

                a[ai++] = c[ci] + b[bi++];
                ci += 7 /* (9301547 % 10)*4*/; /* 28 */

                printf("2 1034         ;lw $3,0($4)\n");
                printf("0 %x \n",0x6000+ei*4);
                printf("2 1038         ;addu $4, $4, 4\n");
                printf("2 103c         ;lw $2,0($5)\n");
                printf("0 %x \n",0x5000+di*4);
```

```
                printf("2 1040          ;addu $5, $5, 4\n");
                printf("2 1044          ;addu $2, $2, $3\n");
                printf("2 1048          ;sw $2,0($6)\n");
                printf("1 %x \n",0x4000+ci*4);
                printf("2 104c          ;addu $8, $8, 4\n");

                c[ci] = d[di++] + e[ei++];
                ci -= 6 /*((9301547 % 10) - 1) * (-4)*/; /* -24 */

                printf("2 104c          ;slt $2, $9, 1000\n");
                printf("2 1050          ;addu $6, $6, (-4*(n-1))=-24\n");
                printf("2 1054          ;bne $2, $0, $L11\n");


        };
        printf("2 1058          ;j $31\n");
        printf(".end        compute\n");
};

main()
{
        compute();
}
```

2 Aufgabe 2

Hier ein Auszug aus dem Tracefile:

```
; compute:
2 1000  ;move $9,0
2 1004  ;la $8,a
2 1008  ;la $7,b
2 100c  ;la $6,c
2 1010  ;la $5,d
2 1014  ;la $4,e
; $L11:
2 1018  ;lw $3,0($6)
0 4000
2 101c  ;addu $6, $6, (4*n)=28
2 1020  ;lw $2,0($7)
0 3000
2 1024  ;addu $7, $7, 4
2 1028  ;addu $2, $2, $3
```

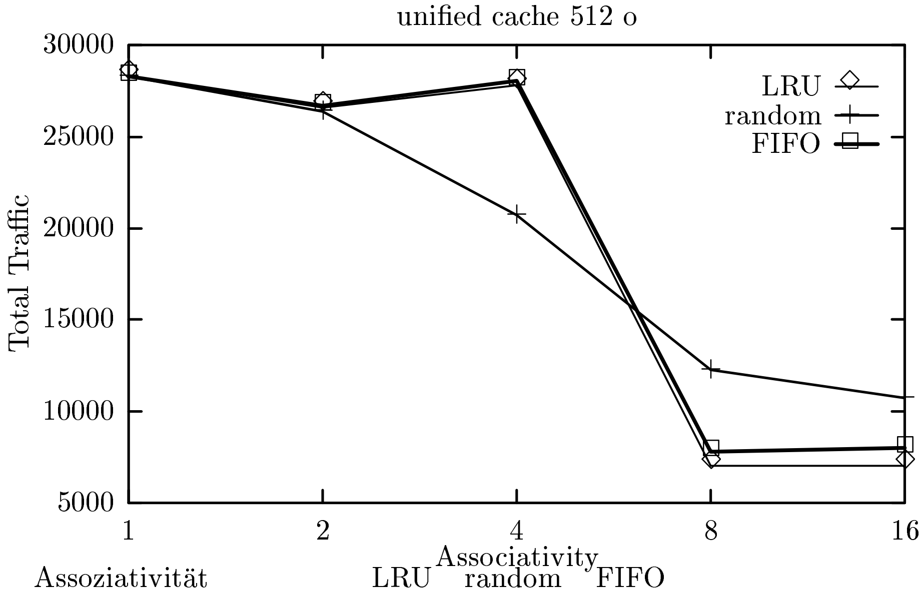
```
2 102c  ;sw $2,0($8)
1 2000
2 1030  ;addu $9, $9, 1
2 1034  ;lw $3,0($4)
0 6000
2 1038  ;addu $4, $4, 4
2 103c  ;lw $2,0($5)
0 5000
2 1040  ;addu $5, $5, 4
2 1044  ;addu $2, $2, $3
2 1048  ;sw $2,0($6)
1 401c
2 104c  ;addu $8, $8, 4
2 104c  ;slt $2, $9, 1000
2 1050  ;addu $6, $6, (-4*(n-1))=-24
2 1054  ;bne $2, $0, $L11
```

...

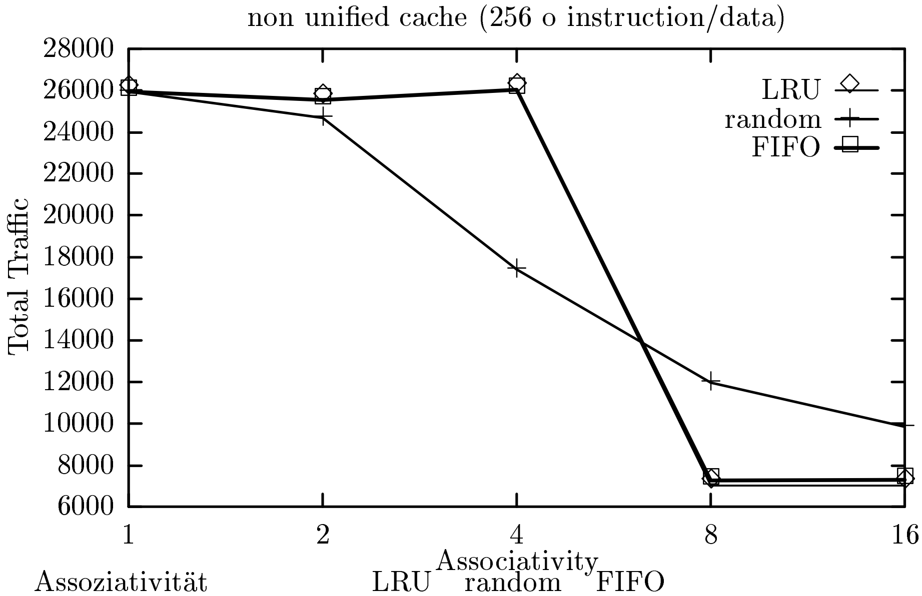
Ad nauseam ac infinitum[1].

3 Aufgabe 3

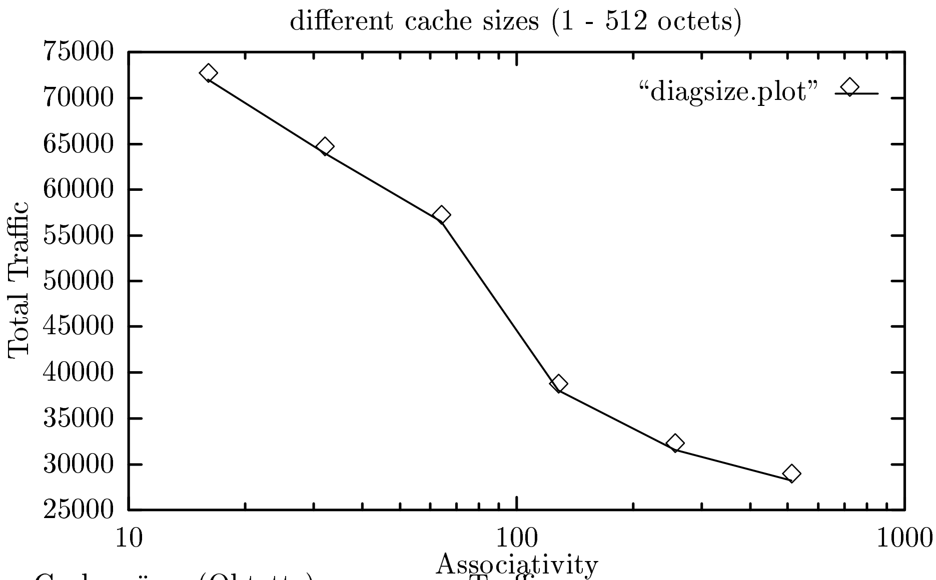
[1] 473143 Oktette, abrufbar unter http://prawda.oeh.net/~aoe/trash/prod.din



Assoziativität	LRU	random	FIFO
1	28292	28292	28292
2	26576	26384	26704
4	27796	20732	28044
8	7040	12280	7812
16	7040	10728	8024



Assoziativität	LRU	random	FIFO
1	25956	25956	25956
2	25552	24696	25552
4	26048	17404	26048
8	7040	11952	7288
16	7040	9848	7324



Cachegrösse (Oktette)	Traffic
16	72016
32	64012
64	56492
128	38056
256	31620
512	28292

4 Aufgabe 4

Bei kleiner Assoziativität ist teilweise random besser, ab 8-fach assoziativem cache schneidet LRU aber besser ab.

Unified caches sind oft die bessere Wahl (bessere Verteilung bei unterschiedlicher lokalität), können aber die Leistung negativ beeinflussen, wenn die Daten sehr viel weniger lokalisiert sind als der Programmcode, der dann ständig ersetzt wird.

Meine Präferenz: Mittelgrosser Non-unified Cache mit möglichst hoher Assoziativität

4.1 Warum ist der Cache effizient?

Grundsätzlich ist der CPU-nahe, schnelle Speicher den in der Speicherhierarchie weiter unten stehenden immer vorzuziehen. In diesem speziellen Fall ist ein groer Performancegewinn zwischen 4 und 8-facher Assoziativität festzustellen.

4.2 Wie erklären Sie die verschiedenen Steigungen derKurven bei verschiedener Assoziativität?

signifikanter Sprung zwischen assoz=2 und assoz=4, da die adressen von programm und daten (0x2000, 0x3000 ...) dann halbwegs gleichzeitig im cache bahalten werden können. diese werden jeweils in das gleiche set des cache geladen, sodaß viele conflict misses auftreten.

4.3 Welche Vorteile haben die Replacementstrategien?

random: leicht zu implmentieren, bei grossen Speichern sehr gut LRU: weniger Gefahr des Replacement für häufig verwendete code/datenteile: gut für kleine caches; aufwendiger! FIFO: durch temporal locality können wichtige blöcke relativ lange gecached werden, allerdings werden sie jedenfalls irgendwann ersatzt – un unwichtige cacheteile bleiben relativ lange aktiv. kaum vorteile.

4.4 Wie und warum teilen Sie die 512 Worte auf Daren- und Instruktionscache auf?

Da Instruktionscaches über 256 Oktette keine signifikante Verbesserung mehr bringen, scheint die für dieses Programmstück beste Aufteilung jeweils 256 Oktette Daten- und Instruktionscache darzustellen.