

# 185.190 Effiziente Programme

## Aufgabe: Hash-Tabelle

Berger G., Hotz-Behofsits C., Reisinger M., Schmidleithner T.

WS12/13

# Ausgangssituation

- ▶ Testaufruf:
  - ▶ `gcc -lm hash.c -o hash`
  - ▶ `perf stat -e  
cycles,cache-misses,branch-misses,instructions  
./hash input input2`
- ▶ Ergebnis:
  - ▶ Cycles: 6,156,600,783
  - ▶ Instructions: 1,939,017,297
  - ▶ Cache-misses: 37,721,251
  - ▶ Branch mispredictions: 18,758,092
- ▶ Testrechner:
  - ▶ Intel Core i5-2520M CPU @ 2.50GHz
  - ▶ Cache-size:
    - ▶ Lvl 3: 3072 KB
    - ▶ Lvl 2: 512 KB
    - ▶ Lvl 1: 128 KB
  - ▶ RAM: 4GB DDR-3

# Schritt 1

```
gcc -O3 -lm hash.c -o hash
```

## Vorher:

- ▶ Cycles: 6,156,600,783
- ▶ Instructions: 1,939,017,297
- ▶ Cache-misses: 37,721,251
- ▶ Branch mispredictions: 18,758,092

## Nachher:

- ▶ Cycles: 3,705,108,800 (+39,82%)
- ▶ Instructions: 1,158,823,277 (+40,24%)
- ▶ Cache-misses: 37,394,499 (+0,87%)
- ▶ Branch mispredictions: 20,203,186 (-7,70%)

# Code: Schritt 2

## Inlining

```
inline unsigned long hash(char *addr, size_t len);  
inline void insert(char *keyaddr, size_t keylen, int  
    value);  
inline int lookup(char *keyaddr, size_t keylen);
```

# Schritt 2

## Inlining

### Vorher:

- ▶ Cycles: 3,705,108,800
- ▶ Instructions: 1,158,823,277
- ▶ Cache-misses: 37,394,499
- ▶ Branch mispredictions: 20,203,186

### Nachher:

- ▶ Cycles: 3,995,922,639 ( $-7,85\%$ )
- ▶ Instructions: 1,158,154,470 ( $+0,06\%$ )
- ▶ Cache-misses: 37,389,502 ( $+0,01\%$ )
- ▶ Branch mispredictions: 20,691,809 ( $-2,42\%$ )

Keine Verbesserung  $\Rightarrow$  entfernt.

# Code: Schritt 3

Packed

```
struct hashnode {  
    char *keyaddr;  
    size_t keylen;  
    int value;  
} __attribute__((__packed__));
```

# Schritt 3

## Packed

### Vorher:

- ▶ Cycles: 3,705,108,800
- ▶ Instructions: 1,158,823,277
- ▶ Cache-misses: 37,394,499
- ▶ Branch mispredictions: 20,203,186

### Nachher:

- ▶ Cycles: 3,760,116,819 (−1,48%)
- ▶ Instructions: 1,158,688,286 (+0,01%)
- ▶ Cache-misses: 37,372,930 (+0,06%)
- ▶ Branch mispredictions: 19,799,458 (+2,0%)

Verschlechterung ⇒ entfernt.

# Code: Schritt 4

## Lineares Sondieren

```
void insert(char *keyaddr, size_t keylen, int value) {
    struct hashnode **l;
    int startPosition = hash(keyaddr, keylen) & (HASHSIZE-1);
    int position = startPosition;
    do {
        l = &ht[position];
        position = (position + 1) % HASHSIZE;
    } while(*l != NULL && position != startPosition);

    if (*l == NULL) {
        struct hashnode *n = malloc(sizeof(struct hashnode));
        n->keyaddr = keyaddr;
        n->keylen = keylen;
        n->value = value;
        *l = n;
    }
}
```



# Schritt 4

## Lineares Sondieren

### Vorher:

- ▶ Cycles: 3,705,108,800
- ▶ Instructions: 1,158,823,277
- ▶ Cache-misses: 37,394,499
- ▶ Branch mispredictions: 20,203,186

### Nachher:

- ▶ Cycles: 4,588,844,030 (−23,85%)
- ▶ Instructions: 1,315,414,647 (−13,51%)
- ▶ Cache-misses: 58,530,839 (−56,52%)
- ▶ Branch mispredictions: 25,859,851 (−28,00%)

Verschlechterung ⇒ entfernt.

# Code: Schritt 5 (1/2)

## Quadratisches Sondieren

```
void insert(char *keyaddr, size_t keylen, int value) {
    struct hashnode **l;
    int startPosition = hash(keyaddr, keylen) & (HASHSIZE-1);
    int position = startPosition; int i = 0;
    do {
        l = &ht[position];
        position = (startPosition + (int) pow(-1, i) + (i*i/2))
% HASHSIZE;
        i++;
    } while(*l != NULL && position != startPosition);

    if (*l == NULL) {
        struct hashnode *n = malloc(sizeof(struct hashnode));
        n->keyaddr = keyaddr;
        n->keylen = keylen;
        n->value = value;
        *l = n;
    }
}
```

# Code: Schritt 5 (2/2)

## Quadratisches Sondieren

```
int lookup(char *keyaddr, size_t keylen) {
    int startPosition = hash(keyaddr, keylen) & (HASHSIZE-1);
    int position = startPosition;
    struct hashnode *l;
    int i = 0;
    do {
        l = ht[position];
        if (l == NULL) {
            break;
        }
        if (keylen == l->keylen && memcmp(keyaddr, l->keyaddr,
keylen) == 0) {
            return l->value;
        }
        position = (startPosition + (int) pow(-1, i) + (i*i/2))
% HASHSIZE;
        i++;
    } while(position != startPosition);
    return -1;
}
```

# Schritt 5

## Quadratisches Sondieren

### Vorher:

- ▶ Cycles: 3,705,108,800
- ▶ Instructions: 1,158,823,277
- ▶ Cache-misses: 37,394,499
- ▶ Branch mispredictions: 20,203,186

### Nachher:

- ▶ Cycles: 5,948,874,039 (−60,56%)
- ▶ Instructions: 2,588,119,362 (−123,34%)
- ▶ Cache-misses: 43,166,841 (−15,44%)
- ▶ Branch mispredictions: 22,792,713 (−12,82%)

Verschlechterung ⇒ entfernt.

# Code: Schritt 6 (1/2)

## Memoization

```
struct cachenode {  
    struct cachenode *next;  
    int value;  
};
```

## Code: Schritt 6 (2/2)

### Memoization

```
int main(int argc, char *argv[]) {
    ...
    struct cachenode *first_cn = NULL;
    struct cachenode *current_cn;
    ...
    if (i == 0) {
        currentLookup = lookup(p, nextp - p);
        if (first_cn == NULL) {
            first_cn = malloc(sizeof(struct cachenode));
            current_cn = first_cn;
        } else {
            current_cn->next = malloc(sizeof(struct cachenode));
            current_cn = current_cn->next;
        }
        current_cn->next = NULL;
        current_cn->value = currentLookup;
    } else {
        currentLookup = current_cn->value;
        current_cn = current_cn->next;
    } ...
}
```

# Schritt 6

## Memoization

### Vorher:

- ▶ Cycles: 3,705,108,800
- ▶ Instructions: 1,158,823,277
- ▶ Cache-misses: 37,394,499
- ▶ Branch mispredictions: 20,203,186

### Nachher:

- ▶ Cycles: 1,223,009,952 (+66,99%)
- ▶ Instructions: 867,096,107 (+25,17%)
- ▶ Cache-misses: 5,077,492 (+86,42%)
- ▶ Branch mispredictions: 7,507,892 (+62,84%)

Verbesserung ⇒ beibehalten.

# Code: Schritt 7

## Memoization 2: dynamisches Array

```
int main(int argc, char *argv[]) {  
    ...  
    int currentLookup;  
    int *cache = calloc(HASHSIZE, sizeof(int));  
    int cacheSize = HASHSIZE;  
    int cacheCounter;  
    ...  
    if (i == 0) {  
        currentLookup = lookup(p, nextp - p);  
        if (cacheCounter >= cacheSize) {  
            realloc(cache, sizeof(int));  
            cacheSize++;  
        }  
        cache[cacheCounter] = currentLookup;  
    } else {  
        currentLookup = cache[cacheCounter];  
    }  
    r = ((unsigned long)r) * 2654435761L + currentLookup;  
    ...  
}
```



# Schritt 7

## Memoization 2 (dynamisches Array)

### Vorher:

- ▶ Cycles: 1,223,009,952
- ▶ Instructions: 867,096,107
- ▶ Cache-misses: 5,077,492
- ▶ Branch mispredictions: 7,507,892

### Nachher:

- ▶ Cycles: 917,739,423 (+24, 96%)
- ▶ Instructions: 702,170,463 (+19, 02%)
- ▶ Cache-misses: 4,647,945 (+8, 46%)
- ▶ Branch mispredictions: 7,421,590 (+1, 15%)

Verbesserung  $\Rightarrow$  beibehalten.

# Code: Schritt 8

## Memoization 3 (Optimierung der Reallokierung)

```
#define CACHE_ALLOC_STEP_SIZE 96
int main(int argc, char *argv[]) {
    ...
    int currentLookup;
    int *cache = calloc(HASHSIZE, sizeof(int));
    int cacheSize = HASHSIZE; int cacheCounter;
    ...
    if (i == 0) {
        currentLookup = lookup(p, nextp - p);
        if (cacheCounter >= cacheSize) {
            cache = realloc(cache, CACHE_ALLOC_STEP_SIZE *
sizeof(int));
            cacheSize += CACHE_ALLOC_STEP_SIZE;
        }
        cache[cacheCounter] = currentLookup;
    } else {
        currentLookup = cache[cacheCounter];
    }
    r = ((unsigned long)r) * 2654435761L + currentLookup;
    ...
}
```

# Schritt 8

## Memoization 3 (Optimierung der Reallokierung)

### Vorher:

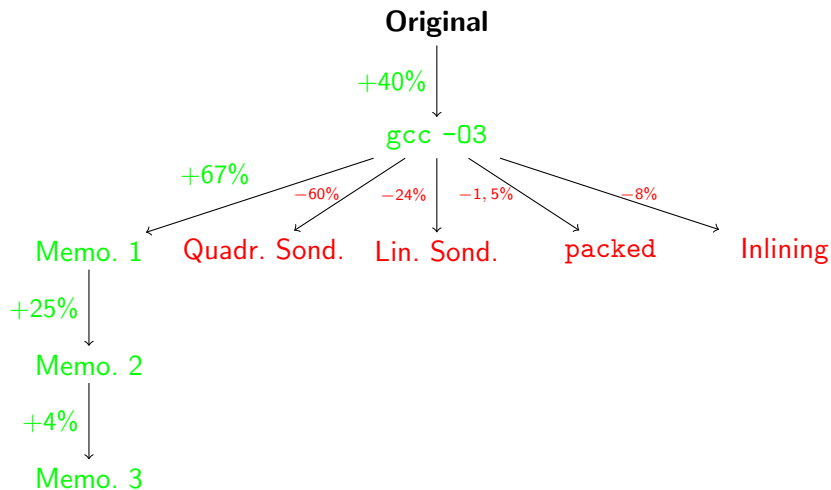
- ▶ Cycles: 917,739,423
- ▶ Instructions: 702,170,463
- ▶ Cache-misses: 4,647,945
- ▶ Branch mispredictions: 7,421,590

### Nachher:

- ▶ Cycles: 879,335,268 (+4, 18%)
- ▶ Instructions: 701,589,819 (+0, 08%)
- ▶ Cache-misses: 4,651,776 (-0, 08%)
- ▶ Branch mispredictions: 7,527,878 (-1, 43%)

Verbesserung  $\Rightarrow$  beibehalten.

# Übersicht



# Mathematischer Ansatz

## Auflösung von Schleifen und explizite Darstellung

```
for (i=0; i<10; i++) {  
    for (p=input2.addr, endp=input2.addr+  
        input2.len; p<endp; ) {  
        nextp=memchr(p, '\n', endp-p);  
        if (nextp == NULL)  
            break;  
        r = ((unsigned long)r) * 2654435761L +  
lookup(p, nextp-p);  
        r = r + (r>>32);  
        p = nextp+1;  
    }  
}
```

Gleiche Berechnung (nur mit anderem Startwert) wird 10 mal durchgeführt.

# Mathematischer Ansatz

Auflösung von Schleifen und explizite Darstellung

```
r = ((unsigned long)r) * 2654435761L + lookup(p,  
      nextp-p);  
r = r + (r>>32);
```

## Differenzengleichung 1. Ordnung

$$x_{n+1} = (x_n * 2654435761 + b_n) * (1 + \frac{1}{2^{32}})$$

# Mathematischer Ansatz

## Auflösung von Schleifen und explizite Darstellung

```
r = ((unsigned long)r) * 2654435761L + lookup(p,  
      nextp-p);  
r = r + (r>>32);
```

$$x_{n+1} = (x_n * 2654435761 + b_n) * (1 + \frac{1}{2^{32}})$$

$$x_0 = 0$$

$$x_1 = 0 * 2654435761 + b_0 + \frac{x_0 * 2654435761 + b_0}{2^{32}} = b_0 + \frac{b_0}{2^{32}}$$

...

# Mathematischer Ansatz

Auflösung von Schleifen und explizite Darstellung

$$\begin{aligned}x_2 &= x_1 * 2654435761 + b_1 + \frac{x_1 * 2654435761 + b_1}{2^{32}} \\x_3 &= ((b_0 + \frac{b_0}{2^{32}}) * 2654435761 + b_1 + \frac{x_1 * 2654435761 + b_1}{2^{32}}) * 2654435761 + \\&b_2 + \frac{((b_0 + \frac{b_0}{2^{32}}) * 2654435761 + b_1 + \frac{x_1 * 2654435761 + b_1}{2^{32}}) * 2654435761 + b_2}{2^{32}} \\&\dots\end{aligned}$$



# Mathematischer Ansatz

Allgemeine iterative Berechnung von Differenzenglg. 1. Ordnung

vereinfachtes Beispiel:

$$x_{n+1} = x_n * c + b_n$$

$$x_3 = ((x_0 * c + b_0) * c + b_1) * c + b_2$$

$$x_3 = c^3 * x_0 + c^2 * b_0 + c^1 * b_1 + c^0 * b_2$$

$$x_n = c^n * x_0 + \sum_{i=0}^{n-1} c^{n-i-1} * b_i$$

Problematik:

Man hat eine Potenz mit der **Anzahl an Iterationen** als Hochzahl.

Bsp: input2 hat 724129 Zeilen über die iteriert wird!