

# 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%)

# 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 ⇒ entfernt.

# 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 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 3

Packed

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

# 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 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 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 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;
}
```