Humboldt University

Computer Science Department
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Operating Systems Principles

Valgrind

Valgrind

What is Valgrind?

- a multipurpose code profiling and memory debugging tool
- monitors memory usage such as calls to malloc and free (or new and delete in C++)
- Valgrinds detects:
  - usage of uninitialized memory
  - write off the end of an array
  - if you forget to free a pointer
- Other tools
  - *Addrcheck*, similar to Memcheck but with much smaller CPU and memory overhead.
  - *Massif*, a heap profiler
  - *Helgrind* and *DRD*, detect race conditions in multithreaded code
  - *Cachegrind*, a cache profiler (GUI KCacheGrind)
  - *Callgrind*, a callgraph analyzer (GUI KCacheGrind)

Valgrind

Usage

- valgrind –tool=*valgrind_tool program_name*
- tools:
  - memcheck
  - callgrind
  - cachegrind

- Example

```
% valgrind --tool=memcheck program_name
...
 =18515== malloc/free: in use at exit: 0 bytes in 0 blocks.
 ==18515== malloc/free: 1 allocs, 1 frees, 10 bytes allocated.
 ==18515== For a detailed leak analysis, rerun with: --leak-check=yes
```

Valgrind - Memcheck

- Code

```
#include <stdlib.h>

int main()
{
 char *x = malloc(100); /* or, in C++, "char *x = new char[100] */
 x = NULL;
 return 0;
}
```

- Run

```
% valgrind --tool=memcheck --leak-check=yes example1
```

- Output

```
==2330== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
 ==2330== at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
 ==2330== by 0x804840F: main (example1.c:5)
```

- Result
  - **Memory leak**
  - forget to call *free()* for „*x*“

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Valgrind - Memcheck

- Code

```
#include <stdlib.h>

int main()
{
  char *x = malloc(10);
  x[10] = 'a';
return 0;
}
```

- Run

```
% valgrind --tool=memcheck --leak-check=yes example2
```

- Output

```
==9814== Invalid write of size 1
==9814== at 0x804841E: main (example2.c:6)
==9814== Address 0x1BA3607A is 0 bytes after a block of size 10 alloc'd
==9814== at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
==9814== by 0x804840F: main (example2.c:5)
```

- Result
  - **Invalid write**

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Valgrind - Memcheck

- Code

```
#include <stdio.h>

int main()
{
  int x;
  if(x == 0) {
    printf("X is zero"); /* replace with cout and include iostream for C++ */
  }
  return 0;
}
```

- Run

```
% valgrind --tool=memcheck --leak-check=yes example3
```

- Output

```
==17943== Conditional jump or move depends on uninitialised value(s)
==17943== at 0x804840A: main (example3.c:6)
```

- Result
  - Use Of Uninitialized Variables

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Valgrind - Memcheck

- Code

```c
#include <stdio.h>

int foo(int x) {
  if(x < 10) {
    printf("x is less than 10\n");
  }
}

int main() {
  int y;
  foo(y);
}
```

- Output

```
==4827== Conditional jump or move depends on uninitialised value(s)
==4827== at 0x8048366: foo (example4.c:5)
==4827== by 0x8048394: main (example4.c:14)
```

- Result
  - Use Of Uninitialized Variables

Valgrind - Callgrind

- Code

```
#include <iostream>
#include <atomic>

std::atomic<int> cnt(0);

void f() {
   for (int n = 0; n < 10000; ++n)
      cnt.fetch_add(1, std::memory_order_relaxed);
}

void g() {
   for (int n = 0; n < 10000; ++n)
      cnt.fetch_add(1, std::memory_order_seq_cst);
}

void h() {
   for (int n = 0; n < 10000; ++n)
      cnt++;
}

int main()
{
   f(); g(); h();

   std::cout << "Counter: " << cnt << '\n';
}
```


Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

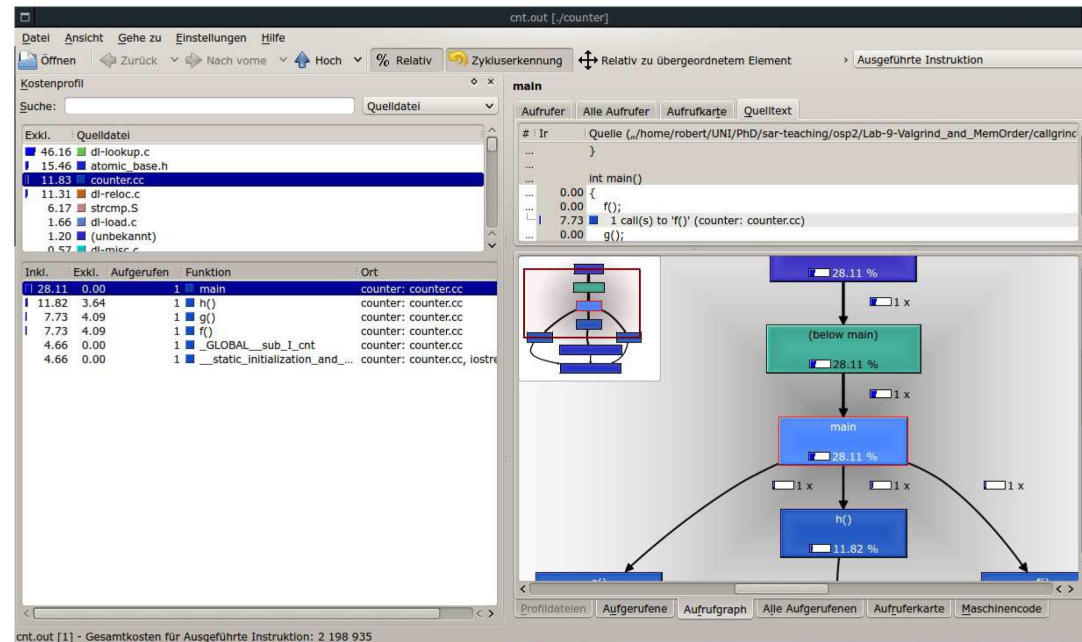
Valgrind - Callgrind

- Run

```
% valgrind --tool=callgrind --collect-jumps=yes --dump-instr=yes --callgrind-out-file=cnt.out ./counter
```

- Result
  - File *cnt.out*
  - can be used with *KCachegrind*

```
% kcachegrind cnt.out
```



KCachegrind

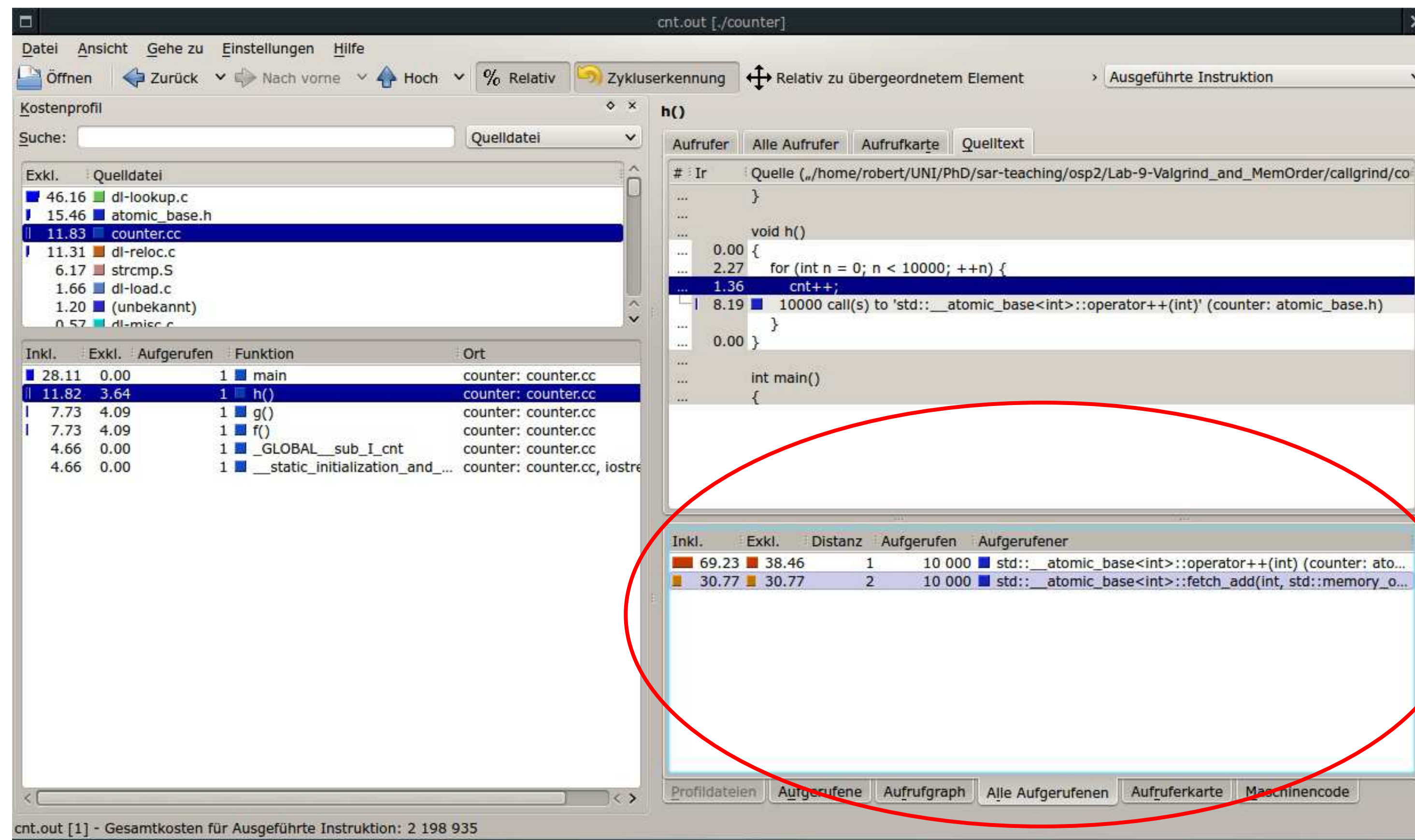
- A profile data visualization

KCachegrind

- Aufrufgraph
  - Welche Funktion ruft welche Funktion auf

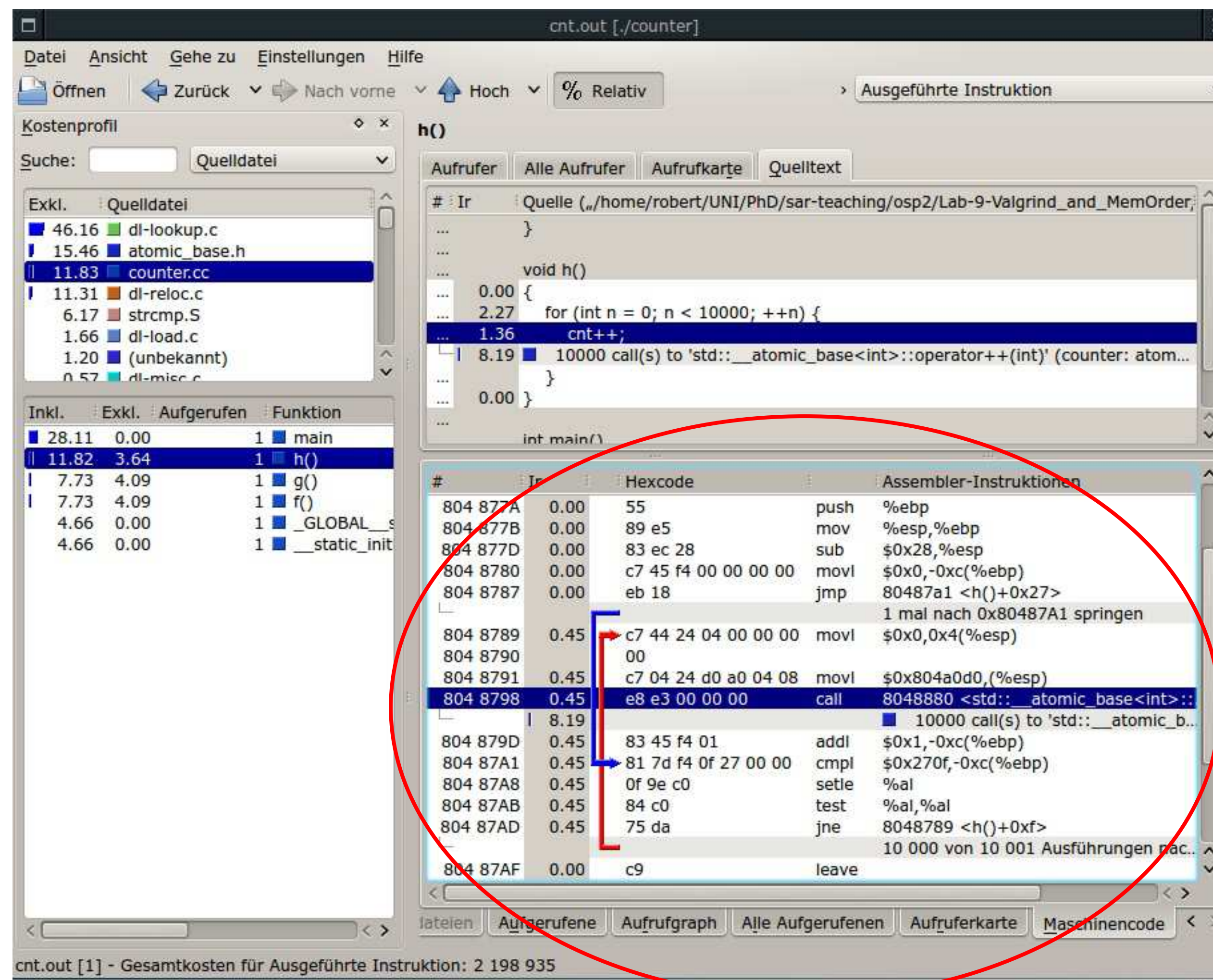
KCachegrind

- Aufgerufene
  - Welche Funktion ruft welche Funktion auf



KCachegrind

- Maschinencode
  - Zeigt den Maschinencode des Programms



Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Links

http://valgrind.org/

http://valgrind.org/docs/manual/manual-core.html

http://www.cprogramming.com/debugging/valgrind.html