Valgrind

Effiziente Programmierung

Niclas Schroeter

Universität Hamburg

2018-05-17

Gliederung (Agenda)

Allgemeines

- Was ist Valgrind?
  - Dynamic Binary Instrumentation Framework
  - Toolbox
- Open Source Software
- Gründer des Projekts: Julian Seward

Verfügbare Tools

- Memcheck
- Cachegrind
- Callgrind
- Helgrind

- noch viele weitere...

Memcheck

- Das populärste Tool
- Entdeckt Probleme bei der Speicherverwaltung
  - Zugriff auf nicht zugewiesenen Speicher
  - Speicherlecks etc.
- Hauptsächlich für C - / C++ - Programme

Cachegrind

- Profiler
- Simuliert den Cache und dessen Nutzung
- Aufteilung
  - L1 $\rightarrow$ I1 und D1
  - L2 bzw. LL

Cachegrind

- Kann auch Sprungvorhersagen simulieren
- Misst die Fehlerrate und sammelt Informationen

- Für jede Sprache nützlich

Callgrind

- Erweiterung von Cachegrind
- Sammelt Informationen über Funktionsaufrufe innerhalb des Programms

  → Erstellt Callgraph

Callgrind

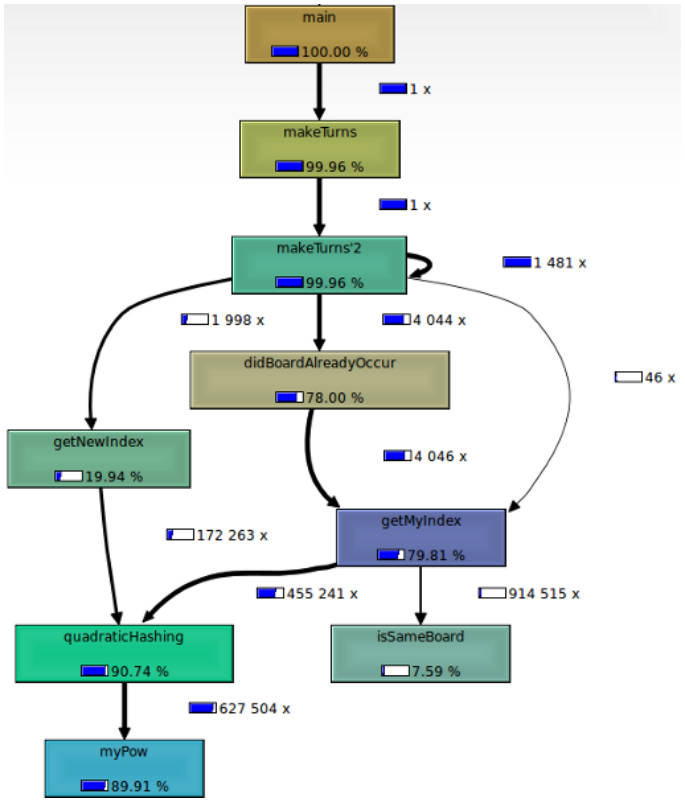

Abbildung: Callgraph für Vier-Gewinnt [Tho12]

Helgrind

- Analysiert Threads und deren korrekte Nutzung
- POSIX Threads
  $\rightarrow$ für C - , C++ - und Fortran-Programme

Funktionsweise

- Was passiert, wenn ein Programm mit Valgrind ausgeführt wird?
  - Tool = Valgrind-Kern + plug-in
  - Kern simuliert CPU

  $\rightarrow$ Programm läuft deutlich langsamer

Shadow Values

- Jedes(!) Register wird durch ein shadow register / shadow memory dargestellt [NS07]
  - Exakter Wert
  - Bestimmte Eigenschaft
- Werden die gesamte Laufzeit über aktualisiert
- Haben keine Auswirkung auf die Ausführung des eigentlichen Programms

Der Kern

- Was passiert genau mit dem Code des Programms?
  1. Programm wird in Codeblocks segmentiert
  2. Disassembler
  3. Injektion von Code zwecks Analyse
  4. Zurück in Maschinencode
  5. Ausführen und nächster Block

Rolle der Shadow Values

- Einige Tools greifen auf sie zu, um bestimmte Werte zu prüfen
- Greifen dadurch nicht in die Ausführung des eigentlichen Programms ein
- Beispiel: Memcheck

Funktion von Memcheck

- Für jedes bit ein "valid-value bit"(V bit) [Dev18]
- V bit wird in gewissen Fällen geprüft
- Je nachdem wird dann Fehlermeldung produziert oder nicht

Funktion von Memcheck

```
1 int i, j;
2 int a[10], b[10];
3 for (i = 0; i < 10; i++){
4     j = a[i];
5     b[i] = j;
6 }
```

- Code sorgt für keine Fehlermeldung von Memcheck

Funktion von Memcheck

```
1 for (i = 0; i < 10; i++){
2     j += a[i]:
3 }
4 if (j == 77)
5     printf("hello there");
```

- Fehlermeldung wird erzeugt

Funktion von Memcheck

- Warum wird in dem zweiten Fall eine Fehlermeldung ausgegeben?
- 3 Fälle, für die V bits geprüft werden
  - Kontrollfluss hängt von nicht initialisiertem Wert ab
  - Wert einer Speicheradresse hängt davon ab
  - Vor/nach Systemaufruf
- Dementsprechend nur Fehlermeldungen in solchen Fällen

Fehlermeldung von Memcheck

```
1  int main()
2  {
3      int x;
4      printf ("x = %d\n", x);
5  }
```

```
1  Conditional jump or move depends on uninitialised
2  value
3      at 0x402DFA94: _IO_vfprintf (_itoa.h:49)
4      by 0x402E8476: _IO_printf (printf.c:36)
5      by 0x8048472: main (tests/manuel1.c:8)
```

Ausgegebene Fehlermeldung

Funktion von Cachegrind

- Daten in drei Strukturen verarbeitet [Net04]
  1. Simulierte Cache Ebenen
  2. Cost Centre Tables
  3. Instr-info Table
- Was genau passiert während der Analyse?

Funktion von Cachegrind

1. Zählen der Maschinenbefehle
2. Code-Injektion
3. Kategorisierung der Befehle
4. Aufzeichnung der Interaktion
5. Ausgabe am Ende der Simulation

Ausgabe von Cachegrind

```
--------------------------------------------------------------------------------
-- User-annotated source: concord.c
--------------------------------------------------------------------------------
Ir         I1mr ILmr Dr       D1mr   DLmr   Dw       D1mw    DLmw

         .    .    .         .      .      .         .       .       void init_hash_table(char *file_name, Word_Node *table[])
         3    1    1         .      .      .         1       0       0  {
         .    .    .         .      .      .         .       .       .      FILE *file_ptr;
         .    .    .         .      .      .         .       .       .      Word_Info *data;
         1    0    0         .      .      .         1       1       1      int line = 1, i;
         .    .    .         .      .      .         .       .       .
         5    0    0         .      .      .         3       0       0      data = (Word_Info *) create(sizeof(Word_Info));
         .    .    .         .      .      .         .       .       .
     4,991    0    0     1,995      0      0       998       0       0      for (i = 0; i < TABLE_SIZE; i++)
     3,988    1    1     1,994      0      0       997      53      52          table[i] = NULL;
         .    .    .         .      .      .         .       .       .
         .    .    .         .      .      .         .       .       .      /* Open file, check it. */
         6    0    0         1      0      0         4       0       0      file_ptr = fopen(file_name, "r");
         2    0    0         1      0      0         .       .       .      if (!(file_ptr)) {
         .    .    .         .      .      .         .       .       .          fprintf(stderr, "Couldn't open '%s'.\n", file_name);
         1    1    1         .      .      .         .       .       .          exit(EXIT_FAILURE);
         .    .    .         .      .      .         .       .       .      }
         .    .    .         .      .      .         .       .       .
   165,062    1    1    73,360      0      0    91,700       0       0      while ((line = get_word(data, line, file_ptr)) != EOF)
   146,712    0    0    73,356      0      0    73,356       0       0          insert(data->;word, data->line, table);
         .    .    .         .      .      .         .       .       .
         4    0    0         1      0      0         2       0       0      free(data);
         4    0    0         1      0      0         2       0       0      fclose(file_ptr);
         3    0    0         2      0      0         .       .       .  }
```

Zusammenfassung

- Warum sollte man Valgrind nutzen?
  - Viele unterschiedliche Tools finden viele Fehler
  - Anhaltspunkte für Optimierung
- Open Source
  - Jeder kann eigene Tools bauen
- Trotzdem gutes Verständnis vom Programmieren nötig, um die Informationen nutzen zu können

Literatur

[Dev18] Valgrind Developers. Valgrind user manual, 2018.

[Net04] Nicholas Nethercote. *Dynamic Binary Analysis and Instrumentation*. PhD thesis, University of Cambridge, 2004. S. 50-59.

[NS07] Nicolas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. 2007.

[Tho12] Martin Thoma. Profiling c programs, 2012.