Code Tuning

Vortrag zum Oberseminar Softwareentwicklung

Übersicht

- Einleitung
- Prinzip
- Regeln für das Code Tuning
- Beispiele
- Quellen

Einleitung

- Code Tuning sollte der letzte Schritt bei der Optimierung eines Programms sein
- „Premature optimization is the root of all evil." -- Donald Knuth
- Viel Aufmerksamkeit für Optimierungen
  - \+ Extrem effiziente Programme
  - – Schwer (oder gar nicht) zu lesen und zu warten
- Zuwenig Aufmerksamkeit für Optimierungen
  - \+ Sehr gut strukturierte Programme
  - – Ineffizient und somit nutzlos
- Code Tuning → Low-Level-Ansatz um rechenintensive Programmteile mit kleinen Änderungen effizienter zu machen

Prinzip

- Code-Tuning sollte selten angewandt werden
- Effizienz
  - Andere Eigenschaften einer Software sind ebenso wichtig, vielleicht sogar wichtiger
  - Zu frühe Optimierung kann sich sehr nachteilig auswirken, z.B. auf Korrektheit, Funktionalität und Wartbarkeit
- Messen der verbrauchten Zeit (Profiling)
  - Oft benutzen nur wenige Funktionen die meiste Zeit (Hot Spots) → Optimierung dort ansetzen
  - Kostenmodelle als Hilfsmittel benutzen [3]

Prinzip II

- Design
  - Bevor Code Tuning zum Einsatz kommt, versuchen andere effektivere Möglichkeiten zu finden (z.B. Ändern der Datenstruktur)
- Beschleunigung/Verlangsamung:
  - Nachdem der Code geändert wurde, erneutes Profiling mit repräsentativen Eingaben → Überprüfen des Effektes

Regeln für das Code Tuning

- Gleiche Fälle finden
- Caching
- Ersetzen von schwierig zu berechnenden Ausdrücken mit gleichwertigen einfacheren
- Prozedurhierarchien kürzen
- Tests kombinieren
- Schleifen aufrollen
- Anreichern von Datenstrukturen

Beispiel 1

- Codefragment aus einer Vektorrotation

```
k = (j + rotdist) % n;
```

- **% → ~100 nsec, +/- → ~10 nsec**

```
k = j + rotdist;
if(k >= n)
    k -= n;
```

- rotdist = 1 → 199nsec ↘ 57 nsec (~70% schneller)
- rotdist = 10 → 206 nsec → 206 nsec ???
- Bei rotdist = 1 wird auf den Speicher sequenziell zugegriffen und die Rechnung verschlingt die meiste Zeit
- Bei rotdist = 10 wird nur jedes 10. Wort gelesen und das Laden von Daten aus dem RAM in den Cache braucht die meiste Zeit

Beispiel 2

- Maximum zweier Werte

```
maxendinghere = max(maxendinghere, 0);
maxsofar = max(maxsofar, maxendinghere);

float max(float a, float b)
{ return a > b ? a : b; }
```

  → 89 nsec

- Ersetzen durch Makro

```
#define max(a, b) ((a) > (b) ? (a) : (b))
```

  ↘ 47 nsec (~50% schneller)

- Aber auch:

  n = 10.000 → 10 msec ↗ 100 s

Beispiel 3

- Sequenzielle Suche

```
int ssearch1(t)
    for i = [0, n)
        if x[i] == 1
            return i
    return -1
```

→ 4.06 nsec

Beispiel 3

- Zusammenfassen der beiden Tests

```
int ssearch2(t)
    hold = x[n]
    x[n] = t
    for (i = 0; ; i++)
        if x[i] == t
            break
    x[n] = hold
    if i == n
        return -1
    else
        return i
```

→ 3.87 nsec (5% schneller)

Beispiel 3

- Aufrollen der Schleife

```
int ssearch3(t)
    x[n] = t
    for (i = 0; ; i += 8)
          if(x[i  ] == t) {          break }
          if(x[i+1] == t) { i += 1; break }
          if(x[i+2] == t) { i += 1; break }
          if(x[i+3] == t) { i += 1; break }
          if(x[i+4] == t) { i += 1; break }
          if(x[i+5] == t) { i += 1; break }
          if(x[i+6] == t) { i += 1; break }
          if(x[i+7] == t) { i += 1; break }
    if i == n
          return -1
    else
          return i
```

→ 1.70 nsec (56% schneller)

Beispiel 4

- Berechnung des nächsten Nachbarn auf dem Globus
  - Punkte mit Längen- und Breitengrad gegeben
  - Zur Berechnung wurden komplizierte trigonometrische Formeln mit 10 Sinus- und Kosinusfunktionen benutzt
  - Einfaches Programm, einfach zu schreiben, gut nutzbar für kleine Datenmengen, bei großen Datenmengen jedoch benötigte der Algorithmus mehrere Stunden

Beispiel 4

- Optimierung
  - Ersetzen der Länge und Breite durch x, y und z Koordinaten
  - Beim Durchlauf werden die Koordinaten des Punktes aus den Längen- und Breitenangaben berechnet
  - Entfernung ist nun nur noch die Summe der drei Quadrate (Euklidsche Entfernung)
  - Nachteil: Es wird mehr Speicherplatz benötigt
  - Vorteil: Programm lief statt mehrerer Stunden nur eine halbe Minute
  - Hinzufügen von einigen Zeilen Code hat das Ziel erreicht, wo eine Änderung von Algorithmus und Daten mehrere hundert Zeilen Code benötigt hätte

Quellen

(1) Programming Pearls - Jon Bentley, Addison-Wesley 2000

(2) Introduction to Code Tuning (http://www.stevemcconnell.com/cctune.htm) - Steven C. McConnell 1993

(3) Cost Models for Time and Space (http://www.cs.bell-labs.com/cm/cs/pearls/appmodels.html) – Jon Bentley, 2000

(4) Rules for Code Tuning (http://www.cs.bell-labs.com/cm/cs/pearls/apprules.html) – Jon Bentley, 2000