

# Technisch-Wetenschappelijke Software

## Code optimalisaties (in C++)

Dirk Nuyens

Academiejaar 2006-2007

### 1 Het gebruik van de TSC op x86

Voor het nauwkeurig timen van stukjes code kan je op heel wat architecturen gebruik maken van *hardware performance counters*. Op de intel architectuur kan dit door middel van de *time stamp counter* (TSC). De assembler instructie om deze uit te lezen is simpelweg `rdtsc` en plaatst een 64 bit unsigned integer in de registers `EDX:EAX`.

De GNU compiler en de intel compiler kunnen de TSC uitlezen met behulp van inline assembly:

```
inline uint64_t rdtsc()
{
    uint32_t lo, hi;
    asm volatile ("rdtsc\n\t" : "=a" (lo), "=d" (hi));
    return (uint64_t) hi << 32 | lo;
}
```

De waarde die je terug krijgt is het aantal clock cycles. Als je de snelheid van je processor weet (b.v. met `cat /proc/cpuinfo | grep 'cpu MHz'`), dan kan je de tijd voor 1 clock cycle vinden als  $S^{-1}$  met  $S$  de snelheid van de CPU in Hz (clock cycles per seconde).

Indien je bovenstaande code gebruikt kunnen er een aantal problemen optreden. Ten eerste kan het zijn dat je start of stop meting niet op de correcte plaats zit vanwege *out of order execution* (hiervoor zou je een extra instructie kunnen bijplaatsen, b.v. `cpuid`, die de instructie stroom serializeert). Ten tweede kan je computer zijn clock snelheid dynamisch wijzigen (b.v. door powermanagement). Ten derde kan je problemen hebben op systemen met meerdere cores, waarbij de waarden van de TSC kunnen verschillen van core tot core en je niet kan garanderen dat je start en stop metingen op dezelfde core zullen doorgaan.

### 2 Opgave voor tijdens de oefenzitting

#### 2.1 Opwarmertjes

1. Wat is de snelheid van de computer waarop je werkt?
2. Indien je TSC uitleest als 32 bit unsigned integer, na hoeveel tijd (in seconden) is je teller dan rond geweest?
3. Na hoeveel tijd is de teller rond geweest voor de volledige 64 bit waarde? (Druk dit uit in een makkelijk verstaanbaardere eenheid dan seconden.)

## 2.2 Een benchmarking klasse

In de file `Bench.h` bevindt zich een klasse `Bench` die gebruik kan maken van de TSC of van de standaard unix routine `getrusage`. Bekijk deze file eens van dichtbij en compileer de voorbeeldcode die in commentaar staat.

1. Interpreteer de uitvoer van de benchmarking klasse.
  - (a) Wat is `warmup` en waarvoor dient dit? Zie je dit effect voor dit voorbeeld? Wat is de verklaring? Gebruik Matlab of Octave om de data te plotten.
  - (b) Voer het programma een aantal keren uit en let op de verhouding `max/min`. Krijg je deze waarde veel groter dan 1? Verhoog de waarde `n` voor het aantal metingen anders eens tot 500. Wat gebeurt er bij uitschieters? Gebruik ook hier Matlab of Octave om de data te plotten.
  - (c) Prent in je hoofd dat je je data alleen gebruik bij een “goede” waarde van `max/min`!
2. Zoek in de man page van `gcc` eens naar mogelijke optimalisatie vlaggen (kijk ook bij de intel specifieke vlaggen) en probeer er eens wat uit.
3. Maak een overzicht voor een aantal verschillende optimalisatievlaggen (probeer zeker `-O0` tot `-O3`). Bewaar voor elke vlag de typische uitvoer van de benchmark en bekijk ook hier het verschil eens grafisch in Matlab of Octave. (De presentatie van deze data maakt deel uit van het huiswerk.)

## 2.3 Loop unrolling (p. 168)

Maak een gelijkaardig C++ programma als hierboven om het inwendig product van twee vectoren te berekenen. Je implementatie is een rechtstreekse implementatie van de formule (geen manuele loop unrolling) waarbij je de vectoren voorstelt door `vector<double>`. (Bekijk de voorbeeldcode uit het vorige punt om te achterhalen hoe je met de template klasse `vector` moet werken.)

1. Test je code met de standaard optimalisatievlaggen `-O0` tot `-O3`. (Als grootte voor de vectoren kan je 100 nemen.) Bewaar de data en stel dit grafisch voor.
2. Gebruik nu de optie `-funroll-loops` samen met de optimalisatievlaggen van hierboven. Vergelijk de uitvoer.
3. Tracht de lus nu ook eens manueel te ontrollen en vergelijk opnieuw. (Dit is een tweede routine.)
4. Gebruik de optie `-S` om de assembler output van de compiler te bekijken en tracht te achterhalen hoeveel keer de lus werd ontrolld door `-funroll-loops`. In plaats van de optie `-S` te gebruiken kan je ook `objdump -D` op je executable uitvoeren. De plaats waar je naar moet kijken is makkelijk te vinden door op de `rdtsc` instructies te zoeken.
5. (Voor thuis.) De standard template library (STL) is een zeer praktische en geavanceerde bibliotheek voor C++. De implementatie zorgt er echter wel voor dat je steeds moet optimaliseren (en dit is zeker zo voor nog geavanceerdere technieken als expression templates). Maak een nieuwe implementatie die in plaats van het `vector` type uit de STL nu met gewone arrays werkt. Vergelijk opnieuw voor de standaard optimalisatievlaggen. Welke versie is sneller (en wanneer)? Plaats ook deze vergelijking in grafiek.
6. (Voor thuis.) Splits de variabele voor de som bij je zelf ontrolde lus in evenveel stukken en stel dan pas op het einde je echte som samen uit deze aparte stukken. Is dit een verbetering ten opzichte van je gewone zelf ontrolde lus?

## 2.4 Data caches (p. 166 en 164)

Implementeer  $y \leftarrow Ax + y$ :

```
for(i = 0; i < m; i++)
  for(j = 0; j < n; j++)
    y[i] = y[i] + A[i][j] * x[j];
```

en ook met de twee lussen omgekeerd.

1. Achterhaal de grootte van de data caches. (De informatie uit `/proc/cpuinfo` volstaat.)
2. Doe de berekening op p. 165 bovenaan en test dit voor je twee routines. Kies hierbij geschikte optimalisaties.

Implementeer de volgende lussen als afzonderlijke tests:

```
for(i = 0; i < n; i++)
  y[i] = x[i] + a;
```

```
for(i = 0; i < n; i++)
  y[i] = x[0] + a;
```

```
for(i = 0; i < n; i++)
  y[0] = x[i] + a;
```

```
for(i = 0; i < n; i++)
  y[0] = x[0] + a;
```

1. Compileer met `-funroll-loops` en vergelijk de timings.
2. Verklaar wat het verschil is tussen de vier varianten in functie van de data cache.

## 3 Opgave voor thuis

Maak een verslag waarin je je resultaten bespreekt.

Stuur de code alsook een PDF van je verslag met figuren naar Dirk Nuyens. Geef aan hoeveel tijd je besteedde aan de opdracht voor thuis. Je hebt hiervoor 1,5 weken.