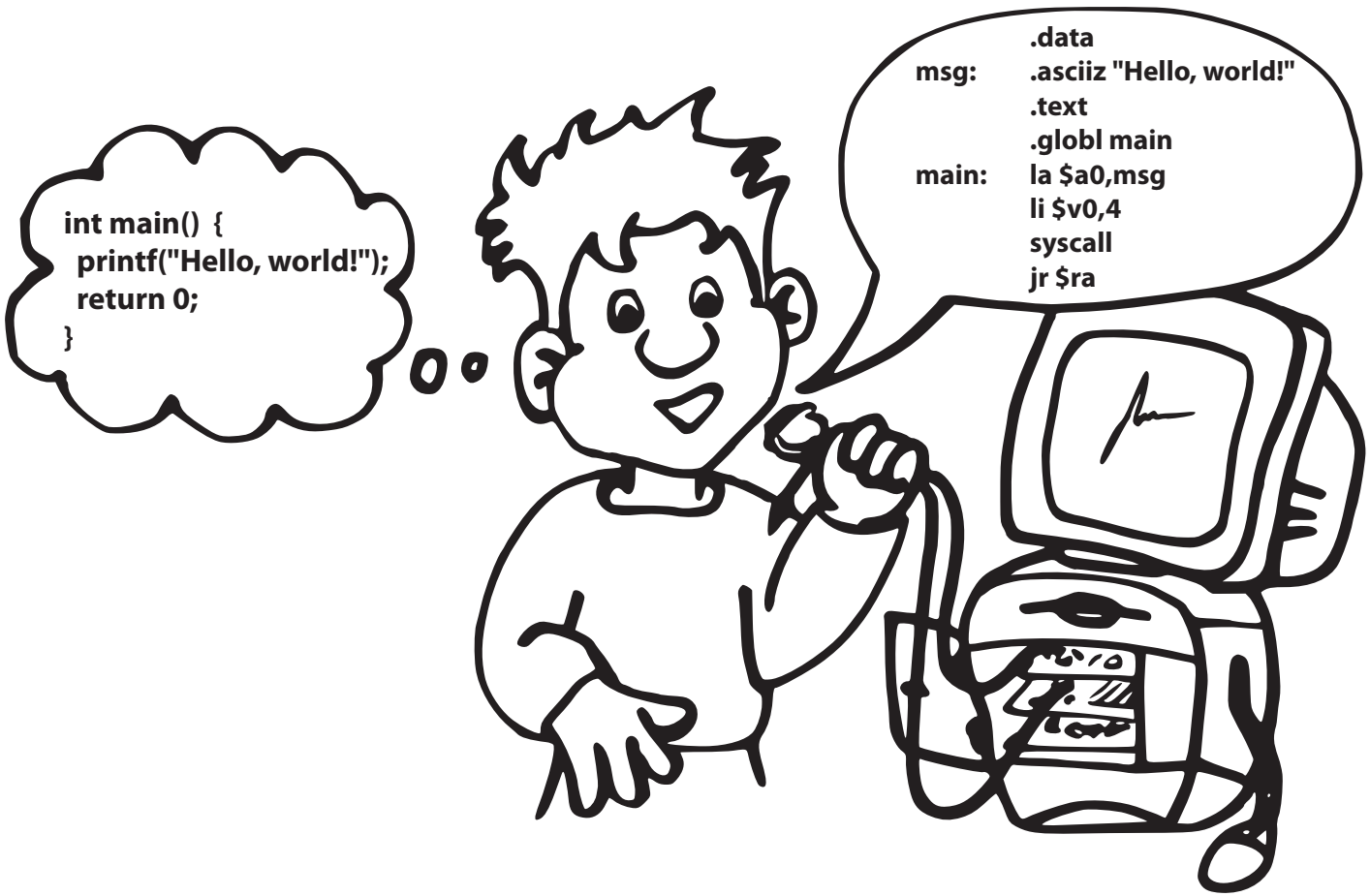
Grundlagen der Informatik III
Wintersemester 2010/2011 – 10. Vorlesung
Dr.-Ing. Wolfgang Heenes

TECHNISCHE UNIVERSITÄT DARMSTADT



Inhalt

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Literatur
2. Analyse von C Programmen
3. OpenMP
4. Intel
5. Zusammenfassung und Ausblick

Literatur

TECHNISCHE UNIVERSITÄT DARMSTADT

[BO10] Bryant, Randal E. und David R. O´Hallaron: *Computer Systems - A Programmer´s Perspective*.
Prentice Hall, 2010.

Analyse von C Programmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für das Schreiben von effizienten Programmen muss man wissen, wo die *Performance Bottlenecks* sind
- Bisher
  - Takte aus Tabellen
  - Tickcounter in Assembler
- Unter Unix (und Windows) verschiedene Programme zur Performance Analyse vorhanden
- Für heute $\Rightarrow$ Performance heißt Ausführungszeit des Programms

Analyse eines C Programms mit zwei Funktionen
Teil I - Unterprogramme

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include<stdio.h>
long fak_ite (int n)
{
  int i;
  long result = 1;
  for (i = 2; i <= n; i++)
    result *= i; /* (result = result * i) */
   return result;
}
long fak_rek (int n)
{ if (n>1) {
 return fak_rek (n-1) * n ;
} else {
return 1;
} }
```

Analyse eines C Programms mit zwei Funktionen, Teil II - Hauptprogramm 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int main (void)
{
int  n = 30;
long erg1, erg2;

long schleife;

for (schleife = 1; schleife < 10000000; schleife++) {
erg1 = fak_ite (n);
//printf("Fakultät: %d\n", erg1);
erg2 = fak_rek (n);
//printf("Fakultät: %d\n:", erg2);
}
return 0;
}
```

Analyse eines C Programms mit zwei Funktionen, Teil II - Hauptprogramm 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int main (void)
{
int  n = 30;
long erg1, erg2;

register schleife;

for (schleife = 1; schleife < 10000000; schleife++) {
erg1 = fak_ite (n);
//printf("Fakultät: %d\n", erg1);
erg2 = fak_rek (n);
//printf("Fakultät: %d\n:", erg2);
}
return 0;
}
```

Analyse von C Programmen
Vergleich der Hauptprogramme

TECHNISCHE UNIVERSITÄT DARMSTADT

- Offensichtlich gibt es in C auch den Datentyp **register**
- Zur Erinnerung: *schleife* ist in C eine lokale Variable des Hauptprogramms
- Hauptprogramm ist eigentlich auch ein Unterprogramm
- Lokale Variablen können in Unterprogrammen auf dem Stack realisiert werden.
- Lokale Variablen können aber auch im Register realisiert werden.
- Zur Erinnerung: Operationen auf den Registern sind schneller als Operationen auf Hauptspeicher
- Vergleich der beiden Programme (clientssh1, using time, gcc):
  - ca. 3.9 s mit long
  - ca. 3.9 s mit register
- Keine große Veränderung

Analyse von C Programmen
Vergleich der Hauptprogramme

TECHNISCHE UNIVERSITÄT DARMSTADT

- Reminder: gcc -S funktion_fak.c generiert das Assemblerprogramm (vgl. Webseite)
- Analyse zeigt: die Schleifenvariable *schleife* wird, trotz der Angabe **register** auf dem Stack realisiert.
- Compiler gcc ignoriert die Anweisung
- Aufruf des Compilers: gcc -O1 funktion_fak.c
  - Verschiedene Optimierungen möglich
  - -O1, -O2 gebräuchlich
- Übersetzung mit -O1 liefert folgendes Ergebnis

Analyse eines C Programms mit zwei Funktionen, Hauptprogramm - Schleifenzähler

TECHNISCHE UNIVERSITÄT DARMSTADT

```
...
pushl %ebp
movl %esp, %ebp
pushl %ebx
pushl %ecx
subl $16, %esp
movl $1, %ebx
.L13:
movl $30, (%esp)
call fak_ite
movl $30, (%esp)
call fak_rek
addl $1, %ebx
cmpl $10000000, %ebx # lokale Variable in %ebx
jne .L13
...
```

Analyse von C Programmen
Vergleich der Hauptprogramme

TECHNISCHE UNIVERSITÄT DARMSTADT

- Was bringt die Optimierung mit -O1 für die Laufzeit des Programms
- Zeiten
  - ca. 3.9 s mit long
  - ca. 3.9 s mit register
  - ca. 2.8 s mit -O1
- Weitere Optimierungen möglich, Aufruf mit -O2
  - ca. 0.0 s mit -O2
- Aber Achtung...nicht immer funktioniert ein Programm nach der Optimierung noch...

Analyse von C Programmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für das sogenannte *Programm Profiling* stehen verschiedene Möglichkeiten zur Verfügung (vgl. [BO10, S. 574])
- Im Beispiel: Hauptprogramm und zwei Funktionen (fak_ite und fak_rek)
- Welche Zeit wird im Hauptprogramm bzw. in den beiden Funktionen verbracht?
- gcc (und andere Compiler) können darüber Aussagen liefern
  - gcc -O1 -pg -o funktion_fak funktion_fak.c (-pg ist ein run-time flag)
  - ./funktion_fak (Aufruf des Programms, Programm läuft langsamer (ca. 9.8 s)
  - gprof funktion_fak (wertet die Profile-Datei aus)

Analyse eines C Programms mit zwei Funktionen, Zeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aufsplittung der Zeiten für das Hauptprogramm und die beiden Funktionen

```
1    %   cumulative   self              total
2   time   seconds   seconds    calls  ns/call  name
3  60.63      2.02     2.02  9999999   202.50  fak_rek
4  33.98      3.16     1.14  9999999   113.50  fak_ite
5   5.39      3.34     0.18                    main
```

- Suchen (und finden) von Funktionen, die viel Zeit brauchen $\Rightarrow$ ggf. Optimierungen
  - z. B. Loop-Unrolling
  - oder bessere (parallele) Algorithmen
  - Thread Programmierung

OpenMP
Allgemeines

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bessere (parallele) Algorithmen zu implementieren ist häufig sehr aufwendig
- Thread Programmierung nicht trivial
- Eine andere Lösung: **OpenMP**[1]
  - Sammlung von Compiler-Anweisungen (Direktiven) und Bibliotheksfunktionen
  - Kombination von C, C++ oder Fortran
  - Erreicht wird $\Rightarrow$ threadparalleles Arbeiten auf Rechnersystemen mit gemeinsamen Adressraum
  - Gut geeignet für Multi-Core-Architekturen
- Zur Erinnerung: Thread bzw. Prozess (Unterschiede später) ist ein Betriebssystemkonzept zur Repräsentation eines Programms in der Ausführung

[1] `http://openmp.org/wp/`

OpenMP
Grundkonzept

TECHNISCHE UNIVERSITÄT DARMSTADT

- OpenMP startet als einzelner Thread
- Für Codeabschnitte, die parallel ausgeführt werden können, *forkt*[2] sich das Programm in zusätzliche Threads
- Diese Codeabschnitte werden als parallele Regionen bezeichnet
- Am Schluss wieder zusammenführen (*join*) zu einem einzelnen Thread
- **Fork-Join-Programmiermodell**
- Beispiel

```
1 #include<stdio.h>
2 int main(void) {
3   int i=5;
4     printf("E pu si muove %d\n",i);
5   }
```

[2] verzweigt

OpenMP
Erstes Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- OpenMP ist eine „Programmiersprache“ mit expliziter Parallelität
- Für einen Compiler ist es schwierig Nebenläufigkeit (Parallelität) zu erkennen.
- OpenMP bietet ein **Pragma** Blöcke parallel als Threads auszuführen
- Beispiel

```
1  #include<stdio.h>
2  #include<omp.h>
3
4  int main(void) {
5   int i=5;
6
7   #pragma omp parallel
8   {
9    printf("E pu si muove %d\n",i);
10  }
11 }
```

OpenMP
Zweites Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Threads sind zu identifizieren

```
1   #include <stdio.h>
2   #include <omp.h>
3   int main(void)
4  { int id,i;
5    omp_set_num_threads(4);
6    #pragma omp parallel for private(id)
7       for (i = 0; i < 4; ++i)
8       {  id = omp_get_thread_num();
9       printf("Hello World from thread %d\n", id);
10   #pragma omp barrier
11    if (id == 0) {
12    printf("There are %d threads\n", omp_get_num_threads());
13    }
14   } return 0; }
```

OpenMP
Drittes Beispiel - Berechnung von PI

TECHNISCHE UNIVERSITÄT DARMSTADT

- Parallelisierung von Schleifen

```
1  #include<stdio.h>
2  #include<math.h>
3  #include<omp.h>
4  int main (void) {
5   int i;
6   long num_steps = 1000000000;
7   double x,pi,step,sum=0.0;
8   step = 1.0/(double) num_steps;
9   #pragma omp parallel for private(x) reduction(+:sum)
10  for (i=0; i< num_steps; i++) {
11   x = (i+0.5)*step; sum += 4.0/(1.0+x*x);}
12  pi = step * sum;
13  printf("PI %lf\n",pi);
14 return 0; }
```

OpenMP
Praxis

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einbinden von #include<omp.h>
- Compileraufruf: gcc -fopenmp trapez_omp.c
- Setzen der Umgebungsvariable für die Anzahl der Threads:
  export OMP_NUM_THREADS=2
- Ergebnisse (Laufzeit in Sekunden) der Parallelisierung:
  - Ein Thread (Pentium D[3], ohne Optimierung): 30.8 s
  - Zwei Threads (Pentium D, ohne Optimierung): 15.4 s
  - Vier Threads (Pentium D, ohne Optimierung): 15.5 s
  - Acht Threads (Pentium D, ohne Optimierung): 20.0 s
- Threads bzw. deren Verwaltung kostet offensichtlich Zeit

[3] Die CPU hat zwei Prozessorkerne.

Intel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Intel bietet einen C Compiler an: Intel C++ Compiler (ICC)
- Freier Download unter `http://software.intel.com/en-us/`
- Unterstützung von OpenMP
- Hochoptimierender Compiler, bei Laufzeitoptimierung wichtige Option
- Weitere Softwarepakete:
  - Intel VTune Amplifier XE 2011 for Linux
  - Intel Inspector XE 2011 for Linux

Zusammenfassung und Ausblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- Analyse von C Programmen
- OpenMP
- Intel

Nächste Vorlesung behandelt

- Leistungsbewertung