Xeración e optimización.

Compiladores e interpretes - Práctica 1A

Barreiro Domínguez, Víctor Xesús

Xaneiro 2022

Índice

1. Introdución.

Para abordar o informe de forma breve salientamos que introducimos nos anexos do documento o código empregado.

Do mesmo xeito, comezamos compilando coas opcións *-O0 -static* e analizando o código ensamblador xerado por gcc coa axuda de Compiler Explorar.

Tras isto centrámonos na función que construimos denominada *realizarProducto*, vendo a súa evolución ao aplicar os disntintos niveis de optimización do compilador.

Finalmente avaliaremos o código obxecto, vendo o seu tamaño e tempo de execución.

2. Ensamblador.

Comezamos analizando o código compilado con -O0, concretamente as funcións que realizan as tarefas indicadas: realizarProducto e producto.

Indicamos sobre o código o que estamos a facer para ver que se está realizando o escrito no código en C.

```
producto:     <- INICIO PRODUCTO
.LFB0:
  .cfi_startproc
  endbr64
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register 6
  movss %xmm0, -4(%rbp)
  movss %xmm1, -8(%rbp)
  movq  %rdi, -16(%rbp)
  movss -4(%rbp), %xmm0
  mulss -8(%rbp), %xmm0
  movq  -16(%rbp), %rax
  movss %xmm0, (%rax)
  nop
  popq  %rbp
  .cfi_def_cfa 7, 8
  ret
  .cfi_endproc
.LFE0:
  .size producto, .-producto
  .globl  realizarProducto
  .type realizarProducto, @function
realizarProducto:     <- INICIO REAKUZARPRODUCTO
.LFB1:
  .cfi_startproc
  endbr64
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq  %rsp, %rbp    <-BUCLE VALORES VARIABLES
  .cfi_def_cfa_register 6
  leaq  -4317184(%rsp), %r11
.LPSRL0:
  subq  $4096, %rsp
  orq $0, (%rsp)
  cmpq  %r11, %rsp
  jne .LPSRL0
  subq  $2864, %rsp
  movq  %fs:40, %rax
  movq  %rax, -8(%rbp)
  xorl  %eax, %eax
  movl  $0, -4320028(%rbp)
  jmp .L3
.L6:
  movl  $0, -4320024(%rbp)
  jmp .L4
.L5:
  movl  -4320028(%rbp), %edx
  movl  -4320024(%rbp), %eax
  addl  %edx, %eax
  cvtsi2sdl %eax, %xmm0
  cvtsi2sdl -4320024(%rbp), %xmm2
```

```
55   movsd  .LC0(%rip), %xmm1
56   addsd  %xmm2, %xmm1
57   divsd  %xmm1, %xmm0
58   cvtsd2ss   %xmm0, %xmm0
59   movl   -4320024(%rbp), %eax
60   cltq
61   movl   -4320028(%rbp), %edx
62   movslq   %edx, %rdx
63   imulq $600, %rdx, %rdx
64   addq   %rdx, %rax
65   movss %xmm0, -4320016(%rbp,%rax,4)
66   movl   -4320028(%rbp), %eax
67   subl   -4320024(%rbp), %eax
68   cvtsi2sdl %eax, %xmm0
69   cvtsi2sdl -4320024(%rbp), %xmm2
70   movsd  .LC1(%rip), %xmm1
71   addsd  %xmm2, %xmm1
72   divsd  %xmm1, %xmm0
73   cvtsd2ss   %xmm0, %xmm0
74   movl   -4320024(%rbp), %eax
75   cltq
76   movl   -4320028(%rbp), %edx
77   movslq   %edx, %rdx
78   imulq $600, %rdx, %rdx
79   addq   %rdx, %rax
80   movss %xmm0, -2880016(%rbp,%rax,4)
81   addl   $1, -4320024(%rbp)
82 .L4:     <-BUCLE VALORES VARIABLES INTERNO
83   cmpl   $599, -4320024(%rbp)
84   jle .L5
85   addl   $1, -4320028(%rbp)
86 .L3:
87   cmpl   $599, -4320028(%rbp)
88   jle .L6
89   movl   $0, -4320028(%rbp)     <-BUCLE PRODUCTO MATRICIAL
90   jmp .L7
91 .L12:
92   movl   $0, -4320024(%rbp)     <-BUCLE PRODUCTO VARIABLES 1
93   jmp .L8
94 .L11:
95   pxor   %xmm0, %xmm0
96   movss %xmm0, -4320032(%rbp)
97   movl   $0, -4320020(%rbp)     <-BUCLE PRODUCTO VARIABLES 2
98   jmp .L9
99 .L10:
100  movl   -4320024(%rbp), %eax
101  cltq
102  movl   -4320020(%rbp), %edx
103  movslq   %edx, %rdx
104  imulq $600, %rdx, %rdx
105  addq   %rdx, %rax
106  movss -2880016(%rbp,%rax,4), %xmm0
107  movl   -4320020(%rbp), %eax
108  cltq
109  movl   -4320028(%rbp), %edx
110  movslq   %edx, %rdx
111  imulq $600, %rdx, %rdx
```

```
112   addq   %rdx, %rax
113   movl   -4320016(%rbp,%rax,4), %eax
114   leaq   -4320036(%rbp), %rdx
115   movq   %rdx, %rdi
116   movaps   %xmm0, %xmm1
117   movd   %eax, %xmm0
118   call   producto
119   movss -4320036(%rbp), %xmm0
120   movss -4320032(%rbp), %xmm1
121   addss %xmm1, %xmm0
122   movss %xmm0, -4320032(%rbp)
123   addl   $1, -4320020(%rbp)
124 .L9:
125   cmpl   $599, -4320020(%rbp)
126   jle .L10
127   movl   -4320024(%rbp), %eax
128   cltq
129   movl   -4320028(%rbp), %edx
130   movslq   %edx, %rdx
131   imulq $600, %rdx, %rdx
132   addq   %rdx, %rax
133   movss -4320032(%rbp), %xmm0
134   movss %xmm0, -1440016(%rbp,%rax,4)
135   addl   $1, -4320024(%rbp)
136 .L8:
137   cmpl   $599, -4320024(%rbp)
138   jle .L11
139   addl   $1, -4320028(%rbp)
140 .L7:
141   cmpl   $599, -4320028(%rbp)
142   jle .L12
143   nop
144   movq  -8(%rbp), %rax
145   xorq   %fs:40, %rax
146   je   .L13
147   call   __stack_chk_fail@PLT
148 .L13:
149   leave
150   .cfi_def_cfa 7, 8
151   ret
152   .cfi_endproc
153 .LFE1:
154   .size realizarProducto, .-realizarProducto
155   .section   .rodata
```

2.1. Compilado con -O1

```
0 producto:
1 .LFB23:
2   .cfi_startproc
3   endbr64
4   mulss %xmm1, %xmm0
5   movss %xmm0, (%rdi)
6   ret
```

```
  .cfi_endproc
.LFE23:
  .size producto, .-producto
  .globl  realizarProducto
  .type realizarProducto, @function
realizarProducto:
.LFB24:
  .cfi_startproc
  endbr64
  movl  $600, %edx
  jmp .L3
.L13:
  subl  $1, %edx
  je  .L10
.L3:
  movl  $600, %eax
.L4:
  subl  $1, %eax
  jne .L4
  jmp .L13
.L14:
  subl  $1, %edx
  je  .L7
.L9:
  movl  $600, %eax
.L6:
  subl  $1, %eax
  jne .L6
  jmp .L14
.L7:
  subl  $1, %ecx
  je  .L15
.L5:
  movl  $600, %edx
  jmp .L9
.L15:
  ret
.L10:
  movl  $600, %ecx
  jmp .L5
  .cfi_endproc
.LFE24:
  .size realizarProducto, .-realizarProducto
  .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
  .string "%d\n"
```

Na compilación feita con -O1, permanecen os búcles antes descritos, pero xa non se realiza nin a reserva de memoria, nin se lle asignan valores as variables, nin se realiza o propio produto. Si que se xera o código da función *producto*.

2.2. Compilado con -O2

```
producto:
```

```
.LFB23:
  .cfi_startproc
  endbr64
  mulss %xmm1, %xmm0
  movss %xmm0, (%rdi)
  ret
  .cfi_endproc
.LFE23:
  .size producto, .-producto
  .p2align 4
  .globl  realizarProducto
  .type realizarProducto, @function
realizarProducto:
.LFB24:
  .cfi_startproc
  endbr64
  ret
  .cfi_endproc
.LFE24:
  .size realizarProducto, .-realizarProducto
  .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
  .string "%d f"
```

Ao aplicar as técnicas de optimización o compilador chega a baleirar completamente a función *realizarProducto*. Cháma a atención que si que xera o código obxecto asociado a función *producto*.

2.3. Compilado con -O3.

Non atopamos diferencias co a versión compilada con -O2.

2.4. Compilado con -Os.

Neste último caso non atopamos cambios sobre o código asociado a *realizarProducto* e a *producto* respecto do atopado con -O2. Destacar que si que existen cambios respecto desta versión na función *main* reordenando instrucións e cambiando algunhas destas.

3. Tamaños.

Tamaños (kB)	**-O0**	**-O1**	**-O2**	**-O3**	**-Os**
Executable	871,8	871,8	871,8	871,8	871,8
Ensamblador	5,1	2,6	2,3	2,3	2,1

Vemos como a medida que aplicamos as técnicas de optimización vanse reducindo as liñas de código, o esperado conforme ao visto no apartado anterior.

4. Tempos.

-O0	-O1	-O2	-O3	-Os
9.271241	0.000000	0.000000	0.000001	0.000000

Cadro 1: Tempos en segundos.

De novo vemos o esperado respecto do exposto cando analizamos os códigos en ensamblador, onde a partir da primeira versión practicamente non se fai nada na función que estamos a medir.

5. Código.

```
0  #include <stdio.h>
1  #include <sys/time.h>
2  #define Nmax 600
3  #define ITER 10
4  void producto(float x, float y, float *z) {
5    *z=x*y;
6  }
7
8  void realizarProducto()
9  {
10   float A[Nmax][Nmax], B[Nmax][Nmax], C[Nmax][Nmax], t, r;
11   int i,j,k;
12   for(i=0;i<Nmax;i++) /* Valores de las matrices */
13     for(j=0;j<Nmax;j++) {
14       A[i][j]=(i+j)/(j+1.1);
15       B[i][j]=(i-j)/(j+2.1);
16     }
17   for(i=0;i<Nmax;i++) /* Producto matricial */
18     for(j=0;j<Nmax;j++)
19     {
20       t=0;
21       for (k=0;k<Nmax;k++)
22       {
23         producto(A[i][k],B[k][j],&r);
24         t+=r;
25       }
26       C[i][j]=t;
27     }
28 }
29
30 int main() {
31
32   struct timeval inicio, final;
33   int i;
34
35     gettimeofday(&inicio,NULL);
36     gettimeofday(&final,NULL);
37     double tempo0 = (final.tv_sec-inicio.tv_sec+(final.tv_usec-
       inicio.tv_usec)/1.e6);
```

```
38
39      gettimeofday(&inicio,NULL);
40      for (i = 0; i < ITER; ++i)
41      {
42        realizarProducto();
43      }
44      gettimeofday(&final,NULL);
45      double tempo = (final.tv_sec-inicio.tv_sec+(final.tv_usec-
        inicio.tv_usec)/1.e6) - tempo0;
46
47      printf("%f", tempo);
48
49      return 0;
50  }
```