2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Néstor Rodríguez Vico
Grupo de prácticas: A1
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz*

Sistema operativo utilizado: *Ubuntu 15.10*

Versión de gcc utilizada: *gcc (Ubuntu 5.2.1-22ubuntu2) 5.2.1 20151010*

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz
stepping        : 3
microcode       : 0x1c
cpu MHz         : 2610.449
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid
sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm ida
arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1
avx2 smep bmi2 erms invpcid xsaveopt
bugs            :
bogomips        : 4988.62
clflush size    : 64
cache_alignment       : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz
stepping        : 3
microcode       : 0x1c
cpu MHz         : 2692.773
cache size      : 3072 KB
physical id     : 0
siblings        : 4
```

```
core id         : 0
cpu cores       : 2
apicid          : 1
initial apicid : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid
sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm ida
arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1
avx2 smep bmi2 erms invpcid xsaveopt
bugs            :
bogomips        : 4988.62
clflush size    : 64
cache_alignment       : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz
stepping        : 3
microcode       : 0x1c
cpu MHz         : 2710.351
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 2
initial apicid : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid
sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm ida
arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1
avx2 smep bmi2 erms invpcid xsaveopt
bugs            :
bogomips        : 4988.62
clflush size    : 64
cache_alignment       : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz
stepping        : 3
microcode       : 0x1c
cpu MHz         : 2647.851
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 3
initial apicid : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
```

```
constant_tsc  arch_perfmon  pebs  bts  rep_good  nopl  xtopology  nonstop_tsc  aperfmperf
eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid
sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm ida
arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1
avx2 smep bmi2 erms invpcid xsaveopt
bugs           :
bogomips       : 4988.62
clflush size   : 64
cache_alignment       : 64
address sizes  : 39 bits physical, 48 bits virtual
power management:
```

1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), y para un programa que implemente la multiplicación de matrices (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

   1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

```
struct {
   int a;
   int b;
} s[5000];

main()
{
   ...
   for (ii=0; ii<40000;ii++) {
      X1=0; X2=0;
      for(i=0; i<5000;i++) X1+=2*s[i].a+ii;
      for(i=0; i<5000;i++) X2+=3*s[i].b-ii;
      if (X1<X2)   R[ii]=X1   else   R[ii]=X2;
   }
   ...
}
```

El cambio principal que se puede realizar es la unión de los dos bucles. A ello se le puede sumar un desenrrollado del bucle. En mi caso, no se porqué, pero el tiempo de ejecución no mejora.

A) MULTIPLICACIÓN DE MATRICES:
CÓDIGO FUENTE: `pmm-secuencial.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
```

```
int main(int argc,char **argv) {
   int i, j, k;
   double t1, t2, total;

   //Leer argumento de entrada (no de componentes del vector)
   if (argc<2){
      printf("Falta tamaño de matriz\n");
      exit(-1);
   }

   unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

   double **M1, **M2, **M3;
   //M1 = M2 * M3;
            M1 = (double**) malloc(N*sizeof(double *));
   M2 = (double**) malloc(N*sizeof(double *));
   M3 = (double**) malloc(N*sizeof(double *));

            if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
                        printf("Error en la reserva de espacio para las
matrices\n");
                        exit(-2);
            }

            for (i=0; i<N; i++){
                        M1[i] = (double*) malloc(N*sizeof(double));
      M2[i] = (double*) malloc(N*sizeof(double));
      M3[i] = (double*) malloc(N*sizeof(double));
                        if ( M1[i]==NULL || M2[i]==NULL || M3[i]==NULL ){
                                    printf("Error en la reserva de espacio
para los matrices\n");
                                    exit(-2);
                        }
            }

            //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

            //Inicializar matriz y vectores

   for (j=0; j<N; j++){
                        for (i=0; i<N; i++){
         M1[i][j] = 0;
                                    M2[i][j] = 2;
                                    M3[i][j] = 2;
                        }
            }

   //Medida de tiempo
   t1 = omp_get_wtime();

   //Calcular producto de matriz por matriz M1 = M2 * M3

   for (i=0; i<N; i++){
                        for(j=0; j<N; j++){
                                    for (k=0; k<N; k++){
                                                M1[i][j] = M1[i][j] +
(M2[i][k]*M3[k][j]);
                                    }
                        }
            }
```

```
   //Medida de tiempo
              t2 = omp_get_wtime();
              total = t2 - t1;

   //Imprimir el resultado y el tiempo de ejecución
   printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ M1[0][0]=%8.6f M1[%d][%d]=
%8.6f\n", total,N,M1[0][0],N-1,N-1,M1[N-1][N-1]);

              for (i=0; i<N; i++){
                           free(M1[i]); free(M2[i]); free(M3[i]);
   }

              free(M1); free(M2); free(M3);

   return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Deserrollado del bucle interno. Se reducen las instrucciones de salto

Modificación b) –explicación-: Inicializar las matrices como su traspuesta. Se mejora el acceso a los datos.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc,char **argv) {
   int h, i, j, k;
   double t1, t2, total;
   int suma = 0; int suma2 = 0;

   //Leer argumento de entrada (no de componentes del vector)
   if (argc<2){
      printf("Falta tamaño de matriz\n");
      exit(-1);
   }

   unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

   double **M1, **M2, **M3;
   //M1 = M2 * M3;
              M1 = (double**) malloc(N*sizeof(double *));
   M2 = (double**) malloc(N*sizeof(double *));
   M3 = (double**) malloc(N*sizeof(double *));

              if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
                           printf("Error en la reserva de espacio para las
matrices\n");
                           exit(-2);
              }

              for (i=0; i<N; i++){
                           M1[i] = (double*) malloc(N*sizeof(double));
      M2[i] = (double*) malloc(N*sizeof(double));
```

```
      M3[i] = (double*) malloc(N*sizeof(double));
                        if ( M1[i]==NULL || M2[i]==NULL || M3[i]==NULL ){
                                    printf("Error en la reserva de espacio
para los matrices\n");
                                    exit(-2);
                        }
            }

            //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

            //Inicializar matriz y vectores

   //Cambiamos el acceso a las matrices
   for (i=0; i<N; i++){
                        for (j=0; j<N; j++){
         M1[j][i] = 0;
                                    M2[j][i] = 2;
                                    M3[j][i] = 2;
                        }
            }

   //Medida de tiempo
   t1 = omp_get_wtime();

   //Calcular producto de matriz por matriz M1 = M2 * M3

   int bucle_interno = N/8;
   for (i=0; i<N; i++){
                        for(j=0; j<N; j++){
                                    suma = 0; suma2 = 0;
                                    for (h=0, k=0; h < bucle_interno; ++h,
k+=8){
                                                suma += (M2[i][k]*M3[k]
[j]);
                                                suma += (M2[i]
[k+1]*M3[k+1][j]);
                                                suma += (M2[i]
[k+2]*M3[k+2][j]);
                                                suma += (M2[i]
[k+3]*M3[k+3][j]);
                                                suma2 += (M2[i]
[k+4]*M3[k+4][j]);
                                                suma2 += (M2[i]
[k+5]*M3[k+5][j]);
                                                suma2 += (M2[i]
[k+6]*M3[k+6][j]);
                                                suma2 += (M2[i]
[k+7]*M3[k+7][j]);
                                    }
                                    M1[i][j] = suma + suma2;
                                    for(k=bucle_interno*8; k<N; ++k){
                                                suma += (M2[i][k]*M3[k]
[j]);
                                    }
                                    M1[i][j] = suma + suma2;
                        }
            }

   //Medida de tiempo
              t2 = omp_get_wtime();
              total = t2 - t1;
```

```
   //Imprimir el resultado y el tiempo de ejecución
   printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ M1[0][0]=%8.6f M1[%d][%d]=
%8.6f\n", total,N,M1[0][0],N-1,N-1,M1[N-1][N-1]);

            for (i=0; i<N; i++){
                        free(M1[i]); free(M2[i]); free(M3[i]);
   }

            free(M1); free(M2); free(M3);

   return 0;
}
```

Capturas de pantalla (que muestren que el resultado es correcto):

```
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B4/2$ ./pmm-secuencial 900
Tiempo(seg.): 3.051889507       / Tamaño:900    / M1[0][0]=3600.000000 M1[899][899]=3600.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B4/2$ ./pmmMod 900
Tiempo(seg.): 4.473921622       / Tamaño:900    / M1[0][0]=3600.000000 M1[899][899]=3600.000000
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*3.051889507*
Modificación a+b	4.473921622

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Una vez más, podemos ver que el resultado no mejora.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s	pmm-secuencial-modificado_b.s	pmm-secuencial-modificado_c.s
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */

B) CÓDIGO FIGURA 1:
CÓDIGO FUENTE: `figura1-original.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
```

```
struct {
            int a;
            int b;
} s[5000];

int main(int argc, char** argv) {
            int i, ii, X1, X2;
            int R[40000];
            struct timespec cgt1, cgt2;
    double tiempo;

    for(i=0; i<5000;i++)  {
                        s[i].a = i;
                        s[i].b = i;
    }

            clock_gettime(CLOCK_REALTIME, &cgt1);

            for (ii=0; ii<40000;ii++) {

                        X1=0; X2=0;

                        for(i=0; i<5000;i++){
                                    X1+=2*s[i].a+ii;
                        }
                        for(i=0; i<5000;i++){
                                    X2+=3*s[i].b-ii;
                        }

                        if (X1<X2)
                                    R[ii]=X1;
                        else
                                    R[ii]=X2;
            }

            clock_gettime(CLOCK_REALTIME, &cgt2);
            tiempo = (double) (cgt2.tv_sec - cgt1.tv_sec)+ (double)
((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
            printf("a: %i - b: %i", R[5000], s[4999].a);
            printf(" Tiempo(seg.): %11.9f\n", tiempo);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Union de los bucles. Solo se recorre una vez.

Modificación b) –explicación-: Deserrollado del bucle interno. Se reducen las instrucciones de salto

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1**`-modificado_a.c`**

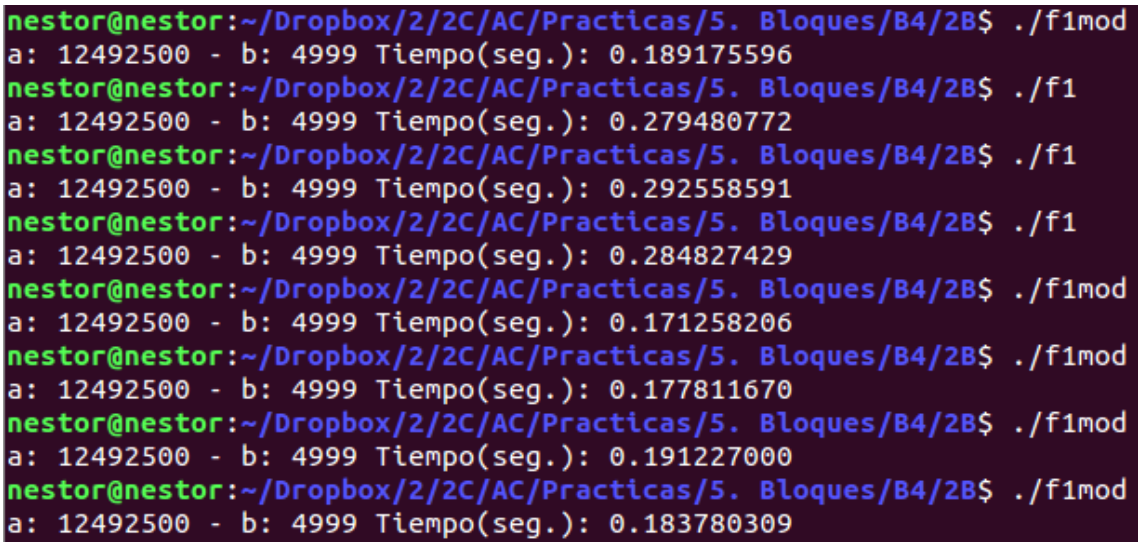
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

struct {
            int a;
            int b;
} s[5000];

int main(int argc, char** argv) {
```

```
            int i, ii, X1, X2;
            int X1a=0, X2a=0;
            int X1b=0, X2b=0;
            int X1c=0, X2c=0;
            int X1d=0, X2d=0;
            int X1e=0, X2e=0;
            int R[40000];
            struct timespec cgt1, cgt2;
    double tiempo;

    for(i=0; i<5000;i++)  {
                        s[i].a = i;
                        s[i].b = i;
    }


            clock_gettime(CLOCK_REALTIME, &cgt1);

            for (ii=0; ii<40000;ii++) {

                        /*X1=0; X2=0;

                        for(i=0; i<5000;i++){
                                    X1+=2*s[i].a+ii;
                                    X2+=3*s[i].b-ii;
                        }*/

                        X1a=0; X2a=0;
                        X1b=0; X2b=0;
                        X1c=0; X2c=0;
                        X1d=0; X2d=0;
                        X1e=0; X2e=0;
                        for(i=0; i<5000;i+=5) {
                                    X1a+=2*s[i].a+ii;
                                    X1b+=2*s[i+1].a+ii;
                                    X1c+=2*s[i+2].a+ii;
                                    X1d+=2*s[i+3].a+ii;
                                    X1e+=2*s[i+4].a+ii;

                                    X2a+=3*s[i].b-ii;
                                    X2b+=3*s[i+1].b-ii;
                                    X2c+=3*s[i+2].b-ii;
                                    X2d+=3*s[i+3].b-ii;
                                    X2e+=3*s[i+4].b-ii;
                        }
                        X1 = X1a+X1b+X1c+X1d+X1e;
                        X2 = X2a+X2b+X2c+X2d+X2e;

                        if (X1<X2)
                                    R[ii]=X1;
                        else
                                    R[ii]=X2;
            }

            clock_gettime(CLOCK_REALTIME, &cgt2);
            tiempo = (double) (cgt2.tv_sec - cgt1.tv_sec)+ (double)
```

```
((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
            printf("a: %i - b: %i", R[5000], s[4999].a);
            printf(" Tiempo(seg.): %11.9f\n", tiempo);

}
```

Capturas de pantalla (que muestren que el resultado es correcto):



1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.292558591*
Modificación a+b	0.171258206

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Se produce una mejora apreciable.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s	pmm-secuencial-modificado_b.s
.file "f1.c" .section .rodata.str1.1,"aMS",@progbits,1 .LC1: .string "a: %i - b: %i" .LC2: .string " Tiempo(seg.): %11.9f\n" .section .text.unlikely,"ax",@progbits .LCOLDB3: .section .text.startup,"ax",@progbits	.file "f1mod.c" .section .rodata.str1.1,"aMS",@progbits,1 .LC1: .string "a: %i - b: %i" .LC2: .string " Tiempo(seg.): %11.9f\n" .section .text.unlikely,"ax",@progbits .LCOLDB3: .section .text.startup,"ax",@progbits

```
.LHOTB3:
                .p2align 4,,15
                .globl          main
                .type           main,
@function
main:
.LFB38:
                .cfi_startproc
                subq            $160072,
%rsp
                .cfi_def_cfa_offset 160080
                movq            %fs:40, %rax
                movq            %rax,
160056(%rsp)
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L2:
                movl            %eax, s(,
%rax,8)
                movl            %eax, s+4(,
%rax,8)
                addq            $1, %rax
                cmpq            $5000, %rax
                jne             .L2
                leaq            16(%rsp),
%rsi
                xorl            %edi, %edi
                call
                clock_gettime
                xorl            %r9d, %r9d
                movl            $s+40000,
%r8d
                .p2align 4,,10
                .p2align 3
.L3:
                movl            %r9d, %edi
                movl            $s, %eax
                xorl            %esi, %esi
                .p2align 4,,10
                .p2align 3
.L4:
                movl            (%rax), %edx
                addq            $8, %rax
                leal            (%rdi,
%rdx,2), %edx
                addl            %edx, %esi
                cmpq            %rax, %r8
                jne             .L4
                movl            $s+4, %eax
                xorl            %ecx, %ecx
                .p2align 4,,10
                .p2align 3
.L5:
                movl            (%rax), %edx
                addq            $8, %rax
                leal            (%rdx,
%rdx,2), %edx
                subl            %edi, %edx
                addl            %edx, %ecx
                cmpq            $s+40004,
%rax
                jne             .L5
                cmpl            %ecx, %esi
                cmovl           %esi, %ecx
                movl            %ecx,
48(%rsp,%r9,4)
                addq            $1, %r9
                cmpq            $40000, %r9
```

```
.LHOTB3:
                .p2align 4,,15
                .globl          main
                .type           main,
@function
main:
.LFB38:
                .cfi_startproc
                pushq           %r15
                .cfi_def_cfa_offset 16
                .cfi_offset 15, -16
                pushq           %r14
                .cfi_def_cfa_offset 24
                .cfi_offset 14, -24
                pushq           %r13
                .cfi_def_cfa_offset 32
                .cfi_offset 13, -32
                pushq           %r12
                .cfi_def_cfa_offset 40
                .cfi_offset 12, -40
                pushq           %rbp
                .cfi_def_cfa_offset 48
                .cfi_offset 6, -48
                pushq           %rbx
                .cfi_def_cfa_offset 56
                .cfi_offset 3, -56
                subq            $160072, %rsp
                .cfi_def_cfa_offset 160128
                movq            %fs:40, %rax
                movq            %rax,
160056(%rsp)
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L2:
                movl            %eax, s(,
%rax,8)
                movl            %eax, s+4(,
%rax,8)
                addq            $1, %rax
                cmpq            $5000, %rax
                jne             .L2
                leaq            16(%rsp),
%rsi
                xorl            %edi, %edi
                xorl            %r15d, %r15d
                movl            $s+40000,
%r14d
                call            clock_gettime
                .p2align 4,,10
                .p2align 3
.L3:
                movl            %r15d, %edx
                movl            $s, %eax
                xorl            %r13d, %r13d
                xorl            %r12d, %r12d
                xorl            %ebp, %ebp
                xorl            %ebx, %ebx
                xorl            %r11d, %r11d
                xorl            %r10d, %r10d
                xorl            %r9d, %r9d
                xorl            %r8d, %r8d
                xorl            %esi, %esi
                xorl            %edi, %edi
                .p2align 4,,10
                .p2align 3
.L4:
                movl            (%rax), %ecx
                addq            $40, %rax
```

```
                jne             .L3
                leaq            32(%rsp),
%rsi
                xorl            %edi, %edi
                call
                clock_gettime
                movq            40(%rsp),
%rax
                subq            24(%rsp),
%rax
                movl            $.LC1, %esi
                pxor            %xmm0, %xmm0
                movl
                s+39992(%rip), %ecx
                movl            20048(%rsp),
%edx
                movl            $1, %edi
                cvtsi2sdq       %rax, %xmm0
                movq            32(%rsp),
%rax
                subq            16(%rsp),
%rax
                movapd          %xmm0, %xmm1
                pxor            %xmm0, %xmm0
                divsd           .LC0(%rip),
%xmm1
                cvtsi2sdq       %rax, %xmm0
                xorl            %eax, %eax
                addsd           %xmm1, %xmm0
                movsd           %xmm0,
8(%rsp)
                call            __printf_chk
                movsd           8(%rsp),
%xmm0
                movl            $.LC2, %esi
                movl            $1, %edi
                movl            $1, %eax
                call            __printf_chk
                xorl            %eax, %eax
                movq
                160056(%rsp), %rsi
                xorq            %fs:40, %rsi
                jne             .L15
                addq            $160072,
%rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 8
                ret
.L15:
                .cfi_restore_state
                call
                __stack_chk_fail
                .cfi_endproc
.LFE38:
                .size           main, .-main
                .section        .text.unlike
ly
.LCOLDE3:
                .section        .text.startu
p
.LHOTE3:
                .comm           s,40000,32
                .section        .rodata.cst8
,"aM",@progbits,8
                .align 8
.LC0:
                .long           0
                .long           1104006501
                .ident          "GCC:
```

```
                leal            (%rdx,
%rcx,2), %ecx
                addl            %ecx, %edi
                movl            -32(%rax),
%ecx
                leal            (%rdx,
%rcx,2), %ecx
                addl            %ecx, %r8d
                movl            -24(%rax),
%ecx
                leal            (%rdx,
%rcx,2), %ecx
                addl            %ecx, %r10d
                movl            -16(%rax),
%ecx
                leal            (%rdx,
%rcx,2), %ecx
                addl            %ecx, %ebx
                movl            -8(%rax),
%ecx
                leal            (%rdx,
%rcx,2), %ecx
                addl            %ecx, %r12d
                movl            -36(%rax),
%ecx
                leal            (%rcx,
%rcx,2), %ecx
                subl            %edx, %ecx
                addl            %ecx, %esi
                movl            -28(%rax),
%ecx
                leal            (%rcx,
%rcx,2), %ecx
                subl            %edx, %ecx
                addl            %ecx, %r9d
                movl            -20(%rax),
%ecx
                leal            (%rcx,
%rcx,2), %ecx
                subl            %edx, %ecx
                addl            %ecx, %r11d
                movl            -12(%rax),
%ecx
                leal            (%rcx,
%rcx,2), %ecx
                subl            %edx, %ecx
                addl            %ecx, %ebp
                movl            -4(%rax),
%ecx
                leal            (%rcx,
%rcx,2), %ecx
                subl            %edx, %ecx
                addl            %ecx, %r13d
                cmpq            %rax, %r14
                jne             .L4
                addl            %r8d, %edi
                addl            %r9d, %esi
                addl            %edi, %r10d
                addl            %esi, %r11d
                addl            %r10d, %ebx
                addl            %r11d, %ebp
                addl            %ebx, %r12d
                addl            %ebp, %r13d
                cmpl            %r13d, %r12d
                cmovl           %r12d, %r13d
                movl            %r13d,
48(%rsp,%r15,4)
                addq            $1, %r15
                cmpq            $40000, %r15
```

```
(Ubuntu 5.2.1-22ubuntu2) 5.2.1 20151010"
                .section        .note.GNU-
stack,"",@progbits
```

```
                jne             .L3
                leaq            32(%rsp),
%rsi
                xorl            %edi, %edi
                call            clock_gettime
                movq            40(%rsp),
%rax
                subq            24(%rsp),
%rax
                movl            $.LC1, %esi
                pxor            %xmm0, %xmm0
                movl
                s+39992(%rip), %ecx
                movl            20048(%rsp),
%edx
                movl            $1, %edi
                cvtsi2sdq       %rax, %xmm0
                movq            32(%rsp),
%rax
                subq            16(%rsp),
%rax
                movapd          %xmm0, %xmm1
                pxor            %xmm0, %xmm0
                divsd           .LC0(%rip),
%xmm1
                cvtsi2sdq       %rax, %xmm0
                xorl            %eax, %eax
                addsd           %xmm1, %xmm0
                movsd           %xmm0,
8(%rsp)
                call            __printf_chk
                movsd           8(%rsp),
%xmm0
                movl            $.LC2, %esi
                movl            $1, %edi
                movl            $1, %eax
                call            __printf_chk
                xorl            %eax, %eax
                movq            160056(%rsp),
%rbx
                xorq            %fs:40, %rbx
                jne             .L13
                addq            $160072, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 56
                popq            %rbx
                .cfi_def_cfa_offset 48
                popq            %rbp
                .cfi_def_cfa_offset 40
                popq            %r12
                .cfi_def_cfa_offset 32
                popq            %r13
                .cfi_def_cfa_offset 24
                popq            %r14
                .cfi_def_cfa_offset 16
                popq            %r15
                .cfi_def_cfa_offset 8
                ret
.L13:
                .cfi_restore_state
                call
                __stack_chk_fail
                .cfi_endproc
.LFE38:
                .size           main, .-main
                .section        .text.unlikel
y
.LCOLDE3:
                .section        .text.startup
```

```
.LHOTE3:
                .comm           s,40000,32
                .section        .rodata.cst8,
"aM",@progbits,8
                .align 8
.LC0:
                .long           0
                .long           1104006501
                .ident          "GCC: (Ubuntu
5.2.1-22ubuntu2) 5.2.1 20151010"
                .section        .note.GNU-
stack,"",@progbits
```

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada `DAXPY` (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
for (i=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CÓDIGO FUENTE: `daxpy.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(int argc,char **argv) {
   unsigned int N,i,j,k, l;
   k=999;

   struct timespec cgt1,cgt2;
   double ncgt; //para tiempo de ejecución
   //Leer argumento de entrada (no de componentes de la matriz)
   if (argc<2) {
      printf("Faltan no componentes del vector\n");
      exit(-1);
   }
   N=atoi(argv[1]);
   int *A,*B,*C;

   A=(int *) malloc(N*sizeof(int));
   B=(int *) malloc(N*sizeof(int));
   C=(int *) malloc(N*sizeof(int));
```

```
   //reservamos memoria dinamica.

   for(i=0; i<N; i++) {
      B[i]=20;
      C[i]=30;
   }
   //inicializamosla matriz y el vector

   clock_gettime(CLOCK_REALTIME,&cgt1);
   for (l=0; l<N; l++)
      A[l]= k*B[l] + C[l];
   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
   (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

   //Imprimir resultado de la suma y el tiempo de ejecución
   printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
   printf("Primer componente del vector resultado [%d] y el ultimo componente del
vector resultado [%d]\n",A[0],A[N-1]);

   free(B);
   free(C);
   free(A);
   //eliminamos la memoria dinamica.
}
```

Tiempos ejec.	-O0	-O2	-O3
	0.004218060	0.002621203	0.002164585

CAPTURAS DE PANTALLA:

```
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B4/2$ ./daxpyO0 1000000
Tiempo(seg.):0.004441512         / Tamaño Vectores:1000000
A[0] = 20010 // A[N-1] = 20010.
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B4/2$ ./daxpyO2 1000000
Tiempo(seg.):0.002187907         / Tamaño Vectores:1000000
A[0] = 20010 // A[N-1] = 20010.
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B4/2$ ./daxpyO3 1000000
Tiempo(seg.):0.002573902         / Tamaño Vectores:1000000
A[0] = 20010 // A[N-1] = 20010.
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR: El código que se saca de la opción -O3 es el más eficiente, pero también es más difícil de entender.

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s

```
        .file   "daxpy.c"
        .section        .rodata
        .align 8
.LC0:
        .string"Faltan no componentes del vector"
        .align 8
.LC2:
        .string"Tiempo(seg.):%11.9f\t / Tama\303\261o Vectores:%u\n"
.LC3:
        .string"A[0] = %d // A[N-1] = %d.\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB2:
        .cfi_startproc
        pushq   %rbp
```

```
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        addq    $-128, %rsp
        movl    %edi, -100(%rbp)
        movq    %rsi, -112(%rbp)
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        movl    $999, -88(%rbp)
        cmpl    $1, -100(%rbp)
        jg      .L2
        movl    $.LC0, %edi
        call    puts
        movl    $-1, %edi
        call    exit
.L2:
        movq    -112(%rbp), %rax
        addq    $8, %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        call    atoi
        movl    %eax, -84(%rbp)
        movl    -84(%rbp), %eax
        salq    $2, %rax
        movq    %rax, %rdi
        call    malloc
        movq    %rax, -80(%rbp)
        movl    -84(%rbp), %eax
        salq    $2, %rax
        movq    %rax, %rdi
        call    malloc
        movq    %rax, -72(%rbp)
        movl    -84(%rbp), %eax
        salq    $2, %rax
        movq    %rax, %rdi
        call    malloc
        movq    %rax, -64(%rbp)
        movl    $0, -96(%rbp)
        jmp     .L3
.L4:
        movl    -96(%rbp), %eax
        leaq    0(,%rax,4), %rdx
        movq    -72(%rbp), %rax
        addq    %rdx, %rax
        movl    $20, (%rax)
        movl    -96(%rbp), %eax
        leaq    0(,%rax,4), %rdx
        movq    -64(%rbp), %rax
        addq    %rdx, %rax
        movl    $30, (%rax)
        addl    $1, -96(%rbp)
.L3:
        movl    -96(%rbp), %eax
        cmpl    -84(%rbp), %eax
        jb      .L4
        leaq    -48(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime
        movl    $0, -92(%rbp)
        jmp     .L5
.L6:
        movl    -92(%rbp), %eax
        leaq    0(,%rax,4), %rdx
        movq    -80(%rbp), %rax
        addq    %rdx, %rax
        movl    -92(%rbp), %edx
        leaq    0(,%rdx,4), %rcx
        movq    -72(%rbp), %rdx
        addq    %rcx, %rdx
        movl    (%rdx), %edx
        imull   -88(%rbp), %edx
        movl    -92(%rbp), %ecx
        leaq    0(,%rcx,4), %rsi
```

```
        movq    -64(%rbp), %rcx
        addq    %rsi, %rcx
        movl    (%rcx), %ecx
        addl    %ecx, %edx
        movl    %edx, (%rax)
        addl    $1, -92(%rbp)
.L5:
        movl    -92(%rbp), %eax
        cmpl    -84(%rbp), %eax
        jb      .L6
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime
        movq    -32(%rbp), %rdx
        movq    -48(%rbp), %rax
        subq    %rax, %rdx
        movq    %rdx, %rax
        pxor    %xmm1, %xmm1
        cvtsi2sdq       %rax, %xmm1
        movq    -24(%rbp), %rdx
        movq    -40(%rbp), %rax
        subq    %rax, %rdx
        movq    %rdx, %rax
        pxor    %xmm0, %xmm0
        cvtsi2sdq       %rax, %xmm0
        movsd   .LC1(%rip), %xmm2
        divsd   %xmm2, %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm0, -56(%rbp)
        movl    -84(%rbp), %edx
        movq    -56(%rbp), %rax
        movl    %edx, %esi
        movq    %rax, -120(%rbp)
        movsd   -120(%rbp), %xmm0
        movl    $.LC2, %edi
        movl    $1, %eax
        call    printf
        movl    -84(%rbp), %eax
        subl    $1, %eax
        movl    %eax, %eax
        leaq    0(,%rax,4), %rdx
        movq    -80(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %edx
        movq    -80(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, %esi
        movl    $.LC3, %edi
        movl    $0, %eax
        call    printf
        movq    -72(%rbp), %rax
        movq    %rax, %rdi
        call    free
        movq    -64(%rbp), %rax
        movq    %rax, %rdi
        call    free
        movq    -80(%rbp), %rax
        movq    %rax, %rdi
        call    free
        movl    $0, %eax
        movq    -8(%rbp), %rsi
        xorq    %fs:40, %rsi
        je      .L8
        call    __stack_chk_fail
.L8:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE2:
        .size   main, .-main
        .section        .rodata
        .align 8
.LC1:
        .long   0
```

```
        .long  1104006501
        .ident "GCC: (Ubuntu 5.2.1-22ubuntu2) 5.2.1 20151010"
        .section       .note.GNU-stack,"",@progbits
```

daxpyO2.s

```
        .file  "daxpy.c"
        .section       .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC0:
        .string"Faltan no componentes del vector"
        .align 8
.LC2:
        .string"Tiempo(seg.):%11.9f\t / Tama\303\261o Vectores:%u\n"
        .section       .rodata.str1.1,"aMS",@progbits,1
.LC3:
        .string"A[0] = %d // A[N-1] = %d.\n"
        .section       .text.unlikely,"ax",@progbits
.LCOLDB4:
        .section       .text.startup,"ax",@progbits
.LHOTB4:
        .p2align 4,,15
        .globl main
        .type  main, @function
main:
.LFB38:
        .cfi_startproc
        pushq  %r14
        .cfi_def_cfa_offset 16
        .cfi_offset 14, -16
        pushq  %r13
        .cfi_def_cfa_offset 24
        .cfi_offset 13, -24
        pushq  %r12
        .cfi_def_cfa_offset 32
        .cfi_offset 12, -32
        pushq  %rbp
        .cfi_def_cfa_offset 40
        .cfi_offset 6, -40
        pushq  %rbx
        .cfi_def_cfa_offset 48
        .cfi_offset 3, -48
        subq   $48, %rsp
        .cfi_def_cfa_offset 96
        movq   %fs:40, %rax
        movq   %rax, 40(%rsp)
        xorl   %eax, %eax
        cmpl   $1, %edi
        jle    .L17
        movq   8(%rsi), %rdi
        movl   $10, %edx
        xorl   %esi, %esi
        call   strtol
        movl   %eax, %r12d
        movq   %rax, %rbp
        movl   %eax, %r13d
        salq   $2, %r12
        movq   %r12, %rdi
        call   malloc
        movq   %r12, %rdi
        movq   %rax, %r14
        call   malloc
        movq   %r12, %rdi
        movq   %rax, %rbx
        call   malloc
        testl  %ebp, %ebp
        movq   %rax, %r12
        je     .L3
        xorl   %eax, %eax
        .p2align 4,,10
        .p2align 3
.L4:
        movl   $20, (%rbx,%rax,4)
        movl   $30, (%r12,%rax,4)
        addq   $1, %rax
```

```
        cmpl    %eax, %r13d
        ja      .L4
        movq    %rsp, %rsi
        xorl    %edi, %edi
        call    clock_gettime
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L6:
        imull   $999, (%rbx,%rax,4), %edx
        addl    (%r12,%rax,4), %edx
        movl    %edx, (%r14,%rax,4)
        addq    $1, %rax
        cmpl    %eax, %r13d
        ja      .L6
.L7:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
        movq    24(%rsp), %rax
        subq    8(%rsp), %rax
        movl    %ebp, %edx
        pxor    %xmm0, %xmm0
        movl    $.LC2, %esi
        pxor    %xmm1, %xmm1
        movl    $1, %edi
        cvtsi2sdq       %rax, %xmm0
        movq    16(%rsp), %rax
        subq    (%rsp), %rax
        cvtsi2sdq       %rax, %xmm1
        movl    $1, %eax
        divsd   .LC1(%rip), %xmm0
        addsd   %xmm1, %xmm0
        call    __printf_chk
        leal    -1(%rbp), %eax
        movl    (%r14), %edx
        movl    $.LC3, %esi
        movl    $1, %edi
        movl    (%r14,%rax,4), %ecx
        xorl    %eax, %eax
        call    __printf_chk
        movq    %rbx, %rdi
        call    free
        movq    %r12, %rdi
        call    free
        movq    %r14, %rdi
        call    free
        xorl    %eax, %eax
        movq    40(%rsp), %rcx
        xorq    %fs:40, %rcx
        jne     .L18
        addq    $48, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 48
        popq    %rbx
        .cfi_def_cfa_offset 40
        popq    %rbp
        .cfi_def_cfa_offset 32
        popq    %r12
        .cfi_def_cfa_offset 24
        popq    %r13
        .cfi_def_cfa_offset 16
        popq    %r14
        .cfi_def_cfa_offset 8
        ret
.L3:
        .cfi_restore_state
        movq    %rsp, %rsi
        xorl    %edi, %edi
        call    clock_gettime
        jmp     .L7
.L18:
        call    __stack_chk_fail
.L17:
        movl    $.LC0, %edi
        call    puts
```

```
        orl     $-1, %edi
        call    exit
        .cfi_endproc
.LFE38:
        .size   main, .-main
        .section        .text.unlikely
.LCOLDE4:
        .section        .text.startup
.LHOTE4:
        .section        .rodata.cst8,"aM",@progbits,8
        .align 8
.LC1:
        .long   0
        .long   1104006501
        .ident  "GCC: (Ubuntu 5.2.1-22ubuntu2) 5.2.1 20151010"
        .section        .note.GNU-stack,"",@progbits
```

daxpy03.s

```
                .file           "daxpy.c"
                .section        .rodata.str1.8,"aMS",@progbits,1
                .align 8
.LC0:
                .string         "Faltan no componentes del vector"
                .align 8
.LC4:
                .string         "Tiempo(seg.):%11.9f\t / Tama\303\261o Vectores:%u\n"
                .section        .rodata.str1.1,"aMS",@progbits,1
.LC5:
                .string         "A[0] = %d // A[N-1] = %d.\n"
                .section        .text.unlikely,"ax",@progbits
.LCOLDB6:
                .section        .text.startup,"ax",@progbits
.LHOTB6:
                .p2align 4,,15
                .globl          main
                .type           main, @function
main:
.LFB38:
                .cfi_startproc
                pushq           %r15
                .cfi_def_cfa_offset 16
                .cfi_offset 15, -16
                pushq           %r14
                .cfi_def_cfa_offset 24
                .cfi_offset 14, -24
                pushq           %r13
                .cfi_def_cfa_offset 32
                .cfi_offset 13, -32
                pushq           %r12
                .cfi_def_cfa_offset 40
                .cfi_offset 12, -40
                pushq           %rbp
                .cfi_def_cfa_offset 48
                .cfi_offset 6, -48
                pushq           %rbx
                .cfi_def_cfa_offset 56
                .cfi_offset 3, -56
                subq            $56, %rsp
                .cfi_def_cfa_offset 112
                movq            %fs:40, %rax
                movq            %rax, 40(%rsp)
                xorl            %eax, %eax
                cmpl            $1, %edi
                jle             .L48
                movq            8(%rsi), %rdi
                movl            $10, %edx
                xorl            %esi, %esi
                call            strtol
```

```
                movl            %eax, %ebp
                movq            %rax, %rbx
                salq            $2, %rbp
                movq            %rbp, %rdi
                call            malloc
                movq            %rbp, %rdi
                movq            %rax, %r14
                call            malloc
                movq            %rbp, %rdi
                movq            %rax, %r12
                call            malloc
                testl           %ebx, %ebx
                movq            %rax, %r13
                je              .L3
                movq            %r12, %rbp
                movl            %ebx, %edx
                andl            $15, %ebp
                shrq            $2, %rbp
                negq            %rbp
                andl            $3, %ebp
                cmpl            %ebx, %ebp
                cmova           %ebx, %ebp
                cmpl            $5, %ebx
                ja              .L49
.L4:
                cmpl            $1, %edx
                movl            $20, (%r12)
                movl            $30, 0(%r13)
                je              .L26
                cmpl            $2, %edx
                movl            $20, 4(%r12)
                movl            $30, 4(%r13)
                je              .L27
                cmpl            $3, %edx
                movl            $20, 8(%r12)
                movl            $30, 8(%r13)
                je              .L28
                cmpl            $5, %edx
                movl            $20, 12(%r12)
                movl            $30, 12(%r13)
                jne             .L29
                movl            $20, 16(%r12)
                movl            $30, 16(%r13)
                movl            $5, %eax
.L6:
                cmpl            %edx, %ebx
                je              .L50
.L5:
                movl            %ebx, %edi
                leal            -1(%rbx), %r15d
                movl            %edx, %ecx
                subl            %edx, %edi
                leal            -4(%rdi), %esi
                movl            %r15d, %r11d
                subl            %edx, %r11d
                shrl            $2, %esi
                addl            $1, %esi
                cmpl            $2, %r11d
                leal            0(,%rsi,4), %r8d
                jbe             .L8
                salq            $2, %rcx
                movdqa          .LC1(%rip), %xmm1
                leaq            (%r12,%rcx), %r10
                xorl            %edx, %edx
                addq            %r13, %rcx
                movdqa          .LC2(%rip), %xmm0
                xorl            %r9d, %r9d
.L9:
                addl            $1, %r9d
```

```
                movaps          %xmm1, (%r10,%rdx)
                movups          %xmm0, (%rcx,%rdx)
                addq            $16, %rdx
                cmpl            %esi, %r9d
                jb              .L9
                addl            %r8d, %eax
                cmpl            %r8d, %edi
                je              .L7
.L8:
                movl            %eax, %edx
                movl            $20, (%r12,%rdx,4)
                movl            $30, 0(%r13,%rdx,4)
                leal            1(%rax), %edx
                cmpl            %edx, %ebx
                jbe             .L7
                addl            $2, %eax
                movl            $20, (%r12,%rdx,4)
                movl            $30, 0(%r13,%rdx,4)
                cmpl            %eax, %ebx
                jbe             .L7
                movl            $20, (%r12,%rax,4)
                movl            $30, 0(%r13,%rax,4)
.L7:
                xorl            %edi, %edi
                movq            %rsp, %rsi
                call            clock_gettime
                cmpl            $4, %ebx
                ja              .L51
                movl            %ebx, %ebp
.L17:
                imull           $999, (%r12), %eax
                addl            0(%r13), %eax
                cmpl            $1, %ebp
                movl            %eax, (%r14)
                je              .L32
                imull           $999, 4(%r12), %eax
                addl            4(%r13), %eax
                cmpl            $2, %ebp
                movl            %eax, 4(%r14)
                je              .L33
                imull           $999, 8(%r12), %eax
                addl            8(%r13), %eax
                cmpl            $4, %ebp
                movl            %eax, 8(%r14)
                jne             .L34
                imull           $999, 12(%r12), %eax
                addl            12(%r13), %eax
                movl            %eax, 12(%r14)
                movl            $4, %eax
.L19:
                cmpl            %ebp, %ebx
                je              .L22
.L18:
                movl            %ebx, %r8d
                movl            %r15d, %esi
                movl            %ebp, %edx
                subl            %ebp, %r8d
                subl            %ebp, %esi
                leal            -4(%r8), %ecx
                shrl            $2, %ecx
                addl            $1, %ecx
                cmpl            $2, %esi
                leal            0(,%rcx,4), %r9d
                jbe             .L21
                salq            $2, %rdx
                xorl            %edi, %edi
                leaq            (%r12,%rdx), %r11
                leaq            0(%r13,%rdx), %r10
                leaq            (%r14,%rdx), %rsi
```

```
                xorl            %edx, %edx
.L13:
                movdqa          (%r11,%rdx), %xmm1
                addl            $1, %edi
                movdqa          %xmm1, %xmm2
                pslld           $4, %xmm2
                movdqa          %xmm2, %xmm0
                pslld           $3, %xmm0
                psubd           %xmm2, %xmm0
                psubd           %xmm1, %xmm0
                movdqa          %xmm0, %xmm1
                pslld           $3, %xmm1
                paddd           %xmm1, %xmm0
                movdqu          (%r10,%rdx), %xmm1
                paddd           %xmm1, %xmm0
                movups          %xmm0, (%rsi,%rdx)
                addq            $16, %rdx
                cmpl            %edi, %ecx
                ja              .L13
                addl            %r9d, %eax
                cmpl            %r9d, %r8d
                je              .L22
.L21:
                movl            %eax, %edx
                imull           $999, (%r12,%rdx,4), %ecx
                addl            0(%r13,%rdx,4), %ecx
                movl            %ecx, (%r14,%rdx,4)
                leal            1(%rax), %edx
                cmpl            %edx, %ebx
                jbe             .L22
                imull           $999, (%r12,%rdx,4), %ecx
                addl            $2, %eax
                addl            0(%r13,%rdx,4), %ecx
                cmpl            %eax, %ebx
                movl            %ecx, (%r14,%rdx,4)
                jbe             .L22
                imull           $999, (%r12,%rax,4), %edx
                addl            0(%r13,%rax,4), %edx
                movl            %edx, (%r14,%rax,4)
.L22:
                leaq            16(%rsp), %rsi
                xorl            %edi, %edi
                call            clock_gettime
                movq            24(%rsp), %rax
                subq            8(%rsp), %rax
                movl            %ebx, %edx
                pxor            %xmm0, %xmm0
                movl            $.LC4, %esi
                pxor            %xmm1, %xmm1
                movl            $1, %edi
                cvtsi2sdq       %rax, %xmm0
                movq            16(%rsp), %rax
                subq            (%rsp), %rax
                cvtsi2sdq       %rax, %xmm1
                movl            $1, %eax
                divsd           .LC3(%rip), %xmm0
                addsd           %xmm1, %xmm0
                call            __printf_chk
                movl            (%r14,%r15,4), %ecx
                movl            (%r14), %edx
                movl            $.LC5, %esi
                movl            $1, %edi
                xorl            %eax, %eax
                call            __printf_chk
                movq            %r12, %rdi
                call            free
                movq            %r13, %rdi
                call            free
                movq            %r14, %rdi
```

```
                call    free
                xorl    %eax, %eax
                movq    40(%rsp), %rbx
                xorq    %fs:40, %rbx
                jne     .L52
                addq    $56, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 56
                popq    %rbx
                .cfi_def_cfa_offset 48
                popq    %rbp
                .cfi_def_cfa_offset 40
                popq    %r12
                .cfi_def_cfa_offset 32
                popq    %r13
                .cfi_def_cfa_offset 24
                popq    %r14
                .cfi_def_cfa_offset 16
                popq    %r15
                .cfi_def_cfa_offset 8
                ret
.L50:
                .cfi_restore_state
                leal    -1(%rbx), %r15d
                jmp     .L7
.L51:
                xorl    %eax, %eax
                testl   %ebp, %ebp
                je      .L18
                jmp     .L17
.L49:
                xorl    %edx, %edx
                xorl    %eax, %eax
                testl   %ebp, %ebp
                je      .L5
                movl    %ebp, %edx
                jmp     .L4
.L34:
                movl    $3, %eax
                jmp     .L19
.L32:
                movl    $1, %eax
                jmp     .L19
.L33:
                movl    $2, %eax
                jmp     .L19
.L28:
                movl    $3, %eax
                jmp     .L6
.L29:
                movl    $4, %eax
                jmp     .L6
.L26:
                movl    $1, %eax
                jmp     .L6
.L27:
                movl    $2, %eax
                jmp     .L6
.L3:
                movq    %rsp, %rsi
                xorl    %edi, %edi
                movl    $4294967295, %r15d
                call    clock_gettime
                jmp     .L22
.L48:
                movl    $.LC0, %edi
                call    puts
                orl     $-1, %edi
                call    exit
```

```
.L52:
                call            __stack_chk_fail
                .cfi_endproc
.LFE38:
                .size           main, .-main
                .section        .text.unlikely
.LCOLDE6:
                .section        .text.startup
.LHOTE6:
                .section        .rodata.cst16,"aM",@progbits,16
                .align 16
.LC1:
                .long           20
                .long           20
                .long           20
                .long           20
                .align 16
.LC2:
                .long           30
                .long           30
                .long           30
                .long           30
                .section        .rodata.cst8,"aM",@progbits,8
                .align 8
.LC3:
                .long           0
                .long           1104006501
                .ident          "GCC: (Ubuntu 5.2.1-22ubuntu2) 5.2.1 20151010"
                .section        .note.GNU-stack,"",@progbits
```