2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Elena Ortiz Moreno
Grupo de prácticas y profesor de prácticas: B2 Niceto Luque
Fecha de entrega: 27/05/21
Fecha evaluación en clase: 28/05/21

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo):

Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

Sistema operativo utilizado: *Ubuntu 18.04.5 LTS*

Versión de gcc utilizada: *gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)*

Volcado de pantalla que muestre lo que devuelve `lscpu` en la máquina en la que ha tomado las medidas:

```
elena@elena97om:/$ lscpu
Arquitectura:                    x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:              Little Endian
CPU(s):                          4
Lista de la(s) CPU(s) en línea:      0-3
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:          2
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                GenuineIntel
Familia de CPU:                  6
Modelo:                          142
Nombre del modelo:               Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Revisión:                        9
CPU MHz:                         799,349
CPU MHz máx.:                    3100,0000
CPU MHz mín.:                    400,0000
BogoMIPS:                        5399.81
Virtualización:                  VT-x
Caché L1d:                       32K
Caché L1i:                       32K
Caché L2:                        256K
Caché L3:                        3072K
CPU(s) del nodo NUMA 0:          0-3
Indicadores:                     fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse
36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perf
mon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave
avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shad
ow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx s
map clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_wind
ow hwp_epp md_clear flush_l1d
```

1. **(a)** Implementar un código secuencial que calcule la multiplicación de dos matrices cuadradas. Utilizar como base el código de suma de vectores de BP0. Los datos se deben generar de forma aleatoria para un número de filas mayor que 8, como en el ejemplo de BP0, se puede usar `drand48()`).

MULTIPLICACIÓN DE MATRICES:

CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

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


#define MAX 1024
int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];


int main(int argc, char** argv){
  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  if(argc < 2) {
    fprintf(stderr,"Falta el tamanio de la matriz\n");
    exit(-1);
  }

  unsigned int n = atoi(argv[1]);
  int i, j;

  if(n > MAX){
    n = MAX;
  }


  for(i = 0; i < n; i++){
    for(j = 0; j < n; j++){
      A[i][j] = j + 2*i;
      B[i][j] = i + 2*j;
      C[i][j] = 0;
    }
  }


  if(n <= 11){
    printf("\nMatriz A:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", A[i][j]);
      }
      printf("\n");
    }
    printf("\nMatriz B:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", B[i][j]);
      }
      printf("\n");
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      for(int k = 0; k < n; k++){
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));

  printf("\nTiempo matriz tamanio %d: %11.9f", n, ncgt);

  if(n <= 11){
      printf("\nMatriz resultado:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", C[i][j]);
      }
      printf("\n");
    }
  }

  return 0;
}
```

(b) Modificar el código (solo el trozo que calcula la multiplicación) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. Incorporar los códigos modificados en el cuaderno.

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-: Desenrrollado de bulce: 4 operaciones por bucle.

Modificación B) –explicación-: Trasponer la matriz B para aprovechar la localidad de los accesos.

…

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de `pmm-secuencial-modificado_A.c`

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


#define MAX 1024
int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];


int main(int argc, char** argv){
  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  if(argc < 2) {
    fprintf(stderr,"Falta el tamanio de la matriz\n");
    exit(-1);
  }

  unsigned int n = atoi(argv[1]);
  int i, j;

  if(n > MAX){
    n = MAX;
  }


  for(i = 0; i < n; i++){
    for(j = 0; j < n; j++){
      A[i][j] = n + (2*j) - i  + 1;
      B[i][j] = n + (2*j) - i  + 1;
      C[i][j] = 0;
    }
  }


  if(n <= 11){
    printf("\nMatriz A:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", A[i][j]);
      }
      printf("\n");
    }
```

```c
    printf("\nMatriz B:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", B[i][j]);
      }
      printf("\n");
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      for(int k = 0; k < n; k += 4){
        C[i][j] += A[i][k] * B[k][j] + A[i][k+1] * B[k+1][j] + A[i][k+2] * B[k+2][j] + A[i][k+3]  * B[k+3][j];
      }
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));

  printf("\nTiempo matriz tamanio %d: %11.9f", n, ncgt);

  if(n <= 11){
      printf("\nMatriz resultado:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", C[i][j]);
      }
      printf("\n");
    }
  }

  return 0;
}
```

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

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer1$ gcc -O2 pmm-secuencial-modificado-a.c -o pmm-secuencial-modificado-a
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer1$ ./pmm-secuencial-modificado-a 500

Tiempo matriz tamanio 500: 0.498274110elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer1$
```

B) Captura de `pmm-secuencial-modificado_B.c`

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


#define MAX 1024
int A[MAX][MAX], B[MAX][MAX], B_T[MAX][MAX], C[MAX][MAX];


int main(int argc, char** argv){
  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  if(argc < 2) {
    fprintf(stderr,"Falta el tamanio de la matriz\n");
    exit(-1);
  }

  unsigned int n = atoi(argv[1]);
  int i, j;

  if(n > MAX){
    n = MAX;
  }


  for(i = 0; i < n; i++){
    for(j = 0; j < n; j++){
      A[i][j] = n + (2*j) - i  + 1;
      B[i][j] = n + (2*j) - i  + 1;
      C[i][j] = 0;
    }
  }


  if(n <= 11){
    printf("\nMatriz A:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", A[i][j]);
      }
      printf("\n");
    }
    printf("\nMatriz B:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", B[i][j]);
      }
      printf("\n");
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for(i = 0; i < n; i++){
    for(j = 0; j < n; j++){
      B_T[i][j] = B[j][i];
    }
  }

  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      for(int k = 0; k < n; k++){
        C[i][j] += A[i][k] * B_T[j][k];
      }
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));

  printf("\nTiempo matriz tamanio %d: %11.9f", n, ncgt);

  if(n <= 11){
      printf("\nMatriz resultado:\n");
    for(i = 0; i < n; i++){
      for(j = 0; j < n; j++){
        printf(" %d ", C[i][j]);
      }
      printf("\n");
    }
  }

  return 0;
}
```

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

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer1$ gcc -O2 pmm-secuencial-modificado-b.c -o pmm-secuencial-modificado-b
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer1$ ./pmm-secuencial-modificado-b 1000

Tiempo matriz tamanio 1000: 0.941819319elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer1$
```

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar N = 500		0.700916746
Modificación A) N = 500	Desenrrollado de bulce: 4 operaciones por bucle.	0.498274110
Modificación B) N=1000	Trasponer la matriz B para aprovechar la localidad de los accesos.	0.941819319
...		

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

Es mejor la modificación B que la A y la no modificada, debido a que solo con trasponer la matriz y por como se realiza la multiplicación de matrices aprovechamos mucho mejor la localidad de los accesos, mientras que en la primera modificación solo nos ahorramos un bucle.

2. **(a)** Usando como base el código de BP0, generar un programa para evaluar un código de la Figura 1. M y N deben ser parámetros de entrada al programa. Los datos se deben generar de forma aleatoria para valores de M y N mayores que 8, como en el ejemplo de BP0.

CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

#define TAM_S 5000
#define TAM_R 40000

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

int R[TAM_R];

int main(){
  int i, ii, X1, X2;

  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  for (i = 0; i < TAM_S; i++ ) {
    s[i].a = i;
    s[i].b = 2*i;
  }


  clock_gettime(CLOCK_REALTIME,&cgt1);

  for ( ii = 0; ii < TAM_R; ii++ ) {
    X1 = 0;
    X2 = 0;

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

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

  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));


  printf("\nVector R: \n");
  printf("R[0]=%d R[39999]=%d\n", R[0], R[TAM_R-1]);
  printf("Tiempo: %11.9f\n", ncgt);

  return 0;
}
```

Figura 1 . Código C++ que suma dos vectores. M y N deben ser parámetros de entrada al programa, usar valores mayores que 1000 en la evaluación.

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

main()
{
   …
   for (ii=0; ii<M;ii++) {
      X1=0; X2=0;
      for(i=0; i<N;i++)  X1+=2*s[i].a+ii;
      for(i=0; i<N;i++)  X2+=3*s[i].b-ii;

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

(b) Modificar el código C (solo el trozo a evaluar) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. En las ejecuciones de evaluación usar valores de N y M mayores que 1000. Incorporar los códigos modificados en el cuader

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-: reducción de bucles y operador ternario.

Modificación B) –explicación-: desenrrollado de bucle y operador ternario.

…

CÓDIGOS FUENTE MODIFICACIONES

A) **Captura** figura1-modificado_A.c

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

#define TAM_S 5000
#define TAM_R 40000

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

int R[TAM_R];

int main(){
  int i, ii, X1, X2;
  int sumatorio;

  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  for (i = 0; i < TAM_S; i++ ) {
    s[i].a = i;
    s[i].b = 2*i;
  }


  clock_gettime(CLOCK_REALTIME,&cgt1);

  for ( ii = 0; ii < TAM_R; ii++ ) {
    X1 = 0;
    X2 = 0;

    for ( i = 0; i < TAM_S; i++ ) {
      X1 += s[i].a;
      X2 += s[i].b;
    }
    sumatorio = TAM_S * ii;

    X1 = (X1*2) + sumatorio;
    X2 = (X2*3) - sumatorio;

    R[ii] = X1 < X2 ? X1 : X2;
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));


  printf("\nVector R: \n");
  printf("R[0]=%d R[39999]=%d\n", R[0], R[TAM_R-1]);
  printf("Tiempo: %11.9f\n", ncgt);

  return 0;
}
```

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

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer2$ gcc figura1-modificado-a.c -o figura1-modificado-a
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer2$ ./figura1-modificado-a

Vector R:
R[0]=24995000 R[39999]=-125010000
Tiempo: 0.674179991
```

B) Captura figura1-`modificado_B.c`

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

#define TAM_S 5000
#define TAM_R 40000

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

int R[TAM_R];

int main(){
  int i, ii, X1, X2;

  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  for (i = 0; i < TAM_S; i++ ) {
    s[i].a = i;
    s[i].b = 2*i;
  }


  clock_gettime(CLOCK_REALTIME,&cgt1);

  for ( ii = 0; ii < TAM_R; ii++ ) {
    X1 = 0;
    X2 = 0;

    for ( i = 0; i < TAM_S; i += 5 ) {
      X1 += 2*s[i].a + ii;
      X1 += 2*s[i+1].a + ii;
      X1 += 2*s[i+2].a + ii;
      X1 += 2*s[i+3].a + ii;
      X1 += 2*s[i+4].a + ii;

      X2 += 3*s[i].b - ii;
      X2 += 3*s[i+1].b - ii;
      X2 += 3*s[i+2].b - ii;
      X2 += 3*s[i+3].b - ii;
      X2 += 3*s[i+4].b - ii;
    }

    R[ii] = X1 < X2 ? X1 : X2;
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));


  printf("\nVector R: \n");
  printf("R[0]=%d R[39999]=%d\n", R[0], R[TAM_R-1]);
  printf("Tiempo: %11.9f\n", ncgt);

  return 0;
}
```

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

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer2$ gcc figura1-modificado-b.c -o figura1-modificado-b
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer2$ ./figura1-modificado-b

Vector R:
R[0]=24995000 R[39999]=-125010000
Tiempo: 0.681195113
```

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*1.197413569*
Modificación A)	reducción de bucles y operador ternario.	0.674179991
Modificación B)	desenrrollado de bucle y operador ternario.	0.681195113
...		

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

En este caso la mejor opción es la a: en ambas modificaciones juntamos todo dentro de un solo bucle for interno, pero funciona mejor llamar al struct el menor número de veces posible acumulando lso resultados que ahorrarnos bucles con múltiples llamadas dentro.

3. 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 que opera con flotantes de doble precisión 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=0;i<N;i++) y[i]= a*x[i] + y[i];
```

Generar los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorporar los códigos al cuaderno de prácticas y destacar las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo `DAXPY.` `N` deben ser parámetro de entrada al programa.

CAPTURA CÓDIGO FUENTE: `daxpy.c`

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

int main (int argc, char **argv){
  struct timespec cgt1, cgt2;     //Medicion de tiempo
  double ncgt;

  const int A = 150;            //Constante arbitraria

  if ( argc < 2 ) {
        fprintf(stderr,"Introduzca un num\n");
        exit(-1);
    }

  int n = atoi(argv[1]);
  int x[n], y[n];

  for (int i = 0; i <= n; i++){
    x[i] = i;
    y[i] = 2*i;
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for (int i = 1;i <= n; i++){
    y[i] = A * x[i] + y[i];
  }

  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));


  printf("\ny[0]: %d, y[%d]: %d\n", y[0], n, y[n]);
  printf("Tiempo: %11.9f\n", ncgt);

  return 0;
}
```

Tiempos ejec. Longitud vectores=1000000	**-O0**	**-Os**	**-O2**	**-O3**
	0.015512014	0.003124218	0.003213841	0.002473006

CAPTURAS DE PANTALLA (que muestren la compilación y que el resultado es correcto)**:**

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ gcc -O0 daxpy.c -o daxpy0 & gcc -O0 daxpy.c -S
[1] 9717
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ gcc -Os daxpy.c -o daxpyS & gcc -Os daxpy.c -S
[2] 9737
[1]   Hecho                   gcc -O0 daxpy.c -o daxpy0
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ gcc -O2 daxpy.c -o daxpy2 & gcc -O2 daxpy.c -S
[3] 9746
[2]   Hecho                   gcc -Os daxpy.c -o daxpyS
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ gcc -O3 daxpy.c -o daxpy3 & gcc -O3 daxpy.c -S
[4] 9771
[3]   Hecho                   gcc -O2 daxpy.c -o daxpy2
```

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ ./daxpy0 1000

y[0]: 0, y[1000]: 152000
Tiempo: 0.000010027
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ ./daxpyS 1000

y[0]: 0, y[1000]: 152000
Tiempo: 0.000001124
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ ./daxpy2 1000

y[0]: 0, y[1000]: 152000
Tiempo: 0.000002250
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP4/ejer3$ ./daxpy3 1000

y[0]: 0, y[1000]: 152000
Tiempo: 0.000001684
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

O0: No tiene ninguna optimización.

Os: Es parecido a O2, pero reduce el tamaño del archivo frente a este.

O2: Se emplean instrucciones más eficientes que las usadas en O0 y se reduce el tamaño de archivo respecto a este.

O3: Se genera un código más complejo con más llamadas a subrutinas y aumenta el tamaño del archivo.

CÓDIGO EN ENSAMBLADOR (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s	daxpyOs.s	daxpyO2.s	daxpyO3.s

jmp .L5
.L6:
movq -128(%rbp), %rax
movl -148(%rbp), %edx
movslq %edx, %rdx
movl (%rax,%rdx,4), %eax
imull -152(%rbp), %eax
movl %eax, %ecx
movq -112(%rbp), %rax
	.L3:
	cmpl %r13d, %eax
	jg .L10
	leal (%rax,%rax), %edx
	movl %eax, (%r15,%rax,4)
	movl %edx, (%r14,%rax,4)
	incq %rax
	jmp .L3
		.L4:
		leal (%rax,%rax), %edx
		movl %eax, (%r14,%rax,4)
		movl %edx, (%rbx,%rax,4)
		addq $1, %rax
		cmpq %rcx, %rax
		jne .L4
			.L9:
			leaq -80(%rbp), %rsi
			xorl %edi, %edi
			call clock_gettime@PLT
			movq -72(%rbp), %rax
			pxor %xmm0, %xmm0
			subq -88(%rbp), %rax
			pxor %xmm1, %xmm1
			movl 0(,%rbx,4), %edx

movl -148(%rbp), %edx
movslq %edx, %rdx
movl (%rax,%rdx,4), %eax
addl %eax, %ecx
movq -112(%rbp), %rax
movl -148(%rbp), %edx
movslq %edx, %rdx
movl %ecx, (%rax,%rdx,4)
addl $1, -148(%rbp)
.L5:
movl -148(%rbp), %eax
cmpl -140(%rbp), %eax
jle .L6
			movl 0(%r13,%r12,4), %r8d
			leaq .LC4(%rip), %rsi
			movl %r14d, %ecx
			movl $1, %edi
			cvtsi2sdq %rax, %xmm0
			movq -80(%rbp), %rax
			subq -96(%rbp), %rax
			cvtsi2sdq %rax, %xmm1
			xorl %eax, %eax
			divsd .LC3(%rip), %xmm0
			addsd %xmm1, %xmm0
			movsd %xmm0, -104(%rbp)
			call __printf_chk@PLT
			movsd -104(%rbp), %xmm0
			leaq .LC5(%rip), %rsi
			movl $1, %edi
			movl $1, %eax
			call __printf_chk@PLT
			xorl %eax, %eax
			movq -56(%rbp), %rbx
			xorq %fs:40, %rbx
			jne .L32
			leaq -40(%rbp), %rsp
			popq %rbx
			popq %r12
			popq %r13
			popq %r14
			popq %r15
			popq %rbp
			.cfi_remember_state
			.cfi_def_cfa 7, 8

			ret
			.L18:
			.cfi_restore_state
			movl $1, -104(%rbp)
			jmp .L5
			.L22:
			movl $2, %esi
			jmp .L11
			.L3:
			leaq -96(%rbp), %rsi
			xorl %edi, %edi
			call clock_gettime@PLT
			jmp .L9
			.L19:
			movl $2, -104(%rbp)
			jmp .L5

4. **(a)** Paralizar con OpenMP en la CPU el código de la multiplicación resultante en el Ejercicio 1.(b). NOTA: usar para generar los valores aleatorios, por ejemplo, `drand48_r()`.

   (b) Calcular la ganancia en prestaciones que se obtiene en atcgrid4 para el máximo número de procesadores físicos con respecto al código inicial no optimizado del Ejercicio 1.(a) para dos tamaños de la matriz.

   (a) MULTIPLICACIÓN DE MATRICES PARALELO:

   CAPTURA CÓDIGO FUENTE: `pmm-paralelo.c`

   (b) RESPUESTA