

2º curso / 2º cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos):

Grupo de prácticas y profesor de prácticas:

Fecha de entrega: 05/06/2021

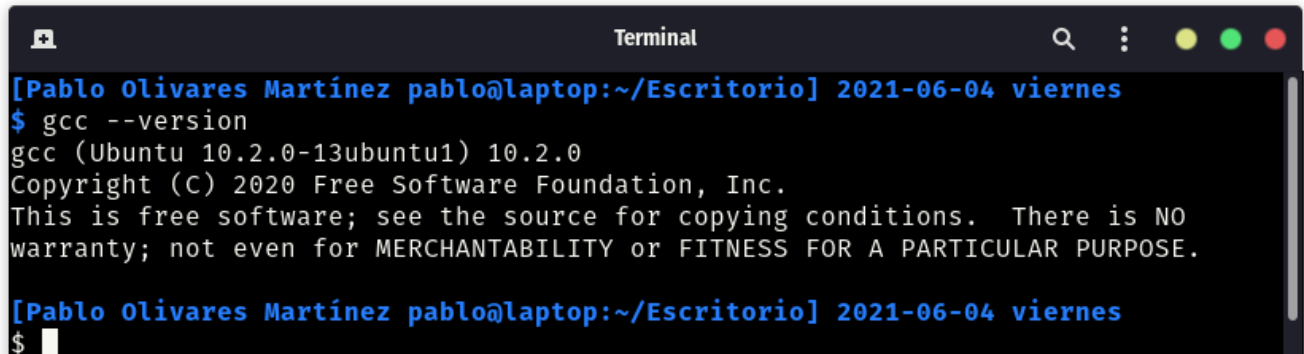
Fecha evaluación en clase: 09/06/2021

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): *(respuesta)*

Sistema operativo utilizado: *Pop! OS 20.10*

Versión de gcc utilizada: 10.2.0



```
Terminal
[Pablo Olivares Martínez pablo@laptop:~/Escritorio] 2021-06-04 viernes
$ gcc --version
gcc (Ubuntu 10.2.0-13ubuntu1) 10.2.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[Pablo Olivares Martínez pablo@laptop:~/Escritorio] 2021-06-04 viernes
$
```

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

```

[Pablo Olivares Martínez pablo@laptop:~/Escritorio] 2021-06-04 viernes
$ lscpu
Arquitectura:                x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes:          Little Endian
Address sizes:                39 bits physical, 48 bits virtual
CPU(s):                       8
Lista de la(s) CPU(s) en línea: 0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:      4
«Socket(s)»:                  1
Modo(s) NUMA:                 1
ID de fabricante:             GenuineIntel
Familia de CPU:                6
Modelo:                       158
Nombre del modelo:            Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Revisión:                     10
CPU MHz:                      2300.000
CPU MHz máx.:                 4000.0000
CPU MHz mín.:                 800.0000
BogoMIPS:                     4599.93
Virtualización:               VT-x
Caché L1d:                    128 KiB
Caché L1i:                    128 KiB
Caché L2:                     1 MiB
Caché L3:                     8 MiB
CPU(s) del nodo NUMA 0:       0-7
Vulnerability Itlb multihit:   KVM: Mitigation: VMX disabled
Vulnerability L1tf:            Mitigation; PTE Inversion; VMX conditional
                                cache flushes, SMT vulnerable
Vulnerability Mds:             Mitigation; Clear CPU buffers; SMT vulnerab
                                le
Vulnerability Meltdown:        Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabl
                                ed via prctl and seccomp
Vulnerability Spectre v1:      Mitigation; usercopy/swapgs barriers and __
                                user pointer sanitization
Vulnerability Spectre v2:      Mitigation; Full generic retpoline, IBPB co
                                nditional, IBRS_FW, STIBP conditional, RSB
                                filling
Vulnerability Srbds:           Mitigation; Microcode
Vulnerability Tsx async abort: Not affected
Indicadores:                   fpu vme de pse tsc msr pae mce cx8 apic sep
                                mtrr pge mca cmov pat pse36 clflush dts ac
                                pi mmx fxsr sse sse2 ss ht tm pbe syscall n
                                x pdpe1gb rdtscp lm constant_tsc art arch_p
                                erfmon pebs bts rep_good nopl xtopology non
                                stop_tsc cpuid aperfperf pni pclmulqdq dte
                                s64 monitor ds_cpl vmx est tm2 ssse3 sdbg f
                                ma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                                movbe popcnt tsc_deadline_timer aes xsave
                                avx f16c rdrand lahf_lm abm 3dnowprefetch c
                                puid_fault epb invpcid_single pti ssbd ibrs
                                ibpb stibp tpr_shadow vnmi flexpriority ep
                                t vpid ept_ad fsgsbase tsc_adjust bmi1 avx2
                                smep bmi2 erms invpcid mpx rdseed adx smap
                                clflushopt intel_pt xsaveopt xsavec xgetbv
                                1 xsaves dtherm ida arat pln pts hwp hwp_no
                                tify hwp_act_window hwp_epp md_clear flush_
                                lld

```

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

```

int main(int argc, char **argv) {
    int i, j;
    double ncgt;
    struct timespec cgt1, cgt2;

    if (argc < 2) {
        printf("Falta tamaño de las matrices\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

#ifdef VECTOR_GLOBAL
    if (N > MAX) N = MAX;
#endif

#ifdef VECTOR_DYNAMIC
    double *M2, *MF, **M1;
    M2 = (double **)malloc(N * sizeof(double *));
    MF = (double **)malloc(N * sizeof(double *));
    M1 = (double **)malloc(N * sizeof(double *));

    // Reservamos memoria ahora para las componentes de la matriz
    for (i = 0; i < N; i++) {
        M1[i] = (double *)malloc(N * sizeof(double));
        M2[i] = (double *)malloc(N * sizeof(double));
        MF[i] = (double *)malloc(N * sizeof(double));
    }
    if ((M2 == NULL) || (MF == NULL) || (M1 == NULL)) {
        printf("No hay suficiente espacio para los vectores \n");
        exit(-2);
    }
#endif

    // Inicializamos la matriz y los vectores
    srand48(time(NULL));
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            M1[i][j] = drand48();
            M2[i][j] = drand48();
            MF[i][j] = 0;
        }
    }

    // Inicializamos la primera variable de tiempo
    clock_gettime(CLOCK_REALTIME, &cgt1);

```

```

// Calculamos el producto
for (i = 0; i < N; i++)
    for (j = 0; j <= i; j++)
        MF[i][j] += M1[i][j] * M2[j][i];

clock_gettime(CLOCK_REALTIME, &cgt2);
ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) + (double) ((cgt2.tv_nsec - cgt1.tv_nsec) / 1000000000.0);

// Para tamaños pequeños (hasta N=10), imprimimos todas las componentes del vector
// y el tiempo de ejecución
if (N <= 10) {
    printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n", N, ncgt);

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            printf("MF[%i][%i] = %f\n", i, j, MF[i][j]);
        }
    }
}

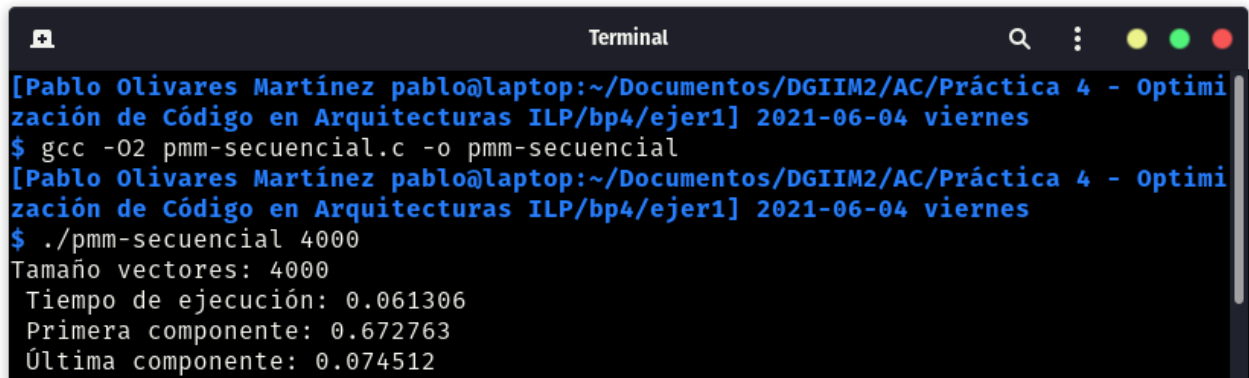
// Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última componente
else {
    printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Última componente: %f\n",
           N, ncgt, MF[0][0], MF[N - 1][N - 1]);
}

#ifdef VECTOR_DYNAMIC
    for (i = 0; i < N; i++) {
        free(M1[i]);
        free(M2[i]);
        free(MF[i]);
    }
    free(M1);
    free(M2);
    free(MF);
#endif

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.



```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer1] 2021-06-04 viernes
$ gcc -O2 pmm-secuencial.c -o pmm-secuencial
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer1] 2021-06-04 viernes
$ ./pmm-secuencial 4000
Tamaño vectores: 4000
Tiempo de ejecución: 0.061306
Primera componente: 0.672763
Última componente: 0.074512

```

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación–: En la modificación A he aprovechado la localidad de accesos creando una matriz 2 transpuesta. Para ello he requerido de código extra que supondrá una ventaja en el cálculo del producto de matrices.

Modificación B) –explicación–: Aquí he usado el desenrollado de bucles para mejorar la velocidad de ejecución rompiendo secuencias de instrucciones independientes.

CÓDIGOS FUENTE MODIFICACIONES**A) Captura de pmm-secuencial-modificado_A.c**

```

// Localidad de Accesos
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        M2_t[i][j] = M2[i][j];
    }
}

// Inicializamos la primera variable de tiempo
clock_gettime(CLOCK_REALTIME, &cg1);

// Calculamos el producto
for (i = 0; i < N; i++) {
    for (j = 0; j <= i; j++) {
        MF[i][j] += M1[i][j] * M2_t[i][j];
    }
}

```

B) Captura de pmm-secuencial-modificado_B.c

```
// Calculamos el producto desenrollando los bucles
for (i = 0; i < N-4; i+=4) {
    for (j = 0; j <= i; j++) {
        MF[i][j] += M1[i][j] * M2_t[i][j];
        MF[i][j+1] += M1[i][j] * M2_t[i][j+1];
        MF[i][j+2] += M1[i][j] * M2_t[i][j+2];
        MF[i][j+3] += M1[i][j] * M2_t[i][j+3];
    }
}

// Calculamos los restantes
for (; i < N; i++)
    for (j = 0; j < N; j++)
        MF[i][j] += M1[i][j] * M2_t[i][j];
```

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

```
Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer1] 2021-06-04 viernes
$ ./pmm-secuencial 10000
Tamaño vectores: 10000
Tiempo de ejecución: 0.474538
Primera componente: 0.301093
Última componente: 0.000922
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer1] 2021-06-04 viernes
$ ./pmm-secuencial-modificado_A 10000
Tamaño vectores: 10000
Tiempo de ejecución: 0.077847
Primera componente: 0.041930
Última componente: 0.084673
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer1] 2021-06-04 viernes
$ ./pmm-secuencial-modificado_B 10000
Tamaño vectores: 10000
Tiempo de ejecución: 0.029471
Primera componente: 0.013348
Última componente: 0.379014
$ gcc -O2 pmm-secuencial-modificado_B.c -o pmm-secuencial-modificado_B
```

TIEMPOS CON N=10.000:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	Ninguna	0,4745s
Modificación A)	Localidad de accesos	0,0778s
Modificación B)	A) + Desenrollado de bucles	0,0294s

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

Como se puede observar, conseguimos una notable mejora de rendimiento tras cada optimización añadida. Principalmente se observa un gran salto de rendimiento aprovechando la localidad de accesos.

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


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

struct S {
    int a;
    int b;
};

int main(int argc, char **argv) {
    int i, j, k, ii, X1, X2;

    struct timespec cgt1, cgt2;
    double ncgt;

    if (argc < 3) {
        printf("Faltan nº componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    unsigned int M = atoi(argv[2]);

    struct S s[N];
    int R[N];

    // Inicializar vectores
    if (N < 9) {
        for (i = 0; i < N; i++) {
            s[i].a = N * 0.1 + i * 0.1;
            s[i].b = N * 0.1 - i * 0.1;
        }
    } else {
        srand(time(NULL));
        for (i = 0; i < N; i++) {
            s[i].a = rand();
            s[i].b = rand();
        }
    }

    clock_gettime(CLOCK_REALTIME, &cgt1);
```



```

clock_gettime(CLOCK_REALTIME, &cgt1);

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;
}

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

printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\t/ R[0](%d) / / R[%d](%d) /\n / S[0].a %d S[0].b %d S[N-1].a %d S[N-1].b %d\n",
        ncgt, N, R[0], M - 1, R[M - 1], s[0].a, s[0].b, s[N - 1].a, s[N - 1].b);

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 cuaderno.

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación–: En la primera modificación aprovecho la localidad de accesos y reduzco el número de operaciones que el procesador debe realizar. Además también reduzco el número de bucles del programa, lo cual debe traducirse en una mejora de rendimiento.

Modificación B) –explicación–: En la segunda modificación he hecho uso de funciones y recursos implementados en C los cuales ya vienen optimizados.

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
for (ii = 0; ii < M; ii++) {
    X1 = 0; X2 = 0;
    for (i = 0; i < N-4; i+=4) {
        X1 += 2 * s[i].a * s[i+1].a * s[i+2].a * s[i+3].a + 4*ii;
        X2 += 3 * s[i].b * s[i+1].b * s[i+2].b * s[i+3].b - 4*ii;
    }
    for ( ; i < N; i++) {
        X1 += 2 * s[i].a * s[i+1].a * s[i+2].a * s[i+3].a + 4*ii;
        X2 += 3 * s[i].b * s[i+1].b * s[i+2].b * s[i+3].b - 4*ii;
    }

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

B) Captura figura1-modificado_B.c

```
for (ii = 0; ii < M; ii++) {
    X1 = 0; X2 = 0;
    for (i = 0; i < N-4; i+=4) {
        X1 += 2 * s[i].a * s[i+1].a * s[i+2].a * s[i+3].a + 4*ii;
        X2 += 3 * s[i].b * s[i+1].b * s[i+2].b * s[i+3].b - 4*ii;
    }
    for ( ; i < N; i++) {
        X1 += 2 * s[i].a * s[i+1].a * s[i+2].a * s[i+3].a + 4*ii;
        X2 += 3 * s[i].b * s[i+1].b * s[i+2].b * s[i+3].b - 4*ii;
    }

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

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

```

[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$ gcc -O2 figura1-original.c -o figura1-original
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$ gcc -O2 figura1-modificado_A.c -o figura1-modificado_A
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$ gcc -O2 figura1-modificado_B.c -o figura1-modificado_B
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$ ./figura1-original 5000 5000
Tiempo:0.068342206 / Tamaño Vectores:5000 / R[0](-952988250)/ / R[4999](-927993250)/
/ S[0].a 766066161 S[0].b 461486994 S[N-1].a 824946572 S[N-1] 1382437306
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$ ./figura1-modificado_A 5000 5000
Tiempo:0.035505862 / Tamaño Vectores:5000 / R[0](-1881459224)/ / R[4999](-1856404236)/
/ S[0].a 24042606 S[0].b 422173805 S[N-1].a 1610125154 S[N-1] 917816070
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$ ./figura1-modificado_B 5000 5000
Tiempo:0.035282842 / Tamaño Vectores:5000 / R[0](-1851477674)/ / R[4999](-1826422686)/
/ S[0].a 1258947859 S[0].b 2008385850 S[N-1].a 807399149 S[N-1] 1647979738
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer2] 2021-06-05 sábado
$

```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	-	0,0683s
Modificación A)	Localidad de accesos y reducción de operaciones	0,0355s
Modificación B)	Uso de recursos optimizados	0,0352s

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

Como vemos, el grueso de la mejora se encuentra en la primera modificación, pues es donde hemos aplicado más modificaciones. Por otro lado, la modificación B, aunque no suponga una gran mejora, produce una optimización del código que reduce su tiempo de ejecución.

- 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 (*Double precision- real Alpha X Plus 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 acarrearán. 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

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

void daxpy(int *y, int *x, int a, unsigned n, struct timespec *cgt1, struct timespec *cgt2) {
    clock_gettime(CLOCK_REALTIME, cgt1);

    unsigned i;

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

    clock_gettime(CLOCK_REALTIME, cgt2);
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "ERROR: Falta tamaño del vector y constante\n");
        exit(1);
    }

    unsigned n = strtoul(argv[1], NULL, 10);
    int a = strtoul(argv[2], NULL, 10);
    int *y, *x;

    y = (int *)malloc(n * sizeof(int));
    x = (int *)malloc(n * sizeof(int));
    unsigned i;

    srand(time(NULL));
    for (i = 0; i < n; i++) {
        y[i] = rand();
        x[i] = rand();
    }

    struct timespec cgt1, cgt2;
    double ncgt;

    daxpy(y, x, a, n, &cgt1, &cgt2);

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

    printf("y[0] = %i, y[%i] = %i\n", y[0], n - 1, y[n - 1]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    free(y);
    free(x);

    return 0;
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
---------------	-----	-----	-----	-----

Longitud vectores=500000000	1,1465s	0,3390s	0,3451s	0,2724s
--	---------	---------	---------	---------

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

```
Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ gcc -O0 daxpy.c -o daxpy_00
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ gcc -O1 daxpy.c -o daxpy_01
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ gcc -O2 daxpy.c -o daxpy_02
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ gcc -O3 daxpy.c -o daxpy_03
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ gcc -Os daxpy.c -o daxpy_0s
```

```
Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ ./daxpy_00 500000000 50000
y[0] = -2013705261, y[499999999] = 1946909684

Tiempo (seg.) = 1.146503316
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ ./daxpy_01 500000000 50000
y[0] = 996360141, y[499999999] = -1727569991

Tiempo (seg.) = 0.334326168
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ ./daxpy_02 500000000 50000
y[0] = -1850299322, y[499999999] = -1374416226

Tiempo (seg.) = 0.345188574
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ ./daxpy_03 500000000 50000
y[0] = 515415715, y[499999999] = 258957789

Tiempo (seg.) = 0.272448791
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$ ./daxpy_0s 500000000 50000
y[0] = 747278794, y[499999999] = 959959561

Tiempo (seg.) = 0.339035808
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer3] 2021-06-05 sábado
$
```


COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Como se puede apreciar, la mejora de tiempo de ejecución viene de la mano de una gran reducción de líneas de código por parte del compilador y/o aplicando notables cambios en este. Por ejemplo, usando O0 usamos direcciones relativas a pila y en O2 se usan los registros de la arquitectura, por eso hay una gran mejora usando O2 en comparación con O0. O3 desenrolla bucles (entre otras optimizaciones), por lo que mejora el rendimiento tal y como hemos visto en el seminario, aunque ocupe más líneas de código, pues reduce el número de saltos. Finalmente, Os se centra en la optimización del tamaño del binario, pero aplicando todas las optimizaciones posibles de O2. Por ello ambas obtienen tiempos tan parecidos pero siendo O2 ligeramente más rápido.

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)

daxpy00.s	daxpy0s.s	daxpy02.s	daxpy03.s
<pre> .file "daxpy.c" .text .globl daxpy .type daxpy, @function daxpy: .LFB6: .cfi_startproc endbr64 pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp 6 .cfi_def_cfa_register subq \$64, %rsp movq %rdi, - 24(%rbp) movq %rsi, - 32(%rbp) movl %edx, - 36(%rbp) movl %ecx, - 40(%rbp) movq %r8, - 48(%rbp) movq %r9, - 56(%rbp) movq -48(%rbp), %rax movq %rax, %rsi movl %0, %edi call clock_gettime@PLT movl \$0, -4(%rbp) jmp .L2: .L3: movl -4(%rbp), %eax leaq 0(,%rax,4), %rdx movq -24(%rbp), %rax addq %rdx, %rax movl (%rax), %ecx movl -4(%rbp), %eax leaq 0(,%rax,4), %rdx movq -32(%rbp), %rax addq %rdx, %rax movl (%rax), %eax imull -36(%rbp), %eax movl %eax, %edx movl -4(%rbp), %eax leaq 0(,%rax,4), %rsi </pre>	<pre> .file "daxpy.c" .text .globl daxpy .type daxpy, @function daxpy: .LFB24: .cfi_startproc endbr64 pushq %r14 .cfi_def_cfa_offset 16 .cfi_offset 14, -16 movq %rsi, %r14 movq %r8, %rsi pushq %r13 .cfi_def_cfa_offset 24 .cfi_offset 13, -24 movl %edx, %r13d pushq %r12 .cfi_def_cfa_offset 32 .cfi_offset 12, -32 movl %ecx, %r12d pushq %rbp .cfi_def_cfa_offset 40 .cfi_offset 6, -40 movq %r9, %rbp pushq %rbx .cfi_def_cfa_offset 48 .cfi_offset 3, -48 movq %rdi, %rbx xorl %edi, %edi call clock_gettime@PLT movq %rdi, %rbx xorl %edi, %edi call clock_gettime@PLT xorl %eax, %eax .L2: cmpl %eax, %r12d jbe .L6: movl (%r14,%rax,4), %edx imull %r13d, %edx addl %edx, (%rbx, %rax,4) incq %rax jmp .L2: .L6: popq %rbx .cfi_def_cfa_offset 40 movq %rbp, %rsi xorl %edi, %edi popq %rbp .cfi_def_cfa_offset 32 popq %r12 .cfi_def_cfa_offset 24 </pre>	<pre> .file "daxpy.c" .text .p2align 4 .globl daxpy .type daxpy, @function daxpy: .LFB39: .cfi_startproc endbr64 pushq %r14 .cfi_def_cfa_offset 16 .cfi_offset 14, -16 movq %r9, %r14 pushq %r13 .cfi_def_cfa_offset 24 .cfi_offset 13, -24 movl %edx, %r13d pushq %r12 .cfi_def_cfa_offset 32 .cfi_offset 12, -32 movl %ecx, %r12d pushq %rbp .cfi_def_cfa_offset 40 .cfi_offset 6, -40 movq %rsi, %rbp movq %r8, %rsi pushq %rbx .cfi_def_cfa_offset 48 .cfi_offset 3, -48 movq %rdi, %rbx xorl %edi, %edi call clock_gettime@PLT testl %ebp, %ebp je .L2 movl %ebp, %ecx xorl %eax, %eax .p2align 4,10 .p2align 3 .L3: movl (%r12,%rax,4), %edx imull %r13d, %edx addl %edx, (%rbx, %rax,4) addq \$1, %rax cmpq %rax, %rcx jne .L3: .L2: popq %rbx .cfi_def_cfa_offset 40 movq %r14, %rsi popq %rbp .cfi_def_cfa_offset 32 xorl %edi, %edi popq %r12 .cfi_def_cfa_offset 24 popq %r13 .cfi_def_cfa_offset 16 popq %r14 </pre>	<pre> .file "daxpy.c" .text .p2align 4 .globl daxpy .type daxpy, @function daxpy: .LFB39: .cfi_startproc endbr64 pushq %r14 .cfi_def_cfa_offset 16 .cfi_offset 14, -16 movq %r9, %r14 pushq %r13 .cfi_def_cfa_offset 24 .cfi_offset 13, -24 movl %edx, %r13d pushq %r12 .cfi_def_cfa_offset 32 .cfi_offset 12, -32 movl %ecx, %r12d pushq %rbp .cfi_def_cfa_offset 40 .cfi_offset 6, -40 movq %rsi, %rbp movq %r8, %rsi pushq %rbx .cfi_def_cfa_offset 48 .cfi_offset 3, -48 movq %rdi, %rbx xorl %edi, %edi call clock_gettime@PLT testl %r12d, %r12d je .L2 leaq 4(%rbp), %rdx movq %rbx, %rax subq %rdx, %rax cmpq \$8, %rax jbe .L3 leal -1(%r12), %eax cmpl \$2, %eax jbe .L3 movd %r13d, %xmm4 movl %r12d, %edx xorl %eax, %eax pshufd \$0, %xmm4, %xmm2 shrl \$2, %edx movdqa %xmm2, %xmm3 salq \$4, %rdx psrlq \$32, %xmm3 .p2align 4,10 .p2align 3 .L4: movdqu 0(%rbp,%rax), %xmm0 movdqu 0(%rbp,%rax), %xmm1 psrlq \$32, %xmm0 </pre>

<pre> %rax movq -24(%rbp), addq %rsi, %rax addl %ecx, %edx movl %edx, (%rax) addl \$1, -4(%rbp) .L2: movl -4(%rbp), %eax cmpl -40(%rbp), %eax jb .L3 movq -56(%rbp), %rax movq %rax, %rsi movl \$0, %edi call clock_gettime@PLT nop leave .cfi_def_cfa 7, 8 ret .cfi_endproc .LFE6: .size daxpy, .- daxpy .section .rodata ta .align 8 .LC0: .string "ERROR: Falta tama\303\261o del vector y constante\n" .LC2: .string "y[0] = %i, y[%i] = %i\n" .LC3: .string "\nTiempo (seg.) = %11.9f\n" .text .globl main .type main, @function main: .LFB7: .cfi_startproc endbr64 pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq %rbx subq \$120, %rsp .cfi_offset 3, -24 movl %edi, - 116(%rbp) movq %rsi, - 128(%rbp) movq %fs:40, %rax movq %rax, - 24(%rbp) xorl %eax, %eax cmpl \$2, - 116(%rbp) jg .L5 movq %rax, %rcx movl \$44, %edx movl \$1, %esi leaq .LC0(%rip), %rdi call fwrite@PLT movl \$1, %edi call exit@PLT .L5: movq -128(%rbp), %rax addq \$8, %rax movq (%rax), %rax movl \$10, %edx movl \$0, %esi movq %rax, %rdi call strtol@PLT </pre>	<pre> popq %r13 .cfi_def_cfa_offset 16 popq %r14 .cfi_def_cfa_offset 8 jmp clock_gettime@PLT .cfi_endproc .LFE24: .size daxpy, .- daxpy .section .rodata ta.str1.1, "ams", @progbits, 1 .LC0: .string "ERROR: Falta tama\303\261o del vector y constante\n" .LC2: .string "y[0] = %i, y[%i] = %i\n" .LC3: .string "\nTiempo (seg.) = %11.9f\n" .section .text .startup, "ax", @progbits .globl main .type main, @function main: .LFB25: .cfi_startproc endbr64 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 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 \$72, %rsp .cfi_def_cfa_offset 128 movq %fs:40, %rax movq %rax, 56(%rsp) xorl %eax, %eax cmpl \$2, %edi jg .L8 movq %stderr(%rip), %rsi leaq .LC0(%rip), %rdi call fputs@PLT movl \$1, %edi call exit@PLT .L8: movq 8(%rsi), %rdi movq %rsi, %rbp movl \$10, %edx xorl %esi, %esi xorl %r13d, %r13d call strtol@PLT movq 16(%rbp), %rdi movl \$10, %edx xorl %esi, %esi movq %rax, %rbx movl %eax, %r15d call strtol@PLT movl %ebx, %r12d </pre>	<pre> .cfi_def_cfa_offset 8 jmp clock_gettime@PLT .cfi_endproc .LFE39: .size daxpy, .-daxpy .section .rodata .str1.8, "ams", @progbits, 1 .align 8 .LC0: .string "ERROR: Falta tama\303\261o del vector y constante\n" .section .rodata .str1.1, "ams", @progbits, 1 .LC2: .string "y[0] = %i, y[%i] = %i\n" .LC3: .string "\nTiempo (seg.) = %11.9f\n" .section .text.s .startup, "ax", @progbits .p2align 4 .globl main .type main, @function main: .LFB40: .cfi_startproc endbr64 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 \$72, %rsp .cfi_def_cfa_offset 128 movq %fs:40, %rax movq %rax, 56(%rsp) xorl %eax, %eax cmpl \$2, %edi jle .L17 movq 8(%rsi), %rdi movq %rsi, %rbx movl \$10, %edx xorl %esi, %esi call strtol@PLT movq 16(%rbx), %rdi xorl %esi, %esi movl \$10, %edx movq %rax, %r14 call strtol@PLT movl %r14d, %r13d leaq 0(%r13,4), %r12 movq %rax, %r15 movq %r12, %rdi call malloc@PLT movq %r12, %rdi movq %rax, %rbp call malloc@PLT xorl %edi, %edi movq %rax, %r12 call time@PLT movq %rax, %rdi call srand@PLT testl %r14d, %r14d je .L12 xorl %ebx, %ebx .p2align 4,,10 .p2align 3 .L13: call rand@PLT movl %eax, 0(%rbp, </pre>	<pre> pmuludq %xmm2, %xmm1 pmuludq %xmm3, %xmm0 pshufd \$8, %xmm1, %xmm1 pshufd \$8, %xmm0, %xmm0 punpckldq %xmm0, %xmm1 movdqu (%rbx,%rax), %xmm0 paddb %xmm1, %xmm0 movups %xmm0, (%rbx, %rax) addq \$16, %rax cmpq %rdx, %rax jne .L4 movl %r12d, %eax andl \$-4, %eax testb \$3, %r12b je .L2 movl %eax, %edx movl 0(%rbp, %rdx,4), %ecx imull %r13d, %ecx addl %ecx, (%rbx, %rdx,4) leal 1(%rax), %edx cmpl %edx, %r12d jbe .L2 movl 0(%rbp, %rdx,4), %ecx addl \$2, %eax imull %r13d, %ecx addl %ecx, (%rbx, %rdx,4) cmpl %eax, %r12d jbe .L2 imull 0(%rbp, %rax,4), %r13d addl %r13d, (%rbx, %rax,4) .L2: popq %rbx .cfi_restore_state .cfi_def_cfa_offset 40 movq %r14, %rsi popq %rbp .cfi_def_cfa_offset 32 xorl %edi, %edi popq %r12 .cfi_def_cfa_offset 24 popq %r13 .cfi_def_cfa_offset 16 popq %r14 .cfi_def_cfa_offset 8 jmp clock_gettime@PLT .p2align 4,,10 .p2align 3 .L3: .cfi_restore_state xorl %eax, %eax .p2align 4,,10 .p2align 3 .L6: movl 0(%rbp, %rax,4), %edx imull %r13d, %edx addl %edx, (%rbx, %rax,4) addq \$1, %rax cmpq %r12, %rax jne .L6 jmp .L2 .cfi_endproc .LFE39: .size daxpy, .-daxpy .section .rodat a.str1.8, "ams", @progbits, 1 .align 8 .LC0: .string "ERROR: Falta tama\303\261o del vector y constante\n" .section .rodat a.str1.1, "ams", @progbits, 1 .LC2: </pre>
--	---	---	---

<pre> movl %eax, - 96(%rbp) movq -128(%rbp), %rax addq \$16, %rax movq (%rax), %rax movl \$10, %edx movl \$0, %esi movq %rax, %rdi call strtol@PLT movl %eax, - 92(%rbp) movl -96(%rbp), %eax salq \$2, %rax movq %rax, %rdi call malloc@PLT movq %rax, - 88(%rbp) movl -96(%rbp), %eax salq \$2, %rax movq %rax, %rdi call malloc@PLT movq %rax, - 80(%rbp) movl \$0, %edi call time@PLT movl %eax, %edi call srand@PLT movl \$0, - 100(%rbp) jmp .L6 .L7: movl -100(%rbp), %eax leaq 0(,%rax,4), %rdx movq -88(%rbp), %rax leaq (%rdx,%rax), %rbx call rand@PLT movl %eax, (%rbx) movl -100(%rbp), %eax leaq 0(,%rax,4), %rdx movq -80(%rbp), %rax leaq (%rdx,%rax), %rbx call rand@PLT movl %eax, (%rbx) addl \$1, - 100(%rbp) .L6: movl -100(%rbp), %eax cmpl -96(%rbp), %eax jb .L7 leaq -48(%rbp), %r8 leaq -64(%rbp), %rdi movl -96(%rbp), %ecx movl -92(%rbp), %edx movq -80(%rbp), %rsi movq -88(%rbp), %rax movq %r8, %r9 movq %rdi, %r8 movq %rax, %rdi call daxpy movq -48(%rbp), %rax movq -64(%rbp), %rdx subq %rdx, %rax pxor %xmm1, %xmm1 cvttsi2sdq %rax, %ymm1 movq -40(%rbp), </pre>	<pre> salq \$2, %r12 movq %rax, %r14 movq %r12, %rdi call malloc@PLT movq %r12, %rdi movq %rax, %rbp call malloc@PLT xorl %edi, %edi movq %rax, %r12 call time@PLT movq %rax, %rdi call srand@PLT .L9: cmpl %r13d, %r15d jbe .L13 call rand@PLT movl %eax, 0(%rbp, %r13,4) call rand@PLT movl %eax, (%r12,%r13,4) incq %r13 jmp .L9 .L13: leaq 40(%rsp), %r9 leaq 24(%rsp), %r8 movl %ebx, %ecx movl %r14d, %edx movq %r12, %rsi movq %rbp, %rdi call daxpy movq 48(%rsp), %rax subq 32(%rsp), %rax leaq .LC2(%rip), %rsi cvttsi2sdq %rax, %ymm0 movq 40(%rsp), %rax subq 24(%rsp), %rax divsd .LC1(%rip), %ymm0 cvttsi2sdq %rax, %ymm1 leal -1(%rbx), %eax movl 0(%rbp), %edx movl \$1, %edi movl 0(%rbp, %rax,4), %r8d movq %rax, %rcx xorl %eax, %eax addsd %xmm1, %xmm0 movsd %xmm0, 8(%rsp) call __printf_chk@PLT movsd 8(%rsp), %ymm0 leaq .LC3(%rip), %rsi movb \$1, %al movl \$1, %edi call __printf_chk@PLT movq %rbp, %rdi call free@PLT movq %r12, %rdi call free@PLT movq 56(%rsp), %rax subq %fs:40, %rax je .L11 call __stack_chk_fail@PLT .L11: addq \$72, %rsp .cfi_def_cfa_offset 56 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 48 popq %rbp </pre>	<pre> %rbx,4) call rand@PLT movl %eax, (%r12,%rbx,4) addq \$1, %rbx cmpq %r13, %rbx jne .L13 .L12: leaq 32(%rsp), %r9 leaq 16(%rsp), %r8 movl %r14d, %ecx movl %r15d, %edx movq %r12, %rsi movq %rbp, %rdi call daxpy pxor %xmm0, %xmm0 pxor %xmm1, %xmm1 movl 0(%rbp), %edx movq 40(%rsp), %rax subq 24(%rsp), %rax movl \$1, %edi leaq .LC2(%rip), %rsi cvttsi2sdq %rax, %ymm0 movq 32(%rsp), %rax subq 16(%rsp), %rax divsd .LC1(%rip), %ymm0 cvttsi2sdq %rax, %ymm1 leal -1(%r14), %eax movl 0(%rbp,%rax,4), %r8d movq %rax, %rcx xorl %eax, %eax addsd %xmm1, %xmm0 movsd %xmm0, 8(%rsp) call __printf_chk@PLT movsd 8(%rsp), %xmm0 leaq .LC3(%rip), %rsi movl \$1, %eax call __printf_chk@PLT movq %rbp, %rdi call free@PLT movq %r12, %rdi call free@PLT movq 56(%rsp), %rax subq %fs:40, %rax jne .L18 addq \$72, %rsp .cfi_restore_state .cfi_def_cfa_offset 56 xorl %eax, %eax 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 .L17: .cfi_restore_state movq stderr(%rip), %rcx movl \$44, %edx movl \$1, %esi leaq .LC0(%rip), %rdi call fwrite@PLT movl \$1, %edi call exit@PLT .L18: call __stack_chk_fail@PLT .cfi_endproc .LFE40: </pre>	<pre> .string "y[0] = %i, y[%i] = %i\n" .LC3: .string "\nTiempo (seg.) = %11.9f\n" .section .text startup, "ax", @progbits .p2align 4 .globl main .type main, @function main: .LFB40: .cfi_startproc endbr64 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 \$72, %rsp .cfi_def_cfa_offset 128 movq %fs:40, %rax movq %rax, 56(%rsp) xorl %eax, %eax cmpl \$2, %edi jle .L28 movq 8(%rsi), %rdi movq %rsi, %rbx movl \$10, %edx xorl %esi, %esi call strtol@PLT movq 16(%rbx), %rdi xorl %esi, %esi movl \$10, %edx movq %rax, %r14 call strtol@PLT movl %r14d, %r13d leaq 0(,%r13,4), %r12 movq %rax, %r15 movq %r12, %rdi call malloc@PLT movq %r12, %rdi movq %rax, %rbp call malloc@PLT xorl %edi, %edi movq %rax, %r12 call time@PLT movq %rax, %rdi call srand@PLT testl %r14d, %r14d je .L23 xorl %ebx, %ebx .p2align 4,10 .p2align 3 .L24: call rand@PLT movl %eax, 0(%rbp, %rbx,4) call rand@PLT movl %eax, (%r12,%rbx,4) addq \$1, %rbx cmpq %rbx, %r13 jne .L24 .L23: leaq 32(%rsp), %r9 leaq 16(%rsp), %r8 movl %r14d, %ecx movl %r15d, %edx movq %r12, %rsi movq %rbp, %rdi </pre>
---	--	--	--

<pre> %rax movq -56(%rbp), %rdx subq %rdx, %rax pxor %xmm0, %xmm0 cvtsi2sdq %rax, %xmm0 movsd .LC1(%rip), %xmm2 divsd %xmm2, %xmm0 addsd %xmm1, %xmm0 movsd %xmm0, - 72(%rbp) movl -96(%rbp), %eax subl \$1, %eax movl %eax, %eax leaq 0(%rax,4), %rdx movq -88(%rbp), %rax addq %rdx, %rax movl (%rax), %edx movl -96(%rbp), %eax leal -1(%rax), %esi movq -88(%rbp), %rax movl (%rax), %eax movl %edx, %ecx movl %esi, %edx movl %eax, %esi leaq .LC2(%rip), %rdi movl \$0, %eax call printf@PLT movq -72(%rbp), %rax movq %rax, %xmm0 leaq .LC3(%rip), %rdi movl \$1, %eax call printf@PLT movq -88(%rbp), %rax movq %rax, %rdi call free@PLT movq -80(%rbp), %rax movq %rax, %rdi call free@PLT movl \$0, %eax movq -24(%rbp), %rbx subq %fs:40, %rbx je .L9 call __stack_chk_fail@PLT .L9: movq -8(%rbp), %rbx leave .cfi_def_cfa 7, 8 ret .cfi_endproc .LFE7: .ta .size main, .-main .section .rodata .align 8 .LC1: .long 0 .long 1104006501 .ident "GCC: (Ubuntu 10.2.0-13ubuntu1) 10.2.0" .section .note .GNU-stack,"",@progbits .section .note .gnu.property,"a" .align 8 .long 1f - 0f .long 4f - 1f .long 5 0: .string "GNU" 1: </pre>	<pre> .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 .cfi_endproc .LFE25: .ta .size main, .-main .section .rodata .align 8 .LC1: .long 0 .long 1104006501 .ident "GCC: (Ubuntu 10.2.0-13ubuntu1) 10.2.0" .section .note .GNU-stack,"",@progbits .section .note .gnu.property,"a" .align 8 .long 1f - 0f .long 4f - 1f .long 5 0: .string "GNU" 1: .align 8 .long 0xc0000002 .long 3f - 2f 2: .long 0x3 3: .align 8 4: </pre>	<pre> .size main, .-main .section .rodata .align 8 .LC1: .long 0 .long 1104006501 .ident "GCC: (Ubuntu 10.2.0-13ubuntu1) 10.2.0" .section .note.G NU-stack,"",@progbits .section .note.g nu.property,"a" .align 8 .long 1f - 0f .long 4f - 1f .long 5 0: .string "GNU" 1: .align 8 .long 0xc0000002 .long 3f - 2f 2: .long 0x3 3: .align 8 4: </pre>	<pre> call daxpy, %xmm0, %xmm0 pxor %xmm1, %xmm1 movl 0(%rbp), %edx movq 40(%rsp), %rax subq 24(%rsp), %rax movl \$1, %edi leaq .LC2(%rip), %rsi cvtsi2sdq %rax, %xmm0 movq 32(%rsp), %rax subq 16(%rsp), %rax divsd .LC1(%rip), %xmm0 cvtsi2sdq %rax, %xmm1 leal -1(%r14), %eax movl 0(%rbp, %rax,4), %r8d movq %rax, %rcx xorl %eax, %eax addsd %xmm1, %xmm0 movsd %xmm0, 8(%rsp) call __printf_chk@PLT movsd 8(%rsp), %xmm0 movl \$1, %edi leaq .LC3(%rip), %rsi movl \$1, %eax call __printf_chk@PLT movq %rbp, %rdi call free@PLT movq %r12, %rdi call free@PLT movq 56(%rsp), %rax subq %fs:40, %rax jne .L29 addq \$72, %rsp .cfi_restore_state .cfi_def_cfa_offset 56 xorl %eax, %eax 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 .L28: .cfi_restore_state movq stderr(%rip), %rcx movl \$44, %edx movl \$1, %esi leaq .LC0(%rip), %rdi call fwrite@PLT movl \$1, %edi call exit@PLT .L29: call __stack_chk_fail@PLT .cfi_endproc .LFE40: .ta .size main, .-main .section .rodata .align 8 .LC1: .long 0 .long 1104006501 .ident "GCC: (Ubuntu 10.2.0-13ubuntu1) 10.2.0" .section .note. GNU-stack,"",@progbits .section .note. gnu.property,"a" .align 8 </pre>
--	--	--	---

<pre> .align 8 .long 0xc0000002 .long 3f - 2f 2: .long 0x3 3: .align 8 4: </pre>			<pre> .long 1f - 0f .long 4f - 1f .long 5 0: .string "GNU" 1: .align 8 .long 0xc0000002 .long 3f - 2f 2: .long 0x3 3: .align 8 4: </pre>
---	--	--	--

4. (a) Paralelizar 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`

```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer4] 2021-06-05 sábado
$ gcc -O2 -fopenmp pmm-paralelo.c -o pmm-paralelo
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 4 - Optimización de Código en Arquitecturas ILP/bp4/ejer4] 2021-06-05 sábado

```

```

// Inicializamos la primera variable de tiempo
t1 = omp_get_wtime();

// Calculamos el producto desenrollando los bucles
#pragma omp parallel for private(j)
for (i = 0; i < N-4; i+=4) {
    for (j = 0; j <= i; j++) {
        MF[i][j] += M1[i][j] * M2_t[i][j];
        MF[i][j+1] += M1[i][j] * M2_t[i][j+1];
        MF[i][j+2] += M1[i][j] * M2_t[i][j+2];
        MF[i][j+3] += M1[i][j] * M2_t[i][j+3];
    }
}

// Calculamos los restantes
#pragma omp parallel for private(j)
for (int k = i; k < N; k++)
    for (j = 0; j < N; j++)
        MF[k][j] += M1[k][j] * M2_t[k][j];

t2 = omp_get_wtime();
tf = t2 - t1;

```

(b) RESPUESTA

```
e1estudiante21@atcgrid:~/bp4

[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp4] 2021-06-05 sábado
$ srun -p ac4 -n1 --hint=nomultithread ./pmm-paralelo 10000
Tamaño vectores: 10000
Tiempo de ejecución: 0.049895
Primera componente: 0.221958
Última componente: 0.000000
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp4] 2021-06-05 sábado
$ srun -p ac4 -n1 --hint=nomultithread ./pmm-paralelo 20000
Tamaño vectores: 20000
Tiempo de ejecución: 0.275219
Primera componente: 0.507001
Última componente: 0.000000
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp4] 2021-06-05 sábado
$
```

```
e1estudiante21@atcgrid:~/bp4

[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp4] 2021-06-05 sábado
$ srun -p ac4 -n1 --hint=nomultithread ./pmm-secuencial 10000
Tamaño vectores: 10000
Tiempo de ejecución: 0.600829
Primera componente: 0.137666
Última componente: 0.595389
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp4] 2021-06-05 sábado
$ srun -p ac4 -n1 --hint=nomultithread ./pmm-secuencial 20000
Tamaño vectores: 20000
Tiempo de ejecución: 3.788470
Primera componente: 0.708571
Última componente: 0.356970
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp4] 2021-06-05 sábado
$
```

GANANCIA = T_SECUENCIAL / T_PARALELO

GANANCIA 10000: 0,600829s / 0,049895s = 12.0418

GANANCIA 20000: 3,78847s / 0,275219s = 13.7652