

BP4RomanAlvarezCarlos.pdf



BrokenQuagga



Arquitectura de Computadores



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



WUOLAH

Fran será lo que quiera.



#NoTeApuntesAWuolah

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



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

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Carlos Roman Alvarez

Grupo de prácticas y profesor de prácticas: A2 – Niceto Rafael Luque Sola

Fecha de entrega: 28/05

Fecha evaluación en clase: 29/05

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) i5-4210U CPU @ 1.70GHz

Sistema operativo utilizado: *Linux Ubuntu 20.10*

Versión de gcc utilizada: 10.2.0

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

```
carlos@carlos-HP-15-Notebook-PC:~$ lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPU(s): 32-bit, 64-bit
Orden de los bytes:    Little Endian
Address sizes:        39 bits physical, 48 bits virtual
CPU(s):               4
Lista de la(s) CPU(s) en línea: 0-3
Número 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:               69
Nombre del modelo:   Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
Revisión:             1
CPU MHz:              1047.502
CPU MHz máx.:        2700.0000
CPU MHz min.:        800.0000
BogoIPs:              4789.07
Caché L1d:             64 KiB
Caché L1i:             64 KiB
Caché L2:              512 KiB
Caché L3:              3 MiB
CPU(s) del nodo NUMA 0: 0-3
Vulnerability Itlb multihit: KVM: Mitigation: VMX unsupported
Vulnerability L1tf:      Mitigation: PTE Inversion
Vulnerability L1tf:      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/swaps 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
```



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 const * argv[])
{
    if (argc < 2)
    {
        fprintf(stderr, "Error falta recibir por parametro un numero.\n");
        exit(-1);
    }

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

    int ***m_r, **a, **b;
    b = (int **)malloc(n * sizeof(int *)); // malloc necesita el tamaño en bytes
    a = (int **)malloc(n * sizeof(int *));
    m_r = (int **)malloc(n * sizeof(int *));

    if ((a == NULL) || (b == NULL) || (m_r == NULL))
    {
        printf("No hay suficiente espacio para los vectores o matriz \n");
        exit(-2);
    }

    for (int i = 0; i < n; ++i)
    {
        m_r[i] = (int *)malloc(n * sizeof(int));
        a[i] = (int *)malloc(n * sizeof(int));
        b[i] = (int *)malloc(n * sizeof(int));

        if (m_r[i] == NULL || a[i] == NULL || b[i] == NULL)
        {
            printf("No hay suficiente espacio para la matriz en la columna %d\n", i);
            exit(-2);
        }
    }

    //Inicializacion
    srand(time(NULL));
    for (int i=0; i<n; ++i)
    {
        for (int e=0; e<n; ++e)
        {
            b[i][e] = e + 1;
            a[i][e] = e + 2;
            m_r[i][e] = 0;
        }
    }

    //Calculo
    double t_ini = omp_get_wtime();
    for (int i = 0; i < n; ++i)
    {
        for (int e = 0; e < n; ++e)
        {
            for (int k = 0; k < n; ++k)
                m_r[i][e] += a[i][k] * b[k][e];
        }
    }

    double tiempo = omp_get_wtime() - t_ini;

    //Resultado
    printf("Tiempo (seg):%11.9f\n", tiempo);
}
```

- (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-: He desenrollado el bucle de 4 en 4

Modificación B) –explicación-: He cambiado los valores de “k” y de “e”, invirtiendo el bucle, para estar más cerca de las posiciones de memoria

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de pmm-secuencial-modificado_A.c

```

int main(int argc, char const * argv[])
{
    if (argc < 2)
    {
        fprintf(stderr, "Error falta recibir por parametro un numero.\n");
        exit(-1);
    }

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

    int **m_r, **a, **b;
    b = (int **)malloc(n * sizeof(int *)); // malloc necesita el tamaño en bytes
    a = (int **)malloc(n * sizeof(int *));
    m_r = (int **)malloc(n * sizeof(int *));

    if ((a == NULL) || (b == NULL) || (m_r == NULL))
    {
        printf("No hay suficiente espacio para los vectores o matriz \n");
        exit(-2);
    }

    for (int i = 0; i < n; ++i)
    {
        m_r[i] = (int *)malloc(n * sizeof(int));
        a[i] = (int *)malloc(n * sizeof(int));
        b[i] = (int *)malloc(n * sizeof(int));

        if (m_r[i] == NULL || a[i] == NULL || b[i] == NULL)
        {
            printf("No hay suficiente espacio para la matriz en la columna %d\n", i);
            exit(-2);
        }
    }

    //Inicilizacion
    srand(time(NULL));
    for (int i=0; i<n; ++i)
    {
        for (int e=0; e<n; ++e)
        {
            b[i][e] = e + 1;
            a[i][e] = e + 2;
            m_r[i][e] = 0;
        }
    }

    //Calculo
    double t_ini = omp_get_wtime();
    for (int i = 0; i < n; ++i)
    {
        for (int e = 0; e < n; ++e)
        {
            for (int k = 0; k < n; ++k)
            {
                m_r[i][e] += a[i][k] * b[k][e];
                m_r[i][k+1] += a[i][e] * b[e][k+1];
                m_r[i][k+2] += a[i][e] * b[e][k+2];
                m_r[i][k+3] += a[i][e] * b[e][k+3];
            }
        }
    }

    double tiempo = omp_get_wtime() - t_ini;

    //Resultado
    printf("Tiempo (seg):%11.9f\n", tiempo);
}

```

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

```

carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./pmm_secuencial 51
Tiempo (seg):0.000378652
m_r[0] = 1377
m_r[50] = 70227
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./pmm-secuencial-A 51
Tiempo (seg):0.000179719
m_r[0] = 1377
m_r[50] = 70227

```

B) ...

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		0.0004543824
Modificación A)	Desenrolado de bucle	0.000378652
Modificación B)	Intercambio de "k" y "e"	0.000179719
...		

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

Como se puede observar la diferencia de tiempo de la modificación final a la original es muy grande, puesto que al hacer que el bucle de menos vueltas reduciendo $n/4$, siendo n el tamaño, ejemplo el de 51, $51/4$ es casi 13, de tal forma que nos ahorramos 38 vueltas que dar, esto implica muchas menos comprobaciones e incrementos.

Otro dato es que intercambiar "k" y "e", hace que las posiciones de memoria de las matrices están más próximas, por lo que el tiempo de acceso a una posición de memoria disminuye bastante

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

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
struct
{
    int a, b;
} s[SIZE_ST];

int main(int argc, char const * argv[])
{
    int i, e, x1, x2;
    int R[SIZE_V];

    // Inicialización
    srand(time(NULL));
    for (i = 0; i < SIZE_ST; ++i)
    {
        s[i].a = rand() % 10;
        s[i].b = rand() % 10;
    }

    // Calculo
    double t_ini = omp_get_wtime();
    for (i = 0; i < SIZE_V; ++i)
    {
        x1 = 0;
        x2 = 0;
        for (e = 0; e < SIZE_ST; ++e)
        {
            x1 += 2 * s[e].a + i;
        }
        for (e = 0; e < SIZE_ST; ++e)
        {
            x2 += 3 * s[e].b - i;
        }

        if (x1 < x2)
            R[i] = x1;
        else
            R[i] = x2;
    }

    double tiempo = omp_get_wtime() - t_ini;
    printf("Tiempo: %11.9f\n", tiempo);
    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-: Modifico el struct para que sea lineal acceder a los datos, de tal forma que tendrá un bucle para cada dato, a y b.

Modificación B) –explicación-: Sustituyo las multiplicaciones por operadores de desplazamiento, haciendo las multiplicaciones mas rápidas. El condicional if else lo he reemplazado por otro de tal forma que creo que ahorra algo de tiempo en la comprobación y almacenamiento del valor.

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
struct
{
    int a[SIZE_ST], b[SIZE_ST];
} s;
int a[SIZE_ST], b[SIZE_ST];

int main(int argc, char const *argv[])
{
    int i, e, x1, x2;
    int R[SIZE_V];

    //inicialización
    srand(time(NULL));
    for (i = 0; i < SIZE_ST; ++i)
    {
        s.a[i] = rand() % 10;
    }
    for (i = 0; i < SIZE_ST; ++i)
    {
        s.b[i] = rand() % 10;
    }

    //calculo
    double t_ini = omp_get_wtime();
    for (i = 0; i < SIZE_V; ++i)
    {
        x1 = 0;
        x2 = 0;

        for (e = 0; e < SIZE_ST; ++e)
        {
            x1 += ((s.a[e]) << 1) + i;
        }

        for (e = 0; e < SIZE_ST; ++e)
        {
            x2 += ((s.b[e] << 1) + s.b[e]) - i;
        }

        R[i] = (x1 < x2) ? x1 : x2;
    }

    double tiempo = omp_get_wtime() - t_ini;
    printf("Tiempo: %11.9f\n", tiempo);
    return 0;
}
```

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

```
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./figural-original 1000
Tiempo: 0.000002901
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./figural-modificado 1000
Tiempo: 0.000000281
```

B) ...

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		0.0000031911
Modificación A)	Modificacion del struct	0.000002901
Modificación B)	Modificacion de las operaciones * por << y modificacion del if else	0.000000281
...		

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

Como se puede ver en el resultado final se reduce bastante, debido que nos ahorraremos las multiplicaciones por operadores de desplazamiento, al hacerse secuencial primero todos los valores de a y luego todos los valores de b y reduciendo bastante el if else, ahorrando ciclos de reloj.

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 (*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 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

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
extern double *x, *y, a;
extern unsigned int N;

unsigned int N = 0;
double a = 0.0;
double *x;
double *y;

int main(int argc, char const * argv[])
{
    if (argc < 3)
    {
        fprintf(stderr, "Error falta recibir por parametro el tamaño y un numero.\n");
        exit(-1);
    }

    // Guardo los valores que recibe por parametro
    N = (unsigned int)atoi(argv[1]);
    a = atof(argv[2]);

    // Inicialización de los vectores x e y
    x = (double *)malloc(N * sizeof(double));
    y = (double *)malloc(N * sizeof(double));

    for (int i = 0; i < N; ++i)
    {
        x[i] = i + 1;
        y[i] = i + 2;
    }

    // calculo
    double t_ini = omp_get_wtime();
    int i;
    // hacemos la multiplicación
    for (i = 0; i < N; i++)
    {
        y[i] = a * x[i] + y[i];
    }

    double tiempo = omp_get_wtime() - t_ini;

    printf("Tiempo (seg.): %11.9f\n", tiempo);
    return 0;
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
Longitud vectores=XXXX	0.427963610	0.195451817	0.198768760	0.187425846

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

```
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ gcc -O0 -fopenmp -o daxpy daxpy.c
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./daxpy 100000000 2.25
Tiempo (seg.): 0.427963610
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ gcc -Os -fopenmp -o daxpy daxpy.c
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./daxpy 100000000 2.25
Tiempo (seg.): 0.195451817
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ gcc -O2 -fopenmp -o daxpy daxpy.c
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./daxpy 100000000 2.25
Tiempo (seg.): 0.198768760
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ gcc -O3 -fopenmp -o daxpy daxpy.c
carlos@carlos-HP-15-Notebook-PC:~/Escritorio/UNIVERSIDAD/AC/Practicas/BP4/CODIGO$ ./daxpy 100000000 2.25
Tiempo (seg.): 0.187425846
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Una de las diferencias que se ve es que en O3 es bastante más extenso que los otros códigos en ensamblador, ocupando 326 lineas, esto se debe al desenrollado de lo de los bucles. Otra diferencia que se ve es que en el código en ensamblador con O3 es que hace primero la comprobación, si lo cumple salta a L10, pero si no se hace la operación, el incremento y un salto a L5 de nuevo, en cambio con O0 hace la comprobación después de hacer la operación y el incremento, lo curioso es que comprueba antes de calcular en la primera vuelta, es decir entra en L3 y alta a L5, comprueba y luego salta a L6 y se repite L5 y L6 hasta que se cumpla la condición, con O2 ocurre algo similar, salvo que tiene 2 comprobaciones, 1 en L3 que comprueba con los valores inicializados, si se cumple salta a L5 y ejecuta el resto del código sin hacer bucle, pero si no entra en L6, calcula y si no cumple repite L6 hasta que se cumpla y con O3 hace una comprobación con los datos inicializados, si se cumple termina el programa, si no continua, hace la operación y se comprueba, si se cumple salta a L9 y termina, que no se calcula otra vez, se comprueba, si se cumple salta a L9 y si no calcula y continua en L10, de tal forma que solo utiliza un bucle en L11 haciendo que tarde menos en ejecutarse.

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> movq %xmm0, %rax movq %rax, -16(%rbp) movl \$0, -20(%rbp) jmp .L5 .L6 movq x(%rip), %rax movl -20(%rbp), %edx movslq %edx, %rdx salq \$3, %rdx addq %rdx, %rax movsd (%rax), %xmm0 mulsd (%xmm0, %xmm0) movq y(%rip), %rax movl -20(%rbp), %edx movslq %edx, %rdx salq \$3, %rdx addq %rdx, %rax movsd (%rax), %xmm1 movq y(%rip), %rax movl -20(%rbp), %edx movslq %edx, %rdx salq \$3, %rdx addq %rdx, %rax addsd %xmm1, %xmm0 movsd %xmm0, (%rax) addl \$1, -20(%rbp) .L5: movl -20(%rbp), %edx movl N(%rip), %eax cmpl %eax, %edx jb .L6 </pre>	<pre> movl N(%rip), %ecx movq x(%rip), %rsi movsd %xmm0, 8(%rsp) movq y(%rip), %rdx xorl %eax, %eax .L5: cmpl %eax, %ecx jbe .L10 movsd (%rsi,%rax,8), %xmm0 mulsd a(%rip), %xmm0 addsd (%rdx,%rax,8), %xmm0 movsd %xmm0, (%rdx,%rax,8) incq %rax jmp .L5 </pre>	<pre> Movl N(%rip), %eax movsd %xmm0, 8(%rsp) testl %eax, %eax je .L5 movq x(%rip), %rsi movq y(%rip), %rdx subl \$1, %eax leaq 8(%rax,8), %rcx xorl %eax, %eax .p2align 4,,10 .p2align 3 .L6: movsd (%rsi,%rax), %xmm0 mulsd a(%rip), %xmm0 addsd (%rdx,%rax), %xmm0 movsd %xmm0, (%rdx,%rax) addq \$8, %rax cmpq %rcx, %rax jne .L6 </pre>	<pre> movl N(%rip), %r9d movsd %xmm0, 8(%rsp) testl %r9d, %r9d je .L8 movq y(%rip), %rsi movq x(%rip), %r8 leaq 16(%rsi), %rax leaq 16(%r8), %rdx cmpq %rax, %r8 setnb %al orl %edx, %eax cmpl \$8, %r9d seta%dl testb %dl, %al je .L9 movl %r9d, %eax leaq (%rsi,%rax,8), %rdx leaq a(%rip), %rax cmpq %rax, %rdx setbe %dl addq \$8, %rax cmpq %rax, %rsi setnb %al orb %al, %dl je .L9 movq %rsi, %rdx xorl %edi, %edi shrq \$3, %rdx andl \$1, %edx je .L10 movsd a(%rip), %xmm0 movl \$1, %edi mulsd (%r8), %xmm0 addsd (%rsi), %xmm0 movsd %xmm0, (%rsi) .L10: movsd a(%rip), %xmm1 movl %r9d, %ebx xorl %eax, %eax subl %edx, %ebx movl %edx, %edx xorl %ecx, %ecx unpcklpd %xmm1, %xmm1 salq \$3, %rdx movl %ebx, %r11d leaq (%r8,%rdx), %r10 addq %rsi, %rdx shrl %r11d .p2align 4,,10 .p2align 3 .L11: movupd (%r10,%rax), %xmm0 addl \$1, %ecx mulpd %xmm1, %xmm0 addpd (%rdx,%rax), %xmm0 movaps %xmm0, (%rdx,%rax) addq \$16, %rax cmpl %ecx, %r11d ja .L11 movl %ebx, %eax andl \$-2, %eax addl %eax, %edi cmpl %eax, %ebx je .L8 movslq %edi, %rdx addl \$1, %edi movsd (%r8,%rdx,8), %xmm0 leaq (%rsi,%rdx,8), %rax </pre>

			<pre>cmpl %r9d, %edi mulsd a(%rip), %xmm0 addsd (%rax), %xmm0 movsd %xmm0, (%rax) jnb .L8 movslq %edi, %rdi movsd (%r8,%rdi,8), %xmm0 leaq (%rsi,%rdi,8), %rax mulsd a(%rip), %xmm0 addsd (%rax), %xmm0 movsd %xmm0, (%rax)</pre>
--	--	--	---

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