

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

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Valentino Lugli

Grupo de prácticas y profesor de prácticas: C1, Christian Morillas

Fecha de entrega: 07/06/21

Fecha evaluación en clase: 08/06/21

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): AMD Ryzen 7 3750H

Sistema operativo utilizado: Kubuntu 20.04.2 LTS

Versión de gcc utilizada: gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

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

```
File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~] 2021-05-18 martes
>>lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         43 bits physical, 48 bits virtual
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             AuthenticAMD
CPU family:            23
Model:                 24
Model name:            AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx
Stepping:              1
Frequency boost:        enabled
CPU MHz:               2373.822
CPU max MHz:           2300,0000
CPU min MHz:           1400,0000
BogoMIPS:              4591.34
Virtualization:        AMD-V
L1d cache:             128 KiB
L1i cache:             256 KiB
L2 cache:              2 MiB
L3 cache:              4 MiB
NUMA node0 CPU(s):     0-7
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf:     Not affected
Vulnerability Mds:      Not affected
Vulnerability Meltdown: Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Full AMD retpoline, IBPB conditional, STIBP disabled, RSB filling
Vulnerability Srbds:     Not affected
Vulnerability Tsx async abort: Not affected
```

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

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

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./pmm-secuencial_EXE n\nn    Dimensión de la matriz.\n");
        exit(-1);
    }

    // Declaración de datos.
    struct timespec cgt1,cgt2; double ncgt;

    int N = atoi(argv[1]);

    double **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\n");
        exit(-2);
    }

    for(int 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\n");
            exit(-2);
        }
    }

    // Inicialización de datos.
    int a = 0, b = N*N;
    struct drand48_data randBuffer;
    double randomNumber;
    srand48_r(time(NULL), &randBuffer);

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            m3[i][j] = 0;
        }
    }
}
```

```

        if (N < 9)
        {
            m1[i][j] = ++a;
            m2[i][j] = ++b;
        }
        else
        {
            drand48_r(&randBuffer, &randomNumber);
            m1[i][j] = randomNumber;
            drand48_r(&randBuffer, &randomNumber);
            m2[i][j] = randomNumber;
        }
    }
}

// Cálculo de la multiplicación de m1 * m2 = m3
clock_gettime(CLOCK_REALTIME, &cgt1);
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        for (int k = 0; k < N; k++)
        {
            m3[i][j] += m1[i][k] * m2[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));

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

printf("Resultado: \n");
if(N<9)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("[%d][%d] = %0.2f ", i, j, m3[i][j]);
        }
        printf("\n");
    }
}
else
{
    printf("[0][0] = %2.2f, ..., [%d][%d] = %2.2f", m3[0][0], N-1,N-1, m3[N-
1][N-1]);
}
printf("\n");

// Liberar memoria
for (int i = 0; i < N; i++)
{
    free(m1[i]);
    free(m2[i]);
}

```

```

        free(m3[i]);
    }

    free(m1);
    free(m2);
    free(m3);

    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) Desenrollado del Bucle:

Se ha decidido desenrollar el bucle y realizar dos multiplicaciones y sumas simultaneas, lo que equivaldría a ir añadiendo de dos en dos valores en la matriz resultante, esto por un lado reduce el número de saltos que realiza el código, además que permite al procesador estar más tiempo ocupado ya que se están accediendo a direcciones de memoria diferentes.

Modificación B) Optimizado de Localidad de Accesos:

Se ha decidido sobre la modificación anterior, optimizar la manera en que la matriz es accedida, esto ya que se sabe que las matrices son almacenadas por el compilador por filas; dado que estos datos son locales son más rápidos para su acceso. Para lograr esto, se modificaron los bucles para que todas las matrices sean recorridas por filas, el cambio es sencillo: intercambiar el bucle de j por el de k.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de pmm-secuencial-modificado_A.c

```

...
// Cálculo de la multiplicación de m1 * m2 = m3
// ¡Solo funciona con matrices pares!
clock_gettime(CLOCK_REALTIME, &cgt1);
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j+=2)
    {
        for (int k = 0; k < N; k++)
        {
            m3[i][j] += m1[i][k] * m2[k][j];
            m3[i][j+1] += m1[i][k] * m2[k][j+1];
        }
    }
}
clock_gettime(CLOCK_REALTIME, &cgt2);
...

```

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

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-05-25 martes
>>make
gcc -O2 -o pmm-secuencial_EXE pmm-secuencial.c
gcc -O2 -o pmm-secuencial-modificado_A_EXE pmm-secuencial-modificado_A.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-05-25 martes
>>./pmm-secuencial_EXE 4
Dimension: 4 Tiempo: 0.000000611
Resultado:
[0][0] = 250.00 [0][1] = 260.00 [0][2] = 270.00 [0][3] = 280.00
[1][0] = 618.00 [1][1] = 644.00 [1][2] = 670.00 [1][3] = 696.00
[2][0] = 986.00 [2][1] = 1028.00 [2][2] = 1070.00 [2][3] = 1112.00
[3][0] = 1354.00 [3][1] = 1412.00 [3][2] = 1470.00 [3][3] = 1528.00

[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-05-25 martes
>>./pmm-secuencial-modificado_A_EXE 4
Dimension: 4 Tiempo: 0.000000441
Resultado:
[0][0] = 250.00 [0][1] = 260.00 [0][2] = 270.00 [0][3] = 280.00
[1][0] = 618.00 [1][1] = 644.00 [1][2] = 670.00 [1][3] = 696.00
[2][0] = 986.00 [2][1] = 1028.00 [2][2] = 1070.00 [2][3] = 1112.00
[3][0] = 1354.00 [3][1] = 1412.00 [3][2] = 1470.00 [3][3] = 1528.00

[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-05-25 martes
>>

```

Symbolab SOLUTIONS GRAPHING PRACTICE GEOMETRY beta NOTEBOOK GROUPS CHEAT SHEETS

Pre Algebra Algebra Pre Calculus Calculus Functions Matrices & Vectors Geometry Trigonometry Statistics Physics

Step-by-Step Calculator

Solve problems from Pre Algebra to Calculus step-by-step

full pad »

x^2 x^\square \log_\square $\sqrt{\square}$ $\sqrt[\square]{\square}$ \leq \geq $\frac{\square}{\square}$ \cdot \div x° π
 $(\square)'$ $\frac{d}{dx}$ $\frac{\partial}{\partial x}$ \int \int_\square \lim \sum ∞ θ $(f \circ g)$ H_2O $\left(\begin{smallmatrix} \square & \square & \square \\ \square & \square & \square \end{smallmatrix}\right)$

Most Used Actions

simplify solve for inverse tangent line See All ▾

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \cdot \begin{pmatrix} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{pmatrix}$$

Go

Examples »

Solution

Keep Practicing >

Show Steps ▾

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \begin{pmatrix} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{pmatrix} = \begin{pmatrix} 250 & 260 & 270 & 280 \\ 618 & 644 & 670 & 696 \\ 986 & 1028 & 1070 & 1112 \\ 1354 & 1412 & 1470 & 1528 \end{pmatrix}$$

```

File Edit View Bookmarks Settings Help
[2][0] = 986.00 [2][1] = 1028.00 [2][2] = 1070.00 [2][3] = 1112.00
[3][0] = 1354.00 [3][1] = 1412.00 [3][2] = 1470.00 [3][3] = 1528.00

[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial_EXE 1250
Dimension: 1250 Tiempo: 5.450523049
Resultado:
[0][0] = 306.08, ..., [1249][1249] = 310.64
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial_EXE 1250
Dimension: 1250 Tiempo: 5.233185562
Resultado:
[0][0] = 314.66, ..., [1249][1249] = 317.09
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_A_EXE 1250
Dimension: 1250 Tiempo: 2.251726746
Resultado:
[0][0] = 313.33, ..., [1249][1249] = 314.76
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_A_EXE 1250
Dimension: 1250 Tiempo: 2.52283202
Resultado:
[0][0] = 315.15, ..., [1249][1249] = 311.51
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>

```

B) Captura de pmm-secuencial-modificado_B.c

```

...
// Cálculo de la multiplicación de m1 * m2 = m3
// ¡Solo funciona con matrices pares!
clock_gettime(CLOCK_REALTIME, &cgt1);
for (int i = 0; i < N; i++)
{
    for (int k = 0; k < N; k++)
    {
        for (int j = 0; j < N; j+=2)
        {
            m3[i][j] += m1[i][k] * m2[k][j];
            m3[i][j+1] += m1[i][k] * m2[k][j+1];
        }
    }
}
clock_gettime(CLOCK_REALTIME, &cgt2);
...

```

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

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>make
gcc -O2 -o pmm-secuencial-modificado_A_EXE pmm-secuencial-modificado_A.c
gcc -O2 -o pmm-secuencial-modificado_B_EXE pmm-secuencial-modificado_B.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_B_EXE 4
Dimension: 4 Tiempo: 0.000000572
Resultado:
[0][0] = 250.00 [0][1] = 260.00 [0][2] = 270.00 [0][3] = 280.00
[1][0] = 618.00 [1][1] = 644.00 [1][2] = 670.00 [1][3] = 696.00
[2][0] = 986.00 [2][1] = 1028.00 [2][2] = 1070.00 [2][3] = 1112.00
[3][0] = 1354.00 [3][1] = 1412.00 [3][2] = 1470.00 [3][3] = 1528.00

[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>

```

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_B_EXE 1250
Dimension: 1250 Tiempo: 1.595917964
Resultado:
[0][0] = 309.11, ..., [1249][1249] = 310.44
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_B_EXE 1250
Dimension: 1250 Tiempo: 1.664653547
Resultado:
[0][0] = 322.46, ..., [1249][1249] = 310.12
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_B_EXE 1250
Dimension: 1250 Tiempo: 1.658494931
Resultado:
[0][0] = 317.23, ..., [1249][1249] = 313.20
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_B_EXE 1250
Dimension: 1250 Tiempo: 1.695730514
Resultado:
[0][0] = 329.38, ..., [1249][1249] = 320.14
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer1] 2021-06-01 martes
>>./pmm-secuencial-modificado_B_EXE 1250
Dimension: 1250 Tiempo: 1.613127807
Resultado:
[0][0] = 318.37, ..., [1249][1249] = 307.86

```

TIEMPOS:

| Modificación | Breve descripción de las modificaciones | -O2, N=1250 |
|-----------------|--|-------------|
| Sin modificar | N/A | ~5.31377 s |
| Modificación A) | Se desenrolló el bucle más interno. | ~2.34023 s |
| Modificación B) | Se optimizó la localidad de los accesos de las matrices. | ~1.64558 s |

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

Se observó que efectivamente los cambios realizados lograron mejorar significativamente el tiempo de ejecución del la multiplicación de las matrices, esto se puede justificar, en el caso del desenrollado de bucles porque el programa está ahora realizando $N^2 * N/2$ bucles en vez de N^3 , de manera que al realizarse menos ejecuciones el tiempo también se reduce.

En el caso de la segunda modificación, se justifica por la manera en que el programa accede a los datos tiene un tiempo dependiendo de la ubicación de los mismos; el compilador hace que las matrices tengan una localidad de datos en sus filas, es decir, que un acceso de fila en fila es más rápido que uno de columna en columna porque estos datos están localmente cerca y a nivel de Hardware quiere decir que estarían en la misma línea de caché, mientras que de columna en columna puede que estén en un nivel superior de memoria y el tiempo que tarda el CPU en traerlo a caché y realizar el cálculo se empieza a acumular, por lo tanto esta mejora también reduce significativamente el tiempo de ejecución.

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>

```

```

#define MAX 64000

int main(int argc, char **argv)
{
    if(argc < 3)
    {
        printf("Uso: ./figura1-original_EXE N M\n");
        exit(-1);
    }

    // Declaración de datos.
    struct timespec cgt1,cgt2; double ncgt;

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

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

    double *R;
    R = (double *)malloc(M * sizeof(double));

    if(R == NULL)
    {
        printf("Error en la reserva de espacio\n");
        exit(-2);
    }

    // Inicialización de datos.
    int a = 0, b = N;
    struct drand48_data randBuffer;
    double randomNumber;
    srand48_r(time(NULL), &randBuffer);

    for (int i = 0; i < N; i++)
    {
        if (N < 9 && M < 9)
        {
            s[i].a = ++a;
            s[i].b = ++b;
        }
        else
        {
            drand48_r(&randBuffer, &randomNumber);
            s[i].a = (int)2*randomNumber;
            drand48_r(&randBuffer, &randomNumber);
            s[i].b = (int)2*randomNumber;
        }
    }

    // Cálculo
    double X1, X2;
    int ii, i;
    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));

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

printf("Resultado: \n");
if(N<9 && M<9)
{
    for (int i = 0; i < M; i++)
    {
        printf("[%d] = %2.0f ", i, R[i]);
    }
}
else
{
    printf("[0] = %2.0f, ..., [%d] = %2.0f", R[0], M-1, R[M-1]);
}
printf("\n");

// Liberar memoria
free(R);

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) Mejorar localidad de accesos unificando los bucles internos:

Debido a que en el código original se posee un vector de registros, para mejorar la localidad de los accesos se decidió unir los dos bucles en uno ya que se está accediendo a la misma localización de memoria, por lo que por localidad de memoria se mejoran los tiempos de ejecución.

Modificación B) Desenrollado de Bucle y reemplazar multiplicación por suma:

Debido a que el código realiza siempre una multiplicación por 2 y 3, se decidió que se realizaran entonces sumas ya que estas operaciones son más veloces que las de multiplicación. Adicionalmente se desenrolló el bucle ya que cada componente del vector no afecta a ninguna otra componente, por lo tanto, se pueden realizar la misma operación en diferentes componentes en una misma ejecución sin problema.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
...
// Cálculo
double X1, X2;
int ii, i;
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;
        X2 += 3 * s[i].b - ii;
    }
    if (X1<X2)
        R[ii] = X1;
    else
        R[ii] = X2;
}
clock_gettime(CLOCK_REALTIME,&cgt2);
...
```

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

```

File Edit View Bookmarks Settings Help
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>make
gcc -O2 -o figura1-original_EXE figura1-original.c
gcc -O2 -o figura1-modificado_A_EXE figura1-modificado_A.c
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_A_EXE 5 5
Dimension: 5 Tiempo: 0.000000381
Resultado:
[0] = 30 [1] = 35 [2] = 40 [3] = 45 [4] = 50
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_A_EXE 5 5
Dimension: 5 Tiempo: 0.000000501
Resultado:
[0] = 30 [1] = 35 [2] = 40 [3] = 45 [4] = 50
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-original_EXE 5 5
Dimension: 5 Tiempo: 0.000000702
Resultado:
[0] = 30 [1] = 35 [2] = 40 [3] = 45 [4] = 50
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>

```

```

File Edit View Bookmarks Settings Help
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-original_EXE 50000 50000
Dimension: 50000 Tiempo: 4.403149600
Resultado:
[0] = 50152, ..., [49999] = -2499875390
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-original_EXE 50000 50000
Dimension: 50000 Tiempo: 4.819922072
Resultado:
[0] = 50204, ..., [49999] = -2499875042
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_A_EXE 50000 50000
Dimension: 50000 Tiempo: 2.788087068
Resultado:
[0] = 50162, ..., [49999] = -2499875249
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_A_EXE 50000 50000
Dimension: 50000 Tiempo: 3.064410547
Resultado:
[0] = 49944, ..., [49999] = -2499874817
[ValentinoLuglii valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>

```

B) Captura figura1-modificado_B.c

```

...
// Cálculo
double X1, X2, Y1, Y2;
int ii, i;
clock_gettime(CLOCK_REALTIME, &cgt1);
for (ii = 0; ii < M; ii++)
{
    X1=0; X2=0;
    for(i=0; i<N;i+=2)
    {
        X1 += ((s[i].a + s[i].a) + ii) + ((s[i + 1].a + s[i + 1].a) + ii);
        X2 += ((s[i].b + s[i].b + s[i].b) - ii) + ((s[i + 1].b + s[i + 1].b +
s[i + 1].b) - ii);
    }

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

```

```

    }
    clock_gettime(CLOCK_REALTIME, &cgt2);
    ...

```

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

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>make
gcc -O2 -o figura1-modificado_B_EXE figura1-modificado_B.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_B_EXE 6 6
Dimension: 6 Tiempo: 0.000000471
Resultado:
[0] = 42 [1] = 48 [2] = 54 [3] = 60 [4] = 66 [5] = 72
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_A_EXE 6 6
Dimension: 6 Tiempo: 0.000000451
Resultado:
[0] = 42 [1] = 48 [2] = 54 [3] = 60 [4] = 66 [5] = 72
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-original_EXE 6 6
Dimension: 6 Tiempo: 0.000000662
Resultado:
[0] = 42 [1] = 48 [2] = 54 [3] = 60 [4] = 66 [5] = 72
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer2] 2021-06-01 martes
>>./figura1-modificado_B_EXE 50000 50000
Dimension: 50000 Tiempo: 2.107850000

```

TIEMPOS:

| Modificación | Breve descripción de las modificaciones | -O2, N=M=50000 |
|-----------------|--|-------------------|
| Sin modificar | N/A | ~4.61614 s |
| Modificación A) | Mejora de localidad de accesos a memoria. | ~2.79657 s |
| Modificación B) | Desenrollar el bucle y cambiar multiplicación por sumas. | ~2.10785 s |

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

La primera modificación se justifica debido a que los dos bucles del código original siempre estaban accediendo al mismo índice del vector de registros `s`, por lo tanto, resultaba mejor juntarlos en un único bucle, de esta manera se mantiene una localidad de memoria. Anteriormente, el CPU cargaba `s` a la caché, y luego volvía a hacerlo para el siguiente bucle, ahora solo realiza la carga una vez en el bucle; esta es una mejora drástica, se puede notar que los accesos a memoria y las transferencias a los niveles de la caché son costosos y optimizaciones en esto siempre acortarán de gran manera el tiempo.

La segunda modificación se justifica porque, en una parte, se están multiplicando constantes. Esto se puede reemplazar por sumas ya que la multiplicación será siempre por algo que no va a variar en las ejecuciones y se sabe que las operaciones de multiplicación son más costosas que las sumas (ligeramente), por lo tanto adicional a esto también se decidió desenrollar el bucle para dar una distinción mayor entre los tiempos; ya que indudablemente mejoran los tiempos cuando se reducen el número de iteraciones que se realizan. Aún así estas dos mejoras conjuntas logran una mejora modesta del tiempo, indica que las operaciones como tal son bastantes similares en costo, pero si existe tal diferencia entre las sumas y las multiplicaciones.

- 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>

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./daxpy N\n");
        exit(-1);
    }

    // Declaración de datos.
    struct timespec cgt1,cgt2; double ncgt;

    int N = atoi(argv[1]);

    double *x, *y;
    x = (double *)malloc(N * sizeof(double));
    y = (double *)malloc(N * sizeof(double));

    if(x == NULL || y == NULL)
    {
        printf("Error en la reserva de espacio\n");
        exit(-2);
    }

    // Inicialización de datos.
    int aAux = 0, bAux = N, i;
    struct drand48_data randBuffer;
    double randomNumber;
    srand48_r(time(NULL), &randBuffer);
    drand48_r(&randBuffer, &randomNumber);
    double a = randomNumber;
    for (int i = 0; i < N; i++)
    {
        if (N < 9)
        {
            x[i] = ++aAux;
            y[i] = ++bAux;
        }
        else
        {
            drand48_r(&randBuffer, &randomNumber);
            x[i] = 10*randomNumber;
            drand48_r(&randBuffer, &randomNumber);
```

```

        y[i] = 10*randomNumber;
    }
}

// Cálculo
clock_gettime(CLOCK_REALTIME, &cgt1);
for (i=0;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));

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

printf("Resultado: \n");
if (N < 9)
{
    for (int i = 0; i < N; i++)
    {
        printf("[%d] = %2.4f ", i, y[i]);
    }
}
else
{
    printf("[0] = %2.4f, ..., [%d] = %2.4f", y[0], N-1, y[N-1]);
}
printf("\n");

// Liberar memoria
free(x);
free(y);

return 0;
}

```

| Tiempos ejec. Longitud vectores=100000000 | -O0 | -Os | -O2 | -O3 |
|---|---------------|---------------|---------------|---------------|
| | 0.496181778 s | 0.165641279 s | 0.158208668 s | 0.148734653 s |

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

```

File Edit View Bookmarks Settings Help
>>make
gcc -O2 -o daxpy_EXE daxpy.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>make
gcc -O2 -o daxpy_EXE daxpy.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>./daxpy_EXE 4
a = 0.47Y = [5.00 6.00 7.00 8.00 ]
X = [1.00 2.00 3.00 4.00 ]
Dimension: 4 Tiempo: 0.000000250
Resultado:
[0] = 5.4665 [1] = 6.9330 [2] = 8.3996 [3] = 9.8661
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>make
gcc -O2 -o daxpy_EXE daxpy.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>./daxpy_EXE 4
a = 0.494971
Y = [ 5.00 6.00 7.00 8.00 ]
X = [ 1.00 2.00 3.00 4.00 ]
Dimension: 4 Tiempo: 0.000000191
Resultado:
[0] = 5.4950 [1] = 6.9899 [2] = 8.4849 [3] = 9.9799
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>

```

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>make
make: Nothing to be done for 'all'.
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>make
gcc -O0 -o daxpy00_EXE daxpy.c
gcc -O2 -o daxpy02_EXE daxpy.c
gcc -O3 -o daxpy03_EXE daxpy.c
gcc -Os -o daxpy0s_EXE daxpy.c
gcc -S -O0 -o daxpy00.s daxpy.c
gcc -S -O2 -o daxpy02.s daxpy.c
gcc -S -O3 -o daxpy03.s daxpy.c
gcc -S -Os -o daxpy0s.s daxpy.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer3] 2021-06-01 martes
>>

```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Compilación con 00: Esta bandera es la compilación por defecto para el código, no se realizan optimizaciones, además hace que sea más sencillo depurar el código con un depurador dedicado. La parte inicial del código se dedican al for, mientras que el cálculo como tal se realiza dentro de la etiqueta .L10; es un traspaso a ensamblador bastante directo del código fuente en C.

A parte de realizarse el “núcleo” de las operaciones con los registros flotantes `%xmm0` y `%xmm1`, utiliza los índices para ubicar las direcciones de memoria de los datos, por eso se ve el uso de los registros acumuladores y de propósito general `eax` y `rdx` también utilizados.

Compilación con Os: Esta bandera indica que se reduzca en tamaño el código, efectivamente se puede comprobar que el código que se obtiene es el más corto de todos los comparados, de igual manera esta bandera también activa todas las optimizaciones O2 excepto aquellas que incrementan el tamaño del código, por lo que además de un ejecutable de tamaño reducido también se encuentra con varias optimizaciones. Se puede notar que el bucle está compactado en la etiqueta .L9; se realizan las multiplicaciones y sumas del Daxpy en ese conjunto de líneas, se accede directamente a la memoria de los vectores, a diferencia que sin optimizar que calculaba en ese instante la dirección para cada elemento, aquí ya se hace de una manera más rápida, el calculo se sigue

realizando con **movsd** y **mulsd** se utiliza para realizar $a \times [i]$, luego eso se almacena en el registro **%xmm0** y se suma luego con $y[i]$ con **addsd** y finalmente se le asigna eso a $y[i]$ con **movsd**; luego se incrementa el contador i con **incq**.

Compilación con 02: Este código es similar al Os, pero no sin importar si las medidas de optimización aumentan el tamaño del código, esencialmente el código que realiza la multiplicación es idéntico al Os, ligeramente largo porque se utilizan más instrucciones para los contadores del for y la comparación de $i < N$, por ejemplo con **addq** y **cmpq** al final del código marcado, el cual se reduce en Os por una sola instrucción.

Compilación con 03: Este es el código con mayores optimizaciones, es notable que es también el código más largo de todos y donde también las partes que resuelven el bucle no se encuentran contiguas como en los otros códigos, se aplican muchas más optimizaciones aunque el núcleo del código se encuentra en la etiqueta **.L14** y **.L20** donde se puede observar las instrucciones de multiplicación de números flotantes; existe más de una versión porque se está optimizando para diferentes casos.

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> ... call clock_gettime movl \$0, -120(%rbp) jmp .L9 .L10: movl -120(%rbp), %eax cltq leaq 0(,%rax,8), %rdx movq -96(%rbp), %rax addq %rdx, %rax movsd (%rax), %xmm0 movapd %xmm0, %xmm1 mulsd -80(%rbp), %xmm1 movl -120(%rbp), %eax cltq leaq 0(,%rax,8), %rdx movq -88(%rbp), %rax addq %rdx, %rax movsd (%rax), %xmm0 movl -120(%rbp), %eax cltq leaq 0(,%rax,8), %rdx movq -88(%rbp), %rax addq %rdx, %rax addsd %xmm1, %xmm0 movsd %xmm0, (%rax) addl \$1, -120(%rbp) .L9: movl -120(%rbp), %eax cmpl -108(%rbp), %eax jl .L10 leaq -48(%rbp), %rax movq %rax, %rsi movl \$0, %edi call clock_gettime ... </pre> | <pre> ... call clock_gettime xorl %eax, %eax .L9: cmpl %eax, %ebx jle .L24 movsd 16(%rsp), %xmm0 mulsd (%r12,%rax,8), %xmm0 addsd 0(%rbp,%rax,8), %xmm0 movsd %xmm0, 0(%rbp,%rax,8) incq %rax jmp .L9 .L24: leaq 64(%rsp), %rsi xorl %edi, %edi xorl %r14d, %r14d call clock_gettime ... </pre> | <pre> ... call clock_gettime movq 16(%rsp), %r10 movsd 8(%rsp), %xmm1 xorl %eax, %eax .p2align 4,,10 .p2align 3 .L10: movsd 0(%rbp,%rax,8), %xmm0 movq %rax, %rdx mulsd %xmm1, %xmm0 addsd 0(%r13,%rax,8), %xmm0 movsd %xmm0, 0(%r13,%rax,8) addq \$1, %rax cmpq %r10, %rdx jne .L10 leaq 112(%rsp), %rsi xorl %edi, %edi xorl %r14d, %r14d call clock_gettime ... </pre> | <pre>L8: xorl %edi, %edi leaq 48(%rsp), %rsi movsd %xmm1, (%rsp) call clock_gettime cmpl \$1, %ebx movsd (%rsp), %xmm1 movl %ebx, %ecx je .L25 .L21: movl %ecx, %edx movapd %xmm1, %xmm2 xorl %eax, %eax shrl %edx unpcklpd %xmm2, %xmm2 salq \$4, %rdx .p2align 4,,10 .p2align 3 .L14: movupd 0(%rbp,%rax), %xmm0 movupd (%r15,%rax), %xmm4 mulpd %xmm2, %xmm0 addpd %xmm4, %xmm0 movups %xmm0, (%r15,%rax) addq \$16, %rax cmpq %rax, %rdx jne .L14 movl %ecx, %eax andl \$-2, %eax andl \$1, %ecx je .L13 .L20: cltq mulsd 0(%rbp,%rax,8), %xmm1 leaq (%r15,%rax,8), </pre> |

| | | | |
|--|--|--|---|
| | | | <pre> %rdx movsd (%rdx), %xmm0 addsd %xmm1, %xmm0 movsd %xmm0, (%rdx) .L13: leaq 64(%rsp), %rsi xorl %edi, %edi xorl %r13d, %r13d call clock_gettimeL41: leaq 48(%rsp), %rsi xorl %edi, %edi movsd %xmm1, (%rsp) call clock_gettime movsd (%rsp), %xmm1 movl %ebx, %ecx jmp .L21 .L5: leaq 48(%rsp), %rsi xorl %edi, %edi call clock_gettime leaq 64(%rsp), %rsi xorl %edi, %edi call clock_gettime ... </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`

```

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

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./pmm-paralelo_EXE n\n      Dimensión de la matriz.\n");
        exit(-1);
    }

    // Declaración de datos.
    double ncgt, cgt1, cgt2;

    int N = atoi(argv[1]);

    double **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\n");
    exit(-2);
}

for(int 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\n");
        exit(-2);
    }
}

// Inicialización de datos.
int k, j;

#pragma omp parallel firstprivate(m1, m2, k, j, N) shared(m3, cgt1, cgt2,
ncgt) default(none)
{
    #pragma omp single
    {
        int a = 0, b = N * N;
        struct drand48_data randBuffer;
        double randomNumber;
        srand48_r(time(NULL), &randBuffer);

        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
            {
                m3[i][j] = 0;
                if (N < 9)
                {
                    m1[i][j] = ++a;
                    m2[i][j] = ++b;
                }
                else
                {
                    drand48_r(&randBuffer, &randomNumber);
                    m1[i][j] = randomNumber;
                    drand48_r(&randBuffer, &randomNumber);
                    m2[i][j] = randomNumber;
                }
            }
        }
    }
}

// Cálculo de la multiplicación de  $m1 * m2 = m3$ 
// !Solo funciona con matrices pares!
#pragma omp single
{
    cgt1 = omp_get_wtime();
}
#pragma omp for

```

```

    for (int i = 0; i < N; i++)
    {
        for (k = 0; k < N; k++)
        {
            for (j = 0; j < N; j+=2)
            {
                m3[i][j] += m1[i][k] * m2[k][j];
                m3[i][j+1] += m1[i][k] * m2[k][j+1];
            }
        }
    }
    #pragma omp single
    {
        cgt2 = omp_get_wtime();
    }
}
ncgt = cgt2 - cgt1;

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

printf("Resultado: \n");
if(N<9)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("[%d][%d] = %0.2f ", i, j, m3[i][j]);
        }
        printf("\n");
    }
}
else
{
    printf("[0][0] = %2.2f, ..., [%d][%d] = %2.2f", m3[0][0], N-1,N-1, m3[N-1][N-1]);
}
printf("\n");

// Liberar memoria
for (int i = 0; i < N; i++)
{
    free(m1[i]);
    free(m2[i]);
    free(m3[i]);
}

free(m1);
free(m2);
free(m3);

return 0;
}

```

(b) RESPUESTA

Se realizaron las ejecuciones con N=1250 y N=2000. De esto, se obtuvo:

Depto. Arquitectura y Tecnología de Computadores

- Para N=1250, el tiempo secuencial fue 3.271847048 s y el tiempo paralelo fue 0.124103710 s.
La versión paralela es 26.3638 veces más rápida que su contraparte secuencial.
- Para N=3000, el tiempo secuencial fue 41.018464185 s y el tiempo paralelo fue 0.398312755 s.
La versión paralela es 102.9805 veces más rápida que su contraparte secuencial.

Se calculó la ganancia en prestaciones como $S(p) = \text{Tiempo}_{\text{secuencial}} / \text{Tiempo}_{\text{paralelo}}$

```
File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer4] 2021-06-02 miércoles
>>make
gcc -O2 -fopenmp -o pmm-paralelo_EXE pmm-paralelo.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer4] 2021-06-02 miércoles
>>../ejer1/pmm-secuencial_EXE 4
Dimension: 4 Tiempo: 0.000000801
Resultado:
[0][0] = 250.00 [0][1] = 260.00 [0][2] = 270.00 [0][3] = 280.00
[1][0] = 618.00 [1][1] = 644.00 [1][2] = 670.00 [1][3] = 696.00
[2][0] = 986.00 [2][1] = 1028.00 [2][2] = 1070.00 [2][3] = 1112.00
[3][0] = 1354.00 [3][1] = 1412.00 [3][2] = 1470.00 [3][3] = 1528.00

[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer4] 2021-06-02 miércoles
>>../pmm-paralelo_EXE 4
Dimension: 4 Tiempo: 0.000001403
Resultado:
[0][0] = 250.00 [0][1] = 260.00 [0][2] = 270.00 [0][3] = 280.00
[1][0] = 618.00 [1][1] = 644.00 [1][2] = 670.00 [1][3] = 696.00
[2][0] = 986.00 [2][1] = 1028.00 [2][2] = 1070.00 [2][3] = 1112.00
[3][0] = 1354.00 [3][1] = 1412.00 [3][2] = 1470.00 [3][3] = 1528.00

[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp4/ejer4] 2021-06-02 miércoles
>>
```

```
File Edit View Bookmarks Settings Help
>>ls
pmm-paralelo_EXE pmm-secuencial_EXE
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>srunc -pac4 -Aac -n1 -c32 --hint=nomultithread --exclusive ./pmm-secuencial_EXE 1250 > sec1.txt
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>cat sec1.txt
Dimension: 1250 Tiempo: 3.271847048
Resultado:
[0][0] = 318.83, ..., [1249][1249] = 310.55
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>srunc -pac4 -Aac -n1 -c32 --hint=nomultithread --exclusive ./pmm-secuencial_EXE 3000 > sec2.txt
srunc: Force Terminated job 108873
srunc: Job step aborted: Waiting up to 32 seconds for job step to finish.
slurmstepd: error: *** STEP 108873.0 ON atcgrid4 CANCELLED AT 2021-06-02T01:26:09 DUE TO TIME LIMIT ***
srunc: error: atcgrid4: task 0: Terminated
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>srunc -pac4 -Aac -n1 -c32 --hint=nomultithread --exclusive ./pmm-secuencial_EXE 2000 > sec2.txt
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>cat sec2.txt
Dimension: 2000 Tiempo: 41.018464185
Resultado:
[0][0] = 509.80, ..., [1999][1999] = 500.96
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>
```

ejer4 : bash (c1estudiante16) atcgrid.ugr.es ejer4 : sftp

```
File Edit View Bookmarks Settings Help
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>srunk -pac4 -Aac -n1 -c32 --hint=nomultithread --exclusive ./pmm-paralelo_EXE 1250 > parall1.txt
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>srunk -pac4 -Aac -n1 -c32 --hint=nomultithread --exclusive ./pmm-paralelo_EXE 2000 > parall2.txt
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>cat parall1.txt
Dimension: 1250 Tiempo: 0.128683889
Resultado:
[0][0] = 305.20, ..., [1249][1249] = 308.95
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>cat parall2.txt
Dimension: 2000 Tiempo: 0.396245362
Resultado:
[0][0] = 510.62, ..., [1999][1999] = 500.24
[ValentinoLugli c1estudiante16@atcgrid:~/bp4/ejer4] 2021-06-02 miércoles
>>
```