2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

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

Sistema operativo utilizado: *(respuesta)*

Versión de gcc utilizada: *(respuesta)*

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

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

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

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

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

A) MULTIPLICACIÓN DE MATRICES:
CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM 1000
int matrizA[TAM][TAM], matrizB[TAM][TAM], resultado[TAM][TAM];
int main(int argc, char const *argv[]) {
  struct timespec cgt1,cgt2;
  double ncgt;
  // Inicializar matrices
  for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
      matrizA[i][j] = (i+j) * 2;
```

```
      matrizB[i][j] = (i+j) * 4;
    }
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  // Realizar calculo matrizA x matrizB
  for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
      for (int k = 0; k < TAM; k++) {
        resultado[i][j] += matrizA[i][k] * matrizB[k][j];
      }
    }
  }
  clock_gettime(CLOCK_REALTIME,&cgt2);
  // Mostrar resultado
  printf("Primer elemento resultado: %u\n",resultado[0][0]);
  printf("Ultimo elemento resultado: %u\n",resultado[TAM-1][TAM-1]);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
  cgt1.tv_nsec)/(1.e+9));
  printf("Tiempo:\t%8.6f\n",ncgt);
  return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Desenrollar el tercer buvcle anidado y vamos a ir inclementando de 5 en 5, cuanto mayor sea este incremento mas mejorara el tiempo deejecucion.

Modificación b) –explicación-: Calcularemos la traspuesta de la matriz de la segunda matriz para hace filas x filas, no filas x columnas.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de `pmm-secuencial-modificado_a.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM 1000
int matrizA[TAM][TAM], matrizB[TAM][TAM], resultado[TAM][TAM];
int main(int argc, char const *argv[]) {
  struct timespec cgt1,cgt2;
  double ncgt;
  // Inicializar matrices
  for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
      matrizA[i][j] = (i+j) * 2;
      matrizB[i][j] = (i+j) * 4;
    }
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  // Realizar calculo matrizA x matrizB
  for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
      for (int k = 0; k < TAM; k+=5) {
        resultado[i][j] += matrizA[i][k] * matrizB[k][j];
        resultado[i][j] += matrizA[i][k+1] *matrizB[k+1][j];
        resultado[i][j] += matrizA[i][k+2]*matrizB[k+2][j];
        resultado[i][j] += matrizA[i][k+3]*matrizB[k+3][j];
```

```
        resultado[i][j] += matrizA[i][k+4]*matrizB[k+4][j];
      }
    }
  }
  matrizB[k][j];
  * matrizB[k+1][j];
  * matrizB[k+2][j];
  * matrizB[k+3][j];
  * matrizB[k+4][j];
  clock_gettime(CLOCK_REALTIME,&cgt2);
  // Mostrar resultado
  printf("Primer elemento resultado: %u\n",resultado[0][0]);
  printf("Ultimo elemento resultado: %u\n",resultado[TAM-1][TAM-1]);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
  cgt1.tv_nsec)/(1.e+9));
  printf("Tiempo:\t%8.6f\n",ncgt);
  return 0;
}
```

```
ignacio@ignacio-PC:~/universidad/AC/P4$ ./pmm-secuencial
Primer elemento resultado: 2662668000
Ultimo elemento resultado: 1450814816
Tiempo: 1.665226
ignacio@ignacio-PC:~/universidad/AC/P4$ ./pmm-secuencial-modificado_a
Primer elemento resultado: 2662668000
Ultimo elemento resultado: 1450814816
Tiempo: 1.712861
```

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

b) ...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM 1000
int matrizA[TAM][TAM], matrizB[TAM][TAM], resultado[TAM][TAM];
int main(int argc, char const *argv[]) {
  struct timespec cgt1,cgt2;
  double ncgt;
  // Inicializar matrices
  for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
      matrizA[i][j] = (i+j) * 2;
      matrizB[i][j] = (i+j) * 4;
    }
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  // Realizar calculo matrizA x matrizB
  for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
      for (int k = 0; k < TAM; k+=5) {
        resultado[i][j] += matrizA[i][k] * matrizB[k][j];
        resultado[i][j] += matrizA[i][k+1] *matrizB[k+1][j];
        resultado[i][j] += matrizA[i][k+2]*matrizB[k+2][j];
```

```
        resultado[i][j] += matrizA[i][k+3]*matrizB[k+3][j];
        resultado[i][j] += matrizA[i][k+4]*matrizB[k+4][j];
      }
    }
  }
  matrizB[k][j];
  * matrizB[k+1][j];
  * matrizB[k+2][j];
  * matrizB[k+3][j];
  * matrizB[k+4][j];
  clock_gettime(CLOCK_REALTIME,&cgt2);
  // Mostrar resultado
  printf("Primer elemento resultado: %u\n",resultado[0][0]);
  printf("Ultimo elemento resultado: %u\n",resultado[TAM-1][TAM-1]);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
  cgt1.tv_nsec)/(1.e+9));
  printf("Tiempo:\t%8.6f\n",ncgt);
  return 0;
}
```

```
ignacio@ignacio-PC:~/universidad/AC/P4$ ./pmm-secuencial-modificado_b
Primer elemento resultado: 2662668000
Ultimo elemento resultado: 1450814816
Tiempo: 0.677797
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*1.665226*
Modificación a)	1.712861
Modificación b)	0.677797
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como podemos ver en los tiempo obtenidos, la mejor opción es calcular la matriz traspuesta para multiplicar FILASxFILAS y asi producir menos fallos de caché. Esta opción es mucho más rapida que la primera modificación por lo que interesa perder tiempo en calcular la traspuesta de una matriz para despues hacer el cálculo con ella.

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

`pmm-secuencial.s`	`pmm-secuencial-modificado_b.s`	`pmm-secuencial-modificado_c.s`

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

```
#include <stdlib.h>
#include <time.h>
#define TAM_VECTOR 5000
#define NUM_ITER 40000
struct{
  int a;
  int b;
} s[TAM_VECTOR];
int R[NUM_ITER];
int main(int argc, char const *argv[]) {
  int X1, X2;
  struct timespec cgt1,cgt2;
  double ncgt;
  for (int i = 0; i < TAM_VECTOR; i++) {
    s[i].a = i*2;
    s[i].b = i*4;
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  for (int ii = 0; ii < NUM_ITER; ii++) {
    X1 = 0; X2 = 0;
    for (int i = 0; i < TAM_VECTOR; i++) {
      X1 += 2*s[i].a+ii;
    }
    for (int i = 0; i < TAM_VECTOR; 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("R[0]: %u\nR[39999]: %u\n\nTiempo:\t
%8.6f\n",R[0],R[39999],ncgt);
  return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: la idea es reducir los dos bucles a uno solo

Modificación b) –explicación-: Cambiar el if(...) {} else {} por R[ii] = (X1 < X2) ? X1 : X2;

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM_VECTOR 5000
#define NUM_ITER 40000
struct{
  int a;
  int b;
} s[TAM_VECTOR];
  int R[NUM_ITER];
  int main(int argc, char const *argv[]) {
```

```
  int X1, X2, ii;
  struct timespec cgt1,cgt2;
  double ncgt;
  for (int i = 0; i < TAM_VECTOR; i++) {
    s[i].a = i*2;
    s[i].b = i*4;
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  for (ii = 0; ii < NUM_ITER; ii++) {
    X1 = 0; X2 = 0;
    for (int i = 0; i < TAM_VECTOR; 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);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
  cgt1.tv_nsec)/(1.e+9));
  printf("R[0]: %u\nR[39999]: %u\n\nTiempo:\t
  %8.6f\n",R[0],R[39999],ncgt);
  return 0;
}
```

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

```
ignacio@ignacio-PC:~/universidad/AC/P4$ ./figura1-original
R[0]: 49990000
R[39999]: 4244942296

Tiempo: 0.281499
ignacio@ignacio-PC:~/universidad/AC/P4$ ./figura1-modificada_a
R[0]: 49990000
R[39999]: 4244942296

Tiempo:  0.197219
ignacio@ignacio-PC:~/universidad/AC/P4$ ./figura1-modificada_b
R[0]: 49990000
R[39999]: 4244942296

Tiempo:   0.292173
```

b) ...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM_VECTOR 5000
#define NUM_ITER 40000
struct{
  int a;
  int b;
} s[TAM_VECTOR];
  int R[NUM_ITER];
int main(int argc, char const *argv[]) {
  int X1, X2, ii;
  struct timespec cgt1,cgt2;
```

```
  double ncgt;
  for (int i = 0; i < TAM_VECTOR; i++) {
    s[i].a = i*2;
    s[i].b = i*4;
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  for (ii = 0; ii < NUM_ITER; ii++) {
    X1 = 0; X2 = 0;
    for (int i = 0; i < TAM_VECTOR; i++) {
      X1 += 2*s[i].a+ii;
    }
    for (int i = 0; i < TAM_VECTOR; i++) {
      X2 += 3*s[i].b-ii;
    }
    R[ii] = (X1 < X2) ? X1 : X2;
  }
  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
  cgt1.tv_nsec)/(1.e+9));
  printf("R[0]: %u\nR[39999]: %u\n\nTiempo:\t
  %8.6f\n",R[0],R[39999],ncgt);
  return 0;
}
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.281499*
Modificación a)	0.197219
Modificación b)	0.292173
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como podemos observar se optiene un mayor beneficio cuando en vez de utilizar 2 for usamos 1 solo. Esto tiene sentido debido a que supongamos un bucle de 60 iteraciones a 1s por iteración, si usamos 2 bucles como estos, se tardaría un total de 2 minutos en ejecutarse mientras que si aprovechamos y en 1 solo hacemos las mismas operaciones el tiempo sereduciría a la mitad (1 minuto).

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

`pmm-secuencial.s`	`pmm-secuencial-modificado_b.s`	`pmm-secuencial-modificado_c.s`

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este

programa es una rutina denominada DAXPY (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

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

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

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

CAPTURA CÓDIGO FUENTE: daxpy.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM 100000000
int vectorA[TAM], vectorB[TAM];
int main(int argc, char const *argv[]) {
  int a = 5;
  struct timespec cgt1,cgt2;
  double ncgt;
  // Inicializar vectores
  for (int i = 0; i < TAM; i++) {
    vectorA[i] = (i+1) * 2;
    vectorB[i] = (i+2) * 4;
  }
  clock_gettime(CLOCK_REALTIME,&cgt1);
  // Benchmark
  for (int i = 0; i < TAM; i++) {
    vectorA[i] = vectorB[i] * a + vectorA[i];
  }
  clock_gettime(CLOCK_REALTIME,&cgt2);
  // Mostrar resultado
  printf("Primer elemento resultado: %u\n",vectorA[0]);
  printf("Ultimo elemento resultado: %u\n",vectorA[TAM-1]);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
  printf("Tiempo:\t%8.6f\n",ncgt);
  return 0;
}
```

Tiempos ejec.	**-O0**	**-O2**	**-O3**
	0.338735	0.115330	0.115421

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

```
ignacio@ignacio-PC:~/universidad/AC/P4$ ./daxpy0
Primer elemento resultado: 42
Ultimo elemento resultado: 2200000020
Tiempo: 0.338735
ignacio@ignacio-PC:~/universidad/AC/P4$ ./daxpy1
bash: ./daxpy1: No existe el archivo o el directorio
ignacio@ignacio-PC:~/universidad/AC/P4$ ./daxpy2
Primer elemento resultado: 42
Ultimo elemento resultado: 2200000020
Tiempo: 0.115330
ignacio@ignacio-PC:~/universidad/AC/P4$ ./daxpy3
Primer elemento resultado: 42
Ultimo elemento resultado: 2200000020
Tiempo: 0.115421
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

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

daxpyO0.s	daxpyO2.s	daxpyO3.s