Laboratorio Sesión 10: Investigando la Jerarquía de Memoria (2)

Objetivo de la Sesión

El objetivo de la sesión es observar el efecto que puede tener la jerarquía de memoria en el rendimiento de un programa.

¿Influye la Jerarquía de Memoria en el tiempo de Ejecución de los Programas?

La mejor forma de comprobar la influencia de la jerarquía de memoria en el tiempo de ejecución de los programas es mediante un ejemplo. En el paquete de programas de esta sesión tenéis 3 programas: `mm-ijk.c`, `mm-jki.c` y `mm-kij.c`. Estos programas son 3 de las 6 formas ijk del producto de matrices.

En el programa `mm-ijk.c` se evalúa la siguiente porción de código (`ijk` denota que el bucle más externo es el bucle controlado por la variable `i` y el más interno el controlado por `k`):

```
t1 = GetTime();

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];

t2 = GetTime();
total = t2 - t1;
```

El conjunto de los tres bucles anidados realiza el producto de matrices C = A * B. Las llamadas a las rutinas `GetTime()` sirven para calcular el tiempo de ejecución de este código. La rutina GetTime() nos da el tiempo empleado por el programa hasta ese instante; la diferencia (`t2-t1`) nos da el tiempo de ejecución del bucle medido en milisegundos. Este es el esquema básico a utilizar para medir el tiempo de ejecución de una porción de código.

En el programa `mm-jki.c`, el código a evaluar es el siguiente:

```
t1 = GetTime();

for(j=0; j<N; j++)
  for (k=0; k<N; k++)
    for (i=0; i<N; i++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];

t2 = GetTime();
total = t2 - t1;
```

La única diferencia con respecto al código anterior es la ordenación de los bucles y, en consecuencia, el orden en que se accede a los elementos de las matrices (algo similar ocurre para el programa `mm-kij.c`). Un detalle fundamental es que los tres programas realizan, de distinta forma, la misma operación: el producto de matrices.

Para compilar y ejecutar estos programas en Linux hay que hacer lo siguiente:

```
$> gcc mm-ijk.c tiempo.c -DN=64 -o IJK64
$> ./IJK64
```

Donde "`tiempo.c`" es el fichero que contiene la rutina `GetTime()`; "`-DN=64`" indica que el tamaño de las matrices es N=64 (si no se pone nada el valor por defecto es N=256); y "`IJK64`" es el nombre del fichero ejecutable.

El ejecutable se comporta de forma diferente según el tamaño de la matriz. Si la matriz tiene un tamaño N=6 (o menor), la aplicación vuelca por pantalla el contenido de la matriz resultante. Esto es útil para comprobar que los tres programas tienen el mismo resultado. Para tamaños mayores la aplicación devuelve el tiempo de ejecución medido en tics de reloj.

En esta sesión hay que hacer lo siguiente:

1) **Compilad y ejecutad** los tres programas para un tamaño N=6. **Comprobad** que los 3 programas dan el mismo resultado.
2) **Rellenad** una tabla similar a esta:

N	Tiempo ejecución (en seg)			MFLOPs[a]		
	mm-ijk	mm-jki	mm-kij			
128						
256						
512						

a. Los MFLOPs son una medida utilizada para comparar el rendimiento de una aplicación en un procesador determinado. Los MFLOPs miden los millones de operaciones en coma flotante ejecutadas por segundo.
Se miden con la siguiente fórmula MFLOPs = (# operaciones en Coma Flotante) / (tiempo * 10^6).

3) Teniendo en cuenta lo que habéis hecho en los apartados anteriores y en el trabajo previo, explicad el porqué de las diferencias de rendimiento en estos tres programas.

Una optimización adicional

Una posible optimización del producto de matrices podría ser el siguiente:

```
t1 = GetTime();

for (i=0; i<N; i++)
  for (j=0; j<N; j++) {
    tmp = C[i][j];
    for (k=0; k<N; k++)
      tmp = tmp + A[i][k] * B[k][j];
    C[i][j] = tmp;
  }
t2 = GetTime();
total = t2 - t1;
```

Esta optimización se puede aplicar a las tres formas anteriores (nota importante: la optimización se aplica de forma diferente en los tres códigos). Este código optimizado lo podéis encontrar en el fichero `mm-ijk2.c`.

Una vez hayáis entendido esta optimización, se pide:

1) Aplicad esta optimización a las otras dos aplicaciones.
2) Compilad y ejecutad los tres programas para un tamaño N=6. Comprobad que los 3 programas dan el mismo resultado.

3) Rellenad una tabla similar a ésta:

N	Tiempo ejecución (en seg)			MFLOPs		
	mm-ijk2	mm-jki2	mm-kij2			
128						
256						
512						

4) Comparad los resultados obtenidos con los obtenidos antes de optimizar los programas, y sacad conclusiones de dica comparación.

Observad que pequeños cambios en el código pueden suponer cambios drásticos en el rendimiento. También es cierto que el compilador debería ser capaz de realizar estos cambios directamente.

Algo Nuevo: El Precompilador de C y la Compilación Condicional

Antes de ejecutar el compilador de C siempre se llama al precompilador. El precompilador de C se llama **cpp**. El precompilador se encarga básicamente de:

- La inclusión de los ficheros de cabeceras (p.e. #include <stdio.h>).
- La expansión de macros.
- La compilación condicional. Usando algunas directivas especiales es posible incluir o excluir partes del programa original en función de condiciones varias.

En los programas de la sesión de hoy hemos utilizado la compilación condicional. En la parte incial de cada programa encontramos lo siguiente:

```
#ifndef N
#define N 256  /* Dimensión por defecto */
#endif
```

Estas tres sentencias indican que si la constante N no está definida, entonces toma el valor 256. La forma de darle un valor diferente a esta constante es al compilar:

```
$> gcc mm-ijk.c tiempo.c -DN=64 -o IJK64
```

La opción "`-DN=64`", es la forma de inicializar N con el valor 64. Este mecanismo es muy útil para redefinir los parámetros de un programa en tiempo de compilación, sin necesidad de editar el programa de nuevo.

Resultados de la sesión

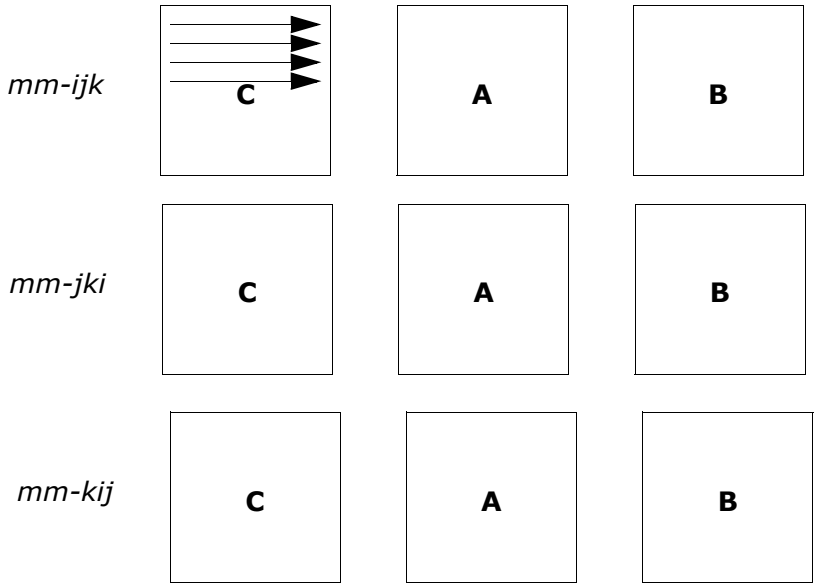
Antes de acabar la sesión tenéis que haber hecho, por lo menos, lo siguiente:

- Compilado y ejecutado los programas de las formas ijk del producto de matrices.
- Contestado **POR ESCRITO** todas las cuestiones que se plantean.
- Comprobado la diferencia de rendimiento entre las tres formas y justificado esta diferencia.
- Compilado y ejecutado el mm-ijk2.c y comparado el tiempo de ejecución con respecto al código no optimizado.
- Aplicado la optimización a los dos programas restantes y comprobado la diferencia de rendimiento.
- Justificado las diferencias de rendimiento entre las versiones optimizadas y las versiones sin optimizar de los tres programas.

Nota: Los ficheros para esta sesión los podéis encontrar en la página web de la asignatura: "`Programas.Sesion09.tar.gz`".

Trabajo previo de la sesión

1) **Dibujad**, para cada una de las formas ijk, en qué orden se recorren las matrices A, B y C.



2) Suponiendo que cada elemento de las matrices ocupa 4 bytes y que nuestra cache de datos tiene líneas de tamaño 32 bytes **calculad**, para cada una de las 3 formas ijk, cuántas líneas se han de mover de Memoria Principal a Memoria Cache para ejecutar completamente el bucle más interno 1 vez. Por ejemplo, en la forma ijk:

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

tenéis que calcular cuántas líneas se han de Mover de MP a MC para ejecutar:

```
for (k=0; k<N; k++)
  C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

Esto es equivalente a calcular los fallos de cache, suponiendo que la MC es de tamaño infinito y completamente asociativa. **Escribid** los resultados en la siguiente tabla:

N	mm-ijk			mm-jki			mm-kij		
	matriz A	matriz B	matriz C	matriz A	matriz B	matriz C	matriz A	matriz B	matriz C
128									
256									
512									

3) Suponiendo que cada elemento de las matrices ocupa 4 bytes, y que el tamaño de página de nuestro sistema es de 8Kbytes calculad, para cada una de las formas ijk, cuántas páginas de memoria virtual se utilizan al ejecutar completamente el bucle más interno 1 vez (como en el apartado anterior). **Escribid** los resultados en la siguiente tabla:

N	mm-ijk			mm-jki			mm-kij		
	matriz A	matriz B	matriz C	matriz A	matriz B	matriz C	matriz A	matriz B	matriz C
128									
256									
512									

Nombre 1:	Fecha:
Nombre 2:	Grupo:

¿Influye la Jerarquía de Memoria en el tiempo de Ejecución de los Programas?

1) **Compilad y ejecutad** los tres programas para un tamaño N=6. **Comprobad** que los 3 programas dan el mismo resultado.
2) **Rellenad** la siguiente tabla:

N	Tiempo ejecución (en seg)			MFLOPs		
	mm-ijk	mm-jki	mm-kij	mm-ijk	mm-jki	mm-kij
128						
256						
512						

3) Teniendo en cuenta lo que habéis hecho en los apartados anteriores y en el trabajo previo, explicad el porqué de las diferencias de rendimiento en estos tres programas.

.....

.....

.....

.....

.....

.....

.....

.....

.....

Una optimización adicional

1) Aplicad esta optimización a las otras dos aplicaciones.
2) Compilad y ejecutad los tres programas para un tamaño N=6. Comprobad que los 3 programas dan el mismo resultado.
3) **Rellenad** la siguiente tabla:

N	Tiempo ejecución (en seg)			MFLOPs		
	mm-ijk2	mm-jki2	mm-kij2	mm-ijk2	mm-jki2	mm-kij2
128						
256						
512						

4) Comparad los resultados obtenidos con los obtenidos antes de optimizar los programas, y sacad conclusiones de dica comparación.

.....

.....

.....

.....

.....

.....

.....

.....