Presentación *Práctica 3*

Arquitectura de Computadores

3º de grado en Ingeniería Informática y

3º de doble grado en Ing. Informática y Matemáticas

Memoria Caché y rendimiento

- El objetivo de esta práctica es experimentar y controlar el efecto de las memorias caché en el rendimiento de un programa de usuario.

- Ej0: Información sobre la caché del sistema
- Ej1: Memoria caché y rendimiento
- Ej2: Tamaño de la caché y rendimiento
- Ej3: Caché y multiplicación de matrices
- Ej4: Configuraciones de Caché en la multiplicación de matrices (Opcional)

Material entregado

- **arqo3.c** – fichero de código fuente de la librería de manejo de matrices.
- **arqo3.h** – fichero de cabeceras de la librería de manejo de matrices.
- **slow.c** – fichero de código fuente del programa de ejemplo que calcula la suma de los elementos de una matriz.
- **fast.c** – fichero de código fuente del programa de ejemplo que calcula la suma de los elementos de una matriz (más eficiente que el anterior).
- **Makefile** – fichero utilizado para la compilación de los ejemplos aportados.
- **slow_fast_time.sh** – fichero que contiene un script Bash de ejemplo para ejecutar los programas slow y fast.

Tiempo de ejecución de un programa

```
int main( int argc, char *argv[]) {
  ...
  m = generateMatrix(n);

  gettimeofday(&ini, NULL);

  res = computation(m, n);

  gettimeofday(&fin, NULL);

  tFinUS = (fin.tv_sec * 1.e6 + fin.tv_usec); /*fin microseg*/
  tIniUS = (ini.tv_sec * 1.e6 + ini.tv_usec); /*inicio microseg.*/
  tElapsedSeg = (tFinUS – tIniUS) * 1.0 / 1.e6;
  printf("Execution time: %f\n", tElapsedSeg);
  free(m);
  ...
}
```

1. Tomar tiempo justo antes de la computación, sin contar tiempo de reservar memoria.

2. Tomar tiempo justo después de la computación, sin contar el tiempo de imprimir por pantalla y liberar memoria.

3. Calcular el tiempo que ha pasado durante la ejecución como la diferencia entre las dos muestras de tiempo tomadas.

Framework Valgrind

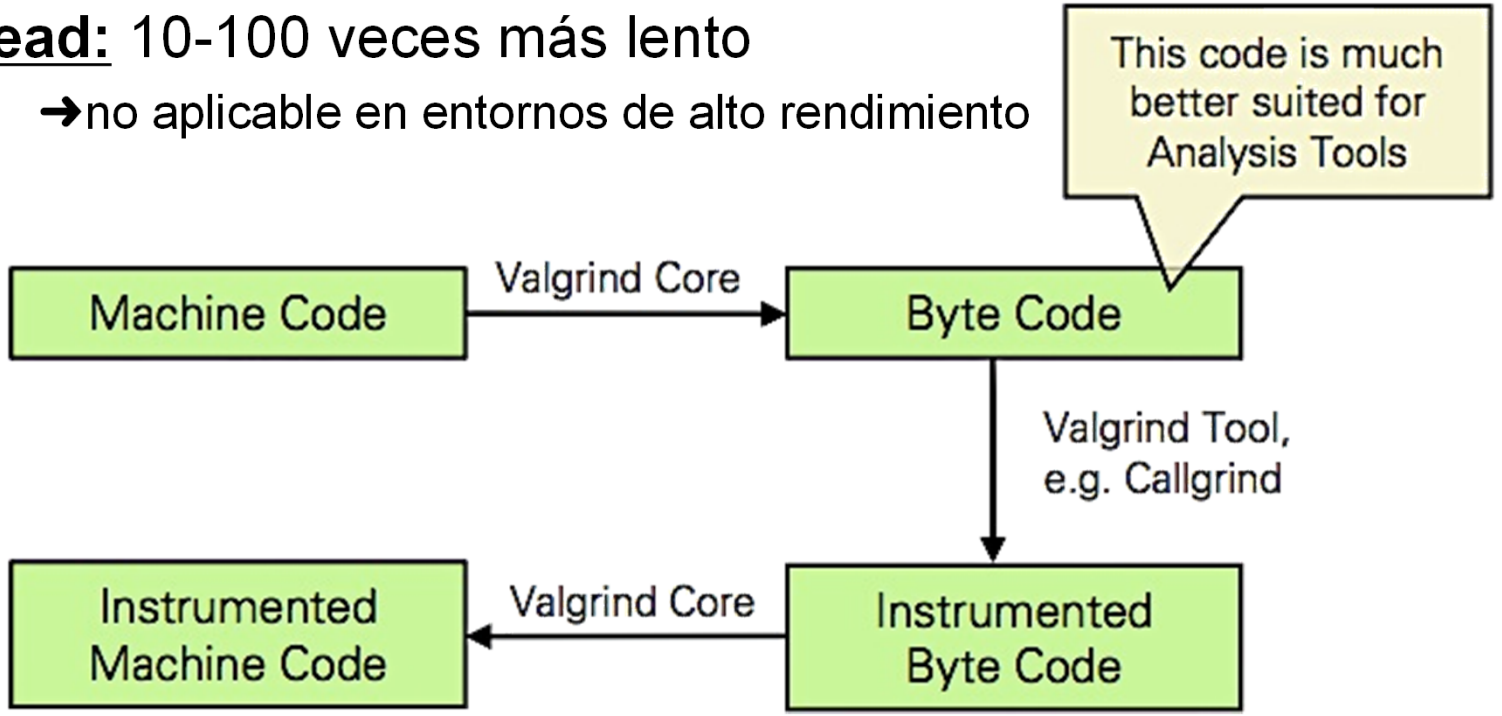
- Conjunto de herramientas basadas en simulación para debug y profiling
- Simula una CPU “software”
- Añade código de análisis
- Open Source
- Paquete estándar de Linux
- Utilizado por Firefox, KDE, …

http://www.valgrind.org

Framework Valgrind

- Recompilación dinámica del código máquina del programa destino en tiempo de ejecución
- El código original no se ejecuta en la CPU
- **Overhead:** 10-100 veces más lento

  ➜no aplicable en entornos de alto rendimiento



Framework Valgrind

- **Memcheck:** la más común, detección de errores en la gestión de la memoria de un programa
- **Cachegrind:** cache *profiling*, localización de las fuentes de fallos de caché
- **Callgrind:** extensión de *cachegrind*, añade generación del grafo de llamadas
- **Massif:** heap (reservas de memoria) profiler
- **Helgrind:** depuración de *threads*

Opciones de Valgrind

```
valgrind –-tool=<herramienta> [opciones]
     <ejecutable> [args_ejecutable]

    --I1=<size>,<associativity>,<line size>
        Level 1 instruction cache

    --D1=<size>,<associativity>,<line size>
        Level 1 data cache

    --LL=<size>,<associativity>,<line size>
        Last-level cache

    --callgrind-out-file=<file>
        Output file name

    --cachegrind-out-file=<file>
        Output file name
```

Ejemplo de Cachegrind

```
> valgrind --tool=cachegrind
    --cachegrind-out-file=slow_out.dat ./slow 1000
    ...
> cg_annotate slow_out.dat
```

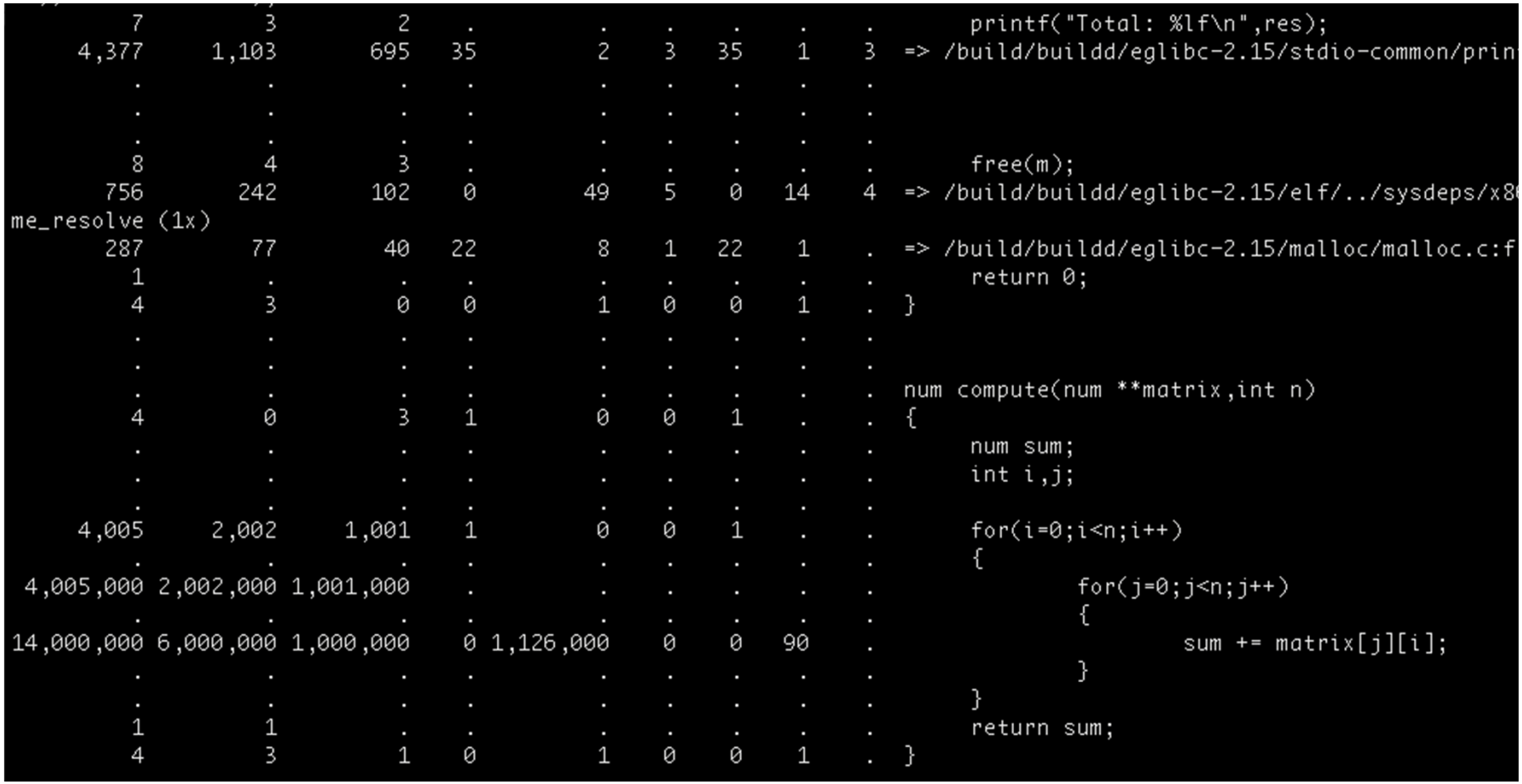
```
--------------------------------------------------------------------------------
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         8388608 B, 64 B, 16-way associative
Command:          ./slow 1000
Data file:        cachegrind.out.31409
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       0.1 100 100 100 100 100 100 100 100
Include dirs:
User annotated:
Auto-annotation:  off

--------------------------------------------------------------------------------
        Ir  I1mr  ILmr         Dr      D1mr  DLmr        Dw    D1mw    DLmw
--------------------------------------------------------------------------------
85,189,296 1,124 1,110 33,063,102 1,128,562 2,079 9,019,008 125,742 125,678  PROGRAM TOTALS

--------------------------------------------------------------------------------
        Ir I1mr ILmr        Dr      D1mr DLmr        Dw    D1mw    DLmw  file:function
--------------------------------------------------------------------------------
25,975,792    3    3 8,002,480         0    0 4,001,240       0       0  /build/buildd/eglibc-2.15/stdlib/ran
19,021,040    5    5 8,010,011         2    2 2,002,013 125,126 125,124  /<ruta>/p3/arqo3.c:generateMatrix
18,009,014    2    2 9,005,006 1,126,001   91 1,001,005       0       0  /<ruta>/p3/slow.c:compute
17,000,000    2    2 6,000,000         0    0 1,000,000       0       0  /build/buildd/eglibc-2.15/stdlib/ran
```

Ejemplo de Callgrind (I)

```
> valgrind --tool=callgrind –-cache-sim=yes
  --cachegrind-out-file=slow_out.dat ./slow 1000
  ...
> callgrind_annotate –-auto=yes slow_out.dat
```

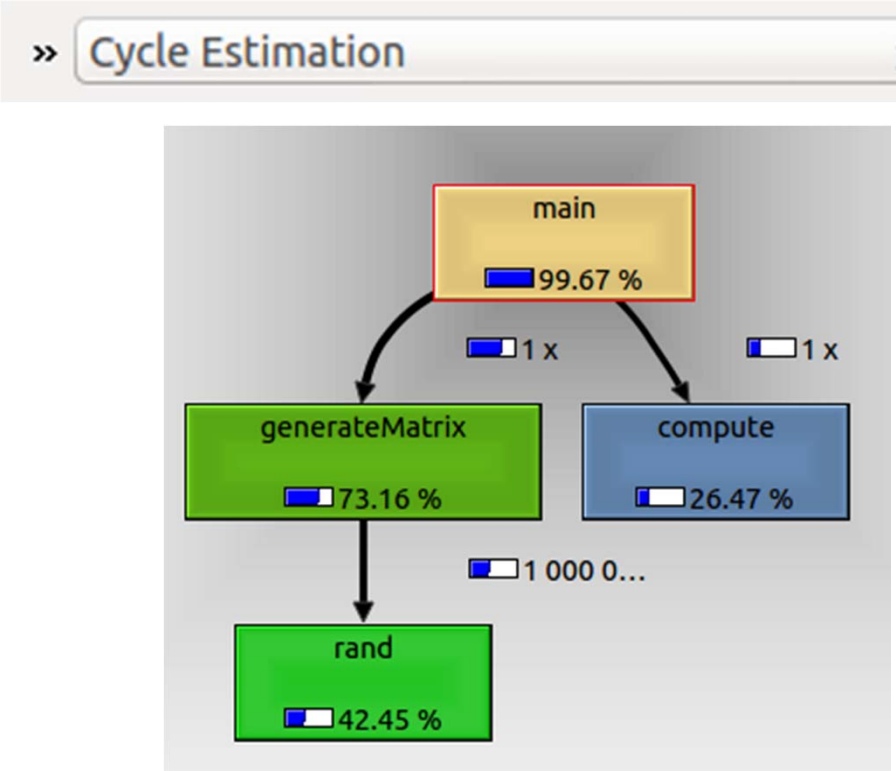
```
        Ir        Dr        Dw I1mr      D1mr D1mw ILmr DLmr DLmw

-- line 2 ----------------------------------------
         .         .         .    .         .    .    .    .    .  #include <stdlib.h>
         .         .         .    .         .    .    .    .    .  #include <sys/time.h>
         .         .         .    .         .    .    .    .    .
         .         .         .    .         .    .    .    .    .  #include "arqo3.h"
         .         .         .    .         .    .    .    .    .
         .         .         .    .         .    .    .    .    .  num compute(num **matrix,int n);
         .         .         .    .         .    .    .    .    .
         .         .         .    .         .    .    .    .    .  int main( int argc, char *argv[])
         6         0         4    .         .    .    .    .    .  {
         .         .         .    .         .    .    .    .    .       int n;
         1         0         1    .         .    .    .    .    .       num **m=NULL;
         .         .         .    .         .    .    .    .    .       struct timeval fin,ini;
         .         .         .    .         .    .    .    .    .       num res;
         .         .         .    .         .    .    .    .    .
         9         3         3    .         .    .    .    .    .       printf("Word size: %ld bits\n",8
     2,135       582       317  101        22   18  100    8    .  => /build/buildd/eglibc-2.15/stdio-co
       788       248       102    0         7    0    0    7    .  => /build/buildd/eglibc-2.15/elf/../s
me_resolve (1x)
         2         1         .    .         .    .    .    .    .       if( argc!=2 )
```

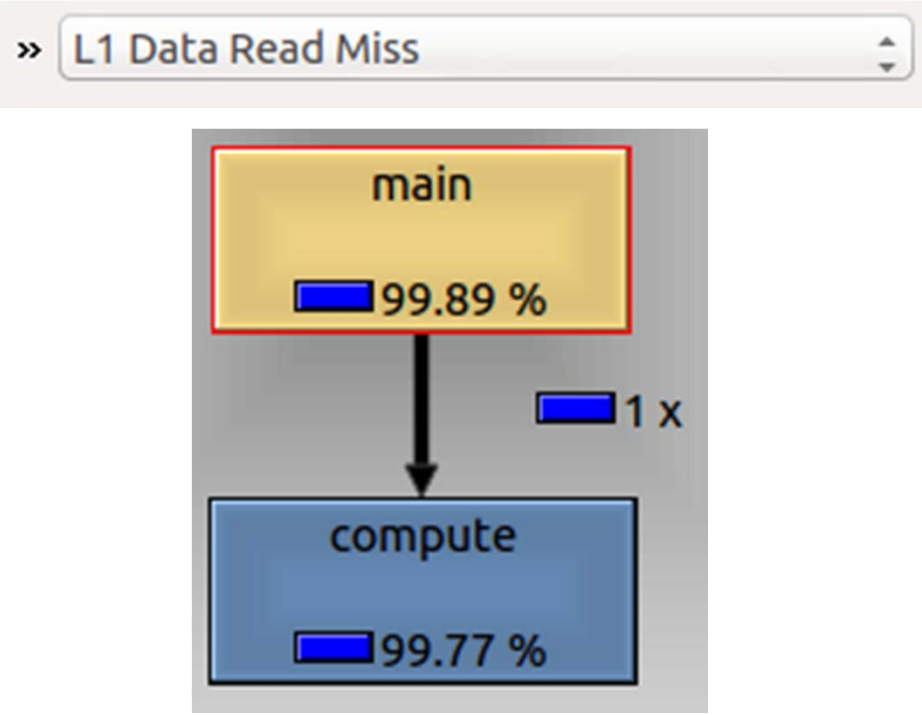
Ejemplo de Callgrind (II)



Callgrind + Kcachegrind

- Obtención del grafo de llamadas de forma gráfica.
  - Permite aplicar diversos criterios
- `> kcachegrind slow_out.dat`





Ejercicio 0: Información sobre la caché del sistema

- Inspeccionar las características de la caché del procesador mediante la línea de comandos.

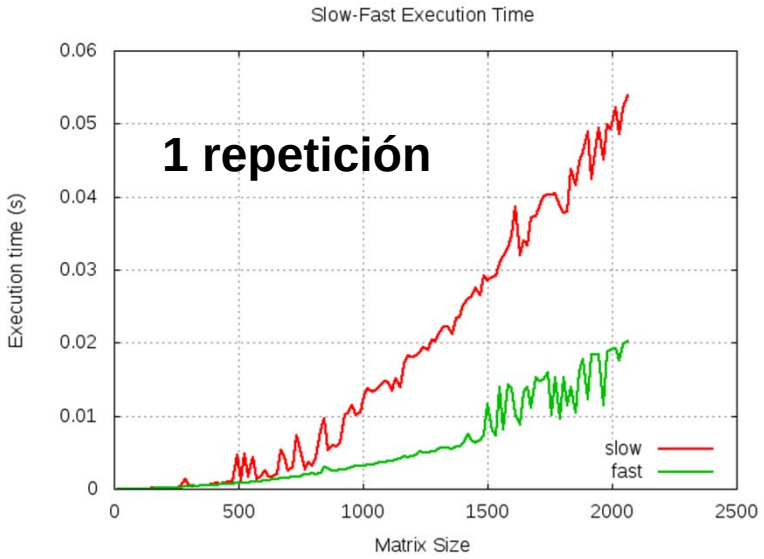
Ejercicio 1: Memoria caché y rendimiento

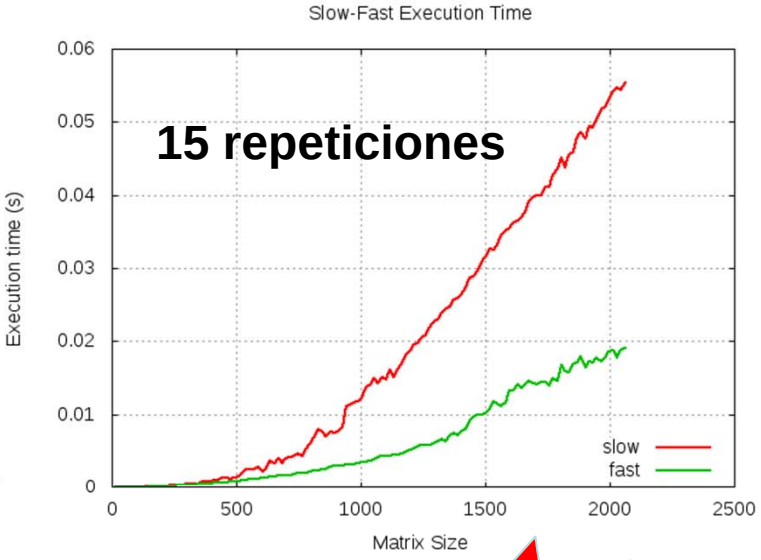
- Evaluar cómo el patrón de acceso a los datos puede mejorar el aprovechamiento de las memorias caché del sistema.
- Programas slow y fast:
  - suma de elementos de una matriz
  - diferentes patrones de acceso
- Realizar múltiples repeticiones intercaladas de cada prueba!!

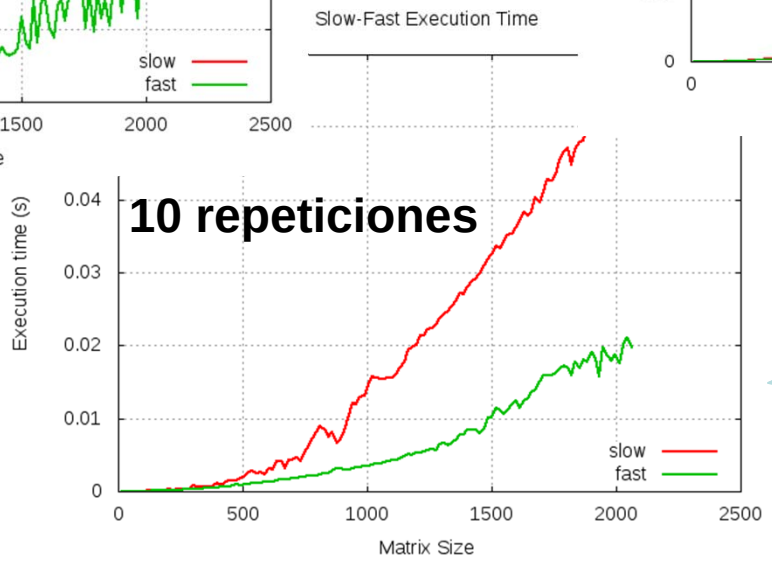
Correcto	**Incorrecto**
./slow N1	./slow N1
./slow N2	./slow N1
./fast N1	./slow N2
./fast N2	./slow N2
./slow N1	./fast N1
./slow N2	./fast N1
./fast N1	./fast N2
./fast N2	./fast N2

Ejercicio 1: Memoria caché y rendimiento

- Repetir las pruebas y hacer media de los resultados.







No olvidar leyenda, nombres en los ejes, unidades …

Ejercicio 2: Tamaño de caché y rendimiento

- Evaluar cómo la configuración de las memorias caché pueden afectar al rendimiento de un programa.

- Usaremos slow y fast con diferentes tamaños de matrix y de caché L1, que ajustaremos mediante Valgrind.

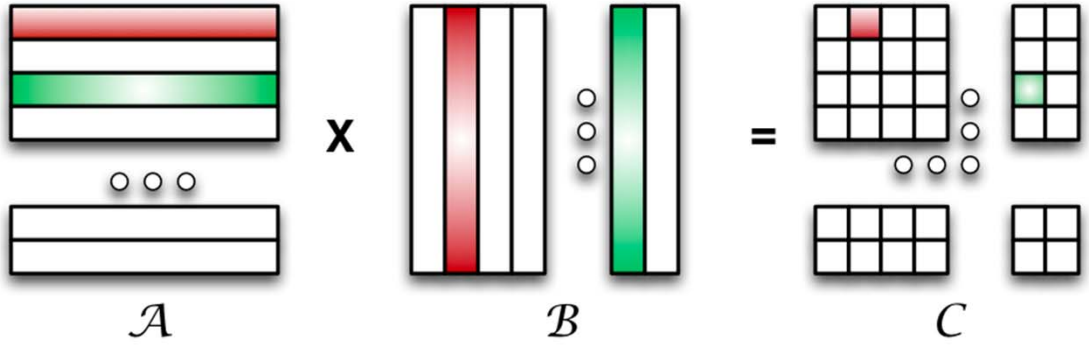
Ejercicio 3: Caché y multiplicación de matrices

- Desarrollar dos programas que multipliquen matrices usando diferentes patrones de acceso a memoria:
  - Multiplicación
  - Multiplicación traspuesta
- Estudiar los resultados de rendimiento y fallos de lectura/escritura en caché.

Pseudocódigo
Multiplicación de matrices

```
function compute( in: matrix a, matrix b, integer N,
                  out: matrix c):
  for i in 1..n do
      for j in 1..n do
          s = 0
          for k in 1..n) do
              s += a[i][k] * b[k][j]
          end do
          c[i][j] = s
      end do
  end do
```

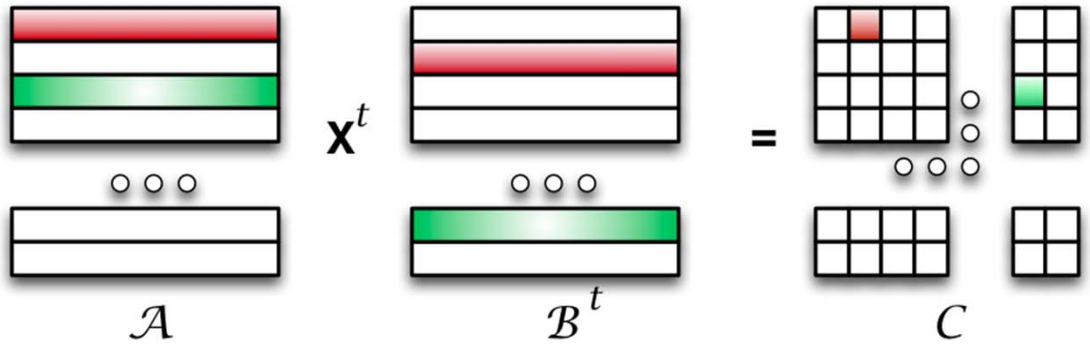
$$c_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b_{kj}$$



Pseudocódigo
Multiplicación Traspuesta

```
function compute( in: matrix a, matrix b^t, integer N,
                  out: matrix c):
  for i in 1..n do
      for j in 1..n do
          s = 0
          for k in 1..n) do
              s += a[i][k] * b^t[j][k]
          end do
          c[i][j] = s
      end do
  end do
```

$$c_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b^{t}_{jk}$$



Ejercicio 4: Opcional

- Configuraciones de Caché en la multiplicación de matrices
- “Jugar” con los diferentes parámetros de configuración de las cachés
  - Tamaño cache, asociatividad, longitud de línea de caché, tamaño de matriz, ¿otros?
- Resultados (ejemplos)
  - errores de lectura y escritura, tiempos de ejecución, ¿otros?
- Justificar experimentos realizados y explicar los resultados y gráficas obtenidos de forma razonada.

Recomendaciones

1. Entender que se pretende estudiar en cada ejercicio.

2. Guardar todos los resultados intermedios (ficheros de datos, logs, gráficos, scripts, etc). ¡Habrá que entregarlos!

3. Documentar los comandos que se utilicen.

4. Utilizar scripts para agilizar la obtención de resultados. Leer los tutoriales si nunca habéis usado scripts.

5. Preguntar al profesor lo que no quede claro (para eso está allí)