Clase 27. Uso de la memoria Cache

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Cache

Introducción

La memoria se copia por bloques de la memoria principal (RAM) a la cache L3 compartida por todos los cores, de esta al L2 y de ahi al L1. Las diferencias de velocidad de la memoria son de entre uno y dos ordenes de magnitud (uno con el L3, 2 con el L1). Utilizar variables que ya están en la cache L1, puede significar un ahorro considerable de tiempo. Todas las técnicas de optimización de la cache se basan en 2 cosas:

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Cache
cachegrind
Técnicas mas usuales para mejorar el uso del cache

Introducción

La memoria se copia por bloques de la memoria principal (RAM) a la cache L3 compartida por todos los cores, de esta al L2 y de ahi al L1. Las diferencias de velocidad de la memoria son de entre uno y dos ordenes de magnitud (uno con el L3, 2 con el L1). Utilizar variables que ya están en la cache L1, puede significar un ahorro considerable de tiempo. Todas las técnicas de optimización de la cache se basan en 2 cosas:

- Cercania temporal de la memoria que se usa. Ej. Reutilizar lo mas pronto posible una variable. Si ocupo x en alguna operación, lo mejor sería reutilizar todo lo que pudiera y realizar todas las operaciones posible con x antes de realizar operaciones con y, ya que el cache es una memoria muy pequeña, y y puede ocupar el lugar de x, lo que requeriria bajar x de la RAM cada que se utilice.

Introducción

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

La memoria se copia por bloques de la memoria principal (RAM) a la cache L3 compartida por todos los cores, de esta al L2 y de ahi al L1. Las diferencias de velocidad de la memoria son de entre uno y dos ordenes de magnitud (uno con el L3, 2 con el L1). Utilizar variables que ya están en la cache L1, puede significar un ahorro considerable de tiempo. Todas las técnicas de optimización de la cache se basan en 2 cosas:

- Cercania temporal de la memoria que se usa. Ej. Reutilizar lo mas pronto posible una variable. Si ocupo x en alguna operación, lo mejor sería reutilizar todo lo que pudiera y realizar todas las operaciones posible con x antes de realizar operaciones con y, ya que el cache es una memoria muy pequeña, y y puede ocupar el lugar de x, lo que requeriria bajar x de la RAM cada que se utilice.
- Cercania espacial. Ej. Si `x[4]` es un vector, dado que la memoria se copia por bloques o líneas, lo mas eficiente sería utilizar x[0], x[1], x[2] y x[3] consecutivamente (en el caso de paralelo tambien sería eficiente usarlos en el mismo core).

Línea de caché

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

La memoria se copia por líneas de y hacia la caché, si se utilizan datos en la misma línea no es necesario descargar de la RAM otra vez la copia de la memoria. Mientra más alejado esté un dato en la memoria más tarda la operación, es decir, no es lo mismo leer/escribir sobre x[1], x[2] que sobre x[1], x[1260]. Si los datos están muy alejados no dejará de importar su distancia ya que no se aprovechará para nada la línea de cache.
Ejemplo:

```
1 const int n=200000000;
  double *x= (double*)malloc(n*sizeof(double));
3 double iniTime=get_time();
    for (register int i=0; i<n; i++)
5         x[i]=5;
    printf("Tiempo de llenado consecutivo=%lf\n",
      get_time()-iniTime);
7   free(x);
```

El tiempo de ejecución de este código es de aproximadamente 0.27 seg.

Ej. saltos

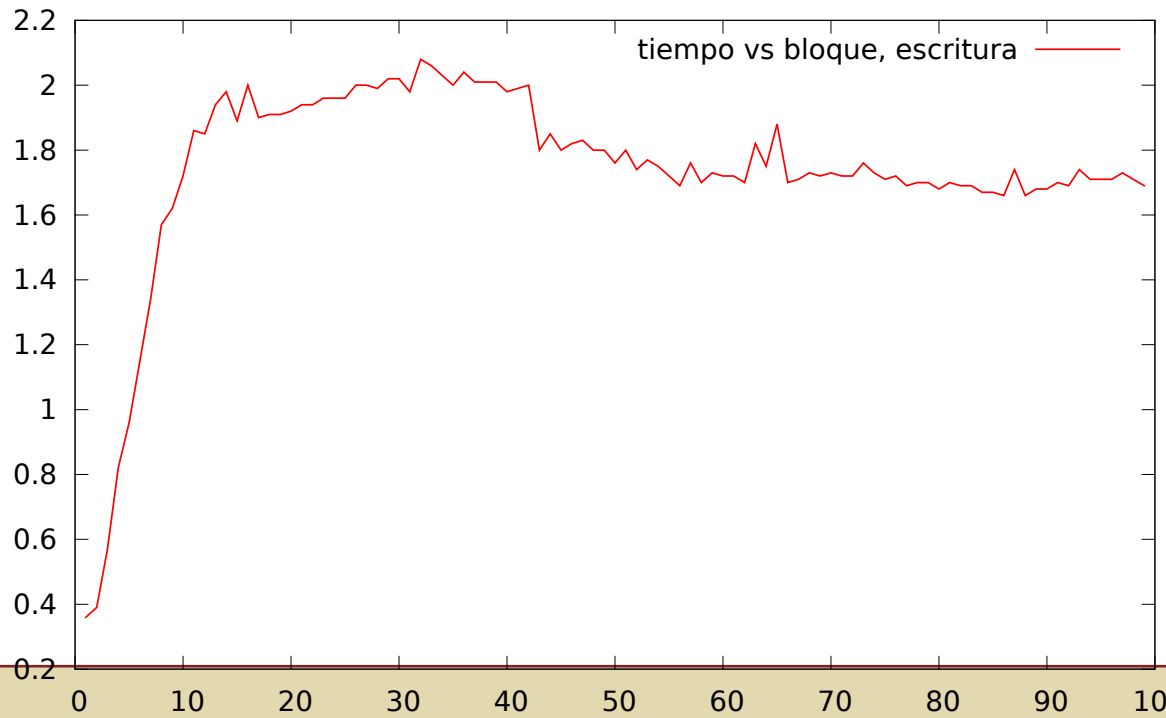
Se hacen las mismas operaciones (más o menos) dando **saltos**, para no acceder a memoria continua. Ejemplo de escritura.

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

```
1  for (register int nb=1; nb<100; nb++){
       double iniTime=get_time();
3      for (register int j=0; j<nb; j++)
         for (register int i=0; (i+j)<n; i+=nb)
5              x[i+j]=5;
       printf("%d %lf \n",nb, get_time()-iniTime);
7  }
```



Ej. saltos

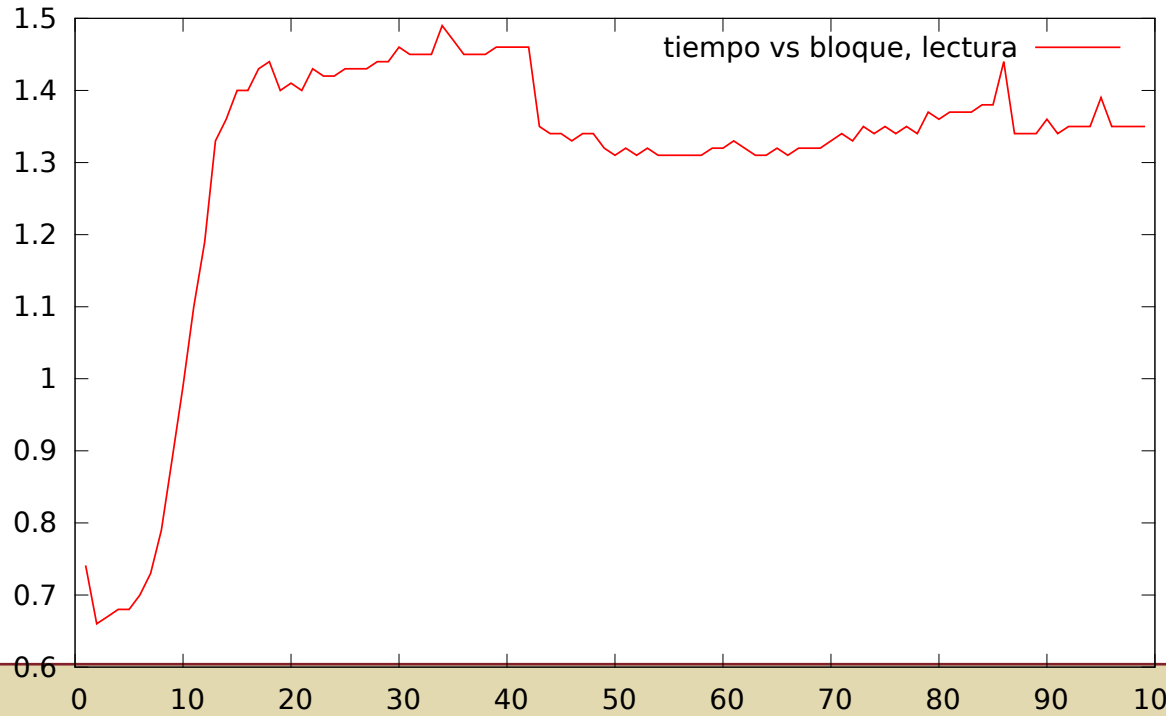
Se hacen las mismas operaciones (más o menos) dando **saltos**, para no acceder a memoria continua. Ejemplo de **lectura**.

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

```
1  for (register int nb=1; nb<100; nb++){
       double iniTime=get_time();       sum=0.0;
3       for (register int j=0; j<nb; j++)
          for (register int i=0; (i+j)<n; i+=nb)
5               sum+=x[i+j];
        printf("%d %lf \n",nb, get_time()-iniTime);
7  }
```



Ej. saltos

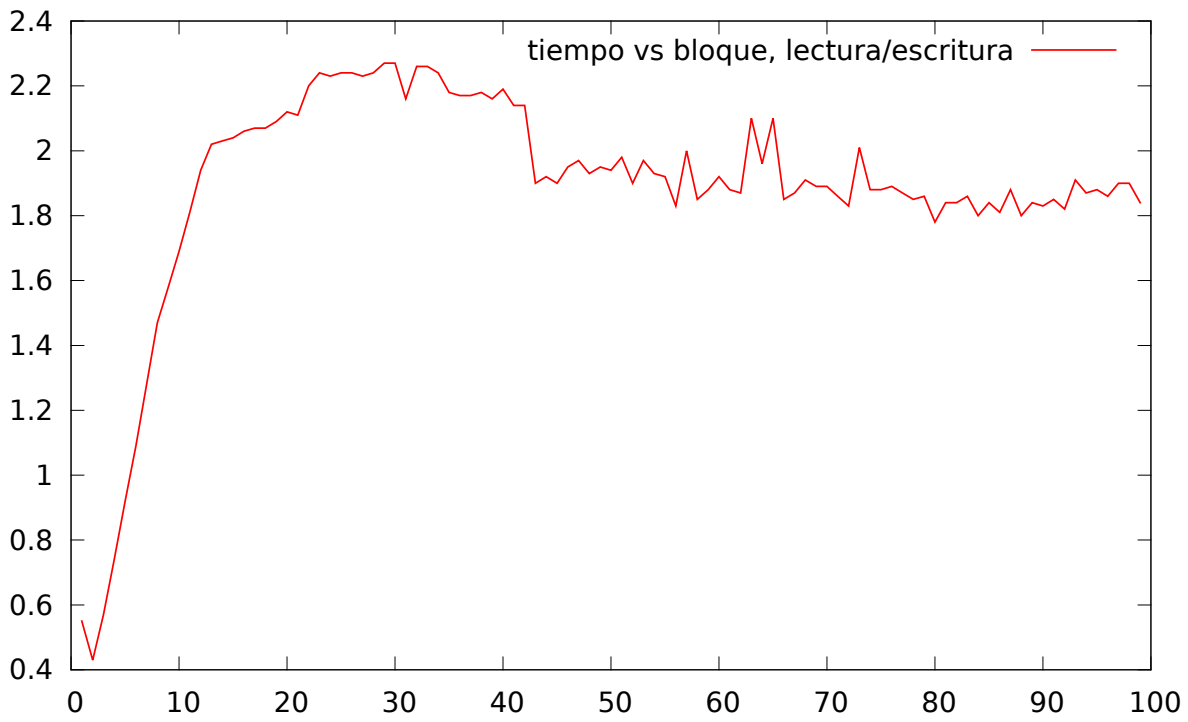
Ejemplo de **lectura/escritura**.

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

```
1 for (register int nb=1; nb<100; nb++){
      double iniTime=get_time();       sum=0.0;
3      for (register int j=0; j<nb; j++)
         for (register int i=0; (i+j)<n; i+=nb)
5             sum+=x[i+j];
       printf("%d %lf \n",nb, get_time()-iniTime);
7 }
```



Ej. cercania temporal

El calculo de los productos en Forma 1 tarda 0.95 seg. la Forma 2 0.41

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

```
#Forma 1
for (register int i=0; i<n; i++)
  px+=x[i]*x[i];
for (register int i=0; i<n; i++)
  py+=y[i]*y[i];
for (register int i=0; i<n; i++)
  pxy+=x[i]*y[i];
#Forma 2
for (register int i=0; i<n; i+=4){
px+=x[i]*x[i]+x[i+1]*x[i+1]+x[i+2]*x[i+2]+x[i+3]*x[i
    +3];
py+=y[i]*y[i]+y[i+1]*y[i+1]+y[i+2]*y[i+2]+y[i+3]*y[i
    +3];
pxy+=x[i]*y[i]+x[i+1]*y[i+1]+x[i+2]*y[i+2]+x[i+3]*y[i
    +3];
}
```

cachegrind

cachegrind

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Valgrind contiene una herramienta para analizar el comportamiento del cache, básicamente los cache hits y cache miss. El uso es el siguiente:

```
valgrind --tool=cachegrind prog
```

Donde prog es el ejecutable. Cachegrind simulara el programa y los accesos al primer y último cache (usualmente L1 y L3). Como se quiere simular el comportamiento real, se recomienda que prog ya sea la versión (compilado) optimizado del programa. Usualmente la simulación de cachegrind se tarda mucho mas que la ejecución normal del programa, se recomienda ejecutar con un ejemplo pequeño.

cachegrind

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Significado:

- I1= Cache de instrucciones del primer nivel (L1).
- D1= Cache de datos del primer nivel (D1).
- LL= Cache de último nivel (instrucciones y datos).
- Ir= lectura de instrucciones, I1mr= misses de lectura de intrucciones del cache 1. ILmr= misses de lectura de instrucciones del cache de último nivel.
- Dr= Lectura de datos, D1mr= misses de lectura de datos en el primer cache, DLmr= misses de lectura de datos en el último cache.
- Dw= Escritura de datos... lo mismo que los de arriba.

cachegrind

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Para algún programa entrega algo como esto.

```
==12035== I   refs:      1,975,017,358
==12035== I1  misses:              760
==12035== LLi misses:              753
==12035== I1  miss rate:          0.00%
==12035== LLi miss rate:          0.00%
==12035==
==12035== D   refs:        713,229,506  (497,049,338 rd   + 216,180
==12035== D1  misses:       27,023,905  (  2,023,359 rd   +  25,000
==12035== LLd misses:       27,023,482  (  2,022,972 rd   +  25,000
==12035== D1  miss rate:           3.7% (        0.4%     +
==12035== LLd miss rate:           3.7% (        0.4%     +
==12035==
==12035== LL refs:          27,024,665  (  2,024,119 rd   +  25,000
==12035== LL misses:        27,024,235  (  2,023,725 rd   +  25,000
==12035== LL miss rate:            1.0% (        0.0%     +
```

cg_annotate

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Además de la información en pantalla, la cual es un resumen de la ejecución, cachegrind genera un archivo con información mas detallada, que tiene mas o menos la siguiente sintaxis:

```
cachegrind.out.12136
```

Donde 12136 es el ID del proceso.

Este archivo se puede visualizar con otra aplicación: cg_annotate.

cg_annotate

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

Se pueden extraer datos de este archivo por medio de **cg_annotate**

```
cg_annotate cachegrind.out.12136
```

Lo primero que despliega es información del procesador:

```
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         6291456 B, 64 B, 12-way associative
```

Tamaño, línea, y asociatividad. Un cache es completamente asociativo si una dirección de la RAM puede copiarse en cualquier posición del cache. Si solo se puede copiar en N posiciones, entonces el cache es N-asociativo.

cg_annotate

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

ejercicio1a: productos puntos

```
     Ir  I1mr  ILmr      Dr  D1mr  DLmr      Dw D1mw DLmw
--------------------------------------------------------------------------------
191,490 1,035 1,024 44,745 2,467 1,952 17,745  819  745  PROGRAM TO

--------------------------------------------------------------------------------
    Ir I1mr ILmr     Dr  D1mr DLmr    Dw D1mw DLmw  file:function
--------------------------------------------------------------------------------
 9,029    3    3      3     1    0 2,009  248  248  ???:ejercicio1a
 9,029    3    3      3     0    0 2,009    6    0  ???:ejercicio1b
```

Linea de ejecución para que solo muestre cierta información (no corresponde a lo de arriba):

```
cg_annotate --show=Dr,Dw --threshold=0.8  cachegrind.out.12136
```

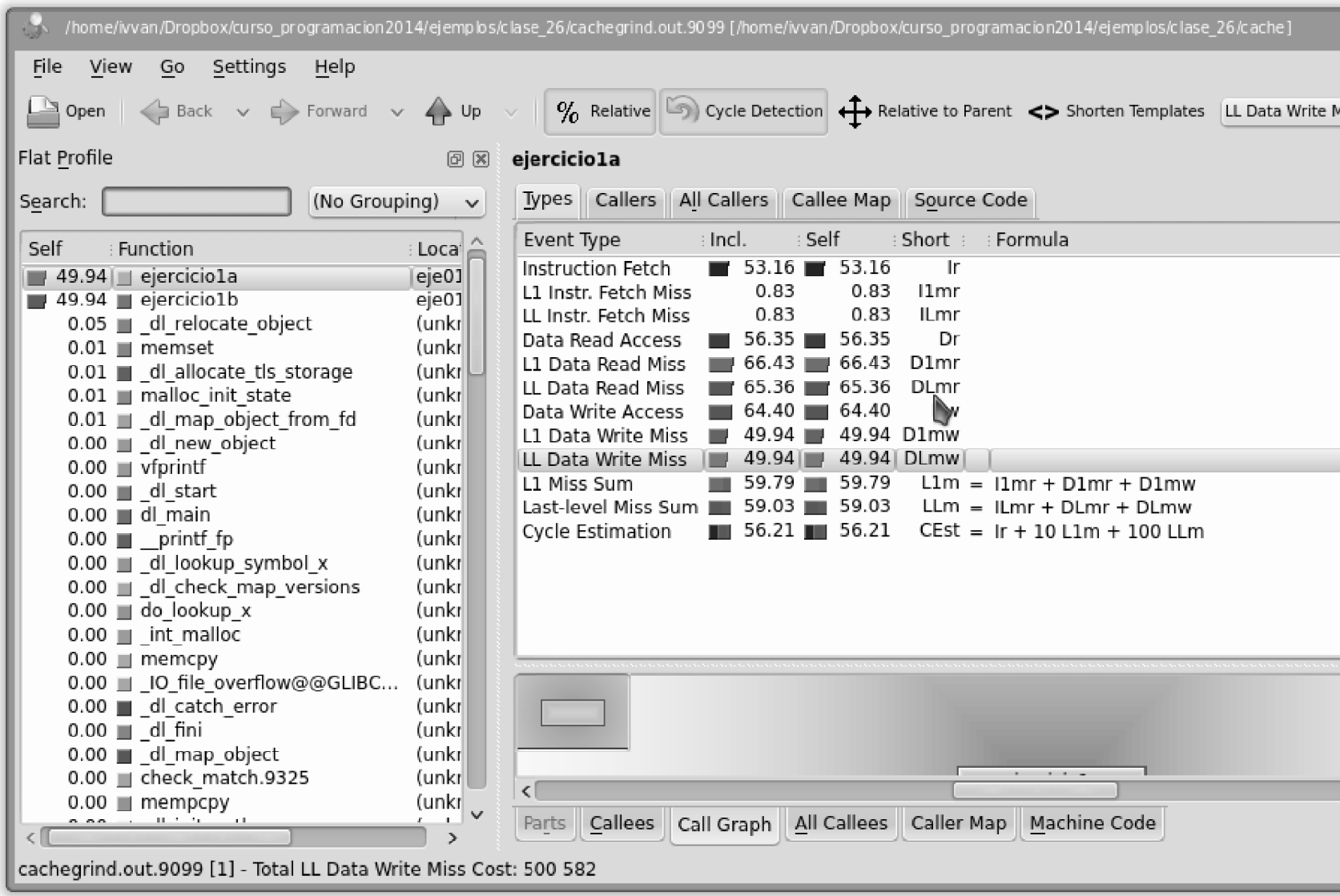
KCacheGrind

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

KCacheGrind es una herramienta para linux/KDE para visualizar la salida de cachegrind, que facilita muchisimo el análisis del código.



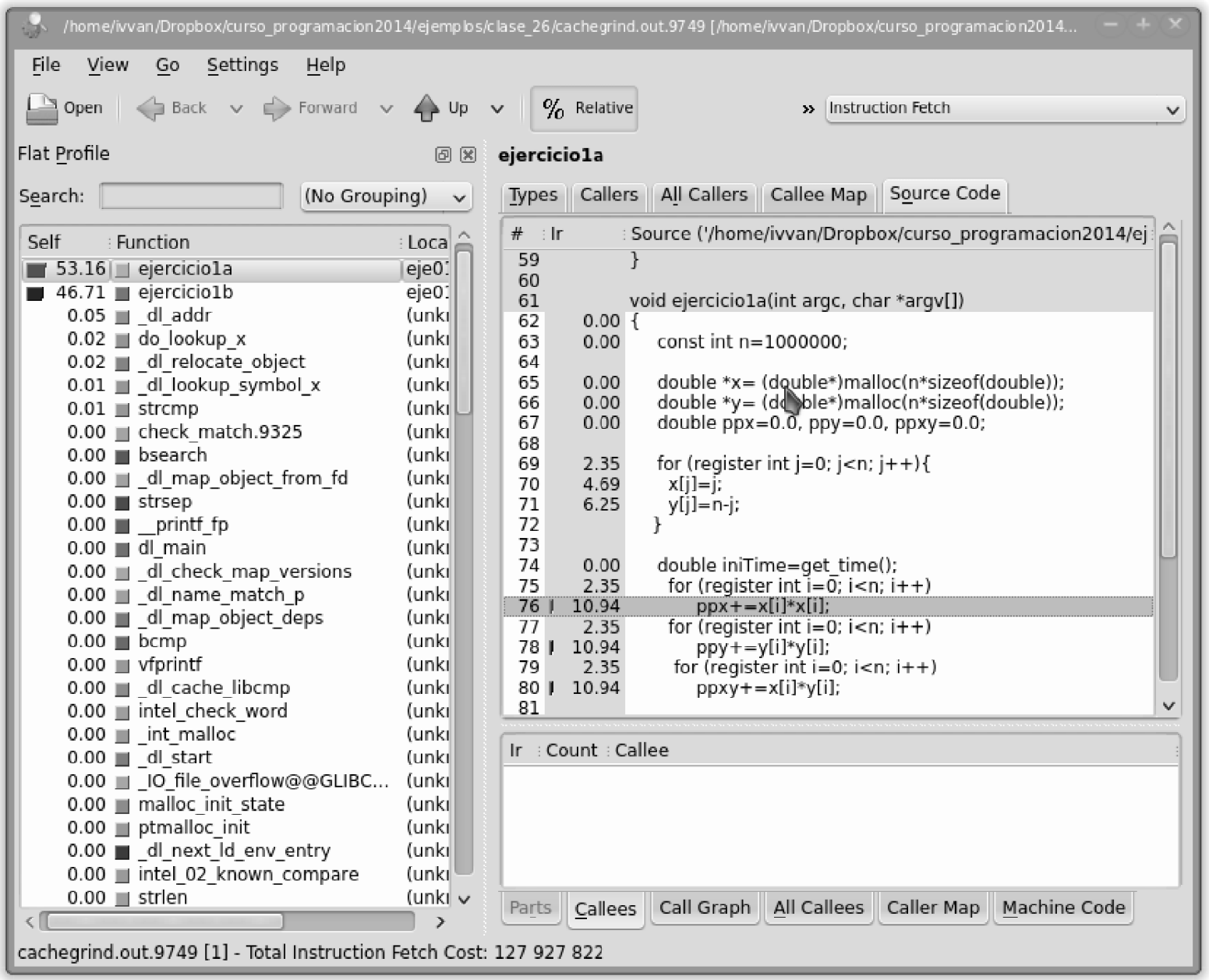
KCacheGrind

Para desplegar toda la información esto si debe de ser compilado con -g.

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache



Técnicas mas usuales para mejorar el uso del cache

Intercambio de loops

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

❖

Kowarschik, Markus, and Christian Weiß.
"An overview of cache optimization techniques and cache-aware numerical algorithms." Algorithms for Memory Hierarchies. Springer Berlin Heidelberg, 2003. 213-232.

Intercambiar los loops para acceder a los datos en el sentido del bloque de datos:

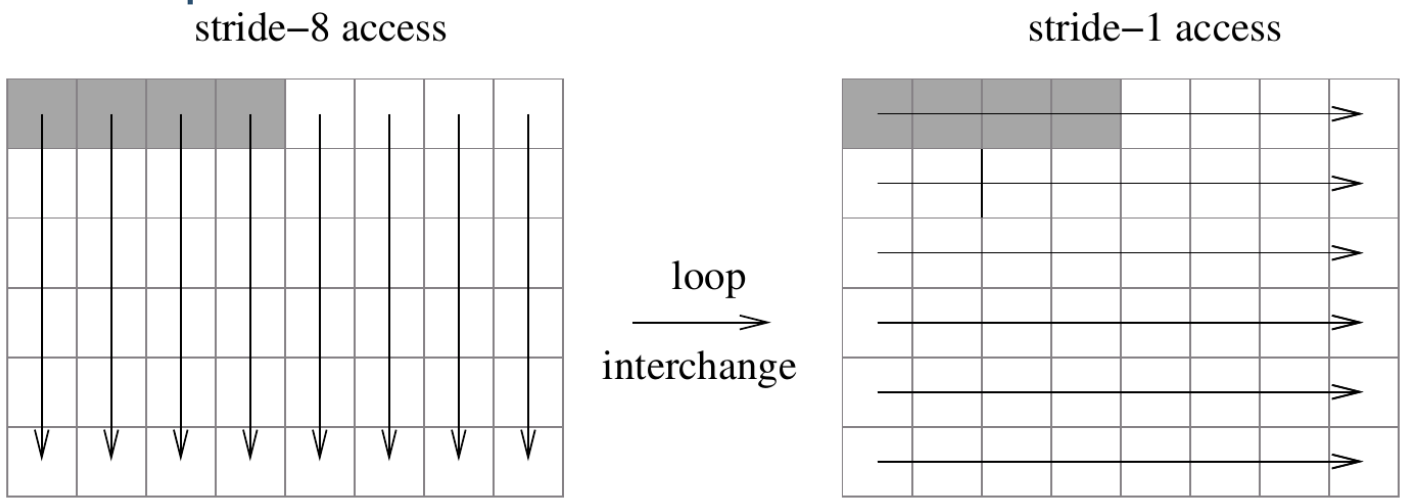


Fig. 2. Access patterns for interchanged loop nests.

Intercambio de loops


```
1  void ejercicio2(int argc, char *argv[])
   {
3   const int nrow=20000,ncol=20000;
    double **M=matrix(nrow,ncol);;
5   double sum=0.0;

7   double iniTime=get_time();
    for (register int i=0; i<nrow; i++)
9     for (register int j=0; j<ncol; j++) sum+=M[i][j];
    printf("Tiempo renglon %lf\n",get_time()-iniTime);
11
    iniTime=get_time();
13  for (register int j=0; j<ncol; j++)
      for (register int i=0; i<nrow; i++) sum+=M[i][j];
15  printf("Tiempo columnas %lf\n",get_time()-iniTime);

17  free_matrix(M,nrow);
   }
19
```

Diferencia de tiempos: renglones=1.19 seg, columnas 5.62 seg.

Fusión de loops

Mejora el loop overhead, la paralalelización , y la espacialidad de los datos, reduciendo los misses.
Fission=0.38seg. Fussion=0.23seg.

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

❖


```
    const int n=50000000;
2   double *x= (double*)malloc(n*sizeof(double));
    double *y= (double*)malloc(n*sizeof(double));
4   double *z= (double*)malloc(n*sizeof(double));
    double iniTime=get_time();
6   for (int i=0; i<n; i++)
        x[i]=y[i]+1.0;
8   for (int i=0; i<n; i++)
        z[i]+=x[i]*4.0;
10  printf("loop fission %lf \n", get_time()-iniTime);
    iniTime=get_time();
12  for (int i=0; i<n; i++){
      x[i]=y[i]+1.0;
14    z[i]+=x[i]*4.0;
    }
16  printf("loop fussion %lf \n", get_time()-iniTime);
    free(x); free(y); free(z);
18
```

Loop Blocking

Se intenta aprovechar los datos que ya han bajado a la cache. normal =0.26seg. por bloques=0.02seg.


```
1  const int n=5000;
   const int B=12;
3  double **a= matrix(n,n);
   double **b= matrix(n,n);
5
   double iniTime=get_time();
7    for (int i=0; i<n; i++)
       for (int j=0; j<n; j++)
9          a[i][j]=b[j][i];
   printf("transpuesta normal %lf\n",get_time()-iniTime);
11
   iniTime=get_time();
13   for (int ii=0; ii<n; ii+=B)
       for (int jj=0; jj<n; jj+=B)
15       for (int i=ii; i<min(ii+B-1,n); i++)
   for (int j=jj; j<min(jj+B-1,n); j+=B)
17               a[i][j]=b[j][i];
   printf("transpuesta bloques %lf\n",get_time()-iniTime);
19 free_matrix(a,n); free_matrix(b,n);
```

Organización de los datos

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

❖

.

- Array padding: Si dos arreglos son mapeados a la misma línea de cache y se acceden alternadamente, se pueden producir un gran número de misses, debido a que un arreglo sobreescribe los datos del otro. Una solución es insertar memoria intermedia en la RAM para que los arreglos mapeen a diferentes locaciones del cache.

Organización de los datos

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

❖

.

- Array padding: Si dos arreglos son mapeados a la misma línea de cache y se acceden alternadamente, se pueden producir un gran número de misses, debido a que un arreglo sobreescribe los datos del otro. Una solución es insertar memoria intermedia en la RAM para que los arreglos mapeen a diferentes locaciones del cache.
- Array merging. Supongamos que almaceno una matriz rala como un conjunto de valores e indices. Por lo general, para operaciones con matrices utilizaría el valor y el indice al mismo tiempo, entonces es lógico bajarlos a la misma linea de cache, esto se puede lograr con una estructura que contenga un valores y los indices y solicitando un arreglo de esta estructura.

Organización de los datos

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

❖

.

- Transposición del arreglo. Similar al intergambio de loop, se puede transponer una matríz (por ejemplo para la multiplicación) y mejorar el acceso desde el arreglo de los datos.

Organización de los datos

Cache

cachegrind

Técnicas mas usuales para mejorar el uso del cache

❖

.

- Transposición del arreglo. Similar al intergambio de loop, se puede transponer una matríz (por ejemplo para la multiplicación) y mejorar el acceso desde el arreglo de los datos.
- Copia de datos. Similar al problema del primer punto. En algoritmos por bloques (como la transpuesta), como usualmente RAM contigua mapea al mismo bloque de cache, puede ser que dos partes del mismo bloque definido en la RAM estén mapeando al mismo bloque de cache produciendo misses, una posible solucion es tener una copia del bloque y usar una parte de uno y una parte de otro.

Terminamos.

¡Gracias!