

Trabajo práctico 2:

Arquitecturas de memoria

Roberto Herman, *Padrón Nro. 84.803*
`berta1108@gmail.com`

Matías Waisgold, *Padrón Nro. 88.464*
`mwaisgold@gmail.com`

Nicolás Suarez, *Padrón Nro. 83.752*
`nicolas.suarez@gmail.com`

Federico Rodriguez, *Padrón Nro. 88.310*
`fedel1726@gmail.com`

2do. Cuatrimestre de 2009

66.20 Organización de Computadoras

Facultad de Ingeniería, Universidad de Buenos Aires

1. Introducción

El presente trabajo práctico pretende familiarizar al alumno con el uso de herramientas de profiling. El objetivo del trabajo será determinar el tamaño de bloque, la cantidad de vías y el tamaño total de una memoria cache L1 de datos. Utilizaremos el módulo Cachegrind de la herramienta Valgrind[1], que nos permite simular diferentes memorias cache y obtener las tasas de misses a partir de las cuales podremos determinar los datos de la cache.

2. Diseño e implementación

2.1. Módulo principal

El módulo principal se encarga de ejecutar cada uno de los módulos y obtener la tasa de misses para realizar el cálculo de la cache.

2.2. Tamaño del Bloque

Para el cálculo del tamaño del bloque se ejecuta el módulo *tamanoBloque*. Este módulo realiza un lazo escribiendo un array de tamaño 1024 KB. Al finalizar el lazo habremos accedido a todos los bloques de la cache, por lo cual obteniendo la cantidad de accesos por escritura (dw) y la tasa de misses de escritura (d1mw) de la cache, podemos calcular el tamaño del bloque haciendo el siguiente cálculo:

$$\text{sizeBloque} = \text{dw}/\text{d1mw}.$$

2.3. Tamaño total de la Cache

Para el cálculo del tamaño total de la cache lo que hacemos es determinar la cantidad total de bloques, luego multiplicando esa cantidad por el tamaño del bloque obtenemos el tamaño de la cache.

Para calcular la cantidad de bloques ejecutamos el módulo *tamanoCache*. Este módulo recibe dos parámetros: el tamaño del bloque (*sizeBloque*) y un valor n que es la cantidad hipotética de bloques de la cache. El módulo realiza un lazo dos veces escribiendo un array cuyo tamaño es $\text{sizeBloque} * n$.

Luego de ejecutar el módulo se obtiene la tasa de misses de escritura, si esa cantidad es menor o igual a n no se ha recorrido todos los bloques por lo cual se vuelve a ejecutar el módulo duplicando el valor de n .

2.4. Cantidad de vías

Para determinar la cantidad de vías lo que hacemos es recorrer un lazo de una matriz de char, la cual tendrá tantas columnas como el tamaño de la cache y tantas filas como la cantidad de vías que suponemos que tiene (inicialmente una).

Al finalizar el lazo, volvemos a acceder a las dos primeras posiciones de cada una de las filas de la matriz, es decir que estaríamos accediendo al primer bloque de cada vía. Si en los dos últimos accesos se producen misses en la cache quiere decir que hemos accedido a todos los bloques de cada una de las vías, sino la cantidad de vías es mayor, por lo cual duplicamos el valor la cantidad de vías y volvemos a ejecutar el lazo.

3. Compilación del benchmark

Para compilar el benchmark se debe correr el makefile. Desde la consola se debe entrar al directorio donde se encuentran el código fuente y el makefile, y luego ejecutar el comando *make*:

```
$> make
```

Al finalizar la compilación se habrán generado cuatro archivos ejecutables:

- tp2 (Este es el programa principal)
- tamanoBloque
- tamanoCache
- cantidadVias

4. Instrucciones de uso

El benchmark se puede utilizar sobre una cache simulada o sobre la cache propia de la maquina que se esté utilizando.

- Para el caso de una cache simulada se debe pasar como parámetro el tamaño de total, la cantidad de vias y tamaño del bloque de la cache a simular. Por ejemplo, para correr el benchmark sobre una cache simulada de 2 vias, con tamaño de bloque de 32 bytes y tamaño total de 32768 bytes:

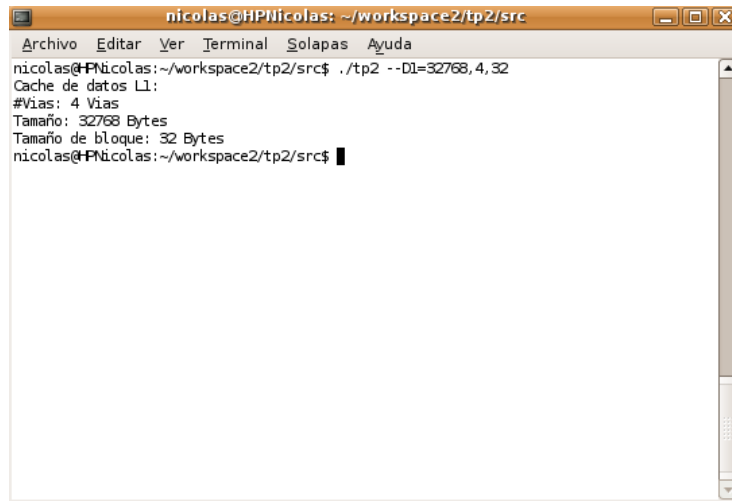
```
$> ./tp2 --D1=32768,2,32
```

- Para el segundo caso simplemente debemos ejecutar el programa sin parámetros:

```
$> ./tp2
```

5. Corridas de prueba

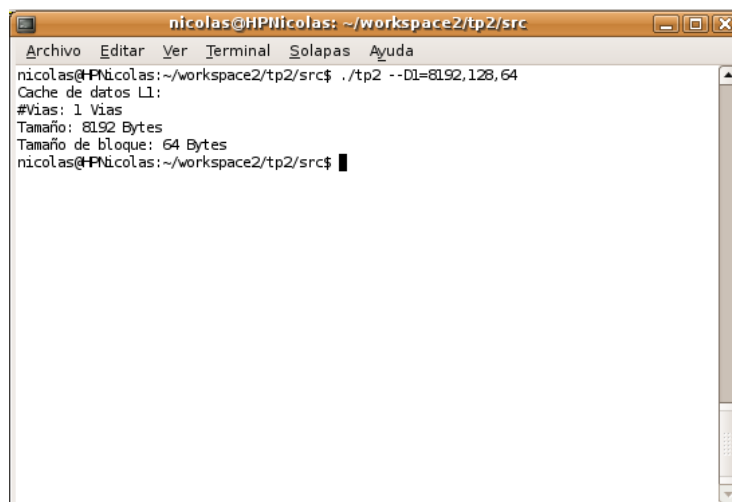
- Prueba 1: Simulando una cache de 32.768 bytes, 4 vias, 32 bytes de línea.



```
nicolas@HPNicolas: ~/workspace2/tp2/src
Archivo Editar Ver Terminal Solapas Ayuda
nicolas@HPNicolas:~/workspace2/tp2/src$ ./tp2 --D1=32768,4,32
Cache de datos L1:
#Vias: 4 Vias
Tamaño: 32768 Bytes
Tamaño de bloque: 32 Bytes
nicolas@HPNicolas:~/workspace2/tp2/src$
```

Figura 1: `./tp2 -D1==32768,4,32`

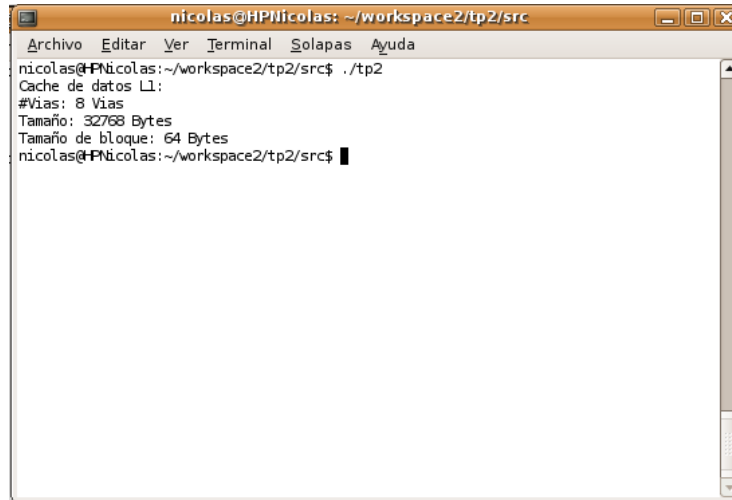
- Prueba 2: Simulando una cache Full Associative de 8.192 bytes y 32 bytes de línea.



```
nicolas@HPNicolas: ~/workspace2/tp2/src
Archivo Editar Ver Terminal Solapas Ayuda
nicolas@HPNicolas:~/workspace2/tp2/src$ ./tp2 --D1=8192,128,64
Cache de datos L1:
#Vias: 1 Vias
Tamaño: 8192 Bytes
Tamaño de bloque: 64 Bytes
nicolas@HPNicolas:~/workspace2/tp2/src$
```

Figura 2: `./tp2 -D1==8192,128,64`

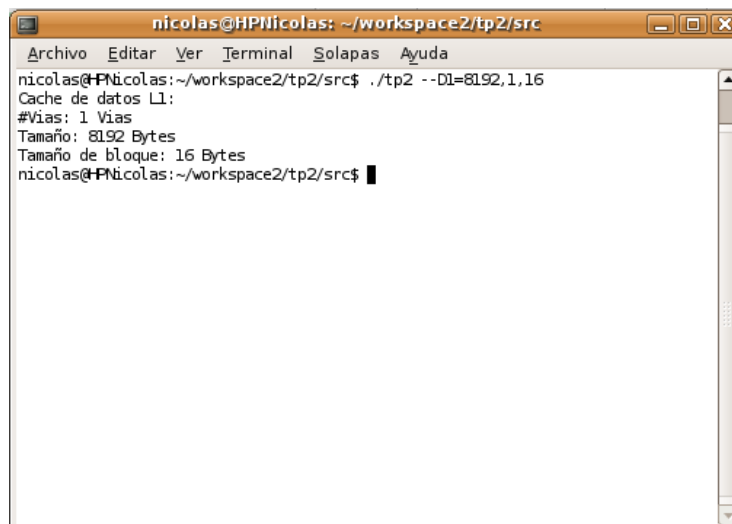
- Prueba 3: Sin simular cache.



```
nicolas@HPNicolas: ~/workspace2/tp2/src
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
nicolas@HPNicolas:~/workspace2/tp2/src$ ./tp2
Cache de datos L1:
#Vias: 8 Vias
Tamaño: 32768 Bytes
Tamaño de bloque: 64 Bytes
nicolas@HPNicolas:~/workspace2/tp2/src$
```

Figura 3: ./tp2

- Prueba 4: Cache Direct Mapped, 16Bytes de línea, 8Kb tamaño total



```
nicolas@HPNicolas: ~/workspace2/tp2/src
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
nicolas@HPNicolas:~/workspace2/tp2/src$ ./tp2 -D1=8192,1,16
Cache de datos L1:
#Vias: 1 Vias
Tamaño: 8192 Bytes
Tamaño de bloque: 16 Bytes
nicolas@HPNicolas:~/workspace2/tp2/src$
```

Figura 4: ./tp2 -D1=8192,1,16

6. Conclusiones

El trabajo práctico nos permitió conocer las características y funcionamiento de la memoria cache. A su vez nos hemos familiarizado con la herramienta de profiling Valgrind, la cual nos permite saber, al correr una aplicación, en que partes de esta se esta accediendo a la memoria cache y obtener la tasa de misses, y en base a esta información podemos mejorar la aplicación para obtener un mayor rendimiento.

Referencias

- [1] Valgrind: <http://valgrind.org/>

A. Apéndice: Código Fuente

A.1. Módulo Principal: tp2.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <getopt.h>

#define TAMANIO_BLOQUE "tamanioBloque"
#define TAMANIO_CACHE "tamanioCache"
#define CANT_VIAS "cantidadVias"
#define CACHEGRIND "valgrind --tool=cachegrind --log-file=salidaValgrind\n--cachegrind-out-file=salidaCachegrind "
#define CG_ANNOTATE "cg_annotate --auto=yes --show=Dr,Dw,D1mr,D1mw salidaCachegrind "
#define FILE_OUTPUT_CACHEGRIND "salidaCachegrind"
#define FILE_OUTPUT_CGANNOTATE "salidaCgannotate"
#define FILE_OUTPUT_GREP "salidaGrep"
#define FILE_OUTPUT_VALGRIND "salidaValgrind"
#define SIZE_BUFFER 512

using std::string;

void imprime_uso (FILE *output)
{
    fprintf(output, "Usage:\n");
    fprintf(output, "    tp2 -h\n"
        "    tp2 -V\n"
        "    tp2 [Cache data]\n"
        "Options:\n"
        "    -h, --help      Imprime ayuda.\n"
        "    -V, --version   Version del programa.\n"
        "Examples:\n"
        "    tp2 --D1=16384,2,64\n");
}

void imprime_version()
{
    printf("Version [66.20] Organizacion de Computadoras\n"
        "Segundo Cuatrimestre 2009\n");
}

bool datosCacheValidos(char* datos)
{
    int i= 5;
    char c;
```

```

c= datos[i];
while(c != ','){
    if(c == '\0')
        return false;

    if((c == '0') || (c == '1') || (c == '2') ||
        (c == '3') || (c == '4') || (c == '5') ||
        (c == '6') || (c == '7') || (c == '8') || (c == '9'))
    {
        i++;
        c= datos[i];
    }
    else{
        return false;
    }
}

i++;
c= datos[i];

while(c != ','){
    if(c == '\0')
        return false;

    if((c == '0') || (c == '1') || (c == '2') ||
        (c == '3') || (c == '4') || (c == '5') ||
        (c == '6') || (c == '7') || (c == '8') || (c == '9'))
    {
        i++;
        c= datos[i];
    }
    else{
        return false;
    }
}

i++;
c= datos[i];

if((c == '0') || (c == '1') || (c == '2') ||
    (c == '3') || (c == '4') || (c == '5') ||
    (c == '6') || (c == '7') || (c == '8') || (c == '9'))
{
    i++;
    c= datos[i];
}
else{
    return false;
}

```



```

    }

    while(c != '\0'){
        if((c == '0') || (c == '1') || (c == '2') ||
            (c == '3') || (c == '4') || (c == '5') ||
            (c == '6') || (c == '7') || (c == '8') || (c == '9'))
        {
            i++;
            c= datos[i];
        }
        else{
            return false;
        }
    }

    return true;
}

void replaceText(string &word, const string &toReplace, const string &replaceBy)
{
    string::size_type posToRepleace = word.find(toReplace);
    while(posToRepleace != string::npos)
    {
        string tmpWord = word;
        if (posToRepleace > 0)
        {
            word = tmpWord.substr(0, posToRepleace);
            word += replaceBy;
            word += tmpWord.substr(posToRepleace + toReplace.size(), tmpWord.size() - 1);
        }
        else
        {
            word = replaceBy;
            word += tmpWord.substr(posToRepleace + toReplace.size(), tmpWord.size() - 1);
        }
        posToRepleace = word.find(toReplace);
    }
}

string intToString( int entero )
{
    std::stringstream cadena("");
    cadena << entero;
    return cadena.str();
}

void parsearDatos(const string &funcion, long int &dr, long int &dw,
                  long int &d1mr, long int &d1mw)
{

```

```

string grep("grep " + funcion + " " + FILE_OUTPUT_CGANNOTATE + " > "
            + FILE_OUTPUT_GREP);
system(grep.c_str());
std::ifstream fileGrep;
fileGrep.open(FILE_OUTPUT_GREP);
if (!fileGrep.good())
{
    std::cerr << "No se pudo parsear la salida del cachegrind\n" << std::endl;
    return;
}
char buffer[SIZE_BUFFER];
memset(buffer, '\0', SIZE_BUFFER);
fileGrep.getline(buffer, SIZE_BUFFER);
if(strlen(buffer)==0)
{
    fileGrep.close();
    std::cerr << "No se pudo parsear la salida del cachegrind\n" << std::endl;
}
string word(buffer);
int posDesde = 0, posHasta = 0;
posDesde = word.find_first_not_of(" ", posDesde);
posHasta = word.find(" ", posDesde);
string strDr = word.substr(posDesde, posHasta-posDesde);
replaceText(strDr, ",", "");
dr = atol(strDr.c_str());

posDesde = word.find_first_not_of(" ", posHasta);
posHasta = word.find(" ", posDesde);
string strDw = word.substr(posDesde, posHasta-posDesde);
replaceText(strDw, ",", "");
dw = atol(strDw.c_str());

posDesde = word.find_first_not_of(" ", posHasta);
posDesde = word.find(" ", posDesde);
string strD1mr = word.substr(posDesde, posHasta-posDesde);
replaceText(strD1mr, ",", "");
d1mr = atol(strD1mr.c_str());

posDesde = word.find_first_not_of(" ", posHasta);
posDesde = word.find(" ", posDesde);
string strD1mw = word.substr(posDesde, posHasta-posDesde);
replaceText(strD1mw, ",", "");
d1mw = atol(strD1mw.c_str());
fileGrep.close();
}

int tamanoBloque(char *datosCache)
{
    string cachegrind(CACHEGRIND);
    if(datosCache != NULL)

```

```

        cachegrind.append(datosCache);
cachegrind.append(" ./");
cachegrind.append(TAMANIO_BLOQUE);
system(cachegrind.c_str());

string cannotate(CG_ANNOTATE);
cannotate.append(TAMANIO_BLOQUE);
cannotate.append(".cpp");
cannotate.append(" > ");
cannotate.append(FILE_OUTPUT_CGANNOTATE);
system(cannotate.c_str());

long int dr=0, dw=0, d1mr=0, d1mw=0;
string fileFuente(TAMANIO_BLOQUE);
fileFuente.append(".cpp:");
fileFuente.append(TAMANIO_BLOQUE);
parsearDatos(fileFuente, dr, dw, d1mr, d1mw);
int sizeBloque = 0;
if(dw>0 && d1mw>0)
    sizeBloque = dw/d1mw;
return sizeBloque;
}

int tamanioCache(char *datosCache, int tamanioBloque)
{
    int n = 128;
    long int dr=0, dw=0, d1mr=0, d1mw=0;
    while(d1mw <= n)
    {
        n = n*2;
        string cachegrind(CACHEGRIND);
        if(datosCache != NULL)
            cachegrind.append(datosCache);
        cachegrind.append(" ./");
        cachegrind.append(TAMANIO_CACHE);
        cachegrind.append(" " + intToString(tamanioBloque) + " " + intToString(n));
        system(cachegrind.c_str());

        string cannotate(CG_ANNOTATE);
        cannotate.append(TAMANIO_CACHE);
        cannotate.append(".cpp");
        cannotate.append(" > ");
        cannotate.append(FILE_OUTPUT_CGANNOTATE);
        system(cannotate.c_str());

        string fileFuente(TAMANIO_CACHE);
        fileFuente.append(".cpp:");
        fileFuente.append(TAMANIO_CACHE);
        parsearDatos(fileFuente, dr, dw, d1mr, d1mw);
    }
}

```

```

        return (n/2 * tamanoBloque);
    }

int cantidadVias(char *datosCache, int sizeCache, int sizeBloque)
{
    int n = 1;
    long int dr=0, dw=0, d1mr=0, d1mw=0;
    while(d1mw == 0)
    {
        string cachegrind(CACHEGRIND);
        if(datosCache != NULL)
            cachegrind.append(datosCache);
        cachegrind.append(" ./");
        cachegrind.append(CANT_VIAS);
        cachegrind.append(" " + intToString(n) + " " + intToString(sizeCache)
            + " " + intToString(sizeBloque));
        system(cachegrind.c_str());

        string cgannotate(CG_ANNOTATE);
        cgannotate.append(CANT_VIAS);
        cgannotate.append(".cpp");
        cgannotate.append(" > ");
        cgannotate.append(FILE_OUTPUT_CGANNOTATE);
        system(cgannotate.c_str());

        string fileFuente("'vector\\[j\\]\\[0\\]'");
        parsearDatos(fileFuente, dr, dw, d1mr, d1mw);
        n = n*2;
    }
    return (n/4);
}

int main(int argc, char* argv[])
{
    char *datosCache = NULL;
    int siguiente_opcion = 0;

    /* Una cadena que lista las opciones cortas validas */
    const char* const op_cortas = "hVd:" ;

    /* Una estructura de varios arrays describiendo los valores largos */
    const struct option op_largas[] =
    {
        { "help",          0, NULL,   'h'},
        { "version",        0, NULL,   'V'},
        { "D1",             1, NULL,   'd'},
        { NULL,              0, NULL,   0 }
    };

    while (siguiente_opcion != -1)

```

```

{
    /* Llamamos a la funcion getopt */
    siguiente_opcion = getopt_long (argc, argv, op_cortas, op_largas, NULL);

    switch (siguiente_opcion)
    {
        case 'h' : /* -h o --help */
            imprime_uso(stdout);
            exit(EXIT_SUCCESS);

        case 'V' : /* -V o --version*/
            imprime_version();
            exit(EXIT_SUCCESS);

        case 'd' : /* --D1 */
            bool datos;
            datos= datosCacheValidos(argv[1]);
            if(datos == false){
                imprime_uso(stdout);
                exit(EXIT_SUCCESS);
            }
            datosCache = argv[1];
            break;

        case '?' : /* opcion no valida */
            imprime_uso(stdout);
            exit(EXIT_SUCCESS);

        case -1 : /* No hay mas opciones */
            break;

        default : /* Algo mas? No esperado. Abortamos */
            abort();
    }
}

int sizeBloque = 0, sizeCache = 0, vias = 0;
sizeBloque = tamanoBloque(datosCache);
sizeCache = tamanoCache(datosCache, sizeBloque);
vias = cantidadVias(datosCache, sizeCache, sizeBloque);
remove(FILE_OUTPUT_VALGRIND);
remove(FILE_OUTPUT_CACHEGRIND);
remove(FILE_OUTPUT_CGANNOTATE);
remove(FILE_OUTPUT_GREP);
std::cout << "Cache de datos L1:" << std::endl;
std::cout << "#Vias: " << vias << " Vias" << std::endl;
std::cout << "Tamano: " << sizeCache << " Bytes"<< std::endl;
std::cout << "Tamano de bloque: " << sizeBloque << " Bytes"<< std::endl;
return 0;
}

```

A.2. tamañoBloque.cpp

```
#include <iostream>

#define ITERACIONES_BLOQUE 1024 * 1024

void tamañoBloque(char *vector){

    register unsigned long i = 0;
    for(i=0; i< ITERACIONES_BLOQUE; i++){
        vector[i]= 'z';
    }
}

int main(int argc, char* argv[]){
    char vector[ITERACIONES_BLOQUE];
    tamañoBloque(vector);
    return 0;
}
```

A.3. tamanhoCache.cpp

```
#include <iostream>
#include <stdlib.h>

void tamanhoCache(register int tamanhoBloque, register int n, char *vector)
{
    register int i;

    for(i=0; i< n; i++)
    {
        vector[i * tamanhoBloque]= 'a';
    }

    for(i=0; i< n; i++)
    {
        vector[i * tamanhoBloque]= 'a';
    }
}

int main(int argc, char* argv[])
{
    if(argc >= 3)
    {
        register int tamanhoBloque = atoi(argv[1]);
        register int n = atoi(argv[2]);
        char *vector = new char[n * tamanhoBloque];
        tamanhoCache(tamanhoBloque, n, vector);
        delete []vector;
    }
    return 0;
}
```

A.4. cantidadVias.cpp

```
#include <iostream>
#include <stdlib.h>

int main(int argc, char* argv[]){

    register unsigned int vias = atoi(argv[1]);
    register unsigned int tamanoCache = atoi(argv[2]);
    register unsigned int tamanoBloque = atoi(argv[3]);
    register unsigned int i;
    register unsigned int j;
    register unsigned int k;
    register unsigned int tamanoFor;

    char **vector = new char*[vias];
    for (i = 0; i < vias; i++){
        vector[i] = new char[tamanoCache];
    }

    tamanoFor = tamanoCache / tamanoBloque / vias; //cantidad de bloques por via

    for (k = 0; k < 100; k++){
        for(i= 0; i< tamanoFor; i++){
            for(j= 0; j< vias; j++){
                vector[j][i * tamanoBloque]= 'a';
            }
        }

        for(j= 0; j< vias; j++){
            vector[j][0]='z';
        }

        for (i = 0; i < vias; i++){
            delete vector[i];
        }
        delete []vector;
        return 0;
    }
}
```