Universidad Nacional de San Agustin
Esc. Profesional de Ciencia de la Computación

Algoritmos Paralelos

Tema: Laboratorio 1 Memoria Caché

Profesor: Alvaro Henry Mamani Aliaga
Alumno: Diego André Ranilla Gallegos

27 de Marzo del 2017

1. Laboratorio

1. Implementar en C la versión simple de la multiplicación de matrices con tres bucles anidados y tratar de evaluar su desempeño para una matriz de tamaño relativamente grande.

```
void multiplicacion3Loops(int C[N][N],int A[N][N],int B[N][N])
{
  int i,j,k;
  for (i=0; i < N; i++){
    for (j=0; j < N; j++){
      for (k=0; k < N; k++){
        C[i][j] += A[i][k]*B[k][j];
      }
    }
  }
}
```

Listing 1: Multiplicación de matrices con 3 bucles

En este programa se implementó un medidor de tiempo con la finalidad de evaluar su rendimiento, en este caso vamos a evaluar la multiplicación de matrices con dimensiones de 800 x 800, cuyo resultado es 3.741124 segundos.

2. Implementar la versión de la multiplicación de matrices en bloques con seis bucles anidados para comprobar si se puede observar un beneficio significativo.

```
void multiplicacion6Loops(int C[N][N],int A[N][N],int B[N][N])
{
  int i1,j1,k1,i,j,k;
    for (i1=0; i1<N; i1+=s){
      for (j1=0; j1<N; j1+=s){
        for (k1=0; k1<N; k1+=s){
          for (i=i1; i<i1+s&&i<N; i++){
            for (j=j1; j<j1+s&&j<N; j++){
              for (k=k1; k<k1+s&&k<N; k++){
                C[i][j] += A[i][k]*B[k][j];
              }
            }
          }
        }
      }
    }
}
```

Listing 2: Multiplicación de matrices con 6 bucles

Para este programa también se evaluó su rendimiento empleando las mismas dimensiones que se usaron con el programa anterior, el resultado obtenido es: 3.196704 segundos.

3. Ejecute estos algoritmos paso a paso para obtener una buena comprensión de los movimientos de datos entre el caché y la memoria y tratar de evaluar su respectiva complejidad en términos de acceso a la memoria distante.

   - En el primer programa se caracteriza en almacenar la memoria caché cada uno de los vectores (filas) pertenecientes a la matriz A, debido

que el programa, este recorre cada elemento contiguo de cada uno de los vectores, por lo cual convertiría en cada vector en un cache line el cual tendría una longitud de N.

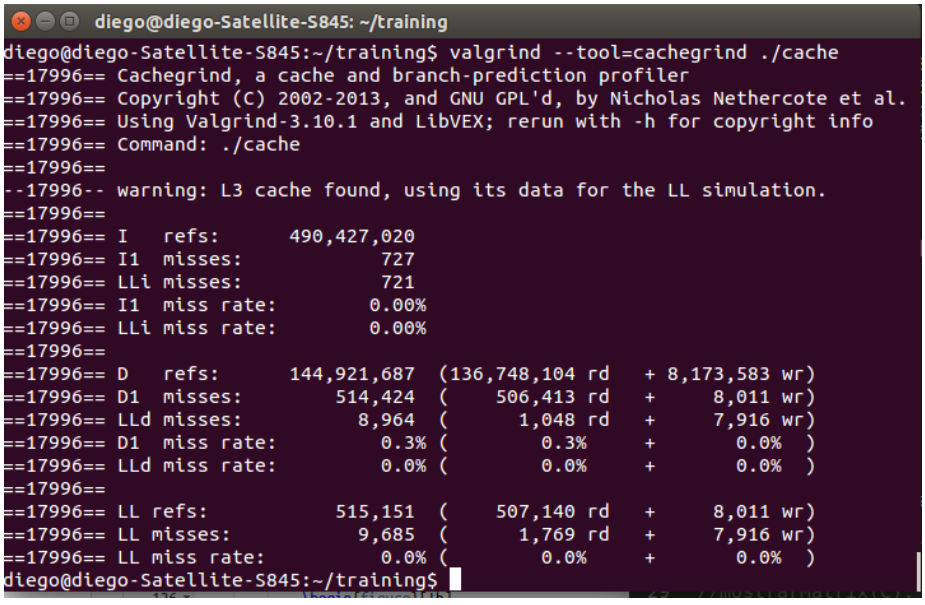
Ahora en cuanto a la matriz B, por las instrucciones que presenta el programa, tiene que recorrer de forma vertical osea por columnas, eso provoca que no tenga el beneficio de la *localidad* el cual tiene cada uno de los vectores de A, entonces eso hace que sea más propenso a que cuando se ejecute el programa, el procesador no encuentre en la cache los valores de las columnas de la matriz B y tenga que llamarlos desde la memoria.

- En el segundo programa se caracteriza por dividir la matriz de NxN en bloques más pequeños o submatrices, con la finalidad de que la tasa de cache misses sea poca por parte de la matriz B a comparación del programa anterior. Con esto quiero decir que en el primer programa, se recorre la columna de N elementos, los cuales a la primera no estarían en la cache, hasta la segunda iteración que se utilice y en el segundo programa sería lo mismo pero con la diferencia que se toma una parte de las columnas y no toda, la cual tiene la longitud de N, y en la segunda iteración y para adelante ya estaría en la memoria caché mucho más antes.

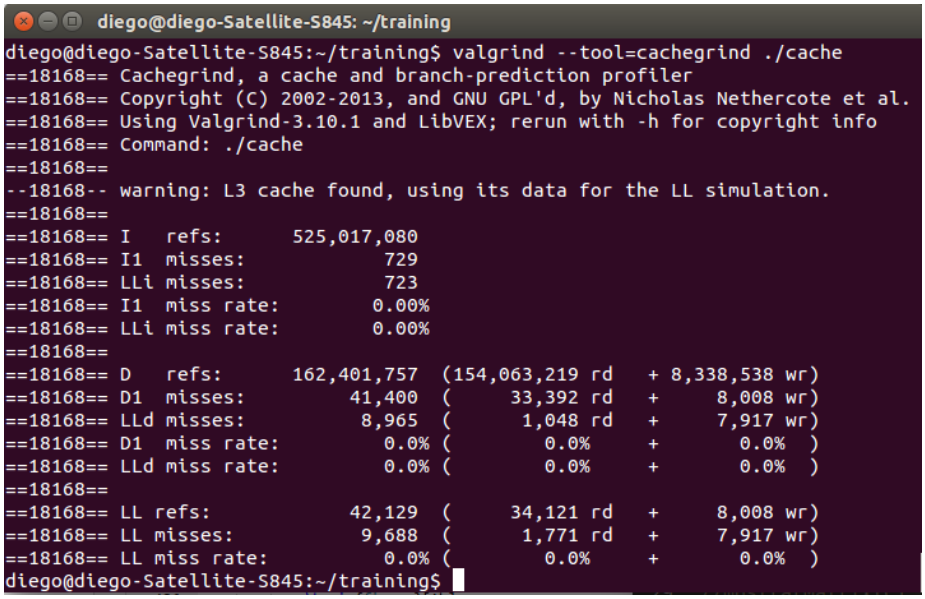
4. Ejecutar estas dos versiones de código con la herramienta *Valgrind* con la finalidad de obtener una evaluación precisa acerca de sus rendimiento en cuanto a los caché misses.

   PRIMERA PRUEBA: N=100 y s=10

   - Primer Programa:



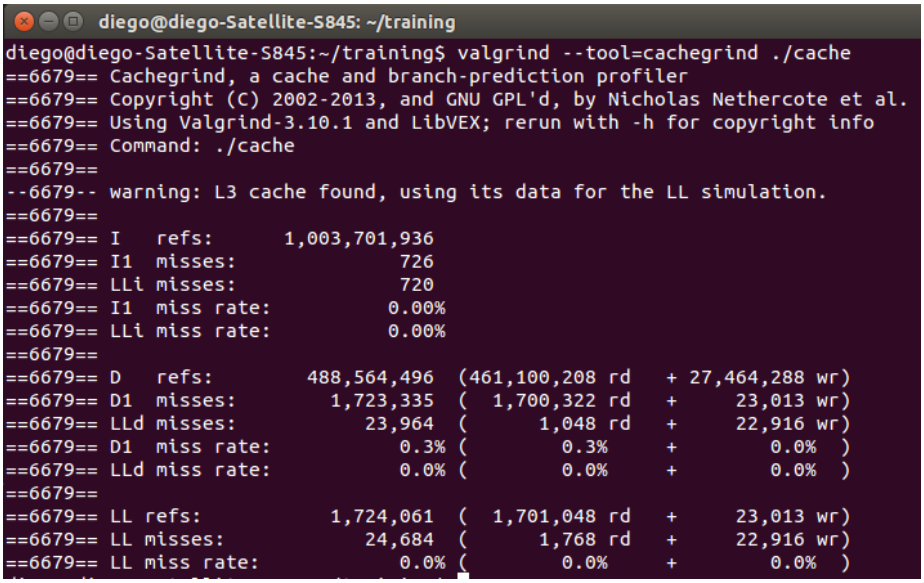
- Segundo Programa:



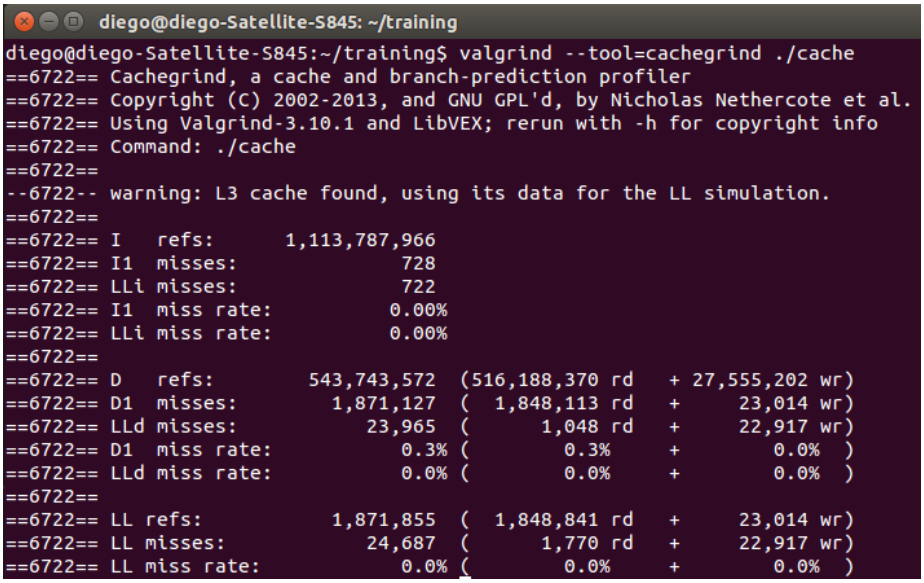
Con el empleo de la herramienta Valgrid podemos rescatar que el segundo programa tiene un mejor manejo de la memoria caché, podemos apreciar esta diferencia cuando comparamos el manejo de los datos cuando se ejecuta el programa, la tasa de caché misses en el nivel 1 ($D1$ *misses*) del programa 1 es más alto que el programa 2, debido a que los datos en el programa 2 fueron almacenados a la caché más antes a comparación del programa 1, por lo que la búsqueda en la caché fue más efectiva.

SEGUNDA PRUEBA: N=300 y s=100

- Primer Programa:

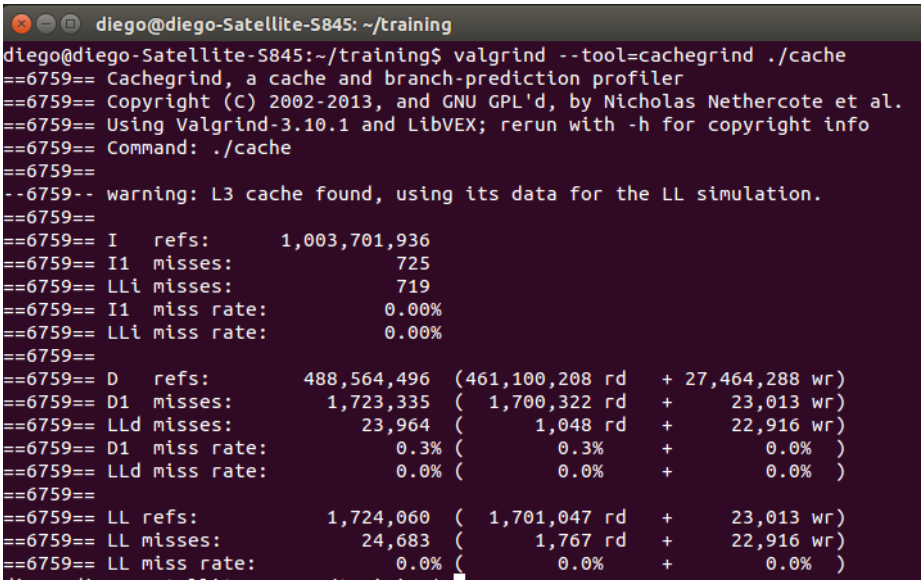


- Segundo Programa:

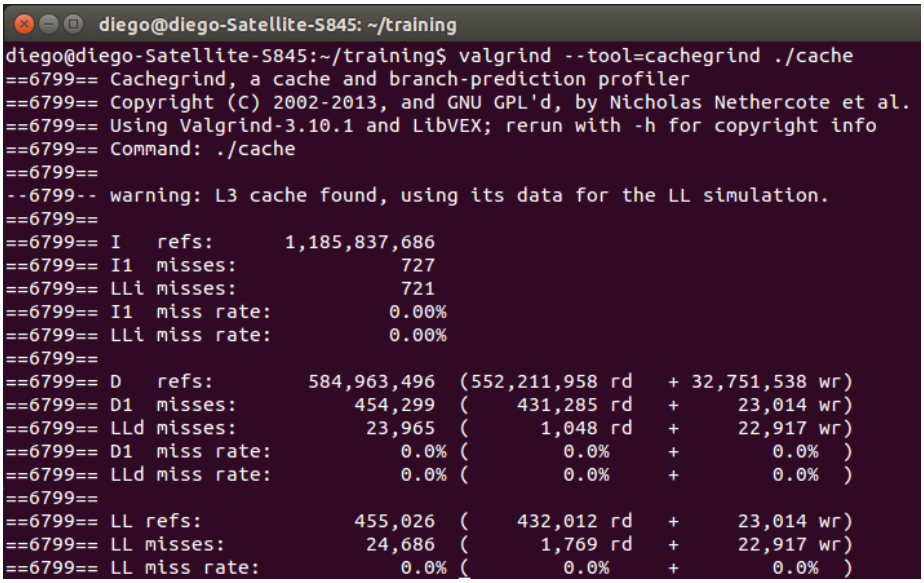


TERCERA PRUEBA: N=300 y s=6

- Primer Programa:



- Segundo Programa:



Al comparar la segunda y tercera prueba, cuyos valores s varían considerablemente, podemos mencionar algo importante acerca de la eficiencia del segundo programa, cuanto más grande es s (en el cual observamos en la segunda prueba), podría llegar hacer ineficiente en el manejo de la caché e incluso superando al primer programa, entonces el valor de s influye mucho en el rendimiento del segundo programa, por lo que tenemos que tener cuidado por lo cual es preferible elegir valores bajos para s .

2. Conclusiones

La implementación de la caché al modelo de Von Neuman ayudó mucho en la mejora del desempeño del sistema, ya que aporta a mejorar uno de los problemas importantes que surgieron a partir de este modelo el cual es el cuello de botella.

Por lo tanto cuando se realiza el diseño e implementación de programas y algoritmos, se debe tomar en cuenta, la presencia del caché ya que es un agente de gran importancia para que el programa o algoritmo ejecute de manera eficiente, como lo hemos visto en este laboratorio, haciendo la comparación de los dos programas.