Universidad ICESI

Memory Mountain y Producto matricial por bloques

Arquitectura de Hardware

- Gabriel Suarez Barón
- Alejandro Varela
- Luis Murcia
- Esteban Mendoza

4-11-2022

Tabla de contenido

Resumen 2
Objetivos 2
Marco teórico 2
Metodología 3
- Identificación del problema 4
- Identificación de factores 4
  - Casos del experimento 4
  - Valores de retorno 4
  - Factores 4
  - Algoritmos 5
- Experimentación preliminar 7
  - Comprobación 7
  - Ejecución inicial de los algoritmos 8
- Diseño del experimento 11
  - Maquinas donde se realizará el experimento 11
  - Valor de N que supera a la memoria caché 13
  - Determinar diferentes valores de N 16
  - Determinar el número de repeticiones 16
- Efectuar el experimento 17
- Análisis de resultados 17
  - Efectos principales 19
  - Grafica de interacción para efectos principales del tiempo normalizado 21
  - Memory Mountain 23
- Conclusiones 25

Bibliografía 26

Resumen

El informe que se presenta a continuación evalúa como los diferentes componentes de hardware y software afectan a la eficacia de un algoritmo. Estos efectos se pueden apreciar en el procesamiento, almacenamiento y acceso a los valores obtenidos por un algoritmo. Se utilizaron 6 algoritmos diferentes que multiplican matrices cuadradas de la forma C = AxB, estos algoritmos fueron programado en el lenguaje C++, se ejecutaron 3 veces cada uno, lo que se representa 3 repeticiones. Cada integrante ejecutó los algoritmos en sus máquinas con sistemas operativos Windows y MacOS. Finalmente, para hacer el análisis y la creación de graficas se utilizó el software que lleva por nombre Minitab, el cual nos facilita esta tarea, permitiendo crear y analizar la prueba ANOVA de dos vías, la cual será la encargada de determinar y comparar el comportamiento de cada uno de los algoritmos presentados.

Objetivos

- Analizar y emitir un juicio sobre el impacto que tiene en el desempeño en la ejecución de diferentes algoritmos equivalentes, pero con diferentes grados de aprovechamiento del principio de localidad, aplicando para ello la metodología de conducción de experimentos y apoyándose en la teoría de la arquitectura de computadores.
- proponer un esquema de multiplicación por bloques y evaluar la mejora de algoritmos deficientes.
- Medir el Memory Mountain para cada uno de los equipos de computadores usados en el experimento.
- Medir el desempeño de ejecución de tres versiones de algoritmos de multiplicación de matrices, cabe destacar que estos tienen la misma complejidad algorítmica.
- Analizar los resultados y proponer y comprobar una mejora mediante a la multiplicación matricial por bloques realizando un análisis de los datos recolectados de las dos primeras partes.

Marco teórico

Para llevar a cabo esta investigación, es necesario definir varios conceptos con el fin de conocer la organización y componentes de un sistema de cómputo usados durante el experimento. El principal concepto que se abordará es el de **lenguaje de alto nivel** el cual se caracteriza por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana, en lugar de la capacidad con que los ejecutan las máquinas, es decir, que permiten una máxima flexibilidad al programador a la hora de abstraerse o de ser literal.

Estos lenguajes se caracterizan por ser los principales actores a la hora de desarrollar las aplicaciones como las conocemos, debido a esto, existen múltiples maneras de codificarlos y por lo tanto existen algunas formas más eficientes que otras. Dicha situación se debe a que los lenguajes funcionan con la secuencia que le dé el programador, sumado a esto, también se encuentra el concepto de **localidad espacial** el cual significa que todas las instrucciones que se almacenan cerca de la instrucción ejecutada recientemente tienen altas posibilidades de ejecución. Se refiere al uso de elementos de

datos (instrucciones) que están relativamente cerca en las ubicaciones de almacenamiento y también del mismo modo a la **localidad espacial** la cual significa que una instrucción que se ha ejecutado recientemente tiene altas posibilidades de volver a ejecutarse. Por lo tanto, la instrucción se guarda en la memoria caché de manera que se pueda recuperar fácilmente y no se demore en buscar la misma instrucción.

Por otro lado, el sistema operativo de ordenadores u otros dispositivos utilizan la **memoria RAM** para almacenar de forma temporal todos los programas y sus procesos de ejecución. Lo que significa que, a mayor cantidad de memoria, mayor velocidad de procesamiento de las acciones que se realicen dentro el computador. Hablando de memoria se puede mencionar que, para mayor eficiencia interna, existe una jerarquía de memoria organizada de la siguiente manera:

Registros: Memoria más rápida y de menos capacidad de todas, está integrada en el microprocesador, permite guardar transitoriamente y acceder dentro de valores muy usados, normalmente en operaciones matemáticas.

Caché: Memoria que guarda temporalmente los datos recientemente procesados. Es un búfer especial de memoria que tienen las computadoras. Existen tres niveles de cache, estos niveles son el L1, L2, y L3, cada uno más grande en términos de almacenamiento que los otros respectivamente.

Memoria principal: Es el lugar de la computadora donde se almacenan temporalmente tanto los datos como los programas que la CPU procesa o tiene que procesar.

Disco Duro: Es un dispositivo de almacenamiento de datos que emplea un sistema de grabación magnética en la que guarda archivos digitales.

Por último, en el experimento se tuvo en cuenta que la ejecución de las **aplicaciones en segundo plano** son los procesos que ejecutan las aplicaciones mientras no las usamos, como la sincronización de datos, ubicación, actualización de información, etc. El inconveniente de esto es que, si hay muchas aplicaciones en el computador ejecutándose en segundo plano al mismo tiempo, estas tienden a consumir muchos recursos y afectar el rendimiento del experimento en el tiempo de ejecución, por lo que se evitó el uso del dispositivo y cada aplicación innecesaria dentro de las horas activas del experimento fue cerrada, para así, evitar alguna anomalía en los datos arrojados por los algoritmos utilizados.

Metodología

La metodología utilizada para llevar a cabo la conducción del experimento fue la planteada por Douglas C. Montgomery, el cual nos plantea los siguientes pasos para llevar a cabo la experimentación.

1. Identificar claramente el problema a resolver
2. Identificar factores
3. Experimentación preliminar
4. Elegir el diseño experimental
5. Efectuar el experimento
6. Analizar datos
7. Conclusiones y toma de decisiones

Identificación del problema

Usualmente se cree que los problemas en tiempo de ejecución y compilación de los programas que se usan día a día son culpa solamente de la jerarquía en la que se codifican las aplicaciones en los lenguajes alto nivel. Pero a medida que se adquiere conocimiento acerca de la estructura del computador y su funcionamiento interno se puede ver que cada máquina contiene una serie de elementos que son requeridos para el buen funcionamiento del sistema.

Elementos como el lenguaje ensamblador el cual implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarios para programar una arquitectura de procesador y el cual también constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador, los tipos de memoria cuya función principal es almacenar todos los datos que ingresan al sistema y un disco duro el cual permite guardar toda la información o el contenido digital.

Por tal razón se debe evaluar y analizar cómo las estructuras hardware y componentes software de un computador afectan el rendimiento y la utilidad de los programas de aplicación, tanto en el procesamiento, acceso y almacenamiento de los datos de los programas que los manipula. De la misma manera se emitirá un juicio sobre el impacto que tiene el desempeño de la ejecución de diferentes algoritmos equivalentes, pero con diferentes grados de aprovechamiento del principio de localidad espacial.

Identificación de factores

Casos del experimento

Para realizar el experimento se tomaron algoritmos de multiplicación de matrices diferentes pero equivalente, los cuales están programados en C++, se evalúa el rendimiento y el tiempo de ejecución de estos.

Valores de retorno

El valor obtenido para determinar el resultado del experimento es el tiempo de ejecución que tardan estos algoritmos en realizar sus operaciones, este tiempo será el normalizado. Cabe recalcar que en cada repetición el tamaño de las matrices cambiara.

Factores

Controlables

Los factores controlables son aquellos que pueden ser ajustados por el experimentador. Estos factores en nuestro experimento son los siguientes:

Cantidad de datos a ordenar, ruido agregado a la experimentación (aplicaciones innecesarias ejecutándose), sistema operativo, componentes del computador el cual realiza la prueba, RAM del computador (se puede limitar).

No controlables

Los no controlables son esas variables que no se pueden controlar durante la operación normal del proceso. En este caso son:

Cantidad de procesos ejecutándose en segundo plano para el funcionamiento de la máquina, procesador de la máquina, posición de registro y memoria donde se guardan las variables de los algoritmos.

Estudiados

Son las variables que se investigan en el experimento para observar cómo afectan o influyen en la variable de respuesta. Donde podemos encontrar:

Tiempo de ejecución del algoritmo, cantidad de datos (n), tipo de algoritmo (versión) y numero de repeticiones de los algoritmos.

Algoritmos

Los algoritmos que multiplican la matriz utilizados para realizar la experimentación son los siguientes:

Algoritmo IJK

(a) Version *ijk*

code/mem/matmult/mm.c

```
1   for (i = 0; i < n; i++)
2       for (j = 0; j < n; j++) {
3           sum = 0.0;
4           for (k = 0; k < n; k++)
5               sum += A[i][k]*B[k][j];
6           C[i][j] += sum;
7       }
```

code/mem/matmult/mm.c

Algoritmo JIK

(b) Version *jik*

code/mem/matmult/mm.c

```
1   for (j = 0; j < n; j++)
2       for (i = 0; i < n; i++) {
3           sum = 0.0;
4           for (k = 0; k < n; k++)
5               sum += A[i][k]*B[k][j];
6           C[i][j] += sum;
7       }
```

code/mem/matmult/mm.c

Algoritmo JKI

(c) Version *jki*

code/mem/matmult/mm.c

```
1   for (j = 0; j < n; j++)
2       for (k = 0; k < n; k++) {
3           r = B[k][j];
4           for (i = 0; i < n; i++)
5               C[i][j] += A[i][k]*r;
6       }
```

code/mem/matmult/mm.c

Algoritmo KJI

(d) Version *kji*

code/mem/matmult/mm.c

```
1   for (k = 0; k < n; k++)
2       for (j = 0; j < n; j++) {
3           r = B[k][j];
4           for (i = 0; i < n; i++)
5               C[i][j] += A[i][k]*r;
6       }
```

code/mem/matmult/mm.c

Algoritmo KIJ

(e) Version *kij*

code/mem/matmult/mm.c

```
1   for (k = 0; k < n; k++)
2       for (i = 0; i < n; i++) {
3           r = A[i][k];
4           for (j = 0; j < n; j++)
5               C[i][j] += r*B[k][j];
6       }
```

code/mem/matmult/mm.c

Algoritmo IKJ

(f) Version *ikj*

code/mem/matmult/mm.c

```
1   for (i = 0; i < n; i++)
2       for (k = 0; k < n; k++) {
3           r = A[i][k];
4           for (j = 0; j < n; j++)
5               C[i][j] += r*B[k][j];
6       }
```

code/mem/matmult/mm.c

Experimentación preliminar

Comprobación

Antes de hacer las pruebas preliminares es necesario comprobar que los 6 algoritmos realicen correctamente la multiplicación, para este propósito se hicieron pruebas donde el tamaño máximo de estas matrices fuera de 5*5 y tuvieran el mismo valor en cada índice, para estas pruebas se decidió que la primera matriz a multiplicar tendría valores de 4 y la segunda de valores 5. Adicionalmente se creó un método que imprime la matriz. Para la comprobación del funcionamiento se utilizó con las matrices A y B.

```
Matrix A
4.000 4.000 4.000 4.000 4.000 ;
4.000 4.000 4.000 4.000 4.000 ;
4.000 4.000 4.000 4.000 4.000 ;
4.000 4.000 4.000 4.000 4.000 ;
4.000 4.000 4.000 4.000 4.000 ;

Matrix B
5.000 5.000 5.000 5.000 5.000 ;
5.000 5.000 5.000 5.000 5.000 ;
5.000 5.000 5.000 5.000 5.000 ;
5.000 5.000 5.000 5.000 5.000 ;
5.000 5.000 5.000 5.000 5.000 ;
```

Ya sabiendo que el método imprime correctamente las matrices se procedió a utilizar los 6 algoritmos e imprimir la matriz C después de cada uno, Obteniendo así los siguientes resultados:

```
Matrix C con Algoritmo 1
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
```

```
Matrix C con Algoritmo 2
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
100.000 100.000 100.000 100.000 100.000
```

```
Matrix C con Algoritmo 3
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
```

```
Matrix C con Algoritmo 4
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;
```

```
Matrix C con Algoritmo 5                Matrix C con Algoritmo 6
100.000 100.000 100.000 100.000 100.000 ;100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;100.000 100.000 100.000 100.000 100.000 ;
100.000 100.000 100.000 100.000 100.000 ;100.000 100.000 100.000 100.000 100.000 ;
```

Con esto se sabe que todos los algoritmos obtienen el mismo valor, pero para comprobar que este valor sea el correcto se realizó el siguiente script en Matlab para comprobar que los resultados sean correctos

```
A(1:5,1:5) = 4
B(1:5,1:5) = 5
C = A*B
```

Que nos da el siguiente resultado

```
A =

     4     4     4     4     4
     4     4     4     4     4
     4     4     4     4     4
     4     4     4     4     4
     4     4     4     4     4


B =

     5     5     5     5     5
     5     5     5     5     5
     5     5     5     5     5
     5     5     5     5     5
     5     5     5     5     5


C =

   100   100   100   100   100
   100   100   100   100   100
   100   100   100   100   100
   100   100   100   100   100
   100   100   100   100   100
```

Con esto, ya se puede afirmar que los 6 algoritmos realizan correctamente la multiplicación de las matrices.

Ejecución inicial de los algoritmos

Ejecutaremos estos algoritmos cada uno con los siguientes valores para N, para así mismo determinar si los resultados arrojados por la experimentación son los adecuados para analizar los datos y cumplir los objetivos del experimento.

Los valores de n serán los siguientes: 100, 134, 179, 239, 319, 426, 569, 759, 1013, 1351, 1802, 2403,3205 y 4274

A continuación, se mostrarán los resultados obtenidos al ejecutar los algoritmos con estos valores de N, para así determinar los ajustes necesarios para llevar a cabo la experimentación final y abarcar la teoría para así comprobarla.

Algoritmo 1			
Versión IJK			
n	**times(s)**	**Gflop/s**	**Normalized(ns)**
100	0.0000	inf	0.0000
134	0.0050	0.9624	2.0780
179	0.0000	inf	0.0000
239	0.0090	30.338	0.6592
319	0.0310	20.943	0.9550
426	0.0700	22.088	0.9055
569	0.1670	22.062	0.9065
759	0.4270	20.480	0.9766
1013	10.300,000	20.185	0.9909
1351	25.240,000	19.539	10.236
1802	179.370,000	0.6524	30.654
2403	512.900,000	0.5411	36.963

Algoritmo 2			
Versión JIK			
n	**times(s)**	**Gflop/s**	**Normalized(ns)**
100	0.0000	inf	0.0000
134	0.0060	0.8020	2.4937
179	0.0040	28.677	0.6974
239	0.0120	22.753	0.8790
319	0.0290	22.387	0.8934
426	0.0810	19.089	10.477
569	0.1800	20.469	0.9771
759	0.4750	18.410	10.863
1013	11.480	18.110	11.044
1351	40.450	12.192	16.404
1802	188.850	0.6197	32.274
2403	537.500	0.5163	38.736

Algoritmo 3			
Versión JKI			
n	**times(s)**	**Gflop/s**	**Normalized(ns)**
100	0.0000	inf	0.0000
134	0.0000	inf	0.0000
179	0.0000	inf	0.0000
239	0.0000	inf	0.0000
319	0.0110	59.021	0.3389
426	0.0100	154.618	0.1294
569	0.0210	175.448	0.1140
759	0.0500	174.898	0.1144
1013	0.1150	180.784	0.1106
1351	0.4240	116.314	0.1719
1802	14.700	79.612	0.2512
2403	34.800	79.747	0.2508

Algoritmo 4			
Versión KJI			
n	**times(s)**	**Gflop/s**	**Normalized(ns)**
100	0.0000	inf	0.0000
134	0.0000	inf	0.0000
179	0.0000	inf	0.0000
239	0.0070	39.005	0.5127
319	0.0110	59.021	0.3389
426	0.0110	140.561	0.1423
569	0.0390	94.472	0.2117
759	0.0740	118.174	0.1692
1013	0.2000	103.951	0.1924
1351	0.7290	67.650	0.2956
1802	26.800	43.668	0.4580
2403	72.450	38.305	0.5221

Algoritmo 5			
Versión KIJ			
n	**times(s)**	**Gflop/s**	**Normalized(ns)**
100	0.0000	inf	0.0000
134	0.0000	inf	0.0000
179	0.0090	12.745	15692
239	0.0100	27.304	0.7325
319	0.0440	14.755	13.554
426	0.0850	18.190	10.995
569	0.2510	14.679	13.625
759	0.5210	16.785	11.916
1013	72.540	0.2866	69.783
1351	180.650	0.2730	73.261
1802	469.710	0.2492	80.272
2403	1.186.900	0.2338	85.537

Algoritmo 6			
Versión IKJ			
n	**times(s)**	**Gflop/s**	**Normalized(ns)**
100	0.0000	inf	0.0000
134	0.0000	inf	0.0000
179	0.0000	inf	0.0000
239	0.0060	45.506	0.4395
319	0.0300	21.641	0.9242
426	0.0500	30.924	0.6468
569	0.1210	30.450	0.6568
759	0.2940	29.745	0.6724
1013	38.320	0.5425	36.864
1351	97.910	0.5037	39.706
1802	253.810	0.4611	43.375
2403	612.900	0.4528	44.170

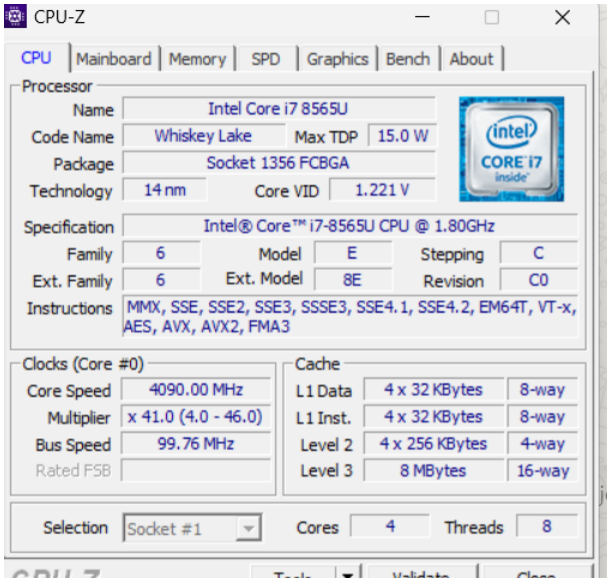
Al ejecutar estos algoritmos de forma preliminar y con algunos valores de N, podemos observar que existen algunos niveles de N donde el sistema no es capaz de determinar el tiempo normalizado de estos, por esta razón se hace necesario modificar los valores de N y determinar el número de repeticiones necesarias para obtener los datos de estos algoritmos con el menor error posible.

Diseño del experimento

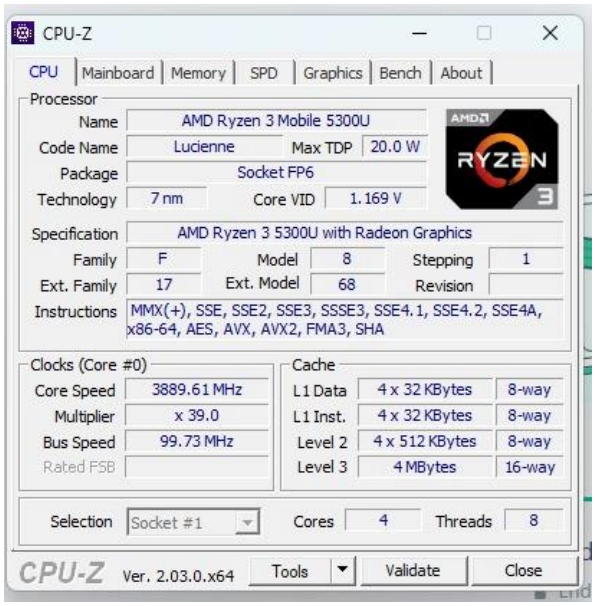
Maquinas donde se realizará el experimento

Para ejecutar el experimento y determinar los valores de N debemos de conocer los componentes de las maquinas donde se ejecutará el experimento. Esto se hace con el fin de determinar valores que sean relevantes a la hora de analizar.

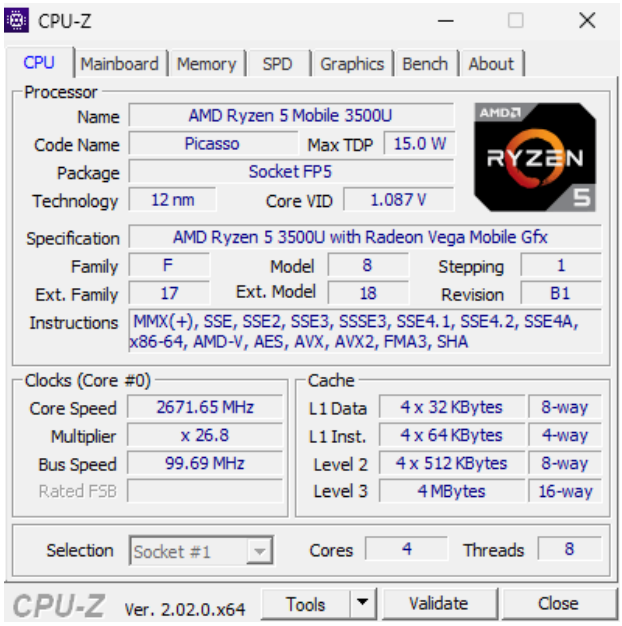
Máquina Gabriel Suárez



Máquina Alejandro Varela



Máquina Luis Murcia



Maquina Esteban Mendoza

Clock	2064 - 3200 MHz
L1 Cache	2 MB
L2 Cache	16 MB
L3 Cache	16 MB
Cores / Threads	8 / 8
Transistors	16000 Million
Technology	5 nm
Features	ARMv8 Instruction Set
iGPU	Apple M1 8-Core GPU
Architecture	ARM

Valor de N que supera a la memoria caché

Para determinar el valor mínimo de N es necesario determinar el tamaño de la Cache, esto se hace con la finalidad de que la maquina no guarde valores parciales y así determinar la fluidez y claridad de los datos obtenidos. Las maquinas tienen diferentes tamaños de memoria en la cache de nivel 3, siendo estos valores 4MB, 8MB y 16MB, por ende, se hace necesario determinar el valor mínimo de acuerdo con estas cantidad de memoria. Utilizaremos el siguiente procedimiento.

En primer lugar, determinaremos el tamaño que deberá de ocupar cada matriz, además de esto, debemos de tener en cuenta que en la matriz se guardaran números enteros, es decir, números de 32 Bits, ahora bien, determinamos cuanto Bytes son 32 Bits

$$8\ Bits\ \rightarrow\ 1\ Byte$$

$$32\ Bits\ \rightarrow ?$$

Planteando esta regla de 3 podemos determinar cuántos Bytes representa cada valor almacenado en la matriz

$$\frac{32\ Bits\ *\ 1\ Byte}{8\ Bits}\ =\ 4\ Bytes$$

Ahora bien, como nuestra unidad de medida serán los bytes, pasaremos la máxima cantidad de memoria cache a esta unidad de medida, para llegar a la unidad esperada, tendremos que pasar de MB a B:

$$1MB\ \rightarrow 1048576\ B$$

$$8MB\ \rightarrow ?$$

Siguiendo la regla de tres tenemos que:

$$\frac{8MB\ *\ 1048576\ B}{1\ MB} = 8388608\ Bytes$$

$$\frac{4MB\ *\ 1048576\ B}{1\ MB} = 4194304\ Bytes$$

$$\frac{16MB\ *\ 1048576\ B}{1\ MB} = 16777216\ Bytes$$

Una vez determinada la cantidad de memoria que utilizará cada elemento en la matriz y la cantidad de memoria cache máxima de las maquinas, podemos determinar el tamaño de cada matriz, recordemos que la matriz será cuadrada y estarán presentes tres matrices en cada repetición. Planteamos la siguiente fórmula para determinar el tamaño de la matriz

$$3\ (n^2\ *\ 4\ Bytes) =\ 8388608\ Bytes$$

$$3\ (n^2\ *\ 4\ Bytes) =\ 4194304\ Bytes$$

$$3\ (n^2\ *\ 4\ Bytes) =\ 16777216\ Bytes$$

Efectuando la formula presentada anteriormente se determinará el valor que deberá de tomar cada matriz

$$\frac{3\ (n^2\ *\ 4\ Bytes)}{4\ Bytes} = \frac{8388608\ Bytes}{4\ Bytes}$$

$$3n^2 = 2097152$$

$$n^2 = \frac{2097152}{3}$$

$$n\ =\ \pm\sqrt[2]{\frac{2097152}{3}}$$

$$n\ =\ \pm 836{,}0925$$

$$\frac{3\ (n^2\ *\ 4\ Bytes)}{4\ Bytes} = \frac{4194304\ Bytes}{4\ Bytes}$$

$$3n^2 = 1048576$$

$$n^2 = \frac{1048576}{3}$$

$$n\ =\ \pm\sqrt[2]{\frac{1048576}{3}}$$

$$n\ =\ \pm 591{,}20668$$

$$\frac{3\ (n^2\ *\ 4\ Bytes)}{4\ Bytes} = \frac{16777216\ Bytes}{4\ Bytes}$$

$$3n^2 = 4194304$$

$$n^2 = \frac{4194304}{3}$$

$$n\ =\ \pm\sqrt[2]{\frac{4194304}{3}}$$

$$n\ =\ \pm 1182{,}41335$$

Dado a que estamos hablando de una cantidad, debemos de utilizar números enteros y positivos, por ende, el tamaño final para que la cantidad memoria utilizada por los algoritmos sea superior a la memoria cache, es igual a.

$$\boldsymbol{n = 837\ para\ 8\ MB\ cache\ level\ 3}$$

$$\boldsymbol{n = 592\ para\ 4\ MB\ cache\ level\ 3}$$

$$\boldsymbol{n = 1183\ para\ 16\ MB\ cache\ level\ 3}$$

Finalmente, para que nuestro experimento tenga los niveles de N adecuados, tomaremos algunos valores por debajo de este límite y otros que superen este límite.

Determinar diferentes valores de N

Determinaremos diez niveles para N, es decir diez casos diferentes en cada algoritmo, tendremos en cuenta valores que están por encima del n limite y valores que están por abajo del n limite, esto para así lograr ver el comportamiento en ambos casos.

Los nuevos valores serán los siguientes: 319, 426, 569, 759, 1013, 1351, 1802, 2403, 3205 y 4274.

Los valores eliminados anteriormente son compensados con nuevos niveles agregados a la lista de casos, cabe recalcar que estos valores de N eliminados no nos sirven para analizarlos dado que, su tiempo normalizado es igual a 0.

Determinar el número de repeticiones

Para determinar la cantidad de repeticiones lo primero es identificar el error porcentual con el cual se quiere contar en la experimentación, lo recomendable para conservar la integridad de los datos y el correcto análisis de los resultados obtenidos es trabajar con error igual o menor al 5%.

Si se desea lograr este porcentaje de error debemos de determinar el número de muestras que debemos de tomar, cabe recalcar que este número de muestras pueden llegar a ser infinitas por lo que nos deberemos de apoyar en la siguiente fórmula, la cual nos determina el tamaño de la muestra necesaria para lograr el objetivo.

$$n = \left(\frac{Z * s}{e}\right)^2$$

Donde:

$n = tamaño\ de\ la\ muestra$

$Z = Nivel\ de\ confianza, con\ un\ nivel\ de\ significancia\ del\ 0.05$

$s = Desviación\ estandar$

$e = error$

Para hacer una versión preliminar del número de repeticiones necesarias empezaremos con tres repeticiones, dado a que este el valor mínimo recomendado para la realización de experimentos. Calcularemos el error con el primer nivel del algoritmo, con el ultimo nivel y con otro valor intermedio. Se calcula el porcentaje de error por cada algoritmo (Solo se tendrá en cuenta el valor normalizado.

La fórmula para calcular el error será la siguiente

$$e = \pm \frac{Z * s}{\sqrt{n}}$$

Además de eso debemos de tener en cuenta que Z tendrá un nivel de confianza del 95% por ende el valor de este será de

$$Z_{95\%} = 1{,}644853$$

Con esta fórmula y los valores dados, podemos empezar a calcular los errores

Versión	n	Repetición 1	Repetición 2	Repetición 3	Desviación Estándar	Error
1-ijk	319	0,955	0,9242	0,9242	0,017782388	**1,7%**
1-ijk	1013	0,9351	0,9658	0,9283	0,019979072	**1,9%**
1-ijk	4274	4,299	4,2275	4,2426	0,037685585	**3,6%**
2-jik	319	0,2464	0,2464	0,2772	0,017782388	**1,7%**
2-jik	1013	0,2443	0,2453	0,2405	0,002532456	**0,2%**
2-jik	4274	1,1034	1,1015	1,1185	0,009315042	**0,9%**
3-jki	319	0,1232	0,154	0,1232	0,017782388	**1,7%**
3-jki	1013	0,1087	0,0991	0,101	0,005083634	**0,5%**
3-jki	4274	0,2699	0,2695	0,2691	0,0004	**0,0%**
4-kji	319	0,1848	0,185	0,2156	0,017724935	**1,7%**
4-kji	1013	0,176	0,178	0,1847	0,00455668	**0,4%**
4-kji	4274	0,4934	0,4938	0,5024	0,005084617	**0,5%**
5-kij	319	0,3389	0,3697	0,37	0,01786962	**1,7%**
5-kij	1013	1,7422	1,7528	1,6968	0,029747605	**2,8%**
5-kij	4274	2,4708	2,5182	2,5254	0,02966412	**2,8%**
6-ikj	319	0,6777	0,6469	0,5953	0,041635241	**4,0%**
6-ikj	1013	3,4464	3,3804	3,3785	0,038665273	**3,7%**
6-ikj	4274	4,5633	4,5608	4,5169	0,026097318	**2,5%**

Una vez calculados los errores, podemos afirmar que el número de repeticiones escogido, es decir, 3 repeticiones, fue el adecuado, ya que, el error presentado en cada uno de los algoritmos y en cada nivel seleccionado no fue superior al 5% el cual era umbral que no se quería superar.

Dado el número de repeticiones necesarias (3 repeticiones), los diferentes niveles de n y el nivel limite donde n superar la cantidad de memoria cache, podemos empezar con nuestro experimento y la toma de datos, para su respectivo análisis.

Efectuar el experimento

Toma de datos, ver el archivo Datos.xls de la entrega además del archivo de MinitabDatos.mpx

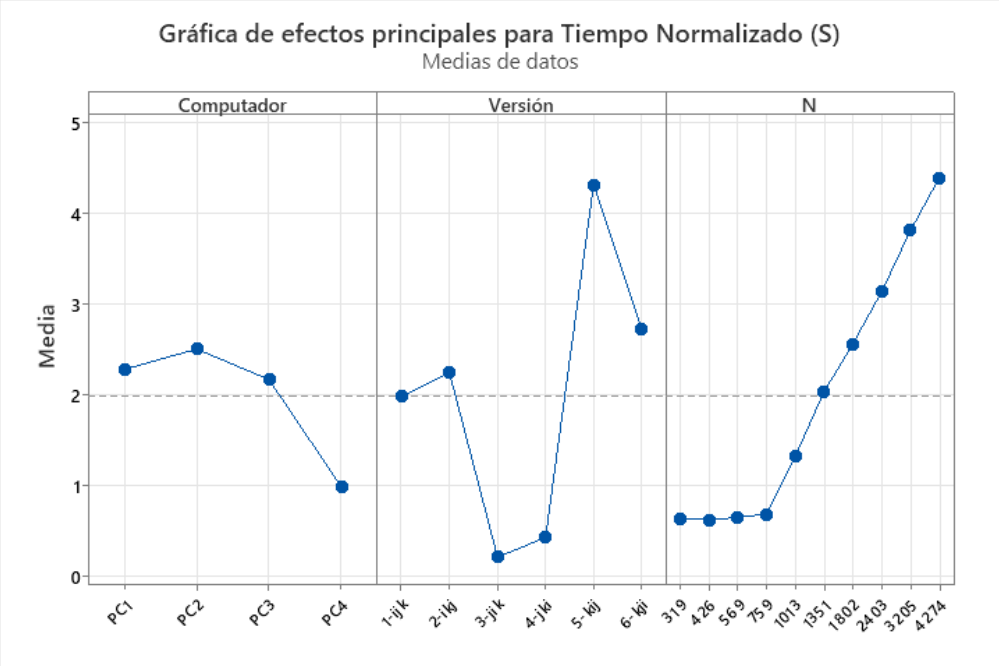
Análisis de resultados

Para comprender nuestros resultados, necesitamos saber cómo se almacenan y acceden realmente los datos. En C++, las matrices bidimensionales se almacenan en orden de fila principal en la RAM (a diferencia de Fortran, donde las matrices bidimensionales se almacenan en orden de columna principal). El orden de fila principal simplemente significa que los elementos consecutivos de una

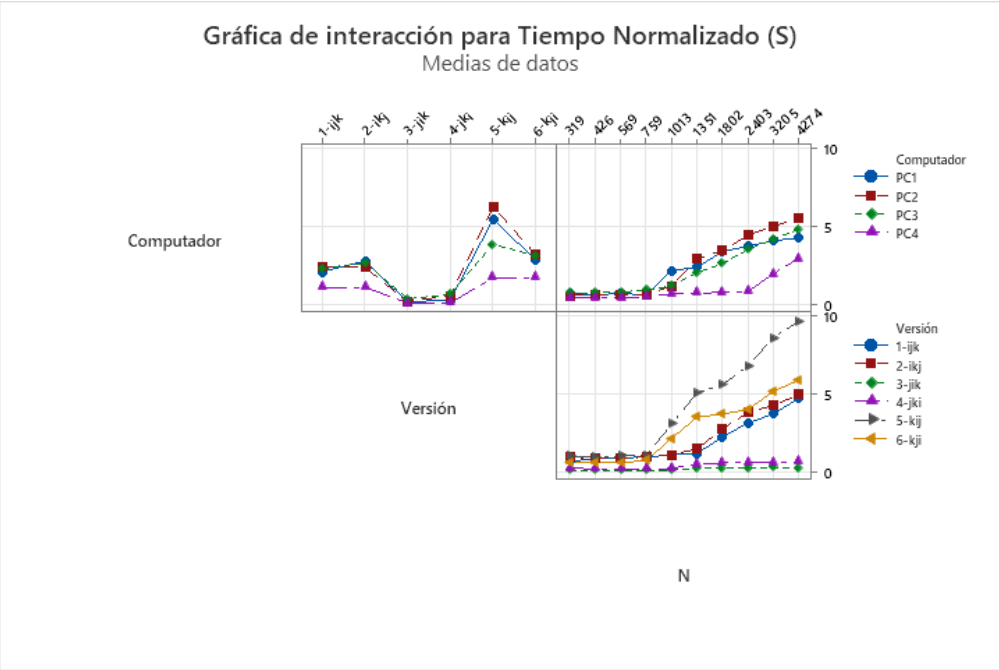
fila residen uno al lado del otro en la memoria, mientras que en el orden de la columna principal los elementos consecutivos de una columna residen uno al lado del otro en la memoria.

Los datos para procesar se cargan en el caché desde la RAM. La memoria caché lee una línea de caché de datos a la vez desde la RAM. Supongamos que el tamaño de la línea de caché es de 64 bytes, aunque necesitamos acceder a una variable de tipo entero que es solo de 4 bytes, el caché tiene que cargar los 64 bytes completos.

Una vez los datos han sido tomados, podemos proseguir a analizar estos, mediante el software de Minitab, el cual nos permitirá crear graficas de efectos principales entre las variables, asignando a la variable respuesta el tiempo normalizado (s) y los factores las variables computadores, versión del algoritmo y valor de N. Obtenemos la siguiente grafica.



Ahora bien, pasando a la gráfica de interacción para el tiempo normalizado, seleccionamos las mismas variables del mismo modo para generar la gráfica esperada



Con una visión global de las gráficas obtenidas gracias a los datos registrados en el experimento podemos darnos cuenta de los factores que afecta el tiempo de operación y realización de cada versión de un algoritmo, vemos que el factor que altera estos datos es el computador, dado que cada maquina tiene características diferentes, los resultados son diferentes, sin embargo, podemos apreciar que los patrones se siguen en cada uno de estos.

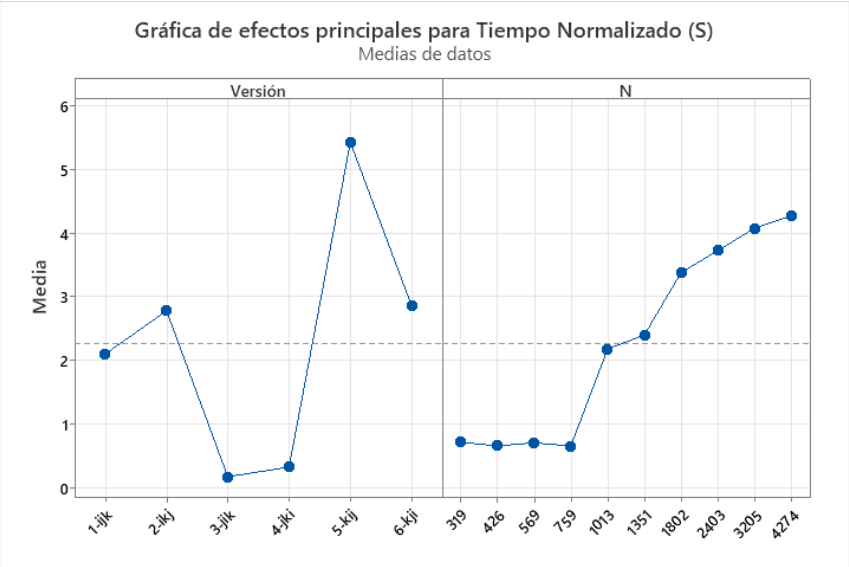
Ahora bien, una vez con una visión general podemos avanzar a ver una gráfica de cada uno de los tiempos tomados por las diferentes maquinas, es decir vamos a estudiar cada dato según el computador donde se realizó.

Efectos principales

Maquina Gabriel Suarez

DATOS GABRIEL SUAREZ

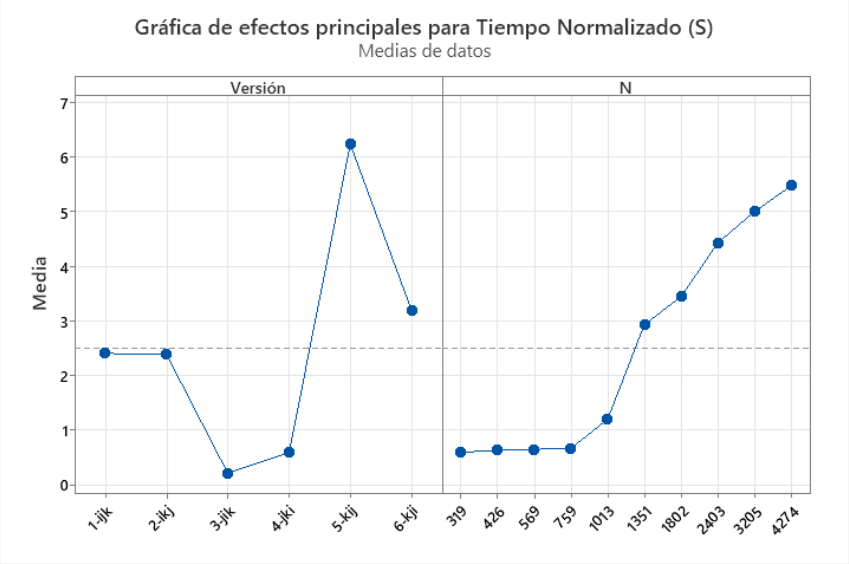
Gabriel Suarez, Gráfica de efectos principales para Tiempo Normalizado (S)



Maquina Alejandro Varela

DATOS ALEJANDRO VARELA

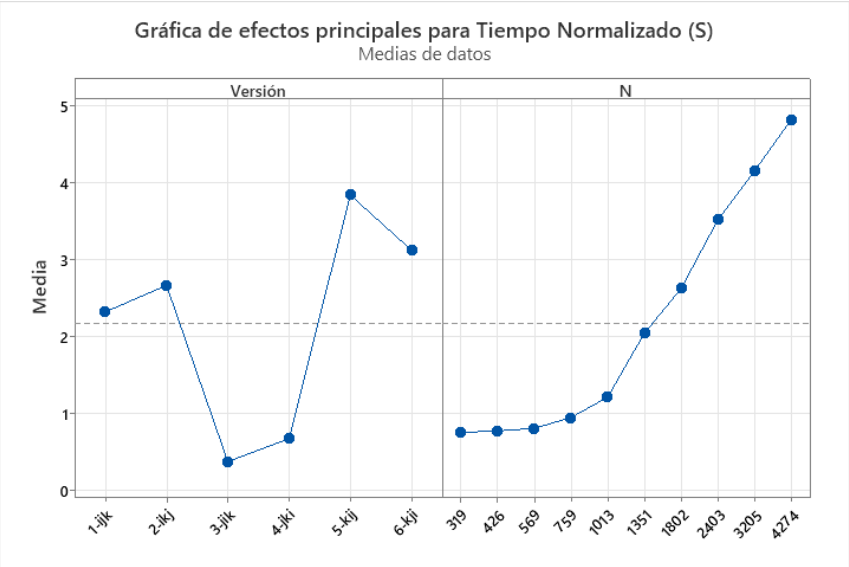
Alejandro Varela, Gráfica de efectos principales para Tiempo Normalizado (S)



Maquina Luis Murcia

DATOS LUIS MURCIA

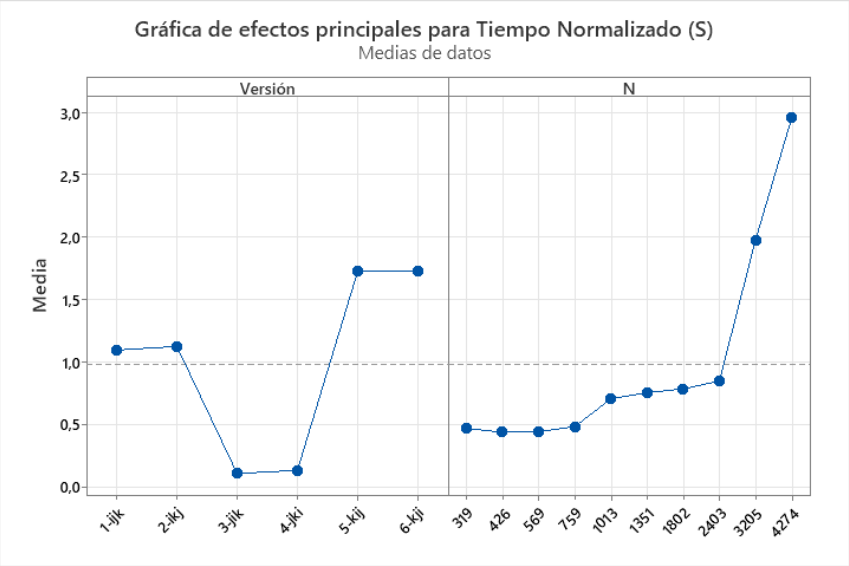
Luis Murcia, Gráfica de efectos principales para Tiempo Normalizado (S)



Maquina Esteban Mendoza.

DATOS ESTEBAN MENDOZA

Esteban Mendoza, Gráfica de efectos principales para Tiempo Normalizado (S)



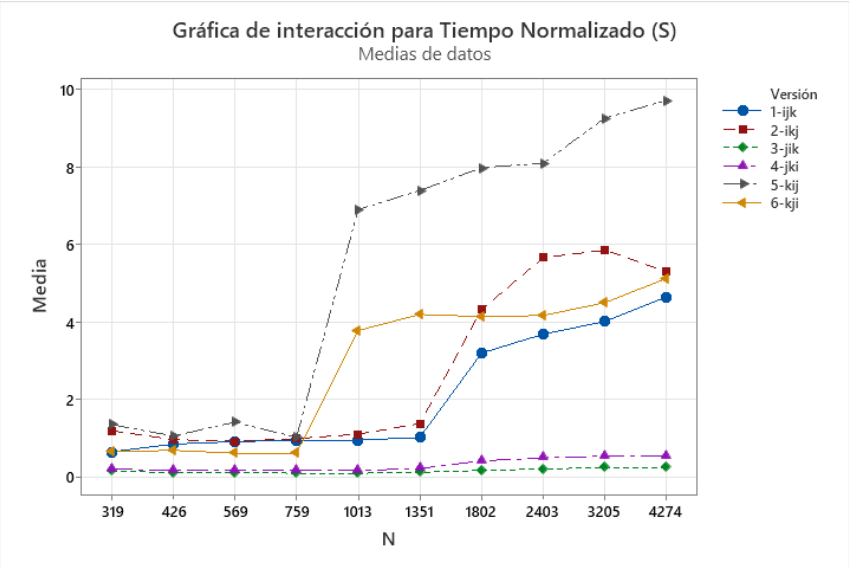
Como se ha podido observar, en todas las maquinas se ha mantenido la misma consistencia de datos, donde el factor más crítico para es el tamaño de la matriz, ya que a medida que el tiempo avanza el tiempo de distribución de los puntos crece de manera significante, a mayor N, mayor tiempo para finalizar el algoritmo. Igualmente, esta grafica obtenida nos deja ver cuál es el algoritmo más eficiente, en este caso, los más eficientes son el 3-jik y el 4-jki y lo menos eficiente en termino de tiempo son el 5-kij y el 6-kji

Grafica de interacción para efectos principales del tiempo normalizado

Gabriel Suarez

DATOS GABRIEL SUAREZ

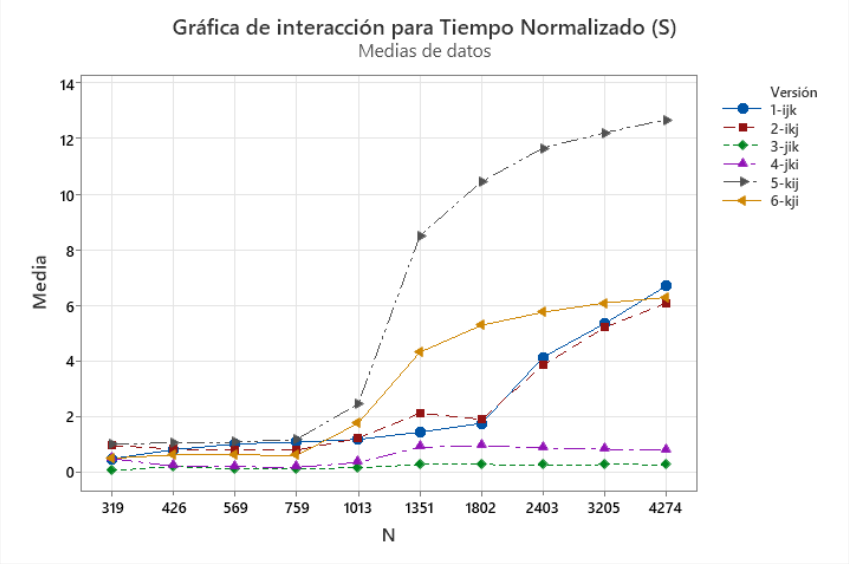
Gabriel Suarez, Gráfica de interacción para Tiempo Normalizado (S)



Alejandro Varela

DATOS ALEJANDRO VARELA

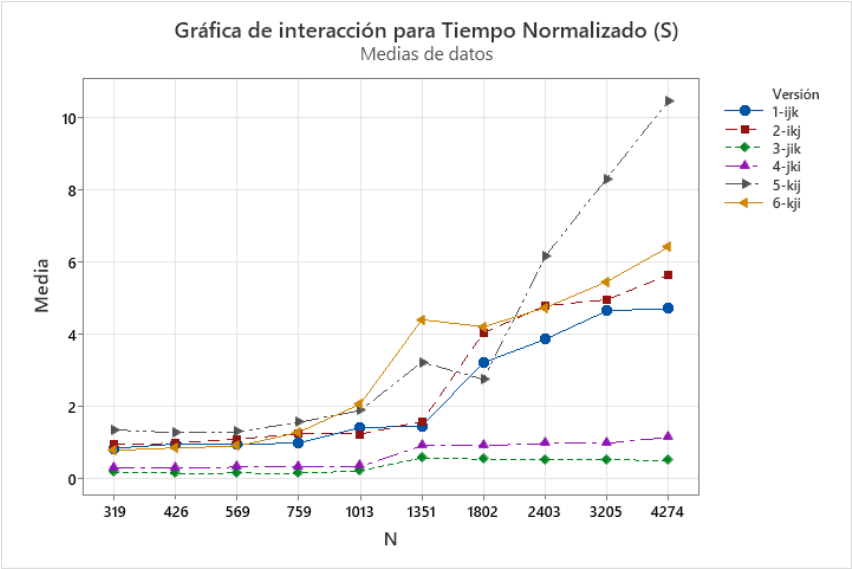
Alejandro Varela, Gráfica de interacción para Tiempo Normalizado (S)



Luis Murcia

DATOS LUIS MURCIA

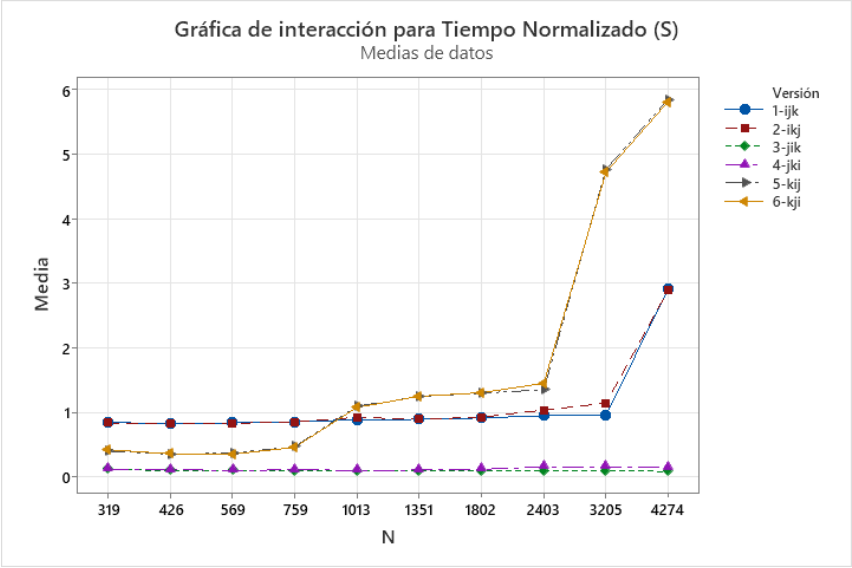
Luis Murcia, Gráfica de interacción para Tiempo Normalizado (S)



Esteban Mendoza

DATOS ESTEBAN MENDOZA

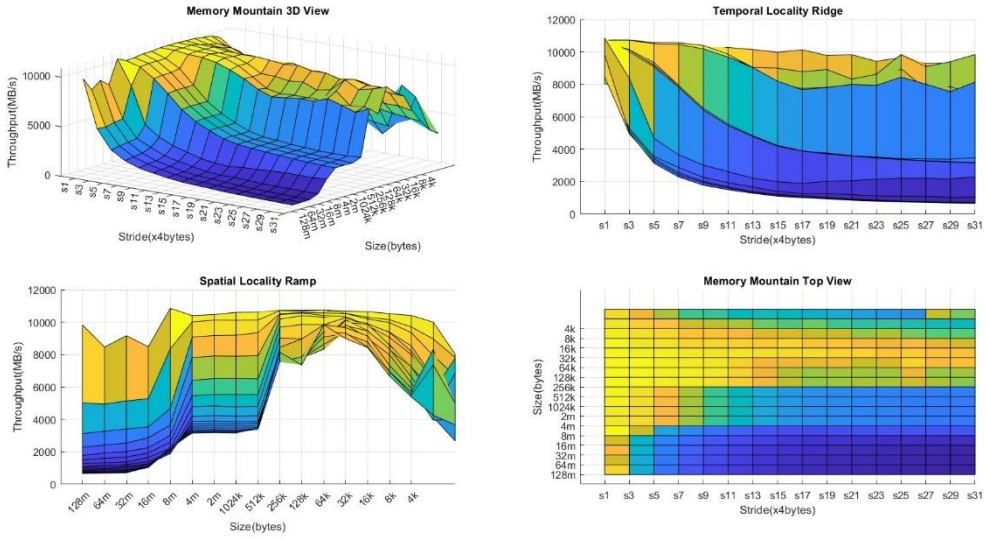
Esteban Mendoza, Gráfica de interacción para Tiempo Normalizado (S)



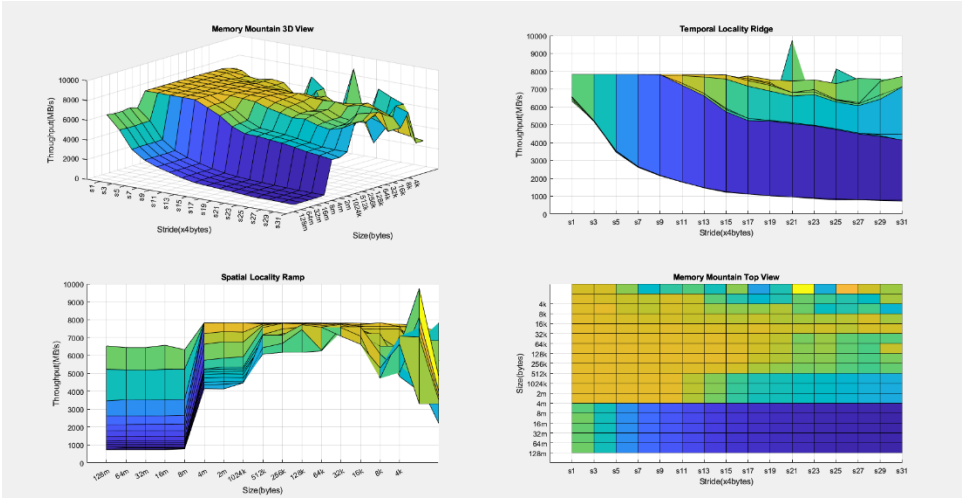
En las gráficas de interacciones de cada computador comprobamos la tendencia vista anteriormente, donde, el tiempo de ejecución de un algoritmo aumenta a medida que el valor de n aumenta, igual que en la gráfica de efectos principales, podemos identificar cual y cuáles son los algoritmos más eficientes y menos eficientes. Además de esto, también se ha identificado que a pesar de que los casos de experimentación fueron lo más limpios posibles, no deja de existir ruido en el experimento, sin embargo, a pesar de estos pequeños errores se puede identificar la eficiencia y eficacia de los algoritmos estudiados.

Memory Mountain

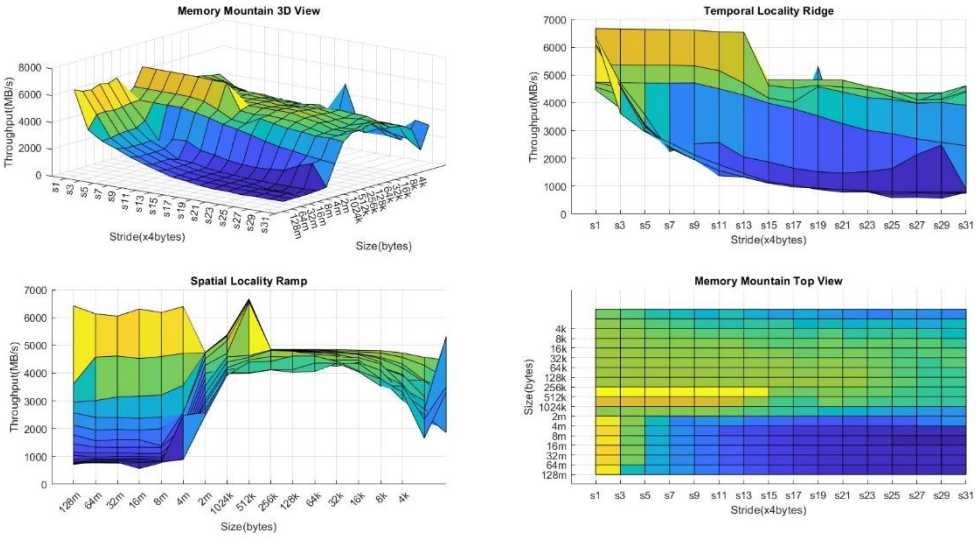
Gabriel Suarez



Alejandro Varela



Luis Murcia



Debido a que el Memory Mountain es un programa diseñado para la arquitectura x86, No se pudo ejecutar en el equipo de Esteban mendoza ya que posee una arquitectura ARM

El Memory Mountain lo podemos interpretar de la siguiente manera, según “One Step! Code”:

“La Montaña de la Memoria es una representación gráfica del ancho de banda de lectura (la velocidad a la que los datos son leídos por un programa en MB/s) en términos del tamaño total de los datos procesados y su patrón de acceso. Proporciona una gran visión de la forma en que la caché de la CPU ayuda a salvar la distancia entre el procesador de alta velocidad y los tiempos de acceso relativamente lentos de la memoria principal”.

La zona azul con menor ancho de banda de lectura corresponde a la memoria principal (RAM). La memoria caché de la CPU no puede almacenar todos los datos para tamaños superiores a 8 MB (dado el hardware de mi sistema). Luego, se requiere que la CPU obtenga constantemente la información de la memoria principal. Este proceso es relativamente lento y reduce drásticamente el ancho de banda de lectura.

Conclusiones

- Hay algoritmos que acceden fácilmente a una gran cantidad de datos debido a que todas las instrucciones que se almacenan cerca de un valor determinado ejecutado recientemente tienen una altas posibilidad de ejecución (localidad temporal), por lo tanto, es mucho más sencillo la acceso a los valores del programa y permite mejorar el rendimiento del computador de una manera notable.
- Cabe destacar que las memorias (tanto caché L1,L2 y/o L3 y la principal) juegan un papel importante, porque, así como soportan grandes cantidades de datos, de la misma manera colaboran a que el acceso a ellas para buscar información sea rápido y efectivo.
- El tiempo de ejecución de un algoritmo aumenta a medida que el valor de n aumenta, pues a mayor cantidad de datos se aumenta la probabilidad de que los valores que se busquen no se encuentren en las cachés y se empiecen a buscar los valores en la memoria principal.

- Las gráficas obtenidas gracias a los datos registrados en el experimento podemos darnos cuenta de los factores que afecta el tiempo de operación y realización de cada versión de un algoritmo, vemos que el factor que altera estos datos es el computador, dado que cada maquina tiene características diferentes los resultados son diferentes, sin embargo, podemos apreciar que los patrones se siguen en cada uno de estos.
- En todas las maquinas se ha mantenido la misma consistencia de datos, donde el factor más crítico para es el tamaño de la matriz, ya que a medida que el tiempo avanza el tiempo de distribución de los puntos crece de manera significante, a mayor N, mayor tiempo para finalizar el algoritmo. Igualmente, esta grafica obtenida nos deja ver cuál es el algoritmo más eficiente, en este caso, los más eficientes son el 3-jik y el 4-jki y lo menos eficiente en términos de tiempo son el 5-kij y el 6-kji.

Bibliografía

"Jerarquía de memorias." Wikia Periféricos de almacenamiento, https://perifericosdealmacenamiento.fandom.com/es/wiki/Jerarqu%C3%ADa_de_memorias#:~:text=Llamamos%20jerarqu%C3%ADa%20de%20memoria%20a,m%C3%A1s%20r%C3%A1pidas%20a%20mas%20lentas. Accesed 8 nov. 2022.

Aprende a detener apps en segundo plano. https://www.claro.com.co/institucional/apps-en-segundo-plano/.Accessed 8 nov. 2022.

"Caché (informática)." Wikipedia, la enciclopedia libre, 25 july 2022. Wikipedia,

https://es.wikipedia.org/wiki/Cach%C3%A9_(inform%C3%A1tica)

"Memoria de acceso aleatorio." Wikipedia, la enciclopedia libre, 21 oct. 2022.

https://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio

"Lenguaje de alto nivel." Wikipedia, la enciclopedia libre, 4 nov. 2022. Wikipedia, https://es.wikipedia.org/wiki/Lenguaje_de_alto_nivel

Diferencia Entre Localidad Espacial y Localidad Temporal – Acervo Lima. https://es.acervolima.com/diferencia-entre-localidad-espacial-y-localidad-temporal/#:~:text=En%20Localidad%20espacial%2C%20es%20probable,vuelva%20a%20ejecutar%20muy%20pronto.Accessed 8 nov. 2022.