

## 1. Optimizaciones.

### 1.1. Inversión de ciclos.

El método consiste en invertir los ciclos exteriores del método básico. Al realizar la multiplicación  $A \times B = C$  con el método básico, las matrices A y C se recorren por filas y la B por columnas. Al intercambiar el orden de los bucles, se cambia el sentido de recorrido de cada matriz, es decir, A y C por columnas y B por filas. Esto permite un mejor aprovechamiento de la localidad espacial que provee la disposición de los elementos contiguos en la memoria bajo el criterio Column Major Order.

### 1.2. Básico con trasposición.

Como se explico en el método anterior, A y C se recorren originalmente por filas. Lo que hace este método es transponer A y luego efectuar la multiplicación recorriendo A y B por columnas simultáneamente. Como se utiliza un acumulador para cada elemento de C, se realizan menor cantidad de accesos a éstos últimos, con respecto a los elementos de las otras matrices. Al igual que en el caso anterior, el beneficio se obtiene debido al aprovechamiento de la localidad espacial.

Sin embargo, el desempeño de este método se ve impactado por la penalidad de calcular la matriz traspuesta, porque deben accederse simultáneamente los elementos simétricos respecto a la diagonal (que pertenecen a distintas filas y columnas de la matriz transponer).

### 1.3. Multiplicación por bloques

Es posible dividir la matriz en bloques de tamaño arbitrario y, utilizando el método básico, realizar multiplicaciones sucesivas de matrices mas pequeñas, mejorando la localidad temporal, ya que se mantienen en la caché los datos que se usarán a corto plazo.

### 1.4. Multiplicación por columnas.

Este método realiza la multiplicación de matrices en forma incremental, efectuando todas las operaciones posibles con los elementos disponibles de una de las matrices y sumando los resultados parciales en la matriz resultado.

Utilizando las propiedades de las matrices se consiguen las siguientes equivalencias que proporcionan fundamento matemático al método. Por simplicidad, se utilizan matrices de  $2 \times 2$ .

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

El producto del miembro izquierdo se puede escribir como

$$\left( \begin{bmatrix} a_{11} & 0 \\ a_{21} & 0 \end{bmatrix} + \begin{bmatrix} 0 & a_{12} \\ 0 & a_{22} \end{bmatrix} \right) \left( \begin{bmatrix} b_{11} & 0 \\ b_{21} & 0 \end{bmatrix} + \begin{bmatrix} 0 & b_{12} \\ 0 & b_{22} \end{bmatrix} \right)$$

Y expandiendo los paréntesis resulta

$$\begin{bmatrix} a_{11} & 0 \\ a_{21} & 0 \end{bmatrix} \begin{bmatrix} b_{11} & 0 \\ b_{21} & 0 \end{bmatrix} + \begin{bmatrix} 0 & a_{12} \\ 0 & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & 0 \\ b_{21} & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & 0 \\ a_{21} & 0 \end{bmatrix} \begin{bmatrix} 0 & b_{12} \\ 0 & b_{22} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} \\ 0 & a_{22} \end{bmatrix} \begin{bmatrix} 0 & b_{12} \\ 0 & b_{22} \end{bmatrix}$$

Así, la matriz producto se puede descomponer en la siguiente suma

$$\begin{bmatrix} a_{11}b_{11} & 0 \\ a_{21}b_{11} & 0 \end{bmatrix} + \begin{bmatrix} a_{12}b_{21} & 0 \\ a_{22}b_{21} & 0 \end{bmatrix} + \begin{bmatrix} 0 & a_{11}b_{12} \\ 0 & a_{21}b_{12} \end{bmatrix} + \begin{bmatrix} 0 & a_{12}b_{22} \\ 0 & a_{22}b_{22} \end{bmatrix}$$

Como se puede apreciar, las tres matrices se acceden por columnas. Esto resulta en una mejor utilización de la memoria cache.

Para que este método funcione correctamente, se requiere que la matriz donde se almacena el resultado haya sido inicializada en 0. Sin embargo, como esto no es tenido en cuenta en el calculo del tiempo (ya que se realiza para todos los metodos), no presenta una desventaja.

Otro factor importante es el grado de asociatividad de la cache, ya que para que no se produzca trashing tiene que tener como minimo 4 vías para evitar que se reemplacen entre si las columnas de las matrices A, B y C.

Por último, si el tamaño de la matriz a multiplicar es muy grande, las columnas de cada matriz no se pueden mapear completamente en la caché, y si el grado de asociatividad no es el suficiente como para que no se produzcan conflictos no solo entre las matrices, sino también entre las columnas que cada una posee, se puede generar trashing.

### 1.5. Multiplicación por columnas, usando operación en bloques

Este método es la unión de los 2 anteriores, combinando las ventajas de cada uno. Básicamente, con la operacion por bloques se logra reducir el problema que se produce en el método de multiplicación por columnas, al operar con matrices de gran tamaño. Si se escoje el valor adecuado para el tamaño de bloque, se puede lograr operar con matrices pequeñas cuyas columnas entran completamente en cada linea de la caché. De esta manera, se hace un gran aprovechamiento de la localidad espacial y temporal.

## 2. Hardware utilizado

El programa de multiplicación de matrices, con cada optimización implementada, se ejecutó en dos configuraciones de hardware que se describen a continuación.

### 2.1. Intel® Pentium® 4 Processor

Especificaciones			
sSpec Number	SL6PE	Package Type	478 pin
CPU Speed	2.66 GHz	Manufacturing Technology	0.13 micron
PCG		Core Stepping	D1
Bus Speed	533 MHz	CPUID String	0F29h
Bus/Core Ratio	20	Thermal Design Power	66.1W
L2 Cache Size	512 KB	Thermal Specification	74°C
L2 Cache Speed	2.66 GHz	VID Voltage Range	1.53V

Cache Info:

1st-level data cache: 8 KB, 4-way set associative, sectored cache, 64-byte line size  
Trace cache: 12 K-uops, 8-way set associative  
2nd-level cache: 512 KB, 8-way set associative, sectored cache, 64-byte line size  
Data TLB: 4 KB or 4 MB pages, fully associative, 64 entries  
Instruction TLB: 4 KB, 2 MB or 4 MB pages, fully associative, 128 entries

### 2.2. Intel® Core™2 Duo Desktop Processor E6400

Especificaciones			
sSpec Number	SL9T9	Package Type	LGA775
CPU Speed	2.13 GHz	Manufacturing Technology	65 nm
PCG	06	Core Stepping	L2
Bus Speed	1066 MHz	CPUID String	06F2h
Bus/Core Ratio	8	Thermal Design Power	65W
L2 Cache Size	2 MB	Thermal Specification	61.4°C
L2 Cache Speed	2.13 GHz	VID Voltage Range	0.85V 1.5V

Cache Info:

1st-level data cache: 32 KB, 8-way set associative, 64-byte line size  
1st-level instruction cache: 32 KB, 8-way set associative, 64-byte line size  
2nd-level cache: 2 MB, 8-way set associative, 64-byte line size  
Data TLB: 4 KB Pages, 4-way set associative, 256 entries  
Data TLB: 4 MB Pages, 4-way set associative, 32 entries  
L0 Data TLB: 4 MB pages, 4-way set associative, 16 entries  
L0 Data TLB: 4 MB pages, 4-way set associative, 16 entries  
Instruction TLB: 4 KB Pages, 4-way set associative, 128 entries  
Instruction TLB: 4 MB Pages, 4-way set associative, 4 entries  
Instruction TLB: 2 MB Pages, 4-way set associative, 8 entries  
64-byte Prefetching

### 3. Corridas

#### 3.1. Intel® Pentium® 4 Processor

Compilador: gcc

Opciones: -O3 -mcpu=pentium3

Plataforma: i686-pc-linux-gnu

Descripcion: Multiplicacion basica de tres bucles.

Tam: 31 mflop/s: 798.86  
Tam: 32 mflop/s: 733.009  
Tam: 96 mflop/s: 724.831  
Tam: 97 mflop/s: 772.978  
Tam: 127 mflop/s: 728.477  
Tam: 128 mflop/s: 639.264  
Tam: 129 mflop/s: 707.624  
Tam: 191 mflop/s: 413.879  
Tam: 192 mflop/s: 416.588  
Tam: 229 mflop/s: 373.999  
Tam: 255 mflop/s: 328.99  
Tam: 256 mflop/s: 46.2338  
Tam: 257 mflop/s: 328.774  
Tam: 319 mflop/s: 307.961  
Tam: 320 mflop/s: 300.856  
Tam: 321 mflop/s: 305.358  
Tam: 417 mflop/s: 266.695  
Tam: 479 mflop/s: 243.731  
Tam: 480 mflop/s: 246.481  
Tam: 511 mflop/s: 234.434  
Tam: 512 mflop/s: 17.426  
Tam: 639 mflop/s: 224.291  
Tam: 640 mflop/s: 21.3779  
Tam: 767 mflop/s: 216.78  
Tam: 768 mflop/s: 17.4243  
Tam: 769 mflop/s: 207.639

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion por 3 bucles invertido.

Tam: 31 mflop/s: 661.754  
Tam: 32 mflop/s: 654.862  
Tam: 96 mflop/s: 665.334  
Tam: 97 mflop/s: 689.371  
Tam: 127 mflop/s: 657.269  
Tam: 128 mflop/s: 669.602  
Tam: 129 mflop/s: 641.112  
Tam: 191 mflop/s: 391.991  
Tam: 192 mflop/s: 403.942  
Tam: 229 mflop/s: 302.083  
Tam: 255 mflop/s: 226.864  
Tam: 256 mflop/s: 37.8476  
Tam: 257 mflop/s: 220.771  
Tam: 319 mflop/s: 193.849  
Tam: 320 mflop/s: 180.229  
Tam: 321 mflop/s: 190.774  
Tam: 417 mflop/s: 172.83  
Tam: 479 mflop/s: 169.165  
Tam: 480 mflop/s: 141.903  
Tam: 511 mflop/s: 163.712  
Tam: 512 mflop/s: 18.5352  
Tam: 639 mflop/s: 162.047  
Tam: 640 mflop/s: 23.031  
Tam: 767 mflop/s: 162.817  
Tam: 768 mflop/s: 18.4353  
Tam: 769 mflop/s: 161.163

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion de matrices simple por bloques.

Tam: 31 mflop/s: 640.929  
Tam: 32 mflop/s: 627.65  
Tam: 96 mflop/s: 614.566  
Tam: 97 mflop/s: 664.167  
Tam: 127 mflop/s: 624.057  
Tam: 128 mflop/s: 616.795  
Tam: 129 mflop/s: 625.757  
Tam: 191 mflop/s: 614.478  
Tam: 192 mflop/s: 586.076  
Tam: 229 mflop/s: 595.737  
Tam: 255 mflop/s: 526.154  
Tam: 256 mflop/s: 527.102  
Tam: 257 mflop/s: 516.869  
Tam: 319 mflop/s: 588.373  
Tam: 320 mflop/s: 559.414  
Tam: 321 mflop/s: 579.484  
Tam: 417 mflop/s: 554.675  
Tam: 479 mflop/s: 528.675  
Tam: 480 mflop/s: 500.063  
Tam: 511 mflop/s: 450.603  
Tam: 512 mflop/s: 398.436  
Tam: 639 mflop/s: 473.97  
Tam: 640 mflop/s: 464.458  
Tam: 767 mflop/s: 452.861  
Tam: 768 mflop/s: 447.658  
Tam: 769 mflop/s: 457.481

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion por columnas de A.

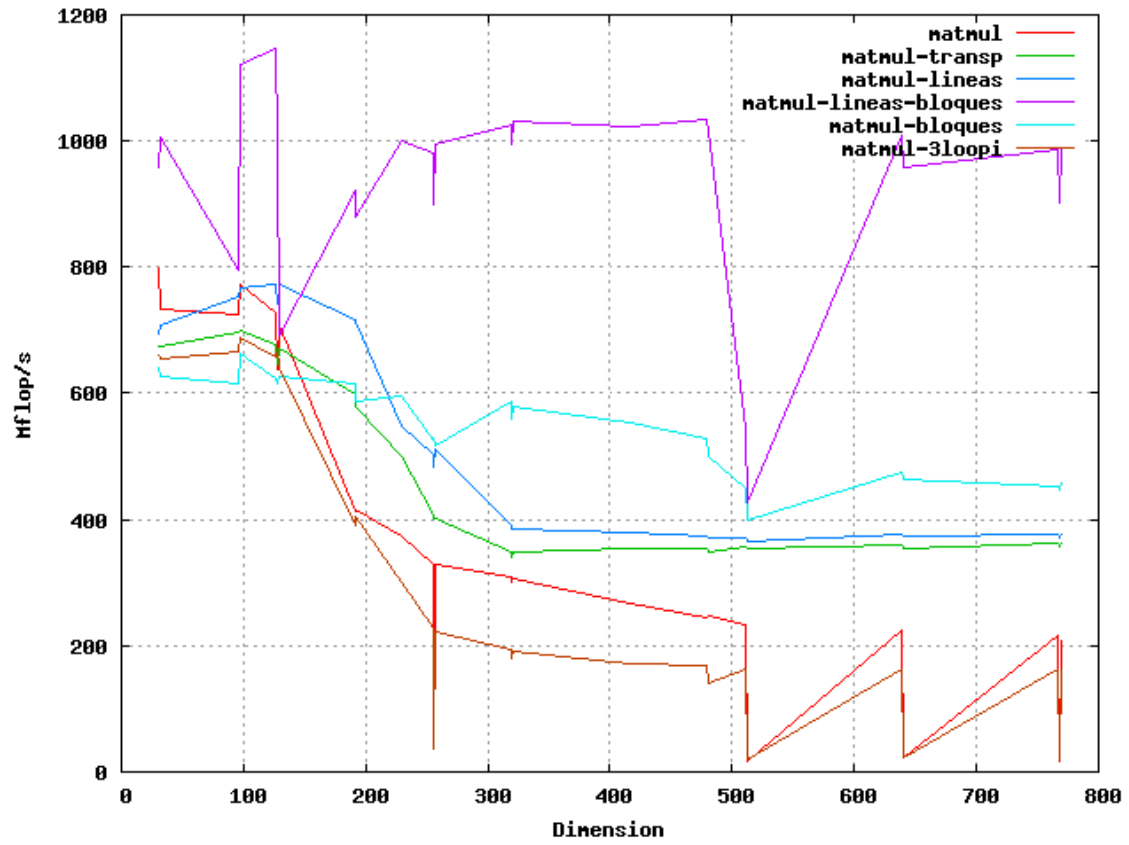
Tam: 31 mflop/s: 694.814  
Tam: 32 mflop/s: 707.835  
Tam: 96 mflop/s: 754.095  
Tam: 97 mflop/s: 768.065  
Tam: 127 mflop/s: 771.848  
Tam: 128 mflop/s: 742.912  
Tam: 129 mflop/s: 772.473  
Tam: 191 mflop/s: 716.61  
Tam: 192 mflop/s: 715.037  
Tam: 229 mflop/s: 548.209  
Tam: 255 mflop/s: 503.168  
Tam: 256 mflop/s: 484.476  
Tam: 257 mflop/s: 510.656  
Tam: 319 mflop/s: 389.803  
Tam: 320 mflop/s: 385.953  
Tam: 321 mflop/s: 383.627  
Tam: 417 mflop/s: 378.702  
Tam: 479 mflop/s: 375.121  
Tam: 480 mflop/s: 372.035  
Tam: 511 mflop/s: 370.83  
Tam: 512 mflop/s: 365.332  
Tam: 639 mflop/s: 377.173  
Tam: 640 mflop/s: 373.133  
Tam: 767 mflop/s: 376.663  
Tam: 768 mflop/s: 370.854  
Tam: 769 mflop/s: 377.592

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion basica de tres bucles luego de trasponer.

Tam: 31 mflop/s: 675.804  
Tam: 32 mflop/s: 673.103  
Tam: 96 mflop/s: 697.113  
Tam: 97 mflop/s: 699.37  
Tam: 127 mflop/s: 676.414  
Tam: 128 mflop/s: 646.485  
Tam: 129 mflop/s: 671.745  
Tam: 191 mflop/s: 598.451  
Tam: 192 mflop/s: 580.218  
Tam: 229 mflop/s: 501.178  
Tam: 255 mflop/s: 407.813  
Tam: 256 mflop/s: 400.691  
Tam: 257 mflop/s: 401.961  
Tam: 319 mflop/s: 347.075  
Tam: 320 mflop/s: 340.803  
Tam: 321 mflop/s: 347.934  
Tam: 417 mflop/s: 353.542  
Tam: 479 mflop/s: 353.902  
Tam: 480 mflop/s: 348.607  
Tam: 511 mflop/s: 355.784  
Tam: 512 mflop/s: 353.872  
Tam: 639 mflop/s: 360.57  
Tam: 640 mflop/s: 353.998  
Tam: 767 mflop/s: 363.04  
Tam: 768 mflop/s: 357.843  
Tam: 769 mflop/s: 362.779



### 3.2. Grafico Comparativo:



### 3.3. Core 2 Duo E6400

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion basica de tres bucles.

Tam: 31 mflop/s: 956.216  
Tam: 32 mflop/s: 944.568  
Tam: 96 mflop/s: 1266.29  
Tam: 97 mflop/s: 1244.16  
Tam: 127 mflop/s: 1283.46  
Tam: 128 mflop/s: 651.338  
Tam: 129 mflop/s: 1277.71  
Tam: 191 mflop/s: 1271.9  
Tam: 192 mflop/s: 645.99  
Tam: 229 mflop/s: 1275.01  
Tam: 255 mflop/s: 1282.98  
Tam: 256 mflop/s: 644.159  
Tam: 257 mflop/s: 1279.6  
Tam: 319 mflop/s: 1278.86  
Tam: 320 mflop/s: 647.424  
Tam: 321 mflop/s: 1266.74  
Tam: 417 mflop/s: 444.455  
Tam: 479 mflop/s: 370.454  
Tam: 480 mflop/s: 372.397  
Tam: 511 mflop/s: 353.249  
Tam: 512 mflop/s: 143.71  
Tam: 639 mflop/s: 339.37  
Tam: 640 mflop/s: 356.774  
Tam: 767 mflop/s: 337.243  
Tam: 768 mflop/s: 166.853  
Tam: 769 mflop/s: 331.102

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion por 3 bucles invertido.

Tam: 31 mflop/s: 1321.45  
Tam: 32 mflop/s: 1282.66  
Tam: 96 mflop/s: 1140.83  
Tam: 97 mflop/s: 1205.84  
Tam: 127 mflop/s: 1270.47  
Tam: 128 mflop/s: 745.792  
Tam: 129 mflop/s: 1227.73  
Tam: 191 mflop/s: 1310.07  
Tam: 192 mflop/s: 742.715  
Tam: 229 mflop/s: 1295.13  
Tam: 255 mflop/s: 1273.96  
Tam: 256 mflop/s: 753.4  
Tam: 257 mflop/s: 1330.58  
Tam: 319 mflop/s: 1202.46  
Tam: 320 mflop/s: 713.453  
Tam: 321 mflop/s: 1005.99  
Tam: 417 mflop/s: 309.13  
Tam: 479 mflop/s: 289.258  
Tam: 480 mflop/s: 333.348  
Tam: 511 mflop/s: 244.422  
Tam: 512 mflop/s: 149.318  
Tam: 639 mflop/s: 200.872  
Tam: 640 mflop/s: 272.988  
Tam: 767 mflop/s: 195.256  
Tam: 768 mflop/s: 180.878  
Tam: 769 mflop/s: 192.686

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion de matrices simple por bloques.

Tam: 31 mflop/s: 1212.9  
Tam: 32 mflop/s: 1208.78  
Tam: 96 mflop/s: 1231.96  
Tam: 97 mflop/s: 1176.52  
Tam: 127 mflop/s: 1276.96  
Tam: 128 mflop/s: 716.413  
Tam: 129 mflop/s: 1288.7  
Tam: 191 mflop/s: 1270.03  
Tam: 192 mflop/s: 1277.75  
Tam: 229 mflop/s: 1258.1  
Tam: 255 mflop/s: 1224.33  
Tam: 256 mflop/s: 625.331  
Tam: 257 mflop/s: 1190.77  
Tam: 319 mflop/s: 1189.44  
Tam: 320 mflop/s: 1200.76  
Tam: 321 mflop/s: 1255.07  
Tam: 417 mflop/s: 1154.13  
Tam: 479 mflop/s: 1251.7  
Tam: 480 mflop/s: 1245.69  
Tam: 511 mflop/s: 1238.65  
Tam: 512 mflop/s: 632.629  
Tam: 639 mflop/s: 1197.49  
Tam: 640 mflop/s: 654.848  
Tam: 767 mflop/s: 1241.31  
Tam: 768 mflop/s: 621.655  
Tam: 769 mflop/s: 1201.25

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion por columnas de A.

Tam: 31 mflop/s: 1627.4  
Tam: 32 mflop/s: 1580  
Tam: 96 mflop/s: 1420.21  
Tam: 97 mflop/s: 1416.65  
Tam: 127 mflop/s: 1474.29  
Tam: 128 mflop/s: 1386.12  
Tam: 129 mflop/s: 1491.18  
Tam: 191 mflop/s: 1457.47  
Tam: 192 mflop/s: 1381.82  
Tam: 229 mflop/s: 1547.82  
Tam: 255 mflop/s: 1527.84  
Tam: 256 mflop/s: 1388.78  
Tam: 257 mflop/s: 1536.32  
Tam: 319 mflop/s: 1525.52  
Tam: 320 mflop/s: 1542.17  
Tam: 321 mflop/s: 1488.85  
Tam: 417 mflop/s: 1355.6  
Tam: 479 mflop/s: 1195.36  
Tam: 480 mflop/s: 1200.6  
Tam: 511 mflop/s: 1122.46  
Tam: 512 mflop/s: 1033.02  
Tam: 639 mflop/s: 971.96  
Tam: 640 mflop/s: 873.935  
Tam: 767 mflop/s: 891.468  
Tam: 768 mflop/s: 930.549  
Tam: 769 mflop/s: 910.804

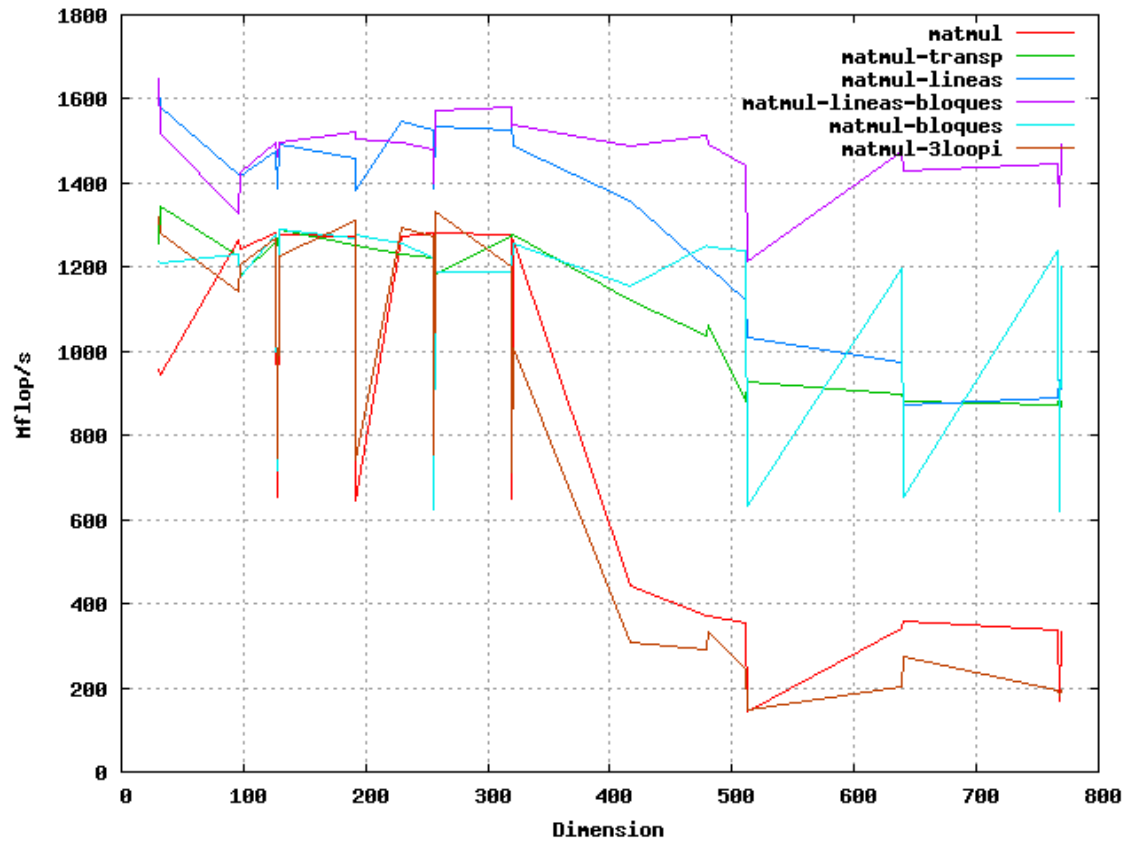
Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion basica de tres bucles luego de trasponer.

Tam: 31 mflop/s: 1258.3  
Tam: 32 mflop/s: 1346.76  
Tam: 96 mflop/s: 1226.76  
Tam: 97 mflop/s: 1180.43  
Tam: 127 mflop/s: 1258.67  
Tam: 128 mflop/s: 1250.07  
Tam: 129 mflop/s: 1291.28  
Tam: 191 mflop/s: 1252.98  
Tam: 192 mflop/s: 1253.05  
Tam: 229 mflop/s: 1232.31  
Tam: 255 mflop/s: 1221.02  
Tam: 256 mflop/s: 1199.36  
Tam: 257 mflop/s: 1185.93  
Tam: 319 mflop/s: 1273.79  
Tam: 320 mflop/s: 1278.29  
Tam: 321 mflop/s: 1279.12  
Tam: 417 mflop/s: 1122.18  
Tam: 479 mflop/s: 1036.39  
Tam: 480 mflop/s: 1063.2  
Tam: 511 mflop/s: 882.452  
Tam: 512 mflop/s: 926.115  
Tam: 639 mflop/s: 899.544  
Tam: 640 mflop/s: 882.849  
Tam: 767 mflop/s: 873.074  
Tam: 768 mflop/s: 893.43  
Tam: 769 mflop/s: 869.49

Compilador: gcc  
Opciones: -O3 -mcpu=pentium3  
Plataforma: i686-pc-linux-gnu  
Descripcion: Multiplicacion de matrices por linea por bloques.

Tam: 31 mflop/s: 1646.9  
Tam: 32 mflop/s: 1517.89  
Tam: 96 mflop/s: 1326.21  
Tam: 97 mflop/s: 1425.29  
Tam: 127 mflop/s: 1495.96  
Tam: 128 mflop/s: 1463.36  
Tam: 129 mflop/s: 1496.36  
Tam: 191 mflop/s: 1522.35  
Tam: 192 mflop/s: 1506.85  
Tam: 229 mflop/s: 1496.02  
Tam: 255 mflop/s: 1477.82  
Tam: 256 mflop/s: 1398.82  
Tam: 257 mflop/s: 1572.1  
Tam: 319 mflop/s: 1579.07  
Tam: 320 mflop/s: 1532.77  
Tam: 321 mflop/s: 1538.13  
Tam: 417 mflop/s: 1489.86  
Tam: 479 mflop/s: 1512.6  
Tam: 480 mflop/s: 1490.32  
Tam: 511 mflop/s: 1443.57  
Tam: 512 mflop/s: 1215.07  
Tam: 639 mflop/s: 1474.68  
Tam: 640 mflop/s: 1428.28  
Tam: 767 mflop/s: 1445.49  
Tam: 768 mflop/s: 1345.2  
Tam: 769 mflop/s: 1491.86

### 3.4. Grafico Comparativo:





## 4. Analisis de resultados y conclusiones:

### 4.1. Inversión de ciclos.

Como se observa en los gráficos, el multiplicar básico es muy inestable respecto del tamaño de matriz utilizado y el más ineficiente. Asimismo, lo único que produce la mejora propuesta es un **pequeño** aumento de la estabilidad y del rendimiento. La curva mantiene la misma tendencia que la del método básico.

### 4.2. Básico con trasposición.

En este caso podemos notar que el método es bastante estable (en contraposición con los anteriores).

Cuando la matriz es pequeña, el costo de transposición es pequeño, y la ventaja de recorrer por columnas no se ve tan perjudicada, obteniendo un buen desempeño.

Cuando la matrices a multiplicar son muy grandes, este método presenta 2 problemas: Por un lado el costo de transposición aumenta, porque los elementos simétricos están más separados en memoria y se pierde localidad espacial, y por el otro lado, cada columna columna sobrepasa el tamaño de la caché y se genera trashing. Este fenómeno se puede apreciar en el gráfico (con la brusca caída de mflops alrededor de un tamaño identificable).

### 4.3. Multiplicación por columnas.

Se puede ver que este método se comporta de manera similar al anterior (porque se basan en el mismo principio) sin el tiempo consumido por el proceso de transposición, por lo que la curva mflops(tamaño) se encuentra desplazada hacia arriba.

### 4.4. Multiplicación por bloques.

Este método evidencia la ventaja de trabajar con matrices de un tamaño deseado (pudiendo elegir el mas conveniente para cada situación).

El desempeño es muy variable y depende de la compatibilidad entre las dimensiones de las matrices y el tamaño del bloque (sin dejar de lado el tamaño de la cache y el grado de asociatividad de la misma). Esto significa que se puede lograr un gran rendimiento al procesar matrices muy grandes, eligiendo correctamente el bloque, cosa que no se puede realizar con los otros métodos.

Todo esto se puede ver claramente en el gráfico.

### 4.5. Multiplicación por columnas y bloques.

Como supusimos, se logran aprovechar las ventajas de los dos métodos anteriores.

Se pudo elegir el tamaño de bloque de manera tal de operar la mayor parte del tiempo con matrices que povocan un rendimiento óptimo para el método de multiplicación por columnas. De esta manera se logra mantener este nivel sin importar el tamaño de la matriz original.

## 5. Código fuente