1. Optimizaciones.

1.1. Inversin de ciclos.

El mtodo consiste en invertir los ciclos exteriores del mtodo bsico. Al realizar la multiplicacin A x B = C con el mtodo bsico, 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 dispocicin de los elementos contiguos en la memoria bajo el criterio Column Major Order.

1.2. Bsico con trasposicin.

Como se explico en el mtodo anterior, A y C se recorren originalmente por filas. Lo que hace este mtodo es transponer A y luego efectuar la multiplicacin recorriendo A y B por columnas simultneamente. Como se utiliza un acumulador para cada elemento de C, se realizan menor cantidad de accesos a stos ltimos, 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 desempeo de este mtodo se ve impactado por la penalidad de calcular la matriz traspuesta, porque deben accederse simultneamente los elementos simtricos respecto a la diagonal (que pertenecen a distintas filas y columnas de la matriz transponer).

1.3. Multiplicacin por bloques

Es posible dividir la matriz en bloques de tamao arbitrario y, utilizando el mtodo bsico, realizar multiplicaciones sucesivas de matrices mas pequeas, mejorando la localidad temporal, ya que se mantienen en la cach los datos que se usarn a corto plazo.

1.4. Multiplicacin por columnas.

Este mtodo realiza la multiplicacin 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 proporcionanan fundamento matemtico al mtodo. Por simplicidad, se utilizan matrices de 2x2.

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

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 parntesis resulta

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

As, 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 utilizacin de la memoria cache.

Para que este mtodo 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 vas para evitar que se reemplacen entre si las columnas de las distintas matrices.

Por ltimo, si el tamao de la matriz a multiplicar es muy grande, las columnas de cada matriz no se pueden mapear completmente en cada lnea de la cach obteniendo una mayor tasa de miss y una consecuente disminucin del rendimiento.

1.5. Multiplicacin por columnas, usando operacin en bloques

Este mtodo es la unin de los 2 anteriores, combinando las ventajas de cada uno. Bsicamente, con la operacion por bloques se logra reducir el problema que se produce en el mtodo de multiplicacin por columnas, al operar con matrices de gran tamao. Si se escoje el valor adecuado para el tamao de bloque, se puede lograr operar con matrices pequeas cuyas columnas entran completamente en cada linea de la cach. De esta manera, se hace un gran aprovechamiento de la localidad espacial y temporal.

2. Hardware utilizado

El programa de multiplicacin de matrices, con cada optimizacin implementada, se ejecut en dos configuraciones de hardware que se describen a continuacin.

3. Corridas

3.1. NIETOOO

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

Tam: 31 mflop/s: 248.333
Tam: 32 mflop/s: 247.404
Tam: 96 mflop/s: 34.3216
Tam: 97 mflop/s: 36.2246
Tam: 127 mflop/s: 33.1154
Tam: 128 mflop/s: 6.83108
Tam: 129 mflop/s: 33.0144
Tam: 191 mflop/s: 15.3798
Tam: 192 mflop/s: 8.32734
Tam: 229 mflop/s: 15.4441
Tam: 255 mflop/s: 15.6119
Tam: 256 mflop/s: 3.29504
Tam: 257 mflop/s: 15.6049
Tam: 319 mflop/s: 15.692
Tam: 320 mflop/s: 4.0728
Tam: 321 mflop/s: 15.7135
Tam: 417 mflop/s: 15.8881
Tam: 479 mflop/s: 15.9132
Tam: 480 mflop/s: 7.00109
Tam: 511 mflop/s: 31.8143
Tam: 512 mflop/s: 6.52512
Tam: 639 mflop/s: 30.8007
Tam: 640 mflop/s: 5.12321
Tam: 767 mflop/s: 15.5318
Tam: 768 mflop/s: 6.51578
Tam: 769 mflop/s: 30.8639

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

Tam: 31 mflop/s: 249.46
Tam: 32 mflop/s: 250.019
Tam: 96 mflop/s: 37.2366
Tam: 97 mflop/s: 28.562
Tam: 127 mflop/s: 23.3121
Tam: 128 mflop/s: 6.36637
Tam: 129 mflop/s: 23.5611
Tam: 191 mflop/s: 16.439
Tam: 192 mflop/s: 8.46746
Tam: 229 mflop/s: 15.9371
Tam: 255 mflop/s: 15.9418
Tam: 256 mflop/s: 4.1076
Tam: 257 mflop/s: 15.9414
Tam: 319 mflop/s: 15.9699
Tam: 320 mflop/s: 4.90446
Tam: 321 mflop/s: 15.9753
Tam: 417 mflop/s: 16.0427
Tam: 479 mflop/s: 15.9177
Tam: 480 mflop/s: 11.5488
Tam: 511 mflop/s: 31.8612
Tam: 512 mflop/s: 8.17854
Tam: 639 mflop/s: 31.0794
Tam: 640 mflop/s: 8.07775
Tam: 767 mflop/s: 31.0675
Tam: 768 mflop/s: 8.14393
Tam: 769 mflop/s: 31.0148

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

Tam: 31 mflop/s: 248.132
Tam: 32 mflop/s: 248.747
Tam: 96 mflop/s: 177.036
Tam: 97 mflop/s: 178.48
Tam: 127 mflop/s: 165.249
Tam: 128 mflop/s: 120.872
Tam: 129 mflop/s: 165.771
Tam: 191 mflop/s: 171.19
Tam: 192 mflop/s: 153.313
Tam: 229 mflop/s: 156.775
Tam: 255 mflop/s: 93.3651
Tam: 256 mflop/s: 17.4113
Tam: 257 mflop/s: 92.5084
Tam: 319 mflop/s: 172.909
Tam: 320 mflop/s: 148.963
Tam: 321 mflop/s: 169.576
Tam: 417 mflop/s: 169.185
Tam: 479 mflop/s: 166.504
Tam: 480 mflop/s: 163.06
Tam: 511 mflop/s: 39.3729
Tam: 512 mflop/s: 7.04778
Tam: 639 mflop/s: 146.114
Tam: 640 mflop/s: 76.2559
Tam: 767 mflop/s: 83.2028
Tam: 768 mflop/s: 10.1927
Tam: 769 mflop/s: 47.4224

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

Tam: 31 mflop/s: 254.374
Tam: 32 mflop/s: 254.856
Tam: 96 mflop/s: 145.172
Tam: 97 mflop/s: 114.614
Tam: 127 mflop/s: 25.4408
Tam: 128 mflop/s: 25.7606
Tam: 129 mflop/s: 25.3055
Tam: 191 mflop/s: 16.9372
Tam: 192 mflop/s: 17.3264
Tam: 229 mflop/s: 15.7353
Tam: 255 mflop/s: 15.6484
Tam: 256 mflop/s: 16.1971
Tam: 257 mflop/s: 16.0698
Tam: 319 mflop/s: 16.1725
Tam: 320 mflop/s: 16.2181
Tam: 321 mflop/s: 16.1775
Tam: 417 mflop/s: 16.2502
Tam: 479 mflop/s: 16.2819
Tam: 480 mflop/s: 16.2959
Tam: 511 mflop/s: 16.2979
Tam: 512 mflop/s: 16.3447
Tam: 639 mflop/s: 16.1341
Tam: 640 mflop/s: 16.5712
Tam: 767 mflop/s: 16.0341
Tam: 768 mflop/s: 16.0436
Tam: 769 mflop/s: 18.2927

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

Tam: 31 mflop/s: 262.949
Tam: 32 mflop/s: 264.199
Tam: 96 mflop/s: 153.68
Tam: 97 mflop/s: 150.534
Tam: 127 mflop/s: 37.9969
Tam: 128 mflop/s: 35.7207
Tam: 129 mflop/s: 35.5977
Tam: 191 mflop/s: 32.474
Tam: 192 mflop/s: 32.6756
Tam: 229 mflop/s: 30.5895
Tam: 255 mflop/s: 30.7529
Tam: 256 mflop/s: 30.5795
Tam: 257 mflop/s: 30.7954
Tam: 319 mflop/s: 31.1137
Tam: 320 mflop/s: 30.9928
Tam: 321 mflop/s: 31.1081
Tam: 417 mflop/s: 31.4333
Tam: 479 mflop/s: 31.5829
Tam: 480 mflop/s: 31.5573
Tam: 511 mflop/s: 31.6399
Tam: 512 mflop/s: 31.608
Tam: 639 mflop/s: 15.8988
Tam: 640 mflop/s: 15.8806
Tam: 767 mflop/s: 15.9071
Tam: 768 mflop/s: 15.8961
Tam: 769 mflop/s: 28.2475