

Paralelización de EigenFaces en un Esquema de Memoria Compartida

Chichizola Franco, Champredonde Raúl, Armando De Giusti

{francoch, rchampre, degiusti}@lidi.info.unlp.edu.ar

Universidad Nacional de La Plata - Facultad de Informática

Laboratorio de Investigación y Desarrollo en Informática

Calle 50 y 115 Primer Piso. 1900 La Plata. Bs.As. Argentina - Tel/Fax: 54 221 422 7707

Resumen

En el ámbito del Laboratorio de Investigación y Desarrollo en Informática de la Facultad de Informática de la Universidad Nacional de La Plata, se ha desarrollado un sistema de reconocimiento de rostros. Luego, con el fin de demostrar la conveniencia de su paralelización para mejorar los tiempos de ejecución, se realizaron una serie de experimentos con los algoritmos más importantes que componen dicho sistema. Finalmente el sistema fue paralelizado utilizando el lenguaje de programación Ada sobre la arquitectura paralela de la supercomputadora SGI Origin 2000 conocida como Clementina 2 y se verificó la ganancia de rendimiento.

En este artículo se describen los algoritmos principales del sistema de reconocimiento de rostros, los experimentos realizados, el análisis de sus resultados y la paralelización del sistema.

1. Introducción

En el marco de los proyectos del Laboratorio de Investigación y Desarrollo en Informática de la Facultad de Informática de la Universidad Nacional de La Plata, se ha desarrollado un sistema para el reconocimiento de rostros [5] como tesina de grado. Básicamente, el sistema utiliza una base de imágenes de rostros y extrae ciertas características de cada uno de estos, en un proceso llamado *entrenamiento*.

El sistema entrenado es capaz de recibir la imagen de un rostro y determinar si se corresponde con alguno de los rostros que se encuentran en su base de imágenes y en tal caso, a cuál de ellos se corresponde. Este proceso es llamado *reconocimiento*.

Se pueden encontrar muchas aplicaciones que utilicen esta tecnología, sin embargo la mayoría de ellas, si no todas, necesitarían del mejor rendimiento posible y sobretodo si requiere de una base de imágenes con numerosos rostros.

La cantidad de cómputo necesario tanto para el entrenamiento como para el reconocimiento, crece en forma proporcional a la cantidad de rostros de la base.

Es claro que el paralelismo de grano grueso beneficiaría el desempeño de la aplicación. Por ejemplo, no es difícil probar que en el proceso de entrenamiento, el cálculo de los eigenvectores de un conjunto de imágenes en paralelo requiere aproximadamente el mismo tiempo que el mismo cálculo para una única imagen. Sin embargo, en esta oportunidad se pretende explotar un paralelismo de menor granularidad.

El lenguaje utilizado es el Ada. La elección responde a varios factores, de los cuales el principal es que, de los lenguajes instalados en Clementina 2, los mecanismos de especificación de concurrencia y paralelismo de Ada son los que mejor se adaptan al esquema de memoria compartida de la arquitectura utilizada.

Los algoritmos de entrenamiento y de reconocimiento se describen en la sección 2. Ellos se basan principalmente en operaciones sobre matrices, de las cuales se considera a la multiplicación como la más representativa, y en el cálculo de autovalores y autovectores. En la sección 3 se describen distintos métodos para el cálculo de estos. En la sección 4 se describe la experimentación realizada tanto sobre la multiplicación de matrices como sobre el cálculo de autovalores y autovectores. Finalmente, en la sección 5 se comentan los resultados de la paralelización de la aplicación de reconocimiento de rostros.

1.1.1 Algoritmos de entrenamiento y reconocimiento

La aplicación de reconocimiento de rostros se basa en el método EigenFaces [8][9], el cual consta fundamentalmente de un proceso de entrenamiento y un proceso de reconocimiento. Ambos utilizan una base de rostros compuesta por un conjunto $\Gamma_1, \Gamma_2, \dots, \Gamma_M$ de imágenes de dimensión de $D \times D$.

El entrenamiento consiste en los siguientes pasos:

- 1- Cada imagen es dividida en bloques de $P \times P$. Cada uno de estos bloques se promedia y se obtiene una nueva imagen de $N \times N$, con $N = D / P$, que se obtiene de reemplazar cada bloque por su promedio.
- 2- Cada imagen Γ_i con $i = 1, 2, \dots, M$ es reorganizada como un vector de dimensión N^2 cuyo valor es construido como la concatenación de cada una de las filas de la imagen, formando así una matriz de $N^2 \times M$.
- 3- Se obtiene el rostro promedio Ψ según la fórmula

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n \quad (1)$$

4- El rostro promedio Ψ obtenido es restado a cada una de las imágenes Γ_i con $i=1..M$ obteniendo un nuevo conjunto de vectores $\Phi_i = \Gamma_i - \Psi$ que conforman la matriz $\Lambda = [\Phi_1, \Phi_2, \dots, \Phi_M]$ de $N^2 \times M$.

5- Se obtiene la matriz de covarianza

$$L = \frac{1}{N^2} \Lambda \Lambda^T \quad (2)$$

de dimensiones $N^2 \times N^2$.

6- Se obtienen los autovectores de L , los que ordenados de mayor a menor según sus correspondientes autovalores, conforman la matriz v

7- Se calculan los “eigenfaces” u como combinación lineal de las columnas de Λ y las columnas de v

$$u_i = \sum_{k=1}^{N^2} v_{ik} \Phi_k \quad (3)$$

8- Se obtiene el patrón $\Omega_i^T = [\omega_1, \omega_2, \dots, \omega_M]$ con $i=1,2,\dots,M$ donde $\omega_k = u_k^T (\Gamma_i - \Psi)$ con $k=1,2,\dots,N^2$.

Dado la imagen de un rostro como entrada del sistema, el proceso de reconocimiento intenta encontrar en la base de imágenes, aquella que se corresponde con el rostro dado, para lo cual se calcula su patrón Ω utilizando el mismo procedimiento anteriormente descrito, y se busca la distancia mínima

$$\min(\|\Omega - \Omega_i\|^2) \text{ con } i=1,2,\dots,M$$

Una vez encontrada la distancia mínima, si la misma está dentro de los límites estipulados por el porcentaje de error admitido que se recibe como parámetro, el sistema indica cuál es la imagen correspondiente. En caso contrario indica la no existencia de correspondencia.

Casi la totalidad de los pasos de los algoritmos descriptos involucran operaciones de matrices simples, con la excepción del paso 6, en el cual se calculan los autovalores y autovectores de una matriz. Este no es un cálculo simple como podrían ser la suma o la multiplicación de matrices. Más bien es un cálculo que consume una gran cantidad de tiempo, que está directamente relacionado con el tamaño de la matriz, pero también con la distribución de sus datos.

De hecho el paso 6 es el que consume la gran mayoría del tiempo necesario para el entrenamiento.

2. Cálculo de autovalores y autovectores

Por definición, dada una matriz simétrica $A \in R^{n \times n}$ existe una matriz ortogonal Q tal que:

$$Q^T A Q = \text{Diag}(\lambda_1, \dots, \lambda_n) \quad (4)$$

donde λ_i con $i=1,\dots,n$, son los autovalores de la matriz A , y Q es la matriz con los autovectores correspondientes. Una forma de calcular los autovalores y autovectores de una matriz simétrica cuadrada de números reales, es el método de Jacobi [10].

En esencia, si A es la matriz para la cual se desea calcular los autovalores y autovectores, y V es la matriz identidad, el método de Jacobi realiza una sucesión de modificaciones sobre ellas. Cada modificación hace que la matriz A sea cada vez “más diagonal”, hasta llegar a un punto en el cual los elementos que se encuentran fuera de la diagonal principal tienen un valor absoluto lo suficientemente pequeño como para considerarlo cero. En ese momento, termina la secuencia de modificaciones, resultando A en una matriz diagonal cuyos elementos de la diagonal principal son los autovalores y V en la matriz que contiene a los autovectores.

El algoritmo correspondiente es el siguiente:

1- Inicialmente V es la matriz identidad

2- Para cada par de índices (p, q) tal que $1 \leq p < q \leq n$:

2.1- Se realiza la *Rotación de Jacobi*. Para eso, utilizando la *descomposición de Schur* [7], se obtiene un par de valores (c, s) , tal que:

$$\begin{vmatrix} b_{pp} & 0 \\ 0 & b_{qq} \end{vmatrix} = \begin{vmatrix} c & s \\ -s & c \end{vmatrix}^T * \begin{vmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{vmatrix} * \begin{vmatrix} c & s \\ -s & c \end{vmatrix} \quad (5)$$

Se forma $J(p, q)$, matriz de $n \times n$, producto de reemplazar en la matriz identidad, c por los elementos de las posiciones (p, p) y (q, q) , s por el elemento de la posición (p, q) , y $-s$ por el elemento de la posición (q, p) , obteniendo entonces la matriz

$$J(p, q) = \begin{vmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{vmatrix} \begin{matrix} p \\ q \end{matrix} \quad (6)$$

2.2- Reemplazar el valor de la matriz A por el resultado de la expresión $J(p, q)^T A J(p, q)$

2.3- Reemplazar el valor de la matriz V por el resultado de la expresión $V J(p, q)$

3- Si los elementos a_{ij} de la matriz A , con $i \neq j$, $i=1, \dots, n$, $j=1, \dots, n$, no tienen un valor suficientemente pequeño como para ser considerado cero, entonces volver al paso 2.

El algoritmo termina con los autovectores en la matriz V y los autovalores correspondientes en la diagonal principal de A .

2.1.1 Optimización del método de Jacobi clásico

El algoritmo de Jacobi clásico podría estar en el orden $O(n^5)$ o mayor aún dependiendo de los valores de la matriz A . Si se pretende aplicarlo a matrices de tamaños relativamente grandes, se hace imprescindible reducir los tiempos de procesamiento necesarios.

Para ello, un primer paso ataca el hecho de que en el método de Jacobi clásico se realizar varias multiplicaciones de matrices para solamente modificar dos filas y dos columnas de una de ellas. Se logra cierta reducción del procesamiento necesario, modificando las filas y columnas mencionadas sin necesidad de trabajar con el resto de los valores, que no afectan a esos datos.

2.1.2 Método de Jacobi por bloques

Suponiendo que $n = N * r$ y que se particiona la matriz A en $N \times N$ bloques de $r \times r$, quedando entonces

$$A = \begin{vmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{N1} & \cdots & A_{NN} \end{vmatrix} \quad (7)$$

donde cada elemento A_{ij} es un bloque de $r \times r$ valores reales.

El algoritmo por bloques es semejante al Jacobi clásico optimizado, pero considerando que mientras antes el par de índices (p, q) indicaba la posición de un valor real a_{pq} dentro de la matriz A , ahora hace referencia a la posición de un bloque A_{pq} . Por lo tanto:

- a- El rango dentro del cual varían los índices p y q ya no es $[1, \dots, n]$ sino $[1, \dots, N]$
- b- En la Rotación de Jacobi (paso 2.1 del algoritmo de Jacobi clásico), en vez de la descomposición de Schur que permite obtener un par de valores (c, s) que cumple con (5), se utiliza la descomposición que permite obtener los bloques V_{pp} , V_{pq} , V_{qp} y V_{qq} tales que

$$\begin{vmatrix} D_{pp} & 0 \\ 0 & D_{qq} \end{vmatrix} = \underbrace{\begin{vmatrix} V_{pp} & V_{pq} \\ V_{qp} & V_{qq} \end{vmatrix}}_{V^{[p,q]}}{}^T * \underbrace{\begin{vmatrix} A_{pp} & A_{pq} \\ A_{qp} & A_{qq} \end{vmatrix}}_{A^{[p,q]}} * \begin{vmatrix} V_{pp} & V_{pq} \\ V_{qp} & V_{qq} \end{vmatrix} \quad (8)$$

La submatriz $V^{[p,q]}$ son los autovectores de $A^{[p,q]}$, y $Diag(D_{pp}) \cup Diag(D_{qq})$ son los autovalores, con D_{pp} y D_{qq} matrices diagonales. Así, para obtener $V^{[p,q]}$, en vez de utilizar la descomposición de Schur se usa el método de Jacobi clásico optimizado. Con los bloques V_{pp} , V_{pq} , V_{qp} y V_{qq} obtenidos, se forma la matriz $J(p, q)$.

2.1.3 Método de Jacobi por bloques paralelo

Se encontró que el método de Jacobi por bloques resulta ser muy adecuado para el cálculo de autovalores y autovectores en paralelo, debido a que en cada rotación de Jacobi, los bloques de las filas y las columnas p y q se modifican recién en los pasos 2.2 y 2.3. Por lo tanto, para todo par de índices (p_1, q_1) y (p_2, q_2) , es posible realizar simultáneamente el paso 2.1 de la rotación de Jacobi sin que se produzca ningún conflicto entre ellos, con la única condición de que los valores de p_1 , p_2 , q_1 y q_2 sean distintos entre sí.

Recién en los pasos 2.2 y 2.3 aparece la necesidad de exclusión mutua sobre las matrices A y V , dado que es el momento en el cual las mismas son modificadas.

Por inducción se puede demostrar que esta propiedad es extensible a $N/2$ pares de índices. Es decir que también se cumple para $(p_1, q_1), \dots, (p_{N/2}, q_{N/2})$ siempre que se cumplan las siguientes condiciones:

$$p_i \neq p_j, \forall i, j = 1..N, i \neq j ; \quad p_i \neq q_j, \forall i, j = 1..N ; \quad q_i \neq q_j, \forall i, j = 1..N, i \neq j$$

Esta propiedad indica que pueden ejecutarse simultáneamente $N/2$ rotaciones de Jacobi (paso 2.1), y luego se realizan las modificación de las matrices A y V (pasos 2.2 y 2.3) en forma secuencial para cada par de índices.

2.1.4 Optimización con BLAS

En los algoritmos por bloque, tanto los secuenciales como el paralelo, las modificaciones correspondientes a los pasos 2.2 y 2.3 que se realizan sobre las filas y las columnas p y q de las matrices A y V , involucran una sucesión de multiplicaciones entre bloques de matrices, que no son necesarias en los métodos que no utilizan bloques.

El uso de librerías existentes de optimización permiten lograr el mejor rendimiento de una arquitectura en particular para las operaciones convencionales sobre matrices. En particular, en este trabajo se utilizó la función `cblas_sgemm` para reducir el tiempo de procesamiento que insumen las multiplicaciones de los bloques de las matrices, obteniendo de esta manera el mejor rendimiento posible para la multiplicación de matrices sobre la arquitectura utilizada.

3. Experimentación

La fase de experimentación de este trabajo, incluyó dos tipos de experimentos distintos para determinar y medir la mejora del rendimiento que se puede lograr por medio de la paralelización.

Por un lado se experimentó con la multiplicación de matrices debido a que se considera que esta operación es la más representativa de BLAS nivel 3 [1] y hay diversas investigaciones realizadas al respecto [2][3][4][6][7]. Por otro lado se experimentó con el cálculo de autovalores y autovectores, por ser este un procesamiento cuyo tiempo de respuesta no sólo está relacionado con el tamaño de la matriz sino también con la distribución de sus valores, y por ser el proceso que insume casi la totalidad del tiempo de entrenamiento.

En ambos casos los experimentos realizados fueron ejecutados en la supercomputadora conocida con el nombre Clementina 2 [11][12][13]. Clementina 2 es un sistema Cray Origin 2000 fabricado por la firma SGI (nuevo nombre de Silicon Graphics). Cuenta con cuarenta procesadores MIPS RISC R12000 de 300 MHz., con un cache secundario de 4 MB, una memoria principal de 10 GB compartida por todos los procesadores, 360 GB de almacenamiento en disco. El sistema operativo de esta supercomputadora es el Irix 6.5 [14], el cual fue desarrollado por Silicon Graphics. Irix es compatible (compliant) con UNIX System V Release 4 y con estándares del The Open Group, incluyendo UNIX 95, Year 2000 y POSIX. Irix posee soporte para multiprocesamiento simétrico (SMP) escalable hasta 128 procesadores y soporte para 32 y 64 bits.

3.1. Multiplicación de matrices

Para esta etapa de la experimentación se utilizaron matrices cuadradas de $N \times N$ con $N = 250, 500, 1000, 1500, 2000$.

La multiplicación se divide entre varias tareas. Cada una de estas se encarga de realizar los cálculos correspondientes a una parte de la matriz resultante. La división de la matriz resultante se realiza por filas, de acuerdo al siguiente esquema: Siendo N la dimensión de la matriz y T la cantidad de tareas utilizadas, se divide la matriz resultante en T grupos de N/T filas cada uno. La tarea T_i se encarga de calcular los valores correspondientes al grupo de filas T_i con $i = 1, 2, \dots, T$.

Las pruebas se realizaron utilizando T tareas con $T = 1, 5, 10, 20, 30, 40$, donde $T = 1$ es la solución secuencial.

Con el fin de utilizar la división de la matriz resultante más conveniente en relación a la influencia que esta puede tener sobre el aprovechamiento de la memoria cache, se identificó la manera en que el lenguaje Ada representa internamente las matrices. El compilador de Ada para la arquitectura utilizada representa a las matrices por filas. El mismo criterio se utilizó entonces para dividir la matriz resultante en tantas porciones como tareas.

Las condiciones de trabajo fueron las de uso normal, es decir, sin contar con acceso exclusivo a la arquitectura. Esas condiciones hacen que la cantidad de procesadores utilizados dependa de la carga de trabajo que cada uno de ellos tenga en el momento de la ejecución del proceso de multiplicación de matrices, y en el caso particular de Clementina 2, esa carga es habitualmente alta.

El sistema operativo permite obtener el tiempo total de ejecución efectiva del proceso. Este tiempo es igual a la suma de los períodos de tiempo en los que cada una de las tareas que componen el proceso ocupan un procesador.

Por otro lado, el tiempo de creación de las tareas y de las matrices operandos y resultante, es despreciable en relación con el tiempo de procesamiento, así como la influencia del uso de la memoria cache.

Por consiguiente, si se tuviera libre acceso a la totalidad de la arquitectura de Clementina 2, el tiempo total de procesamiento sería aproximadamente igual al tiempo de ejecución de una sola de las tareas que el proceso utiliza.

Este es el criterio que se utiliza para calcular el rendimiento con acceso exclusivo, en función del rendimiento en condiciones de trabajo normales.

3.2. Resultados obtenidos para la multiplicación de matrices

Los resultados obtenidos se muestran en las siguientes cinco tablas. En cada tabla se detalla, para cada una de las distintas cantidades de tareas utilizadas, el tiempo de usuario y el tiempo de sistema acumulados que insumió la ejecución del proceso, el tiempo total como suma de los dos anteriores, el tiempo de cada tarea, el incremento de velocidad (speedup) de las soluciones paralelas respecto de la secuencial y los Mflops alcanzados. Todas las medidas de tiempo están expresadas en segundos.

Tabla 1: Matrices de 250x250

Tareas	T.Usuario	T.Sistema	Tiempo total	Tiempo tarea	Speedup	Mflops
1	1,973	0,025	1,998	1,998	1,000	15,609
10	1,989	0,023	2,012	0,201	9,930	155,007
20	1,976	0,034	2,010	0,101	19,880	310,317
30	2,007	0,040	2,047	0,068	29,278	457,006
40	2,009	0,056	2,065	0,052	38,695	604,007

Tabla 2: Matrices de 500x500

Tareas	T.Usuario	T.Sistema	Tiempo total	Tiempo tarea	Speedup	Mflops
1	15,889	0,054	15,943	15,943	1,000	15,665
10	15,881	0,070	15,951	1,595	9,995	156,573
20	16,019	0,075	16,094	0,805	19,812	310,364
30	16,006	0,082	16,088	0,536	29,729	465,713
40	16,108	0,100	16,208	0,405	39,345	616,350

Tabla 3: Matrices de 1000x1000

Tareas	T.Usuario	T.Sistema	Tiempo total	Tiempo tarea	Speedup	Mflops
1	162,420	0,256	162,676	162,676	1,000	12,288
10	163,464	0,276	163,740	16,374	9,935	122,084
20	168,993	0,338	169,331	8,467	19,214	236,106
30	165,542	0,306	165,848	5,528	29,426	361,595
40	166,494	0,390	166,884	4,172	38,991	479,135

Tabla 4: Matrices de 1500x1500

Tareas	T.Usuario	T.Sistema	Tiempo total	Tiempo tarea	Speedup	Mflops
1	661,273	0,646	661,919	661,919	1,000	10,194
10	683,000	1,019	684,019	68,402	9,677	98,649
20	688,620	1,259	689,879	34,494	19,189	195,621
30	687,938	1,312	689,250	22,975	28,810	293,700
40	682,575	1,064	683,639	17,091	38,729	394,814

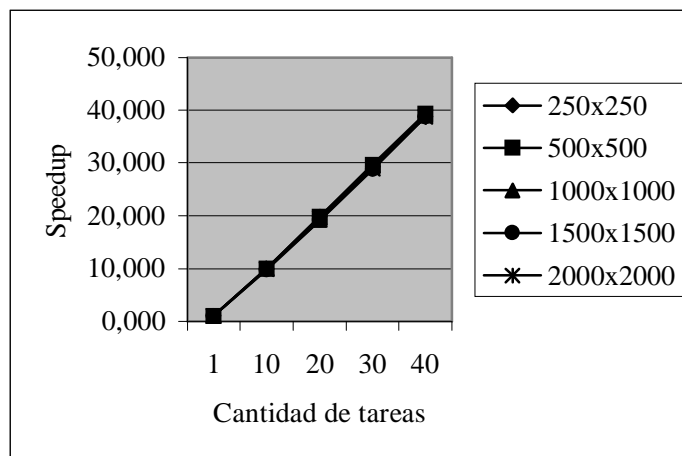
Tabla 5: Matrices de 2000x2000

Tareas	T.Usuario	T.Sistema	Tiempo total	Tiempo tarea	Speedup	Mflops
1	1840,688	1,843	1842,531	1842,531	1,000	8,682
10	1871,282	2,133	1873,415	187,342	9,835	85,384
20	1884,615	2,851	1887,466	94,373	19,524	169,497
30	1907,235	3,122	1910,357	63,679	28,935	251,199
40	1898,128	3,324	1901,452	47,536	38,761	336,501

Estos resultados no pueden mostrarse en un mismo gráfico pues las escalas temporales son excesivamente disímiles. Sin embargo, si se genera un grafico para cada una de las tablas anteriores, estos muestran curvas muy similares entre sí. Esto indica que el rendimiento tiene aproximadamente la misma mejora para todas las pruebas según se incrementa el número de procesadores utilizados. Para ver más precisamente la mejora del rendimiento, en la Tabla 6 se muestran los incrementos de la velocidad (speedup) de las soluciones paralelas respecto de la solución secuencial obtenidos en las distintas pruebas, y en la Figura 1 el gráfico correspondiente.

Tabla 6: Speedups obtenidos

Tam. Matriz	250x250	500x500	1000x1000	1500x1500	2000x2000
Cant. tareas					
1	1,000	1,000	1,000	1,000	1,000
10	9,930	9,995	9,935	9,677	9,835
20	19,880	19,812	19,214	19,189	19,524
30	29,278	29,729	29,426	28,810	28,935
40	38,695	39,345	38,991	38,729	38,761

Figura 1: Speedups obtenidos

De acuerdo a la forma de las curvas que representan los speedups obtenidos a partir de las distintas mediciones, se aprecia que este es lineal y muy cercano al óptimo. La mejora del rendimiento es directamente proporcional a la cantidad de procesadores utilizados.

Este speedup lineal y casi óptimo, se logra simplemente dividiendo el procesamiento en tareas para aprovechar la disponibilidad de varios procesadores. Si además se utiliza la librería de optimización BLAS se obtiene prácticamente la máxima potencia de cálculo de una arquitectura, pudiendo mejorar el rendimiento en unas 25 veces. Las pruebas correspondientes a la multiplicación de matrices usando BLAS fueron obviadas debido a que las mejoras en el rendimiento que se obtendrían, serían directamente proporcionales, y por lo tanto los speedups seguirían siendo los mismos.

3.3. Cálculo de autovalores y autovectores

Esta parte de la experimentación incluyó la prueba de los siguientes métodos:

- Método de Jacobi clásico optimizado
- Método de Jacobi por bloques sin Blas

- (c) Método de Jacobi por bloques con Blas
- (d) Método de Jacobi por bloques paralelo con Blas

Los cuatro algoritmos fueron probados con matrices cuadradas de tamaño $n \times n$ con $n = 100, 200, 300, \dots, 900, 1000$.

En los algoritmos por bloques secuenciales se hizo variar el tamaño de los bloques entre 5 y 500 con pequeñas variaciones entre uno y otro. Se debe considerar que el tiempo de procesamiento necesario para obtener los autovectores a través del método Jacobi, no depende solamente del tamaño de la matriz, sino también de los datos que hay en ella. Se probaron todos los casos con los mismos datos para evitar las posibles influencias de la distribución de los datos. Naturalmente el resultado de las pruebas realizadas puede variar si se utilizan distintas distribuciones de datos.

El tiempo utilizado por los algoritmos secuenciales, se calculó en base al tiempo de uso efectivo de procesadores, el cual es informado por el sistema operativo. Este método de medición no sirve para el caso del algoritmo paralelo debido a que el tiempo de respuesta depende de la carga de trabajo que tenga cada uno de los procesadores del sistema.

3.4. Resultados obtenidos para el cálculo de autovalores y autovectores

La Tabla 7 muestra los tiempos de respuesta obtenidos con el método de Jacobi clásico optimizado, caso de prueba (a), para cada uno de los tamaños de matriz utilizados.

Tabla 7: Método de Jacobi Clásico Optimizado

Tamaño de la matriz	Tiempo de respuesta (seg.)
100	1,281
200	11,480
300	41,808
400	111,066
500	232,260
600	428,852
700	746,798
800	1094,764
900	1634,949
1000	2281,703

Tabla 8: Método de Jacobi por bloques sin Blas

Tam. Matriz	100	200	300	400	500	600	700	800	900	1000
Tam. Bloque										
10	5,959									
25	4,455	45,353								
50	1,447	36,384	137,869	365,230	789,346		2359,443			
75			123,875							
100		12,625		293,904		1212,386		3227,975		
125					639,436					6380,442
150			45,173			1117,207			4204,082	
175							1781,717			
200				119,455				2754,785		
225									3891,822	
250					251,921					5516,127
300						464,495				
350							781,436			
400								1199,372		
450									1749,360	
500										2488,214

Para el caso de prueba (b), los tiempos obtenidos son los que se muestran en la Tabla 8. No fue necesario realizar las mediciones de todas las combinaciones posibles de tamaños de matriz y de bloques, resultando suficiente la muestra tomada. En ella puede verse que el tiempo de respuesta es inversamente proporcional al tamaño de los bloques utilizados. Por lo tanto el mejor tiempo de respuesta se obtiene cuando se utiliza el tamaño de bloque máximo. Es decir que para una matriz de $n \times n$, el mejor tamaño de bloque es $r \times r$ con $n = 2 * r$.

La Tabla 9 muestra los resultados obtenidos para el método de Jacobi por bloques utilizando Blas. En este caso fue necesario un mayor número de pruebas que en el caso anterior para precisar el tamaño de bloque óptimo para cada tamaño de matriz.

Tabla 9: Método de Jacobi por bloques con Blas

Tam. Matriz	100	200	300	400	500	600	700	800	900	1000
Tam. Bloque										
5	1,865	14,316	48,710	123,674	249,648	423,307	721,985	1086,669	1531,781	2112,167
10	1,886	11,111	34,380	80,299	156,531	277,886	431,312	674,344	935,106	1284,376
25	2,367	14,793	41,677	85,646	145,031	245,786	352,080	489,168	695,117	909,138
50	1,304	19,205	54,747	121,438	209,375	332,174	497,340	685,709	846,915	1169,318
75			64,514							
100		11,458		155,695		466,817		988,419		1822,365
125					326,577					1991,821
150			41,865			588,439			1627,207	
175							925,107			
200				110,845				1469,498		
225									2069,692	
250					230,299					2999,736
300						434,975				
350							724,982			
400								1104,597		
450									1617,041	
500										2307,555

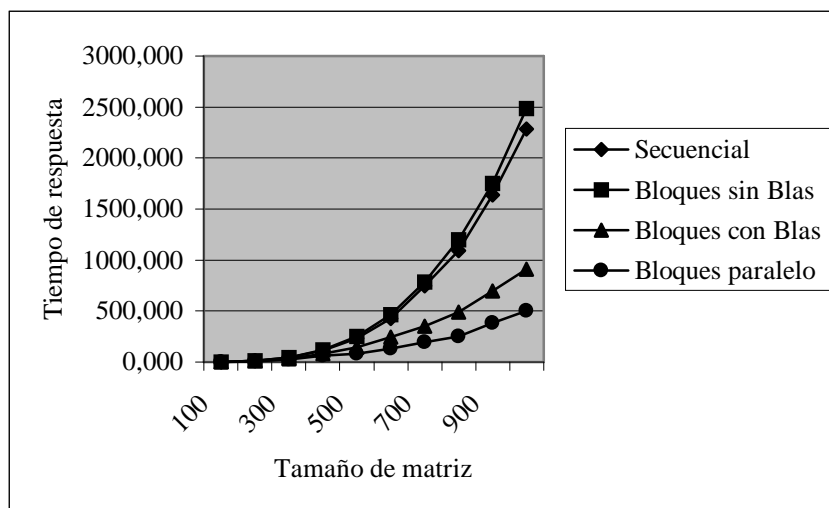
Tabla 10: Método de Jacobi por bloques paralelo con Blas

Tamaño de la matriz	Tiempo de respuesta (seg.)	Tamaño de bloque óptimo	Cantidad tareas
100	1,278	50	1
200	8,304	10	10
300	26,531	10	15
400	64,793	10	20
500	82,033	25	10
600	130,225	25	12
700	193,719	25	14
800	252,635	25	16
900	382,657	25	18
1000	499,879	25	20

Tabla 11: Mejores tiempos de respuesta (seg.)

Método	Secuencial	Bloques sin Blas	Bloques con Blas	Bloques paralelo
Tam. Matriz				
100	1,281	1,447	1,304	1,278
200	11,480	12,625	11,111	8,304
300	41,808	45,173	34,380	26,531
400	111,066	119,455	80,299	64,793
500	232,260	251,921	145,031	82,033
600	428,852	464,495	245,786	130,225
700	746,798	781,436	352,080	193,719
800	1094,764	1199,372	489,168	252,635
900	1634,949	1749,360	695,117	382,657
1000	2281,703	2488,214	909,138	499,879

Figura 2: Tiempos de los distintos métodos comparados



Se encuentra que el tamaño de bloque para el cual se logra el mejor rendimiento depende del tamaño de la matriz, pudiendo identificarse tres rangos. Para matrices de tamaño 100, el mejor rendimiento se obtiene con bloques del mayor tamaño posible, es decir 50. Para matrices de tamaño 200, 300 y 400, el mejor tamaño de bloque es 10, mientras que para los tamaños 500 a 1000 el mejor rendimiento se obtiene con bloques de tamaño 25.

Las pruebas del método de Jacobi por bloques paralelo con Blas fueron realizadas utilizando los tamaños de bloque óptimos para cada tamaño de matriz, obtenidos en las pruebas del método anterior. Los tiempos de respuesta medidos, así como el tamaño de bloque y la cantidad de tareas utilizados se muestran en la Tabla 10.

Finalmente en la Tabla 11 se resumen los mejores tiempos de respuesta obtenidos por los distintos métodos para cada tamaño de matriz. Para los métodos por bloque se utilizan los tiempos de respuesta obtenidos con los mejores tamaños de bloque.

En el gráfico de la Figura 2 se comparan las curvas correspondientes a los tiempos obtenidos por los cuatro métodos. De la experimentación realizada se desprende que:

- El método secuencial muestra una forma del tipo polinomial. El tiempo de respuesta crece más rápidamente que el tamaño de la matriz.
- El método por bloques sin Blas no sólo no mejora los tiempos de respuesta obtenidos por el método clásico optimizado sino que los empeora. El tamaño de bloque óptimo encontrado para el método por bloques sin Blas resultó ser el mayor posible, es decir de la mitad del tamaño de la matriz. En este caso, el método por bloques resulta algorítmicamente igual al clásico optimizado, y por esta razón obtiene un rendimiento similar más un pequeño costo adicional (overhead).
- El método por bloques con Blas obtiene una mejora sustancial en el rendimiento sacando provecho de la cantidad de multiplicaciones de matrices que se deben realizar y de la optimización de las mismas, encontrando el mejor tamaño de bloque en el compromiso (trade off) entre el tamaño de las matrices a multiplicar y la cantidad de multiplicaciones necesarias.
- El método por bloque paralelo logra mejor rendimiento que los demás métodos. Es interesante destacar que para este método los tiempos medidos son tiempos máximos, es decir que podrían mejorarse sensiblemente si se tuviera acceso exclusivo a la arquitectura.

4. Aplicación al sistema de reconocimiento de rostros

Los resultados obtenidos indican que el uso de paralelismo a nivel de operaciones sobre matrices resulta sumamente beneficioso en cuanto al rendimiento del sistema y de toda aplicación basada en cálculos de la misma índole. Con este precedente, se ha realizado efectivamente la paralelización de las operaciones sobre matrices que utiliza el sistema comprobando que los beneficios obtenidos coincidieron con los esperados.

Por otro lado, habiendo probado la conveniencia de paralelizar el cálculo de autovalores y autovectores, se probó el sistema de reconocimiento de rostros, incorporando en el proceso de aprendizaje los métodos por bloques con Blas y por bloques paralelo en reemplazo del clásico optimizado. Se utilizaron para ello matrices de 100x100 y de 400x400.

Para las matrices de 100x100 se obtuvieron los siguientes tiempos:

- Método clásico optimizado: 1,538 seg.
- Método por bloques con Blas: 3,046 seg.
- Método por bloques paralelo: 1,546 seg.

Para las matrices de 400x400 se obtuvieron los siguientes tiempos:

- Método clásico optimizado: 109,894 seg.
- Método por bloques con Blas: 105,139 seg.
- Método por bloques paralelo: 98,047 seg.

Tal como lo probado en la experimentación, con un tamaño de 100x100 no se logra ninguna mejora. Por otro lado, con un tamaño de 400x400 se observan mejoras en el rendimiento aunque no en las proporciones que se podrían prever, debido a la dependencia de la distribución de los valores de la matriz. Evidentemente la distribución de los valores de las matrices tratadas por la aplicación, no contribuyen a obtener grandes de mejoras de rendimiento con la utilización de paralelismo.

5. Conclusiones

Se probó la paralelización de la multiplicación de matrices con distintos tamaños de matrices y distintas cantidades de tareas. El rendimiento mejora en forma directamente proporcional a la cantidad de tareas utilizadas.

Se probaron distintos métodos de cálculo de autovalores y autovectores y se compararon con el método propuesto, método por bloques paralelo. El método propuesto aventaja considerablemente a los demás e insinúa que esta ventaja de acentúa con el crecimiento del tamaño de la matriz.

Con este trabajo se completó la versión paralela del sistema de reconocimiento de rostros quedando así actualmente operativa sobre la arquitectura mencionada.

Como trabajo futuro se está evaluando la migración del sistema de reconocimientos de rostros paralelo al modelo de memoria distribuida de un cluster de microprocesadores sobre el cual se dispone acceso exclusivo.

6. Referencias

- [1] Anderson E., Bai Z., Bischof C., Demmel J., Dongarra J., DuCroz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D. "LAPACK: A Portable Linear Algebra Library for High-Performance Computers". Proceedings of Supercomputing '90, pages 1-10, IEEE Press, 1990.
- [2] Bilmes J., Asanovic K., Chin C., Demmel J. "Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology". Proceedings of the International Conference on Supercomputing, Vienna, Austria, ACM SIGARC. July 1997
- [3] Choi J. "A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers". Proceedings of the High-Performance Computing on the Information Superhighway, IEEE, HPC-Asia'97. 1997.
- [4] Choi J., Dongarra J., Walker D. "PUMMA: Parallel Universal Matrix Multiplication Algorithm on Distributed Memory Concurrent Computers, in Concurrency: Practice and Experience". 6:543-570. 1994.
- [5] Correa Martín, Chichizola Franco. "Sistema de reconocimiento de rostros". Trabajo de grado. Facultad de Informática. 2001.
- [6] Dekel E., Nassimi D., Sahni S. "Parallel matrix and graph algorithms". SIAM Journal on Computing, 10:657-673. 1981.
- [7] Golub G., Van Loan C. "Matrix Computation" Second Edition. The John Hopkins University Press, Baltimore, Maryland. 1989.
- [8] Jun Zhang, Young Yan, and Martin Lades. "Face Recognition: Eigenface, Elastic Matching, and Neural Nets." Proceedings of the IEEE. vol. 85. No. 9, pp.1422-1435, 1997.
- [9] M. Turk and A. Pentland, "Face recognition using eigenfaces", in Proceedings of International Conference on Pattern Recognition , pp. 586-591,1991
- [10] Rutishauser H. "The Jacobi Method for Real Symmetric Matrices". Numer. Math. 9, 1-10. 1966,
- [11] www.cab.cnea.gov.ar/difusion/ClementinaIIINacion.html
- [12] www.setcip.gov.ar
- [13] www.sgi.com/origin/2000
- [14] www.sgi.com/software/irix6.5