Algoritmos no sistólicos para la multiplicación de matrices en FPGA's

Ignacio Bravo, Pedro Jiménez, Manuel Mazo, José Luis Lázaro, J. Javier de las Heras

Departamento de Electrónica. Universidad de Alcalá
Campus Universitario. Ctra. Madrid – Barcelona km 33.600. 28871 – Alcalá de Henares (Madrid)

http://www.depeca.uah.es ibravo@depeca.uah.es

Resumen. *Este trabajo evalúa diferentes algoritmos de multiplicación de matrices para FPGA's (Field Programmable Gate Array). Esta operación es muy habitual en numerosos algoritmos de procesado de señales e imágenes, por lo que su ámbito de actuación es muy diverso. Dentro de este análisis no se recogen los basados en arquitecturas sistólicas ya que éstas consumen un elevado número de multiplicadores, siendo inviable su empleo para aplicaciones con tamaño de matrices grandes.*

1. Introducción

La multiplicación de matrices (M.M.), es una operación cada vez es más habitual en numerosos algoritmos de diferentes áreas de la ingeniería. Por ejemplo, en el área de visión artificial cada imagen equivale a una matriz. Sin embargo, no sólo se centra su ámbito en esta área, sino que también en el procesamiento de señales también su uso es muy habitual.

Normalmente la M.M. lleva asociada un alto número de operaciones aritméticas, por lo que hasta hace poco era inviable su implementación en un dispositivo reconfigurable. Uno de los problemas principales, era la dificultad de implementar un multiplicador hardware. Sin embargo, hoy en día las nuevas familias de FPGA's poseen hasta 400 multiplicadores hw [1] [2]. Con este número de recursos, es factible la implementación de una arquitectura sistólica para la implementación de M.M. [3].

En la Figura 1 se muestra una arquitectura sistólica clásica de M.M., donde se dispone de un interface para gestionar los datos de una matriz de entrada *A* (interface filas) y de una matriz de entrada *B* (interface columnas). En este tipo de algoritmos el número de multiplicadores suele coincidir con el tamaño de la matriz a manejar, por lo que si las matrices a tratar poseen tamaños superiores al número de multiplicadores, se hace inviable su implementación. En el caso de M.M. en visión artificial, el tamaño habitual de imágenes manejados actualmente (256x256, 512x512 ó 1024x1024 píxeles) hace inviable el empleo de arquitecturas sistólicas. Esta causa justifica la búsqueda de alternativas para la M.M. en FPGA's.

Por tanto, a la hora de manejar tamaños de matrices grandes, es imposible implementar algoritmos que conformen una arquitectura puramente sistólica y por consiguiente una ejecución totalmente en paralelo [4]. Por ello, el objetivo será encontrar un algoritmo de M.M., donde el número de recursos internos sea suficiente y donde la ejecución de dicho algoritmo explote al máximo las posibilidades de concurrencia en las FPGA's.

El resto del artículo se divide de la siguiente forma: A continuación se describen las características de la M.M. para su paralelización. El siguiente apartado describe el algoritmo clásico de M.M., para a continuación evaluar el algoritmo de Winograd, Strassen y particionado de matrices. El artículo finaliza con un apartado de resultados obtenidos en diferentes pruebas así como las conclusiones derivadas de este trabajo.

2. Paralelización de los algoritmos de multiplicación.

La M.M. tiene características muy específicas en lo que se refiere al diseño e implementación de un algoritmo paralelo:

- Cada elemento $c_{i,j}$ de la matriz resultado (C), en principio es independiente de todos los demás elementos. Esto es sumamente útil dado que permite un amplio grado de

flexibilidad en lo referente a la paralelización.

- La cantidad y el tipo de operaciones a realizar, es independiente de los datos mismos.
- Regularidad de la organización de los datos y de las operaciones que se realizan sobre los datos: los datos están organizados en estructuras bidimensionales (las matrices mismas) y las operaciones son básicas (multiplicación y suma).

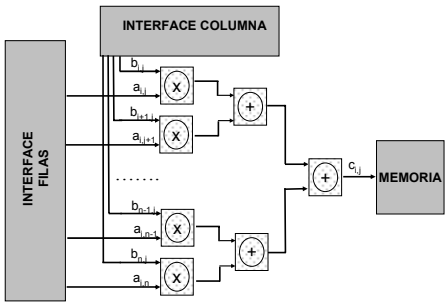

Figura 1. Sistema sistólico de M.M..

La primera característica hace que la M.M. sea especialmente apropiada para sistemas multiprocesadores, donde un conjunto de elementos de procesamiento, comparten una misma memoria. Los algoritmos paralelos para multiprocesadores suelen seguir las ideas básicas de descomposición o división de los datos a calcular.

Las últimas dos características hacen que los algoritmos propuestos para la M.M. en paralelo sigan el modelo de cómputo paralelo SPMD (*Single Program Multiple Data*), en el que un mismo programa se ejecuta en cada procesador de manera independiente y eventualmente se sincroniza y/o comunica con los demás procesadores.

En general, los algoritmos se pueden adaptar de una manera más o menos compleja a cada una de las arquitecturas de procesamiento paralelo disponibles. Por tanto, en función de la plataforma hardware donde se implemente el sistema de M.M., será necesario realizar un algoritmo diferente.

Existen numerosas alternativas para explotar las características anteriores (ver Tabla 1). En este trabajo sólo nos centraremos en tres de ellas: Winograd, Strassen y particionado de matrices.

Tabla 1. Resumen algoritmos M.M. en paralelo.

Método de multiplicación de matrices	Características más importantes
Particionado directo	Varias unidades de procesado, donde cada unidad calcula una parte de la matriz resultado. Problemas con los accesos a memoria compartida.
Divide & Conquer recursivo	Multiplicación mediante submatrices. Llamadas recursivas para la obtención de resultados. Aumento del tamaño de memoria requerido.
Arrays sistólicos	Array de procesadores de dos dimensiones. Distribución simple de los cálculos dentro del array. Requerimientos de memoria iguales para todos los procesadores.
Cannon	Arquitectura sistólica realimentada. Distribución inicial de elementos de entrada. Minimización de los tiempos de ejecución.
Fox	Similar al de Cannon. Mecanismos de broadcast por filas. Mayor requerimiento de memoria

3. Algoritmo clásico de multiplicación de matrices

A la hora de multiplicar dos matrices iniciales *A* y *B*, de tamaños *mxl* y *lxn* respectivamente (ver (1)), una forma muy sencilla de obtener el producto de ambas es acorde a la expresión (2), denominando a este método como clásico.

$$C_{m\times n} = A_{m\times l} \times B_{l\times n} \quad (1)$$

$$c_{i,j} = \sum_{k=1}^{l} a_{i,k} \cdot b_{k,j} \quad \begin{matrix} i = 1...n \\ j = 1...l \end{matrix} \quad (2)$$

La arquitectura hw asociada es bastante elemental, presentándose en la Figura 2 una posible solución. En ésta se observa como con un único multiplicador y un acumulador se consigue implementar la expresión (2). Obviamente el empleo de un número reducido de recursos conlleva un incremento en el tiempo de ejecución, presentándose en (4) el tiempo total de ejecución de esta alternativa (T_{EXE}).

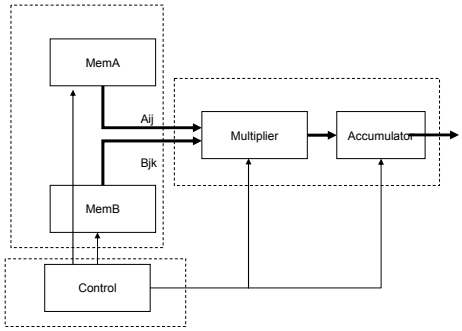

Figura 2. Diagrama de bloques del sistema de multiplicación clásico de matrices.

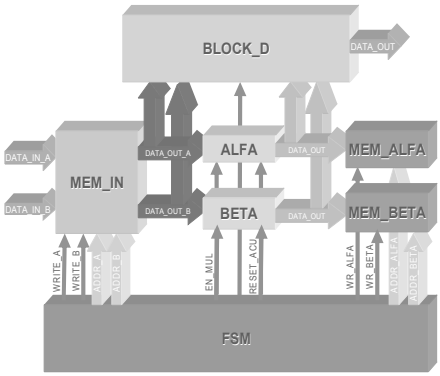

Figura 3. Diagrama de bloques del sistema de M.M. mediante el algoritmo de Winograd.

$$T_E = l \cdot T_{MULT} + (l-1) \cdot T_{ACC} \quad (3)$$

$$T_{EXE} = (n \times p) \cdot T_E \quad (4)$$

donde n, l y p son los rangos de las matrices A y B (ver (1)), T_E es el tiempo de ejecución de un elemento de la matriz de salida, T_{MULT} el tiempo de una multiplicación y T_{ACC} el tiempo que tarda en ejecutarse la acumulación.

Otra forma habitual de evaluar un algoritmo, es en función del número de operaciones aritméticas realizadas (véase (5)).

$$n_{OPS} = m \times n \times (2r-1) \quad (5)$$

Particularizando para el caso de matrices cuadradas (que será la situación a evaluar en el resto de algoritmos) de orden n, la expresión (5) se transforma en (6).

$$n_{OPS} = 2n^3 - n^2 \quad (6)$$

Este número de operaciones usualmente se denomina como *complejidad* de la M.M. y determina el tiempo de ejecución necesario para ser resuelto. En este contexto, también lo usual es encontrar que la M.M. sea $O(n^3)$, enfatizando el término dominante en la ecuación anterior.

4. Algoritmo de Winograd

El método clásico de M.M. posee como principal problema el elevado número de operaciones aritméticas a realizar. Winograd [6] propuso que el cálculo de los elementos de la matriz producto final ($c_{i,j}$), se realizase en base a (7).

$$c_{ij} = d_{ij} - \alpha_i - \beta_j \qquad i,j = 1...n \quad (7)$$

$$d_{ij} = \sum_{k=1}^{n/2} (a_{i,2k} + b_{2k-1,j}) \cdot (a_{i,2k-1} + b_{2k,j})$$

$$\alpha_i = \sum_{k=1}^{n/2} (a_{i,2k} \cdot a_{i,2k-1}) \quad \beta_j = \sum_{k=1}^{n/2} (b_{2k,j} \cdot b_{2k-1,j})$$

La principal ventaja de este algoritmo frente al clásico, es que el cálculo α y β sólo se realiza una vez. Esto reduce a la mitad el número de multiplicaciones requeridas, en comparación con las utilizadas en el método clásico.

El gran inconveniente de este algoritmo, es su fuerte estructura secuencial, siendo muy complicado conseguir algún tipo de paralelismo ya que es necesario esperar a tener todos los elementos de las matrices de entrada, para empezar a procesarlos. Además, el cálculo de d sólo se puede realizar si se tienen calculados los α y β correspondientes. A su vez, los elementos de C no pueden ser calculados hasta que no se tengan los elementos d.

La única fase donde se ha optimizado el procesado y por tanto alcanzando un grado de paralelismo pleno, ha sido en la del cálculo de α y β.

El esquema general del sistema completo se ha dividido en varios módulos funcionales tal y como se puede apreciar en la Figura 3. Por otra parte, los tiempos de ejecución de este algoritmo, se muestran en (8).

$$T_{EXE} = T_{MEM} + L_{MEM\alpha\beta} + T_{\alpha\beta} + L_{\alpha\beta\_dc} + T_{dC} \quad (8)$$

$$T_{MEM} = n^2 \cdot T_{CLK}; T_{\alpha\beta} = 8n \cdot T_{CLK}; T_{dC} = n^3 \cdot T_{CLK}$$

$$L_{MEM\alpha\beta} = 4 \cdot T_{CLK} \quad L_{\alpha\beta\_dc} = 8 \cdot T_{CLK}$$

donde T_{EXE} es el tiempo total, T_{MEM} es el tiempo de escritura de los datos de entrada en la memoria, $L_{MEM\alpha\beta}$ es una latencia para sincronizar

la escritura de datos en la memoria de entrada y el cálculo de α, β, $T_{\alpha\beta}$ es el tiempo consumido por los bloques α y β, $L_{\alpha\beta\_dC}$ es la latencia entre la generación de los coeficientes α y β y la siguiente etapa y finalmente T_{dC} es el tiempo de ejecución de los bloques d y C.

5. Algoritmo de Strassen

Este algoritmo propuesto por Strassen en 1969 [7], basa todo su funcionamiento en la descomposición de una matriz de *nxn* elementos, en submatrices de 2x2 elementos. Así, mediante este algoritmo se reduce el número de multiplicaciones a $n^{2.81}$.

Para explicar su funcionamiento, partamos de dos matrices iniciales de 2x2 elementos (*A, B*), siendo *C* la matriz resultante (ver (9)). Los resultados parciales de multiplicar cada una de estas expresiones (S_p), se pueden expresar en función de los elementos de *A* y *B* (ver (10)).

$$C = A \cdot B \tag{9}$$

$$\begin{pmatrix} c_{11}\ c_{12} \\ c_{21}\ c_{22} \end{pmatrix} = \begin{pmatrix} a_{11}\ a_{12} \\ a_{21}\ a_{21} \end{pmatrix} \cdot \begin{pmatrix} b_{11}\ b_{12} \\ b_{21}\ b_{21} \end{pmatrix}$$

$$\begin{aligned} &S_1 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22}) \\ &S_2 = (a_{21} + a_{22}) \cdot b_{11};\ \ S_3 = a_{11} \cdot (b_{12} - b_{22}) \\ &S_4 = a_{22} \cdot (b_{21} - b_{11});\ \ S_5 = (a_{11} + a_{12}) \cdot b_{22} \\ &S_6 = (a_{21} - a_{11}) \cdot (b_{11} + b_{12}) \\ &S_7 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22}) \end{aligned} \tag{10}$$

Así, cada uno de los elementos que forman la matriz C se puede expresar como:

$$\begin{aligned} &c_{11} = S_1 + S_4 - S_5 + S_7 \quad c_{12} = S_3 + S_5 \\ &c_{21} = S_2 + S_4 \qquad\quad c_{22} = S_1 - S_2 + S_3 + S_6 \end{aligned} \tag{11}$$

A la hora de manejar matrices mayores de 2x2 existen básicamente dos variantes:

- Bottom-up: Para multiplicar dos matrices de orden *n* ($n=2^p$ donde $p>1$), se dividen éstas en bloques de tamaño 2^q ($q<p$). Así, se obtiene una matriz de tamaño *mxm*, donde $m=2^{p-q}$. En la multiplicación de los bloques de *mxm* (algoritmo externo), se utiliza el método clásico, mientras que en la multiplicación de las matrices parciales de $2^q x 2^q$ (algoritmo interno), se emplea el de Strassen.
- Top-down: Esta propuesta utiliza el método de Strassen como algoritmo externo y el clásico como interno. En [8] y [9] se propone una alternativa en donde el método clásico es sustituido por el algoritmo de Winograd, alcanzando un número de multiplicaciones igual a $0.437n^3$.

La alternativa Bottom-up es inviable dentro de una FPGA, ya que el número de recursos internos a utilizar es bastante elevado por lo que se empleará la segunda alternativa. En [9] se propone una variante de ésta (ver (12)), que permite generar coeficientes de salida sin necesidad de leer completamente las matrices de entrada. Para ello, los términos entre paréntesis de (10) son sustituidos por las variables α y β cuyo valor se muestra en (13).

$$\begin{aligned} &S^1{}_{ij} = \sum_{k=1}^{m} \alpha^1_{ik} \cdot \beta^1_{kj} \qquad S^2{}_{ij} = \sum_{k=1}^{m} \alpha^2_{ik} \cdot b_{2k-1,2j-1} \\ &S^3{}_{ij} = \sum_{k=1}^{m} a_{2i-1,2k-1} \beta^2_{kj} \qquad S^4{}_{ij} = \sum_{k=1}^{m} a_{2i,2k} \beta^3_{kj} \\ &S^5{}_{ij} = \sum_{k=1}^{m} \alpha^3_{ik} \cdot b_{2k,2j} \qquad S^6{}_{ij} = \sum_{k=1}^{m} \alpha^4_{ik} \cdot \beta^4_{kj} \\ &S^7{}_{ij} = \sum_{k=1}^{m} \alpha^5_{ik} \cdot \beta^5_{kj} \end{aligned} \tag{12}$$

$$\begin{aligned} \alpha^1_{ik} &= (a_{2i-1,2k-1} + a_{2i,2k}) \\ \alpha^2_{ik} &= (a_{2i,2k-1} + a_{2i,2k}) \\ \alpha^3_{ik} &= (a_{2i-1,2k-1} + a_{2i-1,2k}) \\ \alpha^4_{ik} &= (a_{2i,2k-1} - a_{2i-1,2k-1}) \\ \alpha^5_{ik} &= (a_{2i-1,2k} - a_{2i,2k}) \\ \beta^1_{kj} &= (b_{2k-1,2j-1} + b_{2k,2j}) \\ \beta^2_{kj} &= (b_{2k-1,2j} - b_{2k,2j}) \\ \beta^3_{kj} &= (b_{2k,2j-1} - b_{2k-1,2j-1}) \\ \beta^4_{kj} &= (b_{2k-1,2j-1} + b_{2k-1,2j}) \\ \beta^5_{kj} &= (b_{2k,2j-1} - b_{2k,2j}) \end{aligned} \tag{13}$$

$$T_{EXE} = n^2 \cdot (n \cdot T_{CLK} + (n-1) \cdot T_{CLK}) \tag{14}$$

La Figura 4 muestra la propuesta de este algoritmo, cuyo diseño se ha realizado en VHDL. En ella se pueden identificar cada uno de los bloques que conforman el algoritmo, así como una memoria de entrada FIFO de doble puerto. Una máquina de estados (FSM), se encargará de sincronizar todo el sistema. La implementación del sistema de la Figura 4, tarda en ejecutarse la expresión mostrada en (14).

6. Algoritmo de particionado de matrices.

Derivado de la propuesta de Strassen, otra línea empleada para multiplicar matrices es la basada en la descomposición de las matrices iniciales en submatrices de 2x2 elementos. La principal cualidad de este sistema es el empleo de un bloque multiplicador de matrices de tamaño 2x2. En [10] se presenta un sistema capaz de realizar dicha alternativa pero aplicada al producto de una matriz por un vector.

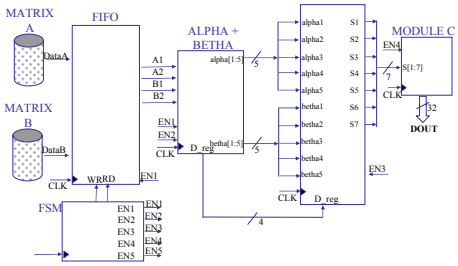

Figura 4. Diagrama de bloques del algoritmo de Strassen.

La propuesta realizada se basa en el siguiente funcionamiento: denominaremos como *C* a la matriz de salida cuyos valores a su vez pueden ser agrupados en submatrices de 2x2 elementos (ver (15)). Cada una de las submatrices que forman *C* se obtendrán mediante el producto y suma de submatrices de *A* ($A_{i,k}$) y *B* ($B_{k,j}$) (ver (16)).

$$C = \begin{pmatrix} c_{1,1}c_{1,2}...c_{1,n} \\ c_{2,1}c_{2,2}...c_{2,n} \\ ..... \\ c_{n,1}c_{n,2}...c_{1,n} \end{pmatrix} = \begin{pmatrix} C_{1,1}C_{1,2}.....C_{1,\frac{n}{2}} \\ C_{2,1}C_{2,2}.....C_{2,\frac{n}{2}} \\ ..... \\ C_{\frac{n}{2},1}C_{\frac{n}{2},2}...C_{\frac{n}{2},\frac{n}{2}} \end{pmatrix} \quad (15)$$

$$C_{i,j} = \begin{pmatrix} c_{(2\cdot i-1),(2\cdot j-1)}c_{(2\cdot i-1),2\cdot j} \\ c_{2\cdot i,(2\cdot j-1)}c_{2\cdot i,2\cdot j} \end{pmatrix} = \sum_{k=1}^{\frac{n}{2}} A_{ik} \times B_{kj} \quad i,j = 1....\frac{n}{2} \quad (16)$$

$$T_{EXE} = (T_{MEM} + T_{MULT} + T_{ADD} + 1) \cdot \frac{n^2}{4} \cdot T_{CLK} \quad (17)$$

En base a esto se ha diseñado un sistema que realiza la multiplicación de submatrices de 2x2 elementos, cuya estructura interna se muestra en la Figura 5. Básicamente, está formada por 4 multiplicadores hw y 4 unidades de suma y acumulación. El empleo de multiplexores permite reducir de 8 a 4 el número de multiplicadores.

Para la optimización del tiempo de ejecución de esta alternativa, se ha segmentado su funcionamiento en un *pipeline* de 3 etapas (multiplicación, acceso a memoria y suma). Así, el tiempo de ejecución total (T_{EXE}) se muestra en (17), donde T_{MULT} es el tiempo consumido en la multiplicación, T_{ADD} el tiempo de suma y T_{MEM} el tiempo de acceso a memoria.

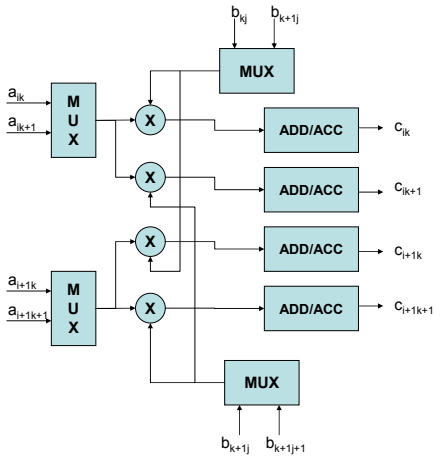

Figura 5. Diagrama de bloques de un multiplicador de submatrices de 2x2 elementos.

Una ventaja de esta alternativa es la implementación de varias unidades permitiendo así la generación de varios datos de salida simultáneos. Otra alternativa eficaz a partir del circuito de la Figura 5 sería la construcción de unidades M.M. mayores, por ejemplo 4x4.

7. Resultados

Las tres alternativas han sido implementadas en VHDL sobre una FPGA 2V500fg256-5 y se han simulado temporalmente. En todos los casos las matrices que se han probado son matrices cuadradas y se han realizado múltiples pruebas con diferentes tamaños, empleando en todos los casos una frecuencia de reloj de la FPGA de 50 MHz. Se han simulado los tres algoritmos para matrices de entrada de 8x8, 32x32, 64x64, 128x128, 256x256 y 512x512, con datos de 16 bits. El resultado temporal se puede observar en la Figura 6 y 7. Se verifica que la alternativa de

división o partición de las matrices es la más rápida y la que mejores resultados presenta desde el punto de vista de recursos consumidos. A medida que el tamaño de las matrices de entrada aumenta, este algoritmo acentúa sus resultados. El tiempo empleado por el algoritmo de Strassen y el de particionado, es el mismo, sin embargo el número de recursos consumidos es casi el doble.

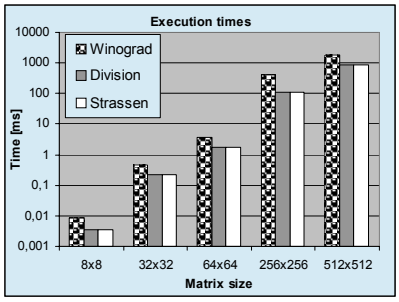

Figura 6. Tiempo de ejecución de la FPGA para los algoritmos evaluados, con diferentes tamaños de matrices.

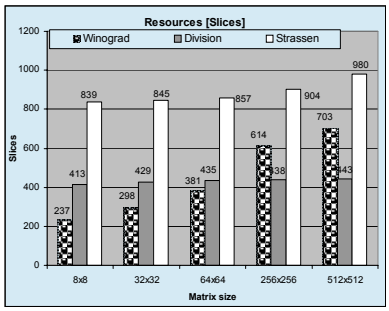

Figura 7. Recursos internos de la FPGA consumidos por los algoritmos evaluados, para diferentes tamaños de matrices.

8. Conclusiones

Este trabajo ha evaluado tres algoritmos de M.M., donde se ha intentado implementar arquitecturas que funcionen en paralelo. Sin embargo, sólo en el algoritmo de Strassen y particionado de matrices se han alcanzado altos niveles de concurrencia. El algoritmo de división de matrices es el que mejores resultados ofrece, tanto por el tiempo de ejecución como por recursos consumidos. Por otro lado, los tiempos de ejecución en los algoritmos evaluados son superiores a los que emplearían arquitecturas sistólicas. Sin embargo, este tipo de arquitecturas no pueden implementarse en las FPGA's actuales cuando el tamaño de las matrices de entrada son superiores a 20x20, debido al número de multiplicadores internos. Así, si se quiere manejar en FPGA's matrices superiores, hay que usar otros algoritmos, no tan concurrentes como los sistólicos. Esta ha sido la razón por la que en este trabajo se han evaluado diferentes algoritmos de multiplicación no sistólicos.

Agradecimientos

Este trabajo ha sido posible gracias al proyecto "Sistema de localización y posicionamiento absoluto de robots. Desarrollo de un espacio inteligente" del Ministerio de Ciencia y Tecnología (ref: DPI2003-05067).

Referencias

[1] Implementing Multipliers in FPGA Devices http://www.altera.com/products/devices/cyclone2/features/multipliers/cy2-multipliers.html

[2] Designing High-Performance Memories and Multipliers. Xcell Journal Q3-2001.

[3] Jagadish, H.V., Kailath, T.K., "A family of new efficient arrays for Matrix", IEEE Trans. on Comput., vol. 38, No. 1, pp 149-155. 1989

[4] Choi, S., Prassana, V.K., "Time and Energy Efficient Matrix Factorization using FPGA's". Proc. Field Programmable Logic and Applications (FPL), pp. 225-234, 2003.

[5] Modi J.J., "Parallel Algorithms and Matrix Computation", Claredon Press Oxford. 1998.

[6] Winograd, S. "A new algorithm for inner product". IEEE Trans. Comput., C-18, 693-694. 1968.

[7] Strassen V. "Gaussian elimination is not optimal". Num. Math. Vol. 13, pp.354-356, 1969

[8] Elfimova L.D., Kapitonova Y. V. "A fast algorithm for matrix multiplication and its efficient realization on systolic arrays". Cybernetics and Systems Analysis, Vol. 37, pp. 109-121, 2001.

[9] Jelfimova L., "A new fast parallel version of Strassen Matrix Multiplicaction Algorithm and its efficient implementation on Systolic-Type Arrays". Proc. of 3rd Int. Conference on Massively Parallel Computing Systems 1998.

[10] Bensaali F., Amira A., Bouridane A. "An FPGA Based Coprocessor for Large Matrix Product Implementation". IEEE Int. Conf. on Field-Programmable Tech. (FPT'03), pp. 292-295, 2003.