Project 3: Implementing Systolic Arrays using Verilog

University of Central Florida
School of Electrical Engineering and Computer Science
CDA 4150 Computer Architecture
Fall 2005 (Due 12/1/05)

An alternative to solve the matrix vector product in parallel are systolic arrays. The name systolic array was proposed by Kung and Leiserson to a network of processing elements that act synchronously to solve specific problems. These networks of processing elements exploit pipelining, parallel processing, and use simple and regular communication paths. At each computation step the processing elements of the systolic array get data, either from another processing element(s) or from outside the network, execute some computation, and pump data out either outside the network or to another processing element(s). The name systolic was proposed by analogy with the way the heart pumps blood through the circulatory system and the manner systolic array pumps data in and out the processing elements.

The systolic array proposed by Kung and Leiserson, in the late seventies, to compute the band matrix vector product is the one we will used through this work and we will refer to it as Kung's Systolic Array (KSA). The basic processing element of KSA is the Inner-Product-Step-Processor (IPSP), which is depicted in Figure 1. An IPSP has three inputs ports and three output ports and is defined as follows:

$IPSP(a, x, y) \rightarrow (a, x, y + a \times x)$

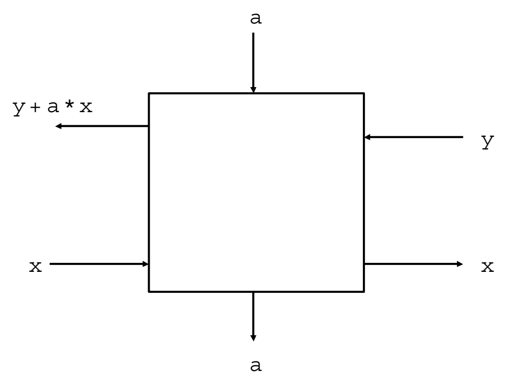

Figure 1: data flow in a systolic processing element.

We will explain how a KSA with three processing elements computes the band matrix vector product

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{11} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{144} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

using the following recurrences:

$$\begin{cases} y_i^1 = 0, \\ y_i^{k+1} = y_i^k + a_{ik} \times x_k, \\ y_i = y_i^{n+1} \end{cases}$$

As in KSA the matrix must enter the systolic array by diagonals the number of processing elements required is equal to the number of diagonals $w = 3$ of matrix A. In Figure 2, we show how the components of the vector x enter the systolic array from left to right, the components of the vector y, initially zero, enter the systolic array from right to left, and the coefficients of the matrix A will enter the systolic array, by diagonals, from top to bottom. At each step of the computation three values enter in each IPSP, a computation is executed, each y_i accumulates its partial result, and three values are pumped out. This computation requires w steps to move the first component y_1 to the leftmost processing element and then, as the components of vector y enter the array every other unit of time, $2n - 1$ additional steps are necessary to pump out all the elements of the resulting vector y for an overall of $T(n) = w + 2n - 1$. The first five steps of the of the computation are shown in Figure 3, where it can be observed that each processing element of the systolic array is working on the matrix vector product on every other unit of time.

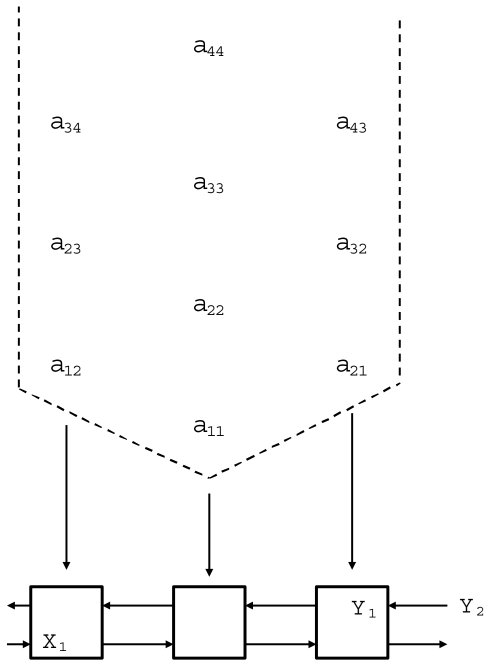

Figure 2: Kung's Systolic array to compute a band matrix vector product.

Surprisingly, the same linear systolic array computes two matrix vector products simultaneously in $T(n) = w + 2n$ steps using perfect shuffling as a spatial data scheduling technique. Therefore,

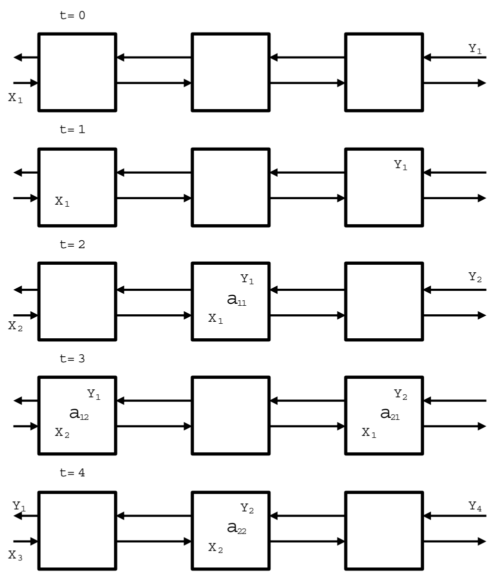

Figure 3: First five steps of the computation.

implementing two matrix vector product multiplication, $Ax||Bz$, on Kung's systolic array is straight forward. To carry out these matrix vector products the elements of x and z and the elements of the matrices A and B must be arranged as illustrated in Figure 4. In this arrangement the components of x and z are merged, using perfect shuffling, into a single array that we will refer to as the carrier vector. The components of x are stored in the carrier vector in the odd locations and the components of z in the even ones. similarly, each diagonal of matrix A is also merged with the corresponding diagonal of matrix B using perfect shuffling.

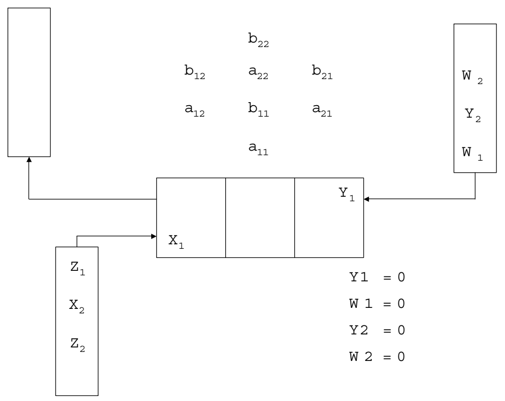

Figure 4: Systolic array to compute the mixed tensor product.

We can simulate a classical bit using a state vectors; for instance, the values 0 and 1 of a classical bit can be realized by a pair of mutually orthonormal state vectors:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Where the symbols $|0\rangle$ and $|1\rangle$, are known as ket zero and ket one according to Dirac's notation.

To move a state vector from one state to another we must use unitary transformations which are represented by 2×2 matrices and we will refer to them as unitary operations or gates. As an example, we show the unitary operations Identity(I), NOT(X) and Hadamard(H): The identity gate just left the state vector as is:

$$\mathrm{I}|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

$$\mathrm{I}|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

the X gate flips $|0\rangle$ into $|1\rangle$ and vise versa:

$$\mathrm{X}|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$\mathrm{X}|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

and the Hadamard (H) gate turns $|0\rangle$ and $|1\rangle$ into a equally weighted superposition state. The H gate is defined as

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and when it is applied to $|0\rangle$ and $|1\rangle$ we obtain the following superposition:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

You have to implement, in Verilog, the following operations in parallel using a systolic array with three processing elements and applying the technique used in the homework:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$I|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

The input to systolic arrays is a 1-Dimensional vector denoted the carrier vector. As the I operation just left the second vector as is, we can use the value "D". For instance.

$$|0\rangle = \begin{bmatrix} 1 \\ D \\ 0 \\ D \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ D \\ 1 \\ D \end{bmatrix}$$

Where "D" stands for a don't care value. The 2 vectors once loaded in the carrier vector can have four possibilities. In Figure 5 we use "D" in the second input vector:

$$|00\rangle => \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle => \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$|10\rangle => \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad |11\rangle => \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

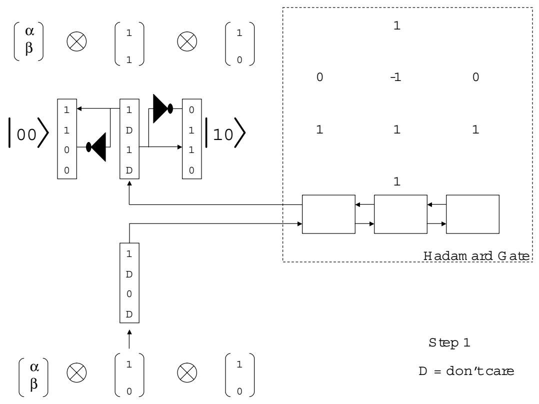

Figure 5: First step of the systolic teleportation design .

The output of the first systolic array (input for the next systolic array) has to be multiplied for two matrices in parallel according to the following rules (see Figure 6):

1. If the value of the first component of the carrier vector is 0 use $I||X$

2.- If the value of the first component of the carrier vector is 1 use $I||I$

You must deliver in a CD:

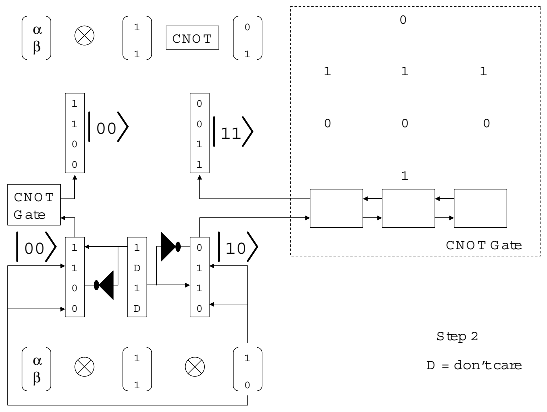

Figure 6: Second step of the systolic teleportation design .

1.- A report indicating the theoretical time and the implementation time for the whole computation.

2.- The value of the final vector(for each of the four cases)

3.- all your files (implementation in Verilog).