International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 7, July - 2014

Evolutionary Program Based Synthesis of Systolic Array

P. Mohamed Faizulla[1],
[1]Student Dept.of VLSI Design and Embedded Systems,
VTU Extension Centre @United Technologies,
Bangalore- 560 022, India.

Dr. V. Venkateswarlu[2]
[2]H.O.D,Dept.of VLSI Design and Embedded Systems,
VTU Extension Centre @United Technologies,
Bangalore-560 022,India.

Abstract: **This paper gives design of "Evolutionary program based synthesis of systolic architecture for 3-Tap FIR filter, 2*2 Matrix-Matrix multiplication, convolution and correlation of two discrete sequences". Results indicate that evolutionary approach is the best search method that can be adopted on systolic architectures as it was inferred from literature survey. In most general terms, evolution can be described as a two–step iterative process, consisting of random variation followed by selection. In this project strategy of *Gaussian perturbation* having ten competitions, thousand trails and varying populations is choosen as the designs are relatively simple. For this each design is represented in Dependency Graph (DG) from design algorithm (for example FIR equation). This DG is further transformed to Space –Time representation after the application of linear mapping technique. The transformation is done based on three vectors viz., *Projection Vector, Processor Vector and scheduling Vector*. The design concept is at system level; hence the terminology used in the project to some extent requires knowledge of advances in computer architecture and also knowledge of multiple disciplines. Only four typical designs are considered; further it can be extended to other design as well in specific to DSP designs. Each MATLAB program written is verified manually for all possible test cases. Results are reported with errors encountered. In future it is expected that evolutionary based concepts are the real contributors for next generation architecture level design.**

Index terms:- Evolutionary Computation, Systolic Architecture, Dependency Graph (DG), Signal Flow Graph (SFG), Projection Vector(d), Processor Vector (P), Scheduling vector (S) , Reduce Dependency Graph (RDG), Regular Iterative Algorithm (RIA).

I. INTRODUCTION

Until recently, computation-intensive tasks were handled by high performance supercomputers, including pipelined computers, array processor and multiprocessor systems. The development of these computer systems has involved a through exploration of parallel computing, efficient programming, and resource optimization. However, the general-purpose nature of these machines has led to complicated system organization and severe system overheads.

A new approach to design (in particular Optimization) called as ―Evolution Computation‖ is just at the budding state and promising the full potential to deal with complicated system organization. Evolution is an optimizing process that can be simulated using the computer or other devices and put to good engineering purpose. Methods of Evolution computing include Evolution Programming, Evolution strategy and Genetic Algorithms. At present the most promising architecture design approach seems to be systolic architecture design and perhaps will continue to be in future. A systolic system is a network of processors which rhythmically compute and pass data through the system. A number of systolic architectures can be designed for any given regular iterative algorithm using linear mapping or projection techniques. The present work under taken is Evolutionary Program based synthesis of Systolic architecture design which is expected to boost the future initiatives based on Evolutionary Computation.

A. Evolutionary Computation

Evolutionary computation has started to receive significant attention during the last decade, although the origins can be traced back to the late 1950's.The most significant advantage of using evolutionary search lies in the gain of flexibility and adaptability to the task at hand, in combination with robust performance (although this depends on the problem class) & global search characteristics. In fact, evolutionary computation should be understood as a general adaptable concept for problem solving, especially well suited for solving

difficult optimization problems, rather than a collection of related and ready-to-use algorithms. [11]

Evolutionary programming was introduced by Fogel. The approach was to evolve finite state machine (FSM) to predict events on the basis of former observations. An FSM is an abstract machine which transforms a sequence of input symbols into a sequence of output symbols. The transformation depends on a finite set of states and a finite set of state transition rules [1]. Evolutionary algorithms generally operated directly on the real values to be optimized, in contrast with genetic algorithms which usually operate on a separately coded transformation of the objective variables for more information refer [1] [2][10][11][13].

B.High-Level Syntheis

High-level synthesis is concerned with the design and implementation of circuits from a behavioral description subject to a set of goals and constraints. Two main tasks performed in high level synthesis, scheduling and mapping. Scheduling assigns operations to clock cycles and resources allocation or mapping is concerned with assigning operations and variables to hardware. During allocation, registers are allocated to store variables, operations assigned to functional units, and connections which are multiplexers, busses, or a combination of both, are used for interconnection. The behavior of a circuit can be specified using a high-level hardware description language, and it should be translated into a suitable intermediate format, e.g. a flow graph [3].

C. Systolic Processors

Systolic processors [5] are a new class of pipelined array architectures [6]. ***" A systolic system is a network of processors which rhythmically compute and pass data through the system".***

The basic principle of a systolic architecture, a systolic array in particular is illustrated in Figure1a & 1b,which is obtained by replacing a single processing element with an array of PEs or cells. A higher computation throughput can be achieved without increasing memory bandwidth. The function of the memory in the diagram is analogous to that of heart; it ―pulses‖ data (instead of blood) through that array of cells. The crux of this approach is to ensure that once a data item is brought out from the memory it can be used effectively at each cell it passes while being ―pumped‖ from cell to cell along the array. This is wide class of compute-bound computations where multiple operations are performed on each data item in a repetitive manner.

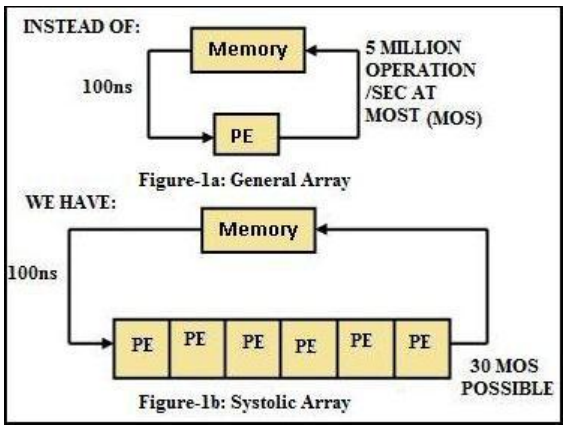

Figure-1a: General Array

Figure-1b: Systolic Array

II.DESIGN OF SYSTOLIC ARCHITECTURE BASED ON EVOLUTIONARY PROGRAM

Algorithm to Hardware Mapping

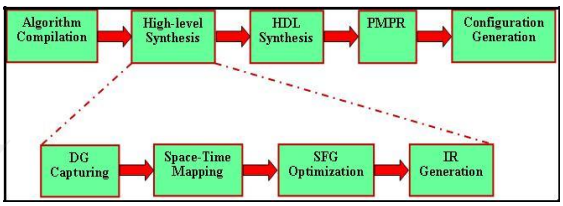

Figure-2: Algorithm to Hardware Mapping.

.

Algorithm Compilation: converts algorithm to graph-based representation.
High-Level Synthesis: converts graph to data path (space) and control (time) to HDL.
HDL (Hardware Description Language) Synthesis: convert behavioral or RTL HDL (VHDL, Verilog) to netlist of gates and flip/flops.
Partitioning, Mapping, Placing, Routing: A netlist to a given architecture Partitioning is required if the design doesn't fit.
Configuration Generation: Equivalent to code generation of compiler.

Dependence Graph Capturing: - *n*-D Dependence Graph (DG) is obtained from a compiler or user. **Space-Time Mapping:**-*n*-D is mapped to *k*-D signal flow graph (SFG) and (*n-k*)-D time schedule, 1<= *k* < *n*. **SFG Optimization:** -*k*-D SFG is optimized mostly by human designer.

IR Generation: -Intermediate Representation (IR) e.g. VHDL, Verilog HDL.

Vertically Integrated VLSI System Design

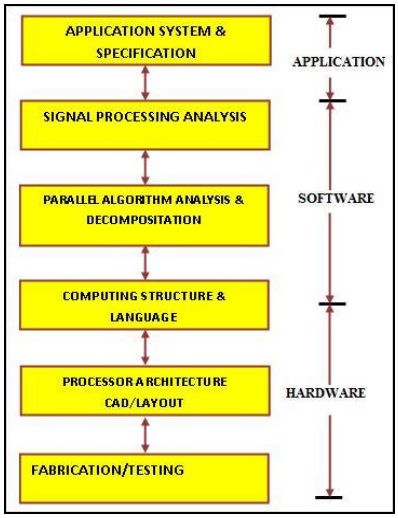

Figure-3: Top-down design integration

The array processors design involves a very broad spectrum of disciplines, including algorithm analysis, parallelism extractions, array architectures, programming techniques, functional primitives, structural primitives and numerical performance of DSP algorithms [4] [7].

A. General Design Flow of Array Processors

A dependence graph (DG) provides a useful first step toward a stationary answer. In deriving an array processor, a series of intermediate design levels are involved. They are creating a (1) DG design,(2)mapping the DG onto a signal flow graph (SFG)array and (3) deriving a systolic array form the SFG [7].

Stage 1: **DG Design** for a given problem, a suitable algorithm described in terms of a certain convenient expression is identified. A recursive algorithm may be easily transformed to a DG by tracing the associated space-time index space and using proper arcs to display the dependencies in the index space.

Stage 2: **SFG Design** The expression primarily consists of processing nodes, communication edges, and delays. A simple (although not the only)way of mapping a DG onto an SFG array is by means of projection, which assigns the operations of all nodes along a line to a single PE. For example, the three-dimensional index space of a DG may be projected onto a two – dimensional SFG array.

Stage 3: **Array Processor Design** The SFG obtained in stage 2 can then be mapped to an SIMD, systolic array, wave front array, or even an MIMD machine. For example, to convert an SFG array into a systolic array, a cut-set based systolization (retiming) procedure may be adopted.

B. Systolic Array Design Methodology

Systolic architecture is designed by using linear mapping techniques on regular dependency graph [7] [8]. The edge in dependency graphs represents precedence constraints. A dependency graph (DG) is said to be regular if the presence of edge in a certain direction at any node in the DG represents presence of edge in the same direction at all nodes in the DG.

Definitions:

Projection vector (also called iteration vector) $\mathbf{d}=\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$

Processor space vector, $\boldsymbol{p}^{T} = (p1\ p2)$ .Any node with index I^T= (i, j) would be executed by processor;

$$p^T I = (p_1\ p_2)\begin{pmatrix} i \\ j \end{pmatrix}$$

Scheduling vector, s^T= (s1 s2). Any node with index I would be executed at time, $s^T I$.

Hardware Utilization Efficiency, **HUE** = $\frac{1}{|S^T d|}$..

This is because two tasks executed by the same processor are spaced $|S^T d|$ time units apart.

Feasible constraints

A number of systolic architectures can be designed for a given problem by selecting different projection, processor space and scheduling vectors. These vectors must satisfy the feasibility constraints derived below.

a. Processor space vector and projection vector must be ***orthogonal to each other***. If points A and B differ by the projection vector, i.e., $\mathbf{I_A}$-$\mathbf{I_B}$ is same as d, then must be executed by the same processor, in other words, $\boldsymbol{P^T I_A = P^T I_B}$. This leads to $\boldsymbol{P^T (I_A - I_B) = 0 \Rightarrow P^T d = 0}$.
b. If A and B are mapped to the same processor, then they cannot be executed at the same time, i.e., $\boldsymbol{S^T I_A \neq S^T I_B}$, i.e., $\boldsymbol{S^T d \neq 0}$.
c. Edge mapping: If an edge ***e*** exists in the space representation or DG, then an edge $\boldsymbol{P^T e}$ is introduced in the systolic array with $\boldsymbol{S^T e}$ delays.

C. Selection of scheduling vector

For any specified projection vector, processor space vector and scheduling vector, the systolic array can be designed using *linear mapping technique*. The method of selecting feasible scheduling vectors using scheduling inequalities is described. Based on the selected scheduling vector $\mathbf{S^T}$, the projection vector and the processor space vector $\mathbf{P^T}$ can be selected accordingly to equations $\boldsymbol{P^T d = 0}$ **and** $\mathbf{S^T d \neq 0}$. . Hence the desired systolic array can be obtained.

D. Selection of S^T based on scheduling inequalities

Selection of S^T based on scheduling inequalities:

For a dependence relation X →Y,

$$X: I_x = \begin{pmatrix} i_x \\ j_x \end{pmatrix} \rightarrow Y: I_y = \begin{pmatrix} i_y \\ j_y \end{pmatrix} \text{---------- (1)}$$

Where, I_x and I_y are the indices of node X and Y respectively. The scheduling inequality for this dependence is given by,

$$S_y \geq S_x + T_x \text{ ---------- (2)}$$

Where T_x is the computation time of node X. The scheduling equations can be classified into the following two types:

1. Linear scheduling,

Where

$$S_x = s^T I_x = (s_1\ s_2) \begin{pmatrix} i_x \\ j_x \end{pmatrix} \quad (3)$$

$$S_y = s^T I_y = (s1\ s2) \begin{pmatrix} i_y \\ j_y \end{pmatrix} \quad (4)$$

2. Affine Scheduling,

Where

$$S_x = s^T I_x + \gamma_x = (s_1\ s_2) \begin{pmatrix} i_x \\ j_x \end{pmatrix} \text{----------} \quad (5)$$

$$S_y = s^T I_y + \gamma_y = (s_1\ s_2) \begin{pmatrix} i_y \\ j_y \end{pmatrix} + \gamma_y \text{ ---------- (6)}$$

Using the forgoing definition, we can rewrite the scheduling equation for affine scheduling as

$$s^T I_x + \gamma y \geq s^T I_x + \gamma_x + T_x \text{ ---------- (7)}$$

Note that the scheduling equation for linear scheduling can be obtained by setting γ_x and γ_y equal to zero.
Define the edge from node X to node Y as $e_{x-y} = I_y - I_x$. Then the scheduling inequality for an edge is described as follows.

$$S^T e_{x-y} + \gamma_y - \gamma_x \geq T_x. \text{ ---------- (8)}$$

Therefore, one scheduling inequalities can be obtained for each fundamental edge in the dependency graph and the scheduling vector S^T can be obtained by solving these inequalities. Hence the selection of scheduling vector consists of 2 steps.

1. Capture all the fundamental edges. The reduced dependence graph (RDG) is used to capture the fundamental edges and the regular iteration algorithm (RIA) description of the corresponding problem is used to construct RDGs [8].

2. Construct the scheduling inequalities according to $S^T e_{x-y} + \gamma_y - \gamma_x \geq T_x$. and solve them for feasible S^T.

RIA Description

The Regular Iterative Algorithm (RIA) is introduced and the method for construction Reduced Dependency Graphs (RDGs) using RIA is illustrated.

a. The RIA is in standard input RIA form if the indexes of the inputs are the same for all equations.

b. The RIA is in standard output RIA form if all output indices, i.e. indices on the left side, are the same [8].

E.Basic Steps in Algorithms [2]

The present effort focuses on the use of ***evolutionary algorithms*** *for real-valued function optimization, without loss of generality attention is restricted to only function minimization*. The basic steps of algorithm are given as follows:

1. The problem is defined as finding the real-valued *n* dimensional vector *x* that is

associated with the external of a functional $F(x)$: R^n —> R. Without loss of generality, let the procedure be implemented as a minimization process.

2. An initial population of parent vectors, x_i, i = 1, . . . , P, is selected at random from a feasible range in each dimension. The distribution of initial trials is typically uniform.

3. An offspring vector, x'_i, i = 1, . . . , P, is created from each parent x_i by adding a Gaussian random variable with zero mean and preselected standard deviation to each component of x_i.

4. Selection then determines which of these vectors to maintain by comparing the errors $F(x_i)$ and F (x'_i), i= 1. . . P. The P vectors that possess the least error become the new parents for the next generation. In other words $F(x_i)$ and F (x'_i) is evaluated against ten other randomly chosen solutions from the population. For each comparison, a ‗win' is assigned if the solution's score is less than or equal to that of its opponent.

5. The process of generating new trials and selecting those with least error continues until a sufficient solution is reached or the available computation is exhausted. In other words, if the P solutions with the greatest number of wins are retained to be parents of the next generation.

In this model, each component of a trial solution is viewed as a behavioral trait, not as a gene. A genetic source for these phenotypic traits is presumed, but the nature of the linkage is not detailed. It is assumed that whatever genetic transformations occur, the result change.

In each behavioral trait will follow a ***Gaussian distribution*** with *zero mean difference* and some *standard deviation*. Specific genetic alterations can affect many phenotypic characteristics due to *pleiotropy* and *polygeny*. It is therefore appropriate to simultaneously vary all of the components of a parent in the creation of a new offspring [1][2][3][14].

F.Design of Systolic Architecture for 3 –Tap FIR Filter[8][12][15][18].

Consider a 3-tap FIR linear filter equation

$y(n) = w_0 * x(n) + w_1 * x(n-1) + w_2 * x(n-2)$ -----(9)

Where x (n), y (n) and (w_0, w_1, w2) are the inputs, output and co-efficients of the filter respectively at time ***n***. The above equation defines a finite impulse response finite time dependent on the number of filter coefficients. A filter design involves determining these coefficients to obtain the desired frequency response. The filter considered here is a special case of the general filter called as ***Moving Average Filter*** (MA). The output sequence can be obtained as a convolution (*) of the input sequence and the impulse response of the filter [7].

Dependency Graph for 3-Tap FIR Filter

Here DG has 3 fundamental edges: input moving upward represented by an edge in $[0\ 1]^T$ direction, coefficients moving toward the right represented by an edge in$[1, 0]^T$ direction, and results moving in$[1, -1]^T$ direction. Since all nodes in the DG contain the same three edges, this DG is regular

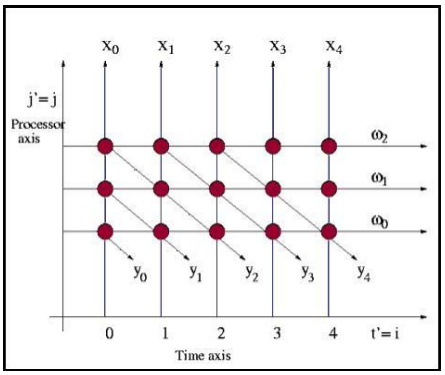

Figure-4: space representation for FIR filter

Nodes co-ordinates (i, j) of FIR filter is shown below

(0, 0)	(0, 1)	(0, 2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)
(3, 0)	(3, 1)	(3, 2)
(4, 0)	(4, 1)	(4, 2)

Table-1: Node coordinates obtain from figure-4

FIR filter design-1a (broadcast Inputs, move results, weights stay)

The systolic design-1 is derived by selecting the projection vector, processor vector and scheduling vector as follows

Projection vector d=$\begin{bmatrix}1 & 0\end{bmatrix}$

Processor vector $P^T = [0, 1]$

Scheduling vector $S^T = [1, 0]$

Checking of systolic array $P^T d = [0, 1] * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$

$S^T d = [1, 0] * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1\ 0$

Result---- → possible with selected vectors

FIR Filter Design-1A Node Mapping to Processor

Any node with index $I^T = (i, j)$ is mapped to processor

$$P^T = [0, 1] * \begin{bmatrix} i \\ j \end{bmatrix}$$

Therefore, all nodes on a horizontal line are mapped to the same processor any node with index $I^T = (i, j)$ is executed at time.

$$S^T = [1, 0] * \begin{bmatrix} i \\ j \end{bmatrix} = i$$

Since

$$S^T d = [1, 0] * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

Then

$$\text{HUE} = \frac{1}{|S^T d|} = 1$$

Edge Mapping For FIR Filter Design-1A

The 3 fundamental edges corresponding to weight, inputs and result can be mapped to corresponding edges in the systolic array accordingly to table-2

e^T	$P^T e$	$S^T e$
Wt(1, 0)	0	1
i/p (0,1)	1	0
Result(1,1)	-1	1

Table-2: Edge Mapping for FIR Filter Design1A

III.SYSTOLIC ARRAY DIAGRAM FOR *FIR FILTER DESIGN 1A*

The block diagram of design-1 systolic array is then constructed as shown in figure-5 the low level implementation of this architecture is shown in figure-7

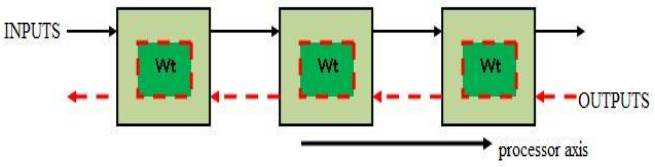

Figure-5: for Design 1A

A. Space-Time Representation for Fir Filter Design-1a

The space-time representation of design-1A is shown in figure-6. From which we can see that the incoming *x* value is available at all the processors at the same time. Specifically, the input data is ―broadcast‖ to the processors, similarly, the weights, *w* appear at the processors at the same spatial coordinates. Thus w_i values ―stay‖, and the output, y_i appears at the processor at different space and time. Hence the outputs ―moves‖

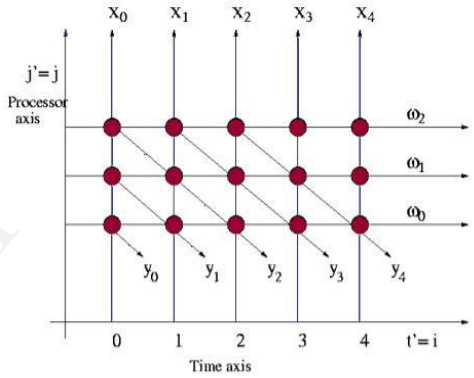

Figure-6: space-time representation of Design-1A

FIR Filter Design-1A Low Level Design Diagram

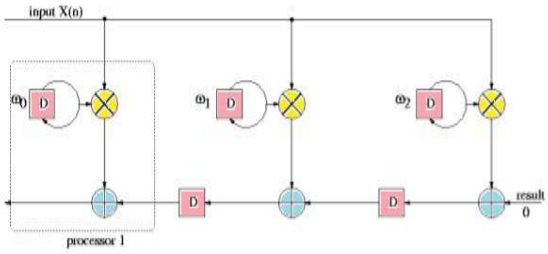

Figure-7: Low Level Design of Design-1A

Note: *Same procedure is also applicable for convolution[4][5][17] and correlation[8][16] or any other design.*

IV.DESIGN OF SYSTOLIC ARCHITECTURE FOR 2*2 MATRIX- MATRIX MULTIPLICATION.

The dependency graph (DG) for designing the systolic array for matrix-matrix multiplication corresponds to a three dimensional (3D) space representation linear projection is used to design 2D systolic architecture array[4] [9] [14].

Given 2 matrices A and B, we can denote their product as C=A*B, where A, B and C are n*n matrices for n=2 we have,

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \Rightarrow \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$C_{11}=a_{11}\, b_{11}+a_{12}\, b_{21}$---------- (a)

$C_{12}=a_{11}\, b_{12}+a_{12}\, b_{22}$---------- (b)

$C_{21}=a_{21}\, b_{11}+a_{22}\, b_{21}$---------- (c)

$C_{22}=a_{21}\, b_{12}+a_{22}\, b_{22}$---------- (d)

These equations can be represented in Space-Representation [9] shown in figure-8

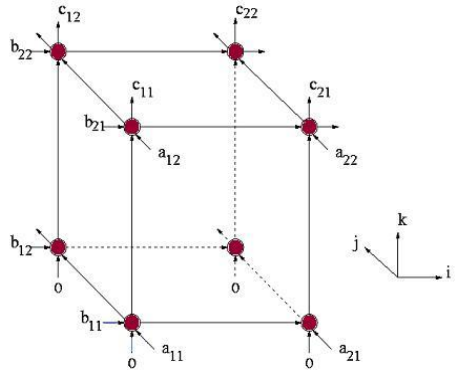

Figure-8: For Space-Representation of matrix multiplication

The iteration in standard output RIA form is as follows:

a (i,j,k) = a(i,j-1,k)

b (i,j,k) = b(i-1,j,k)

c (i,j,k) = c(i,j,k-1) + a(i,j,k) b(i,j,k)

The corresponding RDG (Reduce dependency graph) of matrix product computation is shown in figure-9.

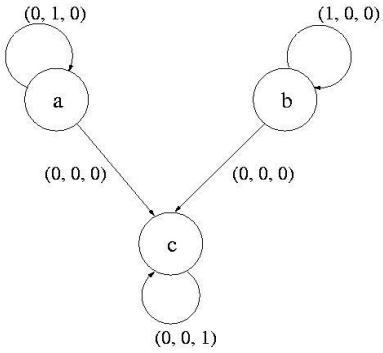

Figure-9: RDG of matrix product computation

Through RDG is difficult to make out projection vector and processor vector. And those will found by using scheduling inequality techniques. With the help of evolutionary programming projection vector and processor vector will be generated.

V.CONCLUSION

The present work undertaken has been successful for the designs considered. Results obtained (MATLAB coding) indicate that evolutionary computation (program) based synthesis of designs considered on systolic architecture are the best optimum results. The design of the DGs and proper selection of the 3 vectors (projection vector, processor vector and scheduling vector) are made in order to achieve an *optimal systolic design*. This project addressed the systolic architecture design methodology based on *Evolutionary Programming.* Systolic architecture designed for given regular iterative algorithms using linear mapping or projection techniques. The methodology is illustrated using FIR filter, matrix-matrix multiplication, convolution and correlation as examples. The MATLAB coding is done for all the designs considered, designs have been verified manually for all possible test cases.

FUTURE WORK

Always there is desired to get the chip module for the particular algorithm at present as well as in future, *to do this, technologies like nano technology has very good scope to implement the systolic architectures based on* Evolutionary Programming. Complex problems on systolic architecture up to certain extent can be implemented on FPGA in hardware module. Implementation of systolic algorithms based on Evolution Programming in ASIC for applications that require high computation precision will be feasible. With further developments in semiconductor industry and the inevitable possibility of implementation of systolic algorithm will result in increased demand for

efficient tools. Progress in Evolution Computation will contribute a lot in future VLSI Design, as it is still at initial stages.

ACKNOWLEDGEMENT

The author wishes to acknowledge Dr. Manoj Kumar (*planet i technologies*, Bangalore), Prof. Dr. Venkateswarlu (H. O. D, UTL, Bangalore) for their valuable contribution to this work.

REFERENCES

[1] Thomas Back, Ulrich Hammel, and Hans-Paul schwerel ‖Evolutionary Computation:-Comments on History and Current Status,‖ IEEE Transactions on evolutionary computation, vol.1 no.1, April 1997

[2] N.Saravanan, David b. Forgel, Kevin M. Nelson ―A comparison of methods for self-adpatation in evolutionary algorithms,‖ Elsevier science Ireland ltd, Biosystems vol.30 pp.157-166. 1995

[3] Soheil Aminzadeh ―using genetic algorithm for High-level sysnthesis,‖ available on http://ece.ut.ac.ir.

[4] S. Y. Kung ―VLSI Array Processor,‖ IEEE ASSP Magazine. July 1985.

[5] H. T. Kung ―Why systolic architectures?‖ IEEE computer magazine, vol-15, pp-37-45. Jan 1982.

[6] Bernard Chazelle ―Computational Geometry on a systolic Chip,‖ IEEE tans on computers, vol.c-33, no.9, sept 1984

[7]S. Y. Kung, VLSI Array processor. Prentice Hall, 1988.

[8]Keshab K. Parhi. VLSI Digital Signal Procrssing System Design and Implementation, john Wiley & sons inc-1999. [9] Halil Snopce, Lavdrim Elimazi. ―Reducing the Number of Processing elements in systolic array for matrix multiplication using linear transformation matrix,‖ Procedings of ICCCC. Vol.III, PP 486-490,2008

[10]David B. Fogel, ―What is evolutionary computation?,‖ IEEE SPECTRUM pp.26-31. Feb-2000.

[11]Chang Wook Ahn, Advances in Evolutionary Algorithms. Springer-Verlag Berlin Heidelberg 2006

[12]V.Visvanathan. Nibeita Mohanty. S. Ramanathan ―An Area-Efficient systolic architecture for Real-Time VLSI Finate Impuse Response Filters,‖ 6th international conference on VLSI Design. pp.166-171. jan 1993.

[13] William M. Spears, Kenneth A. De Jong, Thomas Back. David B. Fogel, Hugo de Garis, ―An Overview of Evolutionary Computation,‖ proceedings of European Conference on Machine Learning. 1993

[14] I. Z. Milovanovic, E. I. Milovanovic, B. M. Rangjelovic, I. C. Jovanovic ―Matrix multiplication on bidirectional linear systolic arrays,‖ mathematics subject classification Filomat 17(2003), pp.135-141.

[15]Mokhtar A. Aboelaze, De-Lei Lee, Benjamin W. Wah ― Two-Dimensional Digital Filtering Using Constant-I/O systolic Arrays,‖ proc.IEEE int'l symp on circiuts and systems, pp.255-258, 1993.

[16]L. J. Deutshch and C. R. Lahmeyer ― A systolic Architecture for the correlation and accumulation of digital sequences,‖ TDA progress report. 62-68 jan-mar 1986.

[17]Avtar Singh and S. Srinivasa, Digital Signal Processing, cengage learning India Pvt Ltd, 2004.

[18]Surin Kittitornkun and Yu Hen Hu, Reconfigurable Processor Array Synthesis, Department of Electrical and Computer Engineering, UW-Madison, PDCAT June 30, 2001.