THE DESIGN OF A SYSTOLIC ARCHITECTURE TO IMPLEMENT GRAPHIC TRANSFORMATIONS

George M. Papadourakis and George N. Babis

University of Crete
Department of Computer Science
P.O. Box 1470, Iraklion, Crete, GREECE
E-mail: papadour@cse.forth.gr

ABSTRACT

A new systolic graphic array architecture is proposed in order to achieve high computational throughput, necessary to generate real-time processing. The proposed systolic architecture will implement graphic transformations such as translations, scaling, and rotations on three dimension vertices. The Logarithmic Number System is utilized to further increase the computational throughput. A comparison between the proposed systolic architecture and the Weitek 7100 SME graphic processor in terms of speed is performed.

1. Introduction

Graphics images are used in sophisticated flight-training simulators for the display of out-of-window scenes viewed by the crew under training. These simulators use high-performance graphics generator hardware capable of generating 3D perspective images at rates of 30 frames per second. In addition, these systems have large data bases containing the information necessary to produce images of runways, buildings, aircraft, vehicles, other aircraft and obstacles. Since an enormous amount of real-time processing is required in order to generate these images, most of the graphic processors use systolic processors. The function of the array processors is to transform, clip, rotate and perspective project the data base into screen coordinates for use by the graphic generator. Since the display screen needs to be updated at a rate of 30 frames per second, for real-time simulation, the amount of computations performed by the array processor determines the quality of the images that are generated.

In order to achieve high computational throughput, the implementation of the above mentioned functions of the array processor, utilizing few Instructions. Multiple Data (FIMD) systolic architectures, is proposed. Systolic arrays have revolutionized the implementation of matrix signal processing and numeric algorithms. They are a network of several interconnected identical cells which satisfy the constraints of locality, modularity, and pipelineability. The fundamental attraction of this architecture is that it lends itself to VLSI implementation and achieves high computational rates due to the use of high data bandwidth and highly localized buses. Systolic designs have been utilized to perform matrix operations [1-4] at high speeds. One problem with systolic arrays is that each cell performs only one basic operation resulting in low processing element (PE) utilization. Papadourakis and Taylor [4] presented a scheme to overcome the above mentioned problem with the introduction of FIMD systolic architectures. The FIMD architecture also makes use of efficient software techniques to perform high level pipelining so a number of operations can be performed simultaneously yielding higher PE utilization and throughput.

In this paper, the application of FIMD systolic architectures to Computer Image Generation Systems will be presented. The proposed FIMD systolic board design will implement the graphics computational function of performing transformations of image vertices. In order to achieve even higher throughput the FIMD systolic board was designed using the Logarithmic Number System (LNS) [5]. Processors utilizing LNS have outperformed existing floating point (FLP) processors. The numbers in LNS are represented as a signed radix raised to some signed exponent. All the arithmetic in this system can be performed using only the exponents [4].

The proposed FIMD board consists of a master controller, an orthogonal array of 16 PEs, a PE controller, an array pipeline, array output buffers, sine and cosine look-up tables, on-board RAM, memory and access controllers for RAM. FLP to LNS and LNS to FLP converters. The PE is the major component of the systolic array and the VLSI PE designed in [6] is utilized. Finally, a detailed timing comparison between the proposed systolic array board and the Weitek 7100 SME board is performed.

2. Graphic Transformation

Three-dimensional images can be represented as polygons made up of a collection of vertices with x, y and z coordinates in right handed cartesian coordinate space. A homogeneous representation of a vertex (x,y,z) is (x,y,z,1). This homogeneous formulation allows the specification of 4x4 transformation matrices. Generally, given a vertex (x,y,z), and a 4x4 transformation matrix T, the transformed vertex is given by:

\[(x',y',z') = T(x,y,z)\]  \[(1)\]

The authors are also affiliated with the:

Institute of Computer Science - FORTH HELLAS

(2)

or its transposed counterpart \[M^T\]  \[(3)\]

3. Proposed Systolic Architecture

The desired systolic architecture must be able to efficiently perform square matrix and matrix multiplication. A 4x4 orthogonal array architecture shown in Figure (1) meets the needs of performing the necessary operations in a parallel manner. Multiplying two matrices serially would take \[n^2(2n-1)\] computational units. An algorithm developed by Kung for matrix multiplication with systolic array of \[n^2\] results in \[4n-1\] computational units [1]. Further improvements were made to this algorithm in [4]. Introducing the ideas of intensive pipelining and the use of more sophisticated PEs, the computational time was reduced to \([2n^2]\) computational units.

The faster matrix multiplication algorithm developed in [4] begins with establishing a pipeline for each PE on the bottom row of the systolic array. These parallel pipelines will pipe data values to the bottom border PEs. The pipelines must be kept full with values for the PEs to work with. This parallelism of pipelines to the border PEs enhances the performance of the systolic array.

The matrices elements to be multiplied are loaded into the pipeline in a skewed manner. The systolic array is then loaded with the first
matrix A. Before the load of A is completed, the elements of the second matrix B are piped into the array for the computation of \( Z = A * B \) to begin.

### 3.1. Matrix Operations

The equations used in graphic transformations (1) and (3), can be analyzed to determine the matrix operations that must be performed by the systolic architecture. The required functions that the systolic architecture must perform are square matrix multiplication and matrix vector multiplication. The square matrix multiplication operation is needed to calculate the final transform matrix \( M \). The matrix vector multiplication operation is needed to perform transformations on vertices. In addition to these matrix multiplication operations, an algorithm is needed to efficiently load the systolic array with matrix elements. The matrix load algorithm is vital to the computational speed with which the matrix multiplication operations are performed.

#### 3.1.1. Matrix Load

The loading order of the matrix elements into the systolic array is important. The loading matrix is loaded into the systolic array with the \( n \)-th row loaded first. Array columns are loaded with corresponding column elements of matrix \( A \) piped in a skewed fashion as shown in Figure (2). Columns are loaded such that element \( a_{kJ} \) is loaded on the first clock cycle, \( a_{k2} \) is loaded on the second clock cycle and so on. This is necessary to keep the pipeline full during multiplication.

![Figure 2. Matrix Load Operation](image)

The loading algorithm inputs element \( a_{ij} \) into \( P_{E1} \) during the first clock cycle; elements \( a_{i1}, a_{i2} \) and \( a_{i3} \) are input to \( P_{E1}, P_{E2}, \) and \( P_{E3} \), respectively, during the second clock cycle and so on. The matrix element \( a_{i4} \) passes through \( P_{E4}, P_{E5} \), and \( P_{E6} \) before it is stored in \( P_{E7} \). This process continues for all array columns. When \( P_{E7} \) has stored the last element of \( A, a_{11} \), it is ready to begin processing of the next clock cycle (7).

#### 3.1.2. Square Matrix Multiplication

The 4x4 orthogonal systolic array is well suited for the square matrix multiplication operation that needs to be performed. The multiplication of homogeneous transformation matrices requires that two square 4x4 matrices must be multiplied. Matrix multiplication is not commutative. The premultiplier matrix \( A \), must be loaded first and matrix \( B \) is then piped into the array for interaction with \( A \).

The postmultiplier matrix \( B \), is piped into the array in a different order than the loading matrix. The first row of \( B \) is input to the first column of the array, the second row of \( B \) is input to the second column of the array, and so on. Again as in the load algorithm, elements are piped in skewed fashion as shown in Figure (3). This pipelining and skewed arrangement produces an efficient machine.

An element of the matrix product \( Z \), where \( Z = A * B \), can be calculated with the recurrent equation

\[
z_{ij} = \sum_{k=1}^{4} a_{ik} b_{kj}
\]

where \( n \) is the dimension of the matrix, which in our case is 4. As mentioned before, the calculation of the product of two \( n \times n \) matrices takes \( 2n^3 \) computational time units. The loading operation requires \( n \) units and the \( n \) units are needed to compute \( z_{11} \).

During the multiplication operation, each appropriate \( PE_{ij} \) takes the sum of partial products from the left neighbor and adds it to the product of \( a_{ij} b_{ij} \). Therefore, each product matrix element is obtained by accumulating data in the rows of the systolic array from left to right as shown in Figure (2). Concurrently with the multiplication shifting is being performed in the \( PE \)s, i.e. \( B \) data is piped one row deeper into the array. The \( PE \)s in the right column output the resulting product matrix.

#### 3.1.3. Matrix Vector Multiplication

Matrix vector multiplication can be performed efficiently with the proposed 4x4 systolic array architecture. The operation will perform a multiply of a 4x4 matrix and 1x4 vectors. The procedure is to load the 4x4 matrix into the array as described previously with the matrix load operation. The vectors will then be piped into the array to be premultiplied by the matrix residing in the array. The product 1x4 vector will be outputs from the right columns of the array as illustrated in Figure (4).

![Figure 3. Square Matrix Multiply Procedure](image)

![Figure 4. Matrix Vector Multiply Procedure](image)

### 3.2. Advantages of Transposed Matrices

There are three significant advantages to using the equations (1) and (3), which use transposed transformation matrices. The first advantage is the loading matrices are available to be loaded into the array without any delay because these matrices are available in the board's memory. Secondly, the pipeline is kept full with the partial products being calculated from the array while the load operation is taking place. Thirdly, the array is kept busy all times either loading or performing matrix multiplication. The final result of utilizing the equations which have tran-
posed matrices is maximum efficiency and throughput of the systolic array.

3.3. Procedure for Loading and Pipelining Matrices

The inputs to the board will come in the form of commands to generate transformation matrices, i.e., ROTX(ang), ROTY(ang), ROTZ(ang), TRANS(x,y,z), and SCALE(x,y,z). The board will generate the transformation matrices corresponding to each command and store it in the board's memory. The proposed board will then calculate a single transformation matrix composed of one to five transformation matrices. This can be illustrated by the following example:

Given: TRANS(x,y,z), SCALE(x,y,z), TRANS(x,y,z), n Vertices

Procedure for Loading and Pipelining Matrices

Step 1. Calculate the three transformation matrices, transpose them, and store in memory as A, B, and C matrices.

Step 2. Load into the array matrix B with the load matrix operation.

Step 3. Pipe in matrix A to be multiplied with matrix B partially loaded in the array.

Step 4. Load into the array matrix C with the load matrix operation.

Step 5. Pipe the product B*A as results come out of the array to be multiplied with matrix C partially loaded in the array.

Step 6. Load into the array the product C*B*A as results come out of the array. This is the final composed transformation matrix M. (Note: There will be a 3 time unit delay as a result of waiting for the calculated elements C*B*A to come from the array.)

Step 7. Pipe in the n vertices to be multiplied with matrix M in the matrix vector multiplication operation.

Step 8. The transformed vertices elements are stored back into memory as they come out of the array.

4. Logarithmic Number System

The use of logarithmic number system (LNS) has been instrumental in the development of high speed computation bound processors. Processors utilizing the LNs have outperformed existing floating point (FLP) processors. The numbers in LNs are represented as a signed radix raised to some signed exponent. All the arithmetic in the system can be performed using only the exponent [8].

LNS has the ability to perform very fast square, square root, multiplication, and division operations. Square and square root calculations are done with a shift of the exponent left or right respectively by one bit position. Multiplication and division calculations are done with an addition or subtraction operation on the exponents. Addition and subtraction calculations are performed with high speed look-up tables on a ROM.

Numbers in LNS are represented as

\[ X = (s)_r e_r \]

where \( r \) is the radix, \( s_r \) is the radix sign bit and \( e_r \) is the signed exponent. The radix is 2 and \( e_r \) is a two's complement number with a 6-bit integer part and a 12-bit fractional part. A LNS number is represented in the word format shown below

\[ s_{ex} \quad 19 \text{-bit exponent magnitude} \]

A zero flag must be provided with each data to represent if \( s = 0 \), since \( \log(0) = \infty \). Equations for LNS arithmetic are below

Multiplication

\[ Z = X \cdot Y = (-1)^{s_x + s_y} r^{e_x + e_y} \]

Division

\[ Z = X / Y = (-1)^{e_x - e_y} r^{e_x - e_y} \]

Addition

\[ Z = X + Y = (-1)^{e_x + e_y} \]

Subtraction

\[ Z = X - Y = (-1)^{e_x - e_y} \]

Addition and subtraction operations must be realized with a ROM look-up table. Equations (6) and (7) can be expressed respectively as

\[ Z = X + Y = (-1)^{e_x + e_y + e_y} \]

and

\[ Z = X - Y = (-1)^{e_x - e_y + e_y} \]

Taking the logarithm of both sides of (8) and (9) produces respectively

\[ e_z = e_x + \log_2(1 + r^{e_y}) \]

\[ e_z = e_x + \log_2(1 - r^{-e_y}) \]

5. Processing Element VLSI Chip

The preprocessing element, PE, is the major component of a systolic array. The systolic array implementation of graphic transforms require a PE design that will support high speed matrix computations. The VLSI PE chip designed by [9], will meet the specifications needed for systolic array implementation of graphic transformations.

Condoridis’s PE design [9], uses LNS to allow high speed arithmetic operations. The cells are able to pass data to neighboring cells while the ALU performs computations. The individual cells can be programmed with microcode and be reconfigurable. Internal parallelism within the array can be achieved as well as parallelism with external components outside the array. The PE design contains an internal and external data routing switches and multiple internal and external buses. Figure (5) shows the block diagram of the PE.

Figure 5. Block Diagram of PE Designed by Condoridis (1987)

5.1. PE Control

The systolic array implementation to perform graphic transformation requires that the PEs will perform in parallel two different tasks. The first task is to perform a matrix load and the second task is to perform matrix multiply computations. The PE controller must be able to partition the array in such a way that while PEs are loading other PEs are performing computations. Efficiency in PE control algorithm is critical to the throughput of the overall system. There must not be a break in the systolic pipeline that would produce inefficiency resulting from processor idle time. Synchronization of data and instructions must be accomplished to provide the individual PEs proper timing with respect to one another. At the same clock interval, PE is provided with synchronized data and instruction inputs. The synchronization algorithm which will be used is a wavefront, where the PE opcodes are propagated in a wave-like manner through the lowest diagonal of the array to the highest diagonal as illustrated in Figure (6). The opcodes will instruct the individual cells what operations need to be performed. The wavefront algorithm begins with an opcode to first diagonal which is the single PE, PE1. On the next clock cycle the instruction is passed from the first diagonal to the second diagonal of cells, PE2, and PE3. During this same clock cycle the first diagonal is given a new opcode. The same pattern holds for the third clock cycle. Opcodes propagate from the second diagonal cells to the third diagonal cells, the first diagonal cell to the second diagonal cells, and the first diagonal cell is given a new opcode. This wavefront control scheme continues through the array in 2e-1 clocks. Each PE receives identical opcodes for each matrix operation. Clock delays within the PE shown in Figure (6) are associated with the receipt of the first opcode by PE.
6. Proposed Systolic Architecture Based Design

The proposed FIMD systolic based design will implement the computer graphics computational function of performing transformations on image vertices. The proposed board is capable of achieving high computational throughput that is required of an array processor within a graphic generator system. The proposed board will consist of a master controller, an orthogonal array of 16 PEs, a PE controller, an array pipeline, array output buffers, sine and cosines look-up tables, on-board RAM, memory access controllers for RAM, FPL to LNS and LNS to FPL converters. Figure (7) is a schematic of the proposed board.

6.1. System Clock

Synchronization of all the components on the board is vital. An 150ns system clock rate will be internal to the proposed systolic board. This clock rate was assumed to allow time for the worst case ALU functions of addition and multiplication (114ns) along with a system overhead margin. All the PEs will be clocked at the system rate, as well as the output buffers, the registers in the pipeline, and the LNS conversion chip.

6.2. The Master Controller

The function of the master controller is to provide the control for all the components on the board. It will directly control the output buffers from the systolic array, the access controller to the on-board RAM, the LNS conversion chips, and the array pipeline controller. Figure (7) represents control lines with dashed lines.

6.3. The Orthogonal Array of PEs

The systolic array will be made up of 16 individual PEs configured as a 4x4 orthogonal array. This network of several interconnected identical cells will satisfy the constraints of locality, modularity and pipelinaability needed of the systolic array.

The PE design of (9) will allow the PEs to pass data and control bits to the neighboring cells directly above and to the right. The LNS ALU within the individual cells provides the high speed arithmetic calculations needed to perform matrix operations.

Inputs are clocked into the bottom row of the array from the systolic array pipeline. Outputs will also be clocked by the same system clock to create a totally synchronized systolic array. The outputs will be clocked from the rightmost column of the array to FIFO buffers.

6.3.1. The PE Controller

The PE controller unit will send control bits to the first PE in the array. As previously discussed, this control will go throughout the array in a wave-like fashion. The controller unit will generate and store the control bits to direct the PEs to calculate the transformation matrix and transform vertices.

6.4. The Systolic Array Pipeline

The systolic array pipeline consists of an 8x4 array of registers each 20 bits wide. These registers are synchronized with the 150ns system clock. A pipeline controller will be used to control flow of data into individual registers. At the beginning of every clock cycle the register contents will shift upward into the neighboring register or into the systolic array.

Included in the pipeline module is a 12-to-4 multiplexer from which the pipeline is filled with data from memory or one of the two array output buffers. Control bits from the master controller select which inputs to the multiplexer will be used. The master controller will also provide the control bits needed by the pipeline controller to select the registers the data will be clocked into.

6.5. The Systolic Array Output Buffers

Three output buffers are needed to hold the values of the calculations done by the systolic array. These buffers are simply FIFO registers. Each buffer module consists of four sets of four register FIFOs. The four register FIFO was assumed because of the skewed nature of the array outputs. All buffer modules are under the control of a master controller and are synchronized with the system clock.

The master controller enables buffer 1 when the array outputs are the transformed vertices coordinates. The transformed vertices coordinates are coming from the array in a skewed form. Delays were added to the outputs so that at a given time interval, all the coordinates that make up a vertex are on the output of buffer module 1. The outputs of this first buffer are sent to be converted to LNS and stored in memory.

Buffer 2 is enabled by the master controller when the array outputs are the elements of the final transformation matrix. Unlike the other two buffer modules, the skewed outputs from this buffer need to be sent to the pipeline as quickly as possible and do not need in-line delays.

Buffer 3 is enabled by the master controller when the array outputs are partial products of the calculation of the final transformation matrix. It is important to note that this buffer's inputs are clocked in a reverse order as compared to the inputs of buffer 2. This difference of order is necessary because the loading of the matrix elements to the array pipeline is not the same. The correct element order must be maintained. Again, the skewed nature of the output requires that delays be put on the outputs of this buffer.

6.6. The Sine and Cosine Look-up Tables

The sine and cosine operations require a look-up table to evaluate the functions of sin(angle) and cos(angle), where angle is a 16-bit FPL integer which specifies the angle of rotation in 360/4096ths of a degree.

The sine and cosine look-up tables will simply be two ROMs with the 20 bit LNS values of sine and cosine operations. The 16 bit FPL integer representing the angle can be decoded to conveniently address 4096 locations within the ROM table corresponding to the desired operation. Therefore, the sine/cosine look-up table unit will contain two 4K x 20-bit ROMs, each with a 16-to-1 decoder to address the ROM locations.

6.7. The On-Board RAM

The board has a 2K x 32-bit RAM synchronized with the system clock. Six reserved blocks of 16 locations are allocated to contain the 5 possible input transformation matrices and the final transformation matrix as calculated by the board. The remaining memory will be used to store up to 488 vertices. It is beneficial for memory to be composed of smaller RAM chips so that while a read can be performed from one chip, a write can be performed to another chip. Each transformation matrix will reside in separate chips.

Vertices coming into the board via I/O controller are stored into RAM in their original 32-bit FLP format. The 32-bit FLP coordinates are sent serially from the FLP to LNS converter and replaced by 20 bit LNS values. The 20 bit LNS coordinates are then sent to the array pipeline to be transformed. The transformed 20 bit LNS vertices are converted back to FLP format and are stored as 32 bit FPL values in the RAM.

6.8. The Memory Controller and the Access Controller

The function of the memory controller unit is to control the accessing to memory locations, the writing to locations and the reading from locations. Parallelism of reading, writing and accessing locations must
be achieved by this unit. The memory controller will be assumed to be an intelligent and sophisticated component. Its tasks are:

1. Store 32 bit FLP format input matrices in memory locations reserved for the individual matrices.
2. Read from matrix locations to fill the pipe to the FLP to LNS converter.
3. Write back to matrix locations the 20 bit LNS values or the matrix from the buffer on the converter's outputs.
4. Read from locations to fill the systolic array pipeline.
5. Write the newly transformed vertices coordinate values to the RAM locations reserved for the vertices.

The benefit of using multiple small RAM chips is that parallelism of tasks involving memory can be accomplished. For example, while the controller is piping one matrix (FLP read) from the FLP to the LNS converter, it can be writing the LNS values coming from the converter to another chip. At the same time, the contents of another chip newly converted LNS matrix values can be sent to the systolic array pipeline.

The memory access controller's function is to access the contents of the locations addressed.

6.9. The FLP to LNS Converter

The FLP to LNS convertor will be used to convert from 32 bit FPL values to 20 bit LNS values. The pipeline to the convertor chip consists of 16 registers, each 20 bits wide. This number of registers was assumed to hold the elements of one transformation matrix. The converted values are sent back to memory via another 16x20 bit buffer.

A divide-by-3 internal clock within the memory module can be used to clock the memory contents into the pipeline every 50ns. The convertor chip is synchronized with this clock rate. This 50ns cycle time will ensure the conversion chip the time to convert to LNS (3811s) along with a marginally overhead time.

6.10. The LNS to FLP Converter

There are four LNS to FPL convertor chips that will be used to convert the newly transformed vertices coordinates from LNS to FLP. These chips are synchronized with the system clock of 150ns. The inputs to these convertors come from the four buffered outputs of the systolic array. The outputs are sent to the memory module to store the transformed vertices.

7. Timing Comparison of Proposed Board vs Weitek 7100 SME

A timing comparison between the proposed systolic array board and the Weitek 7100 SME [10] was performed with regard to computational speed. Analysis for the worst case, the medium case, and the best case is shown. The task each board performs is the calculation of the transformation matrix and the transformation of \( n \) vertices. The Weitek 7100 board performs calculations with a pipeline serial architecture. Calculations involving matrices are performed only with the matrix elements that will have effect on the resultant matrix. Although the Weitek board performs fewer calculations, the speed at which these calculations are performed is much slower than the matrix multiplying done with the proposed systolic board.

It is assumed that the start-up overhead time for both boards is equivalent. In addition, the time to load the pipe for the Weitek board’s ALU will be equivalent to the time it takes to load the pipe to the proposed board’s systolic array. Therefore, these times will not be included in the timing equations.

7.1. Proposed Systolic Architecture Board Timing Equations

Given the task to compose the transformation matrix from \( m \) transformation matrices and to transform \( n \) vertices, the proposed systolic architecture board will perform this according to the equations:

\[
T = (8m + n + 16.33)n + (4t) \text{ for } m=1
\]

\[
T = (n + 21.33)n \text{ for } m=1
\]

where \( t \) is the system clock time of 150ns, and the start-up and flushing times are included. The brackets [ ] indicate that this time is only used if the first matrix requires sine/cosine look-up.

7.2. Weitek 7100 SME Timing Equations

The Weitek board timing equations take in consideration that calculations are done as needed. To come up with accurate timing equations, involves analyzing which resultant matrix elements will be effected by the multiplication of two transformation matrices. The following equations describe the postmultiplication timing for translation \( (T_3) \), scaling \( (T_4) \), and rotation \( (T_5) \) transformation matrices:

\[
T_3 = (n + 8) \text{ of coordinates translated}
\]

\[
T_4 = (16 - 8 \text{ of common matrices elements})
\]

\[
T_5 = (\text{matrix elements in both matrices})
\]

In addition, \( t \) indicates the upper-bound time for a pipelined operation with the WTL 1232 floating point multiplier and the WTL 1233 floating point ALU which is 100ns. Similarly, postmultiplying a vertex with a transformation depends on the vector elements of the vertex and the matrix element in the transformation matrix. Therefore, the time \( T_6 \) required to perform vertex transformations is:

\[
T_6 = \text{matrix elements, vector elements} \times n
\]

where \( n \) is the number of vertices.

8. Performance Comparison

A performance comparison with regard to time is demonstrated on both boards with three different tasks. The first and easiest task is to transform \( n \) vertices with a single transformation matrix. The second task is to compose a transformation matrix from three transformation matrices and transform \( n \) vertices. The third and most complex task is to compose a transformation matrix from five transformation matrices and transform \( n \) vertices.

The first task includes only a translation \( \text{TRANS(dx,dy,0)} \) and it results in the following time equations for each board:

\[
\text{Serial Board } T_1 = (2n + 100)n \text{ns}
\]

\[
\text{Systolic Board } T_1 = (n + 21.33)150ns
\]

The second task consists of a translation \( \text{TRANS(dx,dy,dz)} \), a rotation \( \text{ROTZ(ang)} \), and again a translation \( \text{TRANS(dx,dy,dz)} \). The time equations for each board are:

\[
\text{Serial Board } T_2 = (5n + 11)100ns
\]

\[
\text{Systolic Board } T_2 = (n + 40.33)150ns
\]

Finally, the third task is to compose a transformation matrix from the following five transformation matrices: \( \text{TRANS(dx,dy,dz)} \), \( \text{SCALE(Sx,Sy,Sz)} \), \( \text{ROTZ(ang)} \), \( \text{ROTX(ang)} \), and \( \text{TRANS(dx,dy,dz)} \). The time equations for each board are:

\[
\text{Serial Board } T_3 = (8n + 35)100ns
\]

\[
\text{Systolic Board } T_3 = (n + 55.33)150ns
\]

Figures (8), (9), and (10) illustrate graphically the performances of the proposed systolic board and the Weitek board respectively. Figure (11) is a bar graph illustrating the maximum number of transformed vertices for the time constraint of 30 frames per second for all three tasks. It is clearly demonstrated that the proposed board outperforms the Weitek board.
9. Conclusions

An implementation of graphic transformations was obtained using a systolic architecture. The FIMD (Few Instructions Multiple Data) systolic architecture was able to achieve the high computation throughput that is required by an array processor for graphic generation.

The throughput of this new design exceeded previous implementations employing pipelined serial architecture with speed-ups of 2 to 5 depending on the number of transformations and the number of vertices. More importantly however, larger sets of vertices can be transformed with the proposed systolic architecture than could be transformed with the serial architecture. To achieve the 30 frames per second, all vertices must be transformed within a time constraint of 33.33 milliseconds. When \( m = 3 \), \( m \) is the number of transformation matrices) the serial architecture based board can transform approximately 60K vertices within 33.33 milliseconds. In contrast, the proposed systolic architecture based board can transform 220K vertices within the same time. This is a factor of 3.33 more vertices that can be transformed using the systolic architecture.

This factor increases in favor of the systolic architecture as the number of transformation matrices and vertices increases. For example, when \( m = 5 \), the systolic board and the serial architecture based board will be able to successfully transform 220K and 41K vertices respectively. This is a factor of 5.36 more vertex transformations that can be achieved with the systolic design.

REFERENCES