1. Introduction

The need to double the computation speed every 18 month (Moore law) still exists. However, current VLSI technology is limited to operate below a certain frequency. To cope with the limitations, microprocessor industry suggests the multi-core technology, essentially suggesting the use of parallel computations. This is a dramatic change; the implications are beyond the need to educate programmers to program parallel machines. In fact, it is not always possible to convert a task to several concurrent sub-tasks. The need for fast communication capabilities among the cores introduces new approaches that are based on optical communication over the silicon. The fact that no cross-talks occur in optics is widely used in fiber optics communication. The revolution in the microprocessor architecture may open new opportunities for the use of optical computing. Several new architectures were recently suggested, e.g. [5, 16- 19]. In particular, the architecture based on [5] was implemented in industry.

In this work we suggest new solutions and applications (e.g., [16], [17]) that may enhance the capabilities of architectures that are based on optical vector by matrix multiplications. The fact that optical matrix multiplication can be done in parallel in a significantly faster fashion must be supported by electronic interface architecture that has to convey data to the multiplier in a way that utilizes the multiplier capabilities. Otherwise, the speed of the computation will be determined by this bottleneck.

A vector-by-matrix multiplication is involved in several computationally-intensive applications such as rendering of computer-generated images, beam forming, radar detection, and 3G communication systems. Many of these practical applications require efficient and fast implementation of vector-by-matrix multiplication.

In addition, vector-by-matrix multiplication can be used as the building block for numerous DSP procedures such as convolution, correlation, and certain transformations, as well as, distance and similarity measurements. For example, a discrete Fourier transform (DFT) can be implemented as a special case of vector-by-matrix multiplication.

In this paper we suggest an efficient electro-optical implementation of a vector-by-matrix multiplier (VMM). The proposed electro-optical VMM can perform a general vector (*1x256x8* bit) by matrix (*256x256x8 bit*) multiplication in one cycle of *8* nanoseconds (that is, at a rate of *125* MHz). This rate is the faster than all other VMMs available today and it is expected to improve with introduction of new electro-optical technology. The proposed VMM can serve as a co-processor attached to a DSP or a RISC CPU (referred to as a controller) and significantly enhance the performance of the controller.

The bottleneck of the similar previous design [5] was due to the fact that the spatial light modulator (SLM) used to represent the matrix in the VMM could not be updated in a high rate due to the electrical driver limitation. This bottleneck is solved in this paper by proposing a new electrical driver and SLM designs.

We discuss the proposed implementation, show that it is a feasible implementation, and explain how it can be improved along with technology advances. The proposed VMM is composed of two main units the electro-optical unit and the electrical driver.

2. The VMM Electro-Optical Unit

The ability to perform mathematical operations in free space, in parallel, and without mutual interactions among the various signals is only part of the inherent features of optical data processing. These features are utilized in many optical VMM configurations that have been suggested in the technical literature [1-4]. The VMM unit has two main components: the Optical unit and the electrical driver. The next two sections elaborate on these tow components respectively.

2.1 The VMM Optical Unit

Several configurations can be utilized to implement the optical component of the VMM. One of these configurations is the Stanford multiplier [4] illustrated in Fig. 1. As shown in this figure, the input vector of the VMM is represented by a set of incoherent light sources, the matrix of the VMM is represented by a slide mask or a real-time spatial light modulator (SLM) and the output (product) vector of the VMM is represented by a set of sensitive detectors. The light from each of the sources is spread vertically so that it illuminates a single column in the matrix and then each row in the matrix is summed onto a single detector in the detector array. This VMM configuration can be performed by several optical techniques. One of these techniques uses two sets of lenses. Each of these sets contains a cylindrical lens and a spherical lens and each of these lenses has a focal length of *f*. A single set of lenses has an equivalent focal length of *f*/2 in one direction and *f* in the other direction. As shown in Fig. 1, the first set of lenses is positioned between the input vector (represented by the incoherent light sources) and the matrix (represented by the SLM). This set of lenses is positioned so that the light coming from each of the sources illuminates only a single column in the matrix, which means collimating the light diverging vertically from each of the sources, but imaging it in the horizontal direction.

The other set of lenses is positioned between the matrix (represented by the SLM) and the output vector (represented by the sensitive detectors) and as shown in Fig. 1, it is rotated by *90* degrees compared to the first set of lenses. Therefore, these lenses image a row in the matrix onto a vertical position of a single detector in the detector array and spread the light from a single column of the matrix.

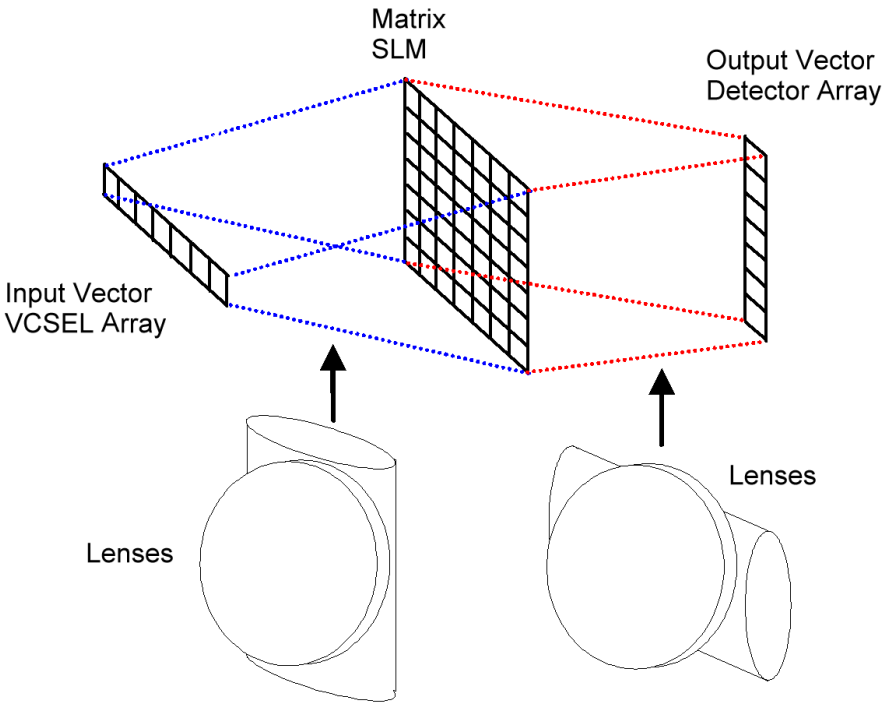

Fig. 1. The Stanford VMM.

An array of *1×256* vertical cavity surface emitting lasers (VCSELs), which are able to convert electrical signals (of *8-9* bits) to optical signals (*256* levels of intensity) at the rate of *125* MHz, is used to represent the input vector. As suggested by the Stanford VMM, each element from the input vector is projected onto a column in the *256×256* matrix, whereas each row of the matrix is summed onto a single detector in the *256×1* detector array.

In order to represent the *256×256* matrix (and in difference to other VMM suggested in the past [5]), we suggest to use *256* units, each of which contains a *1×256×8* bit multiple quantum well (MQW) SLM and is able to modulate light at the frame rate of *125* MHz. Each SLM unit is electrically derived in separate to the other units, and therefore, as we will show later, we are able to update the matrix of the VMM in the true frame rate of *125* MHz (and potentially even higher rates) and we are not restricted to the input bottleneck of a conventional *256×256* SLM (as used in other VMM architectures) [5].

2.2 The VMM Electrical Driver

Figure 2 and Fig. 3 contain high level descriptions of the electrical components of the system. Figure 2 shows the entire VMM electrical driver, whereas Fig. 3 shows the design of a single *1×256×8* bit electrical driver (SED). Section 5, outlines a possible implantation of the system.

The VMM electrical driver has two types of electrical inputs:

a. A *1×256×8* bit vector input '*A*'; and

b. A set of *256* vector inputs 'B_0' to 'B_{255}' (total of *256×256×8* bit).

The output of the VMM is a *1×256×20* bit vector '*C*'.

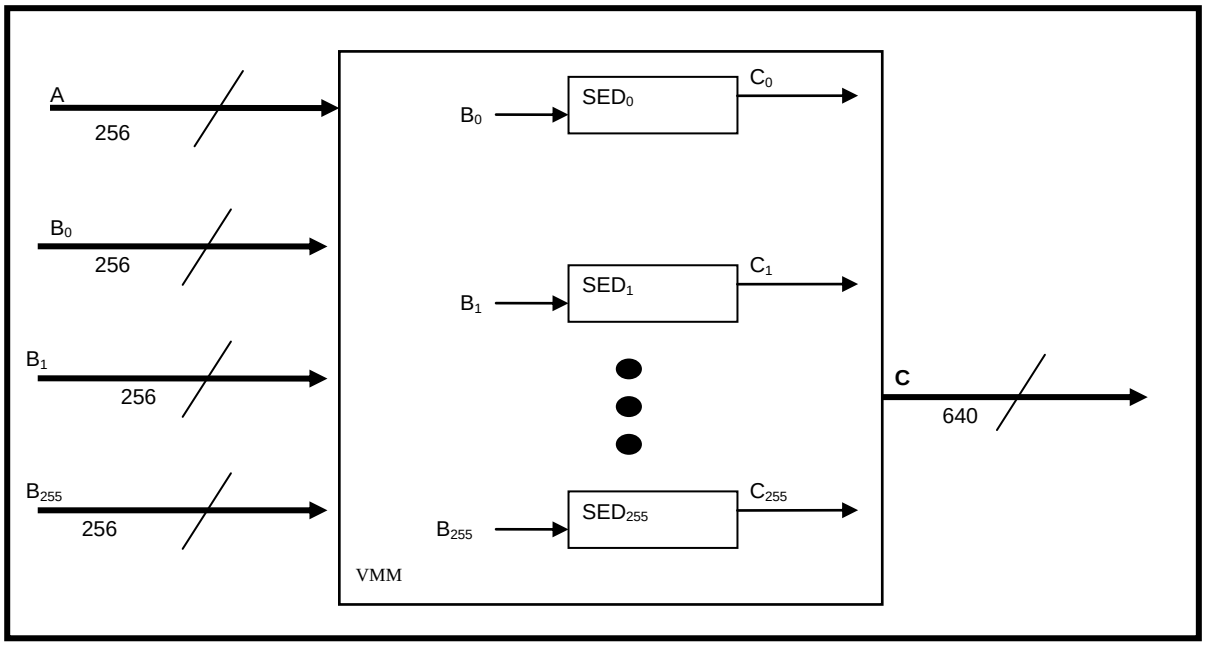

Fig. 2. The VMM electrical driver.

The VMM electrical driver is comprised of *256* units. It is referred to as 'single electrical driver' (SED). The '*A*' input of the VMM drives (see Fig. 1) is the '*row source array*'. The 'B_i' input of the VMM is driving '*SLM-row-j*', which is a part of SED unit '*j*' (SED_j). The VMM output vector '*C*' is an aggregation of the set of scalar outputs ('C_0' to 'C_{255}') emerging from each SED.

Current VCSEL technology operates at a peak rate of *125* MHz with *256* levels of intensity. That is, a resolution of *8* bit per laser light source. The VMM input '*A*' enables this rate using a single *256* bit bus operating at *1.0* GHz. On the other hand, the SLM can be updated at a higher rate.

Figure 3 describes the layout of SED_j. The input and output signals to this unit are electrical. The input '*B*' consists of a single *256* bit bus, which drives an SLM containing a row of *256* values (SLM_j). This single SLM is actually the row '*j*' of the entire SLM matrix. The output 'c_j' is a single *20* bit bus, which is yielded by the output detector array.

The input vector 'B_j' can be stored in an internal dual-port modular memory buffer before being directed to the SLM. The output 'c_j' is directed to the external output, where the *256* outputs from the *256* SED units form one vector '*C*' (*1×256×20* bits).

The configuration depicted in Fig. 3 supports a dot-product operation between the input row vector '*A*' (being converted into light by the VCELs) and one column of the entire SLM matrix. Each SED performs one vector dot-product operations per cycle of 8 ns (the reciprocal of *125* MHz). Combined together, the *256* units perform a vector (*1×256×8* bits) by matrix (*256×256×8* bits) multiplication operation at a rate of *125* MHZ.

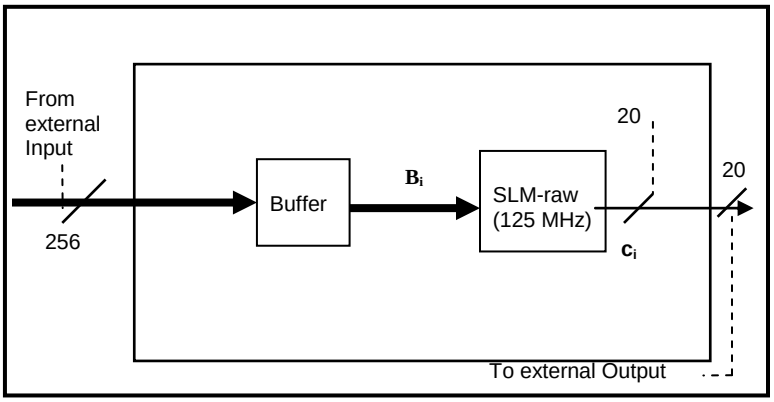

Fig. 3. A 1×256 SLM electrical driver.

Let $\overline{A}$ and $\overline{B}$ represent the *256×8* vectors corresponding to the '*A*' and '*B*' inputs described above. In addition, let 'x_i' denotes the i^{th} element of the vector $\overline{X}$. Then, SED_j enables the operation $c_j = \sum_{i=1}^{256} a_i \times b_i$ on the two vectors, where 'c_j' is a *20* bit scalar. To enable interaction with the SED, the following memory operations (per cycle) are available.

Read

a. Read a vector (*1×256×8* bits) from external input into the buffer

b. Read a vector (*1×256×8* bits) from external input into SLM_j.

c. Read a vector (*1×256×8* bits) from buffer into SLM_j.

Write

d. Write a scalar (*20* bit) from SED_j to the external data source.

3. Performance Analysis of Possible Applications

Each of the units depicted in Fig. 3 is capable of executing *256×125* million multiply-accumulate instructions per second. This is equivalent to *64* Giga integer operations per second (GIPS). The entire VMM system consists of *256* units that are the same as the one unit depicted in Fig. 3. Hence, it can perform in peak performance of *16384* GIPS. The following sections analyze the performance of the unit with respect to DSP procedures and applications.

3.1 DSP Performance Analysis

The proposed VMM can serve as a co-processor attached to a DSP or a RISC CPU, referred to as the controller. In addition, the controller can utilize other co-processors. The entire system (controller, co-processors and VMM) is depicted in Fig. 4. The figure shows an example where one of the additional co-processors is capable of shuffling vector elements. This may be useful for implementing convolution and correlation. It is

assumed that in the typical mode of operation, the controller, along with other co-processors, prepares an input data vector and a matrix to be sent to the VMM for processing. We refer to this as pre-processing. The output of the VMM is sent back to the controller where it may go through additional processing (post-processing) before being sent to other devices or back to the VMM. Sound pre-processing and post-processing operations can reduce the amount of VMM operations. For example, reusing the same input vector as much as possible while altering only [some of] the matrix values can reduce communication between the controller and the VMM and enable the VMM to operate at the peak rate of *>125* MHz. As implied, we are making the reasonable assumption that the controller and or other dedicated co-processors (e.g., a vector-permute co-processor) enable the controller to sustain the rate required for pre and / or post processing.

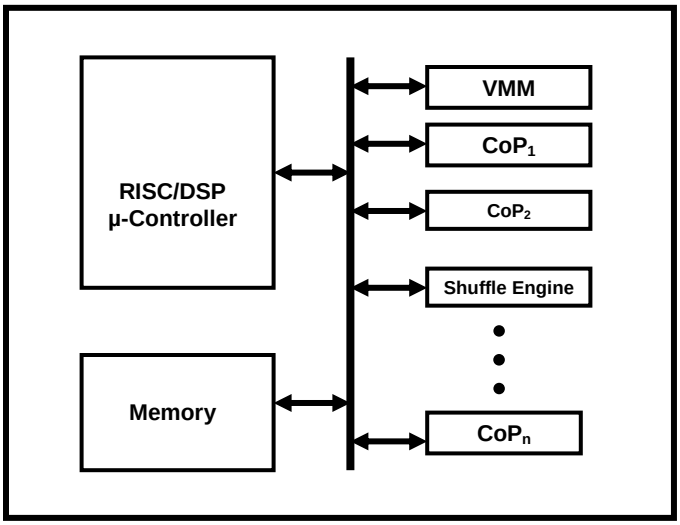

Fig 4. System Architecture

The VMM is capable of performing one generic operation. That is, a vector (1x256x8 bit) by matrix (*256×256×8*) multiplication. Nevertheless, special cases of vector-by-matrix multiplication can be used as the building blocks for numerous DSP procedures. For example, an *L2* norm can be implemented as a special case of vector-by-matrix multiplication.

This section discusses several basic DSP procedures that can be implemented as special cases of vector-by-matrix multiplication. We explain the required pre and post processing and discuss the features (speed, accuracy, etc.) of the DSP / computer graphics (CG) procedure enabled. The following DSP procedures are analyzed:

- Vector-by-matrix multiplication
    - General vector-by-matrix operations
    - Vector-by-matrix with a large number of elements (more than *256* or *256×256* elements).
    - Vector-by-matrix with a small number of elements (less than *256* or *256×256* elements).
- Matrix by matrix multiplication
    - General matrix-by-matrix operations
    - Matrix-by-matrix multiplication with a large number of elements.
    - Matrix-by-matrix multiplication with a small number of elements.
- Dot-product operations
    - Using dot-product to enable convolution, correlation, and transforms (e.g. DFT and DCT)
    - Using dot-product to perform *L2* norm operations.
- Addition, subtraction, complex, and extended-precision operations.

We also discuss applications that can use these DSP building blocks. In specific, we illustrate the usage of vector-by-matrix multiplication to implement a geometrical transformation on a large set of vertices. In addition, we exemplify the use of VMM implementation of *L2* norm for motion estimation and implementing a rake receiver.

3.1.1 Vector-by-Matrix Multiplication

Vector-by-matrix multiplication is an essential operation in several DSP applications such as beam forming, radar signal processing, and multi user detection [6]. The proposed system can perform multiplication of a *1×256×8* bits vector by a *256×256×8*

bits matrix at a rate of *125* million vector-by-matrix multiplications per second. If the matrix and vector are smaller than *1×256* and *256×256*, respectively, then it may be possible to achieve the same rate with multiple small matrices / vectors. This is demonstrated with respect to matrix-by-matrix multiplication.

3.1.2 Matrix-by-Matrix Multiplication

The proposed system can complete the multiplication of *256×256×8* bits matrix by a *256×256×8* bits matrix at a rate of *0.48* MHz ($125/256$ MHz). If the matrix / vector have more than *256* (or *256×256*) then there is an overhead related to decomposing the vector (matrix) into sub vectors (matrices). In most of the cases, this overhead is negligible.

If the matrix (vector) have less than *256* (or *256×256*) elements, then it may be possible to perform pre-processing operations and achieve better performance per vector / matrix. To elaborate, consider implementing two *4×4* matrix multiplications using one *8×8* matrix:

Let '*A*', '*B*', '*C*', and '*D*' be four *4×4* matrices. Then the following constellation can be used to implement the operations $A \times B$ and $C \times D$

$$\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \times \begin{bmatrix} C \\ D \end{bmatrix} = \begin{bmatrix} A \times C \\ B \times D \end{bmatrix} \quad (1)$$

Given the above constellation, the multiplication of 64 4x4 pairs of matrices can be implemented by multiplying a matrix of *4×256* by a matrix of *256×256*. Thus, the peak rate for this essential computer graphics operation is: $64 \times 125/4$ million matrix multiplications per second (a rate of *2* billion matrix multiplications per second). This is an extremely high rate. Nevertheless, it involves data elements of low resolution (*8* bits). We discuss higher precision operations in section 3.1.4.

3.1.3 Dot-Product Operation

Dot-product is the basic operation of a single SLM-SED unit. It is an essential part of convolution, correlation, transforms, and implementation of *L2* norms.

Consider SLM$_j$. Let $A = \{a_i\}_{i=1}^{256}$ be the input raw vector and let $B = \{b_i\}_{i=1}^{256}$ be raw ***'j'*** of the SLM matrix, then the output of the SLM is $c_j = \sum_{i=1}^{256} a_i \times b_i$. That is, each of the SLM units performs a vector dot-product on two *256* elements vectors. The peak rate for the dot-product operation is *125* MHz per SLM.

As in the instance of matrix-by-vector and matrix-by-matrix multiplication we can consider three cases. 1) Dot-product on vectors with *256* elements, 2) vectors with more than *256* elements, and 3) vectors with less than *256* elements.

The dot-product of vectors with *256* elements is performed at a peak rate of *125* MHz per SLM. With *256* SLM units, the system can perform *256* dot-product operations (on vectors of 256 elements) in parallel. This is equivalent to a rate *256×125* million or 32 billion! completions of dot products per second.

Dot-product of vectors with more than *256* elements requires decomposition of the vectors. In most of the cases, the decomposition overhead is negligible (less than *1*%)

Dot-product of vectors with less than *256* elements can be done effectively in several ways. For example consider the dot-product of two *1×4* vectors '*A*' and '*B*' with the two *1×4* vectors '*C'*, and '*D'*. This product can be implemented using the following vector-by-matrix multiplication:

$$\begin{bmatrix} C & 0 \\ 0 & D \end{bmatrix} \times \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} A \bullet C \\ B \bullet D \end{bmatrix} \quad (2)$$

Please note that the matrix is a *1×4* matrix and the vector is an *8×1* vector. To put the above discussion in context, consider the dot-product of two signals each of which contains *40* samples. This is one of the Berkley Design Technology Inc. (BDTI) DSP benchmarks [7]. In this case, we can populate the input row vector with *6* sub-vectors of *40* elements each, and store one sub vector of *40* elements per SLM unit. The end result is a peak rate of *6×125* MHz. In other words, the VMM can complete *750* million dot products of 40 samples per second. This is a 25 times speedup compared to the best published result of *40* element dot-product achieved on the TI-64xx [8].

3.1.3.1 Convolution and Correlation

Convolution (and correlation) between a vector '*X*' and a vector '*S*' can be implemented through dot-product of '*X*' with shifted versions of '*S*'. The following equation illustrates the 4 samples result (*Y={y0..y3}*) of convolution of a *7* samples signal *S={s0..s6}* with a *4* samples signal *X={x0..x3}:*

$$\begin{bmatrix} s0 & s1 & s2 & s3 \\ s1 & s2 & s3 & s4 \\ s2 & s3 & s4 & s5 \\ s3 & s4 & s5 & s6 \end{bmatrix} \times \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix} \quad (3)$$

Hence, in this case it is assumed that the controller (or a co-processor) is generating the shifted values of the signal '*S*'. The VMM can perform a convolution between a signal (mask) of *256* samples and another signal of arbitrary length at a peak rate of *256×125* MHz (32 GHz). A mask with more than *256* elements requires a negligible decomposition overhead. A mask with less than *256* elements can be handled at the same rate. Alternatively, as illustrated in the following equation, several different filters can be implemented in parallel:

$$\begin{bmatrix} s0 & s1 & 0 & 0 \\ s1 & s2 & 0 & 0 \\ 0 & 0 & s0 & s1 \\ 0 & 0 & s1 & s2 \end{bmatrix} \times \begin{bmatrix} x0 \\ x1 \\ z0 \\ z1 \end{bmatrix} = \begin{bmatrix} v0 \\ v1 \\ u0 \\ u1 \end{bmatrix} \quad (4)$$

In this above example, two masks (*X={x0,x1}* and *Z={z0,z1}*) are implemented simultaneously on the signal *S={s0,..,s2}*. The outputs are *V={v0,v1}* and *U={u0,u1}*.

We consider another BDTI DSP benchmark which calls for implementing a *16* tap FIR filter on *40* samples of a given signal [7]. The VMM can sustain a rate of *125* MHz per SLM in this benchmark. Moreover, The VMM can implement *6* different 16 taps / 40 samples FIR filters simultaneously each of this filters completes *125* million filters per second (a speedup of 100 over the TI C64xx) [8].

The fact that the SED contains a buffer with more than 256 bytes (e.g., 2048 bytes) can be used to increase the performance of the VMM. The SLM transistor can be updated at very high rates (>3G Hz) that are well above the rate of updating the input row vector (125 MHz). This can be exploited by storing several masks in the SED buffer and using

them to update the SLM at a rate that is above 125 MHz, while holding the input at a rate of 125 GHz. We plan to address this in future investigation of the VMM implementations.

3.1.3.2 DFT / DCT

The discrete direct Fourier transform (DFT) can be implemented through vector-by-matrix multiplication. That is, via a set of complex dot-products of a [complex] signal with the complex Fourier transform coefficients. Each dot-product represents the DFT at a given frequency. We discuss potential representations of complex data in section [3.1.4.2] and revisit the DFT at that section. On the other hand, the *1-D* DCT can be implemented as a real vector by a real matrix multiplication.

The VMM can complete a *256* points DCT at a rate of *125* MHz (*125* million transforms per second). In addition, the VMM can perform *16* times *2-D* DCT of a *16×16* signal at a rate of *62.5* million transforms per second. Hence an equivalent rate of *1* billion *16x16* DCT per second. This is implemented by running DCT on the rows (*16* rows of *16* elements) through a dot-product of each row with a *16×16* coefficients matrix and then running the resultant 16 vectors through a column by column DCT. In this case, copies of the coefficients matrix of *16×16* elements are stored in the SLM, and the input data is stored in the 'row input vector'.

An *8×8* DCT can be implemented at a peak rate of 4 billion transforms per second.

3.1.3.3 *L2* Norm Operations

The controller can enable *L2* norm operations through pre-processing (vector subtraction) of input vectors. Alternatively, the pre-processing can be carried by the VMM. This is further discussed in section 3.1.4.1.

Consider the vectors $A, B,$ and $C;$ where $C = A - B.$ Then $C \bullet C$ is the L_2 norm of A and $B.$ Assuming that the vector subtraction is done by the controller, then the VMM can perform the *L2* norm of an input vector with a set of *256* other vectors at a rate of *125* MHz per SLM (total of *32* billion norms per second).

The *L2* norm, as well as cross / auto correlation operations can be used for pattern matching and to calculate a distance (or similarity) measure for pattern recognition and data compression applications. We discuss the use of cross correlation for motion estimation in section 4.1.

3.1.4 Addition, Subtraction, Complex Numbers, and Extended-Precision Arithmetic

Along with pre / post processing by the controller, the VMM can be used for addition, and subtraction. Furthermore, the VMM can operate on complex vectors. But, the controller has to perform adjustments. Finally, the VMM can generate partial products which the controller can utilize as a part of higher precision operations.

3.1.4.1 Addition and Subtraction and Multiplication with Integers and Real Numbers.

Internally, the row input matrix, as well as, the SLM values represent unsigned *8* bit integers. Nevertheless, the controller can perform operations such as DC subtraction. This can convert the actual representation to signed representation. Scaling can be used to provide fixed point representation. Trivially, a dot product of an arbitrary vector '*U*' with a vector of the form *V={1,1,..,1}* is equivalent to summation of the element of '*U*' and the dot product of a vector *U={u,0,..,0}* with the vector *V={v,0,..,0}* yields $u \times v$. Hence, the VMM is capable of performing addition and multiplication operations. Subtraction, however, requires conversion to 2's complement by the controller.

3.1.4.2 Complex Number Representation, Complex Dot-Product, and DFT

Complex numbers can be represented as a vector with alternating real and imaginary values. Alternatively, complex numbers can be stored as two vectors; the first includes real values and the second includes the imaginary values. Both may require shuffling and addition or subtraction as a part of the pre or the post processing. The second approach requires fewer manipulations by the controller. It is further elaborated here.

Consider the complex vectors '*A*' and '*B*' with the elements $a_i = u_i + jv_i$, and $b_i = x_i + jy_i$, respectively. Let $U = \{u_i\}_{i=1}^{256}$, $V = \{v_i\}_{i=1}^{256}$, $X = \{x_i\}_{i=1}^{256}$, and $Y = \{y_i\}_{i=1}^{256}$. Then

$A \bullet B$ can be obtained from the dot products $[U \bullet X - V \bullet Y]$ and $[U \bullet Y + V \bullet X]$. Therefore, four dot products per element along with addition / subtraction by the controller are required. As a consequence, the VMM can perform one complex dot-product of *2* vectors of *256*-complex elements at a rate of *31.25* million complex dot products per second (two cycles for the real part and two for the imaginary).

Hence, the VMM can complete a complex vector-by-matrix (*1×256* by *256×256*) at peak rate of *31.25* MHz. Furthermore, it can compute complex matrix-by-matrix multiplication (*256×256* by 256×256) at a rate of *0.12* MHz. Finally, the VMM can complete a complex *256* point DFT at a rate of *31.25* MHz. That is, the VMM can perform 31.25 million complex 256 point DFT per second.

This is a case where the ability to load the SLM at a rate that is faster than *125* MHz (the current limit for updating the raw source array) could improve performance. It is possible to load U into the raw source array and perform the operation: $U \bullet X$ followed by the operation $U \bullet Y$. Where X and Y are loaded to the SLM elements at a rate of *250* MHz. Then, load V to the raw source array and perform the operations $V \bullet Y$ and $V \bullet X$. We plan to consider the utility of this mode of operation, where the SLM is loaded at a rate that is higher than the rate of loading the raw source array, in future generations of the VMM.

4. VMM "Killer" Applications

The DSP procedures listed above can be used as building blocks for several "killer" applications. Goren *et al* [5] have elaborated on wireless related processing such as UTMS rake receiver and multi-user detection. Their proposed VMM, however, is at least two orders of magnitude slower than the current proposed VMM.

We analyze the performance of the system as a component in video compression where motion estimation using *L2* is performed by the VMM. In addition, we evaluate the performance of the system as a part of a *3-D* geometry engine performing transformation on triangles.

4.1 The VMM as a Motion Estimation Engine

The VMM can be used to implement MPEG / H.264 like motion estimation (ME). The VMM ME algorithm can be implemented through *L2* norm or through cross correlation. Both methods yield the same rate but may require different pre / post processing. We consider cross correlation here.

Consider a current-frame macro-block of *16×16* pixels stored in the input row vector and a search window of *32×48* pixels (a slice of *32* rows and *48* columns). The VMM can be used to implement exhaustive search in the following way:

The controller loads the SLM matrix with consecutive macro-blocks from the search window and the VMM implement cross correlation with 256 blocks in each cycle. The entire *32×48* window contains *1536* instances of overlapping macro-block. Hence, it can be loaded into the SLM in 6 cycles of operation or at a rate of $125/6 = 20.8\overline{3}$ MHz. This is the rate at which the VMM can complete the search. The controller has to find the minimum of the cross correlation function calculated by the VMM. Some variants of the above analysis can be of interest. For example, if an "informed" search such as multi-resolution (pyramid) search is implemented, then each stage can use *256* macro-blocks from the search window. For example, a *3* stage search in a *32×48* window at *1/4* pixel resolution can be accomplished at *25* MHz per macro-block. In this implementation, the first stage is a *1/2* sub-sample of the search window. It yields an effective initial search window of *16×32* pixels or *512* overlapping macro-blocks. It requires two cycles to identify the minimum value for this window. Then, one cycle is consumed for full pixel resolution around the maximum, and another cycle for half pixel resolution. Finally, an additional cycle is required for quarter pixel resolution. The total number of cycles (5) is processed at a rate $125/5 = 25$ MHz. A 1/8 resolution will require one more cycle and can be accomplished in $125/6 = 20.8\overline{3}$ MHz

4.2 The VMM in a Geometry Engine System

The VMM can complete *2* billion multiplications of 4x4 matrices per second (see section 3.1.2). Hence it can compute the transformation of *2* billion *3-D* vertices per second; or

$666.\overline{6}$ million triangles per second. The vertices, however, are represented by low resolution *8* bit elements. To support *32* bit floating point operations, i.e., operations with *24* bit mantissa, the VMM has to enable *24* bit multiplication. This can be achieved by generating *9* partial products yielding a rate of *74* million triangles per second. In this case, however, the controller is expected to do a "lot" of pre-processing and post processing (e.g., shift and add of partial products). This may require another dedicated co-processor. Furthermore, each *8*-bit by *8*-bit multiplication must produce no errors, since these will appear high in the *24*-bit result. Thus, the VMM should be capable of generating an exact *16* bit result. This can be accomplished by adding guard bits to the VMM multiply unit.

4.3 NP-Complete Problem Optical Solver

Many researchers have tried to find efficient solutions to NP-complete combinatorial problems. In spite of this, none of them has succeeded in finding an efficient polynomial-time solution to these problems. Due to the difficulty of solving these problems, many approximation and heuristic methods have been proposed in the literature. However, approximation methods tend to prefer good solutions found quickly rather than the best one which may not be found within reasonable time. On the other hand, heuristic methods are able to solve these problems quickly for certain cases only. In fact, the computation time of the heuristic methods may be unexpected and even longer than that of an exhaustive search (checking all possibilities in an exhaustive manner) due to unsuccessful attempts for optimization. Therefore, in applications where deadlines must be met, such heuristic methods are not a good choice and one may prefer to use an exhaustive search. For these cases, a new optical method which can provide a dramatically better guarantied time for the solution is proposed in Ref. [XX].

The proposed optical system is capable of solving NP-complete problems such as the traveling salesman problem (TSP), the Hamiltonian path problem (HPP), etc. by checking all feasible possibilities more efficiently than a conventional computer. This design is based on a fast optical vector-matrix multiplication between a binary-matrix representing all feasible solutions and a weight-vector representing the problem weights. The multiplication product is a vector representing the final solutions of the problem. In the

TSP for example, where the required solution is the best (shortest) tour connecting a certain set of given node coordinates, the multiplication is performed between a binary-matrix representing all feasible tours among the TSP nodes and a grayscale vector-weight representing the weights between the TSP nodes. The multiplication product is a length-vector representing the TSP tour-lengths by peaks of light with different intensities. Finding the shortest tour can be performed by using an optical polynomial-time binary-search which untilizes an optical threshold plate. On the other hand in the HPP, a decision whether there is a path connecting two nodes on the HPP graph is required (which implies that part of the edges may be blocked). In the HPP, the binary-matrix still represents all feasible paths (tours) but the weight-vector in this problem is also binary. After the vector-matrix multiplication, any peak of light obtained in the output of the optical system means that a Hamiltonian path exists.

The advantage of the proposed method is that once the binary-matrix is synthesized, all same-kind problems (TSP, HPP, etc) of the same order (with the same number of nodes) can be solved optically by only changing the weight-vector and performing the vector-matrix multiplication in an optical way. This optical vector-matrix multiplication can be performed by several methods such as the Stanford multiplier and various correlation methods [4]. The input vector (the weight-vector of the problem) is represented by sources of light. The matrix is assumed to be represented by a regular slide since, as mentioned before, a single matrix is suitable for all same-kind problems (TSP, HPP, etc) of the same order (with the same number of nodes) and in addition a pretty large number of nodes (around 16) can be represented on a single slide. For the calculation of the real-time performances of the system, we assume that the binary-matrix representing the required number of nodes has already been synthesized earlier. This means that we assume the existence of a "special hardware" for the task and only the input vector has to be changed per a problem. This is a reasonable assumption since we also provide an efficient algorithm for the synthesis of this matrix (to be generally discussed next).

The physical size of the optical system depends on the size of the binary-matrix (which is dependent on the number of nodes in the problem). The minimum pixel size in a general optical system is $\lambda/(2\cdot NA)$, where NA is the numerical aperture of the optical system. A

realistic value for the minimum pixel size in our implementation is $1\mu\text{m}^2$. Therefore, in an optical aperture of $2.14\text{m}\times 2.14\text{m}$ there are $2.14\times 10^6 \cdot 2.14\times 10^6 = 4.58\times 10^{12}$ pixels. Assuming that one pixel is enough to represent a single element in the binary-matrix and the binary-matrix can be cut and chained in a form of a rectangular binary-matrix, then after preparing the required filter for the VLC with the size of $2.14\text{m}\times 2.14\text{m}$ once, it is possible to solve all problems of 15 nodes within one optical cycle (since the number of elements in the binary-matrix of a 15-node probelm is $(14\cdot 15/2)\cdot(14!/2) = 4.58\times 10^{12}$) plus the time it takes to update the weight vector of the size of $N(N-1)/2 = 15\cdot 14/2 = 210$ on a real-time SLM. The time which this optical cycle consumes depends on the speed of light and on the distance the light passes through the VLC. We can evaluate this time for an aperture of $a\times a$ as $4f/c \approx 2a/(NA\cdot c)$, where f is the focal length of the lenses, c is the speed of light and the approximation $NA \approx a/(2f)$ is used. In the case of having $NA = 0.3$ and $a = 2.14\text{m}$, the optical cycle time is about 4.75×10^{-8} seconds plus the time it take to update the weight vector of the problem, only containing $N(N-1) = 15\cdot 14 = 210$ elements, which is 8×10^{-9} seconds. This mean a total calculation time of 5.55×10^{-8} seconds. For comparison, even assuming that a conventional electronic processor (working in the gigahertz regime) can check 10^9 tours per second, it takes $1/10^9 \cdot 14!/2 = 43.59$ seconds to check all possible tours for a 15-node problem with this computer. Comparison between the computation time of the proposed optical processor and that of the conventional electronic processor yields that the optical processor is faster than the electronic one in a factor of $43.59/(5.54\times 10^{-8}) = 7.87\times 10^8$. Similar calculations have been performed for various TSPs (or HPPs), starting from 3 nodes and reaching 16 nodes. Fig. XX (WILL BE ADDED BY NATI) shows the computation times of the optical and the electronic processors. As seen from this figure, the computation time of the optical processor is always much faster than that of the electronic processor. In addition, the computation time of the electronic processor increases at a faster rate than that of the optical processor. This is the reason for the steeper slope of the electronic processor computation time graph shown in a Figure.

In the Figure, we have also provided a new efficient algorithm for synthesizing the binary-matrix so that it contains all the feasible tours and only them. One advantage of this algorithm is that it synthesizes a binary-matrix of N nodes which also contains the binary-matrices of less than N nodes. Another advantage of this algorithm is that it uses a relatively small number of iterations in order to synthesize the binary-matrix. This binary-matrix contains a huge number of rows ($(N-1)!$), each row represents a different feasible tour and a relatively small number of columns ($N(N-1)$), each column represents an edge related to one of the problem weights. Taking into account the huge number of rows, and the fact that conventional computers cannot duplicate such huge vectors (if the number of nodes N is large), the optical system we have proposed in Ref. [XX] is a useful tool for synthesizing the binary-matrix. According to this optical system design, the binary-matrix long columns are duplicated one after the other by performing a correlation operation with shifted delta point functions for which the shifts are given by the binary-matrix algorithm.

In the optical implementation of the binary-matrix algorithm, the binary-matrix can be represented by a slide containing rectangles, in which a white (transparent) rectangle represents a one in the binary-matrix and a black rectangle represents a zero in the binary-matrix. The input of the binary-matrix algorithm is the binary-matrix of 3 nodes, the rows of which represent 2 tours. The output of the algorithm is the binary-matrix of N nodes, the row of which represent $(N-1)!$ tours. Preparing the initial-slide, containing the binary-matrix of 3 nodes, is quite easy since the binary-matrix of 3 nodes contains only 2 rows and 6 columns. However, as the number of the nodes increases, the number of rows in the binary-matrix increases exponentially (and the number of columns increases quadratically). In fact, when reaching a binary-matrix of 16 nodes for example, the number of rows in the binary-matrix is $(N-1)!=15!=1.3\times10^{12}$ and it may take relatively long time and large memory to duplicate binary-matrix columns containing 1.3×10^{12} elements each. On the other hand, duplications of such big vectors can be performed optically by using optical correlators such as the VanderLugt correlator (VLC) explained in subsection 3.2. By the VLC, the duplications can be done by a correlation with shifted 2D delta point functions (dots of

light), where the shifts are computed by the binary-matrix algorithm. As shown in Fig. 18, a slide containing the binary-matrix of N nodes (in the figure: $N=4$) is used as the input plane of the VLC. A vertical slit selects each time only a single column. This column is Fourier transformed by the first spherical lens and the resulting Fourier transform is multiplied by the filter plane, usually represented on an SLM. After the Fourier transform of the multiplication product by the second spherical lens, the output plane contains the correlation of the source-matrix column and the shifted delta functions. By doing this, we actually perform one or more duplications (depends on the VLC filter used) of the source-matrix column into one or more locations given by the binary-matrix algorithm. A light-sensitive film is positioned in the output plane, so in each duplication, different locations on this film are exposed to light. After the VLC finishes duplicating all columns of the binary-matrix representing N nodes (the source-matrix) into the suitable locations, the binary-matrix representing $(N+1)$ nodes (the target-matrix) appears in one of the diffraction orders of the output plane. The plane containing this target-matrix is used as the input plane of the next stage and the process described above starts all over again in order to yield the binary-matrix of $(N+2)$ nodes. This continues till the binary-matrix with the required number of nodes is produced. The synthesis of the binary matrix is a feasible task since only less than N^3 correlator iterations are needed. This is an excellent complexity for a problem which is usually dependent on $N!$. As soon as this binary-matrix is produced, it can be stored into the proposed device memory and then uploaded in a high rate to the SLM in parts in order to perform different portions of the vector by matrix multiplication.

5. Implementation of the VMM Electrical Driver

Figures 5, 6, and 7 illustrate the components of the VMM electrical driver. These figures include several implementation details demonstrating that it is feasible to construct the system using existing mass production VLSI technology.

The driver consists of *256* units of *1×256×8* bit. Each unit is referred to as a single electrical driver (SED). SED devices are integrated into ALU chips, which are placed on an interface board. Fig. 5 adds more details to Fig. 1 and illustrates the SED unit. A set of SED units integrated into one chip is referred to as an ALU element (see Fig. 6). The

interface board contains several ALU chips (see Fig. 7). Overall, the interface board contains *256* SED units. There is a design flexibility with respect to the number of SED units per ALU element. This number dictates the required number of ALU chips in the interface board. Given current technology we have decided to investigate a configuration with *16* ALU chips, each of which contains *16* SED devices. This configuration is further analyzed in the current appendix.

5.1 The SED Unit

Each SED contains a relatively small dual-port modular memory of several thousands bytes (say *2048* bytes) and a *256* element SLM (i.e., *256* SLM transistors). The SLM element of SED_j (depicted in Fig. 5) represents row '*j*' of the SLM matrix. The input vector '*B*' can be stored in the internal buffer before being directed to the SLM. The buffer drives the SLM either directly with the input B_j or with a set of 256 bytes that has been previously stored in the buffer ($\hat{B}_j$). The output 'c_j' is a single *20* bit bus which is wired to the output detector array. In the implementation considered in this paper, we assume that the input B_j consists of a 256 bits bus that operates at 1 GHz. Hence, it loads the buffer at a rate of *125* M byte / second.

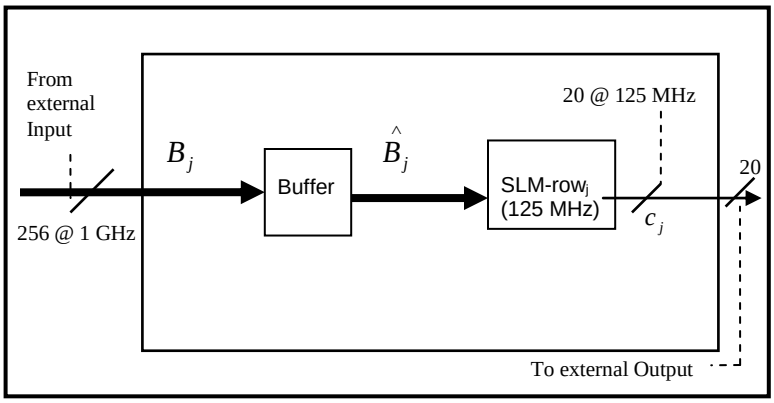

Fig. 5. The SED unit.

5.2 The ALU Chip

Several SED units are integrated into one chip. The chip, referred to as the ALU element, is depicted in Fig. 6. In the implementation investigated in this paper, we assume that *8* SED units are integrated into a single ALU element. This is a reasonable assumption as the estimated number of transistors in each SED is less than *100000* (assuming a *2048x8* bit buffer implemented as an SRAM organized as a FIFO; with 6 transistors per bit this totals a little under 100K transistors). In addition, the number of internal connections within the SED is small.

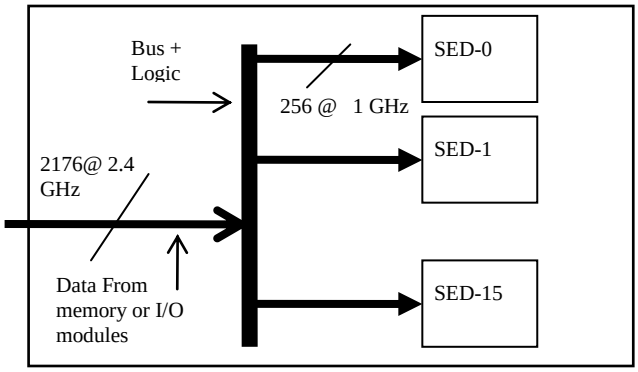

Fig. 6 – The ALU element

An ALU element is connected to an external memory or I/O units through an external bus. An internal bus plus logic distributes the data to the eight SED units inside the ALU element. Each unit gets 256 bits at a rate of 1 GHz, hence a rate of 125 M Byte per second. This means that the entire SLM matrix is updated at the same rate as the input row vector (125 MHz).

Theoretically a bus of *2048* bits operating at *2* GHZ is needed in order to supply *256* bits of data to each SED and sustain a rate of *125* MHz. Practically, we use an internal bus of *2176* bits, which operates at *2.4* GHz, to drive the individual SED units. The reason for

using a bus of *2176* bits is due to the need to synchronize the input data. It is assumed that each set of *16* data lines are synchronized through one synch line. Hence *2048* data lines require *128* synch lines and the total number of input lines is *2176*. In addition, it is assumed that *20%* redundancy in the number of input bits is required in order to supply error correction and handshaking mechanism. This is achieved by raising the frequency of the bus to 2.4 GHz. While this is a relatively wide bus operating at a relatively high frequency, it is well supported by current 0.65nm CMOS technology, where chips such as the IBM Cell contain thousands of pins, thousands of bus lines, and operate at rates that are that above *3* GHz [9]. It is also in line with the ITRS 2005 / 2006 reports [10]. Furthermore, this bus is not a true multi-drop bus; rather, it is implemented as a number of narrower point-to-point interconnects (driving a very wide bus quickly and supporting many sinks would be a design challenge).

The SED units within an ALU chip are synchronized, where each SED operates at a rate of *1* GHz. We discuss the issue of synchronizing the SED units within an ALU element in section 5.4.

The proposed device requires a package with several thousand input pins. This is within the state of the art, as indicated by International Electronics Manufacturing Initiative (INEMI) 2005 Packaging Roadmap Overview, suggesting that packages with over 3000 connections will be available by 2008 (see Table 1 below) [15].

The proposed device requires fairly high input signaling rates. The technology behind high-performance serial interconnects such as PCI Express and Rapid IO are driving very high frequency source-synchronous signaling schemes with rates in the 4-6 GHz region. However, these are intended to be deployed with relatively few connections, and so their power dissipation may be a concern. The proposed VMM electrical interface chip, however, is input-only and so the issues associated with driving a large number of high-frequency signals can be relegated to the system to which the proposed chip is to be connected.

Year of Production	*2004*	*2005*	*2006*	*2007*	*2008*	*2009*
Cost per Pin Minimum for Contract Assembly [1,2] (Cents/Pin)						
Low-cost, hand-held and memory	0.30–.53	.27–.50	.26–.48	.25–.45	.23–.43	.22–.41
Cost-performance	.71–1.24	.67–1.17	.64–1.11	.61–1.05	.58–1.00	.55–.96
High-performance	1.88	1.78	1.69	1.61	1.52	1.45
Harsh	0.32–2.88	0.29–2.60	0.26–2.33	0.25–2.11	0.23–2.00	0.22–1.90
Chip Size (mm^2) [3]						
Low-cost	100	100	100	100	100	100
Cost-performance	140	140	140	140	140	140
High-performance	310	310	310	310	310	310
Harsh	100	100	100	100	100	100
Maximum Power ($Watts/mm^2$) [4]						
Low-cost (Watts) [5]	2.7	2.8	3	3	3	3
Cost-performance	0.6	0.65	0.7	0.74	0.79	0.83
High-performance	0.51	0.54	0.58	0.61	0.64	0.64
Harsh	0.16	0.16	0.18	0.18	0.2	0.2
Package Pincount Maximum [6][7]						
Low-cost	122–500	134–550	144–600	160–660	180–720	180–800
Cost-performance	500–1600	550–1760	550–1936	600–2140	600–2400	660–2800
High-performance	3000	3400	3800	4000	4400	4600
Harsh	500	550	600	660	720	780

Within the devices, several sources distribute gigahertz-rate signals to several destinations. This is also within the capabilities of current technology. As an example, the Tilera TILE 64 implements 64 processors in a 90nm CMOS technology [13]; each processor connects to five point-to-point networks implemented as unidirectional 32-bit interconnects which provide an aggregate bandwidth of 1.28 terabits per second. This bandwidth is provided by 320 bits (lines). Hence, each wire in this older CMOS technology is providing 1280/320, or 4, G bits/ second. Furthermore, Intel has described a 5GHz on-chip network [14].

5.3 The Interface Board

The interface board depicted in Fig. 7 contains *16* ALU chips. In addition, the interface board contains lines that enable transfer of external inputs to the ALU elements. Each ALU element is fed with data independently of the other ALU elements. While there are no internal connections or dependencies between the ALU elements, they work in a source-synchronous mode, where each ALU element is operating at a rate of 2.4 GHz.

The interface board is comparable in complexity to a small size Infiniband board. In fact, in some applications, it is conceivable that the ALU elements occupy more than one interface board.

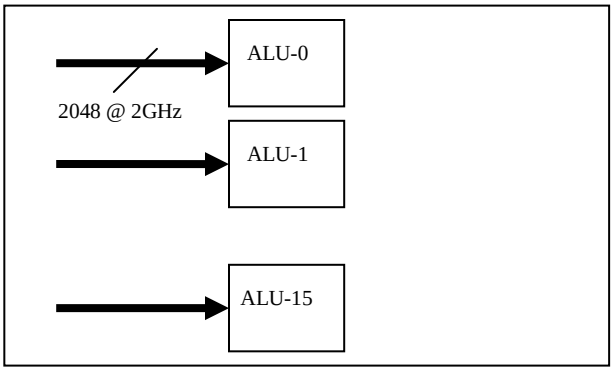

Fig. 7 – The interface board

In these cases, the system is equivalent in complexity to several system-on-chip (SOC) units connected to an Infiniband board. In addition, since there are no dependencies / communication transactions between ALU elements, then the interface board does not require a fully-connected switch. The conclusion from the above discussion is that the level of complexity of the ALU board is moderate. It can be implemented with current of-the-shelve VLSI units.

5.4 System Synchronization

The entire set of *256* SLM elements has to work in a synchronous mode and be synchronized with the conversion of electricity to light at the row input vector. Nevertheless, given the relatively low frequency of the source array, this synchronization is of modest complexity. We consider two synchronization tasks. First, the SED units within an ALU element have to be synchronized. Second, the ALU elements within an ALU board have to be synchronized.

Synchronization of SED units within the ALU chip is relatively straightforward. The SED units are operating at *1* GHZ. Each SED contains small amount of logic and the data path within the SED is short (*4* transistors in serial one of which is an SLM transistor). This is well below the characteristics of current multi-core systems which contain *16* to

32 cores, a data-path with *11* (or more) transistors, and are operating at frequencies of *3* GHz and above [11, 12, 13, 14].

Given the characteristics of an ALU element, synchronizing the ALU elements within the interface board is also at a relatively modest level of complexity. The synchronization can be done by distributing a clock signal through the ALU elements. Commercial systems that distribute clock through a clock tree to hundreds of devices with skew of less than 50 pico seconds (ps) are available. Hence, a two-branch *8* level clock distribution unit can be used to synchronize the *16* ALU elements. Using this tree, an uncertainty of the order of <50 ps is achievable [12]. Nevertheless, since the source array is operating at a rate of *125* MHz, *50* ps is a negligible uncertainty and can be factored into the design. The effect of uncertainty can cause a lose of one least significant bit from an entire set of *256x8* bits. Given the fact that we are using guard bits in the MVMM multiplier this uncertainty is tolerable.

6. References

1. D. G. Feitelson, Optical Computing: *A Survey for Computer Scientists* (MIT Press, Cambridge, MA, 1988).
2. D. McAulay, *Optical Computer Architectures: The Application of Optical Concepts to Next Generation Computers* (Wiley-Interscience, New York, 1991).
3. M. A. Karim and A. A. S. Awwal, *Optical Computing: An Introduction* (John Wiley & Sons, New York, 1992).
4. J. W. Goodman, Introduction to Fourier Optics (McGraw-Hill, New York, 1996).
5. Goren, A. Sariel, S. Levit, Y. Asaf, S. Liberman, B. Sender, T. Tzelnick, Y. Hefetz, E. Moses, and V. Machal, *Vector-Matrix Multiplication*, United States Patent 20040243657 (2001).
6. A. V. Oppenheim, R. W. Schafer, *Discrete time Signal Processing*, 2nd ed., Prentice Hall, 1999.
7. J. Bier, *Byers Guide to DSP Processors (*2005 edition*)*, Berkeley Design Technology, 2006.

8. Anonymous, *C64 Performance Benchmarks*, Texas Instruments, 2008 in: http://focus.ti.com/dsp/docs/dspplatformscontentaut.tsp?sectionId=2&familyId=477&tabId=496

9. M. Gschwind, "Chip Multiprocessing and the Cell Broadband Engine ", *IBM Research Report*, RC2931 2006.

10. Anonymous, *International Technology roadmap for Semiconductors; Assembly and packaging.* ITRS, 2005

11. J. Kim, I. Verbauwhede, and *M.* F. Chang, "Design of an Interconnect Architecture and Signaling Technology for Parallelism in Communication" *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* vol. 15, No. 8, August 2007.

12. T. H. Wood, E. C. Carr, C. A. Burrus, J. E. Henry, A. C. Gossard, and J. H. English, **"**High-speed 2 Ã— 2 electrically driven spatial light modulator made with GaAs/AlGaAs multiple quantum wells (MQWs)", *Electronics Letters*, Volume 23, Issue 17, August 13 1987.

13. D. Wentzlaff et al. "On-Chip Interconnection Architecture of the Tilera Processor," *IEEE Micro*, Volume 27 Number 5, September/October 2007, pp 15-31.

14. Y. Hoskote et al. "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, Volume 27 Number 5, September/October 2007, pp 51-61.

15. J. Wolf, and J. Adams, *2005 Packaging Roadmap Overview*, the International Electronics Manufacturing Initiative (INEMI), 2005 http://thor.inemi.org/webdownload/Industry_Forums/Productronica_2005/2007_Roadmap_Kickoff/2005_Packaging_Roadmap.pdf

16. Shaked, T., N., Messika, S., Dolev, S., and Rosen, Y., "Optical Solution for Bounded NPComplete Problems", *Journal of Applied Optics*, Vol. 46, Issue 5, pp. 711-724, February 2007.

17. Shaked, T. S., Tabib, T., Simon, G., Messika, S., Dolev, S., and Rosen, J., "Optical Binary-Matrix Synthesis for Solving Bounded NP-complete Combinatorial Problems," *Optical Engineering*, Vol. 46, No. 10, October 2007.

18. Dolev, S., and Nir, Y., "Optical Implementation of Bounded Non-Deterministic Turing Machines", US Patent No. 7,130,093 B2, October, 2006.

19. Dolev, S., Fitoussi, H., “The Traveling Beams, Optical Solutions for Bounded NP-Complete Problems”, Fourth International Conference on Fun with Algorithms (FUN2007), LNCS 4475, pp. 120-134, June 2007.