Massively Parallel Inner-Product Array Processor

Roman Genov and Gert Cauwenberghs

Department of Electrical and Computer Engineering
Johns Hopkins University, Baltimore, MD 21218, U.S.A.
E-mail: {roman,gert}@bach.ece.jhu.edu [1]

Abstract

We present a hardware architecture for parallel inner-product array computation in very high dimensional feature spaces, towards a general-purpose kernel-based classifier and function approximator. The architecture is internally analog with fully digital interface. On-chip analog fine-grain parallel processing yields real-time throughput levels for high-dimensional (over 1,000 per chip) classification tasks. The architecture contains an array of computational cells with integrated digital storage and a parallel bank of analog-to-digital converters (ADC). A three-transistor unit cell combines a single-bit dynamic random-access memory (DRAM) and a charge injection device (CID) binary multiplier and analog accumulator. Digital multiplication with enhanced resolution is obtained with bit-serial input vectors and bit-parallel storage of weights, by combining quantized outputs from multiple rows of binary unit cells over time. A prototype 128 × 512 inner-product array processor on a single 3mm × 3mm chip fabricated in standard CMOS 0.5μm technology achieves 8-bit effective resolution, consumes 3.3mW of power and offers 2×10^{12} binary MACS (multiply accumulates per second) per Watt of power. This corresponds to a factor of at least 1,000 increase in computational efficiency compared to modern desktop workstations. Based on the inner-product array processor, an efficient real-time massively-parallel hardware architecture of a Support Vector Machine classifier is presented.

1 Introduction

A general-purpose parallel processor for applications requiring real-time matrix-vector multiplication (VMM) in very high dimensions is presented. The inner-product array processor computes M inner products:

$$Y^{(m)} = \sum_{n=0}^{N-1} W^{(m,n)} X^{(n)} = \mathbf{W}^{(m)} \cdot \mathbf{X} \qquad (1)$$

with N-dimensional input vector $X^{(n)}$, and $N \times M$ matrix of stored elements $W^{(m,n)}$. In artificial neural networks, for instance, the matrix elements $W^{(m,n)}$ correspond to weights, or synapses, between neurons. The elements may also represent templates $X_n^{(m)} = W^{(m,n)}$ (or $\mathbf{X}_m = \mathbf{W}^{(m)}$, in vector form) in a vector quantizer [1], or support vectors in a Support Vector Machine (SVM) [2].

Most of modern neural networks realizations contain a vector-matrix multiplier providing the connections between neurons followed by a nonlinear transformation. Support Vector Machines training and classification, in particular, require the evaluation of a quadratic programming problem to perform vector-matrix multiplication in very large dimensions (100-10,000) which is a very computationally intensive task. Most current SVM implementations are software-based. They use a desktop computer with often a single processor, large memory and a standard quadratic programming package, often partitioning the problem into blocks. However there is the computational limitation of the desktop computer limited by its serial nature of operation and limited memory bandwidth. DSPs also lack parallelism and memory bandwidth needed for efficient real-time implementation. Multiprocessors and networked parallel computers in principle are capable of high throughput, but are costly, and impractical for embedded real-time applications. Extensive silicon area and high power dissipation of a digital multiply-and-accumulate implementation make fully-digital hardware implementations prohibitive for very large (100-10,000) matrix dimensions [4].

Analog VLSI provides a natural medium to implement fully parallel computational arrays with high integration density and energy efficiency [5]. By summing charge or current on a single wire across cells in the array, low latency is intrinsic. Analog multiply-and-accumulate circuits are so small that one can be provided for each matrix element, making massively parallel implementations with large ma-

[1] This work was supported by NSF MIP-9702346 and ONR N00014-99-1-0612. Chips were fabricated through the MOSIS foundry service.

0-7803-7044-9/01/$10.00 ©2001 IEEE

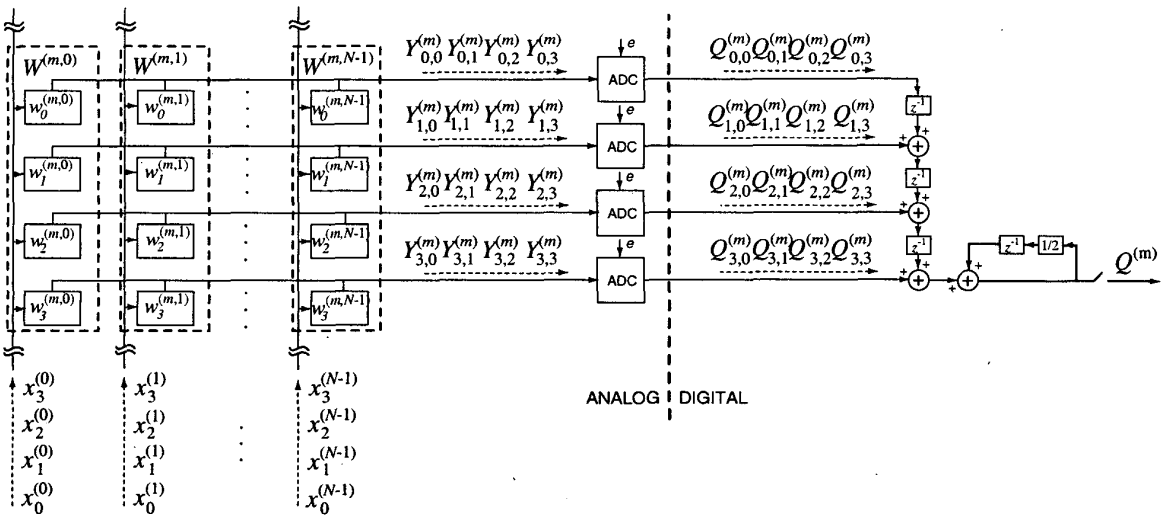

Figure 1: Block diagram of one row in the matrix with binary encoded elements $w_i^{(m,n)}$, for a single m. Data flow of bit-serial inputs $x_j^{(n)}$ and corresponding partial outputs $Y_{i,j}^{(m)}$ before, and $Q_{i,j}^{(m)}$ after quantization. A/D quantization is performed using row-parallel flash A/D converters. The output is constructed using digital post-processing. The example shown is for LSB-first inputs, and $I = J = 4$.

trix dimensions feasible as described in the next section.

2 Mixed-Signal Architecture

2.1 Internally Analog, Externally Digital Computation

The system presented is internally implemented in analog VLSI technology, but interfaces externally with the digital world. This paradigm combines the best of both worlds: it uses the efficiency of massively parallel analog computing (in particular: adding numbers in parallel on a single wire), but allows for a modular, configurable interface with other digital pre-processing and post-processing systems. This is necessary to make the processor a general-purpose device that can tailor the vector-matrix multiplication task to the particular application where it is being used.

The digital representation is embedded, in both bit-serial and bit-parallel fashion, in the analog array architecture (Fig. 1). Inputs are presented bit-serially, and matrix elements are stored locally in bit-parallel form. Digital-to-analog (D/A) conversion at the input interface is inherent in the bit-serial implementation, and row-parallel analog-to-digital (A/D) converters are used at the output interface.

For simplicity, an unsigned binary encoding of inputs and matrix elements is assumed here, for one-quadrant multiplication. This assumption is not essential: it has no binding effect on the architecture and can be easily extended to a standard one's complement for four-quadrant multiplication, in which the significant bits (MSB) of both arguments have a negative rather than positive weight. Assume further I-bit encoding of matrix elements, and J-bit encoding of inputs:

$$W^{(m,n)} = \sum_{i=0}^{I-1} 2^{-(i+1)} w_i^{(m,n)} \tag{2}$$

$$X^{(n)} = \sum_{j=0}^{J-1} 2^{-(j+1)} x_j^{(n)} \tag{3}$$

decomposing (1) into:

$$Y^{(m)} = \sum_{n=0}^{N-1} W^{(m,n)} X^{(n)} = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} 2^{-(i+j+2)} Y_{i,j}^{(m)} \tag{4}$$

with binary-binary VMM partials:

$$Y_{i,j}^{(m)} = \sum_{n=0}^{N-1} w_i^{(m,n)} x_j^{(n)} \,. \tag{5}$$

The proposed mixed-signal approach is to compute and accumulate the binary-binary partial products (5) using an analog VMM array, and to combine the quantized results in the digital domain according to (4).

2.2 Array Architecture and Data Flow

To conveniently implement the partial products (5), the binary encoded matrix elements $w_i^{(m,n)}$ are stored in bit-parallel form, and the binary encoded inputs $x_j^{(n)}$ are pre-

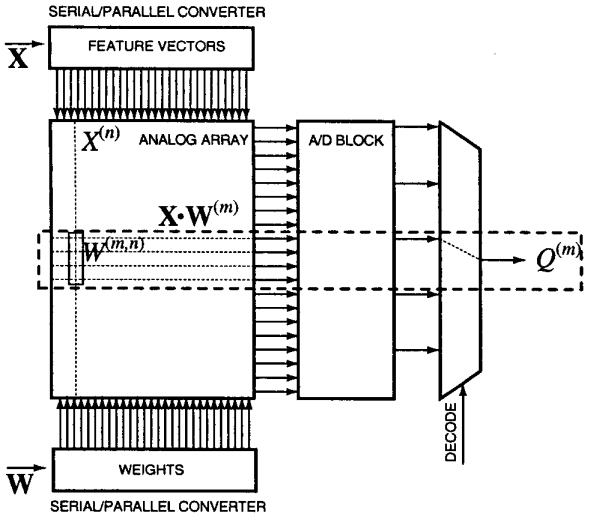

Figure 2: Top level architecture of the inner-product array processor.

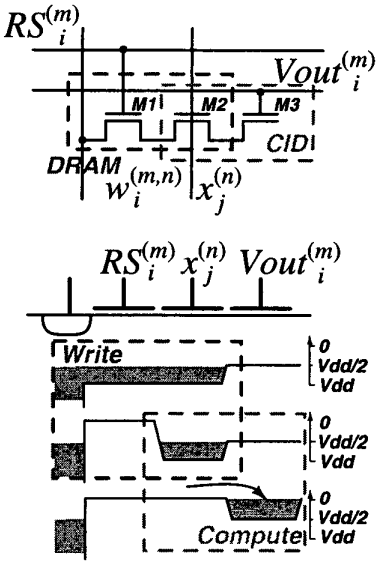

Figure 3: CID computational cell with integrated DRAM storage (*top*). Charge transfer diagram for active write and compute operations (*bottom*).

sented in bit-serial fashion as shown in Figure 2. The bit-serial format was first proposed and demonstrated in [7], with binary-analog partial products using analog matrix elements for higher density of integration. The use of binary encoded matrix elements relaxes precision requirements and simplifies storage [8].

One row of I-bit encoded matrix elements uses I rows of binary cells. Therefore, to store an $M \times N$ *digital* matrix $W^{(m,n)}$, an array of $MI \times N$ *binary* cells ${w_i}^{(m,n)}$ is needed. One bit of an input vector is presented each clock cycle, taking J clock cycles of partial products (5) to complete a full computational cycle (1). The input binary components ${x_j}^{(n)}$ are presented least significant bit (LSB) first, to facilitate the digital postprocessing to obtain (4) from (5) (as elaborated in Section 4).

Figure 1 depicts one row of matrix elements $W^{(m,n)}$ in the binary encoded architecture, comprising I rows of binary cells ${w_i}^{(m,n)}$, where $I = 4$ in the example shown. The data flow is illustrated for a digital input series ${x_j}^{(n)}$ of $J = 4$ bits, LSB first (*i.e.*, descending index j). The corresponding analog series of outputs ${Y_{i,j}}^{(m)}$ in (5) obtained at the horizontal summing nodes of the analog array is quantized by a bank of analog-to-digital converters (ADC), and digital postprocessing (4) of the quantized series of output vectors yields the final digital result (1).

The quantization scheme used is critical to system performance. As shown in Section 4, appropriate postprocessing *in the digital domain* to obtain (4) from the quantized partial products ${Y_{i,j}}^{(m)}$ can lead to a significant enhancement in system resolution, well beyond that of intrinsic ADC resolution. This relaxes precision requirements on the analog implementation of the partial products (5). A dense and efficient charge-mode VLSI implementation is described next.

3 Charge-Mode VLSI Implementation

3.1 CID/DRAM Cell and Array

The elementary cell combines a CID computational unit [7, 8], computing one argument of the sum in (5), with a DRAM storage element. The cell stores one bit of a matrix element ${w_i}^{(m,n)}$, performs a one-quadrant binary-binary multiplication of ${w_i}^{(m,n)}$ and ${x_j}^{(n)}$, and accumulates the result across cells with common m and i indices. The circuit diagram and operation of the cell are given in Figure 3. An array of cells thus performs (unsigned) binary multiplication (5) of matrix ${w_i}^{(m,n)}$ and vector ${x_j}^{(n)}$ yielding ${Y_{i,j}}^{(m)}$, for values of i in parallel across the array, and values of j in sequence over time.

The cell contains three MOS transistors connected in series as depicted in Figure 3. Transistors M1 and M2 comprise a dynamic random-access memory (DRAM) cell, with switch M1 controlled by *Row Select* signal ${RS_i}^{(m)}$. When activated, the binary quantity ${w_i}^{(m,n)}$ is written in the form of charge stored under the gate of M2. Transistors M2 and M3 in turn comprise a charge injection device (CID), which by virtue of charge conservation moves electric charge between two potential wells in a non-destructive manner [7, 8, 18].

The cell operates in two phases: *Write* and *Compute*. When a matrix element value is being stored, $x_j^{(n)}$ is held at Vdd and $Vout$ at a voltage $Vdd/2$. To perform a write operation, either an amount of electric charge is stored under the gate of M2, if $w_i^{(m,n)}$ is low, or charge is removed, if $w_i^{(m,n)}$ is high. The amount of charge stored, $\triangle Q$ or 0, corresponds to the binary value $w_i^{(m,n)}$.

Once the charge has been stored, the switch M1 is deactivated, and the cell is ready to compute. The charge left under the gate of M2 can only be redistributed between the two CID transistors, M2 and M3. An active charge transfer from M2 to M3 can only occur if there is non-zero charge stored, and if the potential on the gate of M2 drops below that of M3 [7]. This condition implies a logical AND, *i.e.*, unsigned binary multiplication, of $w_i^{(m,n)}$ and $x_j^{(n)}$. The multiply-and-accumulate operation is then completed by capacitively sensing the amount of charge transferred onto the electrode of M3, the output summing node. To this end, the voltage on the output line, left floating after being precharged to $Vdd/2$, is observed. When the charge transfer is active, the cell contributes a change in voltage

$$\triangle V_{out} = \triangle Q / C_{M3} \tag{6}$$

where C_{M3} is the total capacitance on the output line across cells. The total response is thus proportional to the number of actively transferring cells. After deactivating the input $x_j^{(n)}$, the transferred charge returns to the storage node M2. The CID computation is non-destructive and intrinsically reversible [7], and DRAM refresh is only required to counteract junction and subthreshold leakage.

The bottom diagram in Figure 3 depicts the charge transfer timing diagram for write and compute operations in the case when both $w_i^{(m,n)}$ and $x_j^{(n)}$ are of logic level 1. A logic level 0 for $w_i^{(m,n)}$ is represented as Vdd, and a logic level 1 is represented as $Vdd/2$, where Vdd is the supply voltage. For $x_j^{(n)}$, logic level 0 is represented as Vdd, and logic level 1 as GND.

Transistor-level simulation of a 512-element row indicates a dynamic range of 43 dB, and a computational cycle of 10 μs with power consumption of 50 nW per cell.

3.2 Experimental Results

We designed, fabricated and tested a VLSI prototype of the inner-product array processor, integrated on a 3×3 mm^2 die in 0.5 μm CMOS technology. The chip contains an array of 512 × 128 CID/DRAM cells, and a row-parallel bank of 128 gray-code flash ADCs. Figure 4 depicts the micrograph and system floorplan of the chip. The layout size of the CID/DRAM cell is $8\lambda \times 45\lambda$ with $\lambda = 0.3\mu m$.

The mixed-signal inner-product array processor interfaces externally in digital format. Two separate shift registers

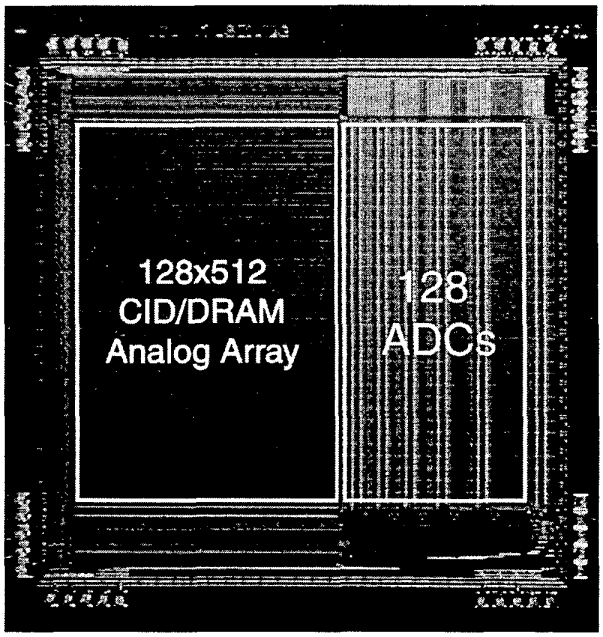

Figure 4: Micrograph of the mixed-signal VMM prototype, containing an array of 512 × 128 CID/DRAM cells, and a row-parallel bank of 128 flash ADCs. Die size is 3 mm × 3 mm in 0.5 μm CMOS technology.

load the matrix elements along odd and even columns of the DRAM array. Integrated refresh circuitry periodically updates the charge stored in the array to compensate for leakage. Vertical bit lines extend across the array, with two rows of sense amplifiers at the top and bottom of the array. The refresh alternates between even and odd columns, with separate select lines. Stored charge corresponding to matrix element values can also be read and shifted out from the chip for test purposes. All of the supporting digital clocks and control signals are generated on-chip.

Figure 5 shows the measured linearity of the computational array. The cases shown are when all binary weight storage elements are actively charged and discharged, and an all-ones sequence of bits is shifted through the input register, initialized to all-zeros bit values. For every 1-bit shift, a computation is performed and the result is observed on the output sense line. The experimentally observed linearity agrees with the simulation results [19]. The feed-through input dependent offsets are compensated for as described in [20].

The chip contains 128 row-parallel 6-bit flash ADCs, *i.e.*, one dedicated ADC for each m and i. In the present implementation, $Y^{(m)}$ is obtained off-chip by combining the ADC quantized outputs $Y_{i,j}^{(m)}$ over i (rows) and j (time) according to (4). Issues of precision and complexity in the

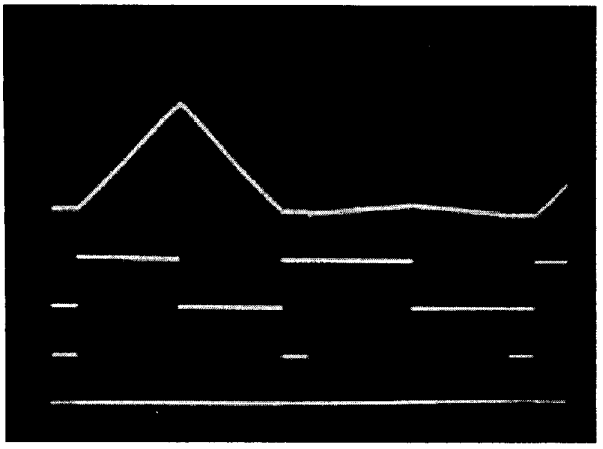

Figure 5: Measured linearity of the computational array. Two cases are shown: all binary weight storage elements are actively charged (*left*) and discharged (*right*). All logic "1" sequence of bits is shifted through the input register, initialized to all-"0" bit values. For every 1-bit shift, a computation is performed. Waveforms shown, *top to bottom*: the analog voltage output on the sense line; input data - on an input pin in common for both input and weight shift register; clock for weight shift register.

implementation of (4) are studied below.

4 Quantization and Digital Resolution Enhancement

Significant improvements in precision can be obtained by exploiting the binary representation of matrix elements and vector inputs, and performing the computation (4) in the digital domain, from quantized estimates of the partial outputs (5).

We quantize all $I \times J$ values of ${Y_{i,j}}^{(m)}$ using row parallel flash A/D converters. Figure 1 presents the corresponding architecture, shown for a single output vector component m. The partials summation is then performed in the digital domain:

$$Q^{(m)} = \sum_{i=0}^{I-1}\sum_{j=0}^{J-1} 2^{-(i+j+2)} Q_{i,j}^{(m)} = \sum_{k=0}^{K-1} 2^{-(k+2)} Q_k^{\prime(m)}, \tag{7}$$

where $k = i + j$, $K = I + J - 1$ and

$$Q_k^{\prime(m)} = \sum_{i=\kappa(k,I)}^{k-\kappa(k,I)} Q_{i,k-i}^{(m)}, \tag{8}$$

with $\kappa(k, I) \equiv \max(0, k-I+1)$ and $\kappa(k, J) \equiv \max(0, k-J+1)$. A block diagram for a digital implementation is shown on the right of Figure 1.

The effect of averaging the quantization error over a large number of quantized values of ${Y_{i,j}}^{(m)}$ boosts the precision of the digital estimate of $Y^{(m)}$, beyond the intrinsic resolution of the analog array and the A/D quantizers used as shown in detail in [20]. We obtain an improvement in signal-to-quantization-noise ratio of a factor 3 and a median resolution gain of approximately 2 bits over the resolution of each ADC.

5 Support Vector Machine Hardware Architecture

The inner-product array processor can be directly used to emulate a class of kernel-based classifiers and regressors, such as Support Vector Machines (SVM).

In its general form, a SVM classifies a pattern vector $\mathbf{X}$ into class $y \in \{-1, +1\}$ based on the training data points $\mathbf{X}_m$ and corresponding classes y_m as:

$$y = sign\Big(\sum_{m=0}^{M-1} \alpha_m y_m K(\mathbf{X}_m, \mathbf{X}) + b\Big), \tag{9}$$

where $K(\cdot, \cdot)$ is a symmetric positive-definite kernel function which can be freely chosen subject to fairly mild constraints [21], α_m and b are coefficients and an offset term respectively, obtained during training. We use inner-product norm functions (*e.g.* sigmoidal connectionist; polynomial):

$$K(\mathbf{X}_m, \mathbf{X}) = f(\mathbf{X}_m \cdot \mathbf{X}) = f(\sum_{n=0}^{N-1} X_m^{(n)} X^{(n)}). \tag{10}$$

SVM training and classification require performing vector-matrix multiplication in very large dimensions in (9), which is very computationally intensive. The scope of the problem, described in more detail in Section 1, calls for special-purpose VLSI hardware to implement the computationally intensive kernel operations in (10) most efficiently, thereby reducing the power consumption and physical size to minimum levels. Our approach, an architecture based on the massively-parallel inner-product array processor described in Sections 2, 3 and 4, efficiently performing this operation is presented in Figure 6.

The inner-product array processor performs the most computationally intensive operation of vector-matrix multiplication. The rest of the computation in equation (9) is implemented in digital domain with coefficients $\alpha_m y_m$ and the kernel table stored in a memory.

The architecture implements a fully autonomous Support

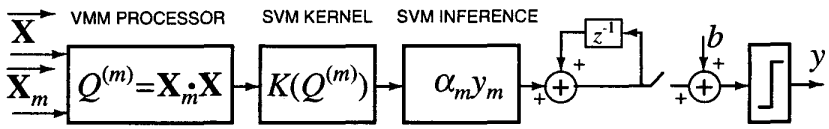

Figure 6: Support Vector Machine hardware architecture.

Vector "Machine". A single $9mm^2$ chip consumes 3.3mW of power and offers 2×10^{12} binary MACS (multiply accumulates per second) per Watt of power. Scaled to a $0.35\mu m$ technology it provides for a throughput of 100 GMACS on a 6mm × 6mm CMOS chip. This corresponds to computing 1,000 inner-products of 1-bit 1,000-dimensional vectors every 10μSec, making the architecture very well suited for real-time implementations of modern artificial vision and human-computer interfaces [22].

6 Conclusions

A novel high-throughput parallel inner-product array processor architecture has been presented. It allows for real-time classification in very high dimensional input spaces. The architecture embeds storage and multiplication in distributed fashion, down to the cellular level. With only three transistors, the cell for multiplication and storage contains little more than either a DRAM or a CID cell. This makes the analog cell very compact and low power, and the regular array of cells provides for a scalable architecture which can be extended to any number of input features and weights as limited by the technology and the number of cascaded chips.

A prototype 128 × 512 inner-product array processor on a single 3mm × 3mm chip fabricated in standard CMOS $0.5\mu m$ technology achieves 8-bit effective resolution, consumes 3.3mW of power and offers 2×10^{12} binary MACS (multiply accumulates per second) per Watt of power. This corresponds to a factor of 100 to 10,000 increase in computational efficiency compared to modern desktop workstations, digital multiprocessors and DSPs.

An efficient real-time massively-parallel Support Vector "Machine", based on the inner-product array processor, for classification in very large dimensional spaces has been proposed.

References

[1] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, Norwell, MA: Kluwer, 1992.

[2] V. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed., Springer-Verlag, 1999.

[3] J. Wawrzynek, et al., "SPERT-II: A Vector Microprocessor System and its Application to Large Problems in Backpropagation Training," in *Advances in Neural Information Processing Systems*, Cambridge, MA: MIT Press, vol. 8, pp 619-625, 1996.

[4] J.C. Gealow and C.G. Sodini, "A Pixel-Parallel Image Processor Using Logic Pitch-Matched to Dynamic Memory," *IEEE J. Solid-State Circuits*, vol. **34**, pp 831-839, 1999.

[5] A. Kramer, "Array-based analog computation," *IEEE Micro*, vol. **16** (5), pp. 40-49, 1996.

[6] A. Chiang, "A programmable CCD signal processor," *IEEE Journal of Solid-State Circuits*, vol. **25** (6), pp. 1510-1517, 1990.

[7] C. Neugebauer and A. Yariv, "A Parallel Analog CCD/CMOS Neural Network IC," Proc. IEEE Int. Joint Conference on Neural Networks (IJCNN'91), Seattle, WA, vol. **1**, pp 447-451, 1991.

[8] V. Pedroni, A. Agranat, C. Neugebauer, A. Yariv, "Pattern matching and parallel processing with CCD technology," Proc. IEEE Int. Joint Conference on Neural Networks (IJCNN'92), vol. **3**, pp 620-623, 1992.

[9] G. Han, E. Sanchez-Sinencio, "A general purpose neuro-image processor architecture," Proc. of IEEE Int. Symp. on Circuits and Systems (ISCAS'96), vol. **3**, pp 495-498, 1996.

[10] M. Holler, S. Tam, H. Castro and R. Benson, "An Electrically Trainable Artificial Neural Network (ETANN) with 10,240 Floating Gate Synapses," in *Proc. Int. Joint Conf. Neural Networks*, Washington DC, pp 191-196, 1989.

[11] F. Kub, K. Moon, I. Mack, F. Long, " Programmable analog vector-matrix multipliers," *IEEE Journal of Solid-State Circuits*, vol. **25** (1), pp. 207-214, 1990.

[12] G. Cauwenberghs, C.F. Neugebauer and A. Yariv, "Analysis and Verification of an Analog VLSI Incremental Outer-Product Learning System," *IEEE Trans. Neural Networks*, vol. **3** (3), pp. 488-497, May 1992.

[13] A. Aslam-Siddiqi, W. Brockherde, B. Hosticka, "A 16x16 nonvolatile programmable analog vector-matrix multiplier," *IEEE Journal of Solid-State Circuits*, vol. **31** (10), pp. 1502-1509, 1998.

[14] A.G. Andreou, K.A. Boahen, P.O. Pouliquen, A. Pavasovic, R.E. Jenkins, and K. Strohbehn, "Current-Mode Subthreshold MOS Circuits for Analog VLSI Neural Systems," *IEEE Transactions on Neural Networks*, vol. **2** (2), pp 205-213, 1991.

[15] C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, 1989.

[16] E. Vittoz, "Micropower Techniques," in *Design of Analog-Digital VLSI Circuits for Telecommunications and Signal Processing*, Franca and Tsividis, Eds., Prentice Hall, 2nd. ed., pp 53-96, 1994.

[17] M. Ismail and T. Fiez, Eds., *Analog VLSI for Signal and Information Processing*, McGraw-Hill, 1995.

[18] M. Howes, D. Morgan, Eds., *Charge-Coupled Devices and Systems*, John Wiley & Sons, 1979.

[19] R. Genov and G. Cauwenberghs, "Charge-Mode Parallel Architecture for Matrix-Vector Multiplication," *Proc. 43rd IEEE Midwest Symp. Circuits and Systems (MWSCAS'2000)*, Lansing MI, August 8-11, 2000.

[20] R. Genov, G. Cauwenberghs, "Analog Array Processor with Digital Resolution Enhancement and Offset Compensation," *Proc. of Conf. on Information Sciences and Systems (CISS'2001)*, Baltimore, MD 2001.

[21] Boser, B., Guyon, I. and Vapnik, V., "A training algorithm for optimal margin classifier," in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp 144-52, 1992.

[22] Papageorgiou, C.P, Oren, M. and Poggio, T., "A General Framework for Object Detection," in *Proceedings of International Conference on Computer Vision*, 1998.