Mapping Image Transformations Onto Pixel Processor Arrays

Laurie Bose[1]   Piotr Dudek[2]

[1]University of Bristol, Bristol, United Kingdom
[2]University of Manchester, Manchester, United Kingdom

Abstract—**Pixel Processor Arrays (PPA) present a new vision sensor/processor architecture consisting of a SIMD array of processor elements, each capable of light capture, storage, processing and local communication. Such a device allows visual data to be efficiently stored and manipulated directly upon the focal plane, but also demands the invention of new approaches and algorithms, suitable for the massively-parallel fine-grain processor arrays. In this paper we demonstrate how various image transformations, including shearing, rotation and scaling, can be performed directly upon a PPA. We also implement an image-wide population count algorithm. The implementation details are presented using the SCAMP-5 vision chip, that contains a 256x256 pixel-parallel array. Our approaches for performing the image transformations efficiently exploit the parallel computation in a cellular processor array, minimizing the number of SIMD instructions required. These fundamental image transformations are vital building blocks for many visual tasks. Thus this paper seeks to both provide a useful reference for future PPA works and also illustrate the flexibility of PPA architectures.**

I. INTRODUCTION

Recent trends in edge computing bring to the fore the concerns about power efficiency of the processing hardware. Some of the most challenging applications are in computer vision, where large amounts of raw sensory data (image pixels) need to be processed. It is well known that data movements are currently the most critical operations responsible for energy consumption as well as the overall speed of the system. Minimising external memory access has become a necessity, and one of the solutions is a distributed architecture, with memory and processing resources collocated on a single device. As many low-level image processing tasks are inherently parallel, with computations localised (results are dependent on pixels and their neighbours), and identical operations executed for all pixels in the image, they are well suited to massively parallel SIMD (Single Instruction Multiple Data) architectures. An extreme level of parallelism can be achieved by allocating a processor per pixel, in a fine-grained SIMD architecture (Figure 1). A very large number of processing elements, each containing local memory and arithmetic logic units, can efficiently execute pixel-parallel algorithms. Such cellular processor architectures have been considered in the past [1]–[4]. With more recent advances in silicon fabrication technologies it is now possible to integrate thousands of elementary processors on a silicon chip, in a pixel-parallel image processor. Furthermore, it is now possible to integrate image sensing elements within the compute-memory fabric of the processor array on a single "vision chip" [5]–[7]. The co-
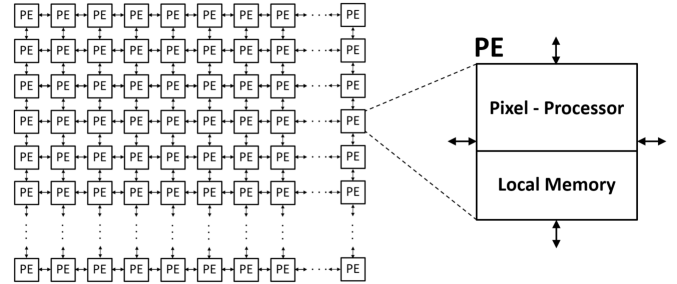

Fig. 1. A Pixel Processor Array contains a SIMD array of processing elements (PEs), on a 2D grid, where each PE contains processing and local memory and is allocated to one pixel in the image.

location of photosensors and processors minimises the sensor-processor communications, providing additional benefits in terms of speed and power consumption of the system. We term such a device a Pixel Processor Array (PPA), where sensing, processing, and local memory are collocated on a processor-per-pixel basis. PPA vision sensors have been demonstrated, with resolutions up to 256×256 pixels [8]. The recent technological trends of 3D silicon wafer stacking provide a vehicle for vertically integrating sensor and processor layers, promising future high-resolution vision sensor devices, where computing power can be placed behind each pixel of the image sensor [9]–[11].

The key advantage of PPA systems is that all low-level image processing occurs on the vision sensor integrated circuit, with no images transmitted off-chip in normal operation. Instead, only results of computations, for instance extracted features [12], classification results [13], or visual odometry information [14], are read-out directly from the device. To ensure that only low-dimensional data is read-out from the device, the PPA must be capable of carrying out all low-level image processing operations in the pixel-parallel array, before using some mechanisms of sparse or summative read-out. While many commonly used image processing operations, for instance local brightness adaptation, corner extraction, image convolution, etc. involve pixel-wise, localised computations, and are easily mapped onto PPA devices, it is not always obvious how to achieve mapping between the pixel-parallel architecture and operations that involve image transformations. In this paper we illustrate how operations such as image rotation, and scaling can be efficiently implemented on a pixel-
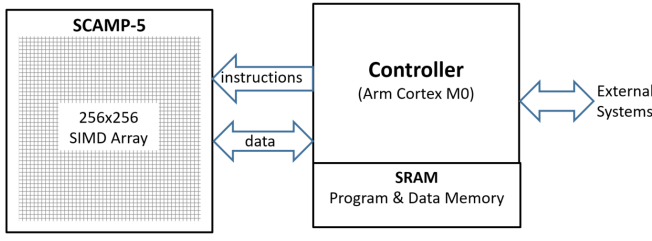
Fig. 2. Overview of the SCAMP vision system. The control program is executed on the ARM M0 core, which instructs the SCAMP-5 massively-parallel SIMD processor array to carry out operations on image arrays. SCAMP-5 has 256x256 Processing Elements.
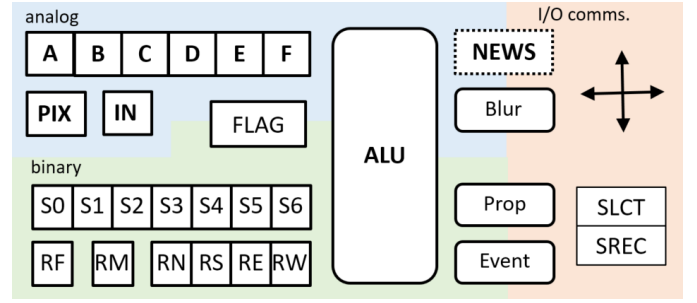

Fig. 3. The architecture of the SCAMP-5 Processing Element. A-F are analog registers, PIX is image sensor input, IN is a global input. S0-S6 are general-purpose binary registers. Rx are special-purpose registers. ALU executes transfers and arithmetic and logic operations, 'Blur' and 'Prop' are additional asynchronous hardware accelerators. FLAG is local activity register. NEWS provides 4-neighbour communications. SLCT and SREC provide array addressing and 'Event' unit enables sparse read-out.

parallel device.

The algorithms we propose are generally applicable to PPA devices, but in our implementation and experiments we use the SCAMP-5 vision sensor device [8]. The architecture of the chip is briefly presented in the next section. Section III will introduce image shear operation which is then used to implement rotations described in Section IV. Section V will present image scaling algorithm. Section **??** will present an algorithm for calculating a population count across a binary image array, exploiting pixel-parallel computations and sparse read-out mechanism of the SCAMP-5 chip.

II. SCAMP-5 ARCHITECTURE

The overall architecture of the hardware system used in this work is illustrated in Figure 2. The SCAMP-5 chip comprises a 256×256 array of Processing Elements (PEs), which receive instructions from a single Controller (Arm Cortex-M0). The controller has its own program and data memory, and is responsible for the overall program flow, and any sequential computing required in the algorithm. It also issues microinstructions to the SCAMP-5 array. All PEs in the array execute the same microinstruction, issued by the Controller, i.e. the array operates as a SIMD processor.

Although it is possible to transfer data from the Controller to the SCAMP-5 array, the primary input to the array is optical, via photosensors in each PE. The typical operation is to acquire an image, and then process it in the SCAMP-5 array, according to the sequence of microinstructions sent by the Controller. The results of computations are read-out from the SCAMP-5 array by the Controller. While reading out entire data arrays is possible (and useful for debugging purposes), the fundamental read-out mechanisms is a sparse, "address-event" type of read-out. As a result of processing, the images are reduced to binary maps, preferably containing only a few non-zero pixels, and the row-column addresses of these pixels are sequentially extracted by the SCAMP-5 readout hardware, so that the array information is reduced to a few 16-bit addresses only.

The detail of the PE architecture is shown in Figure 3. Each PE contains six general-purpose "analog" registers that can store a gray-level pixel value or results of arithmetic operations, and thirteen binary registers. Several binary registers also have special-purpose designations. The ALU provides basic arithmetic and logic operations on the registers, for instance addition or subtraction of two analog registers, or logic AND operation on binary registers.

The FLAG register is a binary activity flag, used to implement conditional instruction execution. In each PE this can be set or reset individually, providing a degree of local autonomy. Only the PEs with FLAG set will execute SIMD instructions issued by the controller, otherwise these instructions are ignored.

The NEWS register is used to provide a mechanism for transferring data between a PE and its four nearest neighbours in the array. For instance, it is possible to move content of register A, to the same register A located in the PE's neighbour to the South. From the point of view of the PE array, this results in a shift of data one pixel to the South. The transfer in binary registers is achieved using a multi-directional propagation operation, with the direction of transfer controlled by additional registers, for instance by setting RN=1 and RS=1, the operation S0=DNEWS(S0) will result in the value of S0 propagating simultaneously in both vertical directions.

The details of the SCAMP-5 implementation can be found in [8]. The datapath is implemented using mixed-signal circuits, in particular storage and arithmetic operations on registers A-F are using analog current-mode signal representation. This has some implications with respect to the precision and accuracy of arithmetic operations, and often requires special care be taken to ensure the inherent processing errors do not adversely affect the computation results. These considerations are beyond the scope of this paper. In many situations the processors can be programmed on the assumption that the computations on registers A-F are equivalent to about 8-bit accuracy.

The analog current-mode computations allow operations such as global summation (all elements of the array are effectively added in one clock cycle) but this has limited precision.

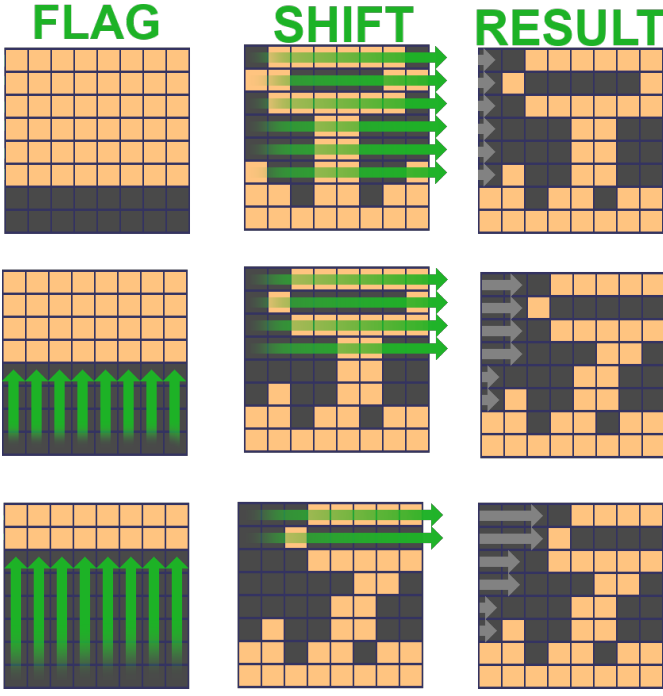When implementing vision algorithms on this architecture,

Fig. 4. Illustration of performing three steps of a horizontal shear. The FLAG register (Left Column) determines along which PE rows data is shifted at each stage. The FLAG register content itself is also shifted upwards in-between each step. As data is repeatedly shifted along the flagged rows the image becomes sheared (Top-Right to Bottom-Right).

a challenge is how to map the required image processing operations onto the constrained processor hardware. Currently, the SCAMP-5 array is programmed using in-line assembler code, while the overall Controller code can be compiled using standard C/C++. The work on more sophisticated compilation tools for this system is on-going [15], [16].

Another challenge, is how to map the required computations onto the pixel-per-processor topology. This is illustrated by the algorithms introduced in the fol1lowing sections, which demonstrate how pixel-parallel operations can be used to perform non-trivial image transformations such as rotations and scaling.

III. SHEAR TRANSFORMATIONS

A 2D shear transformation shifts all points parallel to some line through the origin. For each point the direction and magnitude of this shift is proportional to its signed distance from said line. In this work we only consider shears parallel to the X and Y axi, which when combined correctly can be used to form various other transformations as demonstrated later in Section IV. The matrices for shear transformations parallel to the X and Y axi respectively are given in Equations 1 and 2.

$$\begin{pmatrix} x^{'} \\ y^{'} \end{pmatrix} = \begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + \alpha y \\ y \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} x^{'} \\ y^{'} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y + \alpha x \end{pmatrix} \tag{2}$$

The shear matrix of Equation 1 performs a one-to-one mapping, taking each point (x, y) to its shifted position $(x+\alpha y, y)$. In this case each point's X coordinate is altered proportional to it's Y coordinate, constituting a horizontal shear parallel to the X axis. A vertical shear parallel to the y axis as in Equation 2 is similarly defined.

A. Shearing Upon SCAMP-5 PE Array

A standard image shearing operation is performed by effectively shifting each pixel to a new location upon the 2D plane, and then determining a new image from these shifted pixels. This new image is composed of a grid of pixels whose values are formed by interpolating between the values of the shifted pixels. However performing interpolation is challenging upon current SCAMP-5 hardware, and so we instead consider only nearest neighbour image transformations. With a nearest neighbour shear transformation each pixel in the new image is instead a direct copy of the closest shifted pixel. For images stored upon the registers of the SCAMP-5's processor array, this translates to having to move the Pixel data stored within each PE to a new location on the array, that being the PE closest to the Pixel's shifted position.

As described in Section II, each PE is only capable of directly transferring data to its immediate neighbours in the processor array. Data can still be transferred between any two PEs indirectly by performing a sequence of data transfers, shuffling data from one PE to the next across the array, until it has been copied into the desired PE. This data transfer is performed in parallel across all PEs in the array, however the FLAG register in each PE for conditional execution of SIMD instructions can be used to restrict data transfer operation to only select PEs.

It should be clear given these capabilities that image shearing transformations upon SCAMP-5 should be possible, the question remains how to perform a shear efficiently by exploiting the parallel compute of the SCAMP-5.

B. X and Y Shearing Method

Consider the shear parallel to the X axis as given by Equation 1, horizontally shifting each row of pixels by an amount proportional to the row's Y location. Conducting the equivalent nearest neighbour shear operation upon SCAMP-5 would involve repeated data transfers between PEs, shifting stored pixel data horizontally across the processor array.

This horizontal shift of data could be performed one PE row at a time, however this would be slow and highly inefficient. Instead our proposed approach horizontally shifts the pixel data stored across of multiple rows of PEs simultaneously in parallel.

Note again that the SCAMP-5 PPA consists of a 256×256 array of PEs. Taking the origin of a stored image to be at the center of the PE array, when performing a horizontal shear such as in Equation 1, the ith row of PEs will require a horizontal shift r_i as given by Equation 3.

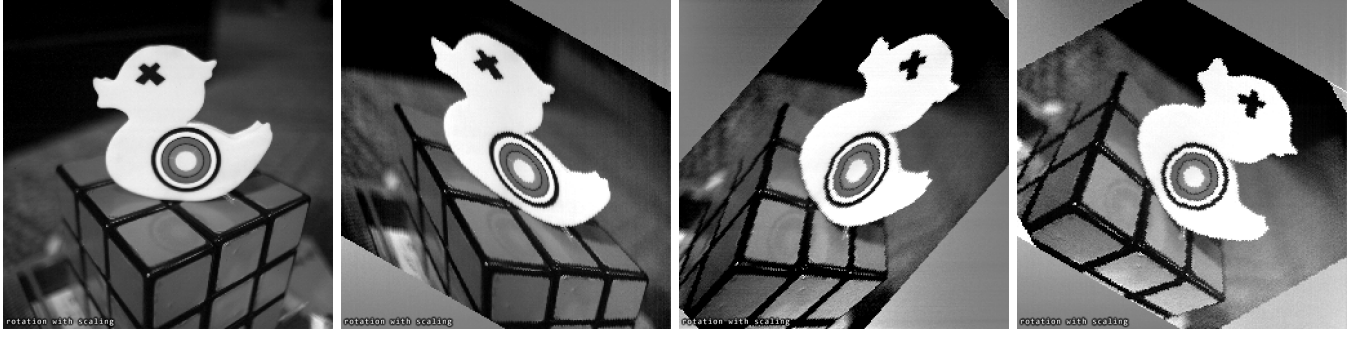$$r_i = Ceil(\alpha(128 - i)) \tag{3}$$

Fig. 5. Example of performing three consecutive shear operations upon an image stored upon SCAMP-5, resulting in an image rotation.

With data in the top and bottom halves of the array being shifted in opposite directions, and with shifts up to a magnitude of $N = Ceil(|\alpha 128|)$. Let S_n denote the set of indices for all rows that require a shift of n as shown in Equations 4.

$$S_n = \{i \mid r_i = n \in \mathbb{Z}\} \qquad (4)$$

Let us assume that $\alpha > 0$, and hence all PE rows in the top half of the array belong to one of the sets $S_1, S_2, S_3...S_N$. To efficiently conduct the shear transformation, the pixel data of these PE rows should be shifted using as few parallel data transfer operations as possible. This can be achieved by performing a single horizontal data transfer operation upon the PE rows of $S_1 \cup S_2 \cup S_3 \cup ...S_N$, then repeating on the rows of $S_2 \cup S_3 \cup S_4 \cup ...S_N$, then $S_3 \cup S_4 \cup S_5 \cup ...S_N$ and so on. Doing so shifts pixel data across multiple rows simultaneously, with each row stopping once its data is correctly shifted according to the shear transformation being performed.

In practice, performing this coordinated data shifting across PE rows requires correct manipulation of the PE FLAG registers controlling conditional execution of SIMD instructions. In the top half of the array the distance to shift each PE row increases going upwards, meaning the FLAG registers within PE rows must be toggled off successively from bottom to top. When viewed as an image, the content of the FLAG registers will then appear as a sweeping curtain, moving upward with each successive horizontally shift as is illustrated in Figure 4. Such a sweeping curtain moving across the FLAG registers can be efficiently created using data transfer operations to shift each PEs FLAG register content upwards.

Once this coordinated shifting of data has been performed for the top half of the PE array, a similar routine can be performed for the bottom half completing the shear operation by transferring all stored pixel into the correct PE locations. The method for this approach is laid out in [1], where Shift(X,DIR) denoted performing a parallel data transfer across all active flagged PEs, copying the content of register X into the same register of a neighbouring PE element. Vertical shearing can be performed in the same manner but now splitting the PE array into left and right halves and vertically shifting columns of PEs in-place of rows.

Algorithm 1 Horizontal Shearing

A *// register holding pixel data in each PE*
$N = Ceil(|\alpha 128|)$ *// Greatest Required Shift*

//Flag all PEs in rows in top half to be shifted
Clear_FLAG (all PEs)
Set_FLAG (PEs in rows from 0 to $Max(S_1)$)

//Shift pixel data in top half of array
for $n = 1$ to N **do**
    //Horizontally shift data in Flagged PEs
    if $\alpha > 0$ **then**
        Shift(A,WEST)
    else
        Shift(A,EAST)
    end if
    //Vertically shift FLAG registers
    $V = Max(S_{n+1}) - Max(S_n)$
    for $i = 0$ to V **do**
        Shift(FLAG,SOUTH)
    end for
end for

//Flag all PEs in rows to be shifted
Clear_FLAG (all PEs)
Set_FLAG (PEs in rows from $Min(S_{-1})$ to 255)

//Shift pixel data in top half of array
for $n = 1$ to N **do**
    //Horizontally shift data in Flagged PEs
    if $\alpha > 0$ **then**
        Shift(A,EAST)
    else
        Shift(A,WEST)
    end if
    //Vertically shift FLAG registers
    $V = Min(S_{-n}) - Min(S_{-(n+1)})$
    for $i = 0$ to V **do**
        Shift(FLAG,NORTH)
    end for
end for

IV. ROTATION BY THREE SHEARS

To rotate images stored upon the SCAMP-5 we make use of the fact that any arbitrary rotation matrix of θ radians as in Equation 5, can be decomposed into a combination of three shear matrices such as shown in Equation 6.

$$\begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} 1 & -tan(\frac{\theta}{2}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ sin(\theta) & 1 \end{bmatrix} \begin{bmatrix} 1 & -tan(\frac{\theta}{2}) \\ 0 & 1 \end{bmatrix} \tag{6}$$

In this case two horizontal shear operations and one vertical in the same form as described previously in Section III. Performing such a set of three shearing operations in sequence will then result in the pixel data being shifted across the PEs of the array such that the appropriate rotated image is produced. An example of such a rotation being performed one shear at a time upon SCAMP-5 is illustrated in Figure 5. The time required to perform such a rotation linearly increases with the rotation angle, with a large rotation of 45 degrees taking $1031\mu s$ to perform.

V. IMAGE SCALING

A typical scaling transformation moves each point (x, y) to a new location $(\alpha x, \beta y)$, bringing said point either closer or further from each of the X and Y axi depending upon scaling factors α and β. Here however we consider how to perform the separate cases of horizontal and vertical scaling operations as shown in Equations 7, 8 upon SCAMP-5. Such horizontal and vertical scaling operations can then be performed in sequence to perform any general image scaling as illustrated in the examples of Figure 7.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \alpha x \\ y \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ \beta y \end{pmatrix} \tag{8}$$

Similar to the image shearing of Section III, we implement nearest neighbour image scaling due to the difficulties in implementing pixel interpolation upon SCAMP-5. Such nearest neighbour scaling involves eliminating or duplicating rows and/or columns of pixel data across the image. Specifically horizontal down-scaling ($\alpha < 1$) involves eliminating columns of pixel data, while horizontal up-scaling ($\alpha > 1$) involves inserting duplicated columns of pixel data. A similar process is performed for vertical scaling, but upon rows of stored pixel data instead. In both cases we regard the origin of the image to be located at the center of the PE array.

A. Scaling Upon SCAMP-5

Duplication or elimination of a row/column of pixel data can be performed on SCAMP-5 using the parallel data transfer operation employed previously for image shearing. By correctly setting up the FLAG registers, limiting the data transfer into
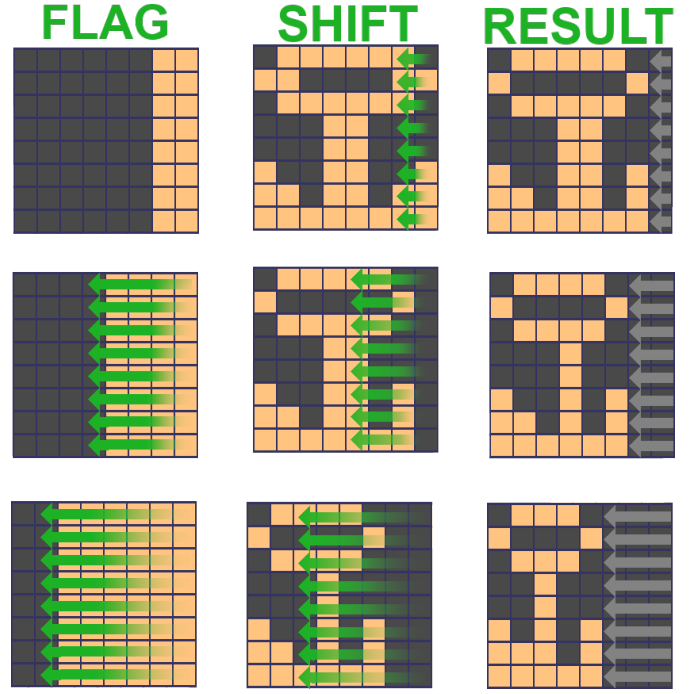

Fig. 6. Illustration of performing horizontal down-scaling over several steps (Top-Right to Bottom-Right). The FLAG register (Left Column) is used to select which columns of PEs should copy data from their rightmost neighbours. This eliminates one column of data in the process by overwriting its content, shrinking the remaining image. The FLAG register content itself is shifted in-between each step, selecting the next column to be eliminated.

only select PEs, this operation can be used not only to shift pixel data but also overwrite it.

Let us examine horizontal down-scaling of an analog image upon SCAMP-5 i.e. $0 < \alpha < 1$ (for $\alpha 1$ in Equation 7), by eliminating columns of pixel data. As the image origin is taken to be the center of the array, this scaling will require horizontally shifting data in the left and right sides of the array in opposite directions, bringing data from both sides towards the array's center. Further, to produce a correctly scaled image the columns eliminated from this data shifting must be evenly spaced across the array. As the PE array is 256 elements in width, the number of columns to eliminate from both the left and right hand sides is given by E in Equation 9

$$E = 128 - Ceil(\alpha 128) \tag{9}$$

with even spacing between these eliminated columns S is then given by Equation 10.

$$K = Ceil(128/E) \tag{10}$$

Let us examine only the right hand side of the array for now. To eliminate the first column of pixel data from this side, the PE FLAG registers are set within that column along with all other columns to the right. A parallel data transfer operation is then performed, instructing each flagged PE to copy over the data from the PE to its right. This causes the pixel data across all flagged columns to be shifted to the left, except in the

Fig. 7. Examples of in-plane image transformations performed on SCAMP-5. Top Left:original, Top Right:Up-scaled, Bottom Left:Up-scaled and Rotated, Bottom right:Down-scaled and Rotated

column to be eliminated, whose pixel data is then overwritten eliminating it from the image.

This process can then be repeated for all remaining columns that must be eliminated from this side of the array. However for each of these subsequent columns, the FLAG registers required to perform this elimination do not need to be setup from scratch. Instead the FLAG register content used for the previous column elimination can itself be shifted horizontally to flag the necessary columns of PEs. Thus, as columns are eliminated the FLAG register content takes the form of a sliding curtain, similar to that used in Section III for image shearing, providing an efficient means to repeatedly eliminate columns of pixel data from the array. This approach is illustrated in Figure 6. The same approach can then be performed to eliminate columns from the left hand side of the image, completing the horizontal down-scaling as listed in Algorithm 2

Vertical down-scaling of an image can be performed in much of the same way, largely just switching the routine from using columns to rows. Similarly up-scaling of an image is performed by in a highly similar fashion, except now duplicating rows or columns of pixel data at evenly spaced intervals, rather than eliminating them. The time taken to perform such scaling operations increases with their magnitude, with up-scaling an image by a factor of 2 (scaling both in x and y) taking $445\mu s$, and down-scaling to half size taking the same time.

VI. Conclusions

This paper presented a set of novel algorithms for conducting image transformations upon PPA devices. In each case we presented a new approach which exploits the parallel processing of the SCAMP-5, but which should also be applicable to PPA architectures in general. Our implementations are fast enough to be used as standard functions in many real-time works providing some essential functions, whose methods implementation are not readily apparent. It is our hope that this paper helps demonstrate how a wide range of tasks are possible on such a device, and that others may use the work presented here to accelerate building their own applications upon pixel-parallel architectures.

Algorithm 2 Horizontal Down-Scaling

A *// register holding pixel data in each PE*
K *// Column skip value determining down-scaling*
$N = Ceil(128/K)$ *//Required Shifts*

//Setup FLAG for scaling right half of array
Clear_FLAG (all PEs)
Set_FLAG (PEs in columns from 0 to K)

//Scale pixel data in right half of array
for $n = 1$ to N **do**
    Shift(A,WEST)   *// Shift data in Flagged PEs*
    for $i = 0$ to K **do**
        Shift(FLAG,WEST)   *//Shift FLAG register content*
    end for
end for

//Setup FLAG for scaling left half of array
Clear_FLAG (all PEs)
Set_FLAG (PEs in columns from $(256 - K)$ to K)

//Scale pixel data in left half of array
for $n = 1$ to N **do**
    Shift(A,EAST)   *// Shift data in Flagged PEs*
    for $i = 0$ to K **do**
        Shift(FLAG,EAST)   *//Shift FLAG register content*
    end for
end for

References

[1] M. J. Duff *et al.*, "Review of the CLIP image processing system," in *Proc. National Computer Conference*. AFIPS Press Arlington, Va, 1978, pp. 1055–1060.

[2] J. C. Gealow, F. P. Herrmann, L. T. Hsu, and C. G. Sodini, "System design for pixel-parallel image processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 32–41, 1996.

[3] M. Ishikawa, K. Ogawa, T. Komuro, and I. Ishii, "A cmos vision chip with simd processing element array for 1ms image processing, 1999 dig. tech. papers of 1999 ieee int," in *Solid-State Circuits Conf.(ISSCC99)(San Francisco, 1999.2. 16)/Abst*, pp. 206–207.

[4] P. Dudek and P. J. Hicks, "A general-purpose cmos vision chip with a processor-per-pixel simd array," in *Proceedings of the 27th European Solid-State Circuits Conference*. IEEE, 2001, pp. 213–216.

[5] J. Poikonen, M. Laiho, and A. Paasio, "MIPA4k: A 64× 64 cell mixed-mode image processor array," in *2009 IEEE International Symposium on Circuits and Systems*. IEEE, 2009, pp. 1927–1930.

[6] A. Lopich and P. Dudek, "A general-purpose vision processor with 160x80 pixel-parallel SIMD processor array," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2017.

[7] A. Rodriguez-Vazquez, J. Fernández-Berni, J. A. Leñero-Bardallo, I. Vornicu, and R. Carmona-Galán, "CMOS vision sensors: embedding computer vision at imaging front-ends," *IEEE Circuits and Systems Magazine*, vol. 18, no. 2, pp. 90–107, 2018.

[8] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535GOPS/W 256× 256 SIMD processor array," in *2013 Symposium on VLSI Circuits*. IEEE, 2013, pp. C182–C183.

[9] T. Yamazaki, H. Katayama, S. Uehara, A. Nose, M. Kobayashi, S. Shida, M. Odahara, K. Takamiya, Y. Hisamatsu, S. Matsumoto *et al.*, "A 1ms high-speed vision chip with 3d-stacked 140 GOPS column-parallel PEs for spatio-temporal image processing," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 82–83.

[10] L. Millet, S. Chevobbe, C. Andriamisaina, L. Benaissa, E. Deschaseaux, E. Beigne, K. B. Chehida, M. Lepecq, M. Darouich, F. Guellec *et al.*, "A 5500-frames/s 85-GOPS/W 3-d stacked BSI vision chip based on parallel in-focal-plane acquisition and processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1096–1105, 2019.

[11] T. Finateu, A. Niwa, D. Matolin, K. Tsuchimoto, A. Mascheroni, E. Reynaud, P. Mostafalu, F. Brady, L. Chotard, F. LeGoff *et al.*, "A 1280× 720 back-illuminated stacked temporal contrast event-based vision sensor with 4.86 μm pixels, 1.066 GEPS readout, programmable event-rate controller and compressive data-formatting pipeline," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 112–114.

[12] J. Chen, S. J. Carey, and P. Dudek, "Feature extraction using a portable vision system," 2017.

[13] L. Bose, J. Chen, S. J. Carey, P. Dudek, and W. Mayol-Cuevas, "A camera that cnns: Towards embedded neural networks on pixel processor arrays," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[14] ——, "Visual odometry for pixel processor arrays," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4604–4612.

[15] J. N. Martel and P. Dudek, "Vision chips with in-pixel processors for high-performance low-power embedded vision systems," in *ASR-MOV Workshop, CGO*, vol. 6, 2016, p. 14.

[16] T. Debrunner, S. Saeedi, and P. H. Kelly, "AUKE: Automatic kernel code generation for an analogue SIMD focal-plane sensor-processor array," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, p. 59, 2019.