Pixel Interlacing to Trade off the Resolution of a Cellular Processor Array against More Registers

Julien N. P. Martel [*], Miguel Chau [†], Matthew Cook [*] and Piotr Dudek [‡]
Email: jmartel@ini.ethz.ch mchau@student.ethz.ch cook@ini.ethz.ch p.dudek@manchester.ac.uk
[*] Institute of Neuroinformatics, University of Zürich / ETH-Zürich, Zürich 8057, Switzerland
[†] Department of Computer Science, ETH-Zürich, Zürich 8092, Switzerland
[‡] Sc. of Electrical & Electronic Engineering, The University of Manchester, Manchester M13 9PL, United-Kingdom

***Abstract*—Recently, several low and mid-level vision algorithms have been successfully demonstrated at high-frame rate on a low power-budget using compact programmable CPA (Cellular Processor Arrays) vision-chips that embed a Processing Element (PE) at each pixel. Because of the inherent constraint in the VLSI design of these devices, algorithms they run are limited to scarce resources, in particular memory – that is the number of registers available per pixel. In this work, we propose an algorithmic procedure to trade off the pixel resolution of a programmable CPA vision-chip against the number of its registers. By grouping pixels into "super-pixels" where pixel registers are interlaced, we virtually expose more registers in software allowing to run more sophisticated algorithms. We implement and demonstrate on an actual device an algorithm that could not have been executed on an existing CPA at full resolution due to its memory requirements.**

I. Introduction

The need of low latency, low power automated vision systems that meet real-world constraints gave rise to the design of hardware devices allowing to directly perform computation in the image plane of an optic system. These so-called vision-chips capture light-intensity in a light-sensitive register which value can be further processed through a processing unit (PU) embedding a dedicated circuitry. Both the PU and the registers are components of a single Processing Element (PE) usually located at each pixel. Such a device can be used to perform massive parallel computation and solely output meaningful pre-processed data – such as the position of a tracked object [1] – thus avoiding the bottleneck of transferring large amount of data such as an entire frame from the image sensor to another device equipped with processing capabilities.

In this work, we consider vision-chips operating in a Single Instruction Multiple Data (SIMD) paradigm where a single instruction is dispatched by a controller to all the PEs and executed simultaneously on their own local registers. The VLSI designs of these devices are subject to many physical and cost constraints, consequently compromises have to be made. For instance, increasing the complexity of the processing circuitry at the pixel level is done at the expense of the fill-factor of the photosensitive element. These design choices restrict the hardware implementation of each PE to having, for instance, only a couple of registers for computation and data storage.

Nevertheless, several vision-chip variants with different pixel processing circuitries have been explored, and fabricated [2], [3]. Specific designs of such devices feature PEs able to communicate with their immediate neighbourhood via a "shared bus" or "register" [1], [4]–[7]. These Cellular Processor Arrays (CPA) are very pertinent for many image processing routines where information needs to be passed efficiently from one pixel (a cell in the array) to another.

This work was funded by SNF grant 143947.

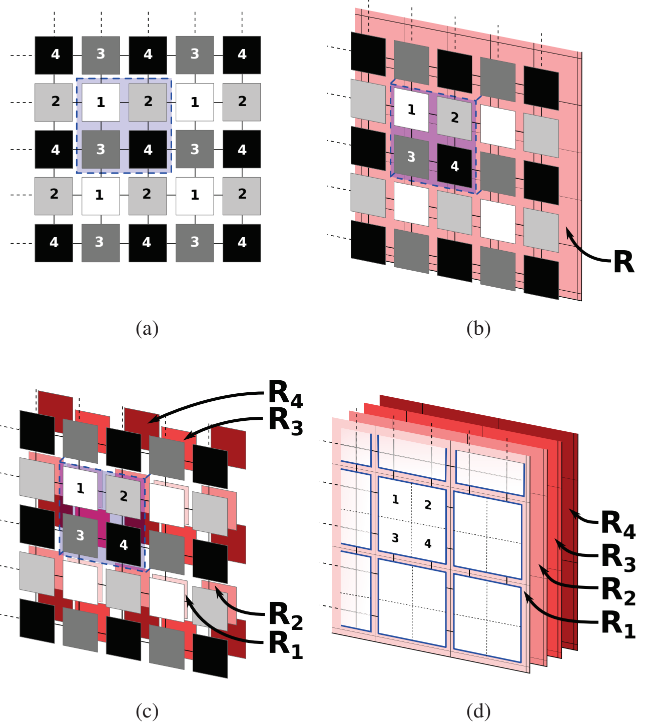

Fig. 1: (a) is an example of the arrangement of pixels in a CPA showing a pattern to group four pixels into super-pixels. (b) shows a single register R at each PE, as designed on hardware. (c) is a view where the register R has been subdivided into four "virtual" registers R_1, R_2, R_3 and R_4 at each pixel of the super-pixel. One can remark the CPA resolution is divided by four since there exists four "virtual" registers only when considering a super-pixel, and a R_i can only be accessed by a PE "i". (d) is an alternative view showing the grouping of four pixels into a super-pixel exposing now four "virtual" registers.

It has been recently demonstrated in simulation that these CPA architectures are well-suited for message-passing algorithms, for instance jointly inferring visual quantities to build mid-level vision systems able to perform on-chip ego-motion estimation [8]. However, the implementation on an actual vision-chip requires more resources than what is constrained by its design. The solution we propose is to trade off available resources on the system in software: concretely by reducing its resolution when "virtually" grouping PEs together in a super-PE.

II. Grouping of pixels in a CPA to interlace array-variables

For the programmable SIMD mode CPAs we consider, registers can be thought as forming an array of data that we refer to as a *data-plane*. If each pixel has n registers, then the device holds n data-planes. A data-plane is used to store an array-variable, for instance the intensity as collected by the light-sensitive registers could be stored in a data-plane while one of the components of a spatial gradient from the light-intensity would require another data-plane.

The grouping of pixels into super-pixels is done by splitting one (or more) data-planes according to the pixel position on-chip so that two pixels grouped into a super-pixel, using the same real data-plane can be used to store two different array-variables. This is what we call the interlacing. Each super-PE exposes now more registers – n grouped PEs give n times more registers –, equivalently more data-planes, which can be used in more sophisticated algorithms.

On CPAs, a corollary of grouping the PEs together is the change of topology: each PE at each pixel has now a different kind of neighbourhood. While the same local registers for different PEs in an ungrouped setting have the same "data-type" as their neighbours, in a grouped setting PEs belonging to super-PEs have neighbours sharing completely independent pieces of information even though they are on-chip in the same data-plane. As a consequence, a careful propagation of information has to be done between registers on the CPA to route it only to relevant places without compromising data on which one does not directly operate.

The idea of clustering pixels and performing partial readout in the data-planes was firstly evoked in [9], however neither any procedure nor any implementation of this concept has been demonstrated to our knowledge despite the flexibility such a procedure offers to design CPA with more pixels and very few registers that can be then used at lower resolution for algorithms requiring more resources.

In the following we detail an example to group four pixels into super-pixels allowing to interlace four array-variables into a single data-plane. We define a way to arrange pixels to group them into super-pixels, we show how to access the virtual registers individually as well as how to keep the CPA property of being able to shift information to the 4-neighbours. We then identify two general components to be able to perform these operations and discuss their role and their instantiation in several CPA hardware.

A. Interlacing and operating on four pixels grouped into super-pixels

1) To define a pattern to interlace four pixels grouped into a super-pixel: As a concrete example, let us say we think of each pixel of a CPA as being labelled with one of the four integers/colors: 1 (white), 2 (light gray), 3 (dark-gray) and 4 (black) to produce an arrangement as seen in Figure 1a.

We will now consider each group of neighbouring pixels constituted of the four different labels, as a super-pixel. If each of the pixels has only one register R (the red array in Figure 1b), each pixel in the super-pixel can be now considered to have its own register R_i split from R as shown in Figure 1c. Then the super-pixel has now virtually four registers as depicted in Figure 1d that will be addressed by accessing R "individually" at each pixel 1, 2, 3 and 4.

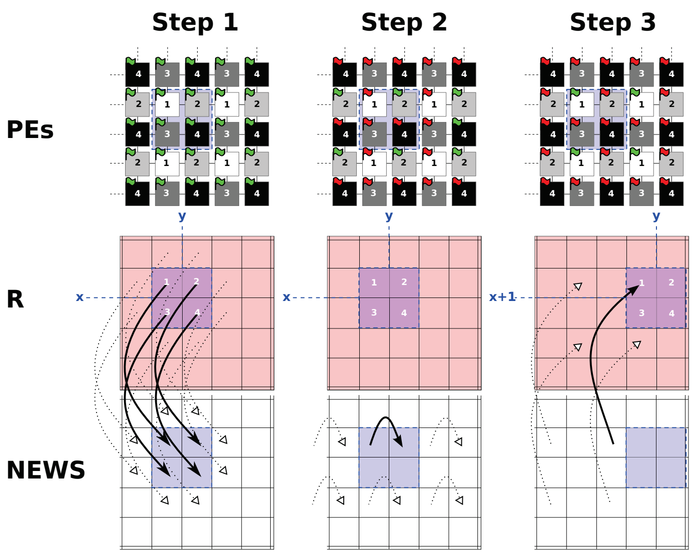

Fig. 2: We show how to execute a shift operation on the CPA to propagate information to the EAST: from the "virtual" register R_1 of the super-pixel at position (x, y) to the "virtual " register R_1 of the super-pixel at position $(x + 1, y)$. The activity flag of the PEs are shown with the following color code: green = "enabled" and red = "disabled". Arrows show the read/write routines that are executed at each step, bold arrows are for the super-pixel of interest while dashed-arrows show what happens for the neighbouring pixels when executing the instruction on the CPA (SIMD mode).

2) To access each of the virtual registers individually: For the PEs grouped into super-PEs we need to be able to operate on a single label independently. For instance, retrieving the content of the register R of pixels 1 – denoted R_1 – consists in operating on the register while "masking" the PEs labelled 2, 3 and 4 to prevent them from working and modifying the content of their own R now they are assumed to contain interlaced array-variables. Because the SIMD mode dispatches the same instruction to all the PEs, when operating on a register/data-plane all the PEs are active irrespective of their label "1", "2" etc. As a consequence the CPA device needs to have a mechanism to be able to operate only on a subset of its pixels –with a particular label– by "masking" the other ones.

3) To propagate the content of the virtual registers individually: In order to expose to the user a CPA which preserves its property of being able to propagate information to its immediate neighbours, a special care needs to be taken for the "shifting operation" since the topology of the array is changed with the interlacing. Precisely, a pixel labelled 1 does not have a 1 neighbour; it means that to propagate information to one of its neighbour it needs to cross a 2 neighbour (if one wants to propagate to EAST or WEST) or a 3 neighbour (if one wants to propagate to NORTH or SOUTH). Because actual connectivity in the CPA only exists between four-adjacent neighbours, we need to take care of shifting first to 2 (resp. 3) in a temporary register when propagating to the EAST/WEST (resp. NORTH/SOUTH) and then shift the content of this temporary register once more to end up into the register of the target 1 pixel. Again, at the chip-level only the R register exists, therefore one has to be careful to "mask" pixels on which one does not want to operate when shifting the content of R_1 not to also modify the content of R_2, R_3 and R_4 as a side effect.

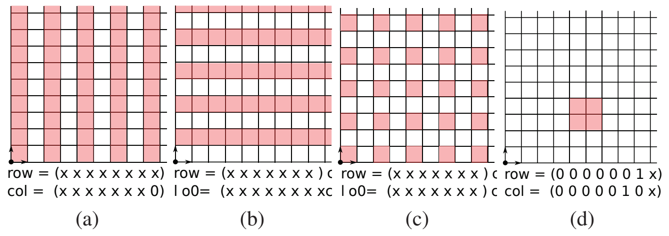

Fig. 3: Four examples of codewords to initialize masking registers from the IPU. (a) and (b) show periodic addressing of rows and columns. (c) shows the periodic selection of a single processor. (d) shows the selection of a particular super pixel in a four pixel grouping setting.

The three steps procedure to shift the content of a register to the EAST from a pixel at (x, y) to a pixel at $(x+1, y)$ is described in the following pseudo-code and is illustrated in Figure 2:

```
Require: $R$ (a register), Masking registers (based on the
    labelling), NEWS (a communication "shared" register)
1: NEWS $\leftarrow$ R // NEWS is shared between adj. neighbs.
2: Mask 1, 3 and 4
3:     NEWS $\leftarrow$ WEST // NEWS$_2$ takes the WEST neighb.'s value
4: Unmask all
5: Mask 2, 3 and 4
6:     R $\leftarrow$ WEST // $R_1$ takes the WEST neighb.'s value
7: Unmask all
```

B. *Components needed in a CPA to operate on virtual registers when pixels are interlaced*

To execute the three-steps procedure we have just outlined, two components are necessary to support the pixel-independent operations of the CPA such as the "individual" access of a particular register and the "shifting" of the content of a register to the immediate neighbours:

An activity flag: is a bistable component whose state indicates whether the PE must operate or not. In the SIMD mode we consider, the activity flag is originally implemented for branching-instructions. Typically one might want this activity flag to be loaded from masking registers that are pre-initialized with appropriate patterns corresponding to labellings of the PE. These patterns specify the possible configurations of activity: for instance all the "1" labelled PEs must operate (mask="off") while "2", "3", and "4" must not (mask="on").

Masking registers: store the patterns of activities that are used to group pixels together. They are loaded in the activity flag to allow individual masking of pixels within the super-pixel and thus allow to perform individual operations on the virtual registers. The masking registers would typically be pre-initialized pixel-wise as described previously. If a circuit performing boolean functions can be used on the masking registers, then only few of them are in fact needed for a four pixel interlacing: one with "0's" (off = PE does not operate) for all even columns of the CPA, another one for the rows and a last one to store any combination of the two other ones. All the other combinations can be trivially computed using operators like (AND, NOT) or (OR, NOT) that have to be present in the boolean circuit. In case masking registers do not exist on-chip then a mechanism needs to be found to change the state of the activity flag of the PE based on predefined activity patterns.

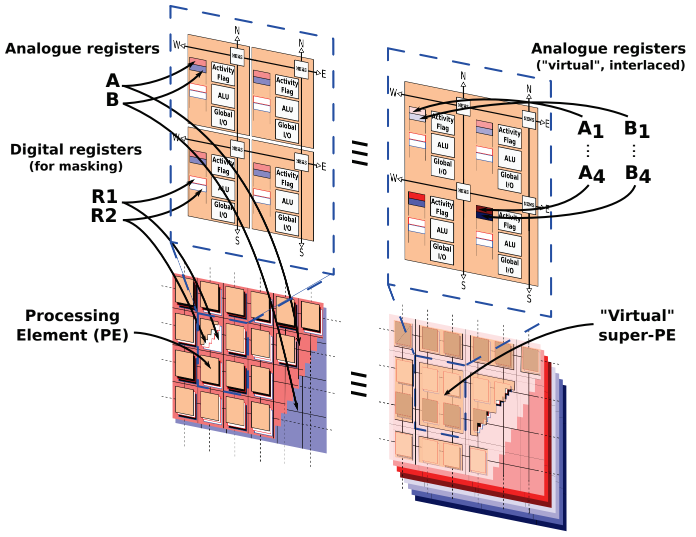

Fig. 4: An instance of a four-pixel interlacing on a sketched version of SCAMP-5 is represented, non-interlaced registers are not shown. In the analogue registers A and B we store using our interlacing scheme: $A_1 = I^t_{x,y}$ as well as $A_2 = I^{t-1}_{x,y}$ the intensities for the current and past frame, $B_1 = V^t_{x,y}$ the temporal gradient, and $A_3 = G^t_{x,y}|_x$, $A_4 = G^t_{x,y}|_y$ the two components of the spatial gradient.

III. Implementation and Results

A. *An implementation on SCAMP-5 vision-chip*

We implemented our interlacing procedure on SCAMP-5, a 256×256 CPA with mixed analogue/digital data-paths consisting of 7 analogue registers (among which one should be reserved for shifting), and 14 single-bit digital registers. A more exhaustive summary of the device's characteristics can be found in [1]. We used the same scheme we presented in Section II where 4 pixels and their PEs are grouped into a super-pixel yielding four times more registers. Thus, when all analog registers are interlaced, it results in a total of 24 "virtual" register-arrays at a resolution of 128×128.

Activity flags and masking registers: exist as such in SCAMP-5. The activity flag can be directly fed with one of the 14 single-bit digital registers. In our implementation we use two of these registers that we initialize such that one has even columns (resp. rows) filled with 1's as shown in Figure 3a and 3b. Since SCAMP-5 embeds circuitries to negate $\neg(\cdot)$ and to disjunct $\vee(\cdot,\cdot)$ digital input registers, other configurations can be created when needed by operating on these two pre-initialized single-bit digital registers. In practice, the values of these registers are initialized by the IPU (Instruction Processing Unit), on a FPGA instructing the CPA vision-chip.

To initialize the digital registers from the IPU: We use the global readout architecture described in [9] to select PEs by generating appropriate codewords to address the array. Each of them is 8-bits so that it is possible to access each of the 256 rows/columns. The codewords are binary, 0's and 1's are used as "do-care bits" and indicate the address of a particular PE. A peculiarity is the introduction of "x" symbols as "don't care bits" which select all the PEs regardless of the bit-value at the "x" position in the address, this provides a flexible mechanism to perform block and periodic selection. Useful examples of codewords for the instantiation of our masking registers are shown in Figure 3.

B. Results: an algorithm using the interlacing

As an example to demonstrate the interlacing procedure we proposed in this work we implemented an algorithm to compute and store 6 array-variables on a SCAMP-5 vision chip: the current intensity frame, the past time step intensity frame, the temporal-gradient, the spatial gradient of the intensity (2 components). In addition, we wrote an algorithm to infer back the light-intensity from the gradients to demonstrate that our interlacing scheme can be used with more sophisticated procedures such as one solving a partial differential equation on the programmable chip.
Note that given that SCAMP-5 offers "only" 6 non-reserved analogue registers [1] it would not be possible to implement an algorithm that can simultaneously compute and store the aforementioned quantities since it would already use 6 registers for the storage. It would not even leave a single temporary register to be used for computation purposes. As an example, a subtraction on SCAMP-5 requires an additional temporary register if one wants to implement an error cancellation scheme to alleviate the mismatch existing between registers [10].
A simplified view of the interlacing and the mapping of the quantities on the SCAMP-5 registers is represented in Figure 4. We use a 4-pixel interlacing for two registers: A (red) and B (blue) to obtain 8 "virtual" registers instead of 2 (8 = 2 reg. $\times$ 4-pix. interlacing). The other registers are not interlaced and are used "as is", we do not represent them on the figure.

Computing the spatio-temporal gradients of the light-intensity is achieved by using standard 1-forward differences as discrete approximations, yielding for the spatial gradient: $\vec{G}^t_{x,y} = (G^t_{x,y}|_x, G^t_{x,y}|_y)^T = (I^t_{x+1,y} - I^t_{x,y}, I^t_{x,y+1} - I^t_{x,y})^T$ and for the temporal gradient: $V^t_{x,y} = I^t_{x,y} - I^{t-1}_{x,y}$. Both these quantities need to shift information on the CPA to be computed, thus exhibiting the shifting procedure of the interlacing. Inferring back the ligth-intensity from the gradients is performed by solving iteratively the Poisson-equation: $\triangle I^t_{\text{rec},x,y} = \vec{\nabla} \cdot \vec{G}^t_{x,y}$, which needs to shift the variable I^t_{rec} in all directions and operate on two data-planes: I^t_{rec} and $\vec{G}^t$.

To check the consistency of our interlacing procedure we transfer the frame readings of the two interlaced registers on a computer on which we perform their deinterlacing to visualize them. The software to deinterlace them is trivial and simply consists in reading the interlaced frames by skipping non-appropriate pixels. It could also be done in hardware by the use of a periodic selection as a readout [9]. In Figure 5 we show both a direct full-frame readout of two interlaced registers from our SCAMP-5 setup and the deinterlaced frames.

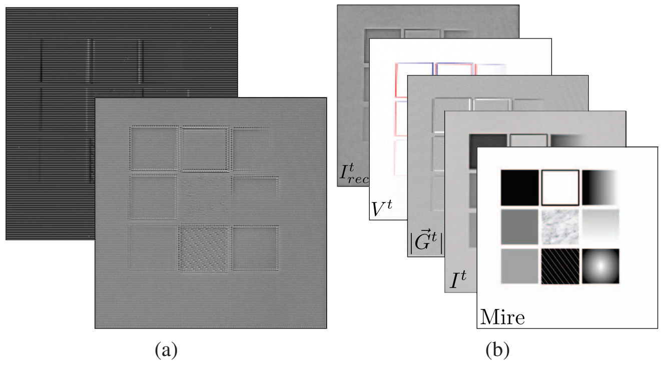

Fig. 5: (a) shows a full-frame readout of the interlaced registers A and B on SCAMP-5. (b) shows the mire and as examples, the quantities $I^t_{x,y}$, $|\vec{G}^t_{x,y}|$, $V^t_{x,y}$, $I^t_{\text{rec},x,y}$ de-interlaced from the registers A and B.

IV. Conclusion

We presented a procedure to trade off the resolution of a CPA against the number of its registers by grouping n PEs at the pixel level in super-PEs. These super-PEs have n times more registers. Each of them can be efficiently addressed and accessed by masking PEs we do not want to operate on by using an activity flag. Masking registers and activity flags are common among the CPA architectures we surveyed [1], [6], [7], [11], [12] even though their implementation varies. We showed it is possible to preserve the property of CPAs to shift information to their neighbours by alternatively masking PEs in the super-PEs and shifting their content within and across the super-PEs. Also, we demonstrated on an actual device that the use of the interlaced PE scheme we introduced in this work allows us to execute an algorithm which could have not been implemented otherwise due to the lack of memory resources for storage and computation. We think that this procedure to trade-off the resolution of CPA vision-chips against more registers offers the opportunity to create devices built with many pixels and few registers, that we can trade-off a-posteriori in software with the help, for instance, of compilers to make this interlacing scheme and the addressing of PEs within super-PEs transparent to the user.

References

[1] S. Carey *et al.*, "A 100'000 fps vision sensor with embedded 535 gops/w 256$\times$256 simd processor array," in *Proc. of the VLSI Circuits Symp. 2013*, 2013, pp. C182–C183.

[2] J. Fernandez-Berni, R. Carmona-Galan, and L. Carranza-Gonzalez, "Flip-q: A qcif resolution focal-plane array for low-power image processing," *IEEE J. Solid-State Circuits*, vol. 46, no. 3, pp. 669–680, 2011.

[3] A. Zarandy, Ed., *Focal-plane Sensor-Processing Chips*. Springer, 2011.

[4] M. Ishikawa *et al.*, "A cmos vision chip with simd processing element array for 1ms image processing," in *Digest of Technical Papers, IEEE Internat. Solid-State Circuits Conf., ISSCC 1999*, 1999, pp. 206–207.

[5] J.-E. Eklund, C. Svensson, and A. Astrom, "Vlsi implementation of a focal plane image processor - a realization of the near-sensor image processing concept," *IEEE Trans. VLSI Syst.*, vol. 4, no. 3, pp. 322–335, 1996.

[6] M. Laiho, J. Poikonen, and A. Paasio, *Focal-plane Sensor-Processing Chips*. Springer, 2011, ch. MIPA4k: Mixed-Mode Cellular Processor Array, pp. 45–71.

[7] A. Lopich and P. Dudek, "An 80$\times$80 general-purpose digital vision chip in 0.18 μm cmos technology," in *Proc. of the IEEE Internat. Symp. on Circuits and Systems, ISCAS 2010*, 2010, pp. 4257–4260.

[8] J. Martel *et al.*, "Toward joint approximate inference of visual quantities on cellular processor arrays," in *Proc. of the IEEE Internat. Symp. on Circuits and Systems, ISCAS 2015*, 2015.

[9] P. Dudek, "A flexible global readout architecture for an analogue simd vision chip," in *Proc. of the IEEE Internat. Symp. on Circuits and Systems, ISCAS 2003*, vol. 3, 2003, pp. 782–785.

[10] P. Dudek and P. Hicks, "An analogue simd focal-plane processor array," in *Proc. of the IEEE Internat. Symp. on Circuits and Systems, ISCAS 2001*, vol. 4, 2001, pp. 490–493.

[11] W. Zhang, Q. Fu, and N. Wu, "A programmable vision chip based on multiple levels of parallel processors," *IEEE J. Solid-State Circuits*, vol. 46, no. 9, pp. 2132–2147, 2011.

[12] A. Rodriguez-Vazquez *et al.*, *Cellular nanoscale sensory wave computing*. Springer, 2010, ch. A CMOS vision system on-chip with multi-core, cellular sensory processing frontend, pp. 129–146.