Toward Joint Approximate Inference of Visual Quantities on Cellular Processor Arrays

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

***Abstract*—The interacting visual maps (IVM) algorithm introduced in [1] is able to perform the joint approximate inference of several visual quantities such as optic-flow, gray-level intensities and ego-motion, using a sparse input coming from a neuromorphic dynamic vision sensor (DVS). We show that features of the model such as the intrinsic parallelism and distributed nature of its computation make it a natural candidate to benefit from the cellular processor array (CPA) hardware architecture. We have now implemented the IVM algorithm on a general-purpose CPA simulator, and here we present results of our simulations and demonstrate that the IVM algorithm indeed naturally fits the CPA architecture. Our work indicates that extended versions of the IVM algorithm could benefit greatly from a dedicated hardware implementation, eventually yielding a high speed, low power visual odometry chip.**

I. Introduction

In a visual information processing system, raw observations made by a sensor are exploited to infer quantities of interest such as optic-flow, ego-motion, or other useful quantities. Inferring these quantities given image frames has been addressed by computer vision since the 60's [2], and remains an active research topic [3]. Recently a new paradigm called event-based vision has emerged with the development of neuromorphic sensors inspired by our retina, reporting asynchronous changes in light-intensity rather than image frames [4]. Both traditional computer vision and event-based vision generally address the problem of inferring other quantities of interest in a *pipelined* fashion using a series of processing blocks.

In Section II we will review features of the model of the interacting visual maps (IVM) originally proposed by Cook et al. in [1]. Unlike more traditional approaches that use a pipelined sequence of algorithmic blocks feeding the output of each block as input to the next, this model softly enforces *relations* to be satisfied between quantities represented by units connected in a *network*. Data flows bidirectionally on each link so as to perform approximate inference on all parts of the model simultaneously, jointly inferring the values of the non-observable quantities. Units containing relevant quantities such as gray-level intensity or optic flow are connected by explicitly-provided relationships between them. The algorithm consists of passing messages between the units, with each unit updating itself to better satisfy its relations with its neighbors. This approach is extendable and robust, and uses a localized style of computation that suggests the use of parallel hardware.

This work was funded by SNF grant 143947.

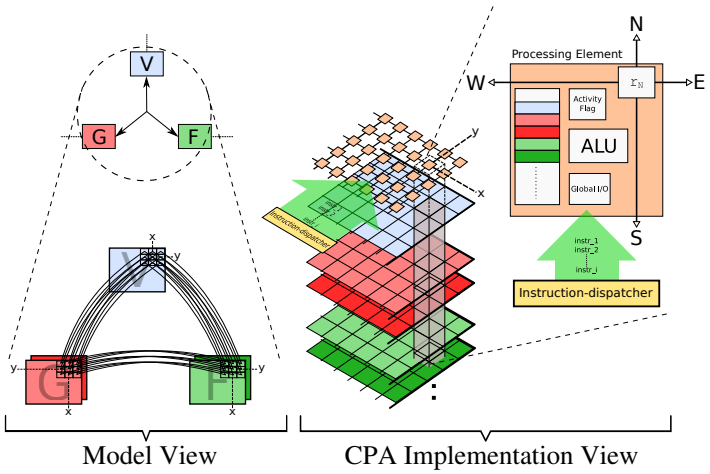
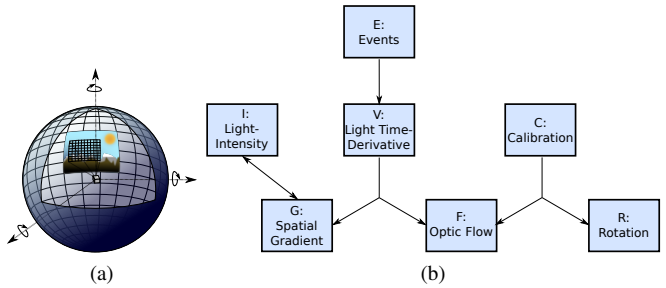


Fig. 1: Schematic view showing the model of the Interacting Visual Maps (IVM) as implemented on a general CPA architecture.
In color we show how units in the model map to registers on processing elements (PEs) of the CPA. All the PEs' same register (represented with the same color in the figure) define a data-plane. All the values of all the units (x, y) in a map of the model are stored in a data-plane at the corresponding (x, y) location.
One PE holds in its registers the values of homologous units at position (x, y) in the different maps of the model.
The NEWS (North, East, West, South) communication plane allows PEs to communicate with their neighbors and is essential for relations in the model that involve adjacent units.

In Section III we will discuss the implementation of the model on a *Cellular Processor Array* (CPA) simulator [5], with a view toward eventual embedding in an integrated circuit. The use of CPAs is motivated by the inherent locality of the computation, its homogeneity for units representing the same quantity, and the ability to efficiently communicate information to adjacent cells. This allows us to make efficient use of these chips to run the message-passing algorithm and shows that CPAs are excellent platforms for this purpose. We present results of inference carried out on a CPA simulator with real-world data, as well as the hardware resources required.

The good performance of our implementation suggests that in the future an extended model could also be implemented, perhaps on extended CPA hardware, providing a complete visual odometry system, a possibility we discuss in Section IV.

978-1-4799-8391-9/15/$31.00 ©2015 IEEE



(a) (b)

Fig. 2: (a) The system being modelled by the interacting visual maps. (b) A representation of the interacting visual maps model we use [1]. The arrows indicate the flow of information. The links between I, G, F, and R are bi-directional. allowing these four quantities to converge to mutually optimally satisfy the relations.

II. Performing joint approximate inference on several visual quantities: The IVM approach

A. Quantities and Relations

The model of the interacting visual maps was first introduced in [1]. It consists of a network of relations between units containing visual quantities such as optic flow or spatial gradient of the grayscale image. Each relation consists of an equation relating some of the visual quantities. Using these equations, the value of one of the visual quantities can be updated so as to better satisfy the equation, given the values the other quantities in the equation. For instance, let us say a unit represents the absolute light-intensity at a certain position $I_{x,y}$ as reported by a camera, and another unit represents events $E_{x,y}$ as reported by a dynamic vision sensor (DVS) [4]. Contrary to traditional cameras reporting frames of integrated light-intensity over a clocked interval that is the same for all pixels, a DVS tracks the light intensity pixel-wise and sends an asynchronous single-pixel event whenever the intensity level increases or decreases by a certain percentage. While traditional cameras report absolute light-intensities, dynamic vision sensors report a signed polarity $\{+1, -1\}$ indicating only a percentage change in the light-intensity. We accumulate these events over a fixed time interval to approximate the temporal derivative of the log light intensity. The equation $E_{x,y} \sim \frac{\partial I_{x,y}}{\partial t} = V_{x,y}$ is an instance of a relation that models the interaction between two units, E and V. The complete network we used for this work is the one described in [1], summarized in Figure 2.

B. A simple message-passing algorithm for approximate inference

A simple message passing algorithm is used to update the quantities stored in the network. Each relation is used to update the quantities it relates. An update will adjust one of the quantities (which we will refer to as the target) based on the other quantities involved, so as to bring the equation a little bit closer to being satisfied. Doing this repeatedly for the various possible targets of the various relations brings the network toward a local optimum, with the relations satisfied as well as possible given the inputs, which are not modified. The convergence of this process can be understood by viewing the updates as performing gradient descent on an error measure consisting of the sum of the squared errors of all the equations [1], or equivalently as converging to a local maximum a posteriori (MAP) estimate of the variable values, assuming Gaussian distributions for the errors of the equations.

To update the value q of a target quantity, the set of values $\mathcal{Q}$ of the source quantities serve to compute a message $\mu_R^t(\mathcal{Q}) = R(\mathcal{Q}^t)$ using the selected relation R as a function $R : \mathcal{Q} \rightarrow R(\mathcal{Q})$ providing the nearest value of the target quantity that would satisfy the relation.. The superscript t stands for the current time step (before the update takes place). Then, to update the target quantity, its current value is blended with the incoming message using the equation:

$$q^{t+1} = (1-\alpha) \cdot q^t + \alpha \cdot \mu_R^t(\mathcal{Q}),\ \alpha \in [0;1] \qquad (1)$$

where α is a hyper-parameter controlling the rate at which the blending occurs.

As a concrete example, let us write the update-equation in the simple case of changing the spatial gradient of the light intensity $\vec{G}_{x,y}$ given the light-intensity itself $I_{x,y}$. There is a single source quantity involved in this relation, $\mathcal{Q} = \{I_{x,y}\}$, and the target is $\vec{G}_{x,y}$. The relation expressed in a functional form for this direction is $R : I_{x,y} \rightarrow \vec{\nabla}(I_{x,y})$. Hence, the update equation becomes in this particular case:

$$\forall (x,y),\ \vec{G}_{x,y}^{t+1} = (1-\alpha) \cdot \vec{G}_{x,y}^{t} + \alpha \cdot \underbrace{\vec{\nabla}(I_{x,y}^t)}_{\mu_R^t(I_{x,y})} \qquad (2)$$

C. Features of the framework

General advantages of the model include its extendability and robustness:

Extendibility is a property of the network in the sense that one does not need to develop any new algorithms when adding new quantities to the network. When new quantities are added, one simply needs to be able to state one or more relations relating the new quantities to the existing quantities. (Of course if the new quantity is not related to the existing quantities in any known way, then there is not much you can do, unless the unknown relationship can be learned from experience, an approach we do not discuss here.)

Extendibility can also be discussed in terms of the sensors that can be connected to the network. A sensor reporting a quantity already modelled can be added to the network simply by feeding the sensor's output to the appropriate units. The model does not need to change when different sensors are used, and in fact multiple sensors can be used simultaneously, in which case the model will perform sensor fusion on the supplied quantities.

Robustness is a crucial issue in a practical system. It turns out that jointly inferring several quantities makes the system inherently robust. Instead of propagating errors in a feed-forward pipeline, the recurrent nature of the model tends to dampen irregularities as the good data "outvotes" the bad, making the system robust to noise and failures.

In terms of computation, the model exhibits interesting features that we exploit to be able to efficiently implement it within a CPA architecture:

Conditional independence exists between maps in the network. Specifically, in a relation, the target unit can be updated independently of all the other units in the network given the source units involved in the relation. Using the Bayesian vocabulary, one would say that the source units of a relation-based function form the Markov Blanket of the target unit. This Markov Blanket lets us benefit from distributivity and parallelism, since two distinct relations using different sets of units can be updated at the same time independently.

Computation locality exists between units in different maps. Most of the relations designed in the network described in Section II-A only relate a value $A_{x,y}$ to its direct counterpart in another map, $B_{x,y}$. In other words, units from different maps only need to communicate if they are located in homologous (x, y) cells. This naturally suggests implementing the model on an architecture where the data lies in array planes interconnected according to the (x, y) coordinates, so that a cell $A_{x,y}$ can communicate with the homologous cell $B_{x,y}$ very efficiently.

In fact, only the relation involving $\vec{F}_{x,y}$, $\vec{R}$ and $\vec{C}_{x,y}$ is non-local, due to $\vec{R}$ being a global property of the system (as opposed to being an (x, y) map). The single 3D vector $\vec{R}$ is inferred from all the cells in $\vec{F}_{x,y}$, making use of the corresponding $\vec{C}_{x,y}$.

Computation homogeneity exists between units in the same maps. The operations that are carried out to update units in a map involved in a relation are the same for each unit in each cell. An operation involving two cells in the arrays A and B at position (x, y) can be done at the same time and independently of the same operation at another pixel (x', y').

III. Implementing the model using a cellular processor array architecture

A. Matching the interacting visual maps with CPAs

In Section II-C we detailed computational features that make the IVM model suited to efficient implementation on a CPA architecture. CPA chips such as ASPA-3 [6] or SCAMP-5 [7] appear to be a close fit for the IVM algorithm, and such an implementation would allow us to run the IVM with significant practical advantages such as low latency, low power-consumption, and a small form-factor. However, before trying to implement these hard-to-debug algorithms directly on a chip, it makes sense to try them on a simulator first (already a non-trivial undertaking), and so this is the approach we have taken.

CPAs are massively parallel arrays working in a single-instruction-multiple-data (SIMD) mode, enabling per-cell operations as well as value transfers to neighboring cells and "inter-plane" communications. CPAs generally consist of several processing elements (PEs), typically one per cell, augmented with one or more registers (constituting the data planes). A central instruction-dispatcher loads instructions into each PE. CPAs come in programmable versions and might also embed a light-sensitive register allowing focal-plane computation, *i.e.*, computation and image acquisition are done at the same place. The CPA architecture avoids the major traditional bottleneck of transmitting acquired data to one or more processing units through a bus of limited bandwidth.

In the previous section we pointed out the local and homogeneous nature of the computations happening in the IVM. Locality is the key property allowing us to perform the computations at each pixel location (each cell) in parallel, and thanks to homogeneity it is possible to execute the same instruction at each of these locations. The CPA architecture is clearly a good match for the IVM algorithm.

B. Demonstrating the feasibility with an implementation of the model on APRON

APRON [5] is a software environment designed to efficiently simulate CPAs. It allows modelling, design, and prototyping of customised CPA-architectures as well as the simulation of CPA algorithms. The simulator mainly consists of two components: the *simulation core*, which provides a virtual CPA, and the IDE, a suite including a compiler, simulator and graphical user interface. The core can be fed with a machine-level *instruction code word* (ICW) stream which is then executed. The compiler transforms a custom language, APRON-script, into an ICW stream to be passed to the simulator. The simulator uses the core to execute the ICW stream in an interactive mode, allowing code inspection and debugging. One feature of APRON is the ability to specify custom *translation rules* to generate different ICW streams that "can be delivered to other applications, and even hardware devices" [5]. However, to execute these ICW streams, customised simulation cores must be provided.

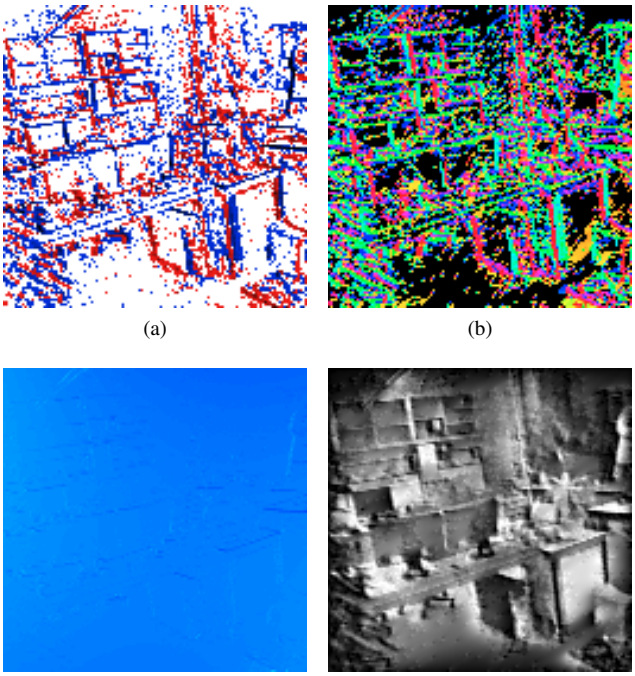
APRON allows us to create *registers*. A register corresponds to one of the data-planes visible in Figure 1. Operations on such registers are always executed in parallel, element-wise on the data at that (x, y) position in the register. To illustrate this local mode of operation, as well as the communication feature of CPAs which enables cells to communicate with their neighbors, we exhibit the following APRON-script implementation of Equation (2), which computes part of the gradient and then uses a previously defined macro to blend it.

```
// Shift intensity map to the right
r[r_temp] = shift.west(r[r_iMap],0)
// Substract intensity map
r[r_temp] = r[r_temp] - r[r_iMap]
// Blend (Message-passing procedure)
r[r_gxMap] = f_blend(r[r_temp], v_ItoGEta)
```

A nice aspect of APRON is that it allows us to write code abstracting ourselves away from just targeting a single CPA chip. Hence, it is perfectly suited to test how the model would behave on a general CPA architecture. Operations that we find are carried out frequently in the model, such as dot products between local registers, suggest specific capabilities that could be built into future dedicated CPA hardware, and in the meantime could be implemented with custom translation rules and simulation cores.

C. Results

We show results of the inference performed in the APRON simulator in Figure 3, and present the required resources for the implementation in Table I. These numbers can help if one is estimating the approximate frame rate or power consumption to be expected when targeting a particular CPA device.



(a) (b) (c) (d)

Fig. 3: Simulation results on the APRON simulator. (a) 25ms of binned events produced by a DVS128 [4], the only input to our simulation. Red represents "+1" events and blue represents "-1" events. (b) The inferred spatial gradient. The vector angle is color-coded by the hue. (c) The inferred optic-flow, again encoded by hue. (d) The inferred intensity.

D. Moving towards a chip implementation

APRON does not impose any constraints on the size or number of registers; it stores floating point values and virtually any kind of arithmetic operation can be performed. A real chip in contrast has only a limited number of analog and digital registers, with poor precision compared to floating point numbers. Also, PEs have a reduced set of arithmetic operations built into their circuitry, for example, division is typically unavailable. Further hardware issues include systematic errors resulting from the analogue components. Some of these issues can be resolved by using more complex instruction sequences so that these systematic errors cancel each other out, but any remaining such issues will have to be analyzed with respect to their effect on performance.

Another difference between the simulation we ran and the CPA vision-chips we consider is the nature of the input to the network: The simulation used intensity-change events while the vision chips report light intensity directly. However, this does not pose a problem for the CPA chips, since the time derivative of light intensity can be provided directly from the light intensity values of successive frames.

IV. Future work

The model we describe in Section II can easily be extended to account for translations in addition to the current estimation of its rotation. Because translation induces motion-parallax, the optic-flow $\vec{F}_{x,y}$ would have to be decomposed into a scene shift component $\vec{S}_{x,y}$, serving the same role as the optic flow map $\vec{F}_{x,y}$ in the current system, plus a perspective component $\vec{P}_{x,y}$, which would contain the effects of motion parallax, being connected to a depth map $D_{x,y}$ and a global translation vector $\vec{T}$. Because of the design of the model, as outlined in Section II-C, adding these components will not require the development of a new algorithm, but simply a single parallax equation. Being able to infer the relative depth as well as the full six-degree-of-freedom ego-motion are the next steps toward achieving a full visual odometry system.

TABLE I: Required hardware resources in APRON to implement the interacting visual maps. "#ops" is the number of instructions used per relation. "#registers" is the total number of registers used per PE for the target quantity. "#add'l registers" is the number of registers used to store temporary results of computations, reused by all calculations.

Quantities	#registers	Relations	#ops	#add'l registers	
$E_{x,y}$	1	$V_{x,y} \leftarrow E_{x,y}$	5	temp.	5
$V_{x,y}$	1	$\vec{G}_{x,y} \leftarrow V_{x,y}, \vec{F}_{x,y}$	33		
$I_{x,y}$	1	$\vec{F}_{x,y} \leftarrow V_{x,y}, \vec{G}_{x,y}$	33		
$\vec{G}_{x,y}$	2	$\vec{F}_{x,y} \leftarrow \vec{R}, \vec{C}_{x,y}$	53		
$\vec{F}_{x,y}$	2	$\vec{R} \leftarrow \vec{F}_{x,y}, \vec{C}_{x,y}$	40		
$\vec{R}$	0	$I_{x,y} \leftarrow \vec{G}_{x,y}$	28		
$\vec{C}_{x,y}$	9	$\vec{G}_{x,y} \leftarrow I_{x,y}$	30		

V. Conclusion

Our implementation of the interacting visual maps worked well in APRON, a CPA simulator, when applied to real-world data. The system was able to perform joint approximate inference of visually relevant quantities, such as optic flow or brightness, based on visual input from a neuromorphic dynamic vision sensor. This is the first step toward extending the model to a full visual odometry system able to run on a real CPA chip meeting real-world constraints.

We think that many problems can be similarly cast into the form of a network of relations, using units representing physical quantities interconnected by equations encoding their relationships. This recasting could be beneficial both for model simplicity and robustness, and for implementability on dedicated architectures like CPAs.

References

[1] M. Cook, L. Gugelmann, F. Jug, C. Krautz, and A. Steger, "Interacting maps for fast visual interpretation," in *Proc. of the International Joint Conference on Neural Networks, IJCNN 2011*, Aug. 2011, pp. 770–776.

[2] L. G. Roberts, "Machine perception of three-dimensional solids," Ph.D. dissertation, Massachusetts Institute of Technology, Dept. of Electrical Engineering, Jul. 1963.

[3] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2nd ed. Prentice Hall, 2011.

[4] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[5] D. R. W. Barr and P. Dudek, "Apron: A cellular processor array simulation and hardware design tool," *EURASIP Journal on Advances in Signal Processing*, pp. 1–9, 2009.

[6] A. Lopich and P. Dudek, "A general-purpose vision processor with 160x80 pixel-parallel simd processor array," in *Proc. of the IEEE Custom Integrated Circuits Conference, CICC 2013*, Sept. 2013, pp. 1–4.

[7] S. J. Carey, D. R. W. Barr, A. Lopich, and P. Dudek, "A 100'000 fps vision sensor with embedded 535 gops/w 256×256 simd processor array," in *Proc. of the VLSI Circuits Symposium 2013*, Jun. 2013, pp. C182–C183.