# Sparse-matrix representation of SNP systems for GPUs

15th Brainstorming Week on Membrane Computing, 2017, Sevilla, Spain

Miguel Ángel Martínez-del-Amor[1]
**David Orellana-Martín**[1]
Francis G. Cabarle[2]
Henry N. Adorna[2]
Mario J. Pérez-Jiménez[1]

*1 Research Group on Natural Computing. University of Sevilla, Spain*
*2 Dept of Computer Science, University of the Philippines Diliman, Philippines*

# Outline

- Motivation
- Sparse matrices
- Proposals
  - SNP systems
  - SNP systems with division
  - SNP systems with buddy
  - SNP systems with plasticity

# Motivation

- Transition of a SNP by matrix representation:
  - Degree m, with n rules

$$C_{t+1} = C_t + S_t * M_{\Pi}$$

Configuration vector: size m

Spiking vector: size n

Transition matrix: size m*n

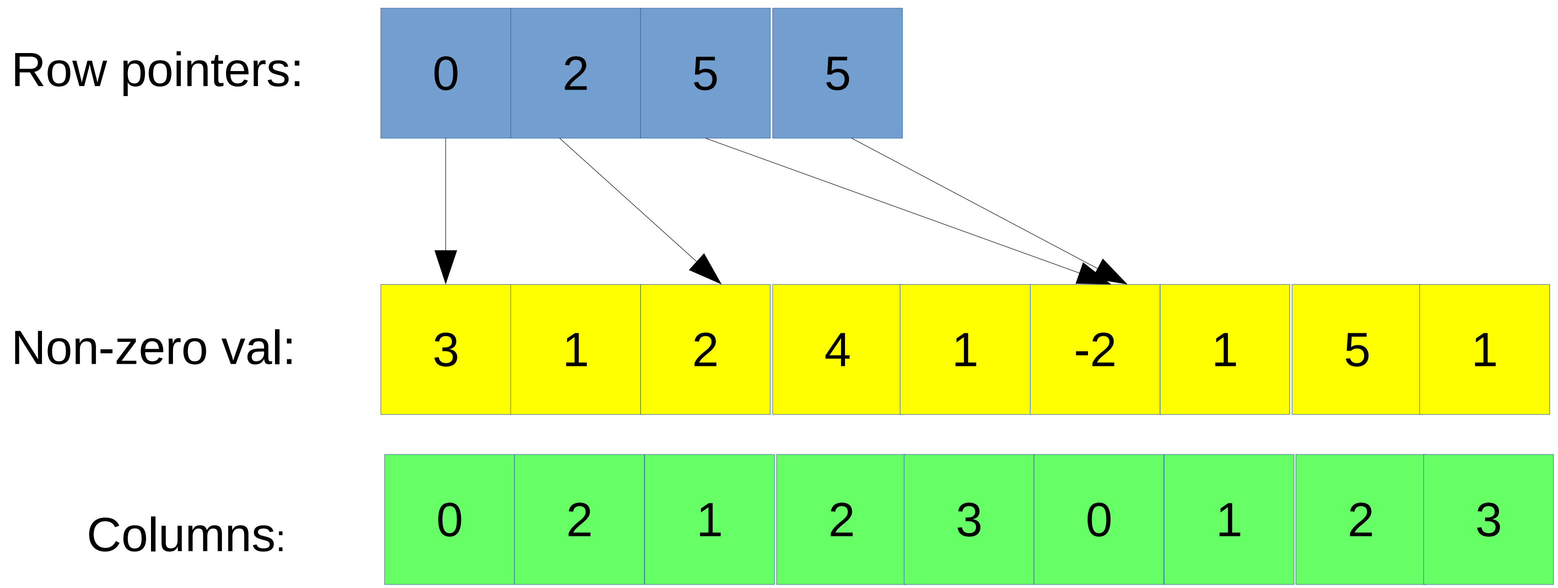
Full of 0's

# SpMV: Sparse Matrix Vector operations

- Reduce size of matrix representation
  - Save memory
  - Save extra operations
- Optimized for GPUs.
- Recall that threads in a GPU should:
  - make coalesced access to mem. (contiguous data)
  - be synchronized (execute same instructions)
- Formats: CSR and ELL

# CSR format

$$\begin{pmatrix} 3 & 0 & 1 & 0 \\ 0 & 2 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ -2 & 1 & 5 & 1 \end{pmatrix}$$




Worth: #non-zero val < #zero val * 2 + #rows

# ELL format

$$\begin{pmatrix} 3 & 0 & 1 & 0 \\ 0 & 2 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ -2 & 1 & 5 & 1 \end{pmatrix}$$

Column

Value

| | | | |
|---|---|---|---|
| (0,3) | (1,2) | X | (0,-2) |
| (2,1) | (2,4) | X | (1,1) |
| X | (3,1) | X | (2,5) |
| X | X | X | (3,1) |

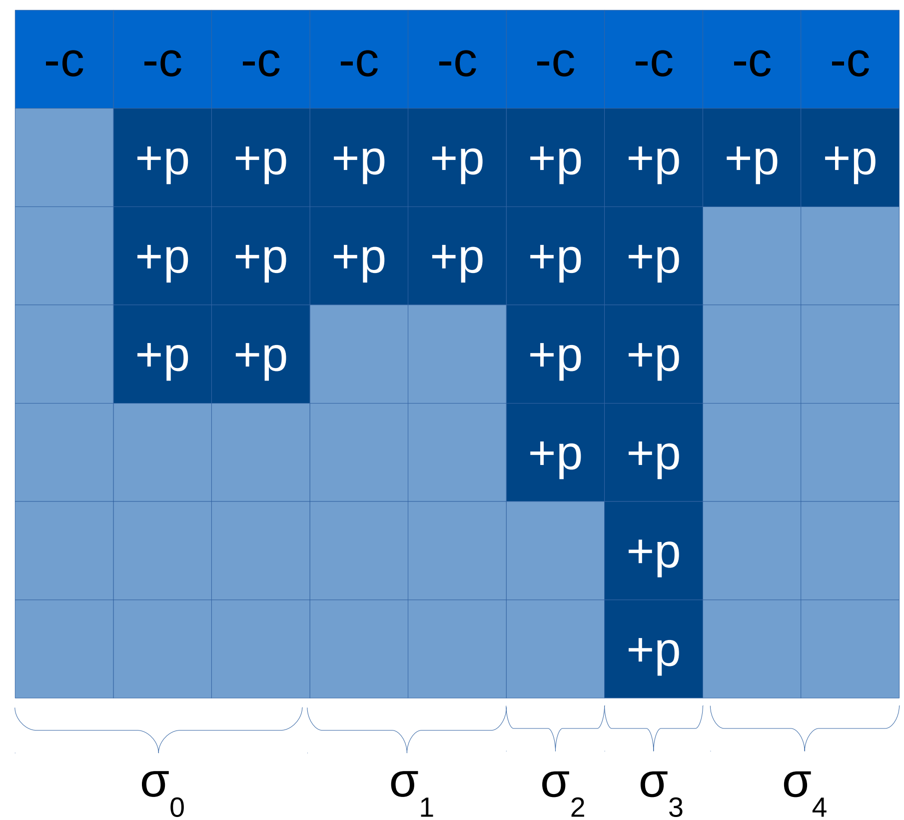
Largest amount
Non-zero values in a row

Worth: length largest row * #rows * 2 < #rows*#columns

# Ideas

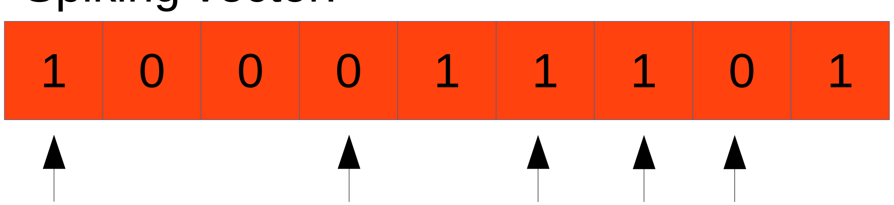
- Take advantage of ELL format
  - For each row (now column), define max size (Z)
  - If we can bound Z, there is room for new values

Sparse transition matrix:



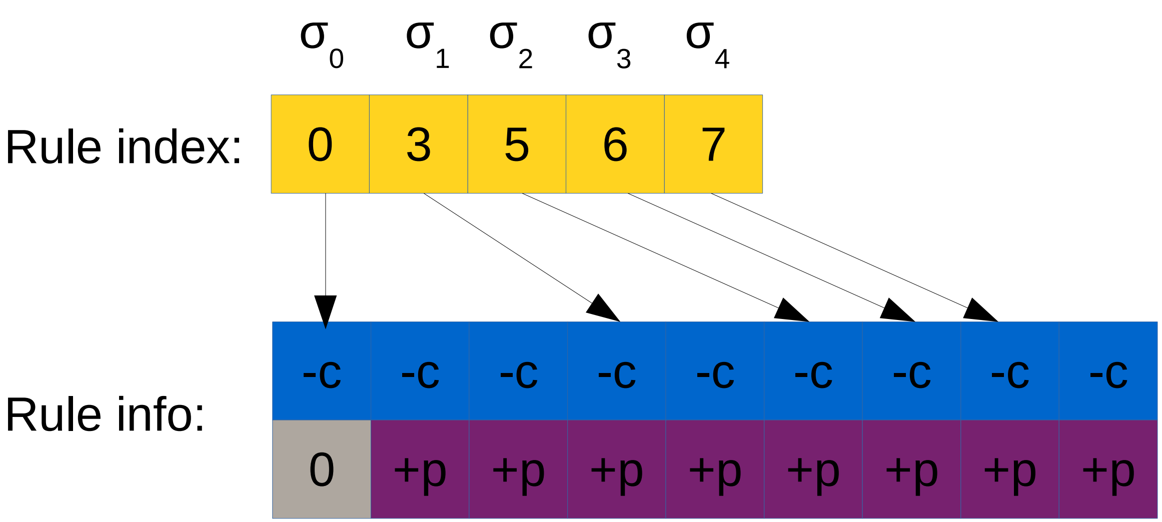

$$E/a^c \rightarrow a^p$$

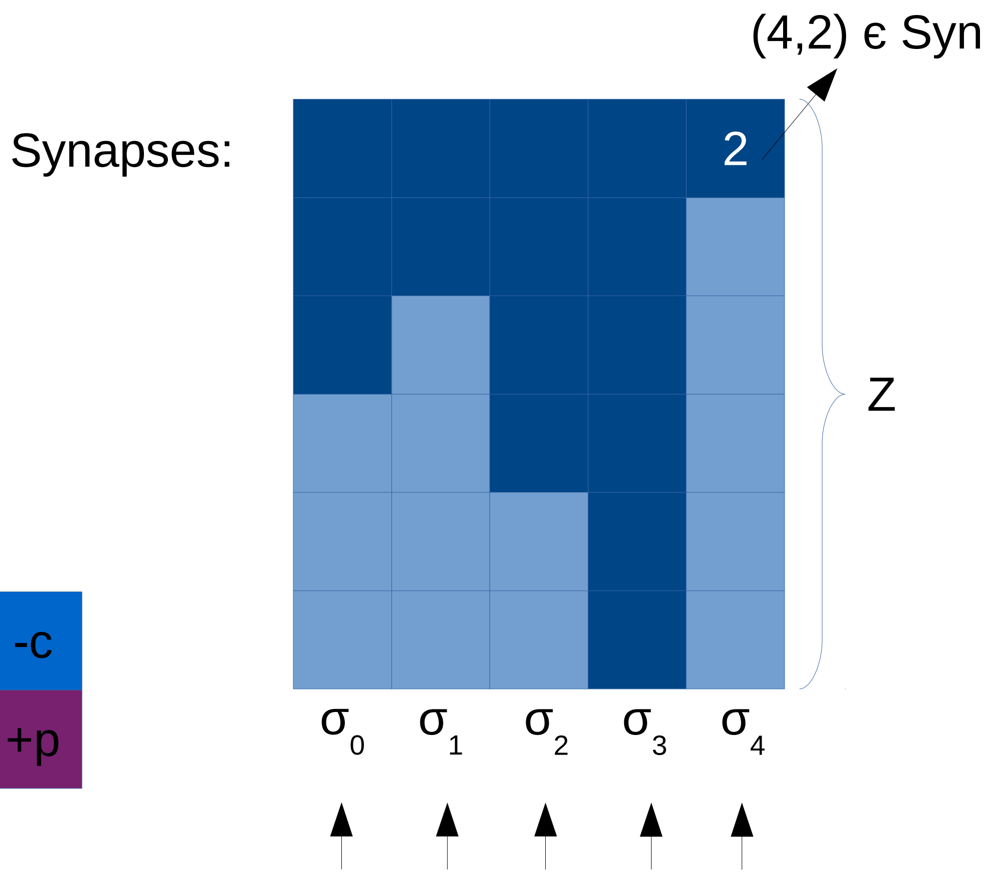
Spiking vector:



# Optimizations

- Access to spiking vector and transition matrix is not coalesced.
- Split transition matrix:



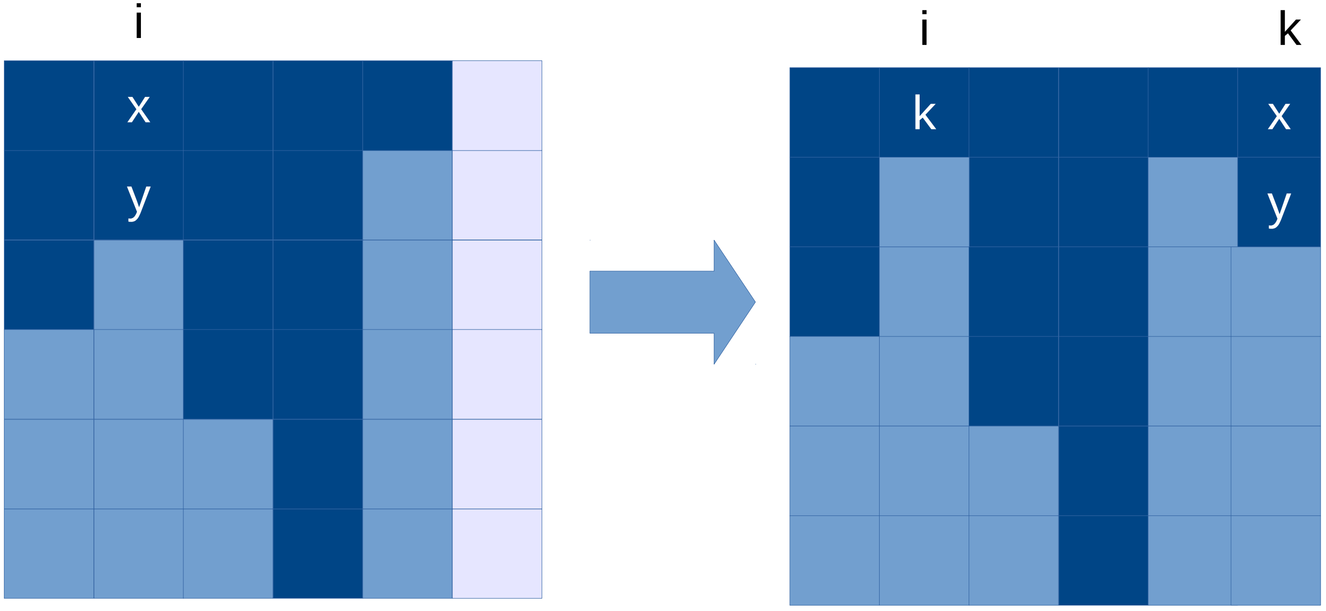



# SNP with budding

1) Copy column i to k

2) Delete column i and write k

$$[E]_i \rightarrow [\ ]_j \,/\, [\ ]_k$$

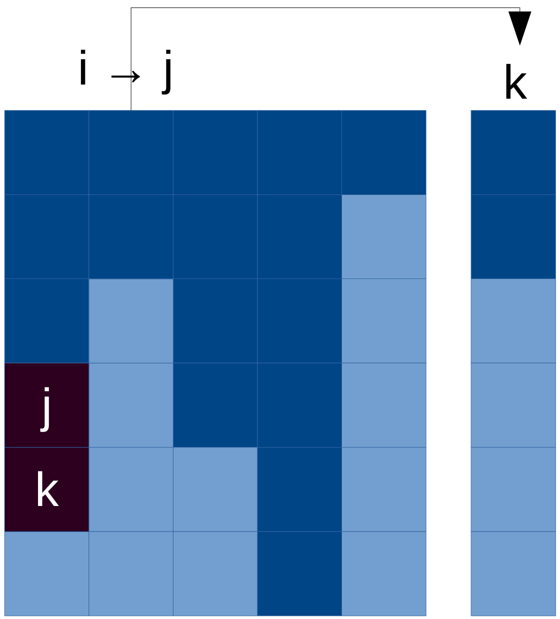



Optimization: swap indexes i and k, so the new column is for i and contains only k

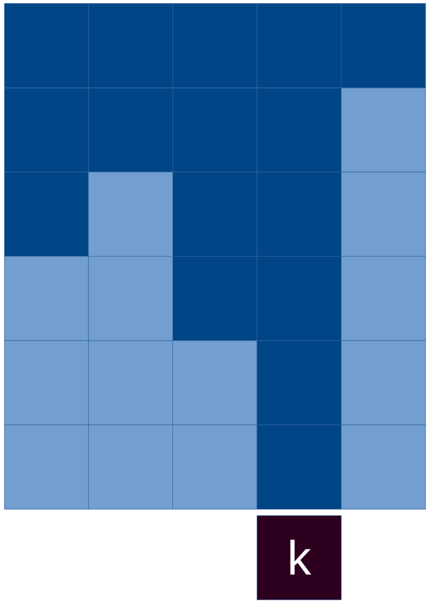
# SNP with division

1) Copy column i to k and i → j

2) Add j, k at the end of (t,i)




Problem: what if the column is “full” already?

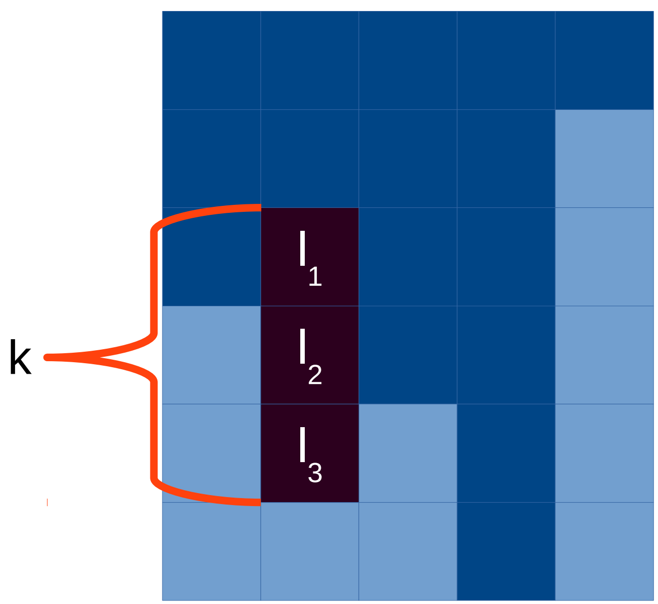



$$[E]_i \rightarrow [\ ]_j \,||\, [\ ]_k$$

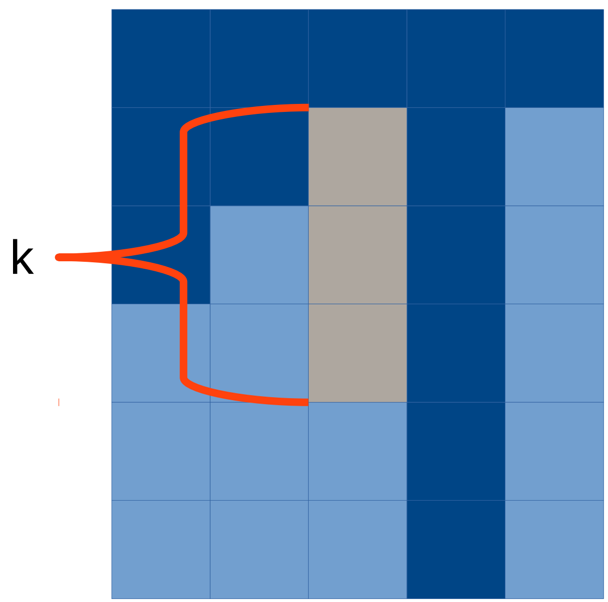
# SNP with plasticity

- Recall:

Plasticity rule: $E/a^c \to \alpha k(i, N)$, where $E$ is a regular expression over $O$, $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $k \geq 1$, and $N \subseteq \{1, \ldots, m\} - \{i\}$;




α=+




α=-

Problems: “holes” in columns, need to compact or to “refill”

# Conclusion

- Plasticity seems to be a better candidate for a dynamic-network SNP on the GPU.
  - It is better to have a fix number of neurons and change the synapses, rather than having to create new neurons and synapses.
- Budding can be made also in an efficient way, and columns are never exceeded.
- Next step: implement the ideas and test with GPUs and examples from literature.