Spiking Neural P system without delay simulator implementation using GPGPU

University of the Philippines Diliman:
Francis Cabarle
Henry Adorna
University of Seville:
Miguel Á. Martínez-del-Amor

Outline

- Spiking Neural P systems (SNP)
- Matrix representation
- GPU computing
- Simulation algorithm
- Future work

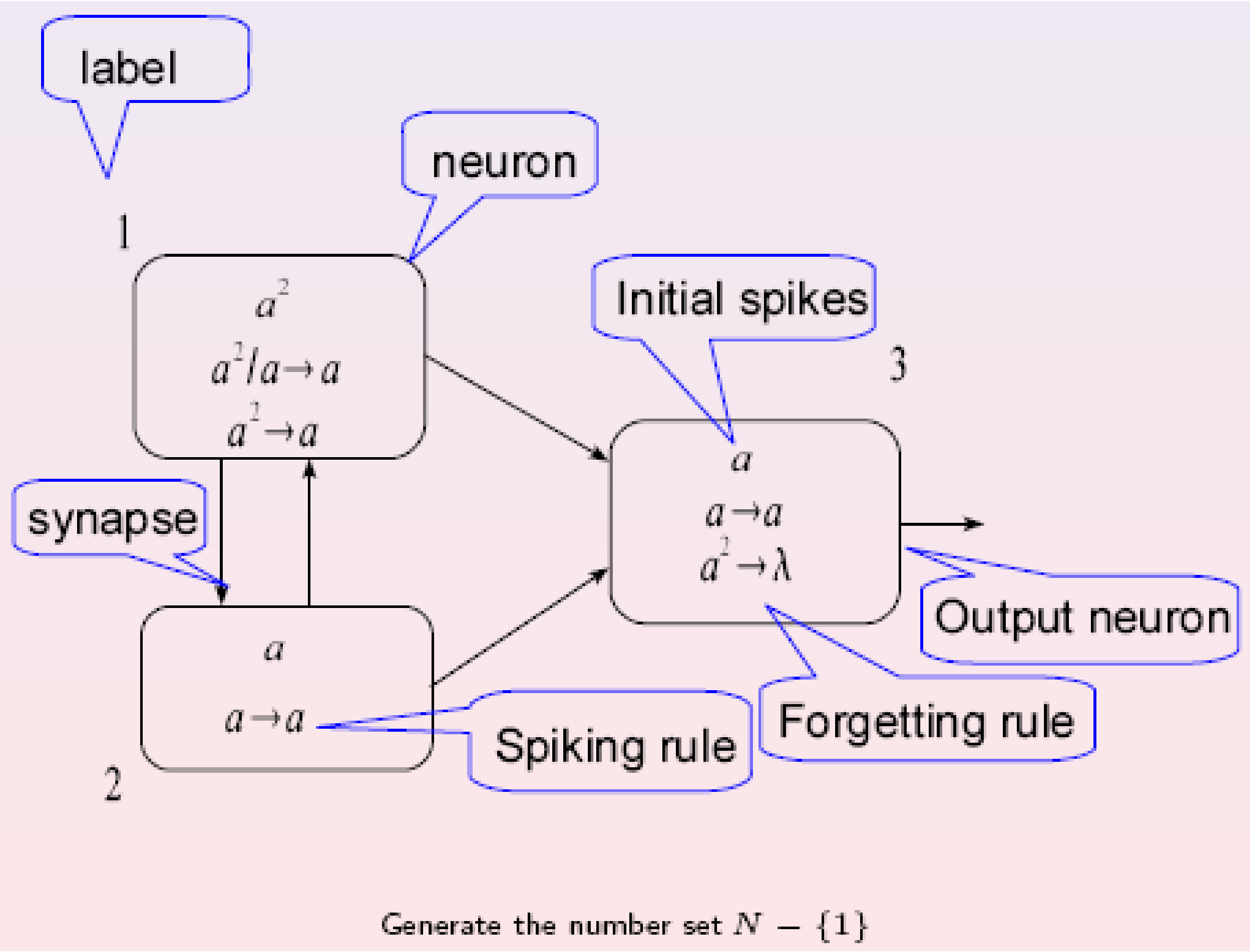
SNP system definition

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

1) $O = \{a\}$, the singleton alphabet of spike a;
2) $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, neurons, where:
    1) n_i : initial number of spikes
    2) R_i : a finite set of rules
        1) spiking rule: $E/a^c \rightarrow a^p$, E is a regular expression over a;
        2) forgetting rule: $a^s \rightarrow \lambda$, for $s \geq 1$;
3) syn, synapses between neurons;
4) in, out $\in \{1, 2, \ldots, m\}$, the input and the output neurons.

- A global clock is used in the synchronous systems.

Simple example of SNP



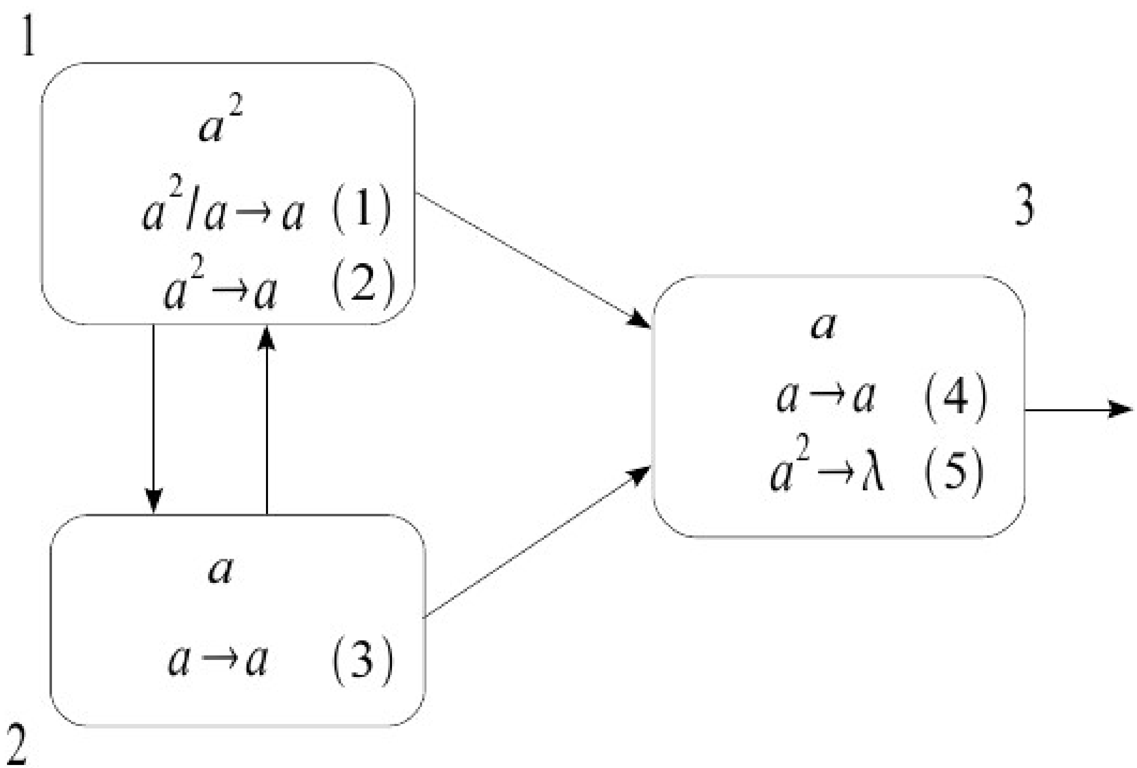
Xiangxiang Zeng

Matrix representation

- Previous work by Xiangxiang Zeng et al. presented in:
  - *X. Zeng, H. Adorna, M.A. Martínez-del-Amor, L. Pan. "When Matrices Meet Brains", 8th BWMC.*
  - *X. Zeng, H. Adorna, M.A. Martínez-del-Amor, L. Pan, M.J. Pérez-Jiménez. "Matrix Representation of Spiking Neural P Systems", CMC11.*

Matrix representation

- Configuration vector (C_k): $C_0 = (2,1,1)$
- Spiking vectors (S_k): (1,0,1,1,0),(0,1,1,1,0)
- Spiking transition matrix (M_Π):



$$M_\Pi = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix}$$

Matrix representation

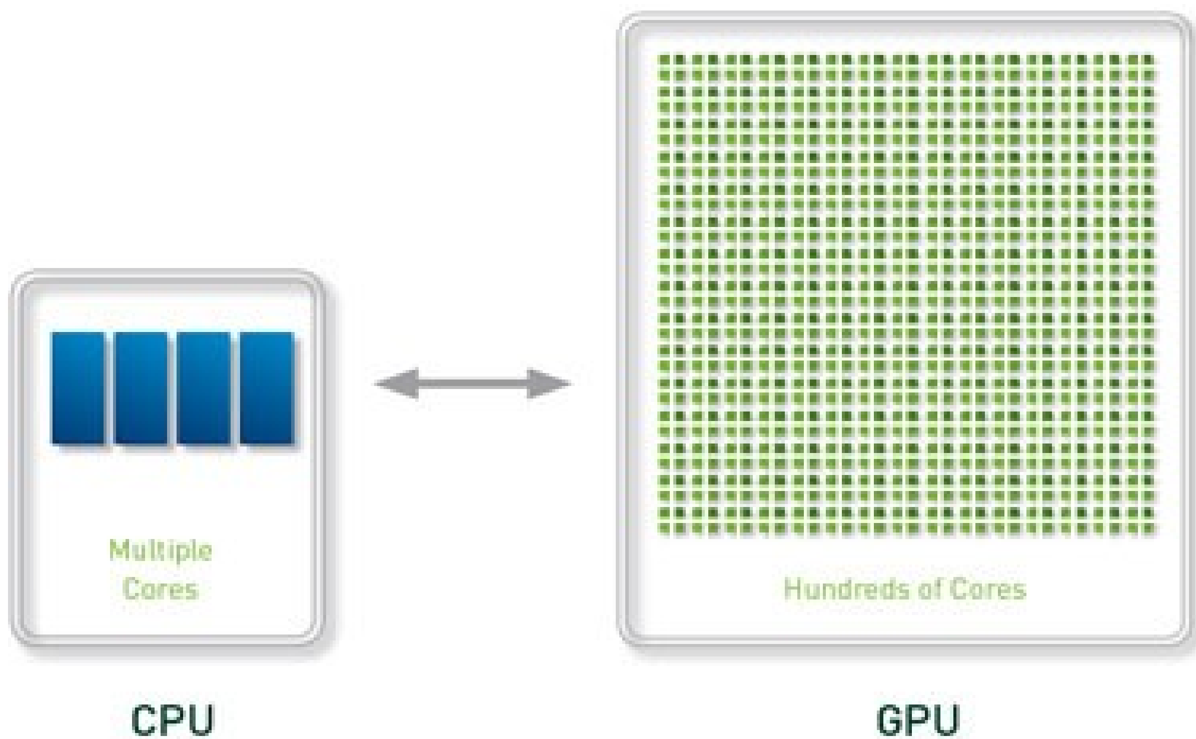
- Next configuration is calculated by:

$$C_{k+1} = C_k + S_k * M_{\Pi}$$

- Matrix operations are very optimized on the GPU.
- A GPU based simulator for SNP system.

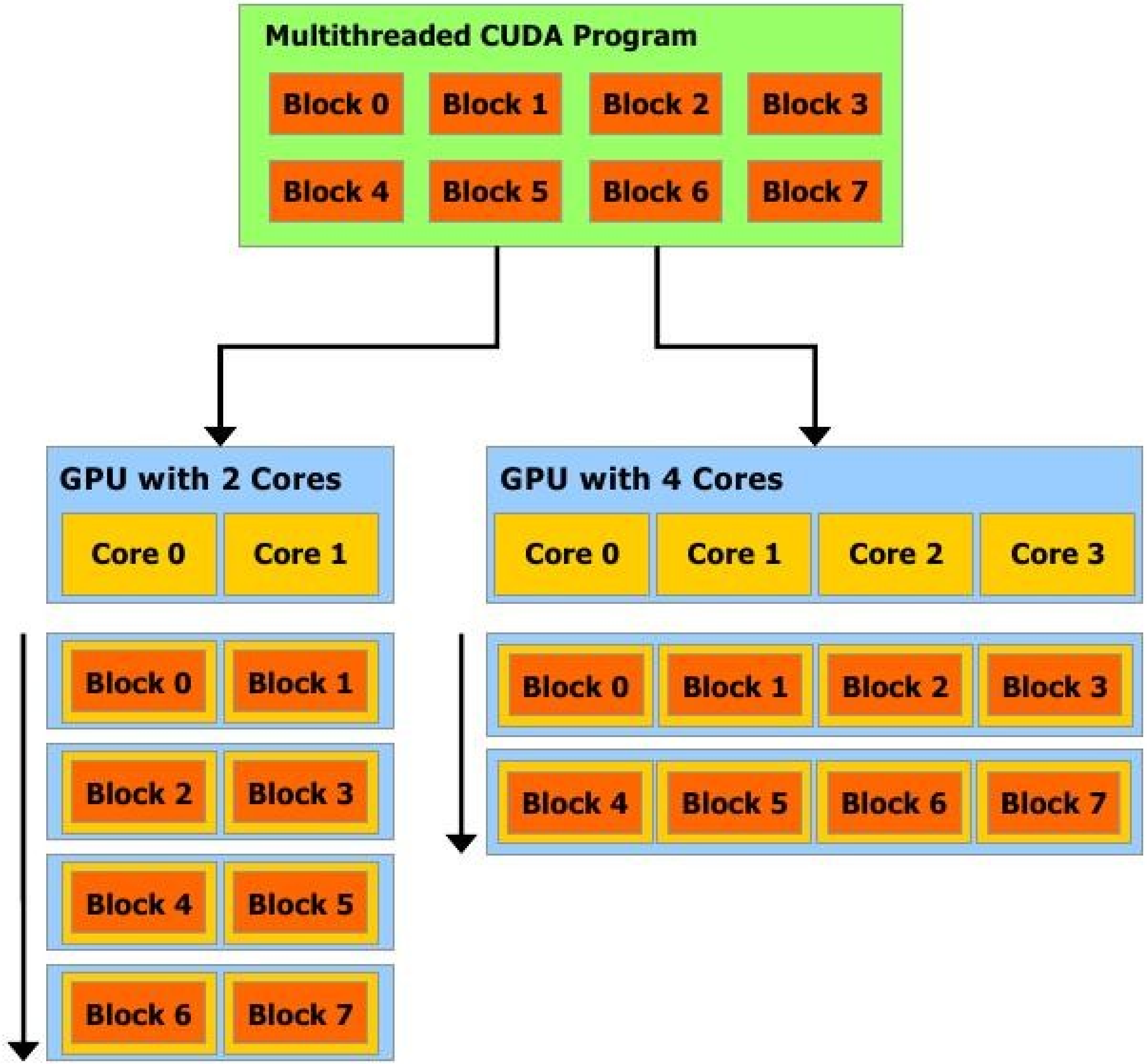
GPU computing

- GPGPU: techniques for using the GPU as a massively parallel co-processor.
- *Host:* the CPU vs *Device:* the GPU



GPU computing: CUDA

NVIDIA
CUDA



Simulation algorithm

- First version:
  - Rules of type $E/a^c \rightarrow a^p$ where $E=a^c$
  - Simulation of non-determinism
- Two stopping criteria:
  - Zero configuration vector
  - Repetition of configuration vectors
- Host side (python/C):
  - Read/write matrices and calculate spiking vectors
- Device side (CUDA):
  - Matrix addition and multiplication

Simulation algorithm

- Overview:
  - I. Load inputs (Host):
    - Configuration vector: *confVec*
    - Transition matrix: *M*
    - Rule criteria: *r*
  - II. Calculate all spiking vectors (Host):
    - All possible *spikVec* from all configurations *confVec*
  - III. Calculate next configurations (Device):
    - For each spiking vector, calculate the next configurations.
  - IV. Repeat I, II and III until stopping criteria satisfies.

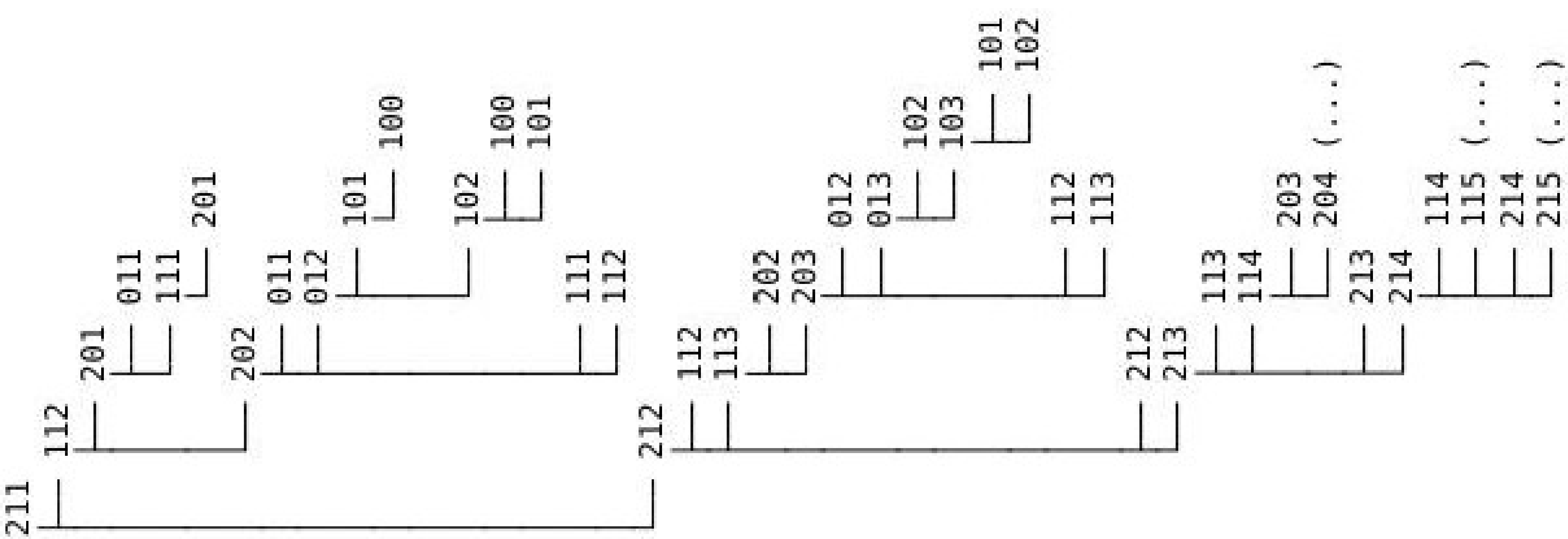
Output of the simulator

- The list of all possible configurations C_k

$allGenCk$ = ['2-1-1', '2-1-2', '1-1-2', '2-1-3', '1-1-3', '2-0-2', '2-0-1', '2-1-4', '1-1-4', '2-0-3', '1-1-1', '0-1-2', '0-1-1', '2-1-5', '1-1-5', '2-0-4', '0-1-3', '1-0-2', '1-0-1', '2-1-6', '1-1-6', '2-0-5', '0-1-4', '1-0-3', '1-0-0', '2-1-7', '1-1-7', '2-0-6', '0-1-5', '1-0-4', '2-1-8', '1-1-8', '2-0-7', '0-1-6', '1-0-5', '2-1-9', '1-1-9', '2-0-8', '0-1-7', '1-0-6', '2-1-10', '1-1-10', '2-0-9', '0-1-8', '1-0-7', '0-1-9', '1-0-8', '1-0-9']

Output of the simulator

- Tree of configurations:



Future work

- To automatically generate the graphical tree of configurations.
- More general regular expressions.
- Simulate SNP system with delay.
- Backwards computation.
- Improve the CUDA code performance:
  - Byte-compiling the python code.
  - Use optimized algorithms for matrix multiplication.

THANKS FOR YOUR ATTENTION

Spiking Neural P system without delay simulator implementation using GPGPU

University of the Philippines Diliman:
Francis Cabarle
Henry Adorna
University of Seville:
Miguel Á. Martínez-del-Amor

Outline

- Spiking Neural P systems (SNP)
- Matrix representation
- GPU computing
- Simulation algorithm
- Future work

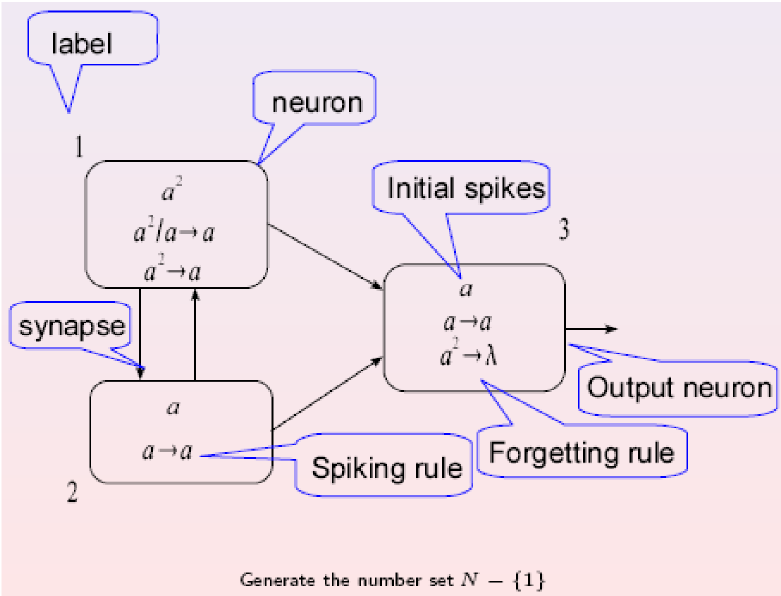
SNP system definition

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

1) $O = \{a\}$, the singleton alphabet of spike a;
2) $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, neurons, where:
   1) n_i : initial number of spikes
   2) R_i : a finite set of rules
      1) spiking rule: $E/a^c \rightarrow a^p$, E is a regular expression over a;
      2) forgetting rule: $a^s \rightarrow \lambda$, for $s \geq 1$;
3) syn, synapses between neurons;
4) in, out $\in \{1, 2, \ldots, m\}$, the input and the output neurons.

- A global clock is used in the synchronous systems.

Simple example of SNP



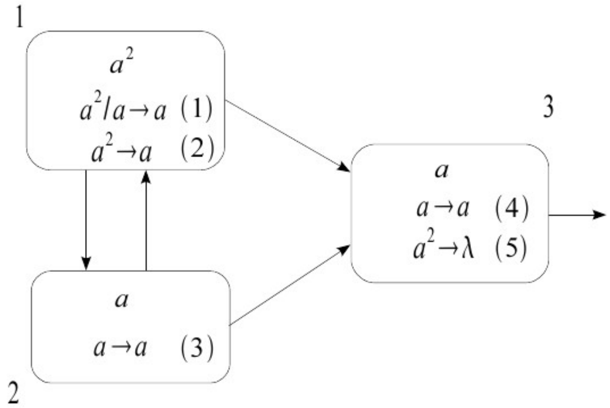
Xiangxiang Zeng

Matrix representation

- Previous work by Xiangxiang Zeng et al. presented in:
  - *X. Zeng, H. Adorna, M.A. Martínez-del-Amor, L. Pan. "When Matrices Meet Brains", 8th BWMC.*
  - *X. Zeng, H. Adorna, M.A. Martínez-del-Amor, L. Pan, M.J. Pérez-Jiménez. "Matrix Representation of Spiking Neural P Systems", CMC11.*

Matrix representation

- Configuration vector (C_k): C_0 = (2,1,1)
- Spiking vectors (S_k): (1,0,1,1,0),(0,1,1,1,0)
- Spiking transition matrix (M_Π):



$$M_\Pi = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix}$$

Matrix representation

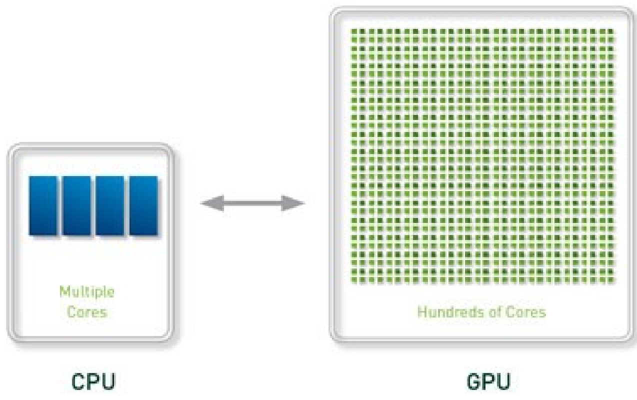
- Next configuration is calculated by:

$$C_{k+1} = C_k + S_k * M_{\Pi}$$

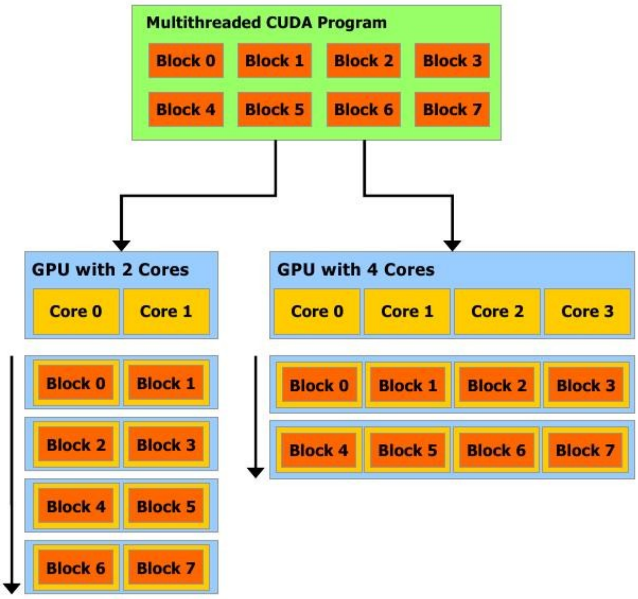
- Matrix operations are very optimized on the GPU.
- A GPU based simulator for SNP system.

GPU computing

- GPGPU: techniques for using the GPU as a massively parallel co-processor.
- *Host:* the CPU vs *Device:* the GPU



GPU computing: CUDA



Simulation algorithm

- First version:
  - Rules of type $E/a^c \rightarrow a^p$ where $E=a^c$
  - Simulation of non-determinism
- Two stopping criteria:
  - Zero configuration vector
  - Repetition of configuration vectors
- Host side (python/C):
  - Read/write matrices and calculate spiking vectors
- Device side (CUDA):
  - Matrix addition and multiplication

10

* Repetition of configuration vectors
=> to prevent infinite loops.

* Host side (python/C):
=> Python (OOP language) for string manipulations.
=> produce All possible *and* valid spiking vectors

* Device side (CUDA):
=> outsourcing of highly repetitive work

Simulation algorithm

- Overview:
  - I. Load inputs (Host):
    - Configuration vector: *confVec*
    - Transition matrix: *M*
    - Rule criteria: *r*
  - II. Calculate all spiking vectors (Host):
    - All possible *spikVec* from all configurations *confVec*
  - III. Calculate next configurations (Device):
    - For each spiking vector, calculate the next configurations.
  - IV. Repeat I, II and III until stopping criteria satisfies.

11

II. => produce All possible *and* valid spiking vectors.

$\Sigma = |r|$, total number of neurons

$\Psi = |\sigma V 1||\sigma V 2|...|\sigma V n|, n \in N, n \leq \infty$

Where $|\sigma V n|$ means the total count of the number of rules in the nth neuron which satisfy the regular expresion E. Ψ gives the expected number of valid and possible Sks which should be produced in a given configuration.

Output of the simulator

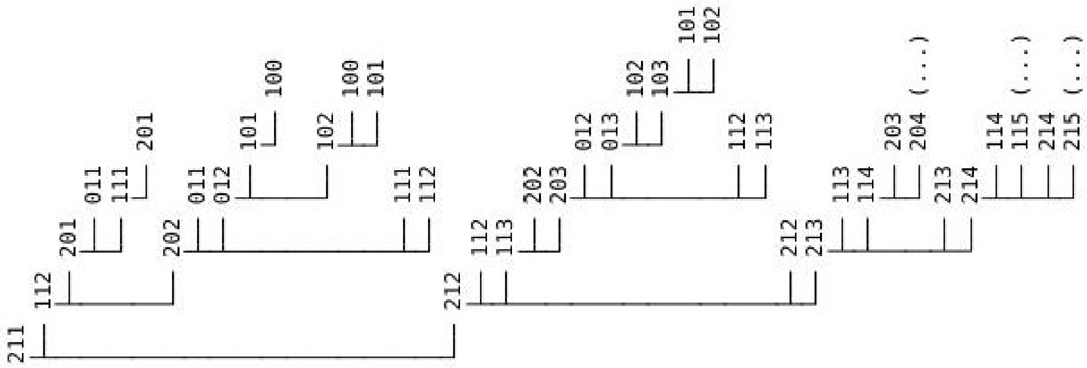
- The list of all possible configurations C_k

allGenCk = ['2-1-1', '2-1-2', '1-1-2', '2-1-3', '1-1-3', '2-0-2', '2-0-1', '2-1-4', '1-1-4', '2-0-3', '1-1-1', '0-1-2', '0-1-1', '2-1-5', '1-1-5', '2-0-4', '0-1-3', '1-0-2', '1-0-1', '2-1-6', '1-1-6', '2-0-5', '0-1-4', '1-0-3', '1-0-0', '2-1-7', '1-1-7', '2-0-6', '0-1-5', '1-0-4', '2-1-8', '1-1-8', '2-0-7', '0-1-6', '1-0-5', '2-1-9', '1-1-9', '2-0-8', '0-1-7', '1-0-6', '2-1-10', '1-1-10', '2-0-9', '0-1-8', '1-0-7', '0-1-9', '1-0-8', '1-0-9']

All possible Ck, else, infinite loops (as seen in the configuration tree next)

Output of the simulator

- Tree of configurations:



Future work

- To automatically generate the graphical tree of configurations.
- More general regular expressions.
- Simulate SNP system with delay.
- Backwards computation.
- Improve the CUDA code performance:
  - Byte-compiling the python code.
  - Use optimized algorithms for matrix multiplication.

* Use optimized algorithms for matrix multiplication => work on optimized algorithms for sparse matrix operations applied to GPGPUs

THANKS FOR YOUR ATTENTION