Some recent simulation efforts on the GPUs

University of the Philippines Diliman
H.N. Adorna, F.G.C. Cabarle, J.P. Carandang, A.R. Lagunda, G.I. Palaganas, J.M. Villaflores

University of Sevilla
M.A. Martinez-del Amor

Outline

- Spiking Neural P systems (SNP)
- Matrix representation (SNP with and without delay)
- SNP simulators and GPU computing
- Simulation algorithm
- NVIDIA CUDA and OpenCL simulators
- Some simulation results
- Next work

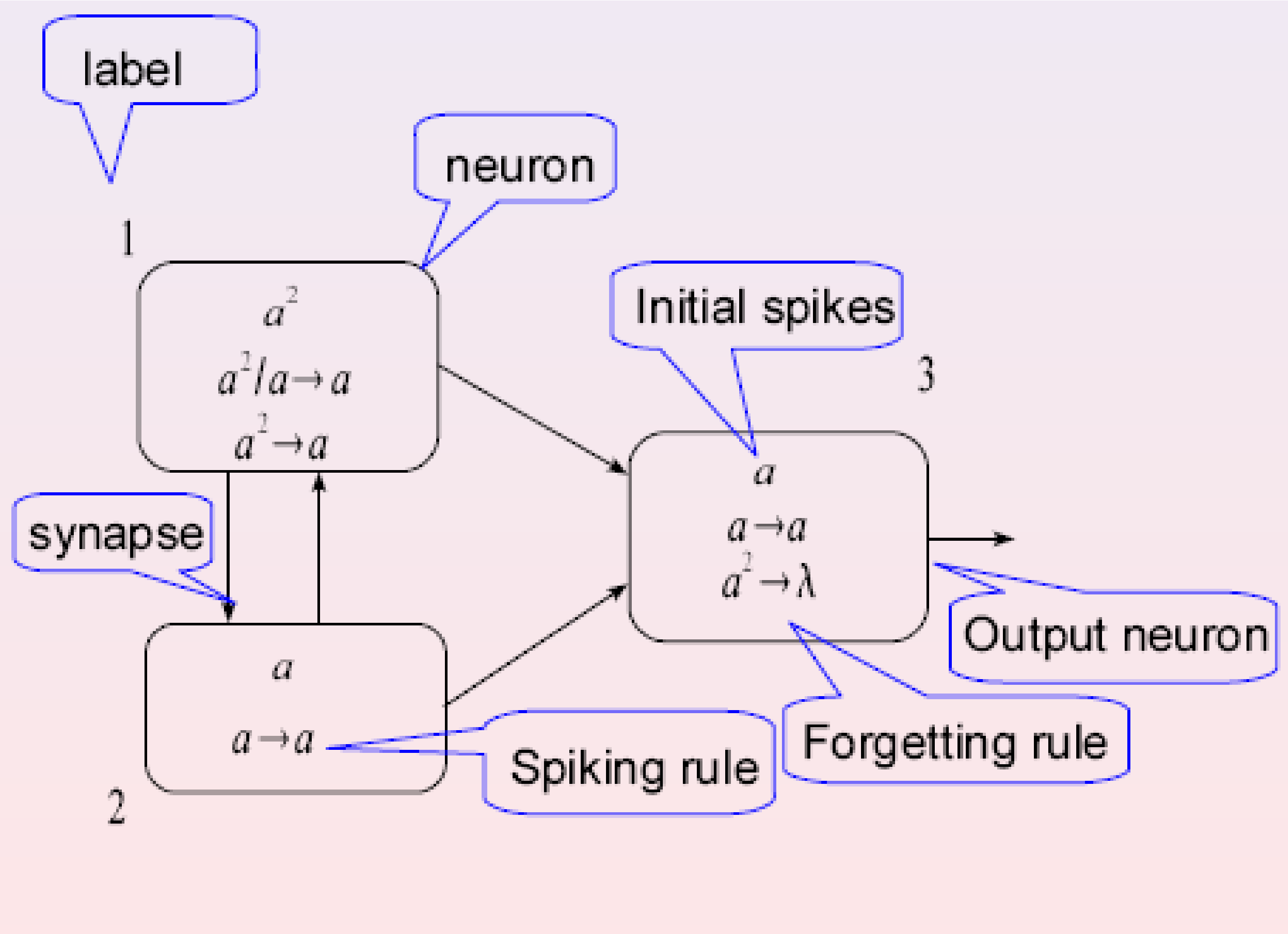
SNP system definition

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

1) $O = \{a\}$, the singleton alphabet of spike a;
2) $\sigma_{\mathrm{i}} = (n_i, R_i)$, $1 \leq i \leq m$, 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 O;
        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



Generate the number set $N - \{1\}$

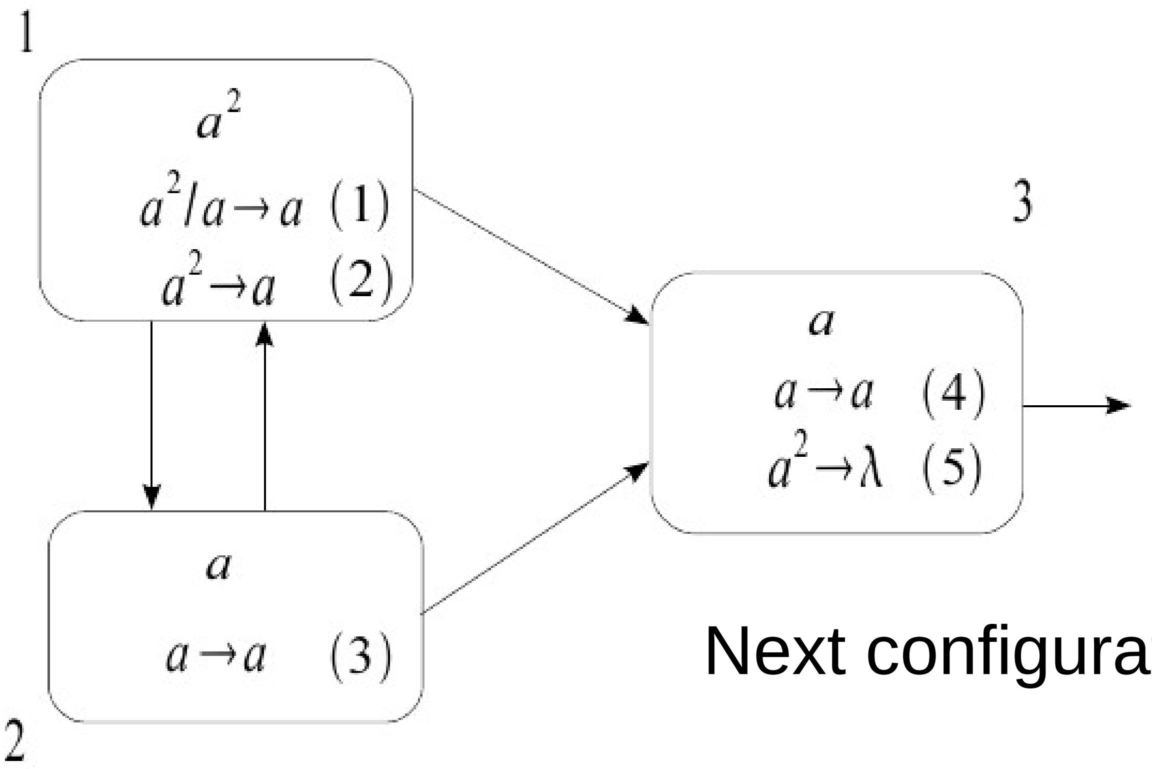
Xiangxiang Zeng

Some previous work

- *X. Zeng, H. Adorna, M.A. Martínez-del-Amor, L. Pan, M.J. Pérez-Jiménez. ``Matrix Representation of Spiking Neural P Systems'', CMC11 (*also in *8BWMC*)
- *F.G.C. Cabarle, H. Adorna, M.A. Martinez-del-Amor. ``Spiking Neural P system simulator based on CUDA'', CMC12* (also in *9BWMC*)
- Z. Bangalan, K. Soriano, R. Juayong, F.G.C. Cabarle, H. Adorna, M. Martínez-del-Amor. ``A GPU Simulation for Evolution-Communication P Systems with Energy Having no Antiport Rules'' *11BWMC*

Matrix representation of SNP systems (without delays)

- 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):



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

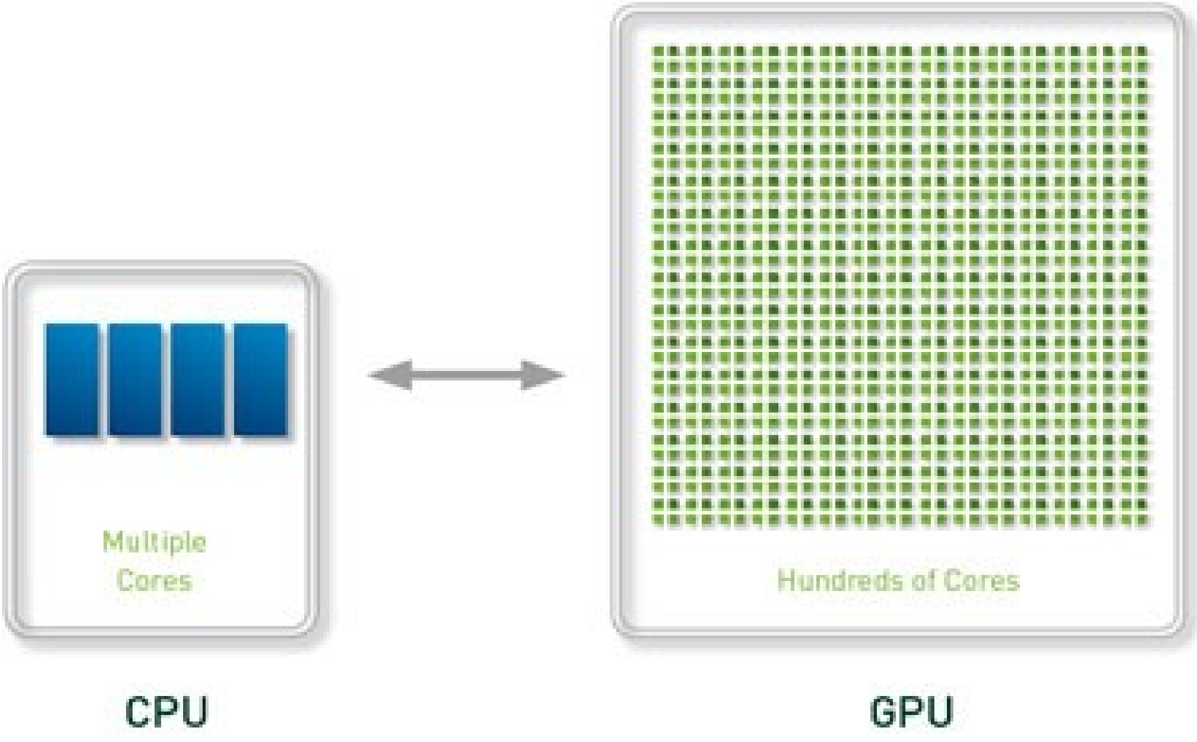
Next configuration: $C_{k+1} = C_k + S_k * M_\Pi$

Matrix representation of SNP with delays

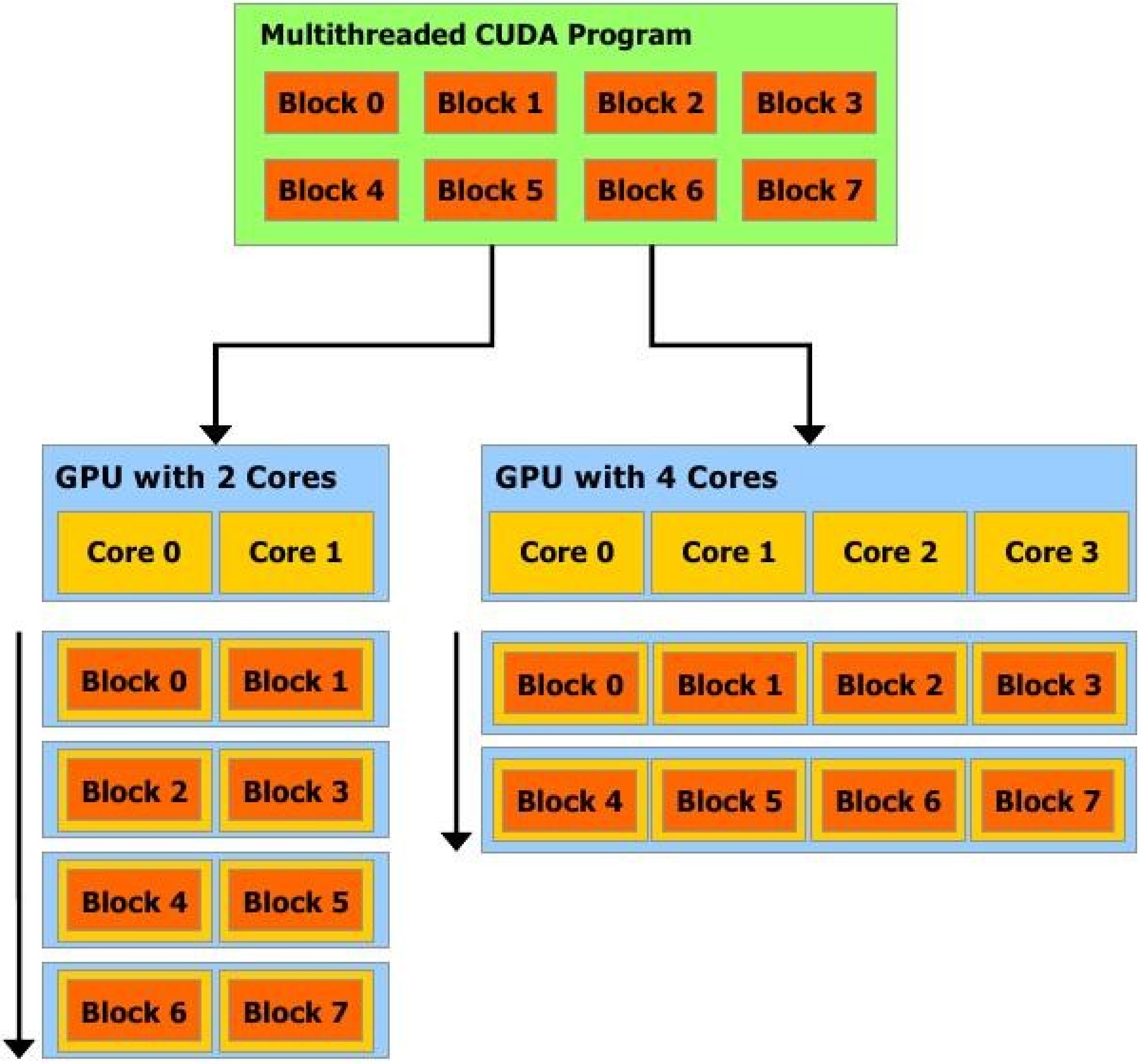
- Use additional vectors to keep track of delays, for lost (due to closed neurons) or gained spikes (open neurons), e.g.
- Status vector: an element is ``1'' if neuron *m* is open, ``0'' if *m* is closed.
- Delay vector: an element is the delay for each rule.
- Linear algebra operations are optimized for graphics processing units (GPUs)

GPU computing

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



Scalable parallel computing with GPUs



Multithreaded CUDA Program

Block 0 Block 1 Block 2 Block 3

Block 4 Block 5 Block 6 Block 7

GPU with 2 Cores

Core 0 Core 1

Block 0 Block 1

Block 2 Block 3

Block 4 Block 5

Block 6 Block 7

GPU with 4 Cores

Core 0 Core 1 Core 2 Core 3

Block 0 Block 1 Block 2 Block 3

Block 4 Block 5 Block 6 Block 7

Simulation algorithm

- Improvements from earlier SNP simulator on GPU:
  - Rules of type $E/a^c \rightarrow a^p$ where $E=a^*$ or $E=a^c$
  - C for entire GPU simulator (host & device)
- Stopping criteria:
  - Zero configuration vector, or repeated configuration vectors
- Host side (C):
  - Allocate space, copy input to (output from) GPU.
- Device side (CUDA C):
  - Deterministic computations of SNP with delay

Simulation algorithm

- Overview:

  I. Load inputs (Host):
  - Vectors, e.g. configuration vector, delay vector
  - Transition matrix: *M*

  II. Calculate all spiking vectors (Device):
  - All possible *spikVec* from all configurations *confVec*

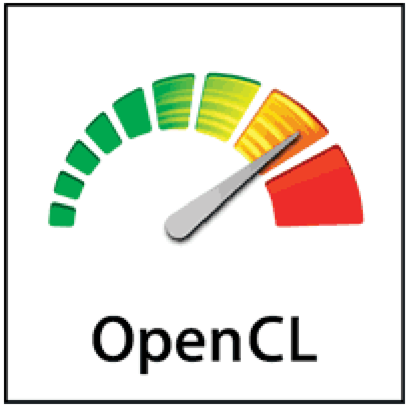
  III. Calculate next configurations (Device):
  - For each spiking vector, calculate the next configurations.

  IV. Repeat II and III until stopping criteria satisfies.

  V. If one stopping criteria is met, copy result to Host.

Open Computing Language (OpenCL)

- Open source framework to execute code on *heterogeneous computing platforms,* e.g. mix of CPUs, GPUs, FPGAs.
- Supports Intel procs, GPUs of NVIDIA and AMD/ATI.
- C-based
- Write code once, execute on several platforms



SNP simulator on OpenCL

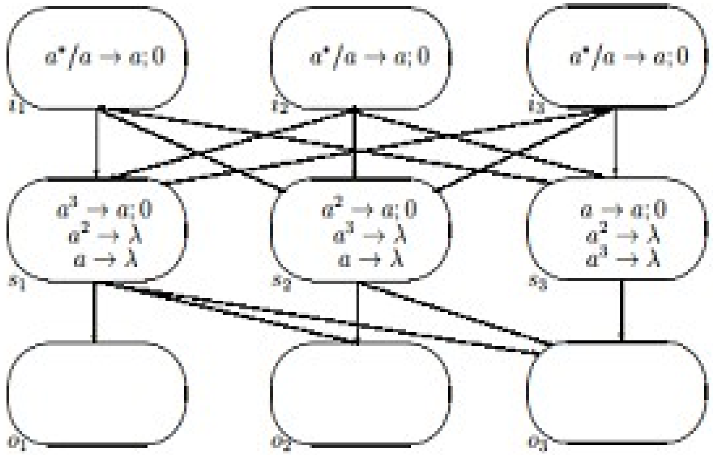
- Can execute as parallel code on Intel CPUs, GPUs of NVIDIA and AMD/ATI, etc.
  - Not like CUDA executing only on NVIDIA GPUs
- So far as we know, first attempt to simulate SN P (any P system?) on OpenCL
- Uses the same algorithm shown earlier

Binary file format for SNP

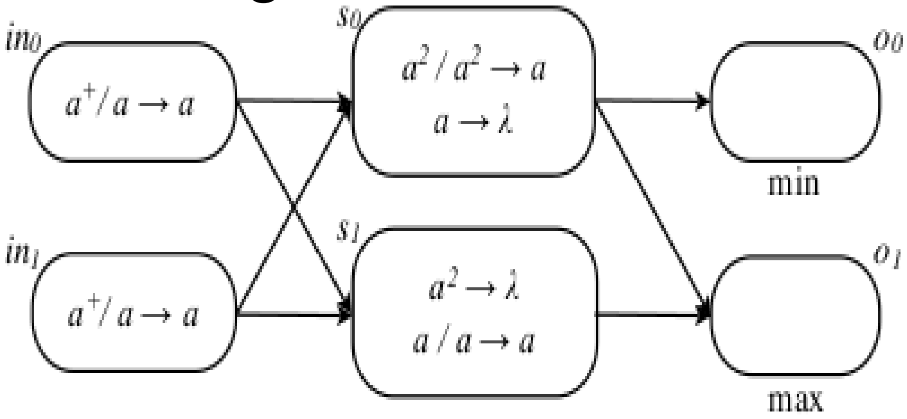
- Based on
  - M.A. Martinez-del-Amor, L.F. Macias-Ramos, M.J. Perez-Jimenez. ``Parallel simulation of PDP systems: updates and roadmap'' *13BWMC*
- From .pli file of PLingua to binary file for CUDA and OpenCL simulations
- SNP systems for simulations: sorting network implemented in SNP
  - R. Ceterchi, A. I. Tomescu. ``Implementing sorting networks with spiking neural P systems'' Fundam. Inf. (2008)
  - M. Ionescu, D. Sburlan. ``Some applications of spiking neural P systems'' Computing & Informatics (2008)

SNP for simulations

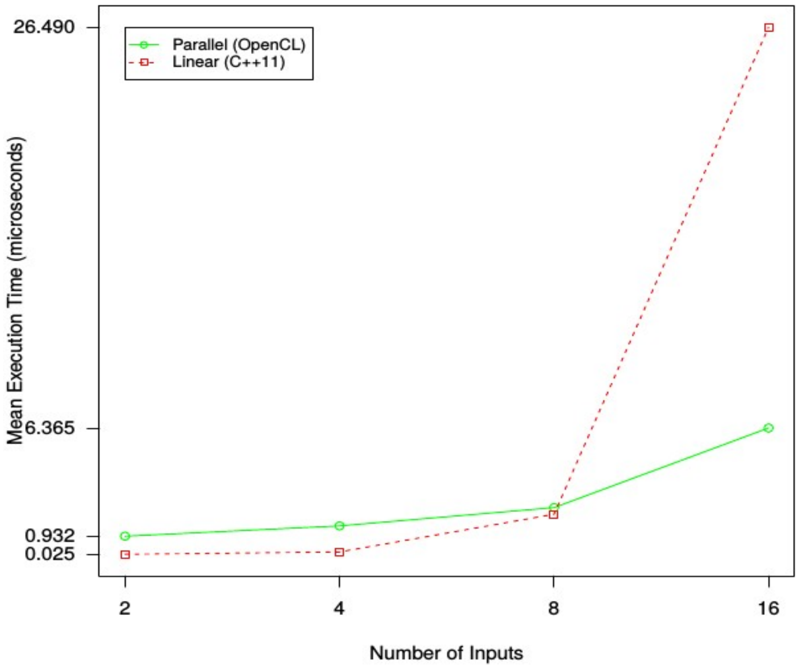
- Generalized sorting network on SNP

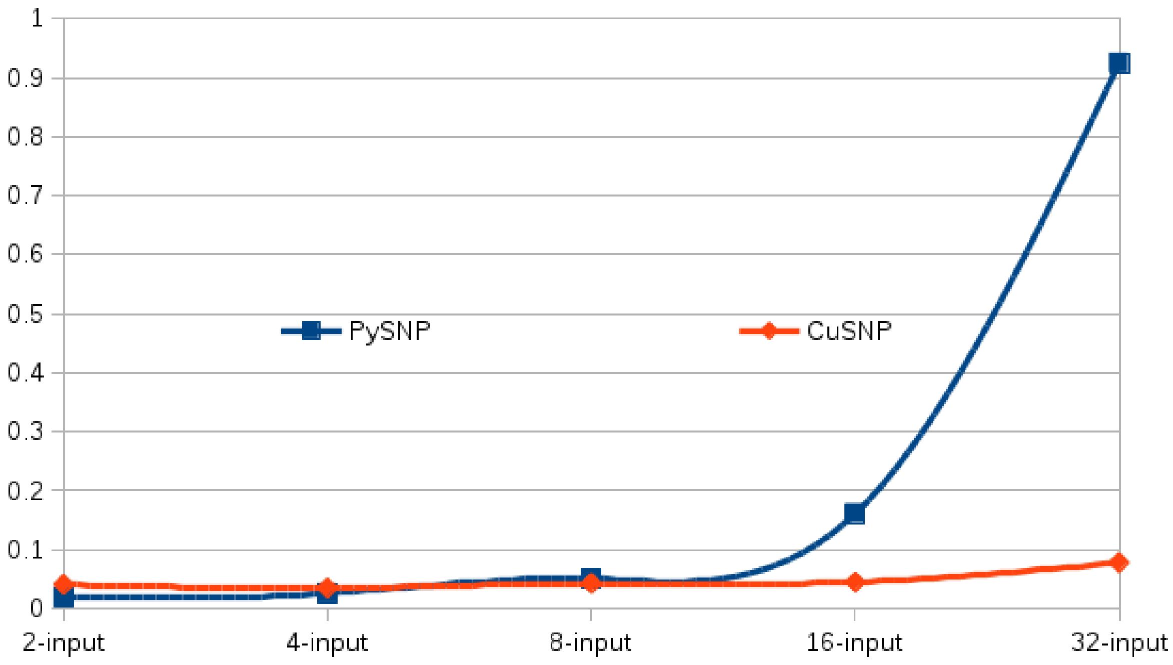


- Bitonic sorting network on SNP



Some simulation results





Next work

- Optimize code for other GPU memory types
- More general regular expressions (e.g. implementing NFA in GPU)
- Simulate on and compare to other platforms e.g. Beowulf clusters, OpenMPI, Intel Many Integrated Core architecture (MIC), multi-GPUs.
- Simulate other SNP systems solving hard problems

Fin.

¡Gracias por su atención!