A Spiking Neural P system simulator based on CUDA

Francis George Cabarle[1], **Henry Adorna**[1], Miguel Martínez-del-Amor[2]

[1]Algorithms & Complexity Lab, Dept. of Computer Science, University of the Philippines Diliman
fccabarle@up.edu.ph, ha@dcs.upd.edu.ph
[2]Research Group on Natural Computing, University of Seville, Spain
mdelamor@us.es

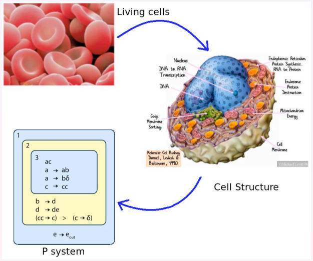
25.August.2011

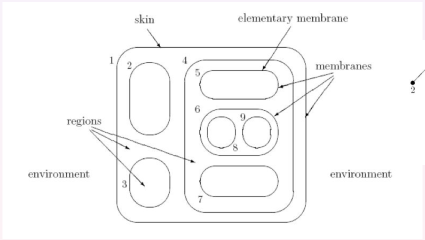
What did we do?

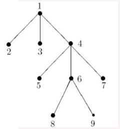
Our Share

1. *Implementation of an SN P Systems without delay in GPU using (Py)CUDA capitalizing on the fact that SN P systems without delay is represented by a matrix. Also, that its (SN P Systems') operation is implementable by some matrix operation.*
2. *As expected, the implementation base on CUDA performs better than that implemented in CPU.*

Membrane Computing



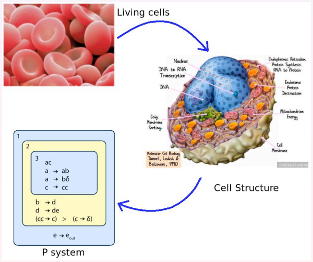


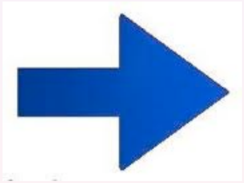


Symbolic presentation

$$[_1 [_2]_2 [_3]_3 [_4 [_5]_5 [_6 [_8]_8 [_9]_9]_6 [_7]_7]_4]_1$$

Implementation . . . anyone?



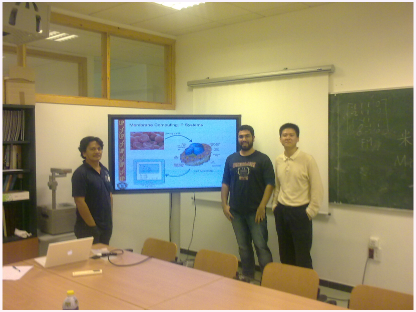


(ref: digstuffs.com)

Early efforts . . .(via P Page) since 2002 . . .

1. January 2002: Transition P Systems Simulator.: by Angel Baranda, 'Natural Computing Group of Madrid (Spain).
2. May 2002: A Membrane Systems Simulator.: by Gabriel Ciobanu and Dorin Paraschiv
3. September 2003: SubLP-Studio v0.1: by Alexandros Georgiou, University of Sheffield, UK
4. October 2004: SimCM Author: M. Isabel Nepomuceno Chamorro, Natural Computing Group, University of Sevilla, Spain
5. November 2004: PSim: by Group for Models of Natural Computing (MNC), University of Verona, Italy
6. September 2005: Simulation Software for Membrane Approximation Algorithm: by T. Nishida, Toyama Prefectural University, Japan
7. March 2006: Two simulators - Vibrio Fischeri and Dynamical Probabilistic P systems: by P. Cazzaniga, D. Pescini, Universita' di Milano-Bicocca, Milan, Italy
8. July 2006: Cyto-Sim: Biological compartment simulator: Microsoft Research - University of Trento Centre for Computational and Systems Biology, Trento, Italy.
9. August 2006: simulators for conformon P systems: Pierluigi Frisco
10. November 2006: Simulators for biological processes available at the University of Sheffield, UK
11. April 2007: Spiking Neural P Systems Simulator, by M.A. Gutierrez Naranjo and D. Ramirez Martinez, University of Sevilla, Spain
12. March 2009. MetaPlab: a virtual laboratory for modeling biological systems by MP systems. University of Verona, Italy.

Primary motivation of this work . . .



Sevilla, (2009)
Xiangxiang Zeng (SN P System), Miguel Martinez-del-Amor (GPGPU with CUDA), myself

Matrix Representation of Spiking Neural P Systems.
Int. Conf. on Membrane Computing 2010

Xiangxiang Zeng, Henry Adorna, Miguel A. Martnez-del-Amor, Linqiang Pan, Mario J. Perez-Jimenez

Question

1. *How do we implement the parallelism in P Systems, in particular SN P Systems?*
2. *Is there a hardware capable enough to implement parallelism of P Systems?*

Since 2004

G. Ciobanu, G. Wenyuan:

1. P Systems Running on a Cluster of Computers. Lecture Notes in Computer Science, 2933, 123-139, 2004.

Van Nguyen, David Kearney, Gianpaolo Gioiosa:

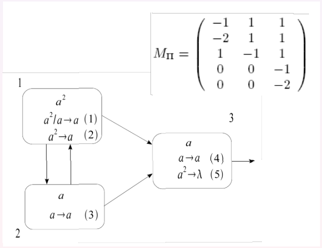
1. Balancing Performance, Flexibility, and Scalability in a Parallel Computing Platform for Membrane Computing Applications. Workshop on Membrane Computing 2007: 385-413
2. An Implementation of Membrane Computing Using Reconfigurable Hardware. Computing and Informatics 27(3+): 551-569 (2008)
3. An Algorithm for Non-deterministic Object Distribution in P Systems and Its Implementation in Hardware. Workshop on Membrane Computing 2008: 325-354
4. A Region-Oriented Hardware Implementation for Membrane Computing Applications. Workshop on Membrane Computing 2009: 385-409

Jose M. Cecilia, Jose M. Garca, Gines D. Guerrero, Miguel A. Martinez-del-Amor, Ignacio Perez-Hurtado, Mario J. Perez-Jimenez:

1. Simulating a P system based efficient solution to SAT by using GPUs. J. Log. Algebr. Program. 79(6): 317-325 (2010)
2. Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics 11(3): 313-322 (2010)

SN P Systems without delay via GPU

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):



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

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.

F. Cabarle, H. Adorna, M.A. Martnez-del-Amor. *Simulating Spiking Neural P System without Delay using GPU.* , 9th BWMC.

SN P System

Spiking Neural P (SNP) systems: directed graph inspired by **neurons** connected by axons w/ synapses [Ionescu et al. 2006]

(ref:heatonresearch.com)

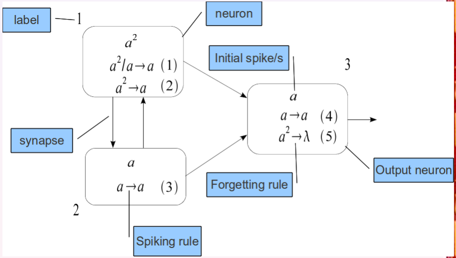


Figure: A sample SNP system Π_1 w/ labeled parts.

SN P Systems

An SNP system is a construct of the form:

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

1. $O = \{a\}$, alphabet of only one object, the system spike a.
2. $\sigma_1, \ldots, \sigma_m$ are m neurons of the form: $\sigma_i = (n_i, R_i), 1 \leq i \leq m$,:
    a) $n_i \geq 0$, initial spike a in neuron σ_i
    b) R_i, finite set of rules of w/ 2 forms: (b-1) **Spiking rules**, $E/a^c \rightarrow a$, E is a reg. exp. over a, $c \geq 1$. (b-2) **Forgetting rules**, $a^s \rightarrow \lambda$, $s \geq 1$, such that for each rule $E/a^c \rightarrow a$ of type (b-1) from R_i, $a^s \notin L(E)$.
3. $syn = \{(i,j) \,|\, 1 \leq i, j \leq m,\ i \neq j\}$ are synapses between neurons.
4. $in, out \in \{1, 2, \ldots, m\}$, input & output neurons.

From Seville and India Group

1. (April 2007) Spiking Neural P Systems Simulator, by M.A. Gutierrez Naranjo and D. Ramirez Martinez, University of Sevilla, Spain.
2. (2011, CMC 12) Simulation of Spiking Neural P Systems Using Pnet Lab, by V.P. Metta, K. Krithivasan,D. Garg, India
3. (2011, CMC 12) P Lingua based Simulator for Spiking Neural P Systems, L.F. Macias-Ramos, et al., University of Sevilla, Spain.

SN P Matrix Representation

Spiking transition matrix M_{SNP} is a matrix comprised of a_{ij} elements where a_{ij} is given as

Definition

$$a_{ij} = \begin{cases} -c, & \text{rule } r_i \text{ is in } \sigma_j \text{ and is applied consuming } c \text{ spikes;} \\ p, & \text{rule } r_i \text{ is in } \sigma_s \ (s \neq j \text{ and } (s,j) \in syn) \\ & \text{and is applied producing } p \text{ spikes in total;} \\ 0, & \text{rule } r_i \text{ is in } \sigma_s \ (s \neq j \text{ and } (s,j) \notin syn). \end{cases}$$

An SN P Matrix

- *Configuration vector C_k*:
  $C_0 =< 2, 1, 1 >$.
- *Spiking vector S_k*:
  $S_0 =< 1, 0, 1, 1, 0 >$,
  $S_0' =< 0, 1, 1, 1, 0 >$.
- Spiking transition matrix:

$$M_{\Pi_1} = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix}$$

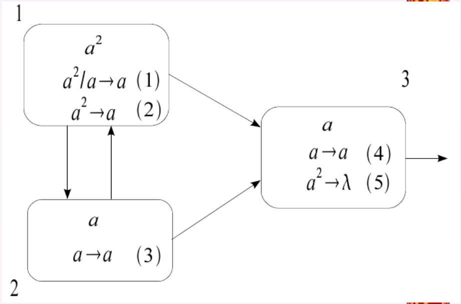


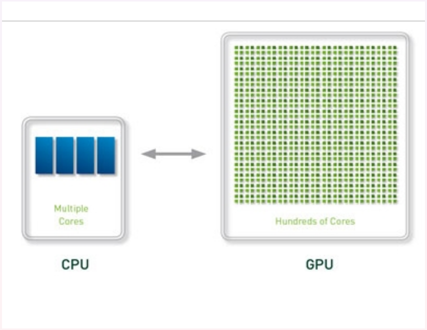
Figure: SNP system Π_1 from [Ionescu et al. 2006].

Next Configuration representation

- Next configuration: $C_{k+1} = C_k \ + \ S_k \cdot M_\Pi$.
- $S''_0 =< 1, 1, 1, 1, 0 >$ is an *invalid* S_k (only one rule per neuron is used).

Our Suggestion:GPU Computing

GPGPU: techniques for using the GPU as a massively parallel co-processor.



Host: the CPU vs Device: the GPU

CPU vs. GPU

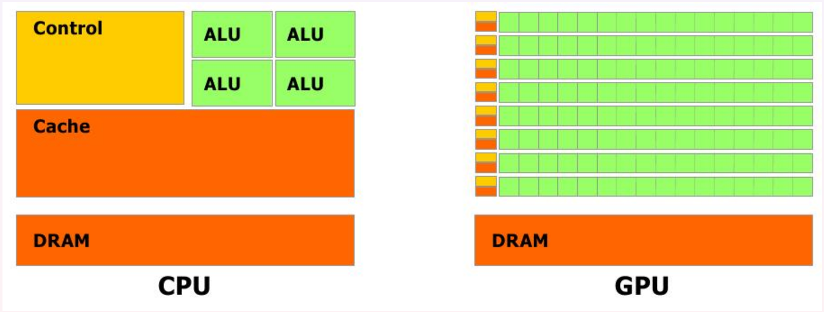


Figure: General CPU vs. general GPU architecture [NVIDIA corp. 2011].

Why GPU?

1. GPUs are the leading exemplars of modern high throughput-oriented architectures [Garland et al, 2010].
2. GPUs have been successfully used to speedup many parallel applications.
3. Modern GPUs are not limited only to graphics processing, as done by the first graphic cards, as they can now be used for general purpose computations [Harris, 2005]; they are now multi-core and data-parallel processors [Kirk and Hwu, 2010].

Why GPU?

1. GPGPU (General Purpose computation on the GPU), a programmer can achieve with a single GPU, a throughput similar to that of a CPU based cluster [NVIDIA, 2010; Harris, 2010]
2. the main *advantages* of using GPUs are their *low-cost*, *low-maintenance* and *low power consumption* relative to conventional parallel clusters and setups, while providing comparable or improved computational power.
3. Moreover, parallel computing concepts such as hardware abstraction, scaling, and so on are handled efficiently by current GPUs.

Parallel Computing w/ GPUs

- Compute Unified Device Architecture (CUDA) by NVIDIA in 2006
- Arch + programming model, extends ANSI C

Figure: Graphics card w/ NVIDIA CUDA enabled GPU.

CUDA Processing Flow

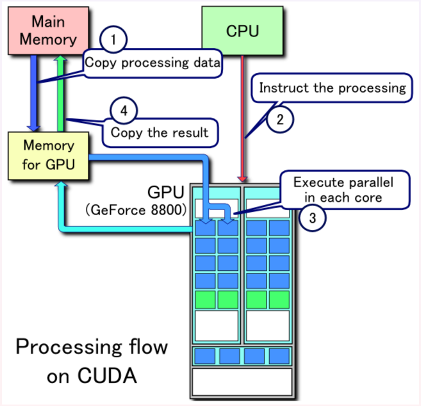


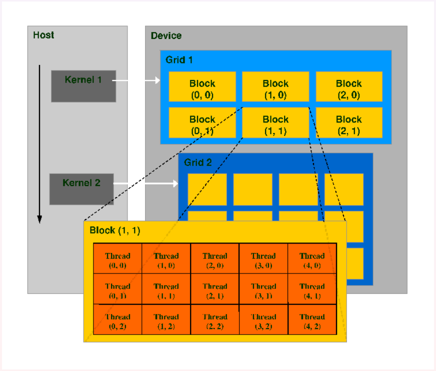
Figure: CUDA Processing Flow, [http://en.wikipedia.org].

Remarks on the Computation

1. The code to be executed in a GPU is written in CUDA C (CUDA *extended* ANSI C programming language).
2. The parallel distribution of the execution units (threads) in CUDA can be split up into multiple threads within multiple thread blocks, each contained within a grid of (thread) blocks. These grids belong to a single device/single GPU.
3. Each device has multiple cores, each capable of running its own *block of threads*.

CUDA model

CUDA uses single program, multiple data (SPMD) parallel paradigm [Kirk, Hwu 2010].



Remarks on the Computation

1. A function known as a *kernel function* is one that is called from the host but executed in the device.
2. GPUs with the same architecture as the one used in this work has a maximum number of threads per block equal to 512.
3. The maximum size of each dimension of a thread block is (512 x 512 x 64), pertaining to the *x*,*y*, and *z* dimensions of a block respectively.
4. Lastly, the maximum size of each dimension of a grid of thread block is (65535 x 65535 x 1) for the grid's *x*,*y*, and *z* dimensions.

Remark: Matrix & parallel hardware

- Matrix operations are very optimized on parallel hardware, including GPUs [Fatahilian et al. 2004].
- GPU simulations of SNP systems using their matrix representations seem more *natural*.

GPU Simulation Consideration

1. **Input files:** file versions of C_k, S_k, M_{SNP}, and a file r (with the list of rules R_i.)
2. **String manipulation:** An OOPL such as Python is suited. *PyCUDA* was chosen in order to fully utilize the speedup of CUDA as well as minimize development time, and is a Python programming interface to CUDA.[1]
3. **Computations involving linear algebra:** C programming language (which NVIDIA extended for their purposes as CUDA C) is suited.

[1] PyCUDA was developed by mathematician Andreas Klöckner for for a more efficient parallel computing on CUDA using Python: safer in terms of memory handling, object cleanup (among others), and faster (in terms of development time via abstractions etc).
In actuality, only the kernel functions are written in C, and those functions are *embedded* within the Python code

Simulation notes

- SNP systems w/o delays, M_Π in row-major order.
- Non-determinism: produce all possible and valid S_k's from given C_k's.
- Use *PyCUDA* for handling of characters + reg exp, *C* for integral computations.
- PyCUDA is Python wrapper for CUDA, used in HPC [Klöckner 2009].
- C_k, S_k, M_Π as mutable PyCUDA *lists*, not strings.
- 2 stopping criteria:
  - Zero C_k
  - Repetition of C_k's

Simulation notes

1. Host/CPU side (python/C)$\rightarrow$ Read inputs (C_0/C_k's, S_k's, M_Π), write and calculate next S_k's, C_{k+1}'s. (***decision making***)
2. Device/GPU side (CUDA) $\rightarrow$ Matrix addition + multiplication (***outsourced parallel work***) i.e. perform ($C_{k+1} = C_k + S_k \cdot M_\Pi$) in parallel.

Simulation Algorithm

Overview:
Inputs: C_0 (C_k's afterwards), M_Π, r (rule file)
Outputs: All valid+possible S_k's, C_k's.

I Load inputs **(Host)**
II Compute all possible+valid S_k's using C_k's **(Host).**
III Per S_k, compute next C_k **(Device).**
IV Repeat I, II and III until at least 1 stopping criteria is satisfied **(Host/Device).**

Simulation Flow Diagram

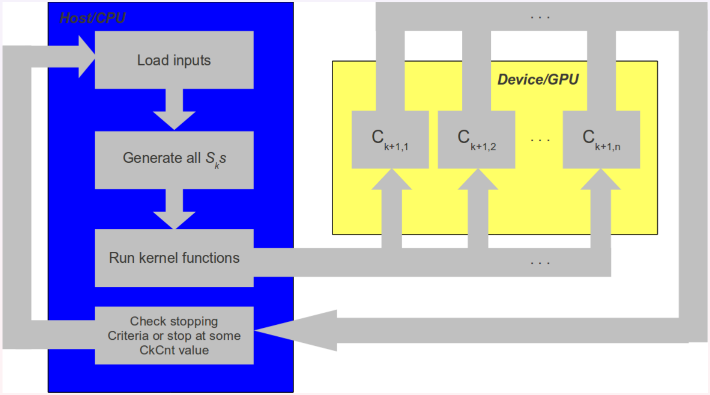
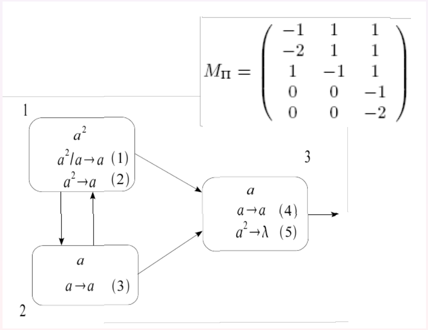


Figure: Diagram showing the simulation flow, with the host and device

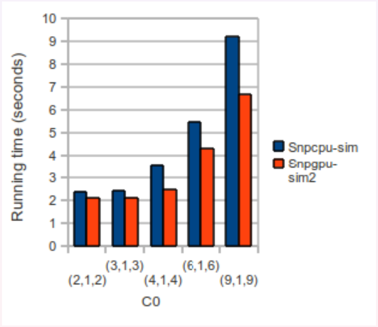
Machine model specification

1. Setup of *snpgpu-sim3* simulated Π_1 and Π_2 using an Apple iMac running Mac OS X 10.5.8, with an Intel Core2Duo CPU at 2.66GHz and with a 6MB L2 cache.
2. The GPU of the iMac is an NVIDIA GeForce 9400 graphics card at 1.15 GHz, with 256 MB Video RAM (or around $266x10^6$ bytes), 16 cores, running CUDA version 3.1.

Simulation results for Π_1



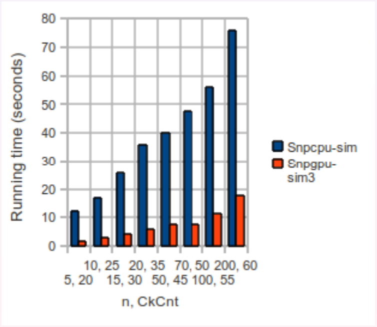
$$M_\Pi = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix}$$



Simulation results for Π_1 using different C_k values.

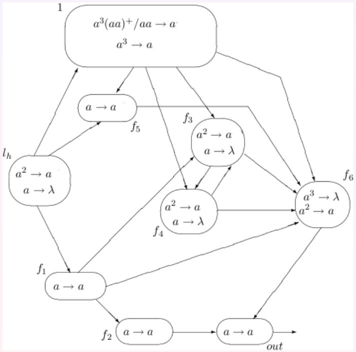
Speedup is $1.4\times$.

Simulation results for Π_2



Simulation results for Π_2 using different C_k values.

Speedup is 6.8$\times$.



SNP system Π_2 from [Ionescu et al. 2006].

14 rules and 9 neurons, 14 $\times$ 9 matrix

Note:

Max number of neurons allowable in current setup:

$$C_k = 266 \times 10^6 \; bytes/(16 \; bytes + 4 \; bytes \cdot |R|),$$

where

- Simulation requirements:
  $GbMem = 4 \cdot sizeof(C_k) + sizeof(M_\Pi)$,
  using standard *C* language *sizeof*() func'n (*int* type is 4 bytes).
- Max Global memory (*GbMem*) of used GPU: 266×10^6 bytes
- M_Π is $|R| \times |C_k|$ (total rules by total neurons).

Simulation limit is a function of R & C_k.

What do we get?

In this paper :

1. we have *snpgpu-sim3*, can now simulate SNP systems with regular expressions (those of the form (b-1)).
2. The speedup of *snpgpu-sim3* over *snpcpu-sim* for Π_1 went up to 1.4 times, while it was 6.8 for Π_2.
3. These results show that SNP system simulation on GPUs can greatly benefit from the parallel architecture of GPUs, and that increasing the parameters (of GPU) offer even larger speedups.
4. This benefit in speedup is coupled with the fact that the CUDA enabled graphics cards are readily available.
5. These cards offer boosts in general purpose computations as co-processors of commonly used CPUs, at a fraction of the power consumption of CPU clusters.

What do we do next?

1. Improve parallelism . . .
2. SNP system variants can be simulated by extending the current GPU simulator.
3. a generic P system parser based using P-lingua formatting for the GPU based SNP system simulator.

Thank You for Your Attention