A Massively Parallel Framework Using P Systems and GPUs

Jose M. Cecilia[1] **and Miguel A. Martínez-del-Amor**[2]

[1] Grupo de Arquitectura y Computación Paralela, Dpto. Ingeniería y Tecnología de Computadores, Universidad de Murcia, Spain
[2] Research Group on Natural Computing, Dpt. of Computer Science and Artificial Intelligence, University of Sevilla, Spain

Abstract

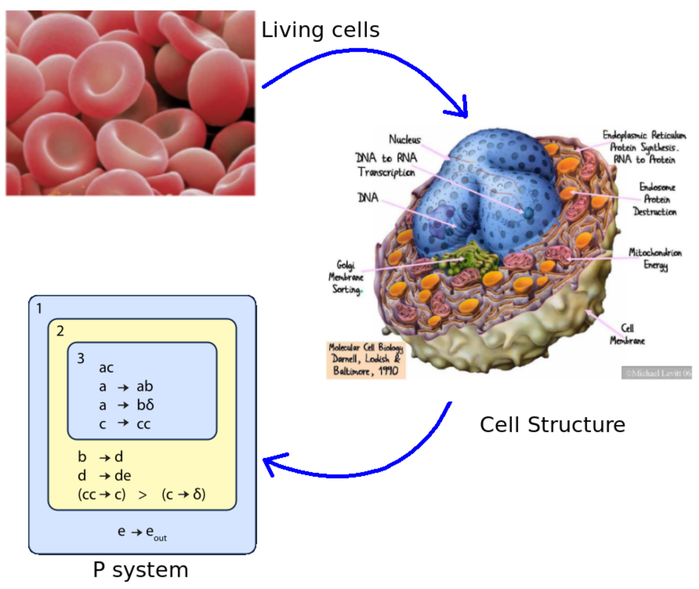
Since CUDA programing model appeared on the general purpose computations, the developers can extract all the power contained in the GPUs (Graphics Processing Unit) across many computational domains. Among those domains, P systems or membrane systems provide a high level computational modeling framework that allows to obtain polynomial time solutions to NP-complete problems by trading time for space, and also to model biological phenomena of computational systems biology. P systems are massively parallel distributed and its computation can be divided in two levels of parallelism: Membranes, that can be expressed as a blocked computation and Objects, that can be expressed as a threaded computation. This computational model naturally fits for CUDA programing model and the GPU. We present a simulator for the class of recognizer P systems with active membranes by using the CUDA programing model in order to exploit the massively parallel nature of those systems at maximum. Experimental results are shown on a Tesla C1060 GPU with a 60X of speed-up compared to the sequential code.

Motivation

- Towards an efficient implementation of a **simulator** for the class of recognizer **P systems** with active membranes, based on the massively parallel nature that they have by their definition:
  - P systems are an approach to obtain, in theory, polynomial time solutions to NP-complete problems by trading time for space.
  - P systems are an alternative to model biological phenomena in the area of computational systems biology.
  - Up to now, it has not been possible to have implementations neither *in vivo* nor *in vitro* of P systems.
  - The manipulation and analysis of these devices is performed by simulations on conventional computers.
- Looking for an alternative approach to extract **parallelism** on the GPU by using P systems.

Union of two areas based on parallelism: Membrane Computing and Graphics Processing Unit

Membrane Computing and P systems



- **Membrane Computing** is an emerging branch within Natural Computing (*Unconventional Computing*).
- The main idea is to consider biochemical processes taking place inside living cells from a computational point of view.
- The devices of this model are called **P Systems**.
- P systems are distributed parallel computing devices, processing multisets of abstract objects by means of various types of rules.
- P systems consist of a set of *syntactic* components:
  - A *membrane structure*: it is formed by a rooted tree of membranes arranged hierarchically inside a root membrane called *skin*, delimiting *regions*.
  - *Multiset of objects*: corresponding to chemical substances present in the compartments of a cell.
  - *Rules*: corresponding to chemical reactions that can take place inside the cell.
- A computation of a P system is a sequence of instantaneous transitions between *configurations*.
- The computation is driven by a global clock that synchronizes the execution.

P systems with active membranes

The model of P system with active membranes is a construct of the form:

$$\Pi = (O, H, \mu, \omega_1, \ldots, \omega_m, R)$$

- $m \geq 1$ is the initial degree of the system.
- O is the alphabet of *objects*.
- H is a finite set of *labels* for membranes.
- μ is a membrane structure (a rooted tree), consisting of m membranes injectively labelled with elements of H.
- $\omega_1, \ldots, \omega_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ.
- R is a finite set of *rules*:

(a) Evolution rules: $[a \rightarrow v]_h^\alpha$ where $h \in H$, $\alpha \in \{+, -, 0\}$ (electrical charges), $a \in O$ and v is a string over O.

(b) Send-in communication rules: $a\,[\ ]_h^\alpha \rightarrow [b]_h^\beta$ where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$.

(c) Send-out communication rules: $[a]_h^\alpha \rightarrow [\ ]_h^\beta b$ where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$.

(d) Dissolution rules: $[a]_h^\alpha \rightarrow b$ where $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in O$.

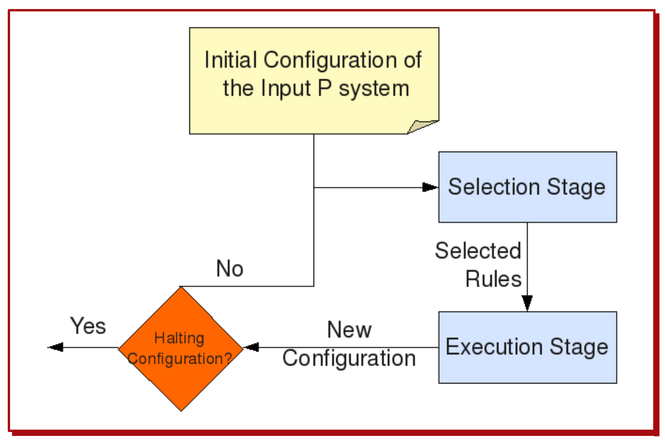
(e) Division rules: $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$ where $h \in H$,$\alpha, \beta, \gamma \in \{+, -, 0\}$, $a, b, c \in O$.

- The rules are used in a maximal parallel way: in one step, each object in a membrane can only be used by at most one rule (non-deterministically chosen), but any object which can evolve by a rule must do it.
- Rules (b) to (e) cannot be applied simultaneously in a membrane.

G. Păun: Membrane Computing, An introduction. Springer-Verlag, Berlín (2002).

Simulation of P systems

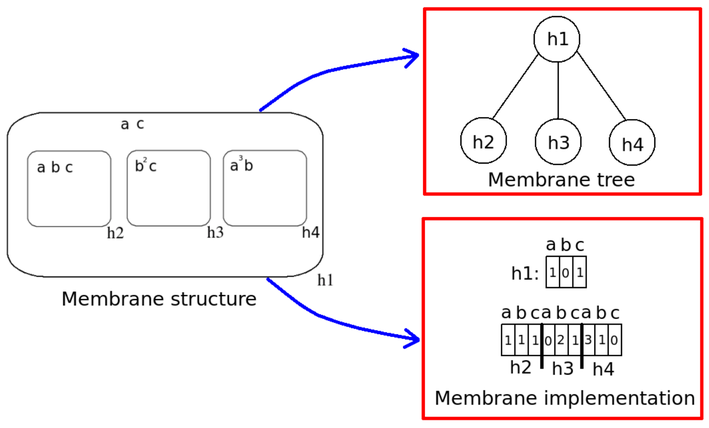
- The simulation is divided into two stages:
  - **Selection stage**: the search for the rules to be executed in each membrane.
  - **Execution stage**: the execution of the rules previously selected.
- Global synchronization is needed between the stages, because the membrane structure is modificated by the execution stage.



- We introduce the following priorities among rules in our simulator to accelerate the simulation: dissolution, evolution, send-out, send-in and division.

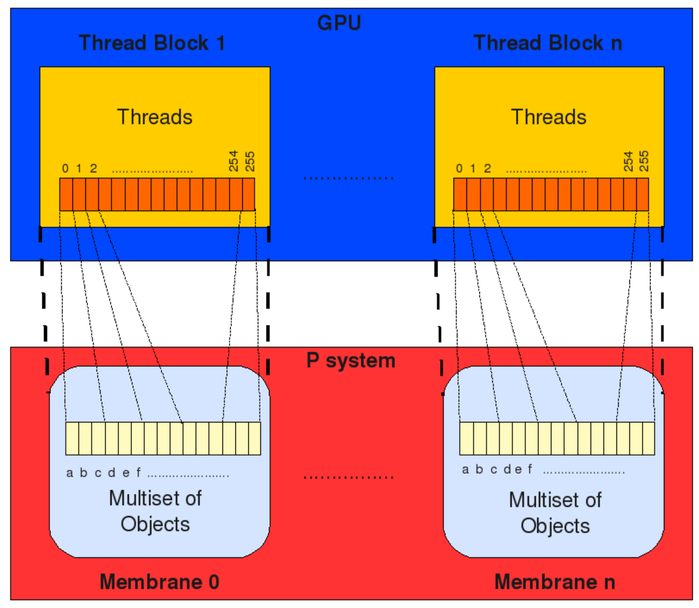
Data structures

- The simulator considers two levels of membrane hierarchy: the skin and the rest of elementary membranes.
- Communication among membranes is only presented between the skin and the elementary membranes.



Design of the simulator in CUDA

- Each CUDA thread block represents a membrane.
- Each thread represents at least one object per membrane.
  - If there are more objects than allowed threads per block (512 in CUDA programing model), they are equally distributed.
- The best performance is reached using 256-threads per thread block.



- The *selection stage* is implemented as a CUDA kernel, which includes part of the *execution stage*:
  - Each individual thread is responsible for identifying if there are some rules associated with the object(s) that it represents.
  - The rules are selected respecting the priorities defined before.
  - All the evolution rules previously selected are stored.
  - For the other rules, each thread block only stores one of them.
  - Finally, the threads executes the evolution rules related to them, after synchronization requirements: they only entails *block-level synchronization*.
- The rest of the *execution stage* is implemented as different CUDA kernels, one kernel per each kind of rule.
  - Dissolution and division kernels use as much thread blocks as membranes.
  - Send-in and send-out kernels only use a thread per membrane.
- The kernels are executed depending on the selected rules.
- The rules that entail communication with the skin membrane (send-in, send-out and dissolution) are implemented by using atomic instructions on the device memory.

Conclusions and future work

- Using the power that provides GPUs to simulate P systems with active membranes is a new concept in the development of applications for membrane computing.
- P systems are an alternative approach to extract all performance available on GPUs due to its parallel nature.
- This simulator is limited by the available resources on the GPU as well as the CPU.
  - In the following versions, we will reduce the memory requirements to handle bigger instances of NP-complete problems.
- Our aim is to fully simulate P systems with active membranes:
  - To include several levels in the membrane struture.
  - To implement not elementary division in the simulator.
- In forthcoming versions, we will adapt our simulator to simulate specific problems at maximum performance.
- We will develop simulators for other kind of P systems, such as probabilistic P system model or stochastic P system model, which are useful to attack within the framework of computational systems biology.
- The newest clusters of GPUs provide a higher massively parallel environment, so we will attempt to scale to those systems to obtain better performance in our simulated codes and also more memory space for our simulations.