GPU technology in Membrane Computing: an overview

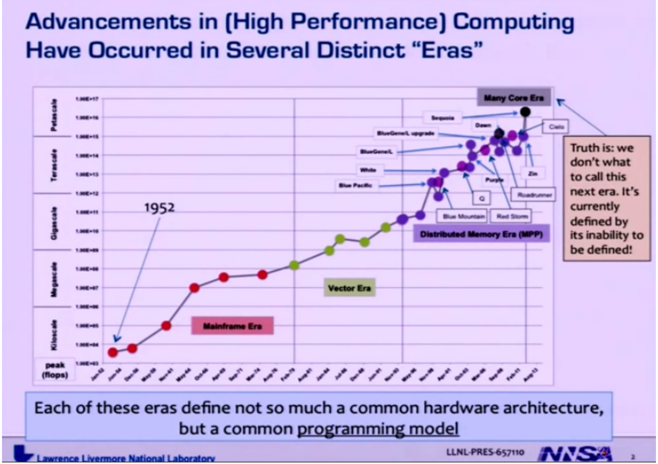
Miguel A. Martínez–del–Amor[1], Manu García–Quismondo[2],
Luis F. Macías-Ramos[3], Mario J. Pérez-Jiménez[3]

[1]Digital Cinema Group, **Fraunhofer IIS**
[2]Department of Fisheries, Wildlife and Conservation Biology, **University of Minnesota**
[3]Research Group on Natural Computing, **University of Seville**

February 3, 2015

Outline

Eras of HPC



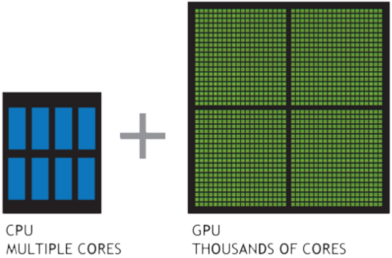
Extracted from `HPCwire.com`

HPC trends

- Current technologies:
  - Coarse-grain parallelism:
    - Grid computing
    - Supercomputers
    - Clusters
  - Fine-grain parallelism:
    - FPGAs
    - Multicore processors (+ vectorial extensions)
    - Manycore processors (e.g. GPUs, Intel Phi, etc.)

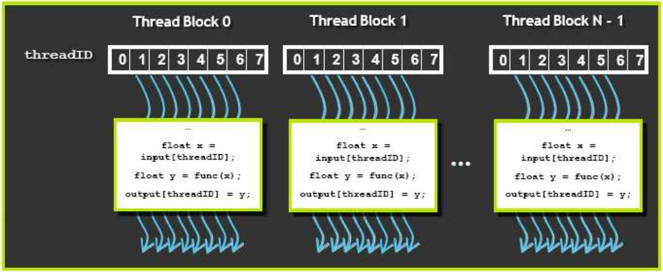
GPU computing

- Graphics Processor Unit (GPU)
- Data-parallel computing model:
  - SPMD programming model (***S**ame **P**rogram for **M**ultiple **D**ata*)
  - Shared memory system
- New programming languages: CUDA and OpenCL
- A GPU features thousand of cores



NVIDIA's technology

- CUDA programming model[1]
  - Heterogeneous model: CPU (**host**) + GPU (**device**).
  - All threads execute the same code (**kernel**) in parallel.
  - Three-level hierarchy of threads (**grid**, **blocks**, **threads**).
  - Memory hierarchy (**global**, **shared** within block).
  - Easy **synchronization** by barrier operations (inside blocks).



[1] W. M. Hwu, D. Kirk: Programming massively parallel processors. Morgan Kaufmann, 2010.

CUDA programming model

Threading hints

- Threads have associated an ID (3 dimensional, inside block).
- Blocks have associated an ID (2 dimensional, inside grid).
- Combine them to get a *global* ID for each thread.
- Parallel granularity: *warp*.
- Emphasize parallelism:
    - Launch as many threads as possible (thousands of them).
    - Keep them synchronized (*inside the blocks*).
    - Use the barrier to synchronize: `__syncthreads()`.

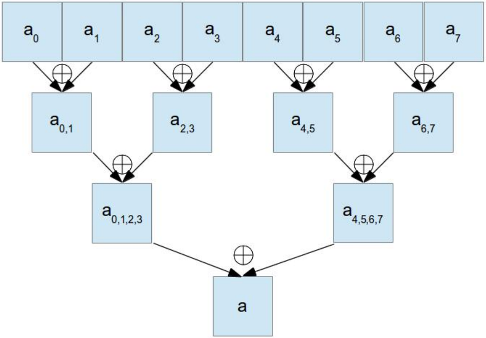
CUDA programming model

Memory hints

- Explicitly and manually managed.
- Store your input and output data in global memory.
- When processing, try to copy the data to shared memory.
- Exploit memory bandwidth:
  - Avoid transferring data CPU-GPU when possible.
  - Exploit shared memory and registers as much as possible.
  - Coalesced access: *contiguous threads accessing to contiguous elements in memory*.

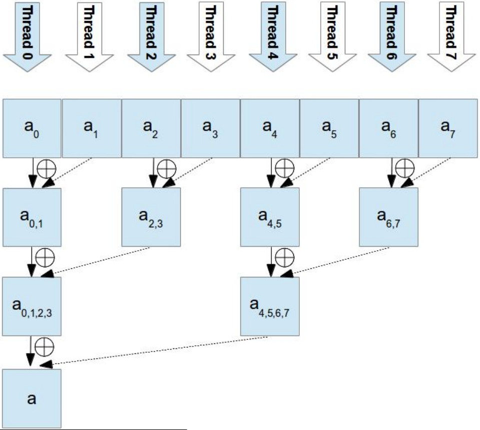
The Reduction Problem

- *How to calculate the total sum of the elements of an array?*
- Input: A set of n elements $\{a_1, \ldots, a_n\}$, and an associative binary operator $\oplus$.
- Output: An element $a = a_1 \oplus a_2 \oplus \ldots \oplus a_n$
- Common parallel solution: a tree of partial solutions



Implementation of the tree solution in CUDA

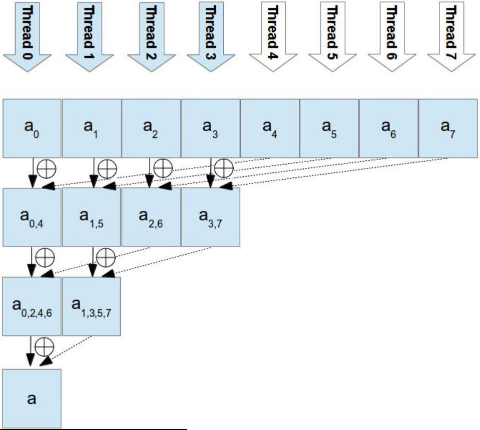
- A very simple implementation of the tree[2]:



[2] R. Ceterchi, M.A. Martínez-del-Amor, M.J. Pérez. The Reduction Problem in CUDA and Its Simulation with P Systems. BWMC14 proc, 2014, 91–102

Implementation of the tree solution in CUDA

- A sequential-addressing implementation of the tree (+4x)[3]:



[3] R. Ceterchi, M.A. Martínez-del-Amor, M.J. Pérez. The Reduction Problem in CUDA and Its Simulation with P Systems. BWMC14 proc, 2014, 91–102

Recent developments in GPU technology

- Nvidia Fermi (Capability 2.x):
  - Up to 448 cores.
  - Cache memories.
  - Support for atomic floating point operations.
  - Support for double precision floating point.
  - Full support for on–device C++.
  - IEEE–32 compliant floating point.

Tesla C2050

Recent developments in GPU technology

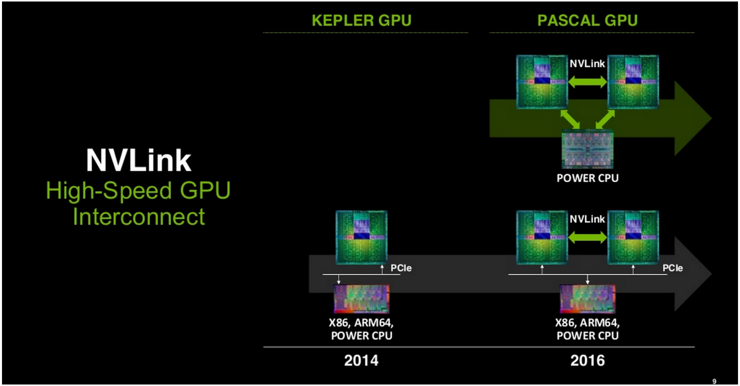
- Nvidia Kepler (Capability 3.x):
  - Up to 2880 cores
  - Dynamic parallelism: Launch kernels from inside kernels.
  - Hyper–Q: Enable multiple CPU cores to use a common GPU device.

Tesla K80

- Nvidia Maxwell (Capability 5.x):
  - Energy consumption improved.
  - Larger memory spaces.

GeForce GM204

NVIDIA Road map



Need for efficient simulation

- P system simulators assist us in:
  - Formal verification of P systems.
  - Computational modeling based on P systems:
    - Experimentally validate models.
    - Conduct virtual experiments.
- Necessity of efficiency to handle large-size instances.
- Sequential simulators serialize the parallelism: twice inefficient.

Simulating P systems on parallel platforms

- Quality attributes of computing platforms[4]:
  - Performance
  - Flexibility
  - Scalability
- Types of computing platforms[1]:
  - Sequential computing platforms.
  - Software-based parallel computing platforms.
  - Hardware-based parallel computing platforms.

[4] V. Nguyen, D. Kearney, G. Gioiosa. Balancing performance, flexibility, and scalability in a parallel computing platform for Membrane Computing applications. LNCS, **4860** (2009), 385-413.

Why is the GPU interesting for simulating P systems?

- Interesting properties:
  - High level of parallelism *(from 16 to 2800 cores)*
  - Shared memory system *(easily synchronized)*
  - Scalability *(multi-GPU systems)*
  - Cheap technology *(cost and maintenance)*

Tesla C1060

- NVIDIA's Tesla GPUs at RGNC
  - Tesla C1060: *240 cores, 4 GB memory.*
  - Tesla K40: *2880 cores, 12 GB memory.*

GPU simulation in Membrane Computing

PMCGPU[5]: Parallel simulators for Membrane Computing on the GPU:

- An intended repository for GPU–based simulators in Membrane Computing.
- Projects:
  - P systems with active membranes
  - SAT solutions in active membranes and tissue with cell division.
  - Probabilistic Dynamic P (PDP) systems.
  - Enzymatic Numerical P systems (ENPS).

[5] `http://sourceforge.org/p/pmcgpu`

Other projects

Projects using GPU for the simulation within Membrane Computing, outside PMCGPU:

- Spiking Neural P systems without delays, no division, no budding, no plasticity (*Cabarle et al. 2012*).
- Evolution-Communication P systems with energy and without antiport rules (*Juayong et al. 2012*).
- Kernel P systems, an ad-hoc solution to Subset Sum problem (*Ipate et al. 2013*).
- Improved simulation of P systems with active membranes (*Maloosi et al. 2014*).
- Tissue P systems for image processing solutions (*Díaz-Pernil et al. 2013*)

PGP systems

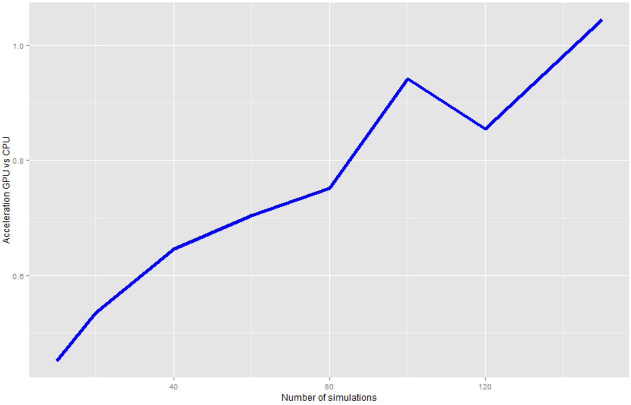
Portfolio:

- A model in Membrane Computing for population dynamics.
- Applied to study the population levels and distribution of several genotypes of butterfly *Pieris oleracea* in a forest ecosystem under different scenarios.

PGP systems
Course of action:

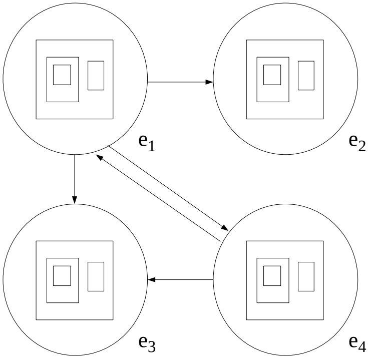
1. Experts on the species determine that 20 simulations per scenario are necessary to verify that the model reproduces the dynamics of the butterfly population under study.
2. A seminar simulator for PGP systems in Java (inside P–Lingua) is developed.
   - Not good enough performance.
3. A simulator in C++ is developed.
   - 10-20 minutes for 20 simulations $\rightarrow$ still not acceptable.
4. A CUDA/C++ simulator is developed.
   - Poor performance, requires large simulation batches to outperform C++.
5. Pivotal design improvement in C++ simulator.
   - Dramatic acceleration of simulations. Throughput = 1 simulation per second.
6. No longer need for CUDA/C++ simulator. Discontinued.

PGP systems



- Acceleration achieved for CUDA/C++ simulator around 1x for 150 simulations.

Population Dynamics P systems (probabilistic model)



Skeleton rules

$$u\ [\ v\ ]_h^{\alpha} \rightarrow u'\ [\ v'\ ]_h^{\beta}$$

Rules of P sytem in environment j

$$u\ [\ v\ ]_h^{\alpha} \xrightarrow{f_{r,j}} u'\ [\ v'\ ]_h^{\beta}$$

Movement rules of environment j

$$(x)_{e_j} \xrightarrow{pr} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$$

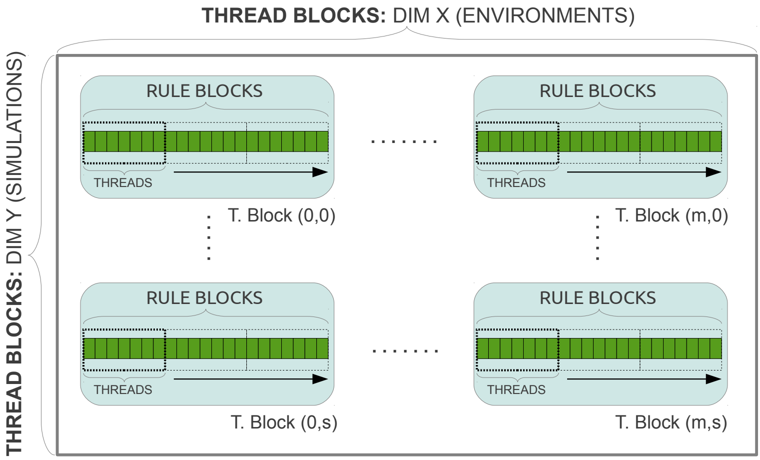
- Rules are applied in a *maximal parallel way according to their probabilities*
- Rules having the same left-hand side are classified into blocks

Simulation Algorithm: DCBA

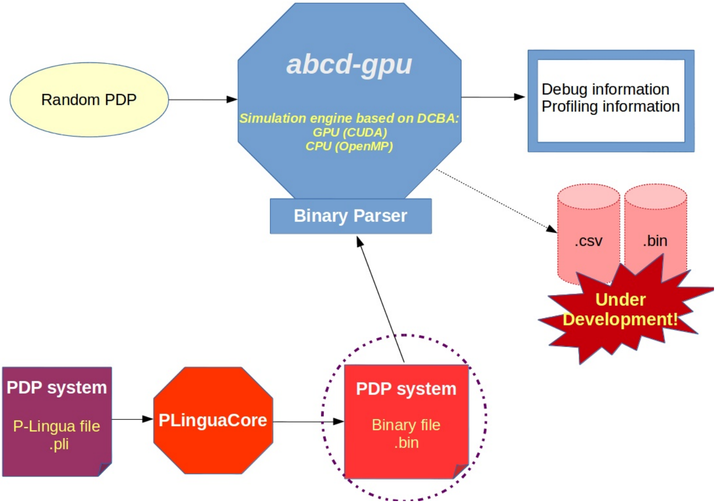
Simulation algorithm

1. **Init.**: static distribution table (**columns:** blocks, **Rows:** objects)
2. **Loop over Time**
3. **Selection** stage:
4. Phase 1 (Distribution of objects along rule blocks)
5. Phase 2 (Maximality selection of rule blocks)
6. Phase 3 (Probabilistic distribution, blocks to rules)
7. **Execution** stage

Parallel general design



ABCD-GPU roadmap



Profiling random PDP systems

	Test A (mean LHS length 1.5)			Test B (mean LHS length 3)		
	% CPU	% GPU	Acc	% CPU	% GPU	Acc
Phase 1	53.7%	30.1%	14.23x	55.3%	12%	8.52x
Phase 2	12.6%	47%	2.13x	18.4%	82.8%	0.4x
Phase 3	22.6%	13.7%	13.2x	14%	2.2%	11.72x
Phase 4	11.1%	9.2%	9.7x	12.3%	3%	7.43x
Total			7.9x			1.8x

Profiling the Bearded Vulture ecosystem model (2008)

	Tesla C1060			GTX550		
	% CPU	% GPU	Acc	% CPU	% GPU	Acc
Phase 1	53.8%	56%	4.2x	53.8%	61.2%	12.6x
Phase 2	1.6%	2%	3.4x	1.6%	6.5%	3.5x
Phase 3	37%	9.4%	17.2x	37%	22.8%	23.3x
Phase 4	7.6%	32.6%	1.02x	7.6%	9.5%	11.6x
Total			4.38x			14.4x

Usefulness of GPU technology in Membrane Computing

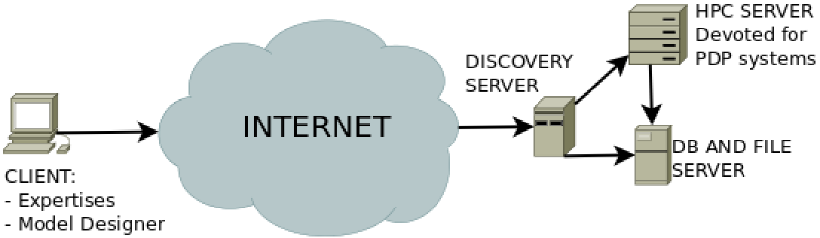
When to apply GPU technology to simulate P systems?

- GPU is a promising technology for simulation in Membrane Computing.
- Nevertheless, its application is not always required/advisable.
- Some drawbacks to take into consideration:
  - It is necessary to assess its potential benefits case by case.
  - Apply to problems involving parallel processing of huge amounts of data (full usage of GPU resources).
  - Requires a good command of the GPU architecture and CUDA.
  - It requires a minimum infrastructure.

Future work

- Connecting simulators with simulation libraries
- Simulating other models with HPC-simulation needs
- Improving existing simulators (more efforts in data structures)
- Improving simulation algorithms (extract more parallelism)
- Using new parallel technology
- Studying models for parallel simulators (*GP systems*)
- Provide theoretical results in GPU computing through MC

Work in progress in a R&D Project in development



Future work: challenges on PDP systems

- Simulation Algorithm and Implementation:
    - Finalize phase 2 (how to disorder+compact an array?)
    - Improve phase 2 (is it necessary?).
    - Improve memory management (push the memory bound).
    - Improve random numbers (real binomial RNG?).
- Simulation of PDP systems:
    - Model oriented optimizations (why DCBA globally?).
    - Parallel P-Lingua (directives from user).
    - Symbolic simulation.
    - Visualization of simulation (OpenGL?).

Thank you very much

Questions?

Interested? Let talk by skype or email

(mdelamor@us.es,mgarciaquismondo@us.es)

Acknowledgments:

M.A. Martínez-del-Amor acknowledges the ERCIM Alain Bensoussan fellowship program.

The authors also acknowledge the support of the project TIN2012-37434, funded by MINECO of Spanish government, and the NVIDIA CUDA Research Center program.