Enhancing the Simulation of PDP Systems in GPUs

Miguel Ángel Martínez-del-Amor, Andrés Doncel and Sevilla Team

[1] Research Group on Natural Computing, Dpt. of Computer Sciences and Artificial Intelligence
University of Seville (Spain)

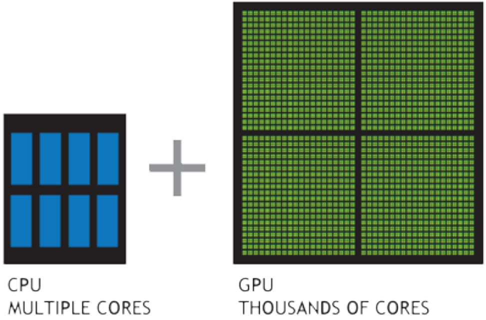
18th Brainstorming Week on Membrane Computing
February 2020 (Sevilla, Spain)

Outline

Outline

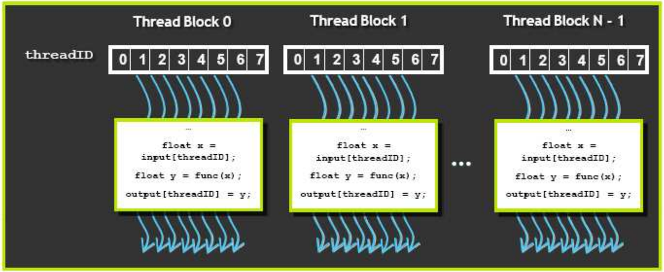
GPU computing

- 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, OpenCL, DirectCompute
- A GPU features thousand of cores



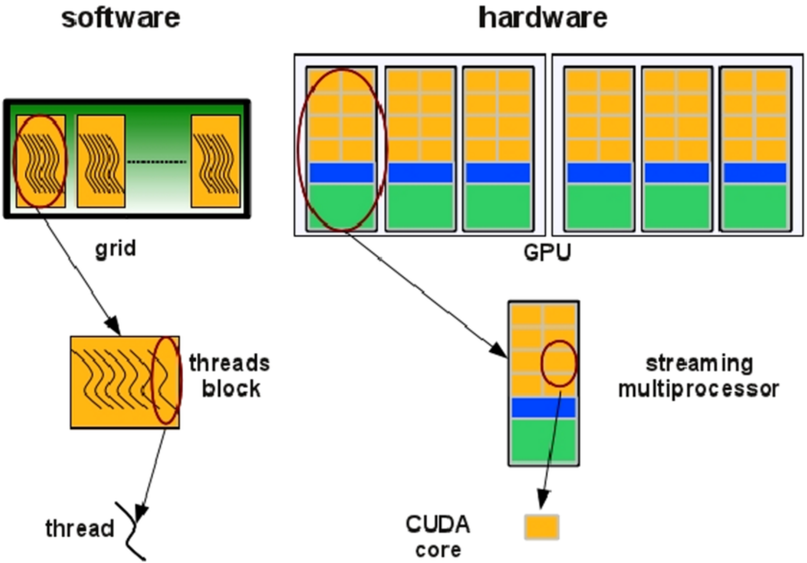
NVIDIA's technology

- 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).

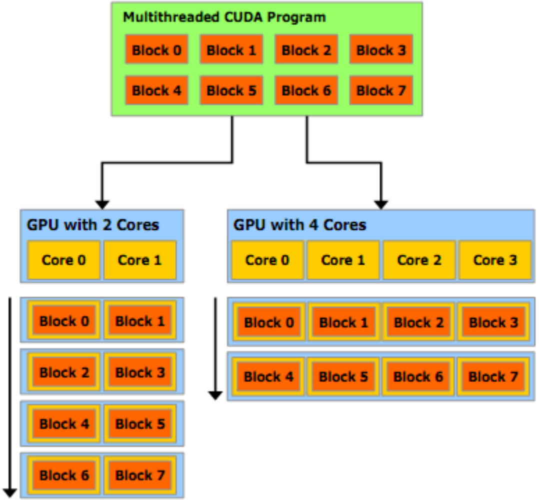


[1] W.-M. Hwu, D. Kirk. Programming massively parallel processors, Morgan Kaufmann, 2010.

GPU: software vs hardware



GPU: scalable architecture



A multithreaded program is partitioned into blocks of threads that execute independently from each other, so that a GPU with more cores will automatically execute the program in less time than a GPU with fewer cores.

NVIDIA GPUs evolution till 2018

	"Fermi"	"Fermi"	"Kepler"	"Kepler"	"Maxwell"	"Pascal"	"Volta"
Tesla GPU	**GF100**	**GF104**	**GK104**	**GK110**	**GM200**	**GP100**	**GV100**
Compute Capability	2.0	2.1	3.0	3.5	5.3	6.0	7.0
Streaming Multiprocessors (SMs)	16	16	8	15	24	56	84
FP32 CUDA Cores / SM	32	32	192	192	128	64	64
FP32 CUDA Cores	512	512	1536	2880	3072	3584	5376
FP64 Units	-	-	512	960	96	1792	2688
Tensor Core Units							672
Threads / Warp	32	32	32	32	32	32	32
Max Warps / SM	48	48	64	64	64	64	64
Max Threads / SM	1536	1536	2048	2048	2048	2048	2048
Max Thread Blocks / SM	8	8	16	16	32	32	32
32-bit Registers / SM	32768	32768	65536	65536	65536	65536	65536
Max Registers / Thread	63	63	63	255	255	255	255
Max Threads / Thread Block	1024	1024	1024	1024	1024	1024	1024
Shared Memory Size Configs	16 KB	16 KB	16 KB	16 KB	96 KB	64 KB	Config
	48 KB	48 KB	32 KB	32 KB			Up Tp

NVIDIA's new GPU: Turing

Key attributes of the new GPUs:

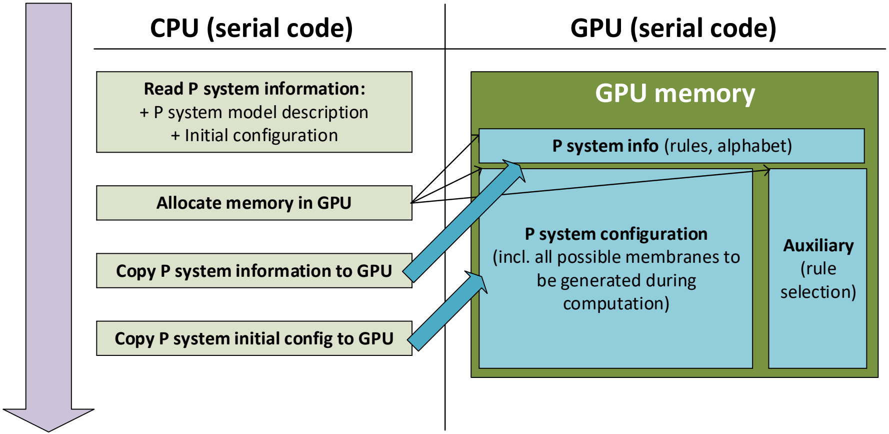
GPU	Memory	Memory with NVLink	Ray Tracing	CUDA Cores	Tensor Cores
Quadro RTX 8000	48GB	96GB	10 GigaRays/sec	4,608	576
Quadro RTX 6000	24GB	48GB	10 GigaRays/sec	4,608	576
Quadro RTX 5000	16GB	32GB	6 GigaRays/sec	3,072	384

Why is the GPU interesting for simulating P systems?

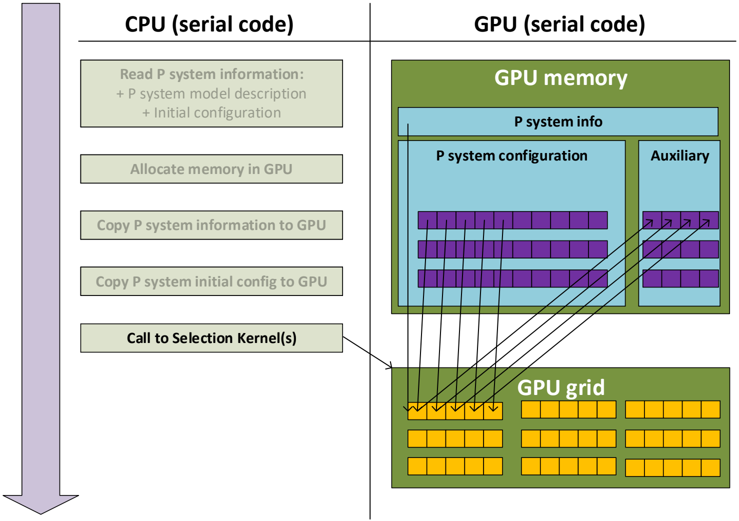
- Desired properties:
  - High level of parallelism *(up to 4000 cores)*
  - Shared memory system *(easily synchronized)*
  - Scalability and *portability*
  - Known languages: C/C++, Python, Fortran...
  - Cheap technology everywhere *(cost and maintenance)*
- Undesired properties:
  - Best performance requires lot of research.
  - Programming model imposes many restrictions

Outline

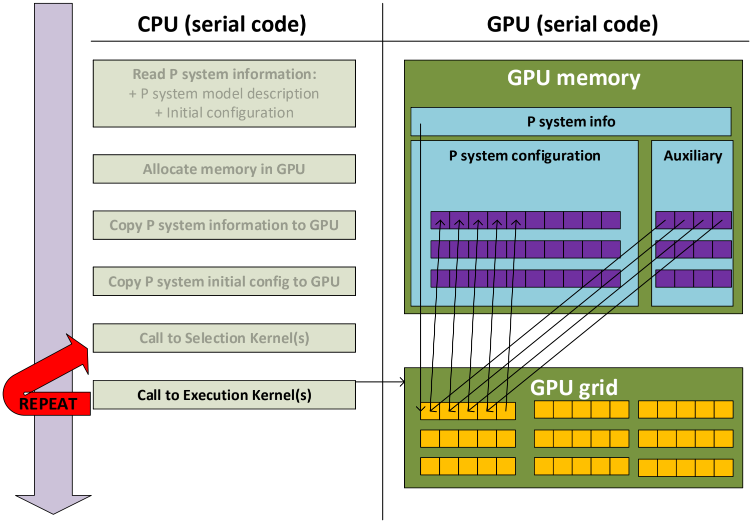
GPU simulator workflow - Initialization (I)



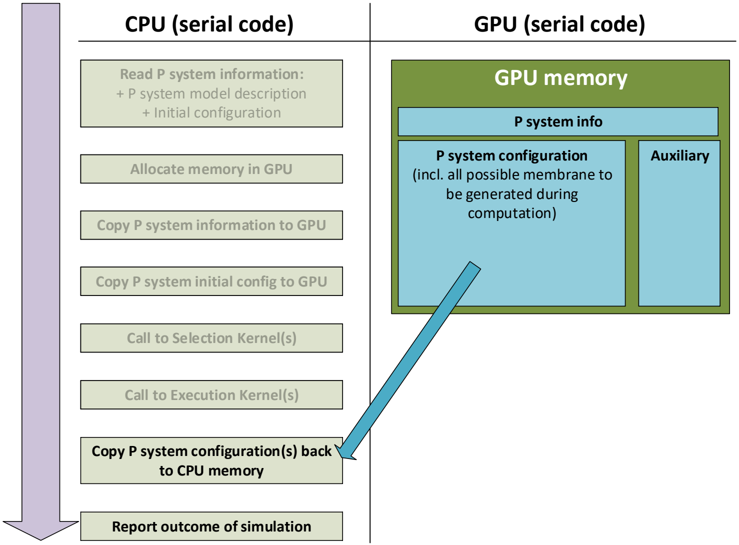
GPU simulator workflow - Simulation - Selection (II)



GPU simulator workflow - Simulation - Execution (III)



GPU simulator workflow - Wrap up (IV)



Simulation approaches

- **Generic** approach: simulator for a variant / class (under restrictions).
- **Specific** approach: simulator for a certain family / model.

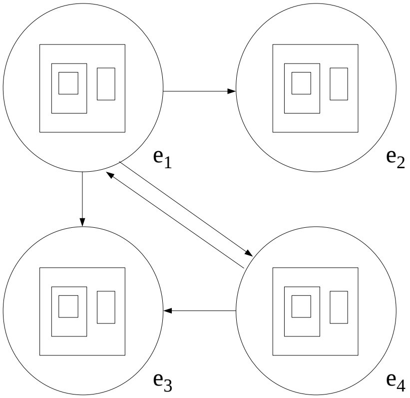
PMCGPU project: http://sourceforge.net/projects/pmcgpu

Simulator Codename	P system model and coverage	Peak speedup	GPU tested
PCUDA	(G) Active membranes	7x (T) 1.67x (R)	C1060
PCUDASAT	(S) Active membranes	63x (R)	C1060
TSPCUDASAT	(S) Tissue w/ cell division	10x (R)	C1060
ABCDGPU	(G) Population Dynamics	18.1x (T) 5x (R)	K40
ENPS-GPU	(G) Enzimatic Numerical	10x (T)	GTX460M
CuSNP	(G) Spiking Neural	50x (R)	GTX750

G= Generic, S=Specific, T=Stress testing, R=Real examples.

Outline

Population Dynamics P systems



Skeleton rules

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

Environment rules

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

Rules are applied in a ***maximal*** *parallel way according to their* ***probabilities***

Simulatin algorithms

Algorithms for probabilistic behaviour:

- BBB[2]: Binomial Block Based algorithm.
- DNDP[3]: Direct Non Deterministic algorithm with Probabilities.
- **DCBA**[4]: Direct distribution based on Consistent Blocks Algorithm.

General scheme

1. **Selection** stage.
2. **Execution** stage.

---

[2] **A uniform framework for modeling based on P Systems**. M.A. Colomer et al, *Proc. BIC-TA*, vol. 1 (2010), pp. 616–621.

[3] **A simulation algorithm for multienvironment probabilistic P systems: A formal verification**. M.A. Martínez-del-Amor et al, *Int. Journal of Foundations of Computer Science*, 22, 1 (2011), 107-118.

[4] **DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution**. M.A. Martínez-del-Amor et al. *Proc. 13th CMC* (2012), pp. 291-310

DCBA

Rules classified into consistent blocks:

- $B_{i,\alpha,\alpha',u,v}$
- $B_{e_j,x}$

Blocks vs competition

1. $a\ [\ b\ ]_1^0 \xrightarrow{0.8} c^3\ [\ ]_1^+$ and $a\ [\ b\ ]_1^0 \xrightarrow{0.2} [\ c^3\ ]_1^+ \Longrightarrow B_{i=1,\alpha=0,\alpha'=+,u=\{a\},v=\{b\}}$
2. $a^2\ [\ b\ ]_2^0 \xrightarrow{1} c\ [\ ]_1^+$ and $a^4\ [\ c^2\ ]_2^0 \xrightarrow{1} [\ d^5\ ]_1^+ \Longrightarrow$ **Competing rules!**

DCBA

Rules classified into consistent blocks:

- $B_{i,\alpha,\alpha',u,v}$
- $B_{e_j,x}$

Blocks vs competition

1. $a\,[\,b\,]_1^0 \xrightarrow{0.8} c^3\,[\,]_1^+$ and $a\,[\,b\,]_1^0 \xrightarrow{0.2} [\,c^3\,]_1^+ \Longrightarrow B_{i=1,\alpha=0,\alpha'=+,u=\{a\},v=\{b\}}$
2. $a^2\,[\,b\,]_2^0 \xrightarrow{1} c\,[\,]_1^+$ and $a^4\,[\,c^2\,]_2^0 \xrightarrow{1} [\,d^5\,]_1^+ \Longrightarrow$ **Competing rules!**

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

DCBA: Direct distribution based on Consistent Blocks Algorithm

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

Constructing static table:

$B1 \equiv [a^2\ c]_1^0$

$B2 \equiv [a^4\ b]_1^0$

$B3 \equiv [b\ d^5]_1^-$

	B1	B2	B3	*Sum*
a	*1/2*	*1/4*		
b		*1*	*1*	
c	*1*			
d			*1/5*	
MIN				

DCBA: Direct distribution based on Consistent Blocks Algorithm

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

$$B1 \equiv [a^2\ c]^0_1 \Big|\ B2 \equiv [a^4\ b]^0_1 \Big|\ B3 \equiv [b\ d^5]^-_1$$

1. Filters:

	B1	B2	B3	*Sum*
a	*1/2*	*1/4*		
b		*1*	*1*	
c	*1*			
d			*1/5*	
MIN				

DCBA: Direct distribution based on Consistent Blocks Algorithm

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

$$B1 \equiv [a^2\ c]_1^0 \Big| \ B2 \equiv [a^4\ b]_1^0 \Big| \ B3 \equiv [b\ d^5]_1^-$$

2. Normalization:

	B1	**B2**	**B3**	*Sum*
a	*1/2 $\mid$ 2/3*	*1/4 $\mid$ 1/3*		*3/4*
b		*1*	*1*	*1*
c	*1*			*1*
d			*1/5*	
MIN				

DCBA: Direct distribution based on Consistent Blocks Algorithm

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

$$B1 \equiv [a^2\ c]_1^0 \,\Big|\, B2 \equiv [a^4\ b]_1^0 \,\Big|\, B3 \equiv [b\ d^5]_1^-$$

3. Minimums:

	B1	B2	B3	*Sum*
a*10	*1/2* $\mid$ *2/3*	*1/4* $\mid$ *1/3*		*3/4*
b*5		*1*	*1*	*1*
c*90	*1*			*1*
d			*1/5*	
MIN	3	0		

Repeat for better accuracy

DCBA: Direct distribution based on Consistent Blocks Algorithm

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

Random order to blocks
Check Maximality

$B1 * 3$, $B2 * 1$

DCBA: Direct distribution based on Consistent Blocks Algorithm

DCBA's general scheme

1. **Initialization**: *static distribution table*
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1**: *Distribution*
5. **Phase 2**: *Maximality*
6. **Phase 3**: *Probability*
7. **Execution** stage

From blocks to rules applications

Using multinomial distribution

$$B1 * 6 =$$

$$r1 \equiv [a^2\ c]_1^0 \xrightarrow{0.7} [d^2]_1^+$$

$$r2 \equiv [a^2\ c]_1^0 \xrightarrow{0.2} [c^2]_1^+$$

$$r2 \equiv [a^2\ c]_1^0 \xrightarrow{0.1} [a^2]_1^+$$

$$M(6, 0.7, 0.2, 0.1) \approx \{3, 2, 1\}$$

Outline

Previous work

DCBA in P-Lingua simulation framework

- The static table is a *hash table*
- **DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution.** M.A. Martínez-del-Amor et al. *Proc. 10th BWMC*, 2, 27–56 (2012)

C++ and OpenMP implementations

- The static table is not implemented (*virtual table*)
- **Parallel Simulation of Probabilistic P Systems on Multicore Platforms.** M.A. Martínez-del-Amor et al. *Proc. 10th BWMC*, 2, 17–26 (2012)

Previous work

DCBA in P-Lingua simulation framework

- The static table is a *hash table*
- **DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution**. M.A. Martínez-del-Amor et al. *Proc. 10th BWMC*, 2, 27–56 (2012)

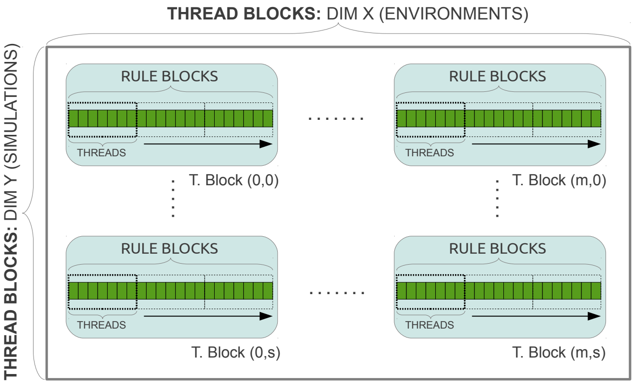
C++ and OpenMP implementations

- The static table is not implemented (*virtual table*)
- **Parallel Simulation of Probabilistic P Systems on Multicore Platforms**. M.A. Martínez-del-Amor et al. *Proc. 10th BWMC*, 2, 17–26 (2012)

A CUDA general design

How to distribute the parallelism?

- Thread blocks: DCBA to **simulations** and **environments** in parallel.
- Threads: DCBA steps to **blocks** in parallel and synchronously.



Typical number of **simulations**: 50–100
Typical number of **environments**: 2–20
Typical number of **rule blocks**: 500–100,000

Phase 1

Each part in a separate kernel

- Kernel for Filters
- Kernel for Normalization and minimums: atomic operations
- Kernel for Updating and filters: atomic operations

Phase 2

The most challenging part when parallelizing by blocks

- Inherently sequential (block after block).
- Requires random order over rule blocks (simulated by CUDA scheduler).
- Improvement: pre-calculate block **competitions** on **shared memory**.

Phase 2

	B0	B1	B2	B3
LHS	A B C	A	D E	A B
order	-	-	-	-

Iteration 0

	B0	B1	B2	B3
LHS	A B C	(0,0)	D E	(0,0)(0,1)
order	0	-	-	-

Iteration 1

	B0	B1	B2	B3
LHS	A B C	(0,0)	D E	(1,0)(0,1)
order	0	1	-	-

Iteration 2

	B0	B1	B2	B3
LHS	A B C	(0,0)	D E	(1,0)(0,1)
order	0	1	0	-

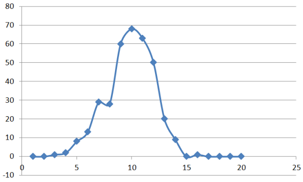
Iteration 3

	B0	B1	B2	B3
LHS	A B C	(0,0)	D E	(1,0)(0,1)
order	0	1	0	2

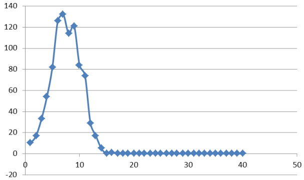
Phase 3

A random number generator on the GPU

- Block to rules applications: **multinomial distribution.**
- Requires generation of binomial random variates: $X \sim B(n,p)$.
- Our implementation: `CURNG_BINOMIAL` library
  - If $n \cdot p > 10$, normal approximation (using `CU_RAND` library)
  - Else, BINV[1] algorithm ($O(n \cdot p)$).



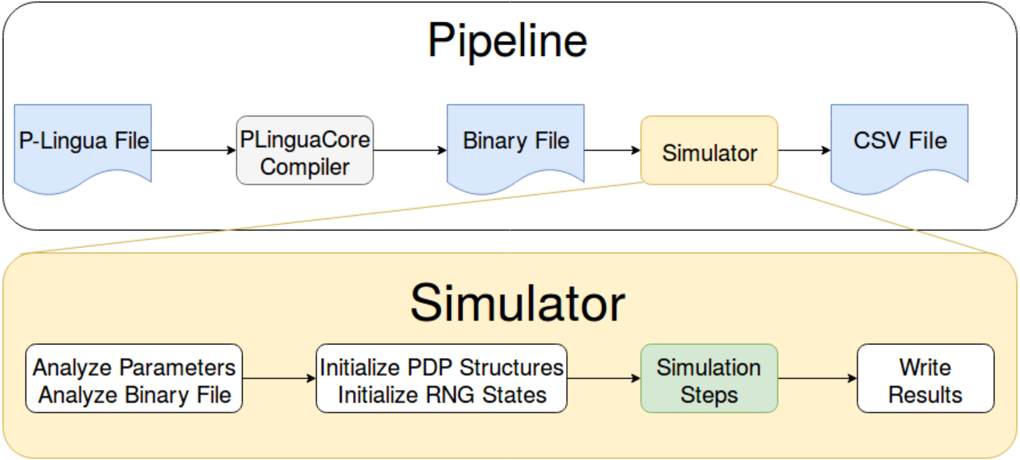
$500 * X \sim B(20, 0.5)$



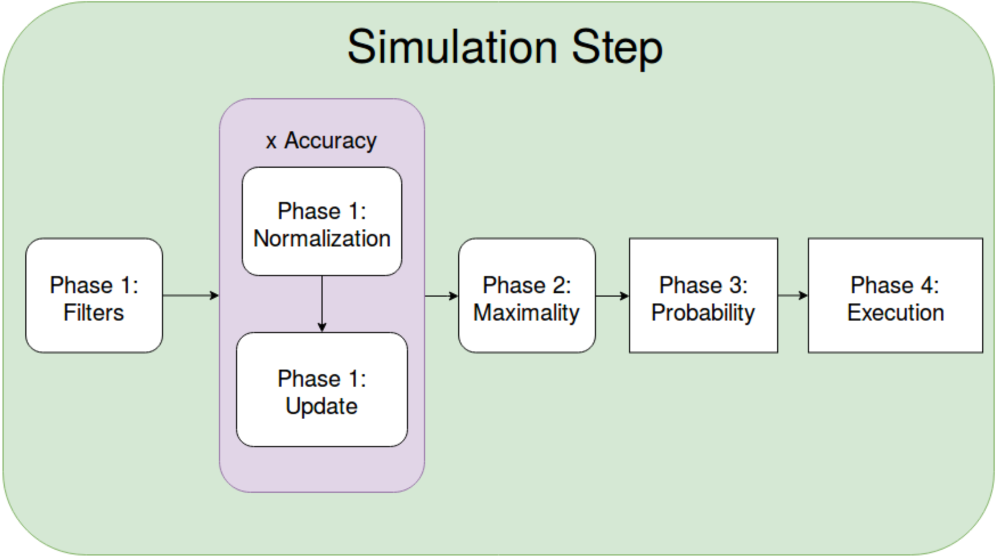
$500 * X \sim B(40, 0.2)$

[1] **Binomial random variate generation**. V. Kachitvichyanukul, B.W. Schmeiser. *Comm. ACM*, 31, 2, 216–222 (1988)

Workflow


Pipeline
P-Lingua File
PLinguaCore Compiler
Binary File
Simulator
CSV File
Simulator
Analyze Parameters
Analyze Binary File
Initialize PDP Structures
Initialize RNG States
Simulation Steps
Write Results

Pipeline


Simulation Step
x Accuracy
Phase 1: Filters
Phase 1: Normalization
Phase 1: Update
Phase 2: Maximality
Phase 3: Probability
Phase 4: Execution

Outline

Let B_i and B_j be two distinct blocks with $LHS(B_i) = (h_j, \alpha_i, u_i, v_i)$ and $LHS(B_j) = (h_j, \alpha_j, u_j, v_j)$:

- **Direct competition**: B_i and B_j directly compete if one of the following holds:
  - if $h_i = h_j \wedge \alpha_i = \alpha_j$, then $v_i \cap v_j \neq \emptyset$.
  - if $parent(h_i) = parent(h_j)$, then $u_i \cap u_j \neq \emptyset$
  - if $parent(h_i) = h_j$, then $u_i \cap v_j \neq \emptyset$.
- **Indirect competition**: B_i and B_j indirectly compete if there exists a distinct block that directly competes with B_i and directly (or indirectly) competes with B_j.
- **Competition**: B_i and B_j compete if and only if directly or indirectly compete.

micro-DCBA

Competition

Let $G = (V, E)$:

- V: rule blocks
- E: blocks that compete directly

Competitive partition = finding connected components.

μ-DCBA (micro-DCBA): running DCBA to each set of the partition. This gives another level of parallelism.

Independent blocks: they do not compete with any other blocks. High level of parallelism.

- By definition, communication rules are independent.

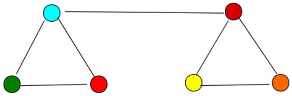
micro-DCBA

Componentes conexas en paralelo

Hooking + Jumping[6]:

1	2	3	4	5	6

1,2	5,6	2,5	1,3	4,5	3,2	4,6

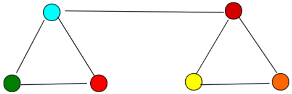


[6] J. Soman, K. Kothapalli y P. J. Narayanan. “Some GPU Algorithms for Graph Connected Components and Spanning Tree”. En: *Parallel Processing Letters* 20 (2010), págs. 325-339.

Hooking + Jumping[6]:

1	2	3	4	5	6

1,2	5,6	2,5	1,3	4,5	3,2	4,6



[6] J. Soman, K. Kothapalli y P. J. Narayanan. “Some GPU Algorithms for Graph Connected Components and Spanning Tree”. En: *Parallel Processing Letters* 20 (2010), págs. 325-339.

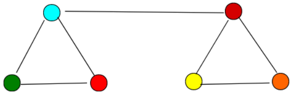
micro-DCBA

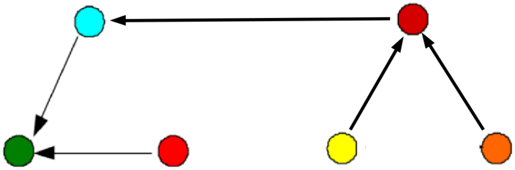
Componentes conexas en paralelo

Hooking + Jumping[6]:

1	2	3	4	5	6

1,2	5,6	2,5	1,3	4,5	3,2	4,6





[6] J. Soman, K. Kothapalli y P. J. Narayanan. “Some GPU Algorithms for Graph Connected Components and Spanning Tree”. En: *Parallel Processing Letters* 20 (2010), págs. 325-339.

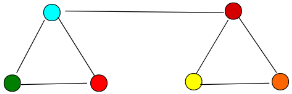
micro-DCBA

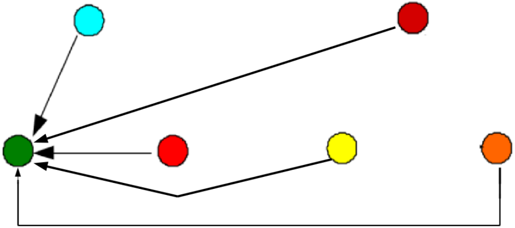
Componentes conexas en paralelo

Hooking + Jumping[6]:

1	2	3	4	5	6

1,2	5,6	2,5	1,3	4,5	3,2	4,6

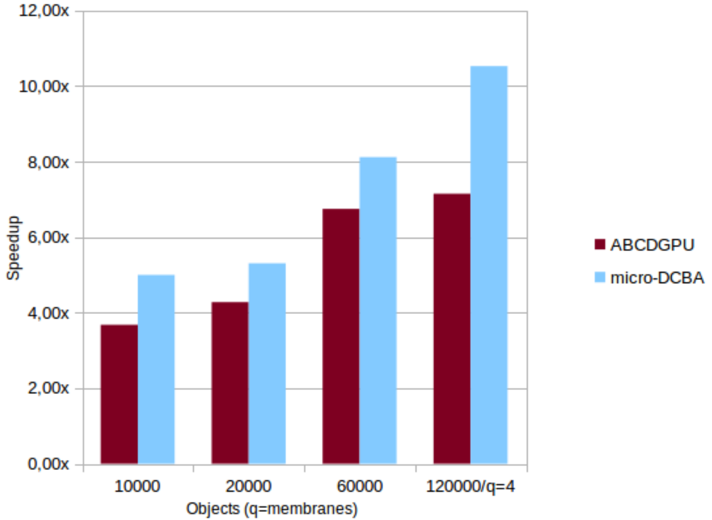




[6] J. Soman, K. Kothapalli y P. J. Narayanan. “Some GPU Algorithms for Graph Connected Components and Spanning Tree”. En: *Parallel Processing Letters* 20 (2010), págs. 325-339.

micro-DCBA

Resultados



Resultados en NVIDIA Tesla K40c

Experimental review

- Example: Bearded Vulture in the Pyrenees model.
- PDP system with 1 environment and 439 rule blocks.
- Results with 100 simulations on an NVIDIA GTX 950M.
  - Initializing states for RNG: 8ms
  - One step fo computation: 352μs
  - A transfer of results: 250μs
  - Writing results in a file: 16ms

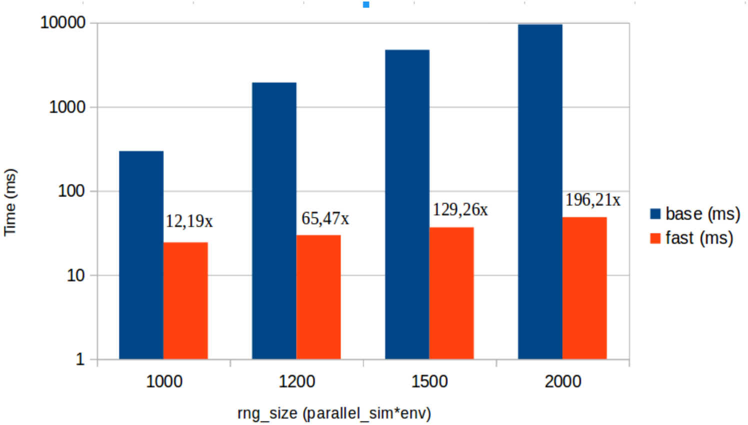
Improvements

- Fast initialization of RNG
- Overlapping sending results and execution.
- Filtering results

Optimizaciones

Inicio rápido RNG

¿Más entornos?



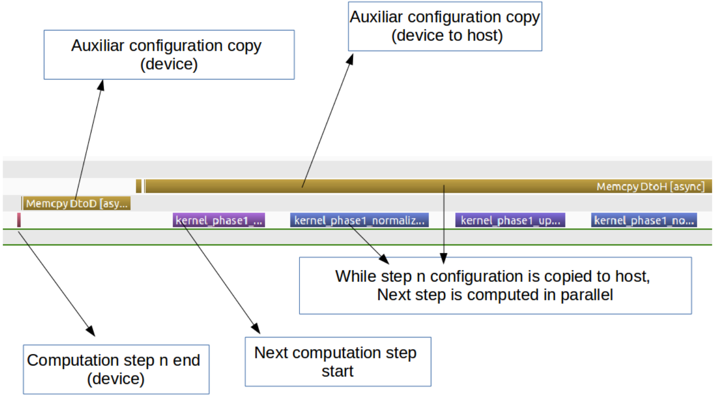
Optimizaciones

Solapamiento envío y ejecución[8]

[8] M. Harris. *How to Overlap Data Transfers in CUDA C/C++*. https://devblogs.nvidia.com/how-overlap-data-transfers-cuda-cc/. [Online; Acceso el 13-5-2018]. 2012.

Optimizaciones

Solapamiento envío y ejecución



Optimizaciones

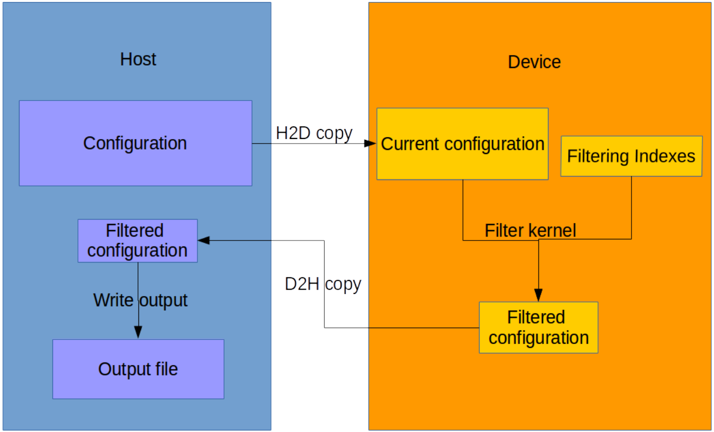
Filtro de resultados

Entrada: Ternas entorno, membrana y objeto

```
0 0 0
1 - 4
- - Z{3,6}
0 1 X{3,1}
```

Optimizaciones

Filtro de resultados



Outline

Work under consideration

- Fully automated Workflow: MeCoSim + P-Lingua5 + ABCDGPU
  - A manual workflow was published in: *L. Valencia-Cabrera, M.Á. Martínez-del-Amor, I. Pérez-Hurtado. A Simulation Workflow for Membrane Computing: From MeCoSim to PMCGPU Through P-Lingua. Enjoying Natural Computing, Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday, Lecture Notes in Computer Science, 11270 (2018), 291-303.*
- Simulation in the cloud (SaaS)
- Auto-generation of CUDA code from P-Lingua 5

Thank you for your attention

Q&A&S?