Simulation of Computing P Systems: A GPU Design for the Factorization Problem

Miguel Á. Martínez-del-Amor, David Orellana-Martín
Ignacio Pérez-Hurtado, Luis Valencia-Cabrera
Agustín Riscos-Núñez, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Dept. Computer Science and Artificial Intelligence
Universidad de Sevilla

CMC19, 4-7 September 2018, Dresden (Germany)

Contents

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).

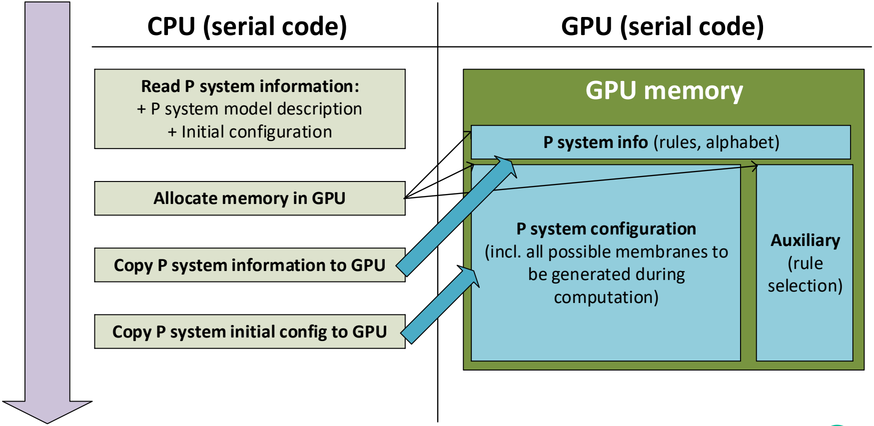
[1] W.-M. Hwu, D. Kirk. Programming massively parallel processors, Morgan Kaufmann, 2010.

Why is the GPU interesting for simulating P systems?

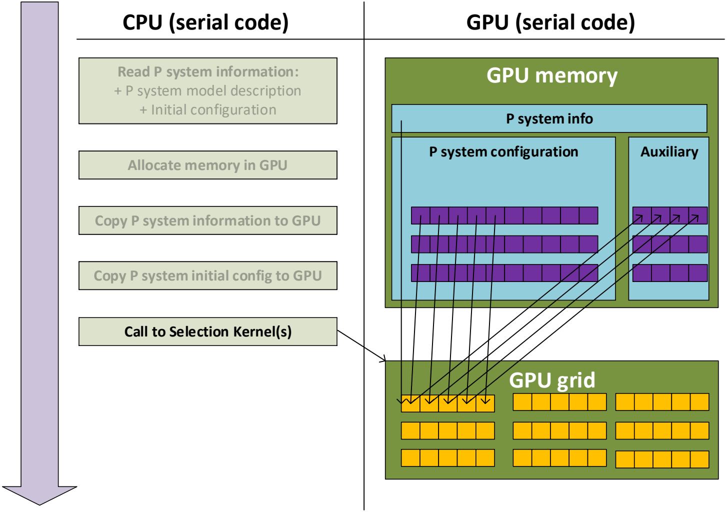
- Desired properties:
  - High level of parallelism *(up to 4000 cores)*
  - Shared memory system *(easily synchronized)*
  - Scalability and *portability*
  - Known languages: C/C++, Python, Fortran...
  - Cheap technology everywhere *(cost and maintenance)*
- Undesired properties:
  - Best performance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

Outline

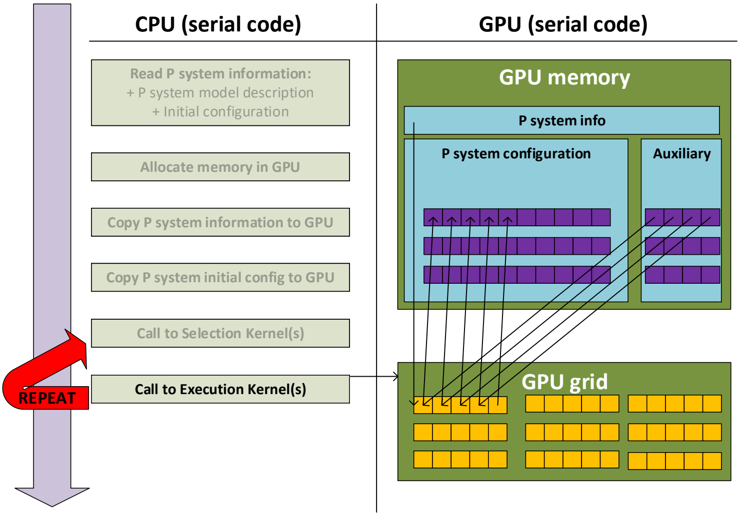
GPU simulator workflow - Initialization (I)



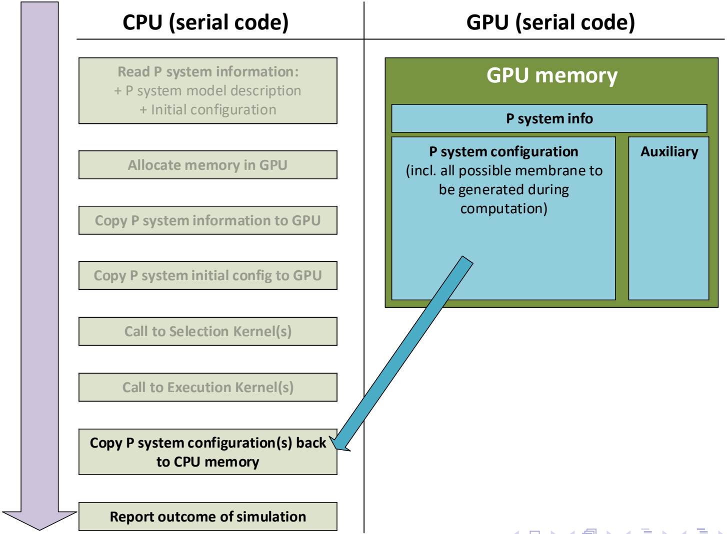
GPU simulator workflow - Simulation - Selection (II)



GPU simulator workflow - Simulation - Execution (III)



GPU simulator workflow - Wrap up (IV)



Outline

Simulation approaches

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

Simulating models (“generic” approach)

P systems with active membranes

- Rooted tree of membranes.
- Polarization and no cooperation (only one object in LHS).
- Rules: Evolution, send-in, send-out, division and dissolution.
- Assumptions to simplify the simulator:
  - Confluent models
  - Only two-level trees (skin and elementary membranes)

Simulating models ("generic" approach)

Mapping double parallelism:

- **Membranes to Thread Blocks**
- **Objects to Threads**: thanks to no-cooperative rules, it is enough to check the existence of one object to trigger a rule.

Simulating models ("generic" approach)

Performance analysis

- Two benchmarks (on a C1060 with 240 cores):
  A. A simple test P system[2]
     - Max speedup: 5.8x
  B. An efficient solution to SAT
     - Max speedup: 1.5x ($n = 18$, 2^{18} membranes)
- Density of objects per membrane: $\frac{Reality}{WorstCase} = \frac{\#Objects}{AlphabetSize}$
  - Test A: 100%
  - Test B: $\sim$ 15%

[2] *One division rule:* $[d]_2 \rightarrow [d]_2\ [d]_2$, *Many evolution rules:* $[o_i \rightarrow o_i]_2,\ 0 \leq i \leq N$

Simulating models ("generic" approach)

Performance analysis

- Two benchmarks (on a C1060 with 240 cores):
  A. A simple test P system[2]
     - Max speedup: 5.8x
  B. An efficient solution to SAT
     - Max speedup: 1.5x ($n = 18$, 2^{18} membranes)
- Density of objects per membrane: $\frac{Reality}{WorstCase} = \frac{\#Objects}{AlphabetSize}$
  - Test A: 100%
  - Test B: $\sim$ 15%

[2] *One division rule:* $[d]_2 \rightarrow [d]_2\ [d]_2$, *Many evolution rules:* $[o_i \rightarrow o_i]_2,\ 0 \leq i \leq N$

Simulating models ("generic" approach)

Foreseen performance by Sevilla Carpets: D. Orellana-Martín et al. Sevilla Carpets revisited: Enriching the Membrane Computing toolbox. Fundamenta Informaticae, 134 (2014), 153-166.

The flatter the carpet, the higher the parallel degree in the system (and so, in the simulation).

Simulating models (“specific” approach)

Cell-like solution to SAT

- P systems with active membranes
- A specific linear time solution to SAT, with exponential workspace
- Encoding:
  - Objects: literals of the formula and auxiliary (counters, etc.)
  - Membranes: truth assignments
- A 4-staged solution:
  1. Generation
  2. Synchronization
  3. Check out
  4. Output

Simulating models ("specific" approach)

Cell-like solution to SAT - parallel design

- **Membranes to Thread Blocks**
- **Objects in initial multiset to Threads**: *we have constrained the number of threads to the amount of different objects in the initial multiset.*

Simulating models (“specific” approach)

Tissue-like solution to SAT

- Tissue P systems with cell division
  - Directed graph of cells.
  - No polarization and cooperation (multisets in LHS)
  - Communication (symport/antiport) and division rules.
  - Active environment.
- A specific linear time solution to SAT, with exponential workspace
- Encoding:
  - Objects: literals of the formula and auxiliary (counters, etc.)
  - Cell: truth assignment
- A 5-staged solution:
  1. Generation
  2. Exchange
  3. Synchronization
  4. Checking
  5. Output

Simulating models ("specific" approach)

Tissue-like solution to SAT

- Tissue P systems with cell division
  - Directed graph of cells.
  - No polarization and cooperation (multisets in LHS)
  - Communication (symport/antiport) and division rules.
  - Active environment.
- A specific linear time solution to SAT, with exponential workspace
- Encoding:
  - Objects: literals of the formula and auxiliary (counters, etc.)
  - Cell: truth assignment
- A 5-staged solution:
  1. Generation
  2. Exchange
  3. Synchronization
  4. Checking
  5. Output

Simulating models ("specific" approach)

Tissue-like solution to SAT - parallel design

- **Cells to Thread Blocks**
- **Objects in initial multiset, objects for truth assignation, and auxiliary objects to Threads**: selection of rules is not direct given that there is cooperation.

Simulating models ("specific" approach)

Performance analysis

- Cell-like approach:
  - Max speedup: 63x ($n = 21$)
- Tissue-like approach:
  - Max speedup: 10x ($n = 21$)

- Conclusion:
  - Charges save space, and help to increase object density
  - No-cooperation avoids synchronization issues
  - Shallow P systems (no more than skin and elementary membranes)

Simulating models ("specific" approach)

Performance analysis

- Cell-like approach:
  - Max speedup: 63x ($n = 21$)
- Tissue-like approach:
  - Max speedup: 10x ($n = 21$)

- Conclusion:
  - Charges save space, and help to increase object density
  - No-cooperation avoids synchronization issues
  - Shallow P systems (no more than skin and elementary membranes)

Outline

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.

Contents

Case study: FACTORIZATION problem

- *Given a natural number which is the product of two prime numbers, find its decomposition*.
- Partial function FACT from N to N^2: $FACT(x) = (y, z)$
- Solution presented in WMC/UCNC 2018:
  - a family $\{\Pi(n) | n \in N\}$ of (binary) computing polarizationless P systems with active membranes
  - makes use of minimal cooperation and minimal production
  - no dissolution rules
  - no division rules for non-elementary membranes

Case study: FACTORIZATION problem

Stages in the computation of $\Pi(n)$:

1. Generation
2. Multiplication
3. Equality checking
4. Trivial solution check
5. First delete
6. Second delete
7. Output 1
8. Output 2

Case study: FACTORIZATION problem

More features:

- After generation, $\Pi(n)$ contains 2^{2n+2}, where $n = k_x$, being x an instance of the problem (x *has* $n+1$ *digits in its binary representation*).
- The computation takes at most $19n+28$ steps.
- Some rules:

$$
\begin{array}{l}
[\, a_j\, \overline{X}_j \rightarrow X_j \,] \ \text{, for } 0 \leq j \leq n \\
[\, \alpha_{1,j,s} \rightarrow \alpha_{1,j,s+1} \,]_2 \ \text{, for } 0 \leq j \leq n \text{ and } 0 \leq s < j \\
[\, \alpha_{2,j,s} \rightarrow \alpha_{2,j,s+1} \,]_2 \ \text{, for } 0 \leq j \leq n \text{ and } 0 \leq s < n+1+j \\
\left.\begin{array}{l} [\, \beta_{1,j,k} \rightarrow \beta_{1,j,k+1} \,]_2 \\ [\, \beta_{2,j,k} \rightarrow \beta_{2,j,k+1} \,]_2 \end{array}\right\} \text{for } 0 \leq j \leq n, 0 \leq k \leq 3n+5 \\
[\, \overline{P}_{j,k} \rightarrow \overline{P}_{j,k+1} \,]_2 \ \text{, for } 0 \leq j \leq 2n+1, 0 \leq k \leq 5n+7 \\
[\, p_{i,j} \rightarrow p_{i,j+1} \,]_2 \ \text{, for } 0 \leq i \leq n, 0 \leq j \leq 5n+9 \\
[\, \tau_{j,k} \rightarrow \tau_{j,k+1} \,]_2 \ \text{, for } 0 \leq j \leq n, 0 \leq k \leq 14n+11 \\
[\, \omega_{j,k} \rightarrow \omega_{j,k+1} \,]_2 \ \text{, for } 0 \leq j \leq n, 0 \leq k \leq 15n+21 \\
[\, \omega_{i,j} \rightarrow \omega_{i,j+1} \,]_1 \ \text{, for } 0 \leq i \leq n, 0 \leq j \leq 17n+25
\end{array}
$$

Design of specific simulators

Considerations:

- No need to do selection - execution (we know the rules)
- No need to do per-transition simulation (we can take short paths)
- No need to store rules in memory (we know the rules!!)
- Under control: *we should be able to know, looking into the simulator, the state of the P system at every transition*

Common designs:

- Models are normally designed by staged computations
- Each one with different behaviour (generation, checking, ...)
- A kernel per stage

Design of specific simulators

Considerations:

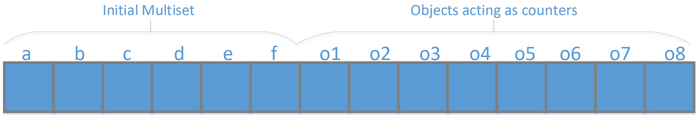
- No need to do selection - execution (we know the rules)
- No need to do per-transition simulation (we can take short paths)
- No need to store rules in memory (we know the rules!!)
- Under control: *we should be able to know, looking into the simulator, the state of the P system at every transition*

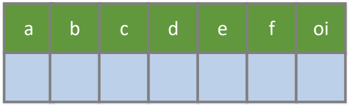
Common designs:

- Models are normally designed by staged computations
- Each one with different behaviour (generation, checking, ...)
- A kernel per stage

Design of specific simulators

Increasing density of objects: from sparse to dense representation





Design of specific simulators

Design decisions:

- Objects acting as counters, variable or in memory?
- Are we able to set an upper-bound of objects appearing in membranes? Minimal production helps!!
- Do we know the maximum amount of membranes?
- A kernel or several kernels per stage?
- Fusing kernels for simple stages?

Outline

GPU-oriented P systems

- MABICAP: Bio-inspired machines over high performance platforms.
- Seeking P system models well-suited for GPU deployments.
- Selection of best ingredients, while keeping computing power and expressibility:
  - Charges
  - Minimal production
  - Minimal (almost no-) cooperation
  - Shallow structure (horizontal parallelism)
- Towards efficient simulation of Spiking Neural P systems (sparse representation)

New GPU features

- Kernel compilation in runtime (customizable to the model)
- Cooperative Groups (for deeper P systems)
- Tensor cores (matrix representations for SNP systems)
- Dynamic Parallelism (to be seen...)
- Faster memory
- Cloud

Coming Calls

BICAS 2019: Biologically Inspired Parallel and Distributed Computing, Algorithms and Solutions

- Part of HPCS 2019 (a CORE B conference)
- Dublin (Ireland), July 15 – 19, 2019
- Deadline: TBD (around March)
- Proceedings in IEEE Xplore
- Special issues in ISI journals (FGCS, CCPE, ...)
- `http://hpcs2019.cisedu.info`

Coming Calls

BWMC 2019: Brainstorming Week on Membrane Computing

- Sevilla (Spain)
- Dates: **5-8 February**
- Announcements at RGNC website: `http://www.gcn.us.es`

Thank you for your attention!

Vielen Dank für Ihre Beachtung!

The authors acknowledge support of the R&D project MABICAP TIN2017-89842-P and REDBIOCOM TIN2015-71562-REDT, funded by the Spanish government and EU FEDER funds.