SynCoP 2015

April 11th, 2015
London, UK

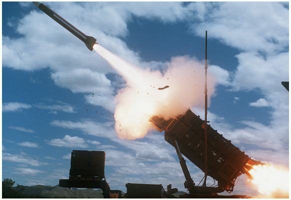
Enhanced Distributed Behavioral Cartography of Parametric Timed Automata

Étienne André, Camille Coti, Hoang Gia Nguyen

LIPN, Université Paris 13, Sorbonne Paris Cité, CNRS, France

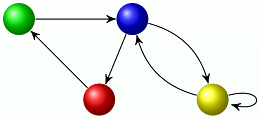
Context: Formal Verification of Timed Systems (1/3)

- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - $\rightsquigarrow$ Need for a thorough modeling and verification phase



Context: Formal Verification of Timed Systems (2/3)

- Use formal methods



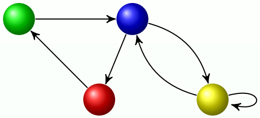
A model of the system

$AG\neg$

A property to be satisfied

Context: Formal Verification of Timed Systems (2/3)

- Use formal methods



$\overset{?}{\models}$

$AG\neg$

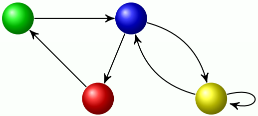
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Context: Formal Verification of Timed Systems (2/3)

- Use formal methods



$\overset{?}{\models}$

AG¬ 

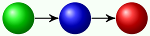
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

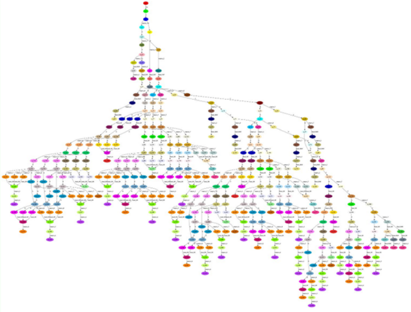
No



Counterexample

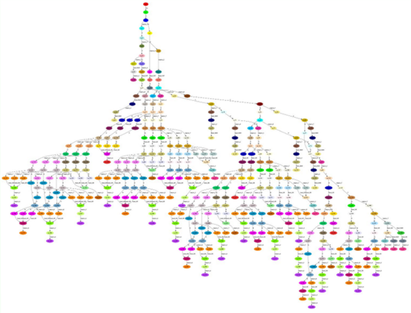
Context: Formal Verification of Timed Systems (3/3)

- Problem: But state space explosion is always painful! Especially real-time systems.



Context: Formal Verification of Timed Systems (3/3)

- Problem: But state space explosion is always painful! Especially real-time systems.



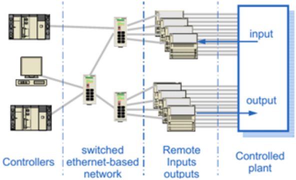
- One solution:
  - Extend to distributed fashion

Outline

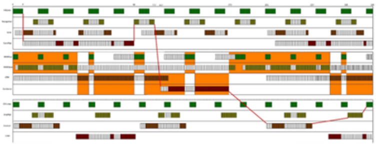
Outline

Parametric Timed Automata (PTA)

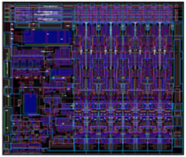
A formalism to model and verify concurrent real-time systems [Alur et al., 1993]



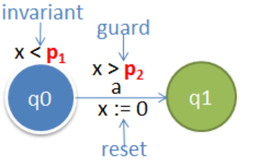
Communication protocols



Processor Scheduling



Asynchronous Circuits

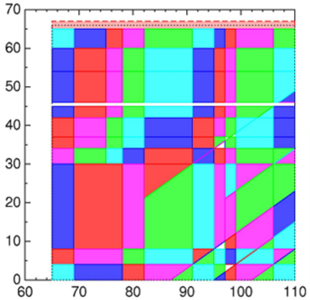


Parametric Timed Automata

x: Clock

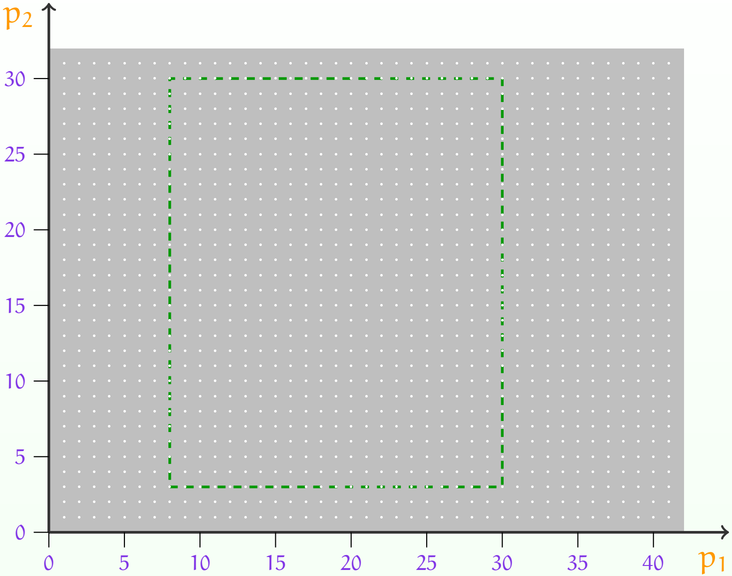
p: Parameters allow to represent unknown values (e.g., a transmission delay or a timeout)

Behavioral Cartography (BC)

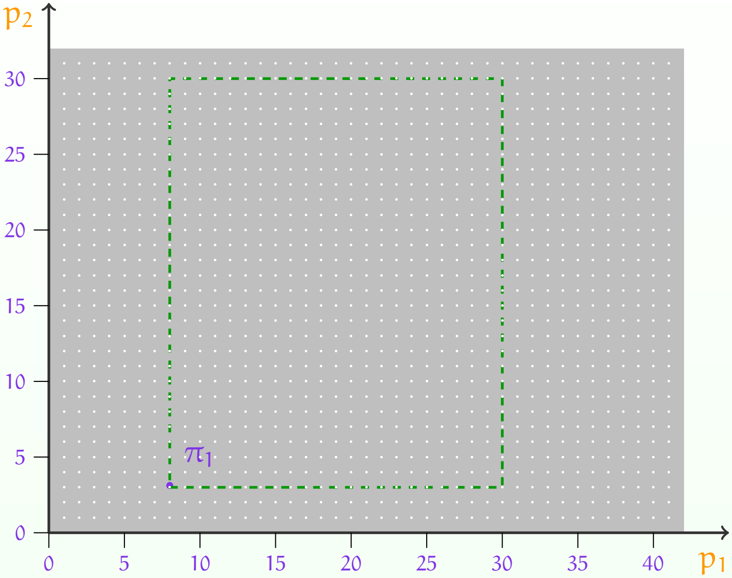


- BC: Partitions a parameter domain into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- Method: enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- All parameter valuations in a tile have the same possible behaviors (same "trace set"), and verify the same linear-time properties

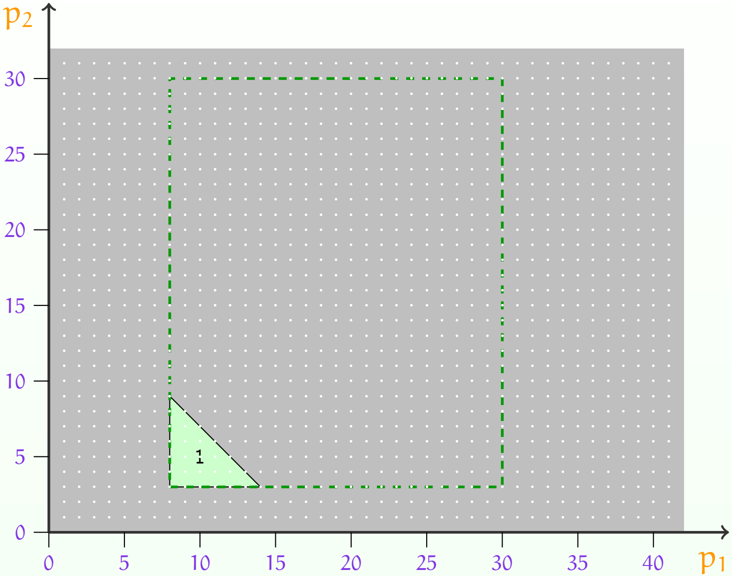
Behavioral Cartography: Example



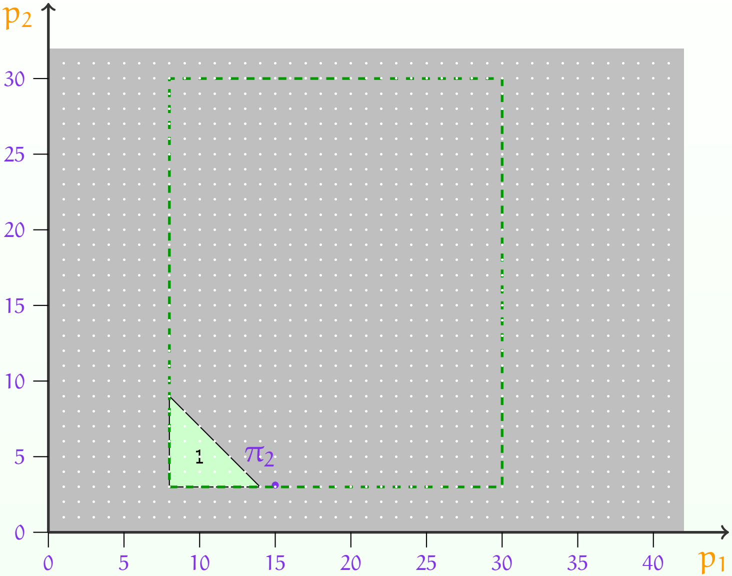
Behavioral Cartography: Example



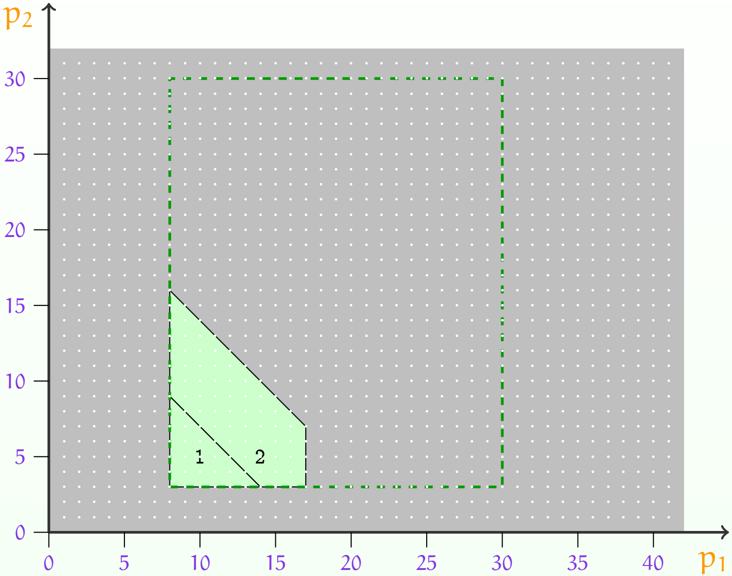
Behavioral Cartography: Example



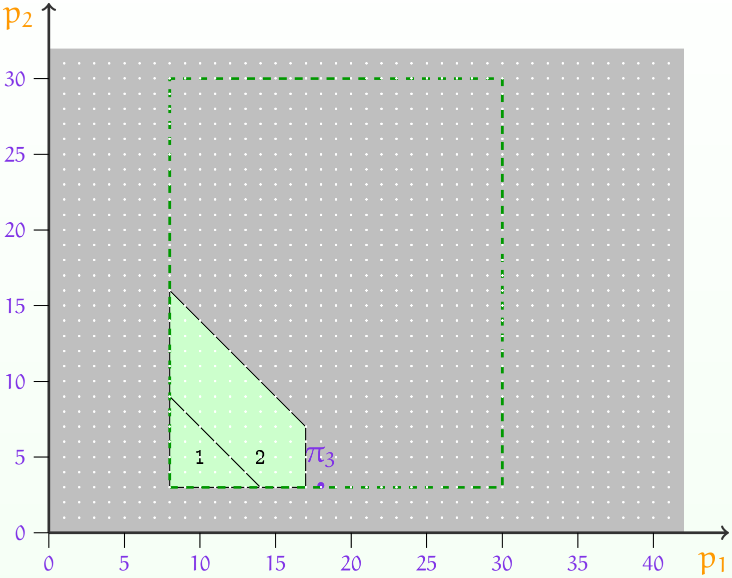
Behavioral Cartography: Example



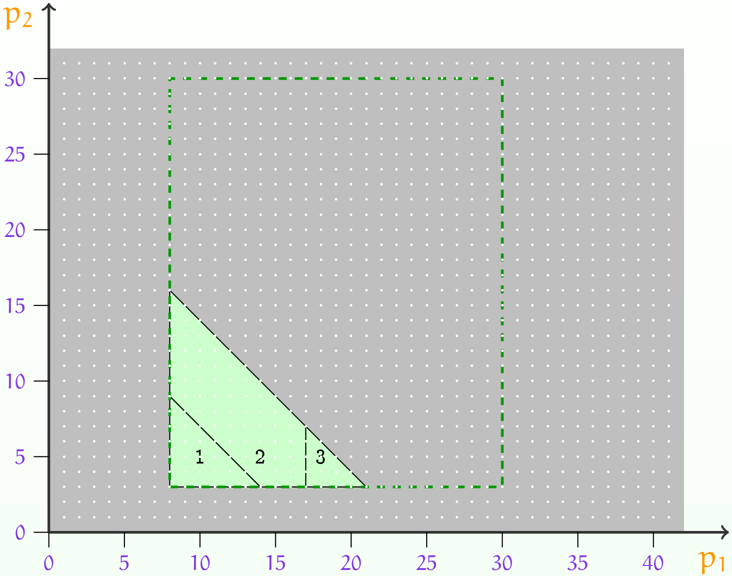
Behavioral Cartography: Example



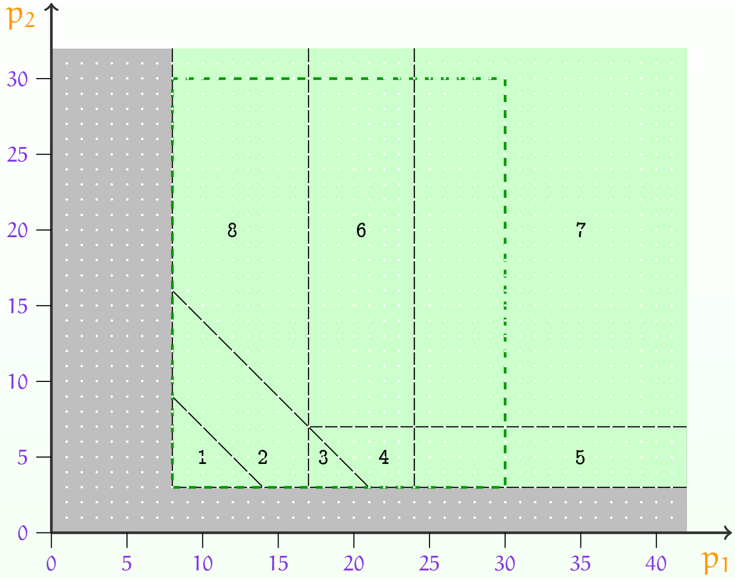
Behavioral Cartography: Example



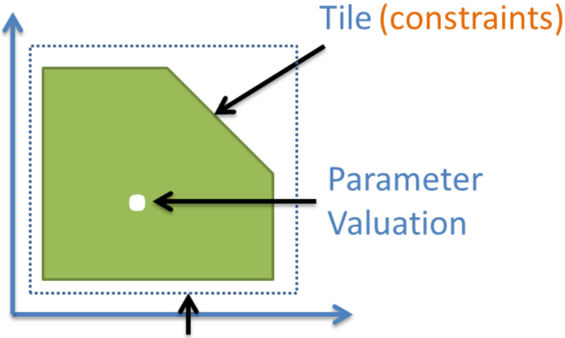
Behavioral Cartography: Example



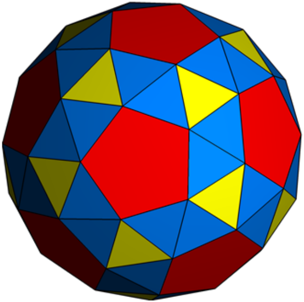
Behavioral Cartography: Example



An n-dimension analysis



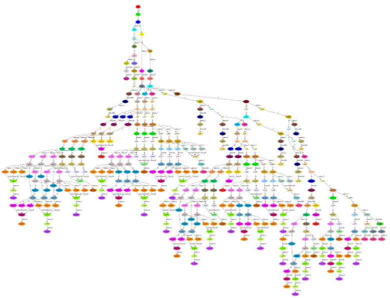
Tile
(2 dimensions)



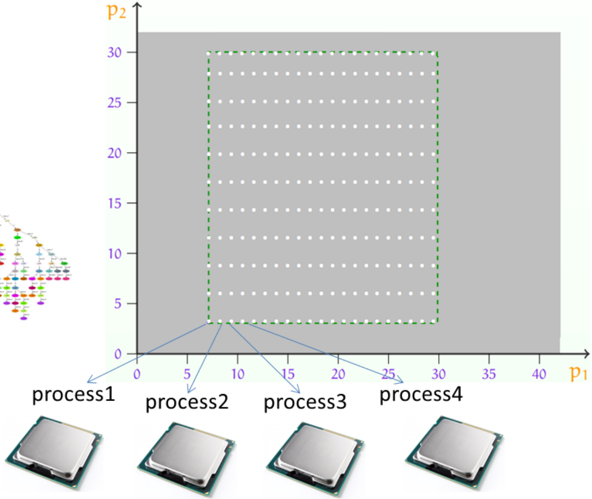
Tile
- Polyhedron
(n dimensions)

Outline

Distributing BC

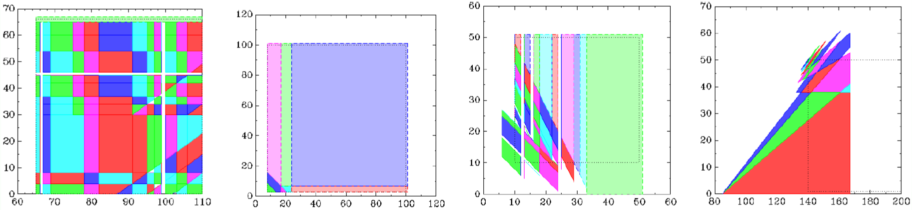


Example of a trace set output by IMITATOR



- Problem: BC is very slow! (up to several hours)
- Goal: distribute BC on a cluster to increase the computation speed

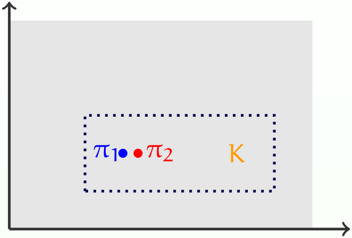
Distributing BC: Problem 1



Problem 1: the general shape of the Cartography is unknown in general

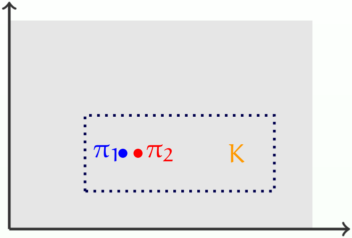
And the time to compute each tile varies a lot (more or less complex trace sets)

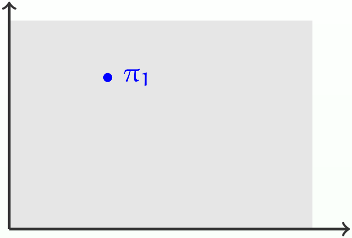
Distributing BC: Problems 2 and 3



Problem 2: two close points will very probably yield the same tile (loss of efficiency)

Distributing BC: Problems 2 and 3

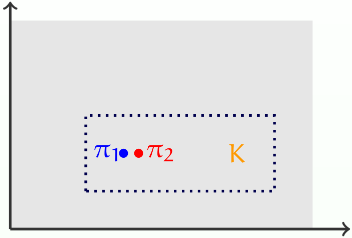


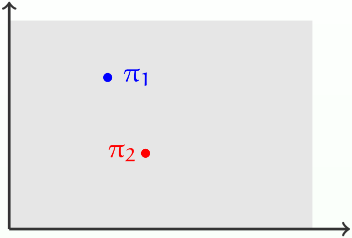


Problem 2: two close points will very probably yield the same tile (loss of efficiency)

Problem 3: Should we stop a process when its reference point ("π_2") was covered by another tile ("K_1") ?

Distributing BC: Problems 2 and 3

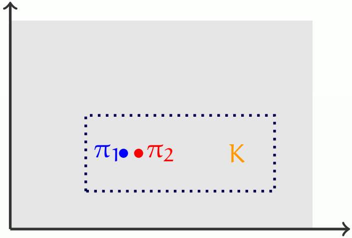


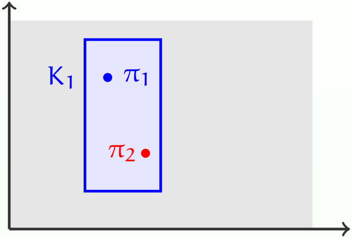


Problem 2: two close points will very probably yield the same tile (loss of efficiency)

Problem 3: Should we stop a process when its reference point ("π_2") was covered by another tile ("K_1") ?

Distributing BC: Problems 2 and 3

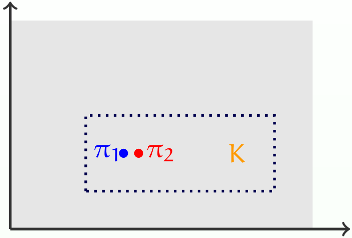


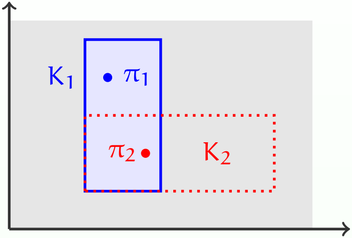


Problem 2: two close points will very probably yield the same tile (loss of efficiency)

Problem 3: Should we stop a process when its reference point ("π_2") was covered by another tile ("K_1") ?

Distributing BC: Problems 2 and 3



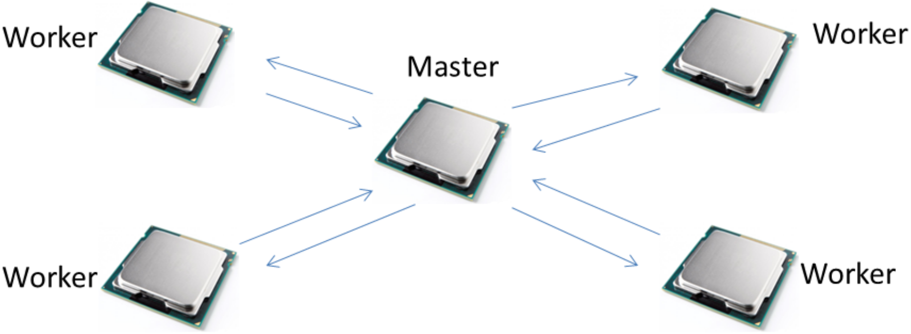


Problem 2: two close points will very probably yield the same tile (loss of efficiency)

Problem 3: Should we stop a process when its reference point ("π_2") was covered by another tile ("K_1") ?

Outline

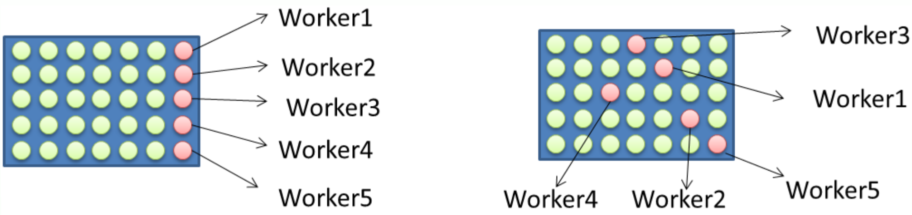
Master Workers Scheme



Traditional Master-Worker communication scheme: [André, Coti, Evangelista, 2014]

- Workers: ask the master for a point, and send the result ("tiles") to the master
- Master: is responsible for smart repartition of data between the workers

Previous Point-based BC Algorithms



Choosing points sequentially   Choosing points randomly

Point-based BC algorithms:

- Sequential: each point is sent to a worker sequentially
- Random: points selected randomly, then switches to Sequential
- Shuffle: similar to the Sequential, but the difference is that master must statically compute the list of all points, then shuffle all points, then store them in array (new)

Outline

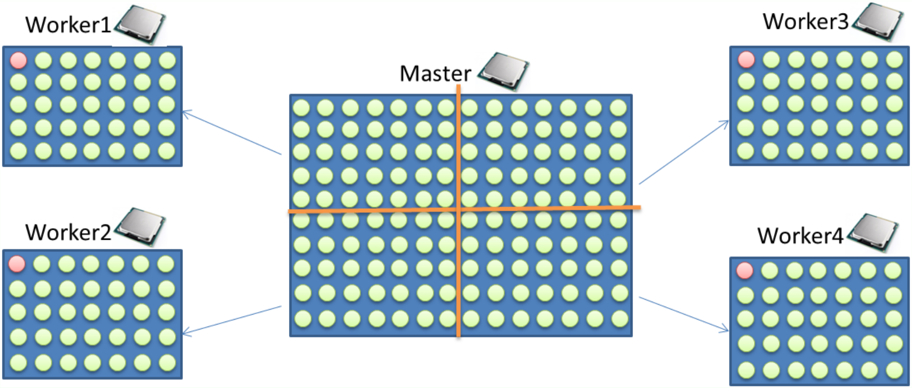
Subpart-based BC Algorithm Scheme

"Domain decomposition" scheme

- Master
  1. initially splits the parameter domain into subparts and send them to the workers
     - Subpart: a subdomain of the parameter domain
  2. when a worker has completed its subpart, the master splits another subpart, and sends it to the idle worker

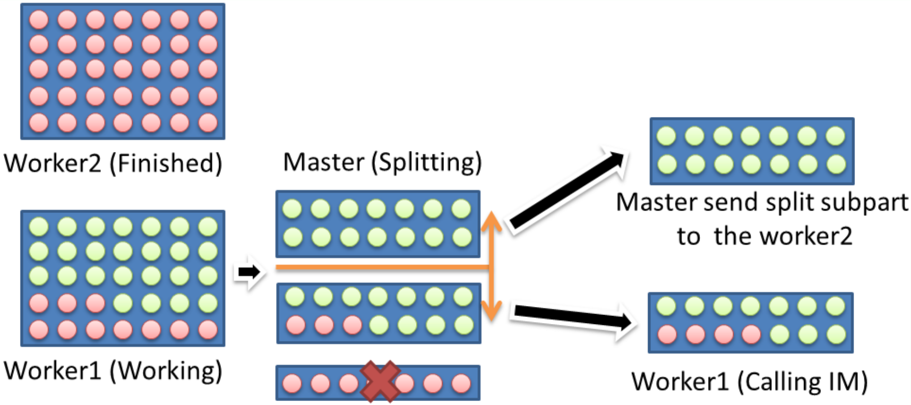
- Workers
  1. receives the subpart from the master
  2. calls IM on the points of this subpart
  3. sends the results (tiles) back to the master
  4. asks for more work

Subpart-based Distribution Scheme: Initial Splitting



- Solved Problem 2! (prevent to choose close points)
- Prevent bottleneck phenomenon at the master side
  - Master only responsible for gathering tiles and splitting subparts

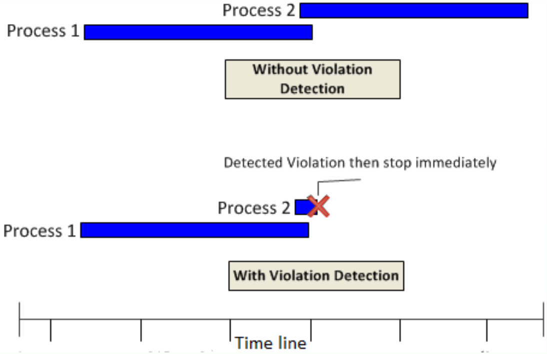
Subpart-based Distribution Scheme: Dynamic Splitting

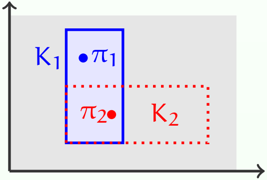


- Master can balance workload between workers

Violation Detection – Heuristic (1/2)

Violation detection: a mechanism to detect and stop process which is calling point in the covered tile.





Violation Detection – Heuristic (2/2)

- Solution proposed: stop immediately when the reference point ("π_2") is covered by another tile ("C_1")
- Workers have ability to self-detects violation
- Is an answer to the previous Problem 3 ("what to do when a point is covered by another tile?")
- Can be used for all previous algorithms

Outline

Implementation in IMITATOR

- IMITATOR [André, Fribourg, Kühne, Soulat, 2012]
  - $26,000$ lines of OCaml code
    - Including $> 3,000$ lines for the distribution algorithms
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license at `www.imitator.fr`
  - Stable version (2.6.2) integrated in *CosyVerif* [AHHKLLP13]
- Distributed version of IMITATOR relying on MPI
  - Using the OcamlMPI library for passing messages between Master and Workers

Implementation in IMITATOR

- IMITATOR [André, Fribourg, Kühne, Soulat, 2012]
  - $26,000$ lines of OCaml code
    - Including $> 3,000$ lines for the distribution algorithms
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license at `www.imitator.fr`
  - Stable version (2.6.2) integrated in *CosyVerif* [AHHKLLP13]

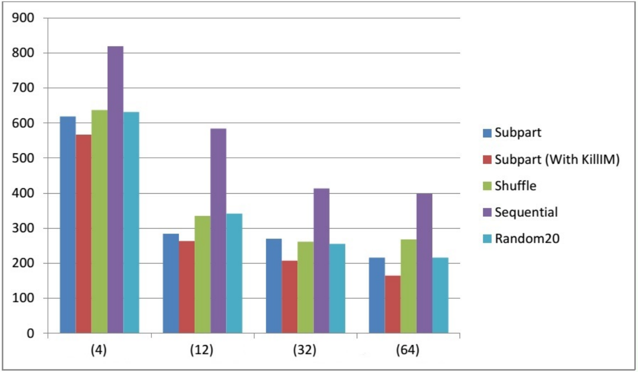
- Distributed version of IMITATOR relying on MPI
  - Using the OcamlMPI library for passing messages between Master and Workers

    ... in which we found a bug!

Experimental Validation

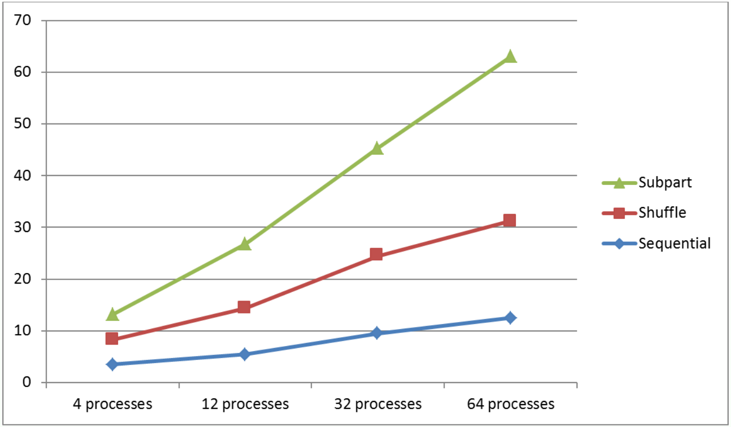
Experimental conducted on a real cluster ("Magi") in the Paris 13 University

Average computation time for a set of case studies, for 4/12/32/64 nodes:



Our new algorithm always outperforms existing algorithms

SpeedUp Chart Diagram

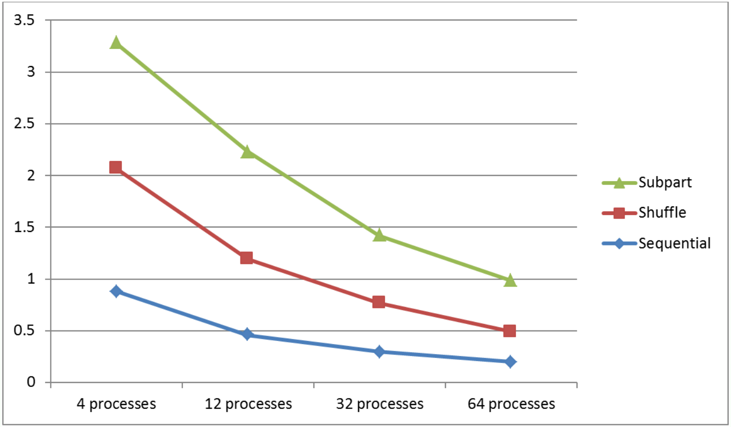


From Amdahl's law, we have $SpeedUp = \frac{T_s}{T_d}$ (Higher is better)

T_s: is run time with single process (sequential)

T_d: is run time with multi-processes (distributed)

Efficiency Chart Diagram



$Efficiency = \frac{T_s}{N \times T_d} = \frac{SpeedUp}{N}$ (Higher is better)

N: is number of processes ("nodes")

All algorithms decrease while number of processes increase

=> loss of efficiency

Outline

Conclusion and Perspectives

- Conclusion:
  - Proposed a new efficient distributed algorithm (Subpart) for Behavioral Cartography
  - Proposed a new heuristic approach improving all BC distribution algorithms
  - Proposed solutions to our three problems
  - Implemented the new algorithm in IMITATOR

- Future works:
  - We will attempt to achieve a more efficient algorithm
  - Design an autonomous distribution scheme for BC
  - Improve heuristics
  - Try BC in GPU's or CPU+GPU's environment
  - ... and prove the deadlock-freeness of our master-worker communication scheme!

Bibliography

References I

Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993).
Parametric real-time reasoning.
In *STOC*, pages 592–601. ACM.

André, É., Coti, C., and Evangelista, S. (2014).
Distributed behavioral cartography of timed automata.
In Dongarra, J., Ishikawa, Y., and Atsushi, H., editors, *21st European MPI Users' Group Meeting (EuroMPI/ASIA'14)*, pages 109–114. ACM.

André, É. and Fribourg, L. (2010).
Behavioral cartography of timed automata.
In *RP*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90. Springer.

André, É., Fribourg, L., Kühne, U., and Soulat, R. (2012).
IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.
In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer.

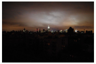
References II

André, É., Hillah, L.-M., Hulin-Hubard, F., Kordon, F., Lembachar, Y., Linard, A., and Petrucci, L. (2013).
CosyVerif: An open source extensible verification environment.
In Liu, Y. and Martin, A., editors, *18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'13)*, pages 33–36. IEEE Computer Society.

Bagnara, R., Hill, P. M., and Zaffanella, E. (2008).
The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21.

Additional explanation

Explanation for the 4 pictures in the beginning



Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

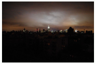
Error screen on the earliest versions of Macintosh

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

Licensing

Source of the graphics used I



Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG
License: CC BY 3.0

Title: Sad mac
Author: Przemub
Source: https://commons.wikimedia.org/wiki/File:Sad_mac.png
License: Public domain

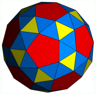
Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0

Source of the graphics used II

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain



Title: Polyhedron
Author: Robert Webb
Source: http://commons.wikimedia.org/wiki/File:Uniform_polyhedron-53-s012.png
License: public domain

Source of the graphics used III

Title: MPI logo
Author: Unknown
Source: http://www.open-mpi.org
License: Unknown

Title: Ocaml logo
Author: Amir Chaudhry
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: CC BY-SA 4.0

Title: IMITATOR logo (Typing Monkey)
Author: Kater Begemot
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: CC BY-SA 3.0

License of this document

Authors: Nguyen Hoang Gia and Étienne André