# Parallel and Distributed Model Checking

**Hubert Garavel**

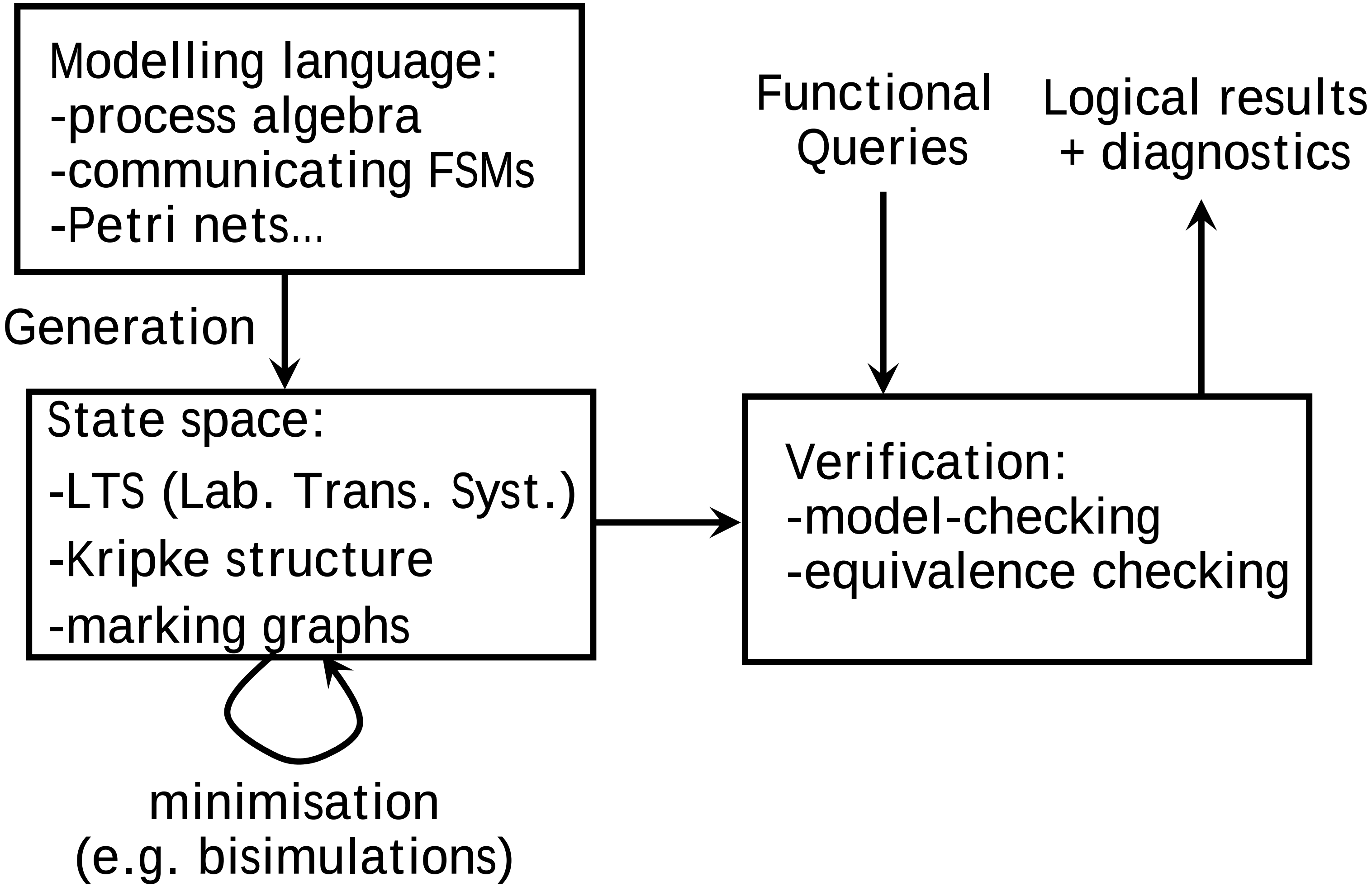
***VASY team***

***INRIA Rhône-Alpes***

*655, avenue de l'Europe*
*38330 Montbonnot Saint Martin*
*France*

# Motivations

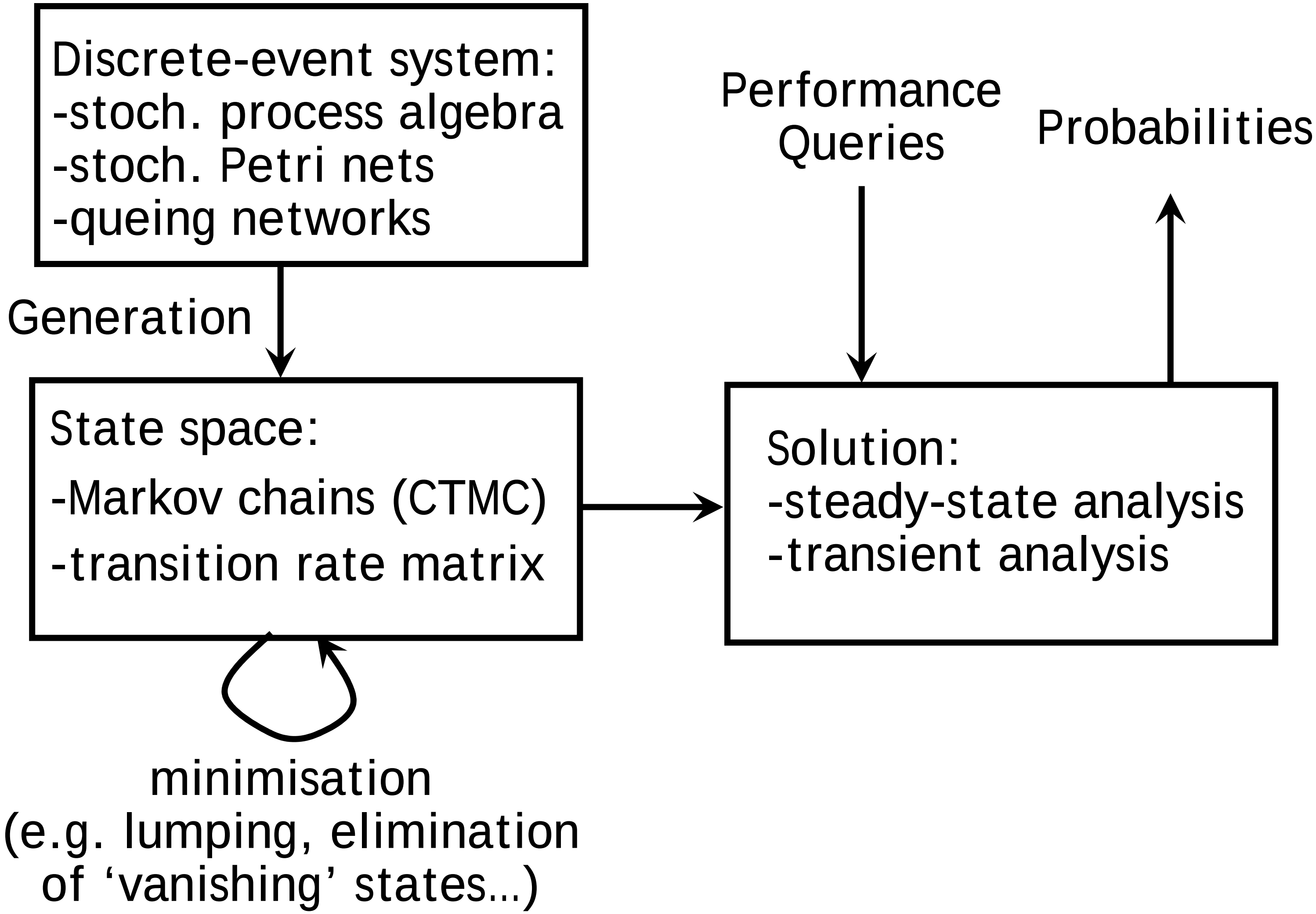
# Functional verification



# Verification state spaces

- State/transition models
- LTS ("black box" vision)
  - no information in states (except initial state)
  - all information on transitions
  - process algebras, bisimulation, branching-time logics (modal mu calculus, ACTL), conformance testing
  - often 'explicit state' model-checking (CWB, CADP, mCRL)
- Kripke structures ("white/grey" box vision)
  - all information in states
  - no information on transitions
  - linear-time logics
  - often 'symbolic' model-checking: set of states (often represented using BDDs, MTDDs...) but also explicit-state approaches
- Timed transition systems *(not covered in this lecture)*

# Performance evaluation



# Performance state spaces

- Continuous-Time Markov Chains (CTMC)
  - no information attached to states
  - stochastic information *rate* (s, s') > *0* attached to each transition s --> s'

- Transition rate matrix

  R [s, s'] = **if** exists s -->s' **then** *rate* (s, s') **else** 0

- Matrix R is often large, sparse (and stiff)

# State space exploration

- Various traversals
  - Breadth-first search (BFS): exhaustive construction, reachability analysis, shortest path, ...
  - Depth-first search (DFS): cycle detection...
  - Synchronous product with an observer or a formula
- Exploration requires a lot of memory
  - Avoid cycles => store visited states
  - BFS requires a FIFO queue
  - DFS requires a stack
  - More (e.g. state table) is often needed to avoid recomputations

# State space EXPLOSION

- The size of state/transition model often grows exponentially in the size of the problem
- Exploration is limited by the physical and virtual memory
- Two problems:
  - state space does not fit into memory
  - state space fits in memory, but is too large for being explored entirely

    (e.g., access to hash table becomes slower as the number of states grows)

# Fighting state explosion

- Two approaches
  - ‘Clever’ methods
  - ‘Brute-force’ methods

# ‘Clever’ methods (1)

- Design ‘better’ modelling languages
  - small languages
  - formal semantics
  - built-in abstractions
  - compositionality properties
- Examples
  - process algebras
  - synchronous languages
  - new generation languages: E-LOTOS [ISO 15437]
- Counter-examples(!)
  - C, C++, Java, SDL, UML/RT, etc.

# ‘Clever’ methods (2)

- Invent better verification algorithms
  - Operate on higher-level models
    - Abstractions, hiding
    - Data flow analysis, static analysis
    - Reductions, property preserving transformations
  - Exploit structure information
    - Hierarchical and compositional verification
    - ‘Symbolic’ models (decision diagrams, Kronecker algebras)
  - Avoid redundancies
    - Partial orders / stubborn sets
    - Symmetries
  - Use locality
    - Caching, bounded-memory algorithms

# ‘Brute force’ methods

- Forget about your PC or workstation
- Use a more powerful machine
  - Increase memory and processing power to handle larger state spaces
  - Use a ‘supercomputer’
- Use N machines instead of one
  - Combine the resources of several machines
  - Ideally, N machines => problems N times larger

# A note about ethics

- Are brute force methods ‘moral’?

- The answer is: Yes!
  - Brute-force is the essence of model-checking
  - Orthogonal to ‘clever’ methods

- Chess programs combine brute force and clever strategies

# Bad news #1

- Brute-force methods will never work
- An exponentially growing problem is attacked by increasing the resources at most linearly!
- That is the fate of model-checking
- In the future, will machine capabilities grow faster than problem complexity?

# Lecture overview

# State of the art in parallel and distributed approaches

- Recent work
  - first paper in 1987
  - but many significant works > 1995
- Many different approaches
  - different problems: explicit or symbolic model checking, Markov solutions...
  - different machine architectures: SIMD, MIMD, shared- or distributed-memory...
- Split accross 'disjoint' scientific fields
  - massively parallel and distributed computers
  - formal verification
  - performance evaluation
  - Petri Nets
- Lack of unifying vision
  - mostly conference papers (never in the mainstream)
  - NEW! dedicated workshop PDMC 02 (*Parallel and Distributed Model-Checking*)
  - few journal publications
  - no survey paper
  - no book

# Organization of the lecture

- Breadth-first search of the various branches
- For each branch, depending on the available material:
  - Bibliographic references
  - Complexity results
  - Summary of the main ideas
  - Experimental results
  - If enough material (publications by different teams) : general ‘laws’, if any

# Contents of the lecture

- Parallelization and Distribution
- Explicit-State Reachability Analysis and State Space Construction
  - SIMD
  - Shared Memory
  - Distributed Memory
- Symbolic Reachability Analysis and State Space Construction
  - Shared Memory
  - Vector Processors
  - SIMD
  - Distributed Memory
- Equivalence checking
  - Distributed Memory
  - Shared Memory
- Model checking
  - LTL
  - CTL
  - Mu-calculus
- Solutions of Markov Chains
  - Numerical Solutions (Steady-State and Transient Analysis)
  - Implicit Representations
- Conclusion

# Parallelization and Distribution

# Five machine architectures

1. Vector processors
   - Pipelined functional units operating on arrays of data
   - Expensive, few publications
2. Data parallel machines (SIMD)
   - Dedicated hardware for data parallel (regular) programs
   - Require special languages and compilers
   - Expensive, few publications
3. Shared-memory multiprocessors
   - Several processors sharing a central memory
   - Programmed using (POSIX) threads and semaphores locks
   - Expensive, but used
4. Distributed-memory multiprocessors (MIMD)
   - Independent machines connected by a high-speed network
   - No shared memory, only local memories
   - Programmed using message passing primitives (eg. MPI)
   - Exemple: Networks Of Workstations (NOW), clusters of PCs, Internet computing grids
   - Cheap, available in most laboratories and companies
   - Many publications
5. Distributed shared memory multiprocessors (DSM)
   - Independent machines with both local memories and shared memories
   - Memory hierarchies, cache coherency protocols (CC-NUMA, etc.)
   - Expensive, few publications

# Speedup

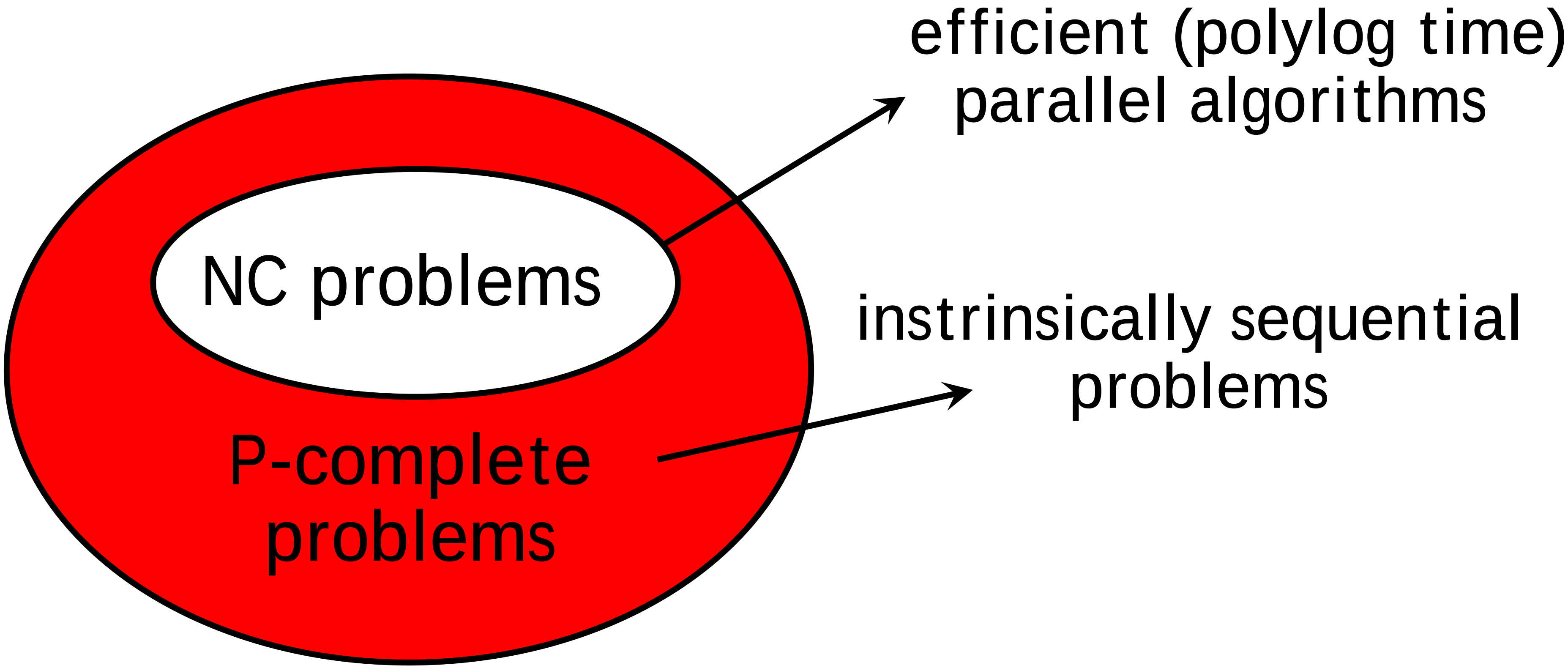
- N = number of processors/nodes/machines
- Speedup(N) = $\dfrac{\text{time taken by sequential version}}{\text{time taken by parallel version with N nodes}}$
- *Cheat(N)* = $\dfrac{\textit{time taken by parallel version with 1 node}}{\textit{time taken by parallel version with N nodes}}$

- Ideally: Speedup(N) = N or even more! (superlinear)
- Practically: Speedup (N) < N due to:
  - parallelization/distribution overhead
  - synchronizations which force tasks to idle
- Speedup often depends from the model: an efficient, general purpose, implementation is hard

# Load balancing

- How to ensure that the N processors have the same amount of work?
- Unbalanced load slows down the whole system (limited by the most loaded machine)
- Different measures:
  - Physical: CPU time used by each node
  - Logical: state space portion explored by each node

# Parallel complexity theory




'P = NC ?' is unknown (everybody failed so far)

An efficient parallel algorithm to solve P-complete problems would be a major algorithmic breakthrough

# Parallel complexity theory

Many useful problems are P-complete

But...

- This is about worst-case time complexity
- Memory space (rather than time) is our primary concern

*Parallel and Distributed*

# Explicit-State Reachability Analysis and State Space Construction

# Definitions

- Explicit state approach
  - explore states one by one
  - forward exploration only (predecessor function not available)
  - rich data types => explicit state
  - tools: CADP, SPIN, etc.
- Reachability analysis
  - (forward) exploration from the initial state
  - breadth-first (or depth-first)
  - stores all encountered states in memory
  - enables simple verifications (deadlocks, state invariants, safety properties)
- State space construction
  - similar to reachability analysis
  - additionally: store all transitions
  - used to generate LTS, Kripke structures, Markov chains
- Parallelizing state space generation is a goal in itself (at least: a prerequisite for deeper verifications)

# Basic sequential algorithm

```
E : set of (explored) states := {}      -- stored in memory
V : set of (visited) states := {S0}     -- stored in memory
T : set of transitions := {}            -- stored on disk
while V not empty do
    S1 := oneof (V)
    move S1 from V to E
    for all L, S2 such that S1 ---L---> S2 do
        if S2 neither in E nor in V then add S1 to V endif
        add transition (S1, L, S2) to T
    done
done     -- the generated state space is given by (E, T)
```

# Two main operations

- Computing the transition function

  *given S1, compute all (L, S2) such that S1 ---L--> S2*

  (done |S| times)

  => language dependent

- Detecting already known states

  *determine whether S2 is in E or in V*

  (done |T| times)

  => hash-tables (CADP, SPIN) or B-trees

# State representations

- States are vectors of values

  (e.g. Petri net markings, variable values...)

But

  - state vectors are memory expensive
  - not needed for equivalence checking, action-based model-checking, Markov chain solution...

- States are also assigned unique numbers
- The hash-table (or B-tree) ensures the mapping 'state vector <--> unique number'

# Reachability / Explicit / SIMD


- [CCBF94] S. Caselli, G. Conte, F. Bonardi, and M. Fontanesi. Experiences on SIMD Massively Parallel GSPN Analysis. Computer Performance Evaluation: Modelling Techniques and Tools. LNCS 794, pp. 266—283, 1994.
- [CCM95] S. Caselli, G. Conte, and P. Marenzoni. Parallel State Space Exploration for GSPN Models. Proc. 16th Int. Conf. on Applications and Theory of Petri Nets, LNCS 935, 181—200, 1995.

# Reachability / Explicit / SIMD

- Gen. Stoch. Petri Nets --> reachability graph

- The sequential algorithm must be revisited to match data flow patterns of the SIMD (Connection Machine)
- The two main operations (transition function and state search) are irregular and do not exhibit the regularity in data structures required for SIMD implementations

- Positive: capability to generate larger state spaces (4-10 Mstates) than on a workstation
- Negative: speed! Even with 32 processors, slower than a workstation (1.5 Mstates => 2 hours)

# Reachability / Explicit / Shared Mem.


- [AH97] S. C. Allmaier and G. Horton. Parallel Shared-Memory State-Space Exploration in Stochastic Modeling. Proc. Int. Conf. on Solving Irregularly Structured Problems in Parallel, LNCS 1253, pp. 207—218, 1997.
- [ASD97] S. Allmaier, S. Dalibor, and D. Kreische. Parallel Graph Generation Algorithms for Shared and Distributed Memory Machine. Proc. Parallel Computing: Fundamentals, Applications and New Directions(ParCo'97), pp. 581—588, Elsevier, 1997.
- [AKH97] S. Allmaier, M. Kowarschik, and G. Horton. State Space Construction and Steady-State Solution of GSPNs on a Shared-Memory Multiprocessor. Proc. 7th IEEE Int. Workshop on Petri Nets and Performance Models PNPM'97, pp. 112—121, IEEE Computer Society Press, 1997.
- [AK99] S. Allmaier and D. Kreische. Parallel Approaches to the Numerical Transient Analysis of Stochastic Reward Nets. Proc. Int. Conf. on Applications and Theory of Petri Nets, pp. 147—167, 1999.

# Reachability / Explicit / Shared Mem.

- Gen. Stoch. Petri Nets --> reachability graph

- The sequential algorithm is almost unchanged
- N threads execute concurrently
  - V implemented as N local stacks + 1 shared stack
  - E union V is implemented as a shared B-tree
- Locks on the shared stack and B-tree nodes

- With 8 processors, 4 Mstates and 25 Mtrans can be generated in 1h40
- Good (linear) speedup

# Reachability / Explicit / Dist. Mem. / Old

## First attempt at parallelizing state space generation


- [AAC87] S. Aggarwal, R. Alonso, and C. Courcoubetis. Distributed Reachability Analysis for Protocol Verification Environments. In Discrete Event Systems: Models and Applications, Lecture Notes in Computer and Information 103, pp. 40-56, Aug. 1987.

- [Aba94] P. Abaziou. Parallélisation d'OPEN/CAESAR dans l'environnement EPEE. Mémoire de DEA en informatique, IFSIC (Rennes, France), 1994.

# Reachability / Explicit / Dist. Mem. / Old

- Target: NOW (Ethernet network of SUN 2-3 workstations)
- Two (main) types of nodes:
  - generators: compute transition function
  - tabulators: state storage and search
- Key idea: the state set is partitionned between the tabulators using a hash function

  H: state vector -> tabulator identifier
- Not implemented
- Much criticized in the litterature [SD97, LS99]
  - complex: six different processes
  - termination relies on timing assumptions that may be difficult to guarantee => complex scheduling problems
  - communication overhead: each states is transferred at least 2 times over the network

# Reachability / Explicit / Dist. Mem. / Old

- 1st implementation: Th. Jéron (INRIA Rennes) 1991
  - Echidna tool for Estelle (communicating FSMs)
  - 2 generators, 2 tabulator processes
  - Target machines: iPSC, TNode
  - No publication available
- 2nd implementation : P. Abaziou (DEA student of Th. Jéron and J-M. Jézéquel, INRIA Rennes) 1993–94
  - Language-neutral platform (Open/Caesar)
  - Code distribution environment (Epee)
  - Architecture-neutral communication library (POM)
  - Target machine: Hypercube
  - Termination : Dijsktra et al. circulating probe algorithm

# Reachability / Explicit / Dist. Mem. / New


- [CCM95] S. Caselli, G. Conte, and P. Marenzoni. Cited above.
- [MCC97] P. Marenzoni, S. Caselli, and G. Conte. Analysis of Large GSPN Models: A Distributed Solution Tool. Proc. 7th IEEE Int. Workshop on Petri Nets and Performance Models, IEEE Computer Society Press , pp. 122—131, 1997.
- [NC97] D. Nicol and G. Ciardo. Automated Parallelization of Discrete State-Space Generation. Journal of Parallel and Distributed Computing, 47(2), pp. 153—167, 1997.
- [SD97] U. Stern and D. Dill. Parallelizing the Murphi Verifier. Proc. Computer-Aided Verification, LNCS 1254, pp. 256—267, June 1997.
- [CGN98] G. Ciardo, J. Gluckman, and D. Nicol. Distributed State Space Generation of Discrete-State Stochastic Models. INFORMS Journal on Computing 10(1), pp. 82—93, 1998.
- [HBB98] B. R. Haverkort, H. C. Bohnenkamp, and A. Bell. Efficiency Improvements in the Evaluation of Large Stochastic Petri Nets. Aug 1998.

# Reachability / Explicit / Dist. Mem. / New


- [KMHK98] W. J. Knottenbelt, M. A. Mestern, P. G. Harrison, and P. Kritzinger. Probability, Parallelism and the State Space Exploration Problem. Proc. 10th Int. Conf. on Computer Performance Evaluation - Modelling, Techniques and Tools. LNCS 1469, pp. 165—179 (Sections 4—6 only), Sep 1998.
- [AK99] S. Allmaier and D. Kreische. Cited above.
- [HBB99] B. R. Haverkort, A. Bell, and H. C. Bohnenkamp. On the Efficient Sequential and Distributed Generation of Very Large Markov Chains from Stochastic Petri Nets. Proc. 8th Int. Workshop on Petri Nets and Performance Models, IEEE Computer Society Press, pp. 12—21, Sep 1999.
- [LS99] F. Lerda and R. Sisto. Distributed-Memory Model Checking with SPIN. Proc. SPIN'99, LNCS 1680, pp. 22—39, July 1999.
- [Cia01] G. Ciardo. Distributed and Structured Analysis Approaches to Study Large and Complex Systems. Proc. 1st EFF/Euro Summer School on Trends in Computer Science, LNCS 2090, pp. 344—374, July 2001.
- [GMS01] H. Garavel, R. Mateescu, and I. Smarandache. Parallel State Space Construction for Model-Checking. Proc. 8th SPIN Workshop, LNCS 2057, pp. 217—234, May 2001. Revised version available as INRIA Research Report RR-4241, Dec. 2001.

# Reachability / Explicit / Dist. Mem. / New

- **Summary: All these algorithms have deep similarities and produce good results**
- Many implementations: GSPN tools, Murphi, SPIN, CADP...
- Teams who started with SIMD or shared-memory eventually switched to distributed-memory [CCM95,AK99]
- My own preferences: [CGN98,Cia01] and (of course!) [GMS01] used in our DISTRIBUTOR tool

# Reachability / Explicit / Dist. Mem. / New

## Principles

- N machines (plus possibly a frontal 'master') connected by a local network or bus
- Each machine can send messages to any other
- The state space is partitioned among the machines using a function (as in [AAC87])
- Each machine M is both a generator and a tabulator
  - It keeps *its* states in its local memory (hash table)
  - It computes the successors of its states
  - It also keeps a part of the transition relation

# Reachability / Explicit / Dist. Mem. / New

## Choosing a partition function

H (s:state vector) -> machine_id

- H can be either a hash function
  - General byte string hashing [GMS01]
  - Universal hashing [SD97]
  - Weighted sums of Petri net places [Cia01]
  - Subset of Petri net places [HBB99]
- or based on lexicographic ordering [CN97]
  - A preliminary random walk in the state space is used to obtain a sampling of reachable states
  - These sample states are lexicographically sorted in N intervals
  - H (S) = M iff state S is in the M-th interval

# Reachability / Explicit / Dist. Mem. / New

## State storage and numbering

- Machine M stores {s:states | H (S) = M} in a local hash-table (or B-tree)
- Each state is assigned a *locally unique* number *local*(S) by its owner M = H (S)
- How to produce a *globally unique* identifier?
  - Most authors use a pair: (H (S), *local*(S))
  - [GMS01] uses a number: (N * *local*(S)) + H (S) (from which projections are obtained using div and mod)

# Reachability / Explicit / Dist. Mem. / New

## Hashing assessment [HBB99] [Cia01] [GMS01]

- Hashing seems to distribute the states evenly between machines

    Ni = number of states on machine i

    Spatial balance = $\mathrm{Max}_{i,j}$ {Ni/Nj} in range 1—1.5

- But the number of cross arcs is harder to control (20%—60%)

# Reachability / Explicit / Dist. Mem. / New

## Transition storage

- Each machine computes the successors (outgoing transitions) of its states
- but receives and stores the predecessors (incoming transitions) of its states
- Machine M receives triples (n1, L, S2) such H (S2) = M and stores triples (n1, L, n2) on disk
- *The transition rate matrix is stored by columns and not by raws*
- Why? Only M knows that the number of S2 is n2
- Reduces the number of messages (contrary to [KMHK98])

# Reachability / Explicit / Dist. Mem. / New

## Distributed termination detection

- Termination: all machines have processed all their states and no more messages are in transit in the queues
- Several algorithms:
  - Dijkstra et al.'s circulating probe algorithm
  - Nicol's non-commital synchronization barrier
  - Mattern's two wave algorithm [GMS01]

# Reachability / Explicit / Dist. Mem. / New

## Remapping [NC97,Cia01]

- Classes = set of states (e.g., 100 states/class)
- Each state belongs to a single class (always the same) given by a partition function H

  H (s:state vector) : class_id

- Each class is stored on one processor (which may change) given by an array T replicated on each machine

  T [c:class_id] : processor_id

- Classes move between processors to balance load

# Reachability / Explicit / Dist. Mem. / New

- Many possible remapping strategies
  - Why? Optimize spatial or temporal balance?
  - How?
  - When?
- Experimental results
  - Remapping CPU overhead: below 5%
  - 8 processors: minor improvement
  - 16 processors: beneficial (speedup 12—13)

*Parallel and Distributed*

# Symbolic Reachability Analysis and State Space Construction

# Symbolic Rechability Analysis

- Mostly done using BDDs
- Different combinations:
  - algorithm: breadth-first or depth-first
  - machine architecture: shared memory, vector processors, SIMD, distributed [shared] memory
  - BDD variant: ‘standard’ BDD, ROBDDs (Reduced Ordered BDDs), etc.

# Reachability / Symbolic / Shared Mem.

- [KC90] S. Kimura and E. M. Clarke. A Parallel Algorithm for Constructing Binary Decision Diagrams. Proc. Int. Conf. on Computer-Aided Design, pp. 220—223, Nov. 1990.
- [KIH92] S. Kimura, T. Igaki, and H. Haneda. Parallel Binary Decision Diagram Manipulation. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science, vol. E75-A, no. 10, pp. 1255—1262, Oct 1992.

- BDDs seen as a minimal finite automata
- Generation/minimization of product automata
- Speedup:
  - 10 for 16 processors [KC90]
  - 14 for 25 processors [KIHM95]

# Reachability / Symbolic / Vector Processors


- [OIY91a] H. Ochi, N. Ishiura, and S. Yajima. Breadth-First Manipulation of SBDD of Boolean Functions for Vector Processing. Proc. 28th ACM/IEEE Design Automation Conf., pp. 413—416, Jun 1991.

- [OIY91b] H. Ochi, N. Ishiura, and S. Yajima. A Vector Algorithm for Manipulating Boolean Functions Based on Shared Binary Decision Diagrams. Proc. Int. Symp. on Supercomputing, pp. 191—200, Nov 1991.

- [OIY91c] H. Ochi, N. Ishiura, and S. Yajima. A Vector Algorithm for Manipulating Boolean Functions Based on Shared Binary Decision Diagrams. Supercomputer 46, vol. 8, no. 6, ASFRA, pp. 101—118, Nov 1991.

# Reachability / Symbolic / SIMD

[GRR95] S. Gai, M. Rebaudengo, and M. S. Reorda. A Data Parallel Algorithm for Boolean Function Manipulation. Proc. 5th Symp. on the Frontiers of Massively Parallel Computations FRONTIERS'95, pp. 28—34, Feb 1995.

- Uses breadth-first search
- Distributes BDD nodes and hash table
- Some ISCAS-85 benchmarks

# Reachability / Symbolic / Distrib. Shared Mem.

[PSC94] Y. Parasuram, E. Stabler, and S. K. Chin. Parallel Implementation of BDD Algorithms Using a Distributed Shared Memory. Proc. 27th Hawaii Int. Conf. on System Sciences Vol I: Architecture, pp. 16—25, Jan 1994.

- BDD nodes and hash table distributed and shared among processors
- Also uses a distributed stack
- Speedup: 20—32 on some ISCAS-85 circuits

[Bas98] S. Basonov. Parallel Implementation of BDD on DSM Systems. M.Sc. thesis, Computer Science Dept., Technion, 1998.

# Reachability / Symbolic / Distrib. Mem.

- [RSBS96] R. K. Ranjan, J. V. Sanghavi, R. K. Brayton, and A. Sangiovanni-Vincentelli. Binary Decision Diagrams on Network of Workstations. Proc. IEEE Int. Conf. on Computer Design, pp. 358—364, 1996.
- [SRBS96] J. V. Sanghavi, R. K. Ranjan, R. K. Brayton, and A. Sangiovanni-Vincentelli. High-Performance BDD Package Based on Exploiting Memory Hierarchy. Proc. Design Automaton Conf., June 1996.

- Distributes BDD nodes on a network of workstations
- Assigns a set of consecutive variables to the same machine
- Allows to handle BDD with several Mnodes

But
  - Not really parallel (only one machine computes at a time)
  - Unimpressive speedup (often < 1)
  - Existential quantification and variable reordering is not efficient

# Reachability / Symbolic / Distrib. Mem.

[SB96] T. Stornetta and F. Brewer. Implementation of an Efficient Parallel BDD Package. Proc. 33rd IEEE Conf. on Design Automation, pp. 641—644, 1996.

- Based on Brace-Rudell-Bryant's BDD package (1990)
- Distributes BDD nodes among processors
- Uses depth-first algorithms
- *Unique table*: distributed, two-level hash-table
- *Computed and uncomputed*: distributed hash tables
- Local LRU caches for fast access to distant BDD nodes
- Speedup: 7—57 for 32 processors on some ISCAS-85 benchmarks

# Reachability / Symbolic / Distrib. Mem.

[HGGS00] T. Heyman, D. Geist, O. Grumberg, and A. Schuster. Achieving Scalability in Parallel Reachability Analysis of Very Large Circuits. Proc. 12th Int. Conf. on Computer-Aided Verification, LNCS 1855, pp. 20—35, July 2000.

- State exploration using BFS on a NOW
- State space is cut in a fixed number of slices
- Slices travel between machines to balance load
- Fast storage: use network instead of disk
- ISCAS'89—93 and IBM benchmarks
- Handles large BDDs up to 1.2 Mnodes
- Linear speedup (0.4N — 0.6N)

# Reachability / Symbolic / Distrib. Mem.

A few other references:


- [AO96] P. Arunachalam and H. Oregon. Distributed Binary Decision Diagrams for Verification of Large Circuits. Proceedings of the IEEE Int. Conf. on Computer Design, pp. 365—370, 1996.
- [BO97] B. Yang and D. R. O'Hallaron. Parallel Breadth-First BDD Construction. ACM Sigplan Notices, 32(7), pp. 145—146, July 1997.
- [CB97] J. S. Cheng and P. Banerjee. Parallel Construction Algorithms for BDDs. Dept. of Electrical and Computer Engineering, Northwestern University (Evanston, IL, USA), 1997.

*Parallel and Distributed*

# Equivalence Checking

# Bad news #2

- Computing strong bisimilarity in finite transition systems is a P-complete problem

[ABGS91] C. Alvarez, J. L. Balcazar, J. Gabarro, and M. Santha. Parallel Complexity in the Design and Analysis of Concurrent Systems. Proc. PARLE'91, LNCS 505, pp. 288—303, 1991.

=> Algorithms for computing bisimulation seem to be inherently sequential and hard to parallelize

# Equiv. checking / Distributed Mem.

- [ZS92] S. Zhang and S. A. Smolka. Towards Efficient Parallelisation of Equivalence Checking Algorithms. Proc. FORTE'92 (Lannion, France), pp. 133—146, 1992.

- [BS02] S. Blom and S. Orzan. A Distributed Algorithm for Strong Bisimulation Reduction of State Spaces. In L. Brim and O. Grumberg, eds., Proc. Workshop on Parallel and Distributed Model Checking PDMC'02 (Brno, Czech Republic), Aug. 2002. To appear in ENTCS.

# Equiv. checking / Distributed Mem.

- Two attempts at parallelizing the Kanellakis-Smolka partition refinement algorithm
  - [ZS92]: The block splitting task is distributed among processors (to optimize time)
  - [BS02]: States are distributed between machines

- Obtained results (in both cases):
  - some improvements, not fully convincing
  - more experimental feedack is needed

# Equiv. checking / Shared Mem.

- [ZS92] S. Zhang and S. A. Smolka. Cited above.
- [RL98] S. Rajasekaran and I. Lee. Parallel Algorithms for Relational Coarsest Partition Problems. IEEE Trans. on Parallel and Distributed Systems, 9(7), July 1998.
- [JKOK] C. Jeong, Y. Kim, Y. On, and H. Kim. A Faster Parallel Implementation of the Kanellakis-Smolka Algorithm for Bisimilarity Checking. Tech. Report, Lab. Computer Software Technology, Electronics and Telecommunications Research Institute, Taejon, Korea.

# Equiv. checking / Shared Mem.

- [RL98] proposes two ‘nearly optimal’ algorithms for CRCW PRAM machines
- [JKOK] proposes an alternative algorithm and claim superior performance

# Four pragmatic remarks

1. The problem remains open for distributed-memory machines (including NOWs)
2. Work focuses on Kanellakis-Smolka algorithm, which seems simpler to parallelize than Paije-Tarjan algorithm.
3. Work focuses on time improvement, but memory can be a problem too.
4. Work is for strong bisimulation only. No work on weaker equivalences (e.g., branching, observational)

*Parallel and Distributed*

# Model Checking

# A tentative classification for a complex situation

- Many potential combinations:
  - type of logic: LTL, CTL, alternation-free or full mu-calculus
  - state space: explicit state or symbolic (BDD)
  - algorithm: global or local (on the fly)
  - machine architecture: vectorial, SIMD, shared- or disstributed-memory
- But
  - many combinations have not been studied yet
  - existing ones have been studied by only one team

# Known combinations

- LTL model-checking
  - explicit state / distributed memory
  - explicit state / shared memory
- CTL model-checking
  - explicit state / vector processors
  - symbolic / vector processors
  - explicit state / SIMD
- Mu-calculus model-checking
  - explicit state / distributed memory
  - symbolic / distributed memory

*Parallel and Distributed*

# LTL Model Checking

# Bad news #3

- LTL model checking relies on a (nested) depth-first search (DFS) of the state space. This algorithm is implemented in SPIN.

- Unfortunately, DFS is a P-complete problem

[Rei85] J. H. Reif. Depth-first search is inherently sequential. Information Processing Letters, 20(5), pp. 229—234, 1985.

# LTL / Explicit State / Distributed Memory


- [LS99] F. Lerda and R. Sisto. Cited above.
- [BBS01] J. Barnat, L. Brim, and J. Stribrna. Distributed LTL Model-Checking in SPIN. Proc SPIN'01 (Toronto, Canada), LNCS 2057, pp. 200—216, May 2001.
- [BCKP01] L. Brim, I. Cerna, P. Krcal, and R. Pelanek. Distributed LTL Model Checking Based on Negative Cycle Detection. Proc. FTS-TCS 2001 (Bangalore, India), Sep. 2001.

# LTL / Explicit State / Distributed Memory

- [LS99] does not perform a DFS and cannot be used to check full LTL (only safety properties)
- [BBS01] proposes a distributed algorithm for nested DFS. No experimental results reported.
- [BCKP01] replaces nested DFS by a shortest path problem (negative cycle detection)
  - Worst-case time complexity worse than nested DFS
  - But easier to distribute on several machines
  - Practically, less messages and better speedup

# LTL / Explicit State / Shared Memory

- [IB02a] C. P. Inggs and H. Barringer. On the Parallelisation of Model Checking. Proc. 2nd Workshop on Automated Verification of Critical Systems AVOCS'02. Tech. report, Univ. of Birmingham, Apr. 2002.
- [IB02b] C. P. Inggs and H. Barringer. Effective State Exploration for Model Checking on a Shared Memory Architecture. In L. Brim and O. Grumberg, eds., Proc. Workshop on Parallel and Distributed Model Checking PDMC'02 (Brno, Czech Republic), Aug. 2002. To appear in ENTCS.

Papers not available before this lecture (presumably related to LTL model checking)

*Parallel and Distributed*

# CTL Model Checking

# Bad news #4

CTL model-checking is a P-complete problem

E. A. Emerson, cited in [ZOS94] S. Zhang, O. Sokolsky, and S. A. Smolka. On the Parallel Complexity of Model Checking in the Modal Mu-Calculus. Proc. 9th IEEE Symp. on Logic in Computer Science, pp. 154—163, July 1994.

# CTL / Explicit State / Vector Processors

[HMH90] H. Hiraishi, S. Meki, and K. Hamaguchi. Vectorized Model Checking for Computation Tree Logic. Proc. Computer-Aided Verification, LNCS 531, pp. 44—53, 1990.

- Bit vectors: $V_F[s]$=value of formula F in state s
- Bottom-up evaluation of $V_F[.]$ on the syntactic structure of formula F
- Vectorial execution was 26—39 times faster than scalar execution (on the same machine)
- It was 1000 times faster than Clarke et al.'s sequential CTL model checker (on a Sun 3/80)

# CTL / Symbolic / Vector Processors

- [OIY91a] [OIY91b] [OIY91c] H. Ochi, N. Ishiura, and S. Yajima. Cited above.
- [HHOY91] H. Hiraishi, K. Hamaguchi, H. Ochi, and S. Yajima. Vectorized Symbolic Model Checking for Computation Tree Logic for Sequential Machine Verification. Proc. Computer-Aided Verification, LNCS 575, pp. 214—224, 1991.

- Based on a vectorial BDD package
- Use BFS algorithm to evaluate CTL, rather than DFS (incompatible with vector processing)
- Vectorial execution was 6—20 times faster than scalar execution (on the same machine)

# CTL / Explicit State / SIMD

- [Bou95] M. Bourdellès. Proposition d'une parallélisation SPMD de l'algorithme de model-checking utilisant la logique CTL. Mémoire de DEA en informatique, IFSIC (Rennes, France), 1995.
- States are partitioned between processors
- Each processor computes the same CTL (sub-)formula (SIMD)
- Local computations altern with propagation to neighbours
- Not implemented

*Parallel and Distributed*

# Mu-Calculus Model Checking

# Bad news #5

- The problem of checking whether an LTS is a model of a formula of the propositional mu-calculus is P-complete.

  [ZOS94] S. Zhang, O. Sokolsky, and S. A. Smolka. Cited above.

- This is even true under strong assumptions
  - the formula is fixed **and** alternation-free
  - **and** the LTS is deterministic **and** acyclic
  - **and** the LTS fan-in **and** fan-out are bounded by 2

# Mu-calculus / Explicit State / Dist. Mem.


- [BLW01a] B. Bollig, M. Leucker, and M. Weber. Parallel Model Checking for the Alternation Free Mu-Calculus. Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems TACAS'01, LNCS 2031, pp. 543—558, 2001.

- [BLW01b] B. Bollig, M. Leucker, and M. Weber. Local Parallel Model Checking for the Alternation Free Mu-Calculus. Tech. Report AIB-04-2001, Aachen University of Technology, 2001. Revised version of [BLW01a].

# Mu-calculus / Explicit State / Dist. Mem.

- Alternation-free fragment of modal mu-calculus
- Parallelization of Stirling's game-based local algorithm
- States of the game graph are partitionned between processors using a hash function
- Successors and predecessors of each state are kept
- Game graph built and coloured simultaneously (BFS traversal)
- Mitigated results
  - NOW with up to 52 processors
  - Up to 1 Mstates (LTS) and 13 Mstates (game graph)

  But
  - Implementation does not work on the fly
  - Seems to be slow (9 minutes for an LTS with 1 Mstates)
  - No speedup below 5 processors

# Mu-calculus / Symbolic / Dist. Mem.

[GHS01] O. Grumberg, T. Heyman, and A. Schuster. Distributed Symbolic Model Checking for Mu-Calculus. Proc. 13th Int. Conf. on Computer Aided Verification, LNCS 2102, pp. 350—362, 2001

- Applies to full propositional mu-calculus
- Global algorithm (not on-the-fly: requires the construction of the whole Kripke structure)
- Symbolic (BDD) representation of the state space
- State space slicing into subsets of the ‘same’ size
- Slices are distributed to processors
- Proof of correctness given
- No implementation reported

*Parallel and Distributed*

# Solution of Markov Chains

# Goals

Given a Markov chain, one wants to compute

- steady-state analysis
  - for ergodic CTMCs: stationary state probabilities
  - for absorbing CTMCs: expected state sojourn state times until absorbtion
- and/or transient analysis

  instantaneous or cumulative measures for a set of user-defined time instants:
  - state probability vector at time t1, t2, ... tn
  - total time spent in each state up to time t

  => In any case, the solution is a real vector indexed by states

# Exemple: Steady State Probabilities

- Transition rate matrix

  R [s, s'] = **if** exists s -->s' **then** *rate* (s, s') **else** 0

- Infinitesimal generator matrix

  Q [s, s'] = **if** s<>s' **then** R [s, s'] **else** —Sum $_{s'' <> s}$ R [s, s'']

- Numerical stationary solution of CTMC R:

  Compute a vector pi[s] of probabilities /

  pi Q = 0 and Sum$_s$ pi[s] = 1

  => solve a linear homogeneous system of equations

# Numerical methods

[Ste94] W. J. Stewart. Introduction to the Numerical Solution of Markov Chains. Princeton University Press, 1994.

- Several algorithms:
  - Power
  - Jacobi
  - Gauss-Seidel
  - SOR iterations
  - Conjugate Gradient Squared (CGS)
  - Block-oriented methods: block-Jacobi
  - ...

# Difficulties of numerical methods

- The state space S is very large
  - Probability vectors pi are of dimention |S|
  - Matrixes R and Q have |S|*|S| elements
  - These matrices are very sparse
  - Memory is a bottleneck
- Key operation: matrix.vector (or vector. matrix) multiplication
  - Floating-point computations are CPU-intensive
  - Time also can be a bottleneck
  - Robust algorithms to ensure num. stability

# Numerical / Steady State / Parallel


- [AKH97] S. Allmaier, M. Kowarschik, and G. Horton. Cited above (see Sections 3—4 of their paper).
- [MCC97] P. Marenzoni, S. Caselli, and G. Conte. Cited above (see Sections 5—6 of their paper).
- [CGN98] G. Ciardo, J. Gluckman, and D. Nicol. Cited above (see Section 4.2 of their paper).
- [MPS99] V. Migallon, J. Penadès, and D. B. Szyld. Experimental Study of Parallel Iterative Solutions of Markov Chains with Block Partitions. In B. Plateau, W. J. Stewart, and M. Silva, Numerical Solution of Markov Chains, pp. 96—110, Prensas Universitarias de Zaragoza, Sep 1999.
- [BB00] A. Bell and B. R. Haverkort. Serial and Parallel Out-Of-Core Solution of Linear Systems Arising from Generalised Stochastic Petri Nets. RWTH Aachen, Germany, 2000.
- [Cia01] G. Ciardo. Cited above.

# Numerical / Steady State / Parallel

Summary:

- Parallel/distributed implementations outperform sequential ones
- The critical issue is the parallel sparse matrix.vector multiplication.
- Gauss-Seidel is efficient sequentially, but difficult to distribute (contrary to Jacobi and CGS).
- Solving large Markov chains still takes time:
  - 50 Mstates requires < 1 day
  - 724 Mstates requires >16 days

  on a cluster of 26 PCs using MPI [BB00]

# Numerical / Steady State / Disk-Based

- [DS98] D. D. Deavours and W. H. Sanders. An Efficient Disk-Based Tool for Solving Large Markov Models. Performance Evaluation, 33(1), pp. 67—84, June 1998.

- A single (bi-processor) workstation
- Gauss-Seidel method
- Matrix stored on disk ('out of core')
- Two cooperating threads:
  - high throughput disk I/O
  - computation
- Successful method: 10 Mstates-100 Mtrans. on a single 128 MB RAM workstation

# Numerical / Steady State / Disk-Based

- [KH99] W. J. Knottenbelt and P. G. Harrison. Distributed Disk-Based Solution Techniques for Large Markov Models. In B. Plateau, W. J. Stewart, and M. Silva, Proc. 3rd Int. Meeting on the Numerical Solution of Markov Chains, pp. 58—75, Prensas Universitarias de Zaragoza, Sep 1999.

- Distributed-memory approach
- Jacobi and CGS methods
- Matrix stored on disk
- Two cooperating processes per node:
  - high throughput disk I/O
  - computation and inter-nodes communications
- Reorder matrix raws/column to improve locality exploiting structure of BFS-generated graphs
- 50 Mstates-500 Mtrans. in 17 hours on a Fujitsu computer with 16 nodes (300 MHz, 256 MB RAM) using MPI

# Numerical / Transient

[AK99] S. Allmaier and D. Kreische. Cited above.

- Shared-memory implementation
  - Parallelization is simple: CTMC in shared memory
  - Solves a CTMC with 2 Mstates and 19 Mtrans in 1 hour 16 on a Convex SPP (8 processors)
  - For larger examples, swapping issues...
- Distributed-memory implementation
  - Main issue: vector.matrix multiplication
  - Solves a CTMC with 2.5 Mstates and 24 Mtrans in 14 minutes on a cluster of 8 PCs.
  - Scales up to 16 PCs

# Implicit representations


- [Don93] S. Donatelli. Superposed Stochastic Automata: A Class of Stochastic Petri Nets with Parallel Solution and Distributed State Space. Performance Evaluation, vol. 18, pp. 21—26, 1993.

- [BFK99] P. Buchholz, M. Fischer, and P. Kemper. Distributed Steady State Analysis Using Kronecker Algebra. In B. Plateau, W. J. Stewart, and M. Silva, eds., Proc. 3rd Int. Meeting on the Numerical Solution of Markov Chains (Zaragoza, Spain), pp. 76—95, Sep 1999.

- [Cia01] G. Ciardo. Cited above (see Section 5).

# Conclusion

# Past and present

- Significant work has been done
- Some clear successes:
  - Reachability analysis / Explicit state / Distributed
  - Reachability analysis / Symbolic / Distributed
  - Markov chains solutions / Steady State / Disk-based
- Approaches are split between different branches of computer science
- A unified view can be fruitful

# Future

- A lot of 'useful' problems are still open
- *Not covered in this lecture: parallel verification of timed systems...*
- Many problems have only been attacked by one team: cross-check the results!
- Implementations/experiments are essential
- The best sequential algorithms are not the best candidates
- NOWs, PC clusters, Internet grids(?) are everywhere

Think distributed!