Parallel State Space Construction for Model-Checking

Hubert Garavel, Radu Mateescu, Irina Smarandache

INRIA Rhône-Alpes / VASY
655, avenue de l'Europe
F-38330 Montbonnot Saint Martin

May 20, 2001

Motivations

- Industrial collaborations at INRIA/VASY
  - Cache coherency protocols (Bull)
  - Embedded applications on smart cards (CP8)
  - Mobile Java agents (MGE-UPS)
- Type of problems
  - Asynchronous, message-passing systems
  - Complex data types
- Approach
  - Use of process algebras: LOTOS, E-LOTOS
  - Explicit-state space methods

Motivations

- Model-checking : memory is the bottleneck
- a) Use more clever methods
  - compositional, partial orders, symmetries
- b) Use more powerful machines
  - "standard" PC : 1 GB RAM (2 GB max)
  - "power" machine (mainframe, SMP, CC-NUMA) => very expensive
  - many small machines:
    - networks of workstations (NOW)
    - clusters of PC
    - meta-computing

Related work

- Low-level models (a lot)
  - Petri nets
  - stochastic Petri nets
  - discrete-time Markov chains
  - continuous-time Markov chains
- Higher-level languages (very few)
  - Murphi (1)
  - SPIN (1)

Definitions

- LTS (Labelled Transition System)
  - S: set of states
  - s_0: initial state
  - A: set of actions
  - T: transition relation
- *Explicit* LTS: entirely generated
- *Implicit* LTS: generated on demand ("*on the fly*")

The CADP toolbox

- Compilers for LOTOS: *CAESAR*, *CAESAR.ADT*
- Language-independent tools:
  - Simulators: *OCIS*, *Xsimulator*
  - Model-checkers: *Evaluator*, *XTL*
  - Bisimulation tools: *Aldebaran*, *BCG_MIN (+ Fc2)*
  - Test generator: *TGV*
- Cross-tool functionalities:
  - Explicit LTSs : *BCG*
  - Implicit LTSs (on the fly): *Open/Caesar*
  - Compositional verification: *SVL*
  - Unified graphical-user interface: *EUCALYPTUS*

Open/Caesar

LOTOS
LTS in BCG format
communicating LTSs
UML/RT (UMLAUT)
SDL (IF)
(untimed) KRONOS

language-dependent compilers

Open/Caesar API

simulation
on the fly verification
LTS generation
test generation (TGV)
test execution (TorX)

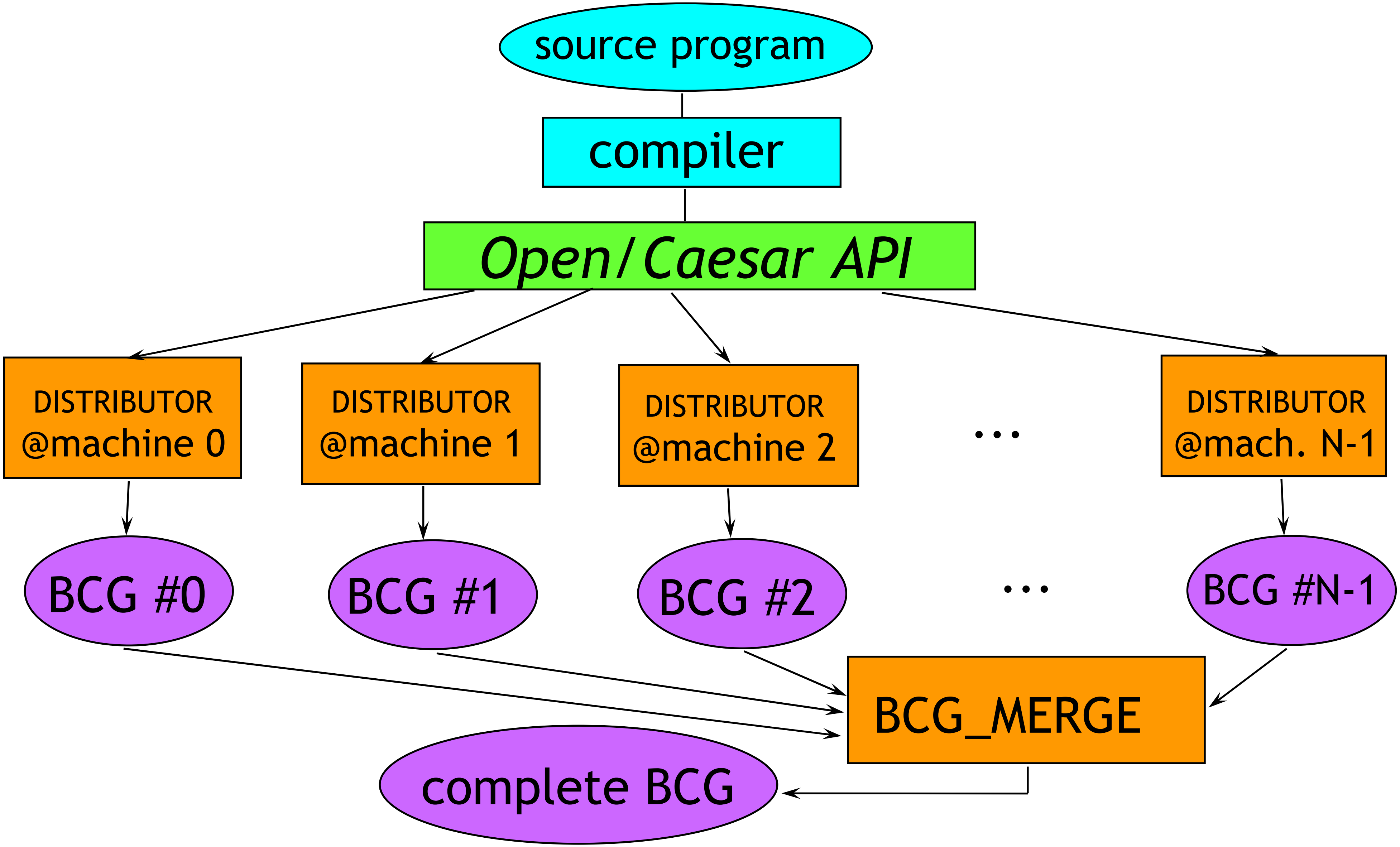
language-independent tools

Goal of our work

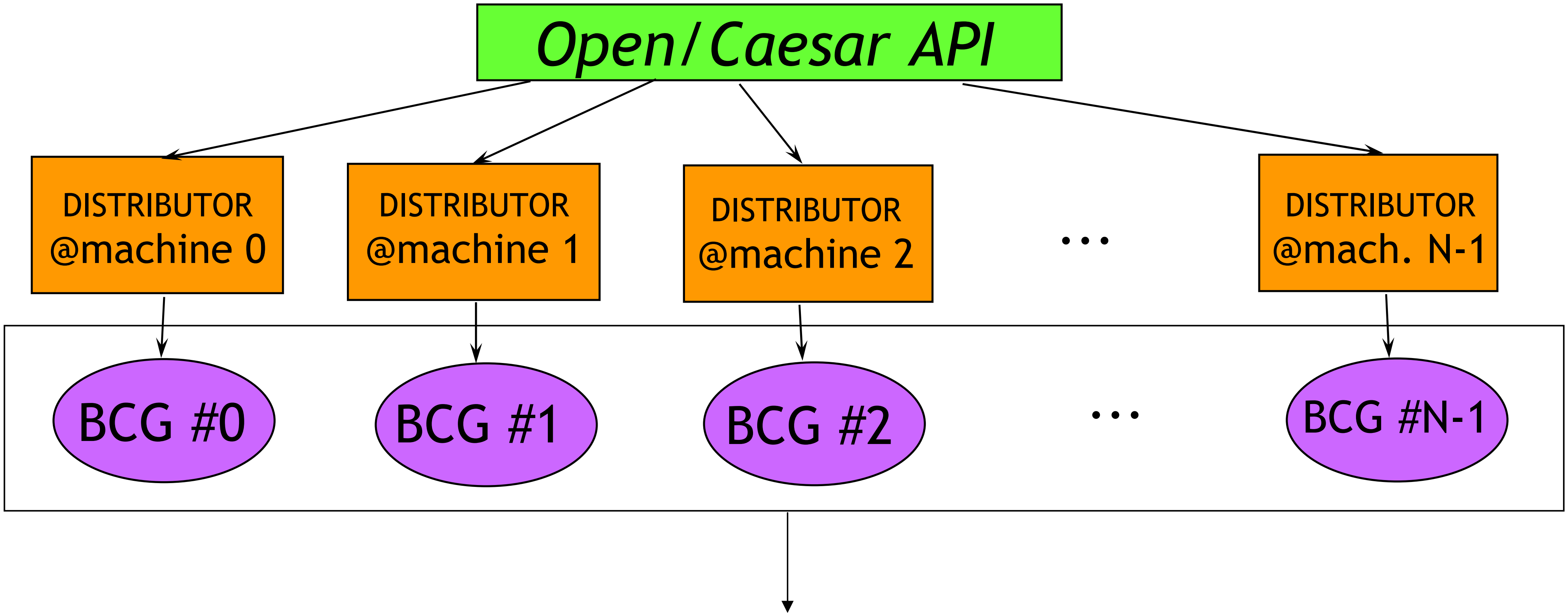
Develop an Open/Caesar software component for parallel state space construction, which can be used for several languages.

caesar.open	*prog.lotos*	distributor	*M1 M2 M3* …
uml.open	*prog.uml*	distributor	*M1 M2 M3* …
if.open	*prog.if*	distributor	*M1 M2 M3* …

Tool architecture



Phase 1 : DISTRIBUTOR



partitioned LTS = (collection of BCG files) + global initial state

Distributed algorithm (1)

- N machines numbered 0 … N-1
- Static partition function h : S -> [0 … N-1]
- Each machine computes/stores an LTS fragment:
  - **Si = { s *in* S | h (s) = i }**
    stored in the local memory (Open/Caesar state table)
    globally unique numbers: s *in* Si <=> n (s) *mod* N = i
  - **Ti = { (s, a, s') *in* T | h (s') = i }**
    stored on the local disk (BCG file)
  - **Ai = { a *in* A | *exists* (s, a, s') *in* Ti }**
    In fact **Si = { s *in* S | h (s) = i *or exists* (s, a, s') *in* Ti }**

Distributed algorithm (2)

- Each machine M[i] receives triples of the form
  (n: *number*, a: *action*, s: *state vector*)
  such that h (s) = i
- M[i] inserts s in its local state table and gives to s a unique number *n* (s)
- M[i] computes the successors (s, a', s') of s
- If h (s') = i then s' is stored locally
  else (*n* (s), a', s') is sent to M[h(s')]

Termination detection

- Global termination <=>

  all local computations are finished and

  all communication channels are empty
- (Virtual) unidirectional ring between machines
- Principles:
  - the initiator machine checks for termination everytime it finishes its local computations
  - termination occurs when the total numbers of sent and received messages are equal

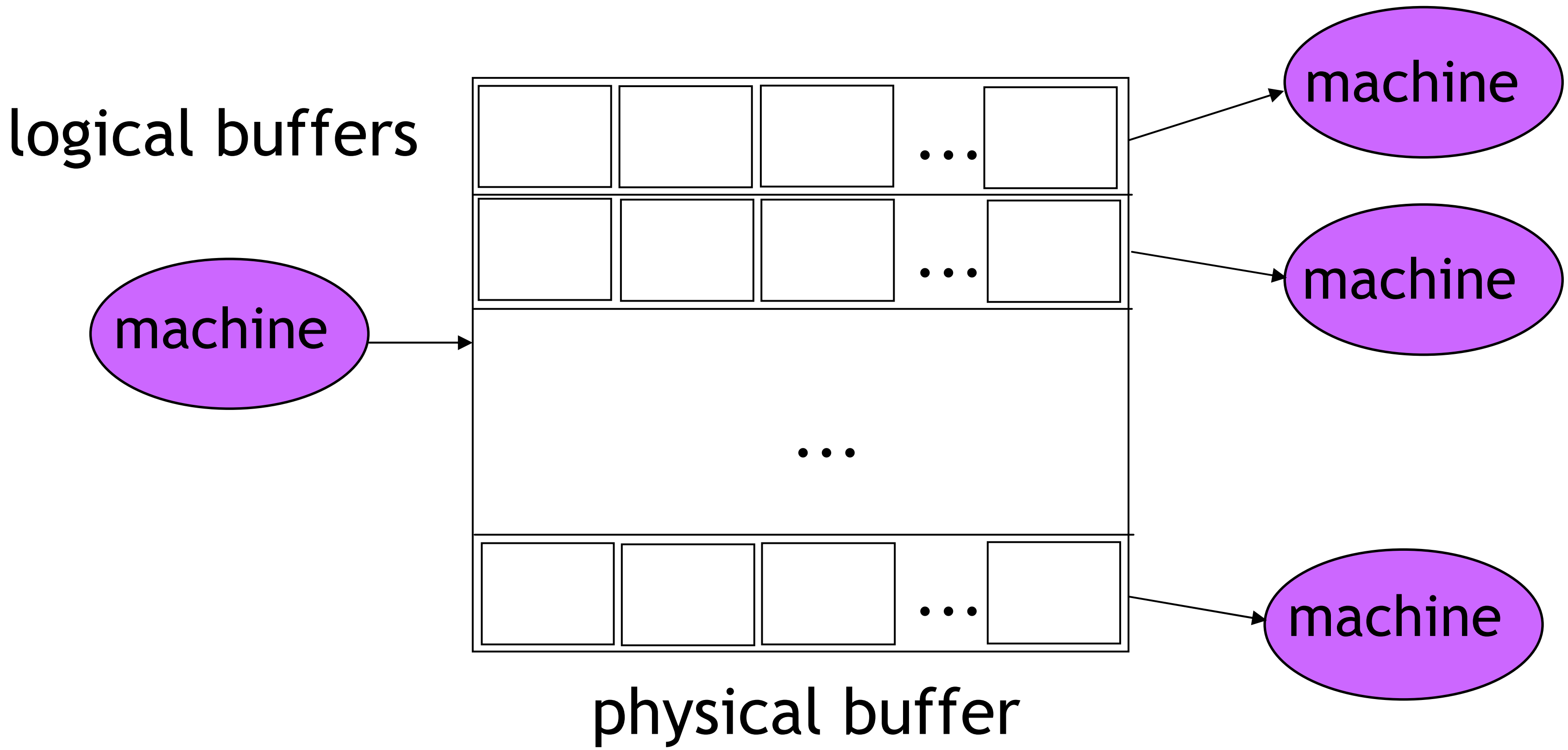
Implementation

- Low-level communication primitives:
  - TCP/IP sockets (MPI not necessary)
  - Non-blocking SEND and RECEIVE primitives
  - Deployment using rsh/rcp
- Several architectures supported:
  - **Debugging** : 3 Sun workstations on Ethernet LAN
  - **Performance measurements** : cluster of 10 Linux PCs (450 MHz, 512 MB RAM) connected by SCI (*Scalable Coherent Interface*)

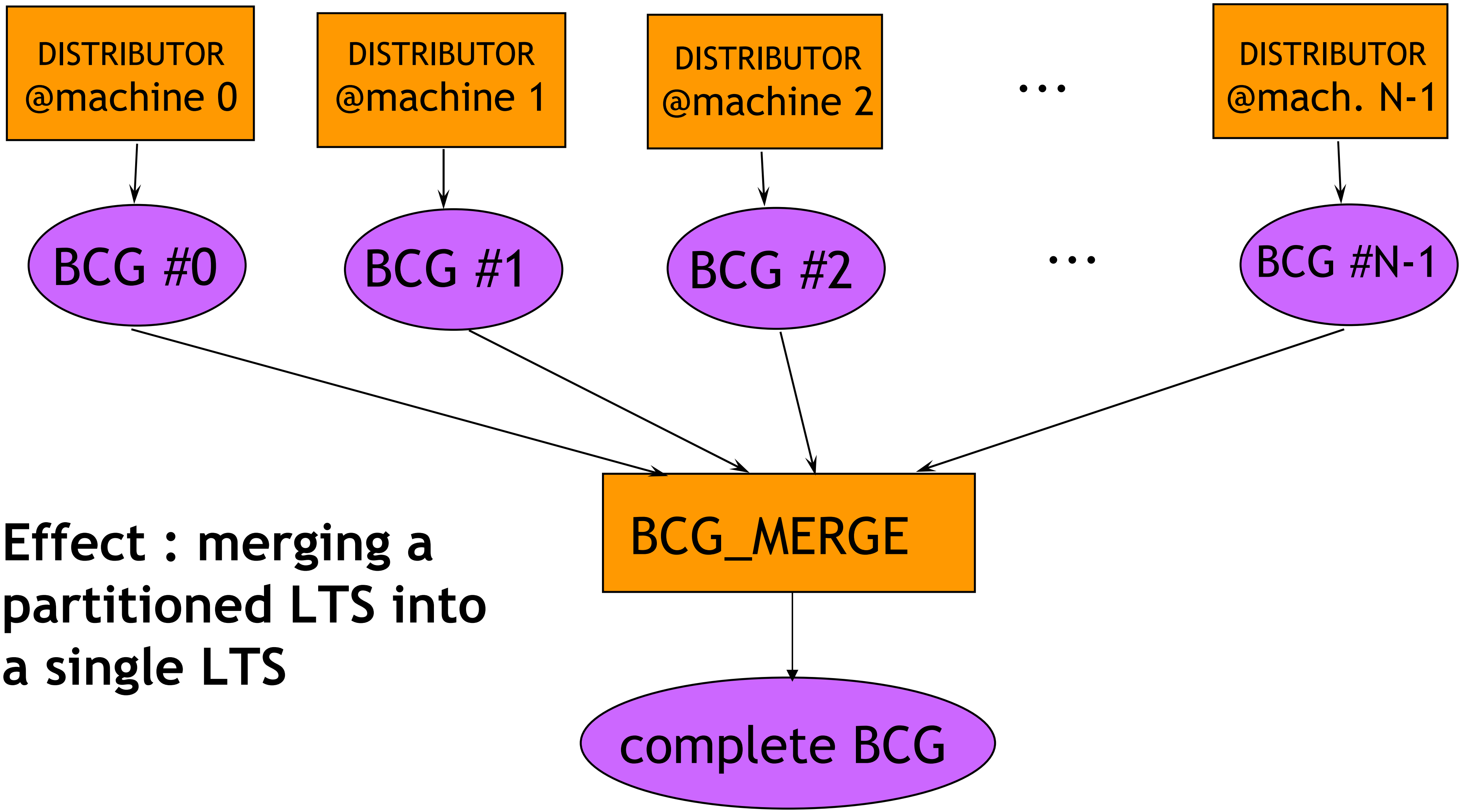
Communication buffers

Buffering messages improves performance

=> use as much buffering as possible



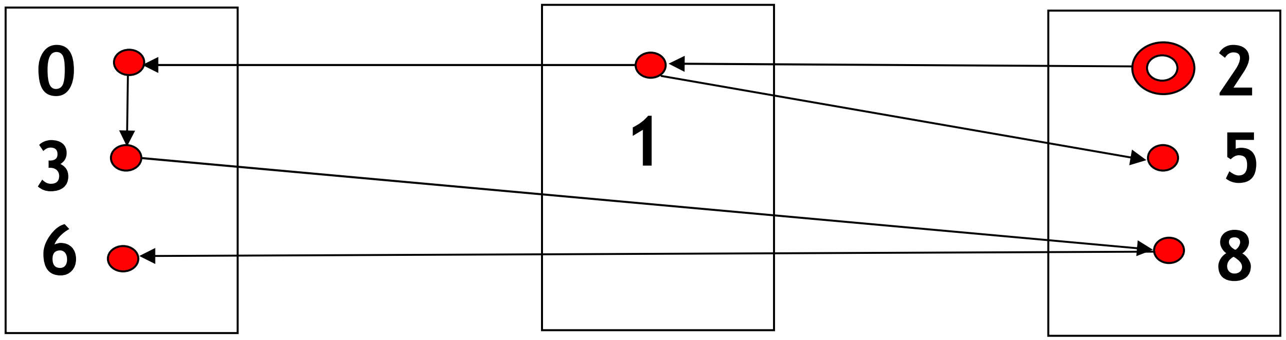
Phase 2: BCG_MERGE



Effect : merging a partitioned LTS into a single LTS

Principles of BCG_MERGE

- Each fragment of a partionned LTS:
  - is **not** a connected graph
  - has **sparse** state numbers



- BCG_MERGE
  - ensures contiguous state numbers (for compaction)
  - processes all fragments one by one (and only once)

Experiments: 3 case-studies

- Philips HAVi protocol [Judi Romijn]

  1.04 Mstates (80 bytes), 3.37 Mtrans

- Token-ring leader election protocol

  12.3 Mstates (6 bytes), 45.3 Mtrans

- SCSI-2 bus arbitration protocol
  - 5 devices : 0.96 Mstates (13 bytes) 6.00 Mtrans
  - 6 devices: 1.20 Mstates (15 bytes) 13.8 Mtrans

Results: speedup

- Parallelization gives an almost linear speedup
  - GAIN: state tables are distributed => linear reduction in searching/inserting states in tables (open hashing)
  - GAIN: distributed computation of transition function
  - LOSS: communication overhead, termination detection
- Expensive transition function => better speedup
- Measurements: T_n = *gen. time with n machines*
  - HAVi: $T_n = T_1 / (\mathbf{0.3\ N})$
  - Token-ring: $T_n = T_1 / (\mathbf{0.4\ N})$
  - SCSI-2: $T_n = T_1 / \mathbf{N}$ *(ideal speedup)*

Results: load balancing

- Good load balancing =>
  the *N* parts of the LTS have similar sizes =>
  h : S -> 0...N-1 distributes states uniformly
- Problems:
  - the state set S is not known in advance
  - the partition function is static
  - language independence => no hint on state vectors
- Approach taken:
  - assume that state vectors are distributed uniformly
  - h (S) = *(integer value of bit string S)* modulo N

Conclusion

- Explicit-state generation/verification seems appropriate for massively parallel computers
- Significant gains can be expected:
  - memory: 1-2 orders of magnitude in state spaces
  - time: linear speedups expected
- Our approach:
  - parallel state-space construction, sequential verification
  - construction of the transition relation (LTS model)
  - language neutral (Open/Caesar)
  - architecture neutral (NOWs, clusters of PCs)

Future work

- Large-scale experiments:
  - INRIA 's cluster of 256 PCs (50-100 GB RAM)
- Distribute DISTRIBUTOR within CADP:
  - support for dynamic data types (lists, trees…)
  - full automation (deployment, merging)
  - more parametrization (different RAM sizes…)
  - separate algorithm from communication code
  - on-line monitoring of LTS construction
- If needed, parallelize verification itself
  - sequential algorithms working on partitioned LTS
  - parallel algorithms working on the fly