SatX10: A Plug&Play Parallel SAT Framework

Bard Bloom, David Grove, Benjamin Herta, Ashish Sabharwal, Horst Samulowitz, and Vijay Saraswat

IBM Watson Research Center, NY, USA
SatX10: Designing Parallel Solvers Made Easier

- A New Framework for designing and executing parallel SAT solvers
  - Composed out of a diverse set of individual solvers
  - Support for Information Sharing
  - Single executable that easily runs on 1 machine or 1000s of machines!

- Key Motivation: separation of expertise

**SAT Experts** focus on high level parallelization strategies:
- Which solvers to use?
- How best to diversify?
- What to share?
- How often to share?
- With whom?

**Parallel Systems Experts** focus on designing an efficient and sound parallel execution system:
- How to share information?
- How to spawn processes?
- Sync. across 1000s of machines
- Support rich sharing topologies
What is X10?

- Parallel programming & execution environment; a high-level language that gets compiled into C++ or Java
  - Java: Runs on Java 1.6 VMs and Ethernet
  - C++: x86, x86 64, PowePC, Sun CPUs, and on the BlueGene/P, and Ethernet and Infiniband interconnects (through an MPI implementation)
    - Runs on Linux, AIX, MacOS, Cygwin OS
    - Same source code can be compiled for all these environments

- Allows parallel solver to run on a single machine with multiple cores and across multiple machines, sharing information such as learned clauses

- Provides many handy concurrency and distribution features, e.g.:
  - `async` block: run asynchronously
  - `at(p)` block: execute block at “place” p (place ≡ independent memory space on same or different machine; think: one solver running per place)
Typical State-of-the-Art Parallel SAT Solvers

1. Essentially based on a portfolio of SAT solvers (Diversity)
   - Often the same underlying baseline solver, but different parameterization (e.g., restart frequency)
   - Main reason for limiting to a single solver is convenience!

2. Exchange of discovered knowledge (Knowledge Sharing)
   - Learned clause sharing (normally restricted to some clause length, e.g., Plingeling, the Gold winning parallel solver in 2011, shares only unit clauses)

Challenge: Can we make it easier for SAT experts to design and test more complex diversification and knowledge sharing strategies?
SatX10 Provides a Unique Solution

- Framework to combine multiple sequential/multi-threaded SAT solvers
  - Interference with / addition to SAT solver code is minimal
    - The framework provides some of the plumbing through base classes
  - Single executable that runs on one machine or 1000s

- Built using the parallel programming and execution language X10
  - Benefits from the years of research in building efficient, reliable, and easy to use parallel programming systems
  - Allows SAT experts to focus on what they do best!

- In principle, can all be done with, e.g., MPI
  - But certainly not easy, as evidenced by the lack of enough parallel SAT solvers today...
SatX10 Architecture

SatX10 Framework 1.0

SolverX10Callback
SolverSatX10Base *

Callback Methods
x10_step()
x10_processOutgoingCls()'

SatX10__Solver
Implement callbacks of base class

CallbackStats (data)
Routines for X10 to interact with solvers
solve()
kill()
bufferIncomingCls() printInstanceInfo()

SatX10__Minisat
Specialized solver: Minisat::Solver *

SatX10__Glucose
Specialized solver: Glucose::Solver *

SolverSatX10Base

Data Objects
Callback *
placeID
maxLenShrCl
outBufSize
incomingClsQ
outgoingClsQ

Pure Virtual Methods
x10_parseDimacs()
x10_nVars()
x10_nClauses()
x10_solve()
x10_printSoln()
x10_printStats()

Other Controls
x10_kill()
x10_wasKilled()
x10_accessInclBuf()
x10_accessOutBuf()

Minisat::Solver
Implement pure virtual methods of base class

Other Methods
bufferOutgoingCls() processIncomingCls()

Glucose::Solver
Implement pure virtual methods of base class

Other Methods
bufferOutgoingCls() processIncomingCls()

Specialization for individual solvers

SatX10.x10
Main X10 routines to launch solvers at various places:
Glucose::Solver *
Minisat::Solver *

...
Configuration of SAT Solvers: Example

- Specified at execution time through a configuration file:

```
example.ppconfig:

// solvers 1-16
Glucose_2.0  -verb=0
Minisat_2.2.0 -verb=0
CirMinisat   -verb=0
Minisat_2.0  -verb=0
Glucose_2.0  -verb=0 -no-luby
CirMinisat   -verb=0 -no-luby
Glucose_2.0  -verb=0 -phase-saving=1
CirMinisat   -verb=0 -phase-saving=1
Minisat_2.2.0 -verb=0 -phase-saving=1
Glucose_2.0  -verb=0 -ccmin-mode=0
CirMinisat   -verb=0 -ccmin-mode=0
Glucose_2.0  -verb=0 -no-luby -phase-saving=1
CirMinisat   -verb=0 -no-luby -phase-saving=1
Minisat_2.2.0 -verb=0 -no-luby
Minisat_2.2.0 -verb=0 -rinc=1.75
Glucose_2.0  -verb=0 -rinc=2.25
```
Configuration of Hosts / Machines: Example

- Specified at execution time through a hostfile:

```
hostfile-example.txt:

localhost
localhost
aiopt-a-11
aiopt-a-11
aiopt-a-12
aiopt-a-12
aiopt-a-13
aiopt-a-13
aiopt-b-11.watson.ibm.com
aiopt-a-11.watson.ibm.com
aiopt-a-12.watson
aiopt-a-12.watson
...
```
Sample Run

Note: The same command line can launch the run on 1000s of cores!

$ make  # compiles everything using x10c++ compiler; produces ./SatX10

$ export X10_NPLACES=2
$ export X10_HOSTFILE=hostfile-example.txt

$ ./SatX10 ppconfigs/example.ppconfig 10 10 example_small.cnf

==== SatX10 Parallel SAT Solver ====
SatX10 parameters:
- CNF filename: example_small.cnf
- ppconfig file: ppconfigs/example.ppconfig
- maxLengthSharedClauses: 10
- outgoingClausesBufSize: 10
- printSolution: false

Place(0): will run LEAF solver Glucose_2_0 [-phase-saving=1 -no-luby]

Instance information:
- filename: example_small.cnf
- number of variables: 3
- number of clauses: 2

Place(1): will run LEAF solver Minisat_2_2_0 [-verb=0 -rnd-seed=1706]

Place(0): starting master process for restart sequence
Place(0): INSTANCE SOLVED!
Support for Arbitrary Communication Patterns

Flat Hierarchical Topology for Large Scale SatX10 (1000+ cores)

Individual Solver a2
- SAT Solver
- Information Containers
  - Buffer Incoming
  - Buffer Outgoing

Center A
- Special SAT solver that employs additional inference and processing
- Information Processing
  - Sorts Clauses and keeps Hashmap (String to Bool) to reduce duplicates
  - Selects clauses to be communicated across centers and individual solvers
  - Information Containers
    - Buffer Incoming/Outgoing (Individual)
    - Buffer Incoming/Outgoing (Centers)

Individual Solver a1
- SAT Solver
- Information Containers
  - Buffer Incoming
  - Buffer Outgoing

Individual Solver aN
- SAT Solver
- Information Containers
  - Buffer Incoming
  - Buffer Outgoing

Solver b2
- Information Containers
  - Buffer Incoming/Outgoing (Individual)

Solver b1

Center B

Center C
Empirical Results: Same Machine

- Same machine, 8 cores, clause lengths=1 and 8

**Note:** Promising but preliminary results; focus so far has been on developing the framework, not on producing a highly competitive solver.
Empirical Results: Multiple Machines

- 8 machine/8 cores vs. 16 machines/64 cores, clause lengths=1 and 8
- Same executable as for single machine – just different parameters!

![Graph showing empirical results](image)
Summary

- Challenge for the Community: Parallel SAT solvers for 1000+ cores
  - Getting it right and efficient with MPI isn’t easy and/or convenient
  - Exploiting proven expertise/research in parallel system design makes sense!

- **SatX10** provides a Framework for SAT experts to “easily”
  design and test parallel solvers
  - The base classes provided with SatX10 include most of the
    plumbing needed to connect your favorite solver
  - Additional code required focuses mainly on SAT specific details:
    - which clauses to send? when? to whom?
    - when to interrupt a receiving solver to incorporate them?

- Framework available** at: [http://x10-lang.org/satx10](http://x10-lang.org/satx10)