HordeQBF: A Modular and Massively Parallel QBF Solver

SAT 2016, Bordeaux, France

Tomáš Balyo, Florian Lonsing | July 4, 2016

KARLSRUHE INSTITUTE OF TECHNOLOGY, VIENNA UNIVERSITY OF TECHNOLOGY

Definitions

Karlsruhe Institute of Technology

QBFs in prenex CNF (PCNF)

- PCNF $\psi := \hat{Q}.\phi$.
- Quantifier prefix $\hat{Q}$.
- Quantifier-free propositional CNF ϕ (assume: no free variables in ψ).
- A cube is a conjunction of literals. A constraint is a clause or a cube.

$$\psi = \exists x_1 \forall y_1 \exists x_3, x_4.(x_1 \vee y_1 \vee \overline{x_4}) \wedge (\overline{x_3} \vee x_1) \wedge (x_1) \wedge (\overline{y_1} \vee \overline{x_4})$$

QBF Satisfiability

- PSPACE-complete problem.
- Recursively instantiate variables starting at left end of prefix.
- The problem of QBF satisfiability is to determine whether a given PCNF formula is satisfiable.

Introduction

Karlsruhe Institute of Technology

Goal

Design a massively parallel and modular QBF solver that runs well on clusters with **hundreds** of processors

Results

- HordeQBF – new parallel solver based on CDCL for QBF (QCDCL)
- Experiments with application benchmarks with up to 1024 processors
- Significant speedups, especially for hard instances

Parallel QBF Solving

- Explicit Search Space Partitioning
  - classical approach, search space does not overlap
  - each solver starts with a different fixed partial assignment
  - learned constraints (clauses and cubes) are exchanged
  - used in solvers for grids and clusters
- Pure Portfolio
  - modern approach, simple but strong
  - different solver(configuration)s work on the same problem
  - learned constraints are exchanged
  - often used in solvers for multi-core PCs

HordeQBF

- HordeQBF is based on HordeSAT
  - HordeSAT was presented at SAT 2015
  - HordeQBF is obtained by replacing Lingeling with DepQBF
  - The ”Horde framework” is not modified
- DepQBF is treated a black box
  - that can solve QBF problems
  - that can produce learned constraints
  - that can accept learned constraints

Design Principles of Horde[SAT/QBF]

Karlsruhe Institute of Technology

- Modular Design
  - blackbox approach to SAT/QBF solvers
  - any QCDCL solver implementing a simple interface can be used
- Decentralization
  - all nodes are equivalent, no central/master nodes
- Overlapping Search and Communication
  - search procedure (QCDCL solver) never waits for clause exchange
  - at the expense of losing some shared clauses
- Hierarchical Parallelization
  - running on clusters of multi-cpu nodes
  - shared memory inter-node clause sharing
  - message passing between nodes

Modular Design

Portfolio Solver Interface

```
void addClause(vector<int> clause);
SatResult solve(); // {SAT, UNSAT, UNKNOWN}
void setSolverInterrupt();
void unsetSolverInterrupt();
void diversify(int rank, int size);
void addLearnedClause(vector<int> clause);
void setLearnedClauseCallback(LCCallback* clb);
void increaseClauseProduction();
```

Diversification

Native Diversification – ”void diversify(int rank, int size)”

- Each (Q)CDCL solver implements it in its own way
- Example: random seed, restart/decision heuristic

Random Diversification in DepQBF

- Initialization of assignment cache (phase saving).
- Parameters of variable activity scores.
- Percentage of learned clauses/cubes to be removed periodically.
- Parameters of nested restart scheme (cf. PicoSAT).
- Toggle long-distance resolution for clause/cube learning.

Clause/Cube (Constraint) Sharing

Regular (every 1 second) collective all-to-all constraint exchange

Exporting Constraints

- Duplicate constraints filtered using Bloom filters
- Constraint stored in a fixed buffer, when full constraints are discarded, when underfilled solvers are asked to produce more constraints
- Shorter constraints are preferred
- Concurrent Access – constraints are discarded

Importing Constraints

- Filtering duplicate constraints (Bloom filter)
  - Bloom filters are regularly cleared – the same constraints can be imported after some time
  - Useful since solvers seem to ”forget” important constraints

Overall Algorithm

The Same Code for Each Process

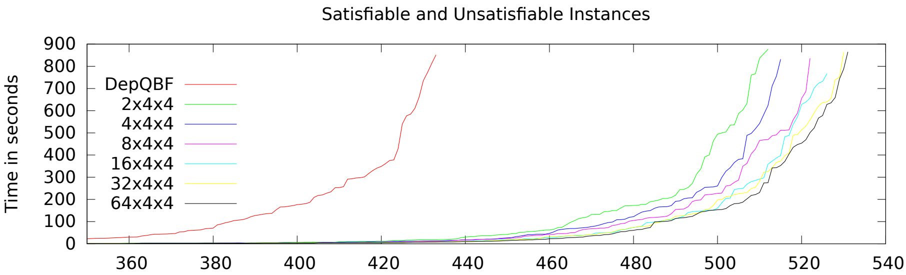
```
SolveFormula(F, rank, size) {
  for i = 1 to #threads do {
    s[i] = new PortfolioSolver(DepQBF);
    s[i].addClauses(F);
    diversify(s[i], rank, size);
    new Thread(s[i].solve());
  }
  forever do {
    sleep(1) // 1 second
    if (anySolverFinished) break;
    exchangeLearnedClauses(s, rank, size);
  }
}
```

Experiments

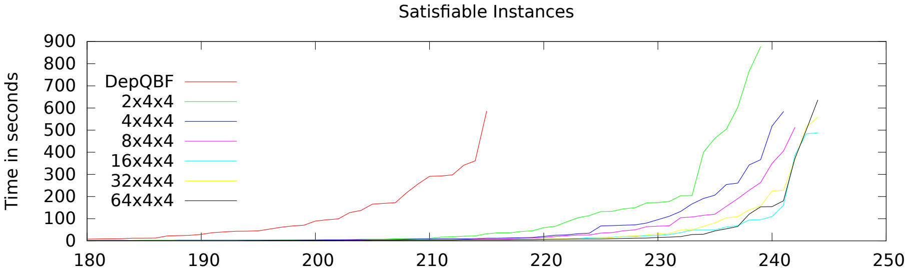
Karlsruhe Institute of Technology

- Benchmarks
  - 2014 QBF Gallery (Competition) Application track instances (735 inst.)
- Computers
  - 64 Nodes of the IC2 cluster
    - each with two octa-core Intel Xeon E5-2670 2.6GHz CPU, 64GB RAM
    - connected by InfiniBand 4X QDR Interconnect
  - In total 128 CPUs and 1024 cores
- Setup
  - Each node runs 4 processes each with 4 threads with DepQBF
  - 1000 seconds time limit (16.7 minutes) for parallel solvers
  - 50000 seconds (13.9 hours) for sequential solvers

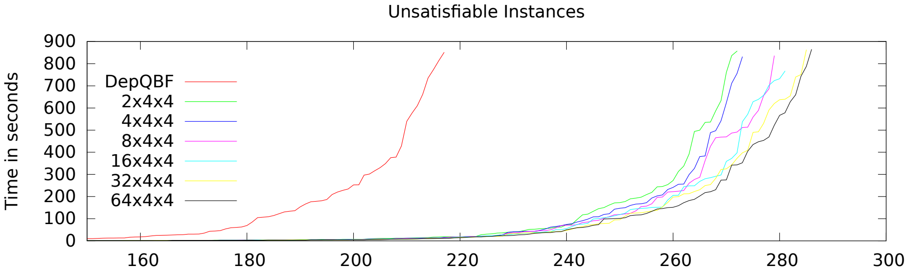
Experiments – All Instances



Experiments – Satisfiable Instances



Experiments – Unsatisfiable Instances



Experiments – Speedups

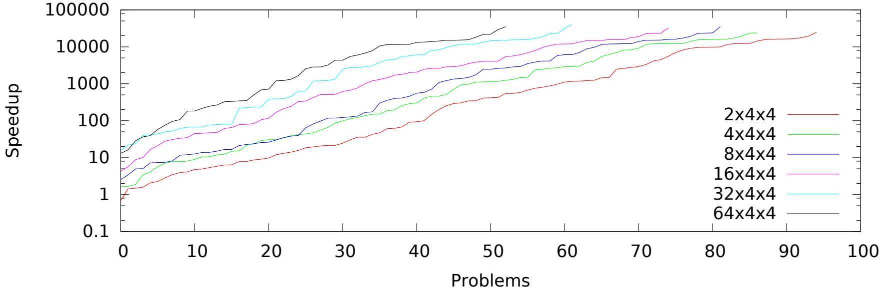
Big Instance = solved after 10 · (#*threads*) seconds by DepQBF

Core Solvers	Parallel Solved	Both Solved	Speedup All Avg.	Speedup All Tot.	Speedup All Med.	Speedup Big Avg.	Speedup Big Tot.	Speedup Big Med.	Speedup Big Eff.
$2\times4\times4$	513	483	622	107.30	0.82	3328	127.36	303.26	9.48
$4\times4\times4$	516	484	667	137.36	0.92	3893	176.27	458.34	7.16
$8\times4\times4$	523	492	748	128.35	0.96	4655	175.26	553.53	4.32
$16\times4\times4$	527	493	754	140.37	0.96	5154	236.18	1449.28	5.66
$32\times4\times4$	531	496	780	132.41	0.96	6282	269.87	2461.84	4.81
$64\times4\times4$	532	496	762	141.99	0.89	6702	307.29	2557.54	2.49

$$\mathit{Eff.} = \mathit{MedianSpeedup} / \#\mathit{threads}$$

Experiments – Speedups on Big Inst.

Big Instance = solved after 10 · (#*threads*) seconds by DepQBF



Conclusion

Karlsruhe Institute of Technology

- HordeQBF is scalable in highly parallel environments.
- Superlinear and nearly linear scaling in average, total, and median speedups, particularly on hard instances.
- Runtimes of difficult QBF instances are reduced from hours to minutes on commodity clusters
  - This may open up new interactive applications