Hochparalleles SAT-Solving

Skalierung paralleler SAT-Solver auf massiv-parallele Systeme

***Dirk Nowotka, Thorsten Ehlers**, AG Zuverlässige Systeme, Institut für Informatik, Christian-Albrechts-Universität zu Kiel*

Kurzgefasst

- SAT-Solving wird als Entscheidungsverfahren in vielen Bereichen wie KI, Konfigurationsmanagement oder Hardware- und Softwareverifikation eingesetzt.
- Im Rahmen des BMBF-Verbundprojekts “Hochparallele Software-Verifikation nebenläufiger Anwendungen in der Automobilindustrie” wird an der massiven Parallelisierung von Methoden der Softwareverifikation geforscht.
- In vielen der dort benutzten Methoden werden SAT-Solver als NP-Orakel eingesetzt.
- Die meisten Arbeiten an SAT-Solvern beschäftigen sich bisher nur mit milder Parallelität auf shared-memory Systemen. In diesem Projekt werden Methoden entwickelt, welche die Skalierbarkeit von SAT-Solvern erhöhen.
- Techniken aus SAT-Solvern werden auch in CP- und SMT-Solvern eingesetzt. Daher bilden Ergebnisse aus diesem Projekt auch eine Grundlage für die Skalierung paralleler Solver für diese Probleme.
- Insbesonderen bei schweren Problemen der beschränkten Modellprüfung (Bounded Model Checking) sind gute Skalierungsergebnisse erzielbar.

Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) fragt, ob zu einer aussagenlogischen Formel eine erfüllende Belegung der Variablen existiert. Frühe Algorithmen wie der DPLL-Algorithmus suchten mittels Backtracking nach solch einer Belegung, und sind daher relativ einfach zu parallelisieren.

Moderne Verfahren implementieren Conflict Driven Clause Learning (CDCL). Bei diesem Verfahren wird ein Backtracking durchgeführt. Stößt der Solver hierbei auf einen Widerspruch, so wird der Grund dafür in eine Klausel kodiert. Diese gelernten Klauseln verhindern, dass der Solver wiederholt ähnliche Teile des Suchraums traversiert. Die Suche nach einer erfüllenden Belegung kann hierdurch massiv beschleunigt werden. Für unerfüllbare Eingabeformeln bilden die gelernten Klauseln eine Resolutionsbeweis für die Unerfüllbarkeit.

Die meisten parallelen SAT-Solver basieren auf dem Portfolioprinzip. Hierbei werden mehrere Solver mit unterschiedlichen Variablenordnungen parallel gestartet, und das Ergebnis des zuerst terminierenden Solvers zurückgegeben. Um die Effizienz dieses Vorgehens zu steigern, können gelernte Klauseln zwischen den einzelnen Solvern ausgetauscht werden. Dieser Informationsaustausch verhindert, dass mehrere der parallel laufenden Solvern redundant im selben Teil des Suchraumes nach Lösungen suchen.

In mäßig parallelen Solvern können Klauseln mit geringer Größe oder niedrigem LBD-Wert [1] von dem lernenden Solver an alle anderen verschickt werden. Da die Menge der ausgetauschten Informationen bei diesem Vorgehen quadratisch mit der Anzahl parallel laufender Solver steigt, ist die Skalierbarkeit limitiert.

In unserem Solver TopoSAT[2] wurde daher die Kommunikation zwischen einzelnen Solver-Prozessen auf Basis von Graphtopologien eingeschränkt. Hierbei entsprachen die Solver-Prozesse Knoten, und ein Informationsaustausch fand nur entlang von Kanten statt. Mit diesem Solver wurden umfangreiche Tests auf einer SGI UV 3000 durchgeführt. Für eine höhere Parallelität ist es erforderlich, die Kommunikation einzuschränken. Eine Möglichkeit ist hierbei die Wahl einer dünner besetzte Topologie. Somit kann das asymtotische Wachstum des Kommunikationsaufwands, je nach verwendeter Kommunikationsstruktur, auf bis zu $\mathcal{O}(n)$ beschränkt werden.

HordeSAT[4] benutzt Kommunikation mittels Broadcasts. Um eine stabile Kommunikation für bis zu 512 Prozesse zu erreichen musste die Menge der von jedem einzelnen Solver versandten Informationen massiv beschränkt werden, was wiederum eine negative Auswirkung auf die Skalierung des Systems hatte.

In diesem Projekt sollen mehrere Fragen über die massive Parallelisierung von SAT-Solvern bearbeitet werden. Die massive Parallelität wird es erlauben, sehr viele Klauseln zu lernen. Von diesen kann entweder ein sehr kleiner Anteil zwischen sehr vielen Solvern ausgetauscht werden, oder aber ein größerer Anteil in einer dünn besetzten Kommunikationsstruktur. Ein globaler Informationsaustausch weniger, aber besonders guter Klauseln liefert bereits gute Ergebnisse.

Im weiteren Projektverlauf soll untersucht werden, wie sich die Skalierung des Solvers verändert, wenn gelernte Klauseln optimiert werden, bevor sie mit an-

deren Solver ausgetauscht werden. Im sequentiellen und mild-parallelen Fall wird hierbei eine Abwägung zwischen dem Zeitaufwand für die Optimierung, und ihrem Nutzen getroffen. Da ausgetauschte Klauseln im massiv-parallelen Fall von sehr vielen Solvern in ihre Datenbanken integriert werden erscheint es hier erfolgsversprechend, mehr Rechenzeit für die Minimierung gelernter Klauseln zu verwenden.

Der Portfolio-Ansatz wird aktuell von den meisten parallelen SAT-Solvern verwendet, und funktioniert in vielen Fällen sehr gut. Die Skalierbarkeit ist allerdings insbesondere auf unerfüllbaren Formeln, welche einen sehr komplexen Beweis für die Unerfüllbarkeit erfordern, eingeschränkt. In diesen Fällen funktioniert eine naive Parallelisierung besser. Für eine Formel F und eine Variable x wird hierbei der Suchraum aufgeteilt, indem F in zwei Formeln $F \wedge x$ und $F \wedge \bar{x}$ aufgeteilt wird. Die Originalformel ist genau dann erfüllbar, wenn mindestens eine dieser Formeln erfüllbar ist. Durch eine rekursive Anwendung dieser Regel und dynamische Lastverteilung kann der Lösungsprozess parallelisiert werden. Erste Experimente zeigten, dass in vielen Fällen dynamisch erkannt werden kann, welcher Lösungsansatz erfolgsversprechender erscheint. Daher soll getestet werden, wie gut ein dynamisches Umschalten zwischen einem Portfolioansatz und einer Partitionierung des Suchraumes in einem massiv-parallelen Kontext funktioniert.

Auf dem HLRN Cluster konnten bereits gute Skalierungsergebnisse für Formeln aus dem Bereich der begrenzten Modellprüfung erzielt werden. In Table 1 sind solche Ergebnisse dargestellt. Die linke Spalte repräsentiert hierbei die Analysegenauigkeit. Für jeden dieser Werte und für Konfigurationen von 96 bis 1536 Threads sind die Laufzeiten unseres Lösers in Sekunden angegeben. Konnte das Problem nicht in 4 Stunden gelöst werden, so wird dies durch "—" dargestellt.

Je höher die Präzision, desto höher fällt der Rechenaufwand aus. Für niedrige Präzisionen ist die Skalierung nicht übermäßig stark, verbessert sich aber mit der Schwierigkeit des Verifikationsproblems. Für Präzisionswerte ab 12 könenn die Formeln sogar nur von Konfigurationen mit einer großen Anzahl von Threads in angemessener Zeit gelöst werden.

Tabelle 1: *Laufzeiten auf sncf-Benchmarks in Sekunden. In der Horizontalen ist die Anzahl der Abrollschritte angegeben, in der Vertikalen die Anzahl verwendeter Threads.*

Präzision	Anzahl Threads				
	96	192	384	768	1536
7	250	198	163	137	123
8	414	319	247	219	187
9	882	496	387	347	269
10	1680	932	581	456	376
11	10574	5641	3149	1960	1360
12	—	9890	5765	3439	2064
13	—	—	9419	5422	3166
15	—	—	—	11845	6749

WWW

http://www.zs.informatik.uni-kiel.de

Weitere Informationen

[1] G. Audemard, L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers", *International Joint Conference on Artificial Intelligence* (IJCAI), 2009. http://ijcai.org/Proceedings/09/Papers/074.pdf

[2] T. Ehlers, D. Nowotka, P. Sieweck, "Communication in Massively-Parallel SAT Solving", *International Conference on Tools with Artificial Intelligence* (ICTAI), 2014. doi:10.1109/ICTAI.2014.111

[3] T. Ehlers, P.J. Stuckey, "Parallelizing Constraint Programming with Learning", *Integration of AI and OR Techniques in Constraint Programming* (CPAIOR), 2016. doi:10.1007/978-3-319-33954-2_11

[4] T. Balyo, P. Sanders, C. Sinz, "HordeSAT: A Massively Parallel Portfolio SAT Solver", *Theory and Applications of Satisfiability Testing* (SAT), 2015. doi:10.1007/978-3-319-24318-4_12

[5] R. Nieuwenhuis, A. Oliveras, C. Tinelli: "Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)", *J. ACM*, 53(6), 937–977, (2006). doi:10.1145/1217856.1217859

[6] C. M. Wintersteiger,Y. Hamadi, L. M. de Moura: "A Concurrent Portfolio Approach to SMT Solving", *Computer Aided Verification* (CAV), 2009. doi:10.1007/978-3-642-02658-4_60

[7] T. Feydy, P. J. Stuckey: "Lazy Clause Generation Reengineered", *Principles and Practice of Constraint Programming* (CP), 2009. doi:10.1007/978-3-642-04244-7_29

Projektpartner

AG Programmiersprachen und Übersetzerbau, Christian-Albrechts-Universität zu Kiel;
Konrad-Zuse-Zentrum für Informationstechnik Berlin;
Model Engineering Solutions GmbH;
Symtavision GmbH

Förderung

BMBF Verbundprojekt HPSV, Förderkennzeichen 01IH15006A