Look-ahead alapú SAT solver-ek párhuzamosíthatóságának vizsgálata

Biró Csaba[1], Kovászani Gergely[2], Tajti Tibor[1], Kusper Gábor[1], Geda Gábor[1]

[1] Eszterházy Károly Főiskola

{birocs, tajti, gkusper, gedag@aries.ektf.hu}

[2] Johannes Kepler University

{Gergely.Kovasznai@jku.at}

Absztrakt

A SAT probléma logikai formulák kielégíthetőségét vizsgálja. Elvárás, hogy a logikai formula konjunktív normál formában legyen. Ekkor, ha a formula n atomból (logikai változóból) áll, akkor legrosszabb esetben 2^n próbálkozásból megállapítható, hogy kielégíthető-e a formula. A SAT probléma jól ismert NP-teljes probléma. Cikkünkben bemutatjuk a CCGrid-et (Cube and Conquer on Grid), amely egy általunk készített grid alapú SAT solver. A CCGrid két fő komponensből áll. A master-en egy szekvenciális look-ahead solver (march_cc) fut, amely particionálja a problémát, majd elkészíti az egyes munkaegységeket. A klienseken, az elkapott munkaegységeket egy párhuzamosított CDCL SAT solver (iLingeling) dolgozza fel. Ezen elosztott számítások elvégzéséhez a BOINC middleware-t, fejlesztéshez pedig a SZTAKI által fejlesztett DC-API-t használtuk. A CCGrid-del csak kielégíthető problémák esetében sikerült sebességnövekedést elérünk, a rendszer egyik gyenge pontjának a look-ahead solver bizonyult. Cikkünkben megvizsgáljuk CCGrid gyorsításának lehetőségeit.

Kulcsszavak: grid, SAT, párhuzamos SAT solver, lookahead, march_cc, iLingeling,

SZTAKI Desktop Grid, BOINC, DC-API

Austro-Hungarian Action Foundation által támogatott.

Project ID: 83öu17

Bevezetés

A logikai kielégíthetőség (satisfiability) problémája alatt azt értjük, hogy valamely 0.-rendű logikai formula változóihoz olyan hozzárendelést keresünk, amely mellett a formula igaz. SAT problémáról beszélünk, ha a formula speciálisan konjunktív normál formában. A SAT probléma NP teljes[1], azaz, nem ismert polinomiális idejű algoritmus, amely megoldaná. A SAT probléma az informatika számos (logika, mesterséges intelligencia, áramkör tervezés) területén napjainkban is intenzíven kutatott téma.

A mai modern CDCL SAT solver-ek [*pl.* glucose, SATzilla, Lingeling, Minisat, ppfolio2012, PrecoSAT] a DPLL [2] algoritmuson alapulnak. A DPLL egy teljes, visszalépéses keresésen alapuló algoritmus, SAT kielégíthetőségi problémára. A formula szemantikus fájában keres, képes a kielégíthetetlen formulák detektálására, hiszen amennyiben nem talál megoldást bejárja a teljes keresési fát. Legnagyobb előnye, hogy az összes változó kiértékelése előtt jó eséllyel megtalálja a megoldást. CDCL (Conflict-driven clause learning) azon elven alapul, hogy a konfliktusok felderítésével csökkenthető a keresési tér. Amennyiben talál egy konfliktust, elugrik abba az ágba ahol az gyökerezik, majd elemezni a konfliktust (elégséges feltétel keresése - bizonyítás), ezután metszi azokat a területeket a keresési térben, ahol ez a feltétel teljesül. Természetesen a modern solver-eben egyéb heurisztikákat is alkalmaznak. Ilyen például a DLIS (Dynamic Largest Individual Sum), amely megszámolja azon kielégíthetetlen klózok számát amelyikben egy adott változó megjelenik, a változókhoz különböző értékeket rendel. Mindez jelentős erőforrást igényel, de a statisztika fenntartása szükséges a heurisztikához. Egy másik fontos heurisztika a VSIDS (Variable State Independent Decaying Sum), amely rangsorolja a változókat literálokban történő előfordulási számuk alapján (kezdeti adatbázis), tanulás útján új klózokat ad hozzá a kezdeti adatbázishoz. Annak érdekében, hogy kevesebb osztály legyen időnként az összegeket elosztja egy konstanssal. Az első look-ahead alapú solver a posit [1995]. A mai modern look-ahead alapú solver-ek [Satz, march, OKSolver, kcnfs] tulajdonképpen egy DPLL alapú solver kiegészítve „előretekintéssel”. Előretekintés célja a lehető legnagyobb csökkenést okozó, elágazási változó megtalálása, amely költéses, ezért a SAT

versenyeken is nem szerepelnek túl jól. Nagy előnyük viszont, hogy segítésükkel jól lehet particionálni a problémát.

SAT solver-ek párhuzamosításra kétféle megközelítés létezik. Egyik a jól ismert „Oszd meg és uralkodj elv", melynek értelmében inkrementálisan szétosztják a keresési teret alterekre, ahol sorra szekvenciális DPLL alapú solver-ek allokálódnak. Az egyes solver-ek kommunikálnak, együttműködnek és dinamikusan megosztják a tudásbázisuk(tanult klózok) egymással.

A másik lehetőség a párhuzamos portfólió (2008). Itt különböző szekvenciális DPLL alapú stratégiákat versenyeztetnek egymással ugyanazon a formulán. Mivel mindegyik ugyanazon a formulán dolgozik, nincs szükség terhelés-kiegyenlítés bevezetésére. A kommunikáció csupán a tanult klózok megosztására szorítkozik. Célja, hogy a keresési téren versengő stratégiák közül megtalálni a legjobbat. Fontosabb state-of-the-art portfólió alapú párhuzamos SAT solver-ek
(pl. ManySAT ppfolio, pfolioUZK, SATzilla)
.

GridSAT[3] volt az első párhuzamosított, teljes SAT solver grid környezetben. Alapja a zChaff szekvenciális solver. Jelentős sebességnövekedést értek el mind kielégíthető és mind kielégíthetetlen SAT problémák esetében. Legnagyobb nehézséget a tanult klózok kezelése okozta.

SAT problémák párhuzamosításának nehézségei [5]

Mivel a SAT probléma eredendően szekvenciális, ezért nehezen párhuzamosítható. SAT problémák párhuzamosításkor felmerülő kérdések:

- Hogyan tudjuk hatékonyan megoldani multi-core rendszereken?
- Milyen adatszerkezetekre, heurisztikákra, és alacsony szintű folyamatokra van szükség?
- Milyen kommunikáció kell alkalmazni?
- Hogyan tudjunk összehasonlítani a párhuzamos SAT solver-ek teljesítményét?

Általánosan elfogadott alapelv, hogy osszuk szét a keresési teret, az egyes altereken indítsunk el egy SAT solver-t. De ekkor újabb kérdésekkel kell szembe néznünk:

- Hogyan osszuk szét a teret?
- Használjunk un. döntési változókat osztó pontoknak! De melyiket?
- Melyik folyamat vagy szál?
- Hogyan lehet hatékonyan kommunikálni, irányítani, szinkronizálni?
- Hogy történjen a konfliktusba került záradékok és tanult klózok megosztása ?

CCGrid

CCGrid[4],[6] (Cube and Conquer on Grid), amely egy grid alapú SAT solver. Két fő komponensből áll.

A master-en egy szekvenciális look-ahead solver (march_cc) fut, amely particionálja a problémát és elkészíti az egyes munkaegységeket. A munkaegységek tartalmazzák az eredeti input.cnf fájlt illetve a feltételezéseket tartalmazó fájl egy - a problémától és a kliensek számától függő - szeletét. Mivel nincs beállítva rögzített vágási mélység, egy adott probléma esetében több ezer, vagy akár több millió feltételezést is létrehozhat.

A klienseken, az elkapott munkaegységeket egy párhuzamos CDCL SAT solver (iLingeling[8]) dolgozza fel, amely a feltételezések alapján bizonyítja az eredeti CNF kielégíthetőségét (SAT/UNSAT). Az iLingeling egy párhuzamos CDCL solver, amely minden szálon futtat egy különálló lingeling példányt. Bemenetként egy un. iCNF fájlt vár, amely legalább egy feltételezést tartalmaz.

1. ábra. CCGrid architektúrája

Eredmények és tesztelési környezet

Teszteléshez, egy SUN szerverből (4 mag, 6 GB memória), és 20-30 kliensből (2-4 mag, 2-4 GB memória) környezetet hoztunk létre. Eredményeinket az alábbi táblázat tartalmazza. A CCGrid-del csak kielégíthető problémák esetében sikerült sebességnövekedést elérünk, a rendszer egyik gyenge pontjának a look-ahead solver bizonyult.

CCGrid gyorsításának lehetőségei

A CCGrid gyorsításának érdekében, az alábbi lehetőségeket vizsgáltuk:

a) A march_cc -t álljon meg hamarabb, így nem vizsgálja olyan mélyen a problémát, gyorsabb lesz. Ekkor, azonban az így előállt feltételezésekkel, a klienseknek nehezebb lesz a bizonyítás.

Annak érdekében, hogy az egyes kliensek, egy-egy részproblémán ne ragadjanak be, a munkaállomások architektúrájának (CPU, memória) függvényében az lingeling-et megállítjuk *X konfliktus/idő* után.

Ez után az iLingeling, a be nem bizonyított feltételezésekre, egyenként meghívja a

march_cc-t. Ezen a ponton érdekes lenne, az alábbi két lehetőséget megvizsgálni:

1. az új feltételezéseket a kliens visszaküldi a master-nek, ami ezután munkaegységek formájában szétosztja azokat a kliensek között,
2. az új feltételezéseket a kliens megtartsa magának, és csak az analizált, rövid konfliktusba került klózokat küldi át a master-nek.

b) A march_cc CPU/GPU alapú párhuzamosítása.

CPU-ra kiosztandó feladatok:

1. párhuzamos Unit propagáció,
2. párhuzamos failed literal keresés.

GPU-ra kiosztandó feladatok:

1. masszívan párhuzamos lokális keresési algoritmus,
2. random walk algoritmus.

Felhasznált irodalom

[1] S. A. Cook., The Complexity of Theorem-Proving Procedures. Proc. of STOC'71, pp. 151-158, 1971.

[2] M. Davis, G. Logemann, D. Loveland., A Machine Program for Theorem Proving. Commun. ACM, vol. 5, no. 7, pp. 394-397, 1962.

[3] W. Chrabakh, R. Wolski, GridSAT: A Chaff-based Distributed SAT Solver for the Grid. Proc. of SC'03, pp. 37-49, 2003.

[4] Csaba Biró, Gergely Kovásznai, Gábor Kusper, Gábor Geda and Armin Biere , Cube-and-Conquer Approach for SAT Solving on Grids Technical Report

[5] Biró Csaba, Kusper Gábor, Geda Gábor, Grid alapú SAT-solverek vizsgálata Networkshop 2012, Veszprém

[6] Marijn Heule, Oliver Kullmann, Siert Wieringa and Armin Biere , Cube and Conquer Guiding CDCL SAT Solvers by Lookaheads Belgrade, February 3, 2012

[7] A. Biere, M. Heule, H. van Maaren, T. Walsh, Handbook of Satisfiability. IOS Press, Amsterdam, 2009.

[8] A. Biere, Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical Report 10/1, FMV Reports Series, JKU, 2010.

[9] P. Kacsuk, J. Kovács, Z. Farkas, A. C. Marosi, G. Gombás, Z. Balaton, SZTAKI Desktop Grid (SZDG): A Flexible and Scalable Desktop Grid System. Journal of Grid Computing, vol. 7, no. 4, pp. 439-461, 2009.