Resolution and parallelizability: barriers to the effficient parallelization of SAT solvers

George Katsirelos, Ashish Sabharwal, Horst Samulowitz, Laurent Simon

Resolution and Parallelizability: Barriers to the Efficient Parallelization of SAT Solvers

George Katsirelos
MIAT, INRA
Toulouse, France
george.katsirelos@toulouse.inra.fr

Ashish Sabharwal
IBM Watson Research Center
Yorktown Heights, NY 10598, USA
ashish.sabharwal@us.ibm.com

Horst Samulowitz
IBM Watson Research Center
Yorktown Heights, NY 10598, USA
samulowitz@us.ibm.com

Laurent Simon
Univ. Paris-Sud
LRI/CNRS UMR8623
Orsay, F-91405, France
simon@lri.fr

Abstract

Recent attempts to create versions of Satisfiability (SAT) solvers that exploit parallel hardware and information sharing have met with limited success. In fact, the most successful parallel solvers in recent competitions were based on portfolio approaches with little to no exchange of information between processors. This experience contradicts the apparent parallelizability of exploring a combinatorial search space. We present evidence that this discrepancy can be explained by studying SAT solvers through a proof complexity lens, as resolution refutation engines. Starting with the observation that a recently studied measure of resolution proofs, namely depth, provides a (weak) upper bound to the best possible speedup achievable by such solvers, we empirically show the existence of bottlenecks to parallelizability that resolution proofs typically generated by SAT solvers exhibit. Further, we propose a new measure of parallelizability based on the best-case makespan of an offline resource constrained scheduling problem. This measure explicitly accounts for a bounded number of parallel processors and appears to empirically correlate with parallel speedups observed in practice. Our findings suggest that efficient parallelization of SAT solvers is not simply a matter of designing the right clause sharing heuristics; even in the best case, it can be — and indeed is — hindered by the structure of the resolution proofs current SAT solvers typically produce.

Introduction

The trend in hardware design has lately been to move away from improving single thread performance and instead towards increasing parallelism. In order for the combinatorial reasoning and optimization community to fully exploit the capabilities of modern hardware, the algorithms we develop must be efficiently parallelizable.

In propositional satisfiability or SAT, the most commonly used sequential algorithms ("solvers") in application domains are based on a set of techniques collectively referred to as Conflict Driven Clause Learning (CDCL). This entails resolution-based clause learning, aggressive restarting, activity based branching heuristics, and efficient data structures for storing and propagating clauses. These solvers perform very well sequentially. Their performance has also been theoretically analyzed using tools from the field of proof complexity (Cook and Reckhow 1979), specifically using proof systems based on the resolution principle (Robinson 1965). While it had been long understood that the operation of CDCL solvers can be viewed as generating resolution refutations, a series of papers (Beame, Kautz, and Sabharwal 2004; Buss, Hoffmann, and Johannsen 2008; Hertel et al. 2008; Pipatsrisawat and Darwiche 2011) analyzed the connection more deeply and found, for instance, that given the freedom to restart often enough, these solvers are in fact exactly as powerful as general resolution.

Our work is motivated by the fact that effectively parallelizing CDCL SAT solvers has been found over the years to be an extremely difficult task, as evidenced by a recent Challenge Paper on this topic (Hamadi and Wintersteiger 2012). There have undeniably been many advances in parallel SAT solving, particularly in the area of clause sharing heuristics. Yet the net efficiency remains rather disappointing. For example, in SAT Competition 2011 (Järvisalo, Le Berre, and Roussel 2011), the most recent such event with a special track for 32-core solvers, the average speedup achieved by top performers such as CRYPTOMINISAT-MT (Soos 2012) and PLINGELING (Biere 2012) was merely a factor of about 3. With modern day hardware providing access to 10,000 and even 100,000 cores, the challenge of scale becomes even greater. As further evidence, the winners in the regular parallel track of the past two SAT Competitions, PPFOLIO (Roussel 2012) and PFOLIOUZK (Wotzlaw et al. 2012), were portfolio-based solvers that simply launch multiple individual (mainly CDCL) solvers with little to no communication between them. The derivation of the underlying resolution refutation itself is not significantly parallelized. If this latter aspect is an obstacle to effective parallelization (as we show), non-information sharing portfolios cannot help us scale well to even a few dozen cores.

In this paper, we initiate a new kind of study that explores *how the proof system underlying state-of-the-art SAT solvers affects the degree to which one may expect parallelization — using current methods — to succeed in practice*. As the SAT and larger CSP and AI communities explore various ways to address the challenge of parallelizability such as new clause sharing heuristics, decompositions, speeding up unit propagation, etc., our goal is to provide a novel perspective that we hope will shed light into the current state of affairs and spark a new set of ideas to tackle the problem.

Preview of Results

At a high level, we find that the resolution proof system itself poses barriers to parallelizability. Our theoretical study begins with an exploration of the *depth* of resolution refutations, which captures the longest chain of dependent resolution steps that must be performed starting from input clauses to the empty clause. Since this is a chain of dependencies, the inverse of depth provides a first upper bound on the speedup achievable in deriving this refutation even with "infinite" processors. To make this bound relevant to CDCL solvers, we employ a notion we call conflict resolution depth. Our experiments suggest that proofs of unsatisfiability of instances with large conflict resolution depth parallelize poorly.

The bound resulting from the notion of depth is, however, weak, in part because depth does not account for the availability of only a bounded number k of processors. Therefore, we propose a refined measure based on recasting the derivation of a proof as *resource constrained scheduling with precedence constraints*. The resulting refined measure, namely the "best case" schedule makespan, converges to proof depth as k increases. For small k, however, it correlates much better with the speedup observed in practice.

Our first empirical finding is that the "shape" or structure of refutations produced by state-of-the-art SAT solvers contains surprisingly narrow bottlenecks (Figure 1) that intuitively impose sequentiality, thus hindering parallelization. Further, we find that these sequential refutations, at least for "industrial" instances, are often indeed not parallelizable beyond a few dozen processors — even in the best case of "offline" scheduling (Figures 3(a) and 3(b)).

Why should the degree of parallelizability of sequential proofs be of interest? To address this, we provide the following somewhat surprising empirical evidence (Figure 4): The speedup observed in practice on an unsatisfiable instance I when using current methods to parallelize a CDCL solver S to k processors correlates well with the degree to which the *sequential* refutation of I found by S can be parallelized by scheduling it on k processors, (which, in light of the above findings, is often very limited). In other words, the makespan of a k-processor schedule of a sequential proof of I provides insight into how parallelizable S will be on I.

We remark that while our empirical results are for unsatisfiable instances, time complexity conclusions drawn from the analysis of unsatisfiability proofs do often transfer to satisfiable instances as well (Achlioptas, Beame, and Molloy 2001; 2004) because finding a solution often involves first proving unsatisfiability of many parts of the search space. Further, speedups on unsatisfiable instances are more consistent and predictable from execution to execution, which makes empirical results more robust.

In summary, our findings suggest that building an effective parallel CDCL solver may be harder than previously thought. The efficiency of such solvers is hindered by the structure of the proofs they produce. Overcoming this barrier may require fundamental changes to the CDCL algorithm that go beyond the emphasis often placed on which clauses to share, how to decompose, or how to speed up unit propagation (Hamadi and Wintersteiger 2012).

Background

The propositional satisfiability problem (SAT) is to determine, for a given propositional formula F, whether there exists an assignment to its variables that satisfies it. A formula in Conjunctive Normal Form (CNF) is expressed as a conjunction of disjunctions of literals. Each disjunction is called a clause. We often refer to a clause as a set of literals and to a CNF formula as a set of clauses.

Resolution (Robinson 1965) is a sound and complete system for reasoning about CNF formulas which consists of the single rule that $(A \vee x) \wedge (B \vee \overline{x}) \vdash A \vee B$, where A and B are disjunctions of literals. We call $A \vee B$ the resolvent of $A \vee x$ and $B \vee \overline{x}$. A CNF formula F is unsatisfiable if and only if the empty clause ($\square$) can be derived using a sequence of resolution steps starting with the clauses of F. More formally, a resolution refutation Γ_r of F is a sequence of clauses $C_1, \ldots, C_m$ such that $C_m = \square$ and for each $i \in [1, m]$, $C_i \in F$ or C_i is the resolvent of two clauses C_j, C_k, $j, k \in [1, i-1]$. Since we study no other systems here, we may omit the word "resolution" and call this a refutation. We say that the length of Γ_r is m. We denote the shortest possible refutation of F by $\Gamma_r^{\min}(F)$.

Given a refutation Γ_r of a CNF formula F, we can define a directed acyclic graph (DAG) called the resolution DAG as follows. There exists a vertex for each unique clause in Γ_r. If C_i is the resolvent of C_j and C_k, there exist edges (C_j, C_i) and (C_k, C_i).

Unit propagation is the following simplification process for CNF formulas. If there exists in F a clause C of size 1 (called a unit clause), say $\{x\}$, we remove from F all clauses that contain x and remove $\overline{x}$ from all clauses that contain it. Since this may generate new unit clauses, we repeat this until fixpoint or until $\square \in F$.

Most of the current successful SAT solvers are based on the Conflict Driven Clause Learning (CDCL) framework; we refer the reader to Biere et al. (2009) for an overview. CDCL solvers alternate branching, i.e., tentatively adding a unit clause to the formula, and performing unit propagation until, after some branching choices, unit propagation generates the empty clause. At that point, a conflict analysis procedure uses the trace of unit propagation to extract a series of resolution steps using the clauses of F and derive a new clause L. The solver then adds L to F and repeats the procedure, either starting with a fresh sequence of branching decisions (a restart), or by reusing a prefix of the current sequence branching decisions such that unit propagation does not derive the empty clause (a backjump). The search terminates when the empty clause is derived, in which case F is unsatisfiable, or until the sequence of branching decisions and unit propagation derive the empty formula, in which case it can extract a solution. We omit discussion of other aspects of CDCL solvers, such as data structures, preprocessing and choice of branching and other heuristics.

Conflict Resolution Depth

To motivate the study of resolution depth, we present the following experiment. We modified the CDCL SAT solver GLUCOSE 2.1 (Audemard and Simon 2009) to produce a file

containing a refutation Γ (termed a *CDCL refutation*) of the input instance F, assuming F is unsatisfiable. The format of this file is such that for each derived clause C (including the empty clause), we can reconstruct all resolution steps involved in the conflict analysis resulting in the derivation of C, including minimization. This associates with C all input or previously derived clauses needed for deriving C. We then remove any clauses that cannot reach the empty clause. All remaining clauses form the CDCL refutation Γ and are annotated with a *conflict resolution depth* attribute, defined as follows:

Definition 1. *Let F be an unsatisfiable CNF formula and Γ a CDCL refutation of F. For every clause $C \in \Gamma$, the* conflict resolution depth *of C, denoted $d_{conf}(C, \Gamma)$, is:*

1. *0 if $C \in F$, i.e., C is an input clause, and*
2. *$1 + \max_{i=1}^{k}\{d_{conf}(C_i, \Gamma)\}$ if C is derived in Γ by resolving clauses $C_1, \ldots, C_k$.*

The conflict resolution depth of the refutation Γ, *denoted $d_{conf}(\Gamma)$, is the maximum of $d_{conf}(C, \Gamma)$ over C in Γ. The* conflict resolution depth of F *itself, denoted $d_{conf}(F)$, is the minimum of $d_{conf}(\Gamma)$ over all CDCL refutations Γ of F.*

In other words, we implicitly construct a directed acyclic proof graph $G_{conf}(\Gamma)$ from Γ with one vertex for each clause C and incoming edges from $C_1, \ldots, C_k$ to C. Then, $d_{conf}(C, \Gamma)$ equals the length of the longest path in $G_{conf}(\Gamma)$ from an input clause to C. We call $G_{conf}(\Gamma)$ the *conflict resolution DAG*, in contrast to the resolution DAG.

We use $|\Gamma|$ to denote the *size* of the CDCL refutation measured as the number of derived clauses in Γ. In the rest of this paper, we will use $|\Gamma|$ as a proxy for the amount of computational resources consumed taken by a sequential SAT solver when deriving Γ, and will refer to it as the number of "steps" taken by the solver. For a parallel SAT solver, the resource consumption will be appropriately adjusted to take into account the derivation of multiple clauses in each "parallel" step. The *step speedup* of a solver S_1 compared to S_2 will then be the ratio of the number of sequential or parallel steps, as appropriate, needed by the two solvers. We will use the term *speedup* (or *parallel speedup*) to denote the ratio of the actual (wall clock) runtimes of the two solvers.

Bottlenecks in Typical CDCL Proofs

Given this setting, we consider a "proof shape" plot of the number of clauses discovered at each conflict depth. Conflict depth is plotted on the X axis, with 0 on the left, and the number of clauses at that conflict depth on the Y axis. The lone clause at maximum conflict depth is the empty clause.

Figure 1 shows a typical proof shape, plotted here for `cmu-bmc-longmult15`, an unsatisfiable model checking instance solved in a few seconds. Plots for other instances are not substantially different. We observe in this plot the existence of deep valleys (i.e., levels such that very few clauses are annotated with it) followed immediately by a peak (i.e., a level with many clauses). Although this may not be apparent from the plot, the size of most of the deep valleys is 1. This means that there exists a *single* clause, of small size in many cases, that is derived at that conflict depth. These

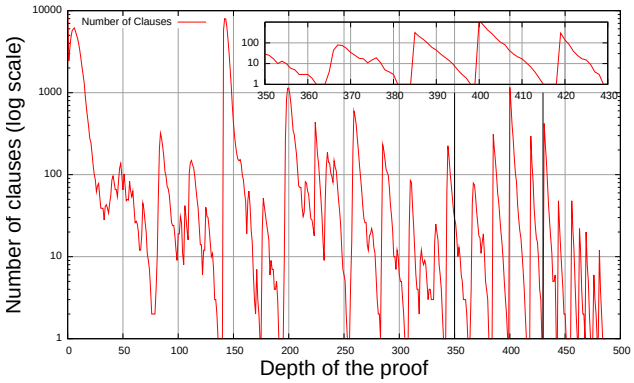


Figure 1: Number of clauses derived at each depth of a typical CDCL proof

deep valleys imply a *bottleneck in the proof*: all clauses discovered at a conflict depth greater than the conflict depth of the valley depend on that single (key) clause. This has an effect on the parallel speedup we could achieve in deriving this particular refutation: until the clause at the first valley is discovered, no processor can progress to a deeper level; the same holds for the second valley, and so on. It may thus be important to detect these key clauses for sharing. Moreover, if it seems that we could detect these clauses by their activity peak in conflict analysis, a good parallel solver would also join the efforts of its threads to produce as soon as possible these key clauses, in the best possible order. Today's parallel SAT solvers do not consider such aspects.

Depth Based Limits to Parallelization

We can formalize the intuition underlying this plot by studying the notion of *resolution depth* of a formula, defined as the minimum over all resolution refutations of the formula of the length of the longest path from an input clause to the empty clause in the corresponding resolution refutation DAG. Formally, given a CNF formula F, a resolution refutation (rather than a CDCL refutation) Γ of it, and a clause $C \in \Gamma$, we can define resolution depths $d_{res}(C, \Gamma)$, $d_{res}(\Gamma)$, and $d_{res}(F)$ in a manner analogous to Definition 1. Note that resolution depth is not to be confused with conflict resolution depth, which is measured on the conflict resolution DAGs generated by CDCL solvers. The two DAGs are, however, related, as the conflict resolution graph simply collapses sequences of unit resolution steps into one step. In particular:

Observation 1. *For any formula F over n variables, $d_{conf}(F) \leq d_{res}(F) \leq n \times d_{conf}(F)$.*

Given that unit propagation is an integral part of clause learning SAT solvers, the measure we are really interested in is conflict resolution depth. Resolution depth serves only as a proxy for it.

The depth of a resolution proof provides a hard upper bound on the step speedup we can achieve by parallelizing the derivation of that refutation using resolution. Indeed, even with an unbounded number of parallel processors and

no communication cost, in the first step we can at best derive all clauses at depth 1. Each clause at depth k has at least one ancestor at level $k-1$ and can therefore not be derived earlier than step k. As we will discuss in the next section, Urquhart (2011) proved that there exist families of instances F_n over n variables such that $d_{res}(F_n) = \Omega(n/\log n)$, i.e., *every* resolution proof of F_n must have depth at least roughly linear in n. This already implies a lower bound of $\Omega(n/\log n)$ on the number of parallel steps needed by any algorithm that performs only one resolution inference per processor in each time step, even with an unbounded number of processors.

This suggests that a theoretical study of barriers to parallelizability is of interest. Unfortunately, instances in the family $\{F_n\}$ proposed by Urquhart are decidable by unit propagation and thus their conflict resolution depth, the notion more applicable to CDCL SAT solvers, is only one. The lower bound argument can, however, be made more relevant by extending it to conflict resolution depth:

Proposition 1. *Let F be a CNF formula and S_k be a parallel CDCL SAT solver using k processors. Let $\mathcal{R}$ denote the set of all CDCL refutations of F. Then, for any k,*

1. *S_k needs at least $d_{conf}(F)$ parallel steps to solve F.*
2. *Let Γ be a CDCL refutation of F produced by a solver S_1 using 1 processor. Compared to S_1, S_k cannot achieve a step speedup larger than $|\Gamma|/d_{conf}(F)$ when deriving Γ.*
3. *S_k cannot achieve a step speedup of more than $\min_{\Gamma\in\mathcal{R}} |\Gamma|/d_{conf}(F)$ compared to an optimal sequential CDCL solver for F.*

Instances with Large Conflict Resolution Depth

Given a DAG G, the *pebbling game* on G is a well-studied single-player game defined as follows. At each step, the player can place a pebble on a source vertex (referred to as "pebbling" the vertex), remove a pebble from any vertex, or, if all ancestors of a vertex v are pebbled, either place a pebble on v or move a pebble from one of v's predecessors to v. The minimum number of pebbles needed to pebble one of the sinks of G is $p(G)$, the *pebbling number* of G. (Note that the number of vertices of G is a trivial upper bound on $p(G)$.) It suffices for our purposes to assume a fanin of two and a single sink.

Kozen (1977) introduced the corresponding *pebbling formulas* $Peb(G)$ with one propositional variable for every vertex capturing whether or not that vertex is pebbled. The clauses of $Peb(G)$ are: v_s for every source vertex s, $\neg v_t$ for the sink vertex t, and $(\neg v_1 \vee \neg v_2 \vee v)$ for every vertex v with predecessors v_1 and v_2. For these formulas, Urquhart (2011) recently showed that resolution depth is tightly related to the pebbling number: $d_{res}(Peb(G)) = p(G) - 2$. Further, he gave a particular family of instances derived from pebbling games on graphs G_n with n vertices and pebbling number (and hence resolution depth) as high as $\Omega(n/\log n)$.

Unfortunately, $Peb(G_n)$ is solvable by unit propagation and therefore $d_{conf}(Peb(G_n)) = 1$. To avoid this behavior, we use a construction introduced by Ben-Sasson and Nordström (2008). They define *XOR-pebbling formulas* $Peb_{\mathrm{XOR}}(G)$ in which there exist two variables v and v' for each vertex v and the axioms are changed to the following: $v_s \oplus v'_s = 0$ for every source vertex v_s, $v_t \oplus v'_t = 1$ for the sink v_t, and $(v_1 \oplus v'_1 = 0) \wedge (v_2 \oplus v'_2 = 0) \rightarrow (v \oplus v' = 0)$ for every vertex v with predecessors v_1 and v_2. These axioms can be encoded with clauses of size at most 6. It is easy to show that Urquhart's result generalizes to XOR-pebbling formulas. Further, it can be shown that unit propagation is ineffective on XOR-pebbling formulas. Formally:

Theorem 1. *For any DAG G, $d_{res}(Peb_{\mathrm{XOR}}(G)) = 2p(G) - 4$ and $d_{conf}(Peb_{\mathrm{XOR}}(G)) = 2p(G) - 4$.*

This allows us to use families of graphs with a known pebbling number to construct instances with a specific depth, which are not trivially solvable, and can in fact be challenging for CDCL solvers (Järvisalo et al. 2012). The instances are, however, easy for variable elimination.

Conflict Resolution Depth and Parallelizability: An Empirical Evaluation

In order to evaluate whether resolution depth or conflict resolution depth is a robust measure for capturing the degree of parallelizability of an instance by a CDCL SAT solver, we report the results of two experiments. All experiments were conducted on AMD Opteron 6134 machines with 32 cores, 64 GB RAM, and 512 KB L2 cache, running at 2.3 GHz.

In both experiments, we measure the speedup or degree of parallelizability observed in practice by comparing the (wall-clock) time needed by the sequential solver GLUCOSE 2.1 compared to our implementation of a parallel version of it, which we call GLUSATX10 (k) when using k processors, built using the SATX10 parallelization framework (Bloom et al. 2012). When launched, GLUSATX10 (k) starts k copies of GLUCOSE (on the same machine, for the purposes of this paper) with different configurations such as random seed, a small fraction of random branching decisions, LBD queue length, etc. These sequential copies share learnt clauses of up to a maximum length, which is dynamically adjusted such that roughly 5% of the clauses learned by each solver are shared with others. Other default features of SATX10, such as clause buffering, are kept unchanged.

In the first experiment, we consider instances with a known lower bound on resolution depth, namely, a subset of instance families recently used to study the notion of "space" of resolution proofs (Järvisalo et al. 2012). In particular, we use XOR-pebbling instances on families of graphs with known pebbling number. We expect that as the ratio $\Gamma_r^{\min}(F)/d_{res}(F)$ increases within a family of graphs, so does the speedup going from GLUCOSE to GLUSATX10. Unfortunately, this hypothesis was not confirmed (data omitted for lack of space). However, CDCL solvers perform poorly on these instances, because the pebbling number of a graph is also a lower bound for resolution space, which correlates with difficulty (Järvisalo et al. 2012). Hence, it is possible that this distorts our results.

In the second experiment, we consider all 300 unsatisfiable instances from SAT Challenge 2012 (Balint et al. 2012) to the subset of 164 instances for which CDCL proof files can be generated using our modified variant of GLUCOSE within 5,000 seconds. From those instances we remove another 10 instances since they can be solved in less than 5 sec-

onds by GLUCOSE. For the remaining 154 instances, there is no known (non-trivial) theoretical lower bound on the resolution depth and computing it is computationally impractical as it involves effectively exploring all possible proofs. Instead, for each instance F, we compute a CDCL refutation Γ_F using (sequential) GLUCOSE and record $d_{conf}(\Gamma_F)$. This is not necessarily close to the desired minimum depth, $d_{conf}(F)$, across all possible CDCL refutations. However, we present (indirect) evidence in the next section that proofs generated by GLUCOSE are in fact similar to the proofs generated by GLUSATX10, and are hence relevant to our current analysis.

For this experiment, we measured how $d_{conf}(\Gamma_F)$ correlates with the speedup observed on F in practice when going from GLUCOSE to GLUSATX10 (8). As one might expect by this point, there is weak correlation between the two. More precisely, the Pearson correlation coefficient is 0.25, which improves to 0.35 after removing one outlier.[1] While this (weak) correlation exists, we also find that the ideal speedup predicted solely by depth, namely $|\Gamma|/d_{conf}(\Gamma_F)$ for a refutation Γ, is often too optimistic (in the 100s, if not in the 1000s).

This suggests that conflict resolution depth is a relevant measure but overall too crude to estimate the degree of parallelizability of resolution refutations in practice.

Proof Schedules: A More Refined Measure

Two possibilities may account for the weak correlation between parallelizability observed in practice and conflict resolution depth. First, it could be that the CDCL refutation produced by one processor is not "similar" enough to the one generated jointly by the k processors. We believe this is not the case, especially for solvers such as GLUSATX10, PLINGELING, and CRYPTOMINISAT-MT which are strongly guided by the same heuristic search strategies that their sequential counterparts use. A second possibility is that the barrier to parallelizability imposed by depth, as outlined in Proposition 1, captures the case where there is no limit on the number of available processors[2] but does not provide any useful information in the case of a few processors. With this as our conjecture, we explore a measure of parallelizability that explicitly takes into account the number of processors.

Definition 2. *Let $G_{conf} = (V, E)$ be a conflict resolution DAG for a CDCL proof Γ of a CNF formula F. A* k-processor schedule *s of G_{conf} is a mapping from V (equivalently, from clauses in Γ) to $M = \{0, 1, \ldots, m\}$ such that:*

1. *for all $i \in M$, $1 \leq |\{v \in V \mid s(v) = i\}| \leq k$;*
2. *$s(v) = 0$ iff v is a leaf of G_{conf}, i.e., v is an input clause;*
3. *if $s(v) = i$ then for all predecessors u of v in G_{conf}, $s(u) \leq i - 1$.*

The makespan *of this schedule, denoted $f(s)$, is m.*

Definition 3. *Let Γ be a CDCL refutation. The* k-processor refutation makespan *of Γ, denoted $f(\Gamma, k)$, is $\min_s f(s)$ where the minimum is over all k-processor schedules s of Γ. Further, the* k-processor instance makespan*, denoted $f(F, k)$, is $\min_\Gamma f(\Gamma, k)$ where the minimum is over all CDCL refutations Γ of F.*

[1] Detailed data is available from the authors.

[2] Since there are only $\Theta(3^n)$ possible clauses over n variables, Proposition 1 also applies with $k = \Theta(3^n)$ processors.

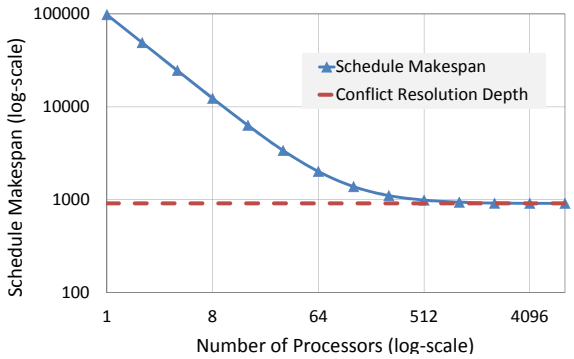


Figure 2: Schedule makespan as a finer characterization than conflict resolution depth

When it is clear from the context, we will, with slight abuse of notation, use the term makespan to also denote refutation makespan and instance makespan. The makespan satisfies the following properties:

Proposition 2. *Let Γ be a CDCL refutation. Then,*

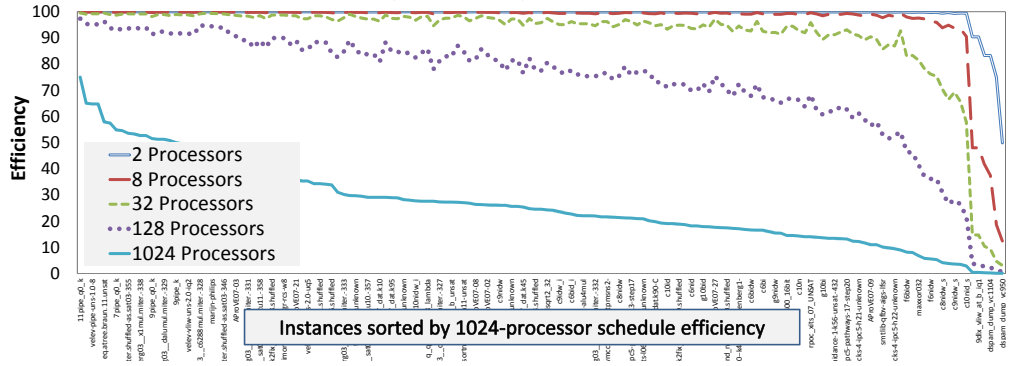
1. *for any 1-processor schedule s of Γ, $f(s) = |\Gamma|$, i.e., 1-processor makespan simply equals the refutation size;*
2. *for any schedule s of Γ, $f(s) \geq d_{conf}(\Gamma)$;*
3. *if $k \leq k'$, then $f(\Gamma, k) \geq f(\Gamma, k')$; and*
4. *as k grows to infinity, $f(\Gamma, k)$ converges to $d_{conf}(\Gamma)$.*

Figure 2 shows an example of the convergence property on an actual CDCL refutation generated by GLUCOSE on an AI planning instance from the SOKOBAN family. The refutation has roughly 100,000 clauses. A close look at the plot indicates near-linear decrease in schedule makespan (i.e., near-optimal speedup) as k, the number of processors, is increased till around 64, after which the curve starts to flatten out and essentially converges to the lower bound of conflict resolution depth after 1024 processors. This refined information for smaller k is what we hope to exploit.

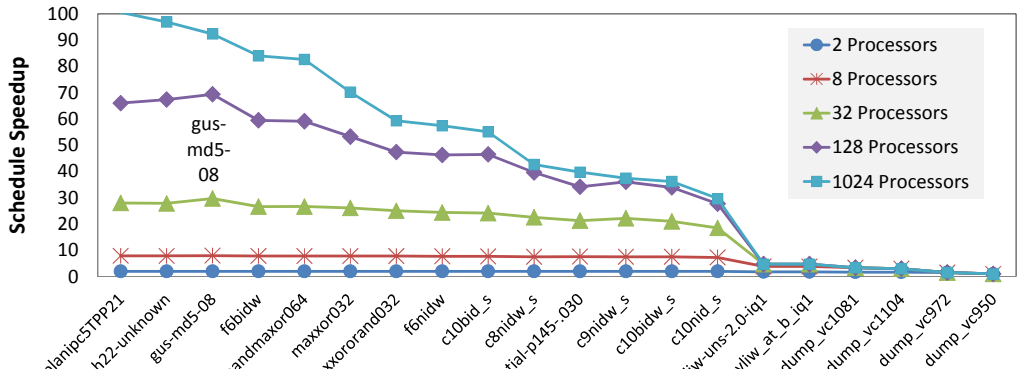
The properties in Proposition 2 naturally generalize to instance makespan as well: for any schedule s of any refutation of F, $f(s) \geq d_{conf}(F)$; and as k grows to infinity, $f(F, k)$ converges to $d_{conf}(F)$. Every k-processor schedule essentially describes one way of deriving the CDCL refutation using k processors, by specifying for each step t precisely up to k clauses that should be derived in parallel in step k. This naturally leads to a notion of speedup:

Definition 4. *Let Γ be a CDCL refutation. The* schedule speedup on Γ *when using k processors compared to using k' processors is $f(\Gamma, k')/f(\Gamma, k)$.*

We are now in a position to employ Proposition 2 and derive a result that generalizes the conflict resolution depth barriers delineated in Proposition 1.



(a) Parallelization efficiency of best case schedules drops significantly as k grows to 128 and 1024



(b) Many instances afford limited best case schedule speedup even with 1024 processors

Figure 3: Efficiency and schedule speedup of $k = 2, 8, 32, 128, 1024$ processor schedules on refutations generated by GLUCOSE

Theorem 2. *Let F be a CNF formula and S_k be a parallel CDCL SAT solver using k processors. Let $\mathcal{R}$ denote the set of all CDCL refutations of F. Then, for any k,*

1. *S_k needs at least $f(F,k)$ parallel steps to solve F; this bound decreases as k increases and converges to $d_{conf}(F)$ as $k \to \infty$;*
2. *Let Γ be a CDCL refutation of F produced by a solver S_1 using 1 processor. Compared to S_1, S_k cannot achieve a step speedup larger than $|\Gamma|/f(\Gamma,k)$ when deriving Γ; this bound increases with k and converges to $|\Gamma|/d_{conf}(F)$ as $k \to \infty$; and*
3. *S_k cannot achieve a step speedup of more than $\min_{\Gamma\in\mathcal{R}} |\Gamma|/\min_{\Gamma'\in\mathcal{R}} f(\Gamma',k)$ compared to an optimal sequential CDCL solver for F; this bound increases with k and converges to $\min_{\Gamma\in\mathcal{R}} |\Gamma|/d_{conf}(F)$ as $k \to \infty$.*

Thus, for finite, and in particular relatively small, values of k, schedule speedup provides a finer grained characterization of the barrier than conflict resolution depth alone. In the rest of this section, we study the correlation between schedule speedup and the typical speedups observed in practice.

Given Γ, k, and m, deciding whether $f(\Gamma,k)$ is at most m is equivalent to the classic Precedence Constrained Scheduling problem which is known to be NP-complete (Garey and Johnson 1979). Computing $f(\Gamma,k)$ is thus NP-hard. In fact, even approximating it within a factor of $4/3$ is NP-hard (Lenstra and Rinnooy Kan 1978). Fortunately, the simplest greedy algorithm — one that essentially performs a topological traversal of $G_{conf}(\Gamma)$ and greedily assigns as many (up to k) eligible vertices to each step as it can — is well-known to provide a $2-1/k$ approximation. A more recent algorithm provides a tighter approximation of $2 - 7/(3k+1)$ (Gangal and Ranade 2008), but for simplicity we use the $2 - 1/k$ approximation algorithm for our empirical study.

Empirical Evaluation

For our experiments, we use the same set of 154 instances from SAT Challenge 2012 as before, along with GLUCOSE and GLUSATX10 as our reference solvers, and the same computational platform as mentioned earlier.

Limited Parallelizability of Sequential Refutations. For the first experiment, for each instance F, we consider the CDCL refutation Γ produced by GLUCOSE on F in a sequential run and ask how much can Γ be parallelized *in the best case* when using k processors. This is captured precisely by $f(\Gamma,k)$, which we compute using the $2 - 1/k$ approximation algorithm.[3] Given this, one can compute a measure often used in systems studies when evaluating the

[3]The approximation algorithm often resulted in a perfect schedule speedup compared to using 1 processor, thereby suggesting that it often finds schedules of quality significantly better than the worst case approximation guarantee.

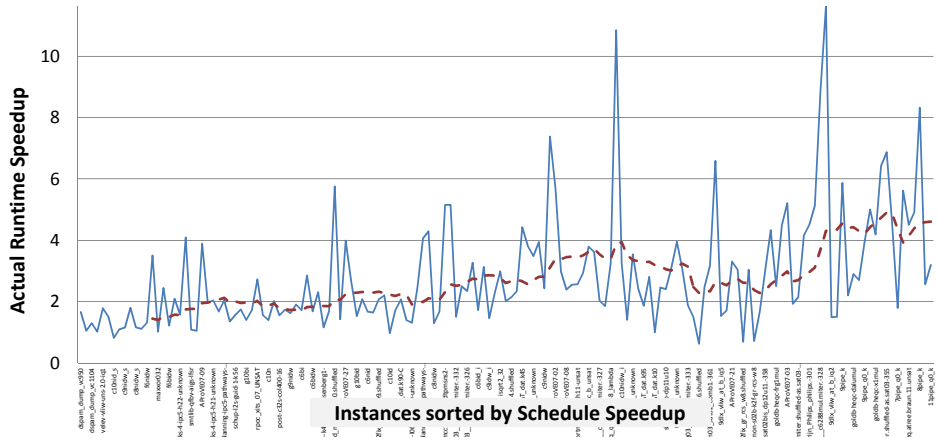


Figure 4: Correlation between Actual Speedup and Schedule Speedup. X-axis: instances sorted by schedule speedup of the sequential GLUCOSE refutation. Y-axis: actual (wall-clock) speedup of GLUSATX10 (8) compared to GLUCOSE.

performance of parallelization techniques, namely, the *efficiency of parallelization*. Efficiency captures as a percentage how close the use of k processors comes to achieving the ideal case speedup of a factor of k. In our context, the *k-processor schedule efficiency* on Γ is defined as $(f(\Gamma,1)/f(\Gamma,k)) \times 100/k$. 100% efficiency thus indicates perfect parallelization.

Figure 3(a) shows the observed parallelization efficiency for $k = 2, 8, 32, 128$, and 1024 processors, with one curve for each k. The 154 instances are sorted along the X coordinate in decreasing order by efficiency with 1024 processors. We see that the efficiency of parallelization starts to decrease significantly as k increases. The dramatic drop in efficiency beyond 128 processors clearly shows that *CDCL refutations produced by state-of-the-art SAT solvers are often inherently not very parallelizable beyond a few dozen processors — even in the best case* where scheduling is performed offline (i.e., in hindsight) with full proof structure in hand.

Figure 3(b) shows another view of the same data, focused particularly on instances towards the right hand side of Figure 3(a), whose GLUCOSE generated sequential refutation is not very parallelizable. It highlights, for example, that *refutations of cryptographic instances such as gus-md5-08.cnf can only be parallelized 100x irrespective of how many processors are available*. Further, to achieve this 100x speedup on this proof, it is not sufficient to use just 128 processors; one must go beyond 1024 processors.

Schedule Speedup Correlates with Parallelizability. For this experiment, we consider the same set of 154 SAT Challenge 2012 instances as in Figure 3(a) and measure (a) their 1024-processor *schedule speedup* and (b) the *actual speedup* observed when running GLUSATX10 (8) versus GLUCOSE. The goal is to demonstrate a correlation between these two quantities.

In Figure 4, the instances are shown sorted along the X coordinate in increasing order of their schedule speedup, with the Y coordinate showing the actual observed speedup. Not surprisingly, there is a significant amount of variation in actual speedup across the instances, which come from a variety of application domains. Nonetheless, as highlighted by the smoother line showing the moving average over a window of 20 instances, there is an upward trend in the data. More precisely, the Pearson correlation coefficient between the observed runtime speedup and the schedule speedup is 0.51, which is typically categorized as "medium" to "strong" statistical correlation. This provides evidence for our conjecture that *the actual observed speedup (in wall-clock time) in modern day CDCL solvers correlates with how parallelizable the CDCL refutations generated by their sequential counterparts are*, which in turn is relatively well *captured by the schedule speedup measure* we introduced.

Conclusions

This paper introduced a new line of study of why, despite tremendous advances in the design of sequential CDCL SAT solvers, researchers have faced difficulty in parallelizing them. We presented empirical evidence that it is the structure of the resolution refutations produced by current solvers that hinders their parallel efficiency. Portfolio approaches are unlikely to help in this context, as they do not even attempt to parallelize the construction of the refutation. Our study suggests that a promising way forward for designing effective parallel clause learning solvers is to alter their behavior so that they discover more parallelizable refutations, even at the expense of worse single-thread performance. More generally, it may turn out that certain resolution refutations are inherently not parallelizable beyond a certain limit, in which case our study would suggest exploring solvers based on other proof systems that produce more parallelizable refutations.

References

Achlioptas, D.; Beame, P.; and Molloy, M. 2001. A sharp threshold in proof complexity. In *33rd STOC*, 337–346.

Achlioptas, D.; Beame, P.; and Molloy, M. 2004. Exponential bounds for DPLL below the satisfiability threshold. In *15th SODA*, 139–140.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *21st IJCAI*, 399–404.

Balint, A.; Belov, A.; Järvisalo, M.; and Sinz, C. 2012. SAT challenge.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Understanding and harnessing the potential of clause learning. *JAIR* 22:319–351.

Ben-Sasson, E., and Nordström, J. 2008. Short proofs may be spacious: An optimal separation of space and length in resolution. In *FOCS*, 709–718.

Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Biere, A. 2012. Plingeling: Solver description. SAT Challenge 2012.

Bloom, B.; Grove, D.; Herta, B.; Sabharwal, A.; Samulowitz, H.; and Saraswat, V. A. 2012. SatX10: A scalable plug&play parallel sat framework - (tool presentation). In *SAT*, 463–468.

Buss, S. R.; Hoffmann, J.; and Johannsen, J. 2008. Resolution trees with lemmas: Resolution renements that characterize DLL-algorithms with clause learning. *Logical Methods in Computer Science* 4(4):13.

Cook, S. A., and Reckhow, R. A. 1979. The relative efficiency of propositional proof systems. *J. Symb. Logic* 44(1):36–50.

Gangal, D., and Ranade, A. G. 2008. Precedence constrained scheduling in $(2-7/(3p+1))$ optimal. *J. Comput. and Syst. Sc.* 74(7):1139–1146.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., New York.

Hamadi, Y., and Wintersteiger, C. M. 2012. Seven challenges in parallel sat solving. In *26th AAAI*.

Hertel, P.; Bacchus, F.; Pitassi, T.; and Gelder, A. V. 2008. Clause learning can effectively p-simulate general propositional resolution. In *23rd AAAI*, 283–290.

Järvisalo, M.; Matsliah, A.; Nordström, J.; and Zivny, S. 2012. Relating proof complexity measures and practical hardness of sat. In *CP*, 316–331.

Järvisalo, M.; Le Berre, D.; and Roussel, O. 2011. SAT competition.

Kozen, D. 1977. Lower bounds for natural proof systems. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, 254–266. IEEE.

Lenstra, J. K., and Rinnooy Kan, A. 1978. Complexity of scheduling under precedence constraints. *Operations Research* 26:22–35.

Pipatsrisawat, K., and Darwiche, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *AI J.* 175(2):512–525.

Robinson, J. A. 1965. A machine oriented logic based on the resolution principle. *Journal of the ACM* 12(1):23–41.

Roussel, O. 2012. ppfolio: Solver description. SAT Challenge 2012.

Soos, M. 2012. CryptoMiniSat 2.9.2. *http://www.msoos.org/cryptominisat2*.

Urquhart, A. 2011. The depth of resolution proofs. *Studia Logica* 99(1-3):349–364.

Wotzlaw, A.; van der Grinten, A.; Speckenmeyer, E.; and Porschen, S. 2012. pfolioUZK: Solver description. SAT Challenge 2012.