Universidade Técnica de Lisboa
Instituto Superior Técnico

Parallel MiniSAT

Luís Filipe Colaço Messias Gil

Diploma Thesis
Degree in Computer Science

Supervisors from INESC–ID:
Prof. Luís Miguel Silveira
Prof. Paulo Flores

Supervisor from DM–IST:
Prof. Jaime Ramos

July 2007

Agradecimentos

Se um homem esvazia a sua carteira para a sua cabeça, ninguém lhe pode ficar com nada. Um investimento em conhecimento paga sempre o melhor proveito.
Benjamin Franklin

Em primeiro lugar quero agradecer aos meus orientadores Luís Miguel Silveira, Paulo Flores e Jaime Ramos todo o apoio, orientação, sugestões e paciência concedidos para que este trabalho pudesse ser concluído com êxito.

Agradeço ao Professor Amílcar Sernadas pela ajuda na escolha do TFC e por me ter indicado o professor Luís Miguel Silveira como especialista em computação paralela.

Ao Dr. Niklas Eén o esclarecimento de dúvidas relacionadas com o funcionamento do MiniSAT.

Aos meus amigos Ana Sofia Graça, Bruno Pereira, Carlos Tamulonis, Hélio Pais, Paulo Abrantes e Sandra Marques pelos comentários e sugestões sobre este relatório e o programa.

Agradeço ao Professor David Matos por tirar dúvidas e resolver os problemas com a *grid*.

Também quero agradecer ao INESC–ID o facto de ter colocado à minha disposição os meios computacionais necessários para desenvolver e testar o programa.

Finalmente quero agradecer aos meus pais que me ofereceram esta licenciatura. Sem eles nada teria sido possível. Obrigado pelo apoio e incentivo para alcançar sempre os objectivos propostos, o carinho e atenção dados nos momentos difíceis, e o mais importante de tudo, o esforço e sacrifício para que eu tivesse sempre uma boa educação.

Acknowledgements

If a man empties his purse into his head, no man can take it away from him. An investment in knowledge always pays the best interest.
Benjamin Franklin

First of all, I want to thank my supervisors Luís Miguel Silveira, Paulo Flores and Jaime Ramos for all the help, guidance, suggestions and patience given, allowing me to finish this work with success.

I also want to thank Professor Amílcar Sernadas for helping me choose my final year project and by suggesting professor Luís Miguel Silveira as an expert in parallel computing.

Thanks to Dr. Niklas Eén for having answered some of our doubts about MiniSAT.

To my friends Ana Sofia Graça, Bruno Pereira, Carlos Tamulonis, Hélio Pais, Paulo Abrantes e Sandra Marques for the comments and suggestions about this report and the program.

Thanks to Professor David Matos for answering my questions and solving the problems with the grid.

I also want to thank INESC–ID for providing the computational means to develop and test the program.

Finally, last but not the least, I want to thank to my parents that offered me this degree. Without them nothing would be possible. Thanks for the support for me to always reach all the proposed goals, the care and attention on the rough times, and most of all, the effort and sacrifice so that I would always get a good education.

Resumo

O problema SAT, verificar se uma fórmula booleana pode ser verdadeira, ocorre em diversas áreas da ciência e engenharia como por exemplo automação de desenho electrónico, pesquisa, planeamento e verificação formal.

Existem vários programas para resolver o problema que utilizam os algoritmos mais avançados que a ciência conhece. Um deles é o MiniSAT que é amplamente utilizado.

Este trabalho apresenta a pesquisa, planeamento e implementação de uma versão paralela do MiniSAT que utiliza a tecnologia MPI (*Message Passing Interface*) para ser executada numa *grid* ou *cluster* de computadores. São descritas as principais características do programa: modos de busca, remoção de hipóteses e partilha de cláusulas aprendidas. Também é avaliada a eficiência da paralelização e do esquema de distribuição de carga entre os processadores.

Palavras chave: Computação Paralela, SAT-Solver, Satisfiabilidade.

Abstract

The SAT problem, determine if a given boolean formula can be true, is largely used in several areas of science and engineering like electronic design automation, search, planning and formal verification.

There are several programs to search for satisfiability, called SAT-solvers, that use some of the most advanced techniques known. One of them is MiniSAT that is widely used.

This work presents a research, planning and implementation of a parallel version of MiniSAT with MPI (*Message Passing Interface*) technology to be executed in clusters or grids of computers. Are described the main features of the program: search modes, assumptions pruning and share of learnt clauses. Is also made an analysis of its performance and load balance.

Keywords: Parallel Computing, SAT-Solver, Satisfiability.

Contents

1 Introduction **1**

2 Logic and complexity **2**
2.1 Boolean Logic 2
2.2 Computational complexity 4

3 Parallel computing **6**
3.1 Technology 6
3.1.1 Hardware 6
3.1.2 Software 8
3.2 Programming 9
3.2.1 Methodology 9
3.2.2 Concerns 11
3.3 Performance measures 12
3.3.1 Metrics 12
3.3.2 Times 13

4 The SAT-Solver **15**
4.1 General algorithms 15
4.2 MiniSAT 17

5 Parallel implementation **19**
5.1 PCAM design 19
5.1.1 Partitioning 19
5.1.2 Communication 20
5.1.3 Agglomeration 20
5.1.4 Mapping 20
5.2 Implementation details 20
5.2.1 Variables selection 21
5.2.2 Assumptions generation 21
5.2.3 Assumptions pruning 26
5.2.4 Sharing learnt clauses 26
5.2.5 Messages 27
5.2.6 Automatic settings 28

5.3 Application's work flow 29
5.4 Modules of the program 31
5.5 Technology 32

6 Experimental results and performance analysis 33
6.1 Grid resources 33
6.2 Methodology 33
6.3 Goals and difficulties 35
6.4 Time measurement 36
6.5 Results 37
6.5.1 Communication delay 38
6.5.2 Modes and options 38
6.5.3 Granularity 40
6.5.4 Load distribution 41

7 Conclusion 42

A User manual 45
A.1 How to use this manual 45
A.2 System requirements and installation 45
A.3 Quick start 45
A.4 Usage 46
A.5 Options 47
A.6 Examples 50
A.7 Inputs and Outputs 50
A.8 Hints for better performance 52
A.9 Error messages 53
A.10 FAQ 54

B Performance tables 56

List of Figures

2.1 Circuit 5

3.1 Pipeline 10
3.2 Task Farm 10

4.1 Davis-Putnam algorithm 15

5.1 Task Farm organization 21
5.2 Function test4SAT() 29
5.3 Program's main() 30

6.1 Timed operations (gray fill) 37

A.1 Program's options 48
A.2 Configuration file 49

List of Tables

4.1 Mapping of modules in the file system 18

5.1 Message for *Equal mode* 27
5.2 Message for *Progressive mode*, assuming $n < k$ literals 27
5.3 Result message 28
5.4 Set of learnt clauses 28
5.5 Model of the formula 28
5.6 Mapping of modules in the file system 32

6.1 Benchmark files 34

B.1 Sequential vs Parallel with master and local worker 58
B.2 Sequential vs Parallel with master and remote worker 59
B.3 Average time of each file taken by the sequential MiniSAT . . . 59
B.4 Average of each file in 3, 6 and 9 workers 60
B.5 Average of each SAT file in 3, 6 and 9 workers 60
B.6 Average of each UNSAT file in 3, 6 and 9 workers 61
B.7 Best performances without options 61
B.8 Worst performances without options 62
B.9 Best performances with conflicts 63
B.10 Worst performances with conflicts 64
B.11 Best performances sharing learnt clauses 65
B.12 Worst performances sharing learnt clauses 66
B.13 Best performances with different granularity 67
B.14 Worst performances with different granularity 68
B.15 Tests with 6 variables to `fpga10_11_uns_rcr.cnf` 69
B.16 Tests with 6 variables to `fpga10_12_uns_rcr.cnf` 70
B.17 Tests with 6 variables to `fpga10_13_uns_rcr.cnf` 71
B.18 Tests with 6 variables to `frb40-19-1.cnf` 72
B.19 Tests with 6 variables to `frb40-19-2.cnf` 73
B.20 Tests with 6 variables to `frb40-19-3.cnf` 74
B.21 Tests with 6 variables to `frb40-19-4.cnf` 75
B.22 Tests with 6 variables to `frb40-19-5.cnf` 76
B.23 Tests with 6 variables to `hole11.cnf` 77
B.24 Tests with 6 variables to `mod2-3cage-unsat-9-11.cnf` 78

B.25 Tests with 6 variables to mod2-3cage-unsat-9-4.cnf 79
B.26 Tests with 6 variables to mod2-3g14-sat.cnf 80
B.27 Tests with 6 variables to mod2c-rand3bip-sat-150-11.cnf . . . 81
B.28 Tests with 6 variables to mod2c-rand3bip-sat-150-15.cnf . . . 82
B.29 Tests with 6 variables to sat2.cnf 83
B.30 Tests with 6 variables to unif-r4.cnf 84
B.31 Tests with 6 variables to unif-r5.cnf 85
B.32 Tests with 6 variables to vmpc_21.renamed-as.sat05-1923.cnf 86
B.33 Tests with 6 variables to vmpc_23.renamed-as.sat05-1927.cnf 87
B.34 Tests with 6 variables to vmpc_25.renamed-as.sat05-1913.cnf 88
B.35 Tests with 6 variables to vmpc_25.shuffled-as.sat05-1945.cnf 89
B.36 Tests with 6 variables to vmpc_26.renamed-as.sat05-1914.cnf 90
B.37 Tests with 6 variables to vmpc_26.shuffled-as.sat05-1946.cnf 91
B.38 Tests with 6 variables to vmpc_27.renamed-as.sat05-1915.cnf 92
B.39 Tests with 6 variables to vmpc_27.shuffled-as.sat05-1947.cnf 93
B.40 Tests of granularity to fpga10_11_uns_rcr.cnf 94
B.41 Tests of granularity to fpga10_12_uns_rcr.cnf 95
B.42 Tests of granularity to fpga10_13_uns_rcr.cnf 96
B.43 Tests of granularity to frb40-19-1.cnf 97
B.44 Tests of granularity to frb40-19-2.cnf 98
B.45 Tests of granularity to frb40-19-3.cnf 99
B.46 Tests of granularity to frb40-19-4.cnf 100
B.47 Tests of granularity to frb40-19-5.cnf 101
B.48 Tests of granularity to hole11.cnf 102
B.49 Tests of granularity to mod2-3cage-unsat-9-11.cnf 103
B.50 Tests of granularity to mod2-3cage-unsat-9-4.cnf 104
B.51 Tests of granularity to mod2-3g14-sat.cnf 105
B.52 Tests of granularity to mod2c-rand3bip-sat-150-11.cnf 106
B.53 Tests of granularity to mod2c-rand3bip-sat-150-15.cnf 107
B.54 Tests of granularity to sat2.cnf 108
B.55 Tests of granularity to unif-r4.cnf 109
B.56 Tests of granularity to unif-r5.cnf 110
B.57 Tests of granularity to vmpc_21.renamed-as.sat05-1923.cnf . . 111
B.58 Tests of granularity to vmpc_23.renamed-as.sat05-1927.cnf . . 112
B.59 Tests of granularity to vmpc_25.renamed-as.sat05-1913.cnf . . 113
B.60 Tests of granularity to vmpc_25.shuffled-as.sat05-1945.cnf . 114
B.61 Tests of granularity to vmpc_26.renamed-as.sat05-1914.cnf . . 115
B.62 Tests of granularity to vmpc_26.shuffled-as.sat05-1946.cnf . 116
B.63 Tests of granularity to vmpc_27.renamed-as.sat05-1915.cnf . . 117
B.64 Tests of granularity to vmpc_27.shuffled-as.sat05-1947.cnf . 118

Chapter 1

Introduction

The SAT problem deals with finding a satisfying assignment to a boolean formula of propositional logic. This problem is associated to computational complexity and was the first NP-Complete problem ever found in 1971 by Stephen Cook [3]. Its applications range from industry to science where many problems are mapped in SAT and solved by specialized programs called SAT-solvers. Since the last decade there has been much research devoted to create better algorithms to allow SAT-solvers to search for solutions faster and solve larger problems with limited resources.

Parallel computing is an old theme in the demand for more computational power. Supercomputers are built since the 1960's but in those early days only some research laboratories and military agencies could support their cost. In recent years the decrease of hardware cost and the rise of programming tools made possible the construction of cheap parallel machines using home and office computers connected by a network, known as clusters or grids. These machines are used to solve scientific and industrial problems like wind tunnel or earthquake simulations.

The aim of this work is to create a parallel version of an existing SAT-solver called MiniSAT, that was developed by Niklas Eén and Niklas Sörensson at Chalmers University in Sweden [5]. Our objectives are to create a tool that runs in generic Linux clusters, can be able to solve larger problems and to research the potential provided by parallel computing to accelerate SAT-solvers

The rest of this document is organized as follows: in Chapters 2 and 3 the necessary concepts about the SAT problem and parallel computing to allow the understanding of the rest of the work. Readers familiar with former concepts can skip these chapters. In Chapter 4 the architecture of MiniSAT is analyzed. Chapter 5 presents the parallel architecture for the program. Chapter 6 shows the results of performance analysis and Chapter 7 the conclusion of the work. Finally as appendix, the user manual and the performance tables.

Chapter 2

Logic and complexity

Logic is the anatomy of thought.
John Locke

In this introductory chapter we will present the main definitions and results about boolean formulas, computational complexity and their relations with the SAT problem.

2.1 Boolean Logic

Definition 2.1.1 *A boolean variable is a symbol that might assume one of two values:* true *or* false.

Definition 2.1.2 *Given a set X of boolean variables, the class of boolean formulas is the smallest class defined by:*

1. true *and* false *are boolean formulas;*
2. *every boolean variable $x \in X$ is a boolean formula;*
3. *if F_1 and F_2 are boolean formulas then $\neg(F_1)$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$ are also.*

The symbols $\neg$, $\wedge$ and $\vee$ are called operators that represent the three basic operations: complement, conjunction and disjunction, respectively. Other operators are defined to represent abbreviations:

1. Implication: $(F_1 \Rightarrow F_2)$ stands for $(\neg(F_1) \vee F_2)$.
2. Equivalence: $(F_1 \Leftrightarrow F_2)$ stands for $((F_1 \Rightarrow F_2) \wedge (F_2 \Rightarrow F_1))$.
3. Exclusive or: $(F_1 \oplus F_2)$ stands for $((F_1 \wedge \neg(F_2)) \vee (\neg(F_1) \wedge F_2))$.

A boolean formula may be *true* of *false*. The operators are used to assert formula's logical value by the following way:

1. $(F_1 \vee F_2) =$ *true* if F_1, F_2 or both are *true*, *false* otherwise;
2. $(F_1 \wedge F_2) =$ *true* if F_1 and F_2 are *true*, *false* otherwise;
3. $\neg(F) =$ *true* if $F =$ *false* and vice versa.

Definition 2.1.3 *A boolean assignment V is a mapping from a set of boolean variables to* {true,false}*. Using assignments we can express the boolean value of a formula F:*

1. *V(F)* = false *if F* = false*;*
2. *V(F)* = true *if F* = true*;*
3. *V(F) = V(x) if F = x;*
4. *V(F) = V(*F_1*)* $\vee$ *V(*F_2*) if F = (*$F_1 \vee F_2$*);*
5. *V(F) = V(*F_1*)* $\wedge$ *V(*F_2*) if F = (*$F_1 \wedge F_2$*);*
6. *V(F)* $= \neg V(F_1)$ *if* $F = \neg(F_1)$*;*
7. *V(F) = V(*F_1*) if* $F = \neg(\neg(F_1))$.

For a formula with n variables there are 2^n different possible assignments, since every variable may be *true* or *false*.

Definition 2.1.4 *A formula F is said to be satisfiable if there is an assignment V that V(F)* = true*; otherwise F is unsatisfiable.*

Definition 2.1.5 (SAT) *The set of all satisfiable boolean formulas is known as* SAT.

The SAT problem is to determine if exists an assignment that makes satisfiable a given formula. That assignment is known as *model*.

Definition 2.1.6 (Literal) *A literal is a boolean variable or its complement.*

Definition 2.1.7 (Clause) *A clause is a disjunction of literals.*

Definition 2.1.8 (CNF) *The conjunctive normal form (CNF) is a conjunction of clauses.*

Example 2.1.9 *The formula* $(\varphi \vee \psi) \wedge (\gamma \vee \neg\delta \vee \neg\varphi)$ *is in CNF.*

Theorem 2.1.10 *Every boolean formula can be written in conjunctive normal form.*

Example 2.1.11 *The boolean formula* $(\varphi \wedge \psi) \vee \gamma$ *may be written in CNF as* $(\varphi \vee \gamma) \wedge (\psi \vee \gamma)$.

The advantage of searching satisfiability in CNF formulas is that each clause must be *true* for the entire formula be *true*.

The search is made by assigning variables successively and verifying conflicts.

Definition 2.1.12 (Unit clause) *An unit clause is a clause that has one unassigned literal and all the others assigned as* false.

Definition 2.1.13 (Implication) *An implication is to assign as* true *a literal in an unit clause.*

Definition 2.1.14 (Conflict) *A conflict happens when the same variable is set to* true *in one implication and* false *in another.*

Example 2.1.15 *Searching the satisfiability of* $(\varphi \vee \psi) \wedge (\gamma \vee \neg\delta \vee \neg\varphi)$*:*
if δ *is assigned to* true*,* ψ *and* γ *to* false*, both clauses become unit clauses. The implications on* φ *assign to it* true *in the first clause and* false *in the second leading to a conflict. But the clause is satisfiable just by assigning* ψ *and* γ *as* true*.*

2.2 Computational complexity

As was said before, in worst case one needs to test an exponential number of assignments while seeking for satisfiability of a formula. Until now nobody has found an algorithm to limit the search time upper-bounded by a polynomial $p(n)$.

As the matter of fact SAT, is a very difficult problem in terms of computational complexity and was proved to be NP-Complete in 1971 by Stephen Cook [3]. We will now present a set of definitions to clarify these assertions and provide a background in computational complexity.

First of all, lets present the two main theoretical methods used to describe the computation in mathematical models like automata and Turing machines [1, 13].

Definition 2.2.1 *A deterministic computation is one that we can determine the next state just by knowing the actual state and the given input.*

Definition 2.2.2 *A nondeterministic computation allows the existence of several ways to proceed from one state for a given input. All transitions are processed in parallel.*

In a nondeterministic computation may be transitions from one state to several others by the same input. In that case the machine is split in several copies and explore the different branches of the computation in parallel.

Notice that every nondeterministic machine can be converted into a deterministic one, which explores the different branches of computations sequentially.

Complexity classes are used in theoretical computer science to classify several types of problems according to the relation between the length of the input and amount of resources (like time or space[1]) required to solve them [1, 13]. We will define the most widely known and relevant time and space complexity classes:

Definition 2.2.3 *The class P is the set of problems computed in polynomial time by a deterministic procedure.*

Definition 2.2.4 *The class NP is the set of problems computed in polynomial time by a non-deterministic procedure.*

The known relation between these classes is P $\subseteq$ NP while the open problem so far is P = NP.

Theorem 2.2.5 *SAT $\in$ NP.*

The combinations of assignments to n variables can be seen as a binary tree with depth n. So one can write a nondeterministic procedure that generates and checks each branch concurrently. The time of execution is upper-bounded by a $p(n)$ for some polynomial p.

Another way to define NP is classifying it as the class of problems whose potential solutions can be verified in polynomial time. By this definition, SAT belongs to NP which means that given an assignment V and a boolean formula F with n variables, one can determine if V satisfies F in polynomial time.

Definition 2.2.6 *A problem G is NP-Complete if belongs to NP and every other problem $e \in E$ of that class can be encoded in $g \in G$ by a deterministic algorithm in polynomial time, such that the answer to g is YES if and only if the answer to e is YES.*

Theorem 2.2.7 *SAT is NP-Complete.*

The previous theorem states that every problem in NP can be represented by a boolean formula and solved in that context. Also indicates that there is no optimal algorithm to solve SAT and probably never will be.

Figure 2.1: Circuit

Example 2.2.8 *The digital circuit in Figure 2.1 can be represented by the boolean formula $(A \wedge B) \vee C$. Is converted to CNF as $(A \vee C) \wedge (B \vee C)$. To know if the circuit can output value 1, one should determine if the formula is satisfiable.*

[1] memory

Chapter 3

Parallel computing

I just bought a Mac to help me design the next Cray!
Seymour Cray - when informed Apple got a Cray supercomputer to help design next Mac

Parallel computing is used with the objective of having more computational power. There are several types of machines to perform parallel computing: supercomputers, clusters and grids. The firsts are the most expensive solution and currently are only used for dedicated tasks while the latest are cheaper. Grids represent an evolution from clusters because they allow the user to abstract from their structure and location.

Definition 3.0.9 (Node) *A node is a computer with one or more processors.*

Definition 3.0.10 (Task) *A task is a process that belongs to a running program. A sequential program has one sequential task while a parallel program may have many tasks running simultaneously.*

3.1 Technology

This section describes the hardware and software technology used in parallel computing. Is presented a historical perspective until today's most recent innovations.

3.1.1 Hardware

Supercomputers

Supercomputers have been made since the 60's by IBM, CDC and UNIVAC. Due to their cost just big companies government's agencies and laboratories, or major universities were able to support them. In 1972 an engineer from

CDC tired of the company's philosophy to avoid risks on new systems got out and created his own computer company. His name was Seymour Cray and wanted to build computers to run simulations rather than data manipulations, to allow designers to model objects before being built. For several years Cray's supercomputers were the fastest in the world.

After Cray, more than twenty supercomputer companies appeared specially during the 80's, but got out of scene some years later due to "supercomputer market crash". The increase of computational power of the workstations and the decrease of costs of the hardware were the main causes of this crash.

Until the early 80's these supercomputers had only a very fast CPU. Then several processors were added, typically between 2 and 8, to allow the accomplishment of parallel calculations. Nowadays they can have several hundreds or thousands of processors, in most cases off the shelf RISC CPUs.

Supercomputers are still expensive and are used in special intensive CPU applications like weather forecast, chess games, cipher breaking, fluids dynamics, analysis of geological data and nuclear energy research.

Clusters

A cluster is a set of dedicated computers, connected by a network, placed in the same physical space, to provide a large amount of computational power. Its objective is to be a cheap alternative to a supercomputer.

The history of clusters started when the lack of financial means to get a supercomputer led a group of engineers at NASA to connect some unused computers, equipped with Linux operating system and open source tools, by an Ethernet network and use it to perform parallel computing. This architecture, known as Beowulf, was a success and many universities, companies, laboratories and home users adopted it and started to build their own clusters based in desktop computers and open source software, that allowed for instance the teaching of parallel programming in universities with few financial resources.
The costs of building a cluster dropped along the 90's because of the reduction of prices of hardware, specially the network components. The programming cost also decreased due to the creation of tools and libraries that help to develop parallel applications. Clusters have been used to perform parallel computing in scientific and engineering problems like fluids dynamics and wind tunnel simulations.

Grids

A grid is different from a cluster because its computer nodes can belong to different owners, may be scattered by a wide geographical area and connected by Internet. May not be composed by dedicated machines (some of them are desktops), having more objectives besides supplying computational power and may fail or be unavailable. Due to the use of special software, the users see the grid as a single computer ignoring the status of the nodes, their amount and location.

The main objective of grid computing is to take advantage of the unused resources of the nodes, like CPU cycles and disk storage, in order to offer services or to perform computation over problems too big for a single supercomputer.

There are several definitions for a grid, one of them is given by Ian Foster in [8]:

Definition 3.1.1 (Grid) *A grid is a system that coordinates resources that are not subject to centralized control using standard, open, general purpose protocols and interfaces to deliver non-trivial qualities of service.*

As one can see, a grid can be more than a set of computers. In fact there are several types of grids:

1. computational: for CPU intensive use;
2. data: for storing and sharing distributed data;
3. equipment: to control remotely a hardware sensor and process the data produced.

There are many projects for grid computing, some of them at a global scale like the ones to use at home: SETI (search for extraterrestrial intelligence), protein folding, stock forecasts. Other applications are video on demand, particles physics, earthquake simulation, genetics and astronomy.

3.1.2 Software

It's not enough to have a cluster or a grid. To take advantage of it, our programs must be written in a way to use all (or most part of) the available resources, by distributing the computation among the nodes. Parallel programming languages have been made but did not become a working tool for scientists and researchers because they wanted to use the languages that already knew like C or Fortran. The solution found was in the form of a library acting as middleware and offering a message passing protocol to control the communication and synchronization of the tasks.

Several, but incompatible message passing systems were made, during the 1980's and early 1990's:

- PICL, PVM - Oak Ridge National Laboratory;
- PARMACS, P4, Chameleon - Argonne National Laboratory;
- LAM - Ohio Supercomputer Center;
- Express - Caltech/Parasoft;
- TCGMSG - special for quantum chemistry.

In 1992 during the Supercomputing conference, a committee known as MPI-Forum was formed to specify a new message passing interface (MPI). The main goal was to create a standard, portable and efficient message passing system, available to several architectures and operating systems and that provided a set of routines for interprocess communication to allow the exchange of messages with data.

Two years later, in 1994, appeared the first standard MPI-1.0 which was implemented by many commercial and open-source vendors. MPI-2.0 standard, published in 1997, added more functionalities. Nowadays MPI has become the de-facto standard for writing parallel applications, specifying the interface and functionality of communication routines, due to the high level of portability and scalability achieved for its programs.

For grids was started a initiative to create software to present the grid as a single machine. This software is called Globus.

3.2 Programming

Writing a parallel program is very different from writing a sequential one. One needs to focus not only in the problem to solve but also in its partition, work distribution, gathering results, data flow, message formats, amount of messages, delays and bottlenecks.

Now are presented the procedures to create parallel programs as well as the main concerns that programmers must have in mind when designing such systems.

3.2.1 Methodology

A methodology to decompose the problem and obtain a parallel algorithm in four stages was proposed by Ian Foster in [7]. This approach was also used to create our program. Its four stages are Partitioning, Communication, Agglomeration and Mapping, also known as PCAM.

Partitioning

In this stage, the computation to solve the problem is partitioned from the functional or domain points of view.

This early partition should be fine-grained, defining small tasks, even if in a large number. The granularity (amount and size of the tasks) will be reviewed in the Agglomeration stage. The partition can be made at the domain level or at functional level:

- **Domain partition:** to divide the input, intermediate or output data in small blocks that can be processed independently.
- **Functional partition:** if computations can be divided in disjoint tasks. If the data required for these tasks is also disjoint, one says the partition is complete.

Example 3.2.1 *A problem where domain partition can be applied is matrix multiplication, where every element of the result does not depend of any other.*

Example 3.2.2 *A functional partition is applied in an image processing system where data suffers successive and independent transformations.*

Many parallel programs have similar architectures or use design patterns to deal with some functional requirements. Two of the most used architectural styles are *Pipeline* and *Task Farm*.

Notation 3.2.3 *We will visually represent architectural style in an informal manner with the notation of boxes and arrows, where the boxes mean processes and the arrows communication channels and the direction of the information flux.*

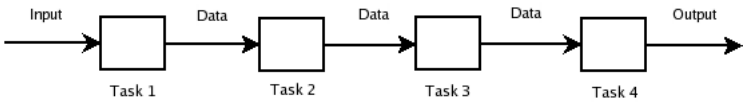


Figure 3.1: Pipeline

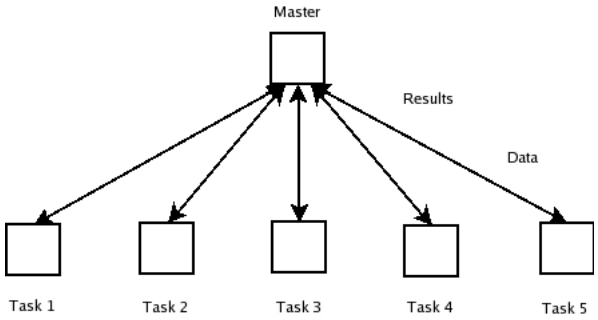


Figure 3.2: Task Farm

A *Pipeline* is used to process a stream of data sequentially with several functions running in the different tasks. Is adequate for a functional partition. Each task receives a stream of data, processes and sends it to other task always different from the previous ones. It is commonly used for image or signal processing where for instance each task may apply one filter to the input and send the output to the next task.

In *Task Farm* there is a master task that sends orders for all others (called the workers) and collects the results. All workers execute the same function, being a good architecture for domain partition, where different parts of the domain are sent to different tasks to be processed. For the matrix multiplication example, each task could calculate a subset of the result matrix.

Obviously, there are problems that have their own particular architectural style with tasks connected *ad hoc*.

Communication

Communication is required to share data among the tasks when data dependency exists, to gather results, to send commands or synchronizing tasks. Can be categorized as follows:

- **Local/Global:** is related with the amount of tasks that each one of them needs to communicate with. Is local when a task only exchanges messages with a small group of other tasks, and global when it sends and receives messages for/from a large group.
- **Structured/Unstructured:** whether a task and its neighbors form a defined communication structure, or have arbitrary connections.
- **Static/Dynamic:** this category is related with the identity of communication partners changes over the computation. If static, a task communicates always with the same partners. If dynamic the partners change, most of the times due to the processed data.
- **Synchronous/Asynchronous:** if the producers and consumers of data are coordinated or not during the transfers.

Agglomeration

After defining the partition and communication, one needs to increase the granularity of the tasks by combining a group of them into a unique one, to distribute them among the available processors and decrease the communication requirements, i.e. the number of messages to send.

The reduction of the number of messages, even if the amount of data is kept, is one of the most important accomplishments in order to decrease the time spent during communication. The tasks must be big enough to spend more time computing rather communicating. It is also important to join tasks that depend on each other.

Mapping

Mapping is about the distribution of the tasks by the machines where they will run. Two ideas to make computation more effective and quick are to distribute concurrent tasks by different machines and tasks that communicate frequently in the same machine. These ideas are inadequate when we have CPU demanding tasks that need to communicate with many others. In this case they cannot be put together on a single machine because they would have to wait a lot before execute or the frequent context switchings could introduce long delays.

3.2.2 Concerns

In both architectural styles some performance issues must be taken into account when designing the algorithm.

The first one, related with agglomeration, is the processing time of each task that must exceed the time of one or two messages transmitted, for *Pipeline* or *Task Farm* respectively, to compensate the wasted time of these. A good agglomeration must make each task work for a significant amount of time.

The second one are the bottlenecks due a high rate of communication to a single task. For instance in a *Pipeline* the first task may process the data very quickly and the second task may take the triple of the time to transform the input, accumulating messages in the second's buffer and introducing delays. The solution is to agglomerate the two tasks in only one, or add more tasks to the second level to process several messages simultaneously.

In a *Task Farm* many results from the workers may come at the same time and the master must process them quickly to avoid that some workers wait too long.

3.3 Performance measures

The performance measure of parallel programs is tricky and non-trivial due to the physical separation of the several tasks.

The performance may be influenced by parameters such as number of nodes, size of the data, amount of memory or cache and communication. For instance if the data structures do not fit in memory in the sequential program but fit in the parallel a speedup is achieved because the access to memory is faster.

Sources of overheads are load imbalances (different work performed by the several tasks), replicated computation and competition for bandwidth (all tasks communicating simultaneously).

3.3.1 Metrics

Four of the parameters most used to estimate performance are execution time, relative efficiency, relative speedup and serial fraction [10].

Definition 3.3.1 (Execution time) *The execution time is the time elapsed since the beginning of the execution in the first processor until the last processor stops and is composed by:*

1. *computation time: time spent performing computations over data;*
2. *communication time: time that takes to send and receive messages;*
3. *idle time: time spent waiting from data sent by other processors.*

Notation 3.3.2 *In the following we will represent T_1 as the execution time of the sequential program and T_p as the execution time of the parallel program on p processors.*

Definition 3.3.3 (Relative Speedup) *Can be classified as sub-linear ($s < p$), linear ($s = p$) or super-linear ($s > p$).*

$$s_p = \frac{T_1}{T_p} \tag{3.1}$$

Definition 3.3.4 (Relative Efficiency) *Efficiency is the quotient between speedup and the number of processors.*

$$e_p = \frac{s_p}{p} = \frac{T_1}{T_p \times p} \tag{3.2}$$

Relative speedup indicates how many times the parallel program was faster while relative efficiency divides that value by the number of processors and shows the relative speedup by processor.

The efficiency and speedup are called absolute when T_1 is the execution time of the fastest sequential algorithm.

Other performance measure is serial fraction proposed by Karp and Flatt in 1990.

Definition 3.3.5 (Serial Fraction) *The serial fraction determines the fraction of the program that is spent performing sequential computing.*

$$f_p = \frac{\frac{1}{s_p} - \frac{1}{p}}{1 - \frac{1}{p}} = \frac{\frac{T_p}{T_1} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{3.3}$$

It can indicate some characteristics of the program:

- if f_p increases as p increases there is an overhead due to load imbalances or synchronizing processors;
- if f_p is negative then there is a super-linear speedup;
- if f_p remains constant as p increases the ideal situation was reached. If e_p decreases that is due to limited parallelism of the program.

3.3.2 Times

There are three types of time: Wall, Elapsed and CPU time.

Definition 3.3.6 (Wall time) *Run time of a process measured by a stop watch.*

Definition 3.3.7 (Elapsed time) *Like Wall time but without the times spent on executing other programs.*

Definition 3.3.8 (CPU time) *Time spent by the CPU on executing the instructions of a program. Can be divided in user time (execution of program's instructions) and system time (execution of the system calls).*

Computers offer timers and clocks to measure CPU and Wall time. Computation time can be measured by the CPU time. Communication time and idle time can only be determined by the Wall time because the program is not running. The Wall time does not provide a good measure for the execution time because it is influenced by the CPU load, i.e. all other programs that are running in the same CPU.

The timing of a program should be done in a dedicated machine, with a small amount of load. It's an artificial environment, but the simpler to be recreated by other user. Should be measured just the CPU time to ignore delays provoked by other programs and because one is just concerned in the time spent by the application. If the Wall time is measured, it should be referred to allow other people to understand the result.

In some situations, we can be only interested in some parts of the program. One can measure only the main algorithm and ignore others, for example the loading of data.

Chapter 4

The SAT-Solver

I can't get no satisfaction!
The Rolling Stones

There are many SAT-solvers available due to new improvements in research made in the last decade. Some of the most popular solvers are Chaff [12], Grasp [11] and MiniSAT [5] that implement the state of the art technologies. For this work we have chosen MiniSAT because it is efficient and well documented.

4.1 General algorithms

In this section we will present the general algorithms and techniques used by most of the actual SAT-solvers and define a set of concepts related with them.

Most of SAT-solvers implement a variation of Davis-Putnam algorithm [4] combined with conflict-driven backtracking, watched literals and dynamic variable ordering. In Figure 4.1 is presented the main loop of a generic SAT-solver.

```
while(true){
        if(!decide()) return SAT;
        while(!BCP()){
                if(!resolveConflict()) return UNSAT;
        }
}
```

Figure 4.1: Davis-Putnam algorithm

The *decide()* function selects an unassigned variable and sets it to *true* or *false* by a heuristic method. This selection may be done by several ways, but the most common are:

- RAND: selects a random variable;

- DLIS (Dynamical Largest Individual Sum): selects the variable that appears more times in the clauses;
- VSIDS (Variable State Independent Decaying Sum): associates a counter to each literal. Every time that a clause is added the counters of its literals are increased. It is chosen the variable with the same polarity of the literal with the highest counter. Periodically, all the counters are divided by a constant.

The *BCP()* function is responsible for carrying the Boolean Constraint Propagation: to identify unit clauses and create implications until there are no more implications or a conflict arises.

Most time of the solvers spend about 90% of their time in *BCP()* [12], so developers try to make the function very efficient.

The decisions are saved in a stack called decision stack. Each is associated an integer tag called *decision level* that corresponds to the height of the decision in the stack. Every implication is related to the corresponding decision by the decision level, what makes easy to find the decision responsible for an implication.

The *resolveConflict()* is responsible to undo all the implications of the current decision level and flip the value of the decision. If both values of the decision have already been tried, is made a *backtrack* in the decision stack (canceling decisions and implications) until find a decision not tried both ways. If no such decision can be found, and we reach the decision level zero, then the problem is UNSAT.

All the canceled decisions are complemented and grouped to form a conflict clause that is added in the database to forbid such decisions. This process is called *learning*.

Definition 4.1.1 (Learnt clause) *A learnt clause is a clause generated by the learning process and contains complemented literals from decisions that led to a conflict.*

For an efficient *BCP()* procedure is need to detect implications easily, i.e., when in a clause with N literals the number of *false* literals goes from N-2 to N-1, meaning that only one literal is not assigned. The watched literals technique to solve this problem consists in select for each clause two unassigned literals and just process the clause when one of the watched literals becomes *false*. In this situation, two things may occur: there are more unassigned literals and another one becomes watched or the clause becomes implied by setting to *true* the watched literal.

Example 4.1.2 *Execution of DPLL algorithm, backtrack and learning on the following formula in CNF format:*

$F = (\alpha \vee \gamma \vee \delta) \wedge (\alpha \vee \gamma \vee \neg\delta) \wedge (\alpha \vee \neg\gamma \vee \delta) \wedge (\alpha \vee \neg\gamma \vee \neg\delta) \wedge (\neg\alpha \vee \beta)$.

1. *assign values to the variables:* $\alpha = false$, $\beta = false$, $\gamma = false$;
2. *conflict when BCP assigns* $\delta = true$ *in first and* $\delta = false$ *in the second clause;*
3. *a learnt clause is added:* $F' = F \wedge (\alpha \vee \gamma)$;
4. *backtrack to assign* $\gamma = true$;
5. *conflict when BCP assigns* $\delta = true$ *in third and* $\delta = false$ *in the fourth clause;*
6. *a learnt clause is added:* $F'' = F' \wedge (\alpha \vee \neg\gamma)$;
7. *backtrack that undoes all assignments;*
8. *assign* $\alpha = true$;
9. *BCP assigns* $\beta = true$;
10. *SAT ! Model:* $\alpha = true$ *and* $\beta = false$.

4.2 MiniSAT

MiniSAT is a SAT-solver written in C++ by Niklas Eén and Niklas Sörensson at Chalmers University in Sweden. Although small with about 600 lines of code, it implements the state of the art SAT–solving techniques [5].

We now present how the previous procedures and techniques are implemented in MiniSAT and explain their particularities.

MiniSAT is also based in the Davis-Putnam algorithm and its propagation is similar to the one described above. It uses the techniques of the watched literals and learnt clauses.

There is a VSIDS activity attached to variables (instead of literals), that is increased when they appear in a learnt clause. All activities are multiplied by a constant less than 1 to decay over time. The next variable to be assigned either is a random variable or the one with the biggest activity.

The new thing in MiniSAT is that learnt clauses are also attached to an activity. When a learnt clause is analyzed its activity increases. To avoid the explosion of the number of this clauses, the database is reduced by half.

Other feature is the possibility to give to the solver a set of literals to be assumed as *true* and search for satisfiability based on that information. When

the search ends the assumptions are undone and the solver returns to initial state even when a contradiction is found (being the result interpreted as UNSAT under assumptions) preserving the database of learnt clauses and filling a vector of conflicts. This vector of conflicts contains a clause contradicting some of the literals assumed, with opposite polarities, that were responsible for unsatisfiability.

Example 4.2.1 *A boolean formula contains the variables φ and ψ. Assuming the literals $\neg\varphi$ and $\neg\psi$ and running the solver, if it returns UNSAT and the vector of conflicts has the literal ψ it means that assuming $\neg\psi$ was enough for the formula be UNSAT.*

The solver is composed by several modules responsible for different data structures and functionalities:

- *Global*: for data types and functions. Specifies a vector data type and lifted booleans (a data type that holds the values *true*, *false* or *undefined*), generation of random numbers, measurement of used resources (memory and CPU time) and memory management.
- *Variables order*: to keep the logic variables ordered. Includes the implementation of a heap.
- *Sorting*: set of functions to sort vectors. Uses *Global* module.
- *Solver*: includes the data types (literals and clauses) and procedures for the SAT-solver algorithm. Uses all other modules.
- *Input readers*: functions to read and parse the input formulas.
- *Standalone application*: program to test satisfiability. Uses all modules above.

We present the mapping between modules/submodules and the files of source code in the following table:

Modules	**File**
Global module	Global.h
Heap submodule	Heap.h
Variables order module	VarOrder.h
Sorting module	Sort.h
Solver methods module	Solver.h and Solver.C
Solver data types module	SolverTypes.h
Input readers	Main.C
Standalone application	Main.C

Table 4.1: Mapping of modules in the file system

Chapter 5

Parallel implementation

If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?
Seymour Cray

This chapter presents the design, structure and details of the parallel algorithm. Is described its design, features, implementation details, application's work flow, modular decomposition and the technology used.

5.1 PCAM design

In this section we describe the design of the parallel algorithm using the PCAM approach.

5.1.1 Partitioning

Due to the architecture of MiniSAT, the partition of the problem cannot be functional, because several of its functions rely on the results of others and must be executed sequentially. They also read and update many data structures in certain sequences in order to keep them coherent.

However, we realize about the complete independence between assignments to variables. Each assignment yields a single boolean value from the formula, independent of any other. The several combinations of assignments can be organized as a binary tree where every branch and subtree are independent of all others, allowing their exploration in parallel.

Definition 5.1.1 (Assumption) *An assumption is a set of literals assumed as true.*

The strategy used makes assumptions over a small set of variables and search for satisfiability on subtrees of the assignments tree. Note that there is no need

to change the solver because, as was said in the previous chapter, it can under assumptions determine whether the problem is SAT or UNSAT. By making a domain partition, in this manner the search space can be split into which can be searched concurrently.

5.1.2 Communication

The most appropriate communication model is to have a master task that sends assumptions to other tasks to test them for satisfiability, i.e. a Task Farm architectural style. This allows to focus the management in one task and the search in the others.

Each worker task communicates only with the master task. After receiving an assumption, the worker runs the solver and reports the solution found (or not) to the master.

This model provides an independent execution of each worker task reducing communication's complexity by avoiding the exchange of messages between workers.

5.1.3 Agglomeration

Here the agglomeration idea is different: the granularity is seen as the amount of generated assumptions. It should be large enough to allow each worker to receive and test more than one assumption. This takes advantage of the reusability of the solver and contributes to balance the time spent on solving the assumptions because each one takes a different time.

5.1.4 Mapping

As much as possible, each task should run on a different machine due to the high processor usage of the solver and because they do not communicate with each other, just with the master. Putting several tasks running in the same machine will only affect their performance because they will dispute the CPU and the memory.

With this design the parallel program has a Task Farm architecture, with domain decomposition and synchronous, static and structured communication and granularity based on the number of generated assumptions.

The organization of the algorithm is presented in Figure 5.1 with the *arrows and boxes* notation, where the arrows represent the direction of communications of data and the boxes represent the tasks of the program.

5.2 Implementation details

In this section are explained in detail the following procedures and features of the application:

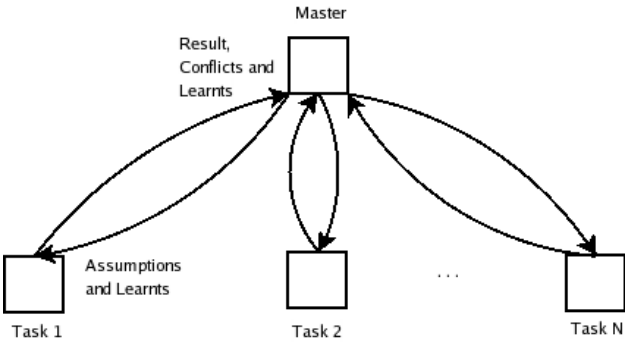


Figure 5.1: Task Farm organization

- variables selection;
- assumptions generation;
- assumptions pruning;
- sharing learnt clauses;
- automatic settings;
- messages.

5.2.1 Variables selection

We can select two types of variables to use in the assumptions: the ones that occur more times or in bigger clauses. The first ones have the objective to affect more clauses, while the second ones are to simplify the biggest clauses.

To do this selection we have two counters for each variable, one for its positive literal and other for its negative literal. As we read the literals of the clauses the respective counter of the variable is increased. For the variables with more occurrences, the counter is increased by one, while for the variables in bigger clauses the counter is increased with the number of literals of the clause. In the end the variables are sorted by the sum of the two counters and the ones with biggest values are chosen.

We use two counters because the program needs to know which literal occurred more times to create the assumptions.

In the following we will refer to the variables with more occurrences, and to their literals, as "the most popular".

5.2.2 Assumptions generation

Given a set of variables, there is more than one way to generate assumptions and explore the assignments tree. It was decided to allow different assumption

generation and work assignment methods, to provide a freedom of choice to the user and to analyze their performance. We propose two major methods to create assumptions subdivided in sub-modes of work assignment:

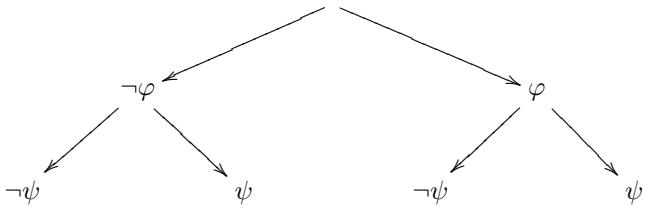
- *Equal*: every assumptions has the same number of literals.
    1. *Random*: the assumptions are chosen randomly.
    2. *Sequential*: the assumptions are chosen in a sequential way.
- *Progressive*: the amount of literals in the assumptions changes.
    1. *Few first*: we start from the assumptions with few literals to those with many literals.
    2. *Many first*: we start from the assumptions with many literals to those with few literals.

In the following subsections we shall analyze and describe each method with more detail. Lets assume that there were already chosen the k most popular variables and we will refer to them just as the k variables.

Equal method

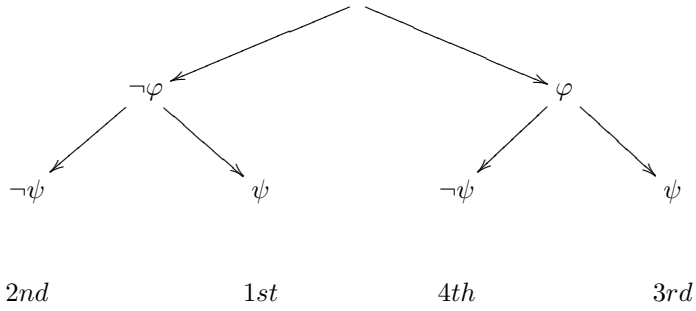
Each assumption is a branch of the assignments tree over the k variables. All the possible combinations are generated, making a total of 2^k different assumptions.

Example 5.2.1 *If the set of most popular variables is* $\{\varphi, \psi\}$*, the four assumptions to test are:* $\{\neg\varphi, \neg\psi\}$*,* $\{\neg\varphi, \psi\}$*,* $\{\varphi, \neg\psi\}$ *and* $\{\varphi, \psi\}$*, arranged in the following tree:*



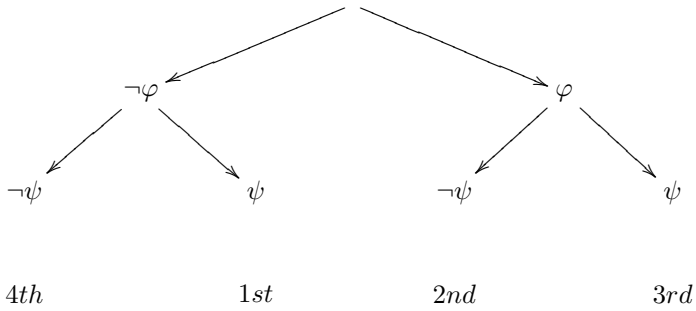
The two sub–modes refer to the testing sequence of the assumptions. In sub–mode *Random* the assumptions are chosen randomly to be tested, to explore at the same time distant subtrees of the assignments tree.

Example 5.2.2 *Choosing randomly the branches:*



In the sub-mode *Sequential* the assumptions are chosen sequentially, starting by the one with the most popular literals and traversing the assignments tree from left to right.

Example 5.2.3 *Sequential mode, having* $\{\neg\varphi, \psi\}$ *as the most popular literals.*



Progressive method

The *Progressive* method is a different approach to explore the assignments tree. While in *Equal* method all the explored subtrees have the same size and the number of assumptions grows exponentially when the amount of their literals increase; in *Progressive* method the intention is to make the number of assumptions grow polynomially as their literals increase but having a variation on the size of the searched subtrees. The main objective is to provide a way to have assumptions with many literals, without having an exponential growth of combinations, to explore more deeply the assignments tree. For k variables this method will create $2 \times k$ assumptions. This is accomplished by generating the assumptions by the following algorithm proposed by us:

Proposition 5.2.4 *The following algorithm provides a set of* $2\times k$ *assumptions, with an amount of literals ranging from* 2 *to* k*, that covers the entire assignments tree:*

1. Organize the most popular literals by the same order, in a list indexed from 1 to k;
2. for each $i-th$ literal from the list, $1 < i \leq k$, create an assumption containing:
   (a) all the previous $j-th$ literals, $1 \leq j < i$;
   (b) the complement of the $i-th$ literal;
3. create an assumption with all the k literals;
4. for each of the previous assumptions, make new ones by complementing their first literal.

Proof: we will prove that the phases 2 and 3 of the algorithm generate assumptions that cover exactly half of the assignments tree and the phase 4 generates the assumptions of the other half.

Proof by induction on the number of literals k in the list.

Base: $k = 1$

1. the list has one literal: $\{\varphi_1\}$.
2. the cycle does not create any assumption because there is no 2nd literal.
3. creates the assumption $h_1 = \{\varphi_1\}$. h_1 covers one branch (half) of the assignments tree for φ_1.
4. creates the assumption $h_2 = \{\neg\varphi_1\}$ that covers the other branch of the tree.

The induction hypothesis asserts that for $k = n$, with a list of n literals $\{\varphi_1, \ldots, \varphi_n\}$, the steps 2 and 3 generate the assumptions $h_1, \ldots h_n$ that cover half of the assignments tree. Complementing the first literal (φ_1) that appears in all $h_1, \ldots, h_n$ by construction, we cover the other half of the tree.

Step: $k = n + 1$

1. the list has $n + 1$ literals $\{\varphi_1, \ldots, \varphi_{n+1}\}$.
2. generates the assumptions $\{h'_1, \ldots, h'_n\}$ where
   $h'_1 = h_1, \ldots, h'_{n-1} = h_{n-1}$ by construction and
   $h'_n = \{\varphi_1, \ldots, \varphi_n, \neg\varphi_{n+1}\} = h_n \cup \{\neg\varphi_{n+1}\}$.
3. creates the assumption $h'_{n+1} = \{\varphi_1, \ldots, \varphi_n, \varphi_{n+1}\} = h_n \cup \{\varphi_{n+1}\}$.
   The assumptions h'_n and h'_{n+1} extend the assumption h_n that covered the longest branch of that half of the tree by including the two combinations for φ_{n+1}. For induction hypothesis as $h_1, \ldots, h_n$ cover half of the assignments tree then $h'_1, \ldots, h'_{n+1}$ also cover half of the assignments tree.

4. generating $n+1$ assumptions by complementing the literal φ_1 of the previous assumptions, the other half of the tree is also covered.

QED

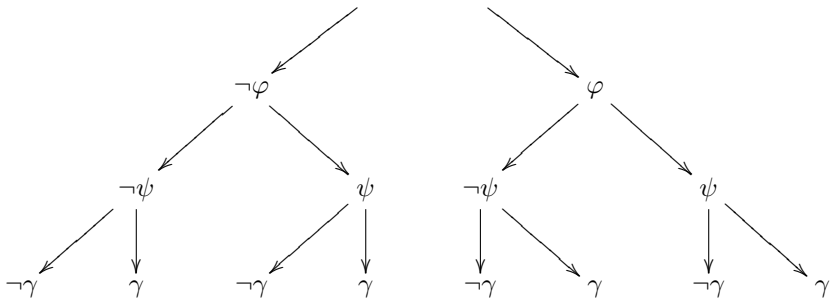
Example 5.2.5 *Consider the most popular literals φ, ψ and γ. Applying the steps of the method:*

1. *Get the list $\{\varphi, \psi, \gamma\}$;*
2. *the assumption for $i = 2$ is $\{\varphi, \neg\psi\}$ and for $i = 3$ is $\{\varphi, \psi, \neg\gamma\}$;*
3. *create the assumption $\{\varphi, \psi, \gamma\}$;*
4. *create the assumptions $\{\neg\varphi, \neg\psi\}$, $\{\neg\varphi, \psi, \neg\gamma\}$ and $\{\neg\varphi, \psi, \gamma\}$.*

Six assumptions were created:

1. $\{\varphi, \neg\psi\}$;
2. $\{\varphi, \psi, \neg\gamma\}$;
3. $\{\varphi, \psi, \gamma\}$;
4. $\{\neg\varphi, \neg\psi\}$;
5. $\{\neg\varphi, \psi, \neg\gamma\}$;
6. $\{\neg\varphi, \psi, \gamma\}$.

As we can see, the assumptions (with the correspondent number under the branches) cover all the assignments tree:



The first and fourth assumptions explore a large subtree composed by two branches.

The two sub–modes allow to decide whether to start from the assumptions with two literals (*Few first*) or from those with k literals (*Many first*).

5.2.3 Assumptions pruning

As was said in the previous chapter, in case of finding a contradiction under assumptions the solver may fill the vector of conflicts with a subset of the assumed literals, with the opposite polarity, responsible for that contradiction.

The polarity of the vector's literals is again reversed to get the original ones and they are added to the result message sent to the master that erases all the assumptions not yet tested that contain them. The objective is to invalidate branches from the search tree even without evaluating them in order to save time.

The vector of conflicts is sent to the master incorporated in the result message that has a fixed size and includes an array with twenty slots for it because are enough for most of the vectors and just one message will be sent most of the times. To assure the communication of vectors with any size, a protocol has been made to break a larger vector through several messages and be rebuilt by the master. Including the vector in the result message avoids sending another one that would introduce more delays.

5.2.4 Sharing learnt clauses

A mechanism to share learnt clauses between the workers was implemented. If the option is enabled the worker sends to the master a set of learnt clauses after finishing a search. Each set may have a maximum number of clauses, each one with a limited size (amount of literals). These restrictions put a threshold on the amount and size of the message sent, because there may exist clauses with hundreds of literals. The selection of the learnt clauses is made after sorting by activity the array where they are kept, traverse it from end to the middle (to share only the most active ones) and choose a limited number of those with less or equal size than the maximum allowed.

The master has a database for each worker where it stores the most recent set of learnt clauses sent by that worker. The idea is to have always the most recent ones so the old ones are replaced by new ones that arrive.

When the master has to send a set of learnt clauses to a worker he chooses one that has not been sent to that worker yet. So the receiver will always get different information.

As was said in the previous chapter, there is an activity associated to each learnt clause. When a worker sends a set of learnt clauses to the master their activities are not included. The responsible for assigning activities to that set is the worker that receives it to respect the scale of values of the existing learnts. The activities of the new learnt clauses are set to the maximum value of those stored in the worker to give them some relevance and to avoid being removed in the next solver's learnts cleanup.

Besides sharing, the worker may also remove all the learnt clauses from the solver after each execution or leave them to be used in the next one.

5.2.5 Messages

The communication between master and workers is made by message passing. This system only allows messages composed by primitive types or structures of them. Therefore, all the objects – literals and clauses – had to be encoded as integers. The variables are already represented internally as integers bigger or equal than zero. To encode a literal object as integer we need to relate its variable and polarity. One might think to encode as the variable with positive or negative sign, but there is the special case of variable 0 (zero). Our solution is to increase the variable by one and add a sign if the literal is negative.

There are four types of messages: request for work, result, set of learnt clauses and model.

Request for work

This is a message sent from the master to a worker containing one assumption to be made by the solver, i.e., a set of literals encoded as integers. Due to the fact that the *Progressive mode* uses assumptions with different amount of literals we had two alternatives: to send messages of different sizes or to send a message with a fixed size and a special value to indicate the end of assumption. We choose the second alternative to avoid being always asking for the size of the message. The end of assumption value is the only one that is not used in the encoding: zero.

The message with assumptions containing k literals has size k:

lit_1	lit_2	lit_3	...	lit_k

Table 5.1: Message for *Equal mode*

lit_1	...	lit_n	0	...

Table 5.2: Message for *Progressive mode*, assuming $n < k$ literals

Result

The result is a message, or a set of messages sent by the worker for the master after solving part of the problem. It is composed by the result (SAT or UNSAT) encoded as 1 or 0 respectively, a flag to indicate if there are more messages, the CPU time spent by the worker to solve the last assumption, an array for the literals returned by the solver as responsible by the conflict and their amount.

All the fields are integers except the time that is a double.

We say that the result may need to be sent in more than one message because the conflicts may be more than the size of the array. The array has a fixed size of 20 and is big enough for the most of the cases, so generally only one message should be sent.

SAT or UNSAT	flag	time	literals in array	array of literals

Table 5.3: Result message

Set of learnt clauses

The sets of learnt clauses are exchanged between the workers and the master. The workers send the learnt clauses with more activity and the master keeps and shares them with other workers.

The clauses, whose literals are encoded by the way described above, are separated by the value zero.

$clause_1$	0	...	$clause_n$	0

Table 5.4: Set of learnt clauses

A limit is imposed to the size of each clause to avoid very long messages and to know how many memory must be allocated to save the set.

Model

The model is the assignment that makes the formula satisfiable. It is not mandatory to have all the variables assigned, being the model the smallest subset of variables that makes the formula being *true*.

So the message with the model contains the assigned variables, that are encoded as literals as described above to represent the values *true* and *false* in the assignments.

lit_1	lit_2	lit_3	...	lit_n

Table 5.5: Model of the formula

5.2.6 Automatic settings

When the search mode or the number of variables to assume are not specified, they are determined automatically by the program, based on the number of workers, the assumptions–CPUs ratio (*acr*) and the search mode. The *acr* indicates the intended relation between the amount of assumptions and workers, i.e., how many assumptions should ideally each worker solve.

Given n workers and an acr, we determine the amount V of variables to assume for *Progressive* or *Equal* mode by the following formulas.

$$V_{progressive} = \lceil n \times acr/2 \rceil \tag{5.1}$$

$$V_{equal} = \lceil \log_2 (acr \times n) \rceil \tag{5.2}$$

In *Progressive* mode the number of assumptions is $2 \times V_{progressive} \approx acr \times n$.

In *Equal* mode the number of assumptions is $2^{V_{equal}} = 2^{\lceil \log_2 (acr \times n) \rceil} \approx acr \times n$.

When only the number of variables is given the search mode is set as *Random* if $2^V \leq acr \times n$, else is set as *Many first*.

When none of the parameters are given, the search mode is set to *Random* and $V = V_{equal}$.

5.3 Application's work flow

```
bool test4SAT(){
  generate all assumptions;
  send one assumption for each worker;
  do{
    receive a result from a worker;
    if(result is SAT) return SAT;

    if(sharing learnt clauses)
      receive learnt clauses and add them into the database;

    if(conflicts enabled)
      erase the assumptions containing the conflicts;

    if(more assumptions to try){
      if(sharing learnt clauses)
        retrieve from database a set of learnt clauses and send it;
      send another assumption;
    }
  }while(not tested all assumptions);
return UNSAT;
}
```

Figure 5.2: Function test4SAT()

```
main(){
Solver S;
  parse parameters;
  read input file into S;
  automatically calculate the missing parameters;

  if(local mode) S.solve() and output result;

  else{ //parallel mode

       if(master){//master code
         choose the most popular variables;
         result = test4SAT();
         output result;
         if(should write the model)
           receive and write the model;
         abort computation;
       }//end master

       else{//worker code
         while(true){
           receive assumption;

           if(sharing learnt clauses){
             receive a set of learnt clauses;
             insert the learnt clauses into S;
           }

           result = S.solve(assumptions);

           if(result is SAT){
             send result;
             if(should write the model) send the model;
           }

           if(sharing learnt clauses)
             get a set of learnt clauses from the solver and send them;

           if(removing all learnt clauses)
             delete all learnt clauses from S;

           if(conflicts enabled)
             get the set of conflicts and insert them in the result;

           send result;
         }//while
       }//end else worker
     }//end else parallel
}
```

Figure 5.3: Program's main()

The application can be divided in two parts: the master and the worker, both sharing a common initialization. Their pseudocode is in Figures 5.2 and 5.3.

The master sends assumptions that the workers use to restrict the search space. After solving, each worker sends learnt clauses or conflicts, if those options are enabled, the result of the search (SAT or UNSAT) and the model (if requested and the result is SAT). The master saves the learnt clauses in a database, removes assumptions that could contain conflicts and sends other set of learnt clauses and assumption to test. When one worker reports SAT or all the assumptions are reported as UNSAT, the master aborts the computation.

The solver of the worker, by default, preserve the learnt clauses to reuse in the next search, but may remove them if the user wishes to.

5.4 Modules of the program

The number of modules has grown significantly:

- *Global*: for data types and functions. Specifies a vector data type and lifted booleans (may assume the values *true*, *false* or *undefined*), generation of random numbers, measurement of used resources (memory and CPU time) and memory management.
- *Variables order*: to keep the logic variables ordered. Also implements a data structure for a heap.
- *Sorting*: set of functions to sort vectors. Uses *Global* module.
- *Solver*: includes the data types (literals and clauses) and methods for the SAT-solver algorithm (new clause, propagate, search, solve). Uses all modules above.
- *Learnts database*: to save and retrieve sets of learnt clauses.
- *Assumptions generation*: to generate assumptions.
- *Counter of variables*: to count the occurrences of each variable in the formula.
- *Statistics*: to register the options chosen as well as execution and communication times.
- *Arguments parser*: to parse the arguments given to the program.
- *Input readers*: functions to read and parse the input formulas.
- *Standalone application*: program to test satisfiability. Uses all other modules.

The module *Arguments parser* belongs to the application `Arg_parser` written by Antonio Diaz Diaz and released under GNU General Public License.

The mapping of the modules into the file system is shown in table 5.6.

Modules	File
Global module	Global.h
Heap submodule	Heap.h
Variables order module	VarOrder.h
Sorting module	Sort.h
Solver methods module	Solver.h and Solver.C
Solver data types module	SolverTypes.h
Learnts database	LearntsDB.h and LearntsDB.C
Assumptions generation	Assumptions.h and Assumptions.C
Counter of variables	OccurVar.h
Statistics	Statistics.h and Statistics.C
Arguments parser	arg_parser.h and arg_parser.C
Input readers	Main.C
Standalone application	Main.C

Table 5.6: Mapping of modules in the file system

5.5 Technology

The implementation of the program was written in C++, to extend the original MiniSAT v1.14, and uses MPI – MPICH 1.2.6 – functions to establish the parallelism and communications between tasks. This program may be executed in clusters or grids with any dimension being completely scalable to any number of CPUs by choosing the right amount of variables to assume.

Chapter 6

Experimental results and performance analysis

You never really know how quick you are before you reach F1.
Jean Alesi

In this chapter we present and analyze the performance achieved by the program developed. As benchmarks we used 25 (7 UNSAT and 18 SAT) files taken from the Internet and SAT competitions. These files were solved by the program using the different search modes and options to see the different performances and load balances attained. The files, its solutions, amount of variables and clauses were are presented in Table 6.1.

6.1 Grid resources

The performance tests were made in the Grid Infrastructure available at INESC–ID, using the 11 nodes available. These nodes have a homogeneous architecture with Pentium 4 processors running at 3.2 GHz, 1 GB of RAM, Linux operating system, MPICH 1.2.6 and the same hard drive shared via NFS.

The advantage of homogeneity is that eliminates delays caused by slower machines that distort the results attained in heterogeneous architectures.

6.2 Methodology

To test the effectiveness of the parallel algorithm it was decided to compare the time spent by the sequential version of MiniSAT and the executions of the parallel program called with different options. To do so, four sets of tests were devised.

File	Solution	Variables	Clauses
fpga10_11_uns_rcr.cnf	UNSAT	220	1122
fpga10_12_uns_rcr.cnf	UNSAT	240	1344
fpga10_13_uns_rcr.cnf	UNSAT	260	1586
hole11.cnf	UNSAT	132	738
mod2-3cage-unsat-9-11.cnf	UNSAT	87	232
mod2-3cage-unsat-9-4.cnf	UNSAT	87	232
unif-r4.cnf	UNSAT	400	1700
frb40-19-1.cnf	SAT	760	43780
frb40-19-2.cnf	SAT	760	43780
frb40-19-3.cnf	SAT	760	43780
frb40-19-4.cnf	SAT	760	43780
frb40-19-5.cnf	SAT	760	43780
mod2-3g14-sat.cnf	SAT	192	768
mod2c-rand3bip-sat-150-11.cnf	SAT	212	1520
mod2c-rand3bip-sat-150-15.cnf	SAT	213	1528
sat2.cnf	SAT	283	1358
unif-r5.cnf	SAT	251	323
vmpc_21.renamed-as.sat05-1923.cnf	SAT	441	45339
vmpc_23.renamed-as.sat05-1927.cnf	SAT	529	59685
vmpc_25.renamed-as.sat05-1913.cnf	SAT	625	76775
vmpc_25.shuffled-as.sat05-1945.cnf	SAT	625	76775
vmpc_26.renamed-as.sat05-1914.cnf	SAT	676	86424
vmpc_26.shuffled-as.sat05-1946.cnf	SAT	676	86424
vmpc_27.renamed-as.sat05-1915.cnf	SAT	729	96849
vmpc_27.shuffled-as.sat05-1947.cnf	SAT	729	96849

Table 6.1: Benchmark files

The first two tests, with just one worker task, have the objective of determining the influence of communication delays.

In the following tests the number of nodes varies between 3 and 11. This means that the amount of worker tasks varies between 2 and 10, the remaining node being used as master.

The third test is to determine the influence of having different numbers of nodes to solve a formula. The number of variables to assume is kept constant. All the search modes are tested with no other options, sharing learnt clauses or pruning/removing assumptions with conflicts.

We decided to assume 6 variables because they provide an adequate amount of assumptions generated by the search modes. Also the performance varies with that amount and there is no particular value that is more accurate than others.

The fourth test is to see the influence of solving a formula with different amounts of variables. Are tested all the search modes with no other options. The amount of variables is automatically calculated according to the *acr* that is equal to the number of worker tasks.

The tests are summarized as follows:

- the master with one worker, both in the same node;
- the master with one worker in a different node;
- for an amount of worker tasks between 2 and 10, for each search mode, assuming 6 variables and:
  1. no other options;
  2. removing assumptions with conflicts;
  3. sharing learnt clauses.
- for an amount of worker tasks and assumptions–CPUs ratio between 2 and 10 (2 workers, $acr = 2$; 3 workers, $acr = 3$; ...), for each search mode and no other options.

6.3 Goals and difficulties

The purpose of this analysis is to study the achievements of the parallel algorithm on:

- effectiveness (speedup, efficiency and serial fraction);
- performance of the different search modes;
- influence of sharing learnt clauses;
- consequences of reduce the search space by removing assumptions with conflicts;
- load distribution between the workers.

Being a search problem and depending on the structure of the boolean formula and on the settings of the SAT–solver (like the amount of variables in assumptions, the number of workers and the search mode) there is no way to predict:

- how much time the algorithm will take;
- if the problem to solve is easy or hard;
- the best search space partition;

- the better search mode;
- who solves each assumption;
- the part of the assignments tree where the solution is.

Thus the results of the performance study, are difficult to be generalized.

6.4 Time measurement

The execution time is one of the most difficult things to measure in parallel programs due to the overlapped computation, communication delays and waits for work.

To make a coherent analysis and comparison between the original MiniSAT and the parallel version, the time measurements must follow the same methods and use the same tools.

MiniSAT measures the CPU time, with the system call `getrusage()`, for the whole execution: the time spent reading the input, solve the problem (find the first solution or UNSAT) and output the result. To be coherent with MiniSAT, the parallel program also measures the CPU time spent until it finds the first solution.

This time must be measured in the master and in the workers and just reflects the effort to find the solution ignoring delays and idle times (that can only be measured with the Wall time).

A software module was designed to record the statistics containing a database for each worker with: its total CPU time, total CPU time spent by the master to process its responses, number of executions completed, amount of sets of learnt clauses sent to and received from the master. It also records the initialization and finalization CPU time spent by the master and the overall Wall time.

The total time is the sum of initialization, finalization, CPU time of the most occupied worker and CPU time spent by the master to process the responses of that worker.

Figure 6.1 shows the interaction between the master and one worker, indicating with a gray fill the timed operations.

The time spent by the worker to send the result is not measured because, as was said, the execution time goes in that message. We do not care about the time spent by the workers on initialization because the master spends even more and we want the maximum.

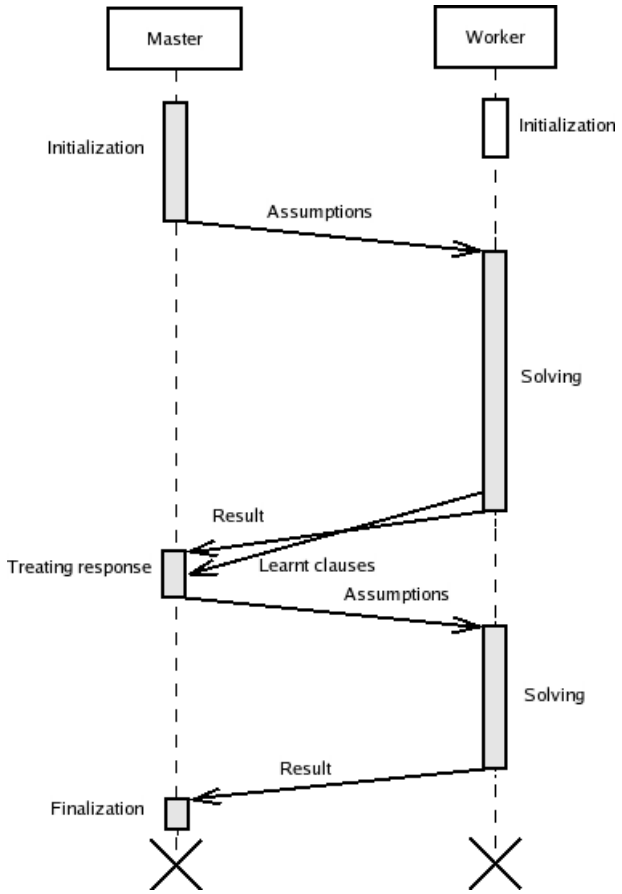


Figure 6.1: Timed operations (gray fill)

Communication delays and the load of the system are not measured by the CPU time. To have an idea of the influence of these two factors, one should compare the Wall time with the Total time.

6.5 Results

In this section we will present the results of the tests over the formulas displayed in Table 6.1. All the times of the tables are in seconds.

The following subsections present the results of the measurements described in the section Methodology.

For the tests with one worker we will discuss the influence of the communication in the time spent on computing.

The tests to different modes and options and to granularity describe and compare the best, worst and average time spent on computing.

The average is made to the time spent to solve all files. Are presented average times for 3, 6 an 9 workers with different search modes, options and granularity. These amounts were chosen to illustrate the variation in the number of workers.

Finally are described the types of load distribution that we have registered among the nodes.

6.5.1 Communication delay

Tables B.1 and B.2 show the results of solving the test files in the sequential MiniSAT and in the parallel version with the master and one worker, running in the same or in distinct nodes, to see the influence of the communication (comparing the CPU and Wall time).

Although the Grid is shared among several users, most of the tests were made in unloaded conditions almost without the interference of other CPU demanding programs. This allowed to get a Wall time very similar with the CPU time.

The options of the parallel version where: 6 variables assumed and *Sequential* mode, while all the others had their default values. The maximum number of requests would be $2^6 = 64$ and the number of messages (2 per request) would be 128.

The influence of the communication, partially determined by the difference between the Wall and CPU time, is minimal in the order of tenths or hundredths of second. This is due to the small number and size of the messages sent.

Due to the random behavior of the solver, the times spent in the local test and in the remote test were different and did not allowed to determine the influence of the time spent on communication, but it is irrelevant because the gains achieved are in orders of sets of ten or hundreds of seconds. There is no problem spending two or three seconds in communication because the gains, when they exist, are widely superiors.

6.5.2 Modes and options

This subsection presents the performance achieved with the test files, with the 4 search modes, an amount of workers between 2 and 10, assuming 6 literals and without options, removing assumptions with conflicts or sharing learnt clauses.

About the best performances (in Tables B.7, B.9 and B.11), most of them occurred with a significant number of workers. The most frequent search modes were *Random* and *Few first*, while the *Many first* and the *Sequential* were only registered a couple of times.

Super–linear speedups occurred frequently, most of them in SAT files. The most incredible speedups were achieved when files that took more than 500,

1000 or even 5000 seconds in the sequential MiniSAT were solved in few seconds by the parallel version, meaning that the partition holding the solution was quickly (and one of the firsts being) explored.

But the parallel solver did not always solved the problems faster. In many situations it took a longer time than the sequential version and sometimes up to 10 times more. Most of the worst times were registered with few workers due to the long computations that occupied them excessively. The worst times are in Tables B.8, B.10 and B.12.

Table B.3 and Table B.4 show that the average time per file decreased from more than 800 seconds taken by the sequential MiniSAT to about 500 seconds taken by the parallel program. The times in Table B.4 indicate that the average times in the modes *Few first* and *Sequential* decreased when the workers increased. The more significant decrease happened with *Few first* mode. The average times of the mode *Many first* decreased from 3 to 6 workers but increased from 6 to 9 workers. With the *Random* mode, the average times increased from 3 to 6 workers and decreased from 6 to 9 workers. There were registered few gains due the share of learnt clauses or the removal of assumptions with conflicts.

The best speedups achieved with the UNSAT files had values around between 1 and 4, i.e sub-linear, with the exception of the file `unif-r4.cnf` with 16. In contrast almost all speedups with SAT files were super-linear, indicating that better performances might be achieved when the solution exists.

The average times for each SAT file varied between 150 and 650 seconds, very inferior to the 916 seconds taken by the sequential MiniSAT. The UNSAT files were solved quickly, with average times varying between 125 and 370 seconds. The average time to solve them with sequential MiniSAT was about 640 seconds.

It was noticed that sometimes the options to remove assumptions with conflicts and share learnt clauses introduced significant speedups, while in the rest of the times the performance remained equal or got even worse.

There are two kinds of benefits when removing assumptions: small time is saved with those that lead to immediate contradictions and many time is saved with those that may take long. But the reduction of tests stops the solvers of increase their database of learnt clauses.

Although the learnt clauses help to guide the search, when they are shared and reused in different conditions may detect conflicts where they do not exist and cause delays.

The amount of assumptions with conflicts removed was significant in the *Random* and *Sequential* modes (more than 30), but as was said, the execution time suffered few speedups. In the *Few first* and *Many first* modes, the amount of erased assumptions was smaller, due to its inferior granularity (12 assumptions to test).

Also the sharing of learnt clauses was not always done in spite of being enabled, because their size exceeded the maximum size allowed. Another fact registered was the difference between the number of sets of learnt clauses sent and received by the workers, due to load imbalances where one or two workers solved most of the assumptions and the others solved few. But there are some examples where the sent and received set of learnt clauses are similar, like with the file `mod2-3cage-unsat-9-11.cnf` in Table B.24 for instance.

In some files like `mod2-3g14-sat.cnf` (Table B.26), none assumption was removed due to lack of conflicts or by the fact that their literals matched with the entire assumption. The file that did not share learnt clauses nor erased assumptions is `unif-r5.cnf`, displayed in Table B.31.

6.5.3 Granularity

This subsection presents the performance of the tests to measure the influence of granularity: the number of variables assumed and, therefore, the amount of assumptions produced.

The conditions assumed for the tests were an equal amount of workers and assumptions–CPUs ratio between 2 and 10 (2 workers, acr = 2; 3 workers, acr = 3; ...), for each search mode, calculating automatically the number of literals to assume and no other options (conflicts or learnts).

The best results for each file are displayed in Table B.13. These performances registered 21 better times than those with 6 variables and without options (Table B.7), in some cases with significant difference between both.

As before, most of the best times were registered with many workers (more than 7), while the search modes that occur more often are now *Many first* and *Few first* with a high number of variables (like 41 and 50).

The amount of super-linear speedups achieved with UNSAT files increased to 4, but the difference between the speedups achieved with SAT and UNSAT files remained.

Also more than half of the worst performances (in Table B.14) exceeded the time spent by the sequential MiniSAT, some of them by a large difference. There is not any predominant search mode because *Many first*, *Random* and *Sequential* appeared 9, 7 and 6 times respectively, while *Few first* just appeared 3 times. The amount of workers was by 18 times less or equal than 4.

The average time per file with different granularity (Table B.4) also decreased to values between 95 to 550 seconds when compared with the 840 seconds in Table B.3. The averages of the *Few first* and *Many first* decreased when the workers increased. On the contrary, the average time of *Sequential* mode became bigger with the increase of the number of workers.

The averages for SAT files, Table B.5, varied between 32 and 700 seconds, yet inferior to the 916 seconds in Table B.3. The UNSAT files had average times between 125 and 360 seconds, very inferior to 640 seconds from Table B.3.

These results show that granularity is so important as the search mode and plays a major role in the effort to find the solution.

6.5.4 Load distribution

In our context we will define the load of a worker as the total amount of CPU time spent on computing. The load distribution, or the time that each worker spends on computing, depends on the time to solve the generated assumptions.

Coinciding the fact that each assumption takes a different time to be solved and there is no mechanism for automatic load balance, was expected potential load imbalances between the workers. But during our tests we found all types of load distributions. We will now report several situations that occurred, although they should not be generalized.

With UNSAT files we found that with few workers the time taken by them was similar although had solved different amounts of assumptions. When the number of workers increased, the load distribution sometimes diverged with the *Few first* and *Many first* modes (some workers ended quickly and others spent much time with few assumptions) or remained similar with the *Random* and *Sequential* modes.

With SAT files, we found three situations: the solution was in the hardest partition and one worker spent much time to find it while the others finished after some seconds; there was a balance and all the workers took similar times; or one worker found the solution while all (or some part of) the others were still solving their first assumption.

The above results show that is hard, or even impossible to predict how the load will be distributed. The positive point of the imbalances is that the resources may be released sooner and the worker used for other purposes.

Chapter 7

Conclusion

This parallel version of MiniSAT gave us an indication of how good is the contribution of parallel computing in this field. It was useful because it showed how a simple idea like domain decomposition can sometimes improve the search. The differences in the performances obtained indicate that they depend on the boolean formula, the search mode and the amount of variables used in assumptions and it is impossible to predict the program's behavior.

The parallel search allowed to solve some files in seconds if the right partition was explored. As a consequence, super-linear speedup is an achievable reality.

There is no best search mode, because the results show that with 6 variables the predominant modes were *Random* and *Few first*, but with different amounts of variables the modes *Few first* and *Many first* emerged as the fastest, indicating that the *Progressive* method should be considered as a serious alternative to *Equal*.

In the tests of granularity most of the best performances were achieved with a large number of variables and workers. This seems to be a good combination to get significant performance improvements.

The average times shown how the time to solve all the files decrease almost by half, or even more, when the parallel program is used.

Features such as removing assumptions with conflicts or sharing learned clauses, presented few good results and did not influence the performance as expected, because we were hoping that they could improve the search and reduce the time more often.

For future research we might suggest several decisions and features to be implemented and studied, like different heuristics to select the variables, new methods to partition the search space or a system of load balancing.

Bibliography

[1] J. L. Balcázar, J. Díaz and J. Gabarró. *Structural Complexity I*, Springer-Verlag, 1995.

[2] W. Chrabakh and R. Wolski, *GrADSAT: A Parallel SAT Solver for the Grid*, Proceedings of IEEE SC03, November 2003.

[3] S. Cook. *The complexity of theorem proving procedures* in Proceedings of the third annual ACM Symposium of Theory of Computing, 1971.

[4] M. Davis, G. Logemann, and D. Loveland. *A machine program for theorem proving*, Communications of the ACM, (5):394-397, 1962.

[5] N. Eén and N. Sörensson. *An extensible SAT-solver* in SAT 2003 Volume 2919 of LNCS, Springer (2004) 502–518.

[6] S. L. Forman and A. M. Segre, *NAGSAT: A Randomized, Complete, Parallel Solver for 3-SAT*, Fifth International Symposium on the Theory and Applications of Satisfiability Testing, May 2002.

[7] I. Foster. *Designing and Building Parallel Programs*, Addison-Wesley, 1995. Online version at `http://www-unix.mcs.anl.gov/dbpp/`.

[8] I. Foster. *What is the Grid? A three point checklist*, Argonne National Laboratory, 2002.

[9] B. Jurkowiak, C. M. Li, and G. Utard, *Parallelizing satz using dynamic workload balancing*, Electronic Notes in Discrete Mathematics, vol. 9, Elsevier Science Publishers, 2001.

[10] A. H. Karp and H. P. Flatt, Measuring parallel processor performance, Comm. ACM 33 (5) (1990), pp 539-543.

[11] J. P. Marques Silva and K. A. Sakallah. *GRASP - A New Search Algorithm for Satisfiability* in ICCAD. IEEE Computer Society Press, 1996.

[12] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik. *Chaff: Engeneering an Efficient SAT Solver* in Proc. of the 38th Design Automation Conference, 2001.

[13] M. Sipser. *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.

[14] D.H.M. Spector. *Building Linux Clusters*, O'Reilly, 2000.

Appendix A

User manual

This manual describes how to use the parallel version of MiniSAT, to provide a clear description of its usage, functionalities and options.

A.1 How to use this manual

If you intend to install the program start reading from the following section; otherwise if you want to learn how to use it, start reading from the section Usage. There is also a quick start guide that explains the basic usage. The inputs and outputs of the program are explained as well as error messages. For doubts consult the FAQ section.

A.2 System requirements and installation

The program should be executed in a cluster or grid containing MPI 1.0 or higher.

To install just extract all the source files to one directory and compile them with the `make` command. You may need to edit the `Makefile` to set the variable `INCLUDE_DIR` where the directory `/mpich/include/` or other similar is located. The executable – `parallel_minisat` – may need to be installed in every computer if they don't share a common file system.

A.3 Quick start

The `parallel_minisat` is executed by the `mpirun` command:

```
mpirun -np number-of-CPUs parallel_minisat [options] input-file [output-file]
```

The parameter `number-of-CPUs` is used to set the number of copies of the program to execute. p CPUs correspond to one master to manage the search and $p-1$ workers to solve the boolean formula.

The formula to test is given in a file with extension `.cnf` or `.bcnf`. This file must be placed in all computers (same directory) if they don't share a common file system.

The most important options of the program are `-n` and `-m` to set the number of variables to assume and the search mode respectively.

For k variables, the search modes *Sequential* (`s`) and *Random* (`r`) make 2^k tests while the *Few first* (`f`) and *Many first* (`m`) make only $2 \times k$ tests.

Example A.3.1 *Setting the number of CPUs to 4 (3 workers and the master), the search mode to* Sequential *and the number of variables to assume to 5.*
`mpirun -np 4 parallel_minisat -m s -n 5 file.cnf`

Example A.3.2 *Setting the number of CPUs to 9 (8 workers and the master), the search mode to* Many first *and the number of variables to assume to 12.*
`mpirun -np 9 parallel_minisat -m m -n 12 file.cnf`

If you provide a name for the output file, a file will be created with the result UNSAT or SAT and the model.

To abort the execution of the program press the keys `Ctrl-C`.

A.4 Usage

The `mpirun` command is used to run the program:
`mpirun [mpirun_options...]  <progname> [options...]`

Its main options are `-np` to indicate the number of processors where the program will run and `-machinefile` to indicate a file with the name of the servers where the application will run. p CPUs correspond to one master to manage the search and $p-1$ workers to solve the boolean formula.

The program `parallel_minisat` is executed through `mpirun`:
`mpirun -np number-of-CPUs parallel_minisat [options] input-file [output-file]`

If the computers of the cluster or grid do not share a common file system, the input file must be copied to every nodes where the program will run and placed in the same directory.

Several behaviours and features can be enabled by the options or from a configuration file described in the next section.

The minimal set of arguments you must give to invoke the program are:

1. amount of CPUs (k workers + 1 master);
2. name of the executable (`parallel_minisat`);
3. input file.

The optional arguments are:

1. file with a list of machines that will run the program;
2. search mode, number of variables and other options;
3. output file.

The program may be executed without the `mpirun` command and perform a sequential search in the local machine, like the original MiniSAT. Its usage is `parallel_minisat -m l input-file [output-file]` where `-m l` stands for *Local* search mode.

The program may be interrupted if you press `Ctrl-C`.

A.5 Options

To see the full list of options of the program (in Figure A.5) type `parallel_minisat -h`. It is not necessary to use the `mpirun` command.

Extra features are set as flags: `-c` to remove assumptions with conflicts, `-l` to share learnt clauses and `-r` to remove the learnt clauses after each execution of the solver.

To configure parameters we have: `-s` to set the method to choose the variables to assume, `-z` and `-t` to set the maximum size and amount of learnt clauses and `-a` to set the assumptions–CPUs ratio (*acr*).

All the previous options when omitted, assume a default value or retrieve it from a configuration file if available. The purpose of the file is to define new default values for parameters and flags without setting them in the command line. But if a configuration file is loaded and some option is given in the command line, it overrides the value in the file.

To manage configuration files there are the options `-g` and `-f` to generate and import configuration files respectively.

There are some precautions to have when editing the file: each line may have a maximum number of 60 characters and there can't be any space between the name of the option and the equal sign, because of the simplicity of the parser.

Example A.5.1 *To create a configuration file.*
`parallel_minisat -g minisat.conf`
The configuration file created by the program is displayed in Figure A.2.

Example A.5.2 *To load a configuration file and override the size and max amount of learnt clauses.*
`mpirun -np 4 parallel_minisat -f minisat.conf -z 40 -t 100 file.cnf`

```
USAGE: mpirun -np number-of-CPUs ./parallel_minisat [options] input-file [output-file]
Options:
  -h, --help  display this help and exit

  -v, --verbose  enable the verbose mode

  -n <value>, --number-of-vars  the number of variables to assume

  -m <arg>, --mode  assumptions generation / search mode:
         l - local execution without assumptions just with 1 CPU
         Progressive mode has <arg>:
         f - start from the assumptions with few literals
         m - start from the assumptions with many literals
         Equal mode has <arg>:
         r - test the assumptions randomly
         s - test the assumptions sequentialy

  -f <file>, --config-file  read a given configuration file.

  -g <file>, --generate-config  generate a configuration file and exit. The
program is able to work without a configuration file.

  The following options may be set in the configuration file
  (command line arguments override them):

  -c, --conflicts  detect and delete assumptions with conflicts

  -l, --learnts  enable the share of learnt clauses

  -z <value>, --learnts-max-size  set the max size of the learnt clauses to
share (default is 20)

  -t <value>, --learnts-max-amount  set the max amount of learnt clauses to
share (default is 50)

  -r, --remove-learnts  remove all the learnt clauses after each solve call.
                If its share is enabled they are sent before removal.
                By default the learnt clauses are kept.

  -a <value>, --assumps-cpus-ratio  set the ratio between the number of
  assumptions to solve and the worker CPUs (default is 3)
  It is used in the automatic calculation of the number of literals and mode.

  -s <arg>, --selection  methods to select the variables to assume with <arg>:
         o - variables with more occurrences(default)
         b - variables in the biggest clauses

  input-file: may be either in plain/gzipped DIMACS format or in BCNF.

  output-file: the file where the result is written.
```

Figure A.1: Program's options

```
#keep comments in separate lines
#Do not insert spaces !

#max size of learnt clauses
LEARNTS_MAX_SIZE=20

#share learnt clauses ?
SHARE_LEARNTS=false

#remove learnt clauses after each solve?
REMOVE_LEARNTS=false

#share conflics ?
CONFLICTS=false

#max amount of learnt clauses to send
LEARNTS_MAX_AMOUNT=30

#Ratio between the number of assumptions.
#and the amount of CPUs.
#Used in automatic calculations of
#variables to assume and search mode
ASSUMPS_CPU_RATIO=3

#how select the variables to assume:
#can be more_occurrences or bigger_clauses
VARIABLE_SELECTION=more_occurrences
```

Figure A.2: Configuration file

The options `-m` and `-n` can only be set by the user or calculated automatically and do not appear in the configuration file. This was decided because you may want to select different modes or assume a distinct amount of literals each time you run the program.

As was said, the most important options are precisely the previous two. Their values, when omitted, are set to respect the assumptions–CPUs ratio, i.e., create an amount of assumptions proportional to the number of workers without exceeding a threshold. The ratio should be bigger than 1 (default is 3).

The search modes conduct the search by different paths and for k literals, the *Sequential* and *Random* modes will make 2^k tests, while the *Few first* and *Many first* will make only $2 \times k$ tests.

A.6 Examples

Here you can find a set of examples with several different ways to invoke the program.

Example A.6.1 *Running the program in 5 CPUs (4 workers and the master) with search mode* Few first *and sharing learnt clauses. The number of variables is automatically calculated.*
`mpirun -np 5 parallel_minisat -m f -l file.cnf`

Example A.6.2 *Running the program in 6 CPUs (5 workers and the master), sharing conflicts and selecting the variables in bigger clauses.*
`mpirun -np 6 parallel_minisat -c -s b file.cnf`

Example A.6.3 *Running the program in 10 CPUs (9 workers and the master), sharing conflicts, removing learnt clauses after each execution of the solver and giving an output file.*
`mpirun -np 10 parallel_minisat -c -r file.cnf out.txt`

Example A.6.4 *Running the program in 8 CPUs (7 workers and the master), reading a configuration file, sharing learnt clauses and setting them to have a maximum size of 10 literals (overriding the value given in the config file).*
`mpirun -np 8 parallel_minisat -f minisat.conf -l -z 10 file.cnf`

A.7 Inputs and Outputs

The input with boolean formulas are given in a file in DIMACS/CNF format (.cnf) or BCNF (.bcnf).

The DIMACS/CNF file format is composed by:

1. comments, that is, lines beginning with the character `c`;
2. a preamble containing informations about the instance: the file format, the number of variables and clauses: `p format nvars nclauses`;
3. the clauses, each one encoded as a sequence of non-null numbers ranging from -nvars to nvars and separated by zero. Positive numbers represent the corresponding variables and negative numbers denote their negations.

It is not necessary that every variable appear in the formula, as one can see in the example where the variable 2 is not used.

Example A.7.1 (DIMACS/CNF file)

```
c comments here
c next is preamble
p cnf 5 4
1 -3 4 0
-1 3 0
-5 -4 0
5 -3 0
```

The result of the search (SAT/UNSAT) is printed to the screen and the model may be saved in a file if a name is given in the arguments.

The other interesting output produced is a file with the extension `.time` containing all the statistics of the execution like the number of calls and the times spent. It shows:

- master's initialization time to read the input file, parse and setup the parameters;
- conditions of the execution: number of workers, variables, search mode, variables's selection mode, erased assumptions and options of learnt clauses;
- workers's execution statistics indicating the number of times the solver ran, sets of learnt clauses sent and received and the CPU time (user plus system time) spent in computation and by the master to process worker's data;
- master finalization time to write outputs and post processing;
- total time as the sum of master's initialization and finalization times and $worker + master$ time from the worker that took more time;
- total wall time to see the influence of the system's load and the delay caused by communication.

The name of this file is made following a pattern, by joining the name of the input file and the conditions of execution: number of CPUs, search mode, amount of variables and method to choose them, existence of conflicts and learnt clauses (and their settings).

Example A.7.2 *After solving the file* `fpga10_11_uns_rcr.cnf` *in 3 CPUs (2 workers and master),* Random *mode, assuming 3 variables, removing assumptions with conflicts and sharing learnt clauses, the program will create a file named* `fpga10_11_uns_rcr.cnf-3-r-3-o-c-l-z20-t50.time` *with the statistics:*

```
Master initialization time: 0.005998 secs

Workers: 2
Variables to be assumed: 3
Search mode: r
Variable's selection mode: o
Erased assumptions: 2
Learnt max amount: 50
Learnts max size: 20

Worker 1:
solve() was executed 2 times
Total time spent by worker: 41.090568 secs
Total time spent by master with this worker: 0.001000 secs
Databases received: 1
Databases sent: 2

Worker 2:
solve() was executed 4 times
Total time spent by worker: 41.034564 secs
Total time spent by master with this worker: 0.001000 secs
Databases received: 1
Databases sent: 4

Master finalization time: 0.000000 secs

Total CPU time: 41.097566 secs

Total wall time: 42.758681 secs
```

A.8 Hints for better performance

There are no better settings to get a good performance. Through experience, you may see what the options make the program take less time to solve the formulas. Here are some, that may not always be the best:

- keep the granularity high: select an amount of variables large enough to generate more assumptions than the amount of workers (4 or 5 assumptions for each worker);
- nevertheless, try a search with few variables and workers (2 or 3). Some problems are easily solved;
- avoid the *Sequential* mode. The others provide better performance;
- do not use the option `-r` to remove the learnt clauses because they will guide the following searches;

- use only conflicts when you have many assumptions and in *Random* and *Sequential* modes. The same for learnt clauses;
- after solving take a look at the file with the statistics because it gives a clear picture where the time was spent.

A.9 Error messages

The error messages that the program can present are:

- `ERROR! Could not open file: filename` – the file does not exits or you don't have permissions to open it.
- `ERROR! Not a BCNF file: filename` – the file has an incorrect format.
- `ERROR! BCNF file in unsupported byte-order: filename` – similar to previous error.
- `PARSE ERROR! Unexpected char:` – a character different than a digit appeared in the input file.
- `ERROR! configuration file filename not found! Setting default values...` – the file with the given name was not found and the program will continue assuming the default internal values for the parameters.
- `ERROR! Could not open file: filename` – could not open the input file.
- `ERROR! Cannot write output to file!` – the program could not write the results for the output file due lack of permissions, disk space or other reason.
- `ERROR! Number of literals to assume is bigger than number of variables in formula!` – if you tried to assume more variables than those in the formula.
- `ERROR! the number of CPUs is 1 but the selected mode is not LOCAL!` – if the program was invoked without `mpirun` and with a search mode different than *Local*. The program will set the search mode to *Local* and continue.
- `ERROR! the number of CPUs is greater than 1 but the selected mode is LOCAL!` – the inverse situation, with many CPUs and the *Local* mode. The search mode is recalculated and the program continues.

A.10 FAQ

What is MPI ?

MPI stands for Message Passing Interface and is a specification of an interface to create parallel programs and manage the communication between the processes. It is considered the de-facto standard for parallel computing. There are several vendors that provide MPI as a library to be called by the parallel programs. These programs are executed in clusters of servers or workstations.

Can I use MPI just in Unix/Linux, or is available for Windows also ?

There are several MPI implementations from different vendors for both operating systems. We used MPICH from Argonne National Laboratory that provides versions for Windows and Unix/Linux.

Can I run MPI in a heterogeneous (Linux and Windows) clusters ?

You should get that information from your MPI vendor. There are implementations that just work on homogeneous clusters (same operating system and version, architecture, libraries) while others may be used in heterogeneous clusters, most of the times with some restrictions (e.g. the size of datatypes).

What are the contents of the file with the list of machines: IPs or hostnames ?

The file given after the option `-machinefile` contains the hostnames of the machines where the processes will be created.

What is the average speedup of the program ? When should I use parallel vs sequential ?

The performance varies with the problem to solve. We noticed that the program is faster, but in certain problems or with certain options it takes the same time or even more. You should use the parallel program with problems that take much time to solve by the sequential program, for instance more than 300 seconds. The objective is to decrease the execution time in sets of ten or even hundreds of seconds.

But is there a general idea based in the size of the problem and the amount of processors ?

No. There are small problems that take a lot of time to solve and big problems that are solved really fast.

What is super–linear speedup ? Does it happens in the program ? Why ?

Speedup is time of the sequential algorithm divided by the time in taken in p processors. Super–linear speedup happens when the speedup achieved is bigger than the number of processors used to solve the problem.

A super–linear speedup occur specially with SAT problems. Their cause is related to how quickly each partition of the subspace is solved and the solution is found. The solving process is made by a search algorithm that is guided by the constraints of the problem. This means that each partition takes a different time to be explored, even when they have the same dimensions. Sometimes the partition with the solution is quickly explored leading to a super–linear speedup.

Appendix B

Performance tables

This appendix contains the tables with the performance of the parallel SAT-solver. All times (except the Wall) refer to CPU time and are in seconds.

The tests to the influence of communication made with one worker are presented in Table B.1 and Table B.2.

The average times spent on solving all files, just the SAT and just the UNSAT with sequential MiniSAT are displayed in Table B.3.

The average of the time spent solving all files, just the SAT and just the UNSAT in 3, 6 and 9 workers are presented in Table B.4, Table B.5 and Table B.6 respectively.

The tests for modes and options are displayed in several tables: the best and worst times from Table B.7 to Table B.12 and the entire set of times for each file are from Table B.15 to Table B.39.

The best and worst times of the tests about granularity are presented in the Tables B.13 and B.14, while the times of all the tests are from Table B.40 to Table B.64.

The tables about the influence of the communication display:

- CPU time spent by the sequential MiniSAT;
- CPU and Wall time spent by the parallel version;
- difference between the previous Wall and CPU time;
- amount of messages exchanged between the master and the worker.

The table with the average times display:

- number of workers (**#W**);

- search mode (**Mode**);
- average time without options (**Avg-no-opts**);
- average time removing assumptions with conflicts (**Avg-confs**);
- average time sharing learnt clauses (**Avg-learnts**);
- average time with different granularities (**Avg-gran**).

The tables of best and worst times display:

- file name (**File**);
- time spent by the sequential MiniSAT (**T-seq**);
- time of the parallel version (**T-par**);
- search mode (**Mode**);
- number of workers (**#W**);
- speedup (**Spd.**);
- efficiency (**Eff.**);
- serial fraction (**S. F.**);
- number of variables assumed (**#V**) in the tests of granularity.

All the tables of the tests to modes and options display:

- number of workers (**#W**);
- search mode (**Mode**);
- time of the execution without options (**T-no-opts**);
- time with conflicts removal (**T-confs**);
- time with share of learnt clauses (**T-learnts**);
- erased assumptions with conflicts (**E.A.**);
- total amount of databases with learnt clauses sent (**Sent**) by the workers;
- total amount of databases with learnt clauses received (**Recv.**) by the workers.

All the tables of the granularity tests display:

- number of workers (**#W**);
- search mode (**Mode**);

- number of variables assumed (**#V**);
- CPU time spent (**CPU Time**);
- speedup (**Spd.**);
- efficiency (**Eff.**);
- serial fraction (**S. F.**).

	Sequential	**Master and worker in the same node**			
File	CPU time	CPU time	Wall time	Wall-CPU	#Messages
fpga10_11	44.999	157.462	157.473	0.011	128
fpga10_12	158.474	162.822	162.838	0.016	128
fpga10_13	161.586	1188.302	1188.758	0.455	128
hole11	723.157	641.080	641.188	0.108	128
mod2-9-11.cnf	78.573	258.808	258.841	0.032	128
mod2-9-4.cnf	80.753	219.774	219.965	0.192	128
unif-r4	3246.920	1617.109	1617.784	0.675	128
frb40-19-1	287.402	41.199	41.437	0.238	2
frb40-19-2	540.394	109.979	110.013	0.034	2
frb40-19-3	6340.410	2563.020	2563.051	0.031	2
frb40-19-4	901.448	2218.831	2218.865	0.035	2
frb40-19-5	4528.720	3016.685	3016.733	0.049	2
mod2-3g14	877.347	1783.715	1784.028	0.313	64
mod2c-150-11	75.333	70.052	70.741	0.688	10
mod2c-150-15	26.554	110.363	110.848	0.486	20
sat2	66.552	95.726	95.898	0.172	30
unif-r5	360.687	2.320	2.508	0.188	2
vmpc_21.ren	51.191	11.089	11.121	0.033	2
vmpc_23.ren	151.893	115.995	116.050	0.055	2
vmpc_25.ren	472.618	790.445	790.627	0.181	10
vmpc_25.shuf	25.826	68.284	68.384	0.100	2
vmpc_26.ren	142.121	697.160	697.362	0.202	2
vmpc_26.shuf	301.119	139.593	139.693	0.100	2
vmpc_27.ren	896.080	897.956	898.531	0.575	2
vmpc_27.shuf	453.440	964.700	964.926	0.226	2

Table B.1: Sequential vs Parallel with master and local worker

	Sequential	**Master and remote worker**			
File	CPU time	CPU time	Wall time	Wall-CPU	#Messages
fpga10_11	44.999	157.934	157.942	0.008	128
fpga10_12	158.474	162.694	162.874	0.180	128
fpga10_13	161.586	1185.690	1185.920	0.230	128
hole11	723.157	643.952	644.046	0.094	128
mod2-9-11	78.573	262.680	262.736	0.055	128
mod2-9-4	80.753	224.994	225.112	0.118	128
unif-r4	3246.920	1614.413	1615.845	1.432	128
frb40-19-1	287.402	41.275	42.250	0.975	2
frb40-19-2	540.394	104.055	104.078	0.024	2
frb40-19-3	6340.410	2406.247	2407.205	0.958	2
frb40-19-4	901.448	2085.334	2085.642	0.308	2
frb40-19-5	4528.720	2830.865	2831.394	0.529	2
mod2-3g14	877.347	1791.196	1791.262	0.066	64
mod2c-150-11	75.333	70.384	71.878	1.493	10
mod2c-150-15	26.554	112.139	113.191	1.052	20
sat2	66.552	95.178	95.309	0.131	30
unif-r5	360.687	2.324	2.936	0.611	2
vmpc_21.ren	51.191	10.701	10.707	0.006	2
vmpc_23.ren	151.893	109.815	109.822	0.007	2
vmpc_25.ren	472.618	741.634	741.764	0.130	10
vmpc_25.shuf	25.826	65.060	65.079	0.019	2
vmpc_26.ren	142.121	651.913	651.998	0.086	2
vmpc_26.shuf	301.119	133.256	133.271	0.015	2
vmpc_27.ren	896.080	956.000	957.027	1.028	2
vmpc_27.shuf	453.440	912.557	912.688	0.131	2

Table B.2: Sequential vs Parallel with master and remote worker

Files	**Average time per file**
All files	839.723
Just SAT files	916.618
Just UNSAT files	641.994

Table B.3: Average time of each file taken by the sequential MiniSAT

#W	Modes	Avg-no-opts	Avg-confs	Avg-learnts	Avg-gran
3	*Few first*	583.171	583.062	354.246	360.804
3	*Many first*	515.157	515.325	507.609	514.925
3	*Random*	233.955	325.698	402.908	479.217
3	*Sequential*	491.947	481.000	487.951	383.589
6	*Few first*	263.225	248.453	297.568	219.419
6	*Many first*	487.311	487.295	490.225	192.874
6	*Random*	442.357	435.161	462.267	448.592
6	*Sequential*	455.172	456.049	452.549	452.214
9	*Few first*	204.162	203.460	202.118	95.589
9	*Many first*	494.989	494.613	503.160	116.089
9	*Random*	437.607	404.607	395.692	472.791
9	*Sequential*	440.508	442.092	442.603	537.086

Table B.4: Average of each file in 3, 6 and 9 workers

#W	Modes	Avg-no-opts	Avg-confs	Avg-learnts	Avg-gran
3	*Few first*	667.143	667.597	344.804	396.191
3	*Many first*	570.140	569.682	567.213	576.319
3	*Random*	232.513	378.467	470.377	572.181
3	*Sequential*	593.173	589.025	587.246	429.391
6	*Few first*	234.601	218.502	280.733	216.414
6	*Many first*	558.833	558.679	559.042	151.095
6	*Random*	549.986	535.787	580.246	560.982
6	*Sequential*	570.729	569.105	564.815	568.754
9	*Few first*	148.731	148.865	149.395	42.726
9	*Many first*	556.352	555.212	556.074	32.758
9	*Random*	541.893	505.616	497.348	598.701
9	*Sequential*	562.493	561.593	565.651	696.683

Table B.5: Average of each SAT file in 3, 6 and 9 workers

#W	Modes	Avg-no-opts	Avg-confs	Avg-learnts	Avg-gran
3	*Few first*	367.242	365.687	378.527	269.810
3	*Many first*	373.769	375.551	354.343	357.054
3	*Random*	237.662	190.008	229.415	240.165
3	*Sequential*	231.650	203.219	232.620	265.811
6	*Few first*	336.829	325.469	340.857	227.144
6	*Many first*	303.398	303.737	313.267	300.308
6	*Random*	165.595	176.409	158.891	159.590
6	*Sequential*	158.026	165.335	163.864	152.541
9	*Few first*	346.699	343.846	337.689	231.520
9	*Many first*	337.199	338.787	367.097	330.369
9	*Random*	169.442	144.868	134.292	149.021
9	*Sequential*	126.830	134.803	126.195	126.693

Table B.6: Average of each UNSAT file in 3, 6 and 9 workers

File	T-seq	T-par	Mode	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	31.563	*Few first*	8	1.426	0.178	0.659
fpga10_12	158.474	71.475	*Many first*	9	2.217	0.246	0.382
fpga10_13	161.586	165.130	*Random*	5	0.979	0.196	1.027
hole11	723.157	218.260	*Few first*	10	3.313	0.331	0.224
mod2-9-11	78.573	29.017	*Many first*	9	2.708	0.301	0.290
mod2-9-4	80.753	28.197	*Random*	9	2.864	0.318	0.268
unif-r4	3246.920	198.606	*Random*	10	16.349	1.635	-0.043
frb40-19-1	287.402	15.303	*Random*	6	18.781	3.130	-0.136
frb40-19-2	540.394	0.233	*Random*	9	2319.228	257.692	-0.125
frb40-19-3	6340.410	8.459	*Few first*	8	749.589	93.699	-0.141
frb40-19-4	901.448	236.657	*Random*	3	3.809	1.270	-0.106
frb40-19-5	4528.720	43.460	*Random*	5	104.205	20.841	-0.238
mod2-3g14	877.347	37.102	*Many first*	6	23.647	3.941	-0.149
mod2c-150-11	75.333	4.713	*Few first*	3	15.983	5.328	-0.406
mod2c-150-15	26.554	0.996	*Random*	7	26.659	3.808	-0.123
sat2	66.552	6.372	*Random*	10	10.444	1.044	-0.005
unif-r5	360.687	1.423	*Random*	10	253.454	25.345	-0.107
vmpc_21.ren	51.191	0.375	*Random*	10	136.505	13.651	-0.103
vmpc_23.ren	151.893	5.560	*Few first*	8	27.317	3.415	-0.101
vmpc_25.ren	472.618	0.732	*Random*	10	645.627	64.563	-0.109
vmpc_25.shuf	25.826	5.107	*Random*	6	5.057	0.843	0.037
vmpc_26.ren	142.121	116.604	*Few first*	5	1.219	0.244	0.776
vmpc_26.shuf	301.119	11.303	*Few first*	10	26.641	2.664	-0.069
vmpc_27.ren	896.080	57.139	*Random*	3	15.683	5.228	-0.404
vmpc_27.shuf	453.440	0.465	*Few first*	5	975.123	195.025	-0.249

Table B.7: Best performances without options

File	T-seq	T-par	Mode	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	95.217	*Sequential*	2	0.473	0.236	3.232
fpga10_12	158.474	221.705	*Random*	3	0.715	0.238	1.598
fpga10_13	161.586	415.011	*Sequential*	3	0.389	0.130	3.353
hole11	723.157	432.901	*Few first*	8	1.670	0.209	0.541
mod2-9-11	78.573	123.353	*Sequential*	2	0.637	0.318	2.140
mod2-9-4	80.753	113.850	*Sequential*	2	0.709	0.355	1.820
unif-r4	3246.920	2091.194	*Many first*	4	1.553	0.388	0.525
frb40-19-1	287.402	271.148	*Few first*	3	1.060	0.353	0.915
frb40-19-2	540.394	571.121	*Few first*	7	0.946	0.135	1.066
frb40-19-3	6340.410	7540.098	*Few first*	7	0.841	0.120	1.221
frb40-19-4	901.448	3548.995	*Few first*	2	0.254	0.127	6.874
frb40-19-5	4528.720	6313.364	*Few first*	2	0.717	0.359	1.788
mod2-3g14	877.347	1970.777	*Few first*	2	0.445	0.223	3.493
mod2c-150-11	75.333	237.513	*Random*	2	0.317	0.159	5.306
mod2c-150-15	26.554	113.870	*Many first*	2	0.233	0.117	7.577
sat2	66.552	149.822	*Many first*	2	0.444	0.222	3.502
unif-r5	360.687	130.581	*Few first*	2	2.762	1.381	-0.276
vmpc_21.ren	51.191	32.319	*Few first*	5	1.584	0.317	0.539
vmpc_23.ren	151.893	197.157	*Few first*	5	0.770	0.154	1.373
vmpc_25.ren	472.618	46.515	*Many first*	2	10.161	5.080	-0.803
vmpc_25.shuf	25.826	384.740	*Few first*	10	0.067	0.007	16.442
vmpc_26.ren	142.121	1480.721	*Few first*	2	0.096	0.048	19.837
vmpc_26.shuf	301.119	551.501	*Few first*	8	0.546	0.068	1.950
vmpc_27.ren	896.080	2904.091	*Random*	6	0.309	0.051	3.689
vmpc_27.shuf	453.440	35.820	*Random*	2	12.659	6.329	-0.842

Table B.8: Worst performances without options

File	T-seq	T-par	Mode	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	24.844	*Random*	2	1.811	0.906	0.104
fpga10_12	158.474	70.419	*Random*	5	2.250	0.450	0.305
fpga10_13	161.586	172.996	*Few first*	6	0.934	0.156	1.085
hole11	723.157	218.559	*Few first*	10	3.309	0.331	0.225
mod2-9-11	78.573	28.136	*Random*	10	2.793	0.279	0.287
mod2-9-4	80.753	29.092	*Sequential*	9	2.776	0.308	0.280
unif-r4	3246.920	207.468	*Random*	10	15.650	1.565	-0.040
frb40-19-1	287.402	14.928	*Random*	2	19.253	9.626	-0.896
frb40-19-2	540.394	33.812	*Few first*	4	15.982	3.996	-0.250
frb40-19-3	6340.410	8.553	*Few first*	9	741.350	82.372	-0.123
frb40-19-4	901.448	37.572	*Random*	6	23.992	3.999	-0.150
frb40-19-5	4528.720	43.486	*Random*	10	104.143	10.414	-0.100
mod2-3g14	877.347	36.575	*Random*	9	23.987	2.665	-0.078
mod2c-150-11	75.333	4.721	*Few first*	6	15.956	2.659	-0.125
mod2c-150-15	26.554	0.993	*Random*	5	26.739	5.348	-0.203
sat2	66.552	7.146	*Random*	9	9.313	1.035	-0.004
unif-r5	360.687	1.430	*Random*	10	252.214	25.221	-0.107
vmpc_21.ren	51.191	0.365	*Random*	6	140.245	23.374	-0.191
vmpc_23.ren	151.893	2.953	*Random*	3	51.434	17.145	-0.471
vmpc_25.ren	472.618	1.952	*Random*	3	242.107	80.702	-0.494
vmpc_25.shuf	25.826	36.989	*Few first*	2	0.698	0.349	1.865
vmpc_26.ren	142.121	116.567	*Few first*	5	1.219	0.244	0.775
vmpc_26.shuf	301.119	11.386	*Few first*	10	26.447	2.645	-0.069
vmpc_27.ren	896.080	33.394	*Random*	8	26.834	3.354	-0.100
vmpc_27.shuf	453.440	0.479	*Few first*	5	946.621	189.324	-0.249

Table B.9: Best performances with conflicts

File	T-seq	T-par	Mode	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	81.459	*Many first*	5	0.552	0.110	2.013
fpga10_12	158.474	165.578	*Sequential*	4	0.957	0.239	1.060
fpga10_13	161.586	335.903	*Many first*	4	0.481	0.120	2.438
hole11	723.157	462.108	*Random*	6	1.565	0.261	0.567
mod2-9-11	78.573	121.923	*Sequential*	2	0.644	0.322	2.103
mod2-9-4	80.753	117.784	*Random*	2	0.686	0.343	1.917
unif-r4	3246.920	2099.032	*Many first*	4	1.547	0.387	0.529
frb40-19-1	287.402	270.852	*Few first*	3	1.061	0.354	0.914
frb40-19-2	540.394	571.028	*Few first*	7	0.946	0.135	1.066
frb40-19-3	6340.410	7532.027	*Few first*	7	0.842	0.120	1.219
frb40-19-4	901.448	2778.452	*Few first*	3	0.324	0.108	4.123
frb40-19-5	4528.720	6305.363	*Few first*	2	0.718	0.359	1.785
mod2-3g14	877.347	1983.184	*Few first*	2	0.442	0.221	3.521
mod2c-150-11	75.333	132.246	*Many first*	2	0.570	0.285	2.511
mod2c-150-15	26.554	115.582	*Many first*	2	0.230	0.115	7.706
sat2	66.552	149.707	*Many first*	2	0.445	0.222	3.499
unif-r5	360.687	131.772	*Few first*	2	2.737	1.369	-0.269
vmpc_21.ren	51.191	32.292	*Few first*	5	1.585	0.317	0.539
vmpc_23.ren	151.893	197.077	*Few first*	5	0.771	0.154	1.372
vmpc_25.ren	472.618	42.167	*Many first*	2	11.208	5.604	-0.822
vmpc_25.shuf	25.826	385.268	*Few first*	9	0.067	0.007	16.658
vmpc_26.ren	142.121	1480.946	*Few first*	2	0.096	0.048	19.841
vmpc_26.shuf	301.119	661.961	*Random*	5	0.455	0.091	2.498
vmpc_27.ren	896.080	2288.459	*Few first*	5	0.392	0.078	2.942
vmpc_27.shuf	453.440	38.587	*Sequential*	2	11.751	5.875	-0.830

Table B.10: Worst performances with conflicts

File	T-seq	T-par	Mode	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	31.374	*Few first*	6	1.434	0.239	0.637
fpga10_12	158.474	62.417	*Few first*	9	2.539	0.282	0.318
fpga10_13	161.586	130.311	*Random*	6	1.240	0.207	0.768
hole11	723.157	218.196	*Few first*	10	3.314	0.331	0.224
mod2-9-11	78.573	26.599	*Random*	10	2.954	0.295	0.265
mod2-9-4	80.753	27.259	*Sequential*	10	2.962	0.296	0.264
unif-r4	3246.920	195.317	*Sequential*	10	16.624	1.662	-0.044
frb40-19-1	287.402	21.192	*Few first*	5	13.562	2.712	-0.158
frb40-19-2	540.394	35.316	*Few first*	4	15.302	3.825	-0.246
frb40-19-3	6340.410	8.499	*Few first*	8	746.060	93.258	-0.141
frb40-19-4	901.448	339.171	*Random*	2	2.658	1.329	-0.247
frb40-19-5	4528.720	43.358	*Few first*	8	104.450	13.056	-0.132
mod2-3g14	877.347	37.069	*Random*	6	23.668	3.945	-0.149
mod2c-150-11	75.333	4.719	*Many first*	8	15.963	1.995	-0.071
mod2c-150-15	26.554	0.998	*Random*	4	26.605	6.651	-0.283
sat2	66.552	7.209	*Random*	7	9.231	1.319	-0.040
unif-r5	360.687	1.430	*Random*	5	252.213	50.443	-0.245
vmpc_21.ren	51.191	0.080	*Random*	8	639.938	79.992	-0.141
vmpc_23.ren	151.893	1.677	*Few first*	4	90.569	22.642	-0.319
vmpc_25.ren	472.618	0.304	*Random*	10	1554.649	155.465	-0.110
vmpc_25.shuf	25.826	4.964	*Random*	8	5.202	0.650	0.077
vmpc_26.ren	142.121	2.111	*Random*	9	67.320	7.480	-0.108
vmpc_26.shuf	301.119	11.678	*Few first*	10	25.786	2.579	-0.068
vmpc_27.ren	896.080	57.257	*Few first*	6	15.650	2.608	-0.123
vmpc_27.shuf	453.440	0.675	*Few first*	5	671.742	134.348	-0.248

Table B.11: Best performances sharing learnt clauses

File	T-seq	T-par	Mode	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	95.287	*Sequential*	2	0.472	0.236	3.235
fpga10_12	158.474	244.117	*Sequential*	2	0.649	0.325	2.081
fpga10_13	161.586	499.687	*Random*	2	0.323	0.162	5.185
hole11	723.157	496.216	*Random*	5	1.457	0.291	0.608
mod2-9-11	78.573	130.889	*Sequential*	2	0.600	0.300	2.332
mod2-9-4	80.753	121.656	*Sequential*	2	0.664	0.332	2.013
unif-r4	3246.920	1850.591	*Many first*	9	1.755	0.195	0.516
frb40-19-1	287.402	299.430	*Random*	6	0.960	0.160	1.050
frb40-19-2	540.394	571.234	*Few first*	7	0.946	0.135	1.067
frb40-19-3	6340.410	7545.371	*Few first*	7	0.840	0.120	1.222
frb40-19-4	901.448	2605.094	*Random*	4	0.346	0.087	3.520
frb40-19-5	4528.720	4690.640	*Random*	3	0.965	0.322	1.054
mod2-3g14	877.347	2017.331	*Few first*	2	0.435	0.217	3.599
mod2c-150-11	75.333	107.473	*Many first*	2	0.701	0.350	1.853
mod2c-150-15	26.554	146.487	*Many first*	2	0.181	0.091	10.033
sat2	66.552	207.412	*Many first*	2	0.321	0.160	5.233
unif-r5	360.687	132.147	*Few first*	2	2.729	1.365	-0.267
vmpc_21.ren	51.191	32.419	*Few first*	5	1.579	0.316	0.542
vmpc_23.ren	151.893	118.318	*Random*	2	1.284	0.642	0.558
vmpc_25.ren	472.618	46.568	*Many first*	2	10.149	5.075	-0.803
vmpc_25.shuf	25.826	430.579	*Random*	2	0.060	0.030	32.345
vmpc_26.ren	142.121	1385.304	*Few first*	2	0.103	0.051	18.495
vmpc_26.shuf	301.119	652.060	*Random*	7	0.462	0.066	2.360
vmpc_27.ren	896.080	2913.734	*Random*	10	0.308	0.031	3.502
vmpc_27.shuf	453.440	34.313	*Many first*	2	13.215	6.607	-0.849

Table B.12: Worst performances sharing learnt clauses

File	T-seq	T-par	Mode	#V	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	13.331	*Random*	2	2	3.376	1.688	-0.408
fpga10_12	158.474	52.116	*Few first*	5	3	3.041	1.014	-0.007
fpga10_13	161.586	181.370	*Few first*	5	3	0.891	0.297	1.184
hole11	723.157	64.229	*Few first*	41	9	11.259	1.251	-0.025
mod2-9-11	78.573	26.452	*Few first*	50	10	2.970	0.297	0.263
mod2-9-4	80.753	24.190	*Sequential*	7	10	3.338	0.334	0.222
unif-r4	3246.920	180.548	*Random*	7	9	17.984	1.998	-0.062
frb40-19-1	287.402	1.919	*Many first*	50	10	149.758	14.976	-0.104
frb40-19-2	540.394	5.743	*Many first*	50	10	94.090	9.409	-0.099
frb40-19-3	6340.410	9.868	*Many first*	50	10	642.548	64.255	-0.109
frb40-19-4	901.448	25.892	*Few first*	41	9	34.816	3.868	-0.093
frb40-19-5	4528.720	1.804	*Few first*	41	9	2510.234	278.915	-0.125
mod2-3g14	877.347	2.433	*Few first*	25	7	360.581	51.512	-0.163
mod2c-150-11	75.333	4.444	*Few first*	25	7	16.951	2.422	-0.098
mod2c-150-15	26.554	0.847	*Random*	5	5	31.348	6.270	-0.210
sat2	66.552	0.508	*Few first*	18	6	131.001	21.833	-0.191
unif-r5	360.687	0.810	*Many first*	41	9	445.267	49.474	-0.122
vmpc_21.ren	51.191	0.102	*Many first*	18	6	501.899	83.650	-0.198
vmpc_23.ren	151.893	0.133	*Few first*	41	9	1142.113	126.901	-0.124
vmpc_25.ren	472.618	0.529	*Random*	6	7	893.391	127.627	-0.165
vmpc_25.shuf	25.826	3.833	*Few first*	41	9	6.737	0.749	0.042
vmpc_26.ren	142.121	2.334	*Few first*	13	5	60.888	12.178	-0.229
vmpc_26.shuf	301.119	4.476	*Many first*	41	9	67.270	7.474	-0.108
vmpc_27.ren	896.080	9.311	*Few first*	41	9	96.243	10.694	-0.113
vmpc_27.shuf	453.440	0.787	*Sequential*	6	6	576.142	96.024	-0.198

Table B.13: Best performances with different granularity

File	T-seq	T-par	Mode	#V	#W	Spd.	Eff.	S. F.
fpga10_11	44.999	138.720	*Many first*	41	9	0.324	0.036	3.343
fpga10_12	158.474	263.936	*Many first*	13	5	0.600	0.120	1.832
fpga10_13	161.586	515.525	*Many first*	8	4	0.313	0.078	3.921
hole11	723.157	970.758	*Few first*	2	2	0.745	0.372	1.685
mod2-9-11	78.573	75.082	*Random*	2	2	1.046	0.523	0.911
mod2-9-4	80.753	78.703	*Random*	2	2	1.026	0.513	0.949
unif-r4	3246.920	2818.094	*Few first*	32	8	1.152	0.144	0.849
frb40-19-1	287.402	315.537	*Sequential*	2	2	0.911	0.455	1.196
frb40-19-2	540.394	445.817	*Many first*	5	3	1.212	0.404	0.737
frb40-19-3	6340.410	5347.412	*Many first*	5	3	1.186	0.395	0.765
frb40-19-4	901.448	2218.451	*Sequential*	6	6	0.406	0.068	2.753
frb40-19-5	4528.720	4980.721	*Random*	6	8	0.909	0.114	1.114
mod2-3g14	877.347	1757.525	*Sequential*	2	2	0.499	0.250	3.006
mod2c-150-11	75.333	119.166	*Many first*	2	2	0.632	0.316	2.164
mod2c-150-15	26.554	60.001	*Many first*	8	4	0.443	0.111	2.679
sat2	66.552	139.505	*Sequential*	2	2	0.477	0.239	3.192
unif-r5	360.687	250.100	*Many first*	2	2	1.442	0.721	0.387
vmpc_21.ren	51.191	55.452	*Random*	2	2	0.923	0.462	1.166
vmpc_23.ren	151.893	116.994	*Sequential*	6	6	1.298	0.216	0.724
vmpc_25.ren	472.618	444.469	*Random*	2	2	1.063	0.532	0.881
vmpc_25.shuf	25.826	469.870	*Sequential*	4	3	0.055	0.018	26.791
vmpc_26.ren	142.121	1957.178	*Random*	4	4	0.073	0.018	18.028
vmpc_26.shuf	301.119	730.186	*Random*	4	3	0.412	0.137	3.137
vmpc_27.ren	896.080	2770.463	*Few first*	2	2	0.323	0.162	5.184
vmpc_27.shuf	453.440	42.710	*Many first*	50	10	10.617	1.062	-0.006

Table B.14: Worst performances with different granularity

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	42.256	37.744	48.027	7	12	4
2	*Many first*	63.162	60.780	69.960	1	12	2
2	*Random*	72.268	24.844	87.504	51	64	9
2	*Sequential*	95.217	30.642	95.287	53	64	5
3	*Few first*	36.124	33.934	49.128	4	12	2
3	*Many first*	39.072	39.295	53.567	1	12	4
3	*Random*	50.817	35.736	44.018	48	64	19
3	*Sequential*	68.363	58.750	51.695	52	64	21
4	*Few first*	49.285	49.544	49.401	4	12	5
4	*Many first*	53.835	54.044	38.331	1	12	5
4	*Random*	46.502	36.764	68.599	45	61	33
4	*Sequential*	34.694	30.334	59.879	49	64	21
5	*Few first*	31.715	36.444	40.837	3	12	5
5	*Many first*	81.508	81.459	38.715	0	12	5
5	*Random*	49.074	51.335	34.636	47	60	34
5	*Sequential*	36.715	60.645	43.542	49	64	17
6	*Few first*	37.024	31.419	31.374	3	12	4
6	*Many first*	51.970	51.932	43.853	0	12	5
6	*Random*	37.426	31.039	55.487	44	63	45
6	*Sequential*	37.442	36.893	36.837	47	64	26
7	*Few first*	31.574	31.725	31.460	2	12	3
7	*Many first*	59.977	60.082	38.486	0	12	4
7	*Random*	44.828	55.052	41.669	42	64	47
7	*Sequential*	38.758	38.933	39.520	44	64	27
8	*Few first*	31.563	31.586	31.667	1	12	2
8	*Many first*	33.181	33.480	33.129	0	12	3
8	*Random*	38.104	56.985	33.036	43	60	48
8	*Sequential*	36.793	36.983	36.639	42	64	34
9	*Few first*	31.603	31.813	32.521	0	12	2
9	*Many first*	31.609	31.452	31.540	0	12	2
9	*Random*	42.103	32.602	42.546	39	64	50
9	*Sequential*	59.207	37.089	38.625	39	64	37
10	*Few first*	33.249	33.104	33.389	0	12	1
10	*Many first*	31.649	31.582	31.481	0	12	1
10	*Random*	61.640	32.325	40.345	37	63	45
10	*Sequential*	45.512	45.616	36.538	39	64	37

Table B.15: Tests with 6 variables to `fpga10_11_uns_rcr.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	121.903	121.476	121.124	3	12	8
2	*Many first*	79.129	78.891	82.234	5	12	3
2	*Random*	217.069	80.914	173.700	50	64	8
2	*Sequential*	196.841	85.049	244.117	47	64	22
3	*Few first*	117.611	116.951	94.383	1	12	8
3	*Many first*	102.795	101.359	100.990	2	12	4
3	*Random*	221.705	93.879	184.577	48	63	17
3	*Sequential*	124.875	103.232	124.782	50	64	5
4	*Few first*	144.716	145.603	160.016	0	12	7
4	*Many first*	115.533	114.216	76.760	3	12	6
4	*Random*	84.194	104.909	194.973	46	63	22
4	*Sequential*	74.254	165.578	124.118	40	64	20
5	*Few first*	92.186	92.745	102.177	0	12	6
5	*Many first*	125.209	115.973	220.822	3	12	3
5	*Random*	159.276	70.419	140.648	45	63	24
5	*Sequential*	127.519	91.171	89.500	44	64	14
6	*Few first*	113.604	113.360	93.316	0	12	5
6	*Many first*	81.612	81.203	81.586	0	12	5
6	*Random*	114.103	97.637	207.832	43	62	39
6	*Sequential*	88.397	119.480	107.504	42	63	25
7	*Few first*	77.864	78.113	113.377	0	12	4
7	*Many first*	72.310	71.700	70.716	1	12	4
7	*Random*	171.703	121.640	108.406	44	63	46
7	*Sequential*	114.020	115.020	109.644	42	64	27
8	*Few first*	97.335	97.574	121.719	0	12	3
8	*Many first*	98.858	98.397	99.777	0	12	3
8	*Random*	125.686	83.010	123.474	38	64	46
8	*Sequential*	123.698	124.795	77.329	37	63	37
9	*Few first*	91.555	91.301	62.417	0	12	2
9	*Many first*	71.475	71.140	70.786	0	12	2
9	*Random*	121.060	144.549	83.631	37	62	49
9	*Sequential*	82.502	115.430	78.397	39	63	38
10	*Few first*	112.564	112.372	113.326	0	12	1
10	*Many first*	116.023	117.161	116.290	0	12	1
10	*Random*	118.655	135.563	102.186	39	54	44
10	*Sequential*	81.544	104.657	140.960	37	64	45

Table B.16: Tests with 6 variables to `fpga10_12_uns_rcr.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	289.975	286.564	286.456	5	12	1
2	*Many first*	237.307	234.616	234.291	2	12	1
2	*Random*	276.760	219.883	499.687	48	64	9
2	*Sequential*	333.348	205.526	452.186	58	64	8
3	*Few first*	296.886	293.914	292.357	4	12	4
3	*Many first*	268.894	298.212	263.706	2	12	2
3	*Random*	272.569	178.478	183.891	48	64	27
3	*Sequential*	415.011	207.806	421.837	48	64	6
4	*Few first*	299.455	293.652	294.086	3	12	6
4	*Many first*	333.341	335.903	339.345	1	11	4
4	*Random*	301.949	320.847	297.697	45	64	17
4	*Sequential*	200.463	212.324	204.419	47	64	8
5	*Few first*	239.987	239.264	214.370	3	12	6
5	*Many first*	269.932	266.215	267.641	1	12	3
5	*Random*	165.130	175.568	177.255	45	64	30
5	*Sequential*	204.479	323.355	294.671	46	63	10
6	*Few first*	175.608	172.996	255.921	2	12	5
6	*Many first*	269.083	269.708	264.947	1	12	4
6	*Random*	280.497	212.930	130.311	41	64	50
6	*Sequential*	268.798	318.760	294.963	41	64	27
7	*Few first*	272.669	270.003	270.179	1	12	3
7	*Many first*	269.268	266.714	492.979	0	12	4
7	*Random*	198.614	277.909	329.006	42	64	44
7	*Sequential*	309.106	299.727	299.568	45	64	29
8	*Few first*	196.998	199.537	199.334	0	12	2
8	*Many first*	314.934	312.176	263.434	0	12	3
8	*Random*	302.322	223.220	267.325	36	64	43
8	*Sequential*	341.354	300.421	302.024	43	64	39
9	*Few first*	268.428	265.383	267.014	0	12	2
9	*Many first*	269.326	264.779	266.876	0	12	2
9	*Random*	280.535	246.403	259.181	36	64	49
9	*Sequential*	188.021	229.872	186.223	42	64	37
10	*Few first*	266.529	269.462	268.015	0	12	1
10	*Many first*	267.944	267.638	265.496	0	12	1
10	*Random*	247.518	213.944	291.822	32	63	50
10	*Sequential*	322.775	185.636	186.444	39	64	49

Table B.17: Tests with 6 variables to `fpga10_13_uns_rcr.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	82.260	82.117	119.129	2	10	2
2	*Many first*	41.336	41.382	41.348	3	11	0
2	*Random*	141.495	14.928	61.003	40	62	5
2	*Sequential*	41.329	41.279	41.286	48	63	0
3	*Few first*	271.148	270.852	75.174	0	9	4
3	*Many first*	41.343	41.305	41.286	1	11	3
3	*Random*	160.502	227.641	53.718	43	55	4
3	*Sequential*	41.331	41.300	41.268	47	63	4
4	*Few first*	87.792	87.849	150.353	0	8	3
4	*Many first*	41.318	41.292	41.257	1	11	4
4	*Random*	150.158	26.373	42.051	40	58	12
4	*Sequential*	41.404	41.243	41.185	44	62	5
5	*Few first*	131.990	125.223	21.192	0	7	2
5	*Many first*	41.314	41.331	41.302	1	11	2
5	*Random*	176.527	31.323	47.477	39	57	7
5	*Sequential*	41.349	41.270	41.263	43	59	13
6	*Few first*	22.551	22.524	22.521	0	7	1
6	*Many first*	41.303	41.285	41.287	1	10	1
6	*Random*	15.303	232.510	299.430	37	41	9
6	*Sequential*	41.291	41.286	41.275	41	58	3
7	*Few first*	166.465	176.929	176.952	0	7	0
7	*Many first*	41.318	41.273	41.281	0	9	1
7	*Random*	41.119	181.215	175.132	40	50	21
7	*Sequential*	41.397	41.292	41.290	40	53	7
8	*Few first*	166.885	166.510	166.827	0	7	0
8	*Many first*	41.360	41.376	41.252	0	8	0
8	*Random*	166.523	39.177	60.059	36	42	19
8	*Sequential*	41.379	41.302	41.290	39	52	1
9	*Few first*	176.471	176.388	176.687	0	7	0
9	*Many first*	41.324	41.206	41.159	0	7	0
9	*Random*	207.271	154.297	142.280	36	38	12
9	*Sequential*	41.333	41.251	41.347	39	39	13
10	*Few first*	166.432	176.884	176.905	0	7	0
10	*Many first*	41.339	41.333	41.303	0	7	0
10	*Random*	227.167	176.651	59.138	35	33	11
10	*Sequential*	41.369	41.329	41.255	38	39	15

Table B.18: Tests with 6 variables to `frb40-19-1.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	55.978	58.966	241.667	0	11	5
2	*Many first*	110.157	110.103	109.937	3	11	0
2	*Random*	118.865	184.106	50.498	41	58	9
2	*Sequential*	110.116	110.176	110.003	49	63	0
3	*Few first*	276.451	276.137	326.596	1	11	2
3	*Many first*	110.098	109.942	109.836	2	11	2
3	*Random*	60.601	129.346	82.469	42	55	3
3	*Sequential*	110.156	110.064	109.877	49	63	2
4	*Few first*	32.066	33.812	35.316	0	10	3
4	*Many first*	110.194	110.012	109.882	1	10	2
4	*Random*	362.841	155.515	326.907	40	46	13
4	*Sequential*	110.181	109.949	109.911	46	63	18
5	*Few first*	38.190	38.238	232.151	0	10	4
5	*Many first*	110.127	109.939	109.797	0	9	2
5	*Random*	146.735	379.071	155.176	38	53	18
5	*Sequential*	110.122	110.021	109.921	45	63	1
6	*Few first*	405.328	431.248	226.887	0	10	2
6	*Many first*	110.164	110.023	109.925	0	9	2
6	*Random*	136.378	175.242	344.178	38	51	12
6	*Sequential*	110.122	110.022	110.004	42	60	2
7	*Few first*	571.121	571.028	571.234	0	9	1
7	*Many first*	110.048	110.044	109.918	0	9	1
7	*Random*	102.464	443.778	182.024	36	34	10
7	*Sequential*	110.194	110.065	109.912	40	60	4
8	*Few first*	146.575	146.385	146.636	0	8	0
8	*Many first*	110.077	110.074	109.904	0	8	0
8	*Random*	119.011	155.014	103.868	38	35	13
8	*Sequential*	110.129	110.164	110.021	39	60	7
9	*Few first*	146.439	146.404	146.314	0	8	0
9	*Many first*	110.089	110.330	110.003	0	8	0
9	*Random*	0.233	236.174	260.976	33	42	22
9	*Sequential*	110.051	110.120	110.006	39	59	8
10	*Few first*	155.106	155.088	155.242	0	8	0
10	*Many first*	109.924	110.172	109.942	0	8	0
10	*Random*	39.833	181.800	110.012	36	44	22
10	*Sequential*	110.020	110.147	110.025	39	56	23

Table B.19: Tests with 6 variables to `frb40-19-2.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	3558.481	3785.749	3107.774	1	10	3
2	*Many first*	2565.576	2563.677	2562.631	3	11	0
2	*Random*	197.348	212.058	368.977	44	60	3
2	*Sequential*	2566.281	2565.355	2562.136	49	63	0
3	*Few first*	5950.518	5975.464	545.551	0	10	4
3	*Many first*	2566.986	2565.770	2565.773	3	11	3
3	*Random*	1057.981	777.812	180.138	40	52	6
3	*Sequential*	2567.241	2563.255	2564.224	49	63	3
4	*Few first*	3329.402	3131.980	536.935	1	10	2
4	*Many first*	2567.709	2562.112	2570.789	1	10	4
4	*Random*	1078.103	6785.172	477.731	43	51	5
4	*Sequential*	2568.228	2566.799	2565.600	48	62	4
5	*Few first*	5641.796	5621.240	2716.937	0	10	3
5	*Many first*	2568.323	2567.703	2564.910	0	9	1
5	*Random*	185.528	7259.543	2606.838	41	44	6
5	*Sequential*	2570.610	2574.174	2563.430	47	62	3
6	*Few first*	941.811	941.892	2782.178	0	10	2
6	*Many first*	2565.717	2566.447	2568.494	0	9	2
6	*Random*	2726.789	3434.602	2267.415	37	45	6
6	*Sequential*	2565.765	2567.541	2561.386	45	62	3
7	*Few first*	7540.098	7532.027	7545.371	0	9	1
7	*Many first*	2569.563	2569.766	2565.263	0	9	1
7	*Random*	2738.961	2561.862	8.738	40	22	3
7	*Sequential*	2569.216	2565.815	2567.072	43	61	3
8	*Few first*	8.459	8.557	8.499	0	3	0
8	*Many first*	2565.841	2563.615	2562.658	0	9	1
8	*Random*	1836.494	8.576	1052.645	38	33	12
8	*Sequential*	2563.445	2561.511	2568.168	41	61	3
9	*Few first*	8.636	8.553	8.566	0	3	0
9	*Many first*	2570.324	2563.441	2564.404	0	9	0
9	*Random*	5739.101	2413.973	3857.758	39	25	5
9	*Sequential*	2569.275	2565.543	2568.176	41	61	15
10	*Few first*	8.505	8.828	8.825	0	3	0
10	*Many first*	2564.430	2567.503	2564.726	0	8	0
10	*Random*	2575.058	2082.067	224.108	36	19	7
10	*Sequential*	2565.439	2566.850	2562.230	41	56	5

Table B.20: Tests with 6 variables to `frb40-19-3.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	3548.995	2078.083	2074.343	0	10	3
2	*Many first*	2221.088	2217.923	2216.024	2	11	0
2	*Random*	508.604	499.020	339.171	43	61	1
2	*Sequential*	2220.341	2217.377	2215.679	53	63	0
3	*Few first*	2780.769	2778.452	2374.939	1	10	4
3	*Many first*	2221.259	2216.195	2214.099	1	11	4
3	*Random*	236.657	2647.885	683.568	45	39	5
3	*Sequential*	2219.970	2214.123	2213.211	51	63	3
4	*Few first*	1055.302	990.827	781.539	0	10	4
4	*Many first*	2233.069	2216.225	2215.692	0	10	4
4	*Random*	417.884	1674.743	2605.094	40	49	6
4	*Sequential*	2219.924	2216.031	2215.591	49	62	5
5	*Few first*	664.325	664.878	665.698	0	9	2
5	*Many first*	2218.510	2214.621	2215.778	0	10	3
5	*Random*	1409.254	219.142	1265.161	44	37	6
5	*Sequential*	2224.505	2218.348	2214.042	47	62	3
6	*Few first*	839.287	839.479	492.311	0	9	2
6	*Many first*	2217.489	2217.751	2217.120	0	10	3
6	*Random*	960.928	37.572	2080.836	40	38	8
6	*Sequential*	2217.409	2215.856	2215.280	45	60	3
7	*Few first*	1288.581	1372.834	1373.873	0	9	1
7	*Many first*	2216.861	2214.907	2216.688	0	9	1
7	*Random*	790.355	1232.207	2486.707	38	34	5
7	*Sequential*	2223.038	2214.655	2216.737	43	59	3
8	*Few first*	1396.644	1394.011	1393.624	0	8	0
8	*Many first*	2217.885	2215.820	2213.506	0	9	1
8	*Random*	1273.268	140.169	1116.823	37	37	3
8	*Sequential*	2223.934	2213.900	2213.553	41	54	2
9	*Few first*	1394.537	1395.689	1394.508	0	8	0
9	*Many first*	2219.532	2215.850	2215.914	0	9	0
9	*Random*	891.417	1393.659	2219.596	36	24	2
9	*Sequential*	2222.455	2215.904	2214.855	41	54	3
10	*Few first*	1393.844	1483.809	1485.947	0	8	0
10	*Many first*	2221.254	2215.708	2219.835	0	8	0
10	*Random*	1372.594	1677.424	496.931	36	12	1
10	*Sequential*	2216.677	2213.174	2214.863	40	40	0

Table B.21: Tests with 6 variables to `frb40-19-4.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	6313.364	6305.363	2909.593	1	10	3
2	*Many first*	3012.264	3010.201	3007.372	5	11	0
2	*Random*	109.749	766.344	4064.790	49	40	0
2	*Sequential*	3013.703	3012.495	3008.973	57	63	0
3	*Few first*	197.180	186.928	187.075	0	8	4
3	*Many first*	3015.763	3013.195	3011.897	5	11	4
3	*Random*	45.828	292.294	4690.640	44	36	6
3	*Sequential*	3018.281	3009.550	3010.253	57	63	4
4	*Few first*	3624.460	3629.614	1381.887	1	8	2
4	*Many first*	3017.545	3005.904	3015.307	3	10	3
4	*Random*	4956.477	5715.788	1029.105	43	38	6
4	*Sequential*	3020.366	3007.901	3012.502	52	62	5
5	*Few first*	3508.971	3737.632	3733.373	0	7	1
5	*Many first*	3015.476	3012.397	3011.739	0	9	2
5	*Random*	43.460	45.802	116.220	39	32	3
5	*Sequential*	3017.933	3013.633	3006.355	48	59	12
6	*Few first*	45.646	45.808	45.827	0	0	0
6	*Many first*	3013.879	3010.830	3011.843	0	8	1
6	*Random*	2031.227	5045.128	3014.929	38	26	1
6	*Sequential*	3018.080	3009.425	3012.205	45	52	1
7	*Few first*	43.638	45.776	45.760	0	0	0
7	*Many first*	3021.036	3007.519	3009.145	1	7	0
7	*Random*	4322.385	3012.146	2215.828	40	9	0
7	*Sequential*	3011.326	3014.809	3008.268	43	37	0
8	*Few first*	43.527	43.546	43.358	0	0	0
8	*Many first*	3018.957	3010.209	3015.237	0	6	0
8	*Random*	4742.856	2828.435	77.909	41	4	0
8	*Sequential*	3011.464	3007.857	3007.436	41	6	0
9	*Few first*	45.706	45.828	45.781	0	0	0
9	*Many first*	3014.278	3009.108	3012.496	0	6	0
9	*Random*	758.259	3681.561	130.037	38	6	0
9	*Sequential*	3013.143	3007.072	3007.817	41	6	0
10	*Few first*	43.492	45.876	45.852	0	0	0
10	*Many first*	3017.375	3007.574	3010.244	0	6	0
10	*Random*	2827.252	43.486	2588.937	37	6	0
10	*Sequential*	3016.002	3012.293	3009.746	41	6	0

Table B.22: Tests with 6 variables to `frb40-19-5.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	426.898	425.993	430.448	0	11	5
2	*Many first*	283.642	285.346	285.652	5	12	0
2	*Random*	373.513	345.657	356.065	41	56	4
2	*Sequential*	340.030	328.905	370.031	57	64	2
3	*Few first*	367.041	364.926	472.674	0	10	6
3	*Many first*	285.556	285.199	287.361	5	12	4
3	*Random*	329.199	303.309	409.247	43	62	18
3	*Sequential*	283.731	286.760	287.196	57	57	3
4	*Few first*	245.601	245.407	238.172	0	12	4
4	*Many first*	285.917	287.234	284.428	4	12	5
4	*Random*	225.809	287.299	427.714	43	60	30
4	*Sequential*	284.556	285.997	286.148	53	55	6
5	*Few first*	342.045	343.066	336.755	0	12	5
5	*Many first*	286.811	289.376	286.187	3	12	2
5	*Random*	284.928	418.480	496.216	40	62	40
5	*Sequential*	286.600	285.580	284.977	48	60	13
6	*Few first*	324.251	253.707	306.749	0	12	4
6	*Many first*	285.398	284.139	284.984	1	12	2
6	*Random*	313.108	462.108	285.514	34	62	33
6	*Sequential*	287.526	287.617	284.405	45	51	13
7	*Few first*	278.901	277.802	278.537	0	12	4
7	*Many first*	286.157	287.138	287.144	0	12	3
7	*Random*	306.769	279.849	283.864	35	63	36
7	*Sequential*	285.272	288.596	287.950	44	60	15
8	*Few first*	432.901	435.235	437.341	0	12	3
8	*Many first*	286.976	285.921	286.665	0	12	3
8	*Random*	311.016	233.729	262.300	31	60	39
8	*Sequential*	286.106	285.853	285.043	42	47	13
9	*Few first*	316.920	292.120	290.469	0	11	2
9	*Many first*	284.940	287.715	285.335	0	12	2
9	*Random*	416.157	271.087	284.361	21	60	46
9	*Sequential*	285.632	287.520	285.007	41	59	23
10	*Few first*	218.260	218.559	218.196	0	12	1
10	*Many first*	286.407	285.126	299.237	0	12	1
10	*Random*	219.422	258.179	285.187	29	63	48
10	*Sequential*	288.590	286.303	285.280	39	59	37

Table B.23: Tests with 6 variables to `hole11.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	85.226	85.154	85.216	1	12	7
2	*Many first*	75.553	75.151	77.612	1	12	6
2	*Random*	119.578	121.058	127.770	25	59	28
2	*Sequential*	123.353	121.923	130.889	28	64	25
3	*Few first*	54.673	55.055	56.250	1	12	5
3	*Many first*	68.458	67.833	67.438	1	10	6
3	*Random*	83.618	81.282	76.696	24	64	40
3	*Sequential*	88.409	85.077	87.097	28	64	33
4	*Few first*	42.934	42.803	43.054	1	12	5
4	*Many first*	43.091	67.299	48.585	0	10	5
4	*Random*	62.334	61.099	66.210	23	62	51
4	*Sequential*	65.888	63.124	59.938	28	64	34
5	*Few first*	38.384	38.198	38.368	1	12	6
5	*Many first*	55.480	55.027	60.089	0	10	4
5	*Random*	52.744	52.596	48.993	23	58	49
5	*Sequential*	50.234	51.574	49.709	28	64	38
6	*Few first*	38.389	38.394	38.407	1	12	4
6	*Many first*	40.926	40.781	44.796	0	10	3
6	*Random*	44.014	45.184	42.911	25	60	48
6	*Sequential*	43.859	44.040	43.398	28	56	43
7	*Few first*	38.358	38.777	38.383	1	12	2
7	*Many first*	44.165	44.577	41.381	0	10	2
7	*Random*	36.596	36.475	36.107	23	59	50
7	*Sequential*	38.650	38.381	37.095	26	56	39
8	*Few first*	38.430	38.348	38.758	1	12	1
8	*Many first*	40.476	41.090	37.454	0	10	1
8	*Random*	33.551	33.847	33.210	24	54	44
8	*Sequential*	33.551	32.370	34.390	24	56	38
9	*Few first*	38.567	38.395	38.163	1	10	0
9	*Many first*	29.017	29.264	28.839	0	10	0
9	*Random*	29.261	31.796	30.366	23	53	39
9	*Sequential*	30.654	30.940	30.838	23	56	41
10	*Few first*	38.785	38.773	38.519	0	10	0
10	*Many first*	38.238	38.310	38.214	0	10	0
10	*Random*	30.146	28.136	26.599	23	54	41
10	*Sequential*	32.286	28.328	29.667	21	52	38

Table B.24: Tests with 6 variables to `mod2-3cage-unsat-9-11.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	88.865	87.962	82.678	1	12	5
2	*Many first*	75.970	75.088	78.055	1	12	7
2	*Random*	111.910	117.784	119.525	29	62	27
2	*Sequential*	113.850	115.233	121.656	30	60	25
3	*Few first*	63.024	61.455	61.114	1	12	6
3	*Many first*	74.510	73.935	58.613	1	10	6
3	*Random*	79.533	73.353	77.021	28	64	40
3	*Sequential*	81.540	79.105	77.084	29	60	32
4	*Few first*	45.926	45.465	45.614	1	12	3
4	*Many first*	62.801	62.476	54.557	0	10	5
4	*Random*	62.769	60.966	54.930	27	60	50
4	*Sequential*	60.137	62.254	63.270	28	60	35
5	*Few first*	45.678	45.486	45.696	1	12	4
5	*Many first*	48.747	48.575	45.741	0	10	4
5	*Random*	47.508	46.019	48.245	28	58	45
5	*Sequential*	46.931	47.281	49.577	27	60	41
6	*Few first*	46.206	46.572	45.772	1	12	3
6	*Many first*	45.315	44.885	42.677	0	10	3
6	*Random*	39.919	39.515	44.982	28	62	52
6	*Sequential*	40.502	40.762	39.246	27	60	43
7	*Few first*	45.762	45.633	46.091	1	12	2
7	*Many first*	41.898	41.685	38.713	0	10	2
7	*Random*	37.365	35.479	34.122	28	56	48
7	*Sequential*	34.388	36.746	37.583	27	60	39
8	*Few first*	45.901	45.466	45.808	1	12	1
8	*Many first*	42.950	42.339	45.818	0	10	1
8	*Random*	31.606	33.876	31.528	25	56	44
8	*Sequential*	30.057	30.271	29.846	27	60	41
9	*Few first*	45.749	45.469	45.663	1	10	0
9	*Many first*	35.033	35.206	35.711	0	10	0
9	*Random*	28.197	30.642	29.256	26	59	47
9	*Sequential*	29.032	29.092	27.729	27	60	40
10	*Few first*	45.608	46.076	46.183	0	10	0
10	*Many first*	35.328	35.319	35.680	0	10	0
10	*Random*	29.344	30.311	27.482	24	55	44
10	*Sequential*	28.680	30.305	27.259	26	52	41

Table B.25: Tests with 6 variables to `mod2-3cage-unsat-9-4.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	1970.777	1983.184	2017.331	0	8	4
2	*Many first*	72.712	69.942	79.636	0	2	1
2	*Random*	1959.462	350.073	1178.658	0	39	28
2	*Sequential*	1016.979	829.480	858.845	0	30	19
3	*Few first*	1311.096	1306.513	1282.021	0	7	3
3	*Many first*	41.745	41.716	43.605	0	2	1
3	*Random*	1187.223	910.177	897.168	0	43	40
3	*Sequential*	626.724	625.952	649.861	0	31	29
4	*Few first*	700.455	701.713	662.182	0	6	4
4	*Many first*	37.162	37.158	37.325	0	1	0
4	*Random*	37.176	611.312	458.580	0	33	32
4	*Sequential*	486.736	497.884	432.305	0	30	29
5	*Few first*	448.255	446.922	460.816	0	6	5
5	*Many first*	37.329	37.142	37.084	0	1	0
5	*Random*	700.841	37.168	141.947	0	9	8
5	*Sequential*	436.856	435.852	371.547	0	31	30
6	*Few first*	415.771	414.670	393.241	0	5	2
6	*Many first*	37.102	37.262	37.175	0	1	0
6	*Random*	514.166	85.656	37.069	0	1	0
6	*Sequential*	337.966	317.816	277.295	0	27	26
7	*Few first*	257.972	258.196	265.222	0	4	2
7	*Many first*	37.147	37.091	37.239	0	1	0
7	*Random*	409.330	319.653	479.619	0	56	55
7	*Sequential*	231.397	232.152	244.663	0	27	26
8	*Few first*	169.474	169.933	174.919	0	2	1
8	*Many first*	37.377	37.621	37.219	0	1	0
8	*Random*	155.336	314.942	303.913	0	37	36
8	*Sequential*	276.467	187.987	196.623	0	26	25
9	*Few first*	68.032	67.879	67.756	0	1	0
9	*Many first*	37.194	37.292	38.028	0	1	0
9	*Random*	108.336	36.575	374.921	0	46	45
9	*Sequential*	185.410	185.224	216.728	0	25	24
10	*Few first*	37.399	37.406	37.404	0	0	0
10	*Many first*	37.368	37.284	37.507	0	0	0
10	*Random*	135.219	369.256	341.372	0	48	47
10	*Sequential*	180.587	187.370	179.547	0	26	25

Table B.26: Tests with 6 variables to `mod2-3g14-sat.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	99.384	99.988	105.496	0	2	1
2	*Many first*	131.798	132.246	107.473	0	4	3
2	*Random*	237.513	35.099	88.340	4	7	4
2	*Sequential*	21.008	21.081	21.030	1	2	0
3	*Few first*	4.713	4.724	4.764	0	0	0
3	*Many first*	80.658	80.945	76.697	0	3	2
3	*Random*	159.311	26.743	99.015	0	9	7
3	*Sequential*	10.708	10.732	10.713	1	1	0
4	*Few first*	4.733	4.739	4.762	0	0	0
4	*Many first*	53.506	53.506	70.288	0	4	3
4	*Random*	23.664	64.800	23.938	5	2	0
4	*Sequential*	26.557	26.709	26.829	0	1	0
5	*Few first*	4.756	4.731	4.733	0	0	0
5	*Many first*	61.607	62.110	47.743	0	2	1
5	*Random*	26.815	47.224	23.734	6	4	2
5	*Sequential*	26.892	26.637	26.963	0	3	2
6	*Few first*	4.742	4.721	4.750	0	0	0
6	*Many first*	42.595	42.293	42.383	0	1	0
6	*Random*	96.979	26.675	64.939	2	16	13
6	*Sequential*	10.683	10.763	26.851	0	5	3
7	*Few first*	4.734	4.722	4.730	0	0	0
7	*Many first*	6.885	6.903	6.973	0	0	0
7	*Random*	56.215	56.471	34.772	9	10	5
7	*Sequential*	12.505	12.536	23.781	0	4	3
8	*Few first*	4.751	4.744	4.726	0	0	0
8	*Many first*	4.748	4.769	4.719	0	0	0
8	*Random*	26.771	48.522	26.655	6	7	6
8	*Sequential*	26.271	26.334	26.634	0	5	4
9	*Few first*	4.759	4.753	4.725	0	0	0
9	*Many first*	4.748	4.749	4.750	0	0	0
9	*Random*	26.685	34.610	26.709	7	8	5
9	*Sequential*	21.199	21.112	26.717	0	3	2
10	*Few first*	4.744	4.745	4.734	0	0	0
10	*Many first*	4.741	4.772	4.753	0	0	0
10	*Random*	26.717	26.707	34.552	5	6	5
10	*Sequential*	23.837	23.793	26.730	0	5	4

Table B.27: Tests with 6 variables to `mod2c-rand3bip-sat-150-11.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	24.642	24.678	24.724	0	0	0
2	*Many first*	113.870	115.582	146.487	0	11	8
2	*Random*	0.998	21.392	9.654	0	2	1
2	*Sequential*	31.417	31.345	47.142	0	9	7
3	*Few first*	24.599	24.772	24.630	0	1	0
3	*Many first*	106.181	105.901	103.062	0	10	8
3	*Random*	44.238	44.374	10.842	0	2	1
3	*Sequential*	24.183	24.386	22.488	0	7	6
4	*Few first*	24.639	24.676	24.583	0	2	1
4	*Many first*	41.855	41.791	44.125	0	9	7
4	*Random*	30.509	16.410	0.998	0	0	0
4	*Sequential*	17.576	17.493	26.947	0	10	9
5	*Few first*	24.700	24.853	24.783	0	3	2
5	*Many first*	33.382	34.090	32.421	0	8	6
5	*Random*	7.881	0.993	11.844	0	4	3
5	*Sequential*	11.801	11.826	15.860	0	6	5
6	*Few first*	24.609	24.615	24.600	0	4	3
6	*Many first*	54.980	54.702	40.918	0	9	5
6	*Random*	8.108	1.000	1.005	0	0	0
6	*Sequential*	10.378	10.407	13.041	0	5	4
7	*Few first*	24.544	24.604	24.608	0	5	4
7	*Many first*	26.428	26.470	24.787	0	8	4
7	*Random*	0.996	0.998	1.002	0	0	0
7	*Sequential*	10.291	10.314	15.172	0	8	7
8	*Few first*	24.569	24.700	24.693	0	6	3
8	*Many first*	13.692	13.794	46.992	0	10	3
8	*Random*	0.998	7.910	1.000	0	0	0
8	*Sequential*	9.597	10.735	11.421	0	7	6
9	*Few first*	24.605	24.635	25.003	0	9	0
9	*Many first*	50.395	50.446	26.465	0	9	2
9	*Random*	11.170	1.001	18.620	0	12	11
9	*Sequential*	6.773	6.826	6.753	0	1	0
10	*Few first*	24.718	24.619	24.895	0	9	1
10	*Many first*	50.180	50.471	26.505	0	10	1
10	*Random*	1.008	1.006	0.999	0	0	0
10	*Sequential*	1.007	1.003	1.005	0	0	0

Table B.28: Tests with 6 variables to `mod2c-rand3bip-sat-150-15.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	100.490	101.772	100.093	0	0	0
2	*Many first*	149.822	149.707	207.412	0	10	9
2	*Random*	27.661	38.429	104.578	0	32	29
2	*Sequential*	54.213	55.599	50.295	0	13	12
3	*Few first*	100.480	100.781	100.703	0	1	0
3	*Many first*	105.383	106.101	128.315	0	10	8
3	*Random*	13.842	82.959	17.035	0	5	4
3	*Sequential*	32.564	35.033	35.304	0	13	12
4	*Few first*	100.445	101.192	100.439	0	1	0
4	*Many first*	52.203	52.052	75.156	0	8	7
4	*Random*	50.314	16.210	10.492	0	4	3
4	*Sequential*	22.297	22.175	30.493	0	13	12
5	*Few first*	99.692	101.278	100.594	0	3	2
5	*Many first*	52.444	52.609	61.660	0	9	6
5	*Random*	43.000	21.164	28.064	0	15	14
5	*Sequential*	19.731	19.802	24.061	0	14	13
6	*Few first*	101.280	101.089	101.245	0	9	4
6	*Many first*	27.175	26.953	99.706	0	10	5
6	*Random*	9.624	32.089	8.478	0	5	4
6	*Sequential*	15.552	15.504	12.281	0	9	8
7	*Few first*	101.842	100.277	101.018	0	9	4
7	*Many first*	77.167	76.769	79.719	0	9	4
7	*Random*	15.208	10.713	7.209	0	3	2
7	*Sequential*	13.949	13.908	10.381	0	9	8
8	*Few first*	100.647	100.654	100.100	0	9	1
8	*Many first*	79.904	79.897	50.079	0	8	3
8	*Random*	48.061	40.180	7.573	0	4	3
8	*Sequential*	8.489	8.495	16.691	0	11	10
9	*Few first*	100.587	101.164	101.927	0	9	2
9	*Many first*	12.678	12.630	12.050	0	4	2
9	*Random*	7.157	7.146	19.435	0	16	15
9	*Sequential*	8.431	8.438	13.657	0	11	10
10	*Few first*	100.229	100.759	100.808	0	9	1
10	*Many first*	78.031	77.930	100.298	0	10	1
10	*Random*	6.372	18.877	32.603	0	39	38
10	*Sequential*	12.600	12.674	14.124	0	12	11

Table B.29: Tests with 6 variables to `sat2.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	1628.368	1663.211	1628.724	0	12	0
2	*Many first*	1806.623	1856.338	1458.557	0	12	5
2	*Random*	827.134	840.816	895.556	0	64	31
2	*Sequential*	862.634	856.078	792.221	0	64	28
3	*Few first*	1635.336	1633.573	1623.783	0	12	6
3	*Many first*	1777.101	1763.021	1648.722	0	12	7
3	*Random*	626.190	564.021	630.454	0	64	47
3	*Sequential*	559.621	601.805	578.648	0	64	49
4	*Few first*	1626.277	1633.814	1618.524	0	12	0
4	*Many first*	2091.194	2099.032	1500.822	0	12	6
4	*Random*	433.829	474.495	409.168	0	64	55
4	*Sequential*	424.977	439.628	457.413	0	64	56
5	*Few first*	1638.634	1634.167	1617.254	0	12	3
5	*Many first*	1599.609	1611.376	1721.390	0	12	6
5	*Random*	393.104	377.238	367.388	0	64	56
5	*Sequential*	343.634	342.404	374.888	0	64	58
6	*Few first*	1622.719	1621.832	1614.463	0	12	4
6	*Many first*	1349.479	1353.514	1430.028	0	12	5
6	*Random*	330.102	346.447	345.199	0	64	56
6	*Sequential*	339.657	309.793	340.693	0	64	57
7	*Few first*	1651.205	1626.947	1627.250	0	12	4
7	*Many first*	1429.253	1429.236	1332.958	0	12	4
7	*Random*	302.019	290.670	319.855	0	64	55
7	*Sequential*	286.223	289.701	339.129	0	64	55
8	*Few first*	1630.633	1629.498	1629.701	0	12	3
8	*Many first*	1667.533	1674.641	1681.235	0	12	3
8	*Random*	221.216	255.354	235.667	0	64	55
8	*Sequential*	278.456	292.451	255.374	0	64	54
9	*Few first*	1634.073	1642.445	1627.575	0	12	1
9	*Many first*	1638.990	1651.951	1850.591	0	12	2
9	*Random*	268.785	256.997	210.706	0	64	53
9	*Sequential*	212.765	213.678	236.545	0	64	54
10	*Few first*	1646.367	1619.387	1626.930	0	12	1
10	*Many first*	1190.945	1195.397	1179.863	0	12	1
10	*Random*	198.606	207.468	211.476	0	64	53
10	*Sequential*	233.451	223.740	195.317	0	64	53

Table B.30: Tests with 6 variables to `unif-r4.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	130.581	131.772	132.147	0	0	0
2	*Many first*	2.311	2.321	2.328	0	0	0
2	*Random*	19.804	6.757	12.178	0	0	0
2	*Sequential*	2.323	2.318	2.318	0	0	0
3	*Few first*	2.942	2.933	2.935	0	0	0
3	*Many first*	2.324	2.316	2.313	0	0	0
3	*Random*	2.323	42.222	31.088	0	0	0
3	*Sequential*	2.316	2.324	2.333	0	0	0
4	*Few first*	2.952	2.932	2.920	0	0	0
4	*Many first*	2.348	2.316	2.314	0	0	0
4	*Random*	33.699	12.116	18.023	0	0	0
4	*Sequential*	2.342	2.333	2.318	0	0	0
5	*Few first*	2.940	2.930	2.911	0	0	0
5	*Many first*	2.318	2.331	2.318	0	0	0
5	*Random*	15.270	7.677	1.430	0	0	0
5	*Sequential*	2.333	2.334	2.316	0	0	0
6	*Few first*	2.941	2.919	2.927	0	0	0
6	*Many first*	2.321	2.358	2.322	0	0	0
6	*Random*	8.388	1.431	1.977	0	0	0
6	*Sequential*	2.320	2.332	2.325	0	0	0
7	*Few first*	2.938	2.914	2.928	0	0	0
7	*Many first*	2.327	2.314	2.325	0	0	0
7	*Random*	2.330	42.492	7.677	0	0	0
7	*Sequential*	2.313	2.311	2.313	0	0	0
8	*Few first*	2.929	2.934	2.925	0	0	0
8	*Many first*	2.323	2.318	2.310	0	0	0
8	*Random*	5.040	19.802	6.782	0	0	0
8	*Sequential*	2.351	2.341	2.321	0	0	0
9	*Few first*	2.942	2.940	2.953	0	0	0
9	*Many first*	2.335	2.333	2.320	0	0	0
9	*Random*	1.431	7.705	6.750	0	0	0
9	*Sequential*	2.339	2.337	2.309	0	0	0
10	*Few first*	2.935	2.923	2.963	0	0	0
10	*Many first*	2.355	2.927	2.938	0	0	0
10	*Random*	1.423	1.430	2.068	0	0	0
10	*Sequential*	2.336	2.334	2.344	0	0	0

Table B.31: Tests with 6 variables to `unif-r5.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	21.717	21.686	21.255	0	10	3
2	*Many first*	11.151	11.141	11.139	5	11	0
2	*Random*	7.613	9.851	3.959	49	61	5
2	*Sequential*	11.125	11.098	11.104	57	63	0
3	*Few first*	13.050	13.083	12.284	0	9	3
3	*Many first*	11.139	11.106	11.115	5	11	3
3	*Random*	11.156	21.032	1.967	42	40	7
3	*Sequential*	11.170	11.090	11.140	56	63	7
4	*Few first*	7.349	7.360	5.379	0	8	2
4	*Many first*	11.168	11.115	11.141	3	10	4
4	*Random*	5.198	15.755	7.664	41	44	6
4	*Sequential*	11.173	11.096	11.091	52	62	9
5	*Few first*	32.319	32.292	32.419	0	7	1
5	*Many first*	11.154	11.090	11.122	1	9	3
5	*Random*	21.982	11.245	11.111	42	43	3
5	*Sequential*	11.163	11.075	11.079	47	59	17
6	*Few first*	11.261	11.273	11.282	0	6	0
6	*Many first*	11.176	11.103	11.132	0	8	0
6	*Random*	11.820	0.365	1.156	39	31	1
6	*Sequential*	11.135	11.114	11.122	43	52	14
7	*Few first*	10.791	11.293	11.382	0	6	0
7	*Many first*	11.191	11.138	11.133	0	7	0
7	*Random*	10.741	5.212	5.177	37	21	0
7	*Sequential*	11.198	11.172	11.130	43	37	0
8	*Few first*	10.800	10.826	11.365	0	6	0
8	*Many first*	11.178	11.106	11.121	0	6	0
8	*Random*	22.262	13.355	0.080	35	0	0
8	*Sequential*	11.179	11.092	11.115	41	6	0
9	*Few first*	11.285	10.819	11.359	0	6	0
9	*Many first*	11.162	11.147	11.112	0	6	0
9	*Random*	5.360	22.326	5.072	32	6	0
9	*Sequential*	11.150	11.128	11.133	41	6	0
10	*Few first*	11.282	11.299	11.279	0	6	0
10	*Many first*	11.201	11.166	11.090	0	6	0
10	*Random*	0.375	11.283	11.147	34	6	0
10	*Sequential*	11.164	11.115	11.105	41	6	0

Table B.32: Tests with 6 variables to `vmpc_21.renamed-as.sat05-1923.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	6.026	5.941	52.634	0	10	3
2	*Many first*	116.382	117.279	116.713	5	11	0
2	*Random*	60.770	26.607	118.318	43	50	1
2	*Sequential*	116.372	116.433	116.536	57	63	0
3	*Few first*	174.335	174.613	18.910	0	9	4
3	*Many first*	116.450	116.366	116.401	4	11	3
3	*Random*	109.881	2.953	6.879	47	43	5
3	*Sequential*	116.416	116.414	116.318	57	63	2
4	*Few first*	70.099	69.977	1.677	0	5	2
4	*Many first*	116.583	116.360	116.175	3	10	3
4	*Random*	39.536	13.890	52.466	43	16	1
4	*Sequential*	116.519	116.700	116.399	51	62	5
5	*Few first*	197.157	197.077	75.606	0	7	1
5	*Many first*	116.451	116.350	116.396	2	9	2
5	*Random*	57.515	5.697	86.859	45	16	10
5	*Sequential*	116.302	116.347	116.365	48	59	3
6	*Few first*	5.711	5.736	5.786	0	6	0
6	*Many first*	116.424	116.488	116.707	0	8	1
6	*Random*	5.724	5.732	116.380	41	17	1
6	*Sequential*	116.436	116.259	116.229	45	52	1
7	*Few first*	5.720	5.724	5.742	0	6	0
7	*Many first*	116.637	116.402	116.523	0	7	0
7	*Random*	5.571	57.749	5.704	38	6	0
7	*Sequential*	116.528	116.476	116.547	43	37	0
8	*Few first*	5.560	5.552	5.746	0	6	0
8	*Many first*	116.389	116.413	116.371	0	6	0
8	*Random*	79.131	60.896	54.358	38	6	0
8	*Sequential*	116.303	116.397	116.265	41	6	0
9	*Few first*	5.756	5.715	5.676	0	6	0
9	*Many first*	116.795	116.403	116.300	0	6	0
9	*Random*	74.825	67.604	5.741	37	6	0
9	*Sequential*	116.317	116.444	116.306	41	6	0
10	*Few first*	5.746	5.743	5.735	0	6	0
10	*Many first*	116.500	116.436	116.298	0	6	0
10	*Random*	87.113	12.174	12.075	37	6	0
10	*Sequential*	116.406	116.447	116.411	41	6	0

Table B.33: Tests with 6 variables to `vmpc_23.renamed-as.sat05-1927.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	5.698	5.689	5.707	0	0	0
2	*Many first*	46.515	42.167	46.568	5	7	0
2	*Random*	5.888	16.939	14.971	48	11	0
2	*Sequential*	24.993	23.063	24.966	46	3	0
3	*Few first*	9.384	9.327	9.368	0	4	1
3	*Many first*	18.454	18.929	11.918	3	6	1
3	*Random*	6.847	1.952	1.182	38	0	0
3	*Sequential*	14.676	14.627	14.726	45	6	1
4	*Few first*	9.330	9.374	9.331	0	5	2
4	*Many first*	5.412	5.382	8.450	0	4	1
4	*Random*	4.080	8.888	4.693	43	4	1
4	*Sequential*	5.697	5.687	5.683	15	0	0
5	*Few first*	9.330	9.363	9.351	0	5	1
5	*Many first*	9.179	9.141	9.168	0	4	1
5	*Random*	1.167	9.677	3.907	44	22	0
5	*Sequential*	9.685	9.694	9.696	47	24	1
6	*Few first*	9.371	9.387	9.324	0	4	0
6	*Many first*	5.526	5.505	5.725	0	4	1
6	*Random*	1.175	5.680	1.208	42	0	0
6	*Sequential*	9.718	9.669	9.634	44	50	1
7	*Few first*	9.366	9.341	9.352	0	3	0
7	*Many first*	5.711	5.544	5.677	0	4	0
7	*Random*	11.131	5.551	5.693	40	10	0
7	*Sequential*	9.683	9.761	9.754	42	35	0
8	*Few first*	9.372	9.352	9.369	0	3	0
8	*Many first*	5.787	5.723	5.684	0	3	0
8	*Random*	2.092	8.927	9.684	39	5	0
8	*Sequential*	9.706	9.763	9.650	41	4	0
9	*Few first*	9.360	9.369	9.357	0	3	0
9	*Many first*	9.729	9.749	9.330	0	3	0
9	*Random*	8.982	9.689	8.930	36	5	0
9	*Sequential*	9.688	9.720	9.657	41	4	0
10	*Few first*	9.343	9.352	9.342	0	3	0
10	*Many first*	9.719	9.758	9.355	0	3	0
10	*Random*	0.732	9.336	0.304	37	0	0
10	*Sequential*	9.670	9.761	9.712	41	4	0

Table B.34: Tests with 6 variables to `vmpc_25.renamed-as.sat05-1913.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	273.056	36.989	62.258	0	10	3
2	*Many first*	68.471	68.199	68.263	0	11	0
2	*Random*	47.532	49.825	430.579	36	63	7
2	*Sequential*	68.233	68.399	68.403	37	63	0
3	*Few first*	71.647	71.528	214.816	0	9	2
3	*Many first*	68.370	68.281	68.339	0	11	4
3	*Random*	383.779	169.311	161.643	34	59	9
3	*Sequential*	68.411	68.208	68.187	37	61	3
4	*Few first*	133.974	134.147	158.035	0	9	2
4	*Many first*	68.307	68.271	68.282	0	11	6
4	*Random*	13.005	267.627	174.864	36	60	8
4	*Sequential*	68.330	68.247	68.252	36	52	5
5	*Few first*	172.179	173.325	172.547	0	9	1
5	*Many first*	68.287	68.405	68.195	0	11	4
5	*Random*	172.747	152.186	24.436	36	57	7
5	*Sequential*	68.510	68.151	68.302	36	51	3
6	*Few first*	130.101	130.065	92.965	0	9	2
6	*Many first*	68.343	68.230	68.277	0	11	4
6	*Random*	5.107	358.525	422.742	34	49	20
6	*Sequential*	68.370	68.422	68.174	35	51	9
7	*Few first*	299.048	299.003	6.691	0	8	2
7	*Many first*	68.177	68.242	68.218	0	11	2
7	*Random*	221.623	68.299	178.551	36	49	29
7	*Sequential*	68.327	68.228	68.177	34	49	10
8	*Few first*	139.628	139.774	139.626	0	9	1
8	*Many first*	68.192	68.145	68.209	0	11	2
8	*Random*	68.342	185.713	4.964	32	46	32
8	*Sequential*	68.202	68.169	68.294	33	48	16
9	*Few first*	383.895	385.268	383.908	0	9	0
9	*Many first*	68.369	68.160	68.373	0	11	1
9	*Random*	171.798	212.433	118.610	34	45	21
9	*Sequential*	68.314	68.361	68.297	31	47	22
10	*Few first*	384.740	384.460	384.320	0	9	0
10	*Many first*	68.460	68.064	68.356	0	10	1
10	*Random*	32.111	68.464	65.061	34	43	23
10	*Sequential*	68.377	68.095	68.204	31	47	21

Table B.35: Tests with 6 variables to `vmpc_25.shuffled-as.sat05-1945.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	1480.721	1480.946	1385.304	0	10	4
2	*Many first*	699.213	698.813	654.370	0	11	0
2	*Random*	161.318	1040.357	396.801	40	62	3
2	*Sequential*	697.276	698.300	653.603	49	63	0
3	*Few first*	314.214	314.660	143.639	0	9	4
3	*Many first*	697.477	698.032	654.960	0	11	2
3	*Random*	587.400	1246.321	299.787	42	62	4
3	*Sequential*	697.504	699.174	652.654	47	63	3
4	*Few first*	466.465	467.827	828.877	0	8	2
4	*Many first*	698.759	699.483	652.449	0	11	6
4	*Random*	897.184	695.736	317.831	43	43	9
4	*Sequential*	697.822	697.883	653.416	45	62	5
5	*Few first*	116.604	116.567	116.102	0	8	1
5	*Many first*	698.703	696.948	654.565	0	11	3
5	*Random*	160.441	315.261	378.062	42	43	13
5	*Sequential*	696.787	700.306	652.971	43	59	3
6	*Few first*	750.540	751.248	704.030	0	8	1
6	*Many first*	697.461	700.174	652.082	0	11	4
6	*Random*	321.777	123.495	1271.070	44	34	13
6	*Sequential*	697.821	698.538	653.677	43	52	5
7	*Few first*	767.832	767.419	767.945	0	8	1
7	*Many first*	697.609	698.995	652.500	0	11	2
7	*Random*	697.809	161.324	539.472	37	38	15
7	*Sequential*	697.815	698.230	653.143	42	52	1
8	*Few first*	151.955	160.724	160.631	0	8	0
8	*Many first*	697.346	698.514	846.481	0	10	1
8	*Random*	508.856	1335.868	160.174	39	40	12
8	*Sequential*	697.539	698.378	696.305	41	52	3
9	*Few first*	160.719	160.560	151.318	0	8	0
9	*Many first*	699.847	699.749	699.131	0	9	0
9	*Random*	455.750	159.453	2.111	36	40	15
9	*Sequential*	698.512	697.763	729.439	41	51	8
10	*Few first*	161.046	161.338	151.150	0	8	0
10	*Many first*	697.351	697.904	701.142	0	8	0
10	*Random*	1163.422	160.631	160.826	34	27	6
10	*Sequential*	697.451	697.099	700.230	40	48	8

Table B.36: Tests with 6 variables to `vmpc_26.renamed-as.sat05-1914.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	281.450	281.318	498.085	0	11	5
2	*Many first*	140.249	140.637	133.185	0	11	0
2	*Random*	455.853	324.441	163.221	20	63	7
2	*Sequential*	140.449	140.293	133.400	21	61	0
3	*Few first*	69.600	69.618	447.135	0	11	6
3	*Many first*	140.354	140.411	133.144	0	11	4
3	*Random*	59.649	125.113	63.839	20	62	14
3	*Sequential*	140.835	140.408	133.410	20	61	11
4	*Few first*	222.331	222.472	40.581	0	11	6
4	*Many first*	140.772	140.211	133.681	0	11	6
4	*Random*	90.720	312.264	144.392	20	60	24
4	*Sequential*	140.422	140.163	133.369	20	61	17
5	*Few first*	18.724	18.706	389.511	0	11	3
5	*Many first*	140.552	140.327	133.076	0	11	4
5	*Random*	95.011	661.961	100.092	20	60	26
5	*Sequential*	140.641	140.530	133.387	20	61	31
6	*Few first*	448.699	133.185	73.120	0	11	4
6	*Many first*	140.294	140.412	133.451	0	11	4
6	*Random*	136.951	19.827	53.587	19	61	41
6	*Sequential*	140.590	140.271	133.415	20	61	38
7	*Few first*	193.197	193.357	34.817	0	11	2
7	*Many first*	140.293	140.559	133.607	0	11	3
7	*Random*	326.327	67.543	652.060	20	62	34
7	*Sequential*	140.564	140.624	133.663	20	61	34
8	*Few first*	551.501	551.363	16.614	0	11	2
8	*Many first*	140.048	140.388	141.029	0	11	2
8	*Random*	12.380	557.533	159.531	18	60	30
8	*Sequential*	140.598	140.454	140.509	20	59	46
9	*Few first*	67.913	68.232	90.829	0	11	2
9	*Many first*	140.450	140.346	171.349	0	11	2
9	*Random*	213.427	79.581	213.187	18	56	37
9	*Sequential*	140.586	140.260	140.751	20	59	45
10	*Few first*	11.303	11.386	11.678	0	11	1
10	*Many first*	140.105	140.128	140.461	0	11	1
10	*Random*	133.073	369.771	97.200	16	59	49
10	*Sequential*	140.437	140.570	171.377	19	59	46

Table B.37: Tests with 6 variables to `vmpc_26.shuffled-as.sat05-1946.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	965.678	903.634	906.187	0	10	3
2	*Many first*	957.029	896.254	897.852	5	11	0
2	*Random*	904.209	682.452	954.997	46	61	0
2	*Sequential*	958.239	895.926	900.563	57	63	0
3	*Few first*	421.557	421.470	421.156	0	9	4
3	*Many first*	899.192	898.380	897.050	5	11	0
3	*Random*	57.139	60.037	1183.062	42	41	5
3	*Sequential*	955.922	898.795	898.093	56	63	4
4	*Few first*	152.387	151.824	1396.148	0	8	2
4	*Many first*	896.482	897.370	900.964	4	10	1
4	*Random*	164.382	897.457	60.243	43	32	4
4	*Sequential*	896.898	899.021	897.893	52	62	3
5	*Few first*	2305.539	2288.459	2299.492	0	7	1
5	*Many first*	903.058	897.974	896.193	3	9	2
5	*Random*	1782.006	482.038	1580.021	42	43	1
5	*Sequential*	897.532	900.640	897.538	48	59	3
6	*Few first*	60.218	60.207	57.257	0	5	0
6	*Many first*	901.741	899.083	898.969	2	8	0
6	*Random*	2904.091	57.337	451.522	43	26	1
6	*Sequential*	898.720	897.896	900.111	45	52	1
7	*Few first*	57.201	60.229	60.087	0	5	0
7	*Many first*	897.918	898.603	902.485	1	7	0
7	*Random*	609.017	60.153	1876.761	38	17	0
7	*Sequential*	895.390	896.696	897.932	43	37	0
8	*Few first*	57.235	60.472	60.120	0	5	0
8	*Many first*	897.481	896.197	897.626	0	6	0
8	*Random*	814.810	33.394	541.318	39	6	0
8	*Sequential*	914.572	897.868	897.839	41	6	0
9	*Few first*	60.219	60.055	57.296	0	5	0
9	*Many first*	899.748	895.576	900.857	0	6	0
9	*Random*	1066.883	577.063	1535.769	37	6	0
9	*Sequential*	897.232	898.510	897.014	41	6	0
10	*Few first*	60.396	60.358	57.298	0	5	0
10	*Many first*	897.045	897.749	896.950	0	6	0
10	*Random*	183.402	2005.047	2913.734	35	6	0
10	*Sequential*	898.433	896.648	898.652	41	6	0

Table B.38: Tests with 6 variables to `vmpc_27.renamed-as.sat05-1915.cnf`

#W	Mode	T-no-opts	T-confs	T-learnts	E.A.	Sent	Recv.
2	*Few first*	21.015	20.937	20.200	0	5	1
2	*Many first*	34.431	34.290	34.313	0	3	0
2	*Random*	35.820	34.830	10.580	22	13	1
2	*Sequential*	33.340	38.587	33.334	18	15	0
3	*Few first*	14.896	14.888	14.771	0	4	0
3	*Many first*	19.351	19.378	20.022	0	4	1
3	*Random*	0.879	4.226	2.753	18	13	2
3	*Sequential*	18.711	17.022	16.370	22	22	2
4	*Few first*	6.656	6.697	3.998	0	3	1
4	*Many first*	2.671	2.685	2.606	0	1	0
4	*Random*	3.152	3.546	3.525	21	15	2
4	*Sequential*	3.028	4.842	5.938	18	13	1
5	*Few first*	0.465	0.479	0.675	0	3	2
5	*Many first*	5.310	5.323	5.270	0	2	0
5	*Random*	0.888	10.927	6.202	19	27	11
5	*Sequential*	2.388	2.387	0.971	22	12	4
6	*Few first*	2.955	2.974	2.945	0	5	3
6	*Many first*	5.295	5.327	5.242	0	3	1
6	*Random*	5.211	1.301	6.510	19	39	37
6	*Sequential*	0.766	0.769	2.365	18	29	12
7	*Few first*	5.315	5.294	5.330	0	5	4
7	*Many first*	5.289	5.307	5.276	0	4	2
7	*Random*	5.159	3.258	3.655	20	50	18
7	*Sequential*	3.772	3.260	5.245	22	57	20
8	*Few first*	5.271	5.313	5.152	0	5	3
8	*Many first*	5.309	5.299	5.288	0	5	3
8	*Random*	2.644	4.835	3.785	20	46	28
8	*Sequential*	3.403	3.397	3.297	22	57	31
9	*Few first*	5.304	5.324	5.156	0	5	2
9	*Many first*	5.342	5.309	5.292	0	5	1
9	*Random*	5.988	6.241	5.751	20	53	37
9	*Sequential*	2.673	2.659	0.755	22	23	13
10	*Few first*	5.281	5.340	5.153	0	5	1
10	*Many first*	5.318	5.318	5.283	0	5	1
10	*Random*	2.696	5.609	7.903	19	53	41
10	*Sequential*	3.807	0.497	0.996	21	42	30

Table B.39: Tests with 6 variables to `vmpc_27.shuffled-as.sat05-1947.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	35.976	1.251	0.625	0.599
2	*Many first*	2	36.089	1.247	0.623	0.604
2	*Random*	2	13.331	3.376	1.688	-0.408
2	*Sequential*	2	35.936	1.252	0.626	0.597
3	*Few first*	5	54.322	0.828	0.276	1.311
3	*Many first*	5	75.245	0.598	0.199	2.008
3	*Random*	4	42.987	1.047	0.349	0.933
3	*Sequential*	4	29.825	1.509	0.503	0.494
4	*Few first*	8	40.766	1.104	0.276	0.875
4	*Many first*	8	92.965	0.484	0.121	2.421
4	*Random*	4	54.057	0.832	0.208	1.268
4	*Sequential*	4	57.028	0.789	0.197	1.356
5	*Few first*	13	69.816	0.645	0.129	1.689
5	*Many first*	13	105.817	0.425	0.085	2.689
5	*Random*	5	40.304	1.116	0.223	0.870
5	*Sequential*	5	48.283	0.932	0.186	1.091
6	*Few first*	18	57.183	0.787	0.131	1.325
6	*Many first*	18	123.143	0.365	0.061	3.084
6	*Random*	6	27.786	1.619	0.270	0.541
6	*Sequential*	6	37.293	1.207	0.201	0.795
7	*Few first*	25	57.161	0.787	0.112	1.315
7	*Many first*	25	100.698	0.447	0.064	2.444
7	*Random*	6	47.307	0.951	0.136	1.060
7	*Sequential*	6	38.712	1.162	0.166	0.837
8	*Few first*	32	73.270	0.614	0.077	1.718
8	*Many first*	32	131.557	0.342	0.043	3.198
8	*Random*	6	62.754	0.717	0.090	1.451
8	*Sequential*	6	36.556	1.231	0.154	0.786
9	*Few first*	41	84.145	0.535	0.059	1.979
9	*Many first*	41	138.720	0.324	0.036	3.343
9	*Random*	7	46.999	0.957	0.106	1.050
9	*Sequential*	7	56.981	0.790	0.088	1.300
10	*Few first*	50	72.420	0.621	0.062	1.677
10	*Many first*	50	99.775	0.451	0.045	2.353
10	*Random*	7	41.851	1.075	0.108	0.922
10	*Sequential*	7	40.192	1.120	0.112	0.881

Table B.40: Tests of granularity to `fpga10_11_uns_rcr.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	153.403	1.033	0.517	0.936
2	*Many first*	2	107.087	1.480	0.740	0.351
2	*Random*	2	107.825	1.470	0.735	0.361
2	*Sequential*	2	107.455	1.475	0.737	0.356
3	*Few first*	5	52.116	3.041	1.014	-0.007
3	*Many first*	5	171.346	0.925	0.308	1.122
3	*Random*	4	173.532	0.913	0.304	1.143
3	*Sequential*	4	159.960	0.991	0.330	1.014
4	*Few first*	8	148.773	1.065	0.266	0.918
4	*Many first*	8	218.099	0.727	0.182	1.502
4	*Random*	4	170.116	0.932	0.233	1.098
4	*Sequential*	4	161.092	0.984	0.246	1.022
5	*Few first*	13	114.755	1.381	0.276	0.655
5	*Many first*	13	263.936	0.600	0.120	1.832
5	*Random*	5	111.992	1.415	0.283	0.633
5	*Sequential*	5	102.197	1.551	0.310	0.556
6	*Few first*	18	88.749	1.786	0.298	0.472
6	*Many first*	18	96.789	1.637	0.273	0.533
6	*Random*	6	119.384	1.327	0.221	0.704
6	*Sequential*	6	158.970	0.997	0.166	1.004
7	*Few first*	25	156.979	1.010	0.144	0.989
7	*Many first*	25	233.559	0.679	0.097	1.553
7	*Random*	6	95.102	1.666	0.238	0.533
7	*Sequential*	6	113.920	1.391	0.199	0.672
8	*Few first*	32	103.854	1.526	0.191	0.606
8	*Many first*	32	217.038	0.730	0.091	1.422
8	*Random*	6	164.187	0.965	0.121	1.041
8	*Sequential*	6	122.622	1.292	0.162	0.741
9	*Few first*	41	100.800	1.572	0.175	0.591
9	*Many first*	41	246.925	0.642	0.071	1.628
9	*Random*	7	130.311	1.216	0.135	0.800
9	*Sequential*	7	166.430	0.952	0.106	1.056
10	*Few first*	50	151.223	1.048	0.105	0.949
10	*Many first*	50	201.664	0.786	0.079	1.303
10	*Random*	7	139.184	1.139	0.114	0.865
10	*Sequential*	7	122.568	1.293	0.129	0.748

Table B.41: Tests of granularity to `fpga10_12_uns_rcr.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	261.553	0.618	0.309	2.237
2	*Many first*	2	277.787	0.582	0.291	2.438
2	*Random*	2	273.079	0.592	0.296	2.380
2	*Sequential*	2	278.164	0.581	0.290	2.443
3	*Few first*	5	181.370	0.891	0.297	1.184
3	*Many first*	5	368.151	0.439	0.146	2.918
3	*Random*	4	263.722	0.613	0.204	1.948
3	*Sequential*	4	267.578	0.604	0.201	1.984
4	*Few first*	8	204.875	0.789	0.197	1.357
4	*Many first*	8	515.525	0.313	0.078	3.921
4	*Random*	4	266.992	0.605	0.151	1.870
4	*Sequential*	4	266.619	0.606	0.152	1.867
5	*Few first*	13	230.736	0.700	0.140	1.535
5	*Many first*	13	228.985	0.706	0.141	1.521
5	*Random*	5	230.449	0.701	0.140	1.533
5	*Sequential*	5	230.450	0.701	0.140	1.533
6	*Few first*	18	285.096	0.567	0.094	1.917
6	*Many first*	18	498.068	0.324	0.054	3.499
6	*Random*	6	255.283	0.633	0.105	1.696
6	*Sequential*	6	201.401	0.802	0.134	1.296
7	*Few first*	25	287.232	0.563	0.080	1.907
7	*Many first*	25	192.263	0.840	0.120	1.221
7	*Random*	6	256.195	0.631	0.090	1.683
7	*Sequential*	6	298.359	0.542	0.077	1.988
8	*Few first*	32	294.774	0.548	0.069	1.942
8	*Many first*	32	301.983	0.535	0.067	1.993
8	*Random*	6	272.463	0.593	0.074	1.784
8	*Sequential*	6	339.090	0.477	0.060	2.255
9	*Few first*	41	222.944	0.725	0.081	1.427
9	*Many first*	41	277.885	0.581	0.065	1.810
9	*Random*	7	303.006	0.533	0.059	1.985
9	*Sequential*	7	278.124	0.581	0.065	1.811
10	*Few first*	50	199.995	0.808	0.081	1.264
10	*Many first*	50	404.268	0.400	0.040	2.669
10	*Random*	7	312.704	0.517	0.052	2.039
10	*Sequential*	7	213.588	0.757	0.076	1.358

Table B.42: Tests of granularity to `fpga10_13_uns_rcr.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	14.735	19.505	9.752	-0.897
2	*Many first*	2	314.960	0.913	0.456	1.192
2	*Random*	2	15.386	18.680	9.340	-0.893
2	*Sequential*	2	315.537	0.911	0.455	1.196
3	*Few first*	5	43.234	6.648	2.216	-0.274
3	*Many first*	5	44.732	6.425	2.142	-0.267
3	*Random*	4	173.262	1.659	0.553	0.404
3	*Sequential*	4	181.933	1.580	0.527	0.450
4	*Few first*	8	91.772	3.132	0.783	0.092
4	*Many first*	8	130.070	2.210	0.552	0.270
4	*Random*	4	52.051	5.522	1.380	-0.092
4	*Sequential*	4	182.232	1.577	0.394	0.512
5	*Few first*	13	16.514	17.404	3.481	-0.178
5	*Many first*	13	49.222	5.839	1.168	-0.036
5	*Random*	5	57.777	4.974	0.995	0.001
5	*Sequential*	5	44.842	6.409	1.282	-0.055
6	*Few first*	18	45.262	6.350	1.058	-0.011
6	*Many first*	18	57.568	4.992	0.832	0.040
6	*Random*	6	45.219	6.356	1.059	-0.011
6	*Sequential*	6	41.206	6.975	1.162	-0.028
7	*Few first*	25	2.740	104.885	14.984	-0.156
7	*Many first*	25	10.348	27.775	3.968	-0.125
7	*Random*	6	183.637	1.565	0.224	0.579
7	*Sequential*	6	41.369	6.947	0.992	0.001
8	*Few first*	32	4.602	62.448	7.806	-0.125
8	*Many first*	32	13.857	20.741	2.593	-0.088
8	*Random*	6	64.098	4.484	0.560	0.112
8	*Sequential*	6	41.297	6.959	0.870	0.021
9	*Few first*	41	6.699	42.900	4.767	-0.099
9	*Many first*	41	10.829	26.541	2.949	-0.083
9	*Random*	7	10.571	27.189	3.021	-0.084
9	*Sequential*	7	251.187	1.144	0.127	0.858
10	*Few first*	50	8.625	33.324	3.332	-0.078
10	*Many first*	50	1.919	149.758	14.976	-0.104
10	*Random*	7	243.348	1.181	0.118	0.830
10	*Sequential*	7	250.908	1.145	0.115	0.859

Table B.43: Tests of granularity to `frb40-19-1.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	396.999	1.361	0.681	0.469
2	*Many first*	2	172.703	3.129	1.565	-0.361
2	*Random*	2	296.884	1.820	0.910	0.099
2	*Sequential*	2	172.879	3.126	1.563	-0.360
3	*Few first*	5	348.856	1.549	0.516	0.468
3	*Many first*	5	445.817	1.212	0.404	0.737
3	*Random*	4	210.774	2.564	0.855	0.085
3	*Sequential*	4	243.907	2.216	0.739	0.177
4	*Few first*	8	135.132	3.999	1.000	0.000
4	*Many first*	8	22.821	23.679	5.920	-0.277
4	*Random*	4	249.196	2.169	0.542	0.282
4	*Sequential*	4	243.999	2.215	0.554	0.269
5	*Few first*	13	61.748	8.752	1.750	-0.107
5	*Many first*	13	217.848	2.481	0.496	0.254
5	*Random*	5	183.573	2.944	0.589	0.175
5	*Sequential*	5	445.334	1.213	0.243	0.780
6	*Few first*	18	235.482	2.295	0.382	0.323
6	*Many first*	18	21.047	25.675	4.279	-0.153
6	*Random*	6	135.243	3.996	0.666	0.100
6	*Sequential*	6	109.987	4.913	0.819	0.044
7	*Few first*	25	62.155	8.694	1.242	-0.032
7	*Many first*	25	42.660	12.668	1.810	-0.075
7	*Random*	6	171.614	3.149	0.450	0.204
7	*Sequential*	6	109.962	4.914	0.702	0.071
8	*Few first*	32	62.253	8.681	1.085	-0.011
8	*Many first*	32	38.555	14.016	1.752	-0.061
8	*Random*	6	203.707	2.653	0.332	0.288
8	*Sequential*	6	110.542	4.889	0.611	0.091
9	*Few first*	41	48.727	11.090	1.232	-0.024
9	*Many first*	41	48.461	11.151	1.239	-0.024
9	*Random*	7	357.864	1.510	0.168	0.620
9	*Sequential*	7	18.656	28.966	3.218	-0.086
10	*Few first*	50	33.121	16.316	1.632	-0.043
10	*Many first*	50	5.743	94.090	9.409	-0.099
10	*Random*	7	24.416	22.133	2.213	-0.061
10	*Sequential*	7	18.726	28.858	2.886	-0.073

Table B.44: Tests of granularity to `frb40-19-2.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	283.521	22.363	11.182	-0.911
2	*Many first*	2	1116.105	5.681	2.840	-0.648
2	*Random*	2	1050.828	6.034	3.017	-0.669
2	*Sequential*	2	1116.282	5.680	2.840	-0.648
3	*Few first*	5	1939.433	3.269	1.090	-0.041
3	*Many first*	5	5347.412	1.186	0.395	0.765
3	*Random*	4	1452.988	4.364	1.455	-0.156
3	*Sequential*	4	3535.978	1.793	0.598	0.337
4	*Few first*	8	4427.766	1.432	0.358	0.598
4	*Many first*	8	812.754	7.801	1.950	-0.162
4	*Random*	4	2930.817	2.163	0.541	0.283
4	*Sequential*	4	3540.866	1.791	0.448	0.411
5	*Few first*	13	483.353	13.118	2.624	-0.155
5	*Many first*	13	868.544	7.300	1.460	-0.079
5	*Random*	5	3919.569	1.618	0.324	0.523
5	*Sequential*	5	5340.404	1.187	0.237	0.803
6	*Few first*	18	122.590	51.721	8.620	-0.177
6	*Many first*	18	323.517	19.598	3.266	-0.139
6	*Random*	6	4039.151	1.570	0.262	0.564
6	*Sequential*	6	2569.352	2.468	0.411	0.286
7	*Few first*	25	194.927	32.527	4.647	-0.131
7	*Many first*	25	170.655	37.153	5.308	-0.135
7	*Random*	6	869.401	7.293	1.042	-0.007
7	*Sequential*	6	2570.362	2.467	0.352	0.306
8	*Few first*	32	248.815	25.482	3.185	-0.098
8	*Many first*	32	76.301	83.098	10.387	-0.129
8	*Random*	6	2952.430	2.148	0.268	0.389
8	*Sequential*	6	2566.808	2.470	0.309	0.320
9	*Few first*	41	246.594	25.712	2.857	-0.081
9	*Many first*	41	130.180	48.705	5.412	-0.102
9	*Random*	7	4489.855	1.412	0.157	0.672
9	*Sequential*	7	4266.898	1.486	0.165	0.632
10	*Few first*	50	261.795	24.219	2.422	-0.065
10	*Many first*	50	9.868	642.548	64.255	-0.109
10	*Random*	7	2208.127	2.871	0.287	0.276
10	*Sequential*	7	4268.087	1.486	0.149	0.637

Table B.45: Tests of granularity to `frb40-19-3.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	1893.267	0.476	0.238	3.201
2	*Many first*	2	753.569	1.196	0.598	0.672
2	*Random*	2	707.978	1.273	0.637	0.571
2	*Sequential*	2	752.964	1.197	0.599	0.671
3	*Few first*	5	1404.943	0.642	0.214	1.838
3	*Many first*	5	641.289	1.406	0.469	0.567
3	*Random*	4	1299.805	0.694	0.231	1.663
3	*Sequential*	4	512.734	1.758	0.586	0.353
4	*Few first*	8	185.048	4.871	1.218	-0.060
4	*Many first*	8	957.261	0.942	0.235	1.083
4	*Random*	4	727.854	1.239	0.310	0.743
4	*Sequential*	4	512.435	1.759	0.440	0.425
5	*Few first*	13	1763.783	0.511	0.102	2.196
5	*Many first*	13	1031.589	0.874	0.175	1.180
5	*Random*	5	2129.295	0.423	0.085	2.703
5	*Sequential*	5	640.635	1.407	0.281	0.638
6	*Few first*	18	189.955	4.746	0.791	0.053
6	*Many first*	18	158.857	5.675	0.946	0.011
6	*Random*	6	186.475	4.834	0.806	0.048
6	*Sequential*	6	2218.451	0.406	0.068	2.753
7	*Few first*	25	804.692	1.120	0.160	0.875
7	*Many first*	25	653.003	1.380	0.197	0.678
7	*Random*	6	267.317	3.372	0.482	0.179
7	*Sequential*	6	2218.024	0.406	0.058	2.704
8	*Few first*	32	59.774	15.081	1.885	-0.067
8	*Many first*	32	158.862	5.674	0.709	0.059
8	*Random*	6	705.357	1.278	0.160	0.751
8	*Sequential*	6	2218.262	0.406	0.051	2.669
9	*Few first*	41	25.892	34.816	3.868	-0.093
9	*Many first*	41	85.838	10.502	1.167	-0.018
9	*Random*	7	1520.749	0.593	0.066	1.773
9	*Sequential*	7	1621.905	0.556	0.062	1.899
10	*Few first*	50	61.610	14.632	1.463	-0.035
10	*Many first*	50	42.099	21.413	2.141	-0.059
10	*Random*	7	984.884	0.915	0.092	1.103
10	*Sequential*	7	1622.259	0.556	0.056	1.888

Table B.46: Tests of granularity to `frb40-19-4.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	3125.763	1.449	0.724	0.380
2	*Many first*	2	2506.659	1.807	0.903	0.107
2	*Random*	2	2815.566	1.608	0.804	0.243
2	*Sequential*	2	2504.600	1.808	0.904	0.106
3	*Few first*	5	728.539	6.216	2.072	-0.259
3	*Many first*	5	1830.210	2.474	0.825	0.106
3	*Random*	4	3776.043	1.199	0.400	0.751
3	*Sequential*	4	863.271	5.246	1.749	-0.214
4	*Few first*	8	2106.309	2.150	0.538	0.287
4	*Many first*	8	1446.983	3.130	0.782	0.093
4	*Random*	4	860.391	5.264	1.316	-0.080
4	*Sequential*	4	862.507	5.251	1.313	-0.079
5	*Few first*	13	1722.834	2.629	0.526	0.226
5	*Many first*	13	2400.639	1.886	0.377	0.413
5	*Random*	5	1720.040	2.633	0.527	0.225
5	*Sequential*	5	1832.168	2.472	0.494	0.256
6	*Few first*	18	3106.062	1.458	0.243	0.623
6	*Many first*	18	661.626	6.845	1.141	-0.025
6	*Random*	6	3678.953	1.231	0.205	0.775
6	*Sequential*	6	3020.710	1.499	0.250	0.600
7	*Few first*	25	981.194	4.616	0.659	0.086
7	*Many first*	25	991.282	4.569	0.653	0.089
7	*Random*	6	3143.725	1.441	0.206	0.643
7	*Sequential*	6	3016.172	1.501	0.214	0.610
8	*Few first*	32	628.764	7.203	0.900	0.016
8	*Many first*	32	90.015	50.311	6.289	-0.120
8	*Random*	6	4980.721	0.909	0.114	1.114
8	*Sequential*	6	3016.091	1.502	0.188	0.618
9	*Few first*	41	1.804	2510.234	278.915	-0.125
9	*Many first*	41	143.133	31.640	3.516	-0.089
9	*Random*	7	2263.274	2.001	0.222	0.437
9	*Sequential*	7	4303.542	1.052	0.117	0.944
10	*Few first*	50	97.558	46.421	4.642	-0.087
10	*Many first*	50	121.311	37.332	3.733	-0.081
10	*Random*	7	3849.541	1.176	0.118	0.833
10	*Sequential*	7	4313.740	1.050	0.105	0.947

Table B.47: Tests of granularity to `frb40-19-5.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	970.758	0.745	0.372	1.685
2	*Many first*	2	947.924	0.763	0.381	1.622
2	*Random*	2	944.699	0.765	0.383	1.613
2	*Sequential*	2	952.244	0.759	0.380	1.634
3	*Few first*	5	531.032	1.362	0.454	0.601
3	*Many first*	5	288.570	2.506	0.835	0.099
3	*Random*	4	283.535	2.551	0.850	0.088
3	*Sequential*	4	359.205	2.013	0.671	0.245
4	*Few first*	8	356.403	2.029	0.507	0.324
4	*Many first*	8	301.006	2.402	0.601	0.222
4	*Random*	4	844.631	0.856	0.214	1.224
4	*Sequential*	4	357.140	2.025	0.506	0.325
5	*Few first*	13	241.104	2.999	0.600	0.167
5	*Many first*	13	155.709	4.644	0.929	0.019
5	*Random*	5	458.609	1.577	0.315	0.543
5	*Sequential*	5	289.405	2.499	0.500	0.250
6	*Few first*	18	138.499	5.221	0.870	0.030
6	*Many first*	18	104.224	6.939	1.156	-0.027
6	*Random*	6	275.617	2.624	0.437	0.257
6	*Sequential*	6	284.553	2.541	0.424	0.272
7	*Few first*	25	84.578	8.550	1.221	-0.030
7	*Many first*	25	141.397	5.114	0.731	0.061
7	*Random*	6	216.460	3.341	0.477	0.183
7	*Sequential*	6	286.506	2.524	0.361	0.296
8	*Few first*	32	82.781	8.736	1.092	-0.012
8	*Many first*	32	100.666	7.184	0.898	0.016
8	*Random*	6	273.108	2.648	0.331	0.289
8	*Sequential*	6	284.446	2.542	0.318	0.307
9	*Few first*	41	64.229	11.259	1.251	-0.025
9	*Many first*	41	68.378	10.576	1.175	-0.019
9	*Random*	7	319.205	2.265	0.252	0.372
9	*Sequential*	7	125.387	5.767	0.641	0.070
10	*Few first*	50	95.637	7.561	0.756	0.036
10	*Many first*	50	114.168	6.334	0.633	0.064
10	*Random*	7	318.462	2.271	0.227	0.378
10	*Sequential*	7	124.076	5.828	0.583	0.080

Table B.48: Tests of granularity to `hole11.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	72.164	1.089	0.544	0.837
2	*Many first*	2	70.843	1.109	0.555	0.803
2	*Random*	2	75.082	1.046	0.523	0.911
2	*Sequential*	2	72.977	1.077	0.538	0.858
3	*Few first*	5	47.290	1.662	0.554	0.403
3	*Many first*	5	53.876	1.458	0.486	0.529
3	*Random*	4	58.352	1.347	0.449	0.614
3	*Sequential*	4	60.513	1.298	0.433	0.655
4	*Few first*	8	43.509	1.806	0.451	0.405
4	*Many first*	8	60.217	1.305	0.326	0.689
4	*Random*	4	47.954	1.639	0.410	0.480
4	*Sequential*	4	47.153	1.666	0.417	0.467
5	*Few first*	13	34.215	2.296	0.459	0.294
5	*Many first*	13	42.974	1.828	0.366	0.434
5	*Random*	5	32.480	2.419	0.484	0.267
5	*Sequential*	5	39.129	2.008	0.402	0.373
6	*Few first*	18	36.903	2.129	0.355	0.364
6	*Many first*	18	50.740	1.549	0.258	0.575
6	*Random*	6	44.290	1.774	0.296	0.476
6	*Sequential*	6	44.011	1.785	0.298	0.472
7	*Few first*	25	33.598	2.339	0.334	0.332
7	*Many first*	25	37.211	2.112	0.302	0.386
7	*Random*	6	35.892	2.189	0.313	0.366
7	*Sequential*	6	36.729	2.139	0.306	0.379
8	*Few first*	32	35.587	2.208	0.276	0.375
8	*Many first*	32	36.549	2.150	0.269	0.389
8	*Random*	6	34.920	2.250	0.281	0.365
8	*Sequential*	6	33.376	2.354	0.294	0.343
9	*Few first*	41	27.309	2.877	0.320	0.266
9	*Many first*	41	27.355	2.872	0.319	0.267
9	*Random*	7	32.571	2.412	0.268	0.341
9	*Sequential*	7	30.454	2.580	0.287	0.311
10	*Few first*	50	26.452	2.970	0.297	0.263
10	*Many first*	50	38.675	2.032	0.203	0.436
10	*Random*	7	29.721	2.644	0.264	0.309
10	*Sequential*	7	30.542	2.573	0.257	0.321

Table B.49: Tests of granularity to `mod2-3cage-unsat-9-11.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	74.752	1.080	0.540	0.851
2	*Many first*	2	77.552	1.041	0.521	0.921
2	*Random*	2	78.703	1.026	0.513	0.949
2	*Sequential*	2	71.913	1.123	0.561	0.781
3	*Few first*	5	55.567	1.453	0.484	0.532
3	*Many first*	5	55.941	1.444	0.481	0.539
3	*Random*	4	57.118	1.414	0.471	0.561
3	*Sequential*	4	59.014	1.368	0.456	0.596
4	*Few first*	8	40.223	2.008	0.502	0.331
4	*Many first*	8	56.059	1.441	0.360	0.592
4	*Random*	4	42.977	1.879	0.470	0.376
4	*Sequential*	4	45.614	1.770	0.443	0.420
5	*Few first*	13	31.158	2.592	0.518	0.232
5	*Many first*	13	44.243	1.825	0.365	0.435
5	*Random*	5	41.656	1.939	0.388	0.395
5	*Sequential*	5	45.598	1.771	0.354	0.456
6	*Few first*	18	35.975	2.245	0.374	0.335
6	*Many first*	18	50.246	1.607	0.268	0.547
6	*Random*	6	40.965	1.971	0.329	0.409
6	*Sequential*	6	42.048	1.921	0.320	0.425
7	*Few first*	25	30.584	2.640	0.377	0.275
7	*Many first*	25	39.566	2.041	0.292	0.405
7	*Random*	6	38.237	2.112	0.302	0.386
7	*Sequential*	6	35.771	2.257	0.322	0.350
8	*Few first*	32	29.656	2.723	0.340	0.277
8	*Many first*	32	34.240	2.358	0.295	0.342
8	*Random*	6	32.673	2.472	0.309	0.320
8	*Sequential*	6	30.060	2.686	0.336	0.283
9	*Few first*	41	38.351	2.106	0.234	0.409
9	*Many first*	41	38.094	2.120	0.236	0.406
9	*Random*	7	30.509	2.647	0.294	0.300
9	*Sequential*	7	27.457	2.941	0.327	0.258
10	*Few first*	50	34.820	2.319	0.232	0.368
10	*Many first*	50	35.701	2.262	0.226	0.380
10	*Random*	7	28.250	2.859	0.286	0.278
10	*Sequential*	7	24.190	3.338	0.334	0.222

Table B.50: Tests of granularity to `mod2-3cage-unsat-9-4.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	424.646	2.066	1.033	-0.032
2	*Many first*	2	1654.469	0.530	0.265	2.772
2	*Random*	2	1666.981	0.526	0.263	2.800
2	*Sequential*	2	1757.525	0.499	0.250	3.006
3	*Few first*	5	917.457	0.956	0.319	1.069
3	*Many first*	5	92.399	9.495	3.165	-0.342
3	*Random*	4	512.433	1.712	0.571	0.376
3	*Sequential*	4	657.699	1.334	0.445	0.624
4	*Few first*	8	365.342	2.401	0.600	0.222
4	*Many first*	8	583.584	1.503	0.376	0.554
4	*Random*	4	6.535	134.245	33.561	-0.323
4	*Sequential*	4	567.343	1.546	0.387	0.529
5	*Few first*	13	507.026	1.730	0.346	0.472
5	*Many first*	13	40.786	21.511	4.302	-0.192
5	*Random*	5	177.272	4.949	0.990	0.003
5	*Sequential*	5	334.038	2.626	0.525	0.226
6	*Few first*	18	100.374	8.741	1.457	-0.063
6	*Many first*	18	758.165	1.157	0.193	0.837
6	*Random*	6	189.008	4.642	0.774	0.059
6	*Sequential*	6	292.802	2.996	0.499	0.200
7	*Few first*	25	2.433	360.581	51.512	-0.163
7	*Many first*	25	303.416	2.892	0.413	0.237
7	*Random*	6	268.626	3.266	0.467	0.191
7	*Sequential*	6	232.881	3.767	0.538	0.143
8	*Few first*	32	28.735	30.533	3.817	-0.105
8	*Many first*	32	730.889	1.200	0.150	0.809
8	*Random*	6	84.620	10.368	1.296	-0.033
8	*Sequential*	6	188.644	4.651	0.581	0.103
9	*Few first*	41	189.468	4.631	0.515	0.118
9	*Many first*	41	29.453	29.788	3.310	-0.087
9	*Random*	7	155.697	5.635	0.626	0.075
9	*Sequential*	7	193.558	4.533	0.504	0.123
10	*Few first*	50	272.895	3.215	0.321	0.234
10	*Many first*	50	278.811	3.147	0.315	0.242
10	*Random*	7	134.539	6.521	0.652	0.059
10	*Sequential*	7	174.040	5.041	0.504	0.109

Table B.51: Tests of granularity to `mod2-3g14-sat.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	95.578	0.788	0.394	1.537
2	*Many first*	2	119.166	0.632	0.316	2.164
2	*Random*	2	118.417	0.636	0.318	2.144
2	*Sequential*	2	118.638	0.635	0.317	2.150
3	*Few first*	5	18.500	4.072	1.357	-0.132
3	*Many first*	5	69.213	1.088	0.363	0.878
3	*Random*	4	115.930	0.650	0.217	1.808
3	*Sequential*	4	52.498	1.435	0.478	0.545
4	*Few first*	8	82.197	0.916	0.229	1.121
4	*Many first*	8	17.370	4.337	1.084	-0.026
4	*Random*	4	52.947	1.423	0.356	0.604
4	*Sequential*	4	52.741	1.428	0.357	0.600
5	*Few first*	13	9.135	8.247	1.649	-0.098
5	*Many first*	13	33.347	2.259	0.452	0.303
5	*Random*	5	11.567	6.513	1.303	-0.058
5	*Sequential*	5	11.519	6.540	1.308	-0.059
6	*Few first*	18	5.520	13.646	2.274	-0.112
6	*Many first*	18	35.413	2.127	0.355	0.364
6	*Random*	6	34.729	2.169	0.362	0.353
6	*Sequential*	6	10.811	6.968	1.161	-0.028
7	*Few first*	25	4.444	16.951	2.422	-0.098
7	*Many first*	25	77.350	0.974	0.139	1.031
7	*Random*	6	26.824	2.808	0.401	0.249
7	*Sequential*	6	12.528	6.013	0.859	0.027
8	*Few first*	32	6.720	11.210	1.401	-0.041
8	*Many first*	32	32.317	2.331	0.291	0.347
8	*Random*	6	26.802	2.811	0.351	0.264
8	*Sequential*	6	26.286	2.866	0.358	0.256
9	*Few first*	41	19.684	3.827	0.425	0.169
9	*Many first*	41	7.187	10.481	1.165	-0.018
9	*Random*	7	40.607	1.855	0.206	0.481
9	*Sequential*	7	5.688	13.243	1.471	-0.040
10	*Few first*	50	9.942	7.578	0.758	0.036
10	*Many first*	50	7.411	10.164	1.016	-0.002
10	*Random*	7	24.006	3.138	0.314	0.243
10	*Sequential*	7	12.935	5.824	0.582	0.080

Table B.52: Tests of granularity to `mod2c-rand3bip-sat-150-11.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	11.515	2.306	1.153	-0.133
2	*Many first*	2	15.492	1.714	0.857	0.167
2	*Random*	2	15.580	1.704	0.852	0.173
2	*Sequential*	2	15.462	1.717	0.859	0.165
3	*Few first*	5	39.804	0.667	0.222	1.749
3	*Many first*	5	22.933	1.158	0.386	0.795
3	*Random*	4	6.996	3.795	1.265	-0.105
3	*Sequential*	4	18.392	1.444	0.481	0.539
4	*Few first*	8	18.807	1.412	0.353	0.611
4	*Many first*	8	60.001	0.443	0.111	2.679
4	*Random*	4	6.962	3.814	0.953	0.016
4	*Sequential*	4	6.966	3.812	0.953	0.016
5	*Few first*	13	37.325	0.711	0.142	1.507
5	*Many first*	13	39.359	0.675	0.135	1.603
5	*Random*	5	0.847	31.348	6.270	-0.210
5	*Sequential*	5	11.361	2.337	0.467	0.285
6	*Few first*	18	2.573	10.319	1.720	-0.084
6	*Many first*	18	6.029	4.404	0.734	0.072
6	*Random*	6	1.000	26.552	4.425	-0.155
6	*Sequential*	6	10.430	2.546	0.424	0.271
7	*Few first*	25	3.006	8.833	1.262	-0.035
7	*Many first*	25	20.943	1.268	0.181	0.754
7	*Random*	6	1.002	26.499	3.786	-0.123
7	*Sequential*	6	10.326	2.572	0.367	0.287
8	*Few first*	32	1.218	21.800	2.725	-0.090
8	*Many first*	32	1.860	14.275	1.784	-0.063
8	*Random*	6	0.994	26.712	3.339	-0.100
8	*Sequential*	6	9.621	2.760	0.345	0.271
9	*Few first*	41	6.692	3.968	0.441	0.159
9	*Many first*	41	6.474	4.101	0.456	0.149
9	*Random*	7	1.992	13.329	1.481	-0.041
9	*Sequential*	7	15.721	1.689	0.188	0.541
10	*Few first*	50	4.918	5.399	0.540	0.095
10	*Many first*	50	21.969	1.209	0.121	0.808
10	*Random*	7	1.990	13.343	1.334	-0.028
10	*Sequential*	7	15.747	1.686	0.169	0.548

Table B.53: Tests of granularity to `mod2c-rand3bip-sat-150-15.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	139.151	0.478	0.239	3.182
2	*Many first*	2	76.619	0.869	0.434	1.303
2	*Random*	2	76.735	0.867	0.434	1.306
2	*Sequential*	2	139.505	0.477	0.239	3.192
3	*Few first*	5	12.558	5.300	1.767	-0.217
3	*Many first*	5	18.635	3.571	1.190	-0.080
3	*Random*	4	36.453	1.826	0.609	0.322
3	*Sequential*	4	55.739	1.194	0.398	0.756
4	*Few first*	8	122.634	0.543	0.136	2.124
4	*Many first*	8	31.185	2.134	0.534	0.291
4	*Random*	4	59.823	1.112	0.278	0.865
4	*Sequential*	4	42.953	1.549	0.387	0.527
5	*Few first*	13	15.551	4.280	0.856	0.042
5	*Many first*	13	21.413	3.108	0.622	0.152
5	*Random*	5	69.184	0.962	0.192	1.049
5	*Sequential*	5	24.478	2.719	0.544	0.210
6	*Few first*	18	0.508	131.001	21.833	-0.191
6	*Many first*	18	1.270	52.400	8.733	-0.177
6	*Random*	6	12.546	5.305	0.884	0.026
6	*Sequential*	6	15.497	4.295	0.716	0.079
7	*Few first*	25	11.051	6.022	0.860	0.027
7	*Many first*	25	10.175	6.541	0.934	0.012
7	*Random*	6	6.326	10.520	1.503	-0.056
7	*Sequential*	6	14.986	4.441	0.634	0.096
8	*Few first*	32	2.310	28.809	3.601	-0.103
8	*Many first*	32	41.222	1.614	0.202	0.565
8	*Random*	6	9.694	6.866	0.858	0.024
8	*Sequential*	6	9.225	7.215	0.902	0.016
9	*Few first*	41	1.825	36.465	4.052	-0.094
9	*Many first*	41	24.551	2.711	0.301	0.290
9	*Random*	7	34.118	1.951	0.217	0.452
9	*Sequential*	7	15.007	4.435	0.493	0.129
10	*Few first*	50	24.818	2.682	0.268	0.303
10	*Many first*	50	0.691	96.308	9.631	-0.100
10	*Random*	7	8.877	7.498	0.750	0.037
10	*Sequential*	7	15.100	4.407	0.441	0.141

Table B.54: Tests of granularity to `sat2.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	1579.641	2.055	1.028	-0.027
2	*Many first*	2	1262.911	2.571	1.285	-0.222
2	*Random*	2	1252.302	2.593	1.296	-0.229
2	*Sequential*	2	1270.520	2.556	1.278	-0.217
3	*Few first*	5	966.974	3.358	1.119	-0.053
3	*Many first*	5	1486.251	2.185	0.728	0.187
3	*Random*	4	801.909	4.049	1.350	-0.130
3	*Sequential*	4	924.586	3.512	1.171	-0.073
4	*Few first*	8	991.063	3.276	0.819	0.074
4	*Many first*	8	1075.701	3.018	0.755	0.108
4	*Random*	4	542.112	5.989	1.497	-0.111
4	*Sequential*	4	621.887	5.221	1.305	-0.078
5	*Few first*	13	910.292	3.567	0.713	0.100
5	*Many first*	13	1192.058	2.724	0.545	0.209
5	*Random*	5	470.510	6.901	1.380	-0.069
5	*Sequential*	5	470.235	6.905	1.381	-0.069
6	*Few first*	18	947.606	3.426	0.571	0.150
6	*Many first*	18	1178.945	2.754	0.459	0.236
6	*Random*	6	353.808	9.177	1.530	-0.069
6	*Sequential*	6	299.513	10.841	1.807	-0.089
7	*Few first*	25	1468.172	2.212	0.316	0.361
7	*Many first*	25	1263.749	2.569	0.367	0.287
7	*Random*	6	267.363	12.144	1.735	-0.071
7	*Sequential*	6	309.131	10.503	1.500	-0.056
8	*Few first*	32	2818.094	1.152	0.144	0.849
8	*Many first*	32	2009.376	1.616	0.202	0.564
8	*Random*	6	253.627	12.802	1.600	-0.054
8	*Sequential*	6	267.920	12.119	1.515	-0.049
9	*Few first*	41	1082.865	2.998	0.333	0.250
9	*Many first*	41	1515.228	2.143	0.238	0.400
9	*Random*	7	180.548	17.984	1.998	-0.062
9	*Sequential*	7	202.020	16.072	1.786	-0.055
10	*Few first*	50	876.269	3.705	0.371	0.189
10	*Many first*	50	1062.537	3.056	0.306	0.252
10	*Random*	7	231.795	14.008	1.401	-0.032
10	*Sequential*	7	196.434	16.529	1.653	-0.044

Table B.55: Tests of granularity to `unif-r4.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	64.529	5.590	2.795	-0.642
2	*Many first*	2	250.100	1.442	0.721	0.387
2	*Random*	2	64.495	5.592	2.796	-0.642
2	*Sequential*	2	248.469	1.452	0.726	0.378
3	*Few first*	5	49.305	7.315	2.438	-0.295
3	*Many first*	5	34.822	10.358	3.453	-0.355
3	*Random*	4	47.918	7.527	2.509	-0.301
3	*Sequential*	4	13.962	25.834	8.611	-0.442
4	*Few first*	8	38.290	9.420	2.355	-0.192
4	*Many first*	8	14.724	24.497	6.124	-0.279
4	*Random*	4	13.938	25.878	6.470	-0.282
4	*Sequential*	4	13.845	26.052	6.513	-0.282
5	*Few first*	13	2.479	145.488	29.098	-0.241
5	*Many first*	13	7.158	50.386	10.077	-0.225
5	*Random*	5	10.550	34.189	6.838	-0.213
5	*Sequential*	5	35.334	10.208	2.042	-0.128
6	*Few first*	18	10.939	32.974	5.496	-0.164
6	*Many first*	18	45.722	7.889	1.315	-0.048
6	*Random*	6	6.736	53.543	8.924	-0.178
6	*Sequential*	6	2.327	154.991	25.832	-0.192
7	*Few first*	25	73.095	4.935	0.705	0.070
7	*Many first*	25	17.056	21.147	3.021	-0.111
7	*Random*	6	11.900	30.310	4.330	-0.128
7	*Sequential*	6	2.314	155.862	22.266	-0.159
8	*Few first*	32	3.482	103.580	12.947	-0.132
8	*Many first*	32	9.560	37.730	4.716	-0.113
8	*Random*	6	2.331	154.725	19.341	-0.135
8	*Sequential*	6	2.329	154.858	19.357	-0.135
9	*Few first*	41	2.479	145.488	16.165	-0.117
9	*Many first*	41	0.810	445.267	49.474	-0.122
9	*Random*	7	15.680	23.003	2.556	-0.076
9	*Sequential*	7	7.167	50.323	5.591	-0.103
10	*Few first*	50	14.933	24.154	2.415	-0.065
10	*Many first*	50	0.957	376.871	37.687	-0.108
10	*Random*	7	1.384	260.596	26.060	-0.107
10	*Sequential*	7	7.166	50.330	5.033	-0.089

Table B.56: Tests of granularity to `unif-r5.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	16.658	3.073	1.537	-0.349
2	*Many first*	2	9.666	5.296	2.648	-0.622
2	*Random*	2	55.452	0.923	0.462	1.166
2	*Sequential*	2	9.669	5.295	2.647	-0.622
3	*Few first*	5	1.797	28.485	9.495	-0.447
3	*Many first*	5	36.909	1.387	0.462	0.582
3	*Random*	4	11.646	4.396	1.465	-0.159
3	*Sequential*	4	6.881	7.439	2.480	-0.298
4	*Few first*	8	14.685	3.486	0.871	0.049
4	*Many first*	8	4.008	12.771	3.193	-0.229
4	*Random*	4	6.834	7.490	1.873	-0.155
4	*Sequential*	4	6.926	7.391	1.848	-0.153
5	*Few first*	13	1.226	41.752	8.350	-0.220
5	*Many first*	13	0.890	57.515	11.503	-0.228
5	*Random*	5	15.322	3.341	0.668	0.124
5	*Sequential*	5	36.871	1.388	0.278	0.650
6	*Few first*	18	1.028	49.794	8.299	-0.176
6	*Many first*	18	0.102	501.899	83.650	-0.198
6	*Random*	6	10.348	4.947	0.825	0.043
6	*Sequential*	6	11.166	4.585	0.764	0.062
7	*Few first*	25	0.203	252.171	36.024	-0.162
7	*Many first*	25	0.437	117.138	16.734	-0.157
7	*Random*	6	18.792	2.724	0.389	0.262
7	*Sequential*	6	11.174	4.581	0.654	0.088
8	*Few first*	32	0.314	163.025	20.378	-0.136
8	*Many first*	32	0.672	76.174	9.522	-0.128
8	*Random*	6	10.849	4.719	0.590	0.099
8	*Sequential*	6	11.133	4.598	0.575	0.106
9	*Few first*	41	0.398	128.617	14.291	-0.116
9	*Many first*	41	0.250	204.761	22.751	-0.120
9	*Random*	7	2.922	17.518	1.946	-0.061
9	*Sequential*	7	5.553	9.218	1.024	-0.003
10	*Few first*	50	0.805	63.589	6.359	-0.094
10	*Many first*	50	0.878	58.301	5.830	-0.092
10	*Random*	7	10.862	4.713	0.471	0.125
10	*Sequential*	7	5.575	9.182	0.918	0.010

Table B.57: Tests of granularity to `vmpc_21.renamed-as.sat05-1923.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	36.562	4.154	2.077	-0.519
2	*Many first*	2	9.324	16.291	8.146	-0.877
2	*Random*	2	45.246	3.357	1.679	-0.404
2	*Sequential*	2	9.326	16.288	8.144	-0.877
3	*Few first*	5	8.835	17.193	5.731	-0.413
3	*Many first*	5	31.176	4.872	1.624	-0.192
3	*Random*	4	28.901	5.256	1.752	-0.215
3	*Sequential*	4	28.860	5.263	1.754	-0.215
4	*Few first*	8	2.273	66.821	16.705	-0.313
4	*Many first*	8	1.038	146.325	36.581	-0.324
4	*Random*	4	27.419	5.540	1.385	-0.093
4	*Sequential*	4	28.825	5.270	1.317	-0.080
5	*Few first*	13	1.285	118.199	23.640	-0.239
5	*Many first*	13	3.536	42.954	8.591	-0.221
5	*Random*	5	92.672	1.639	0.328	0.513
5	*Sequential*	5	31.217	4.866	0.973	0.007
6	*Few first*	18	0.391	388.462	64.744	-0.197
6	*Many first*	18	2.216	68.540	11.423	-0.182
6	*Random*	6	10.708	14.185	2.364	-0.115
6	*Sequential*	6	116.994	1.298	0.216	0.724
7	*Few first*	25	1.160	130.936	18.705	-0.158
7	*Many first*	25	1.072	141.684	20.241	-0.158
7	*Random*	6	110.269	1.377	0.197	0.680
7	*Sequential*	6	116.789	1.301	0.186	0.730
8	*Few first*	32	1.867	81.352	10.169	-0.129
8	*Many first*	32	0.362	419.582	52.448	-0.140
8	*Random*	6	5.560	27.317	3.415	-0.101
8	*Sequential*	6	116.701	1.302	0.163	0.735
9	*Few first*	41	0.133	1142.113	126.901	-0.124
9	*Many first*	41	4.183	36.310	4.034	-0.094
9	*Random*	7	0.134	1133.581	125.953	-0.124
9	*Sequential*	7	83.673	1.815	0.202	0.495
10	*Few first*	50	0.221	687.296	68.730	-0.109
10	*Many first*	50	1.637	92.782	9.278	-0.099
10	*Random*	7	75.024	2.025	0.202	0.438
10	*Sequential*	7	83.661	1.816	0.182	0.501

Table B.58: Tests of granularity to `vmpc_23.renamed-as.sat05-1927.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	271.611	1.740	0.870	0.149
2	*Many first*	2	229.119	2.063	1.031	-0.030
2	*Random*	2	444.469	1.063	0.532	0.881
2	*Sequential*	2	229.245	2.062	1.031	-0.030
3	*Few first*	5	0.931	507.622	169.207	-0.497
3	*Many first*	5	8.212	57.549	19.183	-0.474
3	*Random*	4	0.721	655.476	218.492	-0.498
3	*Sequential*	4	4.472	105.678	35.226	-0.486
4	*Few first*	8	0.624	757.372	189.343	-0.332
4	*Many first*	8	9.732	48.565	12.141	-0.306
4	*Random*	4	0.709	666.571	166.643	-0.331
4	*Sequential*	4	4.459	105.986	26.496	-0.321
5	*Few first*	13	12.624	37.439	7.488	-0.217
5	*Many first*	13	8.236	57.388	11.478	-0.228
5	*Random*	5	1.129	418.596	83.719	-0.247
5	*Sequential*	5	0.945	500.102	100.020	-0.248
6	*Few first*	18	9.360	50.496	8.416	-0.176
6	*Many first*	18	1.259	375.373	62.562	-0.197
6	*Random*	6	1.177	401.525	66.921	-0.197
6	*Sequential*	6	9.750	48.476	8.079	-0.175
7	*Few first*	25	7.045	67.082	9.583	-0.149
7	*Many first*	25	13.583	34.795	4.971	-0.133
7	*Random*	6	0.529	893.391	127.627	-0.165
7	*Sequential*	6	9.752	48.466	6.924	-0.143
8	*Few first*	32	10.858	43.529	5.441	-0.117
8	*Many first*	32	6.060	77.985	9.748	-0.128
8	*Random*	6	9.336	50.626	6.328	-0.120
8	*Sequential*	6	9.726	48.595	6.074	-0.119
9	*Few first*	41	21.425	22.059	2.451	-0.074
9	*Many first*	41	9.069	52.116	5.791	-0.103
9	*Random*	7	3.716	127.177	14.131	-0.116
9	*Sequential*	7	2.634	179.420	19.936	-0.119
10	*Few first*	50	13.163	35.906	3.591	-0.080
10	*Many first*	50	1.867	253.129	25.313	-0.107
10	*Random*	7	1.087	434.771	43.477	-0.109
10	*Sequential*	7	2.615	180.723	18.072	-0.105

Table B.59: Tests of granularity to `vmpc_25.renamed-as.sat05-1913.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	101.581	0.254	0.127	6.867
2	*Many first*	2	20.832	1.240	0.620	0.613
2	*Random*	2	19.886	1.299	0.649	0.540
2	*Sequential*	2	20.800	1.242	0.621	0.611
3	*Few first*	5	27.275	0.947	0.316	1.084
3	*Many first*	5	346.991	0.074	0.025	19.654
3	*Random*	4	162.001	0.159	0.053	8.909
3	*Sequential*	4	469.870	0.055	0.018	26.791
4	*Few first*	8	203.695	0.127	0.032	10.183
4	*Many first*	8	15.839	1.631	0.408	0.484
4	*Random*	4	77.290	0.334	0.084	3.657
4	*Sequential*	4	469.352	0.055	0.014	23.899
5	*Few first*	13	147.330	0.175	0.035	6.881
5	*Many first*	13	155.741	0.166	0.033	7.288
5	*Random*	5	33.998	0.760	0.152	1.396
5	*Sequential*	5	346.559	0.075	0.015	16.524
6	*Few first*	18	9.630	2.682	0.447	0.247
6	*Many first*	18	21.945	1.177	0.196	0.820
6	*Random*	6	44.218	0.584	0.097	1.855
6	*Sequential*	6	68.415	0.377	0.063	2.979
7	*Few first*	25	208.061	0.124	0.018	9.232
7	*Many first*	25	14.059	1.837	0.262	0.468
7	*Random*	6	65.213	0.396	0.057	2.779
7	*Sequential*	6	68.325	0.378	0.054	2.920
8	*Few first*	32	4.442	5.814	0.727	0.054
8	*Many first*	32	17.451	1.480	0.185	0.629
8	*Random*	6	181.972	0.142	0.018	7.910
8	*Sequential*	6	68.315	0.378	0.047	2.880
9	*Few first*	41	3.833	6.737	0.749	0.042
9	*Many first*	41	13.473	1.917	0.213	0.462
9	*Random*	7	60.543	0.427	0.047	2.512
9	*Sequential*	7	150.358	0.172	0.019	6.425
10	*Few first*	50	9.828	2.628	0.263	0.312
10	*Many first*	50	5.749	4.492	0.449	0.136
10	*Random*	7	256.934	0.101	0.010	10.943
10	*Sequential*	7	150.835	0.171	0.017	6.378

Table B.60: Tests of granularity to `vmpc_25.shuffled-as.sat05-1945.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	192.026	0.740	0.370	1.702
2	*Many first*	2	992.176	0.143	0.072	12.962
2	*Random*	2	1217.578	0.117	0.058	16.134
2	*Sequential*	2	994.138	0.143	0.071	12.990
3	*Few first*	5	1170.039	0.121	0.040	11.849
3	*Many first*	5	322.834	0.440	0.147	2.907
3	*Random*	4	695.821	0.204	0.068	6.844
3	*Sequential*	4	42.488	3.345	1.115	-0.052
4	*Few first*	8	591.416	0.240	0.060	5.215
4	*Many first*	8	956.179	0.149	0.037	8.637
4	*Random*	4	1957.178	0.073	0.018	18.028
4	*Sequential*	4	42.429	3.350	0.837	0.065
5	*Few first*	13	2.334	60.888	12.178	-0.229
5	*Many first*	13	41.650	3.412	0.682	0.116
5	*Random*	5	173.724	0.818	0.164	1.278
5	*Sequential*	5	322.802	0.440	0.088	2.589
6	*Few first*	18	13.355	10.642	1.774	-0.087
6	*Many first*	18	30.657	4.636	0.773	0.059
6	*Random*	6	1048.403	0.136	0.023	8.652
6	*Sequential*	6	701.427	0.203	0.034	5.723
7	*Few first*	25	17.728	8.017	1.145	-0.021
7	*Many first*	25	24.417	5.821	0.832	0.034
7	*Random*	6	129.477	1.098	0.157	0.896
7	*Sequential*	6	697.553	0.204	0.029	5.560
8	*Few first*	32	6.095	23.316	2.915	-0.094
8	*Many first*	32	35.643	3.987	0.498	0.144
8	*Random*	6	30.080	4.725	0.591	0.099
8	*Sequential*	6	698.204	0.204	0.025	5.472
9	*Few first*	41	24.766	5.739	0.638	0.071
9	*Many first*	41	20.113	7.066	0.785	0.034
9	*Random*	7	662.989	0.214	0.024	5.123
9	*Sequential*	7	580.875	0.245	0.027	4.473
10	*Few first*	50	33.097	4.294	0.429	0.148
10	*Many first*	50	15.753	9.022	0.902	0.012
10	*Random*	7	540.534	0.263	0.026	4.115
10	*Sequential*	7	579.592	0.245	0.025	4.420

Table B.61: Tests of granularity to `vmpc_26.renamed-as.sat05-1914.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	293.254	1.027	0.513	0.948
2	*Many first*	2	189.681	1.588	0.794	0.260
2	*Random*	2	596.066	0.505	0.253	2.959
2	*Sequential*	2	189.413	1.590	0.795	0.258
3	*Few first*	5	315.062	0.956	0.319	1.069
3	*Many first*	5	229.229	1.314	0.438	0.642
3	*Random*	4	730.186	0.412	0.137	3.137
3	*Sequential*	4	62.168	4.844	1.615	-0.190
4	*Few first*	8	42.624	7.065	1.766	-0.145
4	*Many first*	8	106.090	2.838	0.710	0.136
4	*Random*	4	62.141	4.846	1.211	-0.058
4	*Sequential*	4	62.153	4.845	1.211	-0.058
5	*Few first*	13	15.546	19.370	3.874	-0.185
5	*Many first*	13	26.959	11.170	2.234	-0.138
5	*Random*	5	269.876	1.116	0.223	0.870
5	*Sequential*	5	228.888	1.316	0.263	0.700
6	*Few first*	18	23.979	12.557	2.093	-0.104
6	*Many first*	18	340.459	0.884	0.147	1.157
6	*Random*	6	491.294	0.613	0.102	1.758
6	*Sequential*	6	140.425	2.144	0.357	0.360
7	*Few first*	25	363.340	0.829	0.118	1.241
7	*Many first*	25	86.304	3.489	0.498	0.168
7	*Random*	6	5.687	52.946	7.564	-0.145
7	*Sequential*	6	140.616	2.141	0.306	0.378
8	*Few first*	32	252.193	1.194	0.149	0.814
8	*Many first*	32	286.870	1.050	0.131	0.946
8	*Random*	6	640.664	0.470	0.059	2.289
8	*Sequential*	6	140.393	2.145	0.268	0.390
9	*Few first*	41	124.191	2.425	0.269	0.339
9	*Many first*	41	4.476	67.270	7.474	-0.108
9	*Random*	7	356.239	0.845	0.094	1.206
9	*Sequential*	7	110.925	2.715	0.302	0.289
10	*Few first*	50	157.608	1.911	0.191	0.470
10	*Many first*	50	57.883	5.202	0.520	0.102
10	*Random*	7	192.221	1.567	0.157	0.598
10	*Sequential*	7	110.966	2.714	0.271	0.298

Table B.62: Tests of granularity to `vmpc_26.shuffled-as.sat05-1946.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	2770.463	0.323	0.162	5.184
2	*Many first*	2	439.001	2.041	1.021	-0.020
2	*Random*	2	439.579	2.038	1.019	-0.019
2	*Sequential*	2	439.102	2.041	1.020	-0.020
3	*Few first*	5	103.887	8.625	2.875	-0.326
3	*Many first*	5	833.479	1.075	0.358	0.895
3	*Random*	4	1032.479	0.868	0.289	1.228
3	*Sequential*	4	975.831	0.918	0.306	1.133
4	*Few first*	8	1111.319	0.806	0.202	1.320
4	*Many first*	8	1699.159	0.527	0.132	2.195
4	*Random*	4	532.932	1.681	0.420	0.460
4	*Sequential*	4	975.911	0.918	0.230	1.119
5	*Few first*	13	1047.094	0.856	0.171	1.211
5	*Many first*	13	330.491	2.711	0.542	0.211
5	*Random*	5	1886.283	0.475	0.095	2.381
5	*Sequential*	5	834.746	1.073	0.215	0.914
6	*Few first*	18	17.390	51.528	8.588	-0.177
6	*Many first*	18	225.856	3.967	0.661	0.102
6	*Random*	6	145.173	6.172	1.029	-0.006
6	*Sequential*	6	897.032	0.999	0.166	1.001
7	*Few first*	25	19.674	45.546	6.507	-0.141
7	*Many first*	25	72.453	12.368	1.767	-0.072
7	*Random*	6	865.812	1.035	0.148	0.961
7	*Sequential*	6	897.720	0.998	0.143	1.002
8	*Few first*	32	45.594	19.654	2.457	-0.085
8	*Many first*	32	53.547	16.734	2.092	-0.075
8	*Random*	6	540.611	1.658	0.207	0.547
8	*Sequential*	6	897.016	0.999	0.125	1.001
9	*Few first*	41	9.311	96.243	10.694	-0.113
9	*Many first*	41	44.889	19.962	2.218	-0.069
9	*Random*	7	798.544	1.122	0.125	0.878
9	*Sequential*	7	905.395	0.990	0.110	1.012
10	*Few first*	50	16.387	54.682	5.468	-0.091
10	*Many first*	50	18.227	49.162	4.916	-0.089
10	*Random*	7	1106.976	0.809	0.081	1.262
10	*Sequential*	7	904.271	0.991	0.099	1.010

Table B.63: Tests of granularity to `vmpc_27.renamed-as.sat05-1915.cnf`

#W	Mode	#V	CPU Time	Spd.	Eff.	S. F.
2	*Few first*	2	9.042	50.151	25.075	-0.960
2	*Many first*	2	1.198	378.480	189.240	-0.995
2	*Random*	2	1.196	379.113	189.557	-0.995
2	*Sequential*	2	1.202	377.221	188.610	-0.995
3	*Few first*	5	0.980	462.674	154.225	-0.497
3	*Many first*	5	17.442	25.997	8.666	-0.442
3	*Random*	4	4.906	92.420	30.807	-0.484
3	*Sequential*	4	2.358	192.288	64.096	-0.492
4	*Few first*	8	14.558	31.147	7.787	-0.291
4	*Many first*	8	7.534	60.182	15.046	-0.311
4	*Random*	4	5.253	86.315	21.579	-0.318
4	*Sequential*	4	3.866	117.282	29.321	-0.322
5	*Few first*	13	25.387	17.861	3.572	-0.180
5	*Many first*	13	5.822	77.879	15.576	-0.234
5	*Random*	5	13.808	32.839	6.568	-0.212
5	*Sequential*	5	5.246	86.430	17.286	-0.236
6	*Few first*	18	1.061	427.352	71.225	-0.197
6	*Many first*	18	27.995	16.197	2.700	-0.126
6	*Random*	6	17.299	26.212	4.369	-0.154
6	*Sequential*	6	0.787	576.142	96.024	-0.198
7	*Few first*	25	15.357	29.527	4.218	-0.127
7	*Many first*	25	23.914	18.961	2.709	-0.105
7	*Random*	6	1.569	288.985	41.284	-0.163
7	*Sequential*	6	3.716	122.017	17.431	-0.157
8	*Few first*	32	21.478	21.112	2.639	-0.089
8	*Many first*	32	5.496	82.499	10.312	-0.129
8	*Random*	6	4.771	95.035	11.879	-0.131
8	*Sequential*	6	3.718	121.951	15.244	-0.133
9	*Few first*	41	35.152	12.899	1.433	-0.038
9	*Many first*	41	6.266	72.361	8.040	-0.109
9	*Random*	7	1.131	400.902	44.545	-0.122
9	*Sequential*	7	1.553	291.963	32.440	-0.121
10	*Few first*	50	3.671	123.512	12.351	-0.102
10	*Many first*	50	42.710	10.617	1.062	-0.006
10	*Random*	7	1.732	261.788	26.179	-0.107
10	*Sequential*	7	4.572	99.172	9.917	-0.100

Table B.64: Tests of granularity to `vmpc_27.shuffled-as.sat05-1947.cnf`