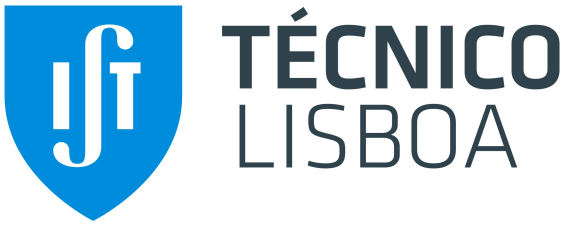

SAT Solver Paralelo

Ricardo de Sousa Marques

Introdução à Investigação e ao Projecto em

Engenharia Electrotécnica e de Computadores

Orientador: Paulo Ferreira Godinho Flores
Co-Orientador: Luís Miguel d'Ávila Silveira

Janeiro 2013

Resumo

Muitos problemas em diversas áreas podem ser convertidas num problema SAT, ou numa sequência de problemas SAT, de tal forma que a solução para este dá-nos a solução do problema original. Apesar do enorme progresso alcançado na última década no desenvolvimento de SAT solvers, há uma necessidade para uma maior eficiência algorítmica para resolver problemas mais largos e mais difíceis.

A ampla disponibilidade de ambientes de computação paralela, nomeadamente *multi-core*, providencia-nos uma oportunidade para esses progressos.

Neste relatório são apresentados os resultados dessa abordagem, sendo apresentados dois solvers que utilizam técnicas distintas para a resolução do problema.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 O problema de Satisfação Booleana (SAT)

A Satisfação Booleana é um dos problemas mais importantes estudados em ciências de computadores. Dada uma fórmula booleana com operações lógicas, nomeadamente ***and***, ***or*** e ***not***, o processo de determinar se existe uma atribuição de variáveis que faça com que a fórmula seja avaliada como verdadeira é chamado de SAT.

Este problema foi considerado o primeiro do tipo NP-Completo. Na electrónica, há várias aplicações que podem ser reduzidas a problemas SAT, entre elas aplicações de *Automatic test pattern generation* (ATPG), verificação de equivalência combinatorial de circuitos, verificação de multiprocessadores, verificação de modelos, entre outros.

1.2 Objectivos

O foco desta dissertação está na investigação e desenvolvimento de técnicas para acelerar a resolução deste tipo de problemas. A utilização de ambientes de computação paralela e distribuída é uma abordagem promissora para o cumprimento deste objectivo. Utilizando plataformas como *multi-core* pretende-se explorar técnicas, quer ao nível algorítmico, quer ao nível de implementação, que permitam ter resultados que consigam competir com os SAT solvers mais eficientes actualmente.

1.2.1 Organização do Relatório

A organização deste relatorio é a seguinte. No capítulo 2 é efectuada uma descrição geral sobre os conceitos utilizados em SAT. Será dada uma maior incidência sobre a estrutura Davis-Putnam Logemann-Loveland (DPLL) [1], pois é a base da maioria dos algoritmos de SAT. Será também efectuada uma breve descrição dos algoritmos de SAT mais conceituados, bem como as técnicas utilizadas por estes.

No capítulo 3 será dado mais ênfase ao tema da dissertação, SAT paralelo, onde serão descritos os algoritmos que utilizam arquitecturas paralelas. Será também efectuada uma breve comparação entre as diferentes técnicas utilizadas para obter aceleração na resolução do problema.

No capítulo 4 será então descrito o trabalho já realizado até ao momento, nomeadamente uma solução paralela de portfolio. Será também descrito o trabalho ainda

que está a ser desenvolvido, um algorimo cooperativo que utiliza clusters de dados.

Capítulo 2

Algoritmos SAT

2.1 Conceitos básicos de SAT

Nesta secção efectua-se uma breve descrição dos diversos termos utilizados para descrever um problema de SAT.

- A ***fórmula CNF* (Conjunctive Normal Form)** é uma conjunção de cláusulas, constituindo a descrição generalizada de um problema SAT. Devido a uma questão de eficiência as fórmulas proposicionais usadas pelos algoritmos de satisfação encontram-se nesta forma.
- Uma ***cláusula*** w é uma disjunção de literais.
- Um ***literal*** l é a ocorrência de uma variável x_i, na sua forma positiva (x_i) ou negativa ($\neg x_i$), numa cláusula w.

Abaixo temos o exemplo de um problema de SAT demonstrado na fórmula CNF:

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_4) \tag{2.1}$$

Rapidamente podemos então concluir que dada uma fórmula CNF $\varphi = (\omega_1 \wedge ... \wedge \omega_n)$ só é satisfeita se for efectuado um conjunto de atribuições ρ tal que satisfaça todas as cláusulas. Uma cláusula é **satisfeita** se pelo menos um dos seus literais for satisfeito, sendo que um literal é satisfeito se, caso seja um literal positivo (x_i) possuir $\rho(x_i) = 1$ e, caso seja um literal negativo ($\neg x_i$), possuir $\rho(x_i) = 0$. Se não existir nenhum conjunto de atribuições ρ que satisfaça esta fórmula diz-se então que o problema é **insatisfeito**.

2.2 Algoritmo Davis Putnam (DP)

Por volta de 1960 Davis e Putnam propuseram o primeiro algoritmo de SAT. Este é baseia-se em técnicas de resolução e aplica as regras da cláusula unitária e do literal puro, enunciadas abaixo:

- ***Resolução*** - Dadas duas cláusulas $\omega_1 = (z \vee x_1 \vee ... \vee x_n)$ e $\omega_2 = (\neg z \vee y_1 \vee ... \vee y_n)$ podemos fazer uma disjunção das duas cláusulas e retirar z, obtendo-se então $\omega_3 = (x_1 \vee ... \vee x_n \vee y_1 \vee ... \vee y_n)$.

- ***Literal Puro*** - Dada uma formula φ, um literal $x_i \in \varphi$ é um literal puro se o seu complementar ($\neg x_i$) não ocorrer em φ.

- ***Regra do Literal Puro*** - Dada uma formula φ que possua literais puros, pode ser efectuada uma atribuição ρ tal que todos esses literais sejam satisfeitos, satisfazendo desta forma as respectivas cláusulas, podendo estas ser removidas da fórmula inicial sem a perda de qualquer informação relativa ao problema.

- ***Regra da Cláusula Unitária*** - Dada uma cláusula ω_i em que todos os literais, excepto um possuem a atribuição 0, implica que o restante literal, denominado ***literal unitário***, tenha a atribuição 1 para que a ω_i seja satisfeita. Esta cláusula é denominada de ***cláusula unitária***.

- ***Propagação Unitária*** - Consiste no processo analisar a fórmula após ter sida efectuada a atribuição de uma variável e, caso necessário, de atribuir 1 a todos os literais unitários. Este processo é também denominado por ***Boolean Constraint Propagation (BCP)*** .

Abaixo esta demonstrado o exemplo de realização de uma propagação unitária:

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_3 \vee x_4) \tag{2.2}$$

Se forem efectuadas as atribuições $x_1 = 0$ e $x_2 = 0$, para a primeira cláusula ser satisfeita terá de ser efectuada a atribuição $x_3 = 1$, e consequentemente para a segunda cláusula ser satisfeita terá de ser efectuada a atribuição $x_4 = 1$.

Esta técnica de propagação requer, porém, o crescimento de memória de forma exponencial, sendo que em 1962 foi proposto outro algoritmo, por Davis, Longemann e Loveland (DLL, ou DPLL), que é a base dos algoritmos de SAT completos actuais. Os mecanismos utilizados por este vão ser descritos com detalhe nas próximas secções.

2.3 Algoritmo Davis Putnam Longemann Loveland(DPLL)

2.3.1 Funcionamento Geral

O pseudo-código deste algoritmo está apresentado na página seguinte.

Dada então uma fórmula φ, o algoritmo irá efectuar atribuições às diversas variáveis com a função de encontrar uma solução. Como podemos observar, a cada passo da procura é escolhida uma variável a atribuir através da função *decide*(). Se todos os literais estiverem atribuídos e nenhuma cláusula estiver instatisfeita estamos perante um caso em que a fórmula é satisfeita. Caso contrário, depois então de ser atribuída uma variável é efectuada uma propagação, onde vão ser atribuídos todos literais unitários e, caso se verifique um conflito, é determinada a cláusula de conflito pela função $conflict\_analysis()$. Esta calcula também o nível de decisão para qual as atribuições efectuadas vão ser desfeitas. Estas são efectuadas com a função *undoDecisions*().

A grande maioria dos algoritmos de SAT é baseada então nos passos de ***propagação***, ***decisão*** e ***análise/backtrack***.

Algorithm 1 DPLL algorithm

```
1: limit = c;
2: while TRUE do
3:     while #conflicts < limit do
4:         lit = decide();
5:         if !lit then
6:             return SAT;
7:         end if
8:         if !BCP(lit) then
9:             cl = conflict_analysis();
10:            if !cl then
11:                return UNSAT;
12:            end if
13:            #conflicts++;
14:        end if
15:    end while
16:    undoDecisions();
17:    increase(limit);
18: end while
```

2.3.2 Heurísticas de Decisão

As heurísticas de decisão têm um papel-chave no desempenho de um SAT solver, visto que determinam a ordem das regiões em que o espaço de procura é explorado. A escolha de uma boa sequência de variáveis a atribuir pode levar à solução do problema muito rapidamente, assim como uma sequência mal-estruturada em relação ao problema pode requerer que a grande maioria do espaço de procura seja explorado até que seja determinada a solução do problema, o que pode levar a uma grande esforço computacional. Os SAT solvers mais antigos implementam mecanismos **estáticos** de escolha de variáveis, onde a selecção é efectuada tendo apenas em conta a estrutura do problema. Os SAT solvers modernos usam modelos **dinâmicos** para a escolha destas variáveis, onde a selecção é efectuada tendo em conta não só a estrutura do problema mas também o seu estado de resolução. A heurística de decisão mais comum e mais eficiente utilizada nos SAT solvers modernos denomina-se por ***Variable State Independent Decaying Sum*** **(VSIDS)**, introduzida pelo algoritmo CHAFF [2].

Este método consiste numa ordenação das variáveis tendo em conta a sua actividade. Inicialmente, é efectuada uma ordenação das variáveis consoante o número de ocorrências de cada literal no problema inicial. Sempre que uma cláusula é aprendida, a VSIDS incrementa o valor de cada variável que a constitui por uma constante, sendo divididadas periodicamente por uma constante de decaimento à medida que o algoritmo progride. Desta forma, os valores atribuídos a cada variável pela VSIDS correspondem ao peso da presença destas nas cláusulas aprendidas mais recentes. Esta heurística diz-se **independente do estado** pois não é mantida qualquer relação com a atribuição das variáveis e o cálculo destes valores.

Esta heurística revela-se bastante eficaz em problemas de larga dimensão, visto que irá sempre escolher a maior variável livre que der mais opções de ramificação.

2.3.3 Propagação

Visto que qualquer algoritmo de SAT requer um acesso e manipulação de grandes quantidades de informação, o modo como as estruturas de dados são definidas é fundamental para obter uma boa performance. A estrutura de dados de algoritmos de SAT que mais melhorias trouxe foi a introdução de ***watched literals***, proposta pelo algoritmo CHAFF [2].

Durante a propagação, apenas cláusulas unitárias podem ser utilizadas para implicar novas atribuições de variáveis. Sendo assim, para cada cláusula, são designados dois literais a ser observados. Quando um literal observado toma a o valor de *false*, a cláusula correspondente é analisada. Caso exista mais um literal livre não observado o ponteiro é movido para este. Caso não exista mais nenhum literal não observado, estamos perante uma cláusula unitária, sendo que o literal unitário corresponde ao outro literal observado da cláusula. Na figura 2.1 está demonstrado um exemplo de realização deste tipo de propagação (retirado de [2]).

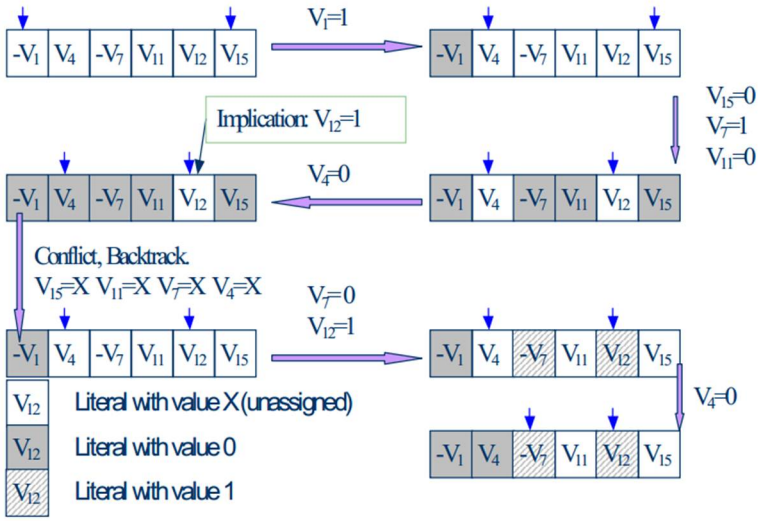

Figura 2.1: Método de propagação com *watched literals*

Esta técnica providencia um método bastante eficiente para determinar se uma cláusula se torna unitária ou não, bem como ao determinar o literal unitário. Uma vantagem bastante interessante desta técnica é o facto de não terem de ser alterados os literais observados quando um *backtrack* é efectuado.

2.3.4 *Backtrack*

Quando é identificado um conflito, é necessário desfazer um conjunto de atribuições para poder retomar a procura. Esse processo denomina-se de retrocesso, ou *backtrack*. Existem essencialmente dois tipos de *backtrack*:

- ***Backtrack* cronológico** - Este processo consiste simplesmente no processo de desfazer a decisão anterior, e consequentes implicações, sendo então retomada a procura.

- ***Backtrack* não-cronológico** - Este processo, introduzido pelo algoritmo GRASP [3], pode desfazer varias decisões. Quando um conflito é identificado, é efectuado um **diagnóstico**, que gera uma cláusula que identifica a origem do conflito. Esta cláusula é aprendida e adicionada ao problema inicial. É procedido então um *backtrack*, que geralmente vai desfazer uma série de decisões até se verificar que não está nenhuma cláusula insatisfeita. A utilização deste processo, ao adicionar informação ao problema original vai permitir também que futuros conflitos sejam detectados mais rápido, sendo a performance do solver melhorada significativamente. Este método tem, porém, a desvantagem de atrasar o processo de propagação, visto ser necessária a análise de mais cláusulas. Um método bastante utilizado para atenuar este atraso é a periódica redução do conjunto de cláusulas aprendidas.

Dados os dois processos de retrocesso, facilmente se conclui que o *backtrack* não-cronológico é o mais eficiente, sendo este o método mais utilizado em SAT solvers modernos.

2.3.5 Restarts

Os algoritmos de SAT podem apresentar uma grande dispersão no que toca ao tempo necessário para resolver uma instância em particular. Podemos observar grandes diferenças de performance se forem utilizadas diferentes heurísticas de decisão. De facto, um estudo efectuado em [4] observa que algoritmos baseados em mecanismos de *backtrack* é caracterizado por realizar procuras demasiado extensas sobre determinadas regiões do espaço de procura.

A estratégia de ***restarts*** é um método para minimizar este problema. Esta estratégia consiste na definição de um limite do número de *backtracks* efectuados. O algoritmo, ao ultrapassar esse limite, desfaz todas as atribuições efectuadas até ao momento e recomeça a procura. Isto implica a adição de um factor de aleatoriedade na heurística de decisão, caso contrário corre-se o risco de ser efectuado o mesmo conjunto de atribuições a cada iteração do algoritmo. Para preservar a plenitude do algoritmo, este limite deverá ser incrementado a cada *restart* efectuado, possibilitando que todo o espaço de procura seja explorado.

Restarts e heurísticas de decisão baseadas em actividade são actividades complementares num SAT solver, visto que as heurísticas permitem-nos mudar a região do espaço de procura a explorar, sendo que os *restarts* permitem que a procura se foque nessa região.

Capítulo 3

SAT Paralelo

A generalização de processadores *multi-core* possibilitou a qualquer utilizador o acesso a ambientes de computação paralela, tendo levado à criação de diversas oportunidades de optimização nas diversas áreas.

Neste contexto, muitos SAT solvers paralelos foram criados, sendo que as competições de SAT inclusivamente apresentam uma vertente de SAT paralelo. O principal objectivo destes solvers é resolver o mais rápido possível problemas. Para tal, diversas métricas são utilizadas para avaliar a performance dos algoritmos, tais como o *speedup* e a eficiência. Os obstáculos que se opõem a estes factores são geralmente os que são encontrados em outros ambientes de computação paralela, tais como balanceamento de trabalho, sincronização entre *threads*, entre outros.

Uma das características que torna promissor o desenvolvimento de algoritmos paralelos de SAT é o facto de ser possível obter *speedups* bastante superiores ao número de recursos utilizados, por se tratar de um agoritmo de procura. Estes *speedups* são possíveis de obter se um dos elementos explorar a região certa do espaço de procura. Desta forma podemos obter aceleração em vastos conjuntos de problemas, demonstrado que combinando a robustez de um algoritmo com a pesquisa extensa em diversos espaços de procura é um uma abordagem ao problema de SAT com grande potencial de resultados.

3.1 Técnicas Utilizadas

A utilização de ambientes de computação paralela é uma abordagem promissora para obter *speedup* na procura de uma solução, comparativamente com SAT solvers sequenciais. Além do mais, SAT solvers paralelos terão a possibilidade de resolver instâncias maiores e mais difíceis, tais como problemas industrais, dos quais uma abordagem sequencial teria muito mais dificuldades em encontrar uma solução num tempo aceitável. Os SAT solvers paralelos podem ser designados em duas categorias distintas:

- **Cooperativos** - Neste caso, o espaço de procura é dividido e cada unidade computacional (seja ela um *core*, um processador ou uma máquina) vai procurar por uma solução no seu sub-espaço de procura. Verifica-se frequentemente que é díficil realizar balanceamento de trabalho entre as várias unidades, sendo para

tal utilizados várias soluções, tais como *pool-threading* e sistemas de *master-slave*.

- **Competitivos** - Nesta solução, cada unidade computacional tenta resolvar a mesma fórmula de SAT, porém utilizando diferentes mecanismos de procura. Isto é efectuado designando diferentes algoritmos a cada unidade de processamento, ou a utilização do mesmo algoritmo com diferentes parâmetros de procura. Estas soluções são também designadas de **portfolios**.

Em ambas as categorias, as diversas unidades podem trabalhar colaborativamente, partilhando informação relevante, tal como cláusulas aprendidas, acelerando desta forma o processo de procura. Isto requer comunicação entre as diferentes unidades e pode levar a algum *overhead*. Podemos desta forma concluir o processo de decisão de que cláusulas partilhar, bem como o processo de partilha tem um impacto bastante significativo na eficiência do solver.

3.2 Algoritmos Existentes

Nesta secção são descritos alguns SAT solvers que utilizam uma abordagem paralela que têm sido bastante sucedidos na última década:

3.2.1 PMSat

Este é um solver baseado no MiniSAT [5], que é um solver sequencial que implementa a maioria das técnicas utilizadas em SAT solvers modernos, sendo também o solver no qual são baseadas as implementações dos dois algoritmos desenvolvidos nesta tese.

O PMSat [6] corre num *cluster* de computadores (*grid*) utilizando *Message Passing Interface (MPI)* para comunicação, sendo baseado numa abordagem *master-slave* com um número fixo de *slaves*. O espaço de procura é dividido pelo *master* utilizando diversas heurísticas de partição. A figura 3.1 demonstra a estrutura deste solver.

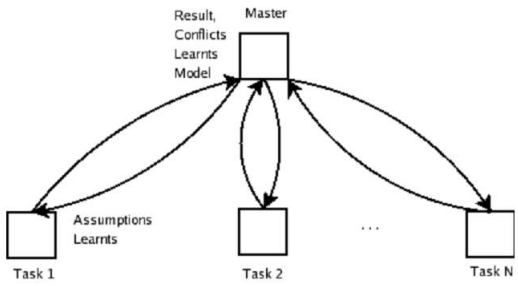

Figura 3.1: Estrutura de funcionamento do PMSat

Cada *slave* procura por uma solução num sub-conjunto de espaço e partilha as respectivas cláusulas aprendidas quando reporta a solução obtida ao *master*. O balanceamento de trabalho é efectuado implicitamente atraves da atribuição de muitas tarefas aos *slaves*.

3.2.2 C-SAT

Esta algoritmo [7] é também uma paralelização do MiniSAT que utiliza MPI com uma arquitectura *master-slave*. Nesta arquitectura, o *master* é responsável pela partilha de cláusula e partição dinâmica do espaço. Os *slaves*, porém, realizam a procura no espaço através de diferentes heurísticas e algumas configurações aleatórias. Podemos então concluir que este solver combina a partição do espaço de procura com a utilização de algoritmos de portfolio.

3.2.3 MiraXT / PaMiraXT

O PaMiraXT [8] é outro SAT solver que segue o modelo de *master-slave* e pode correr em qualquer *cluster* de computadores. Cada *slave* é baseado no solver MiraXT [9], que é por sua vez um portfolio com partilha de cláusulas. Na figura 3.2 esta descrita a arquitectura de funcionamento do PaMiraXT

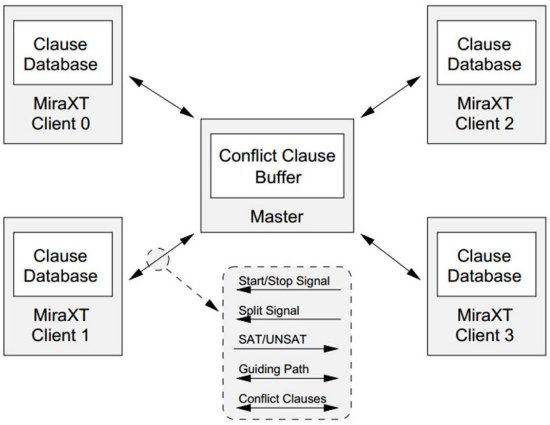

Figura 3.2: Estrutura de funcionamento do PaMiraXT

3.2.4 ySAT

O ySAT [10] é um solver que utiliza uma arquitectura de memória partilhada que sincroniza uma lista de tarefas disponíveis para minizar o número de *threads* inactivas. Neste solver a fórmula, bem como a fila de tarefas são acessíveis de forma global, porém as cláusulas aprendidas são locais para cada *thread*. Devido ao tipo de sincronização deste solver, a eficiência do solver degrada-se à medida que o número *cores* é aumentado.

3.2.5 ManySAT

O ManySAT [11] é um algoritmo de porfolio que utiliza uma arquitectura *multi-thread* que ganhou a competição paralela da SAT Race de 2008 [12]. Desde então, algoritmos portfolio têm aumentado a popularidade. Neste solver, que foi implementado sobre o algoritmo série MiniSAT, são lançadas diversas *threads* com diferentes parâmetros de *restarts*, heurísticas de decisão e de aprendizagem. Adicionalmente

é efectuada a partilha de cláusulas que tenham um tamanho menor que um determinado limite, para acelerar a performance do algoritmo. Na figura 3.3 está demonstrado um exemplo de lançamento do ManySAT com 4 *threads*:

Strategies	**Core 0**	**Core 1**	**Core 2**	**Core 3**
Restart	Geometric $x_1 = 100$ $x_i = 1.5 \times x_{i-1}$	Dynamic (Fast) $x_1 = 100, x_2 = 100$ $x_i = f(y_{i-1}, y_i), i > 2$ if $y_{i-1} < y_i$ $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \lvert cos(1 - \frac{y_{i-1}}{y_i}) \rvert$ else $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \lvert cos(1 - \frac{y_i}{y_{i-1}}) \rvert$ $\alpha = 1200$	Arithmetic $x_1 = 16000$ $x_i = x_{i-1} + 16000$	Luby 512
Heuristic	VSIDS (3% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)
Polarity	if $\#occ(l) > \#occ(\neg l)$ $l = true$ else $l = false$	Progress saving	false	Progress saving
Learning	CDCL	CDCL	CDCL	CDCL
Cl. sharing	size ≤ 8	size ≤ 8	size ≤ 8	size ≤ 8

Figura 3.3: Configurações para diferentes instâncias do ManySAT

3.2.6 SAT4J

O SAT4J é um solver hibrido que mistura a procura competitiva e cooperativa, porém sem partilha de cláusulas. Este solver, implementado em Java, comça com uma abordagem portfolio, onde uma heurística baseada na VSIDS determina as variáveis com a maior actividade. O solver então procede a uma partição do espaço de procura baseado nestas variáveis e, seguido de um determinado número de iterações, este retorna à abordagem porfolio.

3.3 Conclusões

SAT solvers cooperativos utilizam a divisão do espaço de partilha para dividir a fórmula em sub-problemas mais simples. Esta abordagem tem sido uma das mais utilizadas na implementação de algoritmos paralelos de SAT.

Porém, nos últimos anos tem havido uma mudança de tendências, tendo incrementado o número de abordagens competitivas, nomeadamente algoritmos portfolio, que têm demonstrado alcançar performances bastante interessantes, como provado em [7] e [11].

No âmbito desta tese, foram exploradas as duas abordagens, tendo resultado em dois algoritmos. A implementação destes será explicada com mais detalhe nas próximas secções.

Capítulo 4

Projectos desenvolvidos/em desenvolvimento

Neste capítulo é descrita com mais detalhe o trabalho já efectuado, bem como o trabalho que está planeado ser desenvolvido durante a realização da dissertação. Este trabalho focou-se em duas abordagens diferentes, no qual resultadram dois SAT solvers diferentes, a ser descritos com mais detalhe nas seguintes secções

- **PMCSAT** - Consiste numa solução *multi-core* SAT de portfolio, baseada no algoritmo miniSat [5], v2.2.0. Este utiliza diversas *threads* que exploram o espaço de procura de forma independente, cada um com o seu modelo de procura consoante as suas configurações. Neste modelo as diversas *threads* trabalham cooperativamente, partilhando as cláusulas aprendidas mais relevantes, acelerando desta forma o processo de procura. Este algoritmo já foi implementada, estando devidamente documentado, possuindo resultados para diversas instâncias e comprações com outros solvers paralelos. Foi também recentemente submetido um artigo com este algoritmoque foi aceite na conferência Florida Artificial Intelligence Research Society (FLAIRS-26) [13].
- **CLUSTERSAT** - Este algoritmo, também incidindo sobre uma arquitectura *multi-core*, possui uma metodologia bastante diferente do anterior, tratando-se de um algoritmo cooperativo, que utiliza clusters de dados para procurar uma solução. Estes clusters são obtidos por mecanismos já devidamente implementados, nomeadamente os algoritmos Metis e hMetis [14] e [15]. O algoritmo consiste num sistema de *master-slave*, onde o *master* gere os diversos clusters de cláusulas (*slaves*). Este projecto está de momento em desenvolvimento, possuindo já alguns resultados preliminiares mas o seu completo desenvolvimento será o trabalho fulcral a ser implementado durante o 2º semestre.

4.1 PMCSAT

Nesta secção são descritas com mais detalhe as diferentes estratégias utilizadas na implementação deste algoritmo. Utilizando um ambiente *pthread* são lançadas 8 *threads*, cada uma com configurações distintas, descritas nas próximas secções.

4.1.1 Heurísticas de Decisão

As heurísticas de decisão, como foram descritas anteriormente, são o mecanismo utilizado em SAT solvers para designar a próxima variável no problema à qual sera atribuído um valor, quando o processo de propagação é completo. Esta tem um papel fundamental no desempenho de qualquer solver, pois é o que vai determinar o local do espaço de procura que vai ser explorado. Num ambiente paralelo, ao designarmos diferentes espaços a diferentes unidades de procura, com um regime cooperativo o processo de cálculo da solução é facilmente acelerado. As probabilidades de uma das unidades determinar a solução também aumentam significativamente, visto que podem ser designados espaços de procura totalmente diferentes. O PMCSAT explora essa vertente, procurando maximizar o número de locais no espaço de procura a serem trabalhados. Na tabela 4.1 estão exemplificadas mais detalhadamente as abordagens utilizadas para as diferentes *threads*.

Thread #	Esquema de atribuição de prioridades
0	Todas as variáveis têm a mesma prioridade, logo esta thread utiliza a heurística VSIDS original.
1	A primeira metade de variáveis lidas do ficheiro possui prioridade sobre a segunda metade.
2	A segunda metade de variáveis lidas do ficheiro possui prioridade sobre a primeira metade.
3	A prioridade é sequencialmente diminuida à medida que as variáveis são lidas do ficheiro.
4	A prioridade é sequencialmente aumentada à medida que as variáveis são lidas do ficheiro.
5	A prioridade é designada aleatoreamente a todas as variáveis
6	A prioridade é sequencialmente diminuida à medida que as variáveis são lidas do ficheiro, porém com uma componente aleatória que a pode aumentar até 5x.
7	A prioridade é sequencialmente diminuida à medida que as variáveis são lidas do ficheiro, porém com uma componente aleatória que a pode aumentar até 10x.

Tabela 4.1: Esquema de atribuição de prioridades nas *threads* no PMCSAT

Como podemos observar, foram utilizadas técnicas bastantes distintas para explorar o espaço de procura. Em cada *thread* é inicialmente atribuída uma prioridade a cada variável, sendo esta mantida durante toda a execução do programa. Variáveis com prioridade superior serão sempre atribuídas antes de variáveis com prioridade inferior, sendo que o critério de decisão para variáveis com a mesma prioridade é a heurística VSIDS.

Optou-se por utilizar duas *threads* que exploram dois espaços opostos (1 e 2), nomeadamente a primeira e segunda metade das variáveis descritas no ficheiro CNF. Desta forma é possivel aproveitar o facto de que as variáveis que aparecem nestes ficheiros geralmente seguem uma ordem que não é aleatória, estando relacionada com a estrutura do problema. Desta forma podemos garantir que a informação gerada

por estas duas *threads* se vai complementar.

Os outros esquemas de atribuição são bastante diversos, sendo que os esquemas 5, 6, 7 têm inclusive uma componente aleatória associada à atribuição de prioridades. A grande utilidade da inclusão de uma componente aleatória é o facto de poder abranger grandes zonas do espaço de procura. A informação sobre estas, ao ser partilhada vai acelerar o processo de resolução de unidades que possuam uma visão mais global do problema (*threads* 0 a 2).

Na figura 4.1 está exemplificado o modo de funcionamento das *threads* 1, 2 e 3.

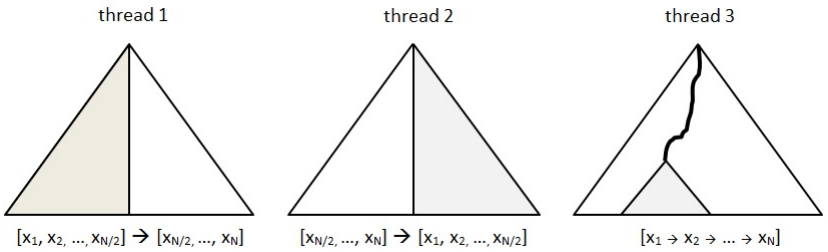

Figura 4.1: Esquema de exploração de diferentes espaços de procura

Como podemos observar na figura, as diferentes *threads* têm atribuídos diferentes espaços de procura, aumentando desta forma a probabilidade de ser encontrada uma solução para o problema.

Utilizando abordagens bastante distintas no critério de atribuição de variáveis a área de pesquisa no espaço de procura é maximizada. Os resultados podem ser ainda mais acelerados utilizando uma política de cooperação entre as diferentes *threads*, através da partilha de cláusulas aprendidas como se descreve na secção seguinte.

4.1.2 Partilha de Cláusulas

As cláusulas aprendidas, durante o processo de resolução de um problema de SAT são uma componente fundamental para o bom desempenho deste. Em SAT paralelo é indispensável uma unidade de trabalho partilhar a informação aprendida de modo a prevenir que outras unidades cometam os mesmos passos que levam ao respectivo conflito. Como tal, é implementado um esquema de partilha de cláusulas no PMC-SAT. O tamanho das cláusulas a partilhar é limitado, evitando assim o *overhead* da partilha de largas cláusulas, que podem conter informação pouco relevante sobre o problema. Em [11], os autores demonstram que a melhor performance é obtida limitando o número de literais de uma cláusula aprendida a 8.

Para minimizar a comunicação ao introduzir este mecanismo no solver, bem como o impacto desta na performance, são utilizadas estruturas de dados armazenadas em memória partilhada por todas as *threads*. Estas foram projectadas de forma a não ser necessária a utilização de *locks* para ler e escrever na memória. A figura 4.2 descreve o mecanismo de partilha de cláusulas.

Como a figura indica, a *thread* que vai partilhar a cláusula aprendida (*source thread*) possui uma série de filas, cada uma destinada a uma *thread* (*target thread*), onde vão ser inseridas as cláusulas. Dado que cada *thread* vai partilhar as cláusulas que aprende com todas as outras, podemos considerar que cada unidade é uma *source*

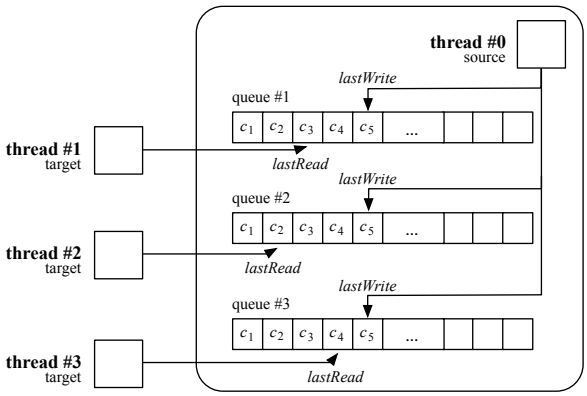

Figura 4.2: Estrutura de dados de partilha de cláusulas

thread que tem como *target thread* todas as outras. Essas estruturas são geridas com os seguintes ponteiros:

- ***lastWrite*** - Ponteiro que indica qual a última cláusula inserida. Este ponteiro pode ser apenas modificado pela *source thread*, podendo porém ser lido por qualquer *target thread*.
- ***LastRead*** - Ponteiro que indica qual a última cláusula lida pela respectiva *target thread*, sendo esta a única que possui controlo sobre o ponteiro.

Esta implementação permite que o mecanismo de partilha de cláusulas não necessite de qualquer ferramenta de sincronizaçao, dado que nunca há conflito entre *threads* sobre a escrita na mesma posição de memória. A partilha de uma cláusula aprendida por uma *target thread* dá-se após a análise de um conflito.

4.1.3 Resultados

Na tabela da página seguinte podemos observar os resultados derivados de aplicar o PMCSAT a um conjunto de instâncias da final da SAT Race de 2008. Estes testes foram efectuados numa máquina Dual Intel Xeon Quad Core processor a 2.33GHz, 24 GB de RAM e correndo o Fedora Linux, *release* 17.

As colunas da tabela fornecem informação sobre os problemas, seguindo-se pelos tempos de execução do algoritmo MINISAT e PMCSAT com partilha de cláusulas. Também estão demonstrada nesta tabela informação sobre que thread *(s)* do PMCSAT encontraram uma solução para cada problema. Quando a partilha de cláusulas é desactivada, as *threads* do algoritmo comportam-se como instâncias do MINISAT com os diferentes parâmetros de configuração descritos. Esta comparação destina-se a determinar qual das estratégias definidas na tabela 1 se adequa melhor para a resolução de um problema, ou de um determinado tipo de problemas.

Quando a partilha é activada, o cenário muda consideravelmente, visto que as cláusulas de conflito aprendidas pelas diversas *threads* são partilhadas. Desta forma, determinar a *thread* vencedora é mais imprevisível.

Seguidamente são demonstrados os cálculos de *speedups*, utilizados para classificar os potenciais ganhos do solver. Primeiro são comparados os resultados do

Instance	Sol.	#Vars	#Clauses	MiniSAT (sec)	pmcSAT (sec)	Winning Thread	Speedup pmcSAT vs no sh	Speedup pmcSAT vs MiniSAT	ManySAT (sec)	Plingeling (sec)	Speedup pmcSAT vs ManySAT	Speedup pmcSAT vs Plingeling
cmu-bmc-barrel6	U	2306	8931	1.32	0.46	0,3,5,6	2.04	2.87	0.47	0.22	1.02	0.47
cmu-bmc-longmult13	U	6565	20438	26.27	7.12	0,1,4	3.06	3.69	7.38	12.78	1.04	1.79
cmu-bmc-longmult15	U	7807	24298	15.60	6.30	0,1	2.08	2.48	5.23	10.84	0.83	1.72
ibm-2002-11r1-k45	S	156626	633125	38.19	4.39	0,1,6	8.70	8.70	21.14	22.47	4.82	5.12
ibm-2002-18r-k90	S	175216	717086	102.30	56.71	1,6	1.80	1.80	82.33	71.51	1.45	1.26
ibm-2002-20r-k75	S	151202	619733	166.08	128.10	0	1.30	1.30	107.25	52.55	0.84	0.41
ibm-2002-22r-k60	U	208590	845248	716.53	554.06	0,2	1.29	1.29	187.18	118.68	0.34	0.21
ibm-2002-22r-k75	S	191166	793646	251.07	8.37	1,6	20.38	30.00	112.03	87.37	13.38	10.44
ibm-2002-22r-k80	S	203961	846921	159.03	19.74	1	2.90	8.06	133.66	119.83	6.77	6.07
ibm-2002-23r-k90	S	222291	922916	680.85	234.78	0	2.90	2.90	158.94	212.08	0.68	0.90
ibm-2002-24r3-k100	U	148043	545315	202.90	104.05	0	1.44	1.95	95.15	74.52	0.91	0.72
ibm-2002-30r-k85	S	181484	888663	850.73	823.43	0	1.03	1.03	248.12	224.22	0.30	0.27
ibm-2004-1_11-k80	S	262808	1023506	145.46	90.96	0,1	1.43	1.60	126.74	51.12	1.39	0.56
ibm-2004-23-k100	S	207606	847320	837.61	62.91	3	11.08	13.31	177.85	89.17	2.83	1.42
ibm-2004-23-k80	S	165606	672840	232.34	16.31	3	14.25	14.25	87.87	147.61	5.39	9.05
ibm-2004-29-k25	U	17494	74526	98.99	34.64	0	2.86	2.86	24.05	27.43	0.69	0.79
mizh-md5-47-3	S	65604	234719	265.53	21.20	0,1	12.53	12.53	68.23	55.71	3.22	2.63
mizh-md5-47-4	S	65604	234811	87.00	17.85	0,1	2.23	4.87	334.92	61.24	18.76	3.43
mizh-md5-47-5	S	65604	235061	563.58	53.09	1	2.44	10.62	56.83	34.39	1.07	0.65
mizh-md5-48-5	S	66892	240181	312.49	30.16	0	6.29	10.36	367.53	42.29	12.19	1.40
mizh-sha0-35-3	S	48689	173748	29.36	5.52	1,3	1.09	5.32	36.72	14.50	6.65	2.63
mizh-sha0-35-4	S	48689	173757	262.22	17.55	1	3.89	14.94	47.27	21.75	2.69	1.24
mizh-sha0-36-1	S	50073	179811	353.95	10.46	1,3	8.51	33.84	339.40	42.22	32.45	4.04
mizh-sha0-36-4	S	50073	179989	217.09	131.42	3	1.65	1.65	733.75	22.26	5.58	0.17
velev-engi-uns-1.0-4nd	U	7000	67553	10.83	4.89	0,1	2.09	2.21	7.25	14.86	1.48	3.04
velev-fvp-sat-3.0-b18	S	35853	1012240	27.05	3.03	0,1	8.94	8.94	2.86	4.59	0.95	1.52
velev-npe-1.0-9dlx-b71	S	889302	14582952	190.91	15.49	1	2.42	12.32	268.84	37.84	17.36	2.44
velev-vliw-sat-4.0-b4	S	520721	13348116	72.90	9.03	0,1,6,7	5.32	8.07	30.98	72.51	3.43	8.03
velev-vliw-sat-4.0-b8	S	521179	13378616	101.53	21.55	0,1	1.49	4.71	42.41	60.81	1.97	2.82
velev-vliw-uns-2.0-iq1	U	24604	261472	435.23	41.77	2,4	6.14	10.42	789.34	17.52	18.90	0.42
velev-vliw-uns-2.0-iq2	U	44095	542252	TO	416.14	2,4	2.31	NA	TO	303.33	NA	0.73
avg. table set (30 inst.)	–	155385	1790335	251.34	211.23	–	4.82	8.19	158.43	69.19	5.81	2.53
avg. full set (78 inst.)	–	168240	1119370	309.48	128.87	–	3.24	37.45	182.56	102.04	9.47	2.48

Figura 4.3: Resultados da aplicação do pmcSAT a um conjunto de instâncias

pmcSAT com partilha de cláusulas vs. sem partilha de cláusulas. Como indicado anteriormente, quando a partilha é desactivada estamos a testar a eficiência das estratégias adoptadas na tabela 1, enquanto que com partilha activa são testadas as vantagens de cooperação entre *threads*. As vantagens da partilha activa são bastante óbvias: utilizando informaçã de outras *threads*, que estão a explorar outra área do espaço de procura fornece-nos informação relevante acelerando o processo de resolução. Este método dá-nos, porém, um *overhead* de exportação e importação de cláusulas periodicamente.

Na tabela seguinte é efectuada uma comparação entre a versão do pmcSAT com partilha de cláusulas activa vs. MiniSAT série. Os resultados são em geral bastante interessantes, com a média total de *speedup* acima de 37. Isto demonstra-nos que o critério de atribuição de variáveis é fulcral para uma boa resolução do problema, que é bastante sensível à porção do espaço de procura que é explorado.

Em seguida são demonstrados os tempos de execução dos algoritmos ManySAT e Plingeling [11] e [16], que utilizam abordagens semelhantes às do pmcSAT. Estão em seguida demonstradas as comparações dos *speedups*. Mais uma vez podemos observar que os resultados são bastante interessantes, sendo os tempos médios de execução do programa da mesma ordem de grandeza. Estes resultados indicam-nos que, como seria de esperar, há uma estrutura substancial a ser delineada em descrições de circuitos e outros tipos de instâncias. Se essa informação estrutural

conseguir ser obtida pela análise da fórmula CNF, então será de esperar um ganho de tempo na procura da solução da fórmula.

No geral, podemos concluir que os resultados sao bastante promissores e justificam um investimento futuro em adicionar diferentes abordagens para acelerar a resolução de problemas SAT.

4.2 clusterSAT

Nesta secção é descrito com mais detalhe o algoritmo clusterSAT. Este pode-se categorizar em duas fases essenciais:

- ***Clustering*** - Nesta fase o problema é dividido num conjunto de *clusters*. Cada *cluster* é constituído por um conjunto de cláusulas, sendo que uma cláusula apenas pode pertencer a um cluster.
- **Resolução** - Depois de partido o problema, é atribuído cada *cluster* a uma respectiva *thread*. Estas *threads* vão ser denominadas por *slaves*, sendo que vão ser controladas por uma *thread master*.

4.2.1 *Clustering*

Como foi indicado anteriormente, o problema vai ser partido com as rotinas de partição de grafos Metis e hMetis [14] e [15]. Estas operam sobre grafos e hipergrafos, respectivamente. O modo como é efectuada a partição em ambos é a seguinte:

- **Metis** - Este programa aceita como *input* um grafo, unidireccional. Considerou-se então que os nós deste grafo seriam as cláusulas do problema, sendo que as arestas representam as variáveis em comum entre cláusulas. Atribuiu-se um peso a estas arestas, tendo este o valor correspondente ao número de variáveis em comum entre essas cláusulas.
- **hMetis** - Este programa aceita como *input* um hipergrafo. A diferença neste tipos de grafos infere no facto de possuir hiper-arestas, nomeadamente arestas que ligam vários nós em simultâneo. Considerou-se então que os nós continuariam a ser as cláusulas do problema, sendo que cada hiper-aresta representa uma variável, ligando todas as cláusulas onde esta está inserida.

Na figura 4.3 temos os grafos de *input* resultantes para a fórmula $\varphi = (\omega_1 \wedge ... \wedge \omega_6)$.

Como podemos observar vamos obter dois grafos distintos. As arestas do grafo *a)* unem duas cláusulas que possuem variáveis em comum (por exemplo, a cláusula 6 partilha a variável x_1 com as cláusulas 1 e 5). Já no grafo *b)* as arestas têm um significado diferente, indicando as cláusulas onde ocorre uma variável (por exemplo, a variável x_1 é uma hiper-aresta que une os nós 1, 5 e 6).

A partição obtida pelos dois algoritmos resulta na indicação do *cluster* a que vai pertencer cada nó do grafo. Sendo assim, cada cláusula apenas pertencerá a um *cluster*. Para tal, vão ser designados dois tipos de variáveis que estão contidas nestes *clusters*:

- **Variáveis Locais** - Este tipo de variáveis são **exclusivas** a um dado *cluster*, ou seja, não ocorrem em nenhuma outra cláusula que não pertença a este.

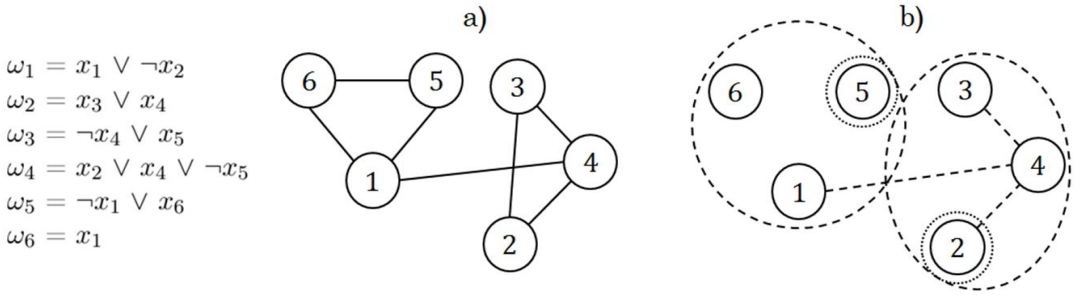

Figura 4.4: Transformação de uma fórmula CNF para um grafo a) METIS, b) HMETIS

Facilmente podemos concluir que um *slave* não necessita de efectuar qualquer tipo de contenção relativamente à atribuição de valores a estas variáveis, pois não influenciam o funcionamento do resto das *threads*.

- **Variáveis Partilhadas** - Este tipo de variáveis **não são exclusivas** a um dado *cluster*, ou seja, ocorrem em mais que um *cluster*. Estas terão um processamento mais ponderado, visto exigirem a sincronização entre *threads*, caso contrário poderão levar a conflitos de atribuições (por exemplo, existindo uma variável x_2, pertencente aos *clusters* 0 e 1, é efectuada atribuição $\rho(x_2) = 1$ no *slave* 0 enquanto que no *slave* 1 é efectuada $\rho(x_2) = 0$).

Quando se dá o início do programa, são lançados paralelamente os dois algoritmos, sendo que é aproveitada a partição do algoritmo que efectuar esta mais rapidamente. Depois de obtido o *cluster* é procedida então a resolução do problema.

4.2.2 Resolução

Para solucionar o problema de sincronização descrito anteriormente foi então implementado o seguinte algoritmo, baseado num sistema de *master-slave*. Estes têm os seguintes papéis:

- ***Master*** - Esta é a *thread* que vai controlar a execução do programa. O seu papel é garantir a mesma atribuição de todas as variáveis partilhadas aos *slaves*, de modo a não exisitir incoerência de dados, bem como analisar os resultados recebidos dos *slaves*.

- ***Slaves*** - Estes são controlados pelo *master*, tendo apenas acesso às cláusulas do *cluster* atribuído. Vão tentar encontrar uma solução do problema tendo em conta as atribuições recebidas do *master* e reportam a este o resultado obtido.

O método de resolução é o descrito nos algoritmos 2 e 3:

Analisando a estrutura do problema, facilmente se conclui que, para não serem efectuados conflitos de atribuições, terão de ser atribuídas todas as variáveis partilhadas antes de partir para a resolução de cada *cluster*. Este passo é então efectuado pelo *master*, sendo o processo é baseado no MINISAT. É utilizada a heurística de decisão desenvolvida no capítulo anterior, sendo atribuida prioridade superior às variáveis que sejam comuns a mais de um *cluster*. Uma vez atribuídas estas variáveis,

Algorithm 2 pmcSAT: Master PseudoCode

```
1: while TRUE do
2:     Assign_Global_Vars();
3:     Export_Assigns();
4:     if 1+ slaves returns UNSAT then
5:         return UNSAT;
6:     else if all slaves return SAT then
7:         return SAT;
8:     else
9:         Import_Clauses();
10:        Backtrack(MIN(slaves_backtrack_level));
11:    end if
12: end while
```

Algorithm 3 pmcSAT: Slave PseudoCode

```
1: while TRUE do
2:     Waitfor_Master_Assignments();
3:     Perform_MiniSAT() :
4:     if backtrack_done && backtrack_level < max(global_assigns.level) then
5:         Export_Clause(conflict_clause);
6:         return UNKNOWN;
7:     end if
8:     if state FOUND then
9:         return STATE;
10:    end if
11: end while
```

as decisões são comunicadas aos *slaves* e estes procederão com o resto do algoritmo. O trabalho de um *slave* pode gerar 3 resultados possíveis:

- O *slave* encontra uma solução para o problema e comunica **SAT** ao *master*.
- O *slave* conclui que o problema é insatisfeito e comunica **UNSAT** ao *master*.
- O *slave* encontra encontra um conflito que reside na atribuição das variáveis partilhadas. Neste caso exporta para o *master* a cláusula de conflito aprendida e comunica **UNKNOWN**.

Recebidas então as respostas de todos os *slaves*, o *master* vai retomar o trabalho, tomando uma das seguintes acções:

- Caso todos os *slaves* retornem **SAT**, visto que as atribuições de variáveis partilhadas entre estes são coerentes podemos considerar que o problema é satisfeito.
- Caso um ou mais *slaves* retornem **UNSAT**, pode-se considerar que o problema é insatisfeito, visto o *slave* estar a trabalhar sobre um sub-problema, implicando que o problema geral seja também insatisfeito.

- Finalmente, se nenhum dos *slaves* retornar UNSAT e pelo menos um *slave* ter chegado a um conflito (**UNKNOWN**), o *master* vai retroceder até ao nível mais superior da árvore onde os *slaves* detectaram o conflito e vai refazer as atribuições (já com as respectivas cláusulas aprendidas), retomando seguidamente o trabalho aos *slaves* . Este procedimento é efectuado até ser sucedida uma das duas situações descritas acima.

4.3 Resultados

Os resultados obtidos são ainda relativos às partições do problema obtidas, visto que o algoritmo ainda está numa fase de desenvolvimento, não possuindo resultados suficientes para determinar a eficiência deste método. Nas seguintes tabelas estão então demonstrados os resultados para as partições efectuadas a algumas das instâncias da SAT Race de 2008 utilizando os algoritmos METIS e hMETIS.

Instance	#Vars	#Clauses	t_Metis 2 clusters (sec)	t_Metis 4 clusters (sec)	%global vars (2 clusters)	%global vars (4 clusters)	%global clauses (2 clusters)	%global clauses (4 clusters)	%global vars/clause (2 clusters)	%global vars/clause (4 clusters)
cmu-bmc-barrel6.cnf.gz	8931	2306	0,23	0,19	9,54%	21,12%	22,26%	51,22%	12,20%	27,47%
cmu-bmc-longmult13.cnf.gz	20438	6565	0,24	0,25	1,25%	3,79%	2,39%	6,74%	1,26%	3,35%
cmu-bmc-longmult15.cnf.gz	24298	7807	0,31	0,28	1,10%	3,88%	2,77%	6,79%	1,52%	3,41%
ibm-2002-11r1-k45.cnf.gz	633125	156626	16,28	16,38	0,32%	0,64%	0,56%	1,03%	0,30%	0,53%
ibm-2002-18r-k90.cnf.gz	717086	175216	17,93	16,37	0,05%	0,24%	0,07%	0,52%	0,03%	0,27%
ibm-2002-20r-k75.cnf.gz	619733	151202	15,17	15,12	0,06%	0,18%	0,10%	0,26%	0,05%	0,12%
ibm-2002-22r-k75.cnf.gz	793646	191166	16,91	17,37	0,12%	0,31%	0,19%	0,49%	0,09%	0,23%
ibm-2002-22r-k80.cnf.gz	846921	203961	19,58	20,08	0,06%	0,17%	0,09%	0,26%	0,04%	0,12%
ibm-2002-23r-k90.cnf.gz	922916	222291	22,22	21,04	0,05%	0,16%	0,08%	0,26%	0,04%	0,12%
ibm-2002-24r3-k100.cnf.gz	545315	148043	9,69	8,42	0,13%	0,41%	0,27%	0,76%	0,12%	0,37%
ibm-2004-29-k25.cnf.gz	74526	17494	1,20	0,89	1,10%	2,26%	2,06%	3,80%	0,93%	1,90%
mizh-md5-47-3.cnf.gz	234719	65604	3,59	3,12	8,04%	12,56%	17,31%	27,27%	6,92%	10,87%
mizh-md5-47-4.cnf.gz	234811	65604	3,55	3,39	6,07%	11,98%	16,29%	26,55%	6,03%	10,55%
mizh-md5-47-5.cnf.gz	235061	65604	3,53	3,26	8,26%	14,20%	17,85%	29,40%	7,11%	12,15%
mizh-md5-48-5.cnf.gz	240181	66892	3,48	3,47	8,33%	12,33%	17,67%	27,54%	7,07%	11,03%
mizh-sha0-35-3.cnf.gz	173748	48689	2,24	2,21	4,08%	8,89%	12,13%	23,94%	5,05%	10,40%
mizh-sha0-35-4.cnf.gz	173757	48689	2,27	2,37	4,12%	9,29%	12,22%	24,18%	5,11%	10,62%
mizh-sha0-36-1.cnf.gz	179811	50073	2,47	2,56	5,00%	9,13%	13,62%	23,99%	5,73%	10,59%
mizh-sha0-36-4.cnf.gz	179989	50073	2,75	2,37	5,02%	9,10%	13,61%	23,77%	5,74%	10,39%
velev-engi-uns-1.0-4nd.cnf.gz	67553	7000	9,88	9,95	42,31%	59,31%	87,58%	97,78%	62,13%	77,92%

Instance	#Vars	#Clauses	t_hMetis 2 clusters (sec)	t_hMetis 4 clusters (sec)	%global vars (2 clusters)	%global vars (4 clusters)	%global clauses (2 clusters)	%global clauses (4 clusters)	%global vars/clause (2 clusters)	%global vars/clause (4 clusters)
cmu-bmc-barrel6.cnf.gz	8931	2306	0,14	0,34	7,85%	13,70%	38,66%	59,88%	24,46%	40,05%
cmu-bmc-longmult13.cnf.gz	20438	6565	0,32	0,71	0,75%	1,90%	3,29%	17,24%	1,73%	8,34%
cmu-bmc-longmult15.cnf.gz	24298	7807	0,38	0,79	0,63%	1,64%	2,80%	9,10%	1,39%	4,54%
ibm-2002-11r1-k45.cnf.gz	633125	156626	29,57	52,77	0,15%	0,48%	1,06%	3,34%	0,52%	1,61%
ibm-2002-18r-k90.cnf.gz	717086	175216	24,48	49,46	0,05%	0,15%	0,44%	1,32%	0,22%	0,65%
ibm-2002-20r-k75.cnf.gz	619733	151202	24,26	40,59	0,06%	0,17%	0,52%	1,35%	0,26%	0,65%
ibm-2002-22r-k75.cnf.gz	793646	191166	35,86	65,80	0,06%	0,18%	0,59%	1,65%	0,30%	0,81%
ibm-2002-22r-k80.cnf.gz	846921	203961	37,38	68,78	0,06%	0,17%	0,56%	1,42%	0,28%	0,70%
ibm-2002-23r-k90.cnf.gz	922916	222291	44,65	84,72	0,05%	0,16%	0,47%	1,08%	0,24%	0,53%
ibm-2002-24r3-k100.cnf.gz	545315	148043	24,75	42,83	0,11%	0,34%	0,29%	1,50%	0,13%	0,71%
ibm-2004-29-k25.cnf.gz	74526	17494	1,94	3,30	0,38%	1,14%	1,23%	6,00%	0,57%	2,80%
mizh-md5-47-3.cnf.gz	234719	65604	7,74	11,63	1,31%	2,02%	18,45%	25,60%	6,07%	8,99%
mizh-md5-47-4.cnf.gz	234811	65604	8,36	12,91	1,31%	2,02%	18,59%	25,39%	6,12%	8,94%
mizh-md5-47-5.cnf.gz	235061	65604	6,83	12,63	1,30%	2,03%	18,61%	25,13%	6,13%	8,85%
mizh-md5-48-5.cnf.gz	240181	66892	6,67	13,37	1,29%	1,98%	18,56%	25,16%	6,09%	8,84%
mizh-sha0-35-3.cnf.gz	173748	48689	6,75	9,44	1,55%	2,89%	13,77%	28,49%	4,86%	10,51%
mizh-sha0-35-4.cnf.gz	173757	48689	4,68	11,10	1,60%	2,90%	13,89%	28,79%	4,92%	10,61%
mizh-sha0-36-1.cnf.gz	179811	50073	4,92	9,81	1,57%	2,87%	14,90%	28,13%	5,44%	10,30%
mizh-sha0-36-4.cnf.gz	179989	50073	6,54	9,71	1,57%	2,83%	14,37%	27,84%	5,23%	10,19%
velev-engi-uns-1.0-4nd.cnf.gz	67553	7000	1,83	3,36	8,83%	11,39%	75,93%	84,96%	40,99%	46,10%

Para estas instâncias, foram utilizadas partições de 2 e 4 *clusters*. Como podemos observar, os resultados das partições são bastante diversificados, o que nos leva a concluir que a partição resultante é altamente dependente da natureza do problema.

Rapidamente observamos que o número de *clusters* no qual vai ser efectuada a partição não tem grande impacto no tempo de execução do METIS, sendo que para o hMetis ao aumentar o número de partições o tempo de execução deste vai aumentar significativamente.

Podemos também verificar, como seria de esperar, que o número de variáveis partilhadas aumenta à medida que o número de *clusters* é aumentado, tanto para o METIS como para HMETIS. Uma observação bastante interessante é o facto do hMetis, apesar de obter partições com um menor número de variáveis partilhadas comparativamente com o METIS, o número de cláusulas que contém variáveis globais (cláusulas globais) é maior que a partição obtida pelo METIS. Isto leva-nos a concluir que a partição efectuada pelo METIS procura difundir as ligações partilhadas entre os vários *clusters* por todo o sub-problema, sendo que o HMETIS realiza o oposto, ou seja, visa concentrar as ligações partilhadas em determinados nós do sub-problema. Como seria então de esperar, e apoiando-nos na tabela, podemos concluir que a percentagem de variáveis globais por cláusula será maior no Metis.

Observando os diversos tipos de instâncias, mais precisamente as da IBM, que são instâncias de verificação de circuitos, são obtidos grandes resultados, tanto para o METIS como para o HMETIS, para 2 e 4 *clusters*, sendo a % de variáveis globais um valor bastante residual. Podemos concluir que este tipo de problemas terá grande potencial de resultados, pois necessita de muita pouca sincronização entre o *master* e os diversos *slaves*.

Capítulo 5

Conclusões

O progresso no desenvolvimento de sistemas *multi-core*, bem como outros ambientes paralelos gera uma oportunidade para acelerar a resolução de problemas SAT.

Neste relatório foram demonstradas duas abordagens distintas ao problema SAT. Numa das abordagens, multíplas instâncias do mesmo solver, utilizando diferentes estratégias para a pesquisa no espaço de procura e partilha de cláusulas, sendo que na outra este espaço é dividido e é utilizado um modelo cooperativo de resolução do problema.

Os resultados obtidos da aplicação destes métodos a instâncias de competições de SAT, com importância prática, demonstram progressos relevantes no estado da arte e capacidade para serem efectuados avanços futuros.

Referências Bibliográficas

[1] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, July 1962.

[2] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of DAC*, Las Vegas, Nevada, USA, June 2001, pp. 530–535.

[3] J. P. M. Silva and K. A. Sakallah, "GRASP: A New Search Algorithm for Satisfiability," in *Proceedings of ICCAD*, San Jose, California, United States, 1996, pp. 220–227.

[4] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman, "Dynamic Restart Policies," in *Proceedings of AAAI*, Edmonton, Alberta, Canada, 2002, pp. 674–681.

[5] N. Een and N. Sorensson, "An Extensible SAT-solver," *Theory and Applications of Satisfiability Testing*, vol. 2919, pp. 502–518, 2004.

[6] L. Gil, P. Flores, and L. M. Silveira, "PMSat: a parallel version of MiniSAT," *JSAT*, vol. 6, pp. 71–98, 2008.

[7] K. Ohmura and K. Ueda, "c-sat: A Parallel SAT Solver for Clusters," in *Theory and Applications of Satisfiability Testing*, ser. LNCS, O. Kullmann, Ed. Springer Berlin / Heidelberg, 2009, vol. 5584, pp. 524–537.

[8] B. B. T. Schubert, M. Lewis, "Pamiraxt: Parallel sat solving with threads and message passing," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 6, pp. 203–222, 2009.

[9] T. Schubert, M. Lewis, and B. Becker, "Pamira - a parallel sat solver with knowledge sharing," in *Proceedings of International Workshop on Microprocessor Test and Verification*. IEEE Computer Society, 2005, pp. 29–36.

[10] Y. Feldmana, N. Dershowitz, and Z. Hanna, "Parallel multithreaded satisfiability solver: Design and implementation," in *Proceedings of International Workshop on Parallel and Distributed Methods in Verification (PDMC)*, ser. Electronic Notes in Theoretical Computer Science (ENTCS), vol. 128, no. 3, 2005, pp. 75–90.

[11] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: A Parallel SAT Solver," *JSAT*, vol. 6, 2009.

[12] C. Sinz and et al, "SAT Race 2008," http://baldur.iti.uka.de/sat-race-2008/index.html, 2008, accessed on May 2012.

[13] D. D. D. II and et al, "FLAIRS-26 Conference," http://www.flairs-26.info/, 2013.

[14] G. Karypis and V. Kumar, "Metis, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 5.0," http://glaros.dtc.umn.edu/gkhome/metis/metis/download, 2011.

[15] G. Karypis and Kumar, "hmetis, a hypergraph partitioning package version 1.5.3," http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download, 2011.

[16] A. Biere, "Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010," FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Austria, Tech. Rep. 10/1, August 2010.