PColorAnt3-RT: Um Algoritmo ACO Paralelo para Coloração de Grafos

Carla Négri Lintzmayer [1]
Mauro Henrique Mulati [2]
Anderson Faustino da Silva [3]

Resumo: Em trabalhos anteriores foram apresentadas investigações de três diferentes algoritmos baseados em *Ant Colony Optimization* (ACO) aplicados ao problema de coloração de grafos com k cores. Os resultados obtidos demonstraram que *ColorAnt*$_3$*-RT* é o melhor dentre os três algoritmos *ColorAnt-RT*, sendo capaz de obter boas e até mesmo ótimas soluções para os melhores valores de k descritos na literatura, além de minimizar a quantidade de conflitos. Porém, em aplicações onde o tempo de execução é um fator crucial, algoritmos ACO não tem sido utilizados. Este artigo propõe e demonstra o uso de uma abordagem paralela para uma arquitetura de *hardware* com memória compartilhada com o objetivo de reduzir o tempo de execução de *ColorAnt*$_3$*-RT*, originando o algoritmo paralelo *PColorAnt*$_3$*-RT* que é capaz de encontrar boas e ótimas soluções em tempo de execução aceitável.

Abstract: In previous work, we presented an investigation of three versions of ACO algorithm applied to graph coloring problem with k colors. The results demonstrated that *ColorAnt*$_3$*-RT* is indeed the best among the three *ColorAnt-RT* algorithms and it is a good option in obtaining good approximations to the best known values of k described in literature, besides minimizing the amount of conflicts. However, a problem of ACO algorithms is their high runtime. Due to this problem ACO algorithms have not been used, especially on this kind of problem – graph coloring. This paper presents a strategy to minimize the *ColorAnt*$_3$*-RT* runtime, the parallel algorithm *PColorAnt*$_3$*-RT* for shared-memory multiprocessor that is able to find good approximations in an acceptable runtime.

[1]Universidade de Campinas
Instituto de Computação
Avenida Albert Einstein, 1251 - Cidade Universitária, Campinas/SP - Brasil - CEP 13083-852
{carla0negri@gmail.com}
[2]Universidade Estadual do Centro-Oeste
Departamento de Ciência da Computação
Campus Cedeteg - Rua Camargo Varela de Sá, 03 - Vila Carli, Guarapuava/PR - CEP 85040-080
{mhmulati@gmail.com}
[3]Universidade Estadual de Maringá
Departamento de Informática
Avenida Colombo, 5790 - Bloco C56 - Maringá/PR - CEP 87020-900
{anderson@din.uem.br}

1 Introdução

Obter uma solução para o problema de coloração de grafos (PCG) consiste basicamente em encontrar uma configuração com uma quantidade k de cores que possam ser atribuídas aos vértices de um grafo G de forma que não existam vértices adjacentes com a mesma cor. O objetivo é, então, minimizar a quantidade de cores, onde o mínimo possível é chamado número cromático de G, representado por $\chi(G)$ [3].

O k-PCG, o PCG descrito como um problema de decisão, consiste em descobrir se um grafo pode ser colorido com k cores fixas. Dessa forma deve-se tentar colorir um grafo com no máximo k cores, com objetivo de minimizar o número de *arestas conflitantes*, sendo que aresta conflitante é aquela que incide em vértices de mesma cor. O caráter $\mathcal{NP}$-difícil do PCG tem levado à realização de trabalhos que exploram metaheurísticas e algoritmos heurísticos [12, 16, 23]. Este problema aparece em diversas situações onde é necessário particionar um conjunto de elementos em grupos com determinadas características compatíveis entre os membros [3].

Dentre as metaheurísticas existentes, o presente trabalho destaca a Otimização por Colônia de Formigas, ou *Ant Colony Optimization* (ACO), que se baseia no comportamento apresentado por formigas durante a busca por alimento em um ambiente [9]. Em algoritmos ACO, cada formiga artificial é geralmente um método construtivo e seu comportamento é percebido principalmente quando, para decidir para onde ela deve dar o próximo "passo", usa-se uma probabilidade calculada com base em dois fatores: a trilha de feromônio e a informação heurística relacionadas ao item. Diversos trabalhos propõem o uso de algoritmos ACO para a resolução de diversos problemas [9] como: roteamento de veículos, atribuição de frequência e coloração de grafos.

ColorAnt$_3$*-RT* é um promissor algoritmo ACO [21] para coloração de grafos, capaz de obter boas soluções para os melhores resultados conhecidos. Além disto, *ColorAnt*$_3$*-RT* é capaz de minimizar a quantidade de conflitos, que é o principal problema de coloração de grafos com uma quantidade fixa de cores.

Uma aplicação prática de *ColorAnt*$_3$*-RT* foi utilizá-lo como uma das fases de um alocador de registradores baseado em coloração de grafos [19]. Os resultados obtidos por este alocador de registradores foi superior aos obtidos pelo alocador tradicional desenvolvido por George e Appel [1, 13], contudo ao custo de um elevado tempo de execução. Neste caso específico, o compilador que utiliza *ColorAnt*$_3$*-RT* durante a fase de alocação de registradores chega a ser 600 vezes mais lento do que aquele que utiliza o algoritmo proposto por George e Appel. Desta forma, é importante investigar estratégias de reduzir o tempo de execução de *ColorAnt*$_3$*-RT*.

O presente trabalho propõe e investiga o algoritmo *PColorAnt*$_3$*-RT*, a versão paralela

de *ColorAnt*$_3$*-RT*. O objetivo principal no desenvolvimento de *PColorAnt*$_3$*-RT* é reduzir o tempo de execução de *ColorAnt*$_3$*-RT*, como também investigar o impacto da programação paralela com memória compartilhada em algoritmos ACO. Os resultados obtidos por *PColorAnt*$_3$*-RT* demonstraram que ele é uma boa opção para encontrar boas soluções em um tempo de execução menor que *ColorAnt*$_3$*-RT*. Desta forma, as contribuições deste artigo são:

1. Uma estratégia para reduzir o tempo de execução de *ColorAnt*$_3$*-RT*;
2. Um algoritmo ACO paralelo capaz de encontrar soluções satisfatórias em um tempo de execução aceitável; e
3. Uma análise detalhada do impacto da programação paralela em *ColorAnt*$_3$*-RT*.

O texto segue organizado da seguinte forma: a Seção 2 apresenta o referencial teórico deste trabalho; a Seção 3 apresenta alguns trabalhos relacionados; a Seção 4 descreve o algoritmo *ColorAnt*$_3$*-RT*; a Seção 5 descreve a versão paralela do algoritmo *ColorAnt*$_3$*-RT*; a Seção 6 apresenta a avaliação experimental; e por fim, a Seção 7 apresenta as considerações finais.

2 Referencial Teórico

2.1 O Problema de Coloração de Grafos

Uma k-coloração de um grafo $G = (V, E)$ é uma atribuição de k cores aos seus vértices, ou seja, um mapeamento $c : V \rightarrow \{1..k\}$. Outra abordagem é o particionamento de V em k conjuntos independentes ou *classes legais* de cores $s = \{C_1, C_2, ..., C_k\}$.

A coloração é *própria* [3] quando ela não possui nenhuma *aresta conflitante*. Um grafo é *k-colorível* se possuir uma k-coloração própria. O valor mínimo de k que permite a um grafo ser k-colorível é o *número cromático* do grafo e é representado por $\chi(G)$. O PCG consiste, portanto, de encontrar o menor valor de k tal que o grafo seja k-colorível. Deseja-se minimizar a função objetivo que, para o PCG, é o número de cores da solução.

O problema da k-coloração descrito como um problema de decisão (k-PCG) é da seguinte forma [25]: dado um grafo G e um número fixo inteiro k de cores possíveis, G é k-colorível? Neste caso, deseja-se minimizar o número de arestas conflitantes da solução, a função objetivo deste problema.

Alguns grafos com características específicas possuem χ conhecido e fixo, como grafos bipartidos (2-coloríveis) e grafos planares (4-coloríveis). Para determinar se um grafo é 2-colorível existem algoritmos em tempo polinomial [3]. Para outros casos não especiais, o

PCG necessita de técnicas que o resolvam de maneira satisfatória, já que não se conhecem (e provavelmente não existem, a menos que $\mathcal{P} = \mathcal{NP}$) algoritmos exatos que processem grafos com mais de 100 vértices [23].

Uma aplicação real do k-PCG é vista na alocação de registradores [1]. Neste problema, a solução não se restringe apenas a verificar se um grafo é k-colorível, mas também deve utilizar alguma heurística que possibilite "eliminar" as arestas conflitantes da melhor forma possível, já que é obrigatório colorir o grafo apenas com k cores.

2.2 Otimização por Colônia de Formigas

Colônias de formigas naturais são organizadas e apresentam comportamento que permite a realização de diversas tarefas que não seriam possíveis por uma única formiga. A comunicação indireta que as coordena e orienta se dá por alterações que elas realizam no ambiente, em um processo denominado "*stigmergy*" [9]. Na maioria dos casos, essa comunicação se dá pelo depósito da substância química *feromônio* na superfície, formando trilhas que guiam o caminho de cada formiga.

Quanto maior a concentração de feromônio em um caminho, maior a probabilidade da formiga escolhê-lo. Tal comportamento é chamado de autocatalítico: um processo que reforça a si mesmo causando convergência [11]. Como o feromônio evapora com o tempo e caminhos mais curtos são percorridos mais rapidamente (incentivando as formigas a passarem mais vezes por eles), a concentração de feromônio nestes caminhos tenderá a ser maior. Portanto, em algum momento a tendência é que a colônia percorra o menor caminho possível entre dois pontos, característica que atraiu a atenção para o fato de que o comportamento das formigas poderia ser mapeado e utilizado computacionalmente.

ACO é uma metaheurística de otimização combinatória que se baseia no comportamento das formigas naturais. Em algoritmos ACO cada formiga artificial é geralmente um método construtivo e seu comportamento é percebido principalmente quando, para decidir para onde ela deve dar o próximo "passo", usa-se uma probabilidade calculada com base em dois fatores: a trilha de feromônio e a informação heurística relacionadas ao item [10]. A informação heurística depende de cada problema.

Diferentes tipos de métodos baseados em colônias de formigas artificiais foram desenvolvidos para o PCG. Eles são classificados em três classes [15]: (1) constituído de algoritmos nos quais cada formiga é um método construtivo que reforça a trilha de feromônio entre pares de vértices não-adjacentes quando eles recebem a mesma cor; (2) composto por algoritmos nos quais as formigas caminham pelo grafo nem sempre previamente colorido e tentam modificar a cor dos vértices de forma a reduzir o número de conflitos existentes; e (3) aqueles nos quais as formigas são algoritmos de busca local, onde, partindo-se de um grafo já colorido, cada formiga encontra uma solução vizinha, que normalmente é a atribuição de

uma nova cor a um vértice conflitante da solução atual.

As duas últimas classes se diferenciam de forma significativa da idéia original de um algoritmo ACO e existem divergências com relação ao fato de poder considerar tais algoritmos como sendo "baseados em colônias de formigas artificiais" [15]. Em geral, os algoritmos que simulam a trilha de feromônio o fazem de maneira semelhante: vértices adjacentes não possuem valor na trilha e vértices não-adjacentes que recebem a mesma cor de alguma formiga têm seu feromônio reforçado. O algoritmo *ColorAnt*$_3$*-RT* pertence a Classe 1.

3 Trabalhos Relacionados

Vários trabalhos investigaram algoritmos ACO sequenciais para o PCG. O primeiro foi o *ANTCOL* [7], onde cada formiga busca o menor valor de k possível, utilizando os métodos construtivos *RLF* (*Recursive Large First*) [4] e *Dsatur* [17]. Este difere de *ColorAnt*$_3$*-RT* no uso da probabilidade (que em ambos envolve o feromônio e a informação heurística), que no *ANTCOL* serve para escolher um novo vértice a ser colorido e no *ColorAnt*$_3$*-RT* serve para escolher uma cor para colorir um vértice.

Em uma abordagem diferente, a cada iteração cada formiga se move probabilisticamente para um vértice adjacente que possua o maior número de arestas conflitantes, onde ela substitui, também probabilisticamente, a cor atual por uma nova que minimize os conflitos [6]. Em outro algoritmo cada formiga colore um único vértice, de forma que a colônia inteira encontra apenas uma solução [14]. *ColorAnt*$_3$*-RT* difere de ambos por não utilizar formigas caminhando pelo grafo.

Em um algoritmo mais recente, o *ALS-COL* (*Ant Local Search*) [23], cada formiga é uma busca local derivada de Busca Tabu para o k-PCG. *ColorAnt*$_3$*-RT* difere deste pelo fato de cada formiga ser um método construtivo e não uma busca local.

Algoritmos ACO paralelos também foram investigados, contudo aplicados a outros problemas, a saber: cobertura de conjuntos [24] e clique máximo de um grafo [5].

O algoritmo AntsLS [24] apresenta uma proposta baseada na combinação de colônia de formigas e busca local, aplicado à cobertura de conjuntos. Dois algoritmos paralelos foram propostos. No primeiro, cada processador tem uma instância do algoritmo e ao final da execução cada instância envia ao processador *mestre* a solução encontrada, que por sua vez seleciona a melhor. No segundo, cada processador representa uma formiga, sendo o feromônio global a todas as formigas. *PColorAnt*$_3$*-RT* possui a mesma estrutura do segundo algoritmo, porém difere na plataforma de hardware utilizada: AntsLS poi projetado para um *cluster* de computadores e desenvolvido com PVM, e *PColorAnt*$_3$*-RT* foi desenvolvido para uma arquitetura de memória compartilhada, utilizando *pthread*.

O algoritmo ABOMC [5] utiliza colônia de formigas artificiais para encontrar o clique máximo de um grafo e segue o modelo de paralelização da versão sequencial utilizando OpenMP. *PColorAnt*$_3$*-RT* é similar a este por utilizar uma plataforma de memória compartilhada.

4 O Algoritmo *ColorAnt*$_3$*-RT*

ColorAnt$_3$*-RT* é um algoritmo ACO para coloração de grafos que usa *React-Tabucol*, uma busca local baseada em Busca Tabu. Cada formiga em *ColorAnt*$_3$-RT é um método construtivo chamado k-ANTCOL, que por sua vez é baseado no método ANTCOL [7] modificado para colorir um grafo com k cores.

No k-ANTCOL, apresentado no Algoritmo 1, a cada passo de construção, deve-se escolher um vértice v (ainda não colorido) com o maior grau de saturação $gsat(v)$[4] e deve-se escolher uma cor c com probabilidade p para atribuir a v.

Algoritmo 1 Pseudocódigo do *k-ANTCOL*.

K-ANTCOL($G = (V, E)$, k) // G: grafo; V: vértices; E: arestas

```
1  NC = V;  // vértices ainda não coloridos
2  cor_i = 0  ∀i ∈ V;  // vetor cor mantém um mapeamento vértice-cor
3  ncoloridos = 0;
4  while ncoloridos < |V| do
5      v = arg max{gsat(v') | v' ∈ NC};
6      escolher uma cor c ∈ {1..k} com probabilidade p(s, v, c) dada pela Equação 1;
7      cor_v = c;
8      NC = NC\{v};
9      ncoloridos++;
10 return cor;
```

A probabilidade p é apresentada na Equação 1 e é calculada com base na trilha de feromônio τ, apresentada na Equação 2, e na informação heurística η, apresentada na Equação 3.

$$p(s,v,c) = \frac{\tau(s,v,c)^{\alpha} \cdot \eta(s,v,c)^{\beta}}{\sum_{i \in \{1..k\}} \tau(s,v,i)^{\alpha} \cdot \eta(s,v,i)^{\beta}} \quad (1)$$

[4]Grau de saturação é o número de cores diferentes que já foram atribuídas aos nós adjacentes de um vértice.

onde α e β são parâmetros do algoritmo que controlam a influência dos valores associados a eles na equação.

$$\tau(s,v,c) = \begin{cases} 1 & \text{se } C_c(s) = \{\} \\ \dfrac{\sum_{u \in C_c(s)} F_{uv}}{|C_c(s)|} & \text{caso contrário} \end{cases} \tag{2}$$

$$\eta(s,v,c) = \begin{cases} 1 & \text{se } N_{C_c(s)}(v) = \{\} \\ \dfrac{1}{|N_{C_c(s)}(v)|} & \text{caso contrário} \end{cases} \tag{3}$$

onde F_{uv} é a trilha de feromônio entre os vértices u e v, $C_c(s)$ é o conjunto de vértices já coloridos com a cor c na solução s, e $N_{C_c(s)}(v)$ são os vértices $x \in C_c(s)$ adjacentes a v na solução s.

A trilha de feromônio é armazenada na matriz $F_{|V|\times|V|}$ e inicializada com 1 nas arestas de vértices não-adjacentes e 0 nas arestas de vértices adjacentes. Sua atualização envolve persistir a trilha atual por um fator ρ ($1 - \rho$ é a taxa de evaporação), conforme a Equação 4, e reforçá-la por meio da experiência obtida nas soluções geradas, cuja forma geral é mostrada na Equação 5.

$$F_{uv} = \rho F_{uv} \quad \forall u, v \in V \tag{4}$$

$$F_{uv} = F_{uv} + \frac{1}{f(s)} \quad \forall u, v \in C_c(s) \mid (u,v) \notin E, c = 1..k \tag{5}$$

onde s é uma solução, $C_c(s)$ é o conjunto de vértices coloridos com a cor c na solução s e f é a função objetivo e retorna o número de arestas conflitantes da solução.

Em relação ao tratamento da trilha de feromônio, *ColorAnt*$_3$*-RT* utiliza a melhor formiga da iteração após a aplicação da busca local (s') e a melhor formiga encontrada durante toda a execução até o momento (s^*), mas não simultaneamente. Inicialmente s' reforça a trilha mais frequentemente do que s^* e uma troca gradual nesta frequência é feita com base na quantidade máxima de ciclos do algoritmo.

Desde as versões iniciais [18, 20, 21], *ColorAnt*$_3$*-RT* passou por algumas alterações em relação às regras de atualização do feromônio, permitindo melhores valores de k e principalmente diminuindo a quantidade de conflitos das soluções encontradas. *ColorAnt*$_3$*-RT* é descrito no Algoritmo 2.

A busca *React-Tabucol* utilizada por *ColorAnt*$_3$*-RT* funciona sobre um espaço de soluções S onde cada solução é formada por k classes de cores e todos os vértices estão

coloridos. Um movimento é a troca da cor de um único vértice que ocorre entre soluções *vizinhas* e quando ocorre, seu inverso é armazenado em uma *lista tabu*, sendo proibido de ser realizado nas próximas t gerações. Partindo de uma solução inicial $s_0 \in S$, a busca gera uma sequência $s_1, s_2, \ldots$ de soluções em S, com s_{i+1} sendo vizinha de s_i a partir de um movimento não-tabu. Entre elas deve-se escolher a solução com o menor número de conflitos.

Algoritmo 2 Pseudocódigo do *ColorAnt*$_3$*-RT*.

COLORANT$_3$-RT($G = (V, E), k$) // G: grafo; V: vértices; E: arestas

```
1  F_uv = 1 ∀(u,v) ∉ E;
2  F_uv = 0 ∀(u,v) ∈ E;
3  f* = ∞;  // melhor valor da função objetivo na execução (número de conflitos)
4  while ciclo < max_ciclos and f* ≠ 0
   and tempo < max_tempo and converg < 4 · √max_ciclos do
5      f' = ∞;  // melhor valor da função objetivo no ciclo
6      for a = 1 to nformigas do
7          s = K-ANTCOL(G, k);
8          REACT_TABUCOL(s);
9          if f(s) < f' then
10             s' = s;
11             f' = f(s');
12     if f' < f* then
13         s* = s';
14         f* = f(s*);
15         converg = 0;
16     F_uv = ρF_uv  ∀u,v ∈ V;
17     if ciclo mod √max_ciclos == 0 then
18         fero_cont = ciclo ÷ √max_ciclos;
19     s = s';
20     if fero_cont > 0 then
21         s = s*;
22     F_uv = F_uv + 1/f(s)   ∀u,v ∈ C_c(s) | (u,v) ∉ V, c = 1..k;
23     fero_cont = fero_cont−1;
24     ciclo = ciclo+1;
25     converg = converg+1;
```

5 O Algoritmo *PColorAnt$_3$-RT*

É importante observar que algoritmos ACO diferem dos algoritmos tradicionais investigados em pesquisas com programação paralela, por exemplo aqueles contidos nos *benchmarks* SPLASH [27] e PARSEC [2]. Tradicionalmente, a abordagem utilizada na paralelização de algoritmos sequenciais tem sido particionar a entrada pelos diversos processadores e então inserir pontos de sincronização onde exista a necessidade de comunicação e/ou consistência dos dados. Contudo, esta abordagem não se encaixa à maioria das aplicações que utilizam um algoritmo ACO, a exemplo do *PColorAnt$_3$-RT*, pois neste contexto cada processador necessita de todos os dados de entrada, pois cada formiga cria uma solução completa. Portanto, não é possível particionar os dados pelos processadores.

Assim, a abordagem utilizada para paralelizar *ColorAnt$_3$-RT* foi distribuir uma quantidade N de formigas pelos processadores (ou núcleos, dependendo da arquitetura de hardware utilizada). Além disto, cada formiga deve ter acesso à matriz de feromônio, que é uma informação global a todos os processadores. A Figura 1 apresenta a arquitetura do algoritmo *PColorAnt$_3$-RT*.

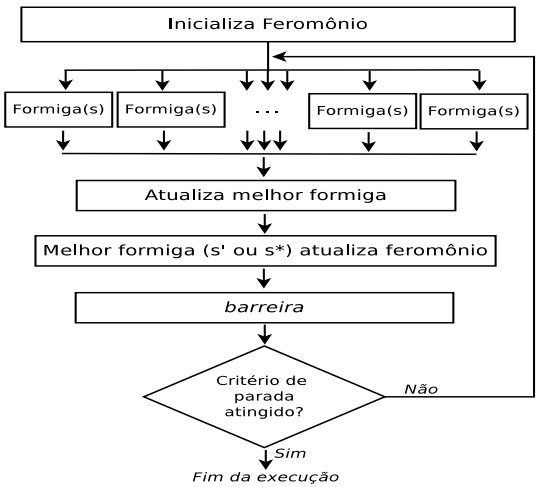


Figura 1. Arquitetura do algoritmo *PColorAnt$_3$-RT*.

Como pode ser observado na Figura 1, inicialmente existe uma distribuição das formigas pelos diversos processadores e a matriz de feromônio é inicializada. A partir deste ponto, cada processador inicia a busca por uma solução. Assim que cada processador encontra uma solução, este atualiza s^* e espera em uma barreira, devido a necessidade de atualização do feromônio. No algoritmo proposto, não existe a necessidade de todos processadores atu-

alizarem a matriz de feromônio, pois isto é dependente apenas de s' ou s^*. Portanto, apenas um processador realiza esta tarefa, enquanto os outros ficam bloqueados em uma barreira. Após todos processadores saírem da barreira, estes verificam se a condição de parada foi atingida. Caso verdadeiro, a execução encerra, caso contrário a execução retoma na tentativa de encontrar uma solução melhor do que aquela encontrada até o presente momento.

Da forma que *PColorAnt*$_3$*-RT* está implementado são necessários dois pontos de sincronização, a saber: (1) exclusão mútua, para garantir que a melhor solução (s^*) encontrada não seja corrompida devido a acessos concorrentes; e (2) uma barreira para garantir que o próximo ciclo (caso exista) tenha as informações necessárias do ciclo anterior.

Em uma visão macro, cada processador tem uma instância de *ColorAnt*$_3$*-RT*, onde as linhas 16 a 23 do Algoritmo 2 (atualização do feromônio) são executadas apenas por um único processador. Vale ainda ressaltar que a condição de parada é uma informação global. Desta forma, é garantido que todos processadores/formigas terão uma visão global da matriz de feromônio.

6 Avaliação Experimental

ColorAnt$_3$*-RT* e *PColorAnt*$_3$*-RT* foram implementados na linguagem `C`, compilados com GCC 4.4.3 utilizando o nível de otimização `O3`. Além disto, a paralelização de *ColorAnt*$_3$*-RT* foi feita com o uso da biblioteca *pthreads* [22]. Ambos foram executados em um computador Intel Xeon E5504 de 2.00 GHz (2 processadores com 4 núcleos cada), 24GB de memória RAM e sistema operacional Ubuntu 10.04.3 LTS com Kernel 2.6.32-37-server.

6.1 Metodologia

Os experimentos foram realizados em 18 grafos[5] do Desafio DIMACS [8], a saber:

- `dsjc500.1`, `dsjc500.5`, `dsjc500.9`: grafos aleatórios padrão `dsjc`$n.d$ que possuem n vértices e d probabilidade de quaisquer dois vértices formarem uma aresta;
- `dsjr500.1`, `dsjr500.1c` e `dsjr500.5`: grafos aleatórios geométricos `dsjr`$n.d$ que foram gerados escolhendo n pontos em um quadrado e configurando arestas entre pares de vértices com distância menor que d. Uma letra 'c' ao final do nome indica que o grafo é complemento do grafo geométrico correspondente;
- `flat300_20_0`, `flat300_26_0` e `flat300_28_0`: grafos `flat`n_χ_0 que possuem n vértices e número cromático χ conhecido;

[5]Disponível em `http://mat.gsia.cmu.edu/COLOR/instances.html`, acessado em novembro de 2011.

- `fpsol2.i.1`, `fpsol2.i.2`, `fpsol2.i.3`, `inithx.i.1`, `inithx.i.2` e `inithx.i.3`: grafos de interferência criados por um alocador de registradores para problemas reais;
- `le450_15d`, `le450_25c` e `le450_25d`: grafos `le450_`χl que possuem sempre 450 vértices e número cromático χ conhecido.

A Tabela 1 apresentas mais detalhes de cada instância utilizada na avaliação. Nesta tabela a segunda coluna apresenta o par χ/k^* (com '?' caso χ não seja conhecido), onde k^* é o valor da melhor solução encontrada até o momento.

Tabela 1. Características de cada instância.

Instância	(χ/k^*)	Vértices	Arestas	Densidade
dsjc500.1	(?/12)	500	12485	0,0999
dsjc500.5	(?/48)	500	62624	0,5020
dsjc500.9	(?/126)	500	112437	0,9013
dsjr500.1	(12/12)	500	3555	0,0285
dsjr500.1c	(84/84)	500	121275	0,9721
dsjr500.5	(122/122)	500	58862	0,4718
flat300_20_0	(20/20)	300	21375	0,4766
flat300_26_0	(20/20)	300	21633	0,4823
flat300_28_0	(28/28)	300	21695	0,4837
fpsol.i.1	(65/65)	496	11654	0,0949
fpsol.i.2	(30/30)	451	8691	0,0856
fpsol.i.3	(30/30)	425	8688	0,0964
inithx.i.1	(54/54)	864	18707	0,0502
inithx.i.2	(31/31)	645	13979	0,0673
inithx.i.3	(31/31)	621	13969	0,0726
le450_15d	(15/15)	450	127650	0,1658
le450_25c	(25/25)	450	17343	0,1717
le450_25d	(25/25)	450	17425	0,1725

Nestes experimentos cada instância foi calibrada com o objetivo de encontrar os melhores valores para os parâmetros: *quantidade de formigas*, α, β, ρ e *quantidade de ciclos da busca local*. Tais parâmetros são apresentados na Tabela 2. Para cada experimento, foram efetuadas 10 execuções. A versão paralela foi executada com 2, 4 e 8 *threads*.

Para garantir que *PColorAnt*$_3$*-RT* tivesse o mesmo comportamento que *ColorAnt*$_3$*-RT* ambos foram parametrizados para utilizarem os mesmos conjuntos de sementes nos geradores de números aleatórios. Cada *thread* utiliza uma ou mais sequências de números aleatórios independente(s) das demais. Dessa forma, para cada execução uma semente inicial foi utilizada

Tabela 2. Parâmetros utilizados por cada instância.

Instância	Formigas	α	β	ρ	Ciclos da Busca Local
dsjc500.1	320	11	19	0.7	1700
dsjc500.5	320	2	12	0.7	1800
dsjc500.9	400	12	2	0.7	1950
dsjr500.1	40	1	1	0.0	50
dsjr500.1c	80	1	4	1.0	1450
dsjr500.5	160	1	20	0.9	1750
flat300_20_0	24	1	1	0.0	100
flat300_26_0	400	1	5	0.6	1950
flat300_28_0	480	1	11	0.6	1250
fpsol.i.1	24	4	1	0.0	50
fpsol.i.2	24	17	1	0.0	50
fpsol.i.3	24	18	1	0.0	50
inithx.i.1	24	16	2	0.0	50
inithx.i.2	24	6	3	0.0	50
inithx.i.3	24	7	3	0.0	50
le450_15d	480	2	16	0.2	1700
le450_25c	240	1	19	0.0	1350
le450_25d	440	1	18	0.2	1700

para gerar outras 8 sementes para as sequências. Cada *thread* utiliza então 8/t sequências de números aleatórios, sendo t a quantidade de *threads* – na versão sequencial considera-se $t = 1$. As mesmas 10 sementes iniciais foram utilizadas para as 10 execuções de todos os experimentos. Desta forma foi possível reproduzir os mesmos resultados, e assim fazer uma comparação justa entre as versões.

6.2 Qualidade dos Resultados

Experimentos com determinada entrada (incluindo as sementes) com variação ou não na quantidade de *threads* possuem exatamente o mesmo comportamento e resultado, à exceção do tempo de processamento. Assim, a Tabela 3 apresenta unificadamente os resultados obtidos por *ColorAnt*$_3$*-RT* e *PColorAnt*$_3$*-RT* em termos de qualidade de solução. Nesta tabela a primeira coluna apresenta o nome do grafo, a segunda o contém o par χ/k^* e a terceira coluna os melhores valores de k encontrados. Nesta tabela ainda são apresentados a quantidade de execuções com sucesso, a quantidade média de conflitos (`Conflitos`) e a quantidade média do total de ciclos (`Ciclos`) do algoritmo. As médias apresentadas são referentes às 10 execuções. Os tempos de execução são apresentados e discutidos na Subseção

6.3.

Tabela 3. Aproximações obtidas.

Instância	(χ/k^*)	k	Sucessos	Conflitos	Ciclos
dsjc500.1	(?/12)	13	(10/10)	0,00	5,5
dsjc500.5	(?/48)	51	(4/10)	2,70	378,30
dsjc500.9	(?/126)	128	(1/10)	3,50	466,90
dsjr500.1	(12/12)	**12**	(10/10)	0,00	35,40
dsjr500.1c	(84/84)	85	(10/10)	0,00	12,80
dsjr500.5	(122/122)	123	(10/10)	0,00	14,50
flat300_20_0	(20/20)	**20**	(9/10)	21,10	167,40
flat300_26_0	(26/26)	29	(1/10)	16,10	426,90
flat300_28_0	(28/28)	32	(8/10)	0,40	189,00
fpsol.i.1	(65/65)	**65**	(7/10)	0,40	112,60
fpsol.i.2	(30/30)	**30**	(4/10)	0,60	221,50
fpsol.i.3	(30/30)	**30**	(8/10)	0,20	73,60
inithx.i.1	(54/54)	**54**	(6/10)	0,40	147,20
inithx.i.2	(31/31)	**31**	(5/10)	0,50	185,10
inithx.i.3	(31/31)	**31**	(6/10)	0,50	185,00
le450_15d	(15/15)	**15**	(10/10)	0,0	19,10
le450_25c	(25/25)	26	(4/10)	1,90	535,70
le450_25d	(25/25)	26	(5/10)	1,60	364,30

Os melhores resultados foram obtidos para os grafos *fpsol* e *inithx*, para todas estas instâncias *ColorAnt-RT*[6] encontrou soluções ótimas. Além destes dois grupos, *ColorAnt-RT* encontrou as soluções ótimas para: `dsjr500.1`, `flat300_20_0` e `le450_15d`. A distância para com as soluções ótimas ou melhores conhecidas (OMC) são: 0% para os grafos *fpsol* e *inithx*; 0,67% para os grafos *geométricos*; 2,67% para os grafos *le_450*; 5,39% para os grafos *aleatórios*; e 8,61% para os grafos *flat*. Estes resultados demonstram que o algoritmo desenvolvido é ideal para ser aplicado à grafos de interferência e ruim para grafos *aleatórios* e *flat*.

Analisando as características dos grafos não é possível identificar um padrão que descreva os resultados encontrados. Contudo, é possível perceber que as piores soluções foram obtidas para grafos com 300 ou 500 vértices. Além disto, era esperado que o aumento da densidade do grafo ocasionasse uma perda na qualidade dos resultados. Porém, os resultados demonstram que a densidade do grafo não influenciou na qualidade dos resultados. A instância `dsjc500.1` cuja densidade é 0,0999 possui uma distância maior (em percentual)

[6]Englobando *ColorAnt*$_3$*-RT* e *PColorAnt*$_3$*-RT*.

do que aquela obtida pela instância `dsjr500.1c`, cuja densidade é de 0,9721. Outro exemplo desta situação é o que ocorre com os gráficos *aleatórios*, nos quais quanto maior a densidade maior é a distância entre os resultados obtidos e as soluções OMC.

As instâncias *fpsol* e *inithx*, bem como `dsjr500.1`, `flat_300_20_0` e `le450_25c` foram parametrizadas com o valor $\rho = 0$, como mostra a Tabela 2. Assim, na iteração, as formigas constroem suas soluções com base no feromônio decorrente apenas do depósito realizado na iteração imediatamente anterior, que pode ter sido feito utilizando s' ou s^*. Desse modo, 0 é atribuído a todas as posições da matriz antes do depósito de feromônio em cada iteração, fazendo com que o feromônio referente a inicialização e histórico de atualizações influenciem as decisões das formigas na iteração atual apenas por meio do que já influenciaram na construção de s^* e s'. Portanto, há um grande favorecimento à exploração do espaço de busca. Tal exploração se mostrou benéfica, como pode ser visto pelos resultados: *fpsol*, *inithx*, `dsjr500.1` e `flat_300_20_0` tem 0% de distância para as soluções OMC, com média de 6,875 execuções com sucesso por instância; ao passo que `le450_25c` tem 4% de distância para soluções OMC. Portanto, mesmo a calibragem do parâmetro ρ sendo diferente do que normalmente ocorre para algoritmos ACO, excelentes resultados foram alcançados, à exceção de `le450_25c`.

Pelas características dos algoritmos ACO não existem garantias que um determinado resultado será encontrado em todas execuções, como pode ser observado pela coluna de execuções com sucesso da Tabela 3. Contudo, é possível observar que *ColorAnt-RT* é um bom algoritmo para reduzir a quantidade de conflitos, sendo que não obteve resultados satifatórios apenas para as instâncias `flat300_20_0` e `flat300_26_0`.

ColorAnt-RT foi calibrado para executar no máximo por 10 horas, exceto se uma solução sem conflitos fosse encontrada em um determinado ciclo ou se o algoritmo parasse de convergir. Nenhuma execução parou por exceder ao limite de tempo, mas ou por achar uma solução sem conflito ou por não convergir. Independente do critério de parada, existem instâncias que necessitam de uma quantidade expressiva de ciclos de execução do algoritmo, para que uma solução seja obtida. Como esta quantidade de ciclos independe da versão do algoritmo (*ColorAnt*$_3$*-RT* ou *PColorAnt*$_3$*-RT*), devido ambas utilizarem a mesma calibragem, a tendência é que o tempo de execução seja reduzido a medida que o custo do algoritmo seja distribuído por vários processadores. Este custo não está relacionado ao tamanho da entrada, mas ao método construtivo k-ANTCOL.

A quantidade de chamadas a este método consome entre 80% e 90% do tempo de execução de *ColorAnt*$_3$*-RT*. Portanto, uma estratégia para diminuir este tempo é distribuir esta quantidade de chamadas pelos diversos processadores. Em outras palavras, distribuir a quantidade de formigas pelos diversos processadores.

6.3 Desempenho de *PColorAnt$_3$-RT*

Idealmente, uma aplicação que leva um tempo `T` para ser executada em um único processador, pode ser executada em um tempo `T/P` para `P` processadores. Porém, existem muitas razões na qual o ideal é raramente alcançado. Primeiro, existe a necessidade de identificar pelo menos as `P` divisões do paralelismo. Segundo, a programação paralela geralmente introduz um *overhead* que não está presente em programação sequencial. Terceiro, mesmo para uma aplicação bem escrita, o desafio de alcançar um tempo `T/P` aumenta quando o valor de `P` aumenta. Desta forma, paralelismo e desempenho estão relacionados mas não são as mesmas coisas.

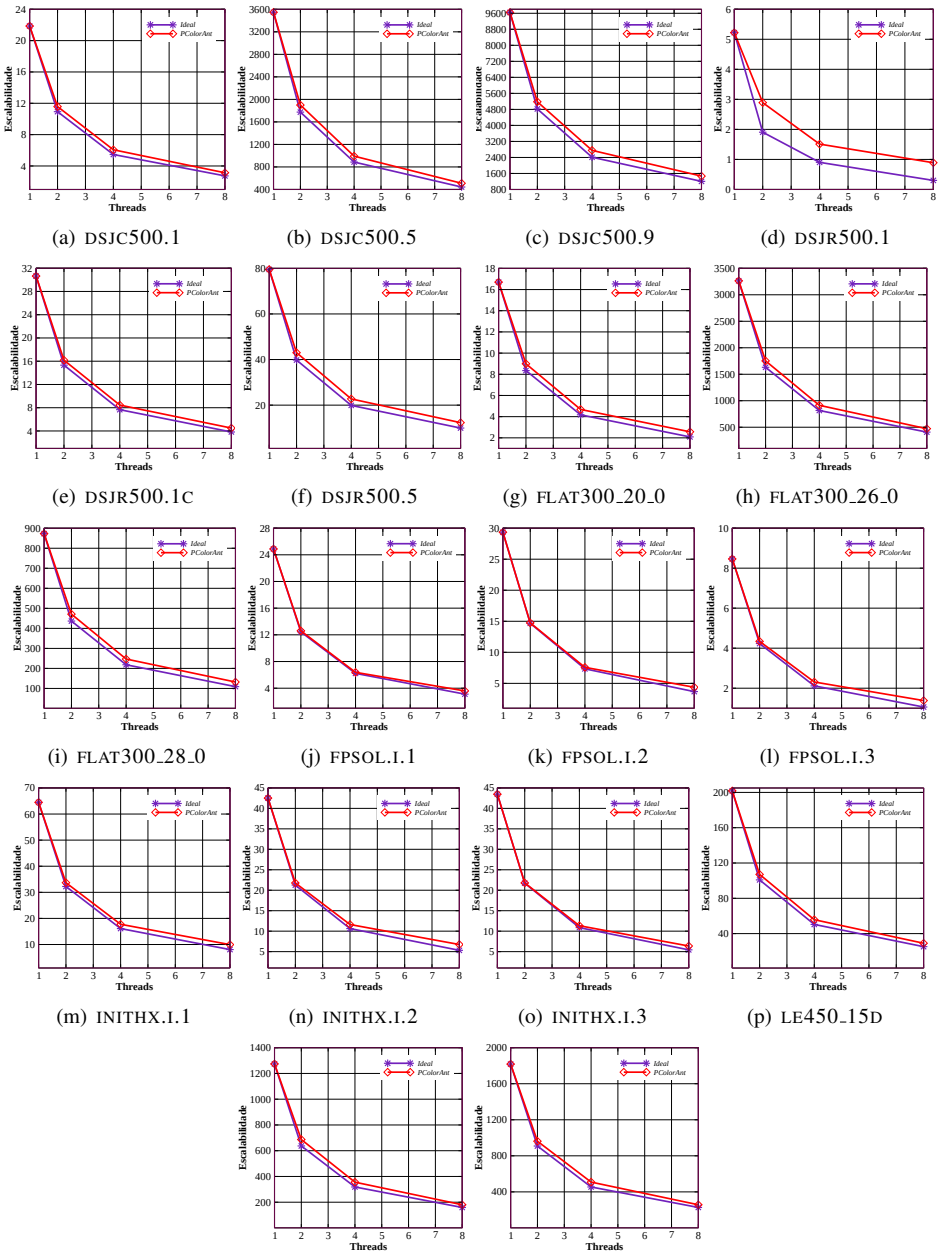
ColorAnt$_3$-RT, embora encontre bons valores para k, possui um alto tempo de execução. A proposta no desenvolvimento de *PColorAnt$_3$-RT* é manter a qualidade dos resultados obtidos por *ColorAnt$_3$-RT*, porém em um menor tempo de execução.

As características dos algoritmos ACO sugerem uma abordagem diferente da tradicional tanto na paralelização da aplicação, quanto na abordagem utilizada para analisar os resultados. Entre tais características é possível enumerar:

1. A paralelização não particiona os dados de entrada, devido as caracteríticas do algoritmo. A estratégia utilizada na paralelização é a distribuição das N formigas pelos processadores (*threads*).
2. É possível que a cada execução seja gerado um resultado diferente, o que pode ser observado na coluna de execução com sucesso da Tabela 3.
3. Mesmos resultados podem ser obtidos em tempos distintos de execuções. Em geral a variação entre o tempo de execução é bem significativa, chegando a um desvio padrão de 1762, considerando apenas execuções bem sucedidas (nenhum conflito) e não o espaço amostral de 10 execuções.
4. A aleatoriedade utilizada pelo algoritmo.

A Figura 2 apresenta a escalabilidade de *PColorAnt$_3$-RT*, para cada instância utilizada na avaliação. Nesta figura, o tempo de execução é apresentado em segundos. E *PColorAnt$_3$-RT* foi executado para 2, 4 e 8 *threads*. Assim, o objetivo é determinar o ponto no qual o modelo utilizado para desenvolver a versão paralela de *ColorAnt-RT* começa a ter uma perda de desempenho, além de investigar a redução do tempo de execução.

Destes experimentos, é possível observar que a aceleração (*speedup*) é similar para todas as instâncias. A variação na aceleração foi entre 1,81 e 1,99 para duas *threads*; 3,50 e 3,90 para quatro *threads*; e 5,88 e 7,07 para oito *threads*. Contudo, apenas para as instâncias `le450_25c` e `le450_25d` a aceleração alcançou um valor superior a 7. Excluindo estas



(a) DSJC500.1 (b) DSJC500.5 (c) DSJC500.9 (d) DSJR500.1

(e) DSJR500.1C (f) DSJR500.5 (g) FLAT300_20_0 (h) FLAT300_26_0

(i) FLAT300_28_0 (j) FPSOL.I.1 (k) FPSOL.I.2 (l) FPSOL.I.3

(m) INITHX.I.1 (n) INITHX.I.2 (o) INITHX.I.3 (p) LE450_15D

(q) LE450_25C (r) LE450_25D

Figura 2. Escalabilidade obtida por *PColorAnt*$_3$*-RT*.

duas instâncias, a aceleração não ultrapassa 6,99. Estes resultados demonstram que as primitivas de sincronização começam a ocasionar um perda de desempenho para uma configuração com 8 *threads* e possivelmente superior. Apenas para a instância `dsjr500.1`, *PColorAnt*$_3$*-RT* obteve um desempenho diferente daquele obtido para as outras instâncias. Um fator que influenciou esta perda de desempenho para esta instância, foi o fato desta possuir um tempo de execução pequeno.

Em geral é esperado que uma aplicação com pontos de sincronização não atinja o *valor ideal*, exceto para casos nos quais exista uma sobreposição entre computação e comunicação. Porém, os resultados demonstraram que existem situações que os resultados obtidos estão bem próximos deste valor, como é o caso das instâncias `fpsol` e `inithx`, para uma configuração com 2 ou 4 *threads*.

Analisando o tipo dos grafos é possível perceber que *PColorAnt*$_3$*-RT* é capaz de reduzir significativamente o tempo de execução do algoritmo de coloração. Um ponto importante a ser destacado é o fato da aceleração obtida por instâncias *menores* – as quais são grafos de interferência criados por um alocador de registradores para problemas reais – ser similar aquela obtida por instâncias *maiores*. Isto demonstra o potencial de *PColorAnt*$_3$*-RT* para ser utilizado em diferentes contexto, por exemplo alocação de registradores.

Como um determinado padrão de desempenho não agrupa um grupo de instâncias, ainda não é possível afirmar que as características de um determinado grupo influenciou nos resultados, por exemplo, ocasionando um custo mais alto para a computação e consequentemente diminuindo o *overhead* de comunicação. Em trabalhos futuros serão analisadas as características de cada instância (grafo), com o objetivo de identificar se estas influenciam no ganho ou perda de desempenho. Contudo, é possível obter uma análise mais detalhada dos resultados apresentados pelos gráficos de escalabilidade analisando o impacto de *PColorAnt*$_3$*-RT* no hardware utilizado.

6.4 Análise Detalhada de *PColorAnt*$_3$*-RT*

Para uma análise mais detalhada das acelerações obtidas por *PColorAnt*$_3$*-RT*, foram coletadas, durante a execução do algoritmo, informações do perfil de hardware, a saber: *quantidade de instruções executadas*, *quantidade de ciclos*, *acessos à hierarquia de cache*, e *falhas nos acessos à hierarquia de cache*. Tais informações foram coletadas pela ferramenta PAPI [26] e são apresentadas na Tabela 4. Nesta tabela, os dados apresentados para *PColorAnt*$_3$*-RT* representam o percentual de redução.

Em uma execução ideal seria esperado que a quantidade de instruções e acessos à hierarquia de cache fossem distribuídas de forma igualitária entre as diversas *threads*. Contudo, isto é somente encontrado em aplicações regulares, nas quais os dados são distribuídos pelas diversas *threads* de forma igualitária e cujo padrão de acesso à memória não é irregular.

Tabela 4. Impacto de *ColorAnt*$_3$*-RT* e *PColorAnt*$_3$*-RT* no Hardware.

Perfil de Instruções e Ciclos								
Instância	*ColorAnt*$_3$*-RT*		*PColorAnt*$_3$*-RT*					
			2-Threads		*4-Threads*		*8-Threads*	
	Instruções (10^9)	*Ciclos* (10^9)	*Instruções* (%)	*Ciclos* (%)	*Instruções* (%)	*Ciclos* (%)	*Instruções* (%)	*Ciclos* (%)
dsjc500.1	60,93	43,27	49,80	48,69	74,84	73,67	87,31	86,99
dsjc500.5	12237,62	7052,06	49,95	48,46	74,98	73,96	87,47	87,12
dsjc500.9	32894,17	19162,32	50,05	48,97	75,02	73,83	87,52	86,59
dsjr500.1	14,33	10,31	49,60	45,08	74,38	72,50	86,83	85,10
dsjr500.1c	95,55	60,51	49,51	48,87	74,62	74,37	87,14	87,23
dsjr500.5	214,15	157,77	50,15	48,47	74,91	73,80	87,19	86,90
flat300_20_0	58,22	33,15	49,78	47,35	74,73	73,24	87,23	85,90
flat300_26_0	12439,85	6489,60	50,08	48,39	75,05	73,93	87,53	87,05
flat300_28_0	2964,88	1734,01	49,27	48,20	75,00	73,58	87,00	86,30
fpsol.i.1	63,23	49,22	50,01	50,02	74,68	74,95	87,18	86,65
fpsol.i.2	76,28	58,06	49,74	50,14	74,68	75,00	86,79	86,48
fpsol.i.3	22,70	16,57	49,17	48,82	74,53	74,04	87,04	85,73
inithx.i.1	167,05	127,89	49,57	49,34	74,42	73,13	86,89	85,73
inithx.i.2	113,31	84,40	49,74	49,17	74,65	73,27	86,80	85,12
inithx.i.3	113,24	86,10	49,60	49,91	74,64	74,32	86,79	86,32
le450_15d	743,34	401,15	50,00	48,75	74,93	74,18	87,46	87,19
le450_25c	3749,65	2532,53	49,99	47,68	74,99	73,42	87,48	86,92
le450_25d	5216,13	3605,67	49,97	48,58	74,98	73,50	87,48	86,86
Perfil de Acessos a Hierarquia de Cache								
Instância	*ColorAnt*$_3$*-RT*		*PColorAnt*$_3$*-RT*					
			2-Threads		*4-Threads*		*8-Threads*	
	Acessos (10^9)	*Falhas* (10^9)	*Acessos* (%)	*Falhas* (%)	*Acessos* (%)	*Falhas* (%)	*Acessos* (%)	*Falhas* (%)
dsjc500.1	50,03	1,26	49,83	48,10	74,73	73,50	87,29	86,81
dsjc500.5	8516,36	437,43	49,95	49,52	74,93	74,50	87,45	87,27
dsjc500.9	18377,31	1359,76	49,94	50,01	74,98	74,77	87,46	87,40
dsjr500.1	10,95	0,38	49,10	47,39	74,29	72,72	86,79	85,95
dsjr500.1c	60,81	3,24	49,98	48,39	74,77	73,32	87,29	85,94
dsjr500.5	156,97	7,49	49,92	47,98	74,76	73,13	87,23	86,11
flat300_20_0	45,03	0,52	49,63	44,71	74,58	71,61	87,01	85,49
flat300_26_0	8592,54	328,61	50,01	50,75	74,93	74,05	87,46	87,06
flat300_28_0	2185,04	77,76	49,15	48,17	74,89	73,80	86,92	86,30
fpsol.i.1	43,10	1,18	49,85	49,93	74,51	74,66	86,89	86,56
fpsol.i.2	49,86	1,44	49,46	49,05	74,26	73,80	86,74	86,30
fpsol.i.3	14,85	0,42	49,79	48,99	74,12	73,54	86,62	86,50
inithx.i.1	110,47	4,09	49,51	49,04	74,21	73,57	86,66	85,48
inithx.i.2	72,18	2,59	49,53	49,10	74,20	73,14	86,46	85,49
inithx.i.3	72,24	2,38	49,23	49,23	74,10	73,53	86,64	85,64
le450_15d	537,67	5,21	49,97	49,23	74,91	72,97	87,41	86,03
le450_25c	3030,65	91,13	49,78	49,21	74,89	74,30	87,43	87,07
le450_25d	4239,33	125,92	49,91	49,05	74,88	74,18	87,43	87,04

Algoritmos ACO podem ser classificados como aplicações irregulares, não relacionado com os acessos aos dados como em aplicações tradicionais, mas sim em relação ao modo como uma solução é encontrada. Eles utilizam alguns dados aleatórios, que são a semente e a sequência de números aleatórios utilizadas a cada execução, as quais influenciam na convergência do algoritmo. Consequentemente, isto pode ocasionar um padrão irregular na execução de cada *thread*. Contudo, como mencionado anteriormente foram utilizados os menos conjuntos de sementes para *ColorAnt*$_3$*-RT* e *PColorAnt*$_3$*-RT* para que esta irregularidade fosse controlada e assim uma comparação mais justa fosse realizada.

Analisando os dados apresentados na Tabela 4 é possível concluir que tanto a redução na quantidade de instruções executadas, quanto nos acessos à hierarquia de *cache* estão próximos dos valores ideais, que são 50%, 75% e 87,5% respectivamente para 2, 4 e 8 *threads*, para que o valor ideal fosse alcançado.

Estes resultados demonstram que para obter um bom desempenho, é necessário uma redução significativa tanto na quantidade de instruções e ciclos, como também na quantidade de acessos e falhas à hierarquia de *cache*. Taxas de redução distantes para estes dois componentes ou um padrão irregular para o mesmo componente ocasionam uma perda desempenho. Este é o caso, por exemplo de `dsjr500.1` e `fpsol.i.3`. Por outro lado, estes resultados demonstram que as instâncias `le450_25c` e `le450_25d` obtiveram os melhores resultados, para uma configuração com 8 *threads*, devido ao fato dos percentuais serem os mais próximos do ideal. Em síntese, uma distância menor entre os percentuais de redução ocasiona um desempenho superior aquele cuja distância entre os percentuais é maior.

Vale ressaltar que embora os dados apresentados na Tabela 4 sejam valores bem próximos para todas as instâncias, uma pequena variação no percentual de redução pode ocasionar um impacto considerável no resultado final. Isto pelo fato dos valores de perfil estarem na casa dos trilhões.

Em síntese, os resultados demonstram que a escalabilidade obtida é decorrente de dois fatores, a saber:

1. Uma boa distribuição da quantidade de instruções, ciclos e acessos à hierarquia de *cache*; e

2. Ao custo do método construtivo k-ANTCOL ser superior ao custo da sincronização.

Como mencionado anteriormente, *PColorAnt*$_3$*-RT* possui chamadas a instruções que realizam exclusão mútua além de barreiras. Estas chamadas de sincronização ocasionam um maior *overhead* para uma execução com 8 *threads*. Contudo, mesmo com a existência deste *overhead PColorAnt*$_3$*-RT* possui uma boa escalabilidade.

7 Considerações Finais

Este artigo apresentou uma estratégia paralela para diminuir o tempo de execução de *ColorAnt*$_3$*-RT*, um algoritmo baseado em colônia de formigas artificiais aplicado ao problema de coloração de grafos, sem degradar a qualidade das soluções até então por ele obtidas. A estratégia utilizada na paralelização é a distribuição das N formigas pelos processadores (*threads*), que proporcionou o desenvolvimento de uma nova versão do algoritmo, denominada *PColorAnt*$_3$*-RT*. Os resultados demonstram que *ColorAnt*$_3$*-RT* e *PColorAnt*$_3$*-RT* possuem a mesma qualidade de solução, o que ocorre pelo fato de utilizarem as mesmas sementes para as execuções.

A análise de tempo demonstrou que o uso de programação paralela pode ser uma excelente estratégia para ser empregada em algoritmos ACO, pois é capaz de reduzir significantemente o tempo de execução de *ColorAnt*$_3$*-RT*, bem como foi possível observar que a aceleração cai levemente em relação à ideal a medida que a quantidade de processadores aumenta, indicando que o algoritmo possui boa escalabilidade.

Por meio de análise das informações do perfil de hardware foi possível verificar que uma boa escalabilidade decorre de uma boa distribuição da quantidade de instruções, ciclos e acessos à hierarquia de *cache*. O fato de o custo da construção da solução de uma formiga ser superior ao custo da sincronização é base para a escalabilidade, que se mantém mesmo com o *overhead* produzido pelas chamadas de sincronização.

PColorAnt$_3$*-RT* é a melhor escolha quando o tempo de execução é um fator crítico, por exemplo para reduzir o tempo de execução do compilador que utiliza *ColorAnt*$_3$*-RT* como uma das fases do alocador de registradores. De fato está será a apróxima investigação a ser realizada. Além desta, outra investigação futura compreende implementar novas versões paralelas, com o objetivo de analisar qual a melhor arquitetura paralela para *ColorAnt*$_3$*-RT* e posteriormente analisar as características de cada instância (grafo) com o objetivo de identificar se estas influenciam no ganho ou perda de desempenho.

Referências

[1] APPEL, A. W. *Modern Compiler Implementation in C*. Cambridge University Press, New York, NY, EUA, 1998.

[2] BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques* (October 2008).

[3] BONDY, J. A., AND MURTY, U. S. R. *Graph Theory*, vol. 244 of *Graduate Texts in Mathematics*. Springer, 2008.

[4] BRÉLAZ, D. New Methods to Color the Vertices of a Graph. *Communications of the ACM 22*, 4 (1979), 251–256.

[5] BUI, T. N., NGUYEN, T., AND RIZZO, JR., J. R. Parallel Shared Memory Strategies for Ant-based Optimization Algorithms. In *Proceedings of the Conference on Genetic and Evolutionary computation* (2009), pp. 1–8.

[6] COMELLAS, F., AND OZÓN, J. An Ant Algorithm for the Graph Colouring Problem. In *International Workshop on Ant Colony Optimization* (1998), pp. 151–158.

[7] COSTA, D., AND HERTZ, A. Ants Can Colour Graphs. *The Journal of the Operational Research Society 48*, 3 (1997), 295–305.

[8] CULBERSON, J. C., AND LUO, F. Exploring the k-colorable Landscape with Iterated Greedy. In *Dimacs Series in Discrete Mathematics and Theoretical Computer Science* (1995), American Mathematical Society, pp. 245–284.

[9] DORIGO, M., BIRATTARI, M., AND STÜTZLE, T. Ant Colony Optimization - Artificial Ants as a Computational Intelligence Technique. *IEEE Computational Intelligence Magazine 1*, 4 (2006), 28–39.

[10] DORIGO, M., AND KRZYSZTOF, S. An Introduction to Ant Colony Optimization. *IRIDIA Technical Report Series* (2006).

[11] DORIGO, M., MANIEZZO, V., AND COLORNI, A. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B 26*, 1 (1996), 29–41.

[12] GALINIER, P., AND HAO, J.-K. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization 3*, 4 (1999), 379–397.

[13] GEORGE, L., AND APPEL, A. W. Iterated Register Coalescing. *ACM Transactions on Programming Languages and Systems 18* (1996), 300–324.

[14] HERTZ, A., AND ZUFFEREY, N. A New Ant Algorithm for Graph Coloring. In *Workshop on Nature Inspired Cooperative Strategies for Optimization NICSO* (2006), pp. 51–60.

[15] HERTZ, A., AND ZUFFEREY, N. Vertex Coloring Using Ant Colonies. In *Artificial Ants: From Collective Intelligence to Real-life Optimization and Beyond* (France, 2010), N. Monmarché, F. Guinand, and P. Siarry, Eds., Wiley.

[16] JOHNSON, D. S., AND TRICK, M. A. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. DIMACS series in discrete mathematics and theoretical computer science. American Mathematical Society, Providence, RI, EUA, 1996.

[17] LEIGHTON, F. T. A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of Research of the National Bureau of Standards 84*, 6 (1979), 489–506.

[18] LINTZMAYER, C. N., MULATI, M. H., AND DA SILVA, A. F. Algoritmo Heurístico Baseado em Colônia de Formigas Artificiais ColorAnt2 com Busca Local Aplicado ao Problema de Coloração de Grafo. In *X Congresso Brasileiro de Inteligência Computacional* (2011).

[19] LINTZMAYER, C. N., MULATI, M. H., AND DA SILVA, A. F. Register Allocation with Graph Coloring by Ant Colony Optimization. In *XXX International Conference of the Chilean Computer Science Society* (2011).

[20] LINTZMAYER, C. N., MULATI, M. H., AND DA SILVA, A. F. RT-ColorAnt: Um Algoritmo Heurístico Baseado em Colônia de Formigas Artificiais com Busca Local para Colorir Grafos. In *Anais do Simpósio Brasileiro de Pesquisa Operacional* (2011).

[21] LINTZMAYER, C. N., MULATI, M. H., AND DA SILVA, A. F. Toward Better Performance of ColorAnt ACO Algorithm. In *XXX International Conference of the Chilean Computer Science Society* (2011).

[22] NICHOLS, B. *Pthreads Programming*. Oreilly, 1996.

[23] PLUMETTAZ, M., SCHINDL, D., AND ZUFFEREY, N. Ant Local Search and Its Efficient Adaptation to Graph Colouring. *Journal of the Operational Research Society 61*, 5 (2010), 819–826.

[24] RAHOUAL, M., HADJI, R., AND BACHELET, V. Parallel Ant System for the Set Covering Problem. In *Proceedings of the International Workshop on Ant Algorithms* (2002), pp. 262–267.

[25] SHAWE-TAYLOR, J., AND ZEROVNIK, J. Ants and Graph Coloring. In *International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA'01* (Berlin, Heidelberg, 2001), Springer, pp. 276–279.

[26] TERPSTRA, D., JAGODE, H., YOU, H., AND DONGARRA, J. Collecting Performance Data with PAPI-C. *Tools for High Performance Computing 2009*, 157–173.

[27] WOO, S. C., OHARA, M., TORRIE, E., SINGH, J. P., AND GUPTA, A. The SPLASH-2 Programs: Characterization and Methodological Considerations. *SIGARCH Computer Architecture News 23* (May 1995), 24–36.