UNIVERSIDADE DE COIMBRA FACULDADE DE CIÊNCIAS E TECNOLOGIA ***Departamento de Engenharia Informática***	**Trabalho Prático N.º 2** **Computação de Alto Desempenho** **2009/10 – 2º Semestre** MEI **Prazo: 21-5-2010**
<u>Nota</u>: A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador.	

Programação com MPI

Objetivos do Trabalho

Obtenção de competências na programação paralela em sistemas de memória distribuída.

Coloração de Grafos

Descrição do Problema

O problema da coloração de um grafo[1] consiste em atribuir cores aos seus vértices, tal que dois vértices vizinhos não partilhem a mesma cor. A Figura 1 ilustra alguns exemplos.

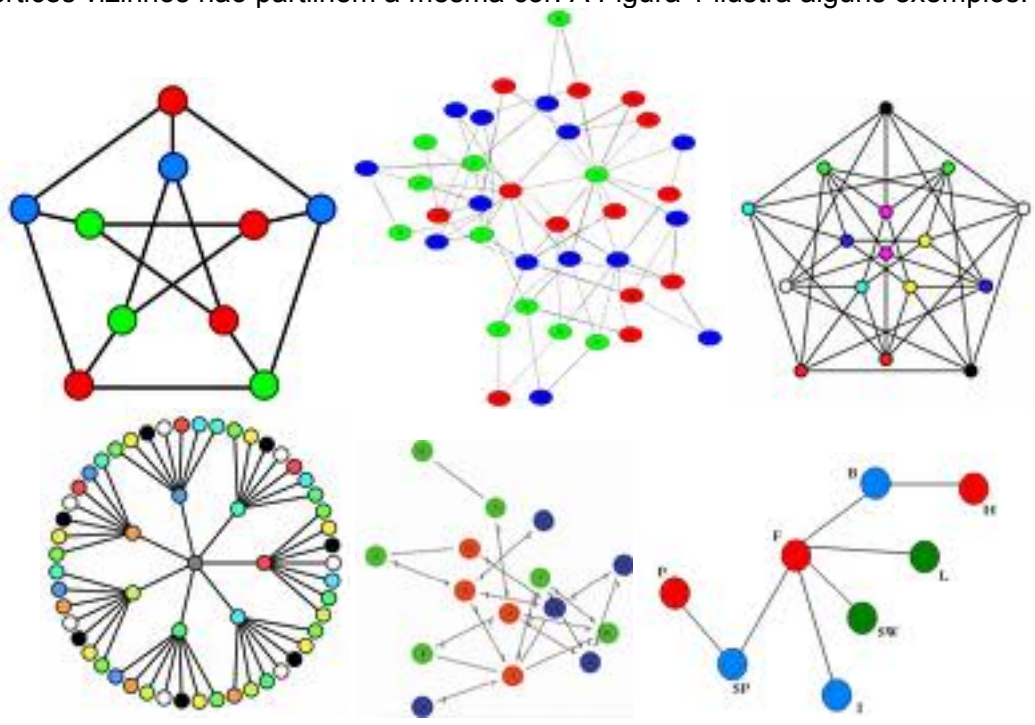

Figura 1 - Exemplos de coloração de grafos

[1] Seria mais preciso designarmos este problema por "coloração de vértices", mas a designação "coloração de grafos" é muito comum.

Note-se que a coloração de um grafo tem muitas soluções triviais. Podemos, por exemplo, atribuir uma cor diferente a cada vértice. No entanto, em geral, queremos soluções mais complexas do que esta. Em particular, uma das questões relacionadas é a de saber qual é o numero cromático de um grafo, i.e., qual o número mínimo de cores com que podemos colorir um grafo. Nalguns casos particulares, por exemplo em grafos planos[2], com os quais podemos representar as relações de fronteira de mapas geográficos, bastam apenas quatro cores para colorir os vértices. Infelizmente, em geral, o problema de determinar o número cromático de um grafo é NP-completo. Para um dado número cromático k, o problema de colorir o grafo com k cores, também é NP-completo.

O problema de coloração de grafos tem muitas aplicações práticas[3]. Por exemplo em algoritmos de escalonamento, na atribuição de larguras de banda a estações de rádio, ou atribuição de registos de uma CPU a um programa pelo compilador. Este problema aparece também em situações recreativas: o puzzle do Sudoku é representável como um grafo de 81 vértices, que temos colorir com 9 cores (números de 1 a 9).

Solução *Greedy*

Uma solução muito simples para o problema e que, nalguns casos poderá ser ótima é a "greedy" ou gananciosa. A ideia é percorrer todos os vértices por uma dada ordem (por exemplo do vértice 1 até n) e para cada um deles escolher a cor mais baixa que estiver disponível (por não ter sido atribuída a qualquer um dos vizinhos).

Solução Paralela

Uma abordagem muito simples para o programa paralelo, consiste em experimentar diferentes ordens na solução *greedy*, na medida em que há uma ordenação dos nós (infelizmente não sabemos qual), que permite utilizar o número óptimo de cores.

Para além desta possibilidade, os alunos deverão usar outros algoritmos mais complexos. Para isso, convém introduzirmos a noção de *conjunto independente*. Dado um grafo G=(V,E), um conjunto de vértices, $I \subseteq V$, é independente se nenhum par de vértices de I estiver conectado por uma aresta. Este conjunto é *maximal* se a adição de qualquer outro vértice de V tornar o conjunto não independente. O conjunto independente com o maior número de vértices é designado de *conjunto independente máximo*.

Com base neste conceito, ilustramos de seguida um algoritmo que esboça uma solução paralela para o problema da coloração [JP95].

$V' \leftarrow V$
Enquanto $V' \neq \emptyset$:
  Escolher um conjunto independente $I \subseteq V'$
  Colorir I em paralelo
  $V' \leftarrow V' \setminus I$

O problema aqui será o de determinar um conjunto independente em paralelo. Podemos, por exemplo, atribuir pesos aleatórios aos vértices do grafo. Depois, os vértices que tiverem um peso mais elevado do que qualquer um dos seus vizinhos farão parte do conjunto independente.

Nesta linha, vamos considerar um algoritmo paralelo, onde começamos por assumir que há um processador para cada vértice. Cada processo determina o peso do seu vértice, envia-o para os processos responsáveis pelos vértices vizinhos e espera pelas mensagens

[2] Um grafo é plano se puder ser representado num plano sem interseções de arestas.
[3] Ver http://en.wikipedia.org/wiki/Graph_coloring.

recíprocas. O peso de um vértice e o peso dos vizinhos determina qual a ordem pela qual cada processo pode colorir o seu vértice. Por exemplo, se um vértice tem 3 vizinhos com um peso mais alto, o processo tem de esperar pelas cores desses 3 vizinhos, antes de colorir o seu vértice. No algoritmo que apresentamos de seguida, designado por Jones e Plassmann [JP93], o peso do vértice υ é $\rho(\upsilon)$, e a cor é $\sigma(\upsilon)$.

```
Escolher ρ(υ)
n-wait = 0
send-queue = ∅
Para cada ϖ ∈ adj(υ):
  Enviar ρ(υ) para o processo responsável por ϖ
  Receber ρ(ϖ)
  Se (ρ(ϖ) > ρ(υ)) então n-wait=n-wait+1
  Senão send-queue←send-queue ∪ {ϖ}
n-recv=0
Enquanto (n-recv < n-wait)
  Receber σ(ϖ)
  n-recv=n-recv+1
Escolher a cor σ(υ) consistente com as cores dos vizinhos
Para cada ϖ ∈ send-queue:
  Enviar σ(υ) ao processo responsável por ϖ
```

Infelizmente, na prática, não é expectável que haja um processo por vértice, sendo necessário que um processo trate de múltiplos vértices. Será necessário, por isso resolver esse problema. Mais uma vez, os alunos poderão recorrer ao trabalho de Jones e Plassmann [JP93].

Gerador de Grafos

Os alunos terão acesso a um programa em python para gerar quatro tipos de grafos: anéis, aleatórios, *crown* e *grid*. Pela sua simplicidade não descreveremos o anel. Nos grafos aleatórios, existe uma aresta entre cada par de nós com probabilidade *p*, dada pelo utilizador. O aspeto dos grafos *crown* e *grid* pode ser observado, respetivamente, na Figura 2 e na

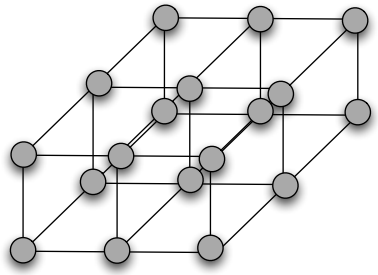

Figura 3. Em todos os grafos é possível alterar o número de vértices.

Os alunos podem ainda utilizar outros tipos de grafos nas suas experiências.

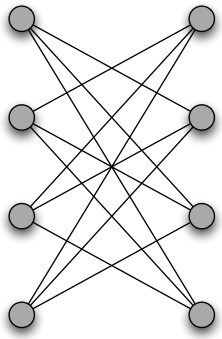

Figura 2 - Crown

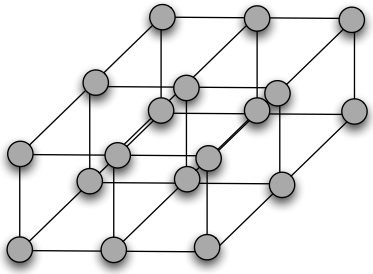

Figura 3 - Grid

Tarefas

Os alunos são encorajados a desenvolver não só este algoritmo, mas também outros que encontrem em [JP93], [A+95] ou noutra referência que consigam encontrar. É obrigatória a utilização do MPI.

Os alunos poderão utilizar os seus próprios recursos (por exemplo, um único computador, com dois cores) na realização das primeiras experiências. Posteriormente, será importante que recolham dados de desempenho com múltiplos processadores. Para isto terão de recorrer às salas de aulas, fora do horário normal das aulas. Solicito aos alunos, que utilizem os recursos das salas de forma contida. Por exemplo:

- Não deverão aceder remotamente
- Não deverão utilizar mais do que 6 computadores
- Não deverão ocupar uma sala mais do que 1 hora, se houver colegas à espera.

Se vier a ser necessário, criarei uma folha de inscrições para que os alunos possam escalonar os seus acessos às salas. **NOTA**: não estarei presente em parte da semana de entrega do trabalho, pelo que os alunos deverão resolver todos os problemas de acesso às salas antes disso.

Código Disponível e a Entregar

Os alunos terão acesso a código escrito em C, para fazer a leitura de grafos. Terão também acesso a alguns grafos de exemplo, bem como a uma solução sequencial de exemplo com o algoritmo *greedy*. A Figura 4 representa o formato dos dados de entrada. O primeiro número indica quantos nós tem o grafo, o segundo quantas arestas, seguindo-se as arestas propriamente ditas. Por exemplo, neste caso o ficheiro representa o grafo da
Figura 5. A função `read_graph_to_adjacency_matrix` converte o formato do ficheiro de entrada para uma matriz de adjacência, como a da Figura 6.

```
4
4
0 1
0 2
1 2
2 3
```

Figura 4 – Formato de entrada de dados

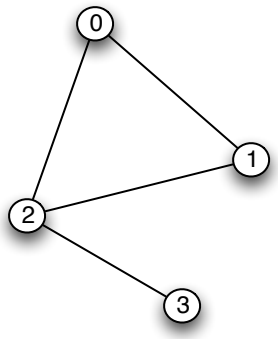

Figura 5 - Exemplo de um grafo

	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

Figura 6 - Matriz de adjacência

O resultado da execução de um algoritmo deverá ser o número de vertices do grafo, seguido do número de cores necessárias, seguido de cada uma das cores dos nós por ordem. As cores devem começar em 1. Por exemplo, neste caso, o resultado poderia ser o da Figura 7: há 4 nós, 3 cores, seguido das cores de cada um dos nós por ordem.

```
4
3
1
2
3
1
```

Figura 7 – Formato dos resultados

Os alunos devem entregar uma **Makefile**, ou equivalente, que permita compilar o código fonte e construir os executáveis de forma automática.

O código dado incluir referências para uma biblioteca de manipulação de conjuntos, que pode ser encontrada em http://www.mission-base.com/peter/source/pbl/doc/set.html, embora esteja também incluída no pacote.

Avaliação

A avaliação do trabalho será feita com base nos seguintes critérios:

- Correção. Os alunos deverão garantir que o seu algoritmo funciona corretamente. Isto significa apenas garantir que não há partilha de cores entre vértices vizinhos. É possível haver soluções diferentes para o mesmo grafo, incluindo soluções ótimas.
- Qualidade dos resultados. Serão privilegiados os algoritmos que utilizem o menor número possível de cores, desde que corretos.
- Desempenho. Os alunos deverão preocupar-se em minimizar o tempo de execução dos seus algoritmos. Os desempenhos dos diferentes trabalhos serão comparados entre si e pesados com a qualidade dos resultados.
- A análise do desempenho será um dos aspectos mais fundamentais na avaliação do trabalho. Serão valorizados os trabalhos que incluam comparações de desempenho de mais do que uma implementação. É importante realçar que a validade das otimizações a realizar deverá ser demonstrada com medições de desempenho. Para além disto é importante manter a clareza do código de forma a que o programa seja de fácil compreensão.

Relatório

O relatório deverá ser o mais curto possível, não deve incluir informação redundante (que já esteja suficientemente coberta no enunciado do trabalho, por exemplo) e deverá ser bem estruturado.

Um dos aspetos mais importantes do relatório reside na avaliação da qualidade (n.º de cores) e no desempenho do(s) algoritmo(s) paralelo(s). Quais os ganhos de desempenho da solução paralela relativamente ao tamanho do grafo de entrada, ou relativamente ao n.º de processos (máximo de 6), por exemplo.

Os alunos não se devem esquecer de indicar os nomes dos programas que criarem, para além da linha de comando necessária para os correr.

Prazo de Entrega

Ver cabeçalho. No total dos dois trabalhos os alunos podem atrasar-se 4 dias.

Bibliografia[4]

[JP93] Jones, M. T. and Plassmann, P. E. 1993. A parallel graph coloring heuristic. SIAM J. Sci. Comput. 14, 3 (May. 1993), 654-669. DOI= http://dx.doi.org/10.1137/0914041.
[A+95] J. R. Allwright , R. Bordawekar , P. D. Coddington , K. Dincer , C. L. Martin. 1995. A Comparison of Parallel Graph Coloring Algorithms.

Entrega do Trabalho

O código do trabalho, deverá ser guardado num ficheiro ZIP. Dentro desse ficheiro ZIP deverá ainda juntar um ficheiro **README.TXT** com TODA A INFORMAÇÃO NECESSÁRIA PARA O DOCENTE EXECUTAR, CONFIGURAR E TESTAR O SEU TRABALHO SEM A PRESENÇA DO(S) ALUNO(S). Trabalhos que não contenham esse ficheiro README com todas as instruções necessárias não serão avaliados. Trabalhos que não executem corretamente também não serão avaliados.

Deverá fazer o upload desse ficheiro ZIP no moodle: http://classes.dei.uc.pt

O relatório poderá ser incluído em formato PDF dentro do ficheiro ZIP. No entanto, é obrigatório entregar o relatório no cacifo de correio do docente da cadeira, até no máximo 1 dia útil após a entrega do trabalho. A chave de inscrição no moodle para CAD é cad$2010.

Bom Trabalho!

[4] Os alunos têm acesso a ambos os artigos conjuntamente com o enunciado do trabalho.