Eternity II

UFABC - MCZA020-13 - Programação Paralela

Emilio Francesquini - e.francesquini@ufabc.edu.br

2020.Q1

Prazo: 12/04/2020

1 O Jogo

O jogo[1] Eternity II[2] foi lançado em 2007 com a promessa de pagar US$ 2 milhões para a primeira pessoa que conseguisse apresentar uma solução completa. Contudo, até hoje, nenhuma solução correta foi descoberta e o prêmio continua sem ganhadores.

Eternity II é um quebra-cabeça clássico de casamento de bordas que envolve a colocação de 256 peças em um tabuleiro de $16x16$. Cada peça tem as suas bordas com diferentes combinações de formas e cores (que neste texto simplificamos para apenas cores). As peças devem ser colocadas no tabuleiro de maneira que todas as cores de suas bordas sejam uma correspondência às bordas das peças adjacentes. As bordas do tabuleiro são um caso com tratamento especial que apenas podem receber peças com bordas cinzas. Cada uma das peças pode ser rotacionada e tem, portanto, 4 possíveis colocações para cada posição do tabuleiro. Há 22 cores no total, sem incluir as bordas cinzas. No jogo original a peça do centro do tabuleiro é pré-determinada e algumas dicas sobre o posicionamento de algumas peças são dadas.

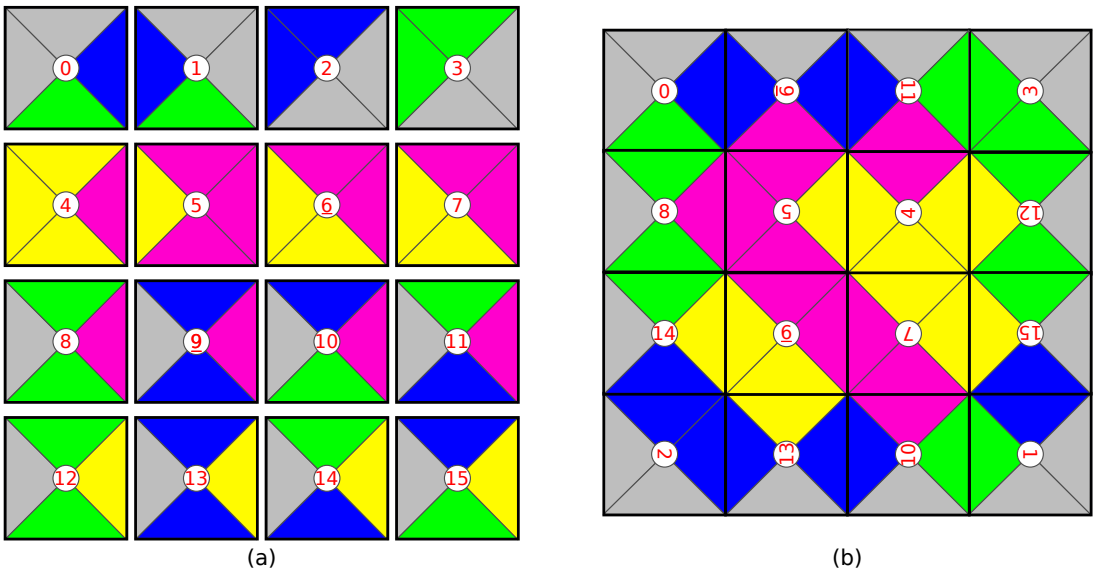

Figure 1: Na esquerda um conjunto de peças em um tabuleiro 4x4, na direita o tabuleiro resolvido. Note que algumas peças foram rotacionadas.

Este jogo foi criado para ser difícil de resolver através de métodos de força bruta e continua a ser intratável na sua configuração original. De fato o número total de combinações (assumindo

[1] Perdi.

[2] A descrição foi adaptada de `https://en.wikipedia.org/wiki/Eternity_II_puzzle`

que todas as peças são diferentes e ignorando as dicas de peças pré-colocadas) é $256! \times 4^{256}$ ou, aproximadamente, 1.15×10^{661}. Um limite superior mais justo pode ser obtido levando em conta as dicas dos criadores do jogo que colocam o número de possibilidades em confortáveis 3.11×10^{545} posições possíveis.

Já que espero que vocês consigam terminar o jogo antes do fim do universo, para nossos propósitos iremos lidar com algumas instâncias menores do problema. Contudo, para manter as coisas interessantes não proveremos dicas sobre o posicionamento inicial de algumas pecas. O quebra-cabeça será dado como entrada contendo o tamanho do tabuleiro, número de cores e peças. Por padrão isso deverá ser lido da entrada padrão e a solução escrita na saída padrão.

Como parte do enunciado você tem uma versão simplista sequencial de um solver. O código só funciona para pequenas instâncias do problema. O seu desempenho poderia ser melhorado por uma divisão paralela mais inteligente do trabalho[3].

A versão sequencial dada usa uma abordagem de força bruta baseada em backtracking. É um bom exercício pensar em estratégias alternativas e heurísticas para melhorar o código fornecido. Note, contudo, que um único quebra-cabeças pode ter mais de uma solução correta.

2 O Projeto

Sua tarefa é escrever uma versão paralela do código **utilizando MPI** para a paralelização. MPI será a única ferramenta de paralelização aceita neste trabalho sendo que o uso de threads, OpenMP ou qualquer outra biblioteca não será aceito.

Fique a vontade para usar quaisquer outras otimizações, além da paralelização do código, para acelerar a execução da versão sequencial. Note, contudo, que dependendo da entrada pode haver mais de uma solução possível. Isto não deve ser um problema contanto que a sua implementação devolva como resposta uma solução válida.

2.1 Entrada

A entrada contém um quebra-cabeça. Ela consiste de uma lista de inteiros separados por espaços e quebras de linha. A primeira linha contém 2 inteiros: o tamanho do tabuleiro g e o número de cores c. Em seguida, serão fornecidas g^2 linhas, que descrevem cada uma das peças. A ordem das peças é importante (ela determina sua numeração que será utilizada na saída) e é contada a partir de 0. Logo as peças são numeradas de 0 a $g^2 - 1$. Cada peça é especificada por 4 inteiros entre 0 e $c - 1$ descrevendo as cores das suas bordas no sentido horário, começando pela borda superior. A cor 0 (cinza) é considerada especial pois deve ser utilizada obrigatoriamente nas bordas. Você pode assumir que $g \leq 16$ e $c \leq 22$.

O exemplo de entrada abaixo representa o tabuleiro e peças mostradas na figura acima. A entrada deve ser lida da *entrada padrão*.

2.2 Saída

A saída esperada deverá possúir g^2 linhas, cada uma representando uma das casas do tabuleiro. A ordem das linhas de saída segue a ordem do tabuleiro da esquerda para a direita, de cima para baixo. Cada linha é composta por 2 inteiros, o primeiro indica o número da peça e o segundo o número de rotações no sentido horário que foram efetuadas por aquela peça. Um saída válida para a a entrada apresentada (e mostrada na figura acima) é mostrada abaixo.

A saída deve ser escrita na *saída padrão*.

[3]Em uma turma anterior de Programação Paralela o mesmo problema foi dado no contexto de PThreads, e não de MPI. Apesar de haver muitas diferenças entre uma implementação feita em threads e outra em MPI, vocês podem se inspirar. O código de ambas está disponível aqui.

2.3 Exemplo de entrada e saída

Entrada	Saída
4 5	0 0
0 1 2 0	9 1
0 0 2 1	11 1
1 0 0 1	3 3
2 0 0 2	8 0
3 4 3 3	5 2
4 4 4 3	4 3
4 4 3 3	12 2
4 4 3 3	14 0
2 4 2 0	6 0
1 4 1 0	7 2
1 4 2 0	15 2
2 4 1 0	2 1
2 3 2 0	13 3
1 3 1 0	10 3
2 3 1 0	1 1
1 3 2 0	

Você pode verificar sua saída usando o programa `checker.c`. A entrada de exemplo acima, assim como outras entradas maiores, podem ser obtidas aqui: entradas.zip.

3 A Entrega

A entrega do código deverá ser feita exclusivamente via GitHub Classroom através do link: `https://classroom.github.com/a/g2oDX_C3` . Será considerado como entrega o último *commit* (não esqueça de dar *push*) no repositório até a **data limite de 12/04/2020**.

Para discussões, dúvidas e comentários utilize o Discord em: `https://discord.gg/A5FZk7m`.

Para **dúvidas específicas sobre o seu projeto**, seu código ou que contenham informações sensíveis (que você não quer que os outros grupos tenham acesso), crie um issue no seu próprio repositório (e não esqueça de marcar o professor para que ele seja notificado).

3.1 Código (4 pontos)

1. O seu repositorio deve conter todo o código fonte do seu programa juntamente com instruções claras sobre sua compilação e execução. Veja alguns ótimos exemplos em alguns projetos abertos no GitHub.
2. Programas com **erros de compilação receberão nota 0**.
3. O ambiente de testes será Linux com OpenMPI e GCC. Você deve entregar uma versão paralelizada com MPI.
4. Programas que não cumprirem os requisitos (resultados incorretos, erros durante a execução, ...) descritos na seção anterior receberão nota 0.
5. Programas sem boa documentação no código terão a sua nota reduzida.

6. Programas desorganizados (nomes de variáveis/funções ruins, falta de identação, ...) terão a sua nota reduzida.

3.2 Relatório (3 pontos)

Juntamente com o seu código você deverá entregar um relatório (máximo de 5 páginas) que contenha:

1. Uma explicação detalhada de como o particionamento das tarefas foi feito entre os processos na sua implementação.
2. Tabelas e gráficos do *speedup* e da eficiência para diversos números de processadores e entradas.
3. Interpretação dos valores mostrados na tabela anterior. Descreva e interprete a variação do *speedup* e da eficiência em relação à ao tamanho do problema e P. Explique os resultados obtidos levando em consideração as características da máquina utilizada.
4. Análise da escalabilidade da sua implementação. Ela é escalável? Apresenta escalabilidade forte ou fraca?
5. Análise do balanceamento de carga. Todos os processos recebem a mesma quantidade de trabalho? Todos acabam a execução aproximadamente ao mesmo tempo?

Dica: Utilize como referência as seções 3.6.2, 3.6.3 e 3.6.4 do livro [PP].

Obs.: Certifique-se de executar o seu programa pelo menos 3 vezes e relate o tempo da **execução mais rápida** (*wall-clock time*). Veja a motivação para fazê-lo nas notas de aula e/ou no livro [PP]-Cap. 3.

3.3 Desempenho (3 pontos)

1. É esperado que você consiga *speedup > 1* na versão paralela.
2. O teste de desempenho será efetuado em uma máquina Intel Xeon E5 com 16 cores (32 threads) com 64 GB de RAM.
3. Os desempenhos tanto da versão sequencial (10%) quanto da versão paralela (20%) do seu algoritmo serão avaliados.

3.4 Política de atrasos

Projetos entregues com atraso sofrerão uma redução da nota conforme a tabela abaixo:

Dias em Atraso	Nota Máxima
0	10
1	7
2	6
3	5
>3	0