UFPR – Departamento de Informática

CI316 – Programação Paralela - Trabalho Prático OpenMP

Prof. Marco Antonio Zanata Alves

Com base na implementação sequencial em linguagem C dos seguintes algoritmos: K-means, Color Histogram e Eternity II (listados no final deste arquivo).

Elabore um relatório com os seguintes itens:

1. Recorte o kernel (parte principal) de cada algoritmo e explique em suas palavras o funcionamento sequencial do trecho.
2. Explique qual a estratégia final (vitoriosa) de paralelização você utilizou.
3. Descreve a metodologia que você adotou para os experimentos a seguir. Não esqueça de descrever também a versão do SO, kernel, compilador, flags de compilação, modelo de processador, número de execuções, etc.
4. Com base na execução sequencial, meça e apresente a porcentagem de tempo que o algoritmo demora em trechos que você não paralelizou (região puramente sequencial).
5. Aplicando a Lei de Amdahl, crie uma tabela com o speedup máximo teórico para 2, 4, 8 e infinitos processadores.
6. Apresente tabelas de overhead, speedup e eficiência. Para isso varie o número de threads entre 1, 2, 4 e 8. Varie também o tamanho das entradas, tentando manter uma proporção. Exemplo de tabela:

		1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Eficiência	N=10.000	1	0,81	0,53	0,28	0,16
	N=20.000	1	0,94	0,80	0,59	0,42
	N=40.000	1	0,96	0,89	0,74	0,58

7. Analise os resultados e discuta cada uma das três tabelas. Você pode comparar os resultados com speedup linear ou a estimativa da Lei de Amdahl para enriquecer a discussão.
8. Seu algoritmo apresentou escalabilidade forte, fraca ou não foi escalável? Apresente argumentos coerentes e sólidos para suportar sua afirmação.

Cuidados gerais para efetuar os experimentos

- Para assegurar a corretude da implementação paralela, deve-se verificar se os resultados paralelos batem com os sequenciais executando diferentes entradas.
- Execute pelo menos 20x cada versão para obter uma média minimamente significativa. Ou seja, todo teste, onde mudamos o número de processos ou tamanho de entrada, devemos executar 20x e obter uma média com desvio padrão.
    - As métricas deverão ser calculadas encima da média das execuções.
- Sugiro escolher um modelo de máquina e sempre utilizar o mesmo modelo até o final do trabalho.
    - Cuidar para não executar em servidores virtualizados ou que contenham outros usuários (processos ativos) utilizando a mesma máquina. Diversos servidores do DINF são máquinas virtualizadas e os testes de speedup não serão satisfatórios/realísticos.
    - Cuide para que não haja outros processos ou usuários usando a máquina no mesmo momento que você esteja executando seus testes.
- **Teste de escalabilidade forte**: Manter um tamanho de entrada N qualquer, e aumentar gradativamente o número de processos. Sugere-se que escolha-se um N tal que o tempo de execução seja maior ou igual a 10 segundos.
- **Teste de escalabilidade fraca:** Aumentar o tamanho da entrada proporcionalmente com o número de processos. Exemplo: 1xN, 2xN, 4xN, 8xN, 16xN. Atenção, escalar N com o número de threads/processos (não de máquinas no caso do MPI).

Regras Gerais

A paralelização dos códigos deve ser feita em C ou C++ utilizando as primitivas OpenMP. A entrega será feita pelo Moodle dividida em duas partes

- Relatório
- Códigos fonte paralelo (OpenMP)

Casos não tratados no enunciado deverão ser discutidos com o professor.

Os trabalhos devem ser feitos individualmente.

Atencão, a cópia do trabalho (plágio), acarretará em nota igual a Zero para todos os envolvidos.

K-means

O agrupamento K-Means é popular na análise de cluster em mineração de dados e consiste em um método que permite a modelagem de funções de densidade de probabilidade pela distribuição de vetores de características.

O método particiona um conjunto de n pontos em k clusters, onde cada ponto será associado ao cluster com a média mais próxima. A distância euclidiana é geralmente a métrica adotada para medir a proximidade.

O agrupamento é um problema NP-difícil, mas existem algoritmos heurísticos eficientes que convergem rapidamente para um ótimo local.

Crie uma versão paralela para o algoritmo kmeans .

O algoritmo:

Nesta implementação, o algoritmo tem como entrada as coordenadas (x, y, z) de k centróides iniciais e um conjunto de pontos (x, y, z).

O algoritmo executa um processo iterativo, onde os pontos são agrupados de acordo com a distância euclidiana mínima entre eles e os centróides. Em seguida, o centróide de cada partição é recalculado com base na média de todos pontos na cluster, e todo o procedimento é repetido até que nenhum centróide seja alterado e nenhum ponto mude de cluster. Após a conclusão, o algoritmo retorna as coordenadas dos k centróides finais

Entrada:

A entrada deve ser lida a partir da entrada padrão.

Exemplo:

./kmeans < 10x1M.txt

As primeiras duas linhas contêm o número de centróides k e o número de pontos n. As próximas k + n linhas contém as coordenadas (x, y, z) para os k centróides e n pontos, respectivamente.

Saída:

A saída deve ser gravada na saída padrão.

O programa deve imprimir k linhas com as coordenadas (x, y, z) dos k centróides re-agrupados.

Arquivos:

kmeans.c - Código sequencial em C do algoritmo kmeans

Color Histogram

No processamento de imagens e fotografia, um histograma de cores é uma representação da distribuição de cores em uma imagem. Para imagens digitais, um histograma de cores representa o número de pixels em cada um dos intervalos de cores encontradas em uma imagem.

Crie uma versão paralela de um algoritmo que gera um histograma de cores de uma imagem digital.

Entrada:

A imagem de entrada é um arquivo de texto no formato PPM (http://netpbm.sourceforge.net/doc/ppm.html). A resolução da imagem é de até 16k.

A entrada deve ser lida a partir da entrada padrão.

Exemplo:

./histogram < 16k-a.ppm

Saída:

A saída contém 64 números que representam a média de pixels calculados para a imagem digital. Cada número é um número é um float com 3 casas decimais.

A saída deve ser gravada na saída padrão.

Eternity II

O quebra-cabeça Eternity II foi lançado em 2007 com a promessa de pagar US $ 2 milhões a primeira pessoa que apresenta-se uma solução completa. Porém, até hoje nenhuma solução correta foi apresentada.

Eternity II é um quebra-cabeça clássico de combinação de arestas que envolve colocar 256 peças quadradas em uma grade 16x16.

Cada ladrilho tem suas bordas marcadas com diferentes combinações de cores. Os ladrilhos devem ser colocados de forma que todas as cores de suas bordas correspondam às cores dos ladrilhos adjacentes. As bordas da grade são um caso especial e devem ser combinadas apenas com ladrilhos de bordas cinza. Os ladrilhos podem ser giradas; portanto, cada bloco tem 4 posicionamentos possíveis em cada posição da grade.

Entrada:

Cada entrada contém 1 quebra-cabeça.

A primeira linha contém 2 inteiros, o primeiro corresponde ao tamanho do grid (g) e o segundo a quantidade de cores (c).

As próximas g^2 linhas correspondem aos ladrilhos, na ordem que são apresentados, cada uma contém quatro inteiros que representam as cores do azulejo. O sentido de preenchimento das cores em cada posição do grid é no sentido horário. Note que a cor 0 (cinza) é um caso especial e deve ser usada apenas nas bordas.

A entrada deve ser lida a partir da entrada padrão.

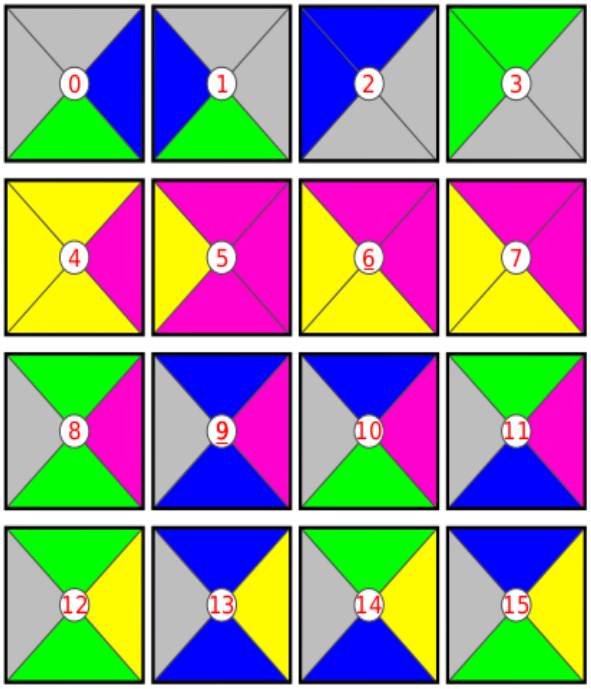
Exemplo:

./eternity < eternity.in

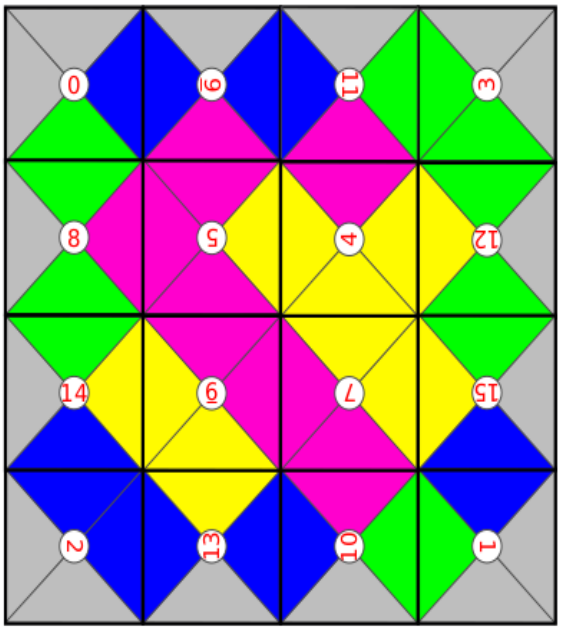
Saída:

A saída contém as g^2 linhas referentes ao grid. Note que a ordem importa e deve ser impressa percorrendo o grid da esquerda para a direira e de cima para baixo. Cada linha contem dois inteiros, o primeiro corresponde ao número do ladrilho e o segundo ao número de rotações realizadas no sentido horário.

Abaixo temos um exemplos ilustrado de entrada (a) e saída (b).



(a)



(b)

Input	Output for the input
4 5 0 1 2 0 0 0 2 1 1 0 0 1 2 0 0 2 3 4 3 3 4 4 4 3 4 4 3 3 4 4 3 3 2 4 2 0 1 4 1 0 1 4 2 0 2 4 1 0 2 3 2 0 1 3 1 0 2 3 1 0 1 3 2 0	0 0 9 1 11 1 3 3 8 0 5 2 4 3 12 2 14 0 6 0 7 2 15 2 2 1 13 3 10 3 1 1