Desafio de Programação Paralela

ERAD-SP 2024

Calebe de Paula Bianchini[1]

[1]Universidade Presbiteriana Mackenzie

Regras Gerais

Leia atentamente todos os problemas apresentados. Cada problema possui uma breve descrição, e exemplos de entradas e saídas (que podem estar nos anexos). Você pode criar sua própria versão paralela e/ou distribuída para o problema ou modificar uma implementação disponibilizada pelos juízes de acordo com sua estratégia, a não ser que o enunciado do problema diga o contrário.

Cada equipe deve criar um arquivo *Makefile* (veja os exemplos nos anexos) com as instruções de compilação. O arquivo binário principal gerado ou um script de execução para (e.g. MPI) deve ser igual a letra que representa o enunciado do problema, em maiúsculo: por exemplo, se o problema se chama ***A***, o executável/script deve-se chamar ***A***.

Para a submissão, cada equipe deve compactar o código fonte e o *Makefile* em um arquivo .zip (obrigatoriamente) e enviá-lo no sistema BOCA. Esse arquivo pode ter, no máximo, 32 Kb de tamanho. Submissões maiores do que este tamanho serão desconsideradas.

O tempo de execução de um problema será medido utilizando a ferramenta *time* no ambiente de testes dos juízes. Será considerada apenas o tempo real de CPU (*real CPU*). Cada submissão será executada três vezes com as mesmas entradas e o tempo médio será utilizado como a métrica de tempo de execução. O tempo de execução das soluções sequenciais dos problemas, produzido pelos juízes, também será medido da mesma forma. A pontuação se dará pelo *speedup*, ou seja, a razão entre o tempo de execução da solução sequencial pelo tempo de execução da submissão. As equipes ganharão pontos em cada problema conforme o valor do *speedup*, sendo este acrescentado no placar.

O time que tiver a maior pontuação será considerado o campeão do Desafio!

Este caderno tem 2 problemas; as páginas estão numeradas de 1 a 5.

Informações Gerais

Infraestrutura de julgamento no BOCA

A infraestrutura disponibilizada é composta por 2 nós computacionais, cada um com 2xIntel Xeon Cascade Lake Gold 6252, 4x NVIDIA Volta V100 GPU e 384Gb de memória RAM. Os nós são interligados por uma rede de interconexão Infiniband.

A infraestrutura utilizada faz parte do supercomputador Santos Dumont (SDumont[1]), e é gerenciada e fornecida pelo Laboratório Nacional de Computação Científica (LNCC).

Compilação

Você deve usar CC ou CXX dentro do seu *Makefile*. Tenha cuidado ao redefini-los! Existe um arquivo *Makefile* dentro do seu pacote de problemas que você pode modificar.

Exemplo:

```
FLAGS=-O3
EXEC=sum
CXX=g++

all: $(EXEC)

$(EXEC):
	$(CXX) $(FLAGS) $(EXEC).cpp -c -o $(EXEC).o
	$(CXX) $(FLAGS) $(EXEC).o -o $(EXEC)
```

Cada tipo submissão utiliza um grupo de compiladores. Veja-os abaixo e escolha bem quando escrever o seu *Makefile*. O compilador marcado como *default* é predefinido nas variáveis **CC** e **CXX**.

Ambiente	Compilador	Comando
host	GCC versão 13.2.0 (GCC)	C = gcc C++ = g++
gpu	NVIDIA CUDA release 11.6, V11.6	C = nvcc C++ = nvcc
mpi	OpenMPI 5.0.0	C = mpicc C++ = mpic++

[1] https://sdumont.lncc.br/

Submissão

Você deve ter um script de execução que tenha o mesmo nome do problema. Este script executa a sua solução da maneira que você a desenvolveu. Existe um script simples dentro do seu pacote de problemas que deve ser modificado.

Exemplo:

```
#!/bin/bash
# This script runs a generic Problem A
# Using 32 threads and OpenMP
export OMP_NUM_THREADS=32
OMP_NUM_THREADS=32 ./sum
```

Submissão de código MPI

Se você está planejando enviar uma solução MPI, você deve compilar usando *mpicc/mpic++*. O script deve chamar *mpirun/mpiexec* com o número correto de processos (máx.: 2). Deve usar um arquivo chamado *machines* que é gerado pelo sistema *auto-judge* - **não** crie este arquivo.

Exemplo:

```
#!/bin/bash
# This script runs a generic Problem A
# Using MPI in the entire cluster (4 nodes)
# 'machines' file describes the nodes
mpirun -np 4 -machinefile machines ./sum
```

Comparando tempos e resultados

Na sua máquina pessoal, meça o tempo de execução da sua solução utilizando o programa *time*. Adicione redirecionamento de entrada/saída ao coletar o tempo. Use o programa *diff* para comparar os resultados originais e da sua solução.

Exemplo:

```
$ time -p ./A < original_input.txt > my_output.txt
real 4.94
user 0.08
sys 1.56

$ diff my_output.txt original_output.txt
```

Não meça o tempo e **não** adicione redirecionamento de entrada/saída ao enviar a sua solução – o sistema *auto-judge* está preparado para coletar seu tempo e comparar os resultados.

Problema A

k-Nearest Neighbors Classifier

Dado um conjunto de pontos bidimensionais *S* divididos em *n* grupos, um inteiro *k* e um ponto bidimensional *P* a ser classificado, o algoritmo *k-Nearest Neighbours* (KNN) pode ser usado para resolver o problema de classificação do ponto *P*, ou seja, classificar o ponto *P* em um dos *n* grupos existentes com base em alguns critérios.

O algoritmo KNN funciona comparando as distâncias de cada ponto $s \in S$ ao ponto *P*, selecionando então os *k* pontos com os menores valores de distância e contando a frequência de cada grupo nesses *k* pontos. O ponto *P* é então classificado como pertencente ao grupo de maior frequência entre os *k* pontos mais próximos.

A Figura A1 ilustra um exemplo do algoritmo de classificação KNN sendo aplicado. Os pontos foram segregados entre 4 grupos diferentes (isto é, A, B, C e D) e um ponto *P* sem grupo é fornecido para ser classificado. O algoritmo KNN então procura pelos *k* pontos mais próximos do ponto *P* e, em seguida, os grupos têm suas frequências contadas, sendo escolhida a maior frequência como classificação de *P*. No caso deste exemplo, o ponto *P* é classificado como pertencente ao grupo C.

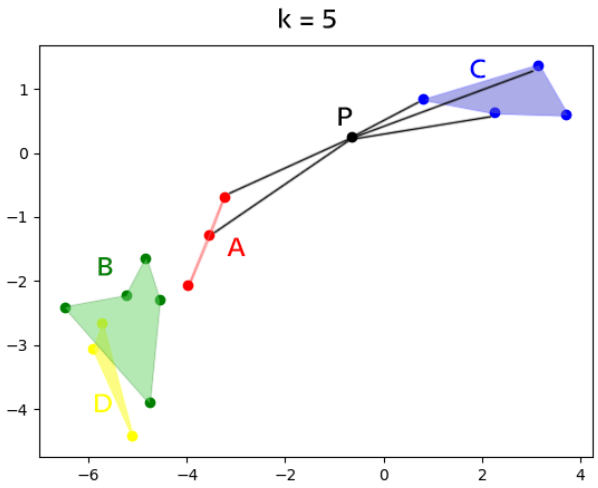

Figura A1. Exemplo simples da classificação KNN.

O seu desafio é escrever uma versão paralela e/ou distribuída do algoritmo KNN.

Entrada

Uma entrada representa a classificação de um único ponto. A primeira linha contém o padrão “n groups=*n*”, onde *n* é o número total de grupos nesta entrada. Então, para cada um dos grupos *n* serão fornecidas as seguintes linhas, a primeira linha possui o padrão “label=*c*” onde *c* é um único caractere que representa o rótulo (ou identificação) do grupo,

a segunda linha segue o padrão "length=*L*", onde *L* é o número de pontos bidimensionais deste grupo e, então, as próximas linhas L contêm o padrão "(x,y)" que representa cada ponto pertencente ao referido grupo. Após todos os grupos terem sido representados, há uma única linha com o padrão "k=*k*", onde *k* é o parâmetro discutido anteriormente. Por fim, a linha final também contém o padrão "(x,y)", representando as coordenadas do ponto bidimensional a ser classificado.

A leitura dos dados é feita a partir da entrada padrão.

Saída

A saída contém um único caractere, sendo ele o rótulo do grupo escolhido como classificação do ponto fornecido.

A escrita dos dados é feita na saída padrão.

Exemplo

Entrada	Saída respectiva a entrada

`n groups=4`
`label=A`
`length=3`
`(-3.55,-1.28)`
`(-3.99,-2.06)`
`(-3.23,-0.70)`
`label=B`
`length=5`
`(-4.85,-1.65)`
`(-5.23,-2.22)`
`(-4.75,-3.89)`
`(-6.48,-2.41)`
`(-4.56,-2.29)`
`label=C`
`length=4`
`(2.25,0.64)`
`(0.80,0.85)`
`(3.13,1.37)`
`(3.71,0.59)`
`label=D`
`length=3`
`(-5.73,-2.65)`
`(-5.11,-4.41)`
`(-5.92,-3.06)`
`k=5`
`(-0.65,0.25)`
	C

Problema B

Smooth

A área de entretenimento tem apresentado tecnologias cada vez mais avançadas. Desde 2013 já existiam modelos de TVs com resolução de 8K (ou *Super Hi-Vision*). Isso significa que esses equipamentos têm 16 vezes "melhor" resolução que o famoso *FullHD*.

Além disso, a partir da incorporação de microprocessadores e microcontroladores nesses aparelhos, será muito comum a execução de algoritmos de filtros e estênceis nas imagens que eles projetam. Todos estes algoritmos já são bem conhecidos da área de processamento de imagem.

Um destes estênceis é denominado *smooth*. Seu principal objetivo é eliminar um pixel não representativo na imagem em relação aos pixels vizinhos, ou seja, eliminar ruídos da imagem.

O exemplo mais conhecido dessa técnica *smooth* é o algoritmo que utiliza um grupo de pixels de tamanho 3x3 e remove o ruído por meio de uma média aritmética, como pode ser visto na Figura B1.

x_0	x_1	x_2
x_3	x_4	x_5
x_6	x_7	x_8

$$x_4' = \frac{x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8}{9}$$

Figura B1. x_4' é o novo valor para o pixel x_4 em um grupo de 3x3 pixels.

Porém, esse algoritmo só funcionaria diretamente no pixel de uma imagem caso ela estivesse em escala de cinza, limitando seu uso em ambientes reais.

Um pixel de uma imagem colorida pode ser decomposto em diversos modelos de cores: RGBA, CMYK etc. Assim, cada pixel é decomposto em cores primárias, conforme o modelo de cores. Por exemplo, um pixel da cor branco pode ser representado, no modelo RGBA (*Red-Green-Blue-Alpha*), como uma quadrupla (R, G, B, A), sendo seus valores (255, 255, 255, 0).

Para aplicar o *smooth* nessa imagem colorida, separa-se cada valor da cor primária do pixel e aplica-se o algoritmo. Ou seja, aplica-se o algoritmo de *smooth* para o valor R do pixel, depois para o valor G do pixel, depois para o valor B e, por fim, para o A. Dessa forma, o novo pixel colorido calculado tem novos valores para (R, G, B, A). A Figura B2 mostra esse cálculo apenas para o valor de R de um pixel.

r_0	r_1	r_2
r_3	r_4	r_5
r_6	r_7	r_8

$$r_4' = \frac{r_0 + r_1 + r_2 + r_3 + r_4 + r_5 + r_6 + r_7 + r_8}{9}$$

Figura B2. r_4' é o novo valor de R para o pixel colorido x_4' em um grupo de 3x3 pixels.

Além disso, é necessário resolver o cálculo do *smooth* para os pixels de borda. Pelo exemplo da Figura B2, percebe-se que alguns pixels não existiriam na imagem real quando r_4 for um pixel de borda. Nesse caso, várias soluções podem ser adotadas, mas a forma mais simples é adotar algum valor fixo para a quadrupla (R, G, B, A) como, por exemplo, (0, 0, 0, 0).

Seu Desafio é escrever uma versão paralela e/ou distribuída do algoritmo de *smooth* que utiliza média aritmética em um grupo de 5x5 pixels e em uma imagem colorida com resolução de até 8K (ou *Super Hi-Vision*).

Entrada

O arquivo de entrada está em formato binário (*little-endian*) com apenas uma imagem. Os dois primeiros valores do arquivo, de tamanho de 16 bits, são a largura ($1 \leq X \leq 7680$) e a altura da imagem ($1 \leq Y \leq 4320$).

Os pixels são coloridos, de 32 bits cada, e são os próximos valores do arquivo. Cada pixel está no formato RGBA, conforme mostra a Figura 3 ($0 \leq R, G, B, A \leq 255$).

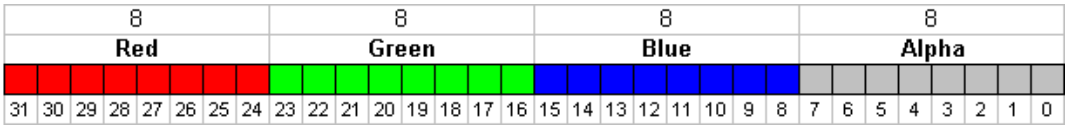

Figura 3. Formato de um pixel no modelo de cores RGBA.

Os X ($1 \leq X \leq 7680$) primeiros pixels formam a primeira linha da imagem, seguidos de X pixels para a segunda linha da imagem, num total de Y linhas ($1 \leq Y \leq 4320$).

Além disso, assuma valores para (R, G, B, A) = (0, 0, 0, 0) para o cálculo dos pixels de borda.

Os dados devem ser lidos de um arquivo denominado image.in.

Saída

A saída deve ser feita em um arquivo binário, mantendo a mesma estrutura citada para o arquivo de entrada: apenas uma imagem, os dois primeiros valores de 16 bits são a largura e a altura da imagem, seguidos dos pixels da imagem com 32 bits cada, no formato RGBA, com X pixels por linha (total de Y linhas).

Os resultados do programa devem ser escritos em um arquivo denominado image.out.