SuperComputação

Insper - Engenharia da Computação - 2022.1 - Profs. André Filipe e Luciano Silva

Avaliação Final

Instruções

- A interpretação do enunciado faz parte da avaliação. Eventuais dúvidas serão resolvidas via *chat* do *Teams*, de maneira individual;
- É permitida a consulta ao material da disciplina (tudo o que estiver no repositório do Github da disciplina e no site http://insper.github.io/supercomp. Isso também inclui suas próprias soluções aos exercícios de sala de aula, mas não inclui materiais não digitais, tampouco outros materiais além dos citados;
- É permitido consultar a documentação de C++ nos sites http://cplusplus.com e https://cppreference.com e o site do OpenMP da Intel/Microsoft e da Nvidia Thrust;
- A prova é individual. Qualquer consulta a outras pessoas durante a prova constitui violação do código de ética do Insper.
- Boa prova!

Questão nro. 1 (2 pontos):

Implemente um programa em C++ que gere n números aleatórios inteiros (entre 1 e 999), populando um vetor. Fazendo uso de OpenMP, calcule a média e desvio padrão dos elementos deste vetor. Você deve testar para n grandes, produzindo um gráfico onde X é o tamanho de n e Y é o tempo de execução. Teste também seu programa para rodar com 2, 3, 5, 7 e 9 threads, avaliando o desempenho.

Entregue o código-fonte, o gráfico gerado e comentários sobre o desempenho conforme o número de threads foi sendo modificado.

Questão nro. 2 (4 pontos): Em nossas aulas tivemos a oportunidade de executar o problema Saxpy com a biblioteca *Thrust*. Você pode encontrar o código executado em aula neste endereço: https://insper.github.io/supercomp/aulas/16-gpu-customizacao/.

Sua tarefa:

1. Implemente uma versão do Saxpy com OpenMP (8 threads por padrão). [2 pontos]
2. Execute a versão com OpenMP e a versão em GPU. Compare os tempos de execução. No caso da GPU ser mais rápida, busque aumentar o número de threads do OpenMP (até um limite de 32). O tempo de execução do OpenMP melhorou? Justifique também caso OpenMP esteja sendo mais rápido que GPU. Há algo relacionado a movimentação de dados entre CPU e GPU? [2 pontos]

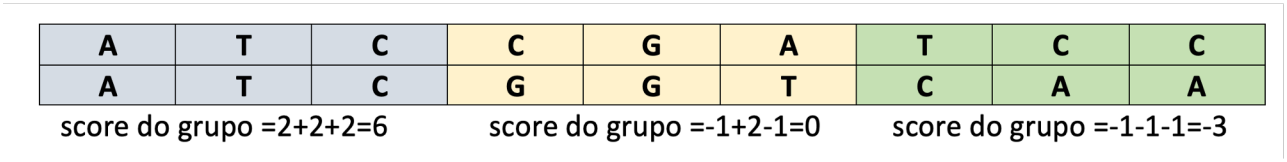
Questão nro. 3 (2 pontos): Implemente a versão do código abaixo com *Thrust*. Atente-se ao tratamento adequado para a geração de números aleatórios. Você deve usar, **obrigatoriamente**, a implementação **transform-reduce** da biblioteca Thrust, disponível em https://tinyurl.com/3namnuhv

```
#include <random>
#include <iostream>
#include <omp.h>
#include <vector>
using namespace std;
int main () {
   long N = 100000000L;
   long sum = 0;
   default_random_engine generator;
   uniform_real_distribution<double> distribution(0.0, 1.0);
   for (long i = 0; i < N; i++) {
        double x, y;

        x = distribution(generator);
        y = distribution(generator);
        if (x*x + y*y <= 1) {
            sum++;
        }
    }
    double pi = 4.0 * sum / N;
    cout << pi << endl;
    cout << sum << endl;

 }
```

Questão nro. 4 (2 pontos): Suponha que queiramos comparar duas sequencias de tamanho múltiplo de k, $k > 1$. Ao invés de cada core da GPU comparar somente dois aminoácidos de cada vez, poderíamos comparar $2k$ aminoácidos de cada vez. Abaixo temos um exemplo com $k = 3$, para duas sequencias com 9 aminoácidos:



Para cada grupo de $k = 3$ aminoácidos, calculamos o score simples entre eles **(+2 para match e -1 para mismatch)**. Calculando o máximo entre os scores dos grupos, obtemos o score máximo de 6.

Implemente um kernel utilizando Thrust que receba o valor de k, assim como as duas sequencias de tamanho múltiplo de k, e calcule o score entre as duas sequencias com os pesos indicados anteriormente.

Implemente, adicionalmente, um programa para testar o seu kernel.
