Supercomputação – Avaliação Final

Instruções

Bem-vindo(a)s à prova final da disciplina de Supercomputação. Leia as instruções abaixo:

SOBRE HORÁRIOS:

- Abertura das salas (07h15); início da prova (07h30); final (09h30); limite tempo extra (10h30);
- A submissão da prova deve ser feita impreterivelmente até 15 min após o deadline (ou seja, 09h45 ou 10h45 (aluno tempo extra), de 26/novembro/2024 – horário de Brasília);
- NÃO serão aceitas entregas após o prazo;

SOBRE SUBMISSÕES DA PROVA:

- O aluno pode fazer múltiplas submissões. O sistema considerará a última como oficial;
- A submissão da prova deve ser um ZIP de uma pasta principal**, nomeada com seu nome completo**, e organizada com uma subpasta para cada questão. Na pasta da questão deve ter (i) o código-fonte, (ii) o arquivo de submissão ao Slurm (.slurm) (se aplicável), (iii) o arquivo de saída do Slurm (ou um print comprovando a execução) e (iv) um TXT com suas respostas textuais (se aplicável). ATENÇÃO: nas questões teóricas, pode enviar apenas o TXT das respostas na pasta.

SOBRE A RESOLUÇÃO DA PROVA:

- 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, anotações e documentações de C++ e bibliotecas pertinentes em sites oficiais;
- Execuções de código no cluster necessitam, obrigatoriamente, serem realizadas via submissão de Jobs não interativos. A execução de códigos da prova diretamente na linha de comando do SMS-HOST fere a boa prática de utilização. O log do cluster será monitorado, e caso seja constatado que o aluno rodou código sem submeter o job, a nota máxima da questão será 50% do valor original. Ex: a questão vale 3, e o aluno acertou tudo, mas executou de forma incorreta diretamente no nó principal, então a nota da questão será 1,5.

SOBRE QUESTÕES DE ÉTICA E PLÁGIO:

- A prova é individual. Qualquer consulta a outras pessoas durante a prova constitui violação do código de ética do Insper;
- Qualquer tentativa de fraude, como trechos idênticos ou muito similares, ou uso de GenAI, implicará em NOTA ZERO na prova a todos os envolvidos, sem prejuízo de outras sanções.

BOA PROVA! \o/

Questões

A interpretação do enunciado faz parte da avaliação. Em caso de dúvida, pode assumir premissas e explicá-las nos comentários.

Questão 1 – Submissão de jobs (Total: 1 ponto)

Esta questão avalia sua compreensão de como solicitar recursos ao cluster via arquivo ".slurm".

Considere que:

- O parâmetro "`--mem=<tamanho>[unidades]`" especifica a memória real necessária por nó. As unidades padrão são megabytes. Diferentes unidades podem ser especificadas usando o sufixo [K|M|G|T].
- Você pode definir o número de threads usando a variável de ambiente OMP_NUM_THREADS. Uma forma simples é antes de chamar o executável fazer o "export" da variável de ambiente da seguinte forma "`export OMP_NUM_THREADS=<number of threads to use>`".

Pede-se:

- Crie um arquivo "script.slurm" que solicita 5 nós de computação, cada um rodando apenas 1 task, e cada task alocando 4 cores, e uma quantidade de 16 gigas de memória. O job deve ser executado numa partição chamada "prova", sendo abortado após 1 hora, exportando a variável de ambiente OMP_NUM_THREADS para usar 15 threads, e solicitar a execução de um código fictício denominado "./programa"
- Nesta questão entregue apenas o .slurm criado
- NOTA: não é necessário submeter o job no cluster! Esta é uma questão teórica! ;D

Questão 2 – Paralelização de códigos sequenciais (Total: 2 pontos)

Contexto:
A transição de códigos sequenciais para paralelos é fundamental para aproveitar o potencial de sistemas de computação modernos com múltiplos núcleos. Nesta questão, você deverá converter diferentes segmentos de código C++ de sequencial para paralelo, utilizando OpenMP. Programe, compile e execute os códigos.

Parte A – Paralelização de Cálculo de Fatorial:

```
#include <iostream>
long long factorial(int n) {
    long long result = 1;
    for (int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}
int main() {
    std::cout << "Fatorial de 10 é " << factorial(10) << std::endl;
}
```

Parte B – Paralelização de Normalização de Vetor:

```
#include <iostream>
#include <vector>
#include <cmath>
std::vector<double> normalize(std::vector<double>& v) {
    double sum = 0;
    for (double num : v) {
        sum += num * num;
    }
    double magnitude = sqrt(sum);
    for (double& num : v) {
        num /= magnitude;
    }
    return v;
}

int main() {
    std::vector<double> vec{1.0, 2.0, 3.0, 4.0};
    vec = normalize(vec);
    for (auto v : vec) {
        std::cout << v << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Parte C – Paralelização de Processamento de Imagem:

```
#include <vector>
void processImage(std::vector<std::vector<int>>& image) {
    for (size_t i = 0; i < image.size(); ++i) {
        for (size_t j = 0; j < image[i].size(); ++j) {
            image[i][j] = (image[i][j] * 2) % 256;
        }
    }
}

int main() {
    std::vector<std::vector<int>> img{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    processImage(img);
    for (auto& row : img) {
        for (auto& col : row) {
            std::cout << col << " ";
        }
        std::cout << std::endl;
    }
    return 0;
}
```

Parte D – Soma de Elementos de Múltiplos Vetores, com print de resultado:

```
#include <iostream>
#include <vector>

// Função que calcula a soma dos elementos de um vetor
double vectorSum(const std::vector<double>& vec) {
    double sum = 0.0;  // Esta será a variável private
    for (double val : vec) {
        sum += val;
    }
    return sum;
}
```

```
int main() {
    std::vector<std::vector<double>> listOfVectors = {
        {1.0, 2.0, 3.0},
        {4.0, 5.0, 6.0},
        {7.0, 8.0, 9.0}
    };

    for (const auto& vec : listOfVectors) {
        double sum = vectorSum(vec);
        std::cout << "Sum of vector elements: " << sum << std::endl;
    }

    return 0;
}
```

Questão 3 – Paralelizando tarefas X Paralelizando dados (Total: 2.5 pontos)

Implemente um programa em C++ que calcule a média, o maior valor, e o produto de um conjunto de números usando OpenMP. Você deve ter uma função para cada cálculo. Utilize #pragmas pertinentes para paralelizar a execução dessas tarefas simultaneamente e, em cada tarefa, sub-paralelizar para distribuir o processamento dos dados. Avalie a eficiência do paralelismo aplicado. Crie o vetor que preferir para demonstrar a corretude do seu código. **Execute seu código no Cluster**!

Questão 4 – Paralelização Híbrida com MPI e OpenMP (Total: 3 pontos)

Enunciado: Desenvolva um programa que utilize MPI para distribuir as linhas de uma grande matriz (dimensão=1000x1000) entre múltiplos processos e use OpenMP dentro de cada processo para calcular a média de cada linha do subconjunto de linhas recebido. Cada processo deve enviar sua lista de médias parciais de volta ao processo raiz, que irá calcular a média total de todas as linhas e exibir os resultados. A inicialização da matriz no rank = 0 está a seu critério.

Configuração do job: O código deve usar 4 processos, cada um com 5 cores. O número máximo de threads por processo é deixado à sua escolha. Considere que o processo raiz também participa dos cálculos.

Questão 5 – Fundamentos de Paralelismo (Total: 1.5 pontos)

- **Parte A (0,75 ponto)**: Descreva com suas palavras o que é *paralelismo*, diferenciando-o do processamento sequencial. Relacione sua reposta com o conceito de *escalabilidade*.
- **Parte B (0,75 ponto)**: O conceito de *balanceamento de carga* em computação paralela refere-se à distribuição equitativa do trabalho entre todos os processadores ou núcleos disponíveis em um sistema de computação paralelo. Por que é importante? Quais técnicas podem ser usadas para alcançar um balanceamento efetivo?