Ministério da Educação
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
UNED Nova Friburgo
Bacharelado em Sistemas de Informação
Disciplina de Programação Paralela e Concorrente
Professor Bruno Policarpo

Trabalho: Paralelização com MPI

Pablo Veiga

1. Descrição

O objetivo deste trabalho é paralelizar, utilizando a biblioteca MPI, o algoritmo ***Counting Sort***.

2. Especificação

No presente trabalho, o aluno deverá responder as questões abaixo, ou desenvolver os programas solicitados. Questões teóricas deverão ser entregues em um documento PDF, contendo a identificação dos alunos, as questões e suas respectivas respostas. Já os códigos-fonte das questões de desenvolvimento deverão ser entregues separadamente.

As questões são:

1. Quais são os dados que deverão ser distribuídos por mensagens? Qual a função MPI você utilizará? Por quê?

   R. Os dados que deverão ser distribuídos por mensagem serão os do *array* a ser ordenado, o seu tamanho e a identificação da chamada.

   A função MPI_Bcast deve ser utilizada pois todos os processos precisam ter acesso ao array completo para saber quantos elementos são menores com relação a cada item da parte da execução de sua responsabilidade.

2. Como o resultado final deverá ser composto? Qual a função do MPI você utilizou? Por quê?

   R. O resultado final deverá ser composto pela chamada da função MPI_Gather disponibilizando os resultados parciais e as posições, recompondo os dados distribuídos anteriormente durante a execução.

3. Escreva um programa em C ou Java que “paralelize” o ***Counting Sort*** utilizando MPI.

   R. Código implementado e enviado (Counting_Sort_MPI.zip)

4. Compare o desempenho da função com a original serial, com a função ***qsort*** da biblioteca do C e com a implementação com ***OpenMP***. Explique as diferenças encontradas. Você acha possível o MPI “ganhar” do OpenMP ou do *qsort*? Por quê?

   R. Considerando um array de 20.000 posições

Serial	**2.945535 segundos**
Counting Sort OpenMP	**1.256732 segundos**
Counting Sort MPI	**1.465168 segundos**
qSort	**0.000003 segundos**

   O algoritmo *qSort* é o mais rápido pois foi construído em baixo nível, pré-compilado e otimizado para este fim.

5. (**1 bônus**) Descubra como executar o programa MPI de forma distribuída, ou seja, com os processos sendo executados em múltiplos computadores.
   R. -