Instituto Tecnológico de Aeronáutica

CE-265: Processamento Paralelo

Exercício 08: Soma de Array Distribuído

Bruno Benjamim Bertucci - Turma 23.2

1 Introdução

Introduzido em 1997, o padrão MPI 2.0 introduziu novas possibilidades de se implementar paralelismo usando o modelo de troca de mensagens, principalmente com a comunicação unilateral. Em suma, essa adição substitui o modelo de troca de dados entre um par de comandos `MPI_Send` e `MPI_Recv` por um novo paradigma onde, por um lado, um processo expõe um trecho da sua memória aos demais processos, sendo esse trecho denominado de janela, enquanto os demais processos podem ler ou alterar o conteúdo dessa janela.

2 Implementação de soma de *Array* distribuído

Para exemplificar o uso desse novo padrão de implementação introduzido no MPI 2.0, foi feita a implementação de um programa paralelo que dispõe de um arquivo contendo os elementos de um vetor de números inteiros, e um segundo arquivo contendo uma série de índices desse mesmo vetor. O programa cria um número de processos definido em tempo de execução. O processo de número zero separa para si próprio um trecho do vetor do arquivo e da série de índices, e manda um dos trechos remanescentes para cada um dos demais processos, de tal forma que cada trecho do vetor tem tamanho igual, o que ocorre também com os trechos da série de índices.

Dessa forma, os demais processos devem receber seus respectivos trechos. Essa comunicação é feita usando os comandos `MPI_Send` no processo zero e `MPI_Recv` nos demais processos. Ao final dessa etapa do programa, portanto, todos os processos, incluindo o zero, devem ter armazenado um trecho do vetor de inteiros, e um da série de índices.

Em seguida, é necessário que cada processo calcule a soma dos elementos do vetor de inteiros com os índices recebidos por ele e, ao final, que as somas calculadas em cada processo sejam acumuladas em uma soma total do processo de número zero. Como os índices podem referenciar trechos do vetor que foram enviados para outros processos, é necessário que um processo consiga acessar os trechos dos demais. Isso foi feito criando uma janela em cada processo, abrangendo todo o espaço de memória que armazena tais trechos. Uma segunda janela foi criada para expor os valores das somas computadas por cada processo.

Para prosseguir para a execução da soma, é necessário que as janelas dos vetores tenham sido criadas, por isso, o comando `MPI_Win_fence` foi colocado para garantir a sincronia dos processos nesse ponto. Em seguida, cada elemento do trecho da série de índices de um determinado processo é analisado, localizando o processo que contém o elemento do vetor desejado, bem como a posição desse elemento no seu respectivo processo. Se ele estiver contido no mesmo processo, o valor desse elemento é armazenado. Caso contrário, esse armazenamento é feito pela leitura do elemento no processo correto usando o comando `MPI_Get`, que busca o valor desejado na janela desse processo. Após um novo comando `MPI_Win_fence` na janela dos vetores para garantir a sincronia, esse valor armazenado é acumulado em uma soma local do processo.

Antes de computar a soma das somas locais de cada processo, é necessário sincronizar a janela referente aos valores de soma local, o que requer um novo comando `MPI_Win_fence`. Feito isso, o comando `MPI_Accumulate()` é executado em todos os processos, exceto o de número zero, com operador `MPI_SUM` e a variável de soma do processo zero como alvo, fazendo com que as somas locais dos outros processos sejam somadas à soma local do processo zero, tendo como resultado a soma de todos os elementos indiciados.

Uma sincronização final, a fim de garantir o término da acumulação de valores, é executada. Finalmente, as janelas são destruídas usando o comando `MPI_Win_free`. Para garantir a correção da computação feita, a soma final calculada é comparada com o valor correto esperado.

3 Teste da paralelização usando MPI-2

Para testar essa implementação, o programa foi executado com números de processos variando entre 1, 2, 4, 8, 16 e 32, no supercomputador SDumont. Os arquivos de saída dessas execuções, que indicam a computação correta realizada, estão no Anexo A.

Durante a implementação do programa, foi possível observar que o modelo de comunicação unilateral é significativamente mais símples de ser implementado do que o modelo inicial definido no padrão MPI-1, necessitando de uma quantidade menor de intervenções no código sequencial para adquirir a paralelização. Ademais, percebeu-se também a necessidade de uso excessivo do comando `MPI_Win_fence`, devido ao fato de os comandos `MPI_Put` e `MPI_Get` serem não bloqueantes, ou seja, não travarem a execução do programa enquanto o envio/recebimento de dados estiver em andamento.

Anexo A – Saídas dos testes realizados

1 processo

```
**RESULTADO CERTO** (48820191429) com 1 ranks MPI
```

2 processos

```
**RESULTADO CERTO** (48820191429) com 2 ranks MPI
```

4 processos

```
**RESULTADO CERTO** (48820191429) com 4 ranks MPI
```

8 processos

```
**RESULTADO CERTO** (48820191429) com 8 ranks MPI
```

16 processos

```
**RESULTADO CERTO** (48820191429) com 16 ranks MPI
```

32 processos

```
**RESULTADO CERTO** (48820191429) com 32 ranks MPI
```