Introdução à programação paralela

Gabriel Martins de Miranda – 13/0111350

1. Título do capítulo

Comunicators and Topologies.
Comunicadores e Topologias.

2. Objetivo do capítulo

Mostrar o uso dos comunicadores e topologias do MPI, sendo o comunicador uma coleção de processos que podem trocar mensagens e topologia estruturas impostas aos processos que permite endereçamento destes de diversas formas. As ideias são abordadas através do algoritmo de multiplicação paralela de matrizes, chamado algoritmo de Fox. São mostradas modificações para ele e para um algoritmo genérico de multiplicação de matrizes.

3. Resumo do capítulo

São abordados dois algoritmos para multiplicação paralela de matrizes e duas ideias, comunicadores e topologias.

O primeiro algoritmo mapeia linhas da matriz a processos, mas este mapeamento requer muita comunicação, o que é custoso.

O segundo algoritmo se propõe a usar o mapeamento checkerboard, que em vez de passar linhas ou colunas à cada processo, passa submatrizes quadradas.

No **algoritmo de Fox**, assumimos que cada processo armazena um único elemento da matriz em vez de uma submatriz inteira usando parâmetros extras de ordem das submatrizes e ordem da malha de processos. Processos vistos como uma malha virtual, e certas malhas como universos de comunicação. Os universos de comunicação são os comunicadores do MPI, e as malhas virtuais correspondem a um tipo de topologia de processos do MPI.

São dois os tipos de comunicadores do MPI, **intra-comunicadores** e **inter-comunicadores**. Os intra podem ser usados em MPI_Send/Recv e em comunicação coletiva. Os inter são usados para comunicação entre processos pertencentes à comunicadores diferentes. Os intra consistem de grupo e contexto. Um grupo é uma coleção ordenada de processos. Um contexto é uma etiqueta única associada a um grupo de um comunicador que serve para checar os argumentos do comunicador nas funções de comunicação. Uma mensagem só pode ser comunicada se o contexto do argumento do comunicador usada pelo processo que a envia é o mesmo da do processo que a recebe. É diferente da tag, o contexto é definido pelo sistema.

Grupos e **comunicadores** são objetos opacos, só acessados pelo usuário através de funções especiais do MPI, como **MPI_Comm_create**, **MPI_Group_incl** e **MPI_Comm_group**. Contextos são inacessíveis ao usuário.

Existem alguns métodos para criar comunicadores: o mais básico é criar um grupo e deixar o sistema associar um contexto a ele. Isto se dá através de três passos, sendo o primeiro obter o grupo do comunicador que contém os processos que queremos e atribuí-lo ao novo comunicador com **MPI_Comm_group**. O segundo criar um array listando os ranks dos processos no grupo antigo de processos que queremos dele. Para criar o novo grupo usamos **MPI_Group_incl**. Após termos nosso novo grupo, o terceiro e último passo é associar um contexto a ela, chamando **MPI_Comm_create**. Vale dizer que as duas primeiras função não envolvem comunicação, enquanto a última é uma função de comunicação coletiva, envolvendo todos os processos do comunicador antigo.

Para dividir um comunicador em vários **subcomunicadores** podemos usar MPI_Comm_split, sendo o número deles dado pelo segundo parâmetro. Se dois processos no mesmo comunicador após o split, seus ranks são definidos pelo terceiro argumento. É uma operação coletiva.

Topologias de processos nos permite endereçar processos. São duas topologias, em grafo ou em malha. Na de malhas, cada processo são vértices num retângulo de qualquer dimensão. É possível associar atributos a comunicadores por um processos chamado caching. Para se criar um comunicador com topologia em malhas em cache usa-se **MPI_Cart_create**. Isto cria um comunicador cujos processos possuem um sistema de coordenadas em cache. Podemos acessar esse sistema com a funções **MPI_Cart_coords** e **MPI_Cart_rank**. A primeira toma o rank to processo e retorna suas coordenadas no grid. A segunda retorna o rank do processo dadas suas coordenadas. Mpi fornece a possibilidade de criar-se submalhas, assim como MPI_Comm_split, chamada **MPI_Cart_sub**, que é coletiva.

Há ainda a função de comunicação ponto-a-ponto, chamada MPI_Sendrecv_replace, que realiza tanto um envio e recebimento através de um buffer.

4. Solução dos exercícios

1. Suponha MPI_COMM_WORLD = p = mn processos. Podemos ver como uma malha de m linhas por n colunas, sendo a primeira linha de {0,1,..,n-1}, a segunda de {n, n+1,...,2n-1}

a) Resolvido a partir do programa comm_create.c. As maiores modificações foram na obtenção dos ranks para criação dos grupos e broadcast.

b) Resolvido a partir de comm_split. A única modificação foi em my_col = my_rank%q.

c) Seriam iguais, já que em ambos associamos os ranks 0,n,2n,3n,...

2. Um comunicador é um conjunto de grupos. Cada grupo pode ser um array cujas entradas são os ranks dos processos em MPI_COMM_WORLD. Cada processo tem uma lista de contextos.

a) Primeiro construir um inteiro único, representando o contexto do comunicador. Então seria necessário ao criar um processo de um grupo determinar o inteiro único para ele dentre os existentes no comunicador.

b) Usaria o split_key para construir o array de comunicadores. Seria necessário também um único inteiro para representar o contexto do comunicador.

3. No algoritmo de Fox, seria irrealista esperar que o sistema fornecesse n^2 processadores, então fizemos com que cada processo armazenasse submatrizes de ordem n/sqrt(p). Já no algoritmo genérico, uma linha por processo também é irrealista. Foi então modificado para suportar um conjunto de linhas. Compare a necessidade de armazenamento entre os dois.

No básico, as matrizes foram distribuídas em blocos em forma de tabuleiro pelos processos. Ele foi modificado de armazenar apenas um elemento por processo para armazenar um conjunto de linhas igual ao tamanho do lado do tablado dividido pelo número de processos, sendo que têm de serem divisíveis um pelo outro. Já no Fox, é necessário que cada processo armazene três submatrizes, sendo A, B e C. Assim, pode-se ver o algoritmos básico requer menos memória, mas há um custo maior por transmissão de mensagens, enquanto que o de Fox necessita de mais memória para armazenamento, mas menos transmissão de mensagens.

4. Um programa usa topologia de três dimensões. Se são l,m e n, use MPI_Cart_sub para criar os seguintes comunicadores cartesianos.

Temos que
int MPI_Cart_sub(MPI_Comm comm, const int remain_dims[], MPI_Comm *newcomm)
comm – comunicador com estrutura cartesiana (entrada)
remain_dims[] - vetor que especifica se i-ésima coordenada é mantida ou não (entrada)
*newcomm – comunicador contendo subgrid (saída)

a) m x m dimensional, cada um com ln

```
coordenadas[0] = 1;
coordenadas[1] = 0;
coordenadas[2] = 1;
MPI_Cart_sub(com_entrada,free_coords,&com_saida);
```

b) vetor lm, cada um com n processos

```
coordenadas[0] = 0;
coordenadas[1] = 0;
coordenadas[2] = 1;
MPI_Cart_sub(com_entrada,free_coords,&com_saida);
```

c) vetor lmn, cada um com uma posição

```
coordenadas[0] = 0;
coordenadas[1] = 0;
coordenadas[2] = 0;
MPI_Cart_sub(com_entrada,free_coords,&com_saida);
```

5. MPI_Sendrecv_replace lida com o buffer para o programador. Implemente o shift circular com MPI_Send e MPI_Recv (divida os processos em dois sets, um envia primeiro, o outro recebe primeiro).

```
// Processos pares mandam MPI_Send primeiro
if( my_rank % 2 == 0 )
{
	// Sender manda mensagem messageS para o receiver
	MPI_Send(messageS,strlen(messageS)+1,MPI_CHAR,dest,tag,MPI_COMM_WORLD);
	// Receiver recebe mensagem na viariável messageR
	MPI_Recv(messageR, 100, MPI_CHAR, recv, tag, MPI_COMM_WORLD, &status);
}
// Processos ímpares recebem MPI_Recv primeiro
else
{
	// Receiver recebe mensagem na viariável messageR
	MPI_Recv(messageR, 100, MPI_CHAR, recv, tag, MPI_COMM_WORLD, &status);
	// Sender manda mensagem messageS para o receiver
	MPI_Send(messageS, strlen(messageS)+1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
}
```

6. Conclusão

Foi abordada de forma mais específica restrições de comunicação em comunicadores com grupos e contextos entre processos e também topologias para endereçamento destes processos, que podem ser em grafo ou malha.

7. Referências consultadas

Parallel Programming with MPI by Peter Pacheco ; Comunicators and Topologies – chapter seven.