Introdução à programação paralela

Gabriel Martins de Miranda – 13/0111350

1. Título do capítulo

Advanced Point-to-Point Communication.

2. Objetivo do capítulo

MPI_Send e **MPI_Recv** são bloqueantes. No capítulo, são estudadas as funções não-bloqueantes do MPI. Algumas implementações de **Allgather** são mostradas com funções bloqueantes e são mostradas formas para tornar as funções aprendidas até aqui mais confiáveis e rápidas.

As funções **MPI_Sendrecv** e **MPI_Sendrecv_replace** são apresentadas. Elas têm a característica de evitar deadlocks usando comunicações emparelhadas. Funções não-bloqueantes fornecem um meio de sobreposição de comunicação e computação, além de melhora de performance. São apresentadas as funções **MPI_Isend** e **MPI_Irecv**, não-bloqueantes.

São abordadas solicitações persistentes de comunicação, com possibilidade ainda maior de sobreposição de comunicação e computação.

Ainda são comentados alguns modos de comunicação, sendo que é possível ao programador explicitamente dizer ao sistema se deve-se usar mensagens bufferizadas e onde elas devem ser alocadas.

3. Resumo do capítulo

Foram estudados dois algoritmos para **Allgather** com a programação de hipercubos.

Foram estudadas as funções **MPI_Sendrecv** e **MPI_Sendrecv_replace**. Ambas tanto enviam quanto recebem informação e gerenciam automaticamente a bufferização. A segunda versão utiliza o mesmo armazenamento tanto para envio quanto recebimento. Úteis para troca de dados entre processos ou deslocamento de dados. Com **MPI_PROC_NULL**, troca-se dados com um processo não existente.

Também foi apresentada comunicação não-bloqueante, que provê maior performance. Para isto, são usados **MPI_Isend**, **MPI_Irecv** e **MPI_Wait**. O buffer usado não pode ser modificado, já que ao retornar de **MPI_Isend** ou **MPI_Irecv** a operação pode não ter sido completada - de enviar e receber. Para completar a operação, deve-se chamar **MPI_Wait**.

Com solicitações permanentes de comunicação podemos evitar o custo de recalcular o envelope de mensagens que muito se repetem. No envio, por exemplo, usamos **MPI_Send_init**, **MPI_Start** e **MPI_Wait**. Com **MPI_Send_init**, as configurações do envio são feitas uma única vez, que cria uma solicitação reusável. Para iniciar o envio, usa-se **MPI_Start**. Para completar o envio, chamamos **MPI_Wait** como de costume. O mesmo esquema é usado no recebimento de dados.

Os modos de comunicação MPI são **standard**, **synchronous**, **ready** e **buffered**. Recebimento de dados só usa modo **standard**, enquanto envio pode usar todos eles. No modo **stardard**, bufferização fica a cargo do sistema. No **synchronous**, bufferização nunca é utilizada, sendo que o envio não completa enquanto que o receptor não começar a receber os dados. No **ready**, envia-se sem saber ainda quem é o receptor até que ele se mostre. No modo **buffered**, sempre ocorre bufferização. Para cada um dos modos, existem envio **bloqueante**, envio **não-bloqueante** e envio **persistente**, sendo escolhidos na chamada da função pela adição da inicial maiúscula. No modo **buffered**, é necessário definir e liberar buffers, sendo a cargo do programador. Deve-se usar um buffer por processo.

São mostradas outras funções ponto a ponto úteis, como variações da **MPI_Wait** para completar múltiplas operações não-bloqueantes e **MPI_Test** para testar se a operação não-bloqueante foi finalizada.

4. Solução dos exercícios

Todos os códigos enviados em anexo.

Para compilar em linux:

```
$ chmod a+x build.sh
$ ./build.sh
```

Os códigos de execução estão comentados no mesmo arquivo.

6. Conclusão

Foi aprendido um pouco sobre comunicação não-bloqueante e sobre solicitações permanentes, e como podem aumentar de forma significativa a performance de um programa. Além disso, foram mostrados os modos de comunicação do MPI, como podem ser utilizados e em quais situações.

7. Referências consultadas

Parallel Programming with MPI by Peter Pacheco; Advanced Point-to-PointCommunication. – chapter thirteen.