Introdução à programação paralela

Gabriel Martins de Miranda – 13/0111350

1. Título do capítulo

Collective Communication. (Comunicação coletiva, Árvores, Broadcast, Redução, Gather. Scatter).

2. Objetivo do capítulo

Explorar alguns problemas encontrados no programa de cálculo de integral usando regra dos trapézios da seção anterior. Problemas de processos ociosos à esperar de outro, que pode estar lendo de entrada e saída. Vários métodos são mostrados para ganhar performance e deixar o mínimo de processos ociosos.

3. Resumo do capítulo

Diversos métodos de comunicação coletiva foram explorados, como Broadcast (um processo envia dados a todos os outros no comunicador), Reduce (cada processo com um operando, sendo todos combinados com um operador binário aplicados sucessivamente a cada um), Gatter (estrutura de dados distribuída reunida por um único processo), Scatter (estrutura de dados armazenada num único processo e distribuída aos outros).

Tanto o Broadcast quanto a redução podem ser melhoradas usando uma estrutura de Tree e percorrendo seus ramos, cuja eficiência depende da topologia do sistema.

As novas funções aprendidas no capítulo são:

A raiz envia a mensagem completa pra todos os outros processo.

int MPI_Bcast (message, count, datatype, root, comm)
void* message; - mensagem
int count; - número de entradas no buffer
MPI_Datatype datatype; - tipo de dados do buffer
int root; - rank da raiz broadcast
MPI_Comm comm; - comunicador

Valores de cada operando dos processos combinados usando operador. Resultado em result do processo raiz.

int MPI_Reduce (operand, result, count, datatype, operator, root, comm)
void * operand; - operando
void* result; - resultado
int count; - número de elementos do buffer que manda
MPI_Datatype datatype; - tipo de dados dos elementos enviados
MPI_Op operator; - operação de redução
int root; - rank do processo raiz
MPI_Comm comm; - comunicador

Reúne os dados armazenados em cada send_data de cada processo na memória referenciada por recv_data no processo raiz. É o oposto do Scatter.

int MPI_Gather(send_data, send_count, send_type, recv_data, recv_count, recv_type, root, comm)

void* send_data;
int send_count;
MPI_Datatype send_type;
void* recv_data;
int recv_count;
MPI_Datatype recv_type;
int root;
MPI_Comm comm;

Processo raiz distribui a memória referenciada por send_data para todos os processos em comm. É diferente do broadcast, pois aqui pedaços diferentes da mensagem são distribuídas pelos diferentes processos, enquanto lá é a mensagem inteira. É o oposto do Gather.

int MPI_Scatter(send_data, send_count, send_type, recv_data, recv_count, recv_type, root, comm)
void* send_data;
int send_count;
MPI_Datatype send_type;
void* recv_data;
int recv_count;
MPI_Datatype recv_type;
int root;
MPI_Comm comm;

Não tem tags, são sincronas. Usar mesmo argumento para dois parâmetros diferentes é ilegal no MPI de for de saída ou entrada/saída. Deadlock é quando todos os processos esperam uns pelos outros.

As vezes é desejável que o resultado de um Gather ou Reduce seja disponível para todos os processos no comunicador. Vimos também o padrão de estrutura Butterfly, que pode ser mais eficiente que os usados normalmente. MPI possibilita sua utilização com as funções a seguir:

Todos os processos com o resultado da redução.

int MPI_Allreduce (operand, result, count, datatype, operator, comm)
void * operand; - operando
void* result; - resultado
int count; - número de elementos do buffer que manda
MPI_Datatype datatype; - tipo de dados dos elementos enviados
MPI_Op operator; - operação de redução
MPI_Comm comm; - comunicador

Todos os processos com o resultado da reunião.

int MPI_AllGather(send_data, send_count, send_type, recv_data, recv_count, recv_type, comm)
void* send_data;
int send_count;
MPI_Datatype send_type;
void* recv_data;
int recv_count;
MPI_Datatype recv_type;
MPI_Comm comm;

É importante notar também que todos os métodos de comunicação coletiva são pontos de sincronização. Isto significa que só são executados quando todos os outros processos o executarem esperando o mesmo root. Caso contrário, a execução do código fica em espera.

4. Solução dos exercícios

Foi usado o MPICH, implementação de alta performance e portabilidade do MPI.

1. O broadcast comum funciona apenas na forma linear, ou seja, não há paralelismo já que o processo principal envia toda a informação aos outros. Logo, no caso de oito processadores comunicáveis linearmente, o broadcast linear é a melhor opção. Nos outros casos, é possível usar de paralelismo, logo o padrão de Tree é melhor, que trabalha bem na forma planar e cúbica. O padrão butterfly é o melhor para o caso do cubo.

2. Antes de fazer a análise do código, é importante lembrar que as funções de broadcast utilizadas são síncronas. Será utiliza a notação Pi para se dirigir ao processo i. Quando o programa inicia, cada processador executado seu case distinto, de acordo com o rank. Como todos possuem **MPI_Bcast** com root em 0, cada um a executa e atualiza seus valores do primeiro parâmetro de acordo com o que é enviado pelo root, no caso o P0. P0 mantém seu valor x, enquando que P1 atualiza seu x para o x do P0 e P2 atualiza seu z para o x do P0. Na segunda linha, apenas P1 tem uma chamada de **MPI_Bcast**, logo fica em espera. P0 tem um **MPI_Send** para P2 na tag 43 e P2 tem um **MPI_Recv** de P0 na tag 43. Assim, o x de P2 recebe o y de P0. P0 e P2 vão para a próxima linha, sendo para ambos um **MPI_Bcast** recebendo de P1. Como P1 estava em espera para poder enviar seu broadcast, todos executam juntos. O z de P0 recebe o y de P1, enquanto que o y de P2 recebe o y de P1 também. Ao final, os valores x, y e z de cada processo são:

P0: x = 0, y = 1, z = 4
P1: x = 0, y = 4, z = 5
P2: x = 1, y = 4, z = 0

3. Ok. Agora os dados lidos do usuário por P0 são enviados no padrão Tree para os outros processos. Isso significa que não será apenas P0 a enviar o dado linearmente para todos os outros processos, mas agora alguns deles irão compartilhar entre si, permitindo certo paralelismo.

4. Ok. Agora os dados lidos por P0 são enviados aos outros processos através da função pronta do MPI chamada MPI_Bcast, que vai utilizar um padrão otimizado pela configuração do sistema para enviar os dados. Pode-se ver o código fica bem mais simples.

5. Temos então uma matriz 2x2 multiplicada por um vetor de tamanho 2x1. Supondo a matriz [a b;c d] e o vetor [e ; f], nossa resposta é um vetor 2x1, [a*e + b*f ; c*e + d*f]. Temos que o scatter pega um dado completo de um processo raiz e distribui em partes para os outros processo. Já o reduce reúne operandos de diversos processos e obtém um resultado usando um operador. Fazendo um scatter de cada linha da matriz inicial, podemos ter um processo P1 com [a b] e um processo P2 com [c d], ambos tendo conhecimento do vetor [e ; f]. Então P1 faria a operação [a*e + b*f] e P2 faria [c*e + d*f]. Ao final, o vetor final seria obtido. Poderia-se também dividir em 4 processos. P1 com a , P2 com b, P3 com c e P4 com d. P1 faria a*e, P2 b*f, P3 c*e e P4 d*f. Com reduce usando operação soma, uniria-se P1 a P2 e P3 a P4. Ao final teríamos o vetor resultado.

6. Scatter pega um dado completo de um processo raiz e distribui em partes para os outros processo. Já o gatter é o inverso, pega dados picados de vários processos e os reúne num dado só completo num processo raiz. Com AllGatter, todos os processos possuem o valor final formado.

5. Trabalhos de programação

1. Escreva duas funções de saída: uma função de saída de vetor e uma função de saída de matriz distribuídas por blocos de linhas. A entrada do programa deve ser apenas a ordem da matriz, sendo que o programa gera as matrizes e vetores com 1’s.

2. Subprograma que calcula soma glabal de coleção distribuída de floats. Resultado da soma global retornada como operando.

3. Subprograma para produto paralelo entre matrizes.

6. Conclusão

Por meio do capítulo foram aprendidas diversas funções com comunicação coletiva, e como são executadas como pontos de sincronização. Foram aprendidas técnicas para aumentar a performance com Tree e Butterfly, além de operações comuns paralelizadas.

7. Referências consultadas

http://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/
Pacheco, P. S., (1997) Parallel Programming with MPI. Morgan Kaufmann.

Obs.: As primeiras linhas nos fontes anexados indicam como compilar.