Relatório Do Trabalho Prático

Problema do caminho mais curto entre todos os pares

Computação Paralela 2019/2020

José Pedro Ribeiro Ferreira Pinto – 201603713

1-Descrição do problema

O problema em questão é o de encontrar o caminho mais rápido num grafo entre todos os pares de nós, isto é, para qualquer nó inicial e nó final encontrar o caminho mais curto do início ao fim.

1.1-Problema de paralelização

Com o objetivo de melhorar a performance do programa e obter experiência pratica num programa com paralelismo foi implementado o algoritmo de fox(Paginas 29-30).

1.1.1-Algoritmo de fox

O algoritmo de fox é um algoritmo de paralelização de operações entre matrizes.

Para poder ser usado o número de processos (ou threads) tem de ser um quadrado perfeito (a sua raiz quadrada é um número inteiro) e a raiz quadrada desse número tem de ser um divisor do tamanho das matrizes.

Inicialmente cada uma das duas matrizes é subdividida em matrizes de menor dimensão. E estas têm todas as mesmas dimensões. Posteriormente cada processo recebe de cada matriz original uma das suas submatrizes.

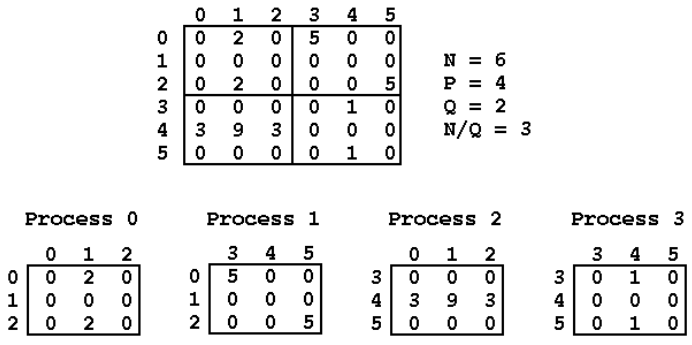

Imagem 1: divisão de uma matriz.

O algoritmo corre em vários passos, com o número de passos igual á raiz do número de processos. Em cada passo é escolhido um processo de cada linha. Cada processo escolhido envia para todos os outros, na sua linha, a sua primeira matriz e de seguida faz a operação de matrizes com as suas próprias matrizes. Os processos que recebem a matriz fazem a operação entre a sua segunda matriz e a que receberam. De seguida todos os processos enviam a sua segunda matriz para o processo acima (no caso de estar na primeira linha envia para a última). Ao receber

a nova matriz substituem a segunda das suas matrizes pela nova e fazem a operação de matrizes entre as duas matrizes que têm.

No primeiro passo os processos escolhidos são os da diagonal e em cada passo posterior os processos escolhidos são os diretamente á direita (se estiver na última coluna é o da primeira que é escolhido).

Todos os resultados são guardados numa terceira matriz de saída que cada processo tem.

2-Implementação

O programa está dividido em quatro partes principais, ler a matriz do stdin, dividir a matriz por cada um dos processos, fazer o algoritmo de fox o número de vezes necessárias e voltar a juntar cada uma das matrizes calculadas para escrever o resultado.

2.1-Estruturas de dados

Diversas estruturas de dados foram implementadas para organizar os dados.

SquareMatrix (matriz quadrada)

Utilizada para guardar as matrizes. Uma vez que para o problema em questão todas as matrizes são de n por n a estrutura foi simplificada, não permitindo ser de n por m, para m diferente n.

Várias funções relativas às matrizes foram criadas, para alocar, escrever, copiar e modificar todos os valores.

Na criação da matriz é alocada memoria contigua para os valores da matriz e um vetor de apontadores, com cada apontador a apontar para uma linha da matriz.

Comms (comunicadores)

Devido é estrutura do problema foi criada uma estrutura para os comunicadores, guardando os comunicadores para a rede, linha e coluna.

Também foram criadas as funções para alocar e escrever os comunicadores.

Os comunicadores foram criados usando topologia de redes cartesianas.

Ranks (Posições)

Para facilitar a obtenção da posição de cada processo num comunicador foi criada a estrutura de posições, com as posições da rede, linha e coluna.

A posição no comunicador de linha é o número da coluna, enquanto que o a posição no comunicador de coluna é o número de linha.

FoxData (informação para algoritmo de fox)

O algoritmo de fox precisa de bastantes dados e como tal foram aglomerados nesta estrutura. A estrutura tem as duas matrizes, os comunicadores e as posições. Também tem os dados sobre o número de processos, o tamanho das matrizes e de quem e para quem são enviadas as matrizes na mesma coluna.

2.2-Leitura de dados

Os dados são lidos do stdin no seguinte formato:

- Na primeira linha o tamanho da matriz e nas linhas seguintes as linhas da matriz (uma linha da matriz por linha).

Os dados são guardados numa matriz quadrada e depois verificados. A verificação confirma que o número de processos é adequado para executar o algoritmo de fox nesta matriz.

2.3-Divisão da matriz

Primeiro o tamanho da matriz é enviado para cada processo usando o Bcast (para estes poderem alocar memoria para a matriz).

A matriz é dividida usando o MPI_TYPE_VECTOR. O tipo é definido tal que uma linha da submatriz é escolhida e depois salta para a linha seguinte.

Cada uma das submatrizes e enviada para cada processo usando o bsend. Utilizar o Bcast seria mais eficiente e seria possível utilizando uma forma diferente de guardar as matrizes em memoria. No final manter o bsend foi escolhido uma vez que tornou o código mais legível.

2.4-Realização do algoritmo de fox

O algoritmo de fox tem de ser realizado log2(tamanho da matriz) vezes para se obter o resultado final(porquê).

No algoritmo, ao contrário do mais usual, não é utilizada a multiplicação de matrizes comum, mas sim uma operação semelhante em que as operações de multiplicação e soma são substituídas pelas de soma e mínimo, respetivamente (operação min-plus). Uma vez que a operação é diferente, a cada passo do algoritmo, ao invés de se acumular os valores na matriz de saída, o mínimo entre os novos valores e os obtidos anteriormente é feito.

Os valores de 0 na matriz podem ter dois significados. Se estiverem na diagonal, o valor representa realmente 0, por outro lado os valores fora da diagonal representam infinito. Como tal os valores 0 fora da diagonal são substituídos por um valor elevado (para representar infinito). Após os cálculos serem concluídos todos os valores de infinito são substituídos por 0.

2.5-Junção das matrizes

Apos o algoritmo ser concluído as matrizes de saída são enviadas para o processo raiz usando o bsend num processo para serem imprimidas. De modo semelhante à divisão das matrizes usar gather seria mais eficiente, mas bsend foi mantido pelo mesmo motivo.

2.6-Notas sobre execução

Várias constantes foram definidas para facilmente controlar a execução do programa.

INF – O número que representa infinito, no caso de os valores da matriz serem demasiado elevados este valor pode ser aumentado

BUFFERSIZE – No caso de uma mensagem de erro aparecer a afirmar que não há espaço no buffer, aumentar este valor.

BECHMARK – 0 não está em modo de medição; 1 está em modo de medição.

BENCHMARKSIZE – No caso de o programa estar em modo de medição o algoritmo de fox será corrido este número de vezes e o tempo de execução será mostrado.

Foi enumerado um número de tags para a comunicação para impedir colisão de mensagens e para mais facilmente detetar error. Cada tag é usada numa secção distinta.

3-Análise de performance

	Número De Processos				
Tamanho Das Matrizes	1	4	9	16	25
5	0.134ms	NA	NA	NA	NA
6	0.275ms	0.354ms	0.552ms	NA	NA
300	5040.0ms	956.46ms	747.94ms	536.29ms	563.25ms
600	44783ms	9872.8ms	7229.0ms	3633.8ms	2851.1ms
900	158517ms	35598ms	23612ms	12044ms	10085ms
1200	400291ms	92935ms	60831ms	33368ms	26102ms

Tabela 1: Tempo de computação para matrizes de diversos tamanhos e diversos números de processos (o valor é a media de 3 medições)

Tal como esperado aumentar o número de processos aumentou bastante a performance na maioria dos casos, tendo o maior impacto nas matrizes de maior dimensão. No entanto há que salientar o caso da matriz de tamanho 6 em que aumentar o número de processos aumenta o tempo demorado. Isto acontece, pois, a computação de cada processo é pequena, mudando pouco, e o tempo de comunicação aumenta. Também se pode observar algo semelhante na matriz de tamanho 300, no entanto apenas se observa o impacto a partir de 25 processos.

Algo estranho a notar é o facto de ao aumentar o número de processos de 1 para 4 o tempo de processamento diminui em mais de 4 vezes. Uma análise ao código não revelou o motivo, este permanecendo um mistério.

Nota: os valores de NA significam que o número de processos não é valido para o tamanho da matriz. Uma vez que 5 é primo o único numero de processos válido é 1.

4-Conclusão

Com este trabalho foi possível observar as vantagens do paralelismo, mas também as suas desvantagens. Se não for feito corretamente, não só não se tem um aumento de performance, mas também esta pode deteriorar-se. Também no caso em que a performance não é essencial, a dificuldade extra de implementação, assim como o tempo de o fazer tornam o paralelismo uma grande desvantagem. Em algumas situações pode não existir mais do que um processador, tornando o paralelismo irrelevante.