Universidade Federal de São João del-Rei
DEpartamento de Ciência da Computação
Programa de Graduação

Relatório sobre a implementação do algoritmo de Floyd-Warshall em paralelo usando a biblioteca OpenMPI

Aluno: Jardel Felipe de Carvalho
Professor: Rafael Saquetto Oliveira

Outubro
2019

Conteúdo

1 Descrição

Este relatório é parte dos requisitos necessários para a conclusão do primeiro trabalho prático da disciplina de Computação Paralela oferecida pela Universidade Federal de São João del-Rei.

Para o trabalho em questão foi proposto a implementação do algoritmo de Floyd-Warshall em paralelo com a utilização da biblioteca OpenMPI. O propósito é realizar uma compararação de desempenho em função do volume dos dados de entrada e do número de processos em uso.

Os resultados obtidos assim como a descrição da metodologia serão especificados e discutidos.

2 Resumo

2.1 Floyd-Warshall

Entre as soluções para problemas de menor caminho entre dois vértices de um grafo se encontra o algoritmo de Floyd-Warshall. Ao contrário do algoritmo de Djikstra o algoritmo de Floyd-Warshall produz resultados plausíveis para grafos com arestas de peso negativo.

Seu funcionamento requer uma matriz ou lista de adjacência como entrada, mediante a isto é realizada a busca por vértices t que possam intermediar o acesso entre os vizinhos u e v e ao mesmo minimizar as suas distâncias dado os pesos das arestas.

O algoritmo de Floyd-Warshall também pode ser executado sob uma aborgem parecida com uma multiplicação de matrizes, a menor distância entre cada vértice é dada pela matriz *Df*, sendo *D2 = D1 x D1*, *D4 = D2 x D2*, *D8 = D4 x D4 ... Df = Dg x Dg*, $f \geq N$ e $g < N$. Ao contrario da multiplicação de matrizes convencional o algoritmo de Floyd-Warshall realiza operações de soma e mínimo ao invés de multplicação e soma.

2.2 Fox

O algoritmo de Fox propõe uma estratégia de multiplicação de matrizes por meio da troca de dados entre nós. Seu funcionamento imita uma multiplicação convencional, entretando, com o particionamento dos dados o algoritmo de Fox torna possível a atuação de múltiplos processos na busca por resultados.

As condições para que o algoritmo de Fox seja aplicado sobre uma matriz de dimensão N dado P processos disponíveis é que P seja um quadrado perfeito e que N mod $\sqrt{P}$ seja igual a zero.

As sua execução é separada em quatro partes. Primeiro é realizado o particionamento da matriz inicial em pedaços B de dimensão N / $\sqrt{P}$ que serão associados e enviados à cada um dos processos respectivamente. O particionamento organiza os processos em linhas de forma que suas numerações se dão na forma RTL (Right To Left Script). Cada processo inicialmente realiza uma cópia R da matriz B.

No segundo passo, é escolhido para cada linha um processo que será responsável por compartilhar a sua submatriz R com os seus vizinhos laterais. Feita a escolha é realizada a troca de dados para com os vizinhos, a matriz recebida é denotada por A.

No terceiro passo, é realizada a multiplicação da matriz recebida A, pela matriz B residente em cada processo atualmente.

No quarto passo a matriz B é enviada para o processo logo acima, assim como é recebida a matriz B do proesso logo abixo.

Os procedimentos dois ao quatro são repetidos $\sqrt{P}$ vezes e o *rank* da matriz ecolhida no segundo passo é denotado por $u = (r + i) \mod \sqrt{P}$ em que r é a linha na qual está sendo verificado valor de u e i é o número da iteração atual.

2.3 MPI

O MPI (Message Passing Interface) é uma especificação que expressa um conjunto chamdas e estruturas de dados uteis para a comunicação entre processos paralelos com memória distribuída. Para este trabalho foi utilizada a implementação OpenMPI para a linguagem C.

3 Abordagem

Sabe-se que algoritmo de Floyd-Warshall pode também trazer resultados com a multplicação sucessiva de matrizes, sendo assim, para trazer a execução para o formato paralelo com OpenMPI, foi utilizado o algoritmo de Fox em cada uma das multiplicações.

4 Implementação

4.1 Comunicação Entre Processos

Foi optado por não fazer o uso de comunicadores na implementação proposta. Sendo assim, algumas operações foram realizadas com base no *rank* do

processo com o intuito cumprir a tarefa de identificação de nós destinatários e nó remetente.

4.2 Vizinhos Laterais

Cada processo escolhido para enviar sua submatriz a seus vizinhos laterais identificou os destinatários com base no seguinte cálculo.

$$q \times r \leq rank < q \times r + q \tag{1}$$

Onde $q = \sqrt{p}$, $p = num\_procs$ e r é a linha do processo escolhido. Aqueles que farão o recebimento na linha r em questão identificam seu remetente conforme a seguir.

$$r \times q + u \tag{2}$$

Vale ressaltar que com esta inequação o processo escolhido pode enviar sua submatriz para si mesmo, o que pode ser um desperdício de recursos. Devido a isto, estruturas condicionais foram inseridas para privar a execução de tal evento.

5 Vizinhos Verticais

Ao fim de cada iteração cada processo com número de identificação *my_rank* compartilha a matriz B com o processo diretamente acima assim como recebe do processo diretamente abaixo. A determinação do destinatário logo acima é dada pelo seguinte cálculo.

$$up = (my\_rank + q \times (q - 1))\%p \tag{3}$$

A determinação do remetente logo abaixo é dada pelo seguinte cálculo.

$$up = (my\_rank + q)\%p \tag{4}$$

Onde *my_rank* é o número de identificação do processo.

6 Análise dos Resultados

6.1 Casos de Teste

Para os testes foram oferecidas para o algoritmo implementado entradas de tamanho $\sqrt{N}$ = 12, 24, 48, 96, 192, 384, 768 e 1536. Cada uma das entradas foram resolvidas 10 vezes por execuções com p =1, 4 e 9 processos.

6.2 Tempos de Execução

Os resultados obtidos mostraram que *T(1)* tende a ser muito maior a medida que se aumenta o tamanho da matriz a ser multiplicada, entretanto para dimensões menores os tempos *T(1), T(4) e T(9)* tendem a ser equiparáveis, mas, com um leve aumento de tempo para *T(4) e T(9)*.

Os tempos *T(4) e T(9)* se mostraram muito eficientes para matrizes maiores, sendo *T(9)* o menor tempo.

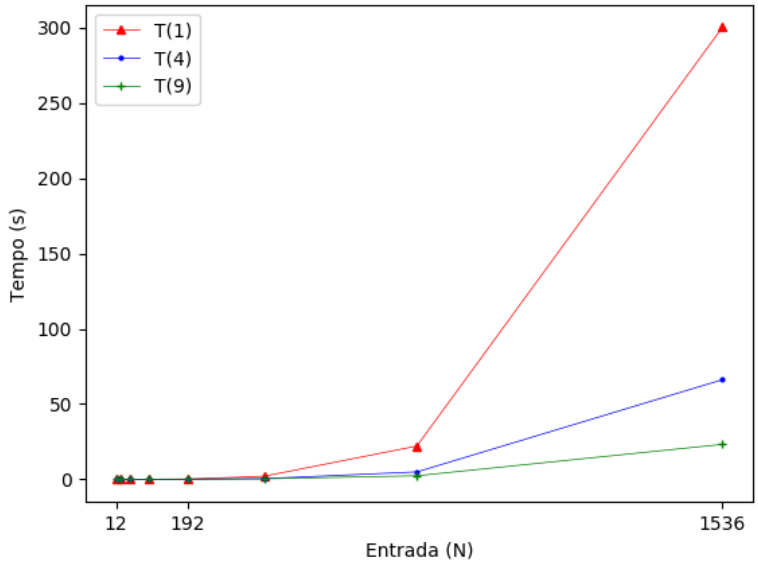

Figura 1: Tempo de execução por tamanho de entrada.

6.3 Speedup

Percebe-se que para $p = 9$ o Speedup tendeu a aumentar em conjunto com o tamanho da entrada. Para os testes em que $p = 4$ o Speedup tendeu a estagnar para os três últimos casos de teste onde $p = 9$ mostrou a sua maior pontuação.

É possível perceber a similaridade de tempos onde $N = 12$ *e* 24, porém, nota-se uma menor pontuação neste intervalo para execuções com mais de um processo.

Para os testes intermediários, onde $N = 48$, *96 e* 192 tem-se que o desempenho para $p = 4$ se apresentou como o mais elevado dos três tipos de execução.

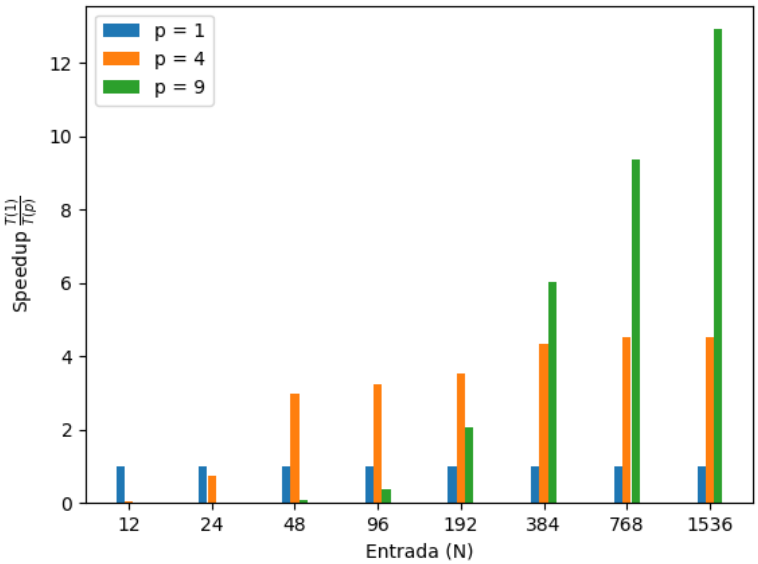

Figura 2: Speedup de cada número de processos por tamanho de entrada.

N | p	T(1)	T(4)	T(9)
12x12	0.000086	0.003174	0.055245
24x24	0.000536	0.000739	0.059596
48x48	0.004532	0.001527	0.064338
96x96	0.024773	0.007664	0.067989
192x192	0.191792	0.054032	0.092463
384x384	2.033856	0.467259	0.338184
768x768	22.127797	4.911317	2.358002
1536x1536	300.394565	66.214253	23.243098

Figura 3: Speedup de cada número de processos por tamanho de entrada.

7 Dificuldades Obtidas e Comentários

Inicialmente, para o desenvolvimento deste trabalho, foi optado por soluciona-lo sem o auxílio de algoritmos já consolidados, entretanto, foi verificada uma

grande dificuldade em se manusear todas as variáveis do problema, sendo estas, a multiplicação consistente de matrizes e o assincronismo entre processos.

Mediante a isto, recorreu-se ao algoritmo de Fox, do qual, simplificou consideravelmente a solução. O autor do documento de base[**1**] para o entendimento do algoritmo, menciona que a tarefa de implementação com o uso de comunicadores pode tornar o caminho mais fácil, entretanto, para este trabalho, foi observado que a identificação dos processos pertencentes a cada comunicador deveriam ser determinados mediante a algum cálculo, tendo em vista este aspecto, foi optado por realizar a comunicação somente através de chamadas *send* e *recv* utilizando os cálculos acima mencionados de maneira direta.

Em linhas gerais, para este trabalho, a maior dificuldade percebida foi a manipulação de todas as variáveis que compõe o problema, sendo estas o tempo levando em consideração o assincronismo dos processos, o particionamento e consistência dos dados como as transformações aplicadas no decorrer da execução e por fim a manutenção da lógica de multiplicação de matrizes.

Todo este conjunto precisa ser satisfeito para o resultado final ser correto e vale ressaltar que a inserção do fator tempo dificulta de maneira significativa o problema.

Bibliografia

[1] PACHECO, Peter S. A User's Guide to MPI $< https://www.lrz.de/services/software/parallel/mpi/mpi_guide.pdf >$