SSC 0903 - Computação de Alto Desempenho (2023/2)

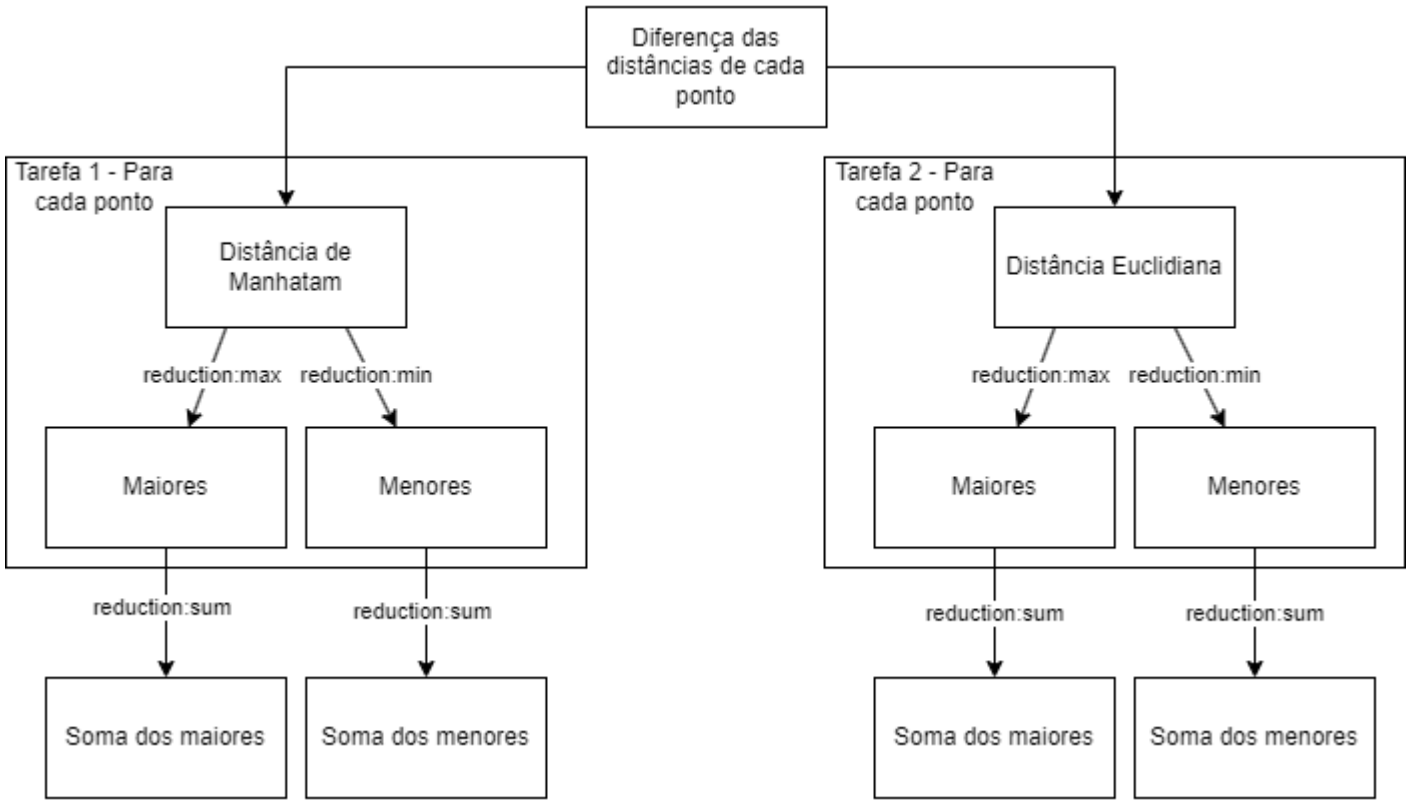
Grupo 3

Pedro Henrique Conrado F. de Oliveira Eduardo	11819091
Figueiredo Freire Andrade	11232820
Milena Correa da Silva	11795401
Olavo Morais Borges Pereira	11297792

PCAM

Particionamento

O método de particionamento escolhido foi inicialmente um **particionamento de dados com certo grau de paralelismo de tarefas**, já que a especificação menciona diferentes cálculos a serem feitos.



A definição das duas tarefas de cálculo das distâncias são iguais e seguem o seguinte formato:

1. Calcular a diferença de cada coordenada entre todos os outros pontos da matriz ($3(N - 1)^2$ tasks)

2. Somar a diferença de cada coordenada calculada acima de acordo com as especificações do método (Manhattan e Euclidiana), encontrando então a distância de cada ponto para os demais ($(N-1)^2$ tasks).
3. Encontrar a menor e maior distância de cada ponto e armazená-las para cálculo futuro ($(N-1)^2$ tasks)

Com isso finalizados, basta:

1. Somar todos os valores do vetor das maiores distância (N^2 tasks)
2. Somar todos os valores do vetor das menores distância (N^2 tasks)

Vale ressaltar que, como a primeira parte das tarefas é igual para os dois métodos, só iremos replicar a partir do 2 ponto, ou seja, no final teremos $N^2 * (3(N-1)^2 + 2(N-1)^2 + 4(N-1)^2) + 2N^2 + 2N^2$ tasks

Comunicação

Como podemos ver pelo diagrama mostrado anteriormente, tirando as duas primeiras comunicações de ordem $(N-1)^2$, teremos mais quatro comunicações para cada ponto ordem $log((N-1)^2)$ já que estará sendo realizada reduções para cada ponto e outras quatro de ordem $log(N^2)$ para as operações de redução por soma. Isso totaliza um custo de comunicação de $N^2 * 2(N-1)^2 + N^2 * 4log(N) + 4log(N)$

Aglomeração

Dentro da estrutura do nosso sistema (MIMD com memória distribuída, em que cada nó é MIMD com memória compartilhada), a criação de uma única tarefa para calcular as distâncias para um ponto isolado revela-se inviável. Dessa forma, decidimos distribuir a carga de trabalho de maneira equitativa, atribuindo a cada nó a responsabilidade por um subconjunto de pontos. Para ilustrar, consideremos, sem perda de generalidade, que np=4 (número de nós). Assim, cada máquina assume a tarefa de calcular as distâncias para um quarto dos pontos, efetuando a soma das menores e maiores distâncias. Além disso, cada máquina identifica as distâncias mínimas e máximas dentro desse quarto de pontos.

Ao término dessa etapa, os valores calculados são enviados para a Máquina 0. Nesse ponto, a Máquina 0 realiza a consolidação dos somatórios, seleciona a menor das menores distâncias e a maior das maiores distâncias, proporcionando uma conclusão eficiente do processo distribuído.

Dentro de cada nó, usamos a criação de threads para permitir o cálculo da distância entre múltiplos pares de pontos de forma paralela.

Mapeamento

Como dito anteriormente, optamos por distribuir os pontos entre os nós do cluster. Dentro de cada nó, implementamos o uso de threads para realizar, de forma paralela, o cálculo da distância entre diversos pares de pontos. Com o intuito de manter um equilíbrio na carga de trabalho, os pontos foram distribuídos de forma alternada entre as máquinas em um estilo *round-robin*. Dessa maneira, evitamos que uma máquina seja responsável apenas pelos pontos que exigem mais esforço (os localizados no início da matriz) ou apenas pelos mais simples (os pontos no final da matriz).

Além disso, em cada nó, implementamos um balanceamento dinâmico entre as threads. Isso foi feito para assegurar uma distribuição uniforme das iterações do loop 'for' e equilibrar o esforço entre as threads. Dessa forma, focamos em otimizar o desempenho e garantir uma execução eficiente.

Por fim, durante a execução, executamos o código com a flag *-bind-to* configurada como *'socket'*. Dessa forma, os processos MPI são vinculados a um processador, ao contrário da configuração padrão, que associa o processo a um núcleo específico do processador. Essa abordagem permite que o processo seja escalonado de forma mais frequente, uma vez que não está restrito a um único núcleo, e ainda assim evita o custo elevado de trocar para outro processador. Ao adotar essa prática, alcançamos um desempenho maior com o uso de OMP e a criação de threads em cada nó do cluster.

Comparação dos Resultados

Para avaliar o desempenho do algoritmo paralelo, realizamos a medição do tempo de execução do algoritmo sequencial para diversos valores de N. Em todas as iterações, o valor da seed foi fixado em 42 para garantir consistência. Vale a pena destacar que o tempo apresentado é resultado da média obtida a partir de três execuções com os mesmos parâmetros e que os resultados podem variar dependendo do uso do cluster.

Tempo Sequencial	
N	**Tempo (s)**
50	0.261
100	4.003
200	63.936

Para fins de cálculo do Speedup relativo, também calculamos o tempo de execução quando rodamos o algoritmo paralelo em um nó com uma thread

Tempo Paralelo NP=1, T=1	
N	**Tempo (s)**
50	0.376
100	4.206
200	66.896

As métricas usadas para medir o desempenho do algoritmo paralelo foram:

- Speedup Absoluto
  - $S_p = \frac{Tempo\ sequencial}{Tempo\ paralelo\ p}$
- Speedup Relativo
  - $S_p = \frac{Tempo\ paralelo\ 1}{Tempo\ paralelo\ p}$
- Eficiência
  - $E_p = \frac{Speedup\ Absoluto_p}{p}$, com $p = NP \cdot T$

Abaixo seguem as tabelas com o tempo de execução para diferentes valores de N, NP e T:

Tempo Paralelo NP=1, T=2				
N	**Tempo (s)**	**Speedup Abs**	**SpeedUp Relativo**	**Eficiência**
50	0.241	1.083	1.560	54.15%
100	2.175	1.840	1.934	92.00%
200	32.953	1.940	2.030	97.00%

Tempo Paralelo NP=1, T=4				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0.183	1.426	2.055	35.65%
100	1.212	3.303	3.470	82.57%
200	17.331	3.689	3.860	92.22%

Tempo Paralelo NP=2, T=1				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0.237	1.101	1.586	55.05%
100	2.176	1.840	1.933	92.00%
200	33.091	1.932	2.022	96.60%

Tempo Paralelo NP=2, T=2				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0,186	1.403	2.022	35.07%
100	1.198	3.341	3.511	83.52%
200	17.328	3.690	3.861	92.25%

Tempo Paralelo NP=2, T=4				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0.157	1.662	2.395	20.77%
100	0,948	4.223	4.437	52.78%
200	14.100	4.534	4.744	56.67%

Tempo Paralelo NP=4, T=1				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0,194	1.345	1.938	33.62%
100	1.300	3.079	3.235	76.97%
200	17.317	3.692	3.863	92.30%

Tempo Paralelo NP=4, T=2				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0.172	1.517	2.186	18.96%
100	0.953	4.200	4.413	52.5%
200	13.527	4.723	4.945	59.03%

Tempo Paralelo NP=4, T=4				
N	**Tempo (s)**	**Speedup Absoluto**	**Speedup Relativo**	**Eficiência**
50	0.168	1.554	2.238	9.71%
100	0.970	4.127	4.336	25.79%
200	13.506	4.734	4.953	29.59%

Comentários Finais

Fica claro que a utilização de MPI e OMP possibilita o desenvolvimento de algoritmos mais rápidos, especialmente dependendo do tamanho de N. Embora existam outras abordagens que poderiam reduzir a comunicação entre as máquinas ou equilibrar a carga de trabalho de maneira diferente para aproveitar possíveis disparidades de desempenho entre os sistemas, por exemplo. Acreditamos que este trabalho tenha atingido seu propósito; tivemos a oportunidade não apenas de aplicar os conceitos aprendidos em sala de aula, mas também de explorar a documentação do Open MPI, aprimorando nossa compreensão sobre o funcionamento dessas bibliotecas.