Relatório do EP3 de MAC0219

Renan Ryu Kajihara, NUSP: 14605762

18 de dezembro de 2024

Resumo

O presente relatório descreve as atividades realizadas no terceiro Exercício-Programa da disciplina "Programação Concorrente e Paralela (MAC0219)", que consistiu na paralelização de um programa que calcula o Conjunto de Julia de forma a simular um sistema distribuído, verificando as diferenças observadas entre a execução do programa paralelizado e a versão sequencial, principalmente em relação ao tempo de execução.

Conteúdo

1 Introdução

Com o intuito de observar a diferença de tempo de execução entre versões sequenciais e versões paralelizadas que simulam um sistema distribuído de um mesmo programa, realizou-se a paralelização de um algoritmo que calculava o Conjunto de Julia utilizando MPI. Além disso, efetuo-se uma série de experimentos para verificar a diferença de performance das duas versões.

2 Objetivos

O exercício teve como objetivo a implementação de técnicas de simulação de programas distribuídos na linguagem de programação C, bem como a análise de perfomance entre as versões sequenciais e as versões paralelizadas.

3 Cronograma

O desenvolvimento deste Exercício Programa envolveu algumas etapas-chave, elencadas abaixo:

- (10/12/2024-11/12/2024) - Escrita do arquivo que calcula o Conjunto de Julia sequencialmente.
- (12/12/2024-15/12/2024) - Paralelização da versão sequencial utilizando MPI.
- (15/12/2024-16/12/2024) - Coleta dos dados dos experimentos.
- (17/12/2024-18/12/2024) - Análise dos dados e redação do relatório do Exercício Programa.

4 Métodos

Primeiramente, para conseguir paralelizar o código sequencial, foi necessário instalar o MPI (The Message Passing Interface), para utilizar as funções características do MPI. Além disso, para simular sistemas distribuídos, foi necessário instalar o SimGrid.

5 Análise dos resultados experimentais

5.1 Experimentos com MPICH

Primeiramente, foram executados testes localmente utilizando MPICH, para comparar o desempenho da versão paralela com a versão sequencial no cálculo do Conjunto de Julia, utilizando diferentes tamanhos de imagem e diferentes números de processos.

A tabela do tempo médio do cálculo do Conjunto de Julia, para diferentes tamanhos de imagem, na versão sequencial pode ser vista a seguir:

Tamanho de imagem	Tempo médio do cálculo em segundos	Intervalo de confiança
500	0,2278	0,0026
1000	0,79583	0,0086
2500	4,78068	0,0126
5000	19,07374	0,0354
10000	75,97345	0,1574

Figura 1: Tabela do tempo médio do cálculo do Conjunto de Julia na versão sequencial.

A tabela do tempo médio do cálculo do Conjunto de Julia, para diferentes tamanhos de imagem e diferentes números de processos, na versão paralelizada pode ser vista a seguir:

Tamanho da imagem	Número de processos	Tempo médio do cálculo em segundos	Intervalo de confiança
500	2	0,09683	0,00325
500	4	0,0954	0,00314
500	8	0,06922	0,00257
500	16	0,06882	0,00278
1000	2	0,32331	0,00687
1000	4	0,26711	0,00625
1000	8	0,18134	0,00489
1000	16	0,18619	0,00402
2500	2	1,92752	0,0175
2500	4	1,47693	0,01557
2500	8	0,98234	0,01223
2500	16	1,23492	0,00965
5000	2	7,62138	0,0584
5000	4	5,82547	0,04892
5000	8	4,03926	0,04447
5000	16	3,94635	0,03986
10000	2	30,28368	0,10587
10000	4	22,98476	0,09624
10000	8	15,45797	0,08874
10000	16	15,25758	0,81025

Figura 2: Tabela do tempo médio do cálculo do Conjunto de Julia na versão paralelizada.

Primeiramente, evidencia-se que o código paralelizado com MPI é muito mais rápido do que o

código sequencial. Nesse sentido, observa-se que a versão paralelizada com MPI em relação a versão sequencial foi de 3 a 6 vezes mais rápida, dependendo do número de processos e do tamanho da imagem.

Além disso, é possível observar que, até 8 processos, quanto maior o número de processos, mais rápido o programa realiza o cálculo do Conjunto de Julia. Entretanto, nota-se que a diferença entre o tempo do cálculo do Conjunto de Julia utilizando 8 processos e 16 processos é muito pequena, pois os testes foram realizados em um computador que possui 8 núcleos de processamento. Dessa forma, ao executar o programa com mais de 8 processos, ocorrerá uma alternação da execução dos processos entre os núcleos de processameto, ou seja, os processos, na realidade, não serão executadas simultaneamente.

5.2 Experimentos com SimGrid

5.2.1 Impacto da heterogeneidade dos nós

É importante observar que, pelo arquivo simple_cluster.xml fornecido, havia uma variação na capacidade computacional entre os nós. Dessa forma, nessa seção, analisaremos o impacto dessa heterogeneidade dos nós no desempenho global.

A tabela que mostra o tempo médio do cálculo do processo mais rápido comparado com o tempo médio do cálculo do processo mais lento pode ser vista a seguir:

Tamanho da imagem	Número de processos	Tempo médio do cálculo do processo mais rápido em segundos	Tempo médio do cálculo do processo mais lento em segundos
500	64	0,0054	0,00485
1000	64	0,0054	0,01933
2500	64	0,0313	0,1189
5000	64	0,13396	0,47215
10000	64	0,53397	1,89961

Figura 3: Tabela que mostra o tempo médio do cálculo do processo mais rápido comparado com o tempo médio do cálculo do processo mais lento.

É evidente pela tabela apresentada que, o fato dos processos terem variação na sua capacidade computacional influencia muito no desempenho global. Isso ocorre porque, ao distribuir as tarefas de forma homogênea, os nós com menor capacidade computacional demorarão muito mais tempo para realizar as suas tarefas. Na tabela apresentada, observa-se que os processos mais lentos demoraram, aproximadamente, 3 a 4 vezes mais do que os processos mais rápidos. Dessa forma, em um ambiente distribuído em que os processos não possuem os mesmos poderes computacionais, é necessário que os processos recebam uma quantidade de tarefas proporcional ao seu poder computacional.

5.2.2 Experimentos com cluster homogêneo

Agora, mostraremos experimentos que demonstram o impacto do nível de latência no tempo total de execução do programa, em um ambiente que todos os nós de computação tenham a mesma capacidade computacional (homogêneos).

A tabela que mostra o tempo médio do processo total, comparando os níveis da latência, pode ser vista a seguir:

Tamanho da imagem	Número de processos	Nível de latência entre os nós	Tempo médio do processo total em segundos	Intervalo de confiança
500	64	10 µs	0,03417	0,00152
500	64	100 µs	0,16243	0,00482
500	64	500 µs	0,4727	0,00778
1000	64	10 µs	0,07708	0,00299
1000	64	100 µs	0,14203	0,00411
1000	64	500 µs	0,44561	0,00597
2500	64	10 µs	0,35898	0,00617
2500	64	100 µs	0,47027	0,00706
2500	64	500 µs	0,74077	0,00684
5000	64	10 µs	1,374	0,01011
5000	64	100 µs	1,49679	0,012
5000	64	500 µs	1,76929	0,01267
10000	64	10 µs	5,657	0,04201
10000	64	100 µs	5,7081	0,04357
10000	64	500 µs	5,94904	0,45327

Figura 4: Tabela que mostra o tempo médio do processo total, comparando os níveis da latência.

Nessa tabela, é possível observar que o nível de latência impacta de forma profunda no tempo de execução do programa. Nesse sentido, nota-se que nos ambientes em que a comunicação entre os nós é mais demorada, o tempo para a execução do programa total é maior. Dessa forma, para que um sistema distribuído seja o mais efetivo possível, é necessário que a latência seja mínima, para que o tempo entre a comunicação dos nós seja baixo, resultando em um processo mais rápido.

6 Conclusão

Primeiramente, é notório que sistemas distribuídos são muito úteis para programação de alto desempenho, uma vez que com vários processos sendo realizados em paralelo, a computação pode se tornar muito mais veloz. Entretanto, evidencia-se que em um ambiente em que os processos possuem capacidade computacional heterogênea, é necessário distribuir as tarefas de forma proporcional ao nível de capacidade computacional de cada processo. Além disso, observa-se que, para um sistema distribuído ser o mais efetivo possível, a latência deve ser mínima, para que os processos possam se comunicar de forma ágil.

Ademais, nota-se que o MPI combinado com SimGrid, pode ser uma forma útil e simples de simulação de sistemas distribuídos, quando não é possível ter acesso a um ambiente distribuído adequado.