Programação Paralela

Paralelização do Problema do Caixeiro Viajante com Threads e Passagem de Messagem

Rogrigo Giraldi Silva

Victor Loquete Pupim

Abstract

Esse relatório tem como objetivo apresentar o processo de paralelização do algoritmo de solução do Problema do Caixeiro Viajante, desenvolvido utilizando técnicas de Algoritmos Genéticos , com o uso de threads e passagem de mensagem. Também contempla a comparação entre os resultados obtidos com os diferentes tipos de paralelização e o algoritmo sequencial.

1. Introdução

A Programação Concorrente é a área da computação que trata de técnicas e ferramentas para o desenvolvimento de algoritmos que em execução multiplicam seus fluxos de processamento e os distribuem entre os processadores disponíveis. Sua aplicação se torna viável em problemas que necessitam de um alto tempo de processamento, tal como a solução do Problema do Caixeiro Viajante utilizando Algoritmos Genéticos.

O Problema do Caixeiro Viajante, ou simplesmente PCV, consiste basicamente em determinar a menor rota, entre várias possíveis, para percorrer um conjunto de cidades ligadas por estradas e retornar para a cidade de origem.

Para a solução do PCV foi utilizada a técnica de algoritmos genéticos, que a partir de um conjunto de soluções iniciais, executa uma série de funções de forma iterativa e devolve como resultado um conjunto de soluções melhoradas.

Com a escolha do problema e da estratégia de solução, foram desenvolvidos os algoritmos genéticos de forma sequêncial, depois paralela com threads e por fim paralela combinando passagem de mensagem e threads.

Com os algoritmos funcionando corretamente, foram executados diversos testes, variando a quantidade de threads, processadores e parametros do algoritmo, e os resultados comparados e compilados em gráficos.

2. Fundamentação

Diante da solução do Problema do Caixeiro Viajante por Algoritmos Genéticos fica evidente que o esforço computacional exercido se dá devido ao grande número de repetições as quais os dados são

submetidos para que o melhoramento da rota aconteça, o que demanda um tempo considerável de processamento, apesar do conjunto de dados de entrada ser relativamente pequeno.
Sendo assim, o PCV com Algoritmos Genéticos apresenta um cenário interessante para se fazer a paralelização, de forma que o processamento iterativo dos dados seja realizada com mais eficiência.

3. Metodologia

A metodologia aplicada foi dividida em 3 etapas:

- Implementação sequencial dos Algoritmos Genéticos
- Paralelização dos Algoritmos Genéticos com o uso de threads
- Paralelização dos Algoritmos Genéticos resultantes da etapa anterior com o uso de passagem de mensagem

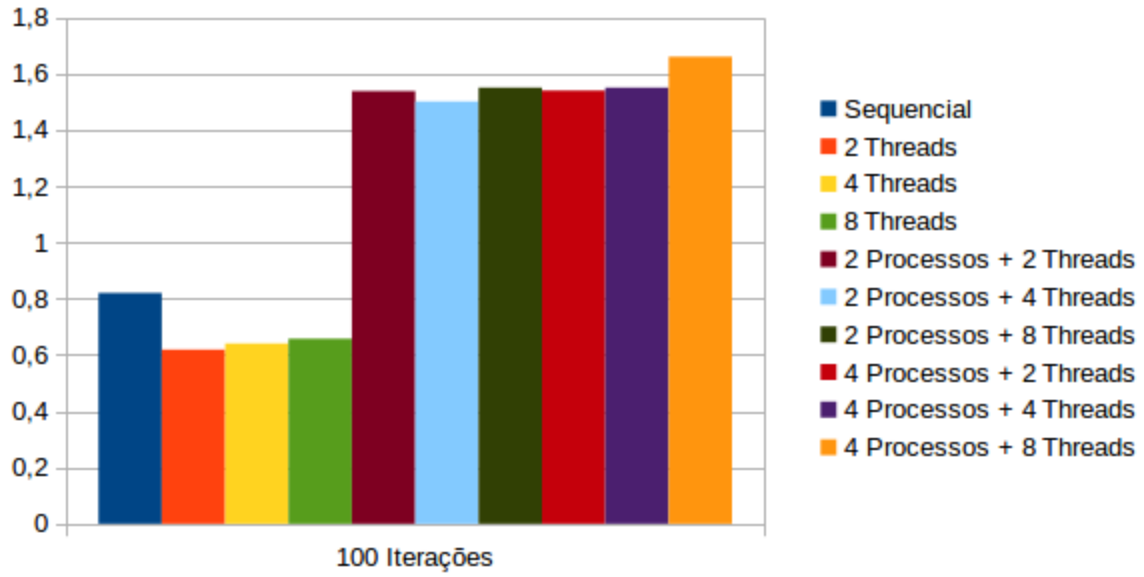
Os Algoritmos Genéticos desenvolvidos realizam os processos básicos para a computação dos dados do PCV, que são inicialmente gerados de forma aleatória e organizados em estruturas chamadas Gene, contendo a sequência de cidades a serem visitadas.
Esses cromossomos são agrupados em uma População, a qual será realizada a melhoria das rotas.
A execução do algoritmo começa com a seleção de dois genes da população atual, sendo escolhido o melhor entre eles e um aleatório para que seja realizado o Crossover.
A técnica de crossover escolhida se baseia na troca de metade de cada cromossomo dos dois escolhidos, os cromossomos pais. O resultado do crossover são dois cromossomos filhos, que são incluidos na População nos lugares de seus pais.
Essa rotina é repetida até que toda população seja renovada e o último passo, a Mutação, possa ser realizado sobre a população, de acordo com uma taxa de mutação pré-estabelecida.
Esse ciclo recebe o nome de Geração e é executado repetidamente até que se chegue ao fim da quantidade de gerações pré-definida e se obtenha uma nova rota com menor caminho.

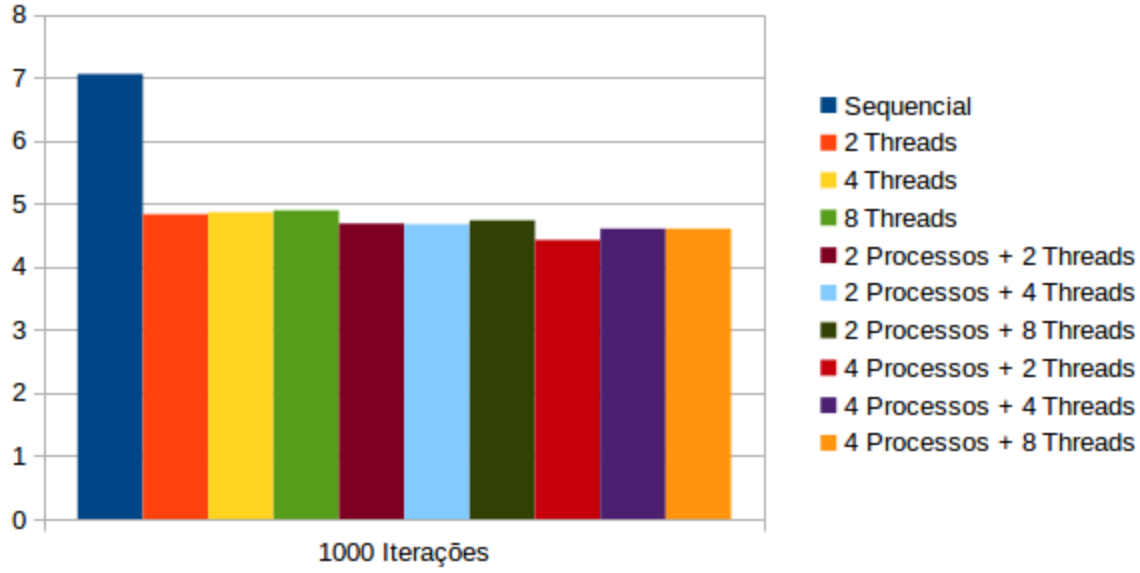
Na segunda etapa foi utilizada a biblioteca PThread do C para a paralelização do algoritmo com o uso de threads.
A abordagem inicial foi a paralelização da função de crossover, pois foi considerada a mais custosa em termos de processamento, e o uso de uma barreira com o objetivo de igualizar o estado de processamento de todas as threads antes da renovação completa da população. Porém não se notou uma diferença significativa no desempenho do algoritmo, pois a quantidade de dados era relativamente pequena, na forma de um vetor com apenas algumas de dezenas de cidades.
A partir disso foi discutida uma nova abordagem para realizar a paralelização do algorítmo, que assim como antes faz o uso de uma barreira antes da renovação da população, porém todos as partes do algoritmo são executadas por threads.
Com essa nova abordagem teve que ser feita uma refatoração do código, mudando o escopo de variáveis e alterando a passagem de parâmetros dos métodos. Também foi tratada a criação das threads de forma que todas elas fossem criadas no inicio da execução apenas.
Dessa forma o algoritmo paralelizado apresentou ganhos significativos em relação ao algoritmo sequencial.

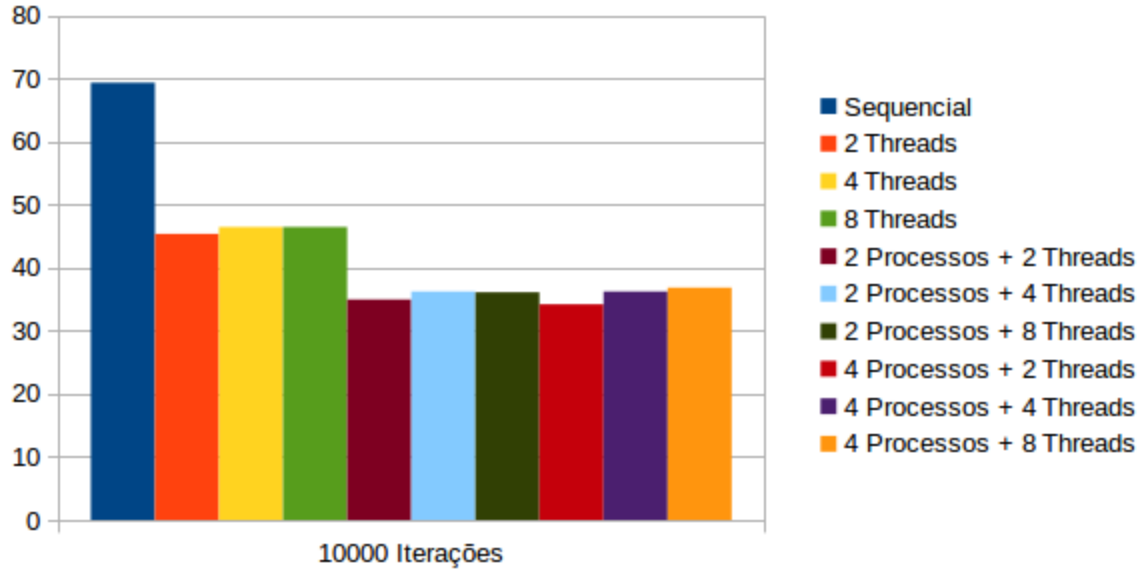
A paralelização por passagem de mensagens do algoritmo, na terceira etapa do desenvolvimento, foi utilizada a biblioteca MPI do C, e ainda mantendo o processamento paralelo por threads.
A estratégia adotada para a computação distribuída dos dados foi a divisão dos cromossomos da população inicial, realizada pelo computador mestre e distribuida entre os nós escravos. O computador mestre mantem uma parte dos dados para realizar a computação do algoritmo também. Junto com o envio dos cromossomos da população, o mestre também envia um array com as coordenadas de cada cidade, para que os escravos consigam fazer os cálculos de distância das rotas.
Assim que os escravos recebem sua porção da população é iniciado a execução do algoritmo até que a quantidade de gerações chegue ao fim. Após finalizado o processamento, cada escravo seleciona seu melhor cromossomo resultado e envia de volta para o mestre, que ao receber as mensagens com os resultados de todos os nós, seleciona o melhor entre eles.
Como as mensagens trocadas entre os computadores são do tipo array de struct foi necessária a criação de MPI datatypes que atendessem as structs criadas para que as funções de send e receive funcionassem corretamente.

4. Avaliação experimental

A seguir a comparação dos resultados obtidos na execução do algoritmo sequencial, paralelo com threads e paralelo com passagem de mensagens, compiladas em gráficos.
Os cálculos de SpeedUp e Eficiência foram feitos para o caso de 10000 iterações no Algoritmos Genéticos e estão apresentados nas tabelas depois dos gráficos.
Os testes foram executados em um processador Intel Core2Duo, com Linux Ubuntu 14.04.




8
7
6
5
4
3
2
1
0
1000 Iterações
Sequencial
2 Threads
4 Threads
8 Threads
2 Processos + 2 Threads
2 Processos + 4 Threads
2 Processos + 8 Threads
4 Processos + 2 Threads
4 Processos + 4 Threads
4 Processos + 8 Threads


80
70
60
50
40
30
20
10
0
10000 Iterações
Sequencial
2 Threads
4 Threads
8 Threads
2 Processos + 2 Threads
2 Processos + 4 Threads
2 Processos + 8 Threads
4 Processos + 2 Threads
4 Processos + 4 Threads
4 Processos + 8 Threads

Tempo Alg. Sequencial	69,28

Configuração	2 Threads	4 Threads	8 Threads
Nº Fluxos	2	4	8
Tempo Real	45,33	46,42	34,94
speed up	1,528347673	1,492460147	1,982827705
Eficiência	0,7641738363	0,3731150366	0,2478534631

Configuração	2 Processos + 2 Threads	2 Processos + 4 Threads	2 Processos + 8 Threads	4 Processos + 2 Threads	4 Processos + 4 Processos	4 Processos + 8 Threads
Nº Fluxos	4	8	16	8	16	32
Tempo Real	36,2	36,08	34,17	36,21	36,21	36,81
speed up	1,982827705	1,920177384	2,027509511	1,913283623	1,913283623	1,882097256
Eficiência	0,4957069262	0,2400221729	0,1267193445	0,2391604529	0,1195802265	0,0588155393

Analisando os resultados dos testes fica claro a melhora de desempenho quando comparadas as performances do algoritmo sequencial com o algoritmo paralelo com 2 threads, quando são manipuladas operações de certo tempo computacional. Nos casos em que o tempo de execução é muito pequeno não torna-se viável o processamento pararelo, sendo que se obtém melhores resultados sequencialmente.
Essa melhora não fica presente quando se aumenta a quantidade de threads e/ou processadores. Acreditamos que esse comportamento acontece devido ao fato dos testes terem sido executados em um computador com 2 núcleos apenas.

5. Intruções para execução

Para executar a aplicação os arquivos (código C e grafo) devem estar no mesmo diretório. Segue os parâmetros para compilar e executar cada um dos algoritmos.

Sequencial - compilação: gcc sequencial.c -o sequencial -std=gnu99 -lm
- execução: ./sequencial

Threads - compilação: gcc thread.c -o thread -std=gnu99 -lm -lpthread
- execução: ./thread

Para definir a quantidade de threads deve alterado o valor da constantes QTDE_THREADS

MPI - compilação mpicc mpiThread.c -o mpiThread -std=gnu99 -lm
- execução mpirun -np 4 mpThread

Para definir a quantidade de threads deve alterado o valor da constantes QTDE_THREADS e a quantidade de processos em QTD_PROC

6. Conclusão

Através dos experimentos e analises realizados pôde-se verificar a vantagem dos algoritmos paralelizados sobre o algoritmo sequencial quanto ao tempo de processamento.
Verificou-se também a importância de se analisar a viabilidade da paralelização do algoritmo, bem como o projeto do código, devido a complexidade encontrada na programação concorrente.