Relatório de Avaliação de Desempenho das Versões MPI e OpenMP de um algoritmo TSP paralelo

Ana Carolina Medrado

Efficiently solving the Traveling Salesman Problem (TSP) is a crucial challenge in various applications, from logistics to electronic circuit optimization. Due to the NP-hard nature of TSP, the search for effective approaches is a constant task in computer science. In this report, we evaluated the performance of parallel versions of a TSP algorithm implemented using the MPI and OpenMP libraries on a high-performance supercomputer, Lobo Carneiro.

1. INTRODUÇÃO

A resolução eficiente do Problema do Caixeiro Viajante (TSP) é fundamental em uma ampla gama de aplicações em diversos campos, desde logística e roteamento até a otimização de circuitos eletrônicos. Como o TSP é um problema NP-difícil, a busca por algoritmos que possam abordá-lo de forma eficaz é uma tarefa contínua na pesquisa em ciência da computação. Uma das estratégias mais promissoras para acelerar a solução do TSP é a paralelização, que permite explorar recursos de hardware avançados, como supercomputadores.

Neste relatório se tem uma avaliação de desempenho das versões paralelas de um algoritmo TSP implementadas utilizando as bibliotecas Message Passing Interface (MPI) e Open Multi-Processing (OpenMP). Essas versões foram executadas no supercomputador Lobo Carneiro, um dos sistemas mais avançados disponíveis para pesquisa científica e computacional no Brasil. O objetivo deste estudo é analisar como as técnicas de paralelização oferecidas pelo MPI e OpenMP podem melhorar a eficiência da resolução do TSP em um ambiente de computação de alto desempenho.

O supercomputador Lobo Carneiro é conhecido por sua capacidade de processamento massivamente paralelo e alta velocidade de comunicação entre nós, tornando-o um ambiente ideal para testar e avaliar algoritmos de otimização complexos, como o TSP. Neste contexto, será comparado o desempenho das versões MPI e OpenMP do nosso algoritmo em termos de tempo de execução, escalabilidade e eficiência. Além disso, examinaremos como o tamanho do problema e o número de nós de processamento afetam o desempenho dessas implementações paralelas.

Ao final deste relatório, será possível fornecer uma visão clara das vantagens e desvantagens de cada abordagem de paralelização (MPI e OpenMP) em relação à resolução do TSP no supercomputador Lobo Carneiro.

2. METODOLOGIA EXPERIMENTAL

Nesta seção, é detalhada a metodologia empregada para avaliar o desempenho das versões MPI e OpenMP do algoritmo de resolução do Problema do Caixeiro Viajante (TSP) no supercomputador Lobo Carneiro. Dados de entrada, métricas de desempenho e parâmetros experimentais essenciais para uma análise sólida e precisa dos resultados.

2.1. Dados de entrada

Os dados de entrada utilizados descrevem duas instâncias do Problema do Caixeiro Viajante (TSP), denominadas "ulysses22.tsp" e "ulysses16.tsp", respectivamente. Esses conjuntos de dados contêm informações essenciais para definir o contexto e as características desses problemas específicos de otimização.

Instância "ulysses22.tsp": Nome: ulysses22.tsp Tipo: TSP (Problema do Caixeiro Viajante) Dimensão: 22 (ou seja, há 22 cidades a serem visitadas) Tipo de Peso das Arestas: GEO (a distância entre cidades é definida como uma função geográfica) Tipo de Exibição de

Dados: COORD_DISPLAY (as coordenadas geográficas das cidades são fornecidas) Seção de Coordenadas dos Nós: Esta seção lista as coordenadas geográficas de cada uma das 22 cidades, identificadas pelo seu número de nó.

Instância "ulysses16.tsp": Nome: ulysses16.tsp Tipo: TSP (Problema do Caixeiro Viajante) Comentário: Odyssey of Ulysses (Groetschel/Padberg) Dimensão: 16 (ou seja, há 16 cidades a serem visitadas) Tipo de Peso das Arestas: GEO (a distância entre cidades é definida como uma função geográfica) Tipo de Exibição de Dados: COORD_DISPLAY (as coordenadas geográficas das cidades são fornecidas) Seção de Coordenadas dos Nós: Esta seção lista as coordenadas geográficas de cada uma das 16 cidades, identificadas pelo seu número de nó.

Ambas as instâncias estão relacionadas à "Odyssey of Ulysses" e apresentam uma variedade de coordenadas geográficas que representam as posições das cidades. Esses dados servirão como entradas para o algoritmo TSP, que buscará encontrar o menor caminho que visita todas as cidades uma vez e retorna ao ponto de partida.

A utilização dessas instâncias de teste permitirá avaliar a capacidade do algoritmo TSP em encontrar soluções eficientes para esses problemas específicos, bem como analisar o desempenho das implementações MPI e OpenMP no supercomputador Lobo Carneiro ao lidar com diferentes tamanhos de instâncias e tipos de dados de entrada.

2.2. Métricas

Na análise comparativa foi avaliado cinco métricas de desempenho e eficiência para entender como cada implementação aborda o Problema do Caixeiro Viajante (TSP) de maneira paralela. O primeiro código utiliza MPI (Message Passing Interface) para distribuir o trabalho entre processos, enquanto o segundo código emprega OpenMP para paralelizar a execução por meio de threads. As métricas incluem tempo de execução, uso de memória, número de threads/processos, qualidade da solução, eficiência da paralelização e oportunidades de otimização. Ao analisar essas métricas, podemos identificar as vantagens e desvantagens de cada abordagem e entender como melhorar o desempenho de ambos os códigos.

2.2.1. Branch and Bound Paralelo com MPI

Tempo de Execução: O tempo de execução desse código dependerá do tamanho do problema (número de cidades), do número de processadores MPI disponíveis e da eficiência da paralelização. É necessário executar o código em diferentes tamanhos de problema e com diferentes números de processadores MPI para obter uma análise precisa do tempo de execução.

Uso de Memória: O uso de memória será influenciado principalmente pelo tamanho do problema (número de cidades) e pelo número de processadores MPI. O código aloca memória para armazenar o grafo e as estruturas de dados para os subproblemas. O uso eficiente de memória é crítico, especialmente para problemas grandes.

Número de Threads: O código MPI não utiliza threads, mas em vez disso, ele funciona com processos separados. O número de processos MPI é igual ao número de threads. Portanto, o número de processadores MPI é uma métrica relevante aqui. A eficiência do código dependerá de como os processos MPI são alocados e gerenciados.

Qualidade da Solução: A qualidade da solução é medida pelo valor mínimo encontrado para o custo da rota (bestCost). O algoritmo Branch and Bound visa encontrar a solução ótima ou uma aproximação dela, portanto, a qualidade da solução é uma métrica crítica.

Paralelização: O código faz uso do MPI para a paralelização, distribuindo subproblemas para processos MPI. A eficiência da paralelização depende da carga de trabalho distribuída entre os processos MPI e de como os subproblemas são divididos e compartilhados.

Otimização: O código pode ser otimizado em várias áreas, incluindo a escolha de heurísticas para limites superiores e inferiores, a estratégia de distribuição de subproblemas e a gestão da memória. A otimização pode afetar significativamente o desempenho do código.

2.2.2. Branch and Bound Paralelo com OpenMP

Tempo de Execução: O tempo de execução deste código depende do tamanho do problema (número de cidades) e do número de threads OpenMP configuradas. É importante executar o código com diferentes tamanhos de problema e números de threads OpenMP para avaliar o tempo de execução.

Uso de Memória: O uso de memória é influenciado principalmente pelo tamanho do problema e pelas estruturas de dados alocadas para o grafo e os subproblemas. É importante medir o uso de memória para verificar se o código é eficiente em relação à memória, especialmente para problemas grandes.

Número de Threads: O código OpenMP usa um número especificado de threads para paralelizar o algoritmo. É importante avaliar o desempenho do código com diferentes números de threads para encontrar o equilíbrio entre paralelismo e sobrecarga de gerenciamento de threads.

Qualidade da Solução: A qualidade da solução é medida pelo valor mínimo encontrado para o custo da rota (final_res). O algoritmo Branch and Bound busca encontrar a solução ótima ou uma aproximação dela, portanto, a qualidade da solução é uma métrica crítica.

Paralelização: O código utiliza a diretiva OpenMP para paralelizar partes do algoritmo, distribuindo a carga

de trabalho entre as threads. É importante avaliar como a paralelização afeta o desempenho e a eficiência do código.

Otimização: Assim como o Código 1, este código também pode ser otimizado em várias áreas, como a escolha de heurísticas para limites superiores e inferiores, estratégias de divisão de subproblemas e gestão de memória. A otimização pode melhorar significativamente o desempenho do código.

A análise dessas métricas permitirá uma comparação detalhada do desempenho e da eficiência dos dois códigos para resolver o Problema do Caixeiro Viajante (TSP) e identificar áreas de melhoria em ambos os casos.

3. RESULTADOS

3.1. Resultado referentes à execução com OpenMP

gcc tsp_openmp.c -fopenmp -lgomp -o tsp_openmp time ./tsp_openmp Minimum cost : 291 Path Taken : 0 12 1 14 8 4 6 2 11 13 9 7 5 3 10 0 0.06user 0.00system 0:00.01elapsed 375%CPU (0avg-text+0avgdata 916maxresident)k 48inputs+0outputs (0major+280minor)pagefaults 0swaps

Comando de compilação: gcc tsp_openmp.c -fopenmp -lgomp -o tsp_openmp gcc: O compilador GCC (GNU Compiler Collection) é usado para compilar o código-fonte em C contido no arquivo "tsp_openmp.c". -fopenmp: Essa opção indica ao compilador para habilitar o suporte ao OpenMP, uma API para programação paralela. -lgomp: Isso especifica a biblioteca do OpenMP a ser usada. -o tsp_openmp: Define o nome do arquivo de saída do programa compilado como "tsp_openmp".

Comando de execução: time ./tsp_openmp time: O comando "time" é usado para medir o tempo de execução do programa. ./tsp_openmp: Isso executa o programa compilado "tsp_openmp".

Saída do Programa: "Minimum cost : 291": Essa linha indica o custo mínimo encontrado pelo algoritmo para resolver o TSP. No caso, o custo mínimo é 291 unidades. "Path Taken : 0 12 1 14 8 4 6 2 11 13 9 7 5 3 10 0": Isso mostra o caminho percorrido que leva ao custo mínimo. O caixeiro viajante começa na cidade 0, visita as cidades na ordem especificada e retorna à cidade de origem (cidade 0). "0.06user 0.00system 0:00.01elapsed": Essa parte fornece informações sobre o tempo de execução do programa: "0.06user": O tempo total da CPU gasto pelo programa foi de 0.06 segundos em modo de usuário. "0.00system": O tempo total da CPU gasto pelo sistema foi de 0.00 segundos. "0:00.01elapsed": O tempo decorrido total, incluindo tempo de espera, foi de 0 minutos e 0.01 segundos.

Outras informações: "375%CPU": Isso indica que a CPU foi utilizada em 375% durante a execução do programa. Isso pode ocorrer em sistemas com múltiplos núcleos de CPU, onde um valor superior a 100% indica uso eficiente de vários núcleos. "916maxresident": A memória máxima residente usada durante a execução foi de 916 kilobytes (KB). "48inputs+0outputs": Indica o número de operações de leitura (inputs) e escrita (outputs) realizadas durante a execução do programa. "0major+280minor": Indica o número de page faults (erros de página) maiores (major) e menores (minor) durante a execução. Erros de página ocorrem quando o programa acessa áreas de memória não carregadas. "pagefaults 0swaps": Essa parte indica que não houve troca de memória (swaps) durante a execução e nenhum erro de página maior foi relatado. Em resumo, o programa "tsp_openmp" resolveu o Problema do Caixeiro Viajante e encontrou um caminho de custo mínimo de 291 unidades. Ele usou uma quantidade significativa de CPU (375%) durante a execução, indicando uma boa utilização de recursos computacionais. O programa não teve problemas de falta de memória ou trocas de memória.

3.2. Resultado referentes à execução com MPI

gcc -c list.c mpicc tsp_mpi.c list.o -o tsp_mpi time mpirun -n 4 ./tsp_mpi Minimum cost is 133 Path is: 0 8 1 9 2536 7 10 4 0 0.23user 3.38system 0:01.70elapsed 2129CPU (Cavgtext+0avgdata 44856maxresident)k 48inputs+outputs (major+51491minor)pagefaults Oswaps

Custo Mínimo: O programa encontrou um custo mínimo de 133 unidades. Isso significa que o caminho encontrado pelo algoritmo tem um custo total de 133 unidades, o que é considerado o custo mínimo possível para visitar todas as cidades no problema TSP.

Caminho: O caminho encontrado pelo programa é listado a seguir: "0 8 1 9 2536 7 10 4 0". No entanto, esse resultado parece ser incorreto devido à presença do número "2536" no caminho. É importante observar que esse número não faz sentido no contexto do problema TSP e sugere um erro no resultado.

Tempo de Execução: O tempo total de execução do programa foi de 1 minuto e 1.70 segundos (ou seja, 61.70 segundos). Esse tempo é composto por dois componentes principais:

Tempo em modo de usuário: 0.23 segundos. **Tempo em modo de sistema:** 3.38 segundos. **Uso de Memória:** Durante a execução do programa, o uso máximo de memória residente atingiu 44.856 KB. Isso indica a quantidade máxima de memória física usada pelo programa durante sua execução.

E/S e Pagefaults: Durante a execução, o programa realizou 48 operações de entrada/saída (E/S). Além disso, houve 51491 erros de página menores, indicando

problemas de gerenciamento de memória durante a execução do programa.

Trocas de Memória (Swaps): Não houve trocas de memória (swaps) durante a execução. Isso significa que o sistema operacional não precisou usar a memória de troca (swap) do disco rígido.

Em resumo, o programa "tsp_mpi" encontrou um custo mínimo de 133 unidades, mas o resultado do caminho parece ser incorreto devido à presença do número "2536". Além disso, o programa teve um tempo de execução relativamente longo (mais de um minuto) e um alto número de erros de página menores. Esses problemas podem indicar possíveis bugs no programa ou problemas de alocação de memória que afetam a precisão dos resultados. É recomendável verificar e depurar o programa para corrigir esses problemas e garantir resultados precisos.

3.3. Comparação dos resultados

Resultado do TSP com MPI:

Custo Mínimo: 133 Caminho: 0 8 1 9 2536 7 10 4 0 Tempo de Execução: 0.23 segundos em modo de usuário, 3.38 segundos em modo de sistema, 1 minuto e 1.70 segundos decorridos. Uso de Memória: Uso máximo de memória residente foi de 44.856 KB. E/S e Pagefaults: 48 operações de entrada/saída, nenhum erro de página maior e 51491 erros de página menores. Trocas de Memória: Nenhuma troca de memória (swaps) ocorreu. Resultado do TSP com OpenMP:

Custo Mínimo: 291 Caminho: 0 12 1 14 8 4 6 2 11 13 9 7 5 3 10 0 Tempo de Execução: 0.06 segundos em modo de usuário, 0.01 segundos decorridos. Uso de Memória: Uso máximo de memória residente foi de 916 KB. E/S e Pagefaults: 48 operações de entrada/saída, nenhum erro de página maior e 280 erros de página menores. Trocas de Memória: Nenhuma troca de memória (swaps) ocorreu. Aqui estão as principais diferenças e observações:

Custo Mínimo: O programa com MPI encontrou um custo mínimo de 133 unidades, enquanto o programa com OpenMP encontrou um custo mínimo de 291 unidades. Portanto, o programa com MPI obteve um resultado melhor em termos de custo mínimo.

Caminho: O caminho encontrado pelo programa com MPI incluiu o valor inesperado "2536" no meio do caminho, o que parece ser um erro nos resultados. O caminho do programa com OpenMP parece estar correto, levando ao custo mínimo encontrado.

Tempo de Execução: O programa com MPI teve um tempo de execução mais longo, com 1 minuto e 1.70 segundos decorridos, em comparação com o programa com OpenMP, que teve um tempo de execução de apenas 0.01 segundos. Isso indica que o programa com OpenMP é muito mais rápido.

Uso de Memória: O programa com MPI usou mais memória, com um uso máximo de memória residente de 44.856 KB, enquanto o programa com OpenMP usou significativamente menos memória, com um uso máximo de 916 KB.

E/S e Pagefaults: Ambos os programas tiveram o mesmo número de operações de entrada/saída (48), mas o programa com MPI teve um número muito maior de erros de página menores (51491) em comparação com o programa com OpenMP (280). Isso pode indicar um comportamento anormal no programa com MPI.

Trocas de Memória: Ambos os programas não tiveram trocas de memória (swaps).

Em resumo, o programa com MPI obteve um resultado de custo mínimo melhor, mas parece ter problemas em seu resultado de caminho e teve um tempo de execução muito mais longo. Por outro lado, o programa com OpenMP teve um tempo de execução muito mais curto, uso de memória mais eficiente e um caminho correto.

4. CONCLUSÃO

A resolução eficiente do Problema do Caixeiro Viajante (TSP) é um desafio crucial em várias aplicações, desde logística até otimização de circuitos eletrônicos. Devido à natureza NP-difícil do TSP, a busca por abordagens eficazes é uma tarefa constante na ciência da computação. Neste relatório, foi avaliado o desempenho das versões paralelas de um algoritmo TSP implementadas com as bibliotecas MPI e OpenMP em um supercomputador de alto desempenho, o Lobo Carneiro.