UFPR – Departamento de Informática

CI1316 – Programação Paralela - Trabalho Prático MPI

Prof. Marco Antonio Zanata Alves

Com base na implementação sequencial em linguagem C do algoritmo **TSP**, listados no final deste arquivo. Leia atentamente e elabore um relatório de no máximo 8 páginas com os seguintes itens:

1. Recorte o kernel (parte principal) do algoritmo e explique em suas palavras o funcionamento sequencial do trecho.
2. Explique qual a estratégia final (vitoriosa) de paralelização você utilizou.
3. Descreve a metodologia que você adotou para os experimentos a seguir. Não esqueça de descrever também a versão do SO, kernel, compilador, flags de compilação, modelo de processador, número de execuções, etc.
4. Com base na execução sequencial, meça e apresente a porcentagem de tempo que o algoritmo demora em trechos que você não paralelizou (região puramente sequencial).
5. Aplicando a Lei de Amdahl, crie uma tabela com o speedup máximo teórico para 2, 4, 8 e infinitos processadores. Não esqueça de explicar a metodologia para obter o tempo paralelizável e puramente sequencial.
6. Apresente tabelas de speedup e eficiência. Para isso varie o número de threads entre 1, 2, 4 e 8. Varie também o tamanho das entradas. Ajuste a quantidade de threads de acordo com o processador que você estiver utilizando. Pense no número de threads que faça sentido. Veja um exemplo de tabela:

		1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Eficiência	N=10.000	1	0,81	0,53	0,28	0,16
	N=20.000	1	0,94	0,80	0,59	0,42
	N=40.000	1	0,96	0,89	0,74	0,58

7. Analise os resultados e discuta cada uma das duas tabelas (speedup e eficiência). Você pode comparar os resultados com speedup linear ou a estimativa da Lei de Amdahl para enriquecer a discussão.
8. Seu algoritmo apresentou escalabilidade forte, fraca ou não foi escalável? Apresente argumentos coerentes e sólidos para suportar sua afirmação.
9. Pense sobre cada um dos resultados. Eles estão coerentes? Estão como esperados? A análise dos resultados exige atenção aos detalhes e conhecimento.

Cuidados gerais para efetuar os experimentos

- Para assegurar a corretude da implementação paralela, deve-se verificar se os resultados paralelos batem com os sequenciais executando diferentes entradas. **Lembre-se que o resultado bater não significa obrigatoriamente que o código está correto**.
- Execute pelo menos 20x cada versão para obter uma média minimamente significativa. Ou seja, todo teste, onde mudamos o número de processos ou tamanho de entrada, devemos executar 20x. Mostrar no relatório a **média com desvio padrão**.
    - As métricas deverão ser calculadas encima da média das execuções.
- Sugiro escolher um modelo de máquina e sempre utilizar o mesmo modelo até o final do trabalho.
    - Cuidar para não executar em servidores virtualizados ou que contenham outros usuários (processos ativos) utilizando a mesma máquina. Diversos servidores do DINF são máquinas virtualizadas e os testes de speedup não serão satisfatórios/realísticos.
    - Cuide para que não haja outros processos ou usuários usando a máquina no mesmo momento que você esteja executando seus testes.
    - Sempre execute com as flags máximas de otimização do compilador, exemplo -O3 para o gcc, afinal queremos o máximo desempenho.
    - Podemos pensar se queremos modificar as configurações de DVFS, também conhecido como turbo-boost, ou seja, fixar a frequência de operação de nossa máquina.
    - Por fim, ainda podemos ter maior controle do experimento, reduzindo a variabilidade ao fixar as threads nos núcleos de processamento.
- **Teste de escalabilidade forte**: Manter um tamanho de entrada N qualquer, e aumentar gradativamente o número de processos. Sugere-se que escolha-se um N tal que o tempo de execução seja maior ou igual a 10 segundos.
- **Teste de escalabilidade fraca:** Aumentar o tamanho da entrada e o número de threads. Atenção, escalar N com o número de threads/processos (não de máquinas no caso do MPI). Lembre-se que o impacto de N no tempo final dependerá da complexidade do algoritmo que estamos trabalhando nesse semestre.
- Seu **algoritmo deve ser genérico** o suficiente para executar com 1, 2, 3, N threads/processos.
- Ambos os códigos (sequencial e paralelo) **devem gerar as mesmas saídas**.
- Evite figuras ou gráficos de resultados muito complexos, opte por formas de apresentação de fácil entendimento.

Regras Gerais de Entrega e Apresentação

A paralelização dos códigos deve ser feita em C ou C++ utilizando as rotinas MPI. A entrega será feita pelo Moodle dividida em duas partes

- **Relatório em PDF (máximo 8 páginas, fonte verdana ou similar tamanho 12pts.)**
- **Código fonte paralelo (MPI)**

- Casos não tratados no enunciado deverão ser discutidos com o professor.
- Os trabalhos devem ser feitos individualmente.
- **A cópia do trabalho (plágio), acarretará em nota igual a Zero para todos os envolvidos.**
- **Os trabalhos serão defendidos presencialmente pelo aluno. A nota irá considerar domínio do tema, robustez da solução e rigorosidade da metodologia.**

The Traveling-Salesman Problem (TSP)

O problema do caixeiro viajante

O Problema do Caixeiro Viajante (TSP) consiste em resolver o problema de roteirização de um hipotético caixeiro viajante. Tal rota deve passar por n cidades, apenas uma vez por cidade, retornar à cidade de origem e ter a menor extensão possível. É um problema NP-difícil muito bem estudado. Mais formalmente, o problema poderia ser representado como um grafo completo não direcionado G = (V, E), |V| = n onde cada aresta (i, j). E tem um custo associado c(i, j) ≥ 0 representando a distância da cidade i a j (Figura B1a). O objetivo é encontrar um ciclo hamiltoniano com custo mínimo, ou um passeio com duração mínima, que visite cada cidade apenas uma vez e termine na cidade de partida.

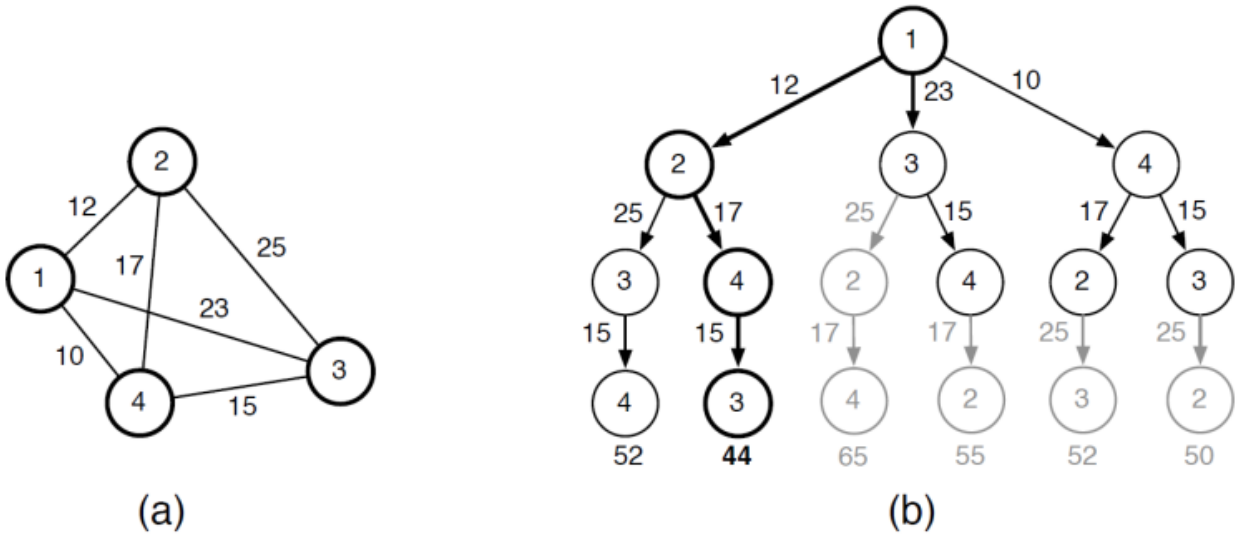

Figure B1. Example of TSP with 4 cities.

Existem várias abordagens diferentes para resolver este problema. Estas soluções normalmente empregam força bruta, heurísticas simples ou complexas, algoritmos de aproximação ou uma mistura deles. **A versão sequencial fornecida do algoritmo emprega deliberadamente um algoritmo exato de força bruta muito simples, baseado em uma heurística gananciosa simples.** Este algoritmo faz uma busca em profundidade procurando o caminho mais curto e possui complexidade O(n!). Ele não explora caminhos que já se sabe serem mais longos que o melhor caminho encontrado até o momento, portanto poda o espaço de busca descartando galhos infrutíferos. A Figura B1b mostra esse comportamento. As arestas sombreadas são aquelas que o algoritmo não segue, pois uma possível solução que as inclua seria mais custosa do que aquela já identificada. Esta técnica simples de poda melhora muito o desempenho do algoritmo. No entanto, também introduz irregularidades no espaço de busca. A profundidade de busca necessária para descartar um dos ramos depende da ordem em que os ramos foram pesquisados.

Escreva uma versão paralela do programa que produza a duração de um passeio mínimo.

Seu programa deve utilizar a mesma heurística gananciosa do exemplo fornecido, pois aqui estamos interessados nas estratégias de paralelização e não em uma heurística melhor para o TSP.

Entrada:

A entrada pode conter diversas instâncias do problema, que são fornecidas pela primeira linha da entrada. A seguir, cada problema é dado sequencialmente como segue. A primeira linha contém o número de cidades (n). As n linhas a seguir são pares inteiros separados por espaço que descrevem as coordenadas cartesianas de cada cidade. **A entrada deve ser lida a partir da entrada padrão.**

Saída:

A saída deve listar os comprimentos dos caminhos mais curtos encontrados para cada problema, uma distância por linha. Para evitar problemas de arredondamento todas as distâncias consideradas pelo programa sequencial são truncadas. Seu programa deve fazer o mesmo. Veja o código fornecido para obter detalhes.

A saída deve ser gravada na saída padrão.