UFPR – Departamento de Informática

CI1316 – Programação Paralela - Trabalho Prático OpenMP

Prof. Marco Antonio Zanata Alves

Com base na implementação sequencial em linguagem C do algoritmo **LCS**, 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, tentando manter uma proporção. 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. 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 proporcionalmente com o número de processos. Exemplo: 1xN, 2xN, 4xN, 8xN, 16xN. Atenção, escalar N com o número de threads/processos (não de máquinas no caso do MPI).
- 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 e primitivas OpenMP. 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 (OpenMP)**

- 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 deverão ser apresentados ao vivo no Zoom pelo aluno com vídeo e áudio. A nota irá considerar domínio do tema, robustez da solução e rigorosidade da metodologia.**

Longest Common Subsequence (LCS) Problem

Problema da Maior Subsequência Comum (MSC)

Encontrar a maior subsequência comum (LCS) é um problema clássico na área de algoritmos de computador e possui domínios de aplicação diversificados.

Uma subsequência comum de duas sequências dadas pode ser definida informalmente como uma série de símbolos em que todos os elementos estão contidos em ambas as sequências e aparecem na mesma ordem. Por exemplo, “ose” e “os” são subsequências comuns de “house” e “browse”. Uma Subsequência Comum Mais Longa (LCS) de duas sequências é uma subsequência que tem um comprimento máximo. Encontrar a subsequência comum mais longa de duas sequências distintas representa um desafio muito importante para várias áreas onde a ciência da computação é aplicada.

O LCS tem várias aplicações em vários campos, incluindo alinhamento de sequência de DNA em bioinformática, reconhecimento de voz e imagem, comparação de arquivos, otimização de consulta de banco de dados, etc. que eles têm relações biológicas entre si (por exemplo, funções biológicas semelhantes).

Na descoberta de padrões entre sequências, o LCS desempenha um papel importante para encontrar a região comum mais longa entre duas sequências. Embora uma quantidade louvável de esforços tenha sido feita na tarefa de descoberta de padrões, com o aumento do comprimento das sequências, os algoritmos aparentemente enfrentam gargalos de desempenho. Além disso, com o advento das tecnologias de sequenciamento de última geração, os dados sequenciais estão aumentando rapidamente, o que demanda algoritmos com tempo de execução mínimo possível.

Dada uma sequência de símbolos S={a0, ɑ1, ..., ɑn}, uma subsequência S' de S é obtida removendo zero ou mais símbolos de S. Por exemplo, dado K={ɑ, b, c, d, e}, K'={b, d, e} é uma subsequência de K. Uma subsequência comum mais longa de duas sequências X e Y é uma subsequência de X e Y com comprimento máximo.

O comprimento de uma subsequência comum mais longa, a distância de Levenshtein, é uma métrica de string para medir a diferença entre duas sequências.

O comprimento c[m, n] de um LCS de duas sequências A={α0, α1, ..., αn} e B={b0, b1, ..., bn} pode ser definido recursivamente da seguinte maneira:

A partir desta definição, um algoritmo de programação dinâmica pode ser derivado diretamente.
Escreva um programa paralelo para calcular o tamanho da subsequência comum mais longa entre duas sequências. Sua solução deve usar programação dinâmica para construir uma matriz de pontuação (como na solução sequencial fornecida).

Entrada:
O programa deve ler duas sequências de arquivos diferentes, contendo letras e números. O primeiro arquivo tem a sequência A (fileA.in). O segundo arquivo tem a sequência B (fileB.in).
As sequências devem ser lidas dos arquivos de entrada corretos.

Saída:
O programa imprimirá o tamanho da Subsequência Comum Mais Longa. Você não precisa imprimir a matriz de pontuação no final, mas a matriz deve ser construída na memória e será usada para verificar soluções (use a função de depuração fornecida para imprimir a matriz durante o desenvolvimento).
A saída deve ser gravada na saída padrão.

Exemplo:

ArquivoA-Entrada

heagawghee

ArquivoB-Entrada

pawheae

Saída

5