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 **SSP**, 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 sequencial e explique em suas palavras o funcionamento do trecho.
2. Explique qual a estratégia final (vitoriosa) de paralelização você utilizou.
3. Descreva a metodologia que você adotou para os experimentos a seguir. Responda também se você desligou o overclock dinâmico (turbo mode), se definiu o modo desempenho, se isolou os núcleos de desempenho, etc. 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 não passiveis de paralelização (região puramente sequencial).
5. Aplicando a Lei de Amdahl, crie uma tabela com o speedup máximo teórico variando o número de processadores (ex. 2, 4, 8, ... ,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/processos (ex. 1, 2, 4 e 8). Varie também o tamanho das entradas. Ajuste a quantidade de threads/processos de acordo com o processador que você estiver utilizando. Pense no número de threads/processos que faça sentido. Veja um exemplo de tabela:

		1 Thread/ Processo	2 Threads/ Processos	4 Threads/ Processos	8 Threads/ Processos	16 Threads/ Processos
Eficiência	N=10k	1.00	0.81	0.53	0.28	0.16
	N=20K	1.00	0.94	0.80	0.59	0.42
	N=40k	1.00	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? Eu sei explicar o desempenho do Simultaneous Multi-threading? O que o desvio padrão está nos mostrando? A análise dos resultados exige atenção aos detalhes e conhecimento.

Cuidados gerais para efetuar os experimentos

- Para assegurar a corretude da implementação paralela, verifique se os resultados paralelos batem com os sequenciais para 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 uma máquina e utilizá-la até o final do trabalho.
  - Cuidar para não executar em servidores virtualizados ou que contenham outros usuários (processos ativos) na 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, e de turbo-boost/turbo-mode, ou seja, fixar a frequência de operação do CPU.
  - Máquinas com núcleos heterogêneos (Performance e Efficiency cores) precisam de maior cautela.
  - Podemos ter maior controle do experimento, reduzindo a variabilidade ao fixar as threads/processos 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 >= 10 segundos.
- **Teste de escalabilidade fraca:** Aumentar o tamanho da entrada e o número de threads/processos. 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.
- Seu **algoritmo deve ser genérico** o suficiente para executar com por exemplo 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 serão defendidos presencialmente pelo aluno. A nota irá considerar domínio do tema, robustez da solução e rigorosidade da metodologia.**

Shortest Superstring Problem (SSP)

O Problema da Super Sequência Mais Curta

O problema da super sequência mais curta pertence à classe NP-Difícil e é definido da seguinte forma:

Dado um conjunto de sequências S onde nenhum elemento é subsequência de outro elemento, encontre a sequência mais curta s que contém cada sequência em S como subsequência. Formalmente, seja S = {s1, s2, ..., sn} um conjunto de sequências tal que

$$\forall s2 , s4 \in S \; s2 \nsubseteq s4 .$$

Assim, o problema da menor substring é encontrar uma string

$$s \in S' = s' \; \forall s2 \in S \; s2 \subseteq s' \text{ tal que } \forall s' \in S' \; s \preceq s' .$$

Por exemplo, considere o seguinte conjunto de strings:

S = {CATGC, CTAAGT, GCTA, TTCA, ATGCATC}.

A string mais curta que contém todas as strings acima como uma substring é

s = GCTAAGTTCATGCATC.

A solução proposta emprega uma estratégia gulosa sobre uma operação chamada "overlap", descrita a seguir.

A operação de sobreposição sobre as strings a e b é sua concatenação ab, onde as partes correspondentes no sufixo de a e no prefixo de b são mescladas. Por exemplo, se a = {ABC} e b = {BCDE}, a sobreposição à esquerda de a e b é {ABCDE}. O valor de sobreposição da operação é o tamanho dos sufixos/prefixos correspondentes de ambas as strings. No exemplo, o valor de sobreposição é 2 (BC).

A ordem importa; a operação de sobreposição e seu respectivo valor de sobreposição não são comutativos. Com a operação de sobreposição definida, podemos agora mostrar o algoritmo guloso que resolve o problema em questão. Sua ideia é simples: a cada passo, substituímos as duas strings cujo valor de sobreposição resultante seria o maior pelo resultado de sua sobreposição. No final, temos a superstring mais curta:

1. Seja T uma cópia de S.

2. Enquanto |T| > 1, faça:

    a. Sejam a e b as duas strings que produzem o maior valor de sobreposição;

    b. Retire a e b de T e insira a string obtida pela sobreposição de a e b.

Agora, o único elemento de T é a menor superstring contendo como substrings todas as strings de S.

É importante notar que a sobreposição das strings x e y provavelmente não é a mesma que a sobreposição das strings y e x devido à sua já mencionada falta de comutatividade. Portanto, ambas as configurações devem ser avaliadas para cada par de strings considerado e sua ordem preservada.

Você deve submeter uma implementação paralela de um algoritmo que resolva o problema da menor superstring.

Entrada

A primeira linha contém o número n de strings a serem lidas e processadas. Cada uma das n linhas seguintes da entrada contém uma string com no máximo 256 caracteres ASCII. Todos os caracteres são legíveis e não há espaços em branco. A entrada deve ser lida da entrada padrão.

Saída

É uma única linha contendo a menor superstring que contém todas as strings da entrada. A saída deve ser escrita na saída padrão.

Exemplo

Entrada	Saída
5 CATG CTAAGT GCTA TTCA ATGCATC	GCTAAGTTCATGCATC