Universidade Federal do Rural do Semi-Árido
Centro Multidisciplinar de Pau dos Ferros
Departamento de Engenharias e Tecnologia
PEX1272 - Programação Concorrente e Distribuída
Docente: Ítalo Assis

Lista de Exercícios - Software paralelo

1. Sendo n o tamanho do problema e p o número de *threads* (ou processos):

   (a) Suponha que o tempo de execução de um programa sequencial seja dado por $T_{sequencial} = n^2$, onde as unidades do tempo de execução estão em microssegundos. Suponha que uma paralelização deste programa tenha tempo de execução $T_{paralelo} = n^2/p + log_2(p)$. Escreva um programa que encontre as acelerações e eficiências deste programa para vários valores de n e p. Execute seu programa com $n = 10, 20, 40, \cdots, 320$ e $p = 1, 2, 4, \cdots, 128$.
   - O que acontece com os *speedups* e eficiências à medida que p aumenta e n é mantido fixo?
   - O que acontece quando p é fixo e n é aumentado?

   (b) Suponha que $T_{paralelo} = T_{sequencial}/p + T_{overhead}$. Suponha também que mantemos p fixo e aumentamos o tamanho do problema.
   - Mostre que, se $T_{overhead}$ crescer mais lentamente que $T_{sequencial}$, a eficiência paralela aumentará à medida que aumentarmos o tamanho do problema.
   - Mostre que se, por outro lado, $T_{overhead}$ crescer mais rápido que $T_{sequencial}$, a eficiência paralela diminuirá à medida que aumentamos o tamanho do problema.

2. Às vezes, diz-se que um programa paralelo que obtém um *speedup* maior que p (o número de processos ou *threads*) tem *speedup* superlinear. No entanto, muitos autores não consideram os programas que superam as "limitações de recursos"como tendo aceleração superlinear. Por exemplo, um programa que deve usar armazenamento secundário para seus dados quando é executado em um sistema de processador único pode ser capaz de acomodar todos os seus dados na memória principal quando executado em um grande sistema de memória distribuída. Dê outro exemplo de como um programa pode superar uma limitação de recursos e obter *speedups* maiores que p.

3. Sendo n o tamanho do problema e p o número de *threads* (ou processos), suponha $T_{serial} = n$ e $T_{parallel} = n/p + log_2(p)$, onde os tempos estão em microssegundos. Se aumentarmos p por um fator de k, encontre uma fórmula para quanto precisaremos aumentar n para manter a eficiência constante.

   (a) Quanto devemos aumentar n se dobrarmos o número de processos de 8 para 16?

   (b) O programa paralelo é escalável?

4. Um programa que obtém *speedup* linear é fortemente escalável? Explique sua resposta.

5. Bob tem um programa que deseja cronometrar com dois conjuntos de dados, *input_data1* e *input_data2*. Para ter uma ideia do que esperar antes de adicionar funções de temporização ao código de seu interesse, ele executa o programa com os dois conjuntos de dados e o comando shell Unix *time*:

```
$ time ./bobs_prog < input_data1
real 0m0.001 s
user 0m0.001 s
sys  0m0.000 s

$ time ./bobs_prog < input_data2
real 1m1.234 s
user 1m0.001 s
sys  0m0.111 s
```

A função de temporização que Bob está usando tem resolução de milissegundos.

(a) Bob deveria usá-la para cronometrar seu programa com o primeiro conjunto de dados? Por que?

(b) E quanto ao segundo conjunto de dados? Por que?

6. Escreva um pseudocódigo para a soma global estruturada em árvore para somar vetores.

   (a) Primeiro considere como isso pode ser feito em um ambiente de memória compartilhada.

      • Quais variáveis são compartilhadas e quais são privadas?

   (b) Depois considere como isso pode ser feito em um ambiente de memória distribuída.

Referência

- PACHECO, Peter S. An introduction to parallel programming. Amsterdam Boston: Morgan Kaufmann, c2011. xix, 370 p. ISBN: 9780123742605.