UFMG/ICEx/DCC — Algoritmos e Estrutura de Dados 3

Trabalho Prático 2

Bacharelado — 1º Semestre de 2011

Esse trabalho prático tem por objetivo fixar o conteúdo de computação paralela ministrado na aula de Algoritmos Estruturados 3.

1 Revisão Paralelização

Geralmente os problemas computacionais que executam em paralelo fazem uso das duas estratégias.

Paralelismo de Dados:
No paralelismo de dados, cada CPU faz operações em um sub-conjunto dos dados a serem processados. A tarefa executada por cada CPU é a mesma, mudando apenas os dados de entrada. Um exemplo é calcular o determinante de várias matrizes, cada CPU pode operar com um conjunto de matrizes diferentes, fazendo sempre a mesma operação.

Paralelismo de Tarefas:
No paralelismo de tarefas, cada CPU executa uma tarefa diferente. Por exemplo, em um caso em que buscamos imagens duplicadas; uma CPU pode estar ocupada lendo as imagens e convertendo para um formato de dados específico, enquanto uma segunda CPU pega os dados convertidos pela primeira e faz a busca por duplicatas.

Como exemplo de computação paralela, tem-se o problema: Dado um cientista da computação que deseja somar o valor do determinante de 10000 matrizes. Apenas 4 matrizes cabem na memória por vez. Por um acaso do destino, o cientista tem um máquina com exatamente 4 núcleos (cores). O cientista poderia resolver o problema utilizando computação paralela da seguinte forma (Algoritmo 1):

As linhas 1-6 do algoritmo não contém as implementações das funções, são apenas possíveis assinaturas dos métodos que foram utilizados para o exemplo. A função DETERMINANT não retorna um valor e sim armazena o resultado na posição de memória indicada. As linhas 7-24 compõem o programa em si que executa da seguinte forma: (1) inicializa algumas variáreis (linhas 8-11) onde são alocadas 4 posições de memória para receber resultados das funções (linha 10); (2) percorre os arquivos que devem ser processados carregando 4 matrizes (linha 13) e dispara 4 threads para processar estas (linhas 14-17), note que a linha 14 cria um apontador para a função e a linha 15 cria a thread que vai chamar esta função com os parâmetros de entrada; (3) espera as 4 threads finalizar e atualiza a soma dos determinantes (linhas 19-22).

Algorithm 1 Sum matrix determinants

```
1: function LOAD_MATRIX(f)
2:     m ← MATRIXIFY(f)                          ▷ Loads matrix from file
3:     return m

4: function DETERMINANT(m, *d)
5:     d ← DET(m)                                ▷ Store on *d the return value
6:     return

7: function MAIN(l)                              ▷ l is a list of open files
8:     i ← 0
9:     t_threads ← [nil, nil, nil, nil]
10:    d_results ← [*d_1, *d_2, *d_3, *d_4]
11:    sum ← 0
12:    for f ∈ l do
13:        m ← LOAD_MATRIX(f)
14:        d_f ← *DETERMINANT                    ▷ Pointer to function
15:        t ← THREAD_CREATE(d_f, m, d_results[i])
16:        t_threads[i] ← t
17:        START_THREAD(t)                       ▷ Threads will compute determinant
18:        i ← i + 1
19:        if i % 4 = 0 then
20:            i ← 0
21:            THREAD_JOIN(t_threads)            ▷ Wait for threads to finish
22:            sum ← sum+ SUM_VECTOR(d_results)
23:    PRINT(sum)
24:    return
```

O pseudocódigo acima, é um exemplo de paralelismo que não necessariamente trabalha de acordo com a biblioteca pthreads. Contudo, o mais importante é a ideia de cada thread ter um endereço de memória próprio para armazenar o resultado da sua computação (visto nas linhas 4, 10 e 16).

2 Paralelização proposta para o TP2

Utilizando o que foi aprendido sobre paralelização de tarefas e CPU na disciplina de AEDS3, implemente o problema descrito no TP0 com paralelismo. Devem ser implementados dois níveis de paralelismo: (1) no primeiro nível, cada thread deve calcular um conjunto de ranqueamentos (2) no segundo nível, várias threads são usadas para computar o placar (valor) do ranqueamento. No final, também é necessário calcular a melhor solução das instâncias do problema usando as threads.

Nesse TP o executável deve ser chamado de tp2 e receber como parâmetro o arquivo de entrada de dados e o número de threads que serão utilizadas para resolver o problema respectivamente, ou seja:

./tp2 <arquivo> <número de threads nível 1> <número de threads nível 2>

Observações:

- Data de entrega : 10/05/2011 .
- A entrada e saída do trabalho serão as mesmas especificadas no TP0.
- Em termos de submissão, a documentação do trabalho deve ser submetida ao minha.ufmg e entregue impressa na secretaria do DCC (solicitar à secretária que coloque no envelope da disciplina). O código fonte do trabalho e o pdf da documentação devem ser compactados (formato zip) e submetidos no moodle.
- A documentação deve explicar os detalhes e decisões de implementação assim como os experimentos utilizados para teste. Os experimentos devem ser analisados variando o número threads nos dois níveis. Além disso, deve haver uma comparação entre o tempo de execução com paralelismo e a solução do TP0. Na análise experimental da documentação deve ser discutido o tempo de execução em relação ao número de threads nos dois níveis (utilizar gráficos).

- As estruturas de dados devem ser alocadas dinamicamente e o código ser modularizado (ou seja, dividido em múltiplos arquivos fonte e fazendo uso de arquivos cabeçalho .h) e o utilitário Make deve ser utilizado para compilar e executar o programa.

Distribuição dos pontos:

- execução
  execução correta: 40%
  saída legível: 10%
- estilo de programação
  código bem estruturado: 10%
  código legível: 10%
- documentação
  discussão dos resultados: 10%
  experimentos realizados: 10%
  análise de complexidade: 10%