Relatório de Atividades

Alunos: André Luiz de Andrade Coelho
João Luis da Silva Guio Soares - 114.094.151
Período: 2015.2
Data: 05/01/2016

O problema apresentado é implementar um código sequencial e concorrente para “Quadratura Adaptativa”, um método de aproximação de integrais, mais especificamente, utilizando retângulos que aproximam a área da figura.

O código sequencial resume-se em:

1. Receber dados de entrada:
    a. Intervalo da função;
    b. Erro máximo;*
    c. Função;
2. Chamar uma função recursiva que (intervalo [a,b], ponto médio m = (a + b) / 2):
    a. Calcular a área retangular do intervalo com a altura no ponto médio;
    b. Calcular a área de dois retangulos menores, que possuem a altura entre o ponto do meio e os pontos do intervalo.
    c. Se o módulo da diferença das áreas é **menor que** o erro, retorna o valor.
    d. Senão, faz a soma dos retangulos com intervalo em [a,m] e [m,b]
    e. A soma final é o valor aproximado da integral.
3. Imprimir o valor na tela, mostrando assim o resultado da integral.

Um problema que já deve ser considerado, é que soma com pontos flutuantes gera imprecisão. Dessa forma, o cálculo do erro não fica em cima do valor final, e sim em cima da diferença das áreas dos retângulos para cada retângulo da recursão.

No problema sequencial, não havia estruturas muito complexas necessárias. Apenas variáveis comuns eram capazes de resolver o problema (até por se tratar apenas de operações aritméticas - soma, subtração, multiplicação e divisão).

O código concorrente executa as seguintes etapas:

1. Thread principal recebe os dados de entrada:
    a. Intervalo da função ([a,b]);
    b. Erro máximo (err);
    c. Quantidade de Threads (nThreads);
    d. Função (func).

2. É chamada então apenas uma thread. Esta thread executa o cálculo do retângulo de intervalo [a,b];
3. A função recursiva funciona então da seguinte maneira:
    a. Acha o ponto médio do intervalo;
    b. Faz os mesmos cálculos de área dos três retângulos necessários
        i. func((a + b) / 2) * (b - a)
        ii. func((a + m) / 2) * (m - a)
        iii. func((m - b) / 2) * (b - m)
    c. Checa a condição de precisão do programa (módulo da diferença das áreas < err);
    d. Se passar:
        i. Checa numa variável global se o número máximo de threads já foi atingido;
        ii. Se não foi atingido, cria-se uma nova thread responsável por calcular o retangulo direito, e a thread atual continua calculando apenas o retangulo esquerdo;
        iii. Se foi atingido, checa-se então uma variável global se "existe uma thread ociosa"
            1. Se existe thread ociosa, a thread atual entra num local de exclusão mútua e anota em variáveis globais o retângulo direito a ser calculado, manda um sinal às threads ociosas e aguarda a inscrição num buffer do novo retangulo, para continuar sua execução com apenas o retângulo esquerdo. (necessário para garantir ordem da soma)
            2. Se não existe thread ociosa, calcula normalmente os dois retângulos.
    e. Se não passar:
        i. Retorna o valor da área.
4. Agora, a função da thread executa os seguintes comandos:
    a. Calcula a área do retângulo que lhe foi designada;
    b. Quando terminar, anota numa variável global (em exclusão mutua) que já terminou(que está ociosa).
    c. Executa o loop: enquanto a variável não for igual ao numero de instanciadas:
        i. Checa novamente em exclusão mútua a varíavel para ver se há necessidade de espera.
        ii. Se não há, sai do loop e continua execução;
        iii. Se há:
            1. Checa se há algo enviado;
            2. Se não há, espera por um novo intervalo;
            3. Se há, avisa que há uma thread em execução (diminuindo a variável de termino) e adiciona no retangulo calculado na posição do buffer que lhe foi pedida.
    d. Se for igual:
        i. Avisa as threads ociosas (broadcast) que não passaram no teste para seguirem. Assim as threads ociosas rechecam a condição que agora vai ser falsa.

e. Assim, apenas uma thread (por exclusão mútua) imprime o resultado final e altera um bool que apenas serve de aviso se algo já foi impresso ou não.

No código concorrente, era necessário que os dados pudessem ser compartilhados por todas as threads. Acesso simultaneo a um mesmo dado ocorre inúmeros problemas, que normalmente são consertados por exclusão mútua, o que não seria eficiente. Foi interessante então usar um vetor onde cada posição seria um pedaço da soma de um retangulo. Da maneira que foi feita, threads que já terminaram recebem um novo intervalo para ser calculado, e somado assim ao seu resultado anterior. Garante-se também uma maior aproximação ao resultado sequencial, pois massiva parte das contas são feitas na mesma ordem.

Foi decidido então, a seguinte lógica para os casos:

- Temos 4 variáveis: **Threads x Funções x Precisão x Intervalos**
- Utilizamos então:
  - 4 Threads (sequencial, 2, 4 e 8 - 8 threads apenas na máquina AMD);
  - 4 funçoes (dadas no próprio texto do trabalho);
  - 3 precisões:
    - Baixa: 0.001 (10^{-3});
    - Média: 0.000000001 (10^{-9});
    - Alta: 0.000000000000000001 (10^{-18}).
  - 2 Intervalos:
    - Pequenos: 0 a 6;
    - Grandes: -100 a 200;
    - A função para letra b recebeu apenas testes entre 0 e 0.8, devido a sua característica de $-1 < x < 1$.

Temos um total então de **96** casos (4 x 4 x 3 x 2)
Cada caso é testado 3 vezes e feito uma média a partir dos 3 testes (para ter uma aproximação mais próxima do correto, pois as vezes há dependência do processador em relação a outras tarefas , ou qualquer outro problema).

Os testes foram feitos em duas máquinas:

EXCEL:

- **CPU:** Intel I7 2.2ghz
- **RAM:** 4gb 1333mhz
- **OS:** OS X El Capítan 10.11.1
- **HD:** 750gb 5400rpm

TXT:

- **CPU:** AMD FX(tm)-6100 Six-Core Processor 3.3ghz
- **RAM:** 8gb 1333MHz
- **OS:** Mint 17 Cinnamon 64-bit
- **HD:** 1TB 7200rpm