Tópicos Avançados em Computação Paralela e Distribuída

Prof. Laércio Lima Pilla

22 de março de 2016

Lista de Exercícios 1

Esta lista de exercícios é composta por quatro problemas a serem paralelizados. Códigos sequenciais estão disponíveis no Google Drive da disciplina para auxiliar no desenvolvimento dos exercícios. Os problemas estão ordenados em ordem de dificuldade sugerida.

A avaliação será feita baseada na geração de resultados corretos, na criatividade das soluções e na quantidade de problemas paralelizados.

As soluções devem ser enviadas para o email laercio.pilla@ufsc.br com o assunto "*[Computação Paralela] Lista de Exercícios 1*" e deve incluir os nomes das pessoas envolvidas (no máximo duplas).

Para a avaliação do desempenho das propostas de paralelização, sugere-se usar tamanhos de entrada que gerem tempos de execução sequencial acima de um segundo.

Índice

1. Integração numérica através de retângulos (integral.cpp)

1.1 Descrição do problema

Uma forma de se tentar descobrir o valor da integral de uma curva (muitas vezes pensado com a área abaixo de uma curva) envolve a decomposição da curva em retângulos e o somatório das áreas destes. A Figura 1 ilustra um exemplo do método.

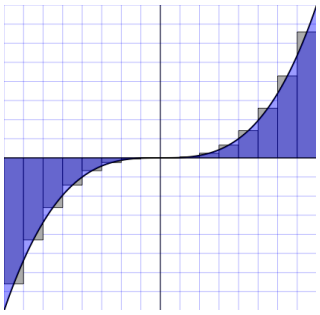


Figura 1. Exemplo de integração numérica.

By Mobius (talk) - http://en.wikipedia.org/wiki/Image:MidRiemann.png, Public Domain, https://commons.wikimedia.org/w/index.php?curid=3539203

O método de integração implementado no exemplo sequencial disponibilizado recebe uma função, os limites do trecho a ser calculado, a quantidade de retângulos a serem calculados e o tipo de ponto a ser tomado para a altura do retângulo (esquerdo, central ou direito). Com base nos valores passados, ele retorna a área estimada.

1.2 Compilação e execução

O código pode ser compilado através do seguinte comando:

```
g++ -o integral integral.cpp -fopenmp -Wall
```

Exemplo de execução e saída (os parâmetros passados são <#retângulos>):

```
./integral 10000
Estimando integrais com 10000 a 1000000 retangulos.
Integral de x^2 de 0 a 1 estimada em 0.333283 em 0.000232 segundos.
Integral de x+2 de 0 a 10 estimada em 69.9995 em 0.001905 segundos.
Integral de x^3+9x^2+18x+27 de 3 a 7 estimada em 1996 em 0.032376 segundos.
```

1.3 Mais informações sobre o problema

https://pt.wikipedia.org/wiki/Integra%C3%A7%C3%A3o_num%C3%A9rica

2. Método de Monte Carlo para a estimativa de π (pi.c)

2.1 Descrição do problema

Métodos de Monte Carlo usam uma amostra de pontos aleatórios para a estimativa de resultados numéricos. Um exemplo é a estimativa do valor de π, o qual é calculado através do sorteio de pontos em um quadrado de lado *r* (*r* = 1 por simplicidade) que cobre um quarto de um círculo com raio *r*. A Figura 2 ilustra o método.

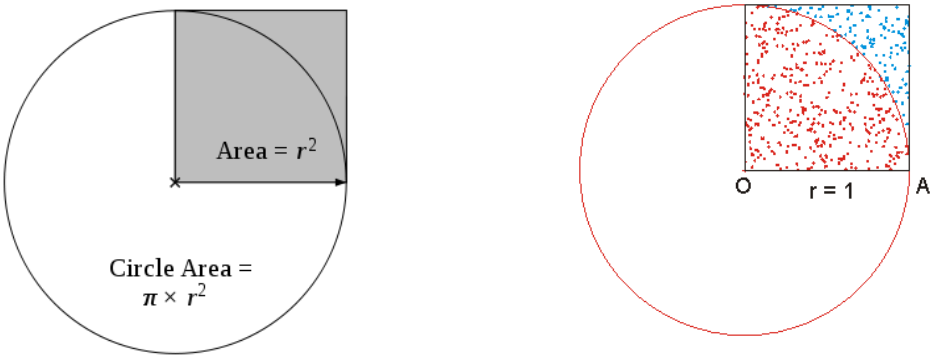


Figura 2. Estimativa de π através de Método de Monte Carlo

Figura da esquerda: By No machine-readable author provided. Limaner assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims)., Public Domain, https://commons.wikimedia.org/w/index.php?curid=770797

Figura da direita: Por Jirah - Obra do próprio, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2912248

A razão entre a área do círculo interna ao quadrado e a área do quadrado é dada pela Equação 1. Dado que os pontos sorteados podem se encontrar dentro ou fora da área do círculo, a razão entre o número de pontos dentro do círculo e o número total de pontos sorteados estima o valor de π/4.

$$(\pi r^2/4)/r^2 = \pi/4 \qquad \text{(Equação 1)}$$

Uma forma simples de descobrir se um ponto (*x*,*y*) se encontra dentro do círculo envolve calcular sua distância à origem (0,0) e verificar se essa distância é menor do que o raio. A distância euclidiana nesse caso é simplificada para o cálculo da hipotenusa.

2.2 Compilação e execução

O código pode ser compilado através do seguinte comando:

```
gcc -o pi pi.c -fopenmp -Wall –lm
```

Exemplo de execução e saída (os parâmetros passados são <#pontos>):

```
./pi 10000
Estimando pi com uma amostra de 10000 pontos
Estimativa do valor de pi = 3.171200
Tempo de execucao = 0.000812 s
```

2.3 Mais informações sobre o problema

https://pt.wikipedia.org/wiki/Pi

3. Merge sort (mergesort.c)

3.1 Descrição do problema

Merge sort é um algoritmo de ordenação baseado na técnica de projeto de algoritmos conhecida como Divisão e Conquista. Ele é conhecido por ter complexidades de pior e médio caso Θ(n log n). Dado um arranjo desordenado, cada etapa do algoritmo é decomposta em três partes:

1) Divisão: os dados do problema são divididos em duas partes (divisão do vetor em primeira metade e segunda metade)
2) Conquista: cada uma das partes é ordenada individualmente (Merge sort recursivo)
3) Combinação: as duas partes do problema ordenadas são combinadas, gerando uma resposta ordenada

A Figura 3 ilustra as etapas do algoritmo Merge sort para um arranjo de sete entradas.

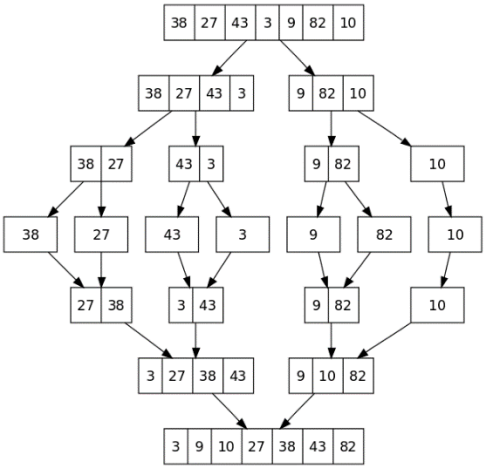


Figura 3. Etapas do Merge sort.

By VineetKumar at English Wikipedia - Transferred from en.wikipedia to Commons by Eric Bauman using CommonsHelper., Public Domain, https://commons.wikimedia.org/w/index.php?curid=8004317

3.2 Compilação e execução

O código pode ser compilado através do seguinte comando:

```
gcc -o mergesort mergesort.c -fopenmp -Wall
```

Exemplo de execução e saída (os parâmetros passados são <#células no arranjo>):

```
./mergesort 100000
Ordenando um vetor de 100000 posicoes.
Ordenacao realizada em 0.022823 s.
Vetor corretamente ordenado.
Verificacao realizada em 0.000601 s.
```

3.3 Mais informações sobre o problema

https://pt.wikipedia.org/wiki/Merge_sort

3.4 Detalhes de implementação

No caso de uma piora no desempenho com a paralelização, o atributo “`if`” pode ser usado junto ao pragma para reduzir o sobrecusto de paralelização. Favor consultar o professor para mais detalhes.

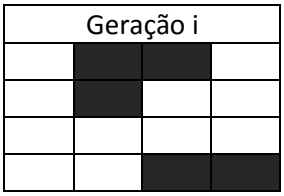
4. Jogo da Vida de Conway (gameoflife.c)

4.1 Descrição do problema

O Jogo da Vida de Conway é um autômato celular que simula a evolução de células em um tabuleiro bidimensional. A cada geração, uma célula no tabuleiro pode estar viva ou morta baseada em seu estado anterior e no estado de seus vizinhos (células ao seu redor) conforme as seguintes regras :

1) Qualquer célula viva com menos de dois vizinhos vivos morre de solidão.
2) Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.
3) Qualquer célula viva com dois ou três vizinhos vivos continua viva na próxima geração.
4) Qualquer célula morta com exatamente três vizinhos vivos se torna uma célula viva.
5) Qualquer célula morta com um número de vizinhos diferente de três continua morta.

A Figura 4 ilustra a evolução de uma geração em um tabuleiro de 4x4, onde as células em branco estão mortas e as células preenchidas estão vivas. Os valores internos às células informa qual das cinco regras foi aplicada para sua evolução



Geração i+1			
5	3	3	5
5	3	4	5
5	5	4	5
5	5	1	1

Figura 4. Evolução de um tabuleiro do Jogo da Vida.

4.2 Compilação e execução

O código pode ser compilado através do seguinte comando:

```
gcc -o gameoflife gameoflife.c -fopenmp –Wall
```

Exemplo de execução e saída (os parâmetros passados são <#iterações> <#colunas> <#linhas>):

```
./gameoflife 1000 50 50
Jogo da Vida de Conway simulado em um espaco 50x50 por 1000 iteracoes
Tempo total de execucao: 0.125499 s.
```

4.3 Mais informações sobre o problema

https://pt.wikipedia.org/wiki/Jogo_da_vida

4.4 Detalhes de implementação

Caso seja de interesse ver a evolução das gerações, as linhas 71 e 73 do código podem ter suas marcações de comentário (//) removidas.