Relatório final

Trabalho: Integracao numerica por regra do Trapezio.
Disciplina: Programação Concorrente (ICP361)
Profesora: Silvana Rossetto
Autor: Eduardo Kota Otomo
DRE: 118147443

Descrição do problema

Programa concorrente escrito em Java para Integração numérica via regra do trapézio.

O objetivo é calcular a integral numérica. `integral.get()`

Projeto e implementacão da solução concorrente.

A interface `Function<Double, Double> f;` permite escolher diferentes funções `f(x)` escolhidas pelo usuário no menu inicial.
A interface `Function<Double, Double> F;` é a primitiva de f(x).

Por exemplo:
`f = x -> x*x*x` <=> `f(x)=x^3`
`F = x -> x*x*x*x/4` <=> `F(x)=x^4/4\n`

`F(x)`usado para calcular a integral exata `F(b)-F(a)` <=> `integralExata = F.apply(b)-F.apply(a);`

Cada thread calcula um subtotal da integral e em seguida soma o resultado ao valor total do recurso compartilhado.

```
class Soma extends Thread{
// ...
  public void run() {
    double h = (b - a)/n;
    for (int i = id; i < n; i += nT) {
      double termo = f.apply(a + i * h) ;
      termo += f.apply(a + (i + 1) * h) ;
      termo *= h;
      termo /= 2;
      subtotal += termo;
    }
    total.inc(subtotal);
  }
// ...
}
```

Exclusão mútua entre threads foi implementado via synchronized para evitar condição de corrida sobre o recurso compartilhado

```
class Total {
  //recurso compartilhado
  private double total;
  public Total() {
     this.total = 0;
  }
  //escrita sobre o recurso compartilhado
  public synchronized void inc(double incremento) {
     this.total += incremento;
  }
  //leitura sobre o recurso compartilhado
  public synchronized double get() {
     return this.total;
  }
}
```

garantindo o mesmo resultado final para cada parâmetro dado pelo usuário.

Caso o usuário insira entradas inválidas, gerando erro de entrada inválida então o erro é tratado e se assume o valor default.

```
System.out.print("Numero de Threads: ");
try{
  nT = sc.nextInt();
  if (nT < 1) {
    nT = 1;
  }
} catch(InputMismatchException e1) {
  nT = 1;
  sc.next();
} finally {
  if (nT==1){
    System.out.println("\n1 Thread\n");
  } else {
    System.out.printf("%d thread(s)\n\n", nT);
  }
}
```

Um timer conta e registra o tempo do momento em que threads são criadas até o término da execução. `long tempo = System.nanoTime() - inicio;`

Testes de corretude

O erro absoluto será a diferença entre integral numérica `integral.get()` e a integral exata `integralExata` `erroAbsoluto = Math.abs(integral.get() - integralExata);`

O erro relativo é obtido do erro absoluto via:

```
erroRelativo = erroAbsoluto;
if (integralExata !=0){
  erroRelativo = erroAbsoluto / integralExata;
}
```

Em todos os casos testado o erro relativo foi bem pequeno nunca excedendo 0.001% comparado ao valor teórico `integralExata`.
Logo os resultados são precisos e acurados.

Avaliação de desempenho.

Testada para as funções: `f(x)= x`, `f(x)= x^2`, `f(x)= x^3`, `f(x)= cos(x)`, `f(x)= sen(x)`.
Nos intervalos `[0,1]` e `[1,2]` 5 vezes cada tomando o valor mediano (=/= médio).
Para 1 ou 4 threads. Onde 1 thread equivale ao caso sequencial.
Para `n = 10` ou `n = 20` subintervalos.

Observamos a seguir que o programa sequencial é um pouco mais lento devido ao tempo de overhead gasto para criar e unir as threads, e somar o resultado de cada thread.

Resultados tabelados

`f(x)= x` integrado em [0,1]

Integral exata: 0.5000000000

subintervalos	n = 10	n = 20
Integral numerica :	0.5000000000	0.5000000000
Erro relativo:	0.0000000000 %	0.0000000000 %
Tempo sequencial:	895001 ns	915100 ns
Tempo p/ 4threads:	1132100 ns	1201602 ns

`f(x)= x` integrado em [1,2]

Integral exata: 1.5000000000

subintervalos	n = 10	n = 20
Integral numerica :	1.5000000000	1.5000000000
Erro relativo:	0.0000000000 %	0.0000000000 %
Tempo sequencial:	896000 ns	910000 ns
Tempo p/ 4threads:	1109650 ns	1191900 ns

`f(x)= x^2` integrado em [0,1]

Integral exata: 0.3333333333

subintervalos	n = 10	n = 20
Integral numerica :	0.3350000000	0.3337500000
Erro relativo:	0.0000500000 %	0.0000125000 %
Tempo sequencial:	915000 ns	959100 ns
Tempo p/ 4threads:	1128200 ns	1132900 ns

`f(x)= x^2` integrado em [1,2]

Integral exata: 2.3333333333

subintervalos	n = 10	n = 20
Integral numerica :	2.3350000000	2.333750000
Erro relativo:	0.0000071429 %	0.0000017857 %
Tempo sequencial:	915100 ns	960100 ns
Tempo p/ 4threads:	1129300 ns	1133000 ns

`f(x)= x^3` integrado em [0,1]

Integral exata: 0.2500000000

subintervalos	n = 10	n = 20
Integral numerica :	0.2525000000	0.2506250000
Erro relativo:	0.0001000000 %	0.0000250000 %
Tempo sequencial:	925100 ns	970100 ns
Tempo p/ 4threads:	1228200 ns	1233010 ns

`f(x)= x^3` integrado em [1,2]

Integral exata: 3.7500000000

subintervalos	n = 10	n = 20
Integral numerica :	3.7575000000	3.7518750000
Erro relativo:	0.0000200000 %	0.0000050000 %
Tempo sequencial:	922100 ns	970530 ns
Tempo p/ 4threads:	1229300 ns	1233000 ns

`f(x)= cos(x)` integrado em [0,1]

Integral exata: 0.8414709848

subintervalos	n = 10	n = 20
Integral numerica :	0.8407696421	0.8412956710
Erro relativo:	0.0000083347 %	0.0000020834 %
Tempo sequencial:	950200 ns	951200 ns
Tempo p/ 4threads:	1090000 ns	1092000 ns

`f(x)= cos(x)` integrado em [1,2]

Integral exata: 0.0678264420

subintervalos	n = 10	n = 20
Integral numerica :	0.0677699106	0.0678123109
Erro relativo:	0.0000083347 %	0.0000020834 %
Tempo sequencial:	949200 ns	951100 ns
Tempo p/ 4threads:	1089000 ns	1093000 ns

`f(x)= sen(x)` integrado em [0,1]

Integral exata: 0.4596976941

subintervalos	n = 10	n = 20
Integral numerica :	0.4593145489	0.4596019198
Erro relativo:	0.0000083347 %	0.0000020834 %
Tempo sequencial:	951300 ns	961900 ns
Tempo p/ 4threads:	1093000 ns	1099000 ns

`f(x)= sen(x)` integrado em [1,2]

Integral exata: 0.9564491424

subintervalos	n = 10	n =2 0
Integral numerica :	3.7575000000	3.7518750000
Erro relativo:	0.0000200000 %	0.0000050000 %
Tempo sequencial:	951100 ns	963100 ns
Tempo p/ 4threads:	1098000 ns	1102000 ns