**Modulo 1 – Trabalho de Implementação (2020/1 REMOTO)**

**Computação Concorrente (MAB-117)**

**Prof. Silvana Rossetto**

1. **Introdução**

O problema escolhido foi o cálculo de π (pi) de forma diferente da abordada em aula. Na ocasião foi solicitada uma implementação do cálculo da variável por meio de uma soma de elementos de uma série infinita e neste trabalho utilizo a integração numérica de uma semicircunferência para calcular seu valor.

1. **Método**

É sabido que a área de uma circunferência é dada por ( π \* R2 ), portanto podemos calcular π como o dobro da área de uma semicircunferência de raio 1. Para tal, quebramos o objeto em retângulos de largura n de forma que a área de cada retângulo é dada pelo produto da altura (fórmula 1) pelo tamanho de cada intervalo com x no ponto médio (fórmula 2)

Fórmula 1:

Fórmula 2:

Desta forma a altura é calculada pelo produto ( y\*n ).

Como estratégia de paralelização o programa coloca cada thread para calcular uma soma parcial dos retângulos atribuídos a elas e somente ao ter concluído todos os seus triângulos a mesma acessa a variável compartilhada (seção crítica) e soma a ela o valor que calculou.

A fim de garantir o balanceamento de carga entre as threads cada uma calcula o retângulo de acordo com seu id, pulando do total de threads até que sejam varridos todos os retângulos.

Ao final do programa, multiplicamos por 2 o valor da variável compartilhada (afinal a função calcula apenas metade da área da circunferência) e obtemos o valor de π.

1. **Resultados**

A seguinte tabela lista os tempos de execução (escolhi os menores dentro de 10 rodados para cada caso) e de ganho de desempenho quando comparado com o tempo sequencial (Ganho de desempenho = Tempo Concorrente / Tempo Sequencial). O tempo de uma thread foi considerado como sendo o sequencial.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Nº de Retângulos | Tempo | Ganho desempenho  (Tseq / Tconc) |
| 1 Thread | 1000 | 0.001248 |
| 1000000 | 0.009226 |
| 1000000000 | 8.063097 |
| 2 Threads | 1000 | 0.001529 | 0.816219751 |
| 1000000 | 0.006767 | 1.363381114 |
| 1000000000 | 4.062308 | 1.984856146 |
| 4 Threads | 1000 | 0.002344 | 0.532423208 |
| 1000000 | 0.0043 | 2.145581395 |
| 1000000000 | 2.092757 | 3.852858693 |
| 8 Threads | 1000 | 0.004947 | 0.252274106 |
| 1000000 | 0.006358 | 1.451085247 |
| 1000000000 | 2.063514 | 3.907459315 |

Com isso é fácil concluir que, apesar de a implementação concorrente ter performance pior para valores baixos, o ganho de desempenho para valores suficientemente grandes de operações tende a aproximadamente ao número de núcleos executando paralelamente o programa (com o ganho de desempenho teórico dado pela lei de Amdahl).