

**INTEGRAÇÃO NUMÉRICA PELA REGRA DE SIMPSON DE 3/8 COMPOSTA**

**FILIPE RODRIGUES**

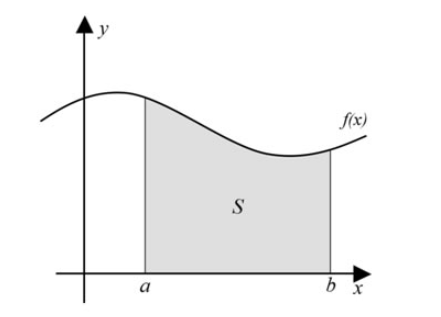
**HEMERSON BATISTA FILHO**

**RAFAELA ROCHA NOGUEIRA**

**RAYDSON FERREIRA CARLOTA**

1. **Definição do problema**

O processo de integração é bastante utilizado para a resolução dos mais variados problemas. Fenômenos físicos e diversos outros contextos relacionados a engenharia, como: problemas de volumes, comprimentos de curvas, estimativas populacionais, trabalho, remanescente de consumo, determinação da posição de um objeto sobre um determinado instante, entre outros problemas, podem ser resolvidos com a utilização da integração. De forma simplificada, a integral de uma função foi originalmente criada para determinar a área sob uma curva no plano cartesiano.



Esse processo pode ser realizado de forma analítica, com manipulações algébricas e técnicas para obter a primitiva em questão. Mesmo com essa solução, para alguns problemas é extremamente difícil, ou até impossível, encontrar uma primitiva. Isto acontece quando as técnicas e manipulações algébricas não possuem suporte para o cálculo da sua primitiva ou quando a primitiva que deseja-se encontrar é de um experimento científico em que pode não haver uma fórmula para a função.

Para problemas como esses, são utilizados métodos numéricos para a obtenção do valor da primitiva. Este valor é uma aproximação. A Regra de Simpson de 3/8 Composta é um exemplo de método para a solução da integração numérica proposto por Thomas Simpson. Pensando nessa problemática, o presente trabalho possui o objetivo de paralelizar a Regra de Simpson 3/8 com a utilização do MPI, na linguagem de programação C++.

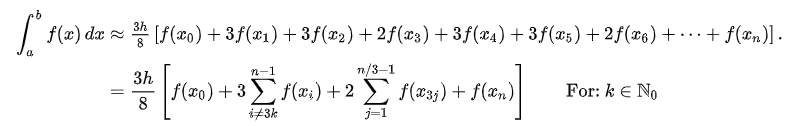
1. **Definição do algoritmo a ser paralelizado**

A Regra de Simpson de 3/8 é baseado em uma interpolação cúbica, obtida através de um caso de Interpolação polinomial de Lagrange. A regra 3/8 de Simpson é definida como a seguir:

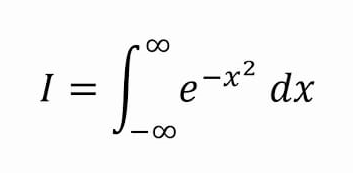


onde .

Dividindo o intervalo em subintervalos de comprimento e introduzindo os nós temos:



O algoritmo desenvolvido resolve cada um desses subintervalos para gerar a aproximação da integral. O programa desenvolvido faz uma integração numérica, integrando de -∞ e +∞.



Porém, por não haver infinito no computador foi escolhido dois valores, um número muito grande negativo e outro positivo para representar o infinito. Foram escolhidos os valores (-100.000 a 100.000).

1. **Decisões de projeto tomadas pelo grupo**

Para a implementação do trabalho, o grupo escolheu a linguagem de programação C++ para o desenvolvimento do algoritmo, visto que todos os participantes possuíam experiência com a linguagem. Na parte de paralelizar o programa decidiu-se pela utilização do MPI, aprendido em sala de aula. O grupo possuía grande interesse na aprendizagem de sua implementação em um problema prático.

Para uma melhor análise dos resultados, decidiu-se então que seria feito um algoritmo para execução sequencial e um paralelizado. Assim, a percepção das diferenças de desempenho entre eles seria visualizada claramente, contribuindo para uma melhor análise do problema.

1. **Descrição do programa paralelo**

O paralelismo é possível e foi usado no cálculo de cada área presente no intervalo entre *a* e *b,* o qual pôde ser feito de maneira que cada processo trabalhasse de forma independente, sendo assim, cada processo ficou responsável por uma parte dessa parcela. No nosso trabalho a quantidade de áreas que são divididas para a realização do cálculo é dada pelo número de processos informados, sendo *n* processos, teremos n áreas entre *a* e *b*, obtendo-se assim o valor da integral através do método pela regra de Simpson de ⅜ composta.

1. **Resultados Obtidos**

Foram realizados testes com 1 (sequencial), 2, 4, 8 e 16 processos e os resultados obtidos podem ser visualizados nas tabelas e gráficos a seguir:

Tabela 1 -Tempo de execução (ms)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Número de interpolações cúbicas** | **1 Processo** | **4 Processos** | **8 Processos** | **12 Processos** | **16 Processos** |
| k = 100,000 | 8,511 | 2,683 | 1,931 | 1,268 | 1,579 |
| k = 200,000 | 14,334 | 4,692 | 4,839 | 4,585 | 2,981 |
| k = 300,000 | 21,476 | 7,801 | 7,39 | 4,996 | 3,955 |
| k = 400,000 | 29,605 | 8,671 | 8,422 | 6,025 | 5,307 |
| k = 500,000 | 35,293 | 12,244 | 9,977 | 10,75 | 7,835 |
| k = 600,000 | 42,487 | 12,883 | 12,072 | 9,391 | 9,717 |
| k = 700,000 | 49,543 | 15,359 | 16,392 | 10,615 | 8,479 |
| k = 800,000 | 56,031 | 17,366 | 16,621 | 17,852 | 8,859 |
| k = 900,000 | 70,605 | 19,223 | 18,635 | 20,685 | 9,825 |
| k = 1,000,000 | 73,549 | 21,099 | 22,242 | 17,271 | 10,241 |

Figura 1 - Gráfico do tempo de execução

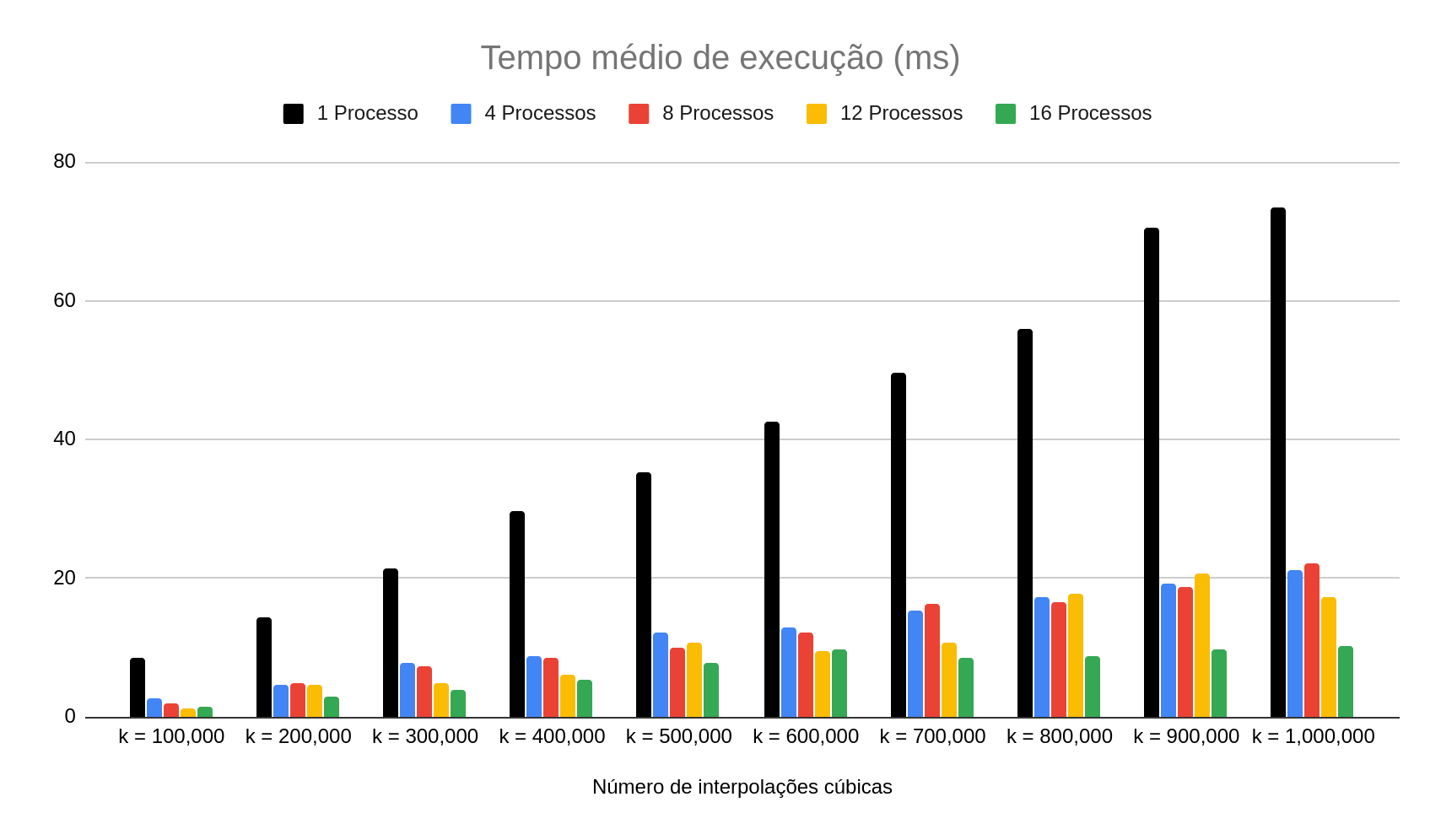


Tabela 2 - Speedup médio

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Número de interpolações cúbicas** | **4 Processos** | **8 Processos** | **12 Processos** | **16 Processos** |
| k = 100,000 | 3,172 | 4,408 | 6,712 | 5,390 |
| k = 200,000 | 3,055 | 2,962 | 3,126 | 4,808 |
| k = 300,000 | 2,753 | 2,906 | 4,299 | 5,430 |
| k = 400,000 | 3,414 | 3,515 | 4,914 | 5,578 |
| k = 500,000 | 2,882 | 3,537 | 3,283 | 4,505 |
| k = 600,000 | 3,298 | 3,519 | 4,524 | 4,372 |
| k = 700,000 | 3,226 | 3,022 | 4,667 | 5,843 |
| k = 800,000 | 3,226 | 3,371 | 3,139 | 6,325 |
| k = 900,000 | 3,673 | 3,789 | 3,413 | 7,186 |
| k = 1,000,000 | 3,486 | 3,307 | 4,259 | 7,182 |

Figura 2 - Gráfico do Speedup médio

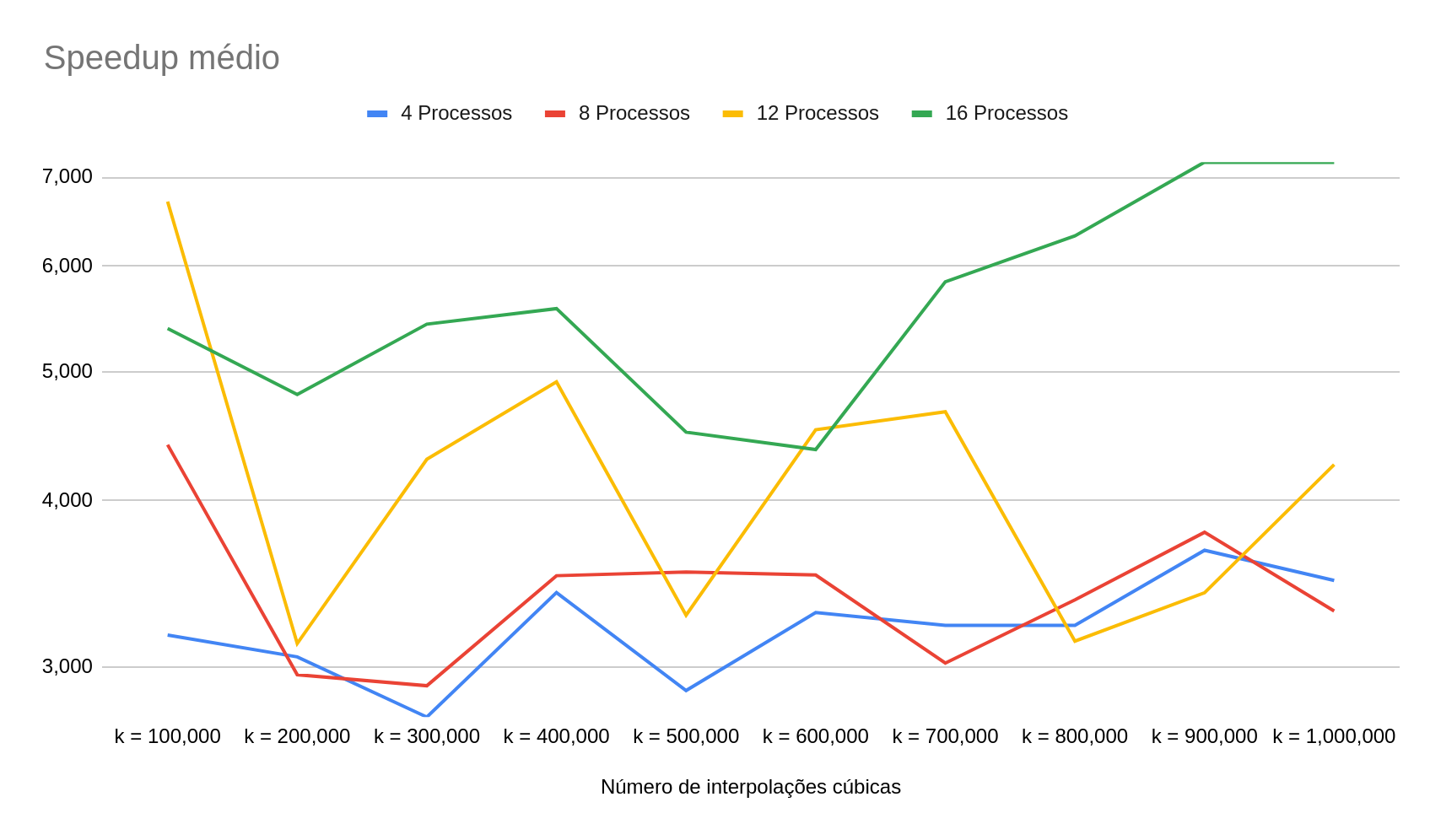
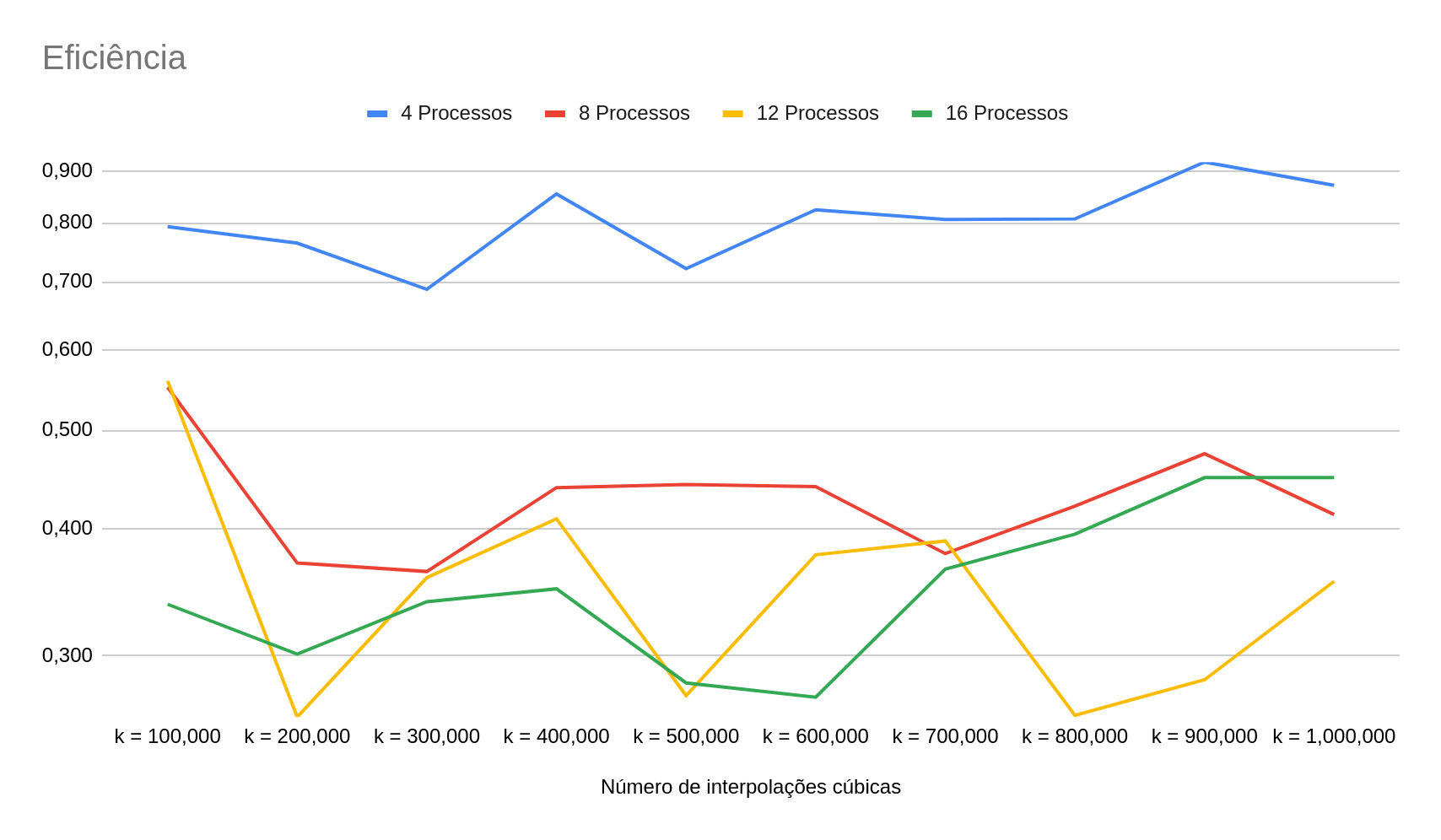


Tabela 3 - Eficiência

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Número de interpolações cúbicas** | **4 Processos** | **8 Processos** | **12 Processos** | **16 Processos** |
| k = 100,000 | 0,793 | 0,551 | 0,559 | 0,337 |
| k = 200,000 | 0,764 | 0,370 | 0,261 | 0,301 |
| k = 300,000 | 0,688 | 0,363 | 0,358 | 0,339 |
| k = 400,000 | 0,854 | 0,439 | 0,409 | 0,349 |
| k = 500,000 | 0,721 | 0,442 | 0,274 | 0,282 |
| k = 600,000 | 0,824 | 0,440 | 0,377 | 0,273 |
| k = 700,000 | 0,806 | 0,378 | 0,389 | 0,365 |
| k = 800,000 | 0,807 | 0,421 | 0,262 | 0,395 |
| k = 900,000 | 0,918 | 0,474 | 0,284 | 0,449 |
| k = 1,000,000 | 0,871 | 0,413 | 0,355 | 0,449 |

Figura 3 - Gráfico da eficiência



1. **Análise dos Resultados**

Como pode ser observado nas tabelas e gráficos descritos nas seções anteriores, a execução do problema com o uso do paralelismo teve um resultado muito eficiente e compensatório, pois, foi identificado que o tempo de execução em paralelo comparado com o tempo de execução sequencial, foi reduzido por mais da metade, e junto a isso, a conforme o número de processos era aumentado, obtinha-se um tempo ainda menor. Isto foi possível, primeiramente pelo tamanho das entradas e da saída, por terem um volume de dados pequeno, a informação de resposta reduz muito o problema de *overhead* paralelo, proporcionando um ganho no desempenho ao executar o algoritmo de modo paralelo.

Outro ponto importante que foi observado, é que a complexidade do cálculo de área utilizando o método de integração de Simpson, por se tratar de uma operação com complexidade de O(n), quanto maior for o número de processos, menor será o trabalho que um cada processo terá que realizar, resultando assim um ganho muito grande de desempenho ao utilizar o paralelismo.