Problema da Montanha Russa

**Alison Orita, Hadyne Biazoto**

Curso de Bacharelado em Informática – Universidade Estadual de Maringá (UEM)

Av. Colombo, 5790, Vila Esperança – Maringá – PR– Brasil

Keijo.alison@hotmail.com, hadynecb@gmail.com

***Abstract.*** *This article describes the problem of the roller coaster and its development in the programming language C using IPC (Inter-Process Communication) that involves dispute over mutually exclusive resources*

***Resumo.*** *Este artigo descreve o problema da montanha russa e seu desenvolvimento na linguagem de programação C utilizando o IPC (Inter-Process Communication) que envolve disputa por recursos mutuamente exclusivos*

# 1. Introdução

O Problema da Montanha Russa é um exemplo de um problema muito comum em controle de processos que pode utilizar o conceito e desenvolvimento em processos. Para resolvê-lo deve-se usar princípios de programação concorrente. Este problema inicialmente utiliza dois processos: os passageiros e os carrinhos.

Neste artigo, iremos apresentar detalhes de implementação do problema da montanha russa utilizando memória distribuída e implementação sequencial, na limguagem C. Em memória distribuída serão utilizadas os recursos IPC (Inter-Process Communication) com MPI (Message Passing Interface), ou seja, o uso de diversos processos para solucionar o problema.

Também apresentaremos comparativos e análises de seus resultados, calculando o Speedup Téorico, Lei de Amdahl, etc.

# 2. Problema

Existem **N** passageiros que aguardam na fila da montanha russa, à espera de um carrinho, para poder realizar o passeio no brinquedo. Cada passageiro poderá entrar apenas em um carrinho, podendo a montanha russa ter mais de um. Todos os carrinhos possuem **C** capacidade de passageiros (C < N) e só começa seu percurso se estiver lotado ou se comportar o restante dos passageiros. Após um tempo determinado, o carrinho retorna a origem, os passageiros saem do carrinho e os que aguardavam entram para iniciar o percurso detalhado.

Descrevendo melhor as condições do problema vamos considerar as seguintes situações:

- Nenhum passageiro irá saltar do carrinho enquanto o mesmo estiver em movimento;

- Nenhum passageiro que estiver aguardando irá entrar no carrinho enquanto o mesmo estiver em movimento;

- O carrinho irá partir apenas quando estiver cheio com exatamente C passageiros ou quando comportar o resto dos passageiros na fila de espera

- Nenhum passageiro pedirá por outro passeio

# 3. Solução com Memória Distribúida

Para resolver este problema utilizamos a técnica do Gerente/Trabalhador. O painel de controle da montanha russa será o processo com o papel do gerente. A partir dele os passageiros serão designados aos seus respectivos carrinhos e suas quantidades serão controladas para anunciar o fim da execução. O papel dos trabalhadores serão atribuídos aos carrinhos, tendo a função de receber os passageiros e quando atingir a capacidade máxima, realizar o passeio. Apenas processo de *rank* igual a zero será o gerente todo resto trabalhador.

Inicialmente o método *controleCarr()* (painel de controle da montanha russa) notifica (por meio da passagem de mensagem) que um passageiro será enviado para cada carrinho da montanha russa e imediatamente aguarda resposta dos mesmos. A partir deste ponto o controle entrará num laço onde aguarda a notificação (mensagem) do processo carrinho, e designa novos passageiros para este mesmo, até o fim da execução, ou seja, quando o número **N** de clientes for atendido (variável *passAtendido* for maior ou igual a *N*).

O método *carrinho()* (carrinho da montanha russa) recebe a mensagem do controle e verifica se adicionando mais este passageiro já está pronto para partir, ou aguardar novos passageiros. Esta verificação ocorre respeintando a condição *nPassNoCarr < CAP,* sendo elas o número de passageiro dentro do carrinho e sua capacidade, respectivamente.

# 4. Análise

Abaixo iremos calcular o Speedup, a Eficiência, a Redundância, a Uitilização, a Qualidade, Lei de Amdahl, Lei de Gustafson-Barsis, Lei de Karp-Flatt e a Isoeficiência da implementação Sequencial e da implementação com uso da Memória Distribuída. Utilizaremos para o teste 10.000 e 100.000 passageiros. O número de instruções foi capturado através da ferramenta PERF.

**Tabela 1. Tabela de tempo/instrução da execução sequencial**

|  |  |  |
| --- | --- | --- |
| Nº Clientes | Tempo | Nº Instrução |
| 10.000 | 00:20,008 | 40.535.849.719 |
| 100.000 | 03:53,333 | 433.974.690.105 |

## 4.1. Speedup

Relação entre o tempo de execução sequencial T(1) e o de tempo de execução em paralelo T(p) em segundos. “p” representa o número de processos.

**Tabela 2. Tabela Speedup com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| T(p) | 00:20,008 | 00:06,878 | 00:3,727 | 00:03,240 | 00:03,368 |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |

**Tabela 3. Tabela Speedup com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| T(p) | 03:53,333 | 00:35,112 | 00:34,837 | 00:24,159 | 00:26,294 |
| S(p) | 1 | 6,645 | 6,697 | 9,658 | 8,874 |

**Figura 1. Análise Speedup**

## 4.2. Eficiência

Relação entre o grau de desempenho (S(p)) e os recursos computacionais disponíveis (p).

**Tabela 4. Tabela Eficiência com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |
| E(p) | 1 | 0,727 | 0,671 | 0,514 | 0,371 |

**Tabela 5. Tabela Eficiência com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 6,645 | 6,697 | 9,658 | 8,874 |
| E(p) | 1 | 1,661 | 0,837 | 0,804 | 0,554 |

**Figura 2. Análise Eficiência**

## 4.3. Redundância

Relação entre o número de operações realizadas pela execução paralela (O(p)) e pela execução sequencial (O(1)).

**Tabela 6. Tabela Redundância com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| O(p) | 40.535.849.719 | 104.114.656.038 | 77.718.652.902 | 65.173.198.924 | 63.062.691.788 |
| R(p) | 1 | 2,568 | 1,917 | 1,607 | 1,555 |

**Tabela 7. Tabela Redundância com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| O(p) | 433.974.690.105 | 1.042.544.776.595 | 818.712.211.237 | 645.190.512.964 | 584.276.212.056 |
| R(p) | 1 | 2,402 | 1,886 | 1,486 | 1,346 |

**Figura 3. Análise Redundância**

## 4.4. Utilização

Relação entre a capacidade computacional utilizada durante a computação e a capacidade disponível (relação entre a Redundância e a Eficiência).

**Tabela 8. Tabela Utilização com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| R(p) | 1 | 2,568 | 1,917 | 1,607 | 1,555 |
| E(p) | 1 | 0,727 | 0,671 | 0,514 | 0,371 |
| U(p) | 1 | 1,867 | 1,286 | 0,827 | 0,577 |

**Tabela 9. Tabela Utilização com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| R(p) | 1 | 2,402 | 1,886 | 1,486 | 1,346 |
| E(p) | 1 | 1,661 | 0,837 | 0,804 | 0,554 |
| U(p) | 1 | 3,991 | 1,579 | 1,196 | 0,746 |

**Figura 4. Análise Utilização**

## 4.5. Qualidade

Relação entre a capacidade computacional utilizada durante a computação e a capacidade disponível (relação entre a Redundância e a Eficiência).

**Tabela 10. Tabela Qualidade com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |
| E(p) | 1 | 0,727 | 0,671 | 0,514 | 0,371 |
| R(p) | 1 | 2,568 | 1,917 | 1,607 | 1,555 |
| Q(p) | 1 | 0,823 | 1,878 | 1,976 | 1,417 |

**Tabela 11. Tabela Qualidade com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 6,645 | 6,697 | 9,658 | 8,874 |
| E(p) | 1 | 1,661 | 0,837 | 0,804 | 0,554 |
| R(p) | 1 | 2,402 | 1,886 | 1,486 | 1,346 |
| Q(p) | 1 | 4,595 | 2,972 | 5,228 | 3,655 |

**Figura 5. Análise Qualidade**

## 4.6. Lei de Amdahl

A lei de Amdahl nos fornece o Speedup máximo que a aplicação paralela com p processadores pode alcançar, não considerando que a fração sequencial do código aumenta conforme aumentam os processadores.

**Tabela 12. Tabela Lei de Amdahl com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |
| S(p) Amdahl | 1 | 1,266 | 1,428 | 1,542 | 1,579 |

**Tabela 13. Tabela Lei de Amdahl com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 6,645 | 6,697 | 9,658 | 8,874 |
| S(p) Amdahl | 1 | 1,282 | 1,434 | 1,583 | 1,665 |

**Figura 6. Análise Lei de Amdahl**

## 4.7. Lei de Gustafson-Barsis

A lei de Gustafson-Barsis nos fornece o Speedup máximo que a aplicação paralela com p processadores pode alcançar, considerando que a fração sequencial do código aumenta conforme aumentam os processadores.

**Tabela 14. Tabela Lei de Gustafson-Barsis com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |
| S(p) G-B | 1 | 2,173 | 2,701 | 2,834 | 2,958 |

**Tabela 15. Tabela comparativa da Lei de Gustafson-Barsis com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 6,645 | 6,697 | 9,658 | 8,874 |
| S(p) G-B | 1 | 2,125 | 2,616 | 2,856 | 2,982 |

**Figura 7. Análise Lei de Amdahl**

## 4.8. Lei de Karp-Flatt

**Tabela 16. Tabela Karp-Flatt com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |
| e | 1 | 0,125 | 0,070 | 0,085 | 0,112 |

**Tabela 17. Tabela Karp-Flatt com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| S(p) | 1 | 2,908 | 5,368 | 6,175 | 5,940 |
| e | 1 | -0,132 | 0,027 | 0,022 | 0,053 |

**Figura 8. Análise Lei de Karp-Flatt**

## 4.9. Eficiência e Escalabilidade

**Figura 9. Eficiência**

**Figura 10. Escalabilidade**

## 4.10. Métrica da Isoeficiência

**Tabela 18. Tabela Isoeficiência com 10.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| E(p) | 1 | 0,727 | 0,671 | 0,514 | 0,371 |
| c | 1 | 2,666 | 2,039 | 1,060 | 0,590 |

**Tabela 19. Tabela Isoeficiência com 100.000 passageiros**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1p | 4p | 8p | 12p | 16p |
| E(p) | 1 | 1,661 | 0,837 | 0,804 | 0,554 |
| c | 1 | -2,512 | 5,143 | 4,124 | 1,245 |

# 5. Conclusão

Verificando os resultados da análise, concluímos que a nossa implementação com memória distribuída, para o problema da montanha russa (com vários carrinhos), se mostrou mais eficiênte quando trabalhado com grande quantidade de dados. Processando 100.000 mil passageiros, o programa teve seu melhor desempenho quando executado com 4 processos, assim como mostra o gráfico de qualidade. A paralelização deixou de compensar quando utilizado 16 processos, o programa começou a demorar mais para resolver o problema em questão. Podemos dizer também que o programa é escalável por manter uma eficiencia relativa à medida que o problema aumentou.

# 6. Referências

Aula 3: Programação Concorrente - Desempenho. Disponível: http://moodlep.uem.br/pluginfile.php/78064/mod\_resource/content/0/Aula%203%20-%20Desempenho.pdf.

Aula 8: Programação Concorrente –Implementação de Memória Distribuída. Disponível: http://moodlep.uem.br/pluginfile.php/81453/mod\_resource/content/0/Aula%208%20-%20Implementa%C3%A7%C3%A3o%20de%20Mem%C3%B3ria%20Distribuida.pdf.

Aula 9: Programação Concorrente – Técnicas Algorítmicas. Disponível: http://moodlep.uem.br/pluginfile.php/81454/mod\_resource/content/0/Aula%209%20-%20T%C3%A9cnicas%20Algor%C3%ADtmicas.pdf.