Segundo Trabalho – Problema do Mercado Modelado em Memória Distribuída

Larissa Yuri Matsuda, Thiago Bucalon

Curso de Bacharelado em Sistemas de Informação – Universidade Estadual de Maringá (UEM) – Campus de Maringá
87020-900 – Maringá – PR– Brasil

ra76743@uem.br, ra68962@uem.br

Resumo. *A necessidade de se fazer várias coisas simultaneamente aparece frequentemente na computação. Dessa forma, o termo programação concorrente está solidamente estabelecido no Brasil. Desenvolver um programa com apenas um fluxo de execução pode gerar um custo altíssimo devido ao tempo de execução que esse programa apresentará, assim cada vez mais estudos confirmam que a paralelização de programas traz diversas melhorias e não apenas na redução do tempo de execução. Esse trabalho tem o objetivo de apresentar a diferença dos resultados do problema do mercado sendo implementado na forma sequencial e paralela.*

1. Introdução

Devido ao elevado poder de processamento dos computadores atuais, bem como a quantidade de núcleos de processamento disponíveis por máquinas, e do custo de aquisição bem inferior ao antes praticado. A programação paralela vem se tornando uma alternativa atrativa para o desenvolvimento de aplicativos mais eficientes e voltados, sobretudo para a resolução de problemas de elevada escala.

Como o aumento do poder computacional, problemas cada vez maiores e mais complexos vêm sendo estudados e resolvidos. Alguns problemas, no entanto possuem dimensão e complexidade tão elevadas que com a utilização da computação tradicional, sequencial, o tempo necessário para obter a solução do problema pode ser tão grande que o torna impraticável.

Na ciência, muitos problemas podem ser modelados matematicamente em termos de sistemas de equações lineares. Para estes sistemas existem vários métodos de resolução já bastante utilizados, principalmente no que diz respeito a matrizes. Muitos exemplos podem ser citados sobre sua vasta aplicação prática em diversas áreas do conhecimento entre muitas outras aplicações na ciência e na engenharia. Devido ao vasto campo de aplicação para sistemas de equações lineares, o estudo de métodos numéricos mais eficientes é necessário.

Dependendo do modelo a ser estudado e das características do problema, as simplificações não serão permitidas e os recursos computacionais podem se mostrar extremamente limitados. Os limites podem ser físicos, como a quantidade de memória requerida e o tempo de cálculo, que depende do problema e do processo de resolução como um todo (método numérico utilizado, precisão, convergência e entre outros).

Primeiramente exibiremos descrição detalhada do problema do supermercado. Em seguida, a modelagem do problema e a análise dos resultados. Por último uma conclusão sobre o desenvolvimento do trabalho e as referências bibliográficas.

2. Descrição do Problema

A proposta desse trabalho é implementar e analisar os resultados de uma versão paralela do Problema do Supermercado, utilizando uma arquitetura de memória distribuída. O algoritmo é composto por uma estrutura de dados, estruturando a quantidade de caixas de pagamento que o supermercado comporta e considerando que há um empregado em cada caixa. Algumas especificação são concedidas para a formulação do problema, como:

- Assim que o cliente for direcionado para o atendimento dos caixas de pagamento, ele deve escolher o caixa com a menor fila.
- Enquanto houver clientes na fila do caixa do empregado ele deve atendê-los.
- Caso não há mais cliente na fila de um dado empregado, ele deve buscar por clientes que estejam em outras filas. Essa busca deve ser pelo caixa que contenha a maior fila naquele instante.
- Caso não haja mais clientes para sempre atendidos na fila do empregado e nem há clientes para se dirigir ao seu caixa, então o empregado se encontra no modo ocioso.
- Uma vez que o cliente escolheu seu caixa, ele não pode mais trocar, exceto para ser atendido conforme descrito anteriormente.
- O número de clientes que se encontram na fila de cada caixa é ilimitada.

3. Modelagem do Problema

Para esse trabalho realizou-se duas modelagens, a modelagem sequencial e a modelagem paralela com memória distribuída. A modelagem sequencial refere-se ao programa sequencial. Nesse programa existe somente um fluxo de controle, de modo que o problema é dividido em uma série de instruções que são executadas uma após a outra e uma dada instrução só pode ser executadas em um determinado instante de tempo.

A modelagem paralela refere-se ao programa concorrente, podendo ser visto como se tivesse vários fluxos de execução. No programa concorrente pode-se utilizar múltiplos recursos computacionais para resolver um problema, de modo que o problema e dividido em partes que podem ser resolvidas concorrentemente e cada uma dessas partes representa um grupo de instruções. Os dois ambientes para desenvolvimento de programas concorrentes são o ambiente de passagem de mensagem, ou memória distribuída, e o ambiente de memória compartilhada.

No ambiente de memória distribuída, as informações são por meio de troca de mensagens, e nesse caso o responsável pelo mecanismo de comunicação entre os processadores é o Sistema Operacional (SO). De modo geral, esse tipo de programação tem uma memória única para cada processador, portanto, existe uma rede de vários computadores conectados.

3.1. Modelagem Sequencial

Nesse modelo o número de caixas e o número de clientes para serem atendidos são definidos logo no início do programa e esse modelo possui três funções principais, a função main( ), a função Cliente( ) e a função Atendente( ). A função main( ) é responsável por inicializar o vetor tamfila, sendo o valor inicial igual a zero, pois no início do programa todas as filas devem estar vazias. A seguir tem-se um laço de repetição onde a condição de parada é ter alcançado o número de clientes a serem atendidos ou que não

haja nenhum cliente na fila de algum caixa. Dentro desse laço é feito um sorteio randômico entre zero e um, de modo que se o número sorteado for igual a zero e o número de clientes ainda não foi atingido é chamado a função Cliente( ). Caso o número sorteado for igual a um e tiver algum cliente na fila é chamado a função Atendente( ). Quando sair desse laço de repetição significa que atingiu o número total de clientes.

A função Cliente( ) é responsável por inserir o cliente que "acabou de chegar" na menor fila dos caixas de pagamento. Para isso, é realizado uma busca pelo caixa que possui a menor fila e então é inserido o cliente nela. A função Atendente( ) é responsável por atender os clientes que estão na fila. Para isso é feito um sorteio variando de zero até o número de caixas menos um para determinar qual caixa irá atender o seu cliente. Feito a escolha do caixa é verificado se o caixa selecionado possui algum cliente na fila, existindo dois casos, O primeiro caso é quando não há ninguém na fila do caixa e ele realiza uma busca pelo caixa que possui a maior fila e atende o cliente que esteja nessa maior fila. O segundo caso, quando há algum cliente na fila do caixa sorteado, é realizado o atendimento do cliente, logo o número de clientes na fila diminui um.

3.2. Modelagem Paralela

Nessa modelagem temos como critério de parada a quantidade máxima de clientes que devem ser atendidos pelos caixas do supermercado. As três funções principais implementadas foram, main( ), Distribui_Cliente( ) e Atende_Cliente( ). A função main() é responsável pela inicialização da biblioteca MPI e pela organização das tarefas de cada processo, de modo que o processo com o número de "*rank*" igual a zero é responsável pela distribuição dos clientes aos caixas de atendimento e o restante dos processos são responsáveis pelo atendimento dos clientes. Esse atendimento foi programado para que cada processo seja um caixa de atendimento.

A função Distribui_Cliente( ), como o próprio nome diz, é encarregado pelo gerenciamento da distribuição dos clientes aos caixas de atendimento. Dessa forma, a função realiza um sincronização com todos os processos para saber qual o tamanho da fila de cada caixa. Adquirido esse dado a função calcula qual é o caixa com a menor fila, pois será esse caixa que receberá um novo cliente. Selecionado o caixa, a função envia uma mensagem avisando o caixa selecionado que a sua fila irá aumentar, pois um cliente entrará nela. Um outra mensagem que essa função envia aos demais processos é a quantidade de clientes que chegaram para serem atendidos por algum caixa. Logo, essa função terminará sua execução quando o número de clientes que chegaram para serem atendidos igualar ao critério de parada.

A função Atende_Cliente( ), é a função responsável por atender os clientes que estão na fila e o critério de parada é conseguir atender todos os clientes. No início da função é realizado uma sincronização com todos os processos para que todos tenham conhecimento do tamanho da fila do outro. Porém essa sincronização possui uma verificação referente ao número de clientes que chegaram, pois enquanto a variável "chegados" não atingir o seu valo máximo, que é a condição de parada, o processo que executa a função Distribui_Cliente( ) também precisa ter o conhecimento do tamanho da fila de cada caixa para que possa escolher qual caixa receberá o novo cliente. Além de que os processos que executam a função Atende_Cliente( ) precisam estar atualizados do valor da variável "chegagos" e se eles receberam mais um cliente na fila (vaiável "miasum"). Após essa sincronização e atualização dos valores das variáveis começa a etapa de atender o cliente que está na fila.

Caso não haja nenhum cliente para ser atendido, isto é, todas as filas estão vazias o caixa executa a função Espera( ) e retorna ao início do loop para tentar atendar algum cliente. Caso haja algum cliente a ser atendido é feito a verificação se a fila do caixa está vazia ou não. Se não estiver vazia ele realiza o atendimento do cliente que está em sua fila e avisa aos demais processos responsáveis pelo atendimento que ele não pegou cliente da fila de ninguém. Caso a fila esteja vazia ele busca pelo caixa que contenha a maior fila, feito esse cálculo o processo avisa ao caixa com a maior fila que ele atendeu um cliente da fila dele. Após a etapa de atendimento é realizado a atualização das variáveis para saber se algum caixa atendeu o cliente de outro caixa e qual é o total de clientes atendidos até o momento. Feito isso, o processo retorna ao início do loop e procede com as etapas até que atenda a todos os clientes.

A função Espera( ) é responsável por deixar o processo "dormindo" por um dado tempo. Ela foi implementada para que os processos responsáveis pelo atendimento quando não houverem nenhum cliente a ser atendido eles possam esperar um dado tempo até retornarem ao início da função, pois nesse dado tempo em que ficam "dormindo" é provável que algum cliente seja inserido em algum fila.

4. Método de Avaliação

O método de avaliação das aplicações se resume a métricas disponibilizadas na literatura sobre programação concorrente. Dentre as principais métricas, pode-se dividir em medidas para avaliar o desempenho que são:

- ***Speedup:*** mede o grau de desempenho. Relação entre o tempo de execução sequencial e o tempo de execução paralelo. Abaixo sua equação:

$$S(p) = \frac{T(1)}{T(p)}$$

  Onde, T(1) é o tempo de execução com um processador e T(p) é o tempo de execução com "p" processadores.

- **Eficiência:** mede o grau de aproveitamento dos recursos computacionais. É a relação entre o *Speedup* e os recursos computacionais disponíveis. Abaixo sua equação:

$$E(p) = \frac{S(p)}{p}$$

- **Redundância:** mede o grau de aumento da computação. Mede a relação entre o número de operações realizadas pela execução paralela e pela execução sequencial. Abaixo sua equação:

$$R(p) = \frac{O(p)}{O(1)}$$

  Onde, O(p) é o número de operações realizas com "p" processadores e O(1) é o número de operações realizadas com um processador.

- **Utilização:** mede o grau de aproveitamento da capacidade computacional. Relação entre a capacidade computacional utilizada durante a computação e a capacidade disponível. Abaixo sua equação:

$$U(p) = R(p) \times E(p)$$

- **Qualidade:** mede o grau de importância de utilizar programação paralela. Abaixo sua equação:

$$Q(p) = \frac{S(p) \times E(p)}{R(p)}$$

5. Avaliação dos Resultados

Cada aplicação foi executada 2 vezes para cada valor de processadores. Assim, para cada número de processador é obtido um tempo médio de execução. Para analisar os dados obtidos e poder assim realizar os gráficos de *Speedup*, foram separados em tabelas as médias dos tempos de execução para cada número de processadores. A tabela inicial conterá os tempos de execução e o número de operações executadas, já a segunda tabela conterá todos os resultados das métricas citadas no capítulo anterior.

Tabela 1: Dados do Programa Sequencial.

Nº de Caixas	Total de Clientes	Tempo de Execução	Nº de Operações
1	100	3min 26.757s	371
1	200	8min20.169s	371
1	300	12min30.312s	371
1	400	16min40.813s	371
1	500	20min50.504s	371

Tabela 2: Dados da Execução Paralela, memória distribuída.

Nº de Processadores	Total de Clientes	Tempo de Execução	Nº de Operações
2	100	0.056s	880
3	100	4min 43.622s	880
4	100	2min 23.289s	880
5	100	2min 7.572s	880
6	100	1min 59.463s	880
2	200	0.064s	880
3	200	8min 27.048s	880
4	200	3min 6.432s	880
5	200	2min 15.488s	880
6	200	2min 32.878s	880
2	300	0.072s	880

3	300	6min2.807s	880
4	300	3mim 26.949s	880
5	300	2min19.769s	880
6	300	3min 8.636s	880
2	400	0.108s	880
3	400	8min 6.992s	880
4	400	3min 57.823s	880
5	400	2min25.849s	880
6	400	3min 30.117s	880
2	500	0.156s	880
3	500	10min4.8146s	880
4	500	4min 7.643s	880
5	500	2min14.348s	880
6	500	3min 52.607s	880

Após as execuções, os dados obtidos foram analisados e as métricas citadas no capítulo anterior foram calculadas. Na Tabela 3 e a Tabela 4 estão disponíveis os resultados para cada medida avaliada:

Tabela 3: Cálculo do Speedup, Eficiência e Redundância.

Nº de Processadores	**Total de Clientes**	***Speedup***	**Eficiência**	**Redundância**
2	100	3692.08929	1846.04464	2.37197
3	100	0.728989	0.242996	2.37197
4	100	1.442937	0.360734	2.37197
5	100	1.628547	0.32571	2.37197
6	100	1.73072	0.28845	2.37197
2	200	7815.5625	3907.78125	2.37197
3	200	0.98643	0.32881	2.37197
4	200	2.68285	0.67071	2.37197
5	200	3.69181	0.73836	2.37197
6	200	3.27169	0.54528	2.37197

2	300	10837.66667	5418.8333	2.37197
3	300	2.06807	0.68936	2.37197
4	300	3.62559	0.9064	2.37197
5	300	5.36823	1.07365	2.37197
6	300	3.97757	0.66293	2.37197
2	400	9266.7870	4633.3935	2.37197
3	400	2.05509	0.68503	2.37197
4	400	4.20823	1.05206	2.37197
5	400	6.86198	1.3724	2.37197
6	400	4.76312	0.79385	2.37197
2	500	83366.9333	41683.4667	2.37197
3	500	2.06758	0.68919	2.37197
4	500	5.04962	1.26241	2.37197
5	500	9.30795	1.86159	2.37197
6	500	5.37604	0.89601	2.37197

Tabela 4: Cálculo da Utilização e Qualidade.

Nº de Processadores	Total de Clientes	Utilização	Qualidade
2	100	4378.76251	2873460.307
3	100	0.576379	0.07468
4	100	0.85565	0.21944
5	100	0.77257	0.22363
6	100	0.68420	0.21047
2	200	9269.13989	12876009.64
3	200	0.77993	0.13674
4	200	1.5909	0.75861
5	200	1.75137	1.14921
6	200	1.293387	0.75211
2	300	12853.31002	24758959.45

3	300	1.63514	0.60104
4	300	2.14995	1.38545
5	300	2.54666	2.42988
6	300	1.57245	1.11167
2	400	10990.27038	18101692.12
3	400	1.62487	0.59351
4	400	2.49545	1.86651
5	400	3.25529	3.97028
6	400	1.88298	1.59412
2	500	98871.93251	1465036563
3	500	1.63474	0.60074
4	500	2.9944	2.68751
5	500	4.41563	7.30515
6	500	2.12253	2.0308

A seguir temos o Gráfico 1 até o Gráfico 5 representamos o *seepdup* e o Gráfico 6 até o Gráfico 10 representando a qualidade.

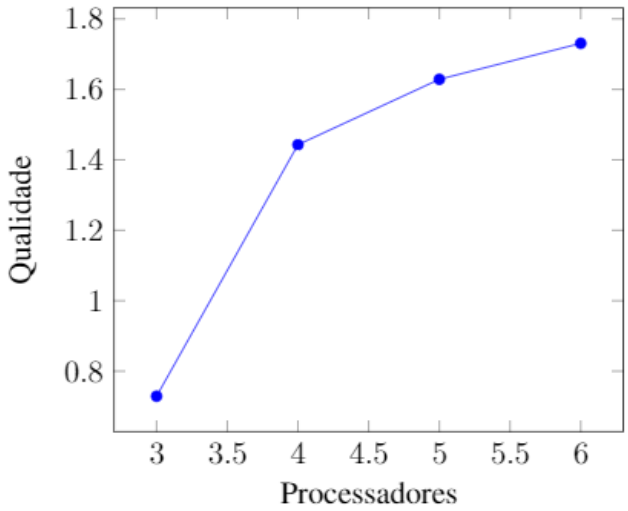

Gráfico 1: Speedup de 100 clientes.

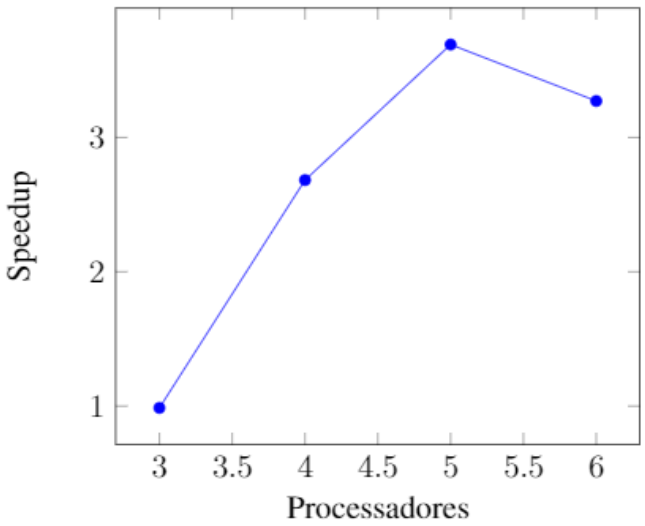

Gráfico 2: Speedup de 200 clientes.

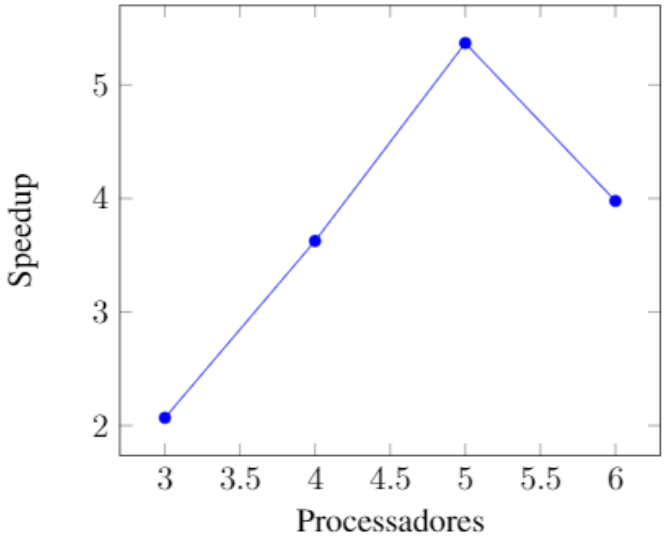

Gráfico 3:Speedup de 300 clientes.

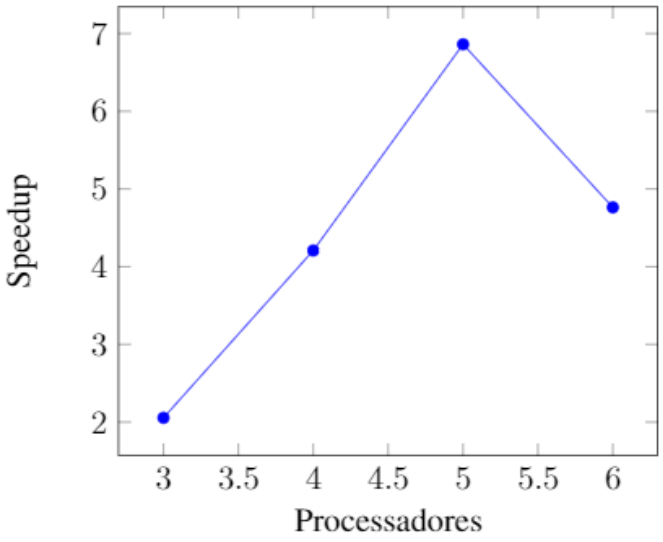

Gráfico 4: Speedup de 400 clientes.

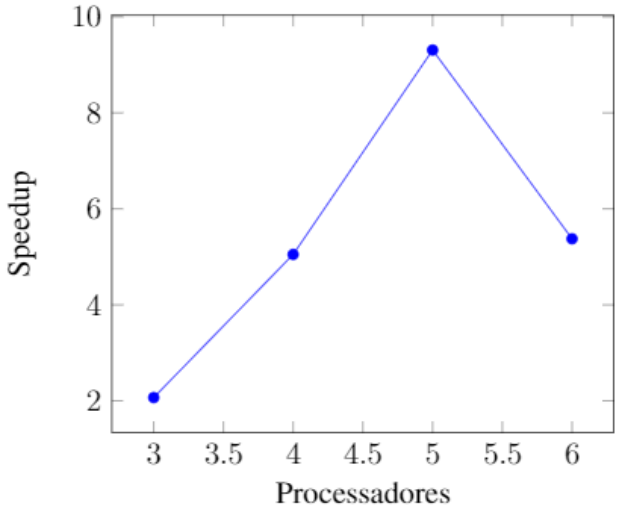

Gráfico 5: Speedup de 500 clientes.

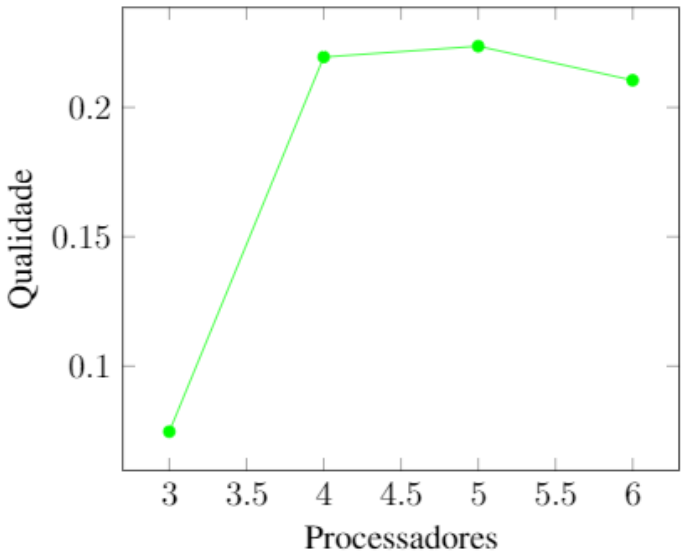

Gráfico 6: Qualidade de 100 clientes.

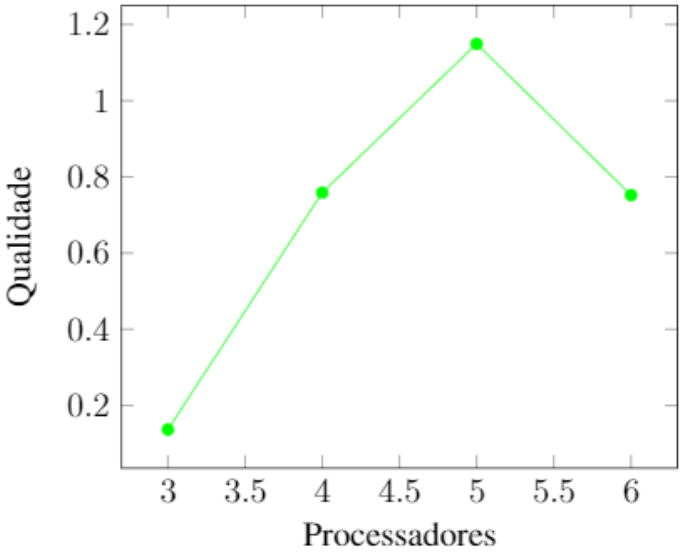

Gráfico 7: Qualidade de 200 clientes.

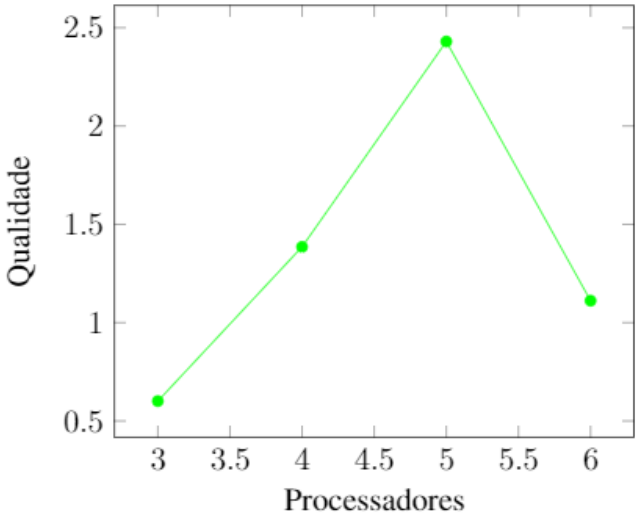

Gráfico 8: Qualidade de 300 clientes.

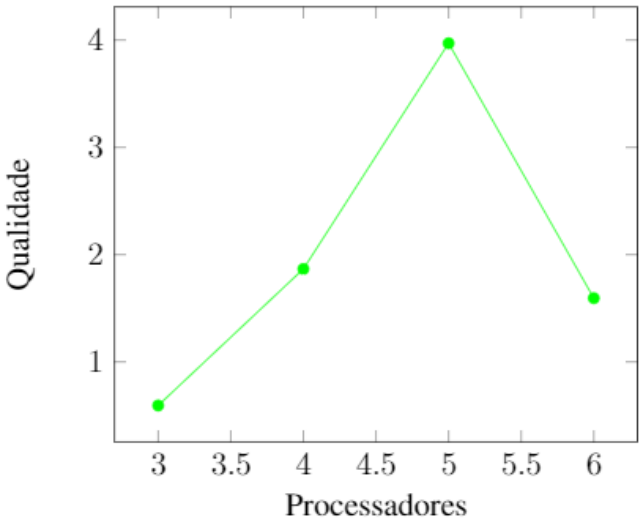

Gráfico 9: Qualidade de 400 clientes.

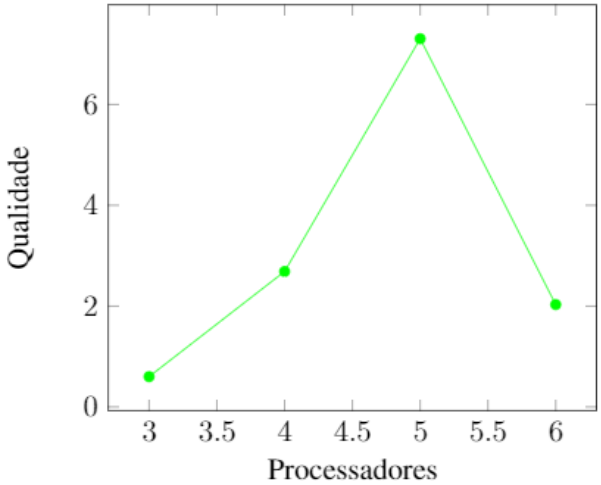

Gráfico 10: Qualidade de 500 clientes.

Após analisar os resultados das execuções pode-se concluir que a solução do problema utilizando MPI obteve um desempenho relativamente bom. A seguir temos uma análise métrica por métrica. Nessa análise foi desconsiderado a execução com dois processadores, pois os resultados divergiram do restante dos casos de teste.

- Speedup: Analisando os resultados exibidos pelas tabelas podemos ver que o tempo de execução da solução paralelo utilizando 2 processadores é bem menor quando comparado com mais processadores. Logo, o *speedup* alcançou um valor altíssimo. Observando os gráficos que exibem a evolução do *speedup* podemos ver que o *speedup* progride conforme vamos aumentando o número de processadores para solucionar o problema. Porém, com 6 processadores podemos ver que há uma queda no desempenho na maioria dos casos, somente com o número de clientes a serem atendidos igual a 100 sempre houve uma melhora do desempenho.
- Eficiência: Estudando de um modo geral todos os casos podemos concluir que a solução paralela obteve mais eficiência com 4 processadores. Há casos em que com 4 e 5 processadores pode-se alcançar o valor igual a um, como os casos em que o valor do total de clientes era igual a 400 e a 500. Já com 500 clientes ao aumentar mais um processador, passando de 5 para 6 a eficiência sofreu uma queda, mas ela continuou próximo a um, valendo 0.89601. Portanto, de um modo geral a eficiência foi evoluindo conforme aumentou-se o número de processadores e ao atingir 4 ou 5 processadores, dependendo do caso, houve uma queda na eficiência.
- Redundância: nessa métrica não conseguimos um valor tão exato, pois os computadores utilizados para a execução dos algoritmos não possuíam a ferramenta Perf, necessária para calcular o número de instruções em cada execução. Dessa forma, transformamos o algoritmo implementado na linguagem de programação C para a linguagem de montagem, Assembly, conseguindo saber o número de instruções que o programa executaria.
- Utilização: na maioria dos casos a solução que utilizou 5 processadores foram as que mais aproveitaram da capacidade computacional. Apenas no caso em que o número de clientes foi igual a 100 a solução que utilizou 4 processadores conseguiu aproveitar melhor a capacidade computacional. De um modo geral, as soluções que utilizam 4 processadores tiveram um melhor aproveitamento computacional do que as que utilizaram 6 processadores.
- Qualidade: Em todos os casos de teste a solução com 5 processadores foi a que obteve melhor qualidade, tento então o maior grau de importância de se utilizar programação paralela na solução do problema. Observamos também que com 4 processadores pode-se atingir uma boa qualidade, já com 6 processadores houve uma redução na qualidade e na maioria dos casos essa redução foi significativa.

6. Conclusão

Um programa concorrente é muito mais complexo do que um código feito com programação sequencial, assim a tentativa de paralelizar um algoritmo muitas vezes é mais simples na teoria do que na prática. Além de todos os erros comuns de um programa sequencial, existe uma variedade de erros gerados devido às interações entre os processos executando simultaneamente e a utilização de dados compartilhados.

Apesar disso, ao obter um código com paralelismo é possível acelerar a execução do programa e chegar a melhores resultados. Por outro lado, a qualidade desses resultados, bem como o *speedup* e sua eficiência, não estão apenas relacionados em criar

e executar um algoritmo com vários processos. Uma boa concepção de abordagem e a modelagem do problema, além de um bom código desenvolvido pelo programador, podem impactar positivamente nesses resultados finais.

Em qualquer modelagem paralela a maior preocupação é na sincronização dos processos e na atualização dos valores das variáveis. Não sendo diferente nesse trabalho, o maior desafio foi a sincronização dos caixas de atendimento, de forma que um novo cliente só poderia ser inserido no caixa que tivesse a menor fila em um dado momento e caso o caixa não possuísse mais nenhum cliente na fila ele teria que atender o cliente do caixa que estivesse com a maior da fila no momento.

Foram realizados diversos casos de teste para a modelagem paralelizada. Do modo que, pudesse ser realizado um estudo sobre o desempenho da modelagem e como ela se comporta com diferentes parâmetros de entrada. Logo, pode-se concluir que a solução elaborada nesse trabalho obteve um melhor desempenho quando foram utilizados cinco processadores na execução.

7. Referência Bibliográficas

GALANTE, G. **Curso de MPI.** Disponível em: <http://www.inf.unioeste.br/~guilherme/tsc/cursompi>. Acesso em: 30 jan. 2017.

ROCHA, Ricardo. **Programação Paralela e Distribuída.** Disponível em: <https://www.dcc.fc.up.pt/~ricroc/aulas/0708/ppd/apontamentos/metricas.pdf>. Acesso em: 30 jan. 2017.

Tanenbaum, A. S. Modern Operating Systems. Prentice Hall, 3ª Ed., 2008.

Tera HPC. **Introdução à programação paralela e mpi.** Disponível em: <http://terahpc.com.br/artigos/artigo-introducao-programacao-paralela-mpi.html>. Acesso em: 30 jan. 2017.