ALGORITMOS PARALELOS – SIMULADOR DE ELEVADOR

Juliana Araújo

Algoritmos e Estrutura de Dados III, Universidade Federal de São João del-Rei

1 INTRODUÇÃO

A computação paralela consiste, basicamente, na divisão de um problema em partes que podem ser resolvidas ao mesmo tempo, paralelamente. Essa forma de computação já vem sendo usada há certo tempo para a resolução de problemas de complexidade elevada e que precisam de uma quantidade considerável de poder de processamento.

Neste trabalho será realizado o paralelismo de um algoritmo que simula um elevador, adicionando vários elevadores ao prédio. Há dois tipos de paralelismo explorados nesse algoritmo: dados e controle.

2 ESPECIFICAÇÕES DO PROBLEMA

O paralelismo de thread consiste na execução em paralelo de independentes conjuntos de operações. Cada conjunto de operação é chamado de thread. Para ser eficiente, as threads devem ser o mais independente possível uma da outra, ou seja, devem existir poucos pontos de interação entre elas, pois cada ponto de interação exige mecanismos de sincronização para evitar que duas threads façam modificação na mesma porção de memória ao mesmo tempo, por exemplo.

O mecanismo de sincronização utilizado é o “Joining Threads”. O princípio básico é que apenas uma thread pode ter efetuado um lock em uma variável do tipo mutex em um dado instante. Portanto, nenhuma outra thread poderá efetuar o lock antes que a primeira thread o libere.

O algoritmo a ser paralelizado foi previamente desenvolvido, sendo uma estratégia que tem como objetivo simular um elevador, pegando os passageiros em seus respectivos andares e deixando-os em seu destino. Para que ocorresse as paralelizações, foram feitas as primeiras modificações nessa estratégia, adicionando mais elevadores ao prédio com o propósito de trabalharem de

maneira simultânea. A partir desse ponto, foi desenvolvido o paralelismo de dados e controle.

3 DESENVOLVIMENTO

Para a elaboração do programa foi utilizada a interface de manipulação de threads Pthreds, padronizada em 1995. Pthreads foi definido como um conjunto de tipos e procedimentos em C, encontrados na biblioteca *pthreads.h*.

Os dados primordiais para execução dos algoritmos são fornecidos por um programa auxiliar e dispostos em arquivos para que seja definido o ambiente do elevador, sendo eles:

- Número de andares do prédio;
- Capacidade do elevador;
- Dados dos passageiros: andar de chamada e andar de destino.

Cada passageiro terá seus dados em uma estrutura, e por fim será formada uma lista de estruturas contendo os dados de todos os passageiros.

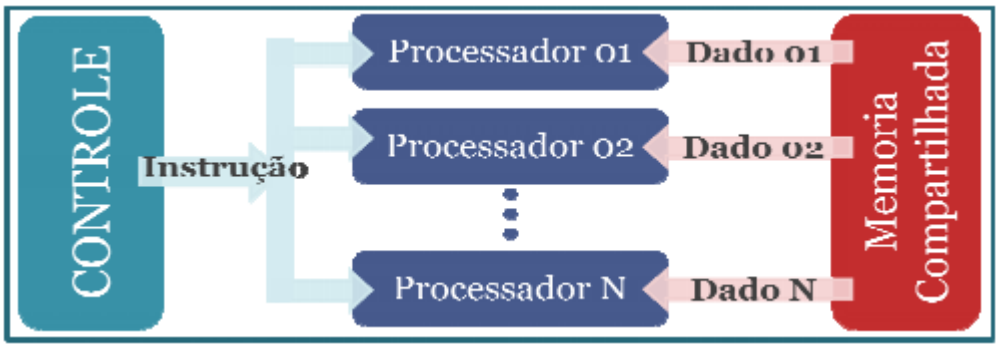
A estratégia desenvolvida busca a aproximação com a realidade de um elevador e consiste em subi-lo, pegando apenas os passageiros que irão subir e, em cada andar, verificar se alguém pode descer. O processo inverso é feito na descida, ou seja, o elevador desce, pegando somente quem irá descer, e sempre verificando se algum passageiro pode deixar o elevador naquele andar. Enquanto a lista não estiver vazia o elevador fará esse processo, já que quando alguém deixa o elevador é retirado da lista.

Os tempos são expressos em uma unidade de tempo Jepslon. Para percorrer a distância de um andar a outro, é necessário 1 Jepslon e para 1 ou mais passageiros entrarem ou saírem do elevador também é necessário 1 Jepslon.

Sempre que o elevador é chamado, o passageiro é sujeito a verificação de sua capacidade máxima. Além disso, há variáveis auxiliares para saber se o passageiro está dentro ou fora do elevador, e no primeiro caso, em qual elevador.

3.2 Paralelismo de Dados

O paralelismo de dados é a execução de uma mesma operação sobre vários elementos de dados ao mesmo tempo.



A implementação realizada consiste em múltiplos elevadores realizando a mesma tarefa no mesmo dado (lista de passageiros), simultaneamente. Cada elevador é responsável por transportar os passageiros de uma parte da lista. Para que houvesse essa definição, cada passageiro recebeu um número de identificação. O cálculo para a thread obter o seu intervalo na lista é dado pelas duas fórmulas abaixo:

- Primeiro passageiro: (id * n) / p;
- Ultimo passageiro: [(id + 1) * n) / p] – 1;

Onde *id* é o número da thread, *n* o total de passageiros e *p* o número total de threads. O intervalo então é calculado buscando ser bem distribuído e não causar sobrecarga em uma thread. Os passageiros que não estão dentro do intervalo em que a thread é responsável são ignorados por ela. Como a lista está ordenada de acordo com o número de chamada dos passageiros, esse intervalo representas andares seguidos do prédio.

A lista é compartilhada entre as threads, portanto o critério de parada das threads é encontrar a lista vazia. Devido a esse compartilhamento, se fez necessário a utilização do mutex no momento de retirada do passageiro da lista. Deste modo, apenas uma thread pode retirar um passageiro em determinado momento.

3.3 Paralelismo de Controle

Trata-se de dividir as operações em estágios ou segmentos, sendo a saída de um segmento a entrada para o próximo segmento.

No paralelismo de controle cada elevador é responsável por um andar, e sua função é apenas subir um andar e descer um andar. Portanto, o número de elevadores é o mesmo que o de andares. O passageiro então irá precisar de um ou mais elevadores para chegar ao seu destino, por exemplo, uma pessoa que está no andar 2 e deseja ir ao quatro, irá pegar um elevador para o andar 3, e depois outro para o 4. A saída de um elevador torna-se então a entrada do próximo. Para que isso seja feito, ao chegar um novo andar, verifica se aquele é o andar de destino do passageiro. Caso seja, esse é retirado, se não, seu andar de chamada é atualizado para o andar atual. Devido a essa atualização, as threads irão terminar apenas quando a lista de passageiro estiver vazia, e enquanto isso irão sempre procurar se existe passageiros no andar em que são responsáveis.

Em razão do compartilhamento da lista, assim como no paralelismo anterior, se fez necessário a utilização de mutex no momento de retirada dos passageiros.

4 ANÁLISE DE COMPLEXIDADE

4.1 Paralelismo de Dados

As threads responsáveis pelo paralelismo de dados irão subir até o último andar e descer até o primeiro. Isso é feito até que a lista de passageiros esteja vazia, por isso o primeiro laço de repetição. Dentro desse laço, há mais quatro, não aninhados: dois para o elevador subir e dois para ele descer, em que um laço é responsável por percorrer toda a lista em busca de alguém que possa deixar o elevador, indo sempre até seu final, e outro que buscará quem está naquele andar e pode entrar. Portando, a complexidade da função em que se executa as threads é O (n*(4n)) = O(n^2), sendo *n* o número total de passageiros.

Há também operações de custo O(n), como os laços que zeram as variáveis da lista, ou os responsáveis por criar a thread. Por fim, há operações O(1). Portanto, a complexidade desse algoritmo é $O(n^2)$, sendo *n* o número de passageiros da entrada.

4.2 Paralelismo de Controle

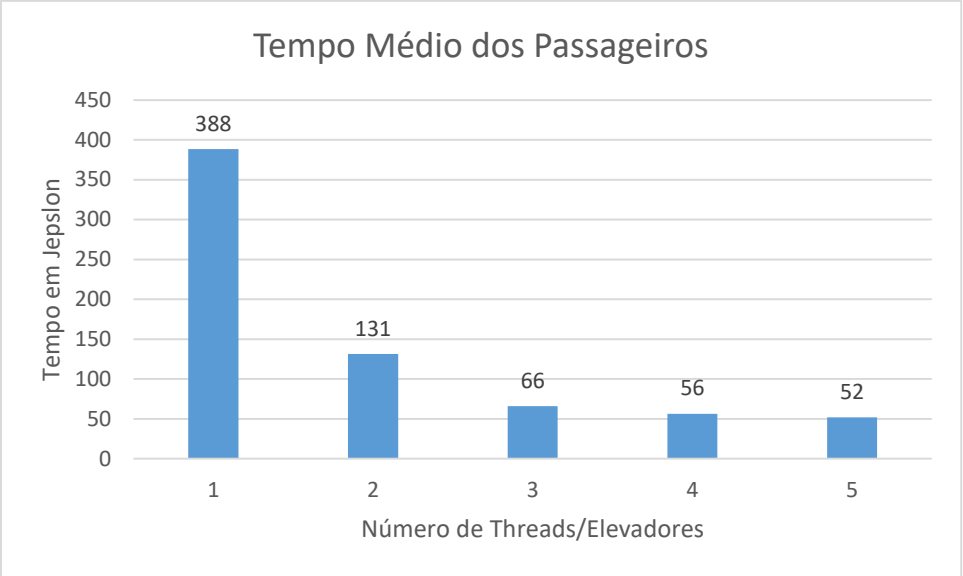
O Paralelismo de controle tem a mesma base detalhada anteriormente para o paralelismo de dados, acrescentando porém mais dois laços de repetição para encontrar quem é o primeiro passageiro que está no andar em que a thread é responsável, tanto no ato de subir o elevador e no de descer. Portanto, a função de complexidade é O(n*(6n)) que também resulta em $O(n^2)$, em que *n* é o número de passageiros da entrada.

5 ANÁLISE DE RESULTADOS

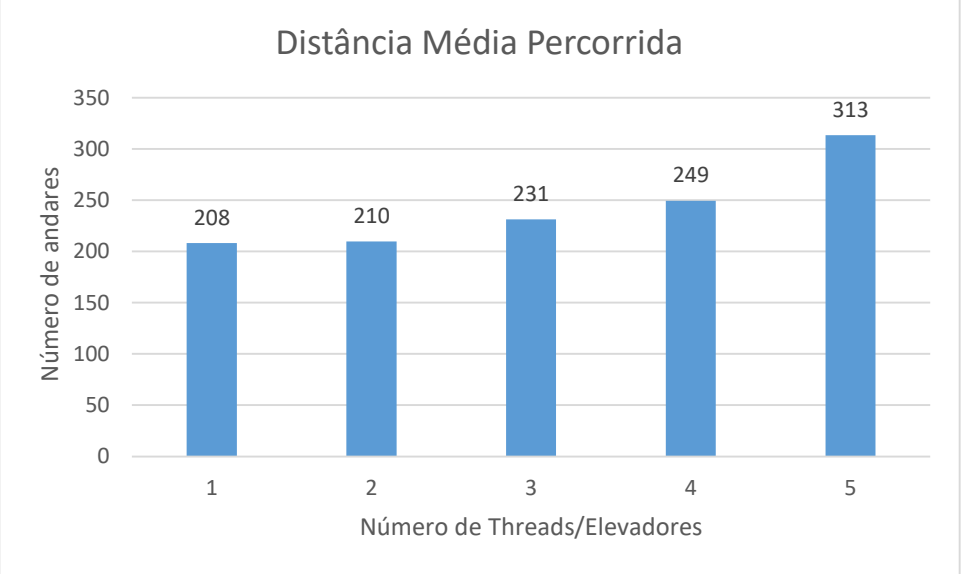
- Paralelismo de Dados

Os testes do algoritmo de paralelismo de dados foram executados variando o número de threads entre 1 e 5. Para cada thread, o número de pessoas variou entre 10, 100 e 1000. O número de andares foi mantido em 50 e a capacidade do elevador em 15 para todos os testes. Os dados dos gráficos apresentados foram obtidos realizando a média dos testes de mesmo número de threads.

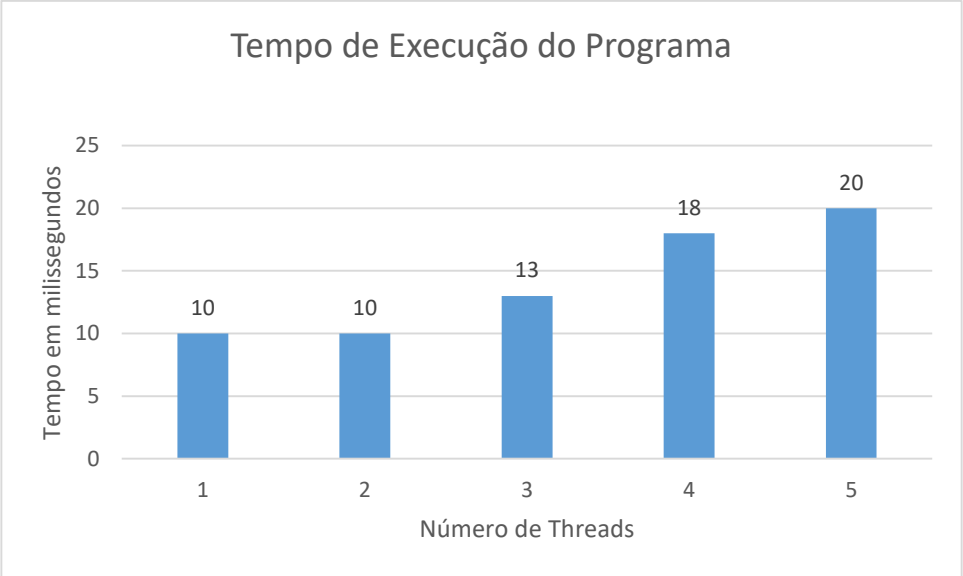
Abaixo, é possível observar a otimização do tempo médio de um passageiro para chegar ao seu destino, adicionando mais elevadores ao prédio e permitindo a divisão de passageiros entre eles. Esse tempo representa a soma do tempo que o passageiro passou fora do elevador mais o tempo que passou dentro.



Por sua vez, a distância média percorrida por elevador aumenta, já que o mesmo tem que percorrer todo o prédio para deixar os passageiros que estão no seu intervalo, e depois voltar a este intervalo ignorando demais passageiros.



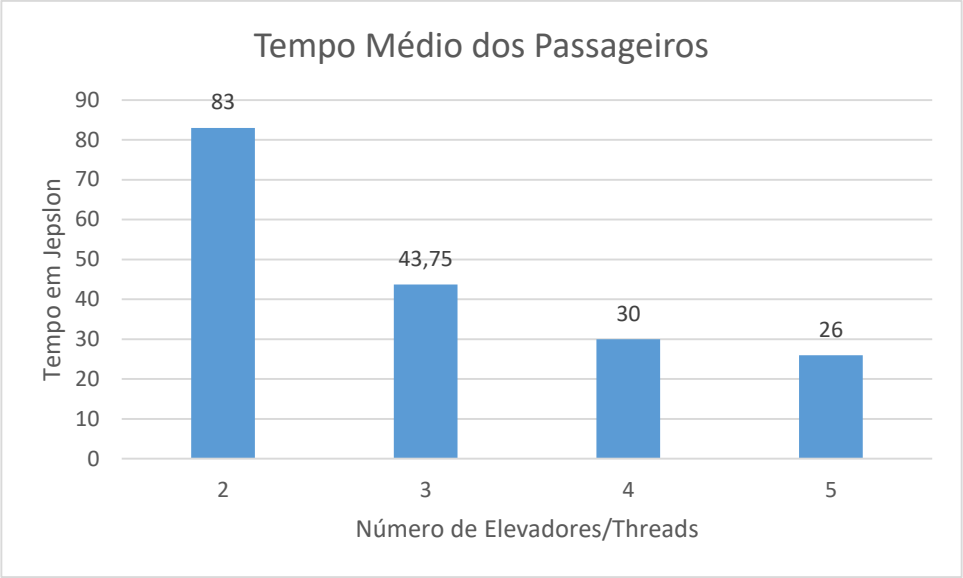
O mesmo acontece no tempo de execução do algoritmo, que aumenta à medida que as threads também aumentam.



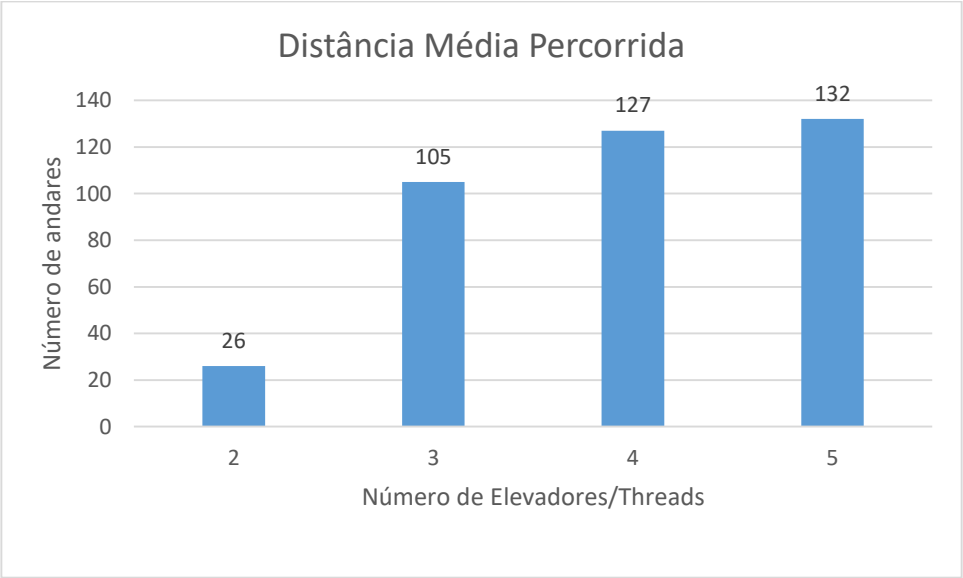
- Paralelismo de Controle

Para efetuar os testes, variou-se o número de andares entre 2 e 5, que também corresponde ao número de threads. Para cada número de thread, foram realizados testes em que o número de passageiros varia entre 10, 100 e 1000 pessoas. A capacidade do elevador foi mantida em 15 pessoas para todos eles. Por fim, foi realizada a média entre os testes de mesmo número de thread. Os resultados estão dispostos nos gráficos a seguir.

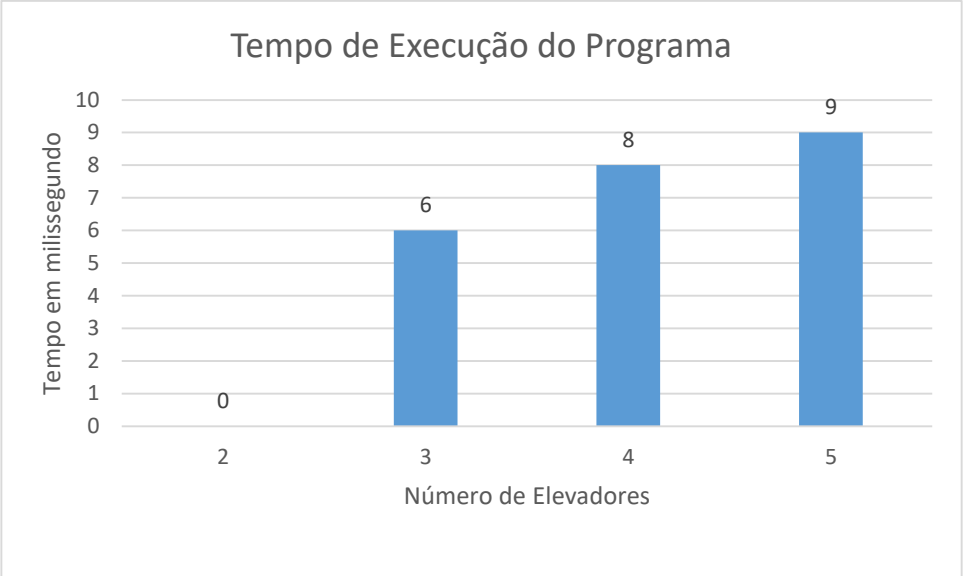
O gráfico abaixo representa o tempo médio de um passageiro para chegar ao seu destino. É possível observar que o passageiro chega de maneira mais rápida ao seu destino ao adicionar mais elevadores no prédio, mesmo que o número de andares também aumente.



Já no gráfico seguinte, percebe-se o aumento da distância média percorrida por um elevador, devido ao aumento de andares que ocorre proporcionalmente ao aumento do número de threads.

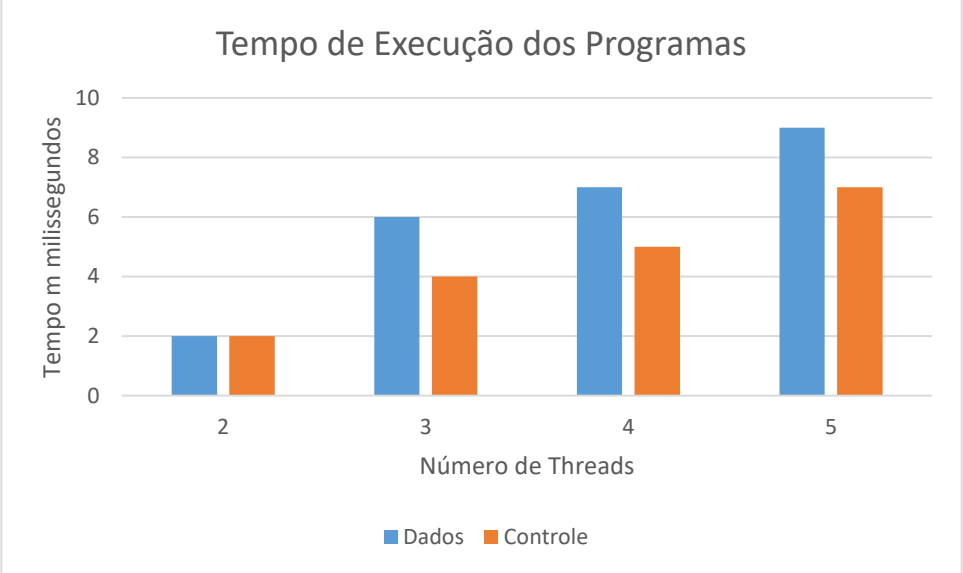


Em razão da estratégia de paralelismo de controle adotada, não é possível realizar testes que contenham apenas uma thread, simulando um programa sequencial, pois não é possível trabalhar com um prédio de apenas um andar. O gráfico abaixo contém o tempo de execução do programa em milissegundos, e pode ser observado seu aumento de acordo com o aumento do número de threads, já que assim aumenta-se também o número de andares e serão necessários mais processos até que o passageiro chegue ao seu destino.



- Comparação de tempos

Os algoritmos de paralelismo possuíam entradas diferentes entre sim, pois cada um possui suas especificidades. Porém, a fim de comparação, um teste com as mesmas entradas foi realizado. A entrada segue os padrões do paralelismo de controle. Os tempos de execução dos dois algoritmos foram muito próximos, e uma mesma execução obteve resultados que variavam muito entre si. A média obtida está no gráfico abaixo.



CONCLUSÃO

Um programa escrito apropriadamente pode dividir uma tarefa em partes e solucioná-las simultaneamente, utilizando mais de um núcleo do processador,

otimizando seus resultados. Sem o paralelismo, não seria possível ter mais de um elevador no prédio se movimentando simultaneamente, e realizar todas as comparações de resultados

REFERENCIAS

Explorando o paralelismo de dados e de thread para atingir a eficiência energética do ponto de vista do desenvolvedor de software. Endereço: “http://www.lbd.dcc.ufmg.br/colecoes/wperformance/2010/002.pdf”.