Insper
Engenharia da Computação - SuperComputação

Luís Filipe Sanchez Carrete

Projeto
Comparações de Desempenho

São Paulo
2022

Introdução

Nós como seres humanos, não somos o animal mais veloz, nem o com melhor visão, nem o animal mais forte, mas nós sabemos criar e utilizar ferramentas muito bem. Sempre que surge algum problema nós criamos ferramentas para resolvê-lo. Precisávamos levar um homem à lua, criamos o foguete, precisamos lavar louça automaticamente, criamos a máquina de lavar, precisamos resolver problemas muito complexos, criamos a supercomputação. A supercomputação é simplesmente isso, técnicas e ferramentas que utilizamos para resolver problemas que antes eram intratáveis, ou seja, precisava de um tempo muito grande para resolvê-los.

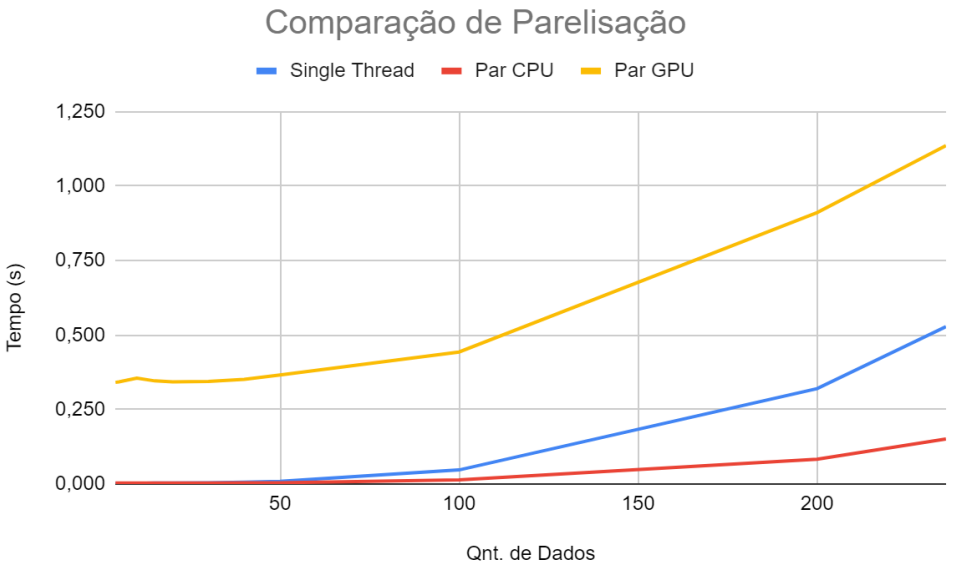
Um dos problemas mais populares é o problema do caixeiro viajante, onde, dado as cidades que o caixeiro deve ir, devemos encontrar a rota que gera a menor distância total. Esse problema com poucos pontos é fácil de resolver, mas quando aumentamos o número de cidades a complexidade do problema aumenta exponencialmente e cresce extremamente rápido. Por exemplo se tivermos 4 cidades só existem 24 possíveis caminhos que o caixeiro pode escolher, quando vemos 11 cidades já pegamos um número muito maior, 39.916.800, e se adicionarmos apenas mais uma, o número de permutações vira 479.001.600, ou seja temos um aumento absurdo. Aí entra a supercomputação, a supercomputação nos permite dividir este problema em várias partes para resolvê-las paralelamente assim cortando o tempo para analisar todas as permutações. Mas para resolver este problema de forma paralelizada temos duas opções, ou podemos paralelizar na GPU ou diretamente na CPU, isto é o que vamos analisar neste roteiro. Para todas as informações abaixo foram utilizados foram testes com 4, 5, 10, 15, 20, 30, 40, 50, 100, 200 e 236 para conseguirmos visualizar o crescimento do tempo de resolução.

Se você pudesse escolher um método para resolver este problema, qual seria?

Para resolver o problema do caixeiro viajante eu escolheria para resolvê-lo utilizando GPU, caso tenha acesso. O problema da GPU é o envio de dados da CPU para a GPU e vice versa, esse acesso gera um overhead muito grande, o jeito correto de utilizar a GPU é enviando todos os dados de uma vez, fazer todos os tratamentos diretamente na GPU, e retornando para a CPU. Como o problema do caixeiro viajante pode ser feito sem gerar dependências podemos resolver o nosso problema com a GPU. O jeito que deve ser feito é o seguinte, deveríamos enviar a lista com o número de cidades, utilizando iteradores devemos gerar todas as permutações possíveis de forma paralela e em seguida calcular a distância de todos os pontos em uma só tacada. Obtendo a melhor solução global.

Abaixo está o gráfico da performance de comparação dos diferentes métodos. Podemos ver que a GPU foi a que demorou mais tempo. Existem duas possibilidades para este resultado. A primeira possibilidade é que os dados ainda não são grandes o suficiente para vermos um ganho tão bom. A GPU, como foi mencionado anteriormente, tem um overhead muito grande quando enviamos e recebemos dados dela, então quando temos poucos dados o programa passa mais tempo enviando e recebendo dados do que realmente

fazendo cálculo por isso é esperado que com um número muito baixo de cidades a GPU seja mais lenta. Outra possibilidade é a implementação do código. Quando escrevemos código para paralelizar na GPU precisamos de uma mentalidade específica, muito diferente de programação single thread, ou seja, o código acaba ficando com outra lógica. Para passar de single thread para paralelização na CPU é extremamente simples pois não precisamos lidar com o envio e o recebimento de dados, e a implementação não é muito distante de um código single thread, por isso a comparação da single thread com a CPU é muito mais justa do que com a GPU, pois a GPU tem uma lógica bem distinta, e específica. Isso pode ser um motivo para o resultado abaixo, simplesmente fazemos mais iterações e outras mudanças no código de GPU.

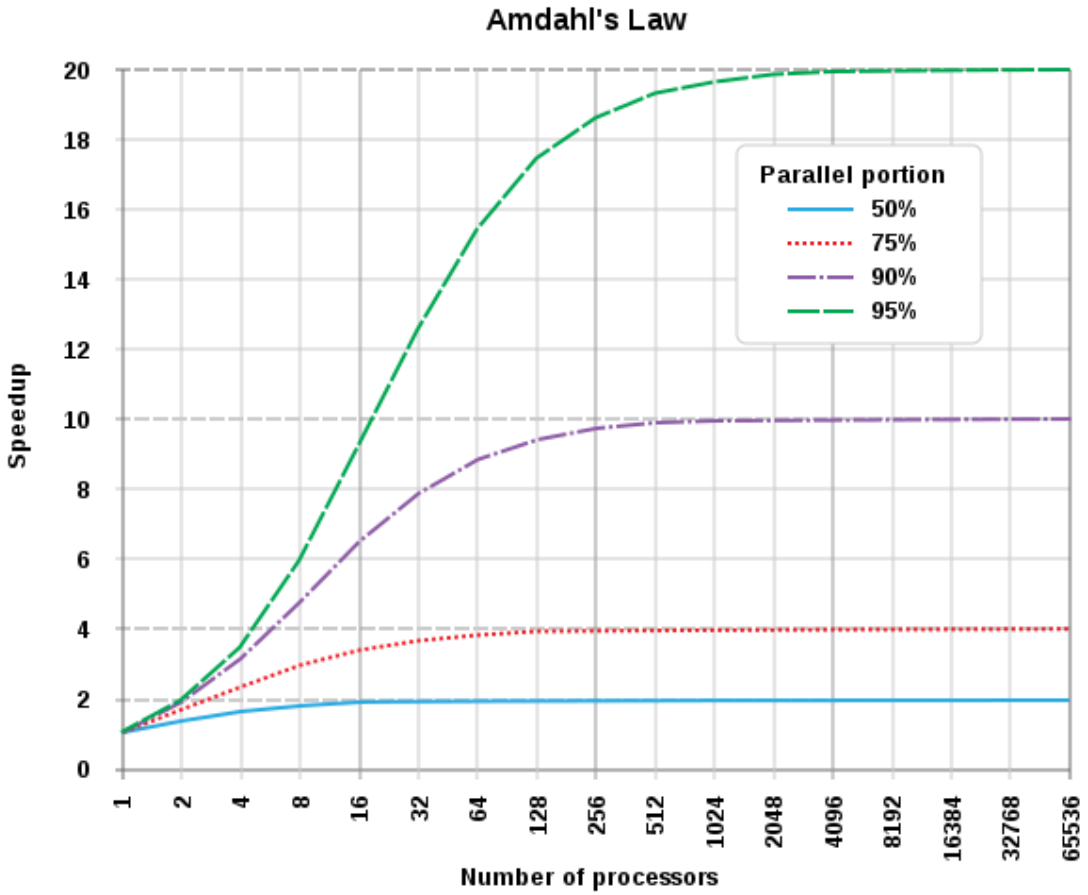


Apesar de nosso resultado não mostrar que a GPU é a melhor para este problema, acredito que se precisar fazer um código do zero para resolver este problema um código paralelizado na GPU faz mais sentido e tem muito mais potencial se for criado com a mentalidade correta. Mas caso já tenha um código single thread para resolver este problema é muito mais simples paralelizar-lo na CPU.

Fizemos implementações paralelas da busca local. Valeria a pena gastar dinheiro comprando uma CPU com mais cores ou uma GPU potente?

Seguindo a mesma linha de raciocínio da questão acima eu acredito que vale a pena investir em uma GPU potente, pois este problema consegue ser resolvido na GPU de forma eficiente e a GPU utilizada corretamente é muito mais rápida que a CPU. A GPU consegue fazer um cálculo em múltiplos dados simultaneamente, seu acesso de memória é um cache extremamente rápido, quanto mais dados conseguimos tratar ao mesmo tempo mais rápido fica o código, por isso optaria por investir em GPU. Já a CPU pode ser bom para outros problemas, problemas com vários acessos de memórias, neste caso eu recomendaria investir em CPU. Apesar da sugestão de comprar um hardware bom, é também preciso considerar o algoritmo, pois podemos gastar 1 milhão em CPUs e GPUs mas caso o algoritmo não for

paralelizável o resultado não vai melhorar. Para isso devemos considerar a lei de Amdahl, o gráfico pode ser visto abaixo. A lei de Amdahl nos diz o limite de número de processos para um algoritmo paralelizado. Podemos ver que não importa o código, todos os códigos tem um certo limite para número de processadores, onde o crescimento não é mais significante. Logo eu recomendo gastar dinheiro em GPU, mas deve sem dúvida alguma também considerar a lei de Amdahl para não gastar dinheiro em mais processador sem nenhum ganho.



Vale a pena esperar pelo resultado da busca exaustiva?

Esta é uma pergunta completamente dependente do contexto. A pergunta correta a se fazer é, o quão preciso deve ser minha resposta. Por exemplo, se estamos calculando o percurso de uma viagem de família utilizando o algoritmo do caixeiro viajante, não precisamos do melhor percurso possível, pois caso demore um pouco mais do que o melhor caminho não vai gerar nenhum problema. Agora caso o problema for mais sério como, o percurso de um astronauta em Marte, onde ele tem combustível limitado, oxigênio limitado e outras limitações, aí sim seria necessário esperar pelo resultado da busca exaustiva pois um resultado não ótimo poderia criar riscos para a vida do astronauta. Concluindo, dependendo da situação, como algo que pode depender de vida ou morte, ou caso o impacto custar muito dinheiro, vale a pena esperar o resultado da busca exaustiva.

Conclusão

Em conclusão, fica evidente o poder da supercomputação, e seu lugar em nossa sociedade. Sem supercomputação não conseguimos, trabalhar problemas muito complexos em tempo hábil, não conseguimos enviar um homem à lua, não conseguimos simular simulações físicas, não conseguimos processar muita coisa que hoje é necessário. Em relação a comparação da CPU com GPU, fica claro que cada paralelização tem seu lugar, as vezes faz mais sentido paralelizar na CPU as vezes na GPU, é importante entender bem seu problema e pensar bem em seu algoritmo para de fato otimizar seu código. Nem sempre o primeiro caminho será o certo, mas quando conseguir, conseguirá as melhores soluções no menor tempo possível.