INSPER - INSTITUTO DE ENSINO E PESQUISA
ENGENHARIA DE COMPUTAÇÃO
GRADUAÇÃO - SUPERCOMPUTAÇÃO

ENRICO FRANCESCO DAMIANI

RELATÓRIO DE DESEMPENHO

São Paulo
2022

Sumário

Introdução
Este relatório posso como objetivo a comparação de desempenho de diferentes implementação para códigos de busca local do *Travelling salesman problem* (TSP). Mais especificamente, são comparadas a implementação rodando em um core da CPU, em múltiplos cores da CPU e por fim na GPU, onde todos os códigos possuem a mesma lógica, entretanto adequados ao respectivo local de implementação. Ao fim, são respondidas perguntas, que visam a análise deste.

Metodologia
Para a realização dos teste forma concebidos três códigos, que estão disponíveis em: https://github.com/enricofd/TSP.git, na pasta busca-local. Todos os códigos foram executados por meio do Google Colab, em sua versão Pro, sendo alocados 80 GB de memória RAM, um processador Intel Xenon e uma placa de vídeo Nvidia A-100 40GB, conforme imagens abaixo:

```
[ ]   1 !cat /proc/meminfo

MemTotal:       87538764 kB
MemFree:        83337436 kB
MemAvailable:   86105296 kB
Buffers:          338236 kB
Cached:          3077952 kB
SwapCached:            0 kB
Active:           628884 kB
Inactive:        3215536 kB
Active(anon):        872 kB
Inactive(anon):   379444 kB
Active(file):     628012 kB
Inactive(file):  2836092 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
```

Imagem retirada do Google Colab, mostrando a quantidade de memória RAM disponível na máquina utilizada

```
 1 !cat /proc/cpuinfo

processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping        : 7
microcode       : 0x1
cpu MHz         : 2200.204
cache size      : 39424 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clfl
bugs            : spectre_v1 spectre_v2 spec_store_bypass mds swapgs taa mmio_stale_data retbl
bogomips        : 4400.40
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

Imagem retirada do Google Colab, mostrando a CPU utilizada pela máquina em questão

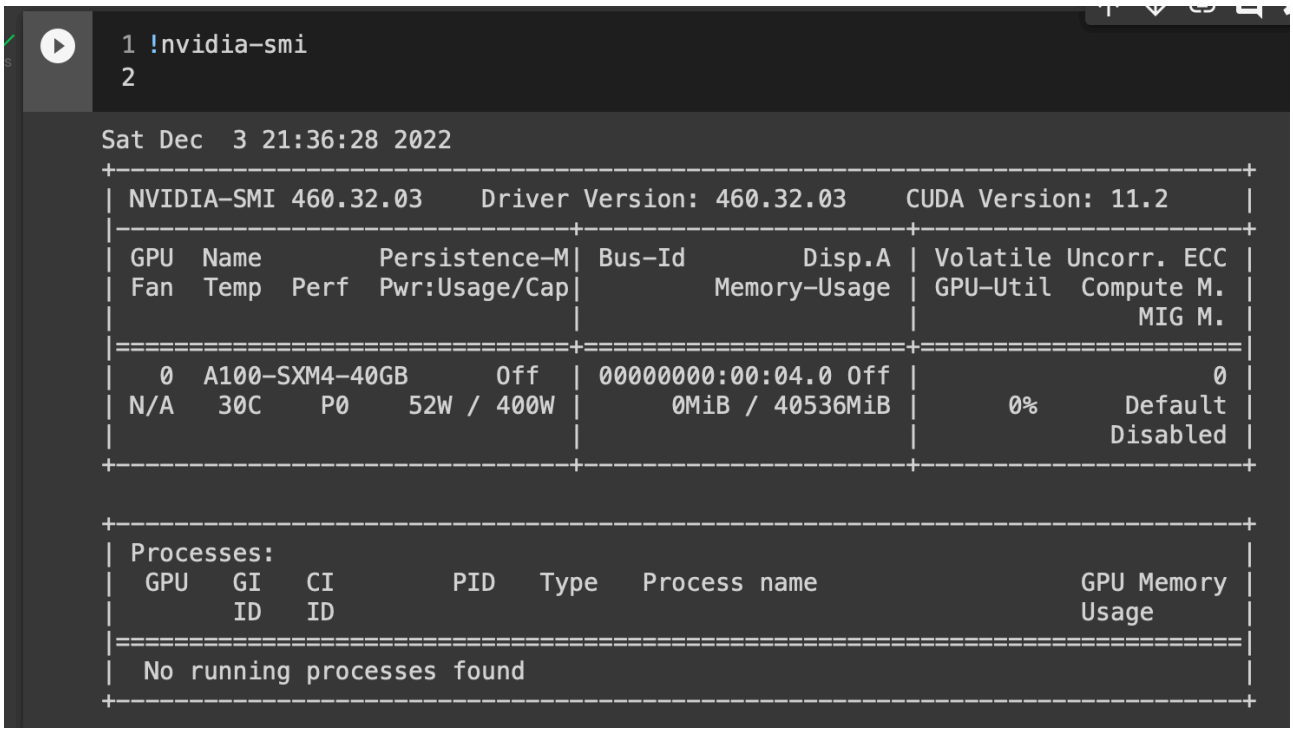


Imagem retirada do Google Colab, mostrando a placa de vídeo utilizada pela máquina

Os códigos concebidos são, main.cpp, mainpar.cpp e maingpu.cu, sendo respectivamente os códigos de, CPU (1 core), CPU (multicore) e GPU. O TSP constitui um clássico problema, em que se tenta otimizar o custo (distância percorrida pelo *salesman*) ao percorrer múltiplos pontos. Em outras palavras, como fazer o *salesman* percorrer todos os pontos, percorrendo a menor distância possível. Desta forma, vale também mencionar que o código paralelizado na CPU utiliza o OpenMP e o em GPU o Thrust. Para tanto, fora elaborado um outro código, input_generator.py, que cria arquivos contendo, 10, 50, 100 e 150 pontos, a serem explorados. Assim, utilizando estes arquivos como entrada dos código de busca local, calcula-se o tempo que cada um levou para alcançar a melhor rota (mínimo local). Vale, além disso, mencionar a lógica utilizada pelos códigos de busca local. Ela parte do princípio de que, para a quantidade de pontos presentes, 10 vezes a quantidade de cidades combinações aleatórias (ordenação dos pontos) devem ser calculadas ou seja, para o caso de 10 pontos, 100 combinações aleatórias serão geradas. A partir disto, para cada uma das combinações, serão efetuadas trocas de elementos, visando a diminuição das distância percorrida. Estas trocas se dão de um elemento, pelo seguinte, e se a nova distância form menor a troca é mantida, caso contrário, revertida. Isto é realizado para todos os pontos de uma dada combinação. Assim, por meio do comando time, obteve-se o tempo de execução para cada combinação de implantação e entrada.

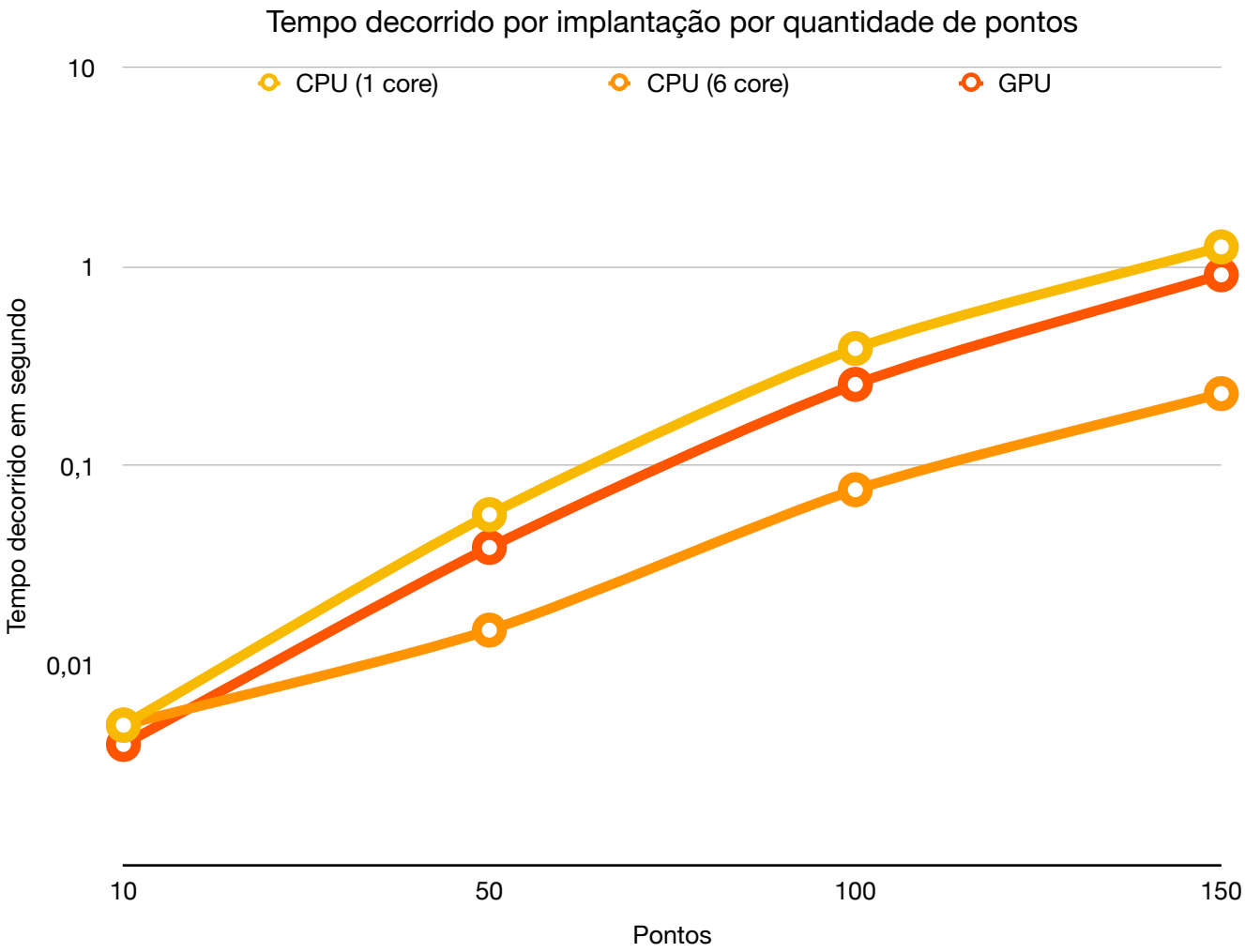
Resultados
Após rodar cada uma das implementações, com cada um dos arquivos gerados, os seguintes resultados abaixo foram obtidos, e se encontram em formato de tabela e gráfico:

Tempo decorrido por implantação por quantidade de pontos

Pontos	CPU (1 core)	CPU (6 core)	GPU
10	0,005	0,005	0,004
50	0,057	0,015	0,039
100	0,389	0,076	0,257

Pontos	CPU (1 core)	CPU (6 core)	GPU
150	1,259	0,231	0,912

A tabela acima mostra a relação entre pontos percorridos com a implementação utilizada.Os tempos acima se encontram em segundos



A gráfico acima mostra a relação entre pontos percorridos com a implementação utilizada.Os tempos acima se encontram em segundos. O gráfico é apresentado em escala logarítmica

Análise de resultados
A análise dos resultados possui como objetivo além de discorrer sobre os dados obtidos, responder a três perguntas. Primeiramente, quanto a análise, os resultados chamam a atenção quanto o tipo decorrido no experimento, por parte da GPU. Era esperado, que o código rodado na GPU fosse significativamente mais rápido que o da CPU paralelizado e ainda mais rápido que o da CPU (1 core). Assim, vale pensar em explicações para justificar tal fenômeno. Dentre as possíveis explicações, três aparecem aparentam ser mais plausíveis. São elas:

1. O código desenvolvido apresenta características não paralelizáveis consideravelmente grandes;
2. A quantidade de dados não é significativamente grande para ver ganhos da CPU;

3. A execução do código por parte do Google Colab, na GPU, é uma “caixa preta".

O primeiro ponto, inicialmente não aparenta fazer sentido. Isso se deve pois desde a criação do código, ele fora feito para não possuir dependências, otimizado o tempo de execução. Entretanto, uma parte importante dele que não fora paralelizada é a criação das combinações de pontos, que ainda ocorre em 1 core da CPU. Assim, esta parte do código pode estar, e muito, aumentando o tempo decorrido para a execução do mesmo. O segundo ponto, por sua vez, é passível de grande dúvida. Isso se da pois com a CPU (6 core) já foi possível ver o ganho apresentado. Entretanto, tenta-se perceber um cenário o qual o processo de executar o código na GPU ainda é muito custoso quando comparado com o tempo ganho ao processar os dados na mesa. O terceiro, por fim, busca não culpar o hardware empregado (indiscutivelmente, é mais do que suficiente para executar o código), mas sim a “caixa preta” por trás do mesmo, vide que não é sabido como se dá realmente a distribuição das tarefas nos servidores do Google.
Por fim, o que é mais plausível, é uma combinação de todos os fatores acima mencionados, culminando nos resultados obtidos. Assim, vamos as perguntas propostas.

Se você pudesse escolher um método para resolver este problema, qual seria?
Acredito que o método escolhido para resolver o problema varia muito com as especificações do mesmo. Mais especificamente, o tempo máximo para a obtenção de uma resposta, o orçamento disponível e a quantidade de pontos a serem processados. Por exemplo, para uma pequena quantidade de dados, um vasto tempo disponível, e um orçamento consideravelmente baixo, a melhor solução pode ser o processamento na CPU em 1 ou, caso necessário, 6 core. E, por sua vez, o processamento de grandes quantidades de pontos em um tempo rápido, possuindo o orçamento, recai na utilização de GPU. Assim, para concluir, depende do cenário.

Fizemos implementações paralelas da busca local. Valeria a pena gastar dinheiro comprando uma CPU com mais cores ou uma GPU potente?
Novamente, depende do cenário. Se pensarmos em uma grande quantidade de dados, a GPU seria vantajosa, entretanto, se a quantidade não for tão grande, talvez a CPU seja mais. Entretanto, de forma geral, é indiscutível que a escalabilidade da GPU é impar. Portanto, consideraria a compra da GPU.

Vale a pena esperar pelo resultado da busca exaustiva?
Mais uma vez, depende do resultado esperado. Se apenas uma “boa solução”for suficiente, a busca local é mais do que suficiente, entretanto, se a melhor solução possível for necessária, resta-se a busca global. O que vale ser mencionado, entretanto, é que para uma quantidade pequena de dados, muitas vezes, o resultado da busca local é similar ao da busca global ou consideravelmente similar. Como mencionado anteriormente, o principal ponto é o resultado esperado, assim como o tempo disponível para o mesmo, vide que a busca exaustiva é muito mais custosa. Isto se dá pois ela passa por todas as possibilidades possíveis para o problema, enquanto a global passa somente por alguns, explorando seus vizinhos.

Conclusão
Assim, conclui-se este relatório mencionando que os resultados obtidos não forma os esperados, vide que a implementação mais rápida se deu pela CPU (6 core) e não pela GPU. Além disso, quanto as perguntas elencadas, para grande parte delas, a resposta acaba dependendo do que é exigido, quanto a tempo, orçamento e pontos a serem calculados, onde as demandas mais exigentes recaem em uma implementação com GPU e as menos exigentes, uma CPU.