In [1]:

```
%matplotlib inline
import os
import subprocess

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Projeto 3

Eduardo Tirta

Introdução

Neste projeto iremos trabalhar em uma área chamada Otimizacão discreta, que estuda problemas de otimizacão em que as variáveis correspondem a uma sequência de escolhas e que tem uma característica especial: a solucão ótima só pode ser encontrada se enumerarmos todas as escolhas possíveis, Ou seja: não existem algoritmos eficientes para sua resolucão. Isto significa que todo algoritmo para sua solucão é O(2n) ou pior. Inclusive, ao recebermos uma solucão só conseguimos saber se ela é a melhor se olhando para todas as outras de novo! Claramente, estes problemas são interessantes para computacão paralela: podemos diminuir seu consumo de tempo consideravelmente se realizarmos testes em paralelo.

Um problema muito popular na área de logística é o Caixeiro Viajante: Um vendedor possui uma lista de empresas que ele deverá visitar em um certo dia. Não existe uma ordem fixa: desde que todos sejam visitados seu objetivo do dia está cumprido. Interessado em passar o maior tempo possível nos clientes ele precisa encontrar a sequência de visitas que resulta no menor caminho.

Vamos assumir que: • o nosso caixeiro usa Waze e já sabe qual é o caminho com a menor distância entre dois pontos; • ele começa seu trajeto na empresa 0 . Ou seja, basta ele encontrar um trajeto que passe por todas as outras e volte a empresa 0 ; • ele não pode passar duas vezes na mesma empresa. Ou seja, a saída é uma permutação de 0 ... (N-1)

Nosso trabalho será encontrar esse caminho e fornecê-lo ao vendedor. Note que esta mesma formulação pode ser usada (ou adaptada) para empresas de entregas com caminhões. Finalmente, os objetivos deste projeto são

1. Implementar um aprimoramento do codigo do projeto 2, enviando os dados para GPU

Desenvolvimento e Otimização

O projeto consiste na otimização do problema do caixeiro viajante, desenvolvido no projeto 2 desta disciplina.

Como o processamento de uma GPU é maior, é aceitável gerar soluções randomicas, possuindo um resultado para a nossa solução. Mas para garantir que o código gere uma solução ótima, é feito um desenvolvimento de uma solução 2-opt. Essa solução é implementada verificando se existe uma solução melhor no momento que invertemos dois pontos, evitando cruzamentos.

Especificação do computador

Os testes são feitos em um computador hospetado na AWS do modelo p2.xlarge

Testes do projeto

O projeto possui 5 arquivos com o código do caxeiro viajante, além de 1 arquivo que gera numeros aleatorios representando os pontos que o viajante precisa passar.

- 2-opt-sol.cu
- Arquivo tsp-par.cpp, otimiza o arquivo tsp-seq.cpp em paralelo, assim, usamos o poder computacional para ganhar velocidade

`python3 gerador.py > [nome_do_arquivo_de_entrada]`

Depois é necessário digitar a quantidade de pontos que deseja gerar para o viajante passar

Além disso, o projeto possui um CMakeLists.txt que possibilita a compilação dos executáveis. São eles:

- 2-opt-sol
- tsp-par (paralelo)

Siga os comando abaixo para gerar os arquivos compiláveis:

```
mkdir build
cd build
cmake ..
make
```

Para rodar o programa, basta utilizar

`./[nome_do_executavel] < ../[nome_do_arquivo_de_entrada]`.

Resultados

Os testes foram realizados com entrada de diversos tamanhos, para verificar se o codigo estava certo, foi feito a comparacao com o arquivo tsp.py em tamanhos menores e testando uma certa quantidade de vezes para garantir que o resultado de entrada e de saida estavam condizentes com o arquivo referencia. O tempo de execucao eh medido usando chrono high resolution para a versao em paralelo na CPU e usado o cudaEvent_t.

Para tamanhos pequenos, o algoritmo mostra que ao rodar em GPU, o código já se mostra muito mais eficiente, estes valores apresentados abaixo com a entrada de 12, roda cerca de 300% mais rapido.

CPU - 2874 milissegundos

GPU - 11.92 milissegundos

```
[ec2-user@ip-172-31-42-14 build]$ ./tsp-par < ../in12
11533.81823 0
0 5 6 11 9 3 10 7 4 2 1 8
Demorou: 3676 ms[ec2-user@ip-172-31-42-14 build]$ ./2-opt-sol < ../in12
11533.8 0
6 11 9 3 10 7 4 2 1 8 0 5
11.9435
[ec2-user@ip-172-31-42-14 build]$
```

Para tamanhos grandes, como 125 pontos, o algoritmo mostra que rodar em GPU é a única solução, mesmo demorando um pouco, ainda é capaz de realizar algumas ações com tamanhos de entradas maiores, enquanto apenas com a CPU, o tempo é indeterminado

CPU - N/A

GPU - 5381.78 milissegundos

```
[ec2-user@ip-172-31-42-14 build]$ ./2-opt-sol < ../in125
117147 0
31 121 39 50 57 52 89 113 73 76 35 93 17 85 42 122 63 44 69 10 30 105 119 3 46 1
11 47 83 18 41 94 45 58 56 114 79 11 115 68 61 64 112 40 0 37 38 110 82 36 26 22
 118 49 65 29 13 109 74 116 97 6 100 90 71 106 24 98 9 78 7 80 12 53 75 117 88 9
6 1 102 92 120 2 19 99 23 55 4 48 34 77 25 87 33 27 59 101 20 124 66 91 43 62 32
 16 70 95 21 5 123 8 84 67 28 51 86 15 81 54 103 104 72 60 107 14 108
5345.68
```