**Relatório trabalho de curso**

**Gabriel Diniz Gisoldo RA: 22214007-1**

A solução proposta faz uso de 8 *threads* para processar paralelamente todos os números contidos na base da dados fornecida.

Foram realizados teste com vários números de *threads* para definir a melhor quantidade de *threads* para o processamento. Esses testes foram executados sobre 50 mil entradas da base de dados variando a quantidade de *threads* usadas.

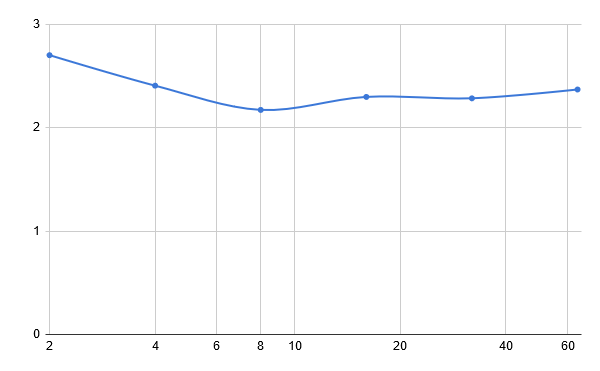


Figura Quantidade de threads X tempo de execução

Para cada número de *threads* a solução foi executada 100 vezes. Após as execuções foi tirada a média dos tempos para diminuir a influência de outros processos em background do sistema operacional. Como pode ser observado na Figura 1, o menor tempo de execução foi obtido usando 8 *threads* e por esse motivo os tempos para o processamento final foram feitos com o uso de 8 *threads*.

Após definir a configuração do número de *threads*, foi realizada a execução sobre toda a base de dados. Foram avaliados um total de 10.006.060 números em 382,09895 segundos. Dessa forma, cada número avaliado levou 0,0381867538 milissegundos em média na operação com 8 *threads*.

O modelo de processamento dos dados foi feito com uso de uma *Queue* do *Python3*, que permite o acesso por várias *threads* e já possui *locks* para inserção e remoção internamente. Dessa forma, todas as *threads* removem um item da *Queue* e fazem os cálculos dos dígitos verificadores desse número. Assim que os cálculos são terminados, o número é inserido em uma variável do tipo *list*, a qual é usada para gerar o arquivo de *output* quando todas as *threads* terminam de processar a *Queue*.

O código que faz o processamento dos números está no arquivo *orquestrador.py* e todo o *output* gerado após o processamento é salvo no arquivo *output.txt* ao final da execução.

O arquivo *funcoes.pyx* contém as funções que fazem o cálculo dos dígitos verificadores dos CPFs e CNPJs. O arquivo foi otimizado com o uso do *Cython* para otimizar o tempo de execução e deixá-lo mais próximo do tempo que o mesmo programa em C levaria para executar.

Todo o código fonte da solução está disponível em:

<https://github.com/huine/sistemas-distribuidos/tree/master/cpf-cnpj>

O vídeo de apresentação está disponível em: