Out[1]:

The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Supercomputação - Projeto Final - Rafael Vieira Rosenzvaig

O Problema

Este projeto tem como objetivo resolver o problema de escolha de projetos finais de engenharia dos alunos do Insper, utilizando técnicas de computação paralela.

Solução:

A solução encontrada, em um primeiro momento, foi a mais intutiva possível. No arquivo, busca_exaustiva_seq função escolhe_alunos() recebe a lista de preferências de cada aluno como entrada e recursivamente calcula a satisfação geral com todas as combinações possíveis. A combinação que possuir satisfação geral melhor será a escolhida e retornada pela função. Porém este método acaba tendo que realizar muitas operações desnecessárias para chegar ao resultado e também é pouco otimizado para entradas muito grandes. Para tentar melhorar isso, foram implementadas técnicas de programação de busca local e busca local na gpu, com versões também paralelizadas (na versao processada na cpu) para melhorar o tempo de processamento e chegar no mesmo resultado que a solução mais ingênua do problema.

Infraestrutura

Executáveis:

- busca_exasutiva_seq
- busca_exaustiva_par
- busca_local_seq
- busca_local_par
- busca_gpu
- backtrack.py
- cria_entrada.py

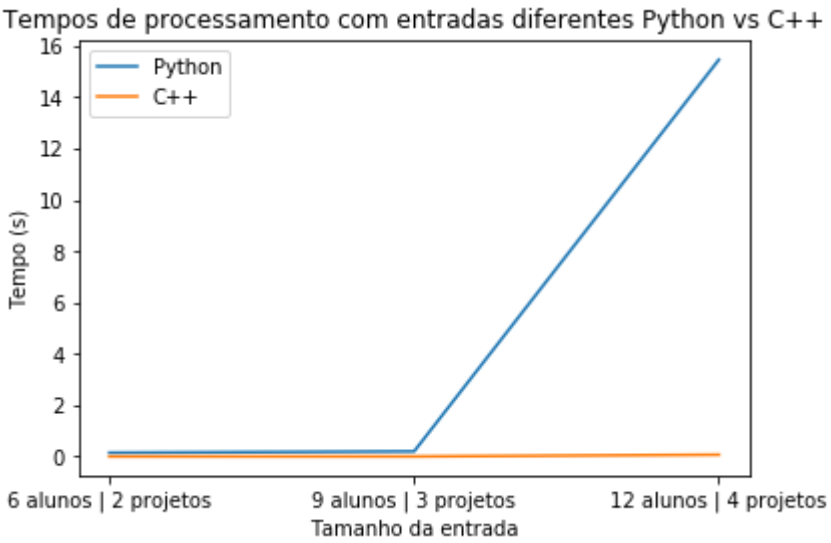
Arquivos:

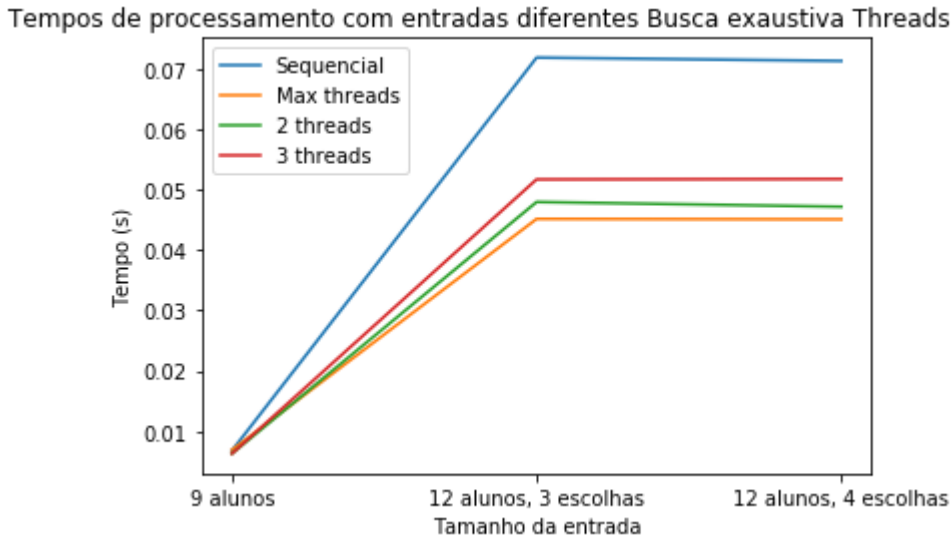
- backtrack.py
  - Este arquivo define as funções:
    - escolhe_alunos(), é a função que recursivamente calcula qual a melhor combinação possível entre a distribuição dos alunos, baseada na preferência de cada um.
    - main(), lê os parâmetros do arquivo de entrada e chama a função escolhe_alunos() com esses parâmetros.
- cria_entrada.py
  - Este arquivo cria uma entrada para o arquivo backtrack.py e/ou ingenua.cpp e dentro dela são definidos os parâmetros de número de alunos, número de projetos e número de escolhas que cada aluno pode fazer para ranquear sua preferência.
- CMakeLists.txt
  - Este arquivo contém os parâmetros de compilação para gerar o executável por meio do CMAKE.

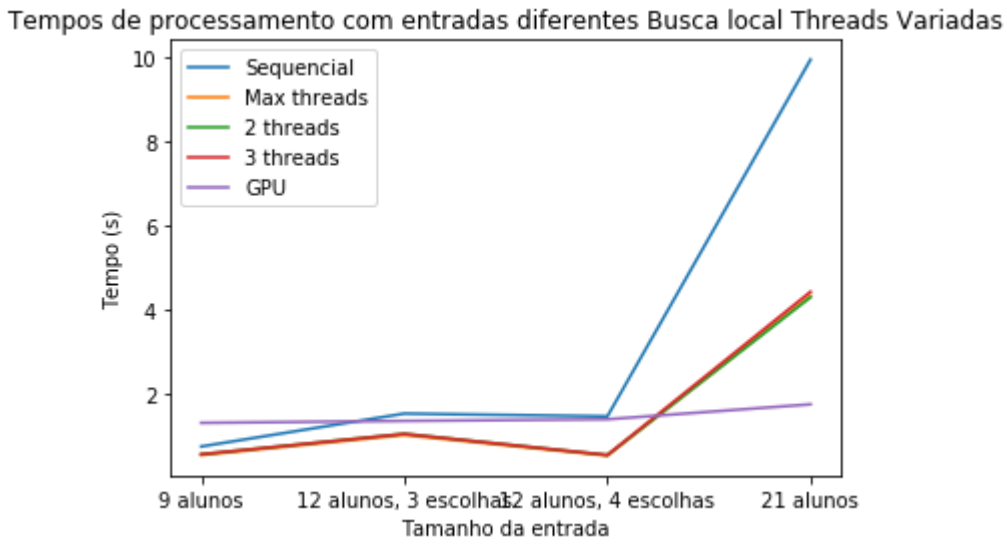
- busca_exaustiva_seq
  - Este arquivo é a tradução direta para C++ do arquivo backtrack.py, portanto as funções fazem as mesmas coisas.
- busca_exaustiva_par
  - Este arquivo é a versão paralelizada do busca_exaustiva_seq. Ele cria tasks paralelizadas para fazer a recursão que também é feita no método sequencial. Este arquivo também possúi as versões com limitação do número de threads: busca_exaustiva_par_2t, busca_exaustiva_par_4t, busca_exaustiva_par_6t e busca_exaustiva_par_8t.
- busca_local_seq
  - Este arquivo realiza a técnica de busca local para identificar qual a melhor solução para o problema de alocação dos alunos nos projetos. Ele define as seguintes funções para isso:
    - A função main() lê o arquivo de entrada e realiza, 100.000 vezes, a verificação da melhor alternativa possível ao gerar um vetor aleatório, utilizando a função random_alunos_dist(), e tentar realizar trocas par a par para ver se há uma melhoria na satisfação, utilizando a função escolhe_alunos(). Além disso, existe a função calc_satisfacao_atual() que faz o cálculo da satisfação atual da distribuição dos alunos em um determinado ponto da iteração.
- busca_local_par
  - Este arquivo é a versão paralelizada do busca_local_seq e realiza também 100.000 iterações, porém de forma paralela, utilizando a função #pragma omp parallel for, da biblioteca OpenMP. Este arquivo também possúi as versões com limitação do número de threads: busca_local_par_2t, busca_local_par_4t, busca_local_par_6t e busca_local_par_8t.
- busca_gpu (busca_local.cu)
  - Este arquivo é baseado no mesmo algoritmo do busca local paralelo, porém é executado por um operador thrust na gpu. Com isso, o calculo é todo feito de maneira pararelizada.

Compilando e executando

Para compilar o programa digite o seguinte código na pasta do projeto: Primeiro `cmake ./build` para usuários de Linux ou `cmake -D CMAKE_CXX_COMPILER=g++-9 -B ./build` para usuários de Mac e depois `make -C ./build`.
Para executar o programa digite o seguinte código na pasta do projeto: `build/ingenua < entradaX` onde X é o número da entrada desejada. Os arquivos de entrada estao disponíveis em: `projeto-validacao/entradas`







Resultados

Observando os resultados é possível afirmar que a mudança de python para C++ proporcionou uma melhora substancial na velocidade em quando os tamanhos de entradas fornecidas eram maiores, onde se viu uma melhora de quase 10 vezes na velocidade de processamento.

Os resultados da paralelização da solução exaustiva mostraram o esperado. Isso significa que o tempo de execução dos programas foi menor para uma maior quantidade de threads disponiveis e o desempenho da solução sequencial foi totalmente superado pelo desempenho das solucões paralelas, vendo uma melhora de quase 2 vezes no tempo de processamento para a maior entrada.

Assim como a sulução exaustiva, a solução de busca local apresentou o resultado esperado. Ou seja, quanto mais threads disponíveis, mais rápido o programa irá rodar devido à paralelização, principalmente para entradas maiores. Alémm disso, a gpu apresentou um desempenho muito superior em relação à solucao paralelizada em CPU, apenas sendo mais lenta para a menor entrada possível. Isso nos revela uma vantagem muito relevante da GPU para entradas grandes, mas uma desvantagem para programas de entradas pequenas.

Considerações finais

Com os resultados esperados, é possivel concluir que a paralelização de programas é sempre mais vantajosa quando é necessário um grande volume de processamento de dados concorrentes de forma eficiente, em relação ao processamento sequencial. E além disso, para problemas que exigem a paralelização, a solução encontrada pode ser implementada em GPU, caso seja possível dividir essas tarefas de forma independente, e assim obter uma grande melhora no tempo de processamento.