Insper

Super Computação

Igor Montagner, Luciano Soares

2019/2

Data de entrega: 29/11

Projeto 4 - Computação Distribuída

Neste projeto continuaremos trabalhando com o problema do 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.

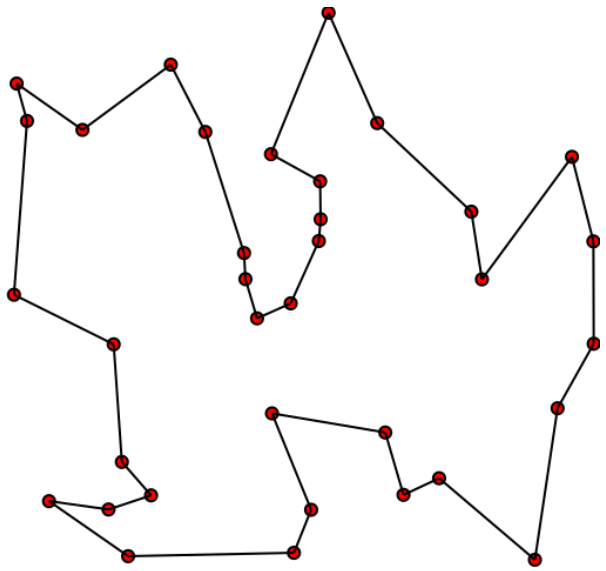

Figure 1: Cada ponto vermelho é uma empresa a ser visitada e a linha preta é o melhor trajeto

No projeto 2 trabalhamos com implementações em multi-core e conseguimos desempenho muito superior a implementação ingênua em Python ao escrever o programa em C++ e paralelizá-lo. No projeto 3 tivemos um desempenho muito bom na Busca Local, mas GPUs não encaixavam naturalmente no Branch and Bound, o que nos impedia de encontrar uma solução comprovadamente ótima. Neste projeto iremos resolver este problema usando computação distribuída.

Parte 1 - implementacão

Nossa resolução do caixeiro viajante envolverá duas etapas de implementação:

- Conceitos **D** e **C** focam na divisão de trabalho entre processos.
- Conceitos **B** e **A** focam em melhorias de desempenho baseada na comunicação eficiente entre os processos.

Divisão de trabalho entre processos

Conceito D

Nossa primeira implementação envolve distribuir o algoritmo de busca local. Já fizemos uma implementação GPU no projeto 3. Neste item você deverá

- implementar uma versão CPU do algoritmo
- distribuir essa versão para vários nós do cluster
- juntar os resultados e retornar o melhor valor encontrado

Não se esqueça de garantir que cada nó gere uma sequência diferente de soluções. Senão você estará resolvendo o mesmo problema repetidas vezes.

Conceito C

A próxima etapa de implementação envolve distribuir o algoritmo de enumeração exaustiva. Você deverá

- criar uma implementação que funcione para qualquer número de processos
- dividir o mais igualmente possível as tarefas entre os processos

Conceito B

A próxima tarefa é trabalhar com tarefas heterogêneas que se comuniquem. Ou seja, você irá rodar dois algoritmos diferentes e fazê-los cooperarem para chegar mais rapidamente em uma solução ótima. Este item irá juntar os dois programas acima em uma única solução. Ou seja, seu programa deverá

- criar *N-1* processos que realizem a enumeração exaustiva
- criar 1 processo que rode a busca local
- o processo da busca local envia sua melhor solução a cada 10 000 iterações para todos os processos de enumeração exaustiva.
- os processos de enumeração exaustiva atualizam sua melhor solução levando em conta esta mensagem recebida.
- seu processo de busca local deve parar de rodar quando a solução ótima for encontrada

Note que ao gerar soluções locais nossa enumeração exaustiva se torna muito melhor: ela pode evitar entrar em ramos de recursão que já são piores do que a solução local encontrada. Isto deverá diminuir consideravelmente o tempo de execução de nosso programa.

Conceito B+

Modifique seu programa acima para que cada processo que faz a enumeração exaustiva envie para os outros sua melhor solução. Desta maneira, além das soluções da busca local eles estarão se atualizando também com as melhores soluções que encontraram. Use mensagens assíncronas para tornar sua comunicação mais eficiente.

Conceito A+

Faça o processo que faz busca local rodar em CUDA. Você pode enviar sua melhor solução a cada 100 000 iterações, neste caso.

Requisitos de projeto

Se os requisitos de projeto abaixo não forem cumpridos sua nota máxima será **D**.

- [ ] CMakeLists.txt que gere um executável por método testado
- [ ] Relatório feito em Jupyter Notebook ou PWeave. Seu relatório deve conter as seguintes seções:
  - [ ] Descrição do problema tratado
  - [ ] Descrição dos testes feitos (tamanho de entradas, quantas execuções são feitas, como mediu tempo, infra usada)

  - [ ] Organização em alto nível de seu projeto.
  - [ ] Comparação com projeto em CPU multi-core
- [ ] Versão já rodada do relatório exportada para *PDF*
- [ ] README.txt explicando como rodar seus testes
- [ ] Conjunto de testes automatizados (via script Python ou direto no relatório)
- [ ] Respeitar os formatos de entrada e saída definidos na seção anterior
- [ ] Seu programa deverá retornar sempre o mesmo resultado.

Como usamos máquinas remotas, não é necessário que seu relatório execute os testes propriamente ditos. Você pode entrar um script Python ou Bash que executa os programas e salva os resultados em um conjunto de arquivos que seu relatório lê e analisa.

Além disto, como estamos rodando uma aplicação distribuída, a escala de tamanho de testes deverá aumentar. Experimente rodar seu programa com entradas de tamanho médio (dezenas de pontos) do TSPlib.

Parte 2 - Relatório e testes

O relatório seguira uma rubrica contendo diversos itens. A nota final de relatório é a média das notas parciais, levando em conta a seguinte atribuição de pontos.

- **I** - 0 pontos: Não fez ou fez algo totalmente incorreto.
- **D** - 4 pontos: Fez o mínimo, mas com diversos pontos para melhora.
- **B** - 7 pontos: Fez o esperado. Não está fantástico, mas tem qualidade.
- **A+** - 10 pontos: Apresentou alguma inovação ou evolução significativa em relação ao esperado.

Rubrica para SuperComputação – Projetos

Critério	Conceito			
	I	D	B	A+
Descrição do problema e da realização dos testes.	O relatório descreve o problema de maneira errada ou que induz ao erro. A descrição da implementação não bate com o código entregue.	O problema é descrito de maneira sucinta e clara. A descrição da implementação omite dados importantes da execução dos testes (máquina usada, arquivos de entrada, etc)	O problema tratado é descrito de maneira sucinta e clara. Detalhes da execução dos testes são apresentados e a escolha dos testes é justificada.	Além do item anterior, é feita alguma tentativa de relacionar características da máquina usada nos testes com os resultados colhidos.
Tamanho das instâncias de teste e quantidade de cenários testados	Não apresenta o resultado de nenhum teste no relatório ou apresenta apenas de casos triviais.	Realiza um conjunto pequeno de testes mas consegue demonstrar alguma diferença significativa de desempenho.	Realiza um conjuto abrangente de testes, levando em conta o consumo de recursos do sistema.	Leva a capacidade de sua máquina próxima de seu limite, realizando também testes intermediários de desempenho.
Medidas de consumo de recursos	Não mede nenhuma medida de desempenho de maneira correta ou consistente.	São feitas medições de tempo imprecisas (comando time) ou misturando partes diferentes do programa.	É medido consumo de tempo com maior precisão (por exemplo, std::chrono).	São feitas várias medições de tempo da mesma tarefa e é os resultados são apresentados como média e desvio padrão dos tempos.
Facilidade de leitura do relatório	O relatório contém erros grosseiros de escrita, não segue nenhuma estrutura clara de organização nem respeita a norma culta de escrita.	Os resultados estão explicados diretamente no texto de maneira confusa. Se utiliza tabelas ou gráficos, o faz incorretamente.	Ilustra as diferenças de desempenho com gráficos e tabelas e comenta brevemente seus resultados.	Ilustra as diferenças de desempenho com tabelas e gráficos e os interpreta de maneira detalhada no texto.