Algoritmo Paralelo para o Problema do Roteamento de Veículo Com Janelas de Tempo utilizando a plataforma CUDA

Thiago William Machado
Universidade Católica Dom Bosco
Campo Grande - MS
thiagueroth@gmail.com

Hemerson Pistori
Universidade Católica Dom Bosco
Campo Grande - MS
pistori@ucdb.br

Ricardo Ribeiro dos Santos
Universidade Federal do Mato Grosso do Sul
Campo Grande - MS
ricr.santos@gmail.com

Resumo

Este artigo apresenta o projeto e desenvolvimento de uma implementação paralela para o Problema de Roteamento de Veículos com Janelas de Tempo utilizando a plataforma CUDA. Este problema consiste em resolver o atendimento de um grupo de consumidores que possuem restrições de horário, visando um custo mínimo. A resolução deste algoritmo utiliza o Método Simplex em conjunto com o Algoritmo de Caminho Mínimo Com Janelas de Tempo, que é uma generalização do Algoritmo de Ford-Bellman-Moore. Uma simulação do algoritmo paralelo foi implementada, onde mostrou-se que, para instâncias com mais de 10 nós, o desempenho chegou a 2.11 vezes melhor do que a versão sequencial do algoritmo.

1. Introdução

O Problema de Roteamento de Veículos com Janelas de Tempo (PRVJT), foco deste trabalho, utiliza o algoritmo Simplex em conjunto com o algoritmo de Caminho Mínimo com Janelas de Tempo (CMJT). Problemas de Roteamento de Veículos, incluindo variações como o PRVJTs, pertencem à classe de problemas NP-completo [2].

Levando em consideração as dificuldades inerentes à obtenção de bons resultados com os algoritmos sequencias para o PRVJT, este artigo apresenta uma implementação paralela para o PRVJT por meio de uma estratégia de resolução proposta em [5]. A solução paralela aqui apresentada utiliza o paralelismo existente em GPUs ou Unidades de Processamento Gráfico da arquitetura CUDA da NVIDIA [3].

Experimentos foram realizados a partir da implementação paralela e da implementação sequencial com instâncias (grafos de rotas) de diferentes tamanhos. Para as instancias utilizadas, o speedup chegou a 2.2.

O artigo está organizando conforme segue: a Seção 2 apresenta a Descrição do Problema, dividida em Problema Mestre e Sub-Problema. O Algoritmo Paralelo para o PRVJT é abordado na Seção 3. A Seção 4 mostra os resultados obtidos. Por fim, a Seção 5 apresenta as conclusões deste artigo.

2. Descrição do Problema

Problemas do Roteamento de Veículos (PRV) consistem em, dado um conjunto de viagem a serem executadas, encontrar um ou mais roteiros para serem compridos por uma frota de veículos, visando um custo mínimo de atendimento. O Problema de Roteamento de Veículo com Janelas de Tempo (PRVJT) possui restrições de atendimento, janelas de tempo, para o comprimento de cada viagem.

A resolução do PRVJT utiliza uma técnica desenvolvida por [5, 6] que compreende a utilização de dois algoritmos muito conhecidos na literatura: O *Algoritmo Simplex*, que resolve o Problema-Mestre e de um *Algoritmo de Caminho Mínimo* baseado no algoritmo clássico de Ford-Bellman-Moore que resolve o Sub-Problema.

2.1 Solução do Problema-Mestre

A entrada para o problema Mestre é um grafo de rotas $G = (N, A)$, em que A é o conjunto de arestas e N o conjunto de vértices. O grafo de rotas é um grafo direcionado com uma única origem (ponto de saída do veículo) e um

único destino (ponto de chegada do veículo). Considere um conjunto de viagens onde cada viagem i é especificada por um intervalo de tempo $[a_i; b_i]$, no qual a viagem deve ser iniciada. Define-se inter-viagem ou viagem de conexão como sendo uma aresta que faz parte de uma viagem. A cada inter-viagem é associada um tempo t_{ij} de duração e um custo c_{ij}, indicando tempo e custo para um veículo ir do vértice i para o vértice j. Considere P como sendo o conjunto das viagens e I o conjunto de inter-viagens.

O Problema-Mestre é resolvido pelo Algoritmo Simplex, como dito anteriormente. A formulação em Programação Linear para o PRVJT é necessário para a aplicação do Método Simplex. A modelagem desta versão é uma das mais utilizadas na literatura, descrita a seguir:

$$
\begin{array}{lll}
\text{Minimizar:} & \sum_{(i,j)\in A} c_{ij}x_{ij} & \\
\text{sujeito a:} & & \\
 & \sum_{j\in N} x_{ji} = 1 & i \in P \quad (1)\\
 & \sum_{k\in N} x_{ik} = 1 & i \in P \quad (2)\\
 & x_{ij} \geq 0 & (i,j) \in A \\
 & x_{ij} > 0 \Longrightarrow t_i + t_{ij} \leq t_j & (i,j) \in N \quad (3)\\
 & a_i \leq t_i \leq b_i & i \in P \quad (4)\\
 & x_{ij} = \{0,1\} & (i,j) \in A \quad (1)
\end{array}
$$

O objetivo é construir um conjunto de rotas com menor custo, satisfazendo as janelas de tempo e incluindo cada viagem exatamente uma vez. A restrição (1) informa que para chegar ao vértice i é necessário sair de algum vértice j e a restrição (2) indica que é necessário partir de algum vértice i para chegar a k. A restrição (3) indica que não podem haver caminhos negativos e tem que respeitar os tempos. Por fim a restrição (4) reflete o intervalo de tempo no qual a viagem deve ser iniciada.

O Método Simplex é utilizado para resolver problemas de Programação Linear. Dado uma matriz A, na primeira fase do método é dividida em duas: básicas B e não básicas N, de forma que $A = (B, N)$. O mesma divisão ocorre com as váriaveis da função objetivo $x = (x_B, x_N)$, no qual x_B é o vetor de variáveis básicas e x_N o vetor de váriaveis não básicas.

A Figura 1 mostra os passos seguintes, segunda fase, para a resolução do Método Simplex:

Cálculo da Solução Básica $\hat{x}_B = B^{-1}b\, \hat{x}_N = 0$

Vetor Multiplicador Simplex $\pi^T = c_B^T B^{-1}$

Custos Relativos $c_k - \pi^T a_k = minimo\{c_j - \pi^T a_j : j \in IN\}$

Determinação da variável a entrar na base $c_k - \pi^T a_k \geq 0$

Cálculo da Direção Simplex $y = B^{-1}a_{Nk}$

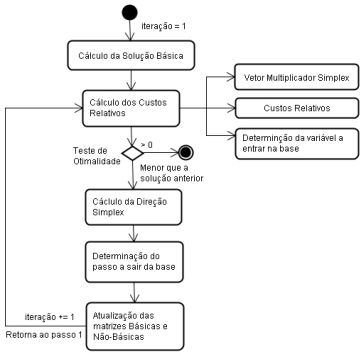

Figura 1. *Segunda Fase - Método Simplex*

Em cada iteração, como visto na Figura 1, o Método Simplex efetua o teste de Otimalidade, verificando se a iteração atual é ótima ($c_k - \pi^T a_k \geq 0$). Se for, a solução é ótima, caso contrário o algoritmo segue a fim de definir a coluna que irá sair da matriz básica B, para isso é feito o cálculo da direção simplex.

É importante ressaltar que a versão utilizada neste trabalho é o Método Simplex com Geração de Colunas. O processo de Geração de Colunas foi incorporado ao Método Simplex, para determinar a nova coluna a entrar na base a cada iteração da Segunda Fase. Para encontrar a nova coluna resolve-se o Sub-Problema do caminho mínimo com janelas de tempo.

2.2 Solução do Sub-Problema

O Sub-Problema é responsável por determinar a rota de custo mínimo respeitando as restrições de horário, chamado Caminho Mínimo com Janelas de Tempo (CMJT). Para o PRVJT, O CMJT consiste em encontrar uma rota de menor custo entre o vértice de origem e o vértice de destino do grafo de rotas, respeitando as restrições de horários de cada um dos vértices [1].

O CMJT é utilizado logo após o cálculo do custo relativo pelo Método Simplex, indicando qual é a rota a ser seguida. Ou seja, quais colunas associadas ao caminho mínimo foram geradas para entrar na base e continuar a execução do Simplex.

Para resolução do problema CMJT foi realizada uma adaptação no algoritmo clássico de Ford-Bellman-Moore para obtenção do menor caminho. Seja $F(i) = \{j : (i,j) \in A, j \in N\}$ o conjunto de sucessores do nó i e L o conjunto de nós que ainda não foram explorados. A árvore de caminho mínimo saindo da origem s é melhorada a cada iteração do algoritmo. O método das etiquetas consiste em retirar do conjunto L um nó i, em seguida tenta-se melhorar todos os sucessores de i. O novo nó melhorado é inserido no con-

junto L, assim, quando o conjunto L for vazio, a etiqueta θ_i associada ao nó i representa o caminho de comprimento mínimo saindo da origem s e terminando no nó i [6].

O Algoritmo de Caminho Mínimo de Ford-Bellman Moore utlizado adiciona uma adaptação para as Janelas de Tempo. A etiqueta θ_i, neste caso, possui dois parâmetros t_i o tempo de chegada da origem s até i, e c_i o custo associado a i.

3. Algoritmo Paralelo Utilizando a Plataforma CUDA

Esta seção apresenta a estratégia adotada pelo algoritmo paralelo implementado utilizando a arquitetura CUDA.

CUDA (*Compute Unified Device Architecture*) é uma platarfoma da NVIDIA criada em 2006, que permite utilizar o poder computacional de Unidades de Processamento Gráfico (GPUs) para computação de propósito geral baseando-se no paradigma SIMT (Single Instruction Multiple Threads). CUDA possui três abstrações básicas: hierarquia de grupo de *threads*, memória compartilhada e sincronização via barreiras [4]. A arquitetura CUDA permite ao programador especificar o que sera executado em paralelo (código do *Device*), podendo despachar centenas de *threads* para execução, e o que será executado sequencialmente (código do *Host*).

Estudos preliminares foram feitos sob a implementação sequencial para o PRVJT. Esta implementação retorna uma solução ótima, porém o tempo computacional para grandes instância mostrou-se inadequado. Foi utilizado a ferramenta *gprof*, que fornece as chamadas e os tempos de cada função do algoritmo para a avaliação de desempenho.

A Figura 2 mostra que a função mais custosa da implementação sequencial foi a "Poda_etiqueta". Esta função pertence ao Sub-Problema (CMJT) e é responsável pela inserção ou remoção de etiquetas.

A Figura 4 mostra o fluxograma de execução do PRVJT. Essa Figura detalha a interface entre o Algoritmo Simplex (Figura 1) com o Algoritmo CMJT. A lista de rotas apresentada no fluxograma em tracejado correspondendo ao Sub-Problema refere-se ao conjunto de rotas possíveis a partir de um vértice i do grafo de rotas.

Na implementação paralela, os passos do CMJT (retângulo tracejado da Figura 4) são executados pelo *device* CUDA enquanto que o restante do Algoritmo PRVJT (basicamente, o algoritmo Simplex) é executado pelo *host*. É importante ressaltar que, a cada ciclo de execução do CMJT, o *host* é quem atualiza a lista de rotas e invoca novamente o *device*. Isso é realizado enquanto a lista for diferente de nula.

A estratégia de implementação paralela consiste em fazer com que cada vértice sucessor do vértice atual i seja

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 75.00      0.03     0.03    13239     0.00     0.00  Poda_etiqueta
 25.00      0.04     0.01    56580     0.00     0.00  nova_etiqueta
  0.00      0.04     0.00    98792     0.00     0.00  insereL
  0.00      0.04     0.00    46553     0.00     0.00  Poda
  0.00      0.04     0.00    11909     0.00     0.00  insere_no3
  0.00      0.04     0.00     7656     0.00     0.00  Remove
  0.00      0.04     0.00     3139     0.00     0.00  consF
  0.00      0.04     0.00     2312     0.00     0.00  procura_next
  0.00      0.04     0.00     1606     0.00     0.00  destruir_Lno
  0.00      0.04     0.00     1606     0.00     0.00  removeL
  0.00      0.04     0.00      146     0.00     0.00  aloca_no
  0.00      0.04     0.00       73     0.00     0.00  Minimocr
  0.00      0.04     0.00       73     0.00     0.55  solucao2
  0.00      0.04     0.00       20     0.00     0.00  vet_custo
  0.00      0.04     0.00        1     0.00    40.00  Calculo
  0.00      0.04     0.00        1     0.00     0.00  Fase_I
  0.00      0.04     0.00        1     0.00     0.00  Le_dados
```

Figura 2. *Avaliação da implementação sequencial do PRVJT*

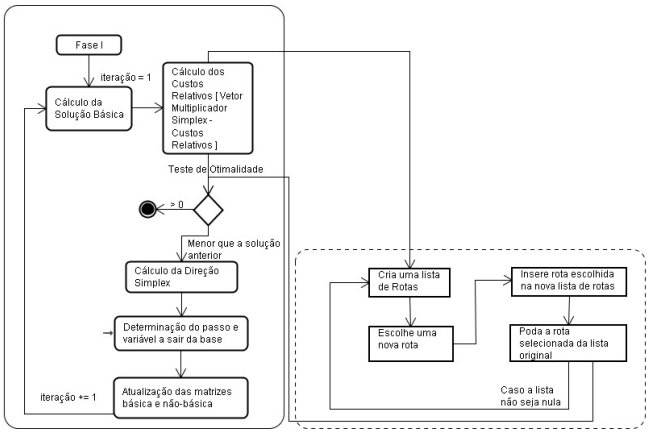

Figura 3. *Fluxograma de execução do PRVJT*

uma *thread*. Esses vértices sucessores pertencem ao conjunto L de vértices. A medida que se explora L as melhores rotas vão sendo definidas. Quando L for nulo significa que a execução do algoritmo chegou ao fim e a etiqueta θ_s, em que s é vértice destino, representa a etiqueta da rota de custo mínimo. Essa rota é enviada ao Simplex, para que ele continue a execução.

4 Experimentos e Resultados

Os experimentos realizados consistem na avaliação de desempenho entre uma versão sequencial e a versão paralela do Sub-Problema do algoritmo PRVJT. Assim, foram implementados algoritmos sequencial e paralelo do algoritmo CMJT seguindo a estratégia de paralelização mencionada na Secão 3.

Os experitmentos para a versão sequencial foram real-

izados em uma plataforma computacional composta por um Processador Intel Core 2 Duo 2.60GHz, 1 GB de memoria RAM, 80GB de disco rígido e sistema operacional Linux Ubuntu 9.10. A versão paralela foi executada sob uma plataforma CUDA composta por uma placa de vídeo GeForce GTS 250, com um 1GB de memória global, 128 processadores e frequência de clock de 1.8GHz.

Para avaliar as implementações foram utilizadas instâncias que variaram de 10 a 2000 vértices. Essas instâncias correspondem a grafos de rotas onde todos os vértices, exceto os vértices origem e destino, são interligados entre si formando um grafo completo. A Tabela 1 apresenta os resultados obtidos com os tempo medidos em segundos.

Tabela 1. Tempos (s) dos Algoritmos e Speedup (T_s/T_p)

Nós	Sequencial	Paralelo	Speedup
10	0.00001	0.00052	0.02
50	0.00052	0.00354	0.02
100	0.00347	0.00965	0.36
200	0.02428	0.03023	0.80
300	0.07856	0.06523	1.22
400	0.18389	0.11899	1.54
500	0.35487	0.20209	1.76
600	0.61075	0.32676	1.87
700	0.96732	0.79466	1.95
800	1.44009	0.72213	1.99
900	2.04894	1.00559	2.04
1000	2.80573	1.35611	2.07
2000	22.8303	10.8183	2.11

A partir de 300 vértices a versão paralela obteve um desempenho melhor sobre a sequencial, com *speedup* (quarta coluna da Tabela 1) de 1.22. O ganho nas demais instâncias segue uma evolução conforme aumenta-se a quantidade de vértices. O gráfico da Figura 4 que apresenta os *speedups* obtidos, mostra essa evolução, onde o ponto alto da paralelização é com entradas entre 1000 e 2000 vértices. Para as instâncias avaliadas não foi possível observar um ponto de saturação no desempenho da implementação paralela. Assim, é possível que para instâncias ainda maiores (5000, 10000 vértices) ainda seja possível obter *speedup*.

5. Conclusão

Este trabalho apresentou o projeto e a implementação de um algoritmo paralelo para Problema de Roteamente de Veículos com Janelas de Tempo (PRVJT) utilizando a aquitetura CUDA da NVIDIA. A estratégia adotada pelo PRVJT consiste na resolução do problema mestre por parte

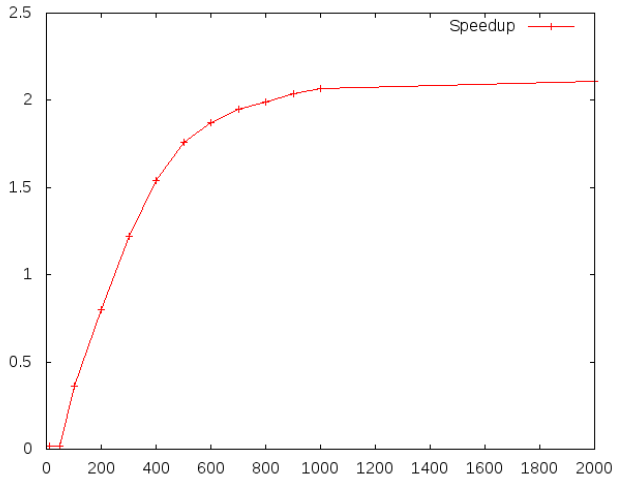

Figura 4. *Gráfico de Speedups*

do algoritmo Simplex e na resolução do sub-projeto de Caminho Mínimo com Janelas de Tempo pelo algoritmo de caminho mínimo de Ford-Bellman-Moore.

Os ganhos de desempenho da implementação paralela sob a sequencial podem ser observados a partir de instâncias com 10 vértices. Os resultados demonstram que o ganho da implementação paralela é significativo e, a princípio, pode continuar com instâncias ainda maiores do que as avaliadas neste trabalho.

Como trabalho futuro pretende-se estender a avaliação de desempenho com instâncias bem conhecidas (instâncias de Solomon) da área de roteamento de veículos. Além disso, é interessante também avaliar o desempenho com a implementação completa (implementações dos problemas mestre e sub-problema) e, por fim, comparar o desempenho com outras implementações paralelas para o PRVJT.

Referências

[1] M. S. F. Desrochers. *A Generealized Permanent Labelling Algorithm for the Shortest Path Problem With Time Windows*, chapter 26. INFOR, 1988.

[2] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. 1979.

[3] T. R. Halfhill. Parallel processing with cuda. *Microprocessor Report*, Jan. 2008.

[4] J. Nickolls, I. Buck, and M. Garland. Scalable parallel programming. *ACM QUEUE*, pages 40–53, Abril 2008.

[5] R. M. Oliveira. A técnica de grereação de colunas aplicada a problemas de roteamento. Master's thesis, ICMC-USP, São Carlos, 2001.

[6] R. M. Oliveira and R. Santos. Problema de roteamento com janelas de tempo: Uma abordagem via geração de colunas. *In XL Anais do Simpósio Brasileiro de Pesquisa Operacional*, 2008.