Estudo Preliminar sobre a Escalabilidade de um Algoritmo Genético Paralelizado com OpenMP

Mateus Fontoura Gomes da Rosa, Márcia Cristina Cera
Universidade Federal do Pampa (UNIPAMPA) – Campus Alegrete
Av. Tiarajú, 810, CEP: 97546-550 – Alegrete – RS - Brasil
mateus.fontourag@gmail.com, marciacera@unipampa.edu.br

Resumo. *Este artigo investiga a escalabilidade de um Algoritmo Genético (AG) aplicado ao Problema de Roteamento de Veículos (PRV) e paralelizado utilizando a API (Aplication Programming Interface) OpenMP. Os testes mostraram que houve aumento no desempenho do AG ao aumentar-se o número de threads na arquitetura alvo, sendo que a configuração de melhor desempenho varia de acordo com a característica da instância computada.*

1. INTRODUÇÃO

Este trabalho visa ampliar o estudo realizado por Gressler e Cera (2014) e comparar os resultados obtidos em Rosa e Cera (2014), buscando analisar a escalabilidade de um Algoritmo Genético (AG) Paralelo para o Problema de Roteamento de Veículos (PRV). O PRV é um problema onde um veículo necessita realizar entrega de mercadorias em um certo número de cidades, usando o percurso com a menor distância possível. O veículo possui capacidade limitada e necessita retornar ao depósito para reabastecer-se.

Utilizou-se um AG para solucionar o PRV pois, segundo Linden (2012), esta meta-heurística possui uma aplicabilidade quase infinita, podendo ser utilizada em qualquer tipo de problema, fornecendo uma solução próxima da ótima. Adicionalmente, AG são meta-heurísticas que adequam-se bem a paralelização (Linden, 2012).

O AG foi paralelizado com OpenMP (CHAPMAN et al., 2008) que é um padrão para a programação em arquiteturas multicore. Ele é uma API que provê diretivas com as quais o usuário pode informar ao compilador como será feito o processo de paralelização do código e como ocorrerá a distribuição da carga de trabalho entre as *threads*. Neste trabalho foram realizados testes em uma arquitetura com 2 processadores de 8 núcleos com *Hyper-threading*, onde o número de *threads* pode ser variado de 2 a 64, para prover subsídios para a análise preliminar da escalabilidade do AG paralelo.

2. ALGORITMOS GENÉTICOS

De maneira similar à teoria evolutiva de Darwin, os AG tratam os dados como populações que evoluem ao passar do tempo. Os indivíduos da população consistem das soluções e os resultados que compõem as soluções são chamados de genes. A Figura 1 apresenta o fluxograma do AG implementado, onde as principais funcionalidades são: **Inicializa População**: inicializa os indivíduos que compõem a primeira geração do AG; **Avalia População**: avalia a qualidade do indivíduo gerado, ou seja calcula a distância percorrida onde as menores distâncias representam as melhores soluções; **Nova Geração**: inicializa a próxima geração do AG através da aplicação dos operadores genéticos; **Cruzamento**: operador genético que recebe dois indivíduos e realiza o cruzamento de seções de seus cromossomos (transferência de carga genética), gerando assim uma nova solução; **Mutação**: operador genético que altera as características de

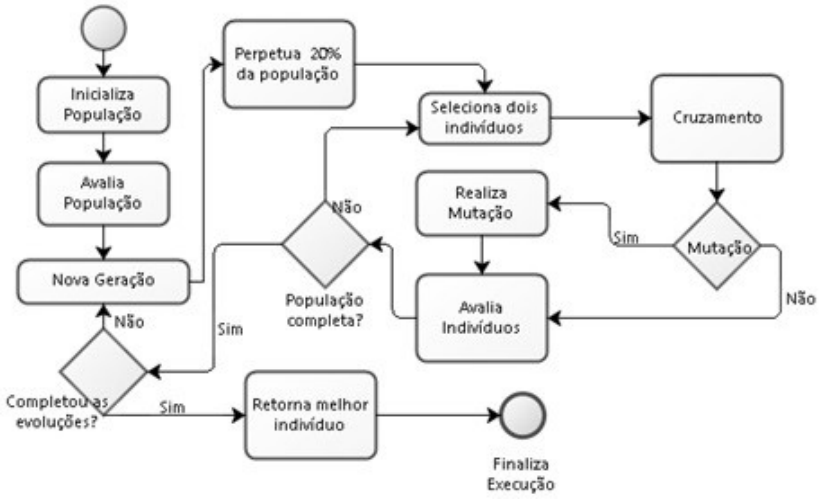

Figura 1: Fluxograma do AG implementado.

alguns genes da solução (alteração em genes de um cromossomo) buscando evitar uma convergência prematura. Ela ocorre numa determinada porcentagem de indivíduos.

3. ALGORITMO GENÉTICO PARALELIZADO COM OPENMP

A paralelização se deu nas regiões de maior processamento do AG identificadas através da análise do perfil de execução do algoritmo (Gressler e Cera, 2014). Foram definidas duas regiões paralelas, uma envolvendo a perpetuação de indivíduos (usando a diretiva `sections/section` do OpenMP) pois ela é feita em duas etapas independentes (perpetuação de 10% dos melhores indivíduos e perpetuação de 10% de indivíduos selecionados aleatoriamente). A outra região paralela ocorre quando são realizadas as chamadas às funções Cruzamento e Avaliação na concepção de uma nova geração (usando `parallel for`).

4. CONFIGURAÇÕES DO AMBIENTE DE EXECUÇÃO

Os testes foram feitos usando uma W*orkstation* Dell Precision T7600 da Universidade Federal do Pampa (UNIPAMPA), com dois processadores Intel® Xeon E5-2650 com frequência de 2.80 GHz, cada um com 8 núcleos e *Hyper-threading*, o qual será chamado de Xeon. Foi feita uma comparação com os resultados apresentados em Rosa e Cera (2014), onde fora utilizado um processador Intel® Core™ i7-3517U com 2 núcleos físicos e 2 lógicos, o qual será chamado i7**.**

Nos testes foram usadas duas instâncias do *Benchmark* de Christofides, a c50 e a c100, que consistem respectivamente de um percurso com 50 e 100 cidades. Cada uma das instâncias foi executada 30 vezes com diferentes números de *threads*: 2, 4, 8, 16, 32 e 64, valor escolhido em função da arquitetura possuir dois processadores com 8 núcleos físicos. Os resultados representam a média do tempo obtido, onde o desvio padrão foi inferior a 1% da média. Conforme mostrado em Rosa e Cera (2014), as políticas de escalonamento de iterações do OpenMP não impactaram de forma significativa no desempenho do AG paralelo, logo foi utilizado apenas a *Static*, onde a distribuição das iterações é feita de forma estática e em tempo de compilação.

Para os parâmetros internos do AG, foram utilizadas as seguintes configurações:*(i)* a técnica de cruzamento de 1 ponto, onde é escolhido um ponto em dois indivíduos,

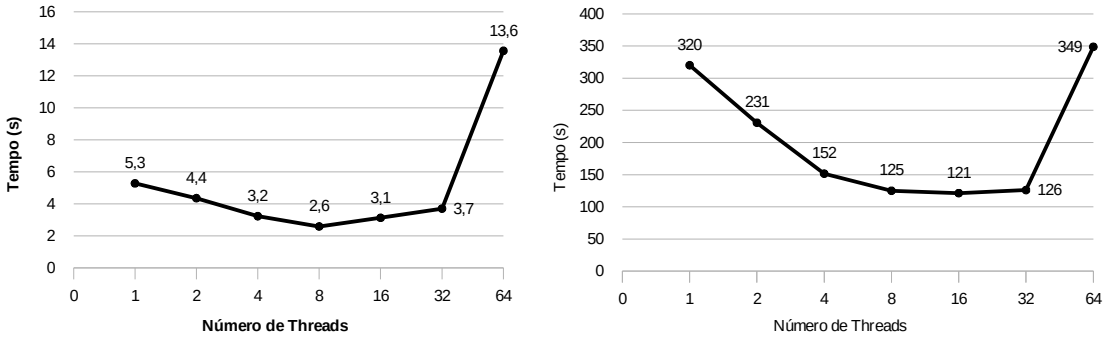

Figura 2: Tempo de execução (em segundos) pelo número de threads para as instâncias c50 (esquerda) e c100 (direita).

seus genes são recombinados, intercalando as partes dos genitores, gerando assim dois novos indivíduos; *(ii)* a técnica de mutação randômica, que consiste em sortear uma das seguintes técnicas: troca, troca bloco, inserção e inversão. As técnicas de troca e troca bloco selecionam um ou um bloco de genes do cromossomo e os trocam de lugar. Na inserção escolhe-se uma seção do indivíduo, removendo-a e inserindo-a em outro lugar no mesmo indivíduo. A inversão seleciona e inverte uma seção de genes do indivíduo.

5. ANÁLISE DOS RESULTADOS

O tempo de execução (em segundos) para a instância c50 conforme varia-se a quantidade de *threads* é demonstrado na Figura 2 à esquerda. Como é possível perceber, com 8 *threads* o AG obteve o menor tempo de execução, que representa um *speedup* levemente superior a 2. O pior desempenho foi observado com 64 *threads*, onde o tempo de execução ultrapassou o tempo sequencial, sendo aproximadamente 2,6 vezes maior. Este comportamento deve-se ao fato de que a instância c50 requer um tempo de computação baixo (em torno de 5,3 segundos na versão sequencial). Logo, o aumento no número de *threads* também aumenta o impacto da sobrecarga (*overhead*) da paralelização e comunicação. Adicionalmente, a justificativa para que o melhor desempenho esteja vinculado a utilização de 8 *threads* deve-se a característica da arquitetura que possui 2 processadores com 8 núcleos – quando a aplicação precisa usar os 2 núcleos concomitantemente a comunicação entre eles impacta no desempenho.

Ainda na Figura 2, à direita, são apresentados os tempos para a instância c100. Nela, o menor tempo de execução foi observado com 16 *threads*, com um *speedup* de 2,64. Embora o *speedup* tenha sido abaixo do ideal em decorrência da característica de como o AG foi implementado, foi possível obter um desempenho melhor utilizando uma carga de trabalho maior (tempo sequencial de 320 segundos). A sobrecarga da comunicação entre os núcleos não impediu o ganho de desempenho com 16 *threads*, já a utilização do *Hyper-threading* (32 *threads*) sim. O pior desempenho continuou sendo com 64 *threads*, o qual ultrapassou o tempo de execução sequencial em 29 segundos.

A Figura 3 compara os *speedups* obtidos para as instâncias c50 e c100 de 2 a 64 *threads* no Xeon e de 2 a 8 *threads* no i7. Para a instância c50 (Figura 3, à esquerda), o aumento da escalabilidade numa arquitetura com um número maior de núcleos levou a um desempenho, porém apenas a configuração de melhor desempenho ficou com um *speedup* em torno de 2. Já para a instância c100, que possui uma maior carga de processamento, o ganho de desempenho é um pouco maior, porém mantêm-se para a maior parte dos números de *threads* acima de 2.

5. CONCLUSÃO

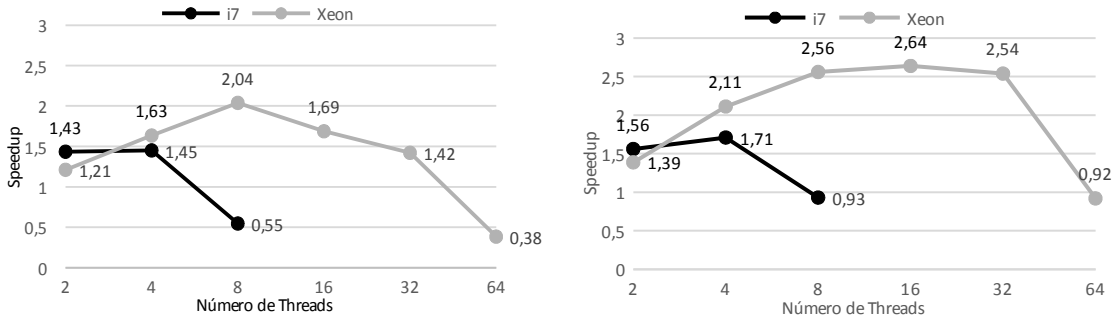

Figura 3: Comparação entre os processadores i7 e Xeon para as instâncias c50 (esquerda) e c100 (direita).

Com os testes foi possível observar que o desempenho do AG paralelizado utilizando OpenMP acelerou a obtenção das soluções. A escalabilidade do AG pode ser observada nas comparações com o processador i7, onde a arquitetura com dois processadores Xeon obteve um *speedup* maior. Os ganhos de desempenho foram melhor observados quando utilizada uma instância maior (que demanda um maior tempo de execução). Sendo possível observar que o melhor desempenho obtido foi para a instância c100 com 16 *threads*, reduzindo o tempo de execução do AG para um pouco menos que duas vezes o tempo de execução. O *speedup* obtido ainda não é ideal, porém reduz consideravelmente o tempo de execução se comparado com os resultados obtidos pelo processador i7. Para trabalhos futuros, serão realizados testes utilizando todas as instâncias do *Benchmark* de Christofides. Adicionalmente, espera-se realizar uma análise de complexidade do AG, a fim de melhorar ainda mais o desempenho do AG.

6. REFERÊNCIAS

ROSA. M. F. G. ; CERA, M. C. (2014) “Análise do desempenho do Algoritmo Genético paralelizado com OpenMP”. Anais do XXVI Congresso Regional de Iniciação Científica e Tecnologica em Engenharia, Alegrete/RS. p. 1-4.

LINDEN, R; (2012) “Algoritmos Genéticos”. Rio de Janeiro, Ciência Moderna, 3ªEd.

TAILLARD E. (2015) “Elementary (capacitated) vehicle routeing - Christofides et al.”, http://mistic.heig-vd.ch/taillard/problemes.dir/vrp.dir/vrp.html, acesso: janeiro-2015.

CHAPMAN B; JOST B; VAN DER PASS R; (2008) “Using OpenMP”. Cambridge, MA, Estados Unidos: Massachusetts Institute of Technology.

GRESSLER, H. O. ; CERA, Márcia C. (2014) “O Impacto da paralelização com OpenMP no desempenho e na qualidade das soluções de um algoritmo genético”. Revista Brasileira de Computação Aplicada, Passo Fundo, v. 6, n. 2, p. 35-47.