Universidade Federal do Pampa
Mateus Fontoura Gomes da Rosa

Melhorando o Desempenho de um Algoritmo Genético Paralelizado com OpenMP

Alegrete
2016

Mateus Fontoura Gomes da Rosa

Melhorando o Desempenho de um Algoritmo Genético Paralelizado com OpenMP

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Márcia Cristina Cera

Alegrete

2016

Mateus Fontoura Gomes da Rosa

Melhorando o Desempenho de um Algoritmo Genético Paralelizado com OpenMP

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 28 de nov de 2016

Banca examinadora:

Márcia Cristina Cera
Orientador

Aline Vieira de Mello
UNIPAMPA

Claudio Schepke
UNIPAMPA

Este trabalho é dedicado a minha mãe, minha eterna e maior apoiadora.

Agradecimentos

Dedico este trabalho aos meus pais, meus eternos apoiadores que permaneceram ao meu lado através das maiores dificuldades e que sempre fizeram tudo em seu alcance para permitir esta oportunidade. Agradeço à minha orientadora, Márcia Cristina Cera, que sempre demonstrou bastante paciência comigo e foi de fato uma orientadora, sempre me orientando ao melhor rumo possível.

"To the scientist there is the joy in pursuing truth,
which nearly counteracts the depressing revelations of truth.
(Howard Philips Lovecraft)

Resumo

Este trabalho propôs a implementação de duas versões de um Algoritmo Genético (AG) utilizando o Modelo de Ilhas. A diferença entre essas duas versões consiste no número de evoluções que cada ilha desempenhou. Cada versão foi então aplicada ao Problema de Roteamento de Veículos (PRV) e paralelizado com OpenMP. O PRV consiste de um problema combinatório, onde um veículo com capacidade limitada precisa atender um número de bairros ou cidades, necessitando retornar a um depósito para reabastecer quando esgotar sua capacidade. A solução do problema consiste em uma trajetória que atenda todas as cidades ou bairros e possua a menor distância total possível. Para solucionar este problema, é utilizado um AG, que consiste em um tipo de Meta-heurística que simula o comportamento evolutivo das espécies. O Modelo de Ilhas consiste de uma variação de AG que utiliza múltiplas populações (ilhas), onde em cada uma dessas populações foi simulado o comportamento evolutivo das espécies. Este AG foi paralelizado utilizando a API (*Application Programming Interface*) OpenMP. Para analisar o desempenho do AG aplicado ao modelo de ilhas, foram feitas execuções do AG, onde estes testes foram divididos em três momentos. O primeiro verifica o desempenho geral do AG e compara a nova versão, que utiliza a abordagem de ilhas, com a versão convencional do AG. O segundo momento compara a eficiência das execuções realizadas na arquitetura utilizada, verificando quais das duas versões do AG aplicado ao Modelo de Ilhas obteve melhor eficiência computacional. O terceiro momento realiza uma comparação entre as soluções obtidas com o AG com o objetivo de verificar qual das duas versões possui uma melhor qualidade de solução. Após os testes, realizados com as instâncias c50, c100, c120 e c150, obteve-se um ganho de *speedup* acima de 5 utilizando-se 6 threads em uma arquitetura com 6 núcleos de processamento (*cores*), obtendo assim um desempenho melhor que os observados nos trabalhos passados, comprovando assim a eficácia do Modelo de Ilhas como uma alternativa para melhorar o desempenho de AG. Também foi verificado que o Modelo de Ilhas permite que o AG encontre soluções de boa qualidade quando comparado as soluções encontradas em trabalhos prévios. Ao comparar as duas Versões da implementação com o Modelo de Ilhas foi possível observar que embora a Versão 2 possua maiores *speedups* em média e melhores soluções, ela sofre de um aumento significativo em seu tempo de execução, favorecendo assim o uso da Versão 1 da implementação com o Modelo de Ilhas em função da diferença entre ambos em *speedup* e qualidade de solução não ser tão significativa.

Palavras-chave: Algoritmos Genéticos. Problema de Roteamento de Veículos. Paralelização. OpenMP. Modelo de Ilhas.

Abstract

This work proposes two versions of a implementation using the Island Model of a Genetic Algorithm(GA). The difference between these two versions consist on the number of evolutions each island will perform. Each version is then applied to the Vehicle Routing Problem (VRP) and parallelized with OpenMP. The VRP is a combinatorial problem, where a vehicle with limited capacity must supply a number of neighborhoods or cities, requiring a return to the deposit to replenish when its capacity is exhausted. The solution of the problem is a path that meets all cities or neighborhoods and has the smallest total distance possible. To solve this problem, a GA is used, which consists on a type of metaheuristic that simulates the evolutionary behavior that each species go through. The Island Model is a variation of GA that utilizes multiple populations (islands), where on each one of these populations the species evolutionary behavior will be simulated. This GA is parallelized using the API (Application Interface) OpenMP. To the performance of the GA executions of the GA will be tested, where these tests will be divided in three moments. The first moment tests the overall performance of the GA and compares the new versions, one that uses the Island Model, and the conventional version of the GA. The second moment compares the efficiency of the executions realized on the architecture that is utilized, verifying which of the two versions of the GA applied to the Island Model obtained the best computational efficiency. The third moment realizes a comparison between the solutions obtained with the GA to better observe which of the two versions obtained a better quality of solution. After the tests, realized with the instances c50, c100, c120 and c150 a gain on speedup higher than 5 was obtained when 6 threads were used on an architecture with 6 processing cores , resulting on a higher performance when compared to past works, thus proving the efficiency of the Island Model as an alternative to increae the performance of the GA. Additionally, it was possible to verify that the Island Model allows the GA to reach solutions of good quality when compared to solutions obtained on previous works. Also, when comparing both versions of the Island Model it was possible to observe that while the Version 2 has better speedups on average and reaches better solutions, it suffers from a significant increase on the execution times, thus favoring the use of the Version 1, since their differences on speedup and quality of solution are not significant.

Key-words:Vehicle Routing Problem. Genetic Algorithm. Parallelization. OpenMP. Island Model.

Lista de ilustrações

Lista de tabelas

Lista de siglas

AG Algoritmo Genético

AGH Algoritmo Genético Híbrido

API *Application Programming Interface*

CUDA *Computer Unified Device Architecture*

GA *Genetic Algorithm*

GPU *Graphics Processing Unit*

IPP Interface de Programação Paralela

MatLab *Matrix Laboratory*

MPI *Message Passing Interface*

OpenMP *Open Multi-Processing*

PCV Problema do Caixeiro Viajante

PRV Problema de Roteamento de Veículos

Pthreads *POSIX Threads*

SPMD *Single Proccess Multiple Data*

TSP *Traveling Salesman Problem*

VRP *Vehicle Routing Problem*

Sumário

1 Introdução

O Problema de Roteamento de Veículos (PRV) consiste de um problema combinatório com uma aplicabilidade para telecomunicações, pesquisa, planejamento de produções e transporte (CRAINIC, 2008). Este problema possui uma complexidade NP-Difícil e, por isso ainda não possui uma solução padronizada , sendo muito utilizado para testar propostas de meta-heurísticas (CRAINIC, 2008). Este trabalho aborda o Algoritmo Genético (AG) como meta-heurística utilizada para solucionar o PRV. AGs consistem de algoritmos baseados em heurísticas de otimização global (LINDEN, 2012). Esta meta-heurística age de maneira semelhante a teoria evolutiva de Charles Darwin, cujo principal aspecto consiste da sobrevivência do organismo mais apto ao ambiente. De maneira semelhante, AGs tratam as possíveis soluções de um determinado problema como indivíduos. O conjunto de indivíduos é denotado como população e esta população selecionará e cruzará seus indivíduos para dar à luz a novos indivíduos. A cada geração os indivíduos são substituídos por indivíduos com melhores soluções para o problema, até que convergem em uma solução definitiva. Esta convergência ocorre após uma certa quantidade de evoluções.

Para melhorar o desempenho do AG implementado, foi utilizada uma nova proposta, usando o Modelo de Ilhas (*Island Model*), conforme o trabalho de Ochi et al. (1998). Esta abordagem consiste em utilizar múltiplas populações para melhorar a solução final do AG. Porém a aplicação do Modelo de Ilhas implica em um aumento considerável no tempo de computação do AG Convencional, para solucionar este problema foi feito uso de paralelização através do uso de OpenMP.

1.1 Objetivo

O objetivo geral deste trabalho é melhorar o desempenho do AG convencional ao aplicar ao Modelo de Ilhas. Desta forma, solucionando o problema de desempenho visto nos trabalhos de Rosa e Cera (2015), Andrade e Cera (2016) e Gressler e Cera (2014), onde é possível ver que os *speedups* obtidos pelo AG convencional estavam abaixo do *speedup* ideal. O Modelo de Ilhas foi escolhido em função de sua maior granularidade, se comparado a versão convencional do AG, esperando-se assim obter melhores métricas de desempenho. Para melhor verificar melhorias de desempenho, foram implementadas duas versões do AG aplicado ao Modelo de Ilhas, denominadas Versão 1 e Versão 2, onde uma versão irá desempenhar um número maior de evoluções que a outra, resultando assim em uma maior carga computacional. Estas duas versões serão aplicadas a quatro instâncias do PRV. Será utilizada uma arquitetura para os testes, esta dispondo de um processador

com 6 *cores*. Este trabalho tem os seguintes objetivos específicos:

- Analisar o desempenho do Algoritmo Genético utilizando ao Modelo de Ilhas;
- Comparar o desempenho do Algoritmo Genético convencional com o modelo de ilhas;
- Avaliar o impacto do Modelo de ilhas na qualidade das soluções.

1.2 Organização do Trabalho

A organização deste trabalho se fará em 6 capítulos. O primeiro capítulo apresentou uma breve introdução acerca do trabalho. O segundo capítulo trará a revisão bibliográfica necessária para a compreensão do trabalho e apresentará trabalhos relacionados ao tema. No terceiro capítulo ocorrerá a especificação de como foram realizadas as implementações do AG (Convencional e Modelo de Ilhas), além de como foram feitas as paralelizações das implementações. O quarto capítulo consistirá da metologia do trabalho, demonstrando os diferentes parâmetros adotados para os testes, como os testes serão realizados e com qual arquitetura será feito. O quinto capítulo apresentará os resultados obtidos nos testes realizados. O capítulo seis apresentará as considerações finais quanto ao trabalho, além de possíveis rumos para os trabalhos futuros.

2 Contextualização

Este capítulo objetiva contextualizar os principais aspectos necessários para a compreensão do trabalho. A Seção 2.1 apresenta de forma breve o problema alvo escolhido, o PRV (Problema de Roteamento de Veículos) e o método de solução, o AG (Algoritmo Genético). A Seção 2.2 demonstra a IPP (Interface de Programação Paralela) utilizada para a paralelização: o *Open Multi-Processing* (OpenMP). A Seção 2.3 contextualiza este estudo frente a alguns trabalhos relacionados na área. E por fim, a Seção 2.4 ressalta os principais pontos levantados no capítulo.

2.1 Problema Alvo e Método de Solução

Esta seção explana sobre o problema alvo (Seção 2.1.1) e o método de solução utilizado (Seção 2.1.2), mencionando suas peculiaridades.

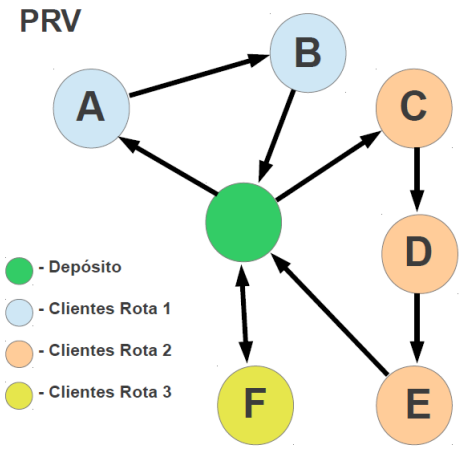
2.1.1 Problema de Roteamento de Veículos

O Problema de Roteamento de Veículos (PRV), do inglês *Vehicle Routing Problem* (VRP) consiste de uma variação do Problema do Caixeiro Viajante (*Traveling Salesman Problem*, ou PCV), onde se necessita descobrir o menor caminho para que um vendedor percorra um certo número de cidades sem que nenhuma delas seja percorrida mais do que uma vez. Já no PRV, ao invés do caixeiro viajante existe um veículo de capacidade limitada que precisa distribuir suas mercadorias percorrendo um conjunto de cidades (clientes). Por ser uma variação do PCV, o PRV pode ser facilmente convertido ao mesmo, uma vez que a capacidade do veículo seja ilimitada (STEEMAN, 2012; TOTH; VIGO, 2001). Como o veículo possui capacidade limitada no PRV, ele necessita passar por um depósito para repor as mercadorias de modo que o custo entre as rotas seja o menor possível.

Cada cliente (cidade) só pode ser visitado por um veículo e apenas uma vez. Na Figura 1 é demonstrado o exemplo de rotas do PRV, onde cada nó simboliza um cliente e o nó central é o depósito (GRESSLER, 2012): a Rota 1 sai do depósito, passa pela cidade A, pela cidade B e retorna ao depósito; a Rota 2 sai do depósito, passa pelas cidades C, D e E e retorna ao depósito; e por fim, a Rota 3 sai do depósito, visita a cidade F e retorna ao mesmo. Assim, essas três rotas permitiram uma única visita a todas as cidades. Como pode-se perceber, existem outras combinações de visitas que permitem solucionar o problema, sendo que o objetivo é encontrar aquela que forneça a menor distância de percorrimento. A seguir, apresentaremos o método utilizado para solucionar o problema

alvo: a meta-heurística Algoritmo Genético (AG).

Figura 1 – Exemplo de rotas no PRV.



Fonte: Gressler (2012)

2.1.2 Algoritmo Genético

AGs consistem de algoritmos baseados nos princípios da teoria evolutiva de Darwin, que introduz o conceito de evolução das formas de vida. Segundo Darwin, uma forma de vida evolui ao ser exposta as adversidades de um ambiente específico. Cada geração tenta adaptar-se da melhor forma ao ambiente a que foi exposto. Os indivíduos que conseguem adaptar-se ao ambiente passam seu código genético adiante, enquanto os que falham, perecem. Algoritmos Genéticos são um ramo dos algoritmos evolucionários e como tal podem ser definidos como uma técnica de busca baseada numa metáfora do processo de evolução natural (LINDEN, 2012).

Os AGs tratam suas soluções como indivíduos e o problema como o ambiente ao qual as soluções serão aplicadas. Cada uma destas soluções possui um código genético, ou seja, o conjunto de genes que a compõe, os quais sofrem ações de operadores genéticos (cruzamentos e mutações) levando a novas gerações de indivíduos, assim provendo uma maior diversidade genética e melhorando a qualidade das soluções obtidas pelo AG.

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.

Os AGs são muitas vezes empregados em problemas de otimização, conforme mencionado por Pacheco (1999) e Michalewicz (2013). Segundo Pacheco (1999) AGs podem ser aplicados aos seguintes problemas: Otimização de Funções Matemáticas, Otimização

Combinatorial, Otimização de Planejamento, Problema do Caixeiro Viajante, Problema de Otimização de Rota de Veículos (PRV), Otimização de Layout de Circuitos, Otimização de Distribuição, Otimização em Negócios e Síntese de Circuitos, provando assim ser uma meta-heurística bastante versátil e permitindo que problemas combinatoriais complexos (comumente classificados como NP-Difícil) sejam executados em computadores atuais normais (MICHALEWICZ, 2013). Os AGs possuem uma série de componentes específicos a eles e que simulam os componentes evolutivos da teoria ao qual são baseados. Os componentes são:

- **População Inicial:** É a primeira população da primeira geração do AGs;
- **Função de Avaliação:** Simula o comportamento natural, onde espécies evoluem e adaptam-se ao ambiente ao qual são aplicadas. O AG simula uma Função de Avaliação, que determina a aptidão dos indíviduos ao ambiente aplicado, desta forma apenas os mais aptos passarão para a próxima geração;
- **Operadores Genéticos:** São operadores que agem sobre os componentes genéticos do AG (genes e cromossomos), sendo estes: Cruzamento e Mutação.

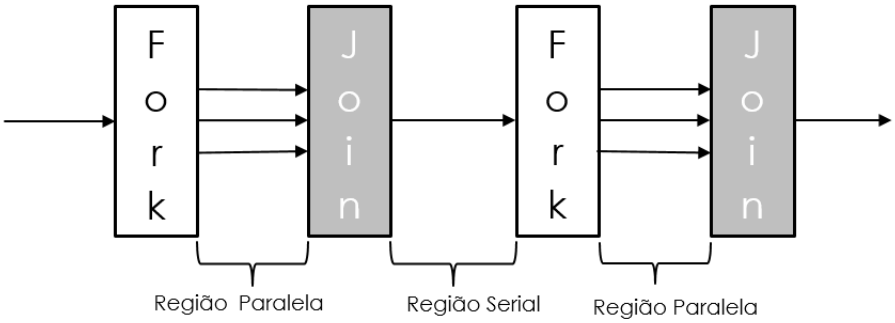
    1. **Cruzamento:** A operação de cruzamento elege dois cromossomos para produzirem um novo indivíduo, combinando seus genes com o objetivo de gerar um novo indivíduo diferente (LINDEN, 2012). Existem algumas técnicas de Cruzamento comumente utilizadas como a técnica de 1 ponto, 2 pontos, uniforme e aritmética (MICHALEWICZ, 2013). A técnica de 1 ponto selecionará um ponto de secção entre os cromossomos-pai, intercalando-os e gerando um novo indivíduo. A técnica de 2 pontos é semelhante a técnica de 1 ponto, porém dois pontos de secção são selecionados. A técnica uniforme seleciona aleatóriamente segmentos de genes que irão compor o novo indivíduo. A técnica aritmética, aplica uma função aritmética para selecionar os genes que irão compor o novo indivíduo.
    2. **Mutação:** A operação de mutação é aplicada a uma certa probabilidade e a uma certa porcentagem da população após o cruzamento. Se a probabilidade for alta demais, há o risco de uma convergência prematura e piora da qualidade do indivíduo (LINDEN, 2012; MICHALEWICZ, 2013). Portanto é necessário que se escolha uma porcentagem pequena e uma probabilidade pequena para que a mutação não impacte de forma negativa. De forma semelhante ao cruzamento, existem técnicas comumente utilizadas na mutação: inserção, troca e inversão (MICHALEWICZ, 2013). A técnica de inserção escolhe um gene e o insere em uma posição diferente no cromossomo. A técnica de troca, seleciona dois genes (ou bloco de genes) escolhidos de forma aleatória e os troca de lugar

no cromossomo. A técnica de inversão, inverte as posições de um gene ou do cromossomo inteiro.

2.2 Paralelização com OpenMP

Como já foi comprovado em trabalhos anteriores (ROSA; CERA, 2014; GRESSLER; CERA, 2014), a paralelização do AG aplicado ao problema PRV reduz o tempo de computação necessário, enquanto utiliza de forma mais eficiente os recursos computacionais da arquitetura paralela sob a qual é executado. Esta seção apresenta a API (*Application Programming Interface*) OpenMP, a qual foi escolhida devido à sua simplicidade e flexibilidade para o desenvolvimento de programas paralelos para arquiteturas de memória compartilhada.

Figura 2 – Representação do Modelo *Fork-Join*.



Fonte: Chapman, Jost e Pas (2008)

O OpenMP 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* (CHAPMAN; JOST; PAS, 2008). Além de diretivas, o OpenMP também fornece variáveis de ambiente e funções de biblioteca, com suporte para C, C++ e Fortran sendo usado em programas do tipo SPMD (*Single Proccess Multiple Data*). Conforme descrito por (RAUBER; RÜNGER, 2013), o modo de execução paralelo pode ser SPMD ou dividido em diferentes *tasks* (tarefas) para diferentes *threads*. O padrão surgiu em 1997 e é mantido pela *Architecture Review Board*, sendo inserido em muitos compiladores desde então. Segundo Chapman, Jost e Pas (2008), o OpenMP segue o modelo *Fork-Join*, sendo este demonstrado na Figura 2, onde partes sequenciais do código são executadas por uma *thread* mestre e em regiões paralelas identificadas pelo programador através de diretivas, criam-se (*fork*) um conjunto de *threads* sendo que ao final da computação essas threads são finalizadas (*join*) mantendo-se apenas a *thread* mestre.

O código dentro do construtor paralelo é denominado de região paralela, exatamente pelo fato de ser executado por múltiplas *threads*. As regiões paralelas consistem de regiões do código com grande carga de trabalho onde o processo de paralelização ocorrerá, ou seja, delimita o bloco de código que será executado em paralelo pelas *threads*. Ao final da região paralela existe uma barreira de sincronização, onde apenas a *thread* mestre prossegue. A declaração de uma região paralela no OpenMP utiliza a seguinte sintaxe:

```
# pragma omp parallel [cláusulas [ ] ...]
{
//trecho de código...
}
```

Fazendo alusão ao modelo *Fork-Join* demonstrado na Figura 2, o *fork* acontece na execução da diretiva de paralelização do OpenMP mostrada acima. Já o *join* ocorre ao final do trecho de código ao qual a diretiva será aplicada, mais especificamente no fechamento das chaves que delimitam o bloco. Chapman, Jost e Pas (2008) explicam que o OpenMP trabalha com dois tipos diferentes de memória: a memória privada (identificada pela cláusula `private` e a compartilhada `shared`). Quando a memória é compartilhada, todas as *threads* acessam uma mesma região de memória, enquanto que no caso da memória privada, cada *thread* possui uma cópia dos dados e os acessa individualmente. Com o objetivo de prover independência entre as operações, o OpenMP provê a diretiva `# pragma omp sections`. Esta diretiva permite que haja a distribuição de tarefas para *threads*, fazendo com que elas possam executar diferentes blocos de instruções de maneira paralela. É possível declarar seções paralelas utilizando a seguinte sintaxe:

```
# pragma omp sections [cláusula [cláusula] ...]
{
[# pragma omp section]
{ //trecho de código...}
[# pragma omp section]
{ //trecho de código...}
}
```

Para aumentar o controle da paralelização, o OpenMP provê a diretiva `# pragma omp parallel for`. Esta diretiva é responsável pela definição de um laço paralelo, onde as iterações do laço são distribuídas entre as *threads* da região paralela. O laço paralelo pode ser declarado usando a seguinte sintaxe:

```
# pragma omp parallel for [cláusulas [ ] ...]
{
```

```
for(inicialização; controle; incremento){
//trecho de código...
}
}
```

O laço paralelo também pode receber a cláusula *schedule* que controla a distribuição de iterações do laço entre as *threads*. Esta cláusula ao ser utilizada permite, dentre outras, três políticas de distribuição (ou escalonamento) de iterações:

- **Static:** A distribuição das iterações é realizada de forma estática e em tempo de compilação, priorizando a alocação contígua de sequências de iterações para uma mesma *thread*;
- **Dynamic:** Realiza a distribuição das iterações de forma dinâmica conforme as *threads* concluem a computação de sua carga atual e solicitam uma nova;
- **Guided:** Esta política realiza a distribuição das iterações de forma dinâmica e em tempo de execução. A distribuição das iterações é guiada, começando grande e reduzindo até chegar ao valor definido no bloco.

2.3 Trabalhos Relacionados

Com o intuito de melhor demonstrar o estado da arte sobre AG e explicar nossas tomadas de decisão, uma tabela foi utilizada. A Tabela 1 resume os trabalhos relacionados (apresentados em linhas) em função de suas características (apresentadas em colunas): API utilizada na paralelização, granularidade das tarefas, ganho de desempenho (*speedup*) obtido e qualidade das soluções encontradas.

API consistem de interfaces cujo objetivo é auxiliar o processo de paralelização do AG. Os trabalhos relacionados utilizaram as seguintes APIs: CUDA[1] (*Computer Unified Device Architecture*) destina-se a prover paralelização em GPU (*Graphics Processing Unit*), que são placas gráficas de alto poder de processamento, em especial para operações matemáticas. MPI[2] (*Message Passing Interface*) consiste de um padrão para troca de mensagens, através de diretivas que auxiliam a comunicação entre processos. MatLab[3] (*Matrix Laboratory*) é uma *interface* voltada ao cálculo numérico, permitindo também a paralelização dos programas apenas através da criação de laços paralelos. Pthreads[4] (*POSIX Threads*) é uma biblioteca para linguagem C que permite a criação e manipulação de *threads*. É uma API poderosa e leve, porém possui funções de uso complexo.

[1] Disponível em: <http://www.nvidia.com.br/object/cuda_home_new_br.html> (último acesso em outubro 2015)
[2] Disponível em: <http://www.mpi-forum.org/docs/> (último acesso em outubro 2015)
[3] Disponível em: <http://www.mathworks.com/products/matlab/> (último acesso em outubro 2015)
[4] Disponível em: <https://computing.llnl.gov/tutorials/pthreads/> (último acesso em outubro 2015)

Características/Propostas	APIs Utilizadas	Granularidade	Speedup	Qualidade de solução
Karpinski e Pacut (2014)	CUDA e MPI	Grossa	≈4	35.71% precisão
Yussof et al. (2009)	MPI	Grossa	8,66	91% precisão
Liu, Jiang e Xie (2009)	MatLab	Média	Não Consta	97% precisão
OLIVEIRA (2010)	OpenMP	Grossa	3,6	99,2% precisão
Borovska (2006)	MPI e OpenMP	Grossa	4,5	96% precisão
Silva e PAIVA (2012)	Pthreads e OpenMP	Fina	43,05	68,33% de precisão
Dornelles et al. (2014)	Pthreads e OpenMP	Fina	1,38	Não Consta
Ochi et al. (1998)	MPI	Grossa	≈6,0	55% de precisão

Tabela 1 – Tabela Comparativa dos Trabalhos Relacionados

OpenMP permite a criação e manipulação de *threads* de forma semelhante a Pthreads, porém utiliza `pragmas` para facilitar o seu uso.

Os trabalhos de Karpinski e Pacut (2014) e Borovska (2006) utilizam duas APIs de paralelização em função dos seus trabalhos serem orientados a *clusters* de computadores. Em ambos os trabalhos a API MPI é utilizada para prover comunicação entre as máquinas, enquanto a outra API explora o potencial local das arquiteturas. O trabalho de Karpinski e Pacut (2014) utiliza CUDA para realizar a paralelização do seu AG na GPU, onde cada GPU utilizada pelo trabalho possuía 448 núcleos. Já Borovska (2006) utiliza OpenMP, pelo foco de sua paralelização ser uma arquitetura *multicore*, seu trabalho utiliza 5 *Workstations* cujas configurações não foram informadas. De maneira similar, os trabalhos de Yussof et al. (2009)e Ochi et al. (1998) possuem como arquitetura alvo *clusters* e utilizaram a MPI para realizar a paralelização de seus trabalhos. O trabalho de Yussof et al. (2009) utiliza 6 máquinas com 2 núcleos físicos cada, enquanto o trabalho de Ochi et al. (1998) utilizou 4 máquinas cada uma com um núcleo físico.

Como a arquitetura foco do nosso trabalho consiste em uma *Workstation* com memória compartilhada e não um *cluster* de computadores com memória distribuída e a paralelização será feita em nível do processador e não da GPU, as APIs CUDA e MPI não foram utilizadas. Silva e PAIVA (2012) e Dornelles et al. (2014) realizaram estudos comparativos entre Pthreads e OpenMP com o intuito de verificar as diferenças entre as APIs. Silva e PAIVA (2012) em seu trabalho utilizou duas arquiteturas, uma com 2 núcleos físicos e outra com 4 núcleos. Em ambos os trabalhos as duas APIs demonstraram desempenhos parecidos, e como OpenMP possui uma simplicidade maior de uso, sem sacrificar seu poder como API de paralelização, foi optado por mantê-la nesse trabalho (GRESSLER, 2012).

A **Granularidade** representa a relação entre a quantidade de processamento e de comunicação necessária para computar partes de um programa paralelo (FOSTER, 1995). A granularidade pode ser fina - representando grãos com tempos de computação pequenos em relação a comunicação demandada; grossa - onde a computação demanda uma grande

parte do tempo; e média - que consiste de um meio-termo entre granularidade grossa e fina. É possível ver na tabela 1 que os trabalhos que utilizaram *clusters* ((OCHI et al., 1998), (KARPINSKI; PACUT, 2014), (BOROVSKA, 2006)) possuem uma granularidade grossa, dado o modo como a paralelização em *clusters* é realizada. Analizando a tabela 1, existem muitos trabalhos que utilizam granularidade grossa, mesmo para arquiteturas que não são *clusters*, isso se deve ao fato de que com granularidade muito fina podemos ter grãos demais para computar em tempo suficientemente hábil, uma vez que toda computação depende de etapas de criação, comunicação e sincronização. Por isso neste trabalho será utilizada granularidade grossa.

O ***Speedup*** denota quantas vezes a versão paralela conseguiu reduzir o tempo de execução em relação ao tempo sequencial (CHAPMAN; JOST; PAS, 2008). O maior *speedup* foi obtido por Silva e PAIVA (2012) em seu trabalho de comparação entre APIs de paralelização. Este *speedup* foi obtido pela API Pthreads, porém a API OpenMP não ficou muito atrás nos *speedups* obtidos. Em alguns casos a API OpenMP obteve *speedups* melhores que a API Pthreads. No trabalho de Dornelles et al. (2014) a API que obteve o melhor *speedup* foi a API OpenMP, neste trabalho a diferença entre as duas APIs foi significativa, porém Dornelles et al. (2014) afirma que pode ter sido em função da arquitetura usada. Borovska (2006) também obteve um bom *speedup* ao utilizar a API de paralelização OpenMP e em função destes resultados, a API OpenMP será utilizada no trabalho.

A qualidade de solução demonstra a porcentagem de soluções ótimas encontrada pelo algoritmo testado. Com o intuito de melhorar a qualidade de solução, vários trabalhos ((KARPINSKI; PACUT, 2014), (BOROVSKA, 2006), (LIU; JIANG; XIE, 2009), (OLIVEIRA, 2010))optaram pelo uso de hibridização de AGs. AGHs (Algoritmo Genético Híbrido) consistem da união entre AGs e uma outra heurística ou meta-heurística (LINDEN, 2012). Com o objetivo de melhorar a eficiência do AG, AGHs são muito utilizados e difundidos pela literatura, sendo em maior parte utilizados para suprir as desvantagens naturais do AG (KONFRŠT, 2004); (CRAINIC, 2008); (MUNAWAR et al., 2008). Porém as desvantagens deste método são o alto custo computacional e o grande conhecimento do problema em questão para definir a etapa do AG em que a metaheurística será inserida e a escolha para melhor otimiza-lo (CRAINIC, 2008). É possível ver na tabela 1 que nem todos os trabalhos que utilizaram hibridização apresentaram uma melhoria significativa em suas qualidades de solução, como o trabalho de (KARPINSKI; PACUT, 2014). Enquanto os que apresentaram uma melhoria significativa, acabaram aumentando consideravelmente seus tempos de execução sequencial, recorrendo à paralelização e arquiteturas em *clusters* para executar seus códigos em tempo hábil.

Santos (2009), menciona que AGHs são geralmente utilizados em arquiteturas com memória distribuída, como a arquitetura a ser utilizada no trabalho trata-se de uma

Workstation com memória compartilhada, é defendido o uso de um AG baseado no modelo de ilhas, proposto por (OCHI et al., 1998). O modelo de ilhas utiliza uma abordagem diferente ao AG, de forma que ao invés de uma população, como a maioria das propostas, eles utilizaram de múltiplas populações, Ochi et al. (1998) nomeia cada população como ilhas, todas independentes. Em seu trabalho Ochi et al. (1998) conseguiram obter melhores métricas de *speedup* e qualidade de solução em relação a execução convencional do AG utilizada para comparações. O trabalho de Silva e PAIVA (2012) fez uso do Modelo de Ilhas proposto por Ochi et al. (1998) e conseguiu obter os maiores *speedups* entre os trabalhos relacionados, comprovando a eficácia do Modelo de Ilhas. Conforme demonstrado na Tabela 1 todas as propostas apresentadas utilizam de diferentes APIs de paralelização e enfatizam a paralelização como um método poderoso de melhorar o desempenho de AGs.

2.4 Balanço do Capítulo

Este capítulo destinou-se a contextualizar os aspectos necessários para a compreensão do trabalho realizado. Foi apresentado o problema alvo, a técnica utilizada para a resolução do mesmo e o método de paralelização empregado. Também foram apresentadas as características do Problema de Roteamento de Veículos (PRV) o qual é o problema alvo de nosso trabalho, a meta heurística Algoritmo Genético (AG) que foi utilizada para encontrar soluções ao PRV e as principais funcionalidades do OpenMP, que é a IPP escolhida para a paralelização do AG. Adicionalmente, os trabalhos relacionados que foram apresentados neste capítulo enfatizam o uso da paralelização para melhorar o desempenho de AGs, onde pode-se observar que esta é uma alternativa que além de reduzir o tempo de execução dos AGs, não prejudica a qualidade das soluções encontradas. No próximo capítulo será demonstrado o AG implementado, como foi paralelizado e as diferenças da nova proposta de AG implementada.

3 Paralelização do Algoritmo Genético

Este capítulo se destina a explicar quanto a implementação e ao processo de paralelização dos AG feito por Gressler e Cera (2014) e a nova versão do AG aplicado ao modelo de ilhas. Para encerrar o capítulo será demonstrado como foi feita a implementação desta nova proposta do AG de Gressler e Cera (2014) e sua paralelização utilizando OpenMP.

3.1 Algoritmo Genético Convencional

Essa seção destina-se a demonstrar o AG implementado por Gressler e Cera (2014). Também será demonstrado como foi realizada a paralelização do AG usando OpenMP.

3.1.1 Algoritmo Genético Sequencial

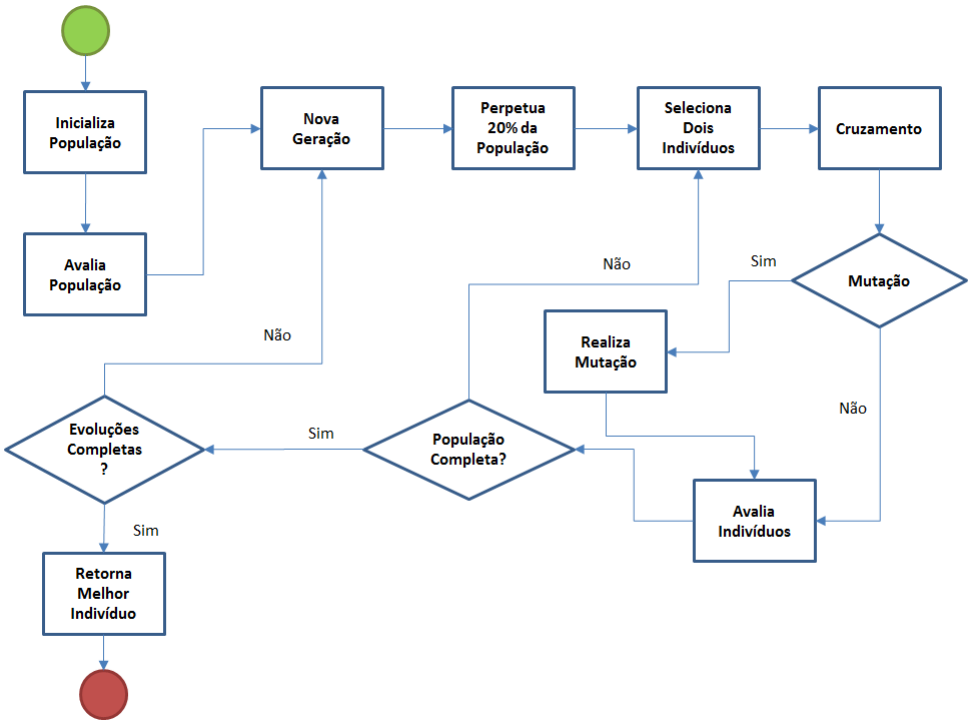
O AG implementado por Gressler e Cera (2014) seguiu o modelo clássico de implementação desta meta-heurística. Nela, os indivíduos consistem de percursos entre cidades (rotas) que solucionariam o PRV, sendo uma população um conjunto de indivíduos ou soluções do problema. O conjunto de cidades a ser percorrido é chamado de cromossomo do indivíduo, sendo este composto por genes, que, por sua vez, são cada uma das cidades que compõem cada percurso. A Figura 3 ilustra o fluxograma do AG implementado onde:

- **Inicializa População:** Consiste da inicialização da primeira população da primeira geração do AG. De acordo com Gressler e Cera (2014) os indivíduos da primeira geração são compostos a partir de uma cidade selecionada aleatoriamente e, a partir dela, são selecionadas as cidades mais próximas no percurso que possuam a menor distância possível entre si e que já não tenham sido visitadas;

- **Avalia População:** De forma semelhante ao comportamento natural, o operador de avaliação realiza o papel da seleção dos indivíduos mais aptos em um ambiente. Esta função avalia os indivíduos e determina os mais aptos entre eles. Onde a aptidão simboliza o quão bem a solução resolve o problema alvo. No caso do problema alvo, a avaliação ordenará os indivíduos sendo que os melhores são aqueles que possuem o menor custo total de percurso, ou seja, a menor distância percorrida;

- **Nova Geração e Perpetuação de 20% da População:** Estas etapas realizam o início da nova geração do AG, perpetuando 20% da população atual. Entre estes 20%, 10% são compostos pelos melhores indivíduos da geração atual, selecionados

Figura 3 – Fluxograma do AG implementado.



pela função de avaliação enquanto outros 10% consistem de indivíduos aleatoriamente selecionados. Esta seleção ocorre sob o propósito de fornecer diversidade genética a nova população. A etapa de Nova Geração prossegue até que o número de evoluções seja alcançado;

- **Cruzamento:** Esta operação realizará a melhoria nas soluções anteriores, utilizando dois indivíduos e combinando segmentos de suas soluções com o objetivo de gerar um novo indivíduo. Para o trabalho, Gressler e Cera (2014) implementou cinco métodos diferentes de cruzamento, sendo 3 métodos básicos conhecidos da Literatura e outros 2 que os combinam, com o intuito de verificar qual deles obteria um maior impacto sobre a solução final do AG. Esta etapa ocorre no AG com o intuito de gerar a nova população e ocorre até que o tamanho da população seja atingido. Os métodos implementados foram:

  1. **Cruzamento 1 ponto:** Neste método, é escolhido um ponto aleatório no cromossomo dos genitores para seccioná-los. Os segmentos dos genitores são intercalados gerando 2 novos indivíduos;

  2. **Cruzamento 2 pontos:** O cruzamento de dois pontos, seleciona dois pontos aleatórios nos cromossomos genitores para seccioná-los, e recombina os segmentos dando origem a 2 descendentes;

3. **Cruzamento Uniforme:** Este método de cruzamento realiza um sorteio de modo a decidir quais genes o indivíduo gerado herdará. O sorteio é feito selecionando gene por gene, onde estes genes são escolhidos para compor os novos indivíduos de modo aleatório;

4. **Híbrida 1:** Este método consiste em escolher aleatoriamente qual das três técnicas acima elencadas deve-se aplicar sobre cada cruzamento;

5. **Híbrida 2:** Este método mantém as técnicas de cruzamento em uma fila, percorrendo entre elas conforme não exista melhoria na solução em N/5 evoluções, onde N é o número de cidades da instância.

- **Mutação:** Esta operação é a responsável por alterar o código genético de um indivíduo, de maneira similar à função que simula na genética. A mutação ocorre sobre cada indivíduo oriundo do cruzamento em uma certa probabilidade. Na mutação as posições de um ou mais genes são trocadas de lugar, assim alterando seu código genético e fornecendo uma outra forma de variedade genética. Gressler e Cera (2014) especificam que foram implementados quatro métodos de mutação no AG, 3 deles conhecidos da literatura e 1 que os combinam, sendo eles:

  1. **Troca (*Swap*):** Este método de mutação pode ocorrer de duas formas: troca de genes e troca de blocos. A troca de genes, seleciona de modo aleatório dois genes no cromossomo e os troca de lugar. A troca de blocos seleciona seções do cromossomo e as troca de lugar;

  2. **Inversão:** A inversão seleciona aleatoriamente um bloco de genes do cromossomo e inverte a posição dos genes dentro deste bloco;

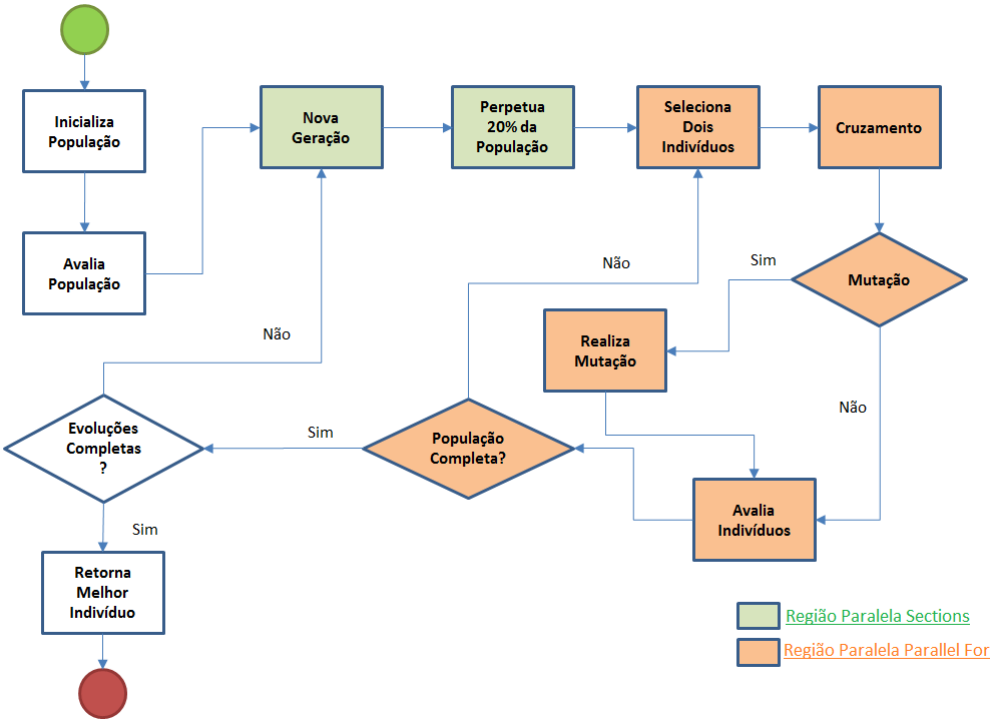
  3. **Inserção:** A inserção realiza a seleção aleatória de um bloco de genes do cromossomo, remove-o e o insere em uma posição diferente no cromossomo;

  4. **Randômica:** Escolhe de modo aleatório uma das 3 técnicas anteriores e as aplica no cromossomo.

- **Avalia Indivíduo:** Realiza a avaliação de cada indivíduo da nova população, ou seja, calcula a distância de cada rota identificando as de menor valor;

- **Retorna Melhor Indivíduo:** Etapa responsável por retornar o indivíduo que melhor se adaptou ao problema através da etapa de avaliação, ou seja, dentre todas as gerações retorna o indivíduo que representa a menor rota de atendimento entre as cidades. Logo, esta etapa ocorre após as gerações terem sido finalizadas e é a etapa que retornará o resultado final da instância do AG.

Figura 4 – Fluxograma da paralelização do AG implementado.

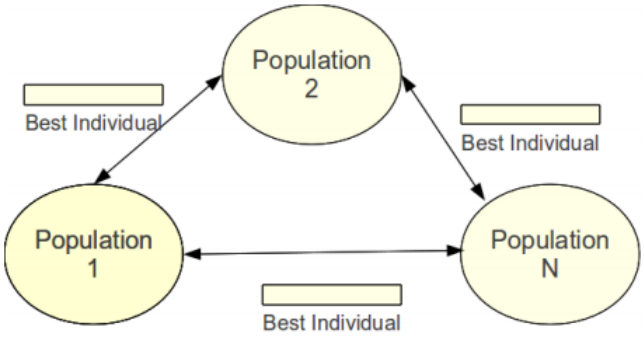


3.1.2 Algoritmo Genético Paralelo com OpenMP

Gressler e Cera (2014) definiu duas regiões paralelas no procedimento que dá origem as novas gerações de indivíduos com base no perfil de execução da versão sequencial (identificação via *profiling* das regiões de maior demanda de processamento). A figura 4 representa as regiões paralelas criadas por Gressler e Cera (2014). A primeira região paralela, destacada em verde na figura 4, envolve a perpetuação dos indivíduos da população atual: 10% dos melhores e 10% de indivíduos escolhidos aleatoriamente. Estas duas operações não possuem dependência de dados e podem ser executadas concomitantemente. Dada a característica desta região do código, foram utilizadas `# pragma omp sections` em sua paralelização.

A segunda região paralela, representada por laranja na figura 4, corresponde a aplicação dos operadores genéticos Cruzamento e Avaliação na geração de novos indivíduos até que uma nova população esteja completa. A geração dos novos indivíduos compreendem iterações de um laço `for`. Como não existem dependências entre as iterações deste laço, Gressler e Cera (2014) optaram pelo uso do `# pragma omp for`. Entretanto, cabe ressaltar que este procedimento será repetido até que um número determinado de evoluções aconteçam, sendo que a população gerada como saída será o ponto de partida para a geração de uma nova população. Logo, devido a esta dependência a paralelização de iterações de laços aplica-se apenas para a geração de indivíduos e não para as evoluções.

Figura 5 – Representação do Modelo de Ilhas



Fonte: Guillén et al. (2014)

3.2 Algoritmo Genético seguindo o Modelo de Ilhas

Uma nova implementaçao do AG foi realizada, tomando como base a proposta de Ochi et al. (1998), que consiste em utilizar múltiplas populações para melhorar a solução final do AG. Esta nova implementação tomou como base o AG implementado por Gressler e Cera (2014), extendendo o e studo realizado por ele e objetivando alcançar soluções melhores, obter melhores *speedups* e uma melhor eficiência do algoritmo.

O Modelo de Ilhas (*Island Model*) consiste de uma proposta feita por Ochi et al. (1998), cuja premissa baseia-se no comportamento adaptativo que diferentes espécies demonstraram quando separadas em ilhas independentes, como por exemplo: aves em diferentes ilhas são diferentes umas das outras, embora possuam o mesmo antepassado. Aplicando esta premissa computacionalmente, Ochi et al. (1998) implementou uma versão do AG, onde são criadas múltiplas populações independentes, cada qual desempenhando as funções padrão do AG padrão. A Figura 5 demonstra o modelo tradicional de ilhas, onde possuímos N populações e cada população pode comunicar seu melhor indivíduo para a outra. Esta comunicação se chama de migração, e pode ou não ocorrer no modelo de ilhas. Whitley, Rana e Heckendorn (1999) realizaram testes com e sem o operador de migração e perceberam que sem o operador as populações ainda assim conseguem obter melhores soluções, porém ao aplicar o operador de migração a média das soluções é melhor. Por mais que o objetivo do operador de migração seja produzir uma maior diversidade genética, ele pode acabar removendo a diversidade genética da população. Como a finalidade do trabalho consiste em uma melhoria em *speedup*, não será implementado um operador de migração no algoritmo neste primeiro momento, visto que a migração apenas auxilia a média final das soluções obtidas pelas ilhas, as ilhas serão completamente isoladas entre si.

3.2.1 Versão Sequencial

A versão sequencial complementa a implementação feita por Gressler e Cera (2014), adicionando um novo laço iterativo que permite a geração de várias populações ao invés de apenas uma. O Pseudocódigo 1 demonstra a implementação do novo modelo. O algoritmo começa recebendo o número de ilhas a serem geradas, após isso cada ilha irá reproduzir as operações presentes no código de Gressler e Cera (2014) (linhas 4 a 30 do pseudocódigo 1), permitindo assim um maior conjunto de possíveis soluções para o AG. Com isso cada ilha conterá uma versão do AG original de Gressler e Cera (2014). Após cada ilha desempenhar e avaliar cada um dos indivíduos gerados, ocorre a comparação entre os indivíduos para selecionar o melhor indivíduo entre todas as ilhas, conforme demonstrado nas linhas 31 a 33 do pseudocódigo 1, este indivíduo será então escolhido como a melhor solução do problema conforme mostrado na linha 34 do pseudocódigo 1. Como esperado, uma implementação destas obterá um tempo de execução muito maior do que a versão anterior do AG, para resolver isso, foram implementadas duas versões do código: **Versão 1** e **Versão 2**. Estas duas versões irão diferir no número máximo de gerações que será aplicada em cada ilha. A Versão 1 irá desempenhar o número padrão de gerações conforme estipulado por Gressler e Cera (2014), este sendo NxNx10, onde N é o número de indivíduos. Para a segunda versão foi dividido a quantidade de gerações pelo número de ilhas totais, gerando assim uma fração menor de gerações por ilha e reduzindo o tempo total do AG. Estas duas versões foram feitas com o objetivo de observar o impacto no *speedup* e qualidade de solução. Como a única diferença entre as duas versões é o valor da variável que controla o número de evoluções que cada população crescerá, o Pseudocódigo 1 é válido para ambas as versões.

3.2.2 Versão paralela com OpenMP

A paralelização foi feita conforme denotada pela figura 6. O primeiro passo da paralelização foi definir uma API de paralelização, como este trabalho extende o trabalho de Gressler e Cera (2014), optou-se por manter o OpenMP como ferramenta de paralelização. Próxima etapa foi definir a abordagem de paralelização e a criação do bloco paralelo. Como nosso possui um laço de repetição que concentra maior parte da carga de processamento, conforme denotado na linha 3 do pseudocódigo 1, foi escolhido utilizar a diretiva `pragma omp parallel for`. Criando a região paralela destacada em laranja na figura 6. Próxima etapa consistiu em analizar o compartilhamento de variáveis. A cláusula `private` foi usada para garantir o acesso exclusivo das *threads* às variáveis, como as variáveis que compõem cada ilha e o custo final retornado pela ilha, a cláusula `shared` foi usada para aquelas variáveis cujos valores eram compartilhados por todas as *threads*.

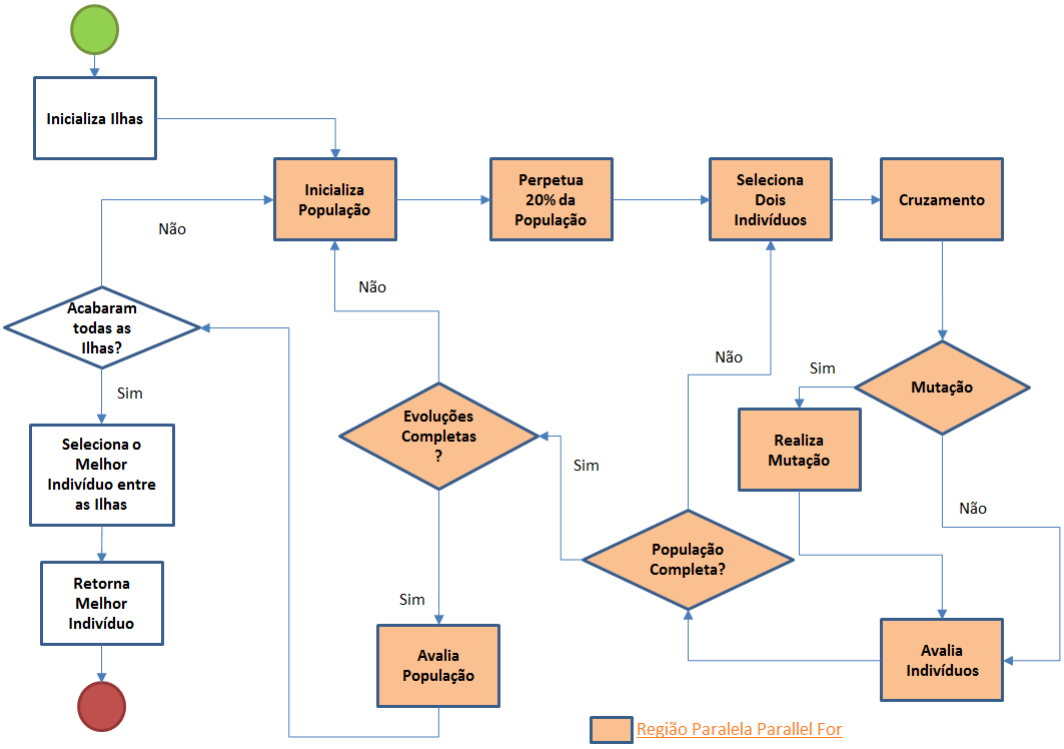
Algorithm 1 Implementação do Algoritmo Genético Sequencial Seguindo Modelo de Ilhas

```
1: Início
2: Inicializa Arquipélago
3: while (Houverem ilhas) do
4:     Gera população inicial de cada ilha
5:     Avalia todos os indivíduos
6:     while ( !Terminou ) do
7:         Copia indivíduos perpetuados para Nova Geração
8:         for (1/2 da taxa de cruzamento) do
9:             if (Iterador menor que 1/4 da taxa de cruzamento) then
10:                Seleciona 1 indivíduo entre os melhores
11:                Seleciona 1 indivíduo aleatoriamente
12:                Cruza os indivíduos selecionados
13:                if (Com certa probabilidade) then
14:                    Aplica mutação em parte dos genes de 1 ou 2 descendentes
15:                end if
16:                Avalia indivíduos gerados
17:            else
18:                Seleciona 2 indivíduos aleatoriamente
19:                Cruza os indivíduos selecionados
20:                if (Com certa probabilidade) then
21:                    Aplica mutação em parte dos genes de 1 ou 2 descendentes
22:                end if
23:                Avalia indivíduos gerados localmente em cada ilha
24:            end if
25:        end for
26:        if (Número de evoluções foi atingido de cada ilha) then
27:            Terminou ← Verdade
28:        end if
29:    end while
30: end while
31: while (Houverem ilhas) do
32:     Compara melhores soluções de cada ilha
33: end while
34: Retorna Melhor Solução Encontrada
35: Fim
```

3.2.3 Balanço do Capítulo

Neste capítulo foram apresentadas as diferenças entre a nova versão do AG em relação a proposta anterior de Gressler e Cera (2014), além de apresentar a proposta de ilhas e o processo de paralelização da mesma. Com o objetivo de extender a apresentação foi incluído o pseudocódigo da nova proposta e fluxogramas demonstrando as regiões que foram paralelizadas. Para o próximo capítulo serão incluídos os parâmetros dos testes realizados para ambas as versões do AG.

Figura 6 – Fluxograma representando a implementação do Modelo de Ilhas



4 Metodologia

Neste capítulo serão apresentadas a infraestrutura de teste (Seção 4.1), a calibragem de parâmetros do AG e de sua paralelização (Seção 4.2), a organização da coleta de dados (Seção 4.3) e finalizando com os principais aspectos do capítulo (Seção 4.4).

4.1 Infraestrutura de Teste

Foi utilizado uma *Workstation* da Universidade Federal do Pampa (UNIPAMPA). Ela possui um processador Intel®Xeon E5-2609 v3 com frequência de 1.90 GHz, com seis núcleos, com 15 MB de memória cache e 16 GB de memória principal. A *Workstation* possui sistema operacional Ubuntu 16.04 LTS e a versão do compilador GCC é a 5.31.

4.2 Calibragem de Parâmetros do AG

Esta seção mostrará os parâmetros que foram utilizados nos testes, sendo as instâncias alvo na Seção 4.2.1, a configuração do próprio AG na Seção 4.2.2 e os parâmetros utilizados pela paralelização com OpenMP na Seção 4.2.3.

4.2.1 Instâncias Utilizadas

Para que o AG possa ser aplicado ao PRV, é preciso fornecer a ele o conjunto de cidades a serem percorridas, o que é nominado de instância do problema. O *benchmark* de Taillard (2015) , contém instâncias de 50 até 199 cidades, consistindo de arquivos de texto que contém informações das cidades, como cidades mais próximas e o custo do percurso para cada cidade, além da melhor solução conhecida para o problema. As instâncias utilizadas do *benchmark* nesta fase do trabalho foram a c50, c100, c120 e c150 que respectivamente são compostas por 50, 100, 120 e 150 cidades. A instância c50 foi escolhida por ser uma instância pequena e que necessitaria de um custo computacional pequeno para solucioná-la. As instâncias c100, c120 e c150 foram escolhidas por serem instâncias maiores e demandarem um maior custo computacional, assim podendo observar melhor o comportamento do AG.

4.2.2 Parâmetros do Algoritmo Genético

O trabalho de Gressler e Cera (2014) identificou os parâmetros do AG capazes de fornecer o melhor desempenho, os quais serão utilizados neste trabalho, dentre estes apenas o número de evoluções será alterado na nova abordagem. Último parâmetro é

o número de ilhas e para esta abordagem considerou-se melhor utilizar uma ilha para cada *core* físico e lógico presente na arquitetura a ser utilizada, baseando-se na decisão tomada por Ochi et al. (1998) e Yussof et al. (2009), onde em cada caso foi utilizado o mesmo número de máquinas presentes no *cluster* como o número de populações. Para a finalidade do trabalho o AG não será aplicado a um *cluster* de computadores, se utilizará o número de núcleos físicos e lógicos presentes na arquitetura como número total de ilhas, se tratando da *Workstation*, que possui 6 núcleos físicos em seu processador, totalizando 12 ilhas isoladas para melhor observar melhor o aumento no *speedup* e qualidade de solução do AG. Os parâmetros estão descritos abaixo:

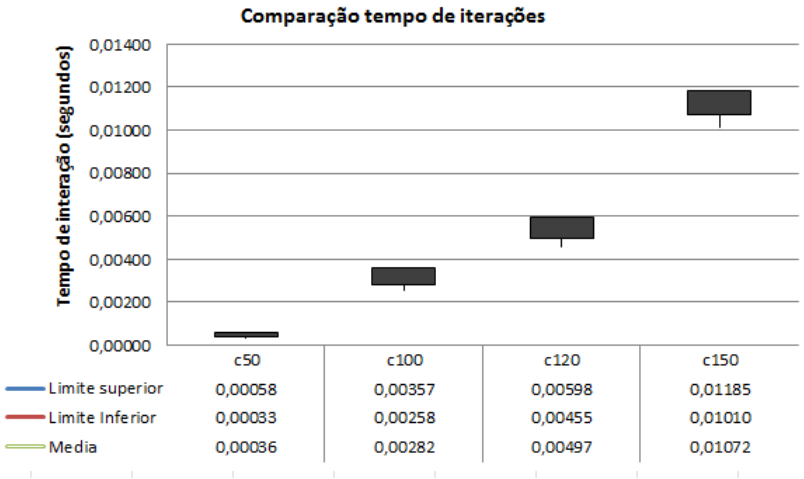
- **Tamanho da População:** $N \times 10$, onde N é o número de cidades envolvidas no problema;
- **Número de Evoluções:** $\frac{N \times N \times 10}{\alpha}$ evoluções, onde α simboliza o número de ilhas independentes para a versão 1 do AG aplicado ao modelo de ilhas e $N \times N \times 10$ para a versão 2;
- **Técnica de Cruzamento:** cruzamento de 1 ponto;
- **Taxa de Mutação:** uma variação de 4 a 10%;
- **Técnica de Mutação:** randômica;
- **Número de Ilhas:** 12 ilhas independentes;

4.2.3 Parâmetros da Paralelização com OpenMP

Para a paralelização foram escolhidos como parâmetros números de *threads* de 2, 4 e 6, valores decididos conforme o número máximo de *cores* do processador, pois trata-se de um processador sem *Hyper-threading*.

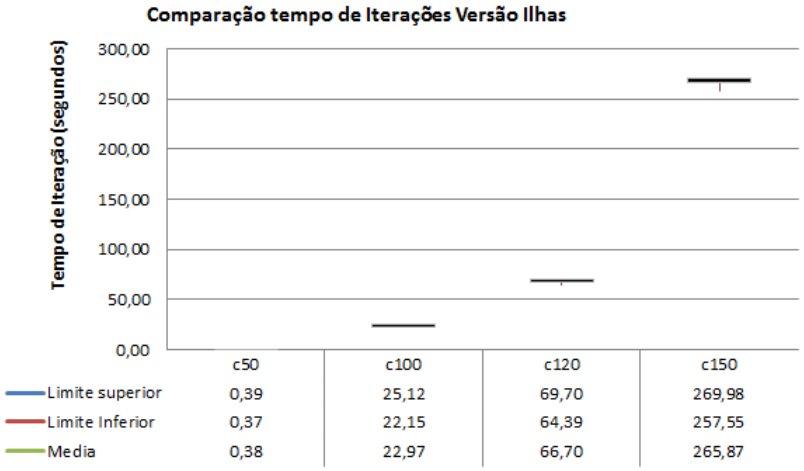
Em ambas as arquiteturas, o número de *threads* foi escolhido de maneira a usar o quantitativo de núcleos disponíveis e inclusive sobrecarregando-os com um número maior de *threads* do que núcleos. Para a política de escalonamento de iterações foi utilizado o valor *static* conforme verificado em trabalhos passados por Rosa e Cera (2015) e Andrade e Cera (2016). Para melhor entender o uso do *static* foram feitos gráficos para demonstrar a variação entre os tempos das iterações, (Figura 7). Para a versão de Gressler e Cera (2014) cada iteração consiste do laço que controla a criação de cada geração do AG. É possível ver que existe uma variação bem pequena entre os tempo máximos, mínimos e médios de cada iteração, com isso demonstra-se que cada iteração leva mais ou menos o mesmo tempo, com isso solidificando a escolha de *static* para a política de distribuição de iterações para a versão de Gressler e Cera (2014). O mesmo teste foi feito para a versão de ilhas do AG.

Figura 7 – Gráfico representando o tempo que cada iteração leva na versão de Gressler e Cera (2014)



A figura 8 demonstra a variação entre o tempo das iterações para a versão do AG de ilhas. Neste gráfico a diferença entre cada instância é maior em função de que cada iteração no modelo de ilhas consiste exatamente em uma versão sequêncial do AG de Gressler e Cera (2014). Porém mesmo assim é possível ver que a variação entre os tempos máximo, mínimo e médio de cada iteração é bem pequeno, mantendo o mesmo comportamento observado pelas iterações do AG de Gressler e Cera (2014). Com isto a decisão de utilizar *static* como parâmetro para a política de distribuição de iterações permanece defendido.

Figura 8 – Gráfico representando o tempo que cada iteração leva na versão de Ilhas



4.3 Organização da Coleta de Dados

Os testes foram executados 8 vezes para cada valor de *thread* em cada instância. Com os valores obtidos foi calculada a média do tempo de execução, tendo esta média obtido um desvio padrão inferior a 1%. Além do tempo de execução, também foi calculado o *speedup* obtido para cada instância conforme o número de *threads* aumenta e a qualidade da solução retornada por cada execução.

4.4 Principais Aspectos

Este capítulo definiu a metodologia do trabalho: infraestrutura de testes, as instâncias utilizadas (c50, c100, c120 e c150), os parâmetros do AG, os parâmetros do OpenMP e a organização e tratamento dos dados coletados. O próximo capítulo fará uma análise do desempenho e da qualidade de solução do AG.

5 Análise de Resultados

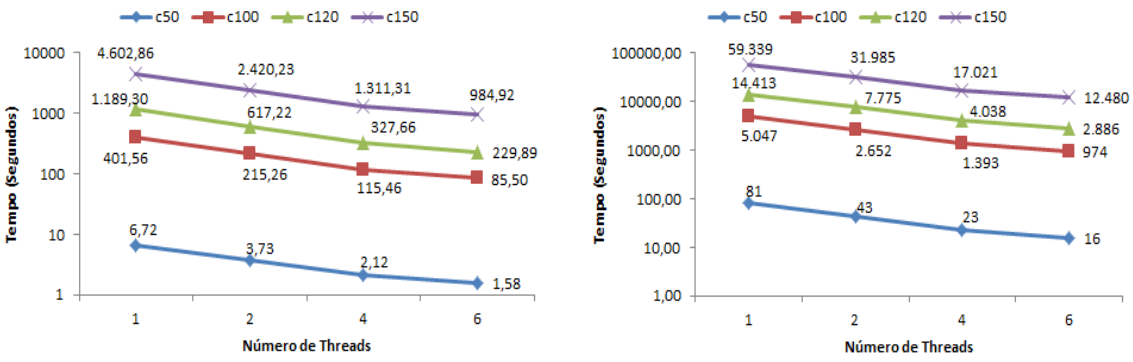
Este capítulo irá demonstrar a análise dos resultados obtidos nos testes descritos conforme o capítulo anterior 4 e compará-los com os resultados obtidos nos trabalhos de Rosa e Cera (2015) e Andrade e Cera (2016). A análise dos resultados ocorreu em três etapas. A primeira comparou o desempenho das instâncias testadas, levando em consideração os tempos de execução e seus *speedups*. A segunda analisou a eficiência do AG paralelo na utilização dos recursos disponíveis. A terceira etapa analisou a qualidade de solução obtidas ao aumentar-se o grau de paralelismo.

5.1 Análise do Desempenho

5.1.1 Analise da Implementação do Modelo de Ilhas

A Figura 9 apresenta o gráfico de tempo de execução para as Versões 1 (9a) e 2 (9b) da implementação com ilhas, enquanto a Figura 10 apresenta seus *speedups*, 10a para a Versão 1 e 10b para a Versão 2.

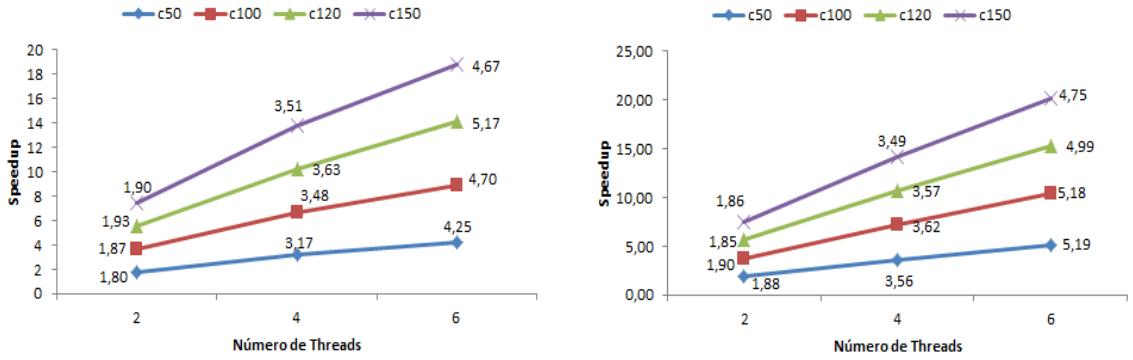
Pode-se observar na Figura 9 que o menor tempo de execução para todas as instâncias ocorreu com 6 *threads*. Isto significa que estas instâncias conseguiram seu máximo desempenho utilizando todos os núcleos físicos disponíveis no processador. É possível ver que na Versão 2 (10b), as instâncias c50 e c100 obtiveram os maiores *speedups*, consistindo de 5,19 e 5,18 respectivamente. Também foi possível observar que a média entre os *speedups* obtidos pela Versão 2 (10b) é maior em relação a Versão 1 (10a). A causa deste comportamento foi a maior carga computacional na Versão 2 em função do maior número



(a) Versão 1

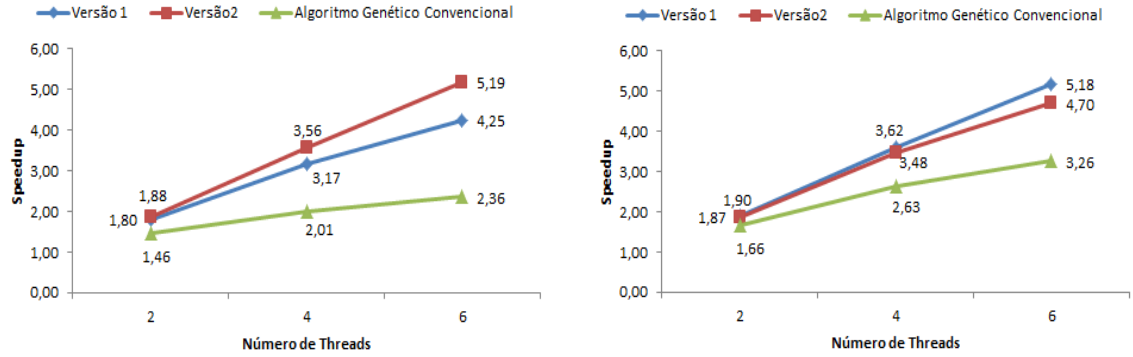
(b) Versão 2

Figura 9 – Tempos de execução (s) para as Versões 1 e 2 do Modelo de Ilhas



(a) Versão 1 (b) Versão 2

Figura 10 – *Speedup* das instâncias para as Versões 1 e 2 do Modelo de Ilhas



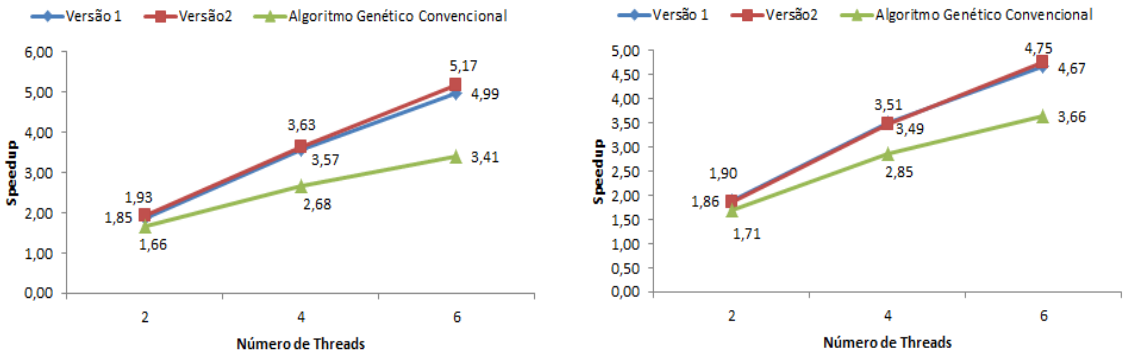
(a) Instância c50 (b) Instância c100

Figura 11 – Comparação entre os *speedups* obtidos pelas Versões da Implementação de ilhas com os obtidos por Andrade e Cera (2016), para as instâncias c50 e c100

de gerações, onde a Versão 1 executa apenas $\frac{N \times N \times 10}{\alpha}$ evoluções, contra $N \times N \times 10$ evoluções da Versão 2. Porém mesmo obtendo *speedups* melhores, a Versão 2 sofreu de tempos de execução em média 12 vezes maiores aos obtidos pela Versão 1.

5.1.2 Comparação do Desempenho entre Modelo de Ilhas e Implementação Convencional

Em seu trabalho Andrade e Cera (2016) realizaram testes com as instâncias c50, c100 e c120 e c150 explorando a escalabilidade até 8 *threads* na mesma arquitetura, utilizando a versão do AG de Gressler e Cera (2014). Segundo Andrade e Cera (2016) a arquitetura utilizada nos testes foi a que demonstrou melhor desempenho na execução do AG Convencional. A Figura 11 mostra uma comparação entre os *speedups* obtidos pelas duas Versões Modelo de Ilhas e os *speedups* de por Andrade e Cera (2016) para as



(a) Instância c120 (b) Instância c150

Figura 12 – Comparação entre os *speedups* obtidos pelas Versões da Implementação de ilhas com os obtidos por Andrade e Cera (2016), para as instâncias c120 e c150

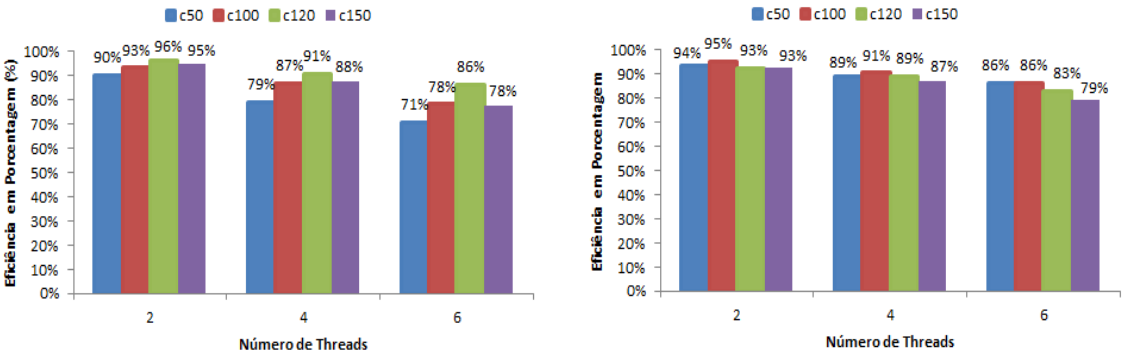
instâncias c50 (11a) e c100 (11b), enquanto a Figura 12 apresenta os *speedups* para as instâncias c120 (12a) e c150 (12b).

Conforme as Figuras 11 e 12, é possível ver que mesmo para instâncias com uma menor carga computacional,como a instância c50, onde o tempo de execução sequencial foi de apenas 6,72 segundos e 81,05, para, respectivamente, a Versão 1 e Versão 2 da proposta, ambas versões do Modelo de Ilhas conseguiram conseguiram melhorar seu *speedup* se comparado aos *speedups* obtidos por Andrade e Cera (2016) em seu trabalho. Logo, as tarefas geradas possuem uma granularidade maior, minimizando impacto do *overhead* da geração e comunicação entre as *threads*, solucionando assim o problema que a implementação convencional do AG possuia. Nesta comparação foi possível confirmar que a aplicação do modelo de ilhas conseguiu uma melhoria no desempenho, tornando os *speedups* mais próximos do *speedup* ideal.

Nas Figuras 11 e 12 nota-se que a Versão 2 atingiu uma melhor média de *Speedups* em relação à Versão 1. Isto se deve ao fato da Versão 2 possuir uma granularidade muito maior em relação a Versão 1, porém conforme a Figura 9 demonstra que essa maior granularidade também implica em um aumento nos tempos de execução para a Versão 2 em relação a Versão 1. Na próxima seção será analisada a eficiência das duas versões do Modelo de Ilhas.

5.2 Análise da Eficiência

A eficiência de um algoritmo indica o quanto dos recursos computacionais ao qual foi aplicado, o algoritmo realmente usa com eficácia. Ela é razão entre *speedup* e o número de *threads* utilizado. A Figura 13 apresenta a eficiência do AG paralelo para as instâncias



(a) Versão 1 (b) Versão 2

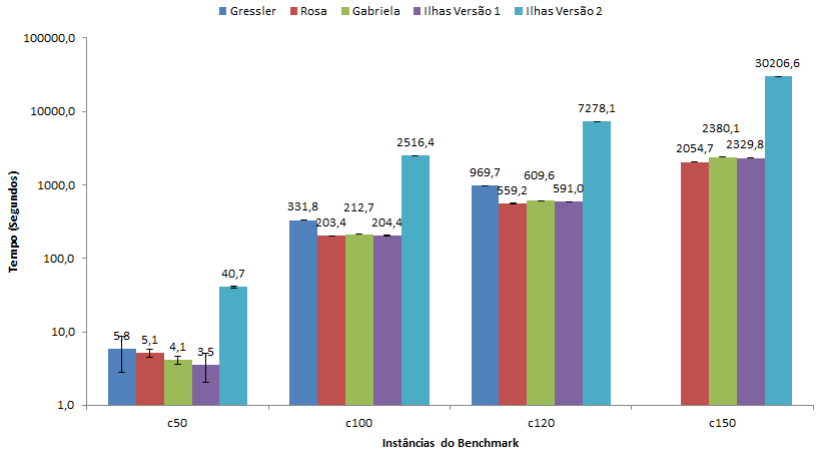
Figura 13 – Eficência das instâncias alvos para as duas Versões 1 e 2 do Modelo de Ilhas

c50, c100, c120 e c150, pelo número de *threads*, para a Versão 1 (13a) e para a Versão 2 (13b). Pode-se idenficar que quanto maior o número de *threads*, menor a eficiência obtida pelo AG paralelo. Isto ocorre pelo impacto das operações de criação, comunicação, sincronização e distribuição da carga de trabalho entre as *threads*, as quais reduzem a eficiência do algoritmo. Adicionalmente, o AG possui uma parcela de computação serial a qual impede que se obtenha uma eficiência de 100%. Isso é evidenciado na Figura 13, onde a porcentagem de eficiência do AG com 2 *threads* começa abaixo de 100% e cai conforme mais *threads* são utilizadas. Como a eficiência é calculada a partir do *speedup*, ela acompanha a seguinte tendência : a pior eficiência foi com a instância c150, eficiência de 79% e a melhor com a instância c100, eficiência de 95% na Versão 2, enquanto na Versão 1 a melhor eficiência foi observada na instância c120 e a pior eficiência na instância c50.

De forma geral, a eficiência do AG foi bem alta, utilizando quase completamente os recursos computacionais e sempre acima de 70% de eficiência. Vale ressaltar que a média de eficiência da Versão 2 do Modelo de Ilhas é maior que a média da Versão 1 em função da maior carga computacional envolvida em cada execução. Com esta análise é possível ver que a Versão 2 mostrou-se melhor que a Versão 1 em função de eficiência, porém assim como em relação ao *speedup*, a diferença entre ambas versões é bem pequena. Na próxima seção será realizada uma comparação de qualidade de solução com intuito de verificar qual é a melhor versão para o Modelo de Ilhas.

Ao comparar a eficiência obtida pelas Versões do Modelo de Ilhas com os resultados obtidos por Andrade e Cera (2016), onde a melhor eficiência foi obtida pela instância c150, obtendo 85% de eficiência e a pior obtida pela instância c50, com 11% de eficiência. Conforme o trabalho de Andrade e Cera (2016) é possível observar que ambas Versões do Modelo de Ilhas conseguiram uma eficiência maior em relação ao AG Convencional.

Figura 14 – Comparação do tempo de trabalhos passados.



5.3 Qualidade da Solução

A Tabela 2 mostra as melhores soluções encontradas por Gressler e Cera (2014) (1^a linha), as melhores soluções encontradas por Rosa e Cera (2015) (2^a linha), as melhores soluções conhecidas para as instâncias Taillard (2015) (3^a linha), as melhores soluções obtidas pela Versão 1 do AG utilizando o modelo de ilhas (4^a linha), as melhores soluções obtidas pela Versão 2 do AG que usa o modelo de ilhas (5^a linha) e as soluções obtidas por Andrade e Cera (2016) (6^a linha). Nela, é possível ver que as versões 1 e 2 da proposta de ilhas paralela conseguiram soluções de qualidade próxima as encontradas por Gressler e Cera (2014) e Rosa e Cera (2015), conforme provado por Andrade e Cera (2016), a *Workstation* utilizada para os testes provê melhores *speedups* porém não fornece boas qualidades de solução. Conforme a tabela 2 nenhuma solução retornada pelas duas versões do Modelo de Ilhas conseguiram ser melhores que as obtidas por Gressler e Cera (2014) e Rosa e Cera (2015).

Para a instância c50, obteve-se uma piora na qualidade de solução se comparado as soluções obtidas por Gressler e Cera (2014) e Rosa e Cera (2015), porém uma qualidade semelhante a encontrada por Andrade e Cera (2016). Enquanto para a instância c120 houve uma pequena piora na qualidade da solução encontrada – aumento de 8,97 Km para a Versão 2 e 15,62 Km para a Versão 1 em relação a Gressler e Cera (2014). Já para a instância c150 tanto a Versão 1 e 2 conseguiram resultados próximos dos obtidos pelo trabalho de Rosa e Cera (2015), embora ambos os resultados estejam 173,2 Km e 13,77 Km, respectivamente para a Versão 1 e 2, distantes das soluções encontradas por Rosa e Cera (2015).

A implementação de ilhas obteve melhores resultados na instância c100 que os demonstrados pelo trabalho de Andrade e Cera (2016), com uma diferença pequena de

Tabela 2 – Demonstração da qualidade das soluções (em Km).

Instância	*c50*	*c100*	*c120*	*c150*
Melhor solução por Gressler e Cera (2014)	543,19	891,57	1066,92	Não Testado
Melhor Solução por Rosa e Cera (2015)	543,19	876,19	1068,18	1100,15
Melhor solução conhecida	524,61	826,14	1042,11	1028,42
Melhor Solução Ilhas Versão 1	560,12	971,07	1082,57	1273,35
Melhor Solução Ilhas Versão 2	560,12	929,27	1075,89	1113,17
Melhor Solução por Andrade e Cera (2016)	560,12	933,07	1075,89	1232,32

3,8 Km. Quanto aos resultados de Rosa e Cera (2015), talvez a implementação de ilhas consiga obter resultados mais próximos se aplicada a mesma arquitetura. A Versão 1 obteve resultados inferiores à Versão 2 justamente pela diferença do número de evoluções entre as duas versões, onde a Versão 1 executa apenas $\frac{N \times N \times 10}{\alpha}$ evoluções, contra $N \times N \times 10$ evoluções da Versão 2. Porém é possível observar que mesmo obtendo melhores soluções em relação as encontradas pela Versão 1, não é recomendado o uso da Versão 2 do Modelo de Ilhas em função do aumento considerável no tempo de execução, conforme demonstrado pela Figura 14.

A Figura 14 apresenta a média de tempos encontrados por todos os trabalhos presentes na Tabela 2, com seus desvios padrão. Como é possível ver, a Versão obteve os maiores tempos de execução, chegando a tempos em média doze vezes maiores que os encontrados pela Versão 1 e mesmo assim não possuindo grandes diferenças entre *speedup* em relação a mesma. Ressaltando também que para compor a solução, cada versão do AG realiza diferentes comparações. A versão convencional do AG realiza exatamente o número de evoluções, $N \times N \times 10$ em comparações para compor a solução. A Versão 1 do Modelo de Ilhas realiza $(\frac{N \times N \times 10}{\alpha}) \times \alpha$ evoluções, totalizando o mesmo número de comparações da versão convencional do AG. Já a Versão 2 do Modelo de Ilhas realiza $(N \times N \times 10) \times \alpha$ comparações para compor sua solução, realizando mais comparações que a versão convencional do AG para compor suas soluções, evidenciado pelo aumento no tempo de execução da Versão 2 em relação a implementação convencional do AG.

5.4 Balanço do Capítulo

Este capítulo mostrou uma análise completa de desempenho, eficiência e qualidade de solução das duas Versões do AG aplicado ao Modelo de Ilhas e paralelizado. Provando que o Modelo de Ilhas conseguiu não só obter melhores métricas de *speedup* e tempo que os obtidos nos trabalhos de Gressler e Cera (2014), Rosa e Cera (2015) e Andrade e Cera (2016), mas também manteve uma qualidade de solução bem próxima destes trabalhos. Porém, as soluções encontradas ainda estão longe das melhores conhecidas Taillard (2015). Adicionalmente, identificou-se que a Versão 2 da Implementação de Ilhas possui melhores

speedups e retornou soluções mais próximas das soluções obtidas pela implementação convencional do AG que a Versão 1, porém o tempo de cada execução da Versão 2 é muito maior que as execuções da Versão 1.

6 Conclusão

Neste trabalho foi realizada uma nova implementação do AG, extendendo assim o trabalho de Gressler e Cera (2014). Esta nova implementação foi baseada no Modelo de Ilhas conforme a proposta feita por Ochi et al. (1998). Esta proposta foi submetida a testes em uma *Workstation* com um processador de com seis núcleos. Com o objetivo de testar diferentes granularidades, duas versões foram implementadas: a Versão 1 e a Versão 2, onde cada versão difere apenas no número de evoluçoes desempenhadas. A Versão 1 realiza $\frac{N \times N \times 10}{\alpha}$ evoluções, enquanto a Versão 2 exerce $N \times N \times 10$ evoluções. Estas duas versões foram paralelizadas utilizando OpenMP e testadas na *Workstation* com o objetivo de verificar qual destas versões obteriam os melhores resultados.

Os testes mostraram que as as duas versões com o Modelo de Ilhas conseguiram não só obter melhores métricas de *speedup* que os obtidos nos trabalhos de Gressler e Cera (2014), Rosa e Cera (2015) e Andrade e Cera (2016), mas também mantiveram uma qualidade de solução bem próxima destes trabalhos. Porém, as soluções encontradas ainda estão longe das melhores conhecidas Taillard (2015). Nestes testes também foi possível conferir que o melhor *speedup* obtido foi com a Versão 2, conseguindo um tempo de execução 5,19 vezes menor que seu tempo sequencial para a instância c50, quando executada com 6 *threads*.

A Versão 2 também obteve uma melhor média de *speedups* para todas as instâncias testadas. Mas a maior granularidade da Versão 2 implicou em um aumento drástico no tempo de execução em comparação tanto com a Versão 1 quanto a implementação convencional do AG por Gressler e Cera (2014). Este aumento no tempo de execução não justifica seu uso por *speedups* e qualidade de solução levemente melhores se comparados a Versão 1, logo para trabalhos futuros considera-se aplicar a Versão 1 do Modelo de Ilhas na mesma arquitetura dos testes de (ROSA; CERA, 2015), com o intuito de observar possíveis melhorias em *speedup* e qualidade de solução. Adicionalmente considera-se também implementar um operador de migração entre as ilhas para a Versão 1 do AG aplicado ao Modelo de Ilhas, visando garantir uma melhor qualidade de solução independente da arquitetura utilizada.

Referências

ANDRADE, G. L.; CERA, M. C. Avaliando a paralelização de um algoritmo genético com openmp. In: *XVII Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*. [S.l.: s.n.], 2016. p. 68 – 73. Citado 10 vezes nas páginas 15, 25, 46, 49, 50, 51, 52, 53, 54 e 57.

BOROVSKA, P. Solving the travelling salesman problem in parallel by genetic algorithm on multicomputer cluster. In: *Int. Conf. on Computer Systems and Technologies*. [S.l.: s.n.], 2006. p. 1–6. Citado 2 vezes nas páginas 33 e 34.

CHAPMAN, B.; JOST, G.; PAS, R. V. D. *Using OpenMP: portable shared memory parallel programming*. [S.l.]: MIT press, 2008. Citado 3 vezes nas páginas 30, 31 e 34.

CRAINIC, T. G. *Parallel solution methods for vehicle routing problems*. [S.l.]: Springer, 2008. Citado 2 vezes nas páginas 25 e 34.

DORNELLES, E. et al. Um estudo comparativo de desempenho utilizando programação sequencial vs paralela aplicado em algoritmos genéticos. *Salão do Conhecimento*, v. 2, n. 01, 2014. Citado 2 vezes nas páginas 33 e 34.

FOSTER, I. *Designing and building parallel programs*. [S.l.]: Addison Wesley Publishing Company, 1995. Citado na página 33.

GRESSLER, H. d. O.; CERA, M. C. O impacto da paralelização com OpenMP no desempenho e na qualidade das soluções de um Algoritmo Genético. *Revista Brasileira de Computação Aplicada*, v. 6, n. 2, p. 35–47, 2014. Citado 17 vezes nas páginas 15, 25, 30, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 50, 53, 54 e 57.

GRESSLER, H. O. *Aumentando a eficiência de um Algoritmo Genético através da paralelização com OpenMP*. Dissertação (Graduação em Ciência da Computação) — UNIPAMPA, Alegrete, 2012. Citado 3 vezes nas páginas 27, 28 e 33.

GUILLÉN, A. et al. Fast feature selection in a gpu cluster using the delta test. *Entropy*, Multidisciplinary Digital Publishing Institute, v. 16, n. 2, p. 854–869, 2014. Citado na página 41.

KARPINSKI, M.; PACUT, M. Multi-gpu parallel memetic algorithm for capacitated vehicle routing problem. *CoRR*, abs/1401.5216, 2014. Disponível em: <http://arxiv.org/abs/1401.5216>. Citado 2 vezes nas páginas 33 e 34.

KONFRŠT, Z. Parallel genetic algorithms: Advances, computing trends, applications and perspectives. *18th IPDPS 2004*, p. 162, 2004. Citado na página 34.

LINDEN, R. *Algoritmos Genéticos*. 3a edição. ed. [S.l.]: Ciência Moderna, 2012. Citado 4 vezes nas páginas 25, 28, 29 e 34.

LIU, X.; JIANG, W.; XIE, J. Vehicle routing problem with time windows: a hybrid particle swarm optimization approach. In: IEEE. *Natural Computation, 2009. ICNC'09. Fifth International Conference on*. [S.l.], 2009. v. 4, p. 502–506. Citado 2 vezes nas páginas 33 e 34.

MICHALEWICZ, Z. *Genetic algorithms+ data structures= evolution programs*. [S.l.]: Springer Science & Business Media, 2013. Citado 2 vezes nas páginas 28 e 29.

MUNAWAR, A. et al. A survey: Genetic algorithms and the fast evolving world of parallel computing. In: IEEE. *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. [S.l.], 2008. p. 897–902. Citado na página 34.

OCHI, L. S. et al. A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. In: *Parallel and Distributed Processing*. [S.l.]: Springer, 1998. p. 216–224. Citado 7 vezes nas páginas 25, 33, 34, 35, 41, 46 e 57.

OLIVEIRA, W. D. *Algoritmo Evolutivo Paralelo para o Problema de Atribuição de Localidades a Aneis em redes SONET/SDH*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE, 2010. Citado 2 vezes nas páginas 33 e 34.

PACHECO, M. A. C. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, 1999. Citado na página 28.

RAUBER, T.; RÜNGER, G. *Parallel programming: For multicore and cluster systems*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 30.

ROSA, M. F. G. da; CERA, M. C. Análise do desempenho do Algoritmo Genético paralelizado com OpenMP. In: *XXVI Congresso Regional de Iniciação Científica & Tecnologia em Engenharia*. Alegrete/RS: [s.n.], 2014. p. 1 – 4. Citado na página 30.

ROSA, M. F. G. da; CERA, M. C. Estudo preliminar sobre a escalabilidade de um Algoritmo Genético paralelizado com OpenMP. In: SBC. *XV Escola Regional de Alto Desempenho (ERAD/RS)*. Gramado/RS, 2015. p. 217 – 220. Citado 6 vezes nas páginas 25, 46, 49, 53, 54 e 57.

SANTOS, O. dos. Estudo comparativo da eficiência de um algoritmo genético sequencial frente um algoritmo genético paralelo. 2009. Citado na página 34.

SILVA, H. H.; PAIVA, C. Avaliação de implementações do algoritmo genético paralelo para solução do problema do caixeiro viajante usando openmp e pthreads. In: *Workshop de Iniciação Científica evento integrante do XIII Simpósio em Sistemas Computacionais WSCAD-SSC*. [S.l.: s.n.], 2012. Citado 3 vezes nas páginas 33, 34 e 35.

STEEMAN, Q. The vehicle routing problem with drop yards: A dynamic programming approach. University of Twente, 2012. Citado na página 27.

TAILLARD, É. *Elementary (capacitated) vehicle routeing - Christofides et al.* 2015. Http://mistic.heig-vd.ch/taillard/problemes.dir/vrp.dir/vrp.html, acessado Julho/2015. Citado 4 vezes nas páginas 45, 53, 54 e 57.

TOTH, P.; VIGO, D. *The vehicle routing problem*. [S.l.]: Society for Industrial and Applied Mathematics, 2001. Citado na página 27.

WHITLEY, D.; RANA, S.; HECKENDORN, R. B. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, UNIVERSITY COMPUTING CENTRE ZAGREB, v. 7, p. 33–48, 1999. Citado na página 41.

YUSSOF, S. et al. A coarse-grained parallel genetic algorithm with migration for shortest path routing problem. In: IEEE. *High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on*. [S.l.], 2009. p. 615–621. Citado 2 vezes nas páginas 33 e 46.