UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

AVALIAÇÃO DE TÉCNICAS DE PARALELIZAÇÃO DE ALGORITMOS BIOINSPIRADOS UTILIZANDO COMPUTAÇÃO GPU: UM ESTUDO DE CASOS PARA OTIMIZAÇÃO DE ROTEAMENTO EM REDES ÓPTICAS

VINCENT WILLIAN ARAÚJO TADAIESKY

DM:10/2015

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2015

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VINCENT WILLIAN ARAÚJO TADAIESKY

AVALIAÇÃO DE TÉCNICAS DE PARALELIZAÇÃO DE ALGORITMOS BIOINSPIRADOS UTILIZANDO COMPUTAÇÃO GPU: UM ESTUDO DE CASOS PARA OTIMIZAÇÃO DE ROTEAMENTO EM REDES ÓPTICAS

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2015

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VINCENT WILLIAN ARAÚJO TADAIESKY

AVALIAÇÃO DE TÉCNICAS DE PARALELIZAÇÃO DE ALGORITMOS BIOINSPIRADOS UTILIZANDO COMPUTAÇÃO GPU: UM ESTUDO DE CASOS PARA OTIMIZAÇÃO DE ROTEAMENTO EM REDES ÓPTICAS

Dissertação submetida à Banca Examinadora do Programa de Pós-graduação em Engenharia Elétrica da UFPA para o obtenção do Grau de Mestre em Engenharia Elétrica na área de Computação Aplicada, elaborada sob a orientação do Prof. Dr. Ádamo Lima de Santana.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2015

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

AVALIAÇÃO DE TÉCNICAS DE PARALELIZAÇÃO DE ALGORITMOS BIOINSPIRADOS UTILIZANDO COMPUTAÇÃO GPU: UM ESTUDO DE CASOS PARA OTIMIZAÇÃO DE ROTEAMENTO EM REDES ÓPTICAS

AUTOR: VINCENT WILLIAN ARAÚJO TADAIESKY

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADDA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM ____ / ____ / _____

BANCA EXAMINADORA:

Prof. Dr. Ádamo Lima de Santana
(ORIENTADOR – UFPA)

Prof. Dr. Diego Lisboa Cardoso
(MEMBRO EXTERNO – UFPA-FCT)

Prof. Dr. Marcelino Silva da Silva
(MEMBRO EXTERNO – UFPA-Castanhal)

Prof.ª. Dr.ª Adriana Rosa Garcez Castro
(MEMBRO INTERNO– UFPA-PPGEE)

VISTO:

Prof. Dr. Evaldo Gonçalves Pelaes
(COORDENADOR DO PPGEE/ITEC/UFPA)

AGRADECIMENTOS

Primeiramente, como de praxe de minha parte, agradeço a Deus, seja lá como ele for, por existir.

Agradeço aos meus pais e minha irmã por me conhecerem o suficiente para confiarem a mim meu trabalho e não a responsabilidade de cuidar de mim mesmo.

Agradeço ao meu orientador, Prof. Dr. Ádamo Lima de Santana, por ser tão companheiro, quanto orientador, me ajudando e confiando em mim, mesmo nos momentos mais complicados, sempre acreditando que posso evoluir mais.

Agradeço à meus amigos Nathalia Nascimento, Petterson Marques e Iury Batalha pelo sacrifício de me ajudar mesmo quando doentes e ocupados, dizendo que “é isso que faz um amigo”.

Agradeço aos meus colegas do LINC, em especial ao grupo de Computação Bioinspirada, Igor, Iago e Paulo, que me auxiliaram no desenvolvimento deste trabalho.

Por último, mas não menos importante, agradeço à banca examinadora desta Dissertação de Mestrado, os Professores Doutores Dr. Diego Lisboa Cardoso, Marcelino Silva da Silva e Adriana Rosa Garcez Castro, que mesmo com os contratempos ocorridos, não hesitaram em avaliar este trabalho e contribuir com sua excelência

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A aplicação em logística de distribuição é diversa, a exemplo do planejamento de transporte e entrega de mercadorias ou no roteamento de dados em redes de telecomunicações. Dado a amplitude e capilaridade desses problemas, trabalhos vêm sendo desenvolvidos visando reduzir os gastos para o funcionamento de redes dessa magnitude, sobretudo no que tange à demanda de energia elétrica. Sendo assim, o presente trabalho apresenta uma proposta de método de resolução de problemas de roteamento com alto grau de demanda. O método proposto é baseado em algoritmos bioinspirados, que aliados a outros métodos, garantem a integridade das soluções obtidas, além de sua proximidade ao ótimo. Entretanto, tais algoritmos se tornam computacionalmente custosos à medida que a complexidade da aplicação em questão aumenta e, portanto, ambientes multiprocessados, como plataformas de computação em GPU, vêm sendo largamente utilizados para aumentar a performance dos mesmos. Sendo assim, este trabalho visa realizar testes sobre as técnicas de paralelização desses algoritmos mais difundidas, com o objetivo de avaliar qual estratégia tem melhor relação com cada algoritmo testado para o problema descrito acima. Os algoritmos que auxiliaram nos testes foram Algoritmos Genéticos e Otimização por Enxame de Partículas, que são altamente difundidos. Os resultados mostram que a estratégia de paralelização a ser utilizada depende tanto da plataforma em que está sendo implementada, quanto do problema a ser tratado.

Palavras-chave: *Algoritmos Bioinspirados, GPU Computing, CUDA, Roteamento*

ABSTRACT

The applications on distribution logistics are diverse, such as the transportation planning and delivery of goods or in telecommunication networks data routing. Given the breadth and capillarity of these problems, studies have been developed to reduce network operating costs of this magnitude, especially regarding the demand for electricity. Therefore, this work proposes a method of resolution of routing problems with high demand. The proposed method is based on bio-inspired algorithms, which combined with other methods, ensure the integrity of the solutions, as well as its proximity to optimum. Nevertheless, such algorithms becomes computationally expensive as the application complexity in question grows and, therefore, multiprocessor environment, like GPU Computing platforms, has being widely used to increase bio-inspired algorithms performance. Thus, this work aims perform tests about the widespread parallelization techniques of these algorithms, intending to make an evaluation of which strategies has better relation with each tested algorithm. In order to do this, the routing problem in WDW optics networks with high demand level was used as a case study, in which it is needed define which are the better routes to demands sent simultaneously. The algorithms that assisted the tests were Genetic Algorithms and Swarm Particle Optimization, which are highly disseminated. The results show that the parallelization strategy to be used depends as much on the platform in which has been implemented, as the problem to be treaty.

Keywords: *Bio-inspired Algorithms, GPU Computing, CUDA, Routing*

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Durante a Segunda Guerra Mundial, observou-se grande necessidade de lidar com problemas de natureza logística, tática e de estratégia militar de grande dimensão e complexidade, que não foram possíveis de serem solucionados com métodos tradicionais. Nesse contexto, surgiu a área de Pesquisa Operacional (PO), que é uma ciência aplicada, multidisciplinar, que tem como foco o suporte à tomada de decisões (SOBRAPO - Sociedade Brasileira de Pesquisa Operacional, 2015).

Uma das principais disciplinas que dá suporte à PO é a Inteligência Computacional (IC), que é o estudo e projeto de agentes inteligentes. Um agente é algo que atua em uma ambiente. Agentes incluem cachorros, termostatos, aviões, homens, etc. Um agente inteligente é um sistema que age de forma inteligente: o que ele faz é apropriado para as circunstâncias e objetivos, é flexível com respeito a ambientes e objetivos variantes no tempo, aprende a partir da experiência, toma as ações apropriadas dados a sua limitação sensorial e a capacidade computacional.

Dentre as tarefas realizadas em IC está a Otimização, presente no processo de desenvolvimento e na execução de diversas aplicações, principalmente em PO. Atualmente, a alta complexidade atribuída aos problemas torna, por vezes, as técnicas tradicionais de otimização ineficazes na obtenção do resultado ótimo, como por exemplo, na solução de problemas NP-completos.

Neste âmbito, a utilização de algoritmos bioinspirados, mais especificamente algoritmos evolucionários e de inteligência coletiva, como Algoritmos Genéticos (AG) e Otimização por Enxame de Partículas (PSO), mostra-se extremamente eficaz na obtenção de resultados aproximados do ótimo, principalmente quando a complexidade do problema aumenta.

Uma aplicação destes algoritmos difundida é a utilização de AG para encontrar o melhor conjunto de pesos sinápticos no treinamento de Redes Neurais Artificiais (RNA). Outra aplicação, desenvolvida recentemente para estas duas classes de algoritmos bioinspirados, é a utilização de AG e PSO no processo de imputação múltipla de dados, sendo

de grande importância em Ciências Sociais e da Saúde, onde ocorrem com maior frequência a ausência de dados em suas bases e as estimativas sem estes podem tornar-se enviesadas.

Entretanto, tais algoritmos se tornam computacionalmente custosos à medida que a complexidade da aplicação em questão aumenta. Para sanar este problema, a utilização de ambientes multiprocessados tornou-se uma opção para a execução destes algoritmos (Moraes, 2011).

Na década de 90, cientistas já estavam utilizando das bibliotecas de programação gráfica para executar tarefas complexas de computação em GPU (NVIDIA, 2015), haja vista que esta é uma arquitetura desenhada para processamento paralelo e de custo monetário muito mais acessível que a construção de um *cluster* de computadores. Por conta disso, a NVIDIA desenvolveu, em 2006, a plataforma de Computação em GPU (*GPU Computing*) CUDA (*Compute Unified Device Architecture*), para facilitar o desenvolvimento de aplicações não gráficas em ambiente GPU, disponibilizando uma interface próxima a utilizada para desenvolver aplicações comuns.

Tal plataforma paralela, já é utilizada em outros trabalhos para a implementação de algoritmos de otimização, inclusive bioinspirados (Oiso, Yasuda, Ohkura, & Matumura, 2011) (Feier, Lemnaru, & Potolea, 2011) (Zhou & Tan, 2009) (Wachowiak & Foster, 2012). Em todos, a plataforma CUDA se mostrou eficiente, reduzindo o tempo de execução destes algoritmos. Porém, não há uma consolidação acerca de qual a melhor estratégia de implementação a se aplicar a eles. Sendo estas dependentes tanto da arquitetura paralela utilizada, quanto do problema a ser tratado.

Sendo assim, o presente trabalho visa realizar testes sobre as técnicas de paralelização de algoritmos bioinspirados mais difundidas, como a estratégia baseada em ilhas, sob a plataforma CUDA, com o objetivo de avaliar quais tem melhor relação com cada algoritmo testado neste ambiente. Para tal, todas as combinações de algoritmo-estratégia serão aplicadas ao problema de roteamento em redes ópticas WDM (*Wavelength-Division Multiplexing*) extensas (que atinjam mais de um fuso-horário) e com múltiplas requisições simultâneas.

1.2 OBJETIVOS E METODOLOGIA

Com base no que foi exposto anteriormente, a presente dissertação pretende realizar um estudo acerca das estratégias de paralelização mais difundidas de AG e de PSO, a fim de verificar quais melhor se adaptam aos conjuntos destes algoritmos para a execução sob a plataforma CUDA.

Estes algoritmos serão implementados nesta plataforma de processamento paralelo, que transforma suas placas gráficas em GPUs de propósito geral (*General Purpose GPU* - GPGPU). A utilização da arquitetura GPU para a solução de problemas computacionalmente custosos vem sendo aplicada desde o final da década de 90 (NVIDIA, 2015), por ser uma opção mais viável em termos econômicos, quando comparada à tradicional arquitetura de *cluster* de computadores.

Tais estratégias serão aplicadas ao problema de roteamento em redes óticas WDM (*Wavelength-Division Multiplexing*) que se estendam por vários fuso-horários e recebam várias requisições simultâneas de comunicações. Para tal, os algoritmos terão que encontrar os caminhos com menor custo, e quais larguras de bandas, que serão utilizados por cada requisição de comunicação feita à rede.

Optou-se por aplicar tais algoritmos a este domínio devido a amplitude e capilaridade dos problemas de roteamento com grande número de restrições e por serem essencialmente problemas de otimização. A aplicação em logística de distribuição, seja de insumo ou de dados, é diversa, como exemplo é possível citar o roteamento de veículos, sejam eles simples, com coleta e entrega, com entregas particionadas, *etc*.; ou ainda, no roteamento de dados em redes de telecomunicações variadas (*e.g.* redes ADSL, wireless, *mesh*).

Os objetivos podem ser verificados na lista a seguir:

- Verificar estratégias de paralelização de algoritmos bioinspirados mais difundidas;
- Adaptar tais estratégias à plataforma CUDA;
- Realizar testes sob o problema de roteamento em redes ópticas WDM; e
- Verificar qual estratégia melhor se adapta a cada algoritmo nesta plataforma.

1.3 ESTRUTURA DO TRABALHO

A presente dissertação está organizada como segue. No Capítulo 2 são apresentados alguns conceitos acerca de Computação Bioinspirada, assim como os algoritmos alvos deste estudo. O Capítulo 3 introduz a plataforma CUDA, que será utilizada para a paralelização dos algoritmos, e alguns conceitos acerca de Programação Paralela. No Capítulo 4, alguns trabalhos relacionados à pesquisa são discutidos. A discussão realizada neste capítulo tem como resultado o planejamento das estratégias que serão utilizadas para paralelizar os algoritmos estudados, que são descritos no Capítulo 5. O Capítulo 6 apresenta o caso de estudos, no qual serão realizados os testes, que consiste no problema de roteamento em redes ópticas com alto grau de demanda. O Capítulo 7 descreve como foram realizados os testes, assim como apresenta os resultados obtidos. As conclusões, dificuldades e trabalhos futuros serão expostos no Capítulo 8.

2 COMPUTAÇÃO BIOINSPIRADA

2.1 CONSIDERAÇÕES INICIAIS

Este capítulo tem por objetivo apresentar conceitos acerca de algoritmos bioinspirados, importantes para a compreensão do presente trabalho, abrangendo os campos de Algoritmos Evolucionários e de Inteligência Coletiva, nos quais se encontram os algoritmos utilizados no desenvolvimento deste estudo, AG e PSO, respectivamente.

2.2 COMPUTAÇÃO NATURAL

A observação da natureza é uma das principais inspirações dos cientistas. Particularmente, na computação, pesquisadores utilizam ideias extraídas destas observações para desenvolver soluções baseadas em sistemas computacionais, campo de estudo chamado de Computação Natural (De Castro, 2006). O mesmo autor divide esse campo em três ramificações:

- Computação bioinspirada: faz uso da natureza como forma de inspiração para o desenvolvimento de técnicas de resolução de problemas. Sua ideia principal consiste em inspirar-se por meio da natureza para resolver problemas complexos a fim de desenvolver ferramentas computacionais ou algoritmos.
- Simulação e emulação da natureza por meio da computação: essa ramificação é basicamente um processo sintético que visa criar padrões, formas, comportamentos e organismos os quais, não necessariamente, se assemelham a "vida-como-nós-conhecemos". Seus produtos podem ser usados para simular vários fenômenos naturais, aumentando assim a compreensão da natureza e as percepções sobre modelos computacionais;
- Computação com materiais naturais: corresponde ao uso de novos materiais para realizar cálculos, constituindo assim um novo paradigma de computação, que surge para substituir ou complementar os computadores atuais à base de silício.

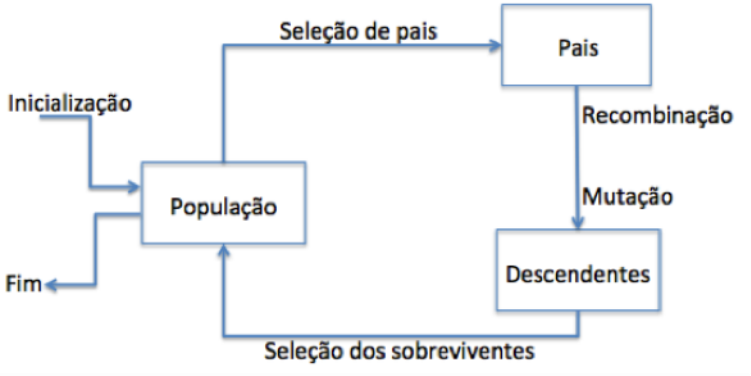
O primeiro campo de pesquisa vem sendo amplamente estudado devido seu custo relativamente baixo e alta aplicabilidade, principalmente em Pesquisa Operacional/Otimização (Eiben & Smith, 2003). Dentre os ramos deste campo de pesquisa está a Computação Evolucionária, que se baseia na teoria da seleção natural, a qual afirma que os indivíduos melhor adaptados ao meio têm maior probabilidade de se reproduzir, ou seja, transmitir o "conhecimento" gravado em seus genes, representada neste trabalho pelo Algoritmo Genético (AG), e o campo da Otimização por Inteligência Coletiva, o qual baseia-se na observação de sociedades naturais e possui em sua essência a ideia de que os indivíduos não aprendem sozinhos, mas com o grupo em que estão inseridos, representado pelo algoritmo de Otimização por Enxame de Partículas (*Particle Swarm Optimization* - PSO).

Tais campos de pesquisa visam desenvolver técnicas de busca e otimização para problemas complexos de otimização (Holland, 1975) (Goldberg, 1989). Os algoritmos utilizados foram escolhidos por serem de fácil implementação e por suas características de *exploration* e *exploitation* – a exploração do espaço de busca como um todo e a busca de um ótimo em uma região específica, respectivamente (Eiben & Smith, 2003). Estes algoritmos serão apresentados nas seções seguintes.

2.3 ALGORITMOS GENÉTICOS

Para De Castro (2006), um algoritmo evolucionário padrão deve se propor a ter alguns itens comuns à todos os outros, como por exemplo, a população deve: passar informação gênica à prole, apresentar diversidade genética e passar pela seleção do mais apto. A Figura 2.1 apresenta um esquema do funcionamento básico de um algoritmo evolucionário.

Figura 2.1 - Esquema geral de um Algoritmo Evolucionário



O fluxograma apresentado é similar ao chamado *algoritmo genético canônico*, que representa a forma mais básica desta classe de algoritmos e apresenta as operações necessárias para que um algoritmo heurístico seja considerado evolucionário. Destas etapas, as presentes no AG são: seleção de pais, recombinação e mutação. Contudo, há outros pontos importantes a se considerar na implementação de AG para solução de problemas de otimização.

Nas seções seguintes serão apresentadas as principais características do algoritmo, que devem ser consideradas durante o planejamento do mesmo para a aplicação, assim como as operações realizadas durante o processo de evolução do mesmo.

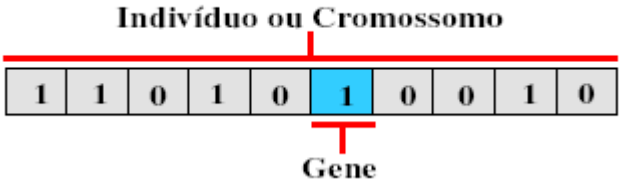
2.3.1 CODIFICAÇÃO GÊNICA

Em AG, cada indivíduo da população possui uma estrutura, chamada *cromossomo* (geralmente um vetor), onde são armazenadas os dados referentes a qual estado o indivíduo está representando no problema a ser resolvido. Tais dados são chamados *genótipo*, por ser a versão codificada das informações originais, chamadas *fenótipo*, de forma a facilitar o processamento realizado pelo algoritmo.

Há diversas formas de se codificar a informação nos genes do indivíduo. As três mais comuns são: codificação binária, onde cada gene armazena um bit; codificação inteira, onde cada gene armazena um número inteiro; e codificação real, na qual os genes armazenam números reais. A Figura 2.2 apresenta um exemplo de cromossomo com codificação binária. A codificação escolhida para o desenvolvimento do presente trabalho foi a inteira, pois serão

armazenados somente os índices dos caminhos a serem utilizados pela rede para a comunicação. Isso será melhor explicado no desenvolvimento do trabalho.

Figura 2.2 - Cromossomo com codificação binária



2.3.2 FUNÇÃO DE APTIDÃO

A *função de aptidão*, também chamada de *função de fitness*, tem o papel de associar o AG ao problema em questão. Esta é utilizada para avaliar o indivíduo, quanto à sua qualidade como solução do problema, para uso posterior pelos operadores de seleção, tanto de pais, quanto de sobreviventes. Isto é feito por meio da utilização do fenótipo do indivíduo que, como dito antes, é a versão decodificada dos dados do cromossomo, ou seja, a informação contida no mesmo.

2.3.3 SELEÇÃO DE INDIVÍDUOS

É na etapa de seleção, tanto de pais quanto de sobreviventes, que a teoria da seleção natural é aplicada. Nesta etapa, um método que leva em consideração os resultados da função de aptidão sob cada indivíduo é aplicado à população, fazendo com que os indivíduos com os melhores valores de aptidão (indivíduos mais aptos) tenham maior probabilidade de serem selecionados e assim passarem suas características à nova geração (seleção de pais), ou sobreviverem para fazerem parte da próxima geração (seleção de sobreviventes). Existem diversos métodos que realizam a seleção de indivíduos no AG. Os mais conhecidos são: dizimação, roleta e torneio.

2.3.3.1 DIZIMAÇÃO

Estratégia determinística simples, que consiste em ordenar os indivíduos pelo valor de sua função objetivo e simplesmente remover um número fixo de indivíduos que apresentarem baixa aptidão, ou seja, criar um patamar e eliminar aqueles que estiverem abaixo desse patamar. Os pais são então sorteados aleatoriamente, usando distribuição uniforme, dentre os que sobreviveram ao processo. A vantagem desta estratégia de seleção é a simplicidade de implementação, que consiste em determinar quais indivíduos possuem aptidão suficiente para permanecer na população e participar do processo de reprodução. A desvantagem é que características genéticas únicas podem ser perdidas uma vez que um indivíduo é removido da população. A perda da diversidade é uma consequência natural das estratégias evolucionárias, mas neste caso ela geralmente ocorre antes que os efeitos benéficos de uma característica única sejam reconhecidos pelo processo evolutivo.

2.3.3.2 SELEÇÃO PROPORCIONAL

Um dos mais populares métodos estocásticos de seleção é a seleção proporcional, também conhecida como roleta. Neste método, os indivíduos são selecionados com base na probabilidade de seleção que é diretamente proporcional à função objetivo. A probabilidade p_i que um indivíduo *i* possui de ser selecionado através de sua função de aptidão $f_{(i)}$ está expressa pela equação (2.1).

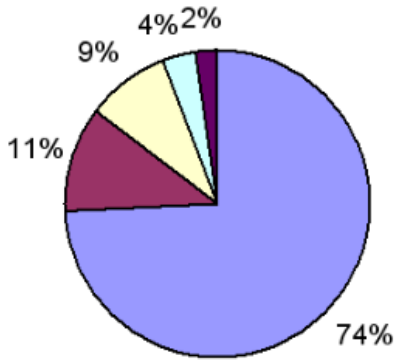
$$p_i = \frac{f(i)}{\sum_i f(i)} \tag{2.1}$$

Através desta equação, pode-se verificar que a probabilidade de um indivíduo ser escolhido é proporcional à seu resultado de sua função aptidão, com relação à soma dos resultados de toda a população, ou seja, é proporcional à sua qualidade como solução do problema na presente geração.

O processo de seleção proporcional pode ser interpretado como uma roleta, onde cada indivíduo da população é representado em uma porção proporcional ao seu índice de aptidão,

como apresentado na Figura 2.3. Desta forma, uma porção maior da roleta é fornecida aos indivíduos com alta aptidão, cabendo aos menos aptos uma porção menor. Cada sorteio da roleta seleciona um indivíduo pai, portanto a roleta é sorteada até que todos os pais sejam selecionados. A grande vantagem deste método é que todos os indivíduos, sem exceção, têm oportunidade de serem selecionados.

Figura 2.3 - Probabilidades de seleção no método de roleta



2.3.3.3 TORNEIO

Neste processo, N indivíduos são selecionados aleatoriamente na população. Estes, então, competem entre si com base no valor da função objetivo, sendo o vencedor aquele que tiver o maior valor. O processo se repete até que a nova população esteja completa. A implementação deste esquema é relativamente simples e requer baixo custo computacional, sendo um dos motivos de sua popularidade.

2.3.4 CRUZAMENTO

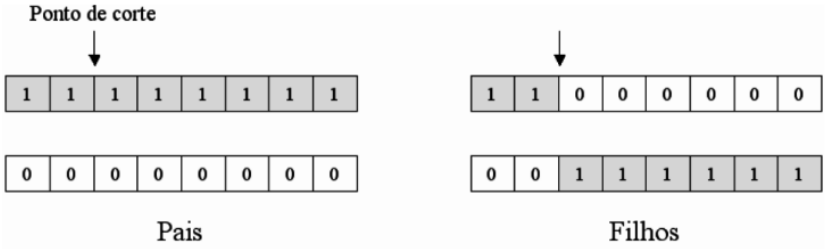
Segundo Konak, Coit, & Smith (2006), o operador de cruzamento (ou recombinação) é o mais importante dos operadores genéticos. Nesta operação, dois indivíduos são quebrados em uma ou mais posições escolhidas aleatoriamente, os fragmentos resultantes são recombinados e dois novos indivíduos são criados.

Existem vários métodos de cruzamento, que diferem-se pela escolha do ponto de corte e pela maneira como será feita esta troca de informações. A seguir, são apresentados os tipos mais comuns de cruzamentos encontrados na literatura.

2.3.4.1 CRUZAMENTO DE UM PONTO

Neste cruzamento um ponto de corte é escolhido aleatoriamente, e a partir deste ponto as informações do código genético dos pais serão trocados. Os filhos serão gerados de acordo com as informações pertencentes a um dos pais anteriores a este ponto e completados com as informações pertencentes ao outro pai posteriores a este ponto (Barbosa, 2005). A Figura 2.4 mostra um exemplo de funcionamento do cruzamento de um ponto.

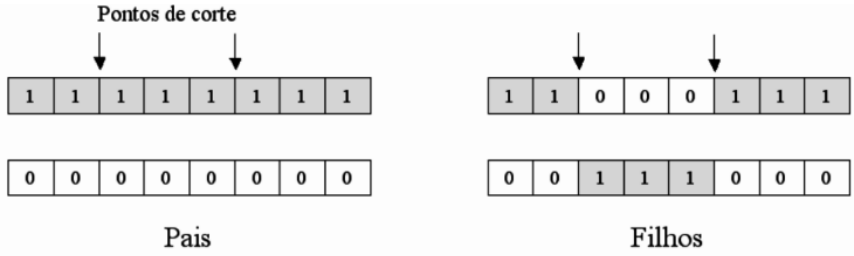
Figura 2.4 - Cruzamento de um ponto



2.3.4.2 CRUZAMENTO DE DOIS PONTOS

Aqui dois pontos são escolhidos aleatoriamente para o corte nos cromossomos dos pais. Todos os materiais genéticos dos pais que estão entre os pontos serão trocados e o restante do material genético permanecerá inalterado (Linden, 2006). A Figura 2.5 mostra um exemplo de funcionamento do cruzamento de dois pontos.

Figura 2.5 - Cruzamento de dois pontos



2.3.5 MUTAÇÃO

O operador de mutação introduz mudanças aleatórias em determinadas características dos indivíduos. A mutação reintroduz a diversidade genética à população e auxilia o algoritmo a fugir de máximos ou mínimos locais (Konak, Coit, & Smith, 2006).

Entre os operadores de mutação para as codificações inteira e real, o mais comumente utilizado é a mutação uniforme. Neste tipo de mutação, posições do indivíduo são sorteadas e os genes correspondentes são redefinidos a partir de uma distribuição de probabilidade uniforme no intervalo permitido para cada gene.

2.3.6 CRITÉRIOS DE PARADA E CONVERGÊNCIA

Atualmente, o maior desafio na implementação de AGs é determinar o momento em que este deve ser interrompido quando não há informação *a priori* acerca da função objetivo (Bhandari, Murthy, & Pal, 2012), pois é impossível saber se o resultado encontrado pelo algoritmo é o ótimo correspondente ao problema ou um sub-ótimo, dito ótimo local, que corresponde apenas a uma região da função objetivo.

Segundo Safe, Carballido, Ponzoni & Brignole (2004), existem três critérios de parada que são tradicionalmente aplicados em AG:

- Por número de iterações;
- Por número de avaliações feitas pela função de aptidão; e
- Quando a chance de se obter uma mudança significativa nas próximas geração é excessivamente baixa.

A escolha para os valores de sensibilidade dos dois primeiros requer algum conhecimento prévio sobre o problema que permita uma estimativa razoável para estes valores. Entretanto, para a terceira alternativa, tal conhecimento prévio tem menor impacto, haja vista que este leva em consideração a convergência da população para uma solução.

Em relação a este critério de parada, há duas variações, chamados critérios de parada fenotípico e genotípico. O primeiro leva em consideração os resultados da função de aptidão aplicada à população, onde se pode avaliar apenas se há variação no melhor resultado, interrompendo o algoritmo caso não haja mudanças durante um certo número de gerações, ou se pode avaliar a variância dos valores obtidos na população, interrompendo o algoritmo caso esta alcance um valor baixo. O segundo leva em consideração a informação codificada, contida nos cromossomos, interrompendo o algoritmo quando houver uma convergência dos genes da população para os mesmos valores. Por exemplo, se um determinado gene assume o valor 1 em 90% (valor escolhido empiricamente) da população, assume-se que este convergiu, e quando 80% (valor escolhido empiricamente) dos genes convergiram, assume-se que o algoritmo convergiu e este é interrompido.

2.4 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

Outro ramo da computação bioinspirada estudado neste trabalho é o de Inteligência Coletiva, o qual baseia-se na observação de sociedades naturais e possui em sua essência a ideia de que os indivíduos não aprendem sozinhos, mas seus conhecimentos e habilidades se disseminam entre toda a população. Logo, uma população pode aprender a partir da observação dos indivíduos ao seu redor (Kennedy & Eberhart, 2001).

O objetivo dos algoritmos dessa categoria é modelar o comportamento de indivíduos, a interação local destes com o ambiente que o cerca e com seus vizinhos individuais, a fim de se obter um comportamento que pode ser utilizado para resolver problemas mais complexos, como problemas de otimização, assim como o AG.

Três são os fatores que regem esses algoritmos: avaliar, comparar e imitar. O primeiro consiste nos indivíduos estimarem seu comportamento baseados na sua capacidade de sentir o ambiente ao seu redor. O segundo, os indivíduos usam uns aos outros como parâmetros de

comparação. E, por fim, a imitação é importante para aquisição e manutenção das habilidades, o que converge a população para um melhor ponto (Eiben & Smith, 2003).

Kennedy & Eberhart (2001) mencionam que os indivíduos aprendem localmente com seus vizinhos, por meio de interações com sua vizinhança, e compartilhando experiências entre si, ocorrendo assim um processo de aprendizado. Tal fator está interligado com a disseminação do conhecimento por meio de resultados de aprendizado social. O qual pode ser observado em crenças, atitudes, comportamento e outros tipos de manifestações entre indivíduos de uma população. Com isso, percebe-se que uma sociedade é um sistema auto-organizado com propriedades globais, as quais não podem ser preditas utilizando apenas as propriedades individuais de quem a compõe.

Por fim, outro item mencionado pelos trabalhos citados, é que o conhecimento é otimizável por culturas, ou seja, embora as interações sejam locais, a experiência e as inovações que ocorrem são transportadas pela cultura até os indivíduos mais afastados da população, havendo interação com esses e gerando resultados mais promissores para todos. Esse efeito global torna-se transparente a todos os indivíduos, fazendo com que a sociedade utilize isso para benefício próprio e, por consequência, de toda a sua população.

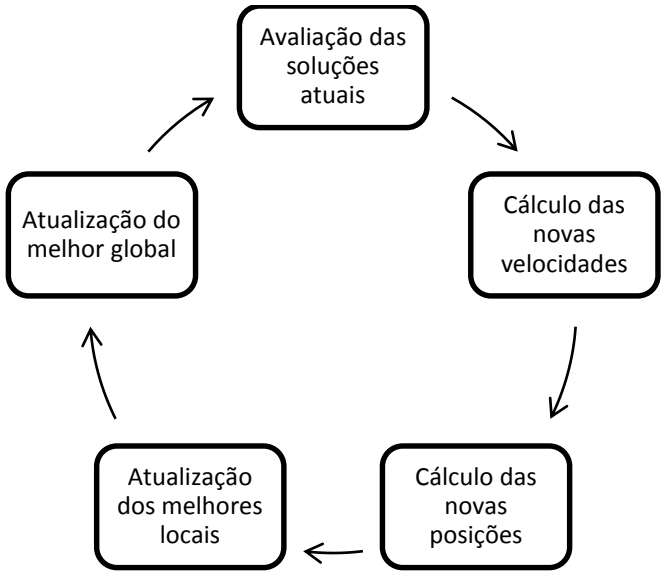
Os comportamentos citados fazem parte do modelo cultural adaptativo, que rege a modelagem dos algoritmos de inteligência coletiva. As abordagens mais conhecidas são os trabalhos inspirados por comportamento sociais de insetos. Dentre estes está o algoritmo de PSO.

A técnica do PSO é estocástica e baseada em populações, a qual foi primeiramente desenvolvida por Kennedy & Eberhart (1995). Seu algoritmo faz alusão ao comportamento social por meio da interação entre indivíduos, neste caso partículas, de um determinado grupo, denominado de enxame. Sua inspiração teve origem a partir da observação de bandos de pássaros e de cardumes de peixes, durante a sua busca por alimento em uma determinada região. Por meio dessa observação, evidenciou-se que o comportamento do grupo é influenciado pela experiência individual acumulada de cada partícula, e também pela experiência acumulada do grupo como um todo em relação a sua tarefa (Kennedy & Eberhart, 2001).

No PSO, cada possível solução do problema corresponde a um ponto no espaço de busca, e sua dimensão é definida pelo número de variáveis do problema. Essas soluções, ou

partículas, por sua vez, possuem um valor associado, o qual é avaliado individualmente e indica o quão adequada esta se encontra para resolver o problema; adicionalmente, cada solução possui uma velocidade que define a direção do movimento de cada partícula. Por meio da modificação da velocidade, que leva em consideração a posição atual da partícula, a melhor posição pela qual a partícula já passou e a melhor posição do grupo ao longo do tempo, o grupo consegue alcançar seu objetivo. O fluxograma da Figura 2.6 mostra o funcionamento de um PSO, na qual os melhores locais caracterizam as melhores posições encontradas por cada partícula e o melhor global caracteriza a melhor posição do enxame.

Figura 2.6 - Ciclo de aprendizagem do PSO



No algoritmo, o enxame é iniciado aleatoriamente, com um conjunto de soluções candidatas representadas por cada partícula, as quais têm seus valores de velocidades também iniciados de forma aleatória. No modelo canônico do PSO a velocidade e a atualização da posição de cada partícula podem ser calculadas de acordo com as Equações (2.2) e (2.3), respectivamente.

$$\boldsymbol{V(t+1)} = \mathrm{w} * \boldsymbol{V(t)} + \mathbf{c} * \boldsymbol{Rnd} * \left(\mathbf{mLocal} - \mathbf{X(t)}\right) + \mathbf{s} * \boldsymbol{Rnd} * \left(\boldsymbol{mGlobal} - \boldsymbol{X(t)}\right) \quad (2.2)$$

$$\boldsymbol{X(t+1) = X(t) + V(t+1)} \quad (2.3)$$

Onde, *V* corresponde a velocidade da partícula na iteração t (anterior) e t+1 (atual); *Rnd* é um valor aleatório o qual varia entre [0,1]; *X(t)* é a posição anterior da partícula; *mLocal* é a melhor posição anterior da partícula; *mGlobal* é a melhor posição anterior do enxame; *w*, *c* e *s* são os parâmetros de inércia, cognitivo e social, respectivamente, os quais determinarão se a partícula tenderá para o *mLocal*, ou para o *mGlobal* ou continuará seu movimento anterior.

A Equação (2.3) demostra como a posição de cada partícula é atualizada dependendo do valor da velocidade. Logo, percebe-se que a partícula considera as melhores posições encontradas individualmente e pelo grupo para atualizar sua velocidade, e cada uma é atualizada independentemente. Vale ressaltar que a conexão entre as dimensões da partícula (cada elemento dentro do vetor posição) dá-se pela função objetivo, assim como no AG, e esta é influenciada pelas posições das partículas. Sendo assim, quanto melhor a combinação de posições em cada dimensão, melhores são os valores encontrados pela função objetivo.

A mudança de velocidade das partículas modifica suas posições fazendo-as se movimentar através do espaço do problema. Com isso, ao longo de sucessivas iterações, o resultado da decisão individual e da influência social, exercida pelo valor do melhor global, faz com que as partículas acabem convergindo para uma solução ótima.

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados conceitos pertinentes ao presente trabalho relacionados aos algoritmos utilizados pelo mesmo, AG e PSO. Percebe-se que ambos são baseados na criação de uma população de possíveis soluções geradas aleatoriamente, que vão se aperfeiçoando conforme as iterações dos mesmos são executadas, até que alcancem um resultado satisfatório. Assim, todas as etapas dos algoritmos são realizadas repetidas vezes para cada indivíduo presente em sua população.

Esta característica torna fácil a implementação destes algoritmos em ambiente multiprocessado do tipo SIMD (*Single Instructions Multiple Data*), onde as mesmas intrusões são executadas para conjuntos diferentes de dados. Esta estrutura está presente no

desenvolvimento de aplicações para a plataforma CUDA, que será melhor apresentada no próximo capítulo.

3 PLATAFORMA CUDA

3.1 CONSIDERAÇÕES INICIAIS

O termo "programação concorrente" vem do inglês *concurrent programming*, onde *concurrent* significa "acontecendo ao mesmo tempo". Uma tradução mais adequada seria “programação concomitante”. Entretanto, o termo “programação concorrente” já está solidamente estabelecido no Brasil. Algumas vezes é usado o termo programação paralela com o mesmo sentido.

O presente capítulo apresentará conceitos de programação concorrente e paralela, bem como a diferença entre ambas, que dão suporte à compreensão do funcionamento da plataforma de programação paralela, CUDA. Tal plataforma foi escolhida para a implementação da versão paralela dos algoritmos alvos deste estudo pelo crescimento em sua utilização no desenvolvimento de aplicações de alta performance.

3.2 PROGRAMAÇÃO PARALELA

Um sistema concorrente contém dois ou mais processos trabalhando em conjunto para a realização de uma tarefa (Andrews, 2000), sendo cada processo, por sua vez, um conjunto de instruções sequenciais. Em sistemas operacionais, processo é um módulo executável único, que compete com outros pela utilização dos recursos computacionais (processador, memória, dispositivos de entrada e saída etc.) (Tanenbaum, 2007). O objetivo da programação concorrente é gerenciar estes recursos.

Segundo Moraes (2011), o termo processamento paralelo pode ser atribuído a qualquer computador ou organização que execute diversas operações simultaneamente. No entanto para Andrews (2000), a diferença primordial entre um sistema paralelo e outros sistemas concorrentes é que o primeiro é desenvolvido com o objetivo principal de reduzir o tempo de execução de um único problema em relação a sua versão sequencial. Portanto, computação paralela é uma subárea de programação concorrente que vise aumentar a performance de algoritmos através de sua paralelização.

Deste modo, é necessário que alguns cuidados sejam tomados a fim de que o correto funcionamento do algoritmo seja mantido, como a sincronização entre processos e a troca de informações entre os mesmos, ambos dependentes da arquitetura de hardware na qual os processos estão em execução.

3.2.1 SINCRONIZAÇÃO ENTRE PROCESSOS

A sincronização entre os processos em execução é crucial para sua comunicação, pois uma informação enviada quando o processo destinatário não estiver a sua espera, será ignorada e se perderá, acarretando no mal funcionamento do sistema implementado.

Segundo Andrews (2000), há dois tipos básicos de sincronização: exclusão mútua e condição de sincronização. Exclusão mútua consiste em assegurar que processos não executem suas seções críticas simultaneamente. A condição de sincronização consiste em fazer com que o processo em questão aguarde (interrompa sua execução) até que uma condição seja satisfeita.

Como exemplo, pode-se citar o clássico problema do leitor e escritor, onde ambos querem acessar o mesmo espaço de memória, um para ler e o outro para escrever, respectivamente (Dijkstra, 1965). Neste problema, a exclusão mútua é necessária para que ambos não acessem o espaço de memória ao mesmo tempo, pois isso faria com que a mensagem lida seja inconsistente devido a simultaneidade das ações. A condição de sincronização é utilizada para garantir que o leitor não tente ler a mensagem antes que o escritor termine de escrevê-la, para que ele sempre leia a mensagem completa.

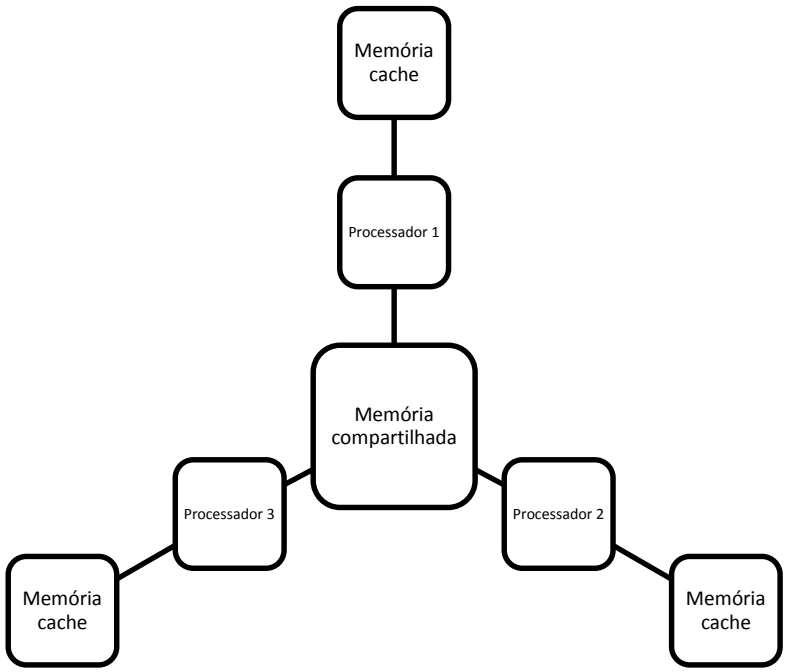
Estando sincronizados, os processos conseguem comunicar-se de maneira eficaz. Contudo, para que esta comunicação seja tão eficiente quanto possível, é necessário que sua estratégia de troca de informações esteja condizente com a arquitetura de hardware da máquina na qual os processos serão executados. Por exemplo, em multiprocessadores com memória compartilhada, a comunicação pode ser por compartilhamento de variáveis, enquanto para os com memória distribuída, esta é feita por meio de troca de mensagens entre os processadores. Na seção a seguir, essas duas arquiteturas básicas de computadores concorrentes serão apresentadas.

3.2.2 ARQUITETURA DE *HARDWARE* PARA COMPUTADORES CONCORRENTES

Segundo Andrews (2000), existem basicamente duas arquiteturas de hardware utilizadas: multiprocessadores com memória compartilhada e multiprocessadores com memória distribuída.

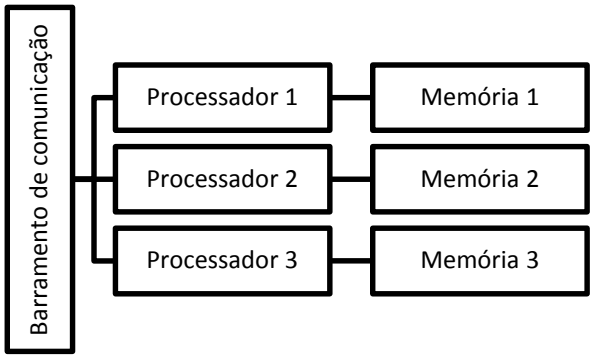
Na arquitetura de multiprocessadores com memória compartilhada, os processadores compartilham a memória principal do computador. Contudo, é importante ressaltar que a memória cache destes continua sendo individual, como apresentado na Figura 3.1. A comunicação entre os processos/processadores, para esse tipo de arquitetura, pode ser implementada por variáveis compartilhadas (*Shared-Variable Programming*), que consiste em alocar um espaço da memória para compartilhar as informações entre processos.

Figura 3.1 - Arquitetura de multiprocessadores com memória compartilhada



Em computadores com múltiplos processadores e memória distribuída, tanto a memória principal quanto a memória cache são individuais para cada processador, ou seja, cada um possui seu conjunto de memórias, como apresentado na Figura 3.2.

Figura 3.2 - Arquitetura de multiprocessador com memória distribuída



Um exemplo de máquinas pertencentes a esta arquitetura são os clusters de computadores, que consistem em um conjunto de computadores interconectados funcionando como sendo um único supercomputador. Como cada computador possui seu próprio processador e suas próprias memórias principais, a memória total do *cluster* fica distribuída, necessitando de um protocolo de comunicação especial para a interação entre os processos.

Esta comunicação pode ser feita por *Message Passing Programmming*, onde os processos se comunicam através da troca de mensagens. Neste paradigma de programação, um processo envia uma mensagem (*send*) e outro recebe a mensagem (*receive*) de maneira sincronizada. A mensagem enviada pode ou não conter dados, a depender do problema.

A arquitetura CUDA possui arquitetura de memória compartilhada em dois níveis, como será melhor abordado nas seções seguintes. No primeiro nível, a memória da placa gráfica é compartilhada entre todas os multiprocessadores existentes. Dentro de um multiprocessador, outro nível de memória, interna a ele, é compartilhada entre os processadores no mesmo.

3.2.3 AVALIAÇÃO DE DESEMPENHO DE SISTEMAS PARALELOS

Em um primeiro contato com a computação paralela, podemos nos deixar levar pela errônea ideia de que ao dividir o trabalho em *n* unidades de processamento, os ganhos em desempenho seriam proporcionalmente em *n* vezes. É importante ressaltar que isto nem sempre é possível. Navaux & De Rose (2003) relacionam os grandes desafios na área de processamento de alto desempenho, que explicam o motivo pelo qual nem sempre é possível

obter tal resultado. Os mesmos estão pautados em 4 (quatro) aspectos: arquiteturas paralelas (apresentadas anteriormente), gerenciamento das máquinas (no caso de *cluster* de computadores), linguagens mais expressivas que consigam extrair o máximo de paralelismo (em GPGPU: bibliotecas de programação gráfica *versus* linguagens de *GPU Computing*) e finalmente uma maior concorrência (concomitância) nos algoritmos.

Sendo assim, algumas métricas de avaliação de desempenho, como *speedup* e eficiência de paralelização, são comumente utilizadas para verificar a melhora de uma solução paralela em relação a sua versão sequencial. Estas métricas levam em consideração o tempo total para a execução de cada estratégia, paralela e sequencial, e serão apresentadas nas próximas seções.

3.2.3.1 SPEEDUP

Para nos auxiliar a mensurar os ganhos ao paralelizar algoritmos seriais, podemos utilizar a métrica denominada *speedup*. O *speedup* mostra o ganho efetivo que a versão paralela do algoritmo tem em relação a versão serial (Moraes, 2011). Este pode ser obtido pela Equação (3.1):

$$speedup = \frac{T_1}{T_N} \tag{3.1}$$

Onde T_1 é o tempo de execução para a versão sequencial do algoritmo e T_N é o tempo para o algoritmo paralelizado para N processos/processadores. O resultado deve variar entre 0 (zero) e N. Caso seja inferior a 1 (um), o algoritmo paralelo desenvolvido é dito como menos eficiente que sua versão sequencial e precisa ser avaliado para a remoção de gargalos, ou ainda, para apurar se foram escolhidas as melhores abordagens durante o projeto do seu código.

O valor ideal de *speedup* é igual ao número de processadores, ou seja, para N processadores, o *speedup* ideal é igual a N. Entretanto, esse valor é muito difícil de ser alcançado por conta do desbalanceamento de carga entre os processadores, conflitos de

acesso à memória em máquinas com memória compartilhada e ao custo de transferência de mensagens pela rede em máquinas com memória distribuída (Amdahl, 1967).

3.2.3.2 EFICIÊNCIA DE PARALIZAÇÃO

A eficiência de paralelização mede o grau de aproveitamento do recursos computacionais, sendo obtida através da razão entre o *speedup* efetivo, calculado pela Equação (3.1), e o *speedup* ideal, apresentado anteriormente, ou seja, mede a relação entre o grau de desempenho e os recursos computacionais disponíveis, conforme a Equação (3.2).

$$\eta = \frac{speedup}{N} \tag{3.2}$$

Multiplicando-se o resultado desta equação por 100 temos a eficiência de paralelização em pontos percentuais, onde se pode perceber que para o valor de *speedup* ideal, a eficiência de paralelização é igual a 100%.

3.2.3.3 GRANULARIDADE

Granularidade é uma métrica de avaliação de programas paralelos que visa determinar, em termos qualitativos, não quantitativos, o balanceamento de carga (de processamento) entre as unidades de processamento disponíveis e a sobrecarga das mesmas, sendo dividida em granularidade grossa, média e fina.

A granularidade de tarefas é inversamente proporcional ao número de tarefas concorrentes em um dado momento da execução, e diretamente proporcional à complexidade de cada tarefa. Assim, quando temos um número muito grande de tarefas pouco complexas, dizemos que a granularidade deste programa é fina. Em contraponto, a granularidade grossa ocorre quando existe um número pequeno de tarefas complexas (da Silva, 2011). A granularidade média é o meio termo entre a fina e a grossa.

A granularidade grossa é apropriada para algoritmos paralelos com alto custo nas comunicações, como em *clusters* de computadores, que utilizam a rede Ethernet para

comunicar seus computadores, e que são regulares em relação ao trabalho das tarefas, ou seja, que as tarefas estejam bem determinadas. Por outro lado, a fina é apropriada em algoritmos com carga de trabalho irregular, pois com tarefas menores é facilitada a realização de balanceamento de carga nesses algoritmos.

Em relação à plataforma CUDA, que possui dois níveis de arquitetura distribuída, duas diferentes formas de granularidades podem ser utilizadas. Para a mais externa, a granularidade grossa é indicada, pois o acesso à memória GDDR da placa gráfica é custosa, em relação ao tempo. Sendo assim, a granularidade do nível interno ao multiprocessador será definida pela frequência de comunicação entre os processadores contidos no mesmo.

3.3 PLATAFORMA CUDA

Inicialmente, a GPU era um processador de função fixa, construído sobre um pipeline gráfico que se sobressaía apenas em processamentos de gráficos em três dimensões. Desde então, a GPU tem melhorado seu hardware, com foco no aspecto programável da GPU. O resultado disso é um processador com grande capacidade aritmética, não mais restrito às operações gráficas.

Tendo em vista esse potencial computacional oferecido pelas GPUs, a empresa NVIDIA, em novembro de 2006, tornou público uma nova plataforma de computação paralela: o CUDA (NVIDIA, 2015). Esta plataforma utiliza suas GPUs para resolver diversos problemas computacionais complexos, transformando-as em GPUs de propósitos gerais (GPGPU - *General Purpose* GPU) (Oiso, Yasuda, Ohkura, & Matumura, 2011). Esta tecnologia tem ganho usuários nos campos científico, biomédico, da computação, da análise de risco e da engenharia devido às características das aplicações nesses campos, as quais são altamente paralelizáveis. Essa tecnologia tem chamado a atenção da comunidade acadêmica devido ao seu grande potencial computacional.

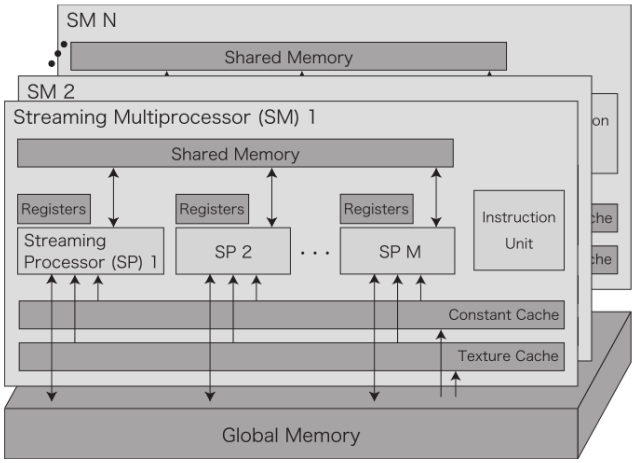
O kit de desenvolvimento CUDA possui ambientes para programação em diversas linguagens, como FORTRAN, DirectCompute, OpenACC. Contudo, a principal linguagem de programação utilizada pelos desenvolvedores desta plataforma é a linguagem C/C++ (NVIDIA, 2014), que será melhor abordada por ser a linguagem utilizada neste trabalho.

3.3.1 ARQUITETURA CUDA

Uma diferença importante entre as GPUs e as CPUs é que, enquanto as CPUs dedicam uma grande quantidade de seus circuitos ao controle, a GPU foca mais em ALUs (*Arithmetic Logical Units*), o que as torna bem mais eficientes em termos de custo quando executam um software paralelo. Consequentemente, a GPU é construída para aplicações com demandas diferentes da CPU: cálculos paralelos grandes com mais ênfase no *throughput* (largura de banda de transferência, que pode ser expressa em kbps, Mbps, etc.) que na latência (demora para a transferência de um único dado). Por essa razão, sua arquitetura tem progredido em uma direção diferente da CPU.

As GPUs CUDA são formadas por um conjunto de multiprocessadores, chamados *streaming multiprocessors* (SMs), que por sua vez é formado por processadores simples, chamados *streaming processors* (SPs) (Oiso, Yasuda, Ohkura, & Matumura, 2011). A Figura 3.3 apresenta a organização dos SMs e SPs e de suas memórias, onde percebe-se que há 3 conjuntos de memórias: a memória global, compartilhada entre todos os SMs; as memórias internas aos SMs, constituídas pelas memórias cache (de variáveis constantes e texturas), que armazenam cópias do que há na memória global, e um conjunto de memória manipulável pelos SPs dentro do SM, chamada memória compartilhada (*shared memory*); e os registradores individuais de cada SP.

Figura 3.3 - Arquitetura CUDA



A memória global se diferencia dos outros dois grupos de memórias por sua utilização, que se dá por meio de alocação dinâmica, o que garante que as variáveis criadas não sejam destruídas ao término de alguma thread, a menos que isso seja feito de forma explícita. Em contraponto, as variáveis criadas na memória compartilhada de um SM e nos registradores dos SPs são removidas quando a execução das threads que as criaram terminam.

No desenvolvimento em CUDA, a CPU e a memória principal do computador são chamadas *host*, enquanto a GPU é chamada *device*. Os SMs também são chamados CUDA cores e é entre estes que os blocos de threads são distribuídos para serem executados. Nas GPUs CUDA cada SM possui um número múltiplo de 32 de SPs, assim cada bloco de threads divide-se, a nível de execução, em sub-blocos que contém 32 threads, chamado *warp* (Oiso, Yasuda, Ohkura, & Matumura, 2011) e cada *warp* é executado por inteiro em um SM, ou seja, os SMs podem executar 32 threads (pertencentes ao mesmo bloco) simultaneamente.

3.3.2 KERNELS

O desenvolvimento para CUDA possui característica heterogênea, em um mesmo programa pode-se colocar instruções para o *host* e para o *device*. Entretanto, a execução deste programa começa no *host*, sendo necessário que se faça uma chamada para a execução no *device*. Esta chamada é feita por uma função especial, denominada *kernel*.

Pode-se definir diversas funções *kernel* em um mesmo programa. O *kernel* é responsável por determinar um procedimento a ser executado e informar o número de blocos e threads que executarão suas instruções no momento de sua chamada pela CPU. Como as instruções contidas no *kernel* serão executadas por todas as *threads* criadas, esta estrutura de programação paralela pode ser considerada SIMD. A Expressão (3.1) mostra a estrutura da assinatura de um *kernel*.

```
__global__ void nome_do_kernel(atributos_de_entrada)      (3.1)
```

O identificador `__global__` informa ao compilador de C/C++ para CUDA que a função a seguir é um *kernel*. Esta função não pode conter valores de saída, portanto utilizasse o valor *void*. Os atributos de entrada podem ser de tipos primitivos, de classes definidas pelo

programador ou podem ser ponteiros para algum dado copiado anteriormente para a memória global do *device*.

A chamada de um *kernel* é feita dentro do código do programa destinado ao *host*, como dito anteriormente, e sua execução é retornada imediatamente a este, fazendo com que outro *kernel* possa ser executado antes da finalização do anterior. A Expressão (4.2) apresenta a estrutura de uma chamada a um *kernel*.

```
nome_do_kernel<<<NB,TPB>>>(variaveis_de_entrada);        (3.2)
```

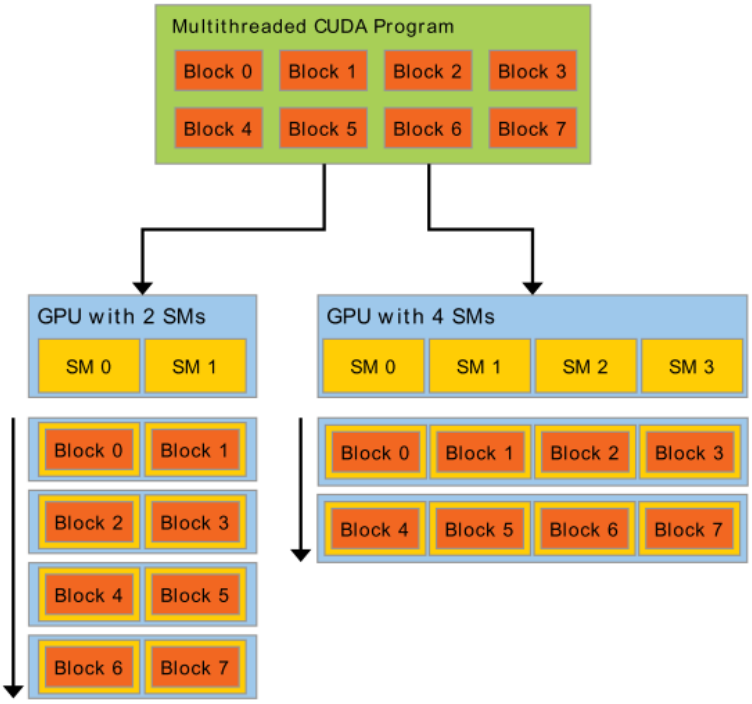
Nesta expressão, os parâmetros `NB` e `TPB` representam, respectivamente, o número de blocos de threads que executarão o código contido no corpo do *kernel* e o número de threads existentes em cada bloco. Esta estrutura constitui um grid de threads, que será melhor explicado a seguir.

3.3.3 THREADS, BLOCKS E GRIDS

Quando é feita uma chamada para a execução de um *kernel*, constrói-se uma estrutura chamada *grid*. Esta estrutura é formada pelo conjunto de threads que executarão o código contido no *kernel* chamado. Estas, por sua vez, são separadas em blocos de threads com números iguais de threads.

Cada bloco é executado em um SM e cada uma de suas threads é executada por um SP. A programação para blocos e threads torna transparente a utilização dos SMs, além de torná-la escalável, pois os blocos de threads são automaticamente distribuídos entre os SMs disponíveis no *device* a nível de execução. A Figura 3.4 apresenta um exemplo da escalabilidade proporcionada pela programação em grid da plataforma CUDA.

Figura 3.4 - Escalabilidade da plataforma CUDA



Nesta figura, temos um exemplo da execução de um programa CUDA que utiliza 8 blocos em duas GPUs diferentes, uma com 2 SMs e outra com 4. Na primeira, cada SM fica responsável pela execução de 4 blocos. Na segunda, cada SM executa apenas 2 blocos. Os blocos são divididos igualmente entre os SMs disponíveis na GPU, para que se otimize sua utilização.

Assim como mascara a utilização dos SMs, o grid também torna transparente a utilização da memória do *device*. Com isso a memória compartilhada em um SM e os registradores de um SP são mapeados, respectivamente, em memória por bloco e memória por thread. Entretanto um grid em si não possui memória reservada, sendo utilizada a memória global para a troca de dados entre os blocos.

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo, foi apresentada a plataforma CUDA, suas características e a contextualização de conceitos de programação paralela à mesma. Que a mesma possui estrutura de programação SIMD, que é uma característica que auxilia na implementação dos

algoritmos alvos deste estudo nesta plataforma. Contudo, vimos que esta possui arquitetura de memória compartilhada em dois níveis (SMs e SPs), o que corrobora com a motivação do presente estudo, de que é preciso avaliar estratégias de paralelização destes algoritmos que se adequem à arquitetura GPU.

O próximo capítulo apresentará alguns trabalhos que utilizaram a plataforma CUDA para implementar versões paralelas de algoritmos bioinspirados, quais as características comuns às estratégias de paralelização utilizadas e os resultados obtidos por cada um.

4 TRABALHOS CORRELATOS

4.1 CONSIDERAÇÕES INICIAIS

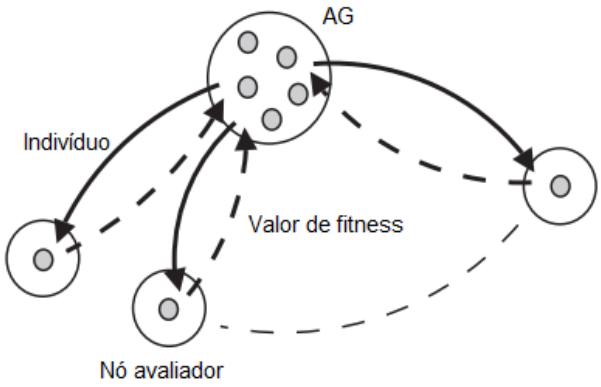
Neste capítulo serão apresentados alguns trabalhos que implementam versões paralelas dos algoritmos genéticos e PSO para clusters de computadores, utilizando o paradigma *message passing*, assim como para a plataforma CUDA em específico. Na maioria destes trabalhos avalia-se principalmente o ganho em performance das versões paralelas dos algoritmos em relação a suas versões sequenciais, com exceção do trabalho de Moraes (2011), que faz a avaliação entre três diferentes implementações paralelas do algoritmo PSO.

4.2 ALGORITMOS GENÉTICOS PARALELOS

Melab, Cahon, & Talbi (2006) afirmam que existem basicamente três grandes modelos de paralelização de algoritmos evolucionários: a avaliação em paralelo da população; a avaliação distribuída de uma única solução; e o modelo baseado em ilhas.

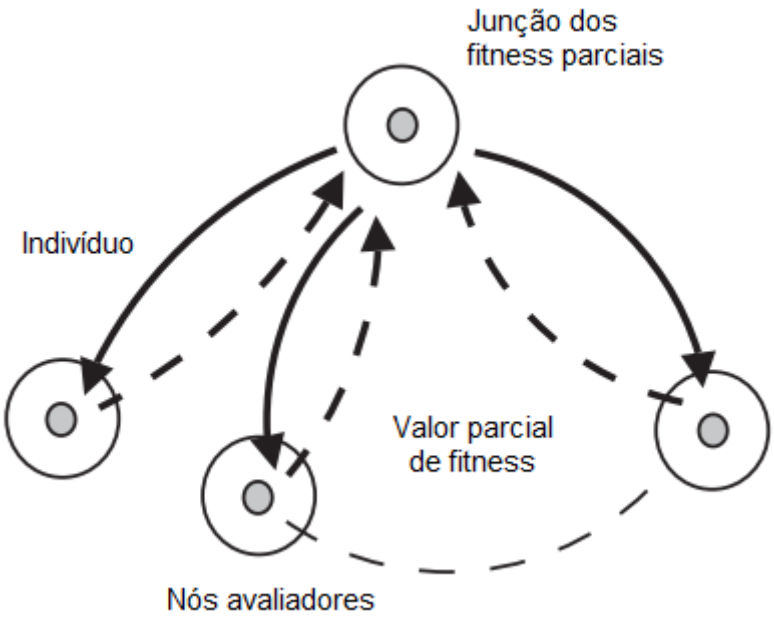
O primeiro modelo segue o princípio de Mestre-Escravo, onde o processo Mestre aplica as operações de seleção, cruzamento e mutação, enquanto os processos Escravos fazem a avaliação das soluções encontradas em paralelo, retornando os resultados para o Mestre. Em geral, esta é a solução menos eficiente em termos de tempo de execução, mas pode ser aperfeiçoada paralelizando-se os operadores genéticos. Neste caso, o processo Mestre passa a servir apenas para a sincronização da população. A Figura 4.1 apresenta este esquema.

Figura 4.1 - AG com avaliação em paralelo dos indivíduos



O segundo modelo, apresentado pela Figura 4.2, também segue o princípio de Mestre-Escravo. Entretanto, neste modelo, a avaliação de uma única solução é distribuída entre os processos Escravos, que computam parte da avaliação, ficando a cargo do processo Mestre a combinação destes resultados para encontrar o resultado completo da avaliação. Em geral, este modelo é utilizado apenas quando a avaliação das soluções pode ser paralelizada.

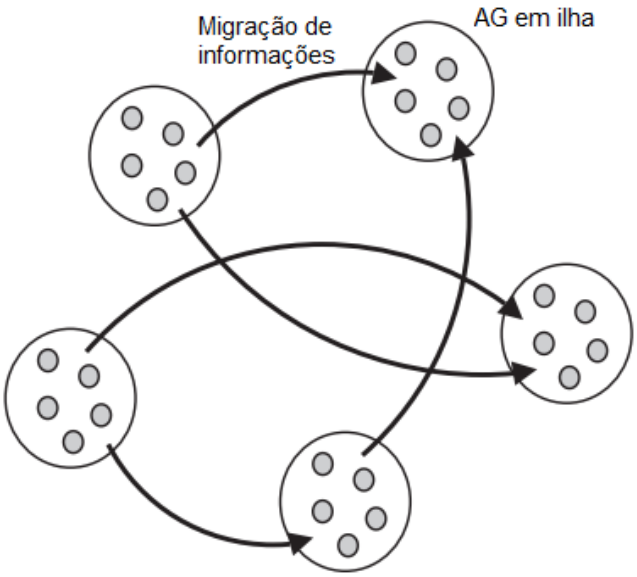
Figura 4.2 - AG com avaliação distribuída de um indivíduo



O último modelo segue um padrão cooperativo, não mais de mestre e escravos, onde várias populações são criadas e separadas em “ilhas”, como apresentado na Figura 4.3. Em

cada ilha, os operadores genéticos são aplicados independentemente das outras e, a cada determinado número de gerações, as melhores soluções são migradas para outra ilha (população), efetuando-se assim, a cooperação entre as mesmas. O objetivo deste modelo é aumentar a exploração do algoritmo no espaço de busca, já que cada população vai evoluir individualmente, sem prejudicar a convergência do algoritmo através da migração das melhores soluções.

Figura 4.3 - AG baseado em ilhas



A seguir serão apresentados alguns trabalhos que utilizaram a plataforma CUDA para a implementação de versões paralelas do AG. Procurou-se classificar os trabalhos desenvolvidos entre os três modelos apresentados acima, com base em suas características, para facilitar a compreensão dos mesmos.

4.3 ALGORITMOS GENÉTICOS EM CUDA

Em Feier, Lemnaru, & Potolea (2011) é implementado um algoritmo genético baseado em ilhas em plataforma CUDA. Neste trabalho, cada ilha é alocada em um bloco de threads, e cada indivíduo (solução) em uma thread, para a execução na GPU. A CPU foi utilizada para sincronizar as ilhas e efetuar a migração de soluções. Este algoritmo genético

foi aplicado ao problema da mochila com 1, 16, 32 e 128 ilhas, para os tamanhos de população de 32 e 128 indivíduos por ilha.

Nos resultados dos testes, observou-se que o aumento da performance do algoritmo é diretamente proporcional ao crescimento do número de ilhas. Entretanto, o mesmo não ocorre para o tamanho da população por ilha, pois em quase todos os testes onde utilizou-se 32 indivíduos por ilha o resultado foi superior aos obtidos com 128 indivíduos por ilha, para o mesmo número de ilhas.

Oiso, Yasuda, Ohkura, & Matumura (2011) implementaram um algoritmo genético semelhante ao primeiro modelo apresentado por Melab, Cahon, & Talbi (2006), onde a avaliação das soluções é feita em paralelo, para CUDA. Em tal implementação, dois indivíduos são alocados por bloco de threads e cada gene em uma thread. A seleção dos indivíduos para os blocos de threads é feita de forma aleatória. Após a seleção, são aplicados os operadores de cruzamento e de mutação. Dois indivíduos são retornados desta operação para a população e são selecionados da seguinte forma: o melhor indivíduo, entre pais e filhos, e um indivíduo selecionado por roleta, dentre os que sobraram.

Este algoritmo genético foi testado em quatro funções comumente utilizadas em *benchmarking* de algoritmos genéticos. Seus resultados foram comparados com os resultados obtidos através da implementação sequencial do algoritmo genético. Nos resultados, observou-se que a versão paralela foi até 6 (seis) vezes mais rápida que a versão sequencial.

4.4 ALGORITMOS DE ENXAME DE PARTÍCULAS PARALELOS

Assim como descrito para algoritmos genéticos, Moraes (2011) descreve três formas de paralelizar PSO. Entretanto, estas estratégias foram planejadas para o modelo de sistema distribuído Mestre-Escravo. Tais algoritmos são classificados quanto ao momento da atualização do melhor global, que pode ser imediata para cada partícula ou feita de uma vez para todo o enxame a cada revoada (iteração), e quanto a sincronização do enxame, que pode ser síncrono (todas as partículas estarão na mesma iteração do algoritmo) ou assíncrono (a partícula A pode estar na iteração 5, enquanto a partícula B pode estar na iteração 3).

O *Swarm Update Parallel* PSO (SU-PPSO) é o modelo de PSO paralelo que utiliza a atualização do melhor global para todo o enxame, ao fim de cada iteração, ou seja, apenas

quando todas as partículas se moverem, o melhor global é atualizado. Dado esta característica, o enxame está sempre sincronizado, ou seja, todas as partículas estão na mesma iteração do algoritmo.

O *Immediate Update Parallel* PSO (IU-PPSO) é o modelo de PSO paralelo que utiliza a atualização imediata do melhor global, ou seja, assim que a partícula é movida. Entretanto, ao fim de cada iteração, o enxame é sincronizado, para que todas as partículas continuem na mesma iteração do algoritmo. Este modelo, alcança resultados melhores que o modelo anterior. Isto ocorre por conta da atualização imediata do melhor global, logo, a próxima partícula terá a posição e a velocidade atualizadas com base no novo ótimo global.

O *Asynchronous and Immediate Update Parallel* PSO (AIU-PPSO) é o modelo de PSO paralelo que aplica a atualização imediata e assíncrona melhor global, ou seja, cada partícula realiza as iterações do algoritmo independentemente das outras. Nos testes, este modelo demonstrou um desempenho elevado, com eficiência superior a 90% em todos os testes.

A seguir serão apresentados alguns trabalhos que utilizaram a plataforma de programação paralela CUDA, da NVIDIA, para a implementação de versões paralelas do PSO. Como na seção anterior, procurou-se classificar os trabalhos desenvolvidos entre os modelos apresentados anteriormente, com base em suas características. Entretanto, um dos trabalhos apresentados na próxima seção pôde ser classificado, também, em um modelo apresentado na seção anterior, para algoritmos genéticos.

4.5 ENXAME DE PARTÍCULAS EM CUDA

Zhou & Tan (2009) implementaram uma versão síncrona de PSO paralelo em CUDA, onde as velocidades e posições das partículas só são atualizadas após computar os ótimos locais e globais de todo o enxame. Dado esta característica, podemos classificar este PSO em CUDA como sendo um SU-PPSO, pois todo o enxame estará na mesma iteração do algoritmo. Neste trabalho, as partículas foram alocadas cada uma em uma thread, sendo que essas threads foram distribuídas em blocos de threads de tamanho fixo, para que as operações sejam distribuídas entre os CUDA cores.

Os testes foram realizados em quatro funções de *benchmark*, onde alcançou resultados melhores, em termos de performance, em todos os testes, em relação ao PSO clássico (implementado apenas para CPU). Observou-se, também, que a eficiência de paralelização aumentava proporcionalmente ao tamanho do enxame, o que corrobora com os resultados apresentados no trabalho de Feier, Lemnaru, & Potolea (2011).

Wachowiak & Foster (2012) apresentaram uma proposta de implementação de um PSO paralelo assíncrono em CUDA. Este trabalho se assemelha ao modelo baseado em ilhas, apresentado anteriormente, pois o enxame é separado entre os blocos de threads e internamente a este o modelo AIU-PPSO é aplicado, e a cada determinado número de iterações do algoritmo o enxame é retornado à CPU para sua sincronização. O autor, entretanto, não especificou como foi avaliado o número de iterações executadas pelo algoritmo dentro da GPU.

Utilizou-se três funções de benchmark para avaliar esta implementação. Em todos os testes a implementação em GPGPU foi superior em performance à implementação clássica. Contudo, não observou-se ganhos consideráveis à performance quando aumentou-se o tamanho do enxame.

4.6 OTIMIZAÇÃO E ROTEAMENTO

O problema de realizar o roteamento em redes com múltiplas condições pode ser abordado de diferentes formas, dependendo da variável de interesse. Por exemplo, é possível reduzir o custo final do cliente, escolhendo meios de transmissão de menor custo; e do ponto de vista operacional, é possível minimizar o uso dos recursos de rede e até mesmo levar em consideração o consumo final de energia elétrica do sistema, além do custo final, aplicável à redes de longa distância.

Tratando-se da redução do custo final ao cliente, por exemplo, Lin, Gan, & Liang (2010) define uma nova arquitetura para reduzir o custo de roteamento de chamadas para Femtocells, que é uma solução para melhorar a cobertura de celulares localizados em um único local com menor custo. A arquitetura combina tanto chamadas de usuários normais quanto usuários de Femto, sem modificar os protocolos dos sistemas de comunicação móveis

existentes. No entanto, é possível perceber que essa é uma técnica complexa e não é de fácil difusão em países como o Brasil.

Em relação aos recursos de rede, os estudos tendem ao desenvolvimento de novos protocolos de roteamento, como o caso de Qureshi, Weber, Balakrishnan, Guttag, & Maggs (2009), que propõe uma forma de rotear as requisições de clientes em grandes redes de servidores distribuídos geograficamente em diferentes fuso-horários. Sua solução pode encontrar tanto a resposta com menor custo quanto a de menor distância, dependendo da distância mínima necessária e usando a diferença de preços por hora como vantagem. Entretanto, não leva em consideração a questão de prever demandas de clientes e possui um tempo de resposta médio às mudanças de preço do mercado.

Outros trabalhos tratam do consumo de energia. Por exemplo, Wu et al. (2009), minimiza o consumo de energia em redes RWA, reutilizando a mesma fibra o máximo possível ao longo de um mesmo caminho, ao invés de distribuir rotas em caminhos e fibras disponíveis. Sua proposta leva em consideração apenas a diminuição do consumo de energia. No mesmo contexto, Çavdar et al. (2011) propõe outro forma de roteamento que minimiza o consumo de energia e custo de uma rede ótica WDM. A solução, chamada de RWA-Bill, realiza o roteamento considerando a localização e os preços desses locais, explorando as duas condições para encontrar matematicamente o caminho mais barato.

Os trabalhos supracitados têm como característica em comum recaírem em um problema de otimização com restrições. Por exemplo, em (Çavdar, 2011), utiliza-se de simplex para sua resolução. No entanto, ressalta-se que tal técnica, além de seu alto custo computacional, requer um tempo de desenvolvimento para que o modelo gerado seja adaptado ao método de resolução (Raidl, G. R.; Puchinger, 2008).

Neste contexto, o presente trabalho apresenta um método de resolução para problemas de roteamento com um grande número de restrições, que analisa o problema de acordo com a demanda exigida à rede, através da combinação de algoritmos já conhecidos de busca de melhores caminhos em grafos e algoritmos bioinspirados.

4.7 CONSIDERAÇÕES FINAIS

Os trabalhos apresentados neste capítulo serviram de inspiração e deram embasamento para o desenvolvimento do presente estudo. Tais trabalhos apresentaram estratégias de paralelização dos algoritmos alvos deste estudo, assim como formas de implementá-las sob a plataforma CUDA. Em todos os trabalhos envolvendo tal plataforma, concluiu-se que esta é uma alternativa viável e de baixo custo para o aumento de performance nestes algoritmos de busca, principalmente quando o custo computacional é extremamente elevado.

Contudo, não há um consenso sobre qual estratégia utilizar para paralelizar estes algoritmos nesta plataforma. Ficando a cargo de cada autor o trabalho de desenvolver seu próprio método, o que dificulta a utilização da plataforma para a implementação dessa classe de algoritmos, tanto para fins acadêmicos, quanto para o mercado.

Sendo assim, procurou-se agrupar as implementações observadas neste capítulo, a fim de padroniza-las. Como resultado, foram obtidas 3 (três) estratégias de paralelização principais, que serão apresentadas no próximo capítulo. Estas serão avaliadas a partir de um problema real, o problema de roteamento em redes ópticas, para que se determine quais se adaptam melhor a cada algoritmo implementado plataforma CUDA.

5 ESTRATÉGIAS DE PARALELIZAÇÃO DE ALGORITMOS BIOINSPIRADOS

5.1 CONSIDERAÇÕES INICIAIS

Com base nos trabalhos expostos no capítulo anterior, observou-se a possibilidade de agrupar as estratégias de paralelização apresentadas, tanto para AG quanto para PSO, em três grupos básicos. Tais grupos foram separados de acordo com a sincronia da população, ou seja, se todos os indivíduos estão na mesma iteração ou não. As estratégias escolhidas foram: Paralelismo Síncrono; Modelo de Vizinhança; e Modelo Baseado em Ilhas. Estas serão descritas nas seções seguintes.

5.2 PARALELISMO SÍNCRONO

A seguinte estratégia objetiva manter a sincronia entre todas as possíveis soluções encontradas até o momento pelo algoritmo. Isso é feito por meio da sincronização entre os processos sempre que for necessária a troca de informações e ao fim de cada iteração do algoritmo. Sendo assim, a próxima operação só será executada após toda a população finalizar a operação anterior. Cada etapa dos algoritmos é realizada para todo o conjunto de soluções simultaneamente, de maneira que cada thread execute as operações em uma única possível solução (indivíduo ou partícula), utilizando-se de 32 threads em cada bloco. Este número de threads foi escolhido com base nos resultados obtidos no trabalho de Feier, Lemnaru & Potolea (2011), apresentado no capítulo anterior, e de acordo com as características da arquitetura das GPUs CUDA.

A população ou enxame é alocado na memória global do *device*, por esta utilizar alocação dinâmica, ou seja, o espaço alocado não será liberado com o término da execução dos *kernels*, e por garantir que qualquer thread possa acessar todo o conjunto de soluções.

As características apresentadas servem tanto para a implementação em AG quanto em PSO, desta estratégia. Contudo, os algoritmos possuem operações diferentes, portanto houve especificidades nas implementações desta estratégia para cada algoritmo, as quais são apresentadas a seguir.

5.2.1 IMPLEMENTAÇÃO AG

Para esta estratégia, os operadores genéticos foram implementados em dois *kernels*, por conta do acesso diferenciado à memória global requerida em cada um. O primeiro, destinado somente à seleção dos indivíduos para o cruzamento, onde a seleção dos indivíduos é feita por meio de torneio, necessita acessar toda a população inalterada. O segundo, que incorpora as etapas de cruzamento e mutação, requer direito de escrita na mesma.

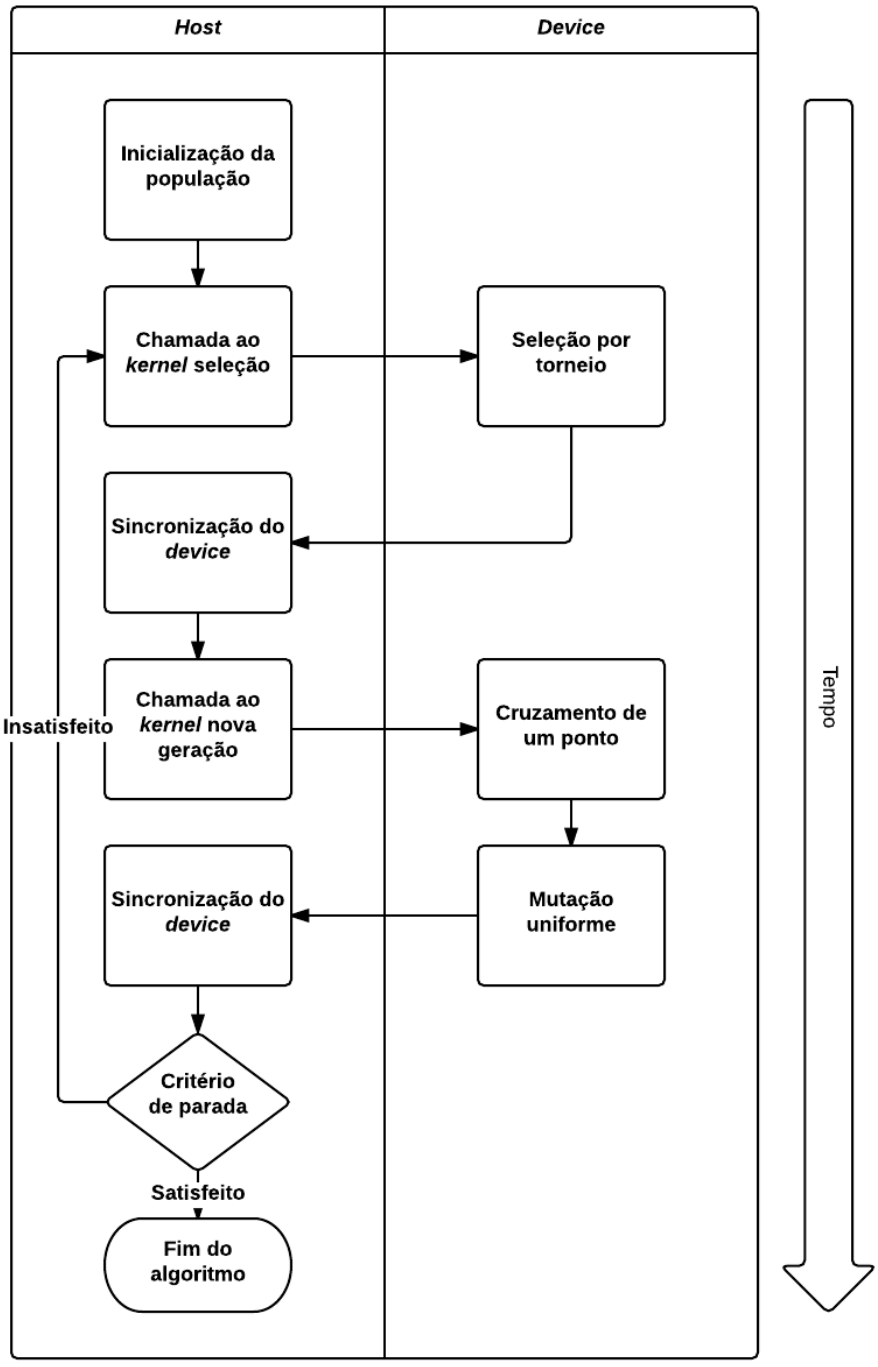
Ao término da execução do primeiro *kernel*, um vetor contendo os indivíduos selecionados é retornado para a memória global. Dado que para valores altos do número de indivíduos que participarão do torneio há menor possibilidade dos menos aptos serem selecionados e para valores muito baixos a seleção passa a ser quase aleatória, optou-se por utilizar 3 indivíduos por torneio, a fim de manter a diversidade da população. Entretanto, convém destacar que este valor depende do tamanho da população utilizada no algoritmo.

Em relação ao segundo *kernel*, optou-se por deixar as operações de cruzamento e mutação juntas por conta da natureza da segunda, que não necessita das informações de outros indivíduos da população para ser realizada, ou seja, não requer que a população esteja na mesma etapa do algoritmo.

O método de cruzamento escolhido para este trabalho foi o cruzamento de um ponto. Com relação a mutação, optou-se por alterar apenas um gene do indivíduo selecionado para tal. O gene a ser alterado é escolhido de maneira uniformemente aleatória e o valor deste é alterado por mutação uniforme.

A Figura 5.1 apresenta como a implementação foi feita. As instruções contidas no *device* são executadas por múltiplas threads simultaneamente, onde cada uma executa para conjuntos de dados diferentes, ou seja, cada thread realiza apenas um torneio, cruzamento ou mutação e a junção de todas as operações torna-se a aplicação o operador sob a população. Ao fim de cada *kernel*, a execução é retornada ao *host*, para sincronizar o *device (*aguardar que todas as threads terminem sua execução).

Figura 5.1 - Implementação de AG por Paralelismo Síncrono

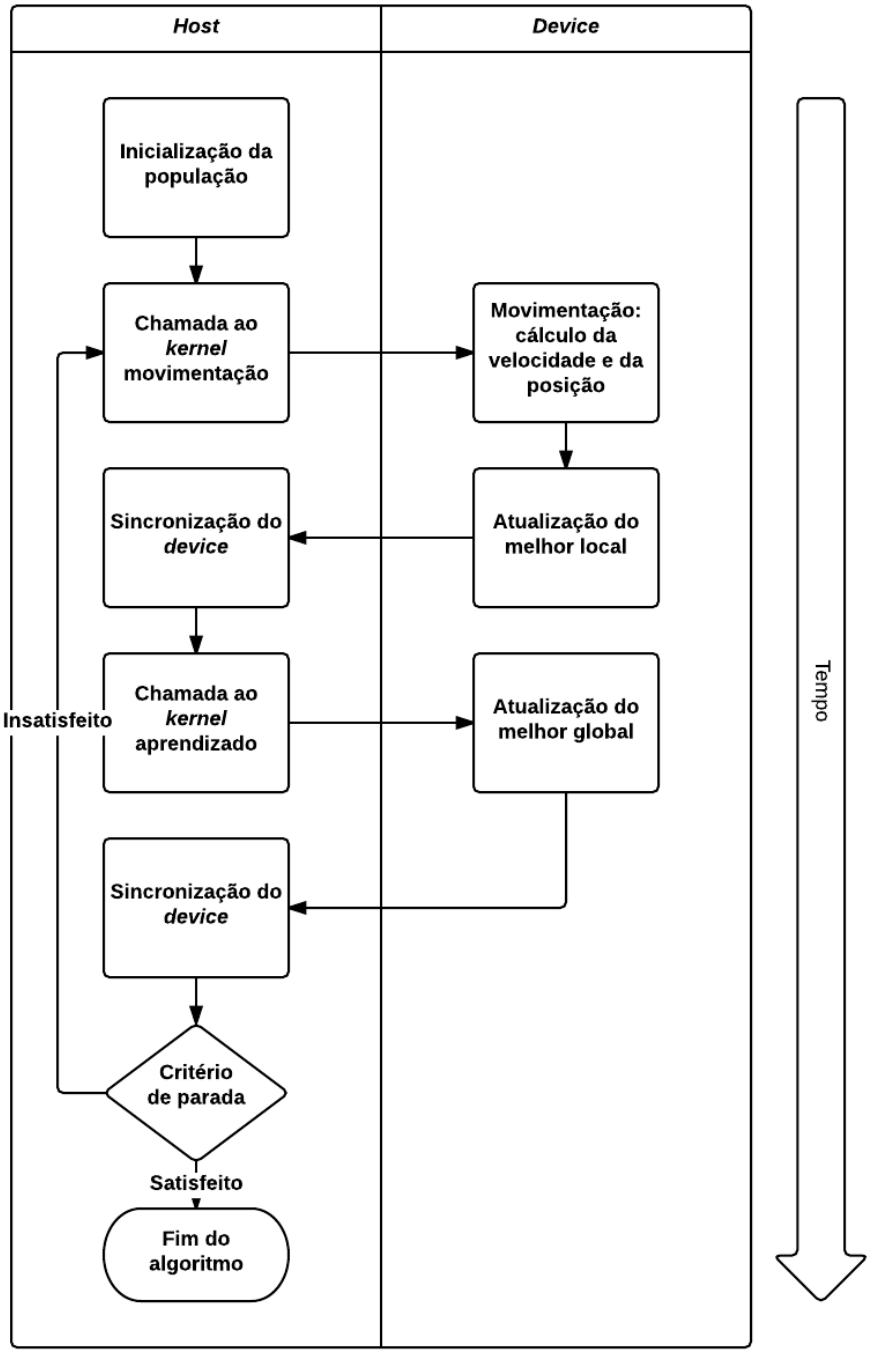


5.2.2 IMPLEMENTAÇÃO PSO

Para a implementação do PSO, utilizou-se também, dois *kernels*, para que a atualização do melhor global seja realizada somente ao fim da iteração do algoritmo. O primeiro executa as etapas de atualização das partículas, que envolve: atualização da

velocidade, da posição e do melhor local, consequentemente a etapa de avaliação do enxame. O segundo realiza a etapa de atualização da melhor posição encontrada pelo enxame, ou seja, o melhor global. A Figura 5.2 apresenta o fluxograma desta implementação.

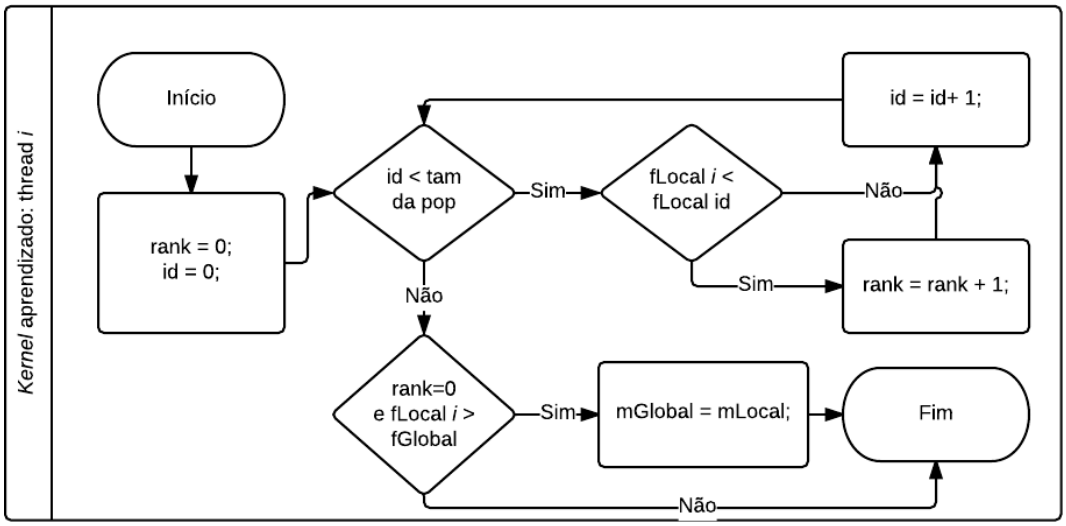
Figura 5.2 - Implementação de PSO por Paralelismo Síncrono



Como o melhor global é comum a todo o enxame, este é armazenado na memória global do *device*. Portanto, fez-se necessário a utilização de uma estratégia que implementasse a exclusão mútua no acesso à este espaço de memória, porém a utilização de semáforos, e outras estruturas comumente utilizadas em programas paralelos, é desencorajada na programação para CUDA (NVIDIA, 2014).

Sendo assim, optou-se por utilizar um algoritmo que encontrasse o melhor global baseado no algoritmo de ordenamento *bubble sort*. Na implementação, cada thread fica responsável por uma partícula e verifica quantas partículas possuem valor de aptidão melhor que a sua, assim as partículas ficam ordenadas como em um *ranking*. Ao final, apenas a thread com maior aptidão acessa o espaço de memória para atualizar o melhor global. A Figura 5.3 apresenta o fluxograma do *kernel* que realiza esta operação, onde *mLocal i* e *fLocal i* são, respectivamente, o melhor local da partícula *i* e o valor de aptidão do mesmo, enquanto *mGlobal* e *fGlobal* têm o mesmo significado para o melhor global

Figura 5.3 - *Kernel* que realiza a atualização do melhor global

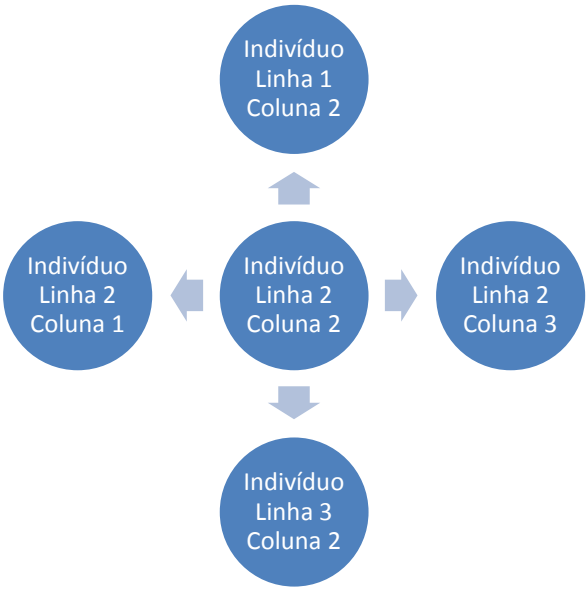


5.3 MODELO DE VIZINHANÇA

O objetivo desta estratégia é fazer com que as soluções evoluam nos algoritmos de forma assíncrona com o restante do conjunto, ou seja, as iterações dos algoritmos ocorrerão de maneira independente para cada indivíduo. Entretanto, nesta estratégia, os indivíduos trocam de informação apenas com os vizinhos mais próximos dada a topologia adotada. A

Figura 5.4 apresenta o esquema de funcionamento deste modelo, onde o processo passa informação aos vizinhos em uma topologia bidimensional.

Figura 5.4 - Troca de informações no Modelo de Vizinhança



Assim como na estratégia anterior, o conjunto de soluções também é armazenado na memória global do *device* e cada possível solução alocada para execução em uma thread. Porém, todas as etapas dos algoritmos foram implementadas em um único *kernel* para cada algoritmo, a fim garantir a corretude dos mesmos, haja vista que se fossem implementadas em mais *kernels*, a população seria sincronizada ao fim de cada um.

Para a implementação do algoritmo genético, este esquema foi utilizado na etapa de seleção de pais, pois passou-se a realizar o torneio apenas com os indivíduos vizinhos; já no caso do PSO, a única informação compartilhada pelo enxame é o melhor global.

5.4 MODELO BASEADO EM ILHAS

O modelo baseado em ilhas é caracterizado por dividir o conjunto total de possíveis soluções dos algoritmos estudados em subconjuntos, nos quais os algoritmos, em suas versões tradicionais (sequenciais), são aplicados simultaneamente. Assim, cada subconjunto

de soluções, ou ilha, se desenvolve separadamente e de maneira assíncrona em relação aos outros subconjuntos.

Nesta estratégia, o conjunto total de possíveis soluções é alocado na memória global do *device*, como nas outras. Contudo, os subconjuntos são alocados na memória compartilhada dos blocos de threads aos quais pertencerão, transformando cada bloco em uma ilha. Para tal, um único *kernel* foi implementando para cada algoritmo.

Esta estratégia, porém, possui uma etapa de sincronização entre as ilhas, a qual deve ser realizada a cada determinado número de iterações dos algoritmos nas mesmas. Isso foi implementado adicionando-se o número de iterações como parâmetro aos *kernels* desenvolvidos. Assim, sempre que as ilhas executam este número de iterações, a execução retorna ao *host* para a sincronização entre os blocos.

Com relação ao tamanho dos subconjuntos, devemos levar em consideração as características da plataforma CUDA e dos algoritmos alvos deste estudo. Para a plataforma, números de threads por bloco diferentes de 32 causam perda de desempenho, como mostrado no trabalho de Feier, Lemnaru & Potolea (2011). Em contraponto, este é um número muito pequeno de indivíduos para o bom funcionamento dos algoritmos estudados. Portanto, optou-se por utilizar 128 indivíduos por ilha, com o intuito de balancear o tempo de execução de um *kernel* inteiro e a convergência de uma ilha. Assim, torna-se possível realizar um menor número de sincronizações entre as ilhas, fazendo com que a execução do algoritmo como um todo seja mais rápida.

5.4.1 IMPLEMENTAÇÃO AG

Como o algoritmo executado pelas ilhas deve estar na sua versão tradicional, implementou-se um AG canônico com método de seleção por torneio, cruzamento com um ponto de corte e mutação uniforme de um gene, sincronizando as threads de uma mesma ilha por meio da instrução __syncthreads(), que sincroniza todas as threads em um mesmo bloco, executada após cada etapa do AG.

Como em AG as informações da população estão distribuídas por todos os indivíduos, a sincronização entre as ilhas se deu por meio do ordenamento da população inteira. Assim, sempre que o algoritmo chega à esta etapa, os indivíduos com menor aptidão passarão para

os blocos com ID maior e os indivíduos com maior aptidão passarão para os bloco com ID menor, o que acaba por sincronizar as ilhas.

5.4.2 IMPLEMENTAÇÃO PSO

A implementação do PSO se deu de forma semelhante ao AG. Entretanto, o compartilhamento da melhor posição global encontrada pelo enxame é suficiente para sincronizar as ilhas, dado que é a única informação conjunta utilizada na movimentação das partículas. Sendo assim, a etapa de sincronização entre as ilhas é feita apenas encontrando-se a melhor posição dentre os melhores globais das ilhas e substituindo-se este no melhor global de cada ilha. Deste modo, as partículas passarão a se mover levando em consideração o novo melhor global.

5.5 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentadas as estratégias de paralelização que serão utilizadas para a realização dos testes no presente trabalho. Observou-se que, por mais que os algoritmos utilizados sejam de classes distintas, suas formas de paralelização são as mesmas, com algumas alterações apenas com relação às informações compartilhadas pelos indivíduos em cada algoritmo.

No próximo capítulo, será apresentado o problema a ser tratado com os algoritmos estudados e o método híbrido proposto para a solução de problemas de roteamento com múltiplas requisições simultâneas e grande quantidade de restrições.

6 OTIMIZAÇÃO DE ROTEAMENTO EM REDES ÓPTICAS WDM COM ALTA ORDEM DE DEMANDAS SIMULTÂNEAS

6.1 CONSIDERAÇÕES INICIAIS

Otimizar rotas não é uma tarefa trivial; dentre estas, identificar aquelas com o menor custo e que satisfaçam um grande número de restrições é ainda mais difícil e o que de fato encontra-se na análise de cenários reais. A aplicação em logística de distribuição, seja de insumo ou de dados, é diversa, como exemplo é possível citar o roteamento de veículos, sejam eles simples, com coleta e entrega, com entregas particionadas, etc.; ou ainda, no roteamento de dados em redes de telecomunicações variadas (e.g. redes ADSL, wireless, ópticas, mesh).

Dado a amplitude e capilaridade dos problemas de roteamento com grande número de restrições, trabalhos vêm sendo desenvolvidos visando reduzir os gastos para o funcionamento de redes dessa magnitude, sobretudo no que tange à demanda de energia elétrica (Cavdar, Yayimli, & Wosinska, 2011) (Yong, Chiaraviglio, Mellia, & Neri, 2009).

Convém destacar que tais trabalhos tiveram por objetivo propor modelos para minimização, limitando-se a aplicar métodos de resolução analíticos, como o *simplex*. No entanto, devido seu alto custo computacional, a aplicação de tais métodos em grandes instâncias torna-se inviável.

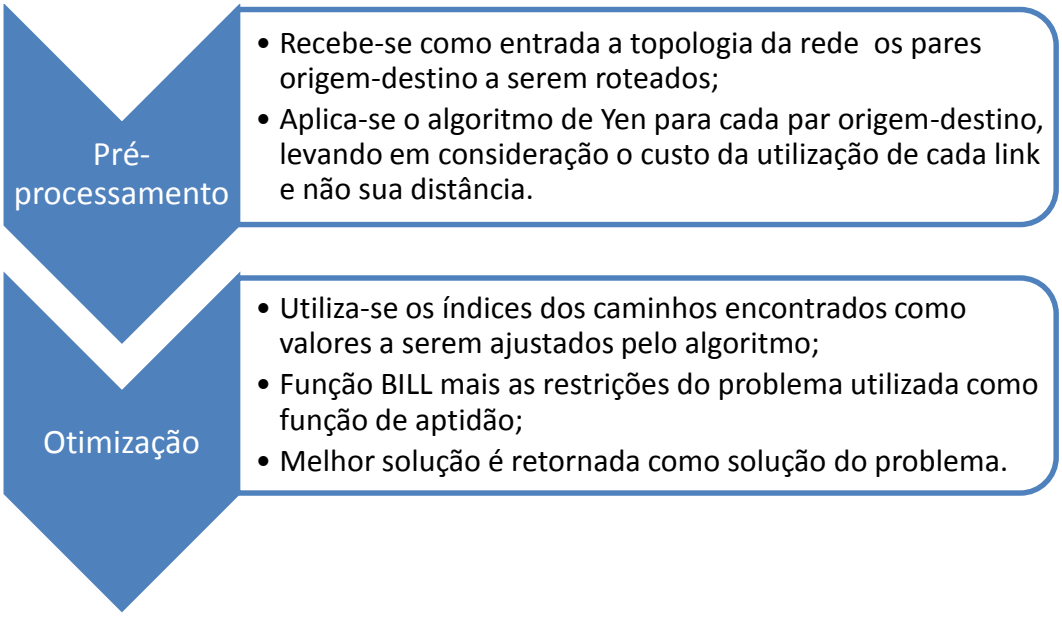
Tendo em vista esse contexto, este capítulo apresenta uma proposta de método de resolução de problemas de roteamento com uma grande quantidade de restrições. O método proposto é baseado em algoritmos bioinspirados, que aliados a outros métodos, garantem a integridade das soluções obtidas, além de sua proximidade ao ótimo. Tal método será paralelizado utilizando a plataforma CUDA, como apresentado no capítulo anterior. Esse trabalho gerou uma publicação no XLVI Simpósio Brasileiro de Pesquisa Operacional, no segundo semestre do ano de 2014

6.2 BA-KSP – ALGORITMOS BIOINSPIRADOS COM K-MENORES CAMINHOS

O método proposto utiliza algoritmos bioinspirados, como AG e PSO, combinados ao algoritmo de Yen (Yen, 1970), que encontra os *n* menores caminhos entre dois pontos em um grafo. Este foi adaptado para levar em consideração o custo da utilização de um link na rede, ao invés da distância do mesmo. Assim, a partir da aplicação deste a cada requisição de comunicação na rede, é possível empregar o algoritmo bioinspirado na busca da combinação entre os caminhos encontrados que minimize o custo para todas as requisições feitas simultaneamente.

O método completo, denominado BA-KSP (*Bioinspired Algorithm with K-Shortest paths*), é apresentado na Figura 6.1 e possui duas etapas. A primeira aplica o algoritmo de Yen para encontrar os menores caminhos para as requisições de comunicação na rede. Nesta etapa, não se leva em consideração as capacidades da rede, quantas conexões simultâneas cada link pode comportar, nem nenhuma restrição do problema e seu resultado é um conjunto de caminhos possíveis. Na segunda etapa, aplica-se o algoritmo bioinspirado para encontrar a melhor combinação dentre os caminhos encontrados para cada requisição. Nesta etapa, todas as condições do problemas são levadas em consideração, pois influenciam na qualidade das soluções encontradas.

Figura 6.1 - Etapas do método proposto



A função BILL mencionada na Figura 6.1 é expressa pela Equação **Erro! Fonte de referência não encontrada.**, que possui dois termos. O primeiro, $w_{x,y}$, representa o custo de utilização de cada link na rede. Custo que é utilizado pelo algoritmo de Yen na primeira etapa do algoritmo. O segundo termo, $n_{x,y}$, consiste no custo de reutilização de cada link na rede, caso o mesmo esteja sendo utilizado por mais de uma requisição de comunicação no problema.

$$BILL = \sum_{\forall(x,y)} w_{x,y} + n_{x,y} \tag{6.1}$$

6.3 CENÁRIO DE TESTES

O método proposto tem por objetivo a resolução de problemas de roteamento com um grande número de restrições, onde os links da rede podem ser utilizados por mais de uma requisição de comunicação e o custo pela reutilização de um dado link seja diferente do custo de sua primeira utilização. Visando validar o método proposto, buscou-se na literatura por um estudo de caso que representa-se o cenário desejado.

Neste contexto, o trabalho de Cavdar, Yayimli, & Wosinska (2011) mereceu destaque por ser inerentemente um problema de roteamento com um grande número de restrições. Como apresentado na seção anterior, os autores propõem um modelo para redução do custo no roteamento de redes óticas em relação à conta de energia elétrica. Tais redes têm como principal característica a grande distância entre seus roteadores, consequentemente, estes ficam localizados em fusos horários distintos. Com isso, o custo pela utilização de cada nó da rede varia de acordo com sua localização e com a hora em que o mesmo está sendo requisitado. Contudo, a característica mais pertinente ao presente trabalho é o fato dos links da rede possuírem mais de um comprimento de onda, que chamaremos neste trabalho de λ (lambda), os quais podem ser utilizados cada um por uma requisição diferente de comunicação.

Todas estas variáveis foram modeladas pela autores, portanto, visando especificar a função objetivo e as restrições do problema. Dessa forma, os autores chegaram a um problema de minimização da seguinte função objetivo, apresentada pela Equação(6.2), na qual o primeiro termo diz respeito ao custo da utilização do link(m,n) isoladamente e o segundo ao custo pela utilização de larguras de bandas adicionais neste mesmo link.

$$BILL'_t = \sum_{\forall(m,n)\in E} a_{mn} l_{mn} \pi_{mt} + \sum_{\forall(m,n)\in E, m\neq s,d} \varepsilon^s L_{mn} \pi_{mt} \tag{6.2}$$

Analogamente, podemos definir os termos desta como o custo pela primeira utilização do link e o custo pela reutilização deste por outras requisições, respectivamente. As variáveis da equação são definidas como: $a_{mn}\pi_{mt}$ é o custo da utilização do link (m,n); l_{mn} indica se o link (m,n) está sendo usado; $\varepsilon^s \pi_{mt}$ é o custo pela reutilização do link (m,n); e L_{mn} indica o número de vezes em que o link (m,n) foi reutilizado.

Sendo assim, o problema em questão pode ser adaptado para tratamento utilizando-se a Equação (6.2) como função *custo* do método. A fim de simplificar o problema, 3 (três) condições foram criadas para que uma solução encontrada pelo método seja considerada válida. Estas são:

1. Cada par origem-destino deve utilizar somente um λ durante todo o percurso;

2. Cada λ é utilizada somente por um par origem-destino; e
3. Os links não podem utilizar mais larguras de banda que as disponíveis (especificadas pelo problema).

Os testes foram realizados em uma rede com 12 nós, conectados por 22 links. Considerou-se o preço do kWh para cada nó em cada uma das 24 horas do dia, utilizando dados reais, disponibilizados pelo *US Energy Information Administration* (http://www.eia.gov). Considerando que seus nós estão dispostos em 4 fusos horários, os nós que estiverem em fusos horários diferentes terão custos diferentes quando analisados no mesmo instante. A topologia da rede utilizada é apresentada na Figura 6.2 e os preços de kWh na Figura 6.3.

Figura 6.2 - Topologia da rede testada

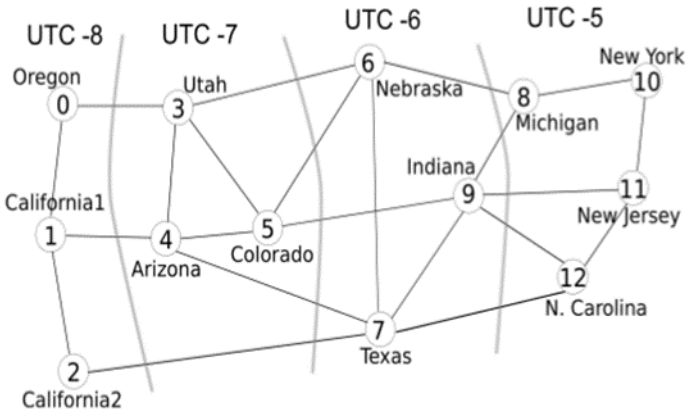
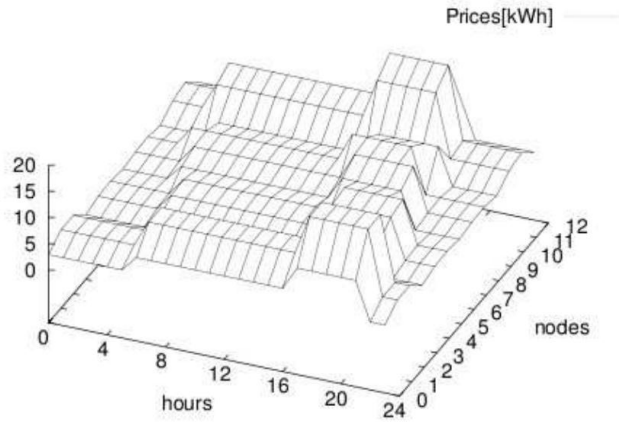


Figura 6.3 - Preços do kWh por nó e por hora



6.4 CONSIDERAÇÕES FINAIS

Neste capítulo, observou-se que o problema de roteamento não é trivial, particularmente no domínio de estudo trabalhado, onde se considera a busca pelo melhor caminho para muitos pares origem-destino simultaneamente; este sendo tratado, via de regra, utilizando técnicas de otimização tradicionais, como o *simplex*, para sua solução. Contudo, tais técnicas são computacionalmente custosas e de difícil adaptação ao problema. Sendo assim, propôs-se um método para a solução desse tipo de problema que combina técnicas de encontrar as melhores rotas em um grafo com algoritmos bioinspirados, a fim de reduzir o custo para a obtenção do melhor resultado.

Foi apresentado também, o cenário que será utilizado para a realização dos testes do método proposto sob a plataforma CUDA. Este cenário consiste em uma rede óptica WDM que se estende por 4 (quatro) diferentes fuso-horários, onde em cada um existe um preço diferenciado pela utilização da energia elétrica.

O próximo capítulo apresentará como foram realizados os testes, assim como uma análise sob os resultados obtidos, a fim de avaliar quais estratégias melhor se adaptam a cada algoritmo bioinspirado testado, bem como se o método de solução proposto é eficiente para a solução de problemas de roteamento em redes com múltiplas requisições simultâneas de comunicação.

7 TESTES E RESULTADOS

7.1 CONSIDERAÇÕES INICIAIS

Este capítulo tem por objetivo descrever a realização dos testes para o presente trabalho, no qual procura-se verificar a melhor estratégia de paralelização de AG e PSO na plataforma CUDA para o problema de roteamento em redes ópticas WDM com grande número de requisições de comunicação. Para isso, expor-se-á os resultados obtidos de maneira a realizar este objetivo e verificar se há grande diferença na utilização das estratégias de paralelização entre os algoritmos estudados.

Testes foram realizados, também, com funções comumente utilizadas em *benchmarking* de algoritmos de otimização, com o intuito de verificar o comportamento geral e específico da plataforma e das estratégias de paralelização.

7.2 TESTES EM FUNÇÕES DE *BENCHMARKING*

Três funções comumente utilizadas em *benchmarks* de algoritmos genéticos foram escolhidas como funções objetivos dos algoritmos implementados para os testes. Estas foram retiradas do trabalho de Dieterich & Hartke (2012), no qual os autores fazem uma análise nas funções mais utilizadas para estes *benchmarks*, e são apresentadas na Tabela 7.1.

Tabela 7.1 - Funções de *benchmarking*

Nome	Função
Ackley	$f(\vec{x}) = -20 \times exp\left(-0.22\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$
Rastrigin	$f(x_0, \cdots, x_n) = 10 \times n + \sum_{i=0}^{n}\left\{\begin{array}{c} x_i > 5.12 ou x_i < -5.12 : 10x_i^2 \\ -5.12 \leq x_i \geq 5.12 : x_i^2 - 10\cos(2\pi x_i)\end{array}\right\}$
Schwefel	$f(x_0, \cdots, x_n) = 418.9829 \times n + \sum_{i=0}^{n}\left\{\begin{array}{c} x_i > 500 ou x_i < -500 : 0.02x_i^2 \\ -500 \leq x_i \geq 500 : -x_i^2\sin(\sqrt{\lvert x_i\rvert})\end{array}\right\}$

7.2.1 CONFIGURAÇÕES DE HARDWARE E SOFTWARE

Para a realização dos testes sob as funções de *benchmarking*, utilizou-se um PC com sistema operacional Ubuntu 12.04 LTS. Suas configurações de *host* e *device* são apresentadas a seguir:

- Processador: Intel Core i5-2310 CPU @ 2.90GHz x 4;
- Memória RAM: 4 GB;
- GPU: NVIDIA GeForce 8400GS, 16 núcleos CUDA;
- Versão do CUDA: 5.0.

7.2.2 PARÂMETROS DOS ALGORITMOS

Tendo em vista que o objetivo deste trabalho não é desenvolver algoritmos especializados na otimização das funções apresentadas, realizou-se testes prévios a fim de ajustar os parâmetros tanto dos AGs como dos PSOs de forma que estes consigam convergir para a solução em todas as funções utilizando os mesmos parâmetros em todas as estratégias de paralelização.

Considera-se que o algoritmo convergiu para a solução quando este encontra um valor igual ao ótimo pelo menos até a quarta casa decimal. A Tabela 7.2 e a Tabela 7.3 apresentam os parâmetros utilizados pelos AGs e pelos PSOs, respectivamente.

Tabela 7.2 - Parâmetros AG *benchmarking*

Parâmetro	*Valor*
Tamanho da população	1024
Taxa de cruzamento	80%
Taxa de mutação	20%

Tabela 7.3 - Parâmetros PSO *benchmarking*

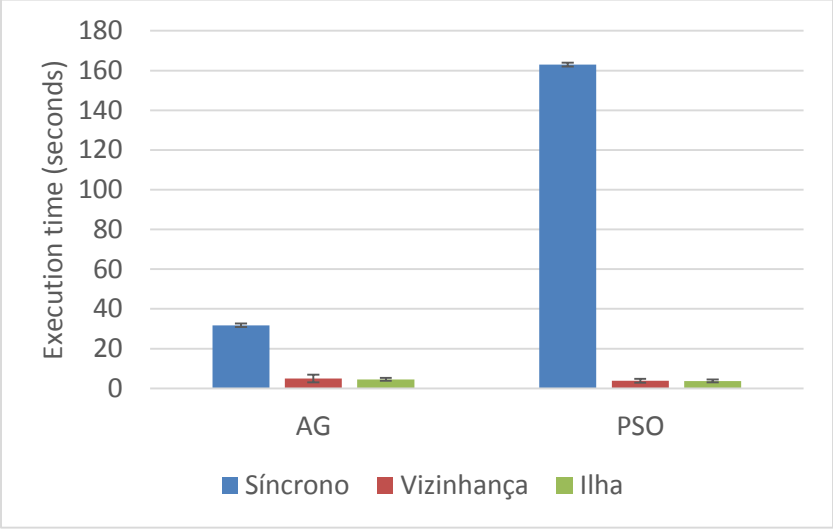
Parâmetro	*Valor*
Tamanho do enxame	1024
Inércia	1
Cognitivo	0,5
Social	0,5

Utilizou-se como critério de parada dos algoritmos o número de iterações igual a 100.000 (cem mil), para que se possa realizar uma comparação em termos de tempo de execução justa entre todas as configurações de testes feitas. Para as estratégias baseadas em ilhas, optou-se por realizar 10 sincronizações em toda a execução do algoritmo, ou seja, esta etapa é realizada a cada 10.000 (dez mil) iterações.

7.2.3 RESULTADOS

Os dados gerados a partir dos testes representam o tempo de execução em milissegundos dos mesmos. Estes serão utilizados como parâmetro de comparação para a avaliação das estratégias utilizadas. Na Figura são apresentadas as médias dos tempos de execução do AG e do PSO, respectivamente, para cada estratégias de paralelização.

Figura 7.1 - Tempos de execução AG e PSO para as funções de *benchmarking*



É possível observar, a partir deste gráficos, que a estratégia de paralelização síncrona possui desempenho inferior às outras duas estratégias, principalmente para o PSO, no qual obteve tempos de execução até 30 vezes maior que as outras estratégias. Entretanto, dado o comportamento estocástico dos algoritmos testados, outra técnica de avaliação deve ser aplicada para que se verifique qual a melhor estratégia de paralelização, além da análise de gráficos.

Para este fim, optou-se por utilizar o *Wilcoxon signed-rank test* (Wilcoxon, 1945). Este é um teste estatístico de hipóteses utilizado para comparar dois conjuntos relacionados de exemplos a fim de verificar se suas distribuições são equivalentes. Tal teste foi realizado para AG e para o PSO, entre as estratégias assíncrona e baseada em ilhas, com um nível de confiança de 95%. Para o AG, o resultado obtido confirmou que não há grandes diferenças entre a utilização das duas estratégias de paralelização. O resultado obtido pelo teste para o PSO mostrou que a estratégia baseada em ilhas é superior à estratégia assíncrona com 95% de confiança.

7.3 TESTES EM REDES ÓPTICAS WDM

Os parâmetros dos algoritmos foram fixados para todas as implementações, de maneira que a comparação fique fidedigna, tanto em relação a tempo de execução, como resultado obtido. Assim, a Tabela 7.4 e a Tabela 7.5 apresentam os parâmetros dos AGs e PSOs. Tais parâmetros foram escolhidos por apresentarem os melhores resultados em testes prévios apenas com as versões sequenciais dos algoritmos. Em relação ao tamanho da população, especificamente, foi escolhido um valor próximo a 10.000 (dez mil) que fosse múltiplo do número de núcleos CUDA na placa de vídeo testada (apresentada à frente).

Tabela 7.4 - Parâmetros AG

Parâmetro	*Valor*
Tamanho da população	9.984
Taxa de cruzamento	80%
Taxa de mutação	20%
Sincronização (ilha)	10
Máximo de iterações	100

Tabela 7.5 - Parâmetros PSO

Parâmetro	*Valor*
Tamanho do enxame	9.984
Inércia	1
Cognitivo	2
Social	2
Sincronização (ilha)	10
Máximo de iterações	100

Para a realização dos testes, utilizou-se o problema de roteamento descrito anteriormente, aplicando-se o método BA-KSP com AG e PSO paralelizados, mas o algoritmo de Yen continuou em versão sequencial. Contudo, observou-se que a configuração de preços dos nós se repetiam em algumas horas dos dias, reduzindo assim, o número de horas para 13 (treze). Sendo assim, executou-se cada implementação (sequencial, síncrono, vizinhança e ilha) dos algoritmos para encontrar os melhores caminhos para as 13 horas do dia.

Os testes foram realizados para as demandas de 25, 30, 50, 60, 70, 75, 80, 90 e 100 requisições simultâneas de comunicação. Cada configuração foi executada 10 vezes para que se obtenha uma distribuição que represente o comportamento real de cada estratégia utilizada.

7.3.1 CONFIGURAÇÕES DE HARDWARE E SOFTWARE

Para a realização dos testes, utilizou-se um servidor Dell PowerEdge T620 com sistema operacional CentOS 6.5. Suas configurações de *host* e *device* são apresentadas a seguir:

- Processador: Intel Xeon E5-2620 v2, 6 núcleos, 15 MB Cache;
- Memória RAM: 16 GB;
- GPU: NVIDIA Tesla K20c, 2496 núcleos CUDA, 5 GB de memória;
- Versão do CUDA: 6.5.

7.3.2 RESULTADOS: CUSTOS OBTIDOS

A Figura 7.2 apresenta os resultados obtidos pelos AGs em termos do custo monetário da utilização das redes ópticas com 25 requisições de comunicação. É possível observar que, embora implementados com os mesmos operadores e parâmetros, as versões paralelas obtiveram resultados piores que a versão sequencial. Este comportamento é observado sob todas as demandas testadas, como se pode verificar na Figura 7.3, que apresenta os resultados para 100 requisições.

Figura 7.2 - Resultados AG para 25 requisições

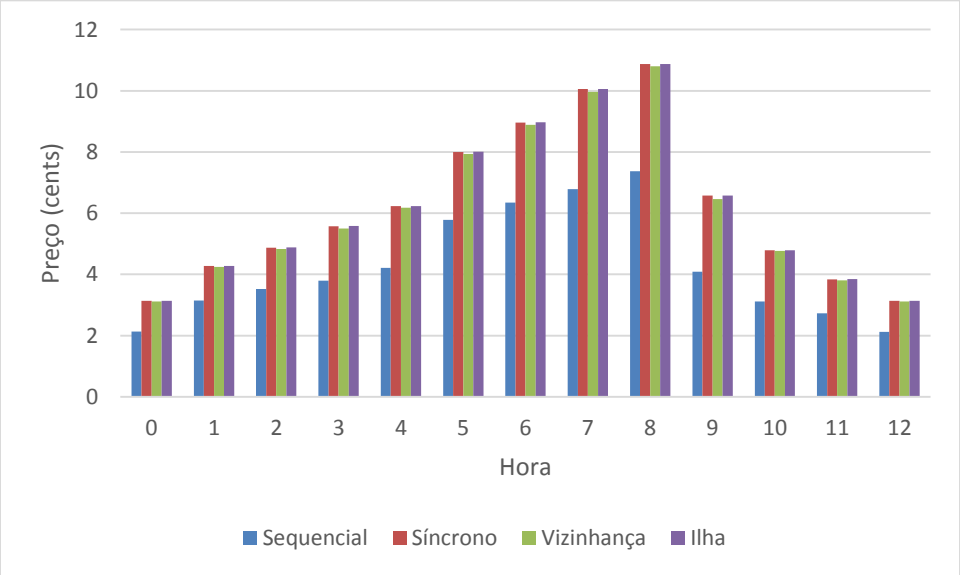
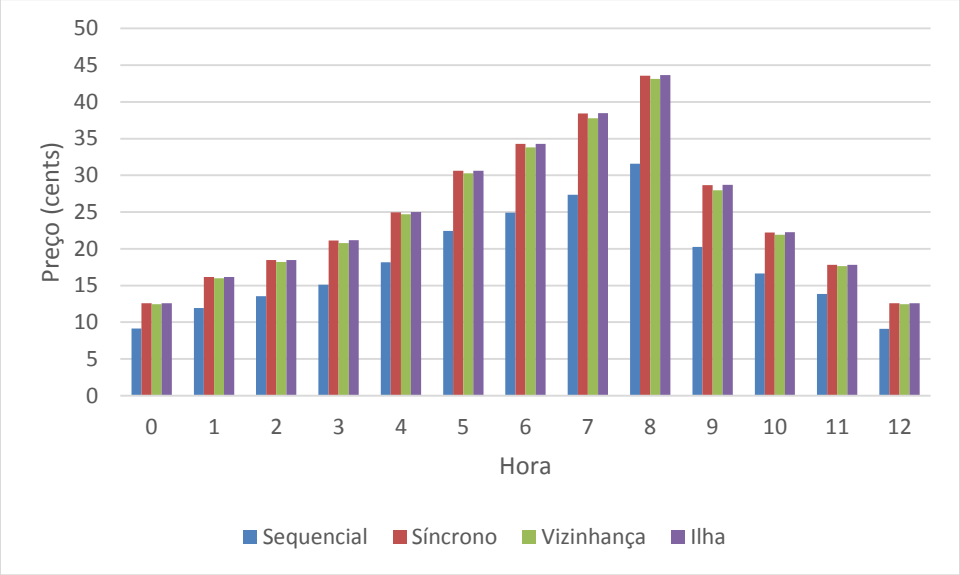


Figura 7.3 – Resultados AG para 100 requisições



Em contraponto, todas as versões do PSO obtiveram resultados aproximados para as demandas menores, como se pode observar na Figura 7.4, que apresenta os resultados do PSO para 25 requisições. A partir da Figura 7.5, que apresenta os resultados do PSO para 100 requisições, percebe-se que o algoritmo paralelo vai se distanciando de sua versão sequencial, obtendo resultados melhores, quando a demanda aumenta.

Figura 7.4 - Resultados PSO para 25 requisições

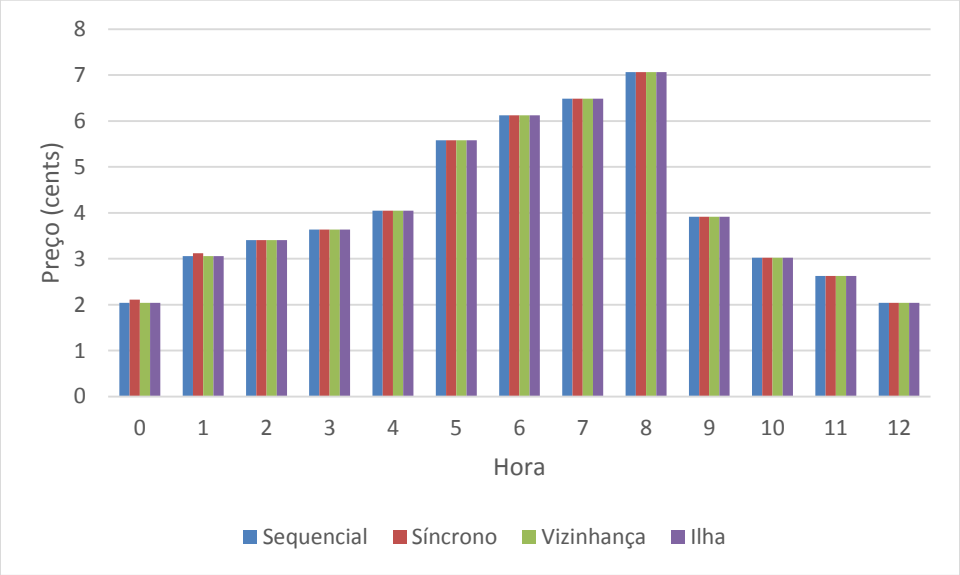
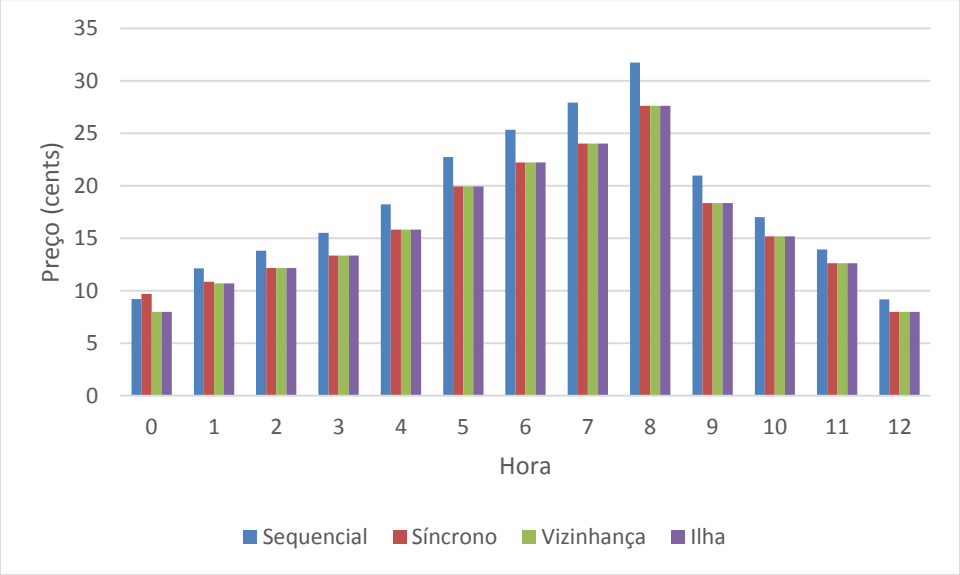


Figura 7.5 - Resultados PSO para 100 requisições



Entretanto, o objetivo não é apenas encontrar o melhor resultado, mas também, determinar qual versão dos algoritmos melhor se adapta ao problema e ao algoritmo reduzindo seu tempo de execução ao máximo quando comparada com as outras versões avaliadas neste trabalho.

7.3.3 RESULTADOS: TEMPOS DE EXECUÇÃO E *SPEEDUP*

Com os resultados anteriores, observou-se que as implementações paralelas, sob a plataforma CUDA, do algoritmo PSO aumentam sua convergência, haja visto que seus resultados foram, em média, melhores que os de sua versão sequencial. Enquanto, para o AG, o oposto é verdadeiro, o que o torna menos atrativo como método de resolução deste problema, principalmente sob a plataforma paralela estudada.

Contudo, a convergência mais rápida em número de iterações não é fator determinante para a escolha de um algoritmo, ou de uma estratégia de implementação paralela do mesmo. Outros fatores devem ser considerados, como uso de memória e tempo de execução. Sendo assim, a Tabela 7.6 apresenta os tempos médios de execução de todas as versões de AG e PSO, em segundos, para todas as configurações de testes (de 25 à 100 requisições), sendo os resultados em negrito, os melhores para cada algoritmo.

Tabela 7.6 - Tempos médios de execução dos algoritmos

		Requisições							
		25	30	50	70	75	80	90	100
AG	Sequencial	3,689	4,068	5,682	7,241	7,613	7,925	8,738	9,517
	Síncrono	0,208	0,228	0,320	0,408	0,429	0,455	0,496	0,547
	Vizinhança	**0,182**	**0,195**	**0,249**	**0,302**	**0,315**	**0,328**	**0,356**	**0,382**
	Ilha	0,361	0,373	0,580	0,774	0,810	0,946	1,009	1,323
PSO	Sequencial	5,691	6,544	9,670	12,786	13,563	14,612	16,056	17,527
	Síncrono	2,335	2,541	2,542	2,714	2,778	2,774	2,868	2,943
	Vizinhança	**0,208**	**0,226**	**0,305**	**0,387**	**0,406**	**0,424**	**0,464**	**0,503**
	Ilha	1,253	1,322	2,250	3,574	3,900	3,900	5,036	6,468

De maneira a observar melhor o comportamento dessas implementações, a Figura 7.6 e a Figura 7.7 foram inseridas. Estas apresentam os tempos de execução das versões do AG e do PSO, respectivamente.

Figura 7.6 - Tempos de execução AG

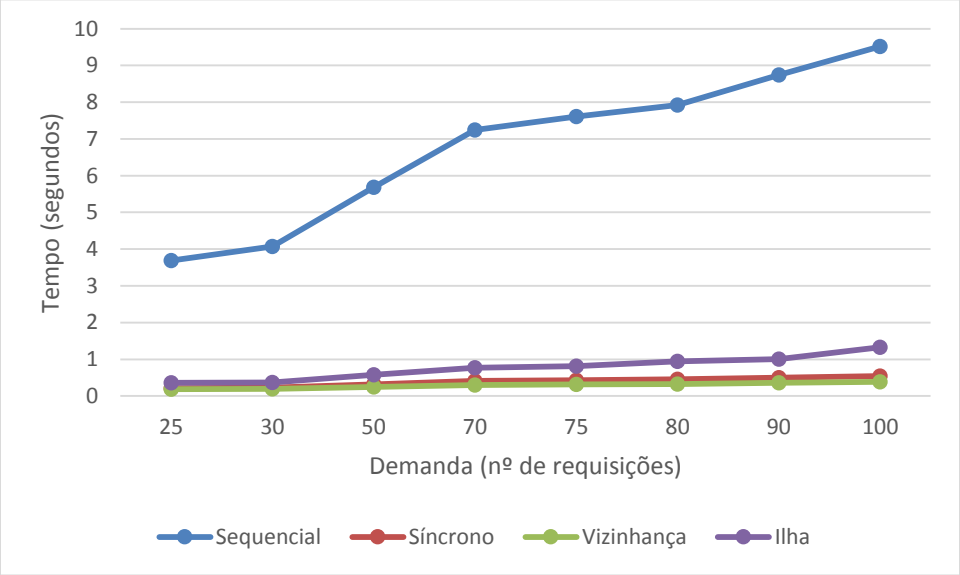
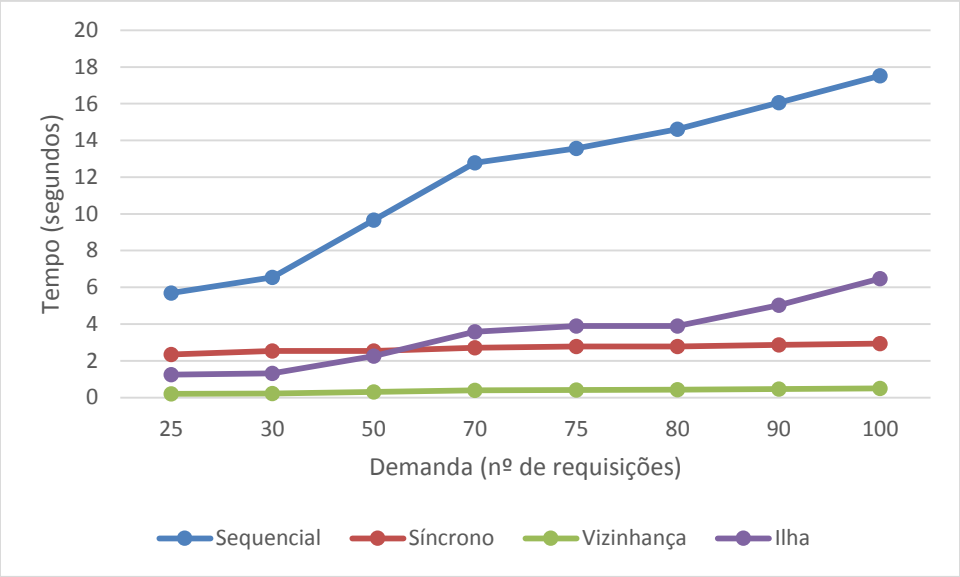


Figura 7.7 - Tempos de execução PSO



A partir destes resultados, percebe-se que a utilização da plataforma CUDA aumenta o desempenho, em termos de tempo de execução, dos algoritmos estudados. Entretanto, observou-se que o PSO baseado em ilhas possui comportamento semelhante ao sequencial, de perda de desempenho com aumento da demanda. Percebe-se também, que o AG, em geral, possui menor tempo de execução que o PSO, considerando-se a mesma quantidade de indivíduos e o mesmo número de iterações.

A Figura 7.8 e a Figura 7.9 apresentam os *speedups* do AG e do PSO, respectivamente. É possível observar que em ambos os algoritmos, o modelo baseado em ilhas sofre queda de performance com o aumento da demanda, o que confronta com os resultados obtidos nos testes com funções de *benchmarking*.

Figura 7.8 - *Speedup* AG

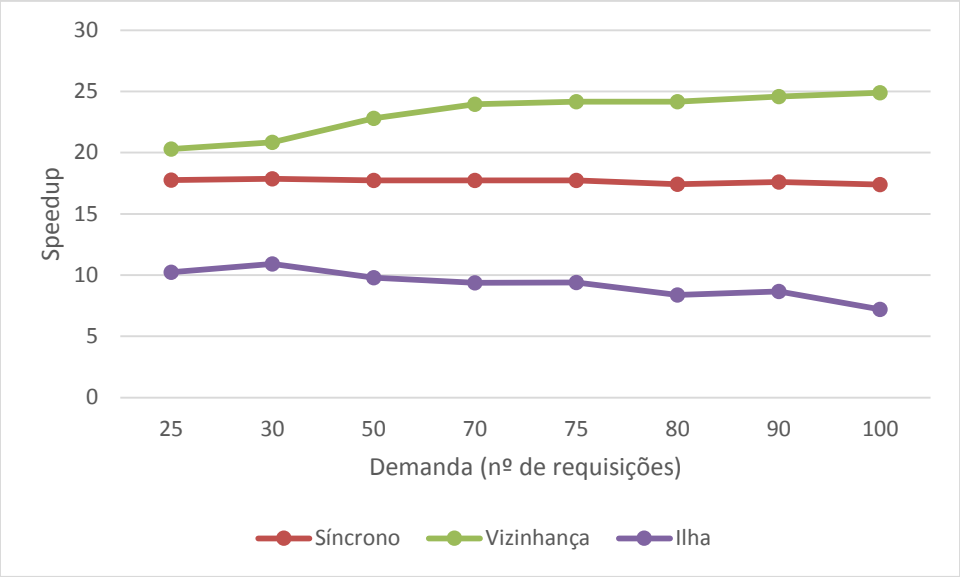
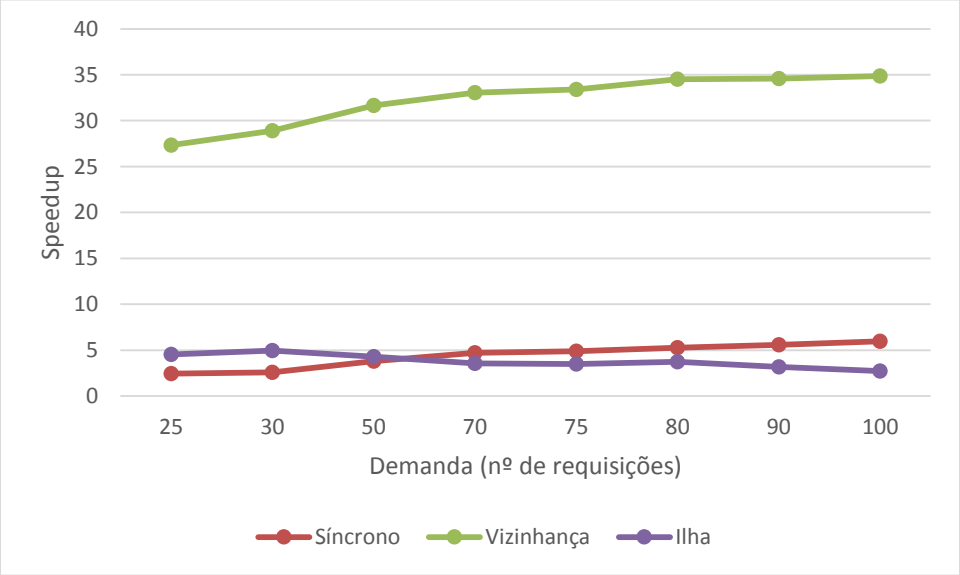


Figura 7.9 - *Speedup* PSO



Outra informação que pode ser extraída desses gráficos é que para ambos os algoritmos, o modelo de vizinhança apresenta os melhores *speedups*, além de sua performance aumentar com o crescimento da demanda. Logo, pode-se afirmar que para o problema de otimização do roteamento de redes ópticas WDM com múltiplas requisições simultâneas de comunicação, a estratégia de paralelização utilizando modelo de vizinhança

é a mais indicada para os algoritmos estudados. Resultado este, levemente diferente do obtido pelos *benchmarks*.

7.4 CONSIDERAÇÕES FINAIS

Neste capítulos foram descritos os testes realizados sob a plataforma CUDA, para o estudo de estratégias de paralelização de AG e PSO. Primeiramente, utilizando-se de funções de *benchmarking* de algoritmos bioinspirados, para em seguida, aplicar os algoritmos implementados ao problema de roteamento em redes ópticas. Com os resultados obtidos, foi possível observar o comportamento de todas as implementações em relação ao caso de estudo. Ademais, no próximo capítulo, algumas conclusões sobre esta dissertação são apontadas, assim como as dificuldades encontradas e trabalhos futuros.

8 CONCLUSÃO

Este trabalho apresentou uma proposta de método para a solução de problemas de otimização do roteamento em redes com múltiplos canais e alta grau de demanda. Este método, chamado BA-KSP, é um híbrido entre o algoritmo de Yen, que encontra as menores rotas em um grafo para um par origem-destino, e Algoritmos Bioinspirados, para encontrar a melhor combinação entre as rotas encontradas para um conjunto de pares origem-destino no mesmo grafo.

Para a realização de testes do método proposto, utilizou-se os algoritmos de Otimização por Enxame de Partículas e Algoritmos Genéticos, por serem os mais difundidos na literatura. Contudo, estes se tornam extremamente custosos à medida que a complexidade do problema aumenta. Portanto, a fim de reduzir o tempo para a execução do método proposto, optou-se por implementá-los em ambiente de Computação em GPU, mais especificamente sob a plataforma CUDA.

A partir da análise de trabalhos que implementassem algoritmos bioinspirados em GPU, observou-se que a plataforma CUDA traz ganho em performance aos algoritmos testados, justificando sua utilização. Contudo, percebe-se que não há um consenso acerca de qual a melhor estratégia de paralelização utilizar nestes algoritmos, ou seja, como implementá-los nesta plataforma, de maneira que eles aproveitem ao máximo os recursos da arquitetura e não sejam onerados.

Sendo assim, três estratégias foram selecionadas para a realização dos testes: Paralelização Síncrona, que busca manter os indivíduos na mesma iteração dos algoritmos utilizados; Modelo de Vizinhança, onde os indivíduos realizam as etapas dos algoritmos de maneira independente dos outros e se comunicam somente com os vizinhos mais próximos; e Modelo Baseado em Ilhas, no qual o conjunto total de indivíduos é dividido em subconjuntos e cada um deste executa o algoritmo inteiro em um núcleo da GPU.

A partir dos resultados obtidos, observou-se que modelo baseado em ilhas apresentou tempos de execução extremamente elevados quando comparados com as outras estratégias para os dois algoritmos testados. Resultado confrontante com testes prévios, realizados com funções de *benchmarking* para estes algoritmos, também apresentados neste trabalho.

Enquanto, o modelo de vizinhança mostrou ser a melhor abordagem para o problema estudado, obtendo os melhores resultados em ambos os algoritmos.

É possível verificar também, que a utilização da plataforma CUDA se torna mais interessante a medida que a complexidade do problema aumenta, pois fica mais expressiva a diferença nos tempos de execução nesta plataforma, comparada às implementações em CPU. É importante notar, contudo, que essa condição é dependente da estratégia de paralelização utilizada e do problema trabalhado, como se pode observar nos resultados do modelo baseado em ilhas, que perde desempenho quando do aumento das demandas.

Já a estratégia síncrona possui comportamento quase constante com relação ao *speedup*, para os dois algoritmos. Este resultado se deu pelo fato de esta estratégia ser somente o algoritmo sequencial com aplicação das operações para a população em paralelo, reduzindo a independência das unidades de processamento, o que acaba por limitar a plataforma.

Em relação ao problema de roteamento em redes ópticas, a maior dificuldade encontrada foi a comparação com outros trabalhos que realizassem a mesma tarefa, haja visto que os mesmos não disponibilizavam os resultados completos, ou omitiam partes de como gerá-los, o que acabava por impossibilitar sua replicação.

Este estudo, porém, visa otimizar o desempenho do método proposto na aplicação do problema apresentado, demonstrando como a forma de implementar estes algoritmos em ambiente de Computação em GPU depende completamente do problema a ser tratado.

A fim de contribuir com o presente estudo, é possível realizar outros testes envolvendo algoritmos bioinspirados e CUDA, acrescentando-se novas estratégias de paralelização e outros algoritmos (*e.g.* Otimização por Colônia de Formigas, ou demais variantes dos modelos de AG e PSO. Da mesma forma, é possível incorporar estes algoritmos ao método BA-KSP para verificar se há ganho em relação aos algoritmos já testados.

REFERÊNCIAS

Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. Em *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (pp. 483-485). New York, NY, USA: ACM.

Andrews, G. R. (2000). *Foundations of Multithreaded, Parallel, and Distributed Programming.* University of Arizona, USA: Wesley.

Barbosa, A. O. (2005). *Simulação e Técnicas da Computação Evolucionária Aplicadas a Problemas de Programação Linear Inteira Mista.* Curitiba, PR: Tese de doutorado, Universidade Tecnológica Federal do Paraná.

Bhandari, D., Murthy, C. A., & Pal, S. K. (abril de 2012). Variance As a Stopping Criterion for Genetic Algorithms with Elitist Model. *Fundameta Informaticae*, pp. 145-164.

Cavdar, C., Yayimli, A., & Wosinska, L. (2011). How to Cut the Electric Bill in Optical WDM Networks with Time-zones and Time-of-use Prices. *37th European Conference and Exposition on Optical Communications*.

da Silva, M. d. (2011). *Controle de granularidade de tarefas em OpenMP*. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul.

De Castro, L. N. (2006). *Fundamentals of Natural Computing.* Chapman and Hall.

Dieterich, J. M., & Hartke, B. (2012). Empirical Review of Standard Benchmark Functions Using Evolutionary Global Optimization. *Applied Mathematics*, 1552-1564.

Dijkstra, E. W. (1965). Solution of a Problem in Concurrent Programming Control. *Communication of the ACM*, pp. 569----.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to Evolutionary Computing.* Springer.

Feier, M. C., Lemnaru, C., & Potolea, R. (2011). Solving NP-Complete Problems on the CUDA Architecture Using Genetic Algorithms. Em *Proceedings of the 2011 10th International Symposium on Parallel and Distributed Computing* (pp. 278-281). Washington, DC, USA: IEEE Computer Society.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Holland, J. H. (1975). *Adaptation in natural artificial systems.* University of Michigan Press.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. *IEEE International Conference on Neural Networks*, (pp. 1942-1948).

Kennedy, J., & Eberhart, R. C. (2001). *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kirk, D. B., & Hwu, W.-M. W. (2010). *Programando para processadores paralelos.* Campus-Elsevier.

Kirk, D. B., & Hwu, W.-m. W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 992-1007.

Lin, Y., Gan, C., & Liang, C. (2010). Reducing Call Routing Cost for Femtocells. *IEEE Transactions on Wireless Communications*, 2302-2309.

Linden, R. (2006). *Algoritmos Genéticos: Uma Importante Ferramenta da Inteligência Computacional.* Rio de Janeiro, RJ: Editora Brasport.

Melab, N., Cahon, S., & Talbi, E. G. (2006). Grid Computing for Parallel Bioinspired Algorithms. *Journal of Parallel and Distributed Computing*, 1052-1061.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms.* Cambridge: USA: MIT Press.

Moraes, A. d. (2011). *DESENVOLVIMENTO E IMPLEMENTAÇÃO DE VERSÕES PARALELAS DO ALGORITMO DO ENXAME DE PARTÍCULAS EM CLUSTERS UTILIZANDO MPI.* Dissertação de Mestrado, Universidade Federal do Rio de Janeiro.

Navaux, P. O., & De Rose, C. A. (2003). *Arquiteturas Paralelas.* Porto Alegre, Brasil: Editora Sagra Luzzatto.

NVIDIA. (agosto de 2014). *CUDA C Programming Guide.* Fonte: CUDA Toolkit Documentation v6.5: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3RUSXl9hM

NVIDIA. (11 de fevereiro de 2015). *CUDA Parallel Computing Platform*. Fonte: nvidia.com.br: https://www.nvidia.com.br/object/cuda_home_new_br.html

Oiso, M., Yasuda, T., Ohkura, K., & Matumura, Y. (2011). Accelerating steady-state genetic algorithms based on CUDA architecture. *2011 IEEE Congress of Evolutionary Computation (CEC)*, 687-692.

Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., & Maggs, B. (2009). Cutting the Electric Bill for Internet-scale Systems. *SIGCOMM Comput. Commun. Rev.*, 123-134.

Raidl, G. R., & Puchinger, J. (2008). Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatory Optimization. *Studies in Computational Inteligence*, 31-62.

Safe, M., Carballido, J., Ponzoni, I., & Brignole, N. (2004). On Stopping Criteria for Genetic Algorithms. Em A. L. Bazzan, & S. Labidi, *Advances in Artificial Intelligence – SBIA 2004* (pp. 405-413). Springer Berlin Heidelberg.

SOBRAPO - Sociedade Brasileira de Pesquisa Operacional. (11 de fevereiro de 2015). *Pesquisa Operacional*. Fonte: SOBRAPO - Sociedade Brasileira de Pesquisa Operacional: http://www.sobrapo.org.br/o_que_e_po.php

Tanenbaum, A. S. (2007). *Modern Operating Systems* (3ª ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Wachowiak, M. P., & Foster, a. E. (outubro de 2012). GPU-Based Asynchronous Global Optimization with Particle Swarm. *Journal of Physics: Conference Series*.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 80-83.

Yen, J. Y. (1970). An Algorithm for Finding Shortest Routes from All Source Nodes to a Given Destination in General Networks. *Quart. Applied Math*, 526-530.

Yong, W., Chiaraviglio, L., Mellia, M., & Neri, F. (2009). Power-Aware Routing and Wavelength Assignment in Optical Networks. *ECOC*, 20-24.

Zhou, Y., & Tan, Y. (maio de 2009). GPU-based parallel particle swarm optimization. *2009 IEEE Congress on Evolutionary Computation, n. 2*, pp. 1493-1500.