2
Trabalhos Relacionados

Este é um trabalho multidisciplinar que aborda questões de Programação Genética, Computação de Alto Desempenho e Computação Quântica. Há uma quantidade considerável de trabalhos nestas áreas. Neste capítulo, concentramos nossas discussões nas pesquisas mais relacionadas aos temas contidos nesta tese. Apresentamos uma revisão da literatura nas áreas de PG, aceleração da PG e algoritmos de inspiração quântica.

2.1
Programação Genética

A Programação Genética foi proposta para resolver uma das principais questões da Ciência da Computação: "Como os computadores podem aprender a resolver problemas sem que sejam explicitamente programados para tal?". A PG tem evoluído rapidamente e tem sido utilizada para resolver uma grande variedade de problemas. Muita literatura foi produzida nesta área.

Cramer [27] foi um dos primeiros a publicar um artigo considerado genuinamente PG. Ele propôs um sistema para gerar automaticamente sequências curtas de funções para computadores utilizando as linguagens JB e TB, as quais foram desenvolvidas pelo autor para esta aplicação. O sistema era composto por um algoritmo genético utilizado para evoluir uma lista de números inteiros. Estes inteiros eram utilizados para definir as instruções, os argumentos e a ordem das instruções nos programas evoluídos. O sistema foi testado na evolução de funções simples para multiplicar dois valores de entrada e gerar um valor de saída, mostrando que na maioria das vezes, o programa evoluído era capaz de desempenhar corretamente a sua função.

Jong [28] estudou a utilização de algoritmos genéticos para realizar a busca no espaço dos possíveis programas capazes de resolver uma determinada tarefa com um nível aceitável de desempenho. Apresentou uma discussão sobre o nível da linguagem que deveria ser empregada para facilitar a busca dos algoritmos genéticos. O ajuste de parâmetros para uma estrutura fixa envolveria o menor nível de complexidade, por outro lado, a necessidade de evoluir tarefas mais flexíveis deveria trabalhar com instruções em um nível mais baixo.

Fujiko e Dickinson [29] adaptaram um algoritmo genético para manipular expressões da linguagem LISP, modificando os operadores de cruzamento, inversão e mutação para adequá-los a esta linguagem. Este modelo foi aplicado

com sucesso para evoluir uma solução para o problema do dilema do prisioneiro.

Koza [30] e De Garis [31] foram os primeiros a utilizar o termo "Programação Genética" para descrever um sistema que realiza a busca por um programa de computador mais adequado, no espaço de todos os possíveis programas, para resolver problemas de diversas áreas da inteligência artificial, tais como processamento simbólico, redes neurais e aprendizado de máquina.

Koza [30] apresenta diversos exemplos de aplicações para o modelo proposto: o problema de aprendizado de máquina dos multiplexadores de 3, 6 e 11 bits; o problema de planejamento aplicado à robótica que consiste em receber as informações dos sensores externos e decidir qual a sequência de ações mais adequada a ser tomada; o problema da identificação de funções simbólicas, por exemplo, a identificação da sequência de Fibonacci, que consiste em encontrar uma função em representação simbólica que se ajusta a pontos de dados amostrados da função original; o problema de regressão simbólica que consiste em encontrar uma função em representação simbólica que melhor se ajusta a uma sequência de pontos de dados de uma curva com comportamento desconhecido; e o problema de encontrar funções para integração e derivação simbólica. O termo "Programação Genética" foi patenteado por Koza [32].

De Garis [31] chamou de Programação Genética a aplicação de algoritmos genéticos para evoluir os pesos e os sinais de uma rede neural, a qual é utilizada para controlar funções tais como caminhar e oscilar, de uma maneira ótima. Para demonstrar o poder do seu modelo de programação genética, De Garis [31] evoluiu a sequência de ações necessárias para controlar um par de pernas mecânicas durante uma caminhada.

Koza [2] publicou seu livro sobre programação genética, desenvolvendo o formalismo da PG. Neste livro são discutidas as questões relativas ao que seria necessário para que computadores resolvessem problemas sem serem explicitamente programados para tal. Foram estendidas as aplicações apresentadas em [30]. Nestas aplicações, Koza utiliza um modelo de programação genética com representação dos indivíduos em árvore, empregando a linguagem LISP para a realização dos cálculos. LISP possui um estrutura de dados que facilita a representação em árvore dos indivíduos.

Nordin [33] propôs a ideia de utilizar a representação dos programas ou indivíduos da população no nível mais baixo, o código de máquina binário. Os resultados experimentais foram realizados em duas arquiteturas: um PC com processador Intel e uma estação SUN SPARC. O *benchmark* utilizado foi um problema de classificação de palavras suíças entre substantivos e não substantivos. Os resultados mostraram que a metodologia proposta tinha um desempenho de até três ordens de magnitude superior aos sistemas de PG

baseados em linguagem interpretada.

Montana [34] propôs um modelo de programação genética fortemente tipificado para trabalhar com programas que manipulam tipos diferentes de dados e com funções projetadas para operar em tipos particulares de dados. Restrições de tipos foram implementadas para garantir a coerência durante o processo evolutivo entre os tipos de dados de entrada e as funções aplicadas sobre estes dados, evitando incompatibilidades. Este modelo de programação genética foi aplicado em problemas que envolvem dois tipos de domínios, manipulação de vetores e matrizes e manipulação de listas. Os resultados experimentais foram obtidos com os problemas de regressão por mínimos quadrados multidimensional e com o filtro de Kalman multidimensional, mostrando resultados melhores para estas classes de problemas envolvendo tipos diferentes de dados quando comparados às soluções obtidas por modelos de PG não tipificados.

O desenvolvimento da área de programação genética levou os pesquisadores a elaborarem outras formas de representar os indivíduos alternativas a representação em árvore até então empregada. Miller e Thomson [35] apresentaram um novo modelo, chamado de programação genética cartesiana, a qual utiliza grafos para representar os indivíduos. Internamente os indivíduos são representados por listas de números inteiros. Estes números inteiros determinam as ligações entre os nós de um grafo, entre as entradas e nós internos, e especificam os nós que serão usados como saídas. Assim, a programação genética cartesiana apresenta uma flexibilidade maior na representação de programas quando comparada a programação genética com representação em árvore. Dois *benchmarks* foram utilizados para obter resultados experimentais: um problema de regressão simbólica polinomial de sexta ordem representado pela equação $x^6 - 2x^4 + x^2$ e o problema da trilha de formigas de Santa Fé. Os resultados experimentais mostraram que o modelo proposto apresentou-se mais efetivo ao solucionar os problemas do que os demais modelos de PG.

Brameier [36] desenvolveu uma tese sobre programação genética linear. Neste caso, os indivíduos são representados por sequências lineares de instruções, as quais são mais adequadas ao que o hardware dos computadores requer para execução. Entre os objetivos deste estudo estão o desenvolvimento de métodos mais avançados e novos operadores de variação para produzir soluções melhores e mais compactas, além de analisar os fenômenos geralmente ocorridos em PG linear, tais como a inclusão de *introns* (blocos de código que não afetam a aptidão), variações neutras e o crescimento do código.

A programação genética tem sido aplicada à diversas áreas de

conhecimento com bastante sucesso. Há diversos problemas na literatura que usam a PG para obter suas soluções. São eles: reconhecimento de padrões, controle de robôs, mineração de dados, análise de imagens, evolução de regras para investimento na bolsa de valores, síntese automática de circuitos elétricos e evolução de regras para o jogo de xadrez [2, 4, 5, 37, 38, 39, 40, 6, 41, 42, 7, 8].

A metodologia de PG proposta neste trabalho utiliza uma representação linear para os indivíduos, a qual é evoluída através de um modelo inspirado nos princípios da computação quântica, tais como utilização do bit quântico, interferência e superposição de estados.

2.2 Acelerando a PG

Como a PG é uma tarefa muito dispendiosa computacionalmente, muitos trabalhos têm sido desenvolvidos para reduzir seu tempo computacional. Estes concentram-se em reduzir o tempo de execução através de: implementações mais eficientes do algoritmo de PG e exploração de paralelismo.

A redução do tempo de execução através de otimizações do algoritmo de PG tem por objetivo diminuir o número de avaliações. O trabalho de Lewis [43] apresenta duas abordagens principais para reduzir o número de avaliações: evitar avaliações que não afetam o processo de seleção e evitar a repetição de avaliações que já foram realizadas. Evitar avaliações que não afetam o processo de seleção significa deixar de calcular as amostras de dados que não apresentam ou que apresentam um efeito muito pequeno no resultado final da avaliação de um indivíduo. O objetivo seria utilizar somente as amostras de dados que apresentam os maiores erros durante o cálculo da avaliação de um indivíduo. O monitoramento do histórico de avaliações durante a evolução pode ser usado para identificar dinamicamente este subconjunto de amostras de dados que apresenta a maior contribuição no cálculo da avaliação. Por outro lado, evitar a repetição de avaliações que já foram realizadas significa armazenar na memória os resultados das avaliações dos indivíduos das gerações anteriores. Isto possibilitaria reutilizar o resultado do cálculo se surgisse um novo indivíduo na população que fosse idêntico a algum dos indivíduos já avaliados.

Alternativamente, muitos autores exploram a computação paralela para reduzir o tempo de computação. Koza e Andre [44] foram os primeiros a desenvolver um algoritmo paralelo de PG usando uma rede de processadores transputer. Eles propuseram um algoritmo distribuído em que subpopulações são atribuídas para diferentes nós de computação da rede. O problema de uma regressão simbólica de uma função booleana para o cálculo da paridade par

com 5 bits, foi resolvido.

Juille e Pollack [45] propuseram uma forma eficiente de usar uma máquina SIMD (*Single Instruction, Multiple Data*) para simular uma máquina MIMD (*Multiple Instruction, Multiple Data*) ao paralelizar um algoritmo de PG, através da utilização de um conjunto reduzido de instruções. Assim, todos os indivíduos da população são avaliados em paralelo no computador MasPar MP-2, sendo atribuído um indivíduo para cada um dos 4096 processadores disponíveis. Desta forma, foram capazes de avaliar até 2350 indivíduos por segundo no problema da descoberta de identidades trigonométricas e até 710 indivíduos por segundo no problema da integração simbólica.

Stoffel e Spector [46] desenvolveram um sistema de programação genética de alto desempenho chamado de HiGP. Este modelo evolui programas lineares para uma máquina virtual, tornando-o flexível e facilmente portável. O algoritmo paralelo empregado foi desenvolvido para usar o modelo de programação SPMD, onde todos os nós de computação executam o mesmo programa sobre conjuntos diferentes de dados. O problema de regressão simbólica representado por $y = x^9$ foi usado como *benchmark*, mostrando que o modelo é capaz de atingir ganhos de desempenho quase lineares com o número de processadores usado.

Eklund [47] implementou um modelo maciçamente paralelo de PG linear em código de máquina de CPUs. A linguagem VHDL foi utilizada para representar o modelo de PG, tornando o modelo flexível e facilmente adaptável a aplicações diferentes. O algoritmo paralelo utilizado divide a população de indivíduos em um grande número de pequenas subpopulações, sendo que cada uma delas é atribuída para executar em um nó de computação. Este modelo foi utilizado na previsão de séries temporais, na previsão do número de manchas solares no período de 1700-1994, obtendo resultados melhores do que os resultados apresentados por modelos estatísticos e por redes neurais.

Folino *et al.* [48] desenvolveram uma metodologia para explorar o paralelismo da PG em problemas de classificação com grandes conjuntos de dados, chamada de técnica de montagem. O algoritmo paralelo consiste em dividir o conjunto de dados em subconjuntos e executar a evolução da PG para cada subconjunto em um nó de computação diferente. Ao final, é utilizada uma estratégia de votação entre os resultados de cada nó, para escolher a classificação final. O problema de classificação, com dados reais do censo realizado nos EUA entre os anos de 1994 e 1995, foi resolvido usando a metodologia proposta e usando a PG normal. Com isso, os autores mostraram que a técnica de montagem pode levar a melhores resultados, com um esforço computacional menor.

Outra metodologia de aceleração comumente empregada é a implementação do algoritmo a ser acelerado em hardware, usando FPGAs reconfiguráveis. Diversos autores estudaram a aceleração da PG em FPGAs.

Koza *et al.* [49] utilizaram uma FPGA Xilinx XC6216 com reconfiguração rápida para implementar um algoritmo de PG com representação dos indivíduos em árvore, com paralelismo maciço. O algoritmo paralelo usado consiste em configurar a FPGA para implementar em hardware a função de avaliação de cada indivíduo, avaliando um indivíduo por vez. O problema das redes de ordenação com 7, 8 e 9 entradas foi resolvido, encontrando-se a solução mínima com 16, 19 e 25 passos, respectivamente.

Sidhu *et al.* [50] apresentaram uma forma rápida e compacta de representar a estrutura em árvore da PG no hardware das FPGAs, de tal forma que é possível executar a evolução e a avaliação dos indivíduos sem interferência externa da CPU. O modelo de paralelismo proposto emprega a execução de todos os indivíduos em paralelo, através da representação em hardware das suas funções de avaliação. Dois problemas foram resolvidos usando este modelo: a função booleana do multiplexador de 11 bits e o problema de regressão simbólica de uma variável.

No trabalho de Heywood e Zincir-Heywood [51] plataformas customizadas de computação baseadas em FPGAs foram propostas para a implementação de PG linear. O algoritmo paralelo utilizado implementa máquinas paralelas baseadas em registradores usando hardware para avaliar rapidamente os indivíduos da PG. O problema de regressão simbólica, representado pela equação $y = x^4 + x^3 + x^2 + x$ foi resolvido.

Martin [52] usou uma linguagem de alto nível Handel-C para desenvolver um algoritmo de PG para FPGAs. O paralelismo utilizado foi desenvolvido para executar todas as funções, inicialização dos indivíduos, avaliação, seleção e operadores de reprodução, usando o paralelismo das FPGAs. Dois *benchmarks* de PG foram utilizados: um *benchmark* de regressão simbólica $x = a + 2b$ e um problema de lógica booleana de dois bits $x = a \oplus b$.

Em um segundo trabalho, Martin [53], realizou um estudo para verificar a influência do gerador de números aleatórios no desempenho do algoritmo paralelo de PG para FPGAs. Foi verificado que a utilização de geradores de números aleatórios baseados em múltiplos LFSRs (*linear-feedback shift register*) contribui para melhorar o desempenho de um sistema de PG em FPGAs.

Recentemente, com o surgimento da programação de propósito geral em GPUs, muitos trabalhos têm sido desenvolvidos para acelerar a execução da PG usando placas gráficas. As primeiras implementações de PG para GPUs

foram apresentadas por Chitty [14] e Harding e Banzhaf [15].

Chitty [14] utilizou a linguagem *C for Graphics* (Cg) da nVidia para desenvolver um algoritmo de PG paralelo para GPUs, com representação dos indivíduos em árvore. O algoritmo utilizado é baseado na metodologia de compilação. Um indivíduo por vez é avaliado na GPU, avaliando as amostras de dados em paralelo. Os *benchmarks* resolvidos na GPU consistem de dois problemas de regressão simbólica, sendo um deles representado pela equação $y = x^4 + x^3 + x^2 + x$ e outro representado por $y = 2.76x^2 + 3.14x$, um problema de classificação com o conjunto de dados *Fisher Iris*, e o problema booleano do multiplexador de 11 bits.

Harding e Banzhaf [15] utilizaram o software *Accelerator*, baseado na interface DirectX, para implementar um modelo de PG paralelo, o qual realiza a compilação dos indivíduos em tempo de execução. O modelo paralelo empregado, compila cada indivíduo para um programa e realiza a avaliação dos dados em paralelo na GPU. Entre os *benchmarks* utilizados, encontram-se um *benchmark* de regressão simbólica representado por $y = x^6 - 2x^4 + x^2$, um *benchmark* de classificação para distinguir pontos pertencentes a duas espirais, e um problema de classificação aplicado à bioinformática com a tarefa de prever a localização de uma proteína numa célula.

Além disso, em ambos os trabalhos de Chitty [14], e Harding e Banzhaf [15], os resultados mostraram um ganho de desempenho modesto para estudos de caso com poucas amostras de dados, devido ao número de operações de ponto flutuante a serem realizadas em paralelo na GPU não ser suficiente para compensar o overhead de geração dos indivíduos e transferência de dados. Ganhos consideráveis foram obtidos para estudos de caso com um grande número de amostras de dados e quando o programa compilado de PG era executado muitas vezes.

Langdon e Banzhaf [18] foram os primeiros a propor uma metodologia interpretada para GPU, com representação dos indivíduos em árvore. O algoritmo paralelo utilizado avalia toda a população de indivíduos em paralelo através de um interpretador desenvolvido em C++ RapidMind. Assim, foi explorado o paralelismo ao nível de indivíduos, enquanto as amostras de dados são executadas sequencialmente. Esta técnica foi chamada de interpretador SIMD e foram utilizadas instruções condicionais para selecionar as operações de PG, o que pode aumentar o overhead quando o conjunto de instruções da PG é grande, trabalhando com muitas amostras de dados. Os resultados experimentais foram obtidos com dois *benchmarks*: o *benchmark* da previsão de séries temporais Mackey-Glass, e o problema de prever a localização de uma proteína numa célula apenas com as informações da

composição dos seus aminoácidos. Estes resultados experimentais mostraram acelerações moderadas, mas apresentaram ganhos de desempenho até mesmo para pequenos programas com poucas amostras de dados, pois o overhead de inicialização é baixo. O interpretador de PG para GPUs foi utilizado posteriormente por Langdon e Harrison [19] aplicado ao problema da previsão da taxa de sobreviventes do câncer de mama num período de dez anos.

Robilliard *et al.* [20] também estudaram uma metodologia interpretada usando a linguagem CUDA, com o objetivo de evitar o overhead das instruções condicionais durante a interpretação de todos os indivíduos da população em paralelo. Para isso, eles propuseram um algoritmo paralelo para o interpretador onde cada indivíduo da PG é avaliado por um bloco diferente de *threads*. Durante a execução, cada bloco de *threads* é mapeado para um multiprocessador diferente da GPU, evitando os desvios no fluxo de execução do programa devidos às diferenças nas instruções entre os indivíduos. Dentro de cada bloco de *threads*, todas as *threads* executam a mesma instrução sobre diferentes subconjuntos de dados. Três *benchmarks* foram utilizados: o problema de regressão simbólica representado por $y = x^6 - 2x^4 + x^2$, o problema booleano dos multiplexadores com 6 e com 11 bits, e o *benchmark* de classificação para distinguir pontos de dados pertencentes a duas espirais. Os resultados experimentais mostraram ganho de desempenho quando comparado com a metodologia proposta por Langdon e Banzhaf [18].

Harding e Banzhaf [16] estudaram uma metodologia de compilação. Neste caso, um cluster de computadores foi usado para reduzir o *overhead* de compilação. A compilação dos indivíduos é feita em paralelo. As máquinas do cluster que estavam equipadas com GPUs simples também foram usadas para o processamento dos dados. Dois problemas foram usados nos experimentos: o problema de classificação de intrusos na rede da *KDD Cup 1999* e o problema da evolução de um filtro *emboss* para processamento de imagens. Acelerações foram obtidas somente para conjuntos de dados grandes. Porém, a utilização de placas gráficas melhores não necessariamente resultaria em melhor desempenho, uma vez que o principal gargalo permanece sendo a compilação.

Langdon e Harman [17] usaram a metodologia de compilação de código com outra finalidade, para criar automaticamente o *kernel* da GPU. Os autores empregaram muitas simplificações, tais como não evoluir a utilização da memória compartilhada, nem a distribuição das *threads*. O problema utilizado nos experimentos foi a geração automática de um código paralelo para GPUs capaz de ter uma funcionalidade idêntica à de um código sequencial altamente otimizado em C para compressão de arquivos, o *gzip*. O melhor indivíduo evoluído foi capaz de desempenhar a sua função corretamente, provando que

é possível elaborar uma metodologia para evoluir automaticamente o código paralelo para GPUs. Contudo, ainda não foi possível verificar acelerações obtidas com esta metodologia quando comparado ao código sequencial para CPU.

A Tabela 2.1 apresenta um resumo dos trabalhos relacionados encontrados na literatura, referentes à aceleração de PG.

De acordo com o nosso conhecimento, nenhum trabalho anterior evoluiu programas de GPU manipulando diretamente o código de máquina das GPUs. Além disso, com relação ao nível de paralelismo empregado no nosso trabalho, a função de avaliação é executada com o máximo de paralelismo possível. O paralelismo de dados é empregado dentro de cada bloco de *threads*, onde todas as *threads* executam as mesmas instruções sobre diferentes amostras de dados. Os indivíduos da PG são avaliados em paralelo usando blocos de *threads* diferentes, que serão executados por multiprocessadores diferentes. Com isso, buscamos evitar a divergência de código.

2.3 Algoritmos de Inspiração Quântica

As metodologias propostas neste trabalho para a aceleração da PG possuem inspiração quântica. A inspiração quântica traz relevantes contribuições para a computação evolucionária e em particular para a PG. Algoritmos evolucionários de inspiração quântica (*Quantum-inspired evolutionary algorithms*–QEAs) têm sido desenvolvidos com crescente interesse.

Moore e Narayanan [54] propuseram uma nova abordagem para explorar os conceitos da computação quântica. Ao invés de desenvolver um algoritmo para computadores quânticos ou tentar tornar viável a sua utilização, eles propuseram a ideia da **computação com inspiração quântica**. Esta nova abordagem visa criar algoritmos clássicos (isto é, executando em computadores clássicos) que utilizam o paradigma da mecânica quântica para melhorar o desempenho na busca de soluções para os problemas. Particularmente, os algoritmos com inspiração quântica (QEAs) têm se tornado objeto de interesse na computação evolucionária recentemente. A superposição linear usada na representação dos estados de um *qubit* possibilita que o QEA represente diversos indivíduos probabilisticamente.

Han e Kim [55] propuseram um algoritmo evolucionário com inspiração quântica (QEA) para utilização em otimização combinatória, o qual é baseado nos conceitos e princípios da computação quântica, tais como no bit quântico e na superposição de estados. Uma porta quântica chamada de *Q-gate* foi

introduzida para atualizar os indivíduos e guiá-los na busca por melhores soluções. O problema da mochila foi utilizado para obter os resultados experimentais e avaliar o desempenho do algoritmo. Os resultados mostraram que o QEA tem um melhor desempenho, mesmo com populações menores, sem convergência prematura, quando comparado aos algoritmos genéticos clássicos.

Kim *et al.* [56] propuseram a utilização de QEAs como uma nova metodologia para alocação de discos, distribuindo arquivos contendo conjuntos de dados muito grandes entre diversos discos para minimizar o tempo de recuperação dos dados. O melhor método utilizado até então era baseado em algoritmos genéticos e com a utilização de QEAs foi possível obter menores tempos de respostas para acesso aos dados, com um menor tempo de convergência do algoritmo.

Jang *et al.* [57] propuseram a utilização de QEAs como um algoritmo aplicado à detecção de rostos. QEAs foram utilizados juntamente com a Análise de Componentes Principais (PCA) com o objetivo de acrescentar a capacidade de aprendizado ao algoritmo. Desta forma é possível otimizar a decisão para distinguir entre imagens que representam rostos e imagens que não representam rostos, melhorando o desempenho do sistema.

Kim *et al.* [58] propuseram um algoritmo evolucionário com inspiração quântica para múltiplos objetivos (QMEA). Este algoritmo é inspirado em conceitos e princípios da computação quântica, tais como incerteza, superposição e interferência. Os resultados experimentais obtidos para o problema da mochila com múltiplos objetivos mostram que QMEAs são capazes encontrar soluções próximas da solução Pareto ótima.

Vlachogiannis e Lee [59] propuseram a utilização de QEAs para otimizar a geração de potência ativa e reativa. O algoritmo de otimização controla variáveis tais como a saída dos geradores, níveis de tensão e as configurações dos dispositivos de compensação e taps dos transformadores, com o objetivo de otimizar os níveis de potência ativa e reativa, levando em consideração os custos. Os resultados são comparados com outras técnicas de otimização encontradas na literatura, mostrando que a utilização dos QEAs contribui para melhorar a solução obtida.

Platel *et al.* [60] estabeleceram em seu artigo que QEA representa um algoritmo original que pertence à classe dos algoritmos de estimação de distribuição (EDAs), realizando uma comparação entre os pontos comuns e específicos entre QEAs e EDAs. Os autores reportaram que os QEAs apresentam maior diversidade na busca, melhor escalabilidade, melhor qualidade das soluções e maior robustez ao ruído. Os QEAs podem se adaptar dinamicamente à velocidade de aprendizado para obter um padrão de

convergência mais suave e robusto. Além disso, os QEAs podem manipular soluções com distribuições mais complexas, levando a uma solução mais eficiente para problemas com maior interação entre as variáveis.

Lau *et al.* [61] propuseram a utilização de QEAs como sendo um novo método para solucionar o problema da Unidade de Compromisso (UC). UC é um problema de otimização do setor elétrico que envolve encontrar a configuração ótima dos recursos de geração para atender a carga elétrica solicitada com um custo mínimo. O método proposto foi aplicado para a otimização de um sistema contendo entre 10 e 100 unidades de geração e com um horizonte de operação de 24 horas, para comparação com os outros métodos de otimização encontrados na literatura. Posteriormente, foi expandido para otimização de um sistema contendo 100 unidades de geração com um horizonte de 7 dias. Os resultados mostraram que os QEAs apresentam um desempenho superior para solucionar esta classe de problemas.

Dias e Pacheco [62] foram os primeiros a propor um algoritmo evolucionário com inspiração quântica para síntese automática de programas em código de máquina de CPUs, chamado QILGP. Este algoritmo utiliza a inspiração nos princípios da física quântica para melhorar o desempenho do algoritmo evolucionário executando em computadores clássicos. QILGP foi comparado com o modelo AIMGP [23], o qual é uma das técnicas clássicas de programação genética de maior sucesso, empregada comercialmente e que também utiliza código de máquina de CPUs. Dois *benchmarks* foram utilizados para comparar o desempenho entre QILGP e AIMGP. O primeiro *benchmark* é um problema de regressão simbólica de duas variáveis representado pela equação $f_1(x, y) = x^2 + y^2 + 2$, no qual QILGP obteve uma taxa de acertos de 100%, maior do que a taxa de 77% obtida pelo modelo AIMGP. O segundo *benchmark* utilizado é o problema de regressão simbólica representado pela equação $f_2(x, y) = x^4 + y^4 + x^3 + y^3 + x^2 + y^2 + x + y + 2$. Neste caso, QILGP teve uma taxa de acertos um pouco menor do que AIMGP, contudo, QILGP continua tendo um desempenho melhor quando são considerados apenas as avaliações do melhor indivíduo, para média e desvio padrão. Dias e Pacheco [63] aplicaram o modelo em uma série de problemas de regressão simbólica e em problemas de classificação binária, mostrando que é capaz de atingir melhores soluções, com um número menor de avaliações e usando uma quantidade menor de parâmetros e operadores do que o modelo clássico AIMGP.

O modelo proposto neste trabalho é um algoritmo evolucionário com inspiração quântica maciçamente paralelo para GPUs, baseado nas ideias utilizadas pelo modelo QILGP, com diferenças, como por exemplo, a substituição da ordenação dos indivíduos por um mecanismo de comparação

em pares de indivíduos, a representação dos indivíduos em linguagem intermediária e também a representação em código de máquina de GPUs.

Tabela 2.1: Tabela comparativa dos trabalhos relacionados encontrados na literatura, referentes à aceleração de PG.

Trabalho	Ano	Metodologia	Criação da População	Paralelismo	Aplicações	Desempenho
Koza e Andre [44]	1995	Interpretador	Árvore	Cluster	Função booleana: Paridade Par com 5 bits.	–
Juille e Pollack [45]	1996	Interpretador	Árvore	Cluster	Identidades trigonométricas, Integração simbólica.	Até 2350 indivíduos por segundo.
Stoffel e Spector [46]	1996	Interpretador	Linear	Cluster	Regressão Simbólica: $y = x^9$.	–
Eklund [47]	2003	Código de Máquina de CPUs.	Linear	Cluster	Previsão de Séries Temporais.	–
Folino *et al.* [48]	2003	Interpretador	Árvore	Cluster	Classificação: censo EUA 94 e 95.	–
Koza *et al.* [49]	1998	Hardware FPGAs.	Árvore	FPGAs	Redes de ordenação com 7, 8 e 9 entradas.	–
Sidhu *et al.* [50]	1999	Hardware FPGAs.	Árvore	FPGAs	MUX 11 bits, Regressão simbólica.	–
Heywood e Zincir [51]	2000	Hardware FPGAs.	Linear	FPGAs	Regressão Simbólica: $y = x^4 + x^3 + x^2 + x$.	–
Martin [52]	2001	Hardware FPGAs.	Linear	FPGAs	Regressão simbólica: $x = a + 2b$, Lógica booleana: $x = a \oplus b$.	–
Chitty [14]	2007	Compilador	Árvore	GPUs	Regressão simbólica: $y = x^4 + x^3 + x^2 + x$, $y = 2.76x^2 + 3.14x$, Classificação: *Fisher Iris*, MUX 11 bits.	–
Harding e Banzhaf [15]	2007	Compilador	Árvore	GPUs	Regressão simbólica: $y = x^6 - 2x^4 + x^2$, Classificação: duas espirais, Bioinformática.	–
Langdon e Banzhaf [18]	2008	Interpretador	Árvore	GPUs	Mackey-Glass, Bioinformática.	895 milhões de GPops.
Langdon e Harrison [19]	2008	Interpretador	Árvore	GPUs	Bioinformática: câncer de mama.	500 milhões de GPops.
Robilliard *et al.* [20]	2009	Interpretador	Árvore	GPUs	Regressão simbólica: $y = x^6 - 2x^4 + x^2$, MUX 6 e 11 bits, Classificação: duas espirais.	2,8 bilhões de GPops.
Harding e Banzhaf [16]	2009	Compilador	Linear	GPUs	*KDD Cup 1999*, Filtro *emboss*.	10,56 bilhões de GPops.
Langdon e Harman [17]	2010	Compilador	Árvore	GPUs	*gzip*	–