9
Conclusões

Este trabalho apresentou duas novas metodologias para a aceleração da Programação Genética (PG) através do uso de Unidades de Processamento Gráfico (GPUs). As metodologias propõem *compilação em linguagem intermediária* e *criação de indivíduos em código de máquina*. Elas criam indivíduos com menores (ou nenhuma) necessidade de compilação e aceleram o processo evolutivo da PG explorando o paralelismo maciço oferecido pela GPU. Avaliação dos indivíduos é realizada com alto nível de paralelismo. Múltiplos indivíduos e múltiplas amostras de dados podem ser avaliadas simultaneamente em paralelo. Além do benefício de explorar linguagens de mais baixo nível para a aceleração da PG em GPU, as metodologias propostas utilizam PG linear com um algoritmo evolucionário inspirado em computação quântica. O uso de algoritmo com inspiração quântica permite o emprego de um algoritmo evolutivo inovador que pode ser mais eficiente do que os algoritmos evolucionários tradicionais. A superposição linear de estados da computação quântica permite a representação de diferentes indivíduos probabilisticamente, provendo um mecanismo de evolução que reduz consideravelmente o número de cromossomos necessários para garantir uma boa diversidade de busca. Além disso, o uso da interferência quântica permite reforçar a estabilidade da busca, fornecendo um guia para a população de indivíduos possibilitando uma boa exploração das vizinhanças das soluções correntes na busca das melhores soluções.

Além de explorar linguagens de mais baixo nível para acelerar a PG em GPUs, este trabalho explora também o ambiente heterogêneo composto por CPU e GPU. Este ambiente heterogêneo permite que diferentes divisões de trabalho possam ser avaliadas. Neste caso, duas abordagens foram propostas: solução híbrida (CPU-GPU) e solução GPU. A solução híbrida (CPU-GPU) implementa apenas a avaliação dos indivíduos na GPU e o restante das tarefas é executado na CPU. Na solução GPU, todas as tarefas são executadas na GPU e a CPU permanece ociosa.

As metodologias propostas foram implementadas e comparadas com as metodologias tradicionais existentes na literatura para PG em GPUs. As metodologias tradicionais utilizam compilação e interpretação. Na metodologia de compilação, os indivíduos são criados na liguagem CUDA e compilados para serem avaliados na GPU. Na metodologia de interpretação, os códigos são interpretados em tempo de execução. Os experimentos foram realizados com

sete *benchmarks* comumente empregados em PG que resolvem problemas de regressão simbólica, previsão de séries temporais, processamento de imagens, classificação e regressão Booleana.

Utilizando uma placa gráfica de última geração, obtivemos resultados de aceleração notáveis. A metodologia de compilação em linguagem intermediária obteve ganhos consideráveis para o problema de regressão simbólica quando comparada à metodologia de compilação. O tempo de compilação dos indivíduos foi reduzido em 5,97 vezes, levando uma redução do tempo total de execução de 5,93 vezes. Esta metodologia, entretanto, se apresentou mais lenta que a metodologia de interpretação. A metodologia de criação de indivíduos em código de máquina executou 7,3 vezes mais rápida que a metodologia de interpretação para o problema de regressão simbólica.

Benchmarks maiores, que representam problemas reais, foram executados apenas para a metodologia de criação de indivíduos em código de máquina. Os *benchmarks* de processamento de imagens e previsão de séries temporais foram avaliados com as duas abordagens de divisão do trabalho entre a CPU e a GPU, híbrida e GPU. Com relação ao uso do ambiente heterogêneo, os seguintes resultados foram obtidos. A implementação de toda a PG na GPU obteve desempenho superior do que a implementação híbrida com criação dos indivíduos na CPU para o problema de previsão de séries. O ganho foi de 1,55 vezes. Para o problema de processamento de imagens, houve pequena perda de 11% para o filtro Sobel e ganhos de 1,43 vezes para a Restauração Salt-and-Pepper. Os *benchmarks* que representam problemas reais com conjunto de dados muito grandes, como o Multiplexador de 20-bits e o Classificador, foram avaliados apenas com a abordagem de criação de indivíduos em código de máquina. Estas avaliações mostraram o potencial de nossa metodologia para tratar problemas de larga escala.

Baseados nos resultados obtidos e na experiência com PG em GPU, propusemos ainda uma otimização para a metodologia de criação de indivíduos em código de máquina. Esta otimização altera a forma como os indivíduos avaliados são selecionados e permite a criação de indivíduos menores. Esta otimização apresentou reduções no tempo de execução de 25% para a previsão de séries, de 29% para o filtro Sobel e 21% para a Restauração Salt-and-Pepper.

Em termos da quantidade de operações de PG, a metodologia de criação de indivíduos em código de máquina obteve para toda a avaliação até 200,5 bilhões de GPops para a regressão simbólica, 4,27 bilhões de GPops para a previsão de séries, 226,0 bilhões de GPops para o filtro Sobel, 346,52 bilhões de GPops para a Restauração Salt-and-Pepper, até 134,30 bilhões de GPops para a classificação e 2.74 trilhões de GPops para a regressão booleana.

Os resultados obtidos apresentam uma nova perspectica sobre a implementação de PG em GPU. A metodologia de criação de indivíduos em código de máquina é escalável e introduz a possibilidade de se tratar problemas realmente grandes para a PG em um período de tempo considerado razoável. Esta metodologia permitiu que pela primeira vez o multiplexador de 20-bits fosse evoluído utilizando todas as amostras de dados durante a evolução. O maior multiplexador evoluído até o momento utilizando todas as amostras de dados emprega 11 bits. Outros multiplexadores utilizam um pequeno percentual das amostras de dados para avaliar cada indivíduo.

9.1 Trabalhos Futuros

O trabalho desenvolvido nesta tese abre uma série de oportunidades para pesquisas futuras. Vivemos um momento sem precedentes no que tange ao acesso a recursos computacionais vastos e de custo reduzido, fato este que vem propiciando um grande salto nas áreas de aprendizado de máquina e mineração de dados. Podemos lidar com conjuntos de dados (instâncias, atributos, classes, etc.) em uma escala inimaginável há alguns anos, no que é conhecido como *Big Data*. Por sua vez, a utilização geral de computação sob demanda maciça (nuvem e/ou GPUs) para aprendizado de máquina é conhecida como *Big Learning*. De maneira geral, técnicas evolutivas de aprendizado de máquina são candidatas ideais para tarefas de *Big Learning*, devido à sua flexibilidade em representações de conhecimento, paradigmas de aprendizado e seu paralelismo inato. Sendo assim, GMGP também apresenta um elevado potencial para obter resultados de grande relevância para esta área, carregando certa vantagem devido a duas de suas características: elevado grau de paralelismo de dados e elevada velocidade de execução, consequência da evolução e avaliação de programas diretamente em código de máquina.

Um dos principais desafios da robótica se caracteriza pelas incertezas na locomoção e no sensoriamento. O paralelismo maciço das GPUs vem propiciando o surgimento de novas e poderosas ferramentas para quantificar incertezas através de sua capacidade de executar simulações de Monte Carlo em tempo real como parte de um sistema de controle de malha fechada [96]. Ao unir a propagação de incertezas baseada em GPU com estratégias de controle ótimo, veículos robóticos podem cumprir seus objetivos em ambientes desconhecidos, além de serem capazes de lidar com perturbações inesperadas. Uma possível abordagem seria usar GMGP para evoluir estratégias de controle em tempo real para tais veículos, tornando-os capazes de evitar obstáculos, de contornar eventuais falhas elétricas ou mecânicas e de se adaptar às mudanças

ambientais. O lançamento do processador Nvidia Tegra K1, para aplicações móveis e de baixo consumo, abre também a possibilidade de se desenvolver este sistema de forma embarcada. Uma vez que tal processador possui núcleos de CPU e de GPU, a GMGP poderia ser executada, em um ou mais destes processadores (redundantes ou não), conjuntamente com os demais processos envolvidos no sistema de controle, tais como atualizações em tempo real do modelo do veículo, simulações de Monte Carlo etc.