



**Cleomar Pereira da Silva**

**Programação Genética Maciçamente Paralela  
em GPUs**

**Tese de Doutorado**

Tese apresentada ao Programa de Pós-graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica da PUC-Rio como requisito parcial para obtenção do título de Doutor em Engenharia Elétrica

Orientador: Prof. Marco Aurélio Cavalcanti Pacheco  
Co-Orientador: Prof. Douglas Mota Dias  
Co-Orientadora: Profa. Cristiana Barbosa Bentes

Rio de Janeiro  
Setembro de 2014



**Cleomar Pereira da Silva**

**Programação Genética Maciçamente Paralela  
em GPUs**

Tese apresentada ao Programa de Pós-graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica do Centro Técnico Científico da PUC-Rio como requisito parcial para obtenção do título de Doutor em Engenharia Elétrica. Aprovada pela Comissão Examinadora abaixo assinada.

**Prof. Marco Aurélio Cavalcanti Pacheco**

Orientador  
Departamento de Engenharia Elétrica — PUC-Rio

**Prof. Douglas Mota Dias**

Co-Orientador  
Departamento de Engenharia Elétrica — PUC-Rio

**Profa. Cristiana Barbosa Bentes**

Co-Orientadora  
Universidade do Estado do Rio de Janeiro – UERJ

**Prof. Ricardo Cordeiro de Farias**

Universidade Federal do Rio de Janeiro – UFRJ

**Prof. Renato Portugal**

Laboratório Nacional de Computação Científica – LNCC

**Profa. Noemi Rodriguez**

Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio

**Prof. Esteban Walter Gonzalez Clua**

Universidade Federal Fluminense – UFF

**Prof. José Franco Machado do Amaral**

Universidade do Estado do Rio de Janeiro – UERJ

**Prof. José Eugênio Leal**

Coordenador Setorial do Centro Técnico Científico — PUC-Rio

Rio de Janeiro, 11 de Setembro de 2014

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

### **Cleomar Pereira da Silva**

Possui graduação em Engenharia Elétrica pela Universidade Federal de Santa Maria (2008) e mestrado em Engenharia Elétrica pela Pontifícia Universidade Católica do Rio de Janeiro (2011).

#### Ficha Catalográfica

Silva, Cleomar Pereira da

Programação genética maciçamente paralela em GPUs / Cleomar Pereira da Silva ; orientador: Marco Aurélio Cavalcanti Pacheco ; co-orientador: Douglas Mota Dias ; co-orientadora: Cristiana Barbosa Bentes. – 2014.

134 f. : il. (color.) ; 30 cm

Tese (doutorado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2014.

Inclui bibliografia

1. Engenharia elétrica – Teses. 2. Programação genética. 3. Inspiração quântica. 4. GPUs. 5. Código de máquina. I. Pacheco, Marco Aurélio Cavalcanti. II. Dias, Douglas Mota. III. Bentes, Cristiana Barbosa. IV. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. V. Título.

CDD: 621.3

À minha namorada Daniela, aos meus pais Antonio Coraci e Maria Rosa e  
aos meus irmãos Edimar e Rudinéia.

## Agradecimentos

Ao CNPq, à PUC-Rio e ao ICA pelo suporte financeiro prestado ao longo destes anos.

Ao Instituto Federal Catarinense por todo apoio e liberação de atividades de trabalho em benefício da minha formação.

Ao meu orientador Professor Marco Aurélio Cavalcanti Pacheco pelo incentivo e por sempre acreditar na viabilidade do meu trabalho.

Ao meu co-orientador Professor Douglas Mota Dias pelo auxílio prestado nas longas discussões técnicas sobre o funcionamento do modelo.

À minha co-orientadora Professora Cristiana Barbosa Bentes pela enorme contribuição prestada na organização das ideias.

Aos meus pais Antonio Coraci e Maria Rosa e aos meus irmãos Edimar e Rudinéia por terem me proporcionado um ambiente de vivência em família.

E em especial, à minha namorada Daniela, pela pessoa especial e querida que ela é, e por ser a principal razão de seguir em frente.

## Resumo

Silva, Cleomar Pereira; Pacheco, Marco Aurélio Cavalcanti; Dias, Douglas Mota; Bentes, Cristiana Barbosa. **Programação Genética Maciçamente Paralela em GPUs**. Rio de Janeiro, 2014. 134p. Tese de Doutorado — Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

A Programação Genética permite que computadores resolvam problemas automaticamente, sem que eles tenham sido programados para tal. Utilizando a inspiração no princípio da seleção natural de Darwin, uma população de programas, ou indivíduos, é mantida, modificada baseada em variação genética, e avaliada de acordo com uma função de aptidão (*fitness*). A programação genética tem sido usada com sucesso por uma série de aplicações como projeto automático, reconhecimento de padrões, controle robótico, mineração de dados e análise de imagens. Porém, a avaliação da gigantesca quantidade de indivíduos gerados requer excessiva quantidade de computação, levando a um tempo de execução inviável para problemas grandes. Este trabalho explora o alto poder computacional de unidades de processamento gráfico, ou GPUs, para acelerar a programação genética e permitir a geração automática de programas para grandes problemas. Propomos duas novas metodologias para se explorar a GPU em programação genética: compilação em linguagem intermediária e a criação de indivíduos em código de máquina. Estas metodologias apresentam vantagens em relação às metodologias tradicionais usadas na literatura. A utilização de linguagem intermediária reduz etapas de compilação e trabalha com instruções que estão bem documentadas. A criação de indivíduos em código de máquina não possui nenhuma etapa de compilação, mas requer engenharia reversa das instruções que não estão documentadas neste nível. Nossas metodologias são baseadas em programação genética linear e inspiradas em computação quântica. O uso de computação quântica permite uma convergência rápida, capacidade de busca global e inclusão da história passada dos indivíduos. As metodologias propostas foram comparadas com as metodologias existentes e apresentaram ganhos consideráveis de desempenho. Foi observado um desempenho máximo de até 2,74 trilhões de GPops (operações de programação genética por segundo) para o *benchmark* Multiplexador de 20 bits e foi possível estender a programação genética para problemas que apresentam bases de dados de até 7 milhões de amostras.

## Palavras-chave

Programação genética; inspiração quântica; GPUs; código de máquina.

## Abstract

Silva, Cleomar Pereira; Pacheco, Marco Aurélio Cavalcanti (Advisor); Dias, Douglas Mota; Bentes, Cristiana Barbosa. **Massively Parallel Genetic Programming on GPUs**. Rio de Janeiro, 2014. 134p. PhD Thesis — Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Genetic Programming enables computers to solve problems automatically, without being programmed to it. Using the inspiration in the Darwin's Principle of natural selection, a population of programs or individuals is maintained, modified based on genetic variation, and evaluated according to a fitness function. Genetic programming has been successfully applied to many different applications such as automatic design, pattern recognition, robotic control, data mining and image analysis. However, the evaluation of the huge amount of individuals requires excessive computational demands, leading to extremely long computational times for large size problems. This work exploits the high computational power of graphics processing units, or GPUs, to accelerate genetic programming and to enable the automatic generation of programs for large problems. We propose two new methodologies to exploit the power of the GPU in genetic programming: intermediate language compilation and individuals creation in machine language. These methodologies have advantages over traditional methods used in the literature. The use of an intermediate language reduces the compilation steps, and works with instructions that are well-documented. The individuals creation in machine language has no compilation step, but requires reverse engineering of the instructions that are not documented at this level. Our methodologies are based on linear genetic programming and are inspired by quantum computing. The use of quantum computing allows rapid convergence, global search capability and inclusion of individuals' past history. The proposed methodologies were compared against existing methodologies and they showed considerable performance gains. It was observed a maximum performance of 2,74 trillion GPops (genetic programming operations per second) for the 20-bit Multiplexer benchmark, and it was possible to extend genetic programming for problems that have databases with up to 7 million samples.

## Keywords

Genetic programming; quantum inspired; graphics processing units; machine code.

## Sumário

1	Introdução	15
1.1	Descrição do problema	15
1.2	Objetivos	16
1.3	Contribuições	18
1.4	Organização do trabalho	19
2	Trabalhos Relacionados	20
2.1	Programação Genética	20
2.2	Acelerando a PG	23
2.3	Algoritmos de Inspiração Quântica	28
3	Programação Genética	33
3.1	Algoritmo Básico	33
3.2	Programação Genética com Representação dos Indivíduos em Árvore	34
3.3	Programação Genética Linear	36
3.4	Evolução de Programas em Linguagem de Montagem ou Linguagem de Máquina	38
4	Algoritmos com Inspiração Quântica e PG	39
4.1	Programação Genética Linear com Inspiração Quântica – QILGP	40
4.2	Representação	41
4.3	Observação	41
4.4	Avaliação de um Indivíduo Clássico	44
4.5	Operador Quântico	44
4.6	Algoritmo Evolutivo	45
5	Unidades de Processamento Gráfico – GPUs	47
5.1	Arquitetura da GPU	49
5.2	Modelo de Programação	50
5.3	Compilação	53
6	Acelerando PG em GPU - Manipulando a Linguagem Intermediária	54
6.1	A linguagem PTX	54
6.2	Modelo Básico	57
6.3	Conjunto de Funções	58
6.4	Indivíduo	59
6.5	Kernel	61
6.6	Detalhes de Implementação	62
7	Acelerando PG em GPU - Manipulando a Linguagem de Máquina	64
7.1	A Linguagem CUBIN	64
7.2	GMGP	67
7.3	Instruções	70
7.4	Indivíduo	71
7.5	GMGP-h	72



7.6	GMGP-gpu	73
7.7	GMGP-gpu+	76
7.8	Detalhes de Implementação	77
8	Resultados Experimentais	<b>78</b>
8.1	Ambiente de Execução	78
8.2	Benchmarks	79
8.3	Base de Comparação	80
8.4	Regressão Simbólica	81
8.5	Avaliando Instruções Booleanas: Multiplexador de 20 bits	93
8.6	O Problema de Classificação: Detecção de Intrusos na Rede	97
8.7	Comparando as Três Versões de GMGP	101
9	Conclusões	<b>122</b>
9.1	Trabalhos Futuros	124

## Lista de figuras

3.1	Fluxograma do algoritmo da Programação Genética.	34
3.2	Operador de cruzamento ou <i>crossover</i> da PG.	36
3.3	Cálculo da Função de Avaliação da Aptidão para os indivíduos da população de programas de computador.	37
4.1	Ilustração de uma implementação de <i>qudit</i> .	43
4.2	Criação de um gene clássico a partir da observação de um gene quântico.	44
4.3	Algoritmo evolucionário da QILGP.	46
5.1	Comparação de desempenho entre CPU e GPU em bilhões de operações de ponto flutuante por segundo.	47
5.2	Comparação da largura de banda da memória entre CPU e GPU em bilhões de bytes por segundo.	48
5.3	SM ( <i>streaming multiprocessor</i> ) da arquitetura de uma GPU Kepler com o chip GK110.	50
5.4	Organização dos espaços de memória de uma GPU.	51
5.5	Hierarquia de organização das <i>threads</i> : grid e blocos de <i>threads</i> .	52
5.6	Etapas de compilação da parte do código destinada a executar na GPU.	53
6.1	Exemplo de código em PTX.	56
6.2	Esquema básico da PG utilizado pela metodologia Pseudo-Assembly.	57
6.3	Esquema de paralelização da PG na metodologia Pseudo-Assembly.	58
6.4	Níveis de paralelismo de Pseudo-assembly na GPU.	59
6.5	Tempo de execução (em segundos) e aceleração obtida quando a metodologia Pseudo-assembly utiliza o nível de otimização -O0, tomando o nível -O4 como referência.	63
7.1	Laço utilizado para evitar a remoção da instrução PTX <i>add</i> .	68
7.2	As duas soluções de GMGP para explorar o paralelismo de um ambiente heterogêneo composto de CPU e GPU: GMGP-h e GMGP-gpu.	69
7.3	Metodologia GMGP-gpu: toda a PG é executada dentro da GPU por diferentes <i>kernels</i> .	74
7.4	Metodologia GMGP-gpu+: toda a PG é executada dentro da GPU por diferentes <i>kernels</i> .	76
8.1	Tempo de execução (em segundos) das metodologias Pseudo-Assembly, Interpreter e GMGP para os <i>benchmarks Chapéu Mexicano</i> e <i>Salutowicz</i> .	87
8.2	Acelerações de GMGP e Interpreter usando Pseudo-Assembly como referência para os <i>benchmarks Chapéu Mexicano</i> e <i>Salutowicz</i> .	88
8.3	Acelerações de GMGP com relação a Interpreter para os <i>benchmarks Chapéu Mexicano</i> e <i>Salutowicz</i> .	89
8.4	Tempos intermediários de GMGP.	90

8.5	Tempos intermediários de Interpreter.	91
8.6	Tempo de execução (em segundos) de GMGP, Pseudo-Assembly e Serial para os <i>benchmarks Chapéu Mexicano</i> e <i>Salutowicz</i> .	92
8.7	Acelerações de Pseudo-Assembly em relação ao Serial para os <i>benchmarks Chapéu Mexicano</i> e <i>Salutowicz</i> .	93
8.8	Acelerações de GMGP com relação a Serial para os <i>benchmarks Chapéu Mexicano</i> e <i>Salutowicz</i> .	94
8.9	As três imagens utilizadas para treinamento do Filtro Sobel em tons de cinza.	109
8.10	As duas imagens utilizadas para validação do Filtro Sobel em tons de cinza.	109
8.11	Imagem utilizada para teste do Filtro Sobel em tons de cinza.	110
8.12	Resultados obtidos ao aplicar os melhores indivíduos evoluídos por GMGP-h, GMGP-gpu e GMGP-gpu+ na imagem de teste para o <i>Filtro Sobel</i>	115
8.13	Resultados obtidos ao aplicar os melhores indivíduos evoluídos por GMGP-h, GMGP-gpu e GMGP-gpu+ na imagem de teste para a <i>Restauração Salt-and-Pepper</i> .	121

## Lista de tabelas

2.1	Tabela comparativa dos trabalhos relacionados encontrados na literatura, referentes à aceleração de PG.	32
4.1	Descrição funcional das instruções para a plataforma Intel x86.	40
6.1	Descrição funcional das instruções em precisão simples de ponto flutuante para Pseudo-assembly.	60
7.1	Descrição funcional das instruções em precisão simples de ponto flutuante, incluindo as instruções adicionais para problemas de classificação, para GMGP.	66
7.2	Descrição funcional das instruções booleanas para GMGP.	67
7.3	Representação hexadecimal do código de máquina de GPU da instrução add.	71
8.1	Parâmetros utilizados para <i>Chapéu Mexicano</i> e <i>Salutowicz</i> .	84
8.2	Tempos intermediários de execução de todas as metodologias de PG em GPU (em segundos).	85
8.3	Desempenho de GMGP, Interpreter e Pseudo-Assembly para <i>Chapéu Mexicano</i> e <i>Salutowicz</i> em GPops.	88
8.4	Erro Absoluto Médio (MAE) na evolução da GPU para <i>Chapéu Mexicano</i> e <i>Salutowicz</i> .	93
8.5	Parâmetros para o Multiplexador de 20 bits.	96
8.6	Resultados da execução de GMGP para o Multiplexador de 20 bits em GPops.	97
8.7	Geração na qual GMGP resolveu o <i>Multiplexador de 20 bits</i> e número total de indivíduos utilizado na evolução.	97
8.8	Número total de amostras utilizadas para cada uma das classes <i>Normal</i> , <i>Probing</i> , <i>DoS</i> , <i>U2R</i> e <i>R2L</i> no <i>benchmark</i> de <i>Detecção de Intrusos na rede</i> .	99
8.9	Parâmetros para o <i>benchmark</i> de <i>Detecção de Intrusos na Rede</i> .	99
8.10	Resultados da execução de GMGP para o <i>benchmark</i> de <i>Detecção de Intrusos na Rede</i> em GPops, para cada uma das classes <i>Normal</i> , <i>Probing</i> , <i>DoS</i> , <i>U2R</i> e <i>R2L</i> .	100
8.11	Resultados da metodologia GMGP e resultados da literatura para o <i>benchmark</i> de <i>Detecção de Intrusos na Rede</i> , apresentando TAR para a classe <i>Normal</i> e TDR para as classes <i>Probing</i> , <i>DoS</i> , <i>U2R</i> e <i>R2L</i> .	101
8.12	Parâmetros para o <i>benchmark</i> <i>Mackey-Glass</i> .	102
8.13	Tempos de execução de GMGP-h, GMGP-gpu e GMGP-gpu+ para <i>Mackey-Glass</i> em segundos.	106
8.14	Resultados de GMGP-h, GMGP-gpu e GMGP-gpu+ para <i>Mackey-Glass</i> em GPops.	107
8.15	Erro RMS para a evolução na GPU do <i>benchmark</i> <i>Mackey-Glass</i> .	108
8.16	Parâmetros utilizados para o <i>Filtro Sobel</i> .	110

8.17	Tempos de execução de GMGP-h, GMGP-gpu e GMGP-gpu+ para o <i>Filtro Sobel</i> em segundos.	112
8.18	Resultados de GMGP-h, GMGP-gpu e GMGP-gpu+ executando o <i>Filtro Sobel</i> em GPops.	113
8.19	MAEs para a evolução na GPU do <i>Filtro Sobel</i> .	114
8.20	Parâmetros utilizados para a <i>Restauração Salt-and-Pepper</i> , para as etapas de classificação e de regressão.	117
8.21	Tempos de execução para o classificador da <i>Restauração Salt-and-Pepper</i> evoluído por GMGP-h, GMGP-gpu e GMGP-gpu+, em segundos.	118
8.22	Resultados de GMGP-h, GMGP-gpu e GMGP-gpu+ para a evolução do classificador da <i>Restauração Salt-and-Pepper</i> em GPops.	119
8.23	Tempos de execução para a etapa de suavização da imagem através de um modelo de regressão linear para a <i>Restauração Salt-and-Pepper</i> evoluído por GMGP-h, GMGP-gpu e GMGP-gpu+, em segundos.	119
8.24	Resultados de GMGP-h, GMGP-gpu e GMGP-gpu+ para a evolução da etapa de suavização da imagem através de um modelo de regressão linear para a <i>Restauração Salt-and-Pepper</i> em GPops.	120

*Eu tentei 99 vezes e falhei, mas na centésima tentativa eu consegui, nunca desista de seus objetivos mesmo que esses pareçam impossíveis, a próxima tentativa pode ser a vitoriosa.*

Albert Einstein