UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

LAURO CÁSSIO MARTINS DE PAULA

Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada

Goiânia
2014

sistema de bibliotecas ufg

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS (TEDE) NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[x] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação

Autor (a):	Lauro Cássio Martins de Paula				
E-mail:	lauro_cassio@hotmail.com				
Seu e-mail pode ser disponibilizado na página? [x]Sim [ ] Não					
Vínculo empregatício do autor	Nenhum (bolsista CAPES/DS)				
Agência de fomento:	Coord. Aperf. Pessoal Nível Superior			Sigla:	CAPES
País:	Brasil	UF:	DF	CNPJ:	00889834/0001-08
Título:	Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada				
Palavras-chave:	Seleção de variáveis, calibração multivariada, GPU, AF				
Título em outra língua:	Parallelization of APS and Firefly Algorithms for Variable Selection in Multivariate Calibration Problems				
Palavras-chave em outra língua:	Variable selection, multivariate calibration, GPU, FA, SPA				
Área de concentração:	Ciência da Computação				
Data defesa: (dd/mm/aaaa)	15/07/2014				
Programa de Pós-Graduação:	Ciência da Computação				
Orientador (a):	Prof. Dr. Anderson da Silva Soares				
E-mail:	anderson@inf.ufg.br				
Co-orientador (a):*					
E-mail:					

*Necessita do CPF quando não constar no SisPG

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [x] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF ou DOC da tese ou dissertação.

O sistema da Biblioteca Digital de Teses e Dissertações garante aos autores, que os arquivos contendo eletronicamente as teses e ou dissertações, antes de sua disponibilização, receberão procedimentos de segurança, criptografia (para não permitir cópia e extração de conteúdo, permitindo apenas impressão fraca) usando o padrão do Acrobat.

Lauro Cássio Martins de Paula

Assinatura do (a) autor (a)

Data: 12 / 09 / 2014

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

LAURO CÁSSIO MARTINS DE PAULA

Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Anderson da Silva Soares

Goiânia
2014

Dados Internacionais de Catalogação na Publicação na (CIP)
GPT/BC/UFG

Paula, Lauro Cássio Martins de.
P324p Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada [manuscrito] / Lauro Cássio Martins de Paula. - 2014.
xv, 81 f. : il., figs, tabs.

Orientador: Prof. Dr. Anderson da Silva Soares
Dissertação (Mestrado) – Universidade Federal de Goiás, Instituto de Informática, 2014.
Bibliografia.
Inclui lista de figuras, abreviaturas, siglas e tabelas.
Apêndices.
1. Algoritmos (computação) 2. Calibração multivariada 3. Regressão linear múltipla I. Título.
CDU: 004.421

Lauro Cássio Martins de Paula

Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada

Dissertação defendida no Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 15 de julho de 2014, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Anderson da Silva Soares
Instituto de Informática - UFG
Presidente da Banca

Prof. Dr. Wellington Santos Martins
Instituto de Informática - UFG

Prof. Dr. Clarimar José Coelho
Departamento de Computação - PUC-GO

Lauro Cássio Martins de Paula (citação bibliográfica: PAULA, L. C. M.)

Graduou-se em Ciência da Computação pela PUC Goiás - Pontifícia Universidade Católica de Goiás, com ênfase em Matemática Computacional. Durante sua graduação, foi monitor no departamento de Computação da PUC Goiás e aluno de iniciação científica pela UFG - Universidade Federal de Goiás. Durante o Mestrado, foi bolsista da CAPES e obteve algumas publicações importantes, as quais contribuíram para o desenvolvimento desta dissertação. Atualmente atua nos seguintes temas de pesquisa: paralelização de métodos para solução de sistemas lineares, paralelização de algoritmos em problemas de calibração multivariada e paralelização de algoritmos para montagem do genoma humano.

Dedico este trabalho especialmente a Deus, ao meu pai e minha mãe.

Agradecimentos

Primeiramente, agradeço a Deus pela vida maravilhosa que sempre me proporcionou, pela sabedoria que tem me fornecido ao longo dos anos e pela oportunidade de realizar mais este trabalho.

Agradeço aos meus pais, Carlos Gardel de Paula e Beti Martins de Paula, pela paciência que sempre tiveram comigo, pela compreensão nos momentos de tribulação, pelo carinho que sempre me deram e pelo apoio em todos os sentidos. Sem eles, eu não teria conseguido chegar até aqui.

Ao meu orientador, Prof. Dr. Anderson da Silva Soares, por sua paciência e importantíssima contribuição para o desenvolvimento deste trabalho, pela confiança em mim depositada, por todos os conselhos que me forneceu e por me ajudar na escrita dos tão importantes artigos, os quais foram importantíssimos para a conclusão deste Mestrado e para a minha aceitação no Doutorado.

Aos meus amigos, Prof. Dr. Leonardo Barra Santana de Souza e Me. Leandro Barra Santana de Souza, por terem me proporcionado a oportunidade de realizar dois projetos de iniciação científica durante a minha Graduação, pelo apoio moral e por me incentivarem a ser um pesquisador. Sem essa parceria, dificilmente eu teria conseguido iniciar esse Mestrado. Em especial, ao Prof. Dr. Leonardo pelo fornecimento de uma das cartas de recomendação tanto para o Mestrado quanto para o Doutorado e por me proporcionar a oportunidade de trabalhar no Centro Integrado de Aprendizagem em Rede (CIAR) da UFG, cuja a bolsa oferecida foi muito importante para minha vida financeira.

Agradeço ao Prof. Dr. Clarimar José Coelho pelas colaborações, pela paciência, pelos seus ensinamentos, por ter sido meu orientador na Graduação, por me proporcionar a oportunidade de ser membro do grupo de pesquisa em Computação Científica da PUC Goiás, lugar onde aprendo cada vez mais, e por ser um dos professores que me forneceu carta de recomendação tanto para o Mestrado quanto para o Doutorado.

À minha namorada, Thais Mirele Coelho Borges, pelo carinho, pela paciência e compreensão em todos os momentos. Sem seu amor, com certeza, essa jornada teria sido mais árdua.

À minha tia, Dalva Aparecida Borges, pela paciência, pelas nossas conversas agradáveis e pelos seus ensinamentos bíblicos, os quais são fundamentais para o equilíbrio

espiritual.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo fornecimento de minha bolsa de estudo durante o Mestrado. Sem esse apoio financeiro, dificilmente eu teria conseguido realizar este trabalho e adquirir os equipamentos que foram necessários para a obtenção de resultados.

Ao Prof. Dr. Wellington Martins por ter sido meu primeiro orientador no Mestrado, por seus conselhos e pelas suas importantes colaborações nas correções desta Dissertação.

Enfim, agradeço a todos aqueles que, de alguma forma, contribuíram para a realização deste Mestrado, que, na verdade, foi um meio para se atingir um fim.

"... Um homem precisa viajar. Por sua conta, não por meio de histórias, imagens, livro ou TV. Precisa viajar por si, com seus olhos e pés, para entender o que é seu. Para um dia plantar suas próprias árvores e dar-lhes valor. Conhecer o frio para desfrutar do calor. E o oposto. Sentir a distância e o desabrigo para estar bem sob o próprio teto. Um homem precisa viajar para lugares que não conhece, para quebrar essa arrogância que nos faz ver o mundo como imaginamos e não simplesmente como ele é ou pode ser..."

Amyr Klink,
Mar sem fim: 360º ao redor da Antártica.

Resumo

de Paula, Lauro Cássio Martins. **Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada**. Goiânia, 2014. 77p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

O problema de seleção de variáveis consiste na seleção de atributos de uma determinada amostra que melhor contribuem para a predição da propriedade de interesse. O Algoritmo das Projeções Sucessivas (APS) tem sido bastante utilizado para seleção de variáveis em problemas de calibração multivariada. Entre os algoritmos bioinspirados, nota-se que o Algoritmo $Firefly$ (AF) é um novo método proposto com potencial de aplicação em vários problemas do mundo real, tais como problemas de seleção de variáveis. A principal desvantagem desses dois algoritmos encontra-se em suas cargas computacionais, conforme seu tamanho aumenta com o número de variáveis. Os avanços recentes das *Graphics Processing Units* (GPUs) têm fornecido para os algoritmos uma poderosa plataforma de processamento e, com isso, sua utilização torna-se muitas vezes indispensável para a redução do tempo computacional. Nesse contexto, este trabalho propõe uma implementação paralela em GPU de um AF (AF-RLM) para seleção de variáveis usando modelos de Regressão Linear Múltipla (RLM). Além disso, apresenta-se duas implementações do APS, uma utilizando RLM (APS-RLM) e uma outra que utiliza a estratégia de Regressões Sequenciais (APS-RS). Tais implementações visam melhorar a eficiência computacional dos algoritmos. As vantagens das implementações paralelas são demonstradas em um exemplo envolvendo um número relativamente grande de variáveis. Em tal exemplo, ganhos de *speedup* foram obtidos. Adicionalmente, realiza-se uma comparação do AF-RLM com o APS-RLM e APS-RS. Com base nos resultados obtidos, mostra-se que o AF-RLM pode ser uma contribuição relevante para o problema de seleção de variáveis.

Palavras–chave

seleção de variáveis, calibração multivariada, GPU, regressão linear múltipla, algoritmo firefly, algoritmo das projeções sucessivas.

Abstract

de Paula, Lauro Cássio Martins. **Parallelization of APS and Firefly Algorithms for Variable Selection in Multivariate Calibration Problems**. Goiânia, 2014. 77p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The problem of variable selection is the selection of attributes for a given sample that best contribute to the prediction of the property of interest. Traditional algorithms as Successive Projections Algorithm (APS) have been quite used for variable selection in multivariate calibration problems. Among the bio-inspired algorithms, we note that the Firefly Algorithm (AF) is a newly proposed method with potential application in several real world problems such as variable selection problem. The main drawback of these tasks lies in them computation burden, as they grow with the number of variables available. The recent improvements of Graphics Processing Units (GPU) provides to the algorithms a powerful processing platform. Thus, the use of GPUs often becomes necessary to reduce the computation time of the algorithms. In this context, this work proposes a GPU-based AF (AF-RLM) for variable selection using multiple linear regression models (RLM). Furthermore, we present two APS implementations, one using RLM (APS-RLM) and the other sequential regressions (APS-RS). Such implementations are aimed at improving the computational efficiency of the algorithms. The advantages of the parallel implementations are demonstrated in an example involving a large number of variables. In such example, gains of speedup were obtained. Additionally we perform a comparison of AF-RLM with APS-RLM and APS-RS. Based on the results obtained we show that the AF-RLM may be a relevant contribution for the variable selection problem.

Keywords

variable selection, multivariate calibration, GPU, multiple linear regression, firefly algorithm, successive projections algorithm.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos

Lista de Abreviaturas e Siglas

APS	Algoritmo das Projeções Sucessivas	18
MIR	Mid-Infrared	19
NIR	Near-Infrared	31
AF	Algoritmo Firefly	19
AG	Algoritmo Genético	18
RLM	Regressão Linear Múltipla	23
PLS	Partial Least Squares	**??**
RMSEP	Root Mean Squared Error of Prediction	21
RS	Regressões Sequenciais	26
NIR	Near Infrared	31
SISD	Single Instruction Single Data	37
SIMD	Single Instruction Multiple Data	37
MISD	Multiple Instruction Single Data	37
MIMD	Multiple Instruction Multiple Data	37
CPU	Central Processing Unit	40
GPU	Graphics Processing Unit	36
CUDA	Compute Unified Device Architecture	41
OpenCL	Open Computing Language	42
API	Application Programming Interface	42
MAPE	Mean Absolute Percentage Error	22
PRESS	Predicted Residual Sums of Squares	22
AIC	Akaike Information Criteria	22
BIC	Bayesian Information Criteria	22

CAPÍTULO 1

Introdução

Uma das características mais interessantes dos métodos instrumentais modernos é o número de variáveis que podem ser obtidas em uma única amostra. Dispositivos como espectrofotômetros[1] têm gerado uma grande quantidade de dados com milhares de variáveis, que vem aumentando consideravelmente ao longo dos anos [79]. Geralmente, isso ocorre porque, para cada amostra, gera-se diversas medidas de absorbâncias em cada comprimento de onda. Logo, quanto maior a resolução do dispositivo, maior será o número de variáveis obtidas [70]. Em contrapartida, o *clock* dos processadores baseados em silício não tem aumentado de forma proporcional ao número de variáveis que podem ser obtidas, sendo necessário processar mais informações com o mesmo poder computacional [92]. Como consequência, a utilização de técnicas eficientes para a seleção de variáveis torna-se importante no sentido de lidar com dados cada vez maiores [41], [19]. Além disso, uma infra-estrutura para computação paralela pode contribuir significativamente para a obtenção de um bom desempenho computacional [16].

A quimiometria é uma área de estudo que se refere à aplicação de métodos estatísticos e matemáticos a problemas de origem química. Uma das sub-áreas de estudo da quimiometria, relacionada à química analítica, é a calibração multivariada. A calibração multivariada consiste no processo de construção de um modelo matemático que fornece a predição do valor de uma propriedade de interesse por meio da seleção de variáveis. A calibração multivariada pode ser definida como o processo de construção de um modelo matemático para relacionar variáveis independentes à variáveis dependentes [62]. No contexto deste trabalho, considera-se como variáveis independentes os conjuntos de dados coletados por meio de instrumentos, como espectrofotômetros, utilizados para a construção de modelos de calibração multivariada. Esses conjuntos podem ser organizados em uma matriz ($\mathbf{X}$), denominada matriz de variáveis e amostras (ou matriz de respostas instrumentais) [9]. As variáveis dependentes são os conjuntos de valores de referência obtidos em laboratório, que servem como parâmetro para a calibração do modelo. Tais conjuntos

[1] Espectrofotômetro é um aparelho de análise utilizado em laboratórios e sua função é medir e comparar a quantidade de luz (energia radiante) absorvida por uma determinada amostra [49].

podem ser representados por um vetor, denominado vetor de variáveis dependentes (ou valores de referência da propriedade de interesse) [59].

A partir do conjunto de variáveis independentes e dependentes é possível realizar a calibração, que estabelece o modelo matemático, e por meio de tal modelo realizar a predição da propriedade de interesse [62]. A predição é o processo de utilização do modelo para estimar a concentração da propriedade de interesse de uma determinada amostra. Por exemplo, a absorbância a um comprimento de onda pela amostra pode ser relacionada com a concentração do composto analisado. A Figura 1.1 mostra o processo de absorção que mede a absorbância de radiação com frequência q. A intensidade de absorção é dada pela Equação (1-1).

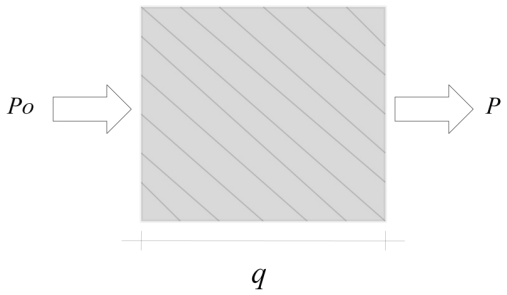


Figura 1.1: *Processo de espectroscopia de absorção.*

$$x(\lambda) = \log \frac{P_0(\lambda)}{P(\lambda)}, \tag{1-1}$$

onde $P_0(\lambda)$ é a radiação emitida pelo instrumento e $P(\lambda)$ a radiação absorvida pela amostra no comprimento de onda λ.

Quando duas ou mais ondas ficam sobrepostas, uma perturbação mútua nas ondas pode ocorrer. A sobreposição de ondas normalmente causa uma alta correlação (dependência linear) entre as variáveis e pode resultar em problemas matemáticos, como a multicolinearidade[2], no processo de obtenção do modelo de calibração [41]. Além disso, um número grande de variáveis pode ser obtido para uma determinada amostra, fazendo com que a matriz $\mathbf{X}$ tenha um número de colunas maior do que o número de linhas, implicando em um mal condicionamento da matriz. Uma matriz mal condicionada pode causar problemas de divergência e interferir diretamente na qualidade de predição da propriedade de interesse da amostra sendo analisada [17]. Portanto, a utilização de técnicas para seleção de variáveis pode ser uma alternativa viável para solucionar esse problema.

[2] A existência de correlação linear entre duas ou mais variáveis é definido como multicolinearidade [20].

Uma técnica tradicional para a seleção de variáveis é o Algoritmo das Projeções Sucessivas (APS), que é utilizado na seleção de variáveis para minimizar problemas de multicolinearidade em modelos de Regressão Linear Múltipla (RLM)[3] [4]. O APS é um algoritmo iterativo composto por três fases, onde na fase 1 são geradas as cadeias de variáveis e cada elemento de uma cadeia é selecionado de modo a obter a menor correlação linear com a anterior. Na fase 2, os subconjuntos de variáveis candidatas são avaliados de acordo com uma medida de erro que determina a precisão do modelo sendo representado. A fase 3 consiste na redução do número de variáveis selecionadas na fase 2, descartando aquelas que não contribuem significativamente para a capacidade preditiva do modelo.

Diversos trabalhos têm utilizado o APS para seleção de variáveis em problemas de calibração multivariada. Por exemplo, Araújo *et al.* [4] propuseram e utilizaram o APS para seleção de variáveis em análise de componentes espectroscópicos. Os resultados mostraram que o APS, em comparação com o PLS e um Algoritmo Genético (AG), pode fornecer os melhores resultados e um desempenho computacional superior.

Por meio da utilização do APS, Breitkreitz *et al.* [13] apresentaram uma estratégia para a determinação de enxofre em amostras de diesel. Os autores mostraram que o APS, em comparação com um AG, foi capaz de fornecer modelos com uma capacidade de predição mais adequada.

Recentemente, Soares *et al.* [93] apresentaram as características básicas do APS para RLM e relataram algumas variantes que têm sido propostas para a seleção de variáveis, transferência de calibração e relacionamento de estudos quantitativos. Para ilustração, foram apresentados dois estudos de caso envolvendo dados de espectros de reflectância no infravermelho próximo (*Near-Infrared* - NIR) para determinação de proteína em trigo.

Soares *et al.* [92] propuseram uma paralelização para o APS, usando modelos de RLM, a fim de explorar a capacidade de processamento de múltiplos núcleos em novas arquiteturas de processadores. Os resultados obtidos mostraram que foi possível reduzir o custo computacional do algoritmo quando dois ou mais núcleos de processamento estão disponíveis. Entretanto, a arquitetura de processamento explorada ficou limitada ao uso de no máximo quatro núcleos, enquanto que as *Graphics Processing Units* (GPU) atuais contém centenas (ou milhares) de núcleos de processamento, permitindo uma exploração de paralelismo mais eficiente.

Com o intuito de reduzir o tempo computacional, Soares *et al.* [91] apresentaram uma implementação da formulação de Regressões Sequenciais[4] para o APS. Tal formula-

[3]RLM é uma técnica estatística utilizada para a construção de modelos que descrevem as relações entre as variáveis explicativas de um determinado processo [68].

[4]Regressões Sequenciais (RS) é um procedimento empregado na fase 2 do APS para a redução do tempo

ção foi capaz de reduzir o esforço computacional evitando recalcular o vetor dos coeficientes de regressão ao se incluir uma nova variável no modelo. Na implementação original do APS, o cálculo é realizado a cada inserção de uma nova variável no modelo, fazendo com que o desempenho computacional seja reduzido quando existe um número relativamente grande de variáveis. A vantagem da implementação proposta foi demonstrada por meio de um exemplo envolvendo um conjunto de dados de espectros de reflectância no infravermelho próximo (NIR) de amostras de trigo. Em tal exemplo, ganhos computacionais foram obtidos em relação à implementação original. No entanto, a possibilidade de paralelização de tarefas para a obtenção de um desempenho computacional mais significativo não foi explorada.

Além do APS, algoritmos genéticos também têm sido utilizados para seleção de variáveis. Por exemplo, Costa Filho e Poppi [34] apresentaram um AG para seleção de variáveis em dados espectroscópicos no infravermelho médio (*Mid-Infrared* - MIR) para a determinação simultânea de glicose, maltose e frutose em solução. Com base nos resultados obtidos, eles mostraram que é possível obter um modelo mais simples em comparação ao método PLS, que realiza regressão por mínimos quadrados parciais para a construção do modelo [62].

Chan e Srinivasan [15] desenvolveram uma versão paralelizada de um AG, baseado em análise de componentes principais, para seleção de variáveis aplicado no problema do desafio *Tennessee Eastman* (*Tennessee Eastman challenge problem*). Os resultados mostraram que a abordagem paralela é doze vezes mais rápida do que o código sequencial. Entretanto, não foi realizada uma comparação com métodos tradicionais de seleção de variáveis como, por exemplo, o APS e o PLS.

1.1 Proposta do Trabalho

Com o aumento considerável do número de variáveis obtidas pelos atuais espectrofotômetros, uma exploração eficiente de paralelismo pelos algoritmos de seleção de variáveis torna-se cada vez mais importante e necessária no sentido de lidar com grandes volumes de dados em um tempo computacional aceitável. Portanto, este trabalho propõe três novas estratégias de paralelização de algoritmos para seleção de variáveis em problemas de calibração multivariada: APS com Regressão Linear Múltipla (APS-RLM), APS com Regressões Sequênciais (APS-RS) e Algoritmo $Firefly$ (AF). O AF é um método de otimização baseado no comportamento das características luminosas de vagalumes, que pode ser utilizado para selecionar variáveis em problemas de calibração multivariada [77].

computacional do algoritmo [91].

Para o APS-RLM, apresenta-se uma estrategia para exploração de paralelismo no cálculo de matrizes inversas usando GPU. A otimização de execução consistiu basicamente em paralelizar o cálculo da operação de inversão de matrizes, necessário para a computação dos coeficientes de regressão. Deve-se ressaltar que a inversão é apenas parte do cálculo necessário para a obtenção dos coeficientes e que, após o cálculo da inversão de forma paralela, é necessário realizar operações sequenciais para concluir o cálculo.

Para o APS-RS, propõe-se uma paralelização em GPU para a formulação das regressões sequenciais. A otimização de execução está em todo o cálculo necessário para a obtenção dos coeficientes. Foi realizado um re-equacionamento matemático de forma a permitir que os cálculos possam ser realizados em paralelo, de modo que todo o cálculo para se obter os coeficientes de regressão seja executado nos núcleos da GPU.

E, para o AF, propõe-se a implementação AF-RLM, que utiliza uma estratégia de paralelização em GPU para o cálculo do vetor dos coeficientes de regressão usando modelos de RLM. É importante destacar que, tanto quanto se sabe, ainda não existe algum trabalho na literatura que utiliza o AF para seleção de variáveis em problemas de calibração multivariada.

Objetiva-se mostrar que é possível reduzir o custo computacional do APS utilizando as implementações propostas. Além disso, com base nos resultados obtidos, mostra-se que, embora o APS seja considerado um algoritmo tradicional e eficiente para seleção de variáveis, o AF-RLM pode ser mais eficaz na construção de um modelo com uma capacidade de predição mais adequada. Adicionalmente, mostra-se que a paralelização do AF-RLM em uma GPU pode apresentar um tempo computacional significativamente reduzido.

O conjunto de dados empregado no trabalho consistiu em amostras integrais de trigo obtidas a partir de material vegetal de produtores canadenses, onde o teor de proteína foi escolhido como propriedade de interesse. Em outras palavras, o modelo de calibração multivariada obtido utilizando as implementações propostas teve como objetivo principal a predição da taxa de proteína contida nas amostras de trigo utilizadas.

1.2 Organização do Trabalho

O Capítulo 2 traz uma revisão sobre a calibração multivariada e sobre os três métodos implementados no trabalho. Os principais conceitos relacionados à computação paralela são descritos no Capítulo 3. O Capítulo 4 apresenta os detalhes relacionados à implementação dos métodos de seleção de variáveis APS-RLM, APS-RS e AF-RLM. O Capítulo 5 detalha os materiais e os métodos utilizados para a obtenção dos resultados. Os resultados obtidos são apresentados e analisados no Capítulo 6. Por fim, o Capítulo 7 mostra as considerações finais e aponta sugestões para trabalhos futuros.

CAPÍTULO 2

Calibração Multivariada

Um modelo de calibração multivariada fornece o valor de uma grandeza y baseado em valores medidos a partir de um conjunto de variáveis explicativas $(x_1, x_2, \ldots, x_k)$ [62], [101]. Tal modelo pode ser definido como:

$$\mathbf{y} = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k + \varepsilon, \tag{2-1}$$

em que β_0, β_1, ..., β_k, k = 1, 2, ..., K, são os coeficientes a serem determinados e ε uma parcela de erro aleatório.

Dada uma matriz $\mathbf{X}$ e o vetor $\mathbf{y}$, a modelagem de dados pode ser dividida em três subconjuntos: calibração ($\mathbf{X}_{cal}$ e $\mathbf{y}_{cal}$), validação ($\mathbf{X}_{val}$ e $\mathbf{y}_{val}$) e predição ($\mathbf{X}_{pred}$ e $\mathbf{y}_{pred}$). Os subconjuntos $\mathbf{X}_{cal}$ e $\mathbf{y}_{cal}$ são utilizados para se obter um modelo matemático que descreve a relação entre as variáveis.

A Equação (2-2) mostra como os coeficientes de regressão podem ser calculados utilizando a Equação (2-2) [57]:

$$\beta = (\mathbf{X}_{cal}^T \mathbf{X}_{cal})^{-1} \mathbf{X}_{cal}^T \mathbf{y}_{cal}, \tag{2-2}$$

em que $\mathbf{X}$ é a matriz de variáveis e amostras (coletadas por instrumentos como espectrofotômetro), $\mathbf{y}$ é o vetor das variáveis dependentes (ou propriedade de interesse obtida em laboratório, a qual serve como parâmetro de referência para a calibração do modelo) e β o vetor dos coeficientes de regressão.

Por meio da utilização do vetor β, é possível estimar a concentração da propriedade de interesse a partir da matriz $\mathbf{X}_{val}$:

$$\hat{\mathbf{y}} = \mathbf{X}_{val} \beta. \tag{2-3}$$

A capacidade preditiva (ou acurácia) de um modelo de RLM pode ser calculada pelo *Root Mean Squared Error of Prediction* (RMSEP), que é uma medida de erro absoluta:

$$\text{RMSEP} = \sqrt{\frac{\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}{N}}, \tag{2-4}$$

em que $\mathbf{y}$ é o conjunto dos valores atuais da propriedade de interesse, $\hat{\mathbf{y}}$ é o conjunto dos valores estimados de $\mathbf{y}$ e N o número de observações (linhas da matriz $\mathbf{X}$).

Uma outra medida de erro para determinar a capacidade preditiva de modelos de RLM que pode ser utilizada é o *Mean Absolute Percentage Error* (MAPE) [61]. MAPE é uma medida relativa que expressa erros como uma percentagem do dado atual e é definido pela Equação (2-5):

$$MAPE = \frac{\sum_{i=0}^{N} \mid \frac{y_i - \hat{y}_i}{y_i} \mid}{N}(100), \tag{2-5}$$

onde $\mathbf{y}$ é o conjunto dos valores atuais da propriedade de interesse, $\hat{\mathbf{y}}$ é o conjunto dos valores previstos (usando algum método) e N o número de observações (ou amostras) utilizadas no cálculo do MAPE.

Na estatística, há também uma técnica chamada *Predicted Residual Sums of Squares* (PRESS), proposta por Allen [2]. PRESS é uma medida útil para comparação de diferentes modelos [98]. É também conhecida como uma forma de validação cruzada, utilizada em análise de regressão para obter uma medida resumida do ajuste de um modelo à uma amostra de observações que não foram utilizadas para estimar o modelo [2]. O PRESS também pode ser utilizado como uma medida de previsibilidade para comparar e selecionar o melhor modelo. A Equação (2-6) mostra como o PRESS pode ser calculado:

$$PRESS = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{2-6}$$

onde $\mathbf{y}$ é o mesmo vetor usado nas Equações (2-4) e (2-5), $\hat{\mathbf{y}}$ é o vetor calculado na Equação (2-3) e N o número de observações.

Outras métricas de avaliação do modelo também podem ser utilizadas. Uma abordagem é a utilização de critérios de informação como o *Akaike Information Criteria* (AIC), proposto por Akayke [1], ou o *Bayesian Information Criteria* (BIC), proposto por Schwarz [85]. AIC é uma medida da qualidade relativa de um modelo para um determinado conjunto de dados. BIC é um critécio para seleção de modelos entre um conjunto finito de modelos, sendo relativamente análogo ao AIC. As Equações (2-7) e (2-8) mostram como o AIC e o BIC podem ser calculados, respectivamente:

$$AIC = \ln(\hat{\sigma}_a^2) + r\frac{2}{N} + 1, \tag{2-7}$$

$$BIC = \ln(\hat{\sigma}_a^2) + r\frac{\ln(N)}{N}, \tag{2-8}$$

em que $\ln(\hat{\sigma}_a^2)$ denota a estimativa de máxima verossimilhança de σ_a^2, r é o número de parâmetros estimados no modelo e N o número de observações [11].

Na abordagem de critérios de informação, modelos que fornecem um valor mínimo para o critério são preferidos. Geralmente, os valores de AIC e BIC são comparados entre vários modelos como base para a seleção do modelo (ou variável). Entretanto, uma desvantagem dessa abordagem é que vários modelos podem ter que ser estimados por máxima verossimilhança, o que pode ser caro por exigir um grande esforço computacional [11].

Um método simples para a obtenção dos coeficientes na Equação (2-1) é a Regressão Linear Múltipla (RLM). RLM é uma técnica estatística que pode ser utilizada para a construção de um modelo de calibração multivariada. Essa técnica pode exigir um número de observações (amostras) maior do que o número de variáveis. Entretanto, o oposto pode ocorrer em algumas aplicações (mais variáveis que amostras). Por exemplo, em problemas que envolvem determinação espectrométrica de uma grandeza física ou química, as variáveis explicativas correspondem a medidas realizadas em vários comprimentos de ondas [89]. Quanto mais comprimentos de ondas na análise espectrométrica, maior a precisão do aparelho e, consequentemente, maior o número de variáveis obtidas. Por outro lado, o número de amostras pode estar limitado, por exemplo, a um custo econômico para coleta de novas amostras [9].

O cálculo da matriz inversa envolvido na Equação (2-2) pode apresentar problemas de estabilidade devido à correlação linear entre as variáveis. A existência de correlação linear entre duas ou mais variáveis é definido como multicolinearidade [20]. Esse tipo de característica de dados pode reduzir a confiabilidade da estimativa dos coeficientes do modelo [17]. Em problemas de predição com modelos de regressão contendo muitas variáveis, a maioria delas podem não contribuir para a capacidade preditiva do modelo. Sendo assim, a seleção de um conjunto reduzido de variáveis, que influenciam positivamente no modelo, é importante para melhorar a eficiência dos algoritmos utilizados para a construção de modelos de RLM. Ainda, a identificação de um pequeno conjunto de variáveis explicativas é normalmente desejada em problemas de regressão [41].

O problema da determinação de uma equação apropriada associada a um subconjunto de variáveis independentes depende do critério usado para: *i*) analisar as variáveis; *ii*) selecionar o subconjunto; e *iii*) estimar os coeficientes na Equação (2-2). De acordo com Miller [67], as razões para utilizar somente algumas das variáveis disponíveis incluem:

- As estimativas de baixo custo ou previsões podem ser alcançadas por meio da redução do número de variáveis;
- A precisão pode ser aprimorada eliminando variáveis não informativas;
- Um conjunto de dados multivariados pode ser parsimoniosamente descrito.

A seleção de variáveis é uma das maneiras possíveis de se contornar o problema da multicolinearidade entre as variáveis. Como mostra a Seção 2.1, o APS e o AF são exemplos de algoritmos que podem ser utilizados para seleção de variáveis [11].

2.1 Métodos para Seleção de Variáveis

2.1.1 Algoritmo das Projeções Sucessivas

Proposto por Araújo *et al.* [4], [94], o Algoritmo das Projeções Sucessivas é uma técnica de seleção de variáveis que tem por objetivo minimizar problemas de multicolinearidade em RLM. O APS é um algoritmo iterativo que, dado uma variável inicial, insere-se uma nova variável (que possua a maior projeção ortogonal em relação à variável anterior) até que um número máximo m seja atingido. A variável de maior projeção contém o mínimo de redundância possível, minimizando o problema de multicolinearidade [62], [20]. O APS é composto por três fases:

1. A fase 1 consiste em operações de projeção realizadas na matriz $\mathbf{X}$. Tais projeções são utilizadas para a geração de cadeias de variáveis. Cada elemento de uma cadeia é selecionado de modo a obter a maior projeção ortogonal conforme a Equação (2-9):

$$\mathbf{P} = \mathbf{I} - \frac{\mathbf{X}^i(\mathbf{X}^i)^T}{(\mathbf{X}_i)^T\mathbf{X}_i}, \tag{2-9}$$

   em que $\mathbf{I}$ é uma matriz identidade de dimensões $N_{cal} \times N_{cal}$, $\mathbf{X}^i$ é a i-ésima coluna da matriz $\mathbf{X}$ e $\mathbf{P}$ a matriz de projeções.
2. Na fase 2, os subconjuntos de variáveis candidatas são avaliados de acordo com o erro do modelo.
3. A fase 3 consiste na redução do número de variáveis selecionadas na fase 2, descartando aquelas que não contribuem significativamente para a capacidade preditiva do modelo.

Como mostra o Algoritmo 2.1, na fase 1 são geradas K cadeias com M variáveis cada, onde

$$M = min(N_{cal} - 1, K). \tag{2-10}$$

Algoritmo 2.1: Fase 1 do APS.

1. **faça** $k = 1$
2. **enquanto** $k < K$
3. **faça** $x_j^1 = x_j$, j = 1, 2, ..., K
4. **faça** $\mathbf{S}_{1,k}$ = k
5. **faça** $i = 1$
6. **enquanto** $i < M$
7. Calcule a matriz **P** de projeções no sub-espaço ortogonal a $\mathbf{X}^i$ conforme a Equação (2-9)
8. Calcule os vetores projetados $x_j^{i+1} = \mathbf{P}^i x_j^i$, j = 1, 2, ..., K
9. Armazene o índice j^* da variável de maior projeção no elemento (i+1, k) da matriz **S**
10. **faça** $i = i + 1$
11. **fim enquanto** i
12. **faça** $k = k + 1$
13. **fim enquanto** k

Algoritmo 2.2: Fase 2 do APS.

1. **faça** $k = 1$
2. **enquanto** $k < K$
3. **faça** $m = 1$
4. **enquanto** $m < M$
5. Seja X_{km} um subconjunto de variáveis formado pelos m primeiros elementos da cadeia k gerada na fase 1
6. Seja ${S_{km}}^{-1}$ a matriz inversa da Equação (2-2)
7. Utilizando as variáveis contidas em X_{km}, calcule a inversa ${S_{km}}^{-1}$ e o restante da Equação (2-2)
8. Calcule o erro da cadeia k com m variáveis
9. **faça** $m = m + 1$
10. **fim enquanto** m
11. **faça** $k = k + 1$
12. **fim enquanto** k

Na fase 2, o APS utiliza o conjunto de validação para avaliar os subconjuntos de variáveis extraídas a partir das cadeias geradas na fase 1. Como resultado, o melhor subconjunto de variáveis é aquele que fornece o menor valor de erro entre os subconjuntos testados. O Algoritmo 2.2 mostra um pseudocódigo para a fase 2.

Devido à necessidade da construção de um modelo de RLM para cada subconjunto de variáveis candidatas, a fase 2 pode ser considerada o gargalo computacional do algoritmo quando comparada com as outras fases [91]. Além disso, o cálculo da matriz inversa envolvida na Equação (2-2) pode exigir um esforço computacional significante,

fazendo com que o desempenho seja reduzido [79]. Com isso, alguns trabalhos têm apresentado propostas para a redução do tempo computacional da fase 2 do APS. Por exemplo, Soares *et al.* [92] apresentaram uma paralelização para o APS na tentativa de explorar a habilidade de processadores *multicore*. Os resultados obtidos mostraram que foi possível reduzir o custo computacional do algoritmo quando dois ou mais núcleos de processamento estão disponíveis. Entretanto, as arquiteturas de processamento exploradas no trabalho estão limitadas ao uso de no máximo quatro núcleos de processamento.

Por existir centenas (ou até mesmo milhares) de núcleos de processamento disponíveis em uma GPU, calcular a inversa de matrizes usando programação paralela em GPUs pode ser computacionalmente mais eficiente [79]. Por esse motivo, este trabalho propõe uma estratégia de paralelização para o cálculo de inversão de matriz envolvido na fase 2 do APS (ver Capítulo 4). A vantagem da implementação proposta (APS-RLM) foi demonstrada em um exemplo envolvendo um conjunto de dados relativamente grandes.

2.1.2 Algoritmo das Projeções Sucessivas com Regressões Sequenciais

Na tentativa de reduzir o tempo computacional, Soares *et al.* [91] propuseram uma nova implementação do APS baseada no uso de um procedimento de regressões sequenciais. Esse procedimento foi empregado na fase 2 do APS. A formulação das regressões sequenciais reduz o tempo computacional evitando o cálculo de matrizes inversas.

Seja $\{x_1, x_2, \ldots, x_M\}$ uma cadeia de variáveis obtidas na fase 1. Na fase 2, essas variáveis são utilizadas para obter M modelos de RLM, iniciando-se a partir de um modelo com uma única variável (x_1), seguindo com duas (x_1, x_2), até M $(x_1, x_2, \ldots, x_M)$ variáveis. Cada um desses modelos pode ser obtido por um procedimento de cálculo dos mínimos quadrados. Tal procedimento requer a inversão de matrizes maiores a medida em que variáveis são adicionadas. Entretanto, a formulação das regressões sequenciais reduz o tempo computacional evitando-se a necessidade de calcular a inversa dessas matrizes [91].

A formulação das regressões sequenciais inicia-se a partir de uma única variável da seguinte maneira:

$$y = \beta_1^{(1)} x_1 + \varepsilon^{y|x_1}, \tag{2-11}$$

onde $\beta_1^{(1)}$ é o coeficiente de regressão e $\varepsilon^{y|x_1}$ o resíduo (parcela de erro aleatório) do modelo. Os sobrescritos (1) e $y|x_1$ denotam que uma variável independente é empregada no modelo e que y é regredido em x_1, respectivamente.

A estimativa dos mínimos quadrados de $\beta_1^{(1)}$ é dada por:

$$\hat{\beta}_1^{(1)} = \frac{\sum_{i=1}^{N} y_i x_{i,1}}{\sum_{i=1}^{N} (x_{i,1})^2}, \tag{2-12}$$

em que y_i e $x_{i,1}$ representam os valores de y e x para o i-ésimo objeto de calibração (amostra), respectivamente.

Por meio da utilização de uma notação similar, o modelo com duas variáveis pode ser escrito como:

$$y = \beta_1^{(2)} x_1 + \beta_2^{(2)} x_2 + \varepsilon^{y|x_1,x_2}. \tag{2-13}$$

De modo a obter $\beta_1^{(2)}$ e $\beta_2^{(2)}$, x_2 é inicialmente regredido em x_1 de acordo com um modelo da forma:

$$x_2 = \hat{\delta}_1^{x_2|x_1} x_1 + \varepsilon^{x_2|x_1}, \tag{2-14}$$

onde o coeficiente $\hat{\delta}_1^{x_2|x_1}$ pode ser calculado por uma regressão univariada como:

$$\hat{\delta}_1^{x_2|x_1} = \frac{\sum_{i=1}^{N} x_{i,2} x_{i,1}}{\sum_{i=1}^{N} (x_{i,1})^2}. \tag{2-15}$$

Então, $\hat{\beta}_1^{(2)}$ e $\hat{\beta}_2^{(2)}$ podem ser obtidos da seguinte forma:

$$\hat{\beta}_2^{(2)} = \frac{\sum_{i=1}^{N} e_i^{y|x_1} x_{i,2}}{\sum_{i=1}^{N} e_i^{x_2|x_1} x_{i,2}}, \hat{\beta}_1^{(2)} = \hat{\beta}_1^{(1)} - \hat{\delta}_1^{x_2|x_1} \hat{\beta}_2^{(2)}, \tag{2-16}$$

onde

$$e_i^{y|x_1} = y_i - \hat{\beta}_1^{(1)} x_{i,1}, \tag{2-17}$$

e

$$e_i^{x_2|x_1} = x_{i,2} - \hat{\delta}_1^{x_2|x_1} x_{i,1}. \tag{2-18}$$

Esse procedimento pode ser generalizado para obter um modelo com m variáveis a partir de um modelo com m - 1 variáveis, onde m varia entre 2 e M [91]. Para tal, a nova variável independente x_m é inicialmente regredida em $\{x_1, x_2, ..., x_{m-1}\}$ de acordo com um modelo da forma:

$$x_m = \hat{\delta}_1^{x_m|x_1,\ldots,x_{m-1}} x_1 + \hat{\delta}_2^{x_m|x_1,\ldots,x_{m-1}} x_2 + \ldots + \hat{\delta}_{m-1}^{x_m|x_1,\ldots,x_{m-1}} x_{m-1} + \varepsilon^{x_m|x_1,\ldots,x_{m-1}}. \tag{2-19}$$

Os coeficientes $\hat{\beta}$ do modelo com m variáveis são calculados como:

$$\hat{\beta}_m^{(m)} = \frac{\sum_{i=1}^{N} e_i^{y|x_1,\ldots,x_{m-1}} x_{i,m}}{\sum_{i=1}^{N} e_i^{x_m|x_1,\ldots,x_{m-1}} x_{i,m}}, \tag{2-20}$$

$$\hat{\beta}_{m-j}^{(m)} = \hat{\beta}_{m-j}^{(m-1)} - \hat{\delta}_{m-j}^{x_m|x_1,\ldots,x_{m-1}} \hat{\beta}_m^{(m)}, j = 1,\ldots,m-1, \tag{2-21}$$

onde

$$e_i^{x_m|x_1,\ldots,x_{m-1}} = x_{i,m} - (\hat{\delta}_1^{x_m|x_1,\ldots,x_{m-1}} x_{i,1} + \hat{\delta}_2^{x_m|x_1,\ldots,x_{m-1}} x_{i,2} + \ldots + \hat{\delta}_{m-1}^{x_m|x_1,\ldots,x_{m-1}} x_{i,m-1}), \tag{2-22}$$

e

$$e_i^{y|x_1,\ldots,x_{m-1}} = y_i - (\hat{\beta}_1^{(m-1)} x_{i,1} + \hat{\beta}_2^{(m-1)} x_{i,2} + \ldots + \hat{\beta}_{m-1}^{(m-1)} x_{i,m-1}). \tag{2-23}$$

Exemplo numérico

Sejam $\mathbf{X}_{3\times3}$ e $\mathbf{y}_{3\times1}$ as matrizes geradas aleatoriamente abaixo:

$$\mathbf{X} = \begin{bmatrix} 0,9528 & 0,5982 & 0,8368 \\ 0,7041 & 0,8407 & 0,5187 \\ 0,9539 & 0,4428 & 0,0222 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 0,3759 \\ 0,8986 \\ 0,4290 \end{bmatrix}. \tag{2-24}$$

Inicialmente, supõe-se um cenário em que seja utilizada apenas a primeira coluna de $\mathbf{X}$: $\mathbf{x}_1 = [0,9528 \quad 0,7041 \quad 0,9539]^T$. O coeficiente de regressão $\beta_1^{(1)}$ é obtido substituindo os respectivos valores na Equação (2-12), o que leva a:

$$\begin{aligned} \beta_1^{(1)} &= \frac{(0,3759 \times 0,9528) + (0,8986 \times 0,7041) + (0,4290 \times 0,9539)}{(0,9528)^2 + (0,7041)^2 + (0,9539)^2} \\ &= 0,6052. \end{aligned} \tag{2-25}$$

Agora adiciona-se a segunda coluna de $\mathbf{X}$: $\mathbf{x}_2 = [0,5982 \quad 0,8407 \quad 0,4428]^T$. Para obter os coeficientes de regressão $\beta_1^{(2)}$ and $\beta_2^{(2)}$, deve-se primeiramente regredir $\mathbf{x}_2$ em $\mathbf{x}_1$ de acordo com as Equações (2-14) e (2-15):

$$\begin{aligned} \hat{\delta}^{x_2|x_1} &= \frac{(0,5982 \times 0,9528) + (0,8407 \times 0,7041) + (0,4428 \times 0,9539)}{(0,9528)^2 + (0,7041)^2 + (0,9539)^2} \\ &= 0,6848, \end{aligned} \tag{2-26}$$

$$\varepsilon^{y|x_1} = \begin{bmatrix} 0,3759 \\ 0,8986 \\ 0,4290 \end{bmatrix} - 0,6052 \begin{bmatrix} 0,9528 \\ 0,7041 \\ 0,9539 \end{bmatrix} = \begin{bmatrix} -0,2007 \\ 0,4725 \\ -0,1483 \end{bmatrix} \tag{2-27}$$

e

$$\varepsilon^{x_2|x_1} = \begin{bmatrix} 0,5982 \\ 0,8407 \\ 0,4428 \end{bmatrix} - 0,6848 \begin{bmatrix} 0,9528 \\ 0,7041 \\ 0,9539 \end{bmatrix} = \begin{bmatrix} -0,0534 \\ 0,3586 \\ -0,2104 \end{bmatrix}. \tag{2-28}$$

Substituindo os valores encontrados na Equação (2-16), tem-se:

$$\begin{aligned} \beta_2^{(2)} &= \frac{(-0,2007 \times 0,5982) + (0,4725 \times 0,8407) + (-0,1483 \times 0,4428)}{(-0,0534 \times 0,5982) + (0,3586 \times 0,8407) + (-0,2104 \times 0,4428)} \\ &= 1,2032, \end{aligned} \tag{2-29}$$

$$\begin{aligned} \beta_1^{(2)} &= 0,6052 - (0,6848 \times 1,2032) \\ &= -0,2188. \end{aligned} \tag{2-30}$$

Finalmente, será adicionada a terceira coluna de $\mathbf{X}$: $\mathbf{x}_3 = [0,8368 \quad 0,5187 \quad 0,0222]^T$. O cálculo do coeficiente de regressão será feito por meio da estimativa de mínimos quadrados como mostra a Equação (2-2) [57]. De modo semelhante ao modelo univariado, deve-se primeiramente regredir $\mathbf{x}_3$ em $\mathbf{x}_2$ e $\mathbf{x}_1$. Para isso, sendo $\mathbf{X}_{12}$ uma matriz que contenha a primeira e a segunda coluna de $\mathbf{X}$, tem-se:

$$\begin{aligned} (\mathbf{X}_{12}^T \mathbf{X}_{12})^{-1} &= \left(\begin{bmatrix} 0,9528 & 0,7041 & 0,9539 \\ 0,5982 & 0,8407 & 0,4428 \end{bmatrix} \times \begin{bmatrix} 0,9528 & 0,5982 \\ 0,7041 & 0,8407 \\ 0,9539 & 0,4428 \end{bmatrix} \right)^{-1} \\ &= \begin{bmatrix} 3,0998 & -3,8952 \\ -3,8952 & 5,6879 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \hat{\delta}^{x_3|x_1,x_2} &= \begin{bmatrix} 3,0998 & -3,8952 \\ -3,8952 & 5,6879 \end{bmatrix} \times \begin{bmatrix} 0,9528 & 0,7041 & 0,9539 \\ 0,5982 & 0,8407 & 0,4428 \end{bmatrix} \times \begin{bmatrix} 0,8368 \\ 0,5187 \\ 0,0222 \end{bmatrix} \\ &= \begin{bmatrix} -0,0176 \\ 0,7728 \end{bmatrix}. \end{aligned} \tag{2-31}$$

Para obter $\varepsilon^{x_3|x_2,x_1}$ e $\varepsilon^{y|x_1,x_2}$, tem-se:

$$\begin{aligned}\varepsilon^{x_3|x_1,x_2} &= \begin{bmatrix} 0,3759 \\ 0,8986 \\ 0,4290 \end{bmatrix} - \left(-0,0176 \begin{bmatrix} 0,9528 \\ 0,7041 \\ 0,9539 \end{bmatrix} + 0,7728 \begin{bmatrix} 0,5982 \\ 0,8407 \\ 0,4428 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0,3913 \\ -0,1187 \\ -0,3033 \end{bmatrix} \end{aligned} \tag{2-32}$$

$$\begin{aligned}\varepsilon^{y|x_1,x_2} &= \begin{bmatrix} 0,8368 \\ 0,5187 \\ 0,0222 \end{bmatrix} - \left(-0,2188 \begin{bmatrix} 0,9528 \\ 0,7041 \\ 0,9539 \end{bmatrix} + 1,2032 \begin{bmatrix} 0,5982 \\ 0,8407 \\ 0,4428 \end{bmatrix} \right) \\ &= \begin{bmatrix} -0,1354 \\ 0,0410 \\ 0,1049 \end{bmatrix} \end{aligned} \tag{2-33}$$

Os coeficientes de regressão $\beta_1^{(3)}$, $\beta_2^{(3)}$ e $\beta_3^{(3)}$ são obtidos substituindo os respectivos valores na Equação (2-20):

$$\begin{aligned}\beta_3^{(3)} &= \frac{(-0,1354 \times 0,8368) + (0,0410 \times 0,5187) + (-0,1049 \times 0,0222)}{0,3913 \times 0,8368) + (-0,1187 \times 0,5187) + (-0,3033 \times 0,0222)} \\ &= -0,3459, \end{aligned} \tag{2-34}$$

$$\begin{aligned}\beta_2^{(3)} &= 1,2032 - 0,7728 \times (-0,3459) \\ &= 1,4706 \end{aligned} \tag{2-35}$$

$$\begin{aligned}\beta_1^{(3)} &= -0,2188 - (-0,0176) \times (-0,3459) \\ &= -0,2249. \end{aligned} \tag{2-36}$$

Finalmente, para fins de comparação, a regressão clássica por mínimos quadrados foi calculada para o exemplo em questão conforme mostrado abaixo:

$$(\mathbf{X}^T\mathbf{X})^{-1} = \left(\begin{bmatrix} 0,9528 & 0,7041 & 0,9539 \\ 0,5982 & 0,8407 & 0,4428 \\ 0,8368 & 0,5187 & 0,0222 \end{bmatrix} \times \begin{bmatrix} 0,9528 & 0,5982 & 0,8368 \\ 0,7041 & 0,8407 & 0,5187 \\ 0,9539 & 0,4428 & 0,0222 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} 3,1010 & -3,9476 & 0,0678 \\ -3,9476 & 7,9925 & -2,9821 \\ 0,0678 & -2,9821 & 3,8587 \end{bmatrix}$$

$$\beta = \begin{bmatrix} 3,1010 & -3,9476 & 0,0678 \\ -3,9476 & 7,9925 & -2,9821 \\ 0,0678 & -2,9821 & 3,8587 \end{bmatrix} \times \begin{bmatrix} 0,9528 & 0,7041 & 0,9539 \\ 0,5982 & 0,8407 & 0,4428 \\ 0,8368 & 0,5187 & 0,0222 \end{bmatrix} \times \begin{bmatrix} 0,3759 \\ 0,8986 \\ 0,4290 \end{bmatrix}$$

$$= \begin{bmatrix} -0,2249 \\ 1,4706 \\ -0,3459 \end{bmatrix}. \quad \text{(2-37)}$$

Como pode-se observar, o resultado expresso em (2-37) é o mesmo ao qual se havia chegado nas Equações (2-34), (2-35) e (2-36). Para ilustração, Soares *et al.* [91] apresentaram um exemplo envolvendo a determinação de proteína em amostras de trigo por espectrometria no infravermelho próximo (NIR). As previsões do modelo resultante exibiram um baixo valor de erro, e a implementação proposta proporcionou ganhos computacionais significantes em relação à implementação tradicional do APS. No entanto, apesar dos resultados obtidos, tal técnica não explorou os avanços recentes das arquiteturas computacionais nem a possibilidade de paralelização de tarefas. Consequentemente, esta dissertação apresenta uma implementação paralela (APS-RS) para a estratégia das regressões sequenciais proposta por Soares *et al.* [91], cujo o objetivo é aumentar o desempenho computacional do algoritmo (ver Capítulo 4).

2.1.3 Algoritmo *Firefly*

Metaheurísticas inspiradas na natureza têm sido ferramentas poderosas na solução de vários tipos de problemas. O Algoritmo *Firefly* (AF) é um algoritmo de otimização proposto recentemente por Yang [102], [103]. Tal algoritmo é baseado no comportamento das características luminosas de vagalumes. O AF simula o sistema de atração de vagalumes. Vagalumes produzem luminosidade como um sistema de sinalização para comunicar com outros vagalumes [6]. Essas características luminosas podem ser destacadas por três condições:

1. Cada vagalume é atraído por outros vagalumes independentemente do sexo;
2. A atratividade é proporcional à luminosidade do vagalume. Logo, para qualquer dois vagalumes, o que possui uma luminosidade menor será atraído pelo de maior luminosidade;
3. O sinal de luz intermitente produzido pelos vagalumes pode ser formulado de tal maneira que ele esteja relacionado com a função objetivo a ser otimizada [104].

No Algoritmo 2.3, existem dois pontos importantes: a variação da intensidade de luz; e a formulação da atratividade. A luminosidade I de um vagalume $\mathbf{x}$ pode ser determinada como $I(\mathbf{x}) \Rightarrow f(\mathbf{x})$. Entretanto, a atratividade ω é relativa. Ela varia com a distância r_{ij} entre o vagalume i e o vagalume j [105]. À medida em que a intensidade de luz diminui com a distância, a atratividade deve variar com o coeficiente de absorção γ.

Na forma mais simples, a intensidade de luz $I(r)$ varia com a distância r exponencialmente da seguinte forma:

$$I = I_o e^{-\gamma r}, \tag{2-38}$$

onde I_o é a intensidade de luz inicial e γ o coeficiente de absorção de luminosidade.

Como a atratividade de um vagalume é proporcional à intensidade de luz vista pelos vagalumes adjacentes, deve-se definir a atratividade ω de um vagalume como:

$$\omega = \omega_o e^{-\gamma r^2}, \tag{2-39}$$

onde ω_o é a atratividade em $r = 0$. Vale ressaltar que o expoente γr pode ser substituído por outras funções, tal como γr^m quando $m > 0$ [105].

A distancia entre dois vagalumes pode ser calculada utilizando a distancia cartesiana:

$$r_{ij} = \sqrt{(x_i - x_j)^2 - (y_i - y_j)^2}. \tag{2-40}$$

Quando um vagalume i é atraído por um vagalume j, seu movimento é determinado pela Equação (2-41):

$$x_i = x_i + \omega_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha\varepsilon_i, \tag{2-41}$$

onde o segundo termo se refere a atratividade entre os vagalumes, $\alpha \in [0, 1]$ e ε é um vetor de números gerados aleatóriamente.

Algoritmo 2.3: AF original.

1. Inicialize uma população **x** de vagalumes
2. Calcule a função objetivo $f(\mathbf{x})$ para cada vagalume
3. Defina o coeficiente de absorção de luminosidade γ
4. **enquanto** $t < MaxGen$, onde $MaxGen$ = número máximo de gerações (iterações)
5. **para** $i = 1$ **até** n
6. **para** $j = 1$ **até** n
7. Intensidade de luz de I_i em $\mathbf{x}_i$ é determinada a partir de $f(\mathbf{x})$ para o vagalume i
8. **se** $(I_j > I_i)$
9. Calcule a atratividade ω entre i e j, a qual varia com a distância r_{ij} por $exp[-\gamma r_{ij}]$
10. Mova o vagalume i em direção ao vagalume j em todas as d dimensões de acordo com a atratividade entre i e j
11. **fim se**
12. Avalie os novos vagalumes e atualize suas intensidades de luz
13. **fim para** j
14. **fim para** i
15. Classifique os vagalumes e encontre os melhores
16. **fim enquanto**
17. Resultados

Geralmente, na maioria dos trabalhos que fizeram uso do AF, assume-se $\omega_0 = 1$ e $\gamma = 1$ [105]. O parâmetro γ caracteriza a variação da atratividade, e seu valor é importante na determinação da velocidade da convergência e como o AF irá se comportar [104]. De acordo com Yang [106], o AF é baseado na técnica *swarm-intelligence* e possui vantagens em relação a outros algoritmos similares. Ainda, Yang [106] afirma que algumas variações do algoritmo *Particle Swarm Optimization* (PSO), tal como o *Accelerated* PSO, são um caso especial do AF quando $\gamma = 0$. No entanto, o AF possui duas principais vantagens:

1. Subdivisão automática: o AF é baseado na atratividade (entre os vagalumes), que diminui conforme a distância aumenta. Portanto, isso implica que toda a população de vagalumes pode se subdividir automaticamente em subgrupos e cada subgrupo pode se agrupar em um ótimo local [106].
2. Habilidade de lidar com multimodalidade: a subdivisão pode permitir que os vagalumes encontrem simultaneamente todos os ótimos desde que o tamanho da população seja suficientemente maior do que o número de modalidades [105].

Exemplo numérico

O AF pode ser adaptado para trabalhar com problemas binários, como o de seleção de variáveis [77]. Para ilustração, considere um problema pequeno de seleção de variáveis com apenas cinco variáveis disponíveis e somente três vagalumes. Pode-se utilizar as cinco primeiras variáveis disponíveis, as três primeiras linhas (amostras) da matriz **X** e os três primeiros elementos do vetor **y**, ambos descritos no início deste capítulo. Para os parâmetros do algoritmo, utilizou-se experimentalmente $\alpha = 0{,}2$, $\gamma = 1$ e $\omega_0 = 0{,}97$. Inicialmente, os vagalumes são números aleatórios uniformemente distribuídos no intervalo [0, 1]:

vagalume 1 = [0,95; 0,48; 0,45; 0,44; 0,92],
vagalume 2 = [0,23; 0,89; 0,01; 0,61; 0,73],
vagalume 3 = [0,60; 0,76; 0,82; 0,79; 0,17].

A seleção de variáveis é um problema binário. Logo, cada vagalume deve ser codificado. Toda variável maior que 0,5 é codificada para 1 (a variável será usada no modelo de regressão), e toda variável menor ou igual a 0,5 é codificada para 0 (a variável não será usada no modelo de regressão):

vagalume 1 codificado = [1; 0; 0; 0; 1],
vagalume 2 codificado = [0; 1; 0; 1; 1],
vagalume 3 codificado = [1; 1; 1; 1; 0].

Agora, cada vagalume poderá ser avaliado usando-se a Equação (2-2). A Equação (2-2) indica a luminosidade, ou seja, a intensidade de luz de cada vagalume. Na Equação (2-2), apenas as colunas da matriz **X** indicada pelo vagalume será usada na regressão. Após calcular a Equação (2-2) para cada vagalume, tem-se: [6,82; 10,97; 9,22]. Logo após, compara-se cada um dos vagalumes (todos contra todos). Por exemplo, o vagalume 2 possui uma luminosidade maior que a do vagalume 1. Então, deve-se mover o vagalume 1 em direção ao vagalume 2. Para isso, primeiramente calcula-se a distância entre os vagalumes usando a Equação (2-40):

r_{12} = [0,71; 0,40; 0,43; 0,17; 0,18].

Utilizando a distancia entre os vagalumes, pode-se calcular a atratividade usando a Equação (2-39). Como resultado, atualiza-se o vagalume 1. Vale ressaltar que o vagalume 1 atualizado exclui a primeira variável utilizada na solução original e passa a utilizar

a segunda e a quarta variável após “caminhar"em direção ao vagalume 2:

Novo vagalume 1 = [0,44; 0,75; 0,02; 0,53; 0,67],
Novo vagalume 1 codificado = [0; 1; 0; 1; 1].

Esse procedimento é repetido até que todas as soluções tenham sido atualizadas. As atualizações permitem que as soluções avancem em direção a aptidão ($fitness$) ótima. A solução que produz o melhor $fitness$ (menor valor RMSEP) é selecionada como a melhor solução global.

Trabalhos recentes têm utilizado o AF para solucionar diversos tipos de problemas. Por exemplo, Yang [103] forneceu uma descrição detalhada de um novo AF para aplicações de otimização multimodal. Lukasik e Zak [60] forneceram uma implementação de um AF para problemas de otimização contínua. Yang [105] propôs o uso de um AF para problemas de projeto não-lineares. Senthilnath *et al.* [87] fizeram uso de um AF para agrupamento em problemas de *benchmark* e compararam seu desempenho com outras técnicas também inspiradas na natureza. Gandomi *et al.* [39] apresentaram um AF para problemas de otimização contínuos e discretos. Horng [46] apresentou um novo método baseado no AF para compressão de imagens.

Husselmann e Hawick [47] propuseram uma paralelização, usando GPU, para um AF e descreveram algumas técnicas de particionamento espacial para a redução de interações entre entidades resultantes. Recentemente, Husselmann e Hawick [48] apresentaram uma implementação do AF para busca em espaço de árvore de expressão e aceleraram a computação do algoritmo usando GPUs. Na maioria desses trabalhos citados, os resultados mostraram que o AF supera outros algoritmos em termos de tempo computacional e otimização.

Baseado no sucesso dos trabalhos que fizeram uso do AF, foi possível descobrir que o mesmo pode ser utilizado para seleção de variáveis na solução de problemas de calibração multivariada [77]. Sendo assim, este trabalho apresenta uma implementação de um AF (AF-RLM) para seleção de variáveis usando RLM (ver Capítulo 4).

CAPÍTULO 3

Processamento Paralelo

Processamento Paralelo (computação paralela) é uma forma eficiente do processamento da informação, com ênfase na exploração de eventos concorrentes no processo computacional [69], [97]. A computação paralela tem contribuído em diversas áreas, que vão desde simulações computacionais a aplicações científicas em aplicativos para mineração de dados e processamento de transações [56]. Nesse sentido, este Capítulo detalha os principais conceitos envolvidos no paradigma de programação paralela.

A Seção 3.1 descreve a taxonomia de Flynn e apresenta algumas motivações para o uso da computação paralela. A arquitetura dos processadores *multicore* é descrita na Seção 3.2. A Seção 3.3 descreve os principais detalhes das GPUs. Já a Seção 3.4 aborda os principais detalhes sobre os dois modelos de programação paralela em GPU mais utilizados atualmente: CUDA [24] e OpenCL [100].

3.1 Visão Geral

Primordialmente, o computador foi desenvolvido como uma máquina sequencial. De forma análoga, a maioria das linguagens de programação requer que o programador especifique um algoritmo como uma sequência de instruções, que são executadas sequencialmente pelo processador. Cada instrução é executada como uma sequência de operações [42], [95]:

1. Busca e decodificação de instruções;
2. Cálculo dos endereços dos operandos;
3. Busca dos operandos na memória;
4. Cálculo com os operandos;
5. Armazenamento dos resultados na memória;

3.1.1 Taxonomia de Flynn

A Taxonomia (ou Classificação) de Flynn, definida por Michael J. Flynn em 1966, classifica a arquitetura computacional de acordo com o processamento do fluxo de

instrução e de dados [29], [86], [35].

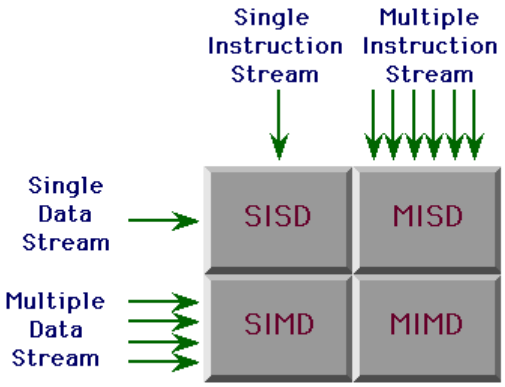


Figura 3.1: *Taxonomia de Flynn.*

Como mostra a Figura 3.1, tem-se as quatro classes descritas a seguir:

• *Single Instruction Single Data* (SISD): uma única unidade de controle é responsável por processar um único fluxo de instruções num único fluxo de dados;

• *Single Instruction Multiple Data* (SIMD): projetos de arquitetura onde uma única instrução é executada simultaneamente em múltiplos fluxos de dados;

• *Multiple Instruction Single Data* (MISD): este tipo de arquitetura é fonte de divergência entre pesquisadores da área de arquitetura de computadores. Navaux [69] afirma que nenhum sistema conhecido se encaixa nesta categoria. Entretanto, Quinn [83] aponta como exemplo para esta categoria o *systolic array*;

• *Multiple Instruction Multiple Data* (MIMD): arquiteturas que apresentam múltiplas unidades de processamento que manipulam diferentes fluxos de instrução com diferentes fluxos de dados.

A Tabela 3.1 mostra um exemplo de aplicação para cada uma das quatro classes [84]:

3.1.2 Motivação

Existem muitas razões para a utilização da computação paralela, destacando-se quatro delas:

Tabela 3.1: *Exemplos de aplicação das classes da Taxonomia de Flynn.*

Classe	Aplicação
SISD	Computadores pessoais com um único processador convencional.
SIMD	Suporcomputadores com arquitetura *processor array* ou *vector pipeline*.
MISD	Computadores utilizados para execução de algoritmos de criptografia.
MIMD	Computer-Aided Design (CAD), simulação, modelagem, etc.

- Execução de várias atividades simultaneamente;
- Capacidade de resolução de problemas maiores;
- Aumento do desempenho computacional;
- Utilização de recursos computacionais não disponíveis localmente.

Tradicionalmente, a computação paralela foi motivada pela resolução (ou simulação) de problemas com grande relevância científica e econômica, denominados *Grand Challenge Problems* (GCP). Tipicamente, os GCPs simulam alguns fenômenos que não podem ser medidos por experimentação (fenômenos climáticos, físicos, químicos, dentre outros). Ainda, as aplicações têm exigido o desenvolvimento de processadores cada vez mais rápidos. Isso ocorre porque a maioria dessas aplicações requerem um grande esforço computacional para processar grandes quantidades de dados.

Há aplicações e oportunidades onde a computação paralela pode apresentar ganhos de desempenho significativos. Na área de análise multivariada, por exemplo, alguns trabalhos têm utilizado a computação paralela na tentativa de aumentar o desempenho computacional. Entre eles, destacam-se os trabalhos de Soares *et al.* [92], Chan *et al.* [15] e Paula *et al.* [79].

Fora do contexto científico, tem ocorrido também uma explosão no volume de dados. Por exemplo, as corporações buscam cada vez mais ferramentas que transformem esses dados em informações valiosas, com o objetivo de fornecer respostas às necessidades de seus negócios e os ajudem a otimizar seus processos. Ao longo dos últimos anos, a expectativa da ciência e do mercado indicam a necessidade de redes de computadores cada vez mais rápidas, sistemas distribuídos altamente escaláveis e arquiteturas de computadores multiprocessados, mostrando claramente que o processamento paralelo tem sido uma alternativa viável para a redução do tempo computacional [53], [80], [74].

Com o avanço da tecnologia, novas arquiteturas computacionais têm sido desenvolvidas. Soluções com vários processadores em uma mesma placa vêm sendo elaboradas, e processadores com vários núcleos de processamento são a nova tendência tecnológica na atualidade [30], [66].

3.2 Arquitetura Multicore

Devido ao constante aumento do *clock* (frequência) dos processadores para aumentar o desempenho do *hardware* e, consequentemente, a obtenção de um superaquecimento do processador, os projetistas têm aumentado o número de núcleos para explorar a lei de Moore[1] [27]. O superaquecimento do processador pode ocorrer por diversos motivos, sendo o *overclocking*[2] um dos principais. Com isso, desenvolveu-se a arquitetura *multicore*, a qual é uma palavra utilizada para definir qualquer processador que contém mais de um núcleo de processamento.

Por ser capaz de executar algumas tarefas em paralelo, os processadores *multicore* representam uma revolução na tecnologia e normalmente contém poucos núcleos, porém com um grande poder de processamento [27]. A Figura 3.2 mostra um exemplo de arquitetura *multicore*:

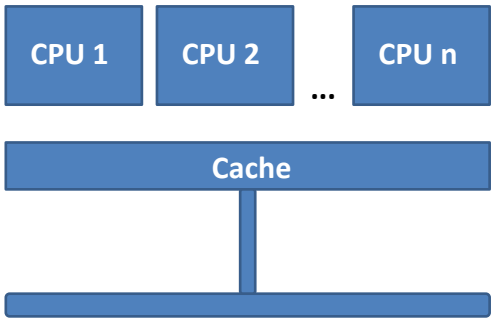


Figura 3.2: *Exemplo de um processador multicore.*

Tais processadores visam minimizar a latência de memória, reservando uma parte do chip para memória *cache*, e permitem um uso moderado de linhas de execução (*threads*) [56], [95]. Entretanto, uma de suas desvantagens encontra-se na largura de banda da memória, a qual pode ser um fator limitante [66]. Esmaeilzadeh *et al.* [27] afirmam que, apesar da organização e topologia do *chip*, o aumento de núcleos em um processador é limitado por um nível de consumo de energia relativamente alto e ainda longe do ideal. Além disso, o barramento também pode ser um fator limitante, pois é o principal gargalo para se aumentar o número de núcleos, e a arquitetura de muitos processadores não foi projetada para suportar múltiplos núcleos de processamento [95].

Por outro lado, os processadores *manycore* são desenvolvidos com dezenas ou centenas de núcleos mais simples, otimizados para uma maior vazão na execução de instruções, utilizando centenas ou até mesmo milhares de *threads* em paralelo [66].

[1] Estabelecida por Gordon Earl Moore, a lei de Moore afirma que o poder de processamento dos computadores dobraria a cada 18 meses.

[2] *Overclocking* é o processo de forçar o processador a executar numa frequência maior do que a especificada pelo fabricante.

Esses processadores são mais comuns em plataformas voltadas para jogos eletrônicos, dispositos embarcados e plataformas computacionais de laboratórios de computação científica [30]. Um caso especial de sucesso é o da evolução das *Graphics Processing Units* (GPU) [72], [79], [77], [76].

3.3 GPU

As GPUs foram inicialmente desenvolvidas como uma tecnologia orientada à vazão, otimizada para cálculos de uso itensivo de dados, onde muitas operações idênticas podem ser realizadas em paralelo sobre diferentes dados [24], [79]. A computação com GPU pode ser considerada como sendo o uso de uma unidade de processamento gráfico como um coprocessador para acelerar as *Central Processing Units* (CPU) para computação científica e de propósito geral [30]. Diferente de uma CPU *multicore*, a qual executa algumas *threads* em paralelo, a GPU foi projetada para executar milhares de *threads* [80], [74]. A Figura 3.3 mostra uma comparação entre a arquitetura de uma CPU com apenas quatro unidades lógica e aritmética (ALU) e uma GPU com 128 ALUs. Na Figura 3.3, as áreas alaranjadas representam a memória (*cache* ou DRAM), a área em amarelo representa a unidade de controle e, em verde, as ALUs.

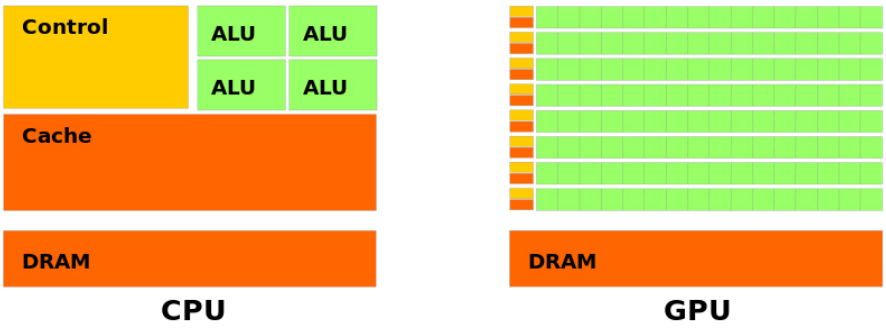


Figura 3.3: *Comparação entre a arquitetura de uma CPU e uma GPU.*

Devido à crescente necessidade de alto poder computacional, a utilização de processadores mais velozes vem se tornando constante [66], [76]. Desde 2003, como mostra a Figura 3.4, as GPUs têm liderado a corrida do desempenho em ponto flutuante.

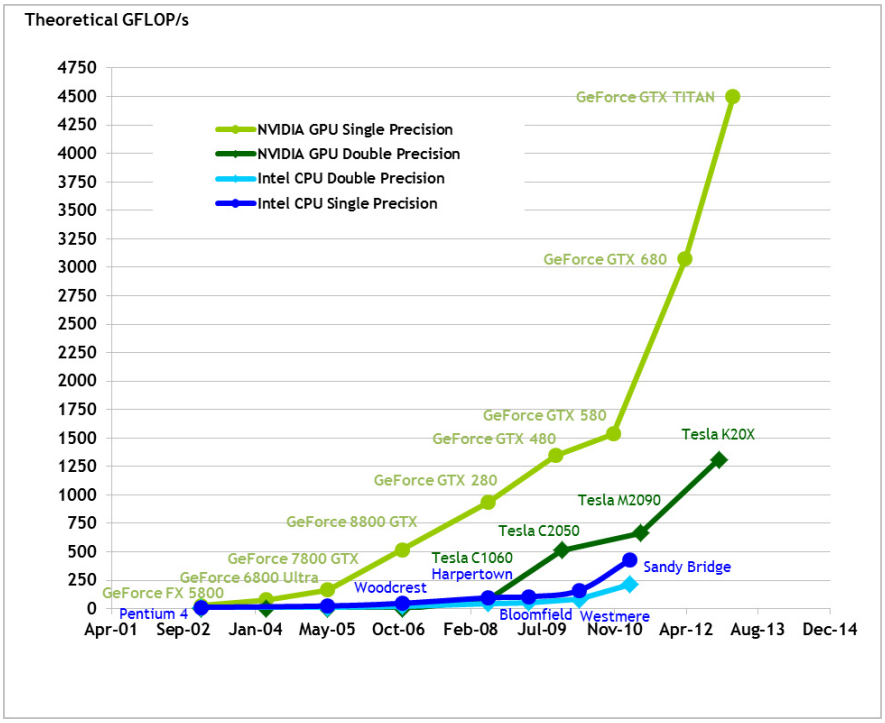


Figura 3.4: *Comparação de capacidades de processamento entre CPUs e GPUs [24].*

As GPUs são melhores adaptadas para endereçar problemas que podem ser expressos através de cálculos realizados de forma paralela [66]. Como o mesmo programa é executado para cada elemento de dado, há menos requisitos referentes a controles de fluxo e, exatamente por ser executado em muitos elementos de dados, a latência de acesso à memória pode ser ocultada pela realização de cálculos. Além do desempenho, as GPUs contam com um importante fator para seu sucesso: a presença de mercado [52]. Ter forte presença de mercado é fundamental para o sucesso de uma arquitetura paralela [30].

Como toda tecnologia, as GPUs possuem suas limitações [66]. Dependendo do volume de dados, o desempenho computacional da GPU pode se mostrar inferior quando comparado ao desempenho da CPU. Isso implica que a quantidade de dados a serem transferidos para a memória da GPU deve ser levado em consideração, devido à existência de um *overhead* associado à paralelização das tarefas na GPU [22], [52]. Fatores em relação ao tempo de acesso em memória também podem influenciar no desempenho computacional [80]. Em outras palavras, o acesso à memória global da GPU geralmente apresenta uma alta latência e pode estar sujeito a um acesso aglutinado aos dados em memória [22].

A *NVIDIA*® e a *AMD*® são exemplos de empresas que desenvolvem GPUs e disputam o mercado de computação paralela. Como mostra a Seção 3.4, modelos específicos para a GPU foram desenvolvidas por essas duas empresas. Modelos de programação como *Compute Unified Device Architecture* (CUDA) e *Open Computing*

Language (OpenCL) permitem que aplicações possam ser executadas mais facilmente na GPU [74].

3.4 Modelos de Programação Paralela em GPU

3.4.1 CUDA

CUDA [24] foi a primeira plataforma e interface para programação de aplicação (API), criada pela *NVIDIA*® em 2006, a permitir que a GPU pudesse ser utilizada para uma ampla variedade de aplicações [52]. CUDA é suportada por todas as placas gráficas da *NVIDIA*®, que são extremamente paralelas devido aos muitos núcleos com diversas memórias *cache* e uma memória compartilhada por todos os núcleos [22].

No ambiente de programação CUDA, o sistema computacional distingue entre o que é executado na CPU (*host*) e o que é executado na GPU (*device*). Um programa em CUDA consiste em partes executadas no *host* e outras partes executadas no *device*. A separação fica a cargo do compilador da *NVIDIA*® (*nvcc*) durante a compilação. O código em CUDA é uma extensão da linguagem computacional C (CUDA-C), onde algumas palavras-chave são utilizadas para rotular as funções paralelas (*kernels*) e suas estruturas de dados [52].

A implementação de uma função a ser executada em paralelo pelas *threads* nos núcleos da GPU é chamada *kernel*. Os *kernels* normalmente geram um grande número de *threads* para explorar o paralelismo de dados. O número de *threads* é especificado pelo programador na invocação da função. Quando um *kernel* é disparado, ele é executado como uma grade (*grid*) de *threads* [24]. Como ilustra a Figura 3.5, as *threads* em um *grid* são organizadas em uma hierarquia de dois níveis, onde cada *grid* consiste em um ou mais blocos de *threads*. Por exemplo, uma matriz $C_{3\times4} = A_{3\times4} + B_{3\times4}$ pode ser obtida pela utilização de um *grid* com um único bloco com doze *threads*. O Algoritmo 3.1 mostra um exemplo de uma função *kernel* que pode somar duas matrizes.

Algoritmo 3.1: Exemplo de uma função *kernel*.

```
__global__ void somaMatrizes(double *A,double *B,double *C,int dim)
{
    //obtem o identificador global da thread
    int id = blockIdx.x * blockDim.x + threadIdx.x;

    if(id < dim)
         C[id] = A[id] + B[id];
}
```

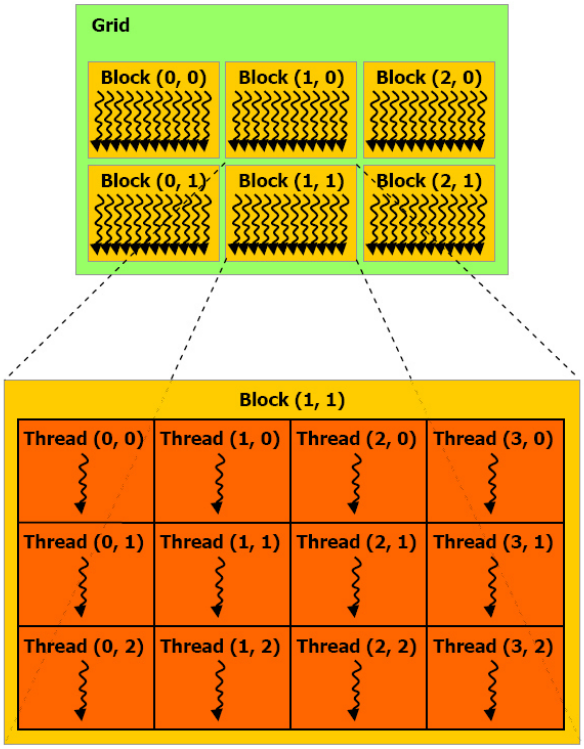


Figura 3.5: *Grid com seis blocos de threads [24].*

Desde o seu surgimento, diversos trabalhos têm utilizado CUDA para a paralelização de vários tipos de problemas. Yldirim e Ozdogan [107] apresentaram um algoritmo como uma abordagem de agrupamento baseado em transformada *wavelet* para paralelização em GPU usando CUDA-C. Atasoy *et al.* [5] apresentaram um método de eliminação implementado em CUDA-C usando o algoritmo de Gauss-Jordan para solução de sistemas de equações lineares. Paula *et al.* [80] utilizaram CUDA-C para paralelizar o método BiCGStab(2), que é um método iterativo utilizado para solução de sistemas lineares. Paula *et al.* [79] propuseram uma estratégia de paralelização para a fase 2 do APS utilizando CUDA-C. Gaioso *et al.* [37], utilizando CUDA-C, apresentaram uma paralelização para o algoritmo Floyd-Warshall, utilizado para encontrar os caminhos mínimos entre todos os pares de vértices de um grafo.

Recentemente, a *MathWorks*® [65] desenvolveu um plugin capaz de fazer a integração entre CUDA e Matlab. Fazer uso do Matlab para computação em GPU pode permitir que aplicações sejam aceleradas mais facilmente [65]. As GPUs podem ser utilizadas com Matlab por meio do *Parallel Computing Toolbox* (PCT). O PCT fornece uma maneira eficiente para acelerar códigos na linguagem Matlab, executando-os em uma GPU. Para isso, o programador deve alterar o tipo de dado para entrada de uma função para utilizar os comandos (funções) do Matlab que foram sobracarregados (*GPUArray*). Por meio da função *GPUArray* é possível alocar dados na memória da GPU e fazer

chamadas a várias funções do Matlab, que são executadas nos núcleos de processamento da GPU. Além disso, os desenvolvedores podem fazer uso da interface *CUDAKernel* no PCT para integrar seus códigos em CUDA-C com o Matlab [63].

O desenvolvimento de aplicações a serem executadas na GPU utilizando o PCT é, geralmente, mais fácil e rápido do que utilizar a linguagem CUDA-C [58]. De acordo com Little e Moler [65], isso ocorre porque vários aspectos de exploração de paralelismo são realizados pelo próprio PCT. Entretanto, a organização e o número de *threads* a serem executadas nos núcleos da GPU não podem ser gerenciados manualmente pelo programador. Ainda, é importante ressaltar que, para poder ser utilizado, o PCT requer uma placa gráfica da *NVIDIA*®.

Após a integração CUDA-Matlab, alguns trabalhos têm utilizado essa tecnologia. Por exemplo, a *NVIDIA*® [21] lançou um livro que demonstra como programas desenvolvidos em Matlab podem ser acelerados usando suas GPUs. Simek e Asn [88] apresentaram uma implementação em Matlab com CUDA para compressão de imagens médicas. Kong *et al.* [55] aceleraram algumas funções do Matlab para processamento de imagens em GPUs. Reese e Zaranek [63] desenvolveram um manual de programação em GPUs usando Matlab. Little e Moler [65] mostraram vários detalhes de como realizar computações do Matlab em GPUs usando CUDA. Mais recentemente, Paula [73] propôs uma paralelização do método BiCGStab(2) para solução de sistemas lineares usando a integração CUDA-Matlab.

3.4.2 OpenCL

OpenCL [100] é um padrão aberto, mantido pelo *Khronos Group*, que permite o uso de GPUs para desenvolvimento de aplicações paralelas. Por meio de sua API, os desenvolvedores podem fazer chamadas à funções *kernels* utilizando um subconjunto limitado da linguagem de programação C em uma GPU [23].

O OpenCL foi projetado para permitir o desenvolvimento de aplicações paralelas que funcionam em plataformas heterogêneas, consistindo em CPUs, GPUs e outros processadores. O modelo de programação em OpenCL é semelhante ao utilizado em CUDA. A Tabela 3.2 mostra uma comparação entre alguns termos utilizados em CUDA e OpenCL.

Um programa em OpenCL consiste em *kernels*, que são executados pelo(s) $device(s)$, e *host*, que gerencia a execução dos *kernels*. Os *kernels* são executados por $workitens$. Os *workitens* são agrupados em *workgroups*. Os *workgroups* são organizados em um *NDRange*. A Figura 3.6 mostra a organização dos *workitens* e *workgroups* em um *NDRange*:

Tabela 3.2: *Comparação entre termos usados em CUDA e OpenCL.*

CUDA	OpenCL
Streaming Multiprocessor (SM)	Compute Unit (CU)
Streaming Processor (SP)	Processing Element (PE)
host	host
device	device
kernel	kernel
thread	workitem
bloco	workgroup
grid	NDRange

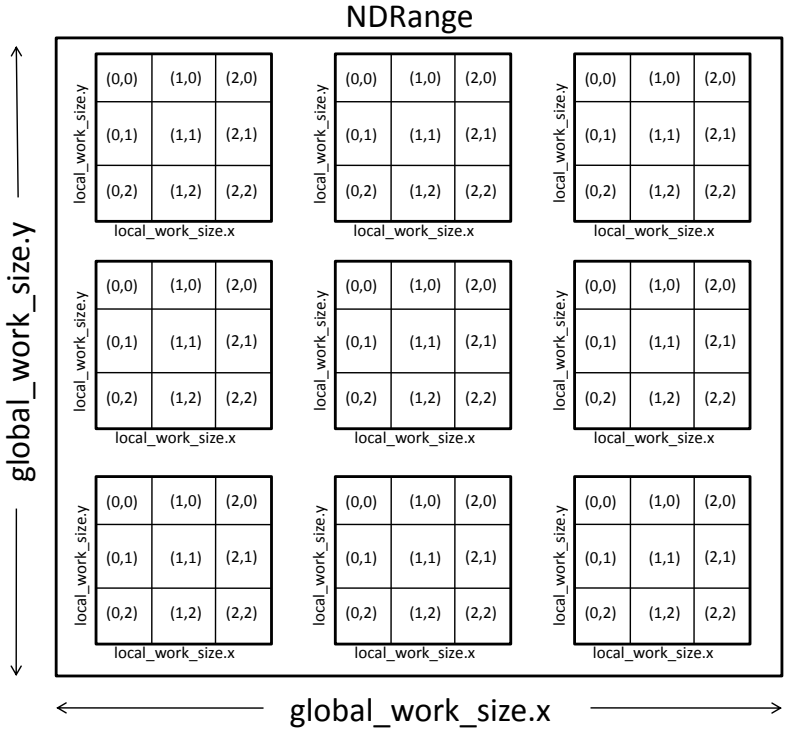


Figura 3.6: *Organização dos workgroups e workitens em um NDRange.*

Após seu surgimento em 2010, alguns trabalhos têm utilizado o OpenCL na tentativa de aumentar o desempenho computacional de problemas paralelizáveis. Entre eles, pode-se citar o trabalho de Komatsu *et al.* [54], que apresentaram uma avaliação de desempenho e portabilidade de programas em OpenCL. Barak *et al.* [7] apresentaram algumas aplicações em OpenCL executadas em *clusters* com muitas GPUs. Jaaskelainen *et al.* [50] descreveram metodologias e técnicas de compilação envolvidas na aplicação de OpenCL como um idioma de entrada para um fluxo de projeto de processadores de aplicações específicas. Grewe *et al.* [44] propuseram um esquema de particionamento portável para programas em OpenCL em sistemas heterogêneos. Recentemente, Suga-

numa *et al.* [96] descreveram suas experiências em programação OpenCL para obter um desempenho escalável para um ambiente de computação heterogênea e distribuída.

A escolha do OpenCL pode parecer uma escolha mais óbvia por ser possível desenvolver programas que poderiam ser executados em qualquer GPU, ao invés de desenvolver uma versão (em CUDA-C) para placas da *NVIDIA*®. No entanto, na prática essa escolha pode ser um pouco mais complicada, já que o OpenCL oferece funções e extensões que são específicas para cada família [53], [72]. Além disso, por ter um modelo de gerenciamento para a portabilidade em multiplataformas e multifornecedores, o OpenCL pode ser considerado um tanto mais complexo [75], [51].

Por outro lado, com milhões de GPUs habilitadas para CUDA já vendidas até hoje, os desenvolvedores de software, cientistas e pesquisadores têm descobrido usos amplamente variados para a computação com GPU utilizando CUDA [24]. Ainda, apesar de poder ser utilizada apenas nas placas gráficas da *NVIDIA*®, CUDA tem sido uma referência e mais utilizada ultimamente [75]. Com efeito, neste trabalho optou-se pelo desenvolvimento dos algoritmos paralelos nesta tecnologia.

CAPÍTULO 4

Implementações Propostas

Neste capítulo, propõe-se uma implementação paralela (usando GPU) para o APS-RLM (Seção 4.1), APS-RS (Seção 4.2) e AF-RLM (Seção 4.3). Para o APS-RLM, apresenta-se uma estratégia para a redução do tempo computacional da fase 2 do algoritmo. Para o APS-RS, propõe-se uma paralelização para a estratégia de regressões sequenciais proposta por Soares *et al.* [91]. E, para o AF-RLM, apresenta-se uma estratégia de paralelização para o cálculo do vetor dos coeficientes de regressão (Equação (2-2)).

4.1 APS-RLM

Como descrito na Seção 2.1.1, o objetivo do APS é selecionar um subconjunto de variáveis com baixa correlação linear que permitam a construção de um modelo de RLM com uma capacidade de predição adequada. Em relação ao custo computacional, por envolver o cálculo de matrizes inversas, a fase 2 representa o maior custo. Sendo assim, esta dissertação apresenta uma estratégia para o cálculo de inversão de matrizes na fase 2 do APS. Mostra-se que calcular a inversa de uma matriz utilizando várias *threads* em uma GPU pode ser computacionalmente mais eficiente do que uma implementação sequencial tradicional.

Uma matriz quadrada $\mathbf{A}$ é chamada invertível (ou inversível) quando existe uma outra matriz $\mathbf{A}^{-1}$ tal que $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ e $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, onde $\mathbf{I}$ é chamada matriz identidade. Calcular a inversa de uma matriz sequencialmente em uma CPU pode exigir um esforço computacional significante. Por exemplo, calcular a inversa de uma matriz grande utilizando o método de Gauss-Jordan[1] pode implicar em um tempo computacional relativamente grande [5]. Logo, fazer uso dos recursos de computação paralela fornecidos por uma GPU usando CUDA-C pode ser mais interessante e viável [80]. Dessa forma, neste trabalho é proposto uma implementação do APS-RLM com paralelização parcial do algo-

[1] O método de Gauss-Jordan possui complexidade de tempo $O(n^3)$ [5].

ritmo para a fase 2, onde a exploração de paralelismo ocorre apenas no cálculo de matrizes inversas.

Sejam $\mathbf{A}_{n\times n}$ e $\mathbf{I}_{n\times n}$ a matriz a ser calculada a inversa e a matriz identidade, respectivamente. Pelo método de Gauss-Jordan, se posicionarmos essas duas matrizes lado a lado, as operações elementares realizadas na matriz $\mathbf{A}_{n\times n}$ também devem ser realizadas na matriz $\mathbf{I}_{n\times n}$. Se $\mathbf{A}_{n\times n}$ é inversível, então a matriz $\mathbf{I}_{n\times n}$ resultante se torna em $\mathbf{A}^{-1}_{n\times n}$ [5]. De forma recursiva, $i = 0, 1, ..., n-1$, onde n é o número de linhas da matriz, é possível executar o método de Gauss-Jordan por meio da utilização de duas funções *kernel* (*kernel*1 e *kernel*2).

No *kernel*1 são utilizados $\sqrt{n}$ blocos[2] com $\sqrt{n}$ *threads* cada, onde cada *thread* acessa um elemento da linha i e o divide pelo pivô da linha i da matriz $\mathbf{A}_{n\times n}$. Por exemplo, se $n = 16$, o algoritmo utiliza 4 blocos com 4 *threads* em cada iteração. Dessa forma, existe um melhor equilíbrio entre o número de blocos e o número de *threads* por bloco, realizando uma utilização mais eficiente dos *Streaming Multiprocessors* (SM) da GPU [79], [37]. Ao final da execução do *kernel*1, o pivô da linha i da matriz $\mathbf{A}_{n\times n}$ é igual a 1 e, logo após, inicia-se a execução do *kernel*2.

No *kernel*2 são utilizados n blocos com n *threads* cada. Cada bloco de *threads* realiza operações sobre uma linha das matrizes. Apenas as *threads* cujo seu identificador global (*id*) dividido pelo número de colunas ($\frac{id}{n}$) é diferente da linha i são permitidas a continuar sua execução e realizam operações nos elementos das matrizes. Por exemplo, na primeira iteração ($i = 0$), as *threads* com $id = 0, 1, ..., n-1$ não satisfazem a condição $\frac{id}{n} \neq i$. Ao final da execução, os elementos abaixo e/ou acima do pivô da linha i da matriz $\mathbf{A}_{n\times n}$ são nulos.

A Figura 4.1 mostra a estratégia utilizada. Cada seta na figura representa uma iteração do algoritmo. Todas as operações realizadas nos elementos da matriz $\mathbf{A}_{n\times n}$ também são realizadas nos elementos da matriz $\mathbf{I}_{n\times n}$ pelas mesmas *threads*. Após a última iteração, a matriz $\mathbf{A}_{n\times n}$ se transforma na matriz identidade e a matriz $\mathbf{I}$ se transforma em $\mathbf{A}^{-1}_{n\times n}$. Os Algoritmos 4.1 e 4.2 mostram, respectivamente, a implementação das funções *kernel*1 e *kernel*2.

$$A = \begin{vmatrix} 5 & 6 & 9 \\ 4 & 1 & 7 \\ 1 & 7 & 6 \end{vmatrix} \Rightarrow \begin{vmatrix} 1 & 1.2 & 1.8 \\ 0 & -3.8 & -0.2 \\ 0 & 5.8 & 4.2 \end{vmatrix} \Rightarrow \begin{vmatrix} 1 & 0 & 1.73 \\ 0 & 1 & 0.05 \\ 0 & 0 & 3.89 \end{vmatrix} \Rightarrow \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$I = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \Rightarrow \begin{vmatrix} 0.2 & 0 & 0 \\ -0.8 & 1 & 0 \\ -0.2 & 0 & 1 \end{vmatrix} \Rightarrow \begin{vmatrix} -0.05 & 0.31 & 0 \\ 0.21 & -0.26 & 0 \\ -1.42 & 1.52 & 1 \end{vmatrix} \Rightarrow \begin{vmatrix} 0.58 & -0.36 & -0.44 \\ 0.22 & -0.28 & -0.01 \\ -0.36 & 0.39 & 0.25 \end{vmatrix}$$

Figura 4.1: *Estratégia de paralelização usada no cálculo da inversa de uma matriz 3×3.*

[2]Conforme detalhado no Capítulo 3, um bloco (*block*) é formado por um conjunto de *threads*.

Algoritmo 4.1: Implementação *kernel*1.

1. **Parâmetros**: A, I, i, $size$ = números de colunas.
2. $id \leftarrow$ identificador global da $thread$
3. **se** $id < size$
4. $\quad pivo \leftarrow A(i, i)$
5. $\quad A(i, \text{id}) \leftarrow \frac{A(i,id)}{pivo}$
6. $\quad I(i, \text{id}) \leftarrow \frac{I(i,id)}{pivo}$
7. **fim se**

Algoritmo 4.2: Implementação *kernel*2.

1. **Parâmetros**: A, I, i, n = número de colunas, $size$ = número de linhas $\times$ número de colunas.
2. $id \leftarrow$ identificador global da $thread$
3. $idBlock \leftarrow$ identificador do bloco
4. $idThread \leftarrow$ identificador local da $thread$
5. **se** $id < size$
6. $\quad$ **se** $(\frac{id}{n}) \neq indice$
7. $\quad\quad m \leftarrow A(\frac{id}{n}, i)$
8. $\quad\quad A(idBlock, idThread) \leftarrow A(idBlock, idThread)$ - $(m \times A(i, idThread))$
9. $\quad\quad I(idBlock, idThread) \leftarrow I(idBlock, idThread)$ - $(m \times I(i, idThread))$
10. $\quad$ **fim se**
11. **fim se**

4.2 APS-RS

Regressões sequenciais é uma estratégia empregada na fase 2 do APS. Tal estratégia objetiva reduzir o tempo computacional do algoritmo evitando o cálculo de matrizes inversas. Isso é possível por meio da utilização das Equações (2-11), (2-12), ..., (2-23). Soares *et al.* [91] apresentaram uma implementação em Matlab que aumenta a eficiência computacional da fase 2 utilizando um procedimento de regressões sequenciais. O ganho computacional obtido foi demonstrado por meio de um exemplo que envolve um grande conjunto de dados de amostras de trigo.

Esta dissertação faz uso da implementação proposta por Soares *et al.* [91] e apresenta uma outra implementação: APS-RS-CUDA. A implementação APS-RS-CUDA é parcialmente paralelizada e é invocada como uma sub-rotina pelo próprio código em Matlab. Isso ocorre devido a utilização de um arquivo *MEX* (*MEX-File*), que permite a invocação de funções implementadas em C/C++/CUDA-C a partir de linhas de comando como se fossem funções padrão do Matlab [64].

Como mostra o Algoritmo 4.3, o código em Matlab inicia-se executando a fase 1 do APS. Os parâmetros do algoritmo representam, respectivamente, a matriz do conjunto de calibração (respostas instrumentais), validação e predição (utilizado para comparar a capacidade preditiva do modelo), vetor das variáveis dependentes do conjunto de calibração (concentrações), validação e predição, número mínimo (normalmente igual a 1) e número máximo (normalmente igual a N_{cal}) de variáveis a serem selecionadas. Logo após a execução da fase 1, a fase 2 pode ser executada pelo próprio código em Matlab ou pelo APS-RS-CUDA.

Algoritmo 4.3: Implementação APS-RS.

1. **Parâmetros**: $Xcal_{N_{cal}\times K}$, $Xval_{N_{val}\times K}$, $Xpred_{N_{pred}\times K}$, $Ycal_{N_{cal}\times 1}$, $Yval_{N_{cal}\times 1}$, $Ypred_{N_{pred}\times 1}$, $N1$, $N2$.
2. Executa a fase 1:
   Centralização na média e auto-escalonamento das colunas de $Xcal$ {para fins de geração das cadeias de variáveis}
   Geração das cadeias de variáveis, contendo no mínimo $N1$ e no máximo $N2$ variáveis, a partir de cada coluna de $Xcal$
3. Executa a fase 2 utilizando o próprio código em Matlab ou o Algoritmo 4.4
4. Executa a fase 3
5. Gera o gráfico das variáveis selecionadas
6. Calcula o erro de predição

Depois da execução da fase 1, a fase 2 pode ser executada por qualquer uma das duas versões: APS-RS-MATLAB ou APS-RS-CUDA. O Algoritmo 4.4 mostra um pseudocódigo para a implementação APS-RS-CUDA. Antes de iniciar a fase 2, todos os dados são transferidos para a memória do *device*, onde o paralelismo de dados é explorado pela execução das *threads*.

O trecho de código entre as linhas 4 a 8 do Algoritmo 4.4 foram paralelizados utilizando funções *kernel* na linguagem CUDA-C [24]. Em cada invocação de um *kernel*, o número de blocos e o número de *threads* por bloco são iguais a $\sqrt{n}$, onde n é o número de linhas da matriz ou vetor em que se realizará as operações. Se $\sqrt{n} \notin \mathbb{N}$, obtém-se o $\lceil\sqrt{n}\rceil$ (teto). Essa estratégia de implementação pode explorar mais eficientemente os núcleos de processamento da GPU, pois evita a utilização de um único bloco com várias *threads* ou vários blocos com uma única *thread* [79], [77], [78].

Na solução do problema proposto, são utilizadas cinco funções *kernel*, não sendo necessário sincronizar as *threads* para obter um desempenho computacional melhor do que o apresentado por implementações tradicionais. O paralelismo é explorado apenas nas operações de adição, subtração e multiplicação matricial. Ou seja, adição e subtração de vetores, multiplicação de matriz por vetor e multiplicação de vetor por escalar. Explora-se também o paralelismo para a cópia de elementos entre matrizes e vetores. A ideia da

implementação paralela destas operações pode ser ilustrada por meio da operação vetorial $\mathbf{z} = \mathbf{v}_N + \mathbf{w}_N$. Nesse caso, a soma dos vetores utiliza *N threads*, onde cada uma executa $\mathbf{z}_i = \mathbf{v}_i + \mathbf{w}_i$ (i = 1, ..., N) da operação vetorial.

Vale ressaltar que os produtos escalares e as operações de soma de vetores são realizadas (sequencialmente) no *host*. Para isso, os dados são transferidos da memória do *device* para a memória do *host* e, logo após, são replicados de volta para a memória do *device*. Poderia ser possível aprimorar o paralelismo dividindo os produtos entre múltiplas somas feitas em paralelo [10]. Entretanto, no contexto deste trabalho, isso implicou em uma alta sobrecarga (*overhead*) e resultou em um ganho de desempenho insignificante. Embora o guia de melhores práticas de programação CUDA [22] sugere que deve-se evitar uma constante transferência de dados entre as memórias do *host* e do *device*, a estratégia utilizada apresentou bons resultados em relação à implementação sequencial (ver Capítulo 6).

Algoritmo 4.4: Implementação APS-RS-CUDA.

1. $L \leftarrow$ matriz onde cada coluna contém os índices das variáveis selecionadas na Fase 1
2. Transferência dos dados da memória do *host* para a memória do *device*
3. **para** $i = 1$ **até** K
4. $\quad lambdas \leftarrow L_i$ {cada *thread* copia um elemento da coluna i da matriz L}
5. $\quad x \leftarrow Xcal_{lambdas}$ {cada *thread* copia um elemento de cada coluna de $Xcal$}
6. $\quad$ Calcula $\beta_1^{(1)}$ utilizando a Equação (2-12)
7. $\quad$ Calcula $\beta_1^{(2)}$ e $\beta_2^{(2)}$ utilizando as Equações (2-14), (2-15), (2-16), (2-17) e (2-18)
8. $\quad$ Calcula $\beta_1^{(m)}, \ldots, \beta_m^{(m)}$, $m = 1, 2, \ldots, K$, utilizando as Equações (2-20), (2-21), (2-22) e (2-23)
9. **fim para**
10. Transferência dos dados da memória do *device* para a memória do *host*
11. Seta os parâmetros de saída

4.3 AF-RLM

Bioinspirado no comportamento de vagalumes, o AF pode ser utilizado para a resolução de problemas de otimização. Apesar de vários trabalhos já terem o utilizado para solucionar diversos tipos de problemas, ainda não existem na literatura outros trabalhos que utilizam o AF para a seleção de variáveis em problemas de calibração multivariada. Portanto, apresenta-se uma implementação do AF (AF-RLM), a qual é uma adaptação do Algoritmo 2.3, para a seleção de variáveis usando modelos de RLM.

O AF-RLM utiliza uma estratégia para a paralelização do cálculo do vetor dos coeficientes de regressão (re-equacionamento da Equação (2-2)). Inicialmente, define-se o

número (s) de vagalumes a serem utilizados e a quantidade de iterações a serem realizadas ($MaxGen$). Em cada iteração, uma população de s vagalumes é gerada aleatoriamente, os coeficientes de regressão e o erro do modelo são calculados s vezes, e os vagalumes são ordenados de acordo com suas intensidades de luz. O Algoritmo 4.5 mostra um pseudocódigo para o AF-RLM.

Algoritmo 4.5: Implementação AF-RLM.

1. **Parâmetros**: $\mathbf{X}_{n\times m}$, $\mathbf{y}_{n\times 1}$, onde $\mathbf{X}_{n\times m}$ é a matriz de amostras por seus valores de variáveis correspondentes, n é o número de linhas, m é o número de colunas e $\mathbf{y}_{n\times 1}$ é o vetor das variáveis dependentes.
2. $s \leftarrow$ número de vagalumes
3. **para** $i = 1$ **até** $MaxGen$
4. Gere aleatoriamente uma população de s vagalumes, onde cada vagalume é um conjunto de d índices para d colunas de $\mathbf{X}_{n\times m}$, $d \leq m$
5. Calcule a Equação (4-1) para cada vagalume
6. Calcule o erro determinando a intensidade de luz de cada vagalume
7. Ordene os vagalumes e encontre o melhor
8. **fim para**
9. Mostre os resultados (RMSEP)

Como descrito no Capítulo 3, as GPUs atualmente disponíveis representam um *hardware* de alto desempenho computacional, com programação flexível e facilitada por meio da utilização de diversas APIs [79], [37]. Ainda, com a integração CUDA-Matlab é possível utilizar funções do Matlab que foram sobrecarregadas para serem executadas na GPU. Tais funções exploram paralelismo nos núcleos da GPU implicitamente. Logo, embora a organização e o número de *threads* não possam ser especificados pelo programador, isso pode fornecer uma maneira eficiente para acelerar códigos em Matlab [88], [55], [21]. Sendo assim, o passo 5 do Algoritmo 4.5 pode ser executado em uma GPU. Nesse caso, a execução paralela é realizada por funções padrão do Matlab [65].

Como discutido anteriormente, calcular a inversa de uma matriz pode elevar muito o tempo computacional, principalmente quando a matriz é grande. Portanto, ao invés de utilizar a Equação (2-2), a estratégia aqui proposta realiza um re-equacionamento dessa equação calculando um sistema linear por meio do comando "\" (*backslash*) da linguagem Matlab, o qual executa uma função otimizada. Em outras palavras, tal função soluciona o sistema:

$$\mathbf{Ax} = \mathbf{b}, \tag{4-1}$$

onde $\mathbf{A} = (\mathbf{X}^T\mathbf{X})$, $\mathbf{b} = \mathbf{X}^T\mathbf{y}$ e $\mathbf{x} = \beta$.

Com a utilização dessa estratégia, evita-se o cálculo da matriz inversa contida na Equação (2-2). O Algoritmo 4.6 mostra um pseudocódigo para o passo 5 do Algoritmo 4.5.

Algoritmo 4.6: Passo 5 do Algoritmo 4.5.

1. **Parâmetros**: $\mathbf{X}_{n\times m}$, $\mathbf{y}_{n\times 1}$ e os vagalumes $\mathbf{x}_1$, ..., $\mathbf{x}_s$.
2. **para** $i = 1$ **até** s
3. Obtenha a submatriz $\mathbf{X}_{n\times d}$, que contém apenas d colunas de $\mathbf{X}_{n\times m}$ indexadas por $\mathbf{x}_i$
4. Aloque as matrizes $\mathbf{X}_{n\times d}$ e $\mathbf{X}^T_{n\times d}$ e o vetor $\mathbf{y}_{n\times 1}$ na memória da GPU, utilizando a função padrão $gpuArray$
5. Calcule a Equação (4-1) na GPU
6. **fim para**

CAPÍTULO 5

Material e Métodos

5.1 Conjunto de Dados

O conjunto de dados empregado neste trabalho consiste em amostras integrais de trigo, obtidas a partir de material vegetal de produtores canadenses. Assim como nos trabalhos de Galvão Filho *et al.* [33], Soares *et al.* [91] e Paula *et al.* [79], os dados padrão foram determinados no laboratório de pesquisa de grãos (*Grain Research Laboratory*). O conjunto de dados para a calibração multivariada consiste em 1090 espectros de reflectância no infravermelho próximo (NIR) de amostras inteiras de grãos de trigo, os quais foram utilizados como dados referenciais na conferência internacional de reflectância difusa, em 2008 [25].

O teor de proteína foi escolhido como propriedade de interesse. Os espectros foram adquiridos por um espectrofotômetro na faixa de 400-2500 *nanometers* (nm), com uma resolução de 2 nm. Neste trabalho, o NIR empregado foi na faixa de 1100-2500 nm. A fim de remover características indesejáveis, o primeiro derivado de espectros foi calculado utilizando o filtro de Savitzky-Golay com um polinômio de segunda ordem e uma janela de onze pontos [79], [92], [91], [90], [33].

Os valores de referência da concentração de proteína nas amostras de trigo foram determinados em laboratório pelo método Kjeldahl [12]. Esse método utiliza a destruição de substâncias orgânicas com ácido sulfúrico concentrado na presença de um catalisador e pela ação do calor, com a subsequente destilação de nitrogênio a partir da amostra. A utilização de métodos instrumentais indiretos, tal como o NIR, e modelos matemáticos como RLM permitem que o nível de proteína seja determinado sem destruir a amostra.

5.2 Pré-tratamento dos Dados

Os espectros de reflectância no NIR podem ser complexos e normalmente ocorrem distorções espectrais causadas pelo espectrofotômetro como, por exemplo, os efeitos causados pela saturação do detector e falhas no varrimento dos comprimentos de onda [3].

Essas perturbações podem gerar nos espectros NIR variações aditivas e multiplicativas. Com isso, torna-se necessário a utilização de um pré-tratamento dos dados, que tem como objetivo remover variações sistemáticas e as informações irrelevantes, evidenciando os parâmetros de interesse e aumentando a seletividade [70]. O pré-tratamento permite que o modelo de calibração não seja afetado por informações irrelevantes e permite uma maior linearidade dos dados [3]. Neste trabalho, para o pré-tratamento dos dados, foi utilizado a centralização na média (ver Seção 5.2.1).

5.2.1 Centralização na média

A centralização na média baseia-se na simples subtração da absorbância em cada comprimento de onda pela absorbância média a esse mesmo comprimento de onda para o conjunto de amostras. Dessa forma, o novo valor de absorbância média para todas as amostras é igual a zero em cada comprimento de onda. Do ponto de vista estatístico, a centralização tem como objetivo prevenir que os pontos mais distantes do centro dos dados tenham maior influência que os mais próximos, dando importância à sua distância do valor médio [3].

5.3 Plataforma Computacional

Todos os testes foram realizados em um computador *desktop* com um processador *Intel core* i7 2600 (3,4 GHz), 8 GB de memória RAM e uma placa gráfica *NVIDIA*® *GeForce* GTX 550Ti com 192 CUDA *cores* e 2 GB de memória configurada. O software *Matlab*® R2013a (8.1.0.604) foi utilizado tanto na geração das matrizes aleatórias, utilizadas para testes com o APS-RLM, quanto na execução dos testes com o APS-RS e AF-RLM.

CAPÍTULO 6

Resultados

6.1 Resultados para o APS-RLM

A Figura 6.1 apresenta o tempo gasto no cálculo de inversão de matrizes envolvido na fase 2, dependendo do número máximo M de variáveis selecionadas. Por exemplo, para $M = 100$, a inversão de matrizes de ordem 100×100 é realizada. Como pode ser observado, o tempo computacional aumenta com o tamanho da matriz, porém o aumento é menos acentuado se a estratégia proposta for utilizada. Para $M = 1000$, a estratégia utilizada no cálculo de inversão de matrizes realizado nos núcleos da GPU é duas vezes mais rápida do que na CPU.

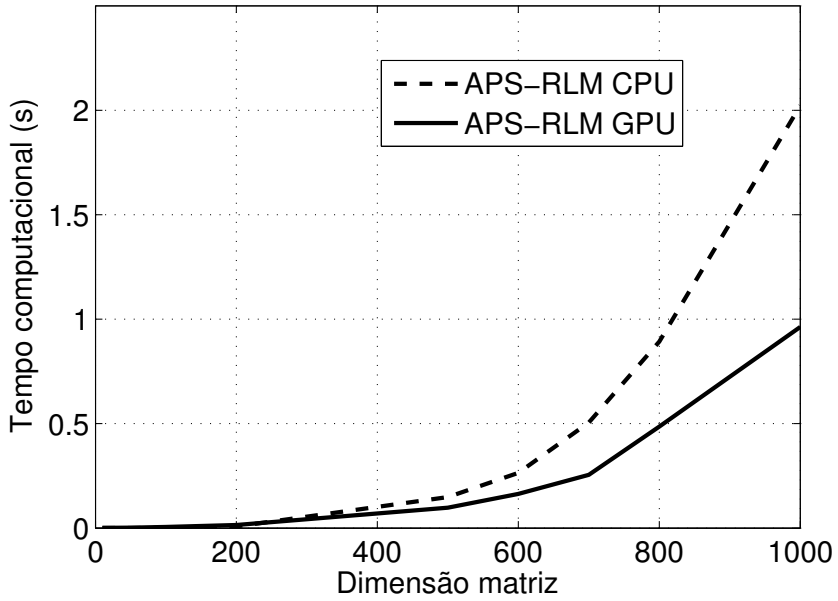


Figura 6.1: *APS-RLM: comparação de desempenho computacional entre CPU e GPU no cálculo de matrizes inversas [79].*

Como mostra a Figura 6.2, a implementação que usa a GPU é menos eficiente para matrizes de tamanho menor que aproximadamente 250×250. Como afirmado na se-

ção 3.3, isso pode ocorrer devido à existência de um *overhead* associado à paralelização das tarefas na GPU [52], [53], [79], [80], [74]. Entretanto, observa-se uma aceleração significativa nos cálculos com a implementação que usa a GPU, que se acentua consideravelmente na medida em que o tamanho das matrizes aumenta. Ainda, apesar desta limitação, acredita-se que essa estratégia de paralelização é importante, pois os dispositivos utilizados na coleta de dados para este tipo de problema têm gerado um número cada vez maior de variáveis. [92]. Até cinco anos atrás, as aplicações geravam matrizes com apenas algumas centenas de variáveis, enquanto que as atuais geram milhares [91]. Em consequência disso, o desenvolvimento de algoritmos computacionais eficientes e a exploração adequada de paralelismo utilizados nesse tipo de problema se tornam cada vez mais importantes para que as predições sejam realizadas com qualidade e em um espaço curto de tempo [79].

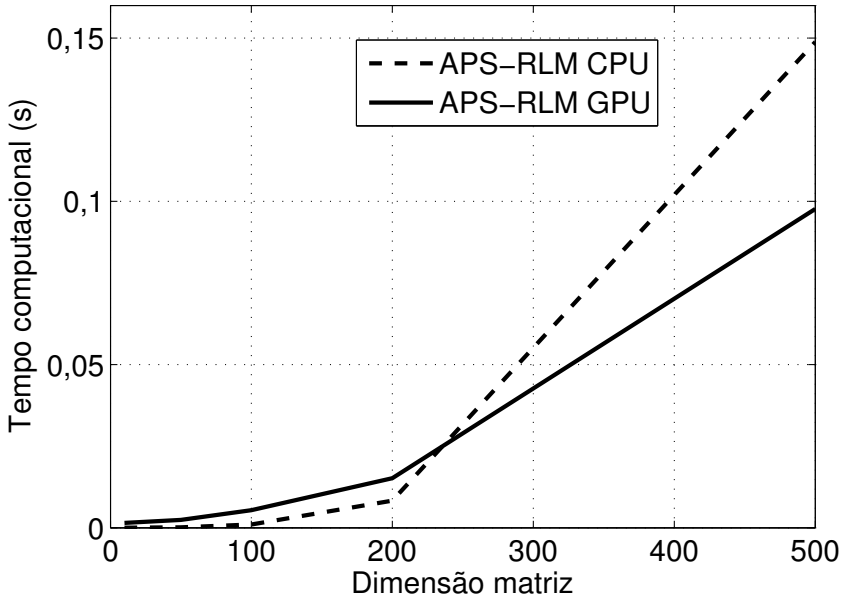


Figura 6.2: *Detalhe da Figura 6.1 mostrando uma comparação de desempenho computacional entre CPU e GPU para matrizes de tamanho até 500×500 [79].*

É importante ressaltar que a estratégia de paralelização aqui proposta não foi comparada com outros algoritmos consolidados na literatura. O objetivo foi apenas mostrar que tal estratégia pode ser viável para problemas que exigem o cálculo de matrizes inversas, como é o caso de seleção de variáveis por modelos de regressão linear múltipla.

6.2 Resultados para o APS-RS

A Figura 6.3 apresenta o tempo gasto para a execução da fase 2 utilizando as implementações APS-RS e APS-RS-CUDA, onde $N2$ representa o número máximo

de variáveis selecionadas. Por exemplo, para $N2 = 100$, foram realizadas regressões envolvendo de 1 até 100 variáveis. Como pode ser observado, o tempo computacional aumenta de acordo com $N2$, porém o aumento é menos acentuado para o APS-RS-CUDA. É possível observar que a implementação APS-RS-CUDA se mostrou, em média, 3x mais rápida que o APS-RS. A Tabela 6.1 apresenta uma comparação de tempo computacional entre as implementações.

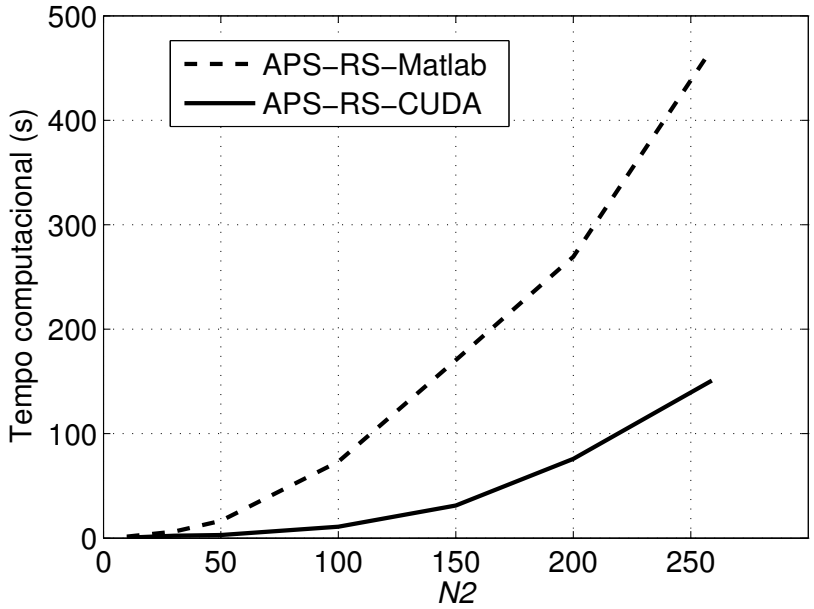


Figura 6.3: *Comparação de desempenho computacional entre APS-RS e APS-RS-CUDA.*

Tabela 6.1: *Tempo computacional (em segundos) para APS-RS-Matlab, APS-RS-CUDA e Soares et al. [91].*

	Número de variáveis		
	100	200	250
APS-RS-Matlab	73,13	269,33	468,42
APS-RS-CUDA	19,88	89,80	170,87
Soares [91]	110	400	700

Em comparação com os tempos computacionais obtidos por Soares *et al.* [91], observou-se que as implementações são mais eficientes que a implementação lá realizada. Por exemplo, para $N2 = 250$, a implementação apresentada em [91] e o APS-RS-CUDA executaram em torno de 700 e 171 segundos, respectivamente. Foi possível observar que o APS-RS-CUDA foi, em média, 4x mais rápido.

6.3 Resultados para o AF-RLM

A Figura 6.4 mostra que o AF-RLM é capaz de reduzir o RMSEP conforme as iterações são realizadas. A curva no gráfico se refere ao erro médio de todos os vagalumes utilizados. A Figura 6.5 mostra como o número de vagalumes afeta o RMSEP. É possível verificar que é necessário um número entre 400 e 500 vagalumes para se alcançar os melhores resultados.

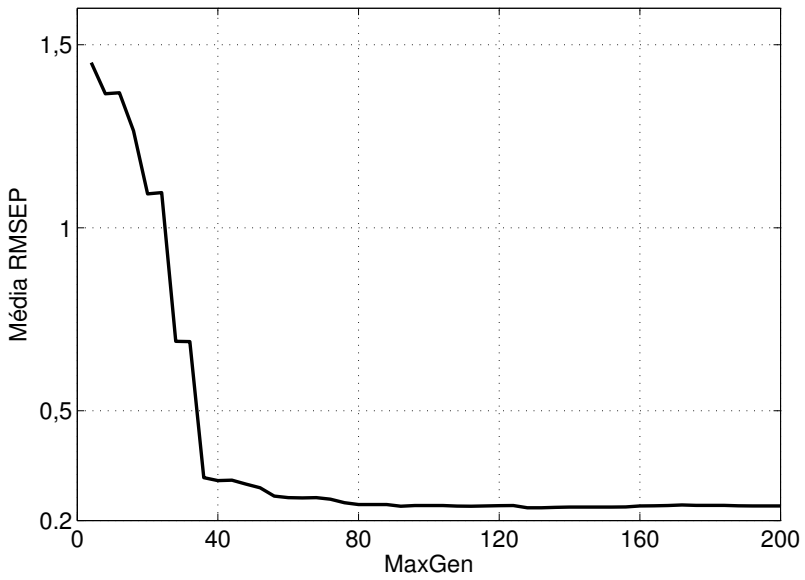


Figura 6.4: *Comportamento entre a média do RMSEP e MaxGen.*

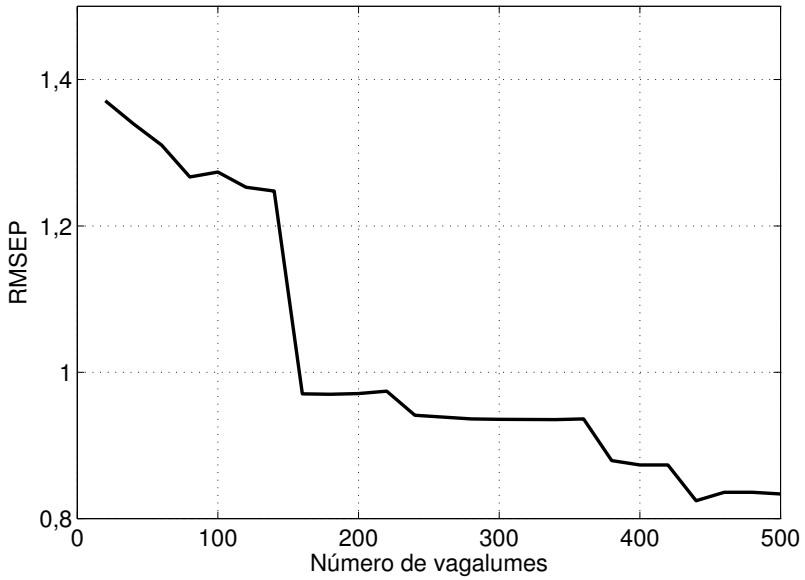


Figura 6.5: *Comportamento entre o RMSEP e o número de vagalumes.*

As variáveis selecionadas utilizando o melhor vagalume obtido podem ser visualizadas na Figura 6.6. Este resultado no gráfico indica que essas regiões são as mais promissoras para serem utilizadas no espectrofotômetro. Na prática, tal resultado implica em um número pequeno de comprimentos de ondas no espectrofotômetro para quantificar a propriedade de concentração de proteína em amostras reais de trigo [77].

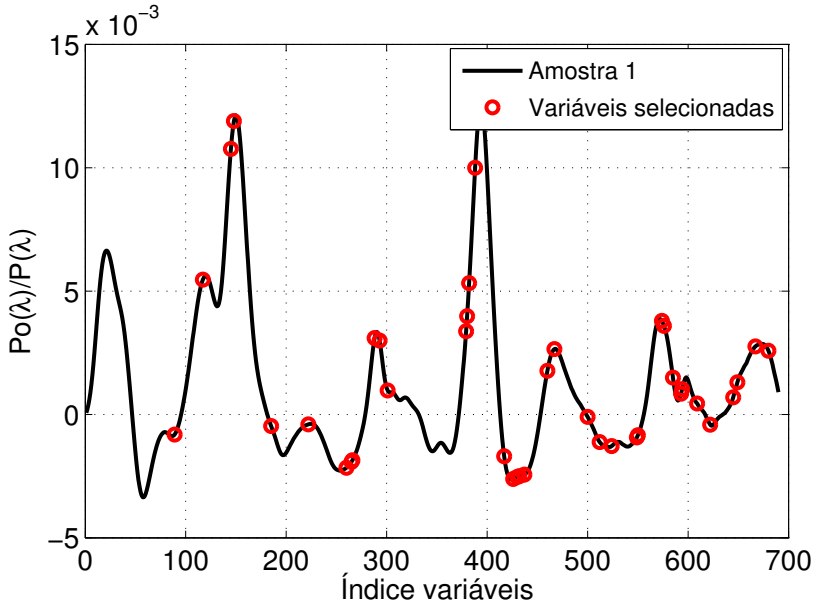


Figura 6.6: *Visualização de variáveis selecionadas.*

6.3.1 Comparação com o APS-RLM e APS-RS

Uma comparação entre o AF-RLM, APS-RLM e APS-RS é apresentada na Tabela 6.2. Como pode ser observado, o APS-RLM e APS-RS selecionam o menor número de variáveis. Entretanto, o AF-RLM apresenta os menores valores de erro (RMSEP, MAPE e PRESS)[1]. Por outro lado, devido o AF-RLM selecionar um número maior de variáveis, o algoritmo perde quando observadas as métricas AIC e BIC. Deve-se notar que o APS-RLM e APS-RS fornecem os mesmos resultados, existindo uma diferença apenas no tempo computacional.

Tabela 6.2: *Resultados do AF-RLM, APS-RLM e APS-RS.*

	RMSEP	MAPE	PRESS	AIC	BIC	Número de Variáveis
APS-RLM	0,20	1,43%	9,95	45,54	120,58	13
APS-RS	0,20	1,43%	9,95	45,54	120,58	13
AF-RLM	0,07	0,5%	3,31	57,90	152,51	39

[1] Vale ressaltar que alguns cientistas podem preferir um número reduzido de variáveis. Nesse caso, o APS-RLM ou APS-RS seria uma escolha mais viável.

A Figura 6.7 apresenta os valores reais versus as predições, utilizando o AF-RLM (círculos azuis), APS-RLM e APS-RS (cruzes vermelhas). Como o APS-RLM e o APS-RS selecionam as mesmas variáveis, ambos são representados pelo mesmo símbolo. As diferenças entre as predições e as concentrações atuais resultam em pontos sobre a reta do gráfico. As concentrações preditas estão próximas das concentrações reais para ambos os métodos. No entanto, o modelo que utiliza as variáveis selecionadas pelo AF-RLM fica mais próximo da reta do que as predições do APS-RLM e APS-RS. Consequentemente, esse resultado indica que o modelo de RLM obtido utilizando as variáveis selecionadas pelo AF-RLM pode produzir o menor erro de predição.

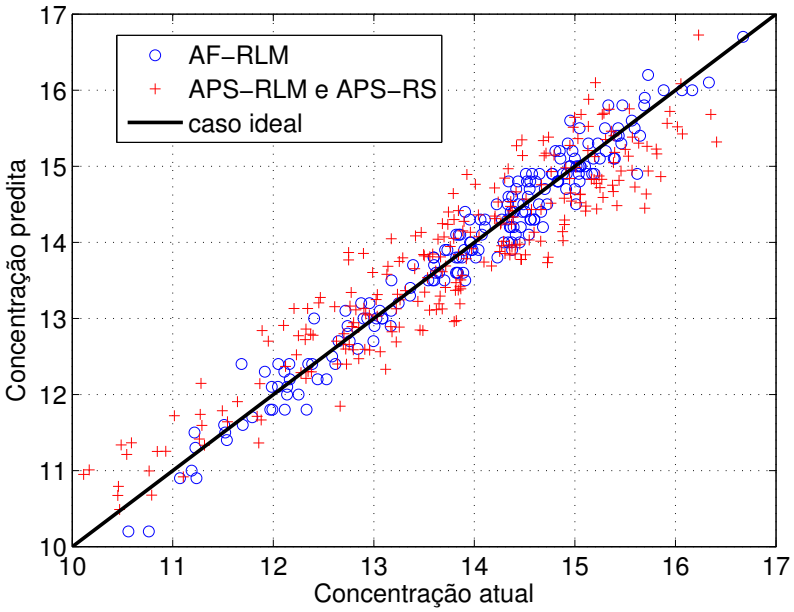
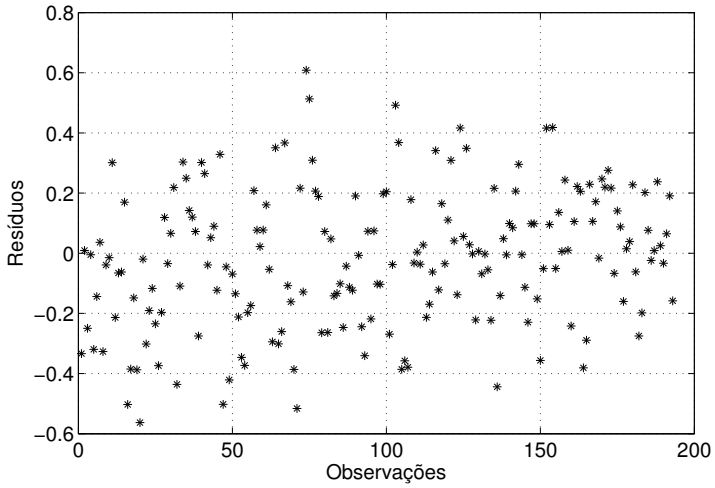
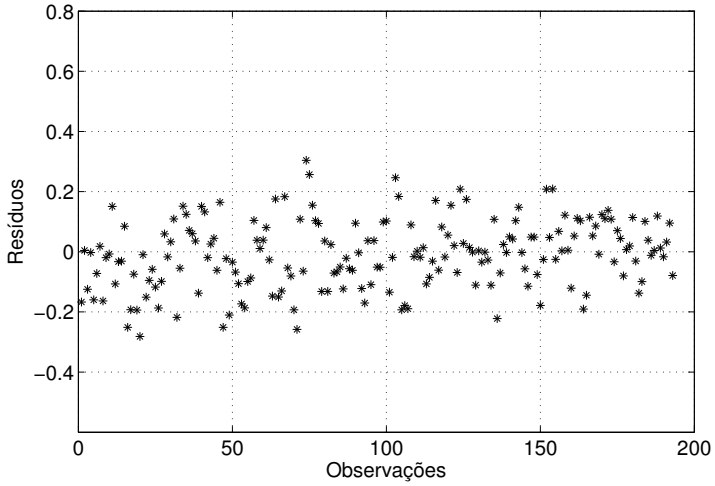


Figura 6.7: *Comparação entre a concentração de proteína predita e atual utilizando AF-MLR, APS-MLR e APS-RS.*

A Figura 6.8 mostra os valores PRESS para todos os algoritmos. Para calcular o PRESS, utilizou-se apenas a diferença entre o valor real (y) e predito ($\hat{y}$) da propriedade de interesse. Ao utilizar apenas a diferença entre y e $\hat{y}$, pode-se obter um valor residual mais exato [77]. É possível observar a partir dos gráficos que os erros estão aleatoriamente distribuídos para ambos os algoritmos, o que indica que ambos possuem baixa sensibilidade à presença de *outliers*.



(a) *APS-RLM e APS-RS.*



(b) *AF-RLM.*

Figura 6.8: *Valores PRESS para os algoritmos: (a) APS-RLM e APS-RS; e (b) AF-RLM.*

6.3.2 Desempenho computacional obtido com o AF-RLM

A Figura 6.9 mostra o desempenho computacional do AF-RLM executado na CPU e também na GPU. A Tabela 6.3 apresenta a comparação de tempo computacional para o AF-RLM de acordo com o número de vagalumes utilizados. Os resultados mostraram que o AF-RLM paralelizado é aproximadamente 5x mais rápido que o AF-RLM sequencial.

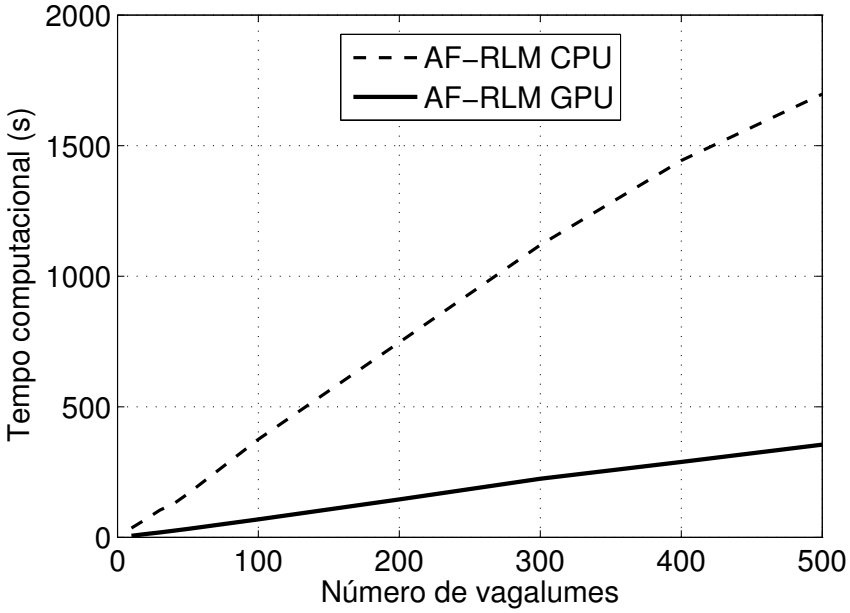


Figura 6.9: *AF-RLM: comparação de desempenho computacional entre CPU e GPU.*

Tabela 6.3: *Tempo computacional (em segundos) para o AF-RLM.*

	Número de vagalumes		
	100	300	500
AF-RLM CPU	375,01	1119,10	1697,01
AF-RLM GPU	68,54	224,18	354,76

Os tempos computacionais na Tabela 6.4 representam o tempo gasto na seleção de até 250 variáveis para o APS-RLM e APS-RS, e usando 250 vagalumes para o AF-RLM.

Tabela 6.4: *Tempo computacional (em segundos) para o APS-RLM, APS-RS e AF-RLM.*

	Tempo
APS-RLM CPU	533,66
APS-RLM GPU	417,23
APS-RS-Matlab	468,42
APS-RS-CUDA	170,87
AF-RLM CPU	932,08
AF-RLM GPU	184,56

Como pode ser observado, os tempos gastos pelo APS-RS-CUDA e o AF-RLM executado na GPU são os menores em comparação com os outros algoritmos. Embora o

tempo do APS-RS-CUDA seja menor, seus resultados indicam um desempenho relativamente equivalente e, devido a estratégia utilizada no cálculo do vetor dos coeficientes de regressão, espera-se que o desempenho computacional do AF-RLM executado na GPU poderá se mostrar superior em problemas com um número maior de variáveis.

CAPÍTULO 7

Conclusões

Apesar de diversos trabalhos já terem apresentado bons resultados para problemas de calibração multivariada, a exploração de paralelismo na tentativa de aumentar o desempenho computacional ainda tem sido pouco investigada. Nesse contexto, este trabalho propôs três novas estratégias de paralelização de algoritmos para seleção de variáveis em problemas de calibração multivariada: APS-RLM, APS-RS e AF-RLM.

Para o APS-RLM, foi apresentada uma estratégia de paralelização para o cálculo de matrizes inversas envolvido na fase 2 do algoritmo. Para o APS-RS, apresentou-se uma proposta de paralelização para ser utilizada na estratégia de regressões sequenciais, que também visa reduzir o tempo computacional da fase 2. É importante ressaltar que, além da paralelização, o APS-RS apresenta um re-equacionamento da formulação matemática do algoritmo, o que também contribuiu para a redução do esforço computacional durante a execução. E, para o AF-RLM, apresentou-se uma estratégia de paralelização para o cálculo do vetor dos coeficientes de regressão.

O objetivo foi mostrar que é possível reduzir o custo computacional do APS utilizando as implementações APS-RLM e APS-RS. Ainda, com base nos resultados obtidos, mostrou-se que, apesar de o APS ser um algoritmo tradicional e eficiente para seleção de variáveis, a implementação AF-RLM pode ser mais eficaz na construção de um modelo com uma capacidade de predição mais adequada. A vantagem da implementação AF-RLM foi demonstrada em um exemplo que envolve um número relativamente grande de variáveis.

O conjunto de dados empregado consistiu em amostrais integrais de trigo, obtidas a partir de material vegetal de produtores canadenses. O teor de proteína foi escolhido como propriedade de interesse. Os resultados mostraram que o modelo de regressão linear múltipla obtido utilizando as variáveis selecionadas pelo AF-RLM pode produzir os menores erros de predição. Adicionalmente, ganhos de *speedup* foram obtidos com a implementação paralela do AF-RLM. O AF-RLM paralelizado mostrou-se cinco vezes mais rápido que sua implementação sequencial. Por outro lado, o APS-RS-CUDA, do ponto de vista computacional, seria uma implementação mais apropriada. No entanto, em comparação com o APS-RLM e APS-RS em termos de RMSEP, MAPE e

PRESS, o AF-RLM pode ser uma contribuição mais relevante para o problema de seleção de variáveis em problemas de calibração multivariada.

7.1 Trabalhos Futuros

Trabalhos seguintes nessa linha de pesquisa poderão envolver problemas de calibração multivariada ainda maiores. Por exemplo, níveis de enxofre em amostras de díesel poderão ser determinados utilizando as implementações APS-RLM, APS-RS e AF-RLM. Em tal exemplo, um número elevado de variáveis poderá estar envolvido na propriedade de interesse. Espera-se, nesse caso, que o desempenho computacional seja ainda mais elevado utilizando as paralelizações propostas. Adicionalmente, alternativas à arquitetura CUDA e integração CUDA-MATLAB, tal como OpenCL [100], poderão ser investigadas para a realização de estudos comparativos.

7.2 Artigos Produzidos

Apresenta-se aqui as premiações obtidas e os trabalhos publicados, elaborados e aceitos para publicação durante a realização desta dissertação. Na Seção 7.2.1, estão destacados os artigos publicados em congressos e periódicos nacionais e internacionais. A Seção 7.2.2 mostra os manuscritos que estão em fase de elaboração e os que ainda estão em processo de avaliação por Pares. Por fim, a Seção 7.2.3 lista as premiações que foram adquiridas até a defesa da dissertação. As listas dos artigos estão ordenadas por ordem decrescente do ano de publicação.

7.2.1 Publicados

1. Parallelization of a Modified Firefly Algorithm using GPU for Variable Selection in a Multivariate Calibration Problem. International Journal of Natural Computing Research (Qualis B5), v. 4, p. 31-42, 2014.
2. Paralelização do Algoritmo das Projeções Sucessivas em GPU usando uma Implementação das Regressões Sequenciais para Seleção de Variáveis em Problemas de Calibração Multivariada. In: Congresso Nacional de Matemática Aplicada e Computacional (Qualis B4), 2014, Natal, RN. Anais do CNMAC, 2014.
3. Partial Parallelization of the Successive Projections Algorithm using Compute Unified Device Architecture. In: The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (Qualis B2), 2013, Las Vegas, USA. Proceedings of PDPTA, 2013. p. 737-741.

7.2.2 Em fase de elaboração e em processo de avaliação

1. A GPU-based Implementation of the Firefly Algorithm for Variable Selection in Multivariate Calibration Problems (EM AVALIAÇÃO). Plos One (Qualis A1).
2. Parallelization of the Successive Projections Algorithm using a Sequential Regression Implementation with GPU for Variable Selection in Multivariate Calibration (EM ELABORAÇÃO). Revista Concurrency and Computation (Qualis A2).

7.2.3 Prêmios

1. 2014 - Convite para publicação de artigo completo no Volume 4, Issue 1 do International Journal of Natural Computing Research.
2. 2013 - Melhor artigo completo publicado no XIV Simpósio em Sistemas Computacionais (WSCAD-SSC).
3. 2013 - Menção de Mérito - Artigo convidado para publicação no Periódico Journal of the Brazilian Computer Society.
4. 2013 - Certificado de Reconhecimento do Conselho Universitário (CONSUNI) da Universidade Federal de Goiás.

Referências Bibliográficas

[1] AKAIKE, H. **A new look at the statistical model identification**. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

[2] ALLEN, D. **The relationship between variable selection and data agumentation and a method for prediction**. *Technometrics*, 16(1):125–127, 1974.

[3] ALMEIDA, F. **Espectroscopia de infravermelho próximo com transformada de fourier (ft-nir) na caracterização de farinhas para alimentação pueril**. *Lisboa: Instituto Superior Técnico da Universidade Técnica de Lisboa*, 2009.

[4] ARAÚJO, M. C. U.; SALDANHA, T. C. B.; GALVÃO, R. K. H.; YONEYAMA, T.; CHAME, H. C.; VISANI, V. **The successive projections algorithm for variable selection in spectroscopic multicomponent analysis**. *Chemometrics and Intelligent Laboratory Systems*, 57(2):65–73, 2001.

[5] ATASOY, N. A.; SEN, B.; SELCUK, B. **Using gauss-jordan elimination method with cuda for linear circuit equation systems**. *Procedia Technology*, 1(0):31–35, 2012.

[6] BANATI, H.; MONIKA, B. **Fire fly based feature selection approach**. *International Journal of Computer Science Issues*, 8(2):473–480, 2011.

[7] BARAK, A.; BEN-NUN, T.; LEVY, E.; SHILOH, A. **A package for opencl based heterogeneous computing on clusters with many gpu devices**. In: *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, p. 1–7. IEEE, 2010.

[8] BARTOLI, A. **On computing the prediction sum of squares statistic in linear least squares problems with multiple parameter or measurement sets**. *International journal of computer vision*, 85(2):133–142, 2009.

[9] BEEBE, K. R.; PELL, R. J.; SEASHOLTZ, M. B. **Chemometrics: a practical guide**. 1998.

[10] BOWINS, E. C. **A comparison of sequential and gpu implementations of iterative methods to compute reachability probabilities**. In: *Proceedings First Workshop on GRAPH Inspection and Traversal Engineering*, p. 20–34, 2012.

[11] BOX, G. E. P.; JENKINS, G. M.; REINSEL, G. C. **Time series analysis: forecasting and control**. Wiley. com, 2013.

[12] BRADSTREET, R. B. **The kjeldahl method for organic nitrogen.** *The Kjeldahl method for organic nitrogen.*, 1965.

[13] BREITKREITZ, M. C.; RAIMUNDO, I. M.; ROHWEDDER, J. J. R.; PASQUINI, C.; FILHO, H. A. D.; JOSE, G. E.; ARAUJO, M. C. U. **Determination of total sulfur in diesel fuel employing nir spectroscopy and multivariate calibration**. *The Analyst*, 128:1204–1207, 2003.

[14] CARNEIRO, R. L. **Algorítimos genéticos para a seleção de variáveis em métodos de calibração de segunda ordem**. PhD thesis, Dissertação de mestrado, Universidade Estadual de Campinas, Instituto de Química, Campinas-SP, 2007.

[15] CHAN, L. M.; SRINIVASAN, R. **A graphic processing unit (gpu) algorithm for improved variable selection in multivariate process monitoring**. In: *11th International Symposium on Process Systems Engineering*, volume 31 de **Computer Aided Chemical Engineering**, p. 1532–1536. Elsevier, 2012.

[16] CHAU, F.-T.; LIANG, Y.-Z.; GAO, J.; SHAO, X.-G. **Chemometrics: from basics to wavelet transform**, volume 234. Wiley, 2004.

[17] CHONG, I.-G.; JUN, C.-H. **Performance of some variable selection methods when multicollinearity is present**. *Chemometrics and Intelligent Laboratory Systems*, 78(1):103–112, 2005.

[18] CHURCHILL, A. W.; HUSBANDS, P.; PHILIPPIDES, A. **Tool sequence optimization using synchronous and asynchronous parallel multi-objective evolutionary algorithms with heterogeneous evaluations**. *IEEE Congress on Evolutionary Computation*, 2013.

[19] COIFMAN, R. R.; WICKERHAUSER, M. V. **Entropy-based algorithms for best basis selection**. *Information Theory, IEEE Transactions on*, 38(2):713–718, 1992.

[20] CORTINA, J. M. **Interaction, nonlinearity, and multicollinearity: Implications for multiple regression**. *Journal of Management*, 19(4):915–922, 1994.

[21] $CUDA^{TM}$, N. **Accelerating MATLAB with CUDA**, volume 1. NVIDIA Corporation, 2007.

[22] $CUDA^{TM}$, N. **NVIDIA CUDA C Programming Best Practices Guide**. NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2009.

[23] $CUDA^{TM}$, N. **OpenCL Programming Guide for the CUDA Architecture**. NVIDIA Corporation, 2009.

[24] $CUDA^{TM}$, N. **NVIDIA CUDA C Programming Guide**. NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 4.0 edition, 2011.

[25] DE REFLECTÂNCIA DIFUSA, C. I. http://www.idrc-chambersburg.org/shootout.html, 2008.

[26] DRAPER, N. R.; SMITH, H. **Applied regression analysis**. 1998.

[27] ESMAEILZADEH, H.; BLEM, E.; BURGER, D. **Power limitations and dark silicon challenge the future of multicore**. *ACM Transactions on Computer Systems (TOCS)*, 30(3):11, 2012.

[28] FABRIS, F.; KROHLING, R. A. **A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on gpu using c-cuda**. *Expert Systems with Applications*, 39(12):10324–10333, 2012.

[29] FERRÃO, L. M. S.; PLOTZE, R. O. **Computação distribuída: o melhor aproveitamento de recursos computacionais**. *Linguagem acadêmica*, 2(1):93–119, 2012.

[30] FERREIRA, E. B. **Processamento paralelo aplicado a métodos filogenéticos comparativos**. Dissertação (ciência da computação), Universidade Federal de Goias, jun 2012.

[31] FERREIRA, M. M. C.; ANTUNES, A. M.; MELGO, M. S.; VOLPE, P. L. O. **Chemometrics i: a tutorial of multivariate calibration**. http://www.scielo.br/scielo.php?pid=S0100-40421999000500016&script=sci_arttext, 1999.

[32] FILHO, A. R. G. **Avaliação do uso de reamostragem e combinação de modelos em regressão linear múltipla empregando o algoritmo das projeções sucessivas**. Dissertação (engenharia eletrônica e computação), Instituto Tecnológico de Aeronáutica, 2010.

[33] FILHO, A. R. G.; GALVÃO, R. K. H.; ARAÚJO, M. C. U. **Effect of the subsampling ratio in the application of subagging for multivariate calibration with the successive projections algorithm**. *Journal of the Brazilian Chemical Society*, 22:2225–2233, 11 2011.

[34] FILHO, P. A. C.; POPPI, R. J. **Aplicação de algoritmos genéticos na seleção de variáveis em espectroscopia no infravermelho médio. determinação simultânea de glicose, maltose e frutose**. *Quim. Nova*, 25(1):46–52, 2002.

[35] FLYNN, M. J.; RUDD, K. W. **Parallel architectures**. *ACM Computing Surveys (CSUR)*, 28(1):67–70, 1996.

[36] FUJIMOTO, N.; TSUTSUI, S. **Parallelizing a genetic operator for gpus**. *IEEE Congress on Evolutionary Computation*, 2013.

[37] GAIOSO, R. R. A.; JRADI, W.; PAULA, L. C. M.; ALENCAR, W.; MARTINS, W. S.; NASCIMENTO, H.; CACERES, E. **Paralelização do algoritmo floyd-warshall usando gpu**. In: *XIV Simposio em Sistemas Computacionais*, p. 19–25, 2013.

[38] GALVAO, R. K. H.; ARAUJO, M. C. U.; FRAGOSO, W. D.; SILVA, E. C.; JOSE, G. E.; SOARES, S. F. C.; PAIVA, H. M. **A variable elimination method to improve the parsimony of {MLR} models using the successive projections algorithm**. *Chemometrics and Intelligent Laboratory Systems*, 92(1):83–91, 2008.

[39] GANDOMI, A. H.; YANG, X.-S.; ALAVI, A. H. **Mixed variable structural optimization using firefly algorithm**. *Computers & Structures*, 89(23):2325–2336, 2011.

[40] GELADI, P.; MARTENS, H. **A calibration tutorial for spectral data. part 1. data pretreatment and principal component regression using matlab**. *Journal of Near Infrared Spectroscopy*, 4:225, 1996.

[41] GEORGE, E. I. **The variable selection problem**. *Journal of the American Statistical Association*, 95(452):1304–1308, 2000.

[42] GOLDMAN, A. **Modelos para computação paralela**. In: *Escola Regional de Alto Desempenho*, volume 3, 2003.

[43] GOLI, M.; MCCALL, J.; BROWN, C.; JANJIC, V.; HAMMOND, K. **Mapping parallel programs to heterogeneous gpu architectures using a monte carlo tree**. *IEEE Congress on Evolutionary Computation*, 2013.

[44] GREWE, D.; OBOYLE, M. F. P. **A static task partitioning approach for heterogeneous systems using opencl**. In: *Compiler Construction*, p. 286–305. Springer, 2011.

[45] HAIR, J. F.; ANDERSON, R. E.; TATHAM, R. L. **Análise Multivariada de Dados**, volume 5. Bookman, 2007.

[46] HORNG, M.-H. **Vector quantization using the firefly algorithm for image compression**. *Expert Systems with Applications*, 39(1):1078–1091, 2012.

[47] HUSSELMANN, A. V.; HAWICK, K. A. **Parallel parametric optimisation with firefly algorithms on graphical processing units**. In: *Proc. Int. Conf. on Genetic and Evolutionary Methods. Number CSTN-141, Las Vegas, USA, CSREA (16-19 July 2012)*, p. 77–83, 2012.

[48] HUSSELMANN, A. V.; HAWICK, K. A. **Geometric firefly algorithms on graphical processing units**. In: *Cuckoo Search and Firefly Algorithm*, p. 245–269. Springer, 2014.

[49] INFOESCOLA. **Espectrofotômetro**. http://www.infoescola.com/materiais-de-laboratorio/espectrofotometro, 2013.

[50] JAASKELAINEN, P. O.; DE LA LAMA, C. S.; HUERTA, P.; TAKALA, J. H. **Opencl-based design methodology for application-specific processors**. In: *Embedded Computer Systems (SAMOS), 2010 International Conference on*, p. 223–230. IEEE, 2010.

[51] KARIMI, K.; DICKSON, N.; HAMZE, F. **A performance comparison of cuda and opencl**. *arXiv preprint arXiv:1005.2581*, 2010.

[52] KIRK, D. B. **NVIDIA cuda software and gpu parallel computing architecture**. NVIDIA Corporation, 2008.

[53] KIRK, D. B.; HWU, W. **Programando para processadores paralelos**. Elsevier, 1 edition, 2011.

[54] KOMATSU, K.; SATO, K.; ARAI, Y.; KOYAMA, K.; TAKIZAWA, H.; KOBAYASHI, H. **Evaluating performance and portability of opencl programs**. In: *The fifth international workshop on automatic performance tuning*, 2010.

[55] KONG, J.; DIMITROV, M.; YANG, Y.; LIYANAGE, J.; CAO, L.; STAPLES, J.; MANTOR, M.; ZHOU, H. **Accelerating matlab image processing toolbox functions on gpus**. In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, p. 75–85. ACM, 2010.

[56] KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. **Introduction to parallel computing**, volume 110. Benjamin/Cummings Redwood City, 1994.

[57] LAWSON, C. L.; HANSON, R. J. **Solving least squares problems**, volume 161. SIAM, 1974.

[58] LIU, X.; CHENG, L.; ZHOU, Q. **Research and comparison of cuda gpu programming in matlab and mathematica**. In: *Proceedings of 2013 Chinese Intelligent Automation Conference*, p. 251–257. Springer, 2013.

[59] LUCENA, D. V. **Algoritmos evolutivo multiobjetivo para seleção de variáveis em problemas de calibração multivariada**. Dissertação (ciência da computação), Universidade Federal de Goias, maio 2013.

[60] LUKASIK, S.; ZAK, S. **Firefly algorithm for continuous constrained optimization tasks**. In: *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, p. 97–106. Springer, 2009.

[61] MAKRIDAKIS, S.; HIBON, M. **Evaluating accuracy (or error) measures**. *INSEAD*, 1995.

[62] MARTENS, H. **Multivariate calibration**. John Wiley & Sons, 1991.

[63] MATHWORKS. **Gpu programming in matlab**. http://www.mathworks.com/company/newsletters/articles/gpu-programming-in-matlab.html, 2011.

[64] MATHWORKS. **Introducing mex-files**. http://www.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html, 2011.

[65] MATHWORKS. **Matlab gpu computing support for nvidia cuda-enabled gpus**. http://www.mathworks.com/discovery/matlab-gpu.html, 2013.

[66] MCINTOSH-SMITH, S. **The gpu computing revolution**. 2011.

[67] MILLER, A. J. **Selection of subsets of regression variables**. *Journal of the Royal Statistical Society. Series A (General)*, p. 389–425, 1984.

[68] NAES, T.; MEVIK, B.-H. **Understanding the collinearity problem in regression and discriminant analysis**. *Journal of Chemometrics*, 15(4):413–426, 2001.

[69] NAVAUX, P. O. A. **Introdução ao processamento paralelo**. *Revista Brasileira de Computação*, 5(2):31–43, 1989.

[70] NEVES, A. C. O. **Espectroscopia no infravermelho próximo e métodos de calibração multivariada aplicados à determinação simultânea de parâmetros bioquímicos em plasma sanguíneo**. Dissertação (química, Universidade Federal do Rio Grande do Norte, fev 2013.

[71] OLIVEIRA, F. C. C. **Modelos de calibração multivariada associados a espectroscopia vibracional para analise de misturas diesel-óleos vegetais**. Dissertação (química), Universidade de Brasilia, 2006.

[72] PAULA, L. C. M. **Programação paralela cuda para simulação de modelos epidemiológicos baseados em indivíduos**. Trabalho de conclusão de curso (ciência da computação), Pontifícia Universidade Católica de Goiás, jun 2012.

[73] PAULA, L. C. M. **Implementação paralela do método bicgstab(2) em gpu usando cuda e matlab para solução de sistemas lineares**. *Revista de Sistemas e Computação*, 3(2):125–131, 2013.

[74] PAULA, L. C. M. **Paralelização e comparação de métodos iterativos na solução de sistemas lineares grandes e esparsos**. *ForScience: Revista Científica do IFMG*, 1(1):01–12, 2013.

[75] PAULA, L. C. M. **Cuda vs. opencl: Uma comparação teórica e tecnológica**. *ForScience: Revista Científica do IFMG*, 2(1):In press, 2014.

[76] PAULA, L. C. M. **Programação paralela de um método iterativo para solução de grandes sistemas de equações lineares usando a integração cuda-matlab**. *Revista de Sistemas e Computação*, 4(1):In press, 2014.

[77] PAULA, L. C. M.; SOARES, A. S.; SOARES, T. W.; DELBEM, A. C. B.; COELHO, C. J.; FILHO, A. R. G. **Parallelization of a modified firefly algorithm using gpu for variable selection in a multivariate calibration problem**. *International Journal of Natural Computing Research*, 4(1):31–42, 2014.

[78] PAULA, L. C. M.; SOARES, A. S.; SOARES, T. W.; FILHO, A. R. G.; COELHO, C. J. **Paralelização do algoritmo das projeções sucessivas em gpu usando uma implementação das regressões sequenciais para seleção de variáeis em problemas de calibração multivariada**. In: *XXXV Congresso Nacional de Matemática Aplicada e Computacional*, 2014.

[79] PAULA, L. C. M.; SOARES, A. S.; SOARES, T. W.; MARTINS, W. S.; FILHO, A. R. G.; COELHO, C. J. **Partial parallelization of the successive projections algorithm using compute unified device architecture**. In: *International Conference on Parallel and Distributed Processing Techniques and Applications*, p. 737–741, 2013.

[80] PAULA, L. C. M.; SOUZA, L. B. S.; SOUZA, L. B. S.; MARTINS, W. S. **Aplicação de processamento paralelo em método iterativo para solução de sistemas lineares**. In: *X Encontro Anual de Computação*, p. 129–136, 2013.

[81] PEREIRA, A. F. C.; PONTES, M. J. C.; NETO, F. F. G.; SANTOS, S. R. B.; GALVAO, R. K. H.; ARAUJO, M. C. U. **Nir spectrometric determination of quality parameters in vegetable oils using pls and variable selection**. *Food Research International*, 41:341–348, 2008.

[82] PONTES, M. J. C.; GALVAO, R. K. H.; ARAUJO, M. C. U.; MOREIRA, P. N. T.; NETO, O. D. P.; JOSE, G. E.; SALDANHA, T. C. B. **The successive projections algorithm for spectral variable selection in classification problems**. *Chemometrics and Intelligent Laboratory Systems*, 78(1):11–18, 2005.

[83] QUINN, M. J. **Parallel Programming**, volume 526. TMH CSE, 2003.

[84] SATO, Y.; SATO, M. **Parallelization and fault-tolerance of evolutionary computation on many-core processors**. *IEEE Congress on Evolutionary Computation*, 2013.

[85] SCHWARZ, G. **Estimating the dimension of a model**. *The annals of statistics*, 6(2):461–464, 1978.

[86] SCOTON, F. M.; LEE, E. J. H.; ALVAREZ, L. G. P. **Uma aplicação científica utilizando processamento paralelo para a arquitetura cell be**. *Escola Politécnica da USP*, 2007.

[87] SENTHILNATH, J.; OMKAR, S. N.; MANI, V. **Clustering using firefly algorithm: Performance study**. *Swarm and Evolutionary Computation*, 1(3):164–171, 2011.

[88] SIMEK, V.; ASN, R. R. **Gpu acceleration of 2d-dwt image compression in matlab with cuda**. In: *Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on*, p. 274–277. IEEE, 2008.

[89] SKOOG, D. A.; HOLLER, F. J.; NIEMAN, T. A. **Principles of instrumental analysis**. 1998.

[90] SOARES, A. S.; DELBEM, A. C. B.; LIMA, T. W.; COELHO, C. J.; SOARES, F. A. **Mutation-based compact genetic algorithm for spectroscopy variable selection in the determination of protein in wheat grain samples**. *Eletronic Letters*, 49:80–92, 2013.

[91] SOARES, A. S.; FILHO, A. R. G.; GALVÃO, R. K. H.; ARAÚJO, M. C. U. **Improving the computational efficiency of the successive projections algorithm by using a sequential regression implementation: a case study involving nir spectrometric analysis of wheat samples**. *Journal of the Brazilian Chemical Society*, 21(4):760–763, 2010.

[92] SOARES, A. S.; GALVÃO, R. K. H.; ARAÚJO, M. C. U.; SOARES, S. F. C.; PINTO, L. A. **Multi-core computation in chemometrics: case studies of voltammetric and nir spectrometric analyses**. *Journal of the Brazilian Chemical Society*, 21:1626–1634, 2010.

[93] SOARES, S. F.; GOMES, A. A.; ARAÚJO, M. C.; FILHO, A. R. G.; GALVÃO, R. K. **The successive projections algorithm**. *TrAC Trends in Analytical Chemistry*, 42:94–98, 2012.

[94] SOARES, S. F. C.; GOMES, A. A.; ARAUJO, M. C. U.; GALVÃO, R. K. H.; OTHERS. **The successive projections algorithm**. *TrAC Trends in Analytical Chemistry*, 42:84–98, 2013.

[95] STALLINGS, W. **Arquitetura e Organização de Computadores**. Prentice Hall, São Paulo, SP, Brasil, 5 edition, 2002.

[96] SUGANUMA, T.; KRISHNAMURTHY, R. B.; OHARA, M.; NAKATANI, T. **Scaling analytics applications with opencl for loosely coupled heterogeneous clusters**. In: *Proceedings of the ACM International Conference on Computing Frontiers*, p. 35. ACM, 2013.

[97] TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos: Princípios e Paradigmas**. Pearson Prentice Hall, 2007.

[98] TARPEY, T. **A note on the prediction sum of squares statistic for restricted least squares**. *The American Statistician*, 54(2):116–118, 2000.

[99] TOBIAS, R. **An introduction to partial least squares regression**. In: *Proceedings of Ann. SAS Users Group Int. Conf., 20th, Orlando, FL*, p. 2–5, 1995.

[100] TSUCHIYAMA, R.; NAKAMURA, T.; IIZUKA, T.; ASAHARA, A.; SON, J.; MIKI, S. **The OpenCL Programming Book**. Fixstars, 2010.

[101] WESTAD, F.; MARTENS, H. **Variable selection in nir based on significance testing in partial least squares regression**. *Journal of Near Infrared Spectroscopy*, 8(2):117–124, 2000.

[102] YANG, X. S. **Nature-inspired metaheuristic algorithms**. Luniver Press, 2008.

[103] YANG, X. S. **Firefly algorithms for multimodal optimization**. In: *Stochastic algorithms: foundations and applications*, p. 169–178. Springer, 2009.

[104] YANG, X.-S. **Firefly algorithm, levy flights and global optimization**. In: *Research and Development in Intelligent Systems XXVI*, p. 209–218. Springer, 2010.

[105] YANG, X. S. **Firefly algorithm, stochastic test functions and design optimisation**. *International Journal of Bio-Inspired Computation*, 2(2):78–84, 2010.

[106] YANG, X.-S.; HE, X. **Firefly algorithm: recent advances and applications**. *International Journal of Swarm Intelligence*, 1(1):36–50, 2013.

[107] YLDIRIM, A. A.; OZDOGAN, C. **Parallel wavelet-based clustering algorithm on gpus using cuda**. *Procedia Computer Science*, 3(0):396–400, 2011.