UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE FÍSICA
PROGRAMA DE PÓS GRADUAÇÃO EM FÍSICA AMBIENTAL

GeneSE: Um ambiente computacional para cálculo de balanço de energia da superfície utilizando imagens de satélites

Raphael de Souza Rosa Gomes

Orientador: **Prof. Dr. Josiel Maimone de Figueiredo**

Cuiabá - MT
Março/2015

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE FÍSICA
PROGRAMA DE PÓS GRADUAÇÃO EM FÍSICA AMBIENTAL

GeneSE: Um ambiente computacional para cálculo de balanço de energia da superfície utilizando imagens de satélites

Raphael de Souza Rosa Gomes

Tese apresentada ao Programa de Pós Graduação em Física Ambiental da Universidade Federal de Mato Grosso, como parte dos requisitos para obtenção do título de Doutor em Física Ambiental.

Prof. Dr. Josiel Maimone de Figueiredo

Cuiabá, MT

Março/2015

Dados Internacionais de Catalogação na Fonte.

S729g Souza Rosa Gomes, Raphael de.
GeneSE: Um ambiente computacional para cálculo de balanço de energia da superfície utilizando imagens de satélites / Raphael de Souza Rosa Gomes. -- 2015
87 f. : il. color. ; 30 cm.

Orientador: Josiel Maimone de Figueiredo.
Tese (doutorado) - Universidade Federal de Mato Grosso, Instituto de Física, Programa de Pós-Graduação em Física Ambiental, Cuiabá, 2015.
Inclui bibliografia.

1. SEB. 2. paralelismo. 3. GPU. 4. CUDA. 5. OpenCL. I. Título.

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE FÍSICA
Programa de Pós-Graduação em Física Ambiental

FOLHA DE APROVAÇÃO

TÍTULO: **GeneSe: UM AMBIENTE COMPUTACIONAL PARA CÁLCULO DE BALANÇO DE ENERGIA DA SUPERFÍCIE UTILIZANDO IMAGENS DE SATÉLITES**

AUTOR: **RAPHAEL DE SOUZA ROSA GOMES**

Tese de Doutorado defendida e aprovada em 17 de março de 2015, pela comissão julgadora:

Prof. Dr. Josiel Maimone de Figueiredo
Orientador
Instituto de Computação – UFMT

Prof. Dr. Marcelo Sacardi Biudes
Examinador Interno
Instituto de Física – UFMT

Prof. Dr. Maurício Fernando Lima Pereira
Examinador Interno
Instituto de Computação – UFMT

Prof. Dr. Marcos Rodrigues Vieira
Examinador Externo
University of California – Riverside - EUA

Prof. Dr. Geison Jader de Melo
Examinador Externo
Instituto Federal de Mato Grosso/IFMT

DEDICATÓRIA

À minha família e principalmente à minha amada.

Agradecimentos

Gostaria de agradecer primeiramente à UFMT e ao PGFA pelo apoio que sempre deram ao trabalho.

Gostaria de agradecer a todas as pessoas do PGFA que fizeram parte deste trabalho, em especial Victor Hugo e o Prof. Marcelo.

Quero agradecer também aos meus alunos de iniciação científica, Cristhian, Phellipe, Fábio, Vinícius e Mattyws pela ajuda quando necessário, ou simplesmente por aturar um cara "pegando no pé".

Agradeço também ao professor Dr. José de Souza Nogueira (Paraná) pelo apoio em aprofundarmos o relacionamento da Computação com a Física Ambiental.

Agradeço ao meu orientador Dr. Josiel Maimone Figueiredo por sempre me cobrar a fazer algo a mais, e pelas conversas sobre o que queríamos com o trabalho.

Mas agradeço principalmente à minha família, aos meus filhos Clara e Davi, pelo raio de luz que são para meu ser, e não tem como expressar minha gratidão pela minha Daniela, sempre ao meu lado me chamando para fazer o que é melhor, a aprender e evoluir. Sem eles jamais chegaria aonde cheguei, por isso, Muito Obrigado!!!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

S-SEBI	Simplified Surface Energy Balance Index	2
SEB	Balanço de Energia de Superfície	59
SEBAL	Surface Energy Balance Algorithm for Land	2
SEBS	Surface Energy System	2
SEBTA	Surface Energy Balance with Topography Algorithm	2
SM	Streaming Multiprocessors	32
SP	Streaming Processor	32
SSEB	Simplified Surface Energy Balance	2
TSM	Two - Source Model	2

RESUMO

GOMES, R. S. R. *GeneSE*: Um ambiente computacional para cálculo de balanço de energia da superfície utilizando imagens de satélites. Cuiabá, 2015, 87f. Tese (Doutorado em Física Ambiental) - Instituto de Física, Universidade Federal de Mato Grosso.

O conhecimento a respeito das formas de utilização da água está se tornando cada vez mais necessário, uma vez que esse recurso é cada vez mais escasso. Assim, utilizar técnicas que estimem a quantidade de água evapotranspirada em uma determinada área é importante para o gerenciamento de processos hidrológicos, de irrigações, de processos industriais, entre outros. Contudo, tais técnicas são relativamente complexas e com diversas etapas envolvidas para o cálculo final. Assim, o presente trabalho teve como objetivo desenvolver a plataforma GeneSE, que é uma plataforma computacional integrada que facilita o uso dessas técnicas para o cálculo da evapotranspiração utilizando balanço de energia de superfície a partir de imagens de satélites, tais como as técnicas SEBAL, SEBTA, SSEB e S-SEBI. Todas as técnicas envolvidas neste trabalho foram implementadas contemplando processamento sequencial e paralelo, como CPU e GPU. As principais facilidades fornecidas pela plataforma GeneSE são: (1) abstração e encapsulamento das técnicas; (2) facilidade no uso de técnicas diversas; (3) automatização de diversas etapas do processamento. Para validar a plataforma GeneSE foi utilizado um conjunto de 85 imagens de satélite de quatro áreas diferentes do estado de Mato Grosso. Todas essas imagens fora processadas em 4 dispositivos GPU's distintos, com 4 plataformas desenvolvidas no ambiente computacional (Java, OpenCL GPU, OpenCL CPU e CUDA). Os testes mostraram que o ambiente gasta uma média de 150 milissegundos para criar e compilar o código-fonte. Os resultados evidenciaram também que a utilização das plataformas paralelas diminui o tempo de processamento em 80 vezes, além de utilizarem a mesma quantidade de memória que uma plataforma sequencial. Após os testes de desempenho, realizou-se a validação com os dados medidos de torres e em duas áreas o R^2 apresentou valores maiores que 0,6 mostrando assim, que os algoritmos são significativos para a estimativa da evapotranspiração.

Palavras-chave: SEB, paralelismo, GPU, CUDA, OpenCL

ABSTRACT

GOMES, R. S. R. *GeneSE*: A computational environment for the surface energy balance calculation using satellite images. Cuiabá, 2015, 87f. Tese (Doutorado em Física Ambiental) - Instituto de Física, Universidade Federal de Mato Grosso.

The knowledge about the forms of water use is becoming increasingly necessary, since this resource is also becoming scarce. Thus, using techniques that estimate the amount of water evapotranspired in a particular area is important for the management of hydrological processes, irrigation, industrial processes, among others. However, such techniques are relatively complexes and involve several steps for the final calculation. Thus, this study developed the GeneSE platform, which is an integrated computing platform that facilitates the use of these techniques for the determination of evapotranspiration using surface energy balance based on satellite images, such as SEBAL, SEBTA, SSEB and S-SEBI techniques. All the techniques involved in this work were implemented covering sequential and parallel processing, with CPU and GPU. The main facilities provided by the GeneSE platform are: (1) abstraction and encapsulation of the techniques; (2) ease of use of several techniques; (3) automation of various processing stages. To validate the Genesis platform was used a set of 85 satellite images from four different areas of the state of Mato Grosso. All these images were processed by 4 different GPU devices with 4 platforms developed in the computing environment (Java, OpenCL GPU, CPU OpenCL and CUDA). The tests showed that the environment spends an average of 150 milliseconds to create and compile the source code. The results also showed that the use of parallel platforms reduces the processing time up to 80 times with use the same amount of memory that sequentially platform does. After the performance tests, their validation was performed with the measured data from towers and three areas. The R^2 technique results presented values higher than 0.6, thus showing that the algorithms are significant for estimating evapotranspiration.

Keywords: SEB, paralelism, GPU, CUDA, OpenCL

Capítulo 1

INTRODUÇÃO

1.1 PROBLEMÁTICA

Uma grande quantidade de processos ecológicos da Terra tem a água como componente principal para sua realização. Esse elemento é um dos principais componentes para a determinação, por exemplo, do clima e da meteorologia. Além disso, a água é fundamental para a manutenção e sobrevivência do homem, principalmente na produção e disponibilização de alimentos. Com o atual crescimento populacional, sua demanda também cresce, gerando escassez desse recurso, o que representa uma ameaça não só para a sociedade humana, mas também para a vida terrestre. Para investigar o consumo/uso da água pela agricultura e pela atividade industrial tem-se utilizado o indicador da evapotranspiração, que tem demostrado ser o melhor para essas medidas.

A evapotranspiração (ET), definida como a quantidade de água evaporada e transpirada, representa cerca de 60% do total de água que sai da superfície terrestre e vai para atmosfera, sendo o segundo maior componente do ciclo hidrológico em escala global. Ela utiliza mais da metade da energia solar total absorvida pela Terra, sendo assim um componente importante para o balanço de energia. Portanto, nos estudos dos ecossistemas, um dos maiores desafios é a determinação do valor da ET. Um dos fatores desse desafio é a grande quantidade de modelos disponíveis para sensoriamento remoto, cada um com a sua especificidade, fazendo com que o pesquisador passe mais tempo implementando a solução do que de fato analisando os resultados. Outro fator é o tempo necessário para processar cada um desses modelos, pois apresentam etapas manuais e subjetivas, necessitando de uma análise prévia do especialista, desperdiçando tempo de análise dos resultados.

1.2 JUSTIFICATIVA

Diante do quadro apresentado, realizar estimativas confiáveis de ET é essencial para a melhoria do manejo da água em irrigações, em hidroelétricas, processos industriais, entre outros. Atualmente, existem vários modelos para estimar a ET a partir do balanço de energia terrestre (BE) utilizando vários dados, desde informações de torres meteorológicas, até aqueles baseados em dados de sensoriamento remoto em diferentes resoluções espaço-temporais. Esses modelos vêm demostrando eficiência no mapeamento da ET diária e sazonal em escala regional com grande precisão. Os modelos (ou algoritmos) mais comumente utilizados para estimar ET baseados no balanço de energia utilizando dados de sensores orbitais são:

1. *Surface Energy Balance Algorithm for Land* – SEBAL;
2. *Simplified Surface Energy Balance Index* – S-SEBI;
3. *Simplified Surface Energy Balance* – SSEB;
4. *Surface Energy Balance with Topography Algorithm* – SEBTA;
5. *Two – Source Model* – TSM;
6. *Surface Energy Balance System* – SEBS;
7. *Mapping Evapotranspiration at High Resolution with Internalized Calibration* – METRIC.

Para cada um desses algoritmos, pode-se utilizar diferentes sensores espectrais, sendo que os mais utilizados são os provenientes do satélite LandSat e dos sensores MODIS presentes nos satélites Aqua e Terra. Contudo, utilizar esses algoritmos não é uma tarefa fácil, pois o pesquisador precisa conhecer várias técnicas e vários tipos de satélites e sensores, além de cada técnica apresentar etapas manuais e subjetivas, necessitando de uma análise prévia. Outro aspecto complicador é a comparação dos resultados gerados, pois a complexidade de desenvolvimento de cada um muitas vezes impossibilita suas aplicações e comparações. Essas características dificultam a difusão e a utilização de tais algoritmos, fazendo com que a análise da evapotranspiração aconteça somente em algumas regiões por somente uma ou outra técnica, com somente um ou outro satélite com poucas imagens, devido ao fato também do processamento ser trabalhoso e demorado.

Este trabalho, portanto, tem como objetivo geral criar e validar um ambiente computacional que facilite e flexibilize o cálculo do balanço de energia da superfície utilizando vários algoritmos de sensoriamento remoto, possibilitando também a execução dos cálculos por dispositivos de processamento paralelo. Para atingir essa meta têm-se os seguintes objetivos específicos:

- Estudar e implementar várias técnicas que calculam o balanço de energia utilizando imagens de satélites (SEBAL, SEBTA, S-SEBI e SSEB);
- Estudar e implementar cada método do item anterior para as imagens do satélite LandSat e do sensor MODIS;
- Aplicar técnicas de paralelismo em GPU e CPU para os algoritmos implementados com o uso de CUDA e OpenCL;
- Desenvolver ambiente computacional (GeneSE: Generic Surface Energy) que integre todos os algoritmos implementados;
- Desenvolver editor de fórmulas que abstraia os aspectos de programação dos algoritmos;
- Testar o ambiente para analisar o tempo gasto para calcular uma imagem, bem como a quantidade de memória utilizada
- Validar os algoritmos com dados medidos

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Como a água é de fundamental importância para uma gama de processos dos sistemas da Terra, e a evapotranspiração representa cerca de 60% de toda a água que volta para a atmosfera, existem diversos modelos para estimar a ET, sendo que em Gowda et al. (2008) é apresentada uma análise detalhada de diferentes algoritmos para estimativa de ET (TSM, SEEBI, S-SEBI, SEBS, SEBAL, kB^{-1}, Aproximação Beta β, METRIC), relatando que a precisão da estimativa de ET varia de 67 a 97% para ET diária, enquanto que para ET sazonal esta precisão ultrapassa 94%, indicando que a junção de técnicas de sensoriamento remoto com algoritmos apropriados, tem grande potencial para estimar adequadamente a ET.

A utilização de modelos para balanço de energia de superfície não é um processo fácil, pois é necessário que o pesquisador possua conhecimentos detalhados do modelo, bem como de conhecimentos avançados nos aspectos computacionais para conseguir implementá-los. Outro aspecto complicador é a comparação dos resultados gerados pelos diversos métodos, pois a complexidade de desenvolvimento de cada método muitas vezes impossibilita suas aplicações e comparações. Isso dificulta a difusão dos algoritmos e consequentemente a aplicação desses modelos.

Os modelos que estimam balanço de energia da superfície terrestre por meio de sensoriamento remoto requerem poucos dados de superfície para determinar os componentes: fluxo de calor no solo (G), fluxo de calor latente (LE), saldo de radiação (Rn) e o fluxo de calor sensível (H), sendo esses componentes determinados a partir das imagens obtidas por satélites em todos os modelos

Neste capítulo são apresentados os aspectos teóricos para o entendimento do cálculo do balanço de energia de superfície através de imagens de satélite. Na Seção 2.1 estão conceituados os tipos de satélites mais utilizados para essa

classe de métodos; posteriormente na Seção 2.2 são apresentados os algoritmos utilizados para o desenvolvimento deste trabalho, juntamente com toda a definição matemática necessária para o seu entendimento. Nas Seções 2.3 e 2.4 estão definidos resumidamente os conceitos de Processamento Paralelo e Programação Paralela, e na Seção 2.5 é apresentado o conceito principal de programação em GPU, bem como as duas linguagens utilizadas, CUDA e OpenCL.

2.1 Sensores Orbitais

2.1.1 Land Remote Sensing Satellite (LandSat)

LandSat é um projeto da NASA que captura informações espectrais da superfície da Terra, que possui resolução espacial moderada (NASA, 2014a). A Tabela 1 mostra um resumo da série histórica do programa, incluindo as datas de lançamentos e os sensores presentes em cada um deles, sendo o primeiro satélite lançado em 1972, o Landsat-1.

Tabela 1: Resumo dos satélites Landsat lançados (MARKHAM; HELDER, 2012; NASA, 2014a)

Satélite	Data de Lançamento	Sensores	
Landsat-1	23 de Julho de 1972	MSS (4 bandas)	RBV (3 bandas)
Landsat-2	22 de Janeiro de 1975	MSS (4 bandas)	RBV (3 bandas)
Landsat-3	5 de Março de 1978	MSS (5 bandas)	RBV (somente pan)
Landsat-4	16 de Julho de 1982	TM (7 bandas)	MSS (4 bandas)
Landsat-5	1 de Março de 1984	TM (7 bandas)	MSS (4 bandas)
Landsat-6	5 de Outubro 1993	ETM (8 bandas)	
Landsat-7	15 de Abril de 1999	ETM + (8 bandas)	
Landsat-8	11 de Fevereiro 2013	OLI (9 bandas)	TIRS (2 bandas)

Para utilizar os dados dos satélites nos cálculos de cada algoritmo, faz-se necessário realizar uma calibração, que é o processo de ajustar os valores de cada banda. Os pixels de cada imagem original, no caso dos sensores TM e ETM+, variam de 0 a 255 (8 bits), pois os sensores dos satélites medem a radiância espectral e os transformam em escala de tons de cinza. Essa quantidade de bits para armazenamento varia de acordo com o sensor. A Tabela 2 apresenta a composição das bandas do Landsat 5, juntamente com os coeficientes de calibração e a irradiação espectral.

Tabela 2: Descrição das Bandas do Landsat 5

Bandas	Comprimento de Onda (μm)	Coeficientes de Calibração ($Wm^{-2}sr^{-1}\mu m^{-1}$)		Irradiação Espectral no Topo da Atmosfera ($Wm^{-2}\mu m^{-1}$)
		a	b	
1 (azul)	0,45 – 0,52	-1,52	193	1957
2 (verde)	0,52 – 0,60	-2,84	365	1826
3 (vermelho)	0,63 – 0,69	-1,17	264	1554
4 (IV-próximo)	0,76 – 0,79	-1,51	221	1036
5 (IV-médio)	1,55 – 1,75	-0,37	30,2	215
6 (IV-termal)	10,4 – 12,5	1,2378	15,303	-
7 (IV-médio)	2,08 – 2,35	-0,15	16,5	80,67

O processo começa com a realização da calibração radiométrica, banda a banda, de acordo com a equação:

$$L_{\lambda i} = a_i + \left(\frac{b_i - a_i}{255}\right) ND \tag{1}$$

onde $L_{\lambda i}$ representa a radiância espectral da banda i, a_i e b_i são coeficientes de calibração de cada banda ($Wm^{-2}sr^{-1}\mu m^{-1}$), e ND é o número digital do pixel, valor entre 0 e 255. Após a calibração radiométrica é realizado o cálculo da *reflectância*, que representa a quantidade de energia detectada pelo sensor, ou seja, a porção da radiação eletromagnética refletida e capturada pelo sensor, determinada por:

$$\rho_{\lambda i} = \frac{\pi L_{\lambda i}}{k_{\lambda i} \cos Z d_r} \tag{2}$$

onde $k_{\lambda i}$ é a irradiância solar espectral ($Wm^{-2}\mu m^{-1}$) da banda i (Tabela 2), Z representa o ângulo zenital e d_r é o quadrado da razão entre a distância média Terra-Sol e a distância Terra-Sol em um dado dia do ano, denominado dia juliano (DJ). O ângulo zenital é definido como o ângulo entre a reta vertical local e os raios solares, tal qual mostra a Figura 1. O d_r é definido por (IQBAL, 1983):

$$d_r = 1 + 0,033 \cos\left(\frac{2\pi DJ}{365}\right) \tag{3}$$

onde DJ representa o dia sequencial do ano, sendo o argumento do cosseno em radianos. Neste ponto as calibrações dos dados estão finalizadas, então tais dados podem ser utilizados nos cálculos do balanço de energia, descritos nas seções posteriores.

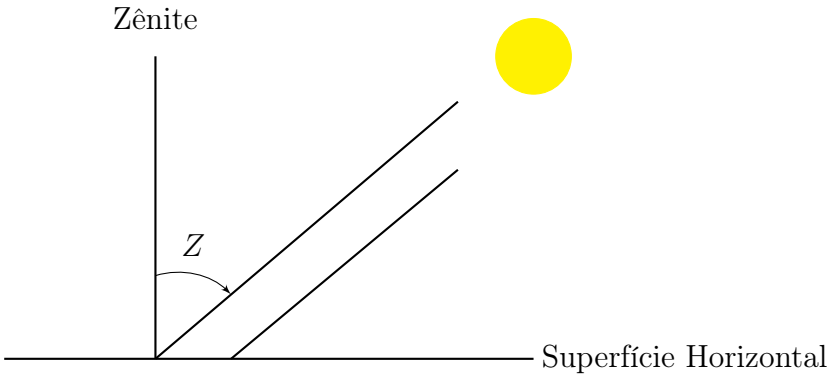


Figura 1: Ângulo Zenital

2.1.2 Moderate Resolution Imaging Spectroradiometer (MODIS)

MODIS é um tipo de dispositivo que está implantado em dois satélites em órbita: o Terra (1999) e o Aqua (2002), ambos compõem o Sistema de Observação da Terra (*EOS, Earth Observing System*), da NASA. Esses sensores possuem 36 bandas espectrais com resoluções diferentes (NASA, 2014b). As bandas 1 e 2 têm 250 metros de resolução espacial, já as bandas de 3 a 7 têm 500 metros de resolução e o restante tem 1 km de resolução. É um instrumento que tem uma sensibilidade radiométrica alta (16 bits) (NASA, 2014b).

O processo de calibração do MODIS é diferente do LandSat, sendo o cálculo da radiância espectral definido por:

$$L_{\lambda i} = Rad\_scale_i + (SI - Rad\_offset_i) \tag{4}$$

onde $L_{\lambda i}$ representa a radiação espectral da banda i, os coeficientes de calibração de cada banda são representados por Rad_scale_i e Rad_offset_i, e SI é o número que representa a conversão digital do pixel em um inteiro, valor entre 0 e 32767 (16 bits). Após essa etapa é realizado o cálculo da *reflectância* determinada por:

$$\rho_{\lambda i} = Ref\_scale_i + (SI - Ref\_offset_i) \quad (5)$$

onde Ref_scale_i e Ref_offset_i são coeficientes de calibração da reflectância para a banda i. Tanto os coeficientes da Equação 5 quanto da equação anterior são obtidos na própria imagem ficando armazenada em seus metadados.

2.2 ALGORITMOS PARA CÁLCULO DA ET DA SUPERFÍCIE UTILIZADOS NESSE TRABALHO

Foram utilizados os seguintes algoritmos dentre aqueles mais comumente estudados para estimar ET utilizando dados provenientes de sensores orbitais, baseados no balanço de energia (BE):

- Surface Energy Balance Algorithm for Land – SEBAL (BASTIAANSSEN et al., 1998a; BASTIAANSSEN et al., 1998b)
- Surface Energy Balance with Topography Algorithm – SEBTA (GAO et al., 2011)
- Simplified Surface Energy Balance – SSEB (SENAY et al., 2007; GOWDA et al., 2009)
- Simplified Surface Energy Balance Index – S-SEBI (ROERINK et al., 2000)

Para facilitar a compreensão da relação entre os algoritmos e seus parâmetros, a Figura 2 ilustra o esquema de todas as equações envolvidas no processo, e em qual parte dele é diferente para cada algoritmo. A apresentação da descrição de cada uma das variáveis contidas em cada uma delas será realizada ao longo desta seção.

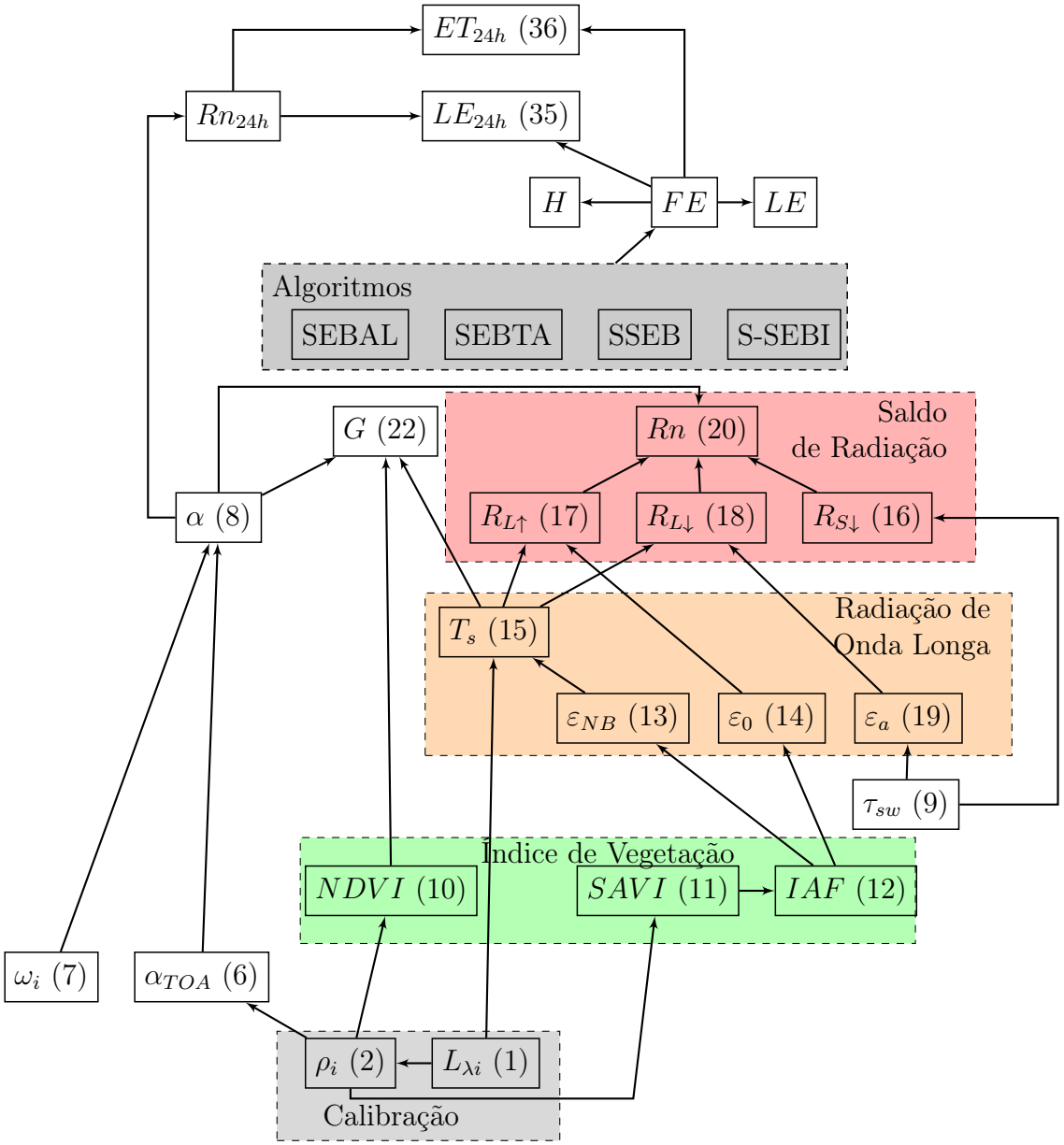


Figura 2: Diagrama de Equações para o Balanço de Energia, onde "(x)" é o número da equação listada neste documento.

Todos os algoritmos utilizados partem de um mesmo conjunto de equações. Após o processo de calibração de cada satélite o *albedo planetário* é calculado utilizando a *reflectância* definido por:

$$\alpha_{toa} = \sum(\omega_i \rho_i) \tag{6}$$

onde ρ_i é a *reflectância* da banda i e ω_i o peso daquela banda para o cálculo do *albedo*. O peso (ω_i) pode ser calculado por:

$$\omega_i = \frac{k_i}{\sum k_i} \tag{7}$$

onde o k_i representa a *irradiação* no topo da atmosfera na banda i, que no caso do Landsat 5 seus valores encontram-se na Tabela 2. Após o cálculo do *albedo planetário*, realiza-se a determinação do *albedo de superfície* por:

$$\alpha = \frac{\alpha_{toa} - \alpha_p}{\tau_{sw}^2} \tag{8}$$

onde α_p é a porção da *radiação solar incidente* refletida pela própria atmosfera, geralmente variando entre $0,025$ e $0,04$, sendo muito utilizado o valor $0,03$ (BASTIAANSSEN, 2000). Já τ_{sw} representa a *transmissividade atmosférica*, que pode ser definida como sendo a *radiação incidente* transmitida pela atmosfera, ou seja, aquilo que não foi absorvido e nem refletido. A *transmissividade*, quando há condições de céu claro, pode ser obtida pela equação a seguir (ALLEN et al., 2007):

$$\tau_{sw} = 0,75 + 2,10^{-5}z \tag{9}$$

onde z representa a altitude de cada pixel (metros).

Terminado o cálculo do *albedo*, os índices de vegetação são calculados, sendo mais utilizados os valores de NDVI, SAVI e IAF definidos pelas Equações 10, 11 e 12 respectivamente:

$$NDVI = \frac{\rho_{NIR} - \rho_{VIS}}{\rho_{NIR} + \rho_{VIS}} \tag{10}$$

$$SAVI = \frac{(1+L)\,(\rho_{NIR} - \rho_{VIS})}{L + \rho_{NIR} + \rho_{VIS}} \tag{11}$$

$$IAF = -\frac{\log\left(\frac{0,69-SAVI}{0,59}\right)}{0,91} \tag{12}$$

onde ρ_{NIR} representa a *reflectância* da banda do IV-próximo (vermelho próximo) e ρ_{VIS} é a *reflectância* da banda do vermelho visível, que no caso do Landsat 5, por exemplo são as bandas 4 e 3 respectivamente e no MODIS pelas bandas 1 e 2. A variável L é um fator que depende do tipo de solo da região analisada, sendo comumente utilizado o valor $0,1$.

Obtidos os valores dos índices de vegetação, o próximo cálculo a ser realizado é com referência às *emissividades da superfície*, ou seja, quanto de energia foi emitido de volta para a atmosfera. Duas emissividades são calculadas para representar porções diferentes do espectro, definidas pelas Equações 13 e 14:

$$\varepsilon_{NB} = 0,97 + 0,0033IAF \tag{13}$$

$$\varepsilon_0 = 0,95 + 0,01IAF \tag{14}$$

Ambas as equações são calculadas quando se tem $IAF < 3$, caso contrário, $\varepsilon_{NB} = \varepsilon_0 = 0,98$, contudo em Allen et al. (2002) é utilizado mais um filtro. Quando $NDVI < 0$ então $\varepsilon_{NB} = 0,99$ e $\varepsilon_0 = 0,985$.

Posteriormente às emissividades, é realizado o cálculo da *temperatura de superfície* e no caso do Landsat 5 é utilizado a equação:

$$T_s = \frac{K_2}{\ln\left(\frac{\varepsilon_{NB}K_1}{L_{\lambda,6}} + 1\right)} \tag{15}$$

onde K_1 e K_2 são constantes de calibração para cada satélite, sendo que para o Landsat 5 são utilizados os valores $607,76$ ($Wm^{-2}sr^{-1}\mu m^{-1}$) e $1260,56$ (K) respectivamente (ALLEN et al., 2002; SILVA et al., 2005).

Após todas essas variáveis determinadas começa-se o processo de obtenção dos componentes do *balanço de energia*, sendo que o primeiro é o *saldo de radiação*, que é dividido em três partes (Equações 16, 17 e 18):

$$R_{S\downarrow} = S\cos(Z)\, d_r \tau_{sw} \tag{16}$$

$$R_{L\uparrow} = \varepsilon_0 \sigma T_a^4 \tag{17}$$

$$R_{L\downarrow} = \varepsilon_a \sigma T_a^4 \tag{18}$$

onde σ é a constante de Stefan-Boltzman ($Wm^{-2}K^{-4}$), T_a é a *temperatura do ar*, S é a *constante solar*, e ε_a é a *emissividade da atmosfera* que é definido por:

$$\varepsilon_a = 0,85\left(-\ln\left(\tau_{sw}\right)\right)^{0,09} \tag{19}$$

A equação a seguir representa todo o *balanço de energia*, que é o objetivo final de todos os algoritmos:

$$Rn = LE + H + G \tag{20}$$

onde Rn é o *saldo de radiação*, LE a *densidade de fluxo de calor latente*, H a *densidade de fluxo de calor sensível* e G a *densidade de fluxo de calor no solo*, todos em $(\frac{W}{m^2})$. Depois de calcular os resultados das equações anteriores (Equações de 1 até 19) é possível obter o primeiro componente do *balanço de energia*, que é o *saldo de radiação* (Rn), computado pela equação a seguir (SILVA et al., 2005; ALLEN et al., 2002):

$$Rn = R_{S\downarrow}(1-\alpha) - R_{L\uparrow} + \varepsilon_0 R_{L\downarrow} \tag{21}$$

onde $R_{S\downarrow}$ é a *radiação de onda curta incidente*, α é o *albedo* de cada pixel, $R_{L\downarrow}$ é a *radiação de onda longa* emitida pela atmosfera na direção de cada pixel, $R_{L\uparrow}$ é a *radiação de onda longa* emitida por cada pixel e ε_0 é a *emissividade* de cada pixel.

O *fluxo de calor no solo* (G) é a taxa de armazenamento de calor no solo, devido à condução. Calcula-se a razão G/Rn utilizando a seguinte equação empírica desenvolvida por Bastiaanssen (2000) representando valores próximos do meio-dia:

$$G = (T_s - 273.16) \times (0.0038 + 0.0074\alpha) \times (1 - 0.98 \times NDVI^4) \times Rn \quad (22)$$

onde T_s é a *temperatura da superfície* ($^\circ C$), α é o *albedo da superfície* e NDVI é o *índice de vegetação* da diferença normalizada. Para efeito de correção dos valores do *fluxo de calor no solo* para corpos de água ($NDVI < 0$), pode ser utilizada a seguinte expressão: $G = 0.3Rn$ ou $G = 0.5Rn$, segundo Allen et al. (2002).

A obtenção dos outros componentes da Equação 20 é realizada como um resíduo da mesma. Todos os algoritmos usados neste trabalho realizam todos cálculos explicados até esta seção, contudo cada um deles modifica o modo como será obtido o resíduo do balanço de energia, que são os valores de LE e H, conforme explicação a seguir.

2.2.1 Surface Energy Balance Algorithm For Land (SEBAL)

No algoritmo SEBAL, o *fluxo de calor sensível* (H) é a *taxa de perda de calor* para o ar por convecção e condução, devido a uma diferença de *temperatura*. O valor de H é estimado com base na *velocidade do vento* e da *temperatura da superfície* usando uma calibração interna da diferença da temperatura próxima à superfície entre dois níveis da superfície, o que segundo Bastiaanssen et al. (1998a):

$$H = \frac{\rho c_p dT}{r_a} \quad (23)$$

onde ρ é *densidade do ar* ($\frac{kg}{m^3}$), cp é o *calor específico do ar* ($\frac{1004J}{Kg\ K}$), dT (K) é a diferença de *temperatura* (T1 – T2) entre as duas alturas ($z1$ e $z2$), e r_a é a *resistência aerodinâmica* para o transporte de calor ($\frac{s}{m}$).

O valor de H é uma função do gradiente de *temperatura* (dT), *rugosidade da superfície*, e da *velocidade do vento*. Para facilitar sua determinação são utilizados dois pixels "âncoras" (em que os valores de H podem ser previstos e de dT estimado) e a *velocidade do vento* na altura dada (MENDONÇA et al., 2012).

A Figura 3 mostra como é o processo iterativo que acontece com o SEBAL após a escolha do pixel quente e frio pelo especialista. A figura faz referência às equações envolvidas em cada processo, sendo todas descritas a seguir.

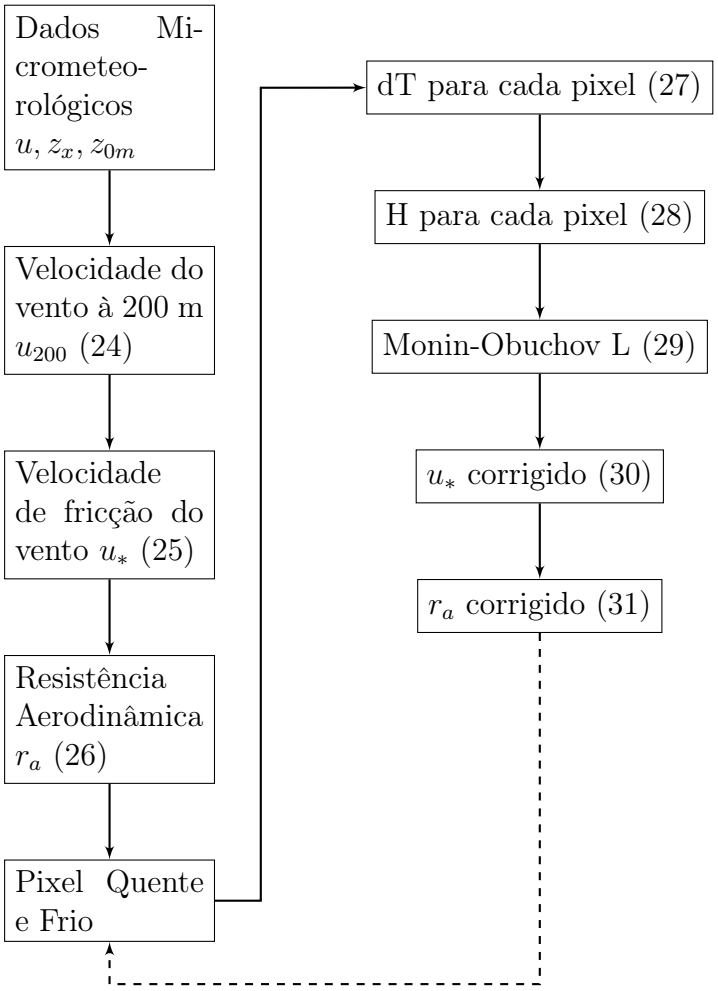


Figura 3: Processo Iterativo para a determinação dos coeficientes A e B.

Todo o processo começa com a aquisição de alguns dados de campo, tais como *velocidade do vento* , *pressão atmosférica*, *umidade relativa*. A partir de então, a *velocidade do vento* a $200m$ é calculada com:

$$u_{200} = u_* \frac{\ln\left(\frac{z_{200}}{z_{0m}}\right)}{k} \tag{24}$$

onde z_{200} representa a *altura* de $200m$, z_{0m} é a *rugosidade de superfície* e u_* representa a *velocidade de fricção do vento*, ou seja, a velocidade com que o vento muda na direção vertical (cisalhamento do vento) que é definida por:

$$u_* = \frac{k u_x}{\ln\left(\frac{z_{200}}{z_{0m}}\right)} \tag{25}$$

onde u_x é o dado de velocidade medido. A partir daqui calcula-se a *resistência aerodinâmica* com:

$$r_{ah} = \frac{\ln\left(\frac{z_2}{z_1}\right)}{u_* k} \tag{26}$$

onde z_1 e z_2 são *alturas* . Com o r_{ah}, obtêm-se alguns dados das imagens, tais como *temperatura* do pixel quente e frio, além do *saldo de radiação* , do *fluxo de calor no solo* e do *índice SAVI* do pixel quente, para então calcular os valores de dT e H dos pixels escolhidos como quente e frio, de acordo com as Equações 27 e 28:

$$dt = aT_s + b \tag{27}$$

$$H = \rho cp \frac{dT}{r_{ah}} \tag{28}$$

Com o valor de H é calculado o comprimento de Monin-Obuchov (L) para descrever os efeitos da turbulência através de:

$$L = -\frac{\rho\ cp\ u_*^3 T_s}{k.g.H} \tag{29}$$

onde g é a *aceleração da gravidade*, ρ é a *densidade do ar*, cp é o *coeficiente de calor sensível* e H é o *fluxo de calor sensível*. O comprimento de Monin-Obuchov serve para corrigir a *velocidade de fricção* (u_*) e a *resistência aerodinâmica* (r_{ah}) através das Equações 30 e 31:

$$u_* = \frac{k\ u_{200}}{\ln\left(\frac{z_{200}}{z_{0m}}\right) - \Psi_{m(200)}} \tag{30}$$

$$r_{ah} = \frac{\ln\left(\frac{z_2}{z_1}\right) - \Psi_{h(z_2)} + \Psi_{h(z_1)}}{u_*.k} \tag{31}$$

onde Ψ é uma função definida por:

$$\begin{cases} L < 0 & \Psi_m(y) = 2\ln\left(\dfrac{1+x(y)}{2}\right) + \ln\left(\dfrac{1+x(y)^2}{2}\right) - 2\arctan\left(x(y)\right) + 0,5\pi \\ & \Psi_h(y) = 2\ln\left(\dfrac{1+x(y)^2}{2}\right) \\ L > 0 & \Psi_m(y) = -5\left(\dfrac{2}{L}\right) \\ & \Psi_h(y) = -5\left(\dfrac{y}{L}\right) \\ L = 0 & \Psi_m(y) = \Psi_h(y) = 0 \end{cases} \tag{32}$$

onde a função $x(y)$ é definida como:

$$x(y) = \left(1 - 16\frac{y}{L}\right)^{0,25} \tag{33}$$

Esse processo de corrigir a *resistência aerodinâmica* é realizado até que a diferença entre a resistência corrigida e a anterior seja menor que um valor determinado pelo pesquisador, que representa a porcentagem de erro dessa variável.

Finalizado esse processo o *fluxo de calor sensível* (H) é determinado para então se calcular o restante da equação de *balanço de energia*. Uma vez que o *fluxo de calor latente* (LE) é computado para cada pixel, uma quantidade equivalente de LE instantâneo ($\frac{mm}{h}$) é calculada dividindo pelo *calor latente de vaporização* (λ). Esses valores são então extrapolados utilizando uma razão de ET de referência da cultura para obter níveis diários ou sazonais de ET.

A ET é determinada com base na *fração evaporativa instantânea* (FE_i), definida a seguir, como uma razão entre LE e $(Rn - G)$. Estudos mostram que FE_i é igual a fração evaporativa diária (FE_{24h}) (ALLEN et al., 2011; BASTIAANSSEN, 2000).

$$FE_i = \left(\frac{LE}{Rn - G}\right) = FE_{24h} = \frac{LE_{24h}}{Rn_{24h}} \tag{34}$$

Assumindo que o *fluxo de calor no solo diário* é igual a 0 ($G_{24h} = 0$), o *fluxo de calor latente diário* (LE_{24h}) é estimado por:

$$LE_{24h} = FE_i \times Rn_{24h} \tag{35}$$

No qual $Rn_{24h} = R_{S24h}(1 - albedo) - 110 \times \tau_{24h}$, onde R_{S24h} é a *radiação de onda curta* incidente diária (W/m^2) e o τ_{24h} é *transmissividade diária* (ALLEN et al., 2002). A conversão de LE_{24h} para *evapotranspiração diária* (ET_{24h}) (mm/dia) é:

$$ET_{24h} = \left(\frac{FE_i \times Rn_{24h} \times 86.4}{2450} \right) \tag{36}$$

Aqui todo o processo de obtenção da *evapotranspiração* é finalizado para o SEBAL, a seguir o SEBTA é explicado.

2.2.2 Surface Energy Balance With Topography Algorithm (SEBTA)

O algoritmo SEBTA utiliza as mesmas equações que o algoritmo SEBAL para o cálculo da evapotranspiração, apresentando diferença no computo da *temperatura de superfície*, pois o mesmo leva em consideração a elevação do terreno para corrigir a temperatura utilizando a carta digital do terreno (GAO et al., 2011).

$$Ts_{dem} = T_S + 0.0065(h - h_{mean}) \tag{37}$$

onde Ts_{dem} é a *temperatura de superfície* corrigida pela *elevação do terreno* no pixel dado, T_s é a *temperatura de superfície*, h é a *elevação* acima do nível do mar no pixel e h_{mean} é a *média de elevação* da área estudada. Outra consideração que o algoritmo do SEBTA faz é com relação à determinação dos pixels denominados "quente" e "frio". Enquanto no SEBAL sua determinação é feita manualmente, o SEBTA propõe um critério dessa escolha.

A escolha para o pixel "quente", é realizada através da determinação do pixel de maior temperatura de superfície e menor *índice de vegetação* $mSAVI$ (Equação 38). Já o pixel "frio" é determinado através do pixel com menor *temperatura de superfície* e maior *índice de vegetação*. Esses critérios pressupõem que numa região com pouca vegetação a temperatura aumente, ou seja, é uma região com o valor de *calor sensível* (H) máximo e de *calor latente* (LE) mínimo. Já uma região com vegetação mais densa representa *calor latente* máximo e *calor sensível* mínimo (GAO et al., 2011).

$$mSAVI = (0.5)\left((2.0 * \rho_4 + 1) - \sqrt{(2 * \rho_4 + 1)^2 - 8 * (\rho_4 - \rho_3)}\right) \quad (38)$$

onde ρ_i é a *reflectância* da banda i, assim a determinação dos pixels "âncoras" obedecem a um trapézio ilustrado no Figura 4.

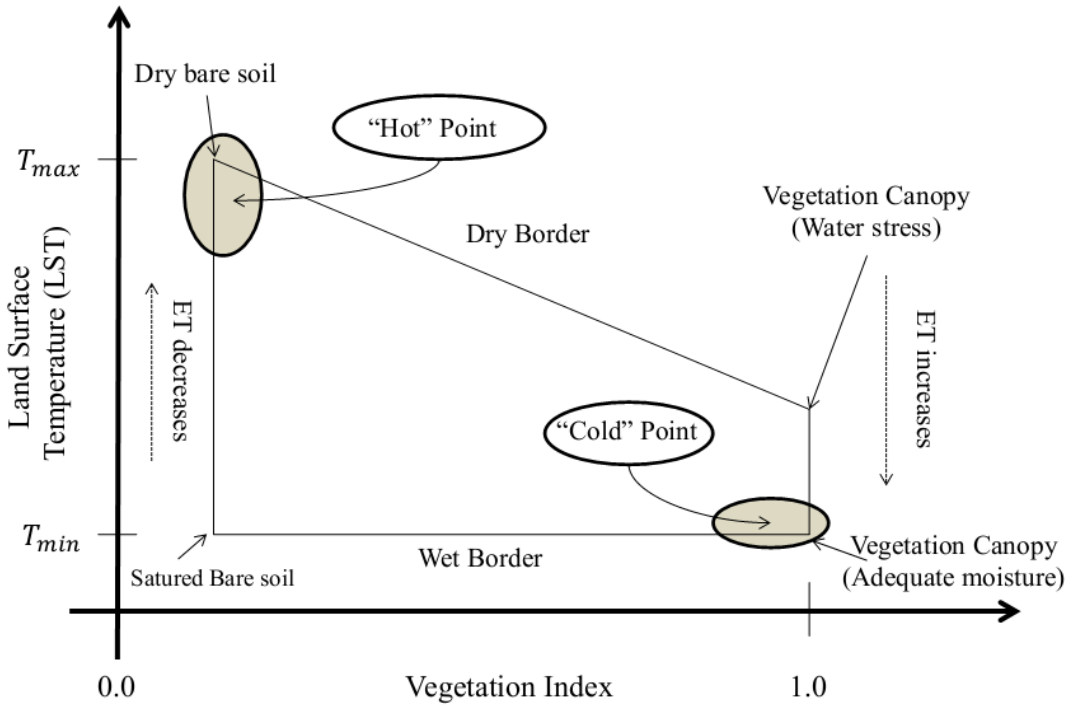


Figura 4: Estrutura esquemática do Trapézio para identificação dos pixels quente e frio, onde a abscissa esta relacionada aos índices de vegetação (mSAVI) e o eixo vertical é a temperatura da superfície (Ts)

Após a determinação dos pixels "âncoras", todo o processo iterativo descrito na Figura 3 é realizado também para o SEBTA. Assim, a principal diferença entre o SEBTA e o SEBAL está no modo como os pixels são escolhidos.

2.2.3 Simplified Surface Energy Balance (SSEB)

Abordagens de balanço de energia de superfície são baseados na *razão da evapotranspiração* como uma função de mudança de estado da água utilizando a energia disponível no ambiente para vaporização (SU et al., 2005). Sensoriamento remoto baseado no balanço de energia converte imagens de satélite em características de superfície, como *albedo*, *índices de vegetação* (IAF, NDVI ou SAVI), *temperatura de superfície*, entre outros, para estimar a *ET* como um resíduo da equação (GOWDA et al., 2009):

$$LE = Rn - H - G \tag{39}$$

O algoritmo SSEB diz que o LE varia linearmente entre o pixel "quente" e "frio" em proporção com a *temperatura de superfície* (GOWDA et al., 2009), assumindo que a diferença de temperatura entre o solo e o ar são correlacionados linearmente com a *quantidade de água no solo* (SADLER et al., 2000). Portanto, o SSEB calcula a *fração evaporativa* de cada pixel seguindo a equação:

$$FE_i = \frac{(T_H - T_s)}{(T_H - T_C)} \tag{40}$$

onde T_H e T_C representam a média da *temperatura de superfície* dos 3 pixels "quente" e 3 pixels "frio" respectivamente, sendo que o pixel "quente" é determinado pelos 3 pixels com menores $NDVI$ e o pixel "frio" pelos maiores $NDVI$; já a T_s é o valor da *temperatura de superfície* para cada pixel da imagem e o FE_i representa a *fração evaporativa* (SENAY et al., 2011). A partir disso calcula-se o LE e o H, para por fim calcular a ET. Portanto, neste algoritmo não existe o processo iterativo dos algoritmos anteriores, pois os pixels âncoras são calculados pelas médias.

2.2.4 Simplified Surface Energy Balance Index (S-SEBI)

Se as condições atmosféricas da área estudada forem consideradas constantes e se essa área reflete uma variação suficiente nas condições de hidrologia, então os fluxos podem ser determinados utilizando somente a própria imagem de detecção (ROERINK et al., 2000).

Esse algoritmo parte da Equação 20, sendo que o mesmo calcula o *fluxo de calor no solo* (G) e o *saldo de radiação* (Rn), para então calcular a fração evaporativa pela equação:

$$\Lambda = \frac{LE}{LE + H} = \frac{LE}{Rn - G} \tag{41}$$

Considerando constantes a *radiação global* e a *temperatura do ar*, a *temperatura de superfície* varia de acordo com o aumento do *albedo* (ROERINK et al., 2000). Assim, a *fração evaporativa* pode ser calculada pela equação:

$$\Lambda = \frac{T_H - T_s}{T_H - T_{LE}} \tag{42}$$

onde T_H e T_{LE} representam as *temperaturas de superfícies* no pixel "quente" e "frio" respectivamente.

Para a determinação dos valores de T_H e T_{LE} plota-se os valores da *temperatura de superfície* pelo *albedo* da imagem (Figura 5), e determina-se duas retas, uma superior e outra inferior, que representaram as retas "quente" e "fria" respectivamente.

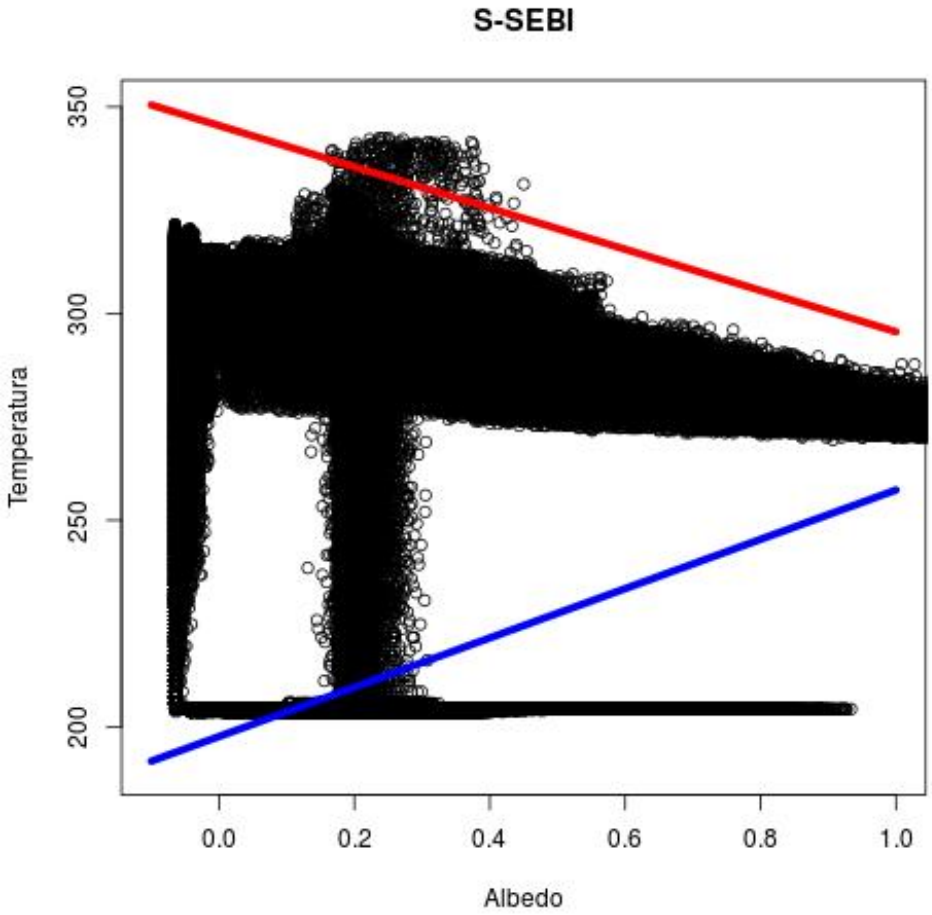


Figura 5: Representa o gráfico do *albedo* pela *temperatura de superfície* da imagem.

Com as retas determinadas podem-se calcular as *temperaturas* T_H e T_{LE} pelas equações:

$$T_H = a_H + b_H \alpha \tag{43}$$

$$T_{LE} = a_{LE} + b_{LE} \alpha \tag{44}$$

onde os valores a_H e b_H representam os coeficientes da reta "quente" e os valores de a_{LE} e b_{LE} os da reta "fria", e α representa o *albedo da superfície* do pixel.

Assim o cálculo da fração evaporativa fica:

$$\Lambda = \frac{a_H + b_H \alpha - T_s}{a_H + b_H \alpha - (a_{LE} + b_{LE} \alpha)} \tag{45}$$

Resolvendo tem-se:

$$\Lambda = \frac{a_H + b_H \alpha - T_s}{a_H - a_{LE} + (b_H - b_{LE}) \alpha} \tag{46}$$

Com isso, os valores de H, LE e ET podem ser calculados de modo similar aos outros algoritmos.

Em resumo, todos os algoritmos apresentados possuem uma base em comum, mas mudam radicalmente quanto à forma como vão calcular o LE e H. Pelo formato dos cálculos e dos dados utilizados, a paralelização dos mesmos pode ser uma necessidade, principalmente se a quantidade de dados a serem processadas for muito elevada.

2.3 PROCESSAMENTO PARALELO

Processamento paralelo pode ser definido como sendo uma forma eficiente de processar informações que enfatiza a exploração de eventos no processo computacional (HWANG; BRIGGS, 1984). Esse processamento paralelo aparece de várias formas: *pipeling*, vetorização, simultaneidade, concorrência, paralelismo de dados, particionamento, multiplicidade, replicação, compartilhamento de tempo, compartilhamento de espaço, multitarefa, multiprogramação ou multicomputadores, entre outros (HWANG, 1993), sendo alguns desses explicados mais a frente.

Todas as formas de processamento paralelo demandam algum tipo de gerenciamento com o objetivo de sincronizar as entradas e saídas de cada unidade de processamento. Por exemplo, a Figura 6 mostra a arquitetura típica para servidores distribuídos, onde é necessário existir uma política de tolerância a falhas e portanto algum controle relacionado com replicação, falha de transmissão, entre outros. Na Figura 6 tem-se a tarefa a ser distribuída e o *dispatcher* que determinará o local para qual a tarefa será enviada para o processamento. Após a finalização, o resultado é encaminhado de volta para o *dispatcher*, que por sua vez reenvia para a origem da tarefa.

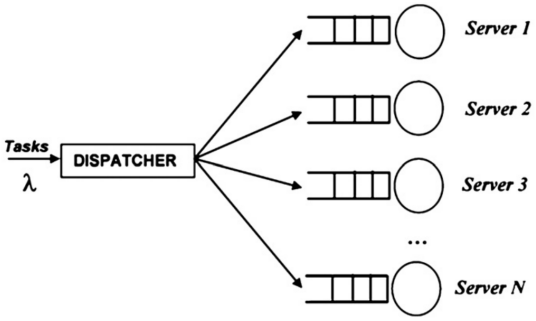


Figura 6: Modelo de servidor distribuído (SEMCHEDINE et al., 2011).

2.3.1 Classificação da Arquitetura Paralela de Computadores

A arquitetura dos computadores pode ser classificada segundo a taxonomia de Flynn (FLYNN, 1972). Essa taxonomia propõem a arquitetura em duas dimensões: instruções e dados, sendo que cada dimensão possui dois valores (*single* ou *multiple*), com isso tem-se quatro classificações ilustradas nas Figuras 7(a), 7(b), 7(c), 7(d).

- *Single Instruction, Single Data* (SISD): Uma instrução é executada em um conjunto de dados. Exemplo: computadores mono *thread*.

- *Multiple Instruction, Single Data* (MISD): Múltiplas instruções são executadas em um conjunto de dados. Exemplo: vários algoritmos de criptografia para decodificar uma mensagem.

- *Single Instruction, Multiple Data* (SIMD): Uma instrução é executada em múltiplos conjunto de dados. Exemplo: processamento de imagens em GPU.

- *Multiple Instruction, Multiple Data* (MIMD): Múltiplas instruções são executadas em múltiplos conjunto de dados. Exemplo: multiprocessadores e multicomputadores.

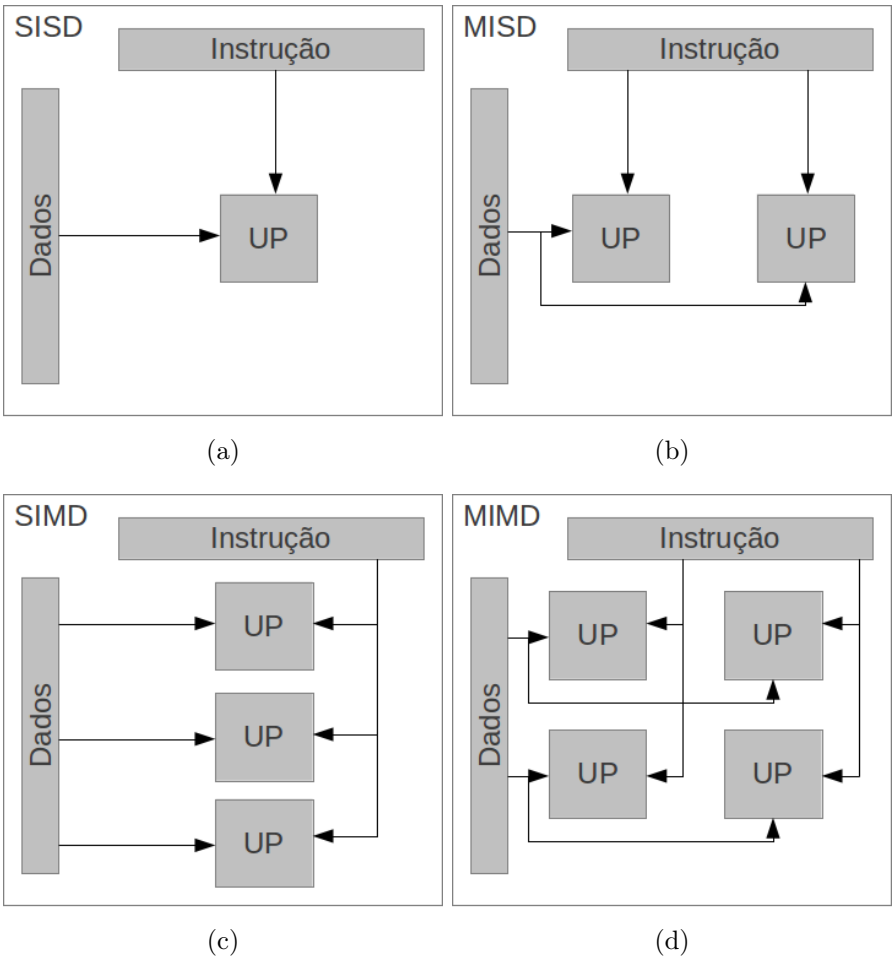


Figura 7: Classificações das arquiteturas computacionais.

2.3.2 CPU *Multinúcleos* e GPU

CPU *Multinúcleos* e GPU possuem em seu processador muitos núcleos, sendo o GPU voltado para paralelismo em dados (ELLIOTT; ANDERSON, 2012), enquanto CPU pode ter paralelismo de dados ou funcional. Portanto, CPU possui seus núcleos otimizados para processamento serial, enquanto GPU para processamento paralelo, sendo mais eficiente para lidar com múltiplas tarefas simultaneamente. No tópico 2.5 os conceitos sobre GPU são detalhados. A figura 8 ilustra a divisão dos processadores e GPU's em núcleos (CPU).

Atualmente os processadores podem atingir até 12 núcleos enquanto que as GPU's podem ter mais de 2000 núcleos, com isso pode-se visualizar um grande

potencial para cálculos científicos devido a grande quantidade de processos ou *threads* que podem ser executadas simultaneamente.

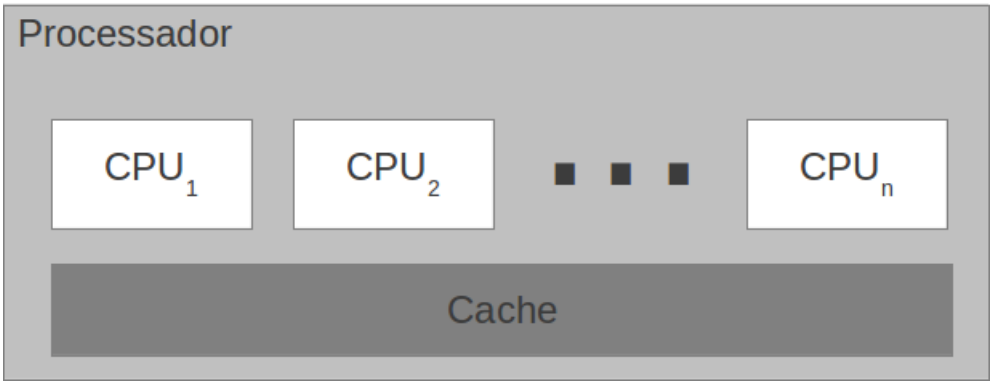


Figura 8: Divisão do processadores em núcleos

Outra característica importante dos multinúcleos ou *multi-cores*, além de executarem vários processos simultâneamente, é o baixo consumo de energia se comparado a um *single-core* (processador único). Isso acontece pelo fato dos *multi-cores* possuírem tempos de *clock* menores que os *single-cores*. Outro ponto é a menor produção de calor pelo fato de necessitarem de menos energia. Portanto, pode-se visualizar muito ganhos ao utilizar o processamento paralelo, contudo no presente estudo o fator analisado foi o ganho de desempenho sobre cálculos utilizando as GPU's em comparação com um *single-core*. Por essas características os dispositivos multicores são muito utilizados para acelerar aplicações em carros, smartphones, tablets, drones e robôs.

2.3.3 *Clusters*

Cluster pode ser descrito como a integração de mais de um computador e recursos incorporados através de hardware, redes e software para criar uma única imagem do sistema. Nas abordagens tradicionais os termos de HPC (*High-Performance Computing*) e *cluster* de computadores referenciam o mesmo tipo de ambiente computacional (VALENTINI et al., 2011).

Clusters são compostos por computadores sendo que cada um recebe o nome de nó. Cada nó pode ter diferentes características como a arquitetura de processador único ou múltiplo. Uma rede de computação em *cluster* é uma rede dedicada (VALENTINI et al., 2011).

Esse tipo de paralelismo é muito utilizado em sistemas bancários, sistemas meteorológicos, ferramentas de mapeamento genético, simuladores geotérmicos, programas de renderização de imagens tridimencionais, entre outros. Principal-

mente pelo fato de conseguir realizar manutenção no *cluster* sem a necessidade de parar o sistema, com isso é possível retirar ou colocar nós no *cluster* mesmo com o sistema funcionando.

2.3.4 *Grids*

Grid é um conjunto de sistemas heterogêneos e autônomos de larga escala, de vários domínios administrativos, geograficamente distribuídos e interligados por uma rede ampla(TORKESTANI, 2011).

Os recursos da *grid* podem ser livremente adicionadas ou retiradas a qualquer momento a critério do proprietário. O desempenho dos nós da rede e sua carga frequentemente mudam com o tempo. *Grids* permitem a seleção, agregação e compartilhamento dos recursos de software e hardware de computadores diferentes em uma forma distribuída (TORKESTANI, 2011). Sendo esse tipo de paralelismo muito utilizado no meio acadêmico, como forma de compartilhamento de recursos computacionais. Um exemplo de *grid* é do CERN (*Conseil Européen pour la Recherche Nucléaire* ou Organização Europeia para a Pesquisa Nuclear) para simulações de colisões de partículas.

2.3.5 *Cloud Computing* (Computação em Nuvens)

Cloud Computing pode ser definido como um novo estilo de computação em que os recursos são dinamicamente escaláveis, muitas vezes virtualizados e são fornecidos como um serviço através da Internet (FURHT, 2010).

A principal diferença entre *cloud computing* e *clusters* é que o primeiro muitas vezes pode assumir a forma de ferramentas baseadas na web, e suas aplicações normalmente são acessadas através de um navegador de internet (VALENTINI et al., 2011), tanto *grids* quanto *cloud computing* podem ser estruturados utilizando *clusters* (VALENTINI et al., 2011).

Esse tipo de computação é muito utilizado como serviços disponíveis pela WEB tais como o Google Docs[1], Amazon[2], Panda Cloud Antivirus[3], entre outros.

Para aproveitar todo esse processamento faz-se necessário utilizar linguagens que suportam a programação paralela.

[1] <http://www.google.com/docs/about/>
[2] <http://www.amazon.com>
[3] <http://www.cloudantivirus.com/>

2.4 PROGRAMAÇÃO PARALELA

A programação paralela consiste em desenvolver soluções utilizando o paralelismo suportado pelas máquinas atuais, incrementando a quantidade de processos executados simultaneamente. A programação paralela é desenvolvida de maneira diferente para cada tipo de arquitetura paralela: *cluster*, *grid*, *cloud*, GPU, entre outros. Pode-se obter o paralelismo dessas estruturas de várias formas:

- Aprender uma nova linguagem de programação voltada exclusivamente para essas estruturas como Occam, Ada ou HPF.
- Utilizar bibliotecas para programação paralela que utilize uma outra linguagem como hospedeira, como por exemplo, OpenMP, MPI, OpenCF, PVM, CUDA ou OpenCL.
- Utilizar compiladores que paralelizem o programa sequencial como por exemplo, Oxygen, OSCAR e PARADIGM para Fortran.

Para utilizar a programação paralela é necessário entender como modelar para paralelismo, pois a compreensão do problema é um fator decisivo na determinação de quais partes podem ser paralelizadas. Além da modelagem, também é importante entender como os processos são escalonados, pois o escalonamento pode influenciar diretamente na modelagem afim de se obter um maior ganho da estrutura que se está utilizando para o processamento paralelo. Outro fator relevante é entender quais são os tipo de acesso à memória, pois pode-se deixar que determinados dados fiquem mais próximos daqueles que se relacionam frequentemente, diminuindo assim o tempo de latência entre um acesso e outro.

2.4.1 Metodologia de paralelização

O paralelismo tenta transformar determinado problema, em problemas menores para que cada um possa ser processado de forma independente e paralela, otimizando o tempo de resposta. Para utilizar a programação paralela deve-se identificar e solucionar os problemas fundamentais do paralelismo (SARKAR, 1989):

- Identificar o paralelismo do problema;
- Particionar o problema em tarefas sequenciais e independentes;

- Dividir as tarefas em processos.

Ao resolver os problemas do paralelismo, teoricamente pode-se diminuir o tempo de resposta pela metade, se dividirmos o problema em dois. Contudo, é muito difícil atingir essa marca, pois a maioria dos problemas possui uma determinada parte que não pode ser subdividida, ou seja, permanece de forma sequencial. Com isso pode-se determinar o aumento de velocidade de um algoritmo pela lei de Amdahi (AMDAHL, 1967):

$$S = \frac{1}{r_s + \frac{r_p}{n}} \tag{47}$$

onde, S representa o ganho de velocidade, r_s é taxa sequencial do programa, r_p é a taxa paralelizável do programa e n é a quantidade de processos. Observe que $r_s + r_p = 1$. Ao analisar a lei de Amdahi, pode-se concluir que entender o problema é de fundamental importância na implementação de programas paralelos, pois se faz necessário conhecer qual parte é paralelizável e qual não é, para a partir desse ponto verificar as possíveis tecnologias para paralelismo e escolher a que melhor lhe convém. Contudo, ainda há pouca utilização do paralelismo, principalmente pela existência do problema da decomposição do algoritmo (HWANG; BRIGGS, 1984).

Para auxiliar o desenvolvimento de aplicações paralelas pode-se utilizar a metodologia de Foster (FOSTER, 1995), denominada PCAM, que é dividida em 4 partes:

- Decomposição/Partição (*Partition*): Esta etapa é destinada a realizar a divisão da tarefa em tarefas menores, de tal forma que tanto o cálculo quanto os dados associados a esse cálculo possam ser divididos.

- Comunicação (*Communicate*): A divisão das tarefas determinam o padrão de comunicação entre as tarefas, pois uma tarefa pode precisar acessar um dado que pertence a outra tarefa, sendo necessário determinar as estruturas e os algoritmos para realizar as comunicações entre as tarefas.

- Aglomeração (*Agglomerate*): As etapas anteriores eram etapas abstratas, portanto esta etapa tem por objetivo concretizar as abstrações anteriores, visando obter uma maior reutilização do código sequencial e diminuir o tempo de comunicação entre tarefas. Nesta etapa precisa-se também tomar

o cuidado para não limitar a escalabilidade do algoritmo, como por exemplo, se for utilizada uma matriz com as dimensões $512x256x2$, e a mesma for agrupada nas duas primeiras dimensões. Nesse caso ocorrerá uma limitação na escalabilidade em somente 2 processos ou processadores.

- Mapeamento (*Map*): Aqui é definida em qual processador as tarefas serão executadas para poder maximizar a ocupação dos processadores e minimizar sua comunicação.

A Figura 9 mostra todas as etapas da metodologia PCAM, onde o problema é analisado e particionado em tarefas menores. Então são verificadas quais tarefas se comunicam entre si, para a partir disso aglomerar aquelas que possuem muita comunicacão e finalizando, distribuir as tarefas entre os processadores.

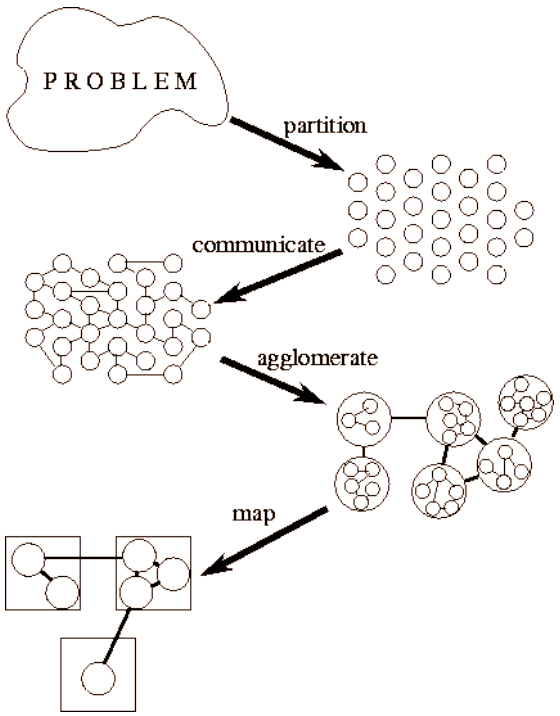


Figura 9: PCAM: uma metodologia de projeto para programas paralelos (FOSTER, 1995).

2.4.2 Tipos de Paralelismo

Além da taxonomia de Flynn existem também alguns tipos de paralelismo como por exemplo:

- Paralelismo Funcional: Operações diferentes são executadas por tarefas diferentes em um conjunto de dados diferente.

Algorithm 1 Algoritmo Funcional

1: $x = 1$;
2: $y = 1$;
3: $z = x + y$;
4: $w = x * y$;

No algoritmo 1 pode-se observar que as linhas 1 e 2 são independentes, e que as linhas 3 e 4 dependem das linhas anteriores, contudo, são independentes entre si.

- Paralelismo de Dados: A mesma operação pode ser realizado sobre conjunto de dados diferentes.

Algorithm 2 Algoritmo Dados

1: **for** $i = 0 \rightarrow tam\_vetor$ **do**
2: $a[i] = b[i] + c[i]$;
3: **end for**

No algoritmo 2 observa-se que a soma dos dados pode ser realizada de forma independente.

- *Pipeline*: Divide o problema em etapas, de forma que cada etapa pode ser executada juntamente com outra mas em objetos diferentes, como por exemplo, uma linha de montagem. A figura 10 mostra um *pipeline* dividido em 4 fases: busca, decodificação, execução e escrita. Pode-se observar que depois do 4 ciclo de *clock*, a cada novo ciclo é obtido o resultado de uma execução, reduzindo assim o tempo de resposta.

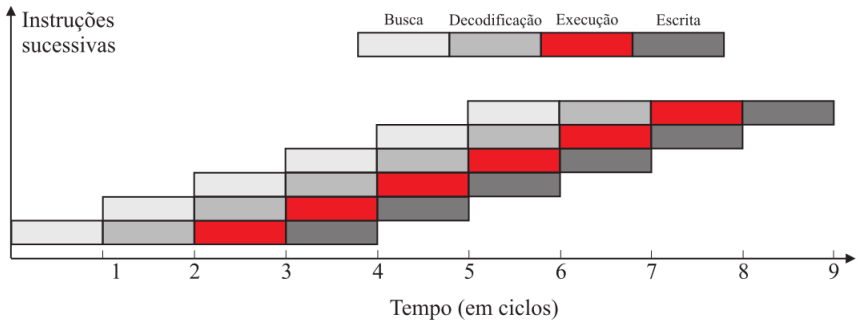


Figura 10: Execução em *pipeline* com uso de instruções sucessivas, onde o sistema passa a executar, após alguns ciclos, uma instrução por ciclo (PEREIRA, 2007).

2.4.3 Acesso à memória

Uma outra problemática da programação paralela é o acesso à memória, que pode apresentar diferentes formas. Duas delas são:

- Memória Compartilhada: Os dados ficam armazenados em um único local , onde todos os processos podem ter acesso a esse determinado espaço, como mostra a Figura 11.

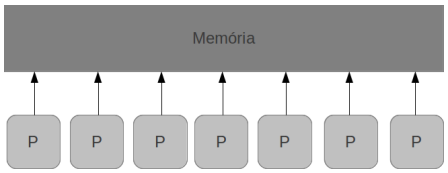


Figura 11: Memória Compartilhada

- Memória Distribuída: Nesse tipo de acesso cada processo possui um local de armazenamento de dados onde estes buscam as informações, como mostra a Figura 12.

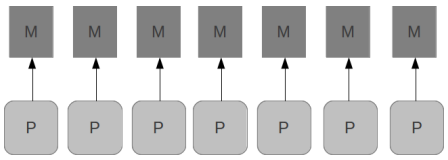


Figura 12: Memória Distribuída

Geralmente existem as duas abordagens na comunicação entre processos, devido ao fato de a memória local ter um acesso mais rápido e a memória compartilhada permitir que todos os processo utilizem os mesmos dados.

2.4.3.1 Classificação dos sistemas de memória compartilhada

A Figura 11 mostra que os processos P's podem acessar os dados da memória. Quando uma requisição de acesso chega à memória, esta é direcionada para o controlador da memória. Se aquele módulo que a requisição deseja acessar não estiver ocupado, o controlador responde a requisição permitindo o acesso e marca aquele módulo como ocupado. Se o módulo requerente estiver ocupado, o controlador emite um sinal de ocupado para a requisição, fazendo com que o processo requerente espere a liberação de acesso (HWANG, 1993). Segundo Hwang

(1993), baseado nesse processo de comunicação, pode-se classificar a memória compartilhada em:

1. *Uniform Memory Access* (UMA): A memória é acessível por todos os processadores através de uma rede de interconexão da mesma forma que um único processador acessa a memória. Todos os processadores têm tempo de acesso igual a qualquer local de memória. A interligação da rede usada em UMA pode ser um único barramento, barramentos múltiplos, ou um *switch*. Como o acesso à memória compartilhada é balanceado, esses sistemas são chamados de sistemas SMP (*symmetric multiprocessor*). Cada processador tem igual oportunidade de leitura / gravação à memória, incluindo velocidades de acessos iguais.

2. *Nonuniform Memory Access* (NUMA): Cada processador liga-se a uma parte da memória compartilhada. A memória tem um único espaço de endereço. Portanto, qualquer processador pode acessar qualquer local de memória diretamente através do seu endereço real. No entanto, o tempo de acesso aos módulos depende da distância do processador.

3. *Cache-Only Memory Architecture* (COMA): Similar ao NUMA, cada processador liga-se a uma parte da memória compartilhada. No entanto, neste caso, a memória partilhada é a memória cache. O COMA requer a transferência dos dados para o processador requisitante. Não há hierarquia de memória e o espaço de endereço é feito de todos os caches.

Dentro do contexto dessas classificações temos as GPU, classificadas como UMA, ou seja, todos os núcleos da GPU podem acessar uma mesma memória. Portanto o programador tem a disposição em qualquer núcleo os mesmos dados. Esses e outros detalhes da GPU são explicadas a seguir.

2.5 GENERAL PURPOSE GRAPHICS PROCESSING UNIT (GPGPU)

A programação para GPGPU é utilizada desde 1978, sendo amplamente difundida no mundo científico a partir dos anos 2000, como é visto em Weigel (2012), Yamanaka et al. (2011), além de estar presente em mais de 70% dos supercomputadores no mundo (TOP500, 2015). Essa presença acontece devido ao fato do seu poder de processamento, quantidade de núcleos e custo baixo.

Por essas características, a arquitetura escolhida para este trabalho foi a GPU, explicada a seguir.

Por volta de 1980, as primeiras GPU's foram modeladas utilizando o conceito de *pipeline* gráfico, ou seja, um modelo conceitual de estágios que os dados são enviados através de uma combinação de hardware (núcleos GPU) e software da CPU. O termo "GPU" (Unidade de Processamento Gráfico) não existia ainda, sendo introduzido somente em 1999 pela NVIDIA[4] com o lançamento da placa GeForce 256.

A abordagem de *pipeline* é amplamente utilizada pelas principais fabricantes como NVIDIA, ATI[5], etc, acelerando o uso das GPU's nos computadores (CROW, 2005). Conforme a programação com estágios (*pipeline*) se tornava mais geral, o hardware pode ser transformado para *stream processor* de propósito geral (PURCELL et al., 2002).

A arquitetura das GPU's tem como abordagem o uso de um vetor escalável de multi-thread, Streaming Multiprocessors (SM) (NVIDIA, 2011), e cada Streaming Processor (SP) gerencia a alocação de memória, sincronização e a comunicação entre os SP's (PAPAKONSTANTINOU et al., 2009). Essa arquitetura facilita o acesso ao paralelismo das GPU's. A realização de instruções na GPU acontece utilizando pipeline, sendo que cada estágio é executado por um hardware específico paralelo. Contudo, existe a possibilidade dos estágios serem executados em um único hardware utilizando a unidade programável (OWENS et al., 2008).

Com essas características as GPU's estão sendo amplamente utilizadas em diversas resoluções de problemas em áreas distintas como encontra-se em: matemática (LUTSYSHYN, 2015), física (JOSELLI et al., 2015), química (CHENG et al., 2015), médica (ZHANG et al., 2015), física nuclear (GOVENDER et al., 2014), teoria da complexidade (GOMES; FIGUEIREDO, 2014).

Para utilizar as GPU se faz necessário a utilização de uma linguagem, sendo as duas principais: CUDA e OpenCL. A seguir são apresentadas as características de cada uma delas.

2.5.1 Compute Unified Device Architecture (CUDA)

CUDA é uma arquitetura e um modelo de programação paralela em placas GPU NVIDIA, voltada para resolver problemas computacionais complexos de uma forma mais eficiente do que em uma CPU. Foi desenvolvida também para suportar várias linguagens como o CUDA C, CUDA Fortran e OpenCL (NVIDIA,

[4] <http://www.nvidia.com.br/>
[5] <http://www.amd.com/pt-br/products/graphics/>

2011).

O CUDA funciona agrupando *threads* criadas pelo programa em blocos (Figura 13(a)), e cada bloco é então executado independentemente do outro em um determinado núcleo (NVIDIA, 2011), como mostra a Figura 13(b). Os blocos também são agrupados em *grids*.

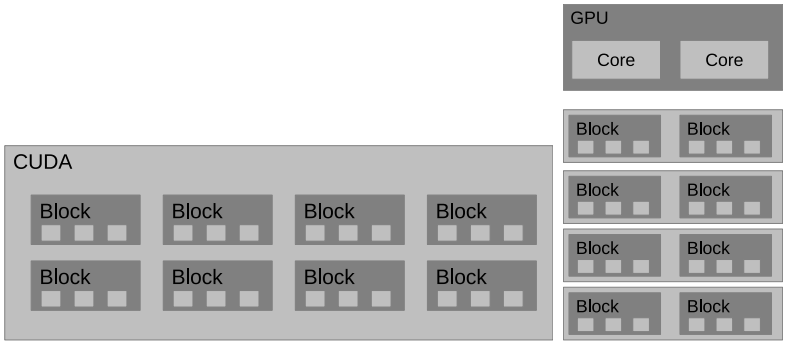


Figura 13: (a) Divisão em blocos das *threads* em CUDA; (b) Divisão dos blocos nos núcleos em CUDA

No CUDA existem algumas restrições e uma delas é que o total da multiplicação da quantidade de *threads* por blocos de cada dimensão, não pode ultrapassar a quantidade total de *threads* por blocos suportadas pela placa (Equação 48).

$$T_T >= T_1 * T_2 * T_3 \tag{48}$$

onde T_T é a quantidade de total de *threads* por bloco suportada pela placa e T_i (i = 1..3), é a quantidade de *threads* por bloco da dimensão i. Outra restrição é que a quantidade de blocos por *grid* por dimensão não pode ultrapassar a quantidade de blocos por *grid* dessa dimensão suportada pela placa (Equação 49).

$$G_i >= g_i \tag{49}$$

onde G_i representa a quantidade de blocos por *grid* da dimensão i (i = 1, 2 ou 3) suportada pela placa, e g_i representa a quantidade de blocos por *grid* da dimensão i definida no programa desenvolvido.

2.5.2 Open Computing Language (OpenCL)

OpenCL é um padrão aberto para programação paralela de plataformas heterogêneas, que fornece um acesso de alto e baixo nível para processos em dispositivos paralelos. Funciona tanto em CPU's, quanto em GPU's de vários fabricantes (AUGUSTO; BARBOSA, 2012; NVIDIA, 2009a). OpenCL é também um framework que inclui uma linguagem, bibliotecas e uma API (GROUP, 2011). Cada dispositivo no OpenCL possui p *computing unit* (CU), que por sua vez é composta por q *processing elements* (PE), como mostra a Figura 14 (AUGUSTO; BARBOSA, 2012).

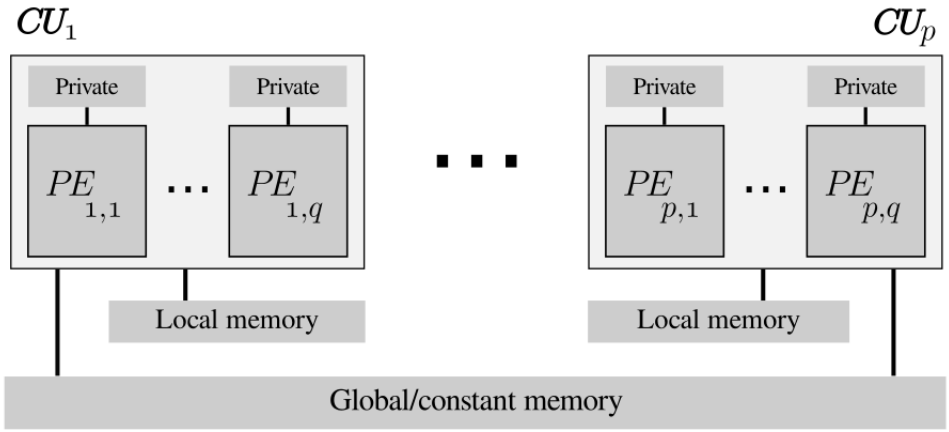


Figura 14: Conceito da arquitetura de dispositivos (AUGUSTO; BARBOSA, 2012).

A Figura 15 mostra como o OpenCL realiza a paralelização na GPU. Dado valores para X e Y, a multiplicação de entre eles $(X * Y)$ determina a quantidade de *work-itens* (processos ou *threads*) que irão existir na placa. Ao agrupar uma quantidade p de *work-itens*, determina-se um *work-group*, que é executado utilizando a arquitetura SIMD. Essa divisão pode ser realizada entre uma e três dimensões, sendo obrigatório que o resultado da multiplicação da quantidade de *work-groups* de cada dimensão (Equação 50), não deve ultrapassar a quantidade de *work-groups* máxima da placa (Equação 51), sendo que essa quantidade é fornecida pela própria placa em uso.

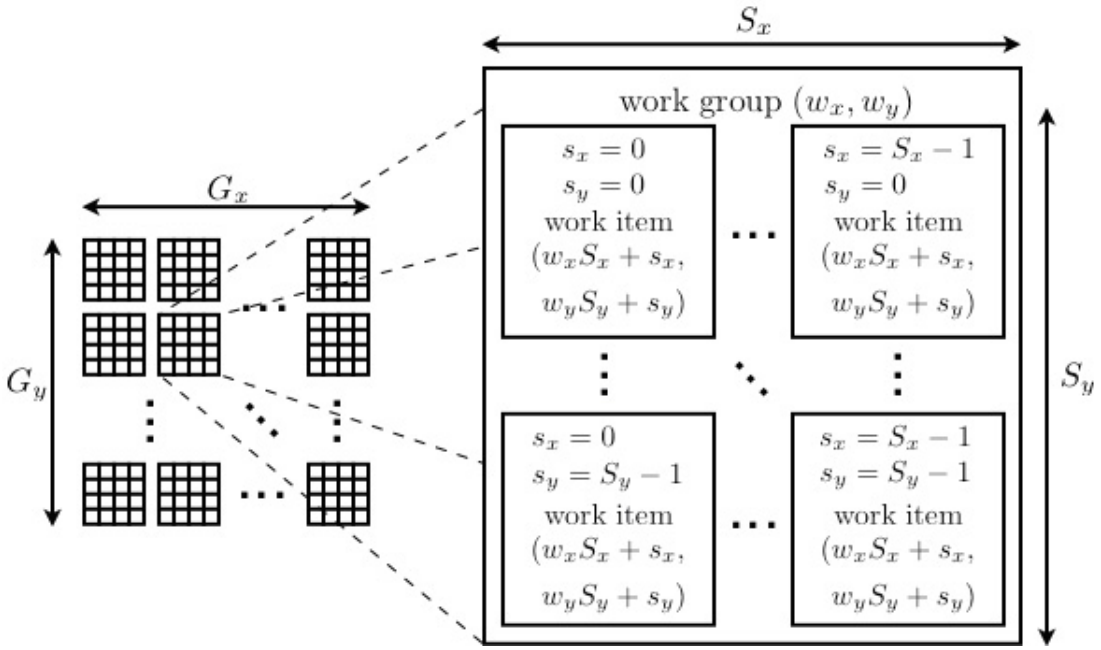


Figura 15: *Work-itens* e *work-group* do OpenCL.

$$WG_t = D_1 * D_2 * D_3 \tag{50}$$

onde, WG_t representa a quantidade de *work-groups* total e D_i representa a quantidade de *work-groups* daquela dimensão.

$$WG_t <= WG_{MAX} \tag{51}$$

onde WG_{MAX} representa a quantidade de *work-groups* suportada pela placa de vídeo. Outra obrigatoriedade é que a quantidade de *work-itens* de cada dimensão deve ser múltipla da quantidade de *work-group* da dimensão em questão (Equação 52), ou seja, a divisão deve resultar em um número inteiro $\mathbb{Z}$.

$$\mathbb{Z} = \frac{WI_i}{WG_i} \tag{52}$$

onde WI_i representa a quantidade de *work-itens* da dimensão $i(i = 1,\ 2\ ou\ 3)$. Com essa estrutura pode-se dividir tarefas em tarefas menores, que são executadas simultaneamente em *work-itens* diferentes, gerando ganho de desempenho para resolução do problema aplicado.

Capítulo 3

MATERIAL E MÉTODOS

Neste capítulo são apresentados os locais de origem dos dados de estudo, bem como as características dos mesmos e os equipamentos utilizados em cada um deles. É descrito também como foi todo o desenvolvimento do ambiente proposto, as linguagens utilizadas e a definição da arquitetura, além de descrever quais e como foram os testes realizados no ambiente e as estatísticas utilizadas no presente trabalho.

3.1 LOCAL DE ORIGEM DOS DADO DE ESTUDO

Os dados utilizados neste trabalho foram obtidos de quatro localidades distintas no estado de Mato Grosso que apresentam características climáticas semelhantes, com temperatura média anual entre $24,9$ a $25,4°C$, precipitação entre 1300 a 1400 $\frac{mm}{ano}$, uma estação seca entre abril e setembro e uma chuvosa entre outubro e março, sendo a quarta área experimental com precipitação média anual de 2037 mm (VOURLITIS et al., 2011). As descrições das áreas são:

- Fazenda Miranda (FM) – com coordenadas 15°43′53.65″S e 56°4′18.88″O e altitude de 157 m, no município de Cuiabá – MT. Essa área é caracterizada por uma pastagem, com dominância da vegetação herbácea que surgiu depois da derrubada parcial da vegetação original, contendo fragmentos que conservam as características de Cerrado.

- Fazenda Experimental (FE) da Universidade Federal de Mato Grosso – com coordenadas 15°47′11″S e 56°4′47″O e altitude de 140 m, no município de Santo Antônio do Leverger – MT, distante 33 km de Cuiabá – MT. Essa área é caracterizada por uma pastagem de *Brachiaria humidicola*.

- Reserva Particular do Patrimônio Natural - RPPN SESC Pantanal (CAM) – com coordenadas 16°39′50″S e 56°47′50″O e altitude de 120 m, no município de Barão de Melgaço – MT, distante 160 km de Cuiabá – MT. Essa área apresenta vegetação monodominante de Cambará (*Vochysia divergens Pohl*), conhecida localmente como cambarazal, com altura do dossel variando entre 28 a 30 m e em uma forma de faixa contínua de aproximadamente 25 km de extensão e 4 km de largura, paralela ao rio Cuiabá.

- A quarta área é localizada a 50 km de Sinop, Mato Grosso, Brasil, na Fazenda Macaraí (SINOP) – com coordenadas 11°24′43.4″S e 55°19′25.7″O e altitude de 435 m. No local, havia uma torre micrometeorológica de 42 m de altura. Essa região é de transição entre a Amazônia e o Cerrado, definida como Floresta Tropical Semidecídua, com árvores em torno de $25 - 28$ m de altura. O *índice de área foliar* (IAF) varia de $5 - 6$ $m^2 m_{-2}$ na estação úmida e de $2 - 2,5$ $m^2 m_{-2}$ na estação seca Vourlitis et al. (2002), Sanches et al. (2008).

A Figura 16 apresenta no mapa do estado de MT, a localização das quatro áreas de onde foram retirados os dados utilizados nesse trabalho.

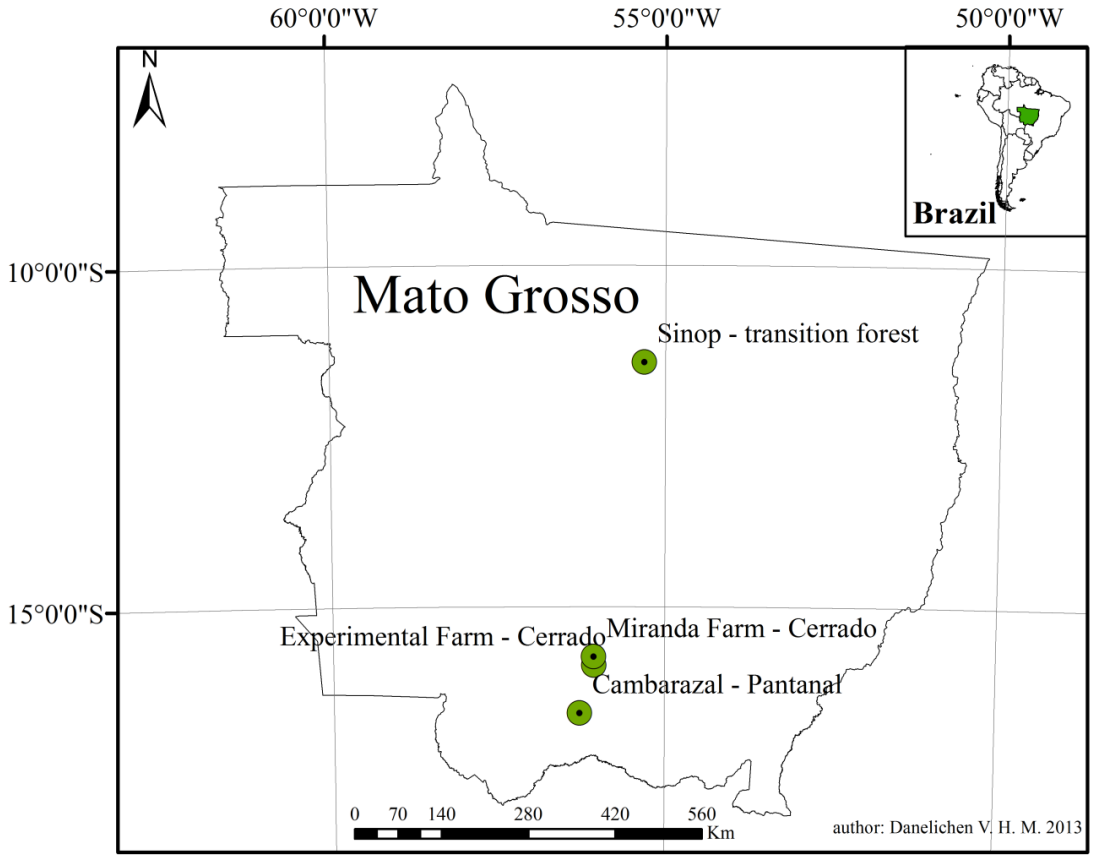


Figura 16: Localização das áreas de estudos, onde a floresta de transição é designada como SINOP, cerrado como Fazenda Miranda – FM e Fazenda experimental – FE e a floresta de cambarazal – CAM.

3.2 INSTRUMENTAÇÃO MICROMETEOROLÓGICA

O *saldo de radiação* foi medido por meio de um saldo radiômetro (Net Radiometer, Kipp & Zonen Delft, Inc., Holland), e a *radiação solar incidente* através de um piranômetro (LI-200, Campbell Sci, Inc., USA) a 33 m de altura no cambarazal (CAM), 42 m na floresta de transição (SINOP) e a $2,5$ m nas áreas de pastagem (FE e FM). O *fluxo de calor no solo* foi obtido por meio de dois fluxímetros de calor no solo no cambarazal, um em SINOP (HFT-3.1, REBS, Inc., Seattle, Washington) e um fluxímetros de calor no solo nas áreas de pastagem a 2 cm de profundidade. Os *gradientes de temperatura e umidade do ar* foram estimados por meio de dois termohigrômetros (HMP 45 C, Vaisala, Inc., Helsinki, Finland) instalados a $33,7$ e $37,7$ m no cambarazal, a $40m$ na floresta de transição, e a $0,5$ e $2,8$ m nas áreas de pastagem.

No cambarazal, os equipamentos foram alimentados com tensão de 12 V por duas baterias de 150 Ah, carregadas por um painel solar de 64 W com regulador de tensão, e nas áreas de pastagem todos os equipamentos foram alimentados com tensão de 12 Vpor uma fonte AC/DC de 10 A, alimentada pela rede de corrente alternada de 127 V. Nas duas áreas, os dados produzidos por sinais e pulsos elétricos dos transdutores foram processados e armazenados por um datalogger (CR 10X, Campbell Scientific, Inc., Ogden, Utah), com médias de 15 minutos. Para aumentar o número de canais de entrada do registrador foi utilizada uma placa multiplexadora (AM16/32A-ST-SW, Campbell Scientific, Inc., Ogden, Utah). Na torre da floresta de transição os dados produzidos por sinais dos transdutores foram processados em uma frequência de 10 Hz e armazenados a média desses dados a cada 30 minutos por um datalogger (CR5000, Campbell Scientific, Inc., Logan, UT, USA). A flutuação da velocidade tridimensional do vento foi medida por meio de um anemômetro sônico (CSAT-3, Campbell Scientific, Inc., Logan, UT, USA).

3.3 DESENVOLVIMENTO DO AMBIENTE GENESE

Para o desenvolvimento do ambiente foi necessário utilizar diversas ferramentas computacionais, sendo a linguagem Java escolhida, como linguagem hospedeira, juntamente com a plataforma de desenvolvimento NetBeans. Para as linguagens paralelas foram escolhidas o CUDA, utilizando a biblioteca JCUDA (JCUDA, 2012) para comunicação com o Java e a linguagem OpenCL, utilizando a biblioteca JOCL (JOCL, 2012), para integração com o mesmo, neste caso tanto para GPU quanto CPU.

Para realizar a integração das linguagens paralelas com o ambiente desenvolvido foi necessário utilizar a framework JEDIGPU, também desenvolvida pelo autor deste trabalho, que representa uma framework voltada especificamente para automatizar todas as configurações, além de executar os códigos paralelos tanto do CUDA quanto do OpenCL (GOMES, 2012). Assim, a JEDIGPU é uma intermediária entre o ambiente desenvolvido e as plataformas paralelas, utilizando o componente da framework JSeriesCUDA para o CUDA e JSeriesCL para o OpenCL.

A Figura 17 mostra todas as ferramentas utilizadas para desenvolver o ambiente, denominado de GeneSE (Generic Surface Energy). Além do ambiente foi desenvolvido uma interface com usuário (GUI) para testar o uso do ambiente.

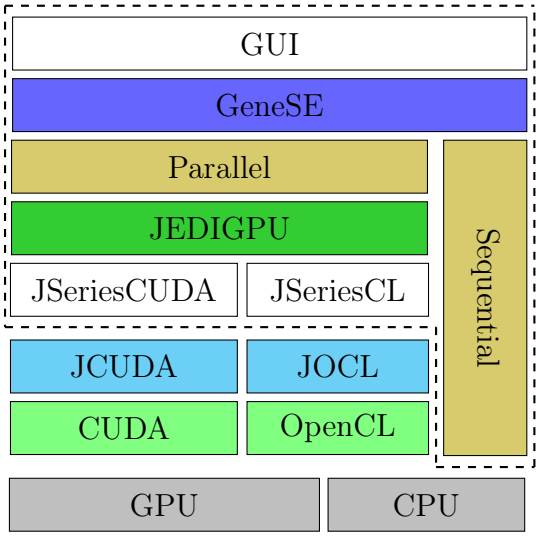


Figura 17: Arquitetura do ambiente GeneSE

3.3.1 Testes

Para a realização dos testes foi utilizado um conjunto com 85 imagens da região de Mato Grosso, Brasil. Foi definido também um conjunto de 8 algoritmos, sendo 4 para calcular somente os índices de cada algoritmo automatizado e 4 para calcular até a evapotranspiração. Todos os cálculos e imagens foram processados em 4 GPUs diferentes (Tabela 3) e também em quatro CPU diferentes (Tabela 4), onde cada imagem foi executada em quatro linguagens diferentes, resultando em um total de 10880 imagens processadas.

Tabela 3: Configurações das GPU das máquinas testadas.

Nome	Núcleos	Frequência (MHz)
GT520m	48	810
GT730m	384	725
GTX560Ti	384	822
GT220	2 x 48	625

Tabela 4: Configurações das CPU das máquinas testadas.

Nome	Núcleos	Frequência (GHz)
i5	2	2,3
i5	2	2,5
i7	4	3,4
i7	4	2,6

Os testes foram divididos em 2 partes:

- Teste 1: foi medido o tempo gasto para o ambiente criar o código e compilá-lo, afim de verificar a influência desse tempo no total. A matriz de pixels de cada imagem teve seu tamanho variado de $1000x1000$ até $6000x6000$ pixels, acrescentando sempre 1000 pixels de cada lado da imagem.
  - Criação: Tempo gasto pelo ambiente para criar o código a ser compilado e executado
  - Compilação: Tempo gasto pelo ambiente para compilar o código criado
- Teste 2: realizou as medidas de tempo de processamento e de quantidade de memória ocupada para variações no tamanho da imagem. Sendo que esse tamanho variou de $1000x1000$ até $6000x6000$ pixels, acrescentando sempre 1000 pixels de cada lado da imagem. Para cada algoritmo e para cada plataforma, foram armazenados o *tempo de execução* (ms) e a *memória ocupada* (MB) para cada teste.

3.3.1.1 Validação

Após a execução dos testes realizou-se uma validação dos dados obtidos com os dados disponíveis de campo, sendo utilizado os indicadores: *acurácia* - índice de Willmott (d) (Equação 53); *erro quadrado médio* R̈MSE¨ (Equação 54) e o *erro absoluto médio* M̈AE¨ (Equação 55). A *acurácia* está relacionada com a distância dos valores estimados com os observados. Matematicamente, esta abordagem é dada por um índice que pode ser amplamente aplicado para a comparação entre os modelos (WILLMOTT et al., 1985). Seus valores variam de zero, sem relação, a 1 com perfeita relação.

$$d = 1 - \left[\sum (P_i - O_i)^2 / \sum (|P_i - O| + |O_i - O|)^2\right] \tag{53}$$

onde P_i é o valor estimado, O_i o valor observado e O é a média dos valores observados. O $RMSE$ indica como o modelo falha para estimar a variabilidade das medições em torno da média e mede a variação nos valores estimados em torno dos valores medidos (WILLMOTT; MATSSURA, 2005). O limite inferior de $RMSE$ é 0, o que significa que existe total conformidade entre as estimativas e medições modelo.

$$RMSE = \sqrt{\frac{\sum (P_i - O_i)^2}{n}} \tag{54}$$

O MAE indica a distância (desvio) dos valores médios absolutos estimados a partir dos valores medidos. É ideal que os valores de MAE estejam próximos de 0 (WILLMOTT; MATSSURA, 2005).

$$MAE = \frac{\sum |P_i - O_i|}{n} \tag{55}$$

Capítulo 4

RESULTADOS

Neste capítulo são apresentadas as definições do desenvolvimento do ambiente, bem como todas as modificações necessárias em outros componentes para automatizar toda a execução dos códigos gerados pelo GeneSE. São apresentados os impactos computacionais gerados pelo ambiente com enfoque principal nos resultados dos testes em relação ao tempo médio gasto para cada plataforma e algoritmo, bem como a média de utilização da memória dos computadores testados.

4.1 DESENVOLVIMENTO DO AMBIENTE GENESE

Como parte dos resultados da dissertação de mestrado desenvolvida pelo autor deste trabalho foi implementada toda a automatização para configuração e execução automática para GPU tanto em OpenCL quanto em CUDA (GOMES, 2012), contudo alguns aprimoramentos foram necessários para o desenvolvimento do ambiente GeneSE, sendo esse aprimoramento descrito a seguir.

4.1.1 Aprimorando JSeriesCL

O primeiro aprimoramento realizado para JSeriesCL (GOMES, 2012), que é o componente para execução de códigos em OpenCL da framework JEDIGPU, foi a adição do suporte do OpenCL para CPU, seguindo os mesmos princípios utilizados para GPU, pois a linguagem OpenCL é para plataformas heterogêneas, assim flexibilizou-se em que tipo de plataforma deseja-se executar o código.

Outro aprimoramento foi a determinação automática dos *work-groups* e *work-items* (Seção 4.1.1.1) do OpenCL, descrito a seguir (GOMES et al., 2014).

A Figura 18 apresenta a estrutura da classe *ParameterGPU* que é utilizada na determinação dos parâmetros utilizados pela JSeriesCL.

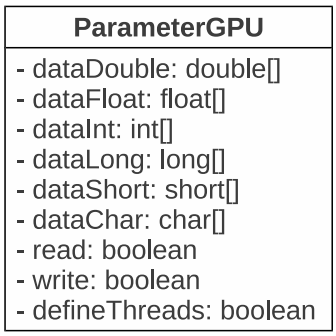


Figura 18: Classe *ParameterGPU*

Os atributos *dataDouble*, *dataFloat*, *dataInt*, *dataLong*, *dataShort* e *dataChar* são utilizados para enviar e informar qual tipo de dado será utilizado na GPU. Já os atributos *read* e *write* especificam se o dado terá operações de leitura ou escritas, respectivamente.

JSeriesCL define o nível de paralelismo utilizando o atributo *defineThreads*. Se esse atributo estiver marcado como *TRUE*, então aquele conjunto de dados que o representa definirá uma dimensão na GPU, essa configuração é utilizada para determinar quantas dimensões irão existir na GPU, além da quantidade de dados auxiliar na determinação dos *work-groups* e *work-items*. A seguir esse processo de determinação automática é descrito.

4.1.1.1 Determinando os *work-groups* e *work-items*

A quantidade de *work-groups* e *work-items* é obtida resolvendo algumas equações. O primeiro passo considera o número de valores *TRUE* utilizados no atributo *defineThreads*, portanto, apenas três parâmetros com esse valor marcado são permitidos, um para cada dimensão da GPU disponível. O próximo passo é determinar o número de *work-groups* e *work-items* para cada dimensão. Isto é conseguido usando os sistemas de Equações 56, 57, 58 para dimensões 1, 2 e 3, respectivamente:

$$\left\{ WG_1 = WG_{MAX} \right. \tag{56}$$

$$\begin{cases} a = \dfrac{Q_1}{Q_2} \\ WG_1 = a \cdot WG_2 \\ WG_1 \cdot WG_2 \leq WG_{MAX} \end{cases} \tag{57}$$

$$\begin{cases} a & = & \dfrac{Q_1}{Q_2} \\ b & = & \dfrac{Q_1}{Q_3} \\ WG_1 & = & a \cdot WG_2 \\ WG_1 & = & b \cdot WG_3 \\ WG_1 \cdot WG_2 \cdot WG_3 & \leq & WG_{MAX} \end{cases} \tag{58}$$

onde, Q_1, Q_2 e Q_3 são as quantidades de dados nos parâmetros com o atributo *defineThreads* ativado. Os valores de WG_1, WG_2 e WG_3 representam o número de *work-groups* em cada dimensão.

A resolução é realizada diretamente para uma dimensão, em outras palavras, o número de *work-groups* será o número máximo permitido pelo dispositivo a ser executado (Equação 56). Além disso, com 2 dimensões, o sistema de Equação 57 é usado, resolvendo o sistema anterior tem-se Equação 59:

$$WG_1 \leq \sqrt{WG_{MAX} \cdot a} \tag{59}$$

No caso de 3 dimensões, o número de *work-groups* é obtido usando o sistema de Equação 58, que quando resolvido tem-se a Equação 60:

$$WG_1 \leq \sqrt[3]{WG_{MAX} \cdot a \cdot b} \tag{60}$$

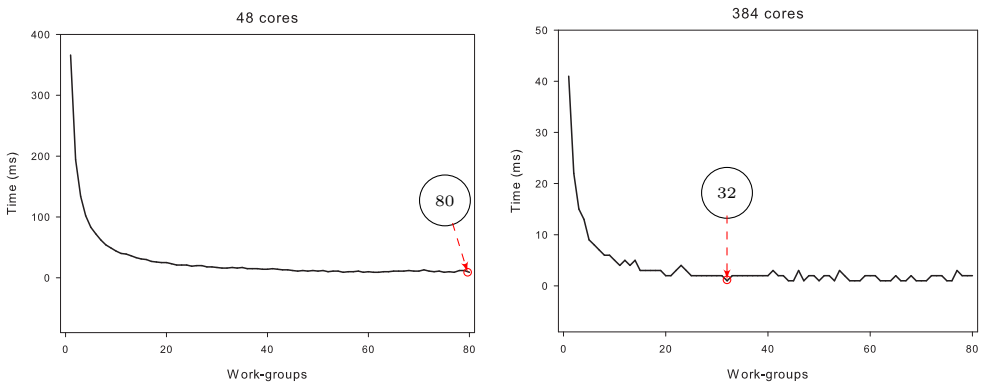
Finalmente, o número de *work-items* é determinado pela Equação 61 para cada dimensão:

$$WI_i = \left\lceil \frac{Q_i}{WG_i} + 1 \right\rceil \cdot WG_i \tag{61}$$

onde, WI_i representa o número de *work-items* na dimensão i, Q_i representa o número de dados da dimensão e WG_i é o número de *work-groups* obtidos a partir das equações acima. Portanto, todos os *work-items* calculados obedecem a Equação 52, que trata a quantidade de *work-groups* compatíveis com o dispositivo e seu respectivo *work-items*. É importante ressaltar que todas as condições estão automaticamente disponíveis no JSeriesCL, ou seja, os cálculos são feitos sem a (re)implementação do código-fonte pelo programador. Por exemplo, se o contexto exige a execução do mesmo código em dispositivos distintos, sem JSeriesCL é obrigatório realizar o cálculo do número de *work-groups* e de *work-items* para cada dispositivo e incluir esses valores no código-fonte, já com JSeriesCL não há essa necessidade, então alcança-se a portabilidade entre dispositivos distintos.

4.1.1.2 Impacto da escolha dos work-groups

A quantidade de *work-group* tem um grande impacto sobre o desempenho do dispositivo paralelo. Para demonstrar esse impacto no desempenho global, o algoritmo SEBAL foi aplicado a uma imagem com 14.400 pixels, variando o número de *work-groups* entre 1 e 80. As Figuras 19(a) e 19(b) ilustram o comportamento global de duas GPU's distintas, com 48 e 384 núcleos, em que a diferença entre o mais lento e o mais rápido, para cada GPU, foi de 97%. A Figura 19 mostra ainda a quantidade de *work-groups* com melhor desempenho para esse teste. No caso da GPU com 48 núcleos foi 80 *work-groups* e de 32 para a GPU com 384 núcleos. É importante perceber que, em alguns casos, o aumento produz uma deterioração menor no desempenho, pois isso depende da capacidade de computação da GPU (NVIDIA, 2009b).



(a) (b)

Figura 19: Impacto do número de *work-groups* sobre o desempenho da GPU, onde o eixo X representa a quantidade de *work-groups* e o eixo Y representa o tempo gasto para executar o pequeno problema: (a) mostra o resultado com uma GPU com 48 núcleos. (b) mostra o resultado com uma GPU com 384 núcleos.

É importante compreender que a política da JSeriesCL para a escolha do número de *work-groups* é a utilização da quantidade máxima dos *work-groups* suportado pelo dispositivo. Os cálculos do *work-items* e *work-groups* foram realizados automaticamente para cada contexto testado, portanto, o código fonte final, tornou-se mais generalizado, legível e portável.

A Tabela 5 mostra a realização de outros testes variando os algoritmos utilizados, exibindo a quantidade de dados disponíveis, o número de *work-groups* e o número de *work-items* para cada dimensão (valores entre colchetes). Para a verificação do funcionamento da JSeriesCL foram feitas simulações em diferentes conjuntos de dados com mais de 1 dimensão, de forma que a framework tenta

utilizar o potencial máximo da GPU, além de automatizar todas as Equações de 50 até 61.

Tabela 5: Resultado dos cálculos automatizados do JSeriesCL.

Data	Qtd_i	WG_{MAX}	WG_i	WI_i
Image (Pixels)	[14400]	576	[576]	[14976]
Image (Pixels)	[35361578]	576	[576]	[35362154]
Micro-meteorological	[140448]	1024	[1024]	[141312]
Simulate 1	[14321, 3456]	960	[64,15]	[14336, 3465]
Simulate 2	[63729, 3772]	960	[137, 17]	[63842, 3773]
Simulate 3	[231229,14321, 3456]	1024	[170, 6, 1]	[231370, 14322, 3456]
Simulate 4	[23167,7473, 1234]	1024	[42, 12, 2]	[23184, 7476, 1236]
Simulate 5	[17275, 3421, 2134]	896	[37, 6, 4]	[17279, 3426, 2136]

4.1.2 Aprimorando JSeriesCUDA

O aprimoramento realizado para CUDA, que na framework JEDIGPU é implementada pelo componente JSeriesCUDA, foi o cálculo da quantidade de *threads por bloco* e de *blocos por grid*, obedecendo ao critério de máxima ocupação na GPU descrito a seguir.

4.1.2.1 Determinando a Máxima Ocupação

A ocupação em CUDA é obtido a partir da quantidade de *warps* ativos e da quantidade máxima de *warps* (Equação 65) que a GPU suporta. *Warps* é definido como um conjunto de *threads* que são criados, gerenciados e executados pela GPU, sendo que um conjunto de 32 *threads* formam um *warp* (esse valor é denominado de *warp size* (ω)) (NVIDIA, 2011).

Todas as Equações de 62 até 74 são implementadas em uma planilha eletrônica disponibilizada pela própria NVIDIA denominada CUDA GPU Occupancy Calculator (NVIDIA, 2012). Portanto é necessário a determinação manual da quantidade de *threads por bloco* para cada código fonte, para cada GPU e para cada configuração da *compute capability* escolhida.

A Figura 20 representa um esquema das equações necessárias para se calcular a taxa de ocupação de uma GPU. Os nós redondos representam constantes que variam de GPU para GPU e da configuração da compilação da GPU, como, por exemplo, a *compute capability*. Os nós representados como diamantes representam os parâmetros do código-fonte a ser executado na GPU, assim a taxa varia de equipamento para equipamento e também para cada código-fonte. Os nós retangulares são as variáveis a serem calculadas, tendo entre parentêses o número da fórmula seguindo a listagem deste documento. O cálculo da ocupação

máxima acontece das folhas para a raiz. Cada fórmula da figura é detalhada a seguir.

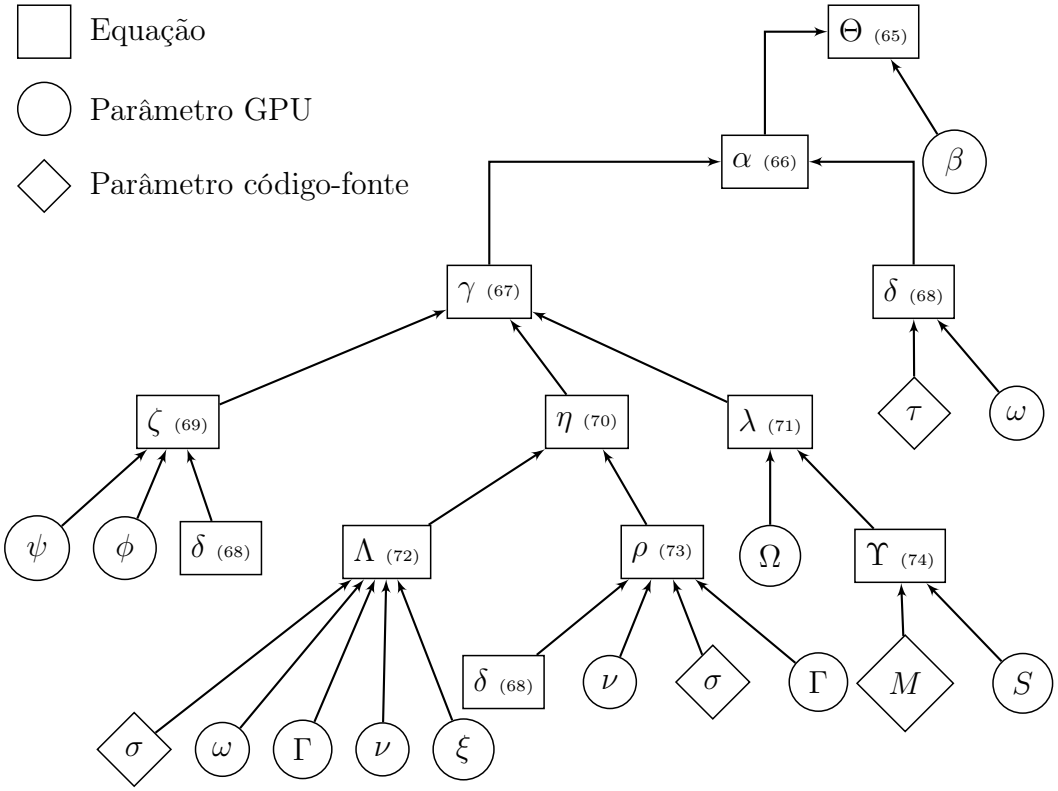


Figura 20: Diagrama que ilustra as equações necessárias para se calcular a taxa de ocupação de uma GPU, onde ' (x) ' é o número da equação listada neste documento.

Primeiramente foram definidas três operações (Equações 62, 63 e 64), a primeira encontra um valor próximo a x que seja múltiplo de y e que seja no máximo igual a x, a segunda encontra um valor próximo a x que seja múltiplo de y e que seja no mínimo igual a x e a terceira realiza a fatoração de um número retornando um vetor dos valores encontrados.

$$fm(x, y) = \left(\left\lfloor \frac{x}{y} \right\rfloor \right) * y \tag{62}$$

$$cm(x, y) = \left(\left\lfloor \frac{x}{y} \right\rfloor \right) * (y + 1) \tag{63}$$

$$fac(x) = fatora\ x \tag{64}$$

A taxa de ocupação é determinada pela variável Θ que é definida por:

$$\Theta = \frac{\alpha}{\beta} \tag{65}$$

onde, α representa a quantidade de *warps* ativos para cada multiprocessor e β representa a quantidade máxima de *warps* que um multiprocessor suporta, assim o resultado da Equação 65, varia de 0 à 1, sendo que 1 representa a GPU totalmente ocupada e 0 ao contrário. A variável α é definida como (Equação 66):

$$\alpha = \gamma * \delta \tag{66}$$

onde γ representa a quantidade *threads* ativos por multiprocessor e δ representa a quantidade *warps* por bloco que o código fonte irá ocupar. Os termos γ e δ são definidos pelas Equações 67 e 68 respectivamente.

$$\gamma = \min(\zeta, \eta, \lambda) \tag{67}$$

onde ζ representa uma taxa, que é limitada pelo número máximo de *warps* ou de blocos por multiprocessor, η representa uma taxa que é limitada pela quantidade de registros utilizadas por multiprocessor e λ representa uma taxa que é limitada pela quantidade de memória compartilhada por multiprocessor. Voltando a Equação 66, δ é definido por:

$$\delta = cm\left(\frac{\tau}{\omega}, 1\right) \tag{68}$$

onde τ representa a quantidade de *threads por bloco* definida para esta execução e ω representa o *warp size*.

Os termos da Equação 67, ζ, η, λ são definidos pelas Equações 69, 70 e 71 respectivamente.

$$\zeta = \min\left(\psi, fm\left(\frac{\phi}{\delta}, 1\right)\right) \tag{69}$$

onde ψ representa a quantidade de *thread* bloco por multiprocessor e ϕ representa o limite de *warps* por Multiprocessor que a GPU suporta.

$$\begin{cases} \sigma > \mu & \eta = Error \\ \sigma > 0 & \eta = fm\left(\frac{\Lambda}{\rho}, 1\right) \\ \sigma \leq 0 & \eta = \psi \end{cases} \tag{70}$$

onde σ representa a quantidade de registro que será utilizado por cada *thread* pelo código fonte, μ representa o limite de registros por *thread* suportado pela GPU,

Λ representa a quantidade máxima de *warps* ativos que a quantidade de registros suporta e ρ representa a quantidade de registros por bloco que será utilizado na execução.

$$\begin{cases} \Upsilon > 0 & \lambda = fm\left(\frac{\Omega}{\Upsilon}, 1\right) \\ \Upsilon \leq 0 & \lambda = \psi \end{cases} \tag{71}$$

onde Υ representa a quantidade de memória compartilhada utilizada pelo código fonte por bloco que é definida pela Equação 74, Ω representa o limite de memória compartilhada por multiprocessor (bytes).

Os termos da Equação 70 Λ e ρ são definidos pelas Equações 72 e 73 respectivamente:

$$\begin{cases} g = \text{"}block\text{"} & \Lambda = \xi \\ g <> \text{"}block\text{"} & \Lambda = fm\left(\frac{\xi}{cm\left(\sigma * \omega, \Gamma\right)}, \nu\right) \end{cases} \tag{72}$$

onde g representa a granularidade da GPU, ξ representa o limite total de registros, Γ representa o tamanho da alocação da GPU e ν representa a granularidade da alocação do *warp*.

$$\begin{cases} g = \text{"}block\text{"} & \rho = cm\left(cm\left(\delta, \nu\right) * \sigma * \omega, \Gamma\right) \\ g <> \text{"}block\text{"} & \rho = \delta \end{cases} \tag{73}$$

onde δ representa a quantidade de *warps per block* da GPU.

$$\Upsilon = cm\left(M, S\right) \tag{74}$$

onde M representa o quanto de memória compartilhada que cada *thread* irá ocupar e S representa a quantidade a ser alocada por vez.

Portanto, para determinar a ocupação da GPU é necessário três parâmetros do código fonte:

1. Threads per Block (τ)
2. My Shared Memory (M)
3. Registers (σ)

Por exemplo, se o código em CUDA do pesquisador/desenvolvedor ocupar 512 bytes de memória compartilhada (M) e 32 registros por *thread* (σ) então constrói-se a Figura 21 que mostra a variação da ocupação de acordo com a quantidade de *threads por bloco* (τ) escolhida.

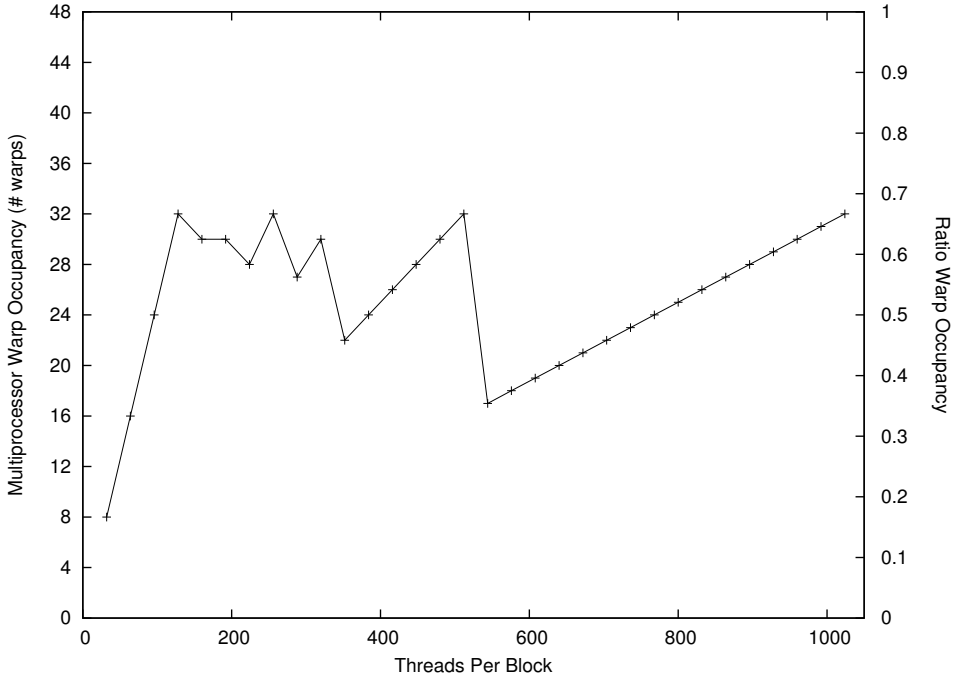


Figura 21: Impacto da variação no tamanho do blocos.

A partir dessa figura o pesquisador/desenvolvedor deve escolher qual a quantidade de *threads por bloco* que deseja utilizar na execução do seu código.

4.1.2.2 Determinando o número de *blocos por grid* e de *threads por bloco*

A quantidade de *threads por bloco* é obtida resolvendo as equações citadas anteriormente, assim a framework recebe o código fonte a ser executado e determina quantos bytes de memória compartilhada irá ocupar além de quanto registros irá precisar. Com esses dois parâmetros, calcula-se todos os pontos múltiplos de ω até o limite máximo da GPU, pois assim tem-se uma quantidade exata de quantos warps ativos a GPU irá conter. Esses dados formam um gráfico parecido com a Figura 21, então a framework escolhe a quantidade de *threads por bloco* tem a maior ocupação e também o maior número absoluto. Portanto, garante-se que a framework irá escolher o valor que irá realizar a maior ocupação da GPU. No caso da Figura 21, o valor escolhido será 1024 pois representa uma ocupação de 32 *warps* ou uma taxa de ocupação de 0,66 do total da GPU.

Com o valor τ definido, a framework irá calcular a quantidade de *threads por bloco* para cada dimensão necessária (T_1, T_2 e T_3). O primeiro passo considera o número de valores TRUE utilizados nos atributos defineThreads, então apenas três parâmetros com este valor são permitidas, um para cada dimensão da GPU

disponível. O próximo passo é determinar o número de *threads por bloco* e *blocos por grid* para cada dimensão. Isto é alcançado usando os seguintes sistemas de Equações 75, 76, 77 para dimensões 1, 2 e 3, respectivamente:

$$\begin{cases} T_1 = \tau \end{cases} \tag{75}$$

$$\begin{cases} T_1 = \omega \\ T_2 = \dfrac{\tau}{\omega} \end{cases} \tag{76}$$

$$\begin{cases} T_1 = \omega \\ T_2 = \lceil fac(\dfrac{\tau}{\omega}) \rceil \\ T_3 = \lfloor fac(\dfrac{\tau}{\omega}) \rfloor \end{cases} \tag{77}$$

onde na Equação 77 realiza-se a fatoração de $\frac{\tau}{\omega}$ e encontra-se dois valores, então T_2 irá receber o maior valor da fatoração e T_3 o menor valor da fatoração, portanto a Equação 48 fica garantida.

Finalmente, o número de *blocos por grid* é determinada pela Equação 78 para cada dimensão:

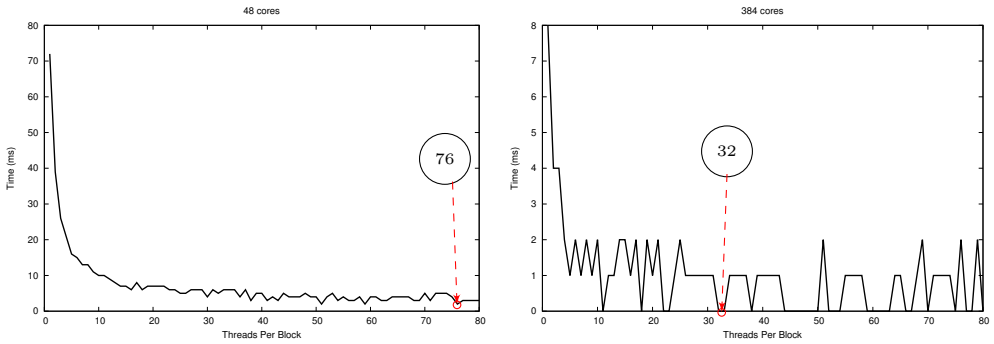
$$g_i = \left\lceil \frac{Q_i}{T_i} + 1 \right\rceil \cdot T_i \tag{78}$$

onde g_i representa a quantidade de *blocos por grid* da dimensão i, Q_i representa a quantidade de dados no parâmetro com o atributo *defineThreads* ativado e T_i representa a quantidade de *threads por bloco* da dimensão i. É importante ressaltar que todas as condições estão automaticamente disponíveis no JSeriesCUDA, ou seja, os cálculos são feitos sem a (re)implementação do código-fonte pelo programador. Por exemplo, se o contexto exige a execução do mesmo código em dispositivos distintos, sem JSeriesCUDA é obrigatório calcular os valores de *threads por bloco* e *blocos por grid* para cada dispositivo, e incluir esses valores no código-fonte, utilizando JSeriesCUDA não há essa necessidade, então alcança-se a portabilidade entre GPU's também na linguagem CUDA.

4.1.2.3 Impacto da escolha da quantidade de *threads por bloco*

O valor de *threads por bloco* tem um grande impacto sobre o desempenho da GPU. Para demonstrar como é esse impacto no desempenho global, o algoritmo SEBAL foi aplicado a uma imagem com 14.400 pixels, variando o número de *threads por bloco* entre 1 e 80. As Figuras 22(a) e 22(b) ilustram o comportamento

global de duas GPU's distintas, com 48 e 384 núcleos, em que a diferença entre o mais lento e mais rápido foi de 97% para cada GPU. A Figura 22 mostra ainda a quantidade de *threads por bloco* com melhor desempenho nesse teste. Para a GPU com 48 núcleos obteve-se uma quantidade de *threads por bloco* de 76, já para a GPU com 384 núcleos o melhor foi com 32 *threads por bloco*. É importante perceber que, em alguns casos, o aumento produz uma deterioração menor no desempenho, pois isso depende da capacidade de computação da GPU (NVIDIA, 2009b).



(a) (b)

Figura 22: Impacto do número de *threads por bloco* sobre o desempenho da GPU, onde o eixo X representa a quantidade de *threads por bloco* e o eixo Y representa o tempo dispendido para executar o pequeno problema: (a) apresenta o resultado com uma GPU com 48 núcleos. (b) mostra o resultado com uma GPU com 384 núcleos.

É importante compreender que a política definida em JSeriesCUDA para a escolha do número de *threads por bloco* é a utilização da máxima ocupação do dispositivo.

A ocupação máxima da GPU foi calculada automaticamente, alterando alguns parâmetros de compilação - como o poder da GPU computação (sm) e se usará a opção $fastmath$ (fm). O número de registros e o tamanho de memória compartilhada usada pelo algoritmo também foram fixados em valores diferentes. Com esses dados, JSeriesCUDA calcula a taxa de ocupação máxima da GPU, seguido pelo número máximo de *threads por bloco* na ocupação escolhida. Todos esses valores são apresentados na Tabela 6, mostrando que o mesmo algoritmo usado em diferentes GPU's com parâmetros de compilação distintos, podem levar a outros valores de registros e memórias compartilhadas (conforme destacado). Assim, o número de *threads por bloco* segue essa variação, afetando consequentemente a ocupação máxima.

Tabela 6: Resultado do cálculo da máxima ocupação pela JSeriesCUDA.

	GT 520m				GTX 560			
	Registros	Shared Memory	Threads	Ocupação	Registros	Shared Memory	Threads	Ocupação
Sm_10 fm	24	196	320	0.417	24	212	320	0.417
Sm_11 fm	24	196	320	0.417	24	212	320	0.417
Sm_12 fm	24	196	320	0.625	24	212	320	0.625
Sm_13 fm	27	196	512	0.500	27	212	512	0.500
Sm_20 fm	42	228	768	0.500	25	228	608	0.792
Sm_21 fm	42	228	768	0.500	27	228	576	0.750
Sm_10	25	196	320	0.417	25	212	320	0.417
Sm_11	25	196	320	0.417	25	212	320	0.417
Sm_12	25	196	320	0.625	25	212	320	0.625
Sm_13	29	196	512	0.500	29	212	512	0.500
Sm_20	46	228	640	0.417	29	228	1024	0.667
Sm_21	46	228	640	0.417	31	228	1024	0.667

A Tabela 6 também mostra que com os cálculos automatizados da JSeriesCUDA é possível utilizar dispositivos GPU diferentes, com independência de código-fonte; de outro modo, o algoritmo deve ser adaptado a cada GPU diferente utilizada.

Os cálculos de *threads por bloco* e *blocos por grid* foram realizados automaticamente para cada contexto de teste (SSEB, SEBAL, e SEBTA), portanto, o código fonte final tornou-se mais generalizado, legível e portável. A Tabela 7 mostra a quantidade de dados disponíveis (Qtd_i) para cada contexto, o número de *threads por bloco* (T_i), e do número de *blocos por grid* (g_i) para cada uma das dimensões; a coluna τ mostra o número de *threads por bloco* que representa a ocupação máxima para esta execução. Alguns conjuntos de dados foram simulados com dimensões superiores a 1 para verificar como funciona a biblioteca. A framework calcula a ocupação máxima com dimensões diferentes, como pode-se visualizar na linha em destaque da Tabela 7.

JSeriesCUDA tenta usar todo o potencial da GPU e automatiza as Equações (48, 49 e de 62 até 78) descrito neste trabalho. Assim, é possível concluir que JSeriesCUDA irá configurar a GPU para fornecer automaticamente toda a configuração necessária para executar o algoritmo de forma a obter um tempo gasto muito próximo do melhor.

Tabela 7: Resultado dos cálculos automatizados do JSeriesCUDA

Tipos de Dados	Qtd_i (dados disponíveis)	τ (threads /bloco)	T_i (τ em cada dimensão)	g_i (blocos / grid)
Image (SSEB)	[14400]	576	[576]	[14976]
Image (SEBAL)	[35361578]	1024	[1024]	[35361792]
Image (SEBTA)	[2944760]	960	[960]	[2945280]
Micro-meteor.	[140448]	1024	[1024]	[141472]
Simulate 1	[14321, 3456]	960	[32, 30]	[14353, 3486]
Simulate 2	[63729, 3772]	960	[32, 30]	[63761, 3802]
Simulate 3	[231229,14321, 3456]	1024	[32, 8, 4]	[231261, 14329, 3460]
Simulate 4	[23167,7473, 1234]	960	[32, 6, 5]	[23199, 7479, 1239]
Simulate 5	[17275, 3421, 2134]	896	[32, 7, 4]	[17307, 3428, 2138]

A partir dos aprimoramentos descritos nos componentes da framework JEDIGPU (JSeriesCL e JSeriesCUDA) foi possível desenvolver o ambiente GeneSE, que tem por objetivo facilitar o uso de sensoriamento remoto, a seguir é explicado o seu desenvolvimento.

4.1.3 Desenvolvimento da GeneSE

A Figura 23 mostra os módulos e o fluxo de processamento.

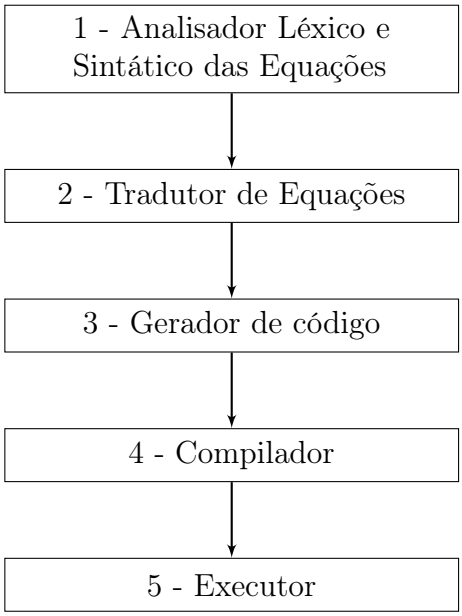


Figura 23: Etapas de processamento do ambiente GeneSE.

O ambiente GeneSE é focado em permitir a execução do processamento tanto em dispositivo GPU quanto em CPU. Outro aspecto importante é o suporte flexível de equações matemáticas, que representam a manipulação a ser realizada

sobre séries de dados. Para tanto, foram definidos alguns conceitos: *constantes*, *estruturas* e *equações*, descritos a seguir.

4.1.3.1 Definindo GeneSE

O ambiente GeneSE utiliza quatro grupos de definição:

1. **Constants**: este grupo é representado como $\Theta = \{c_1, c_2, \ldots c_n\}$, onde cada elemento c_i é um par $\langle$`constant name`, `constant value`$\rangle$ que descreve um conjunto de números reais, no qual os valores não serão alterados durante todo o processamento. Como exemplo podem ser utilizados a constante de Stefan-Boltzman e o valor de PI (π), sendo esse grupo equivalente a $\Theta = \{SB = 5.67E-8, PI = 3.1415\}$.

2. **VariableValue**: este grupo é representado como $\Psi = \{v_1, v_2, \ldots v_n\}$, onde cada elemento v_i é um par $\langle$`variable_name`, `variable_value`$\rangle$ que descreve um conjunto de estruturas compostas pelo nome da variável e seus valores, que são um conjunto de números reais. Um exemplo são as variáveis *temperatura* e *umidade relativa*, juntamente com todos os valores do ano, no qual esse grupo é equivalente a
$\Psi = \{temp = \{20, 21, 20, ..., 23\}, RH = \{70, 65, 63, ..., 40\}\}$.

3. **forVariables**: este grupo é representado como $\Omega = \{eq_1, eq_2, \ldots eq_n\}$ onde eq_i é um par $\langle$`equation name` = `equation components`$\rangle$ que descreve um conjunto de equações que serão executadas somente uma vez para todo o conjunto. Por exemplo, o cálculo da declinação solar que é realizada uma única vez para toda a imagem,
$\Omega = \{dec = radians(23.45 * sin(radians(360.0 * (DJ - 80)/365)))\}$

4. **forEachValue**: este grupo é representado como $\Phi = \{eq_1, eq_2, \ldots eq_n\}$ onde eq_i é um par $\langle$`equationname` = `equationcomponents`$\rangle$ que representa um conjunto de equações que serão executadas para cada item definido no grupo VariableValue. Por exemplo, o cálculo do índice de vegetação NDVI que é realizado para cada pixel, $\Phi = \{NDVI = (\rho_4 - \rho_3)/(\rho_4 + \rho_3)\}$

A Figura 24 ilustra quais parâmetros o GeneSE precisa para criar e executar o código.

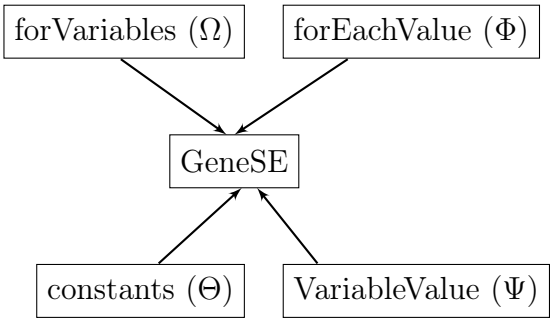


Figura 24: Definição do GeneSE

A partir das definições desses grupos que o Analisador Léxico e Sintático processa as informações.

4.1.3.2 Análise Léxica e Sintática do GeneSE

A interpretação dos parâmetros utilizados pela estrutura GeneSE realiza a análise léxica e sintática para cada grupo de parâmetros (Θ, Ψ, Ω e Φ), sendo esse processo realizado na primeira etapa de processamento, conforme ilustrado na Figura 23. Supondo o seguinte grupo de constantes (Θ):

$$\Theta = \begin{cases} a &= 1 \\ b &= 2 \end{cases} \tag{79}$$

De equações $forVariables$ (Ω) :

$$\Omega = \begin{cases} c &= a + b \end{cases} \tag{80}$$

De equações $forEachValue$ (Φ) :

$$\Phi = \begin{cases} z &= ax + b \\ y &= zx + c \end{cases} \tag{81}$$

E de dados $VariableValue$ (Ψ):

$$\Psi = \begin{cases} x &= \{25, 28, 29, 30, 31, 30, 30, 30.5\} \end{cases} \tag{82}$$

Parte-se para a etapa de criação, que realizará a análise léxica, verificando a existência dos termos nos parâmetros. Portanto a equação que resulta no valor y, terá seus termos z, x ou c, ou no conjunto de constantes (79) ou no conjunto de fórmulas do *forVariables* (80), ou em alguma fórmula anterior dentro do *forEachValue* (81), ou por fim no parâmetro VariableValue. Toda essa análise é

feita concomitantemente com a verificação da sintaxe das fórmulas, ou seja, se a estrutura obedece a escrita matemática.

Nessa etapa também é verificado quais são os resultados que o cientista /desenvolvedor deseja que seja retornado da execução, o que é ativado com o sinalizador 'O_' (de Output) no início de cada fórmula que está no *forEachValue* (Φ), portanto, caso deseje retornar o resultado da variável y basta escrever no conjunto de fórmulas do *forEachValue* o sinalizador 'O_':

$$\Phi = \begin{cases} z & = & ax + b \\ O\_y & = & zx + c \end{cases} \tag{83}$$

Um outro recurso adicionado ao ambiente é a possibilidade de executar determinada fórmula, se a mesma obedecer a alguma condicional. Por exemplo, quando a variável z for maior que 0, então a fórmula da variável y tem uma definição, caso o contrário outra. Para informar essa condição para o ambiente basta colocar _(<condição>) depois do nome da variável no conjunto de fórmulas do *forEachValue*, por exemplo:

$$\Phi = \begin{cases} z & = & ax + b \\ O\_y & = & zx + c \\ O\_y\_(z > 0) & = & z + c \end{cases} \tag{84}$$

Depois da fase de interpretação, o código-fonte é criado (segunda e terceira etapa da Figura 23). O pseudo Algoritmo 3 mostra o código fonte criado com os parâmetros $\Theta(79), \Omega(80)$, $\Phi(84)$ e $\Psi(82)$.

Algorithm 3 – Pseudo algoritmo para $\Theta(79), \Omega(80)$, $\Phi(84)$ e $\Psi(82)$.

```
1: a ← 1
2: b ← 2
3: c ← a + b
4: while tem item em 'x' do
5:     z ← a * x + b
6:     y[i] ← z * x + c
7:     if z > 0 then
8:         y[i] ← z + c
9:     end if
10: end while
11: return y
```

4.1.3.3 Compilação e Execução do GeneSE

Após a fase de criação de código segue-se a fase de compilação (quarta etapa da Figura 23), onde o código Java hospedeiro que utiliza OpenCL ou CUDA é compilado em tempo de execução. O código gerado do OpenCL ou CUDA é então passado como um parâmetro para a framework JEDIGPU (GOMES, 2012).

Todos os parâmetros $VariableValue$ (Ψ) são utilizados pelo código criado como um conjunto hash $\langle string, float[]\rangle$, que representa as equações marcadas com $'O\_'$ no conjunto $forEachValue$ (Φ). Após a compilação entra na última etapa do GeneSE (Figura 23) que é simplesmente executar o código gerado e retornar a resposta para quem chamou-a.

Contudo, para adicionar os algoritmos SEBAL, SEBTA, SSEB e S-SEBI ao ambiente foi necessário primeiramente automatizá-los, assim na seção a seguir todo o processo de automatização de cada um deles é descrito.

4.1.4 Automatização dos Algoritmos SEB

Para incluir cada um dos algoritmos utilizados pelo ambiente o primeiro passo foi automatizar os processos que eram manuais em cada um deles e adicioná-los.

4.1.4.1 SEBAL

A automatização do algoritmo SEBAL consistiu em encontrar o pixel "quente" e o pixel "frio", onde esses pixels representam o maior e o menor valor para o índice escolhido. A partir disso, calcula-se os valores de Rn, G e SAVI para o pixel "quente". A partir daí o processo iterativo descrito em Allen et al. (2011) é ativado para a determinação de "a" e "b" da equação:

$$dT = aT_s + b \tag{85}$$

Com os coeficientes (a e b), a Equação 23 é resolvida e o processo de cálculo da ET concluído, assim o ambiente incorporou todo o processo iterativo, como também flexibilizou qual índice utilizar para determinar os pixels âncoras, basta o pesquisador, no conjunto de equações enviado para o ambiente, determinar qual índice ele deseja. Abaixo, segue um exemplo do conjunto Φ inserido no ambiente GeneSE para execução do cálculo de sensoriamento remoto, utilizando o SEBAL no ambiente:

$$\Phi = \begin{cases} rad\_espectral & = & coef\_calib\_a + ((coef\_calib\_b - coef\_calib\_a)/255.0) * pixel \\ reflectancia & = & (pi * rad\_espectral)/(irrad\_espectral * cosZ * dr) \\ albedo & = & (sumBandas - reflectanciaAtmosfera)/(transmissividade * transmissividade) \\ NDVI & = & (bandaRefletida4 - bandaRefletida3)/(bandaRefletida4 + bandaRefletida3) \\ SAVI & = & ((1.0 + L) * (bandaRefletida4 - bandaRefletida3))/(L + bandaRefletida4 + bandaRefletida3) \\ IAF & = & (-ln((0.69 - SAVI)/0.59)/0.91) \\ emissividadeNB & = & 0.97 + 0.0033 * IAF \\ emissivity & = & 0.95 + 0.01 * IAF \\ T_s & = & K2/ln(((emissividadeNB * K1)/banda6) + 1.0) \\ LWd & = & emissivity * StefanBoltzman * (pow(T_s, 4)) \\ Rn & = & ((1.0 - albedo) * SWd) + (emissivity * (LWdAtm) - LWd) \\ G_0 & = & Rn * ((T_s - T_0) * (0.0038 + 0.0074 * albedo) * (1.0 - 0.98 * NDVI * NDVI * NDVI * NDVI)) \\ sebal & = & T_s \\ LE & = & Rn - H - G_0 \\ evap\_fr & = & LE/(Rn - G_0) \\ Rn_{24h} & = & Rg_{24h} * (1.0 - albedo) - 110.0 * Tao_{24h} \\ O\_ET_{24h} & = & (evap\_fr * Rn_{24h} * 86.4)/2450.0 \end{cases} \quad (86)$$

Portanto, para informar ao ambiente que deseja-se utilizar a automatização SEBAL basta adicionar uma equação da seguinte forma:

$$sebal =< equacao > \quad (87)$$

onde $< equacao >$ representa a equação utilizada pelo ambiente para calcular o índice, e escolher com base nele os pixels âncoras, com isso o ambiente GeneSE fará todo o processo do cálculo do índice e adicionará o valor de "H" no escopo dos cálculos automaticamente, podendo o pesquisador utilizá-lo nas suas equações seguintes, como observa-se pela equação de LE no conjunto Φ:

$$LE = Rn - H - G_0 \quad (88)$$

Todo esse processo foi implementado no ambiente GeneSE para executar de forma sequencial, quando utilizada a linguagem Java e também de forma paralela para CUDA e OpenCL para GPU e CPU, sendo que a plataforma de execução é escolhida pelo pesquisador. A paralelização nas escolhas dos pixels utilizando os índices foi embutida no ambiente, deixando a critério do cientista qual equação deseja utilizar junto com o paralelismo, sendo toda a implementação realizada da seguinte maneira:

- Para cada imagem foi definido uma *thread* para cada linha, assim se uma imagem tem 3000 linhas, tem-se 3000 *threads* executando, sendo que cada *thread* irá buscar os pixels âncoras dentro das colunas da sua respectiva linha.

- Após essa execução tem-se então 3000 pixels quente e 3000 frio.
- Dentro dos 3000 valores são encontrados o menor e o maior valor para o índice determinado
- A partir daí o processo iterativo é ativado e finaliza-se o algoritmo

4.1.4.2 SEBTA

Para esse algoritmo a automatização seguiu o mesmo princípio do SEBAL, deixando em aberto o índice que deseja-se utilizar para o eixo X (Figura 4), mas obrigando a utilização da T_s para o eixo Y, pois o intuito do SEBTA é encontrar uma relação entre algum índice de vegetação com a temperatura de superfície. Assim para o pesquisador utilizar a técnica do SEBTA dentro do ambiente basta substituir na Equação 86, a linha que contém a Equação 87 por:

$$sebta =< equacao > \tag{89}$$

Portanto, se o pesquisador desejar utilizar o índice NDVI em vez do índice mSAVI, que é o padrão do SEBTA, basta definir dentro do conjunto Φ (Equações 86) a equação:

$$sebta = NDVI \tag{90}$$

No exemplo do conjunto Φ, a equação do NDVI já foi definida, podendo utilizar diretamente a variável, ou é possível colocar a equação da mesma ficando da seguinte forma:

$$sebta = \frac{bandaRefletida4 - bandaRefletida3}{bandaRefletida4 + bandaRefletida3} \tag{91}$$

Todo esse processo foi implementado no ambiente para executar de forma sequencial, quando utilizando a linguagem Java e também de forma paralela para CUDA e OpenCL para GPU e CPU e a paralelização do índice seguiu o mesmo princípio do desenvolvido no SEBAL.

4.1.4.3 SSEB

A automatização da técnica SSBEB consistiu em encontrar os 3 pixels com os menores valores para o índice escolhido, e calculando a média das suas respectivas T_s, denominado de pixel "quente", e os 3 pixels com os maiores valores para o índice, calculando a média das suas respectivas T_s, denominado pixel "frio". Abaixo está um exemplo de conjunto Φ que utiliza a técnica do SSEB para seus cálculos:

$$\Phi = \begin{cases} rad\_espectral & = & coef\_calib\_a + ((coef\_calib\_b - coef\_calib\_a)/255.0) * pixel \\ reflectancia & = & (pi * rad\_espectral)/(irrad\_espectral * cosZ * dr) \\ albedo & = & (sumBandas - reflectanciaAtmosfera)/(transmissividade * transmissividade) \\ NDVI & = & (bandaRefletida4 - bandaRefletida3)/(bandaRefletida4 + bandaRefletida3) \\ SAVI & = & ((1.0 + L) * (bandaRefletida4 - bandaRefletida3))/(L + bandaRefletida4 + bandaRefletida3) \\ IAF & = & (-ln((0.69 - SAVI)/0.59)/0.91) \\ emissividadeNB & = & 0.97 + 0.0033 * IAF \\ emissivity & = & 0.95 + 0.01 * IAF \\ T_s & = & K2/ln(((emissividadeNB * K1)/banda6) + 1.0) \\ LWd & = & emissivity * StefanBoltzman * (pow(T_s, 4)) \\ Rn & = & ((1.0 - albedo) * SWd) + (emissivity * (LWdAtm) - LWd) \\ G_0 & = & Rn * ((T_s - T_0) * (0.0038 + 0.0074 * albedo) * (1.0 - 0.98 * NDVI * NDVI * NDVI * NDVI)) \\ sseb & = & NDVI \\ evap\_fr & = & (T_H - T_s)/(T_H - T_C) \\ H & = & (1 - evap\_fr) * (Rn - G_0) \\ LE & = & (evap\_fr) * (Rn - G_0) \\ Rn_{24h} & = & Rg_{24h} * (1.0 - albedo) - 110.0 * Tao_{24h} \\ O\_ET_{24h} & = & (evap\_fr * Rn_{24h} * 86.4)/2450.0 \end{cases} \tag{92}$$

Da mesma maneira que os outros algoritmos, após os cálculos do SSEB o ambiente já disponibiliza no escopo das equações as variáveis T_H e T_C, definidas internamente no ambiente para o SSEB, como pode ser observado na equação a seguir, presente no conjunto Φ:

$$evap\_fr = \frac{T_H - T_s}{T_H - T_C} \tag{93}$$

O processo de paralelização do índice no GeneSE foi definido de forma similar aos anteriores, é criada uma *thread* para cada linha, contudo neste caso são criadas também mais 3 *threads* para cada linha criada, que representa os 3 maiores valores e os 3 menores valores do índice, assim há uma paralelização de duas dimensões tanto em linhas, quanto em colunas.

4.1.4.4 S-SEBI

A automatização do algoritmo S-SEBI consistiu em encontrar os 4 pontos extremos da Figura 5 denominados:

- (x_1, y_1): ponto médio inferior para o índice e para a T_s;

- (x_2, y_2): ponto médio superior para o índice e para a T_s;
- (x_3, y_3): ponto médio inferior para o índice e superior para a T_s;
- (x_4, y_4): ponto médio superior para o índice e inferior para a T_s;

No qual cada ponto médio representa a média entre 10 valores da sua respectiva região. Após o cálculo desses pontos, duas equações de reta são calculadas, uma utilizando os pontos (x_1, y_1) e (x_4, y_4) e outra utilizando os pontos (x_2, y_2) e (x_3, y_3), assim as equações 43 e 44 são resolvidas e o processo concluído até a ET.

Para utilizar esse recurso dentro do conjunto Φ escolhido basta colocar a equação:

$$ssebi =< equacao > \tag{94}$$

onde $< equacao >$ pode ser qualquer parâmetro já calculado anteriormente ou uma nova equação, assim o ambiente irá calcular os valores de T_H e T_{LE} e deixá-los disponíveis no escopo das equações podendo ser utilizados logo em seguinte como a seguir:

$$evap\_fr = \frac{(a_H + b_H * \alpha - T_s)}{(a_H - a_{LE} + (b_H - b_{LE}) * \alpha)} \tag{95}$$

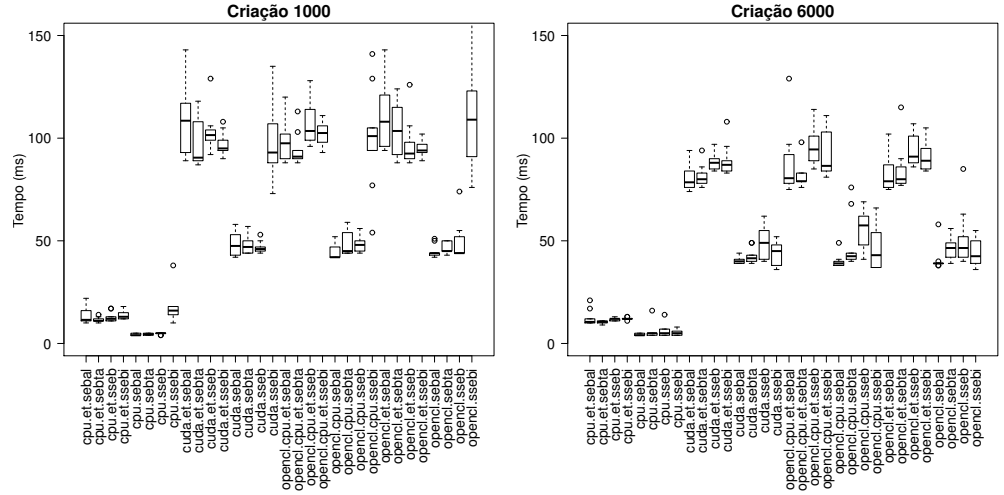
O processo de paralelização do índice seguiu o mesmo princípio do desenvolvido no SSEB, contudo em vez de criar somente 3 *threads* por linha foram criados 10, pois no caso do SSEB a técnica realiza a média de 3 pixels, já a automatização do S-SEBI procura pelos pontos médios, ou seja, uma região média, assim 10 pixels podem representar melhor uma pequena região da Figura 5.

Após a implementação de todos os algoritmos dentro do GeneSE, foi realizado os testes e seus resultados estão descrito a seguir.

4.2 IMPACTO DO AMBIENTE NO DESEMPENHO (TESTE 1)

A Figura 25 mostra o resultado dos testes para o tempo de criação de código para cada linguagem e para cada algoritmo implementado no ambiente,

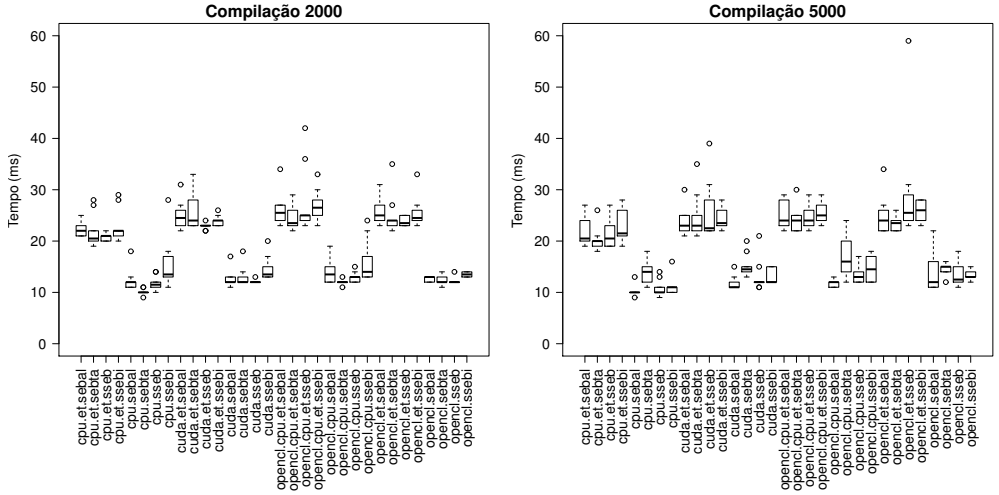
em imagens com tamanho de 1000x1000 e 6000x6000. Observa-se na figura que tempo para a criação dos códigos para as plataformas paralelas (CPU ou GPU), é um pouco maior se comparado com a plataforma sequencial (CPU). Essa diferença se deve ao fato da implementação nas plataformas paralelas possuírem um custo de desenvolvimento maior, além da necessidade de realizar mais verificações do que na sequencial. Outra observação é que o tamanho da imagem não alterou o comportamento e a média, mostrando que o ambiente não é interferido pela quantidade de dados no quesito de criação do código.



(a) (b)

Figura 25: Tempo gasto de criação de código para cada linguagem e algoritmo, com uma imagem de tamanho $1000x1000$ (a) e $6000x6000$ (b).

A Figura 26 mostra o tempo gasto para compilar o código gerado para cada linguagem e algoritmo utilizado, em imagens com tamanho de $2000x2000$ e $5000x5000$. Observa-se que nesse aspecto não houve diferença entre as plataformas e o tamanho das imagens, contudo houve diferença entre algoritmos, o que era esperado, pois os algoritmos com o termo "et" são constituídos dos algoritmos sem "et".

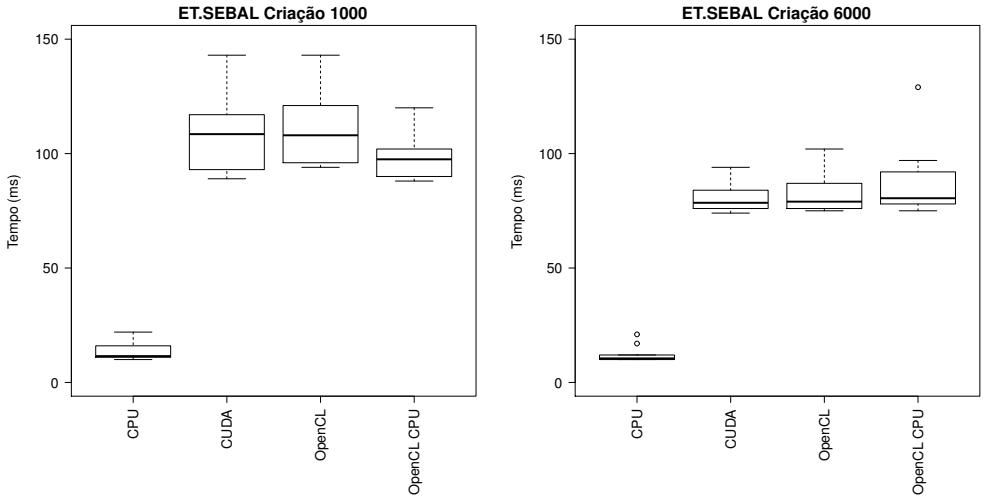


(a) (b)

Figura 26: Tempo gasto de compilação do código para cada linguagem e algoritmo, com uma imagem de tamanho $2000x2000$(a) e $5000x5000$ (b).

Ao analisar as Figuras 25 e 26 observa-se uma média de 150 ms de tempo gasto entre criação de código e compilação dos mesmos, isso mostra que os processamentos adicionais do ambiente não trazem grande impacto no tempo total de execução.

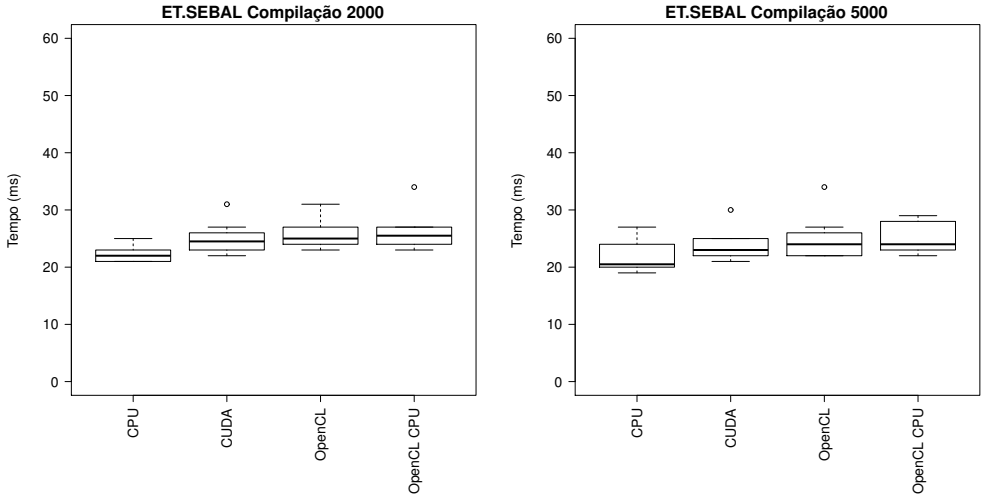
As Figuras 27(a) e 27(b) mostram os tempos de criação para um algoritmo específico, no caso do ET.SEBAL (evapotranspiração pelo SEBAL), para cada plataforma utilizada, com imagens de tamanho diferentes. Observa-se pela Figura 27 que o tempo de criação tem uma diferença somente para a plataforma CPU, pois trata-se de um algoritmo sequencial, enquanto os outros são algoritmos paralelos. Outra informação é a não diferença de tempo quando o tamanho das imagens são comparados, reforçando a independência do ambiente com a quantidade de dados.



(a) (b)

Figura 27: Tempo gasto para criação do algoritmo de evapotranspiração pelo SEBAL, em imagens de 1000x1000 (a) e 6000x6000 (b).

As Figuras 28(a) e 28(b) mostram os tempos de compilação para um algoritmo do ET.SEBAL (evapotranspiração pelo SEBAL), para cada plataforma utilizada, com imagens de tamanho diferentes. Observa-se que não houve variação no tempo de compilação médio, independentemente da solução criada e do tamanho da imagem, esses resultados reforçam a pouca influência do ambiente para a execução dos algoritmos nas plataformas paralelas.



(a) (b)

Figura 28: Tempo gasto para compilação do algoritmo de evapotranspiração pelo SEBAL, em imagens de $2000x2000$ (a) e $5000x5000$ (b).

Terminado os testes de impacto do ambiente, partiu-se para os testes de

desempenho dos algoritmos desenvolvidos.

4.3 PERFORMANCE (TESTE 2)

4.3.1 Tempo

A Figura 29 mostra o tempo de processamento (ms) de cada plataforma, para cada algoritmo desenvolvido (as linhas das Figuras). Pode-se observar que as plataformas paralelas foram superiores à sequencial (Figura 29(d)) em pelo menos 8 vezes (OpenCL CPU), chegando até 80 vezes mais rápida (OpenCL GPU). Entre as plataformas paralelas, observa-se que o OpenCL CPU apresenta uma tendência de crescimento superior às outras plataformas, OpenCL GPU e CUDA. Essa tendência acontece porque a quantidade de núcleos disponíveis é inferior, portanto, conforme cresce a quantidade de *threads* (dados) a serem processadas, maior a probabilidade de tais *threads* entrarem em uma fila. Fazendo o tempo de espera das *threads* aumentar proporcionalmente com a quantidade de dados, ocasionando um aumento no tempo total de processamento.

A plataforma CUDA foi a que apresentou menor variação no tempo - diferença entre o maior e o menor tempo - (cerca de 1567.7 ms) em comparação com o OpenCL GPU (2127.4), assim a plataforma CUDA foi o que apresentou melhor resultado em relação ao tempo de processamento seguido do OpenCL GPU, OpenCL CPU e Java.

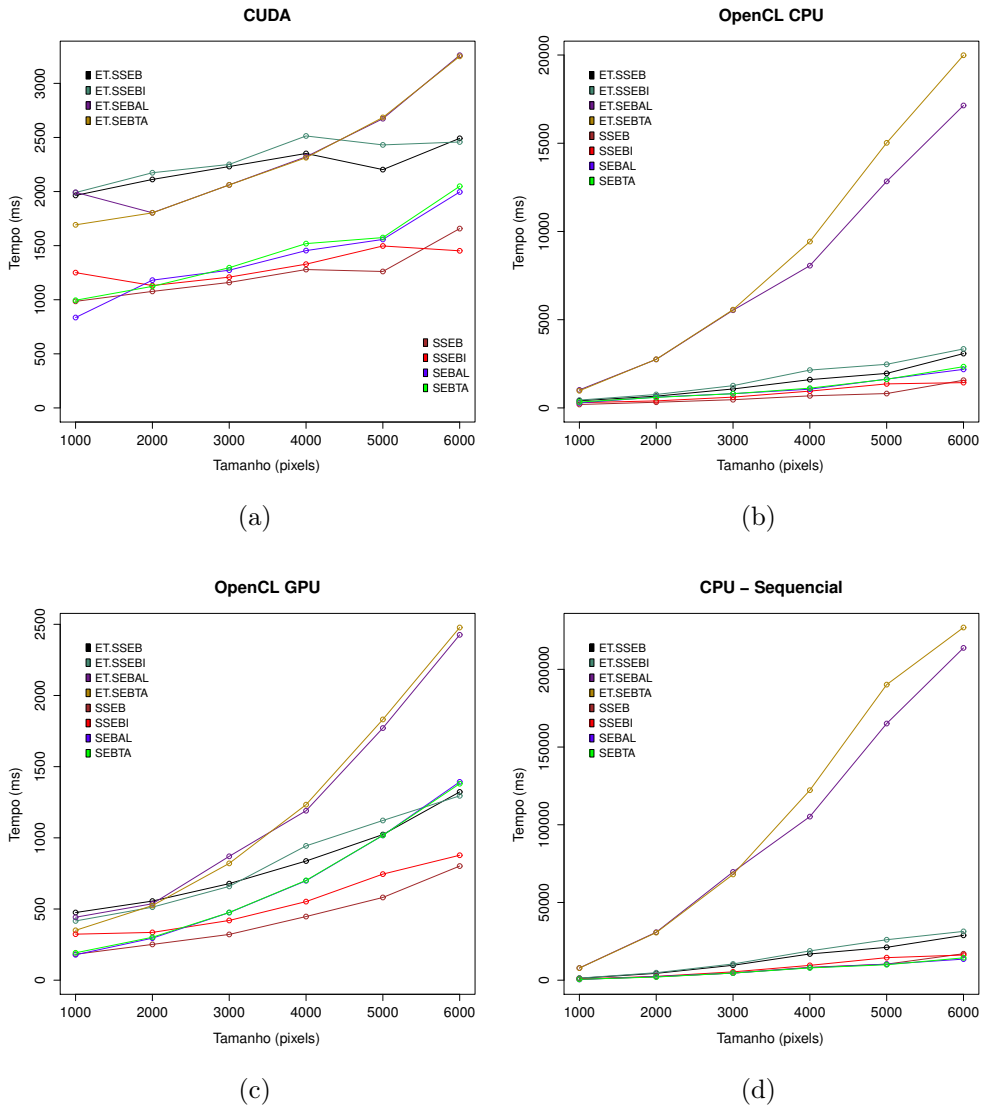


Figura 29: Representa a média do tempo de processamento para cada linguagem e para cada algoritmo.

Conclui-se que o paralelismo para o cálculo de sensoriamento remoto consegue diminuir o tempo de processamento em pelo menos 80 vezes. Considerando que o pesquisador tenha 10000 imagens para processar, e que leve uma média de 240 segundos para processar cada imagem, sem considerar o tempo de leitura e escrita, então em vez de demorar aproximadamente 27 dias de processamento irá levar aproximadamente 7 horas utilizando um dispositivo paralelo. É importante ressaltar que os dispositivos GPU usados neste trabalho não representam dispositivos de última geração, pois a melhor GPU testada tinha somente 384 núcleos, sendo que existem GPU's com mais de 2880 núcleos, ou seja, as GPU utilizadas representam somente 15% do poder de processamento das mais avançadas.

Contudo, o funcionamento independe da quantidade de núcleos.

Após o teste de tempo, foi realizado também o teste de memória ocupada, para cada plataforma e para cada algoritmos desenvolvido no GeneSE, aumentando também os tamanhos das imagens utilizadas.

4.3.2 Memória

A Figura 30 mostra o comportamento do espaço de memória utilizado (MBytes) de cada plataforma, para cada algoritmo.

O teste de memória consistiu em executar cada algoritmo de cada plataforma até o fim, via chamada no terminal, para que a máquina virtual Java pudesse efetivamente limpar a memória, com isso eliminou-se o problema de conter memória ocupada de outras execuções, problema conhecido na plataforma Java devido ao comportamento do *garbagge collect*.

Observa-se que independentemente da plataforma executada o comportamento no uso de memória foi o mesmo, com o tamanho mínimo de 50 MB para o algoritmo do SEBAL, até 746.75 MB para o ET.SSEB. Portanto, o ambiente não interfere na quantidade de espaço necessário para o processamento de cada algoritmo.

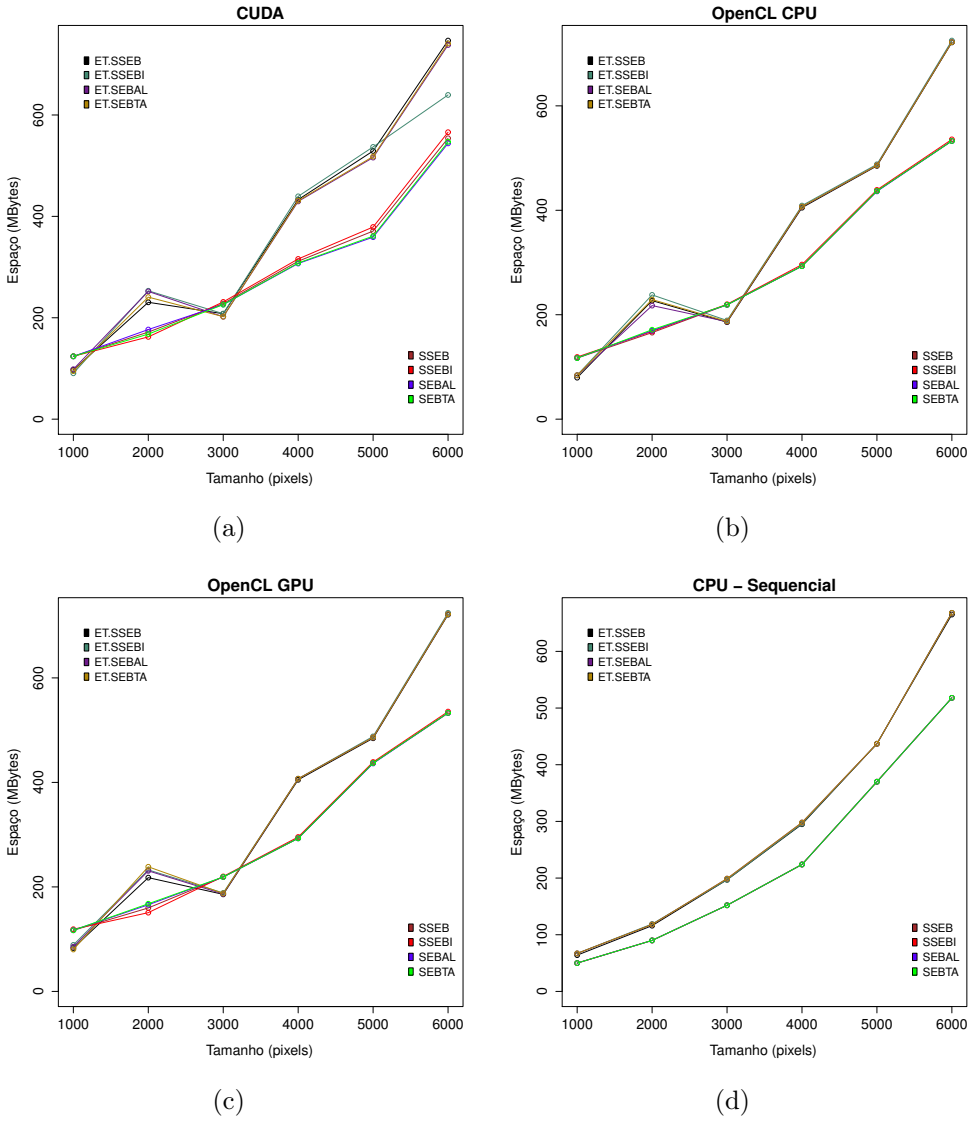


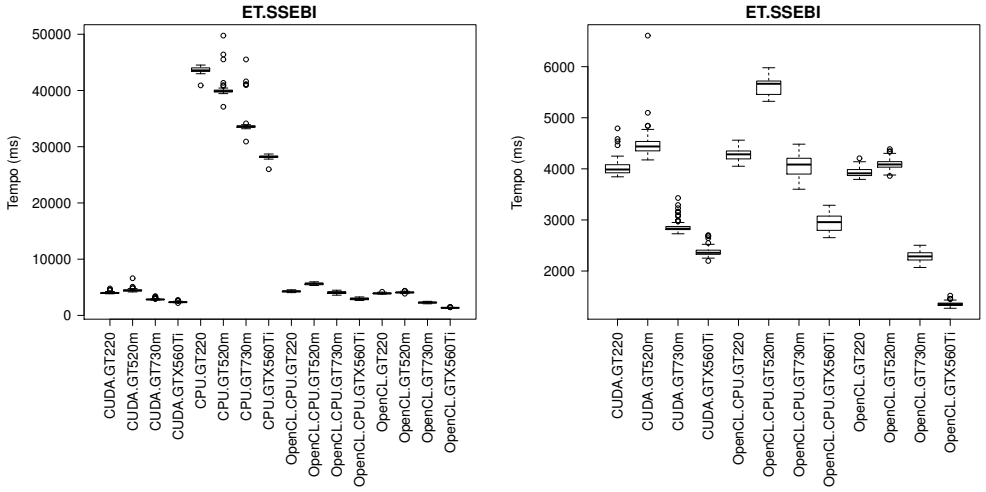
Figura 30: Média de espaço de memória utilizado por cada linguagem para cada algoritmo

Com as Figuras 29 e 30, pode-se concluir que ao utilizar o ambiente o pesquisador tem a garantia de não ocupar mais memória do que iria ocupar, e ainda terá um ganho significativo na redução de tempo de processamento em até 80 vezes, ao utilizar o processamento paralelo disponível no ambiente, além de facilitar a alteração de qualquer uma das equações, tirando do pesquisador a necessidade de se conhecer uma linguagem de programação.

A Figura 31 mostra a variação do tempo de processamento de uma imagem com 54.000.000 de pixels para cada máquina testada ao executar o algoritmo ET.SSEBI, sendo que em cada máquina foram executadas todas as plataformas disponíveis no ambiente. Observa-se que não houve grande variação entre as

plataformas paralelas (OpenCL CPU, OpenCL GPU e CUDA), mas houve uma grande variação no tempo de processamento entre as CPU's quando executado o algoritmo sequencial.

Já a Figura 31(b) mostra a variação do tempo mas somente entre os dispositivos paralelos.



(a) (b)

Figura 31: Representa a variação do tempo de processamento para cada dispositivo testado.

4.4 ANÁLISE DOS RESULTADOS ENTRE PLATAFORMAS

Após as análises de tempo e memória, foi averiguado se houve diferença nos valores calculados da evapotranspiração entre as plataformas utilizadas no ambiente, CUDA, CPU, OpenCL GPU, OpenCL CPU. Foi possível verificar se o paralelismo possui influência nos algoritmos. Para tanto foi definido como valor padrão os resultados calculados pela plataforma CPU, assim todas as outras plataformas foram comparadas com a sequencial.

A Tabela 8 apresenta os cálculos de R^2 da variável evapotranspiração calculada pelo ambiente utilizando a plataforma sequencial (CPU) com as demais plataformas, CUDA, OpenCL GPU e OpenCL GPU, para cada algoritmo utilizado. Pode-se observar que somente o cálculo de ET.SSEBI apresentou resultados inferiores a 0,97, isso se dá pelo fato de o S-SEBI precisar determinar regiões para o cálculo das retas, assim ao utilizar o paralelismo, o mesmo pode encontrar uma região melhor diferente do executado na forma sequencial. Contudo, obteve-se

0,60 de R^2 independente da plataforma utilizada. Já para os outros algoritmos os resultados se mostraram satisfatórios, demonstrando a não influência do paralelismo nos cálculos.

Tabela 8: Representa o R^2 da variável evapotranspiração do Java com todas as plataformas desenvolvidas

	R^2		
	CUDA	OpenCL GPU	OpenCL CPU
ET.SEBAL	1.00	1.00	0.99
ET.SEBTA	0.97	0.98	0.98
ET.SSEB	0.99	0.98	0.98
ET.SSEBI	0.60	0.63	0.61

Porém, para atestar a qualidade das retas de regressão foi calculado também os coeficientes de cada regressão 'a' e 'b' (Tabelas 9 e 10 respectivamente). A proximidade dos coeficientes 'a' à 1 e os 'b' à 0, indica a precisão dos cálculos, sendo os algoritmos de SEBAL, SEBTA e SSEB os mais precisos independente da plataforma utilizada, já para o S-SEBI o mesmo não acontece.

Tabela 9: Representa coeficientes 'a' das retas de regressão

	a		
	CUDA	OpenCL GPU	OpenCL CPU
ET.SEBAL	1.02	0.99	0.90
ET.SEBTA	1.16	1.07	1.06
ET.SSEB	0.98	1.06	0.98
ET.SSEBI	0.64	0.68	0.65

Tabela 10: Representa coeficientes 'b' das retas de regressão

	b		
	CUDA	OpenCL GPU	OpenCL CPU
ET.SEBAL	-0.06	0.02	0.33
ET.SEBTA	-0.35	-0.39	-0.32
ET.SSEB	0.02	0.13	0.10
ET.SSEBI	5.78	4.80	4.78

Conclui-se então que o ambiente consegue obter resultados estáveis para a evapotranspiração independente do algoritmo utilizado, e que esse resultado pode ser calculado por diferentes plataformas de tal modo que o resultado final não é afetado pelo dispositivo ou linguagem escolhida.

4.5 ANÁLISE DOS RESULTADOS ENTRE ALGORITMOS

Após a análise entre as plataformas realizou-se a validação dos dados calculados com dados obtidos de torres micrometeorológicas de cada área estudada para cada algoritmo por plataforma.

A Tabela 11 mostra o resultado da validação realizada por área, por algoritmo e por plataforma. Observa-se que para a área CAM (Cambarazal) o algoritmo do SEBAL obteve bons resultados, chegando a 0,61 de R^2, já para FE (Fazenda Experimental) o melhor foi o SSEB com 0,44, para a FM (Fazenda Miranda) o SEBTA foi o que apresentou melhor resultado com 0,75, já para SINOP não obteve-se resultados satisfatórios pois os melhores (SEBTA e S-SEBI) conseguiram obter 0,18, isso se deve ao fato dos dados medidos dessa região não possuírem uma variação durante o ano.

Tabela 11: Índices utilizados para a validação dos algoritmos por área.

Área	Algoritmo	Plataforma	MAE	RMSE	d	R^2
CAM	SEBAL	OpenCL GPU	0,35	0,96	0,98	0,61
CAM	SEBAL	OpenCL CPU	0,35	0,96	0,98	0,61
CAM	SEBAL	CUDA	0,35	0,96	0,98	0,61
CAM	SEBAL	CPU	0,35	0,96	0,98	0,61
CAM	SEBTA	CPU	1,57	1,95	0,88	0,30
CAM	SEBTA	OpenCL GPU	1,68	2,02	0,87	0,29
CAM	SEBTA	OpenCL CPU	1,68	2,02	0,87	0,29
CAM	SEBTA	CUDA	1,68	2,07	0,86	0,29
CAM	SSEB	OpenCL GPU	1,59	1,92	0,87	-0,08
CAM	SSEB	OpenCL CPU	1,82	2,11	0,83	-0,07
CAM	SSEB	CUDA	2,16	2,52	0,72	-0,06
CAM	SSEB	CPU	2,34	2,65	0,66	-0,06
CAM	S-SEBI	OpenCL CPU	1,24	1,60	0,93	0,39
CAM	S-SEBI	OpenCL GPU	1,24	1,60	0,93	0,39
CAM	S-SEBI	CUDA	1,22	1,58	0,93	0,39
CAM	S-SEBI	CPU	1,16	1,56	0,93	0,39
FE	SEBAL	CPU	2,62	8,13	0,41	0,02
FE	SEBAL	CUDA	2,62	8,13	0,41	0,02
FE	SEBAL	OpenCL GPU	2,62	8,13	0,41	0,02
FE	SEBAL	OpenCL CPU	2,62	8,14	0,41	0,02

FE	SEBTA	CPU	0,42	0,89	0,96	0,21
FE	SEBTA	OpenCL GPU	0,97	1,25	0,89	0,19
FE	SEBTA	OpenCL CPU	0,97	1,25	0,89	0,19
FE	SEBTA	CUDA	1,22	1,47	0,82	0,18
FE	SSEB	CUDA	0,86	1,18	0,90	0,43
FE	SSEB	CPU	0,90	1,17	0,90	0,43
FE	SSEB	OpenCL CPU	0,79	1,06	0,92	0,44
FE	SSEB	OpenCL GPU	0,60	0,88	0,95	0,45
FE	S-SEBI	CPU	0,34	0,92	0,95	0,15
FE	S-SEBI	CUDA	0,53	0,83	0,96	0,15
FE	S-SEBI	OpenCL GPU	0,49	0,79	0,96	0,15
FE	S-SEBI	OpenCL CPU	0,48	0,79	0,96	0,15
FM	SEBAL	OpenCL CPU	0,58	1,02	0,96	0,51
FM	SEBAL	CPU	0,58	1,02	0,96	0,51
FM	SEBAL	OpenCL GPU	0,58	1,02	0,96	0,51
FM	SEBAL	CUDA	0,58	1,02	0,96	0,51
FM	SEBTA	CPU	1,05	1,26	0,94	0,75
FM	SEBTA	OpenCL CPU	1,06	1,26	0,94	0,75
FM	SEBTA	OpenCL GPU	1,06	1,26	0,94	0,75
FM	SEBTA	CUDA	1,26	1,31	0,93	0,74
FM	SSEB	OpenCL CPU	1,29	1,63	0,88	-0,09
FM	SSEB	OpenCL GPU	1,28	1,64	0,88	-0,09
FM	SSEB	CUDA	1,46	1,77	0,86	-0,09
FM	SSEB	CPU	1,76	2,32	0,71	-0,07
FM	S-SEBI	OpenCL CPU	1,15	1,50	0,90	0,29
FM	S-SEBI	CUDA	1,16	1,49	0,90	0,29
FM	S-SEBI	OpenCL GPU	1,15	1,49	0,90	0,29
FM	S-SEBI	CPU	1,12	1,47	0,91	0,29
SINOP	SEBAL	OpenCL CPU	0,04	0,69	0,99	0,19
SINOP	SEBAL	OpenCL GPU	0,04	0,69	0,99	0,19
SINOP	SEBAL	CUDA	0,04	0,69	0,99	0,19
SINOP	SEBAL	CPU	0,04	0,69	0,99	0,19
SINOP	SEBTA	OpenCL GPU	1,75	2,04	0,81	-0,02
SINOP	SEBTA	OpenCL CPU	1,75	2,04	0,81	-0,02
SINOP	SEBTA	CUDA	1,72	2,04	0,82	-0,02
SINOP	SEBTA	CPU	1,60	1,91	0,84	-0,02
SINOP	SSEB	OpenCL CPU	1,27	1,43	0,92	-0,57

SINOP	SSEB	OpenCL GPU	1,34	1,50	0,92	-0,56
SINOP	SSEB	CPU	1,47	1,73	0,88	-0,54
SINOP	SSEB	CUDA	1,53	1,84	0,86	-0,53
SINOP	S-SEBI	CUDA	0,84	1,03	0,96	0,18
SINOP	S-SEBI	OpenCL GPU	0,77	0,99	0,97	0,18
SINOP	S-SEBI	OpenCL CPU	0,76	0,98	0,97	0,18
SINOP	S-SEBI	CPU	0,73	0,96	0,97	0,18

Conclui-se que os algoritmos possuem comportamentos diferentes para áreas diferentes não sendo um sempre melhor que o outro. Portanto, uma determinada área pode ser estudada por todos os algoritmos e o especialista decide qual se adéqua melhor ao perfil da área.

4.6 GENESE GUI

Com os dados analisados e comparados desenvolveu-se uma interface de usuário (GUI) adequada para facilitar o uso do ambiente GeneSE pelo usuário final. A Figura 32 mostra a parte da GUI na qual as constantes utilizadas no conjunto de equações foram inseridas.

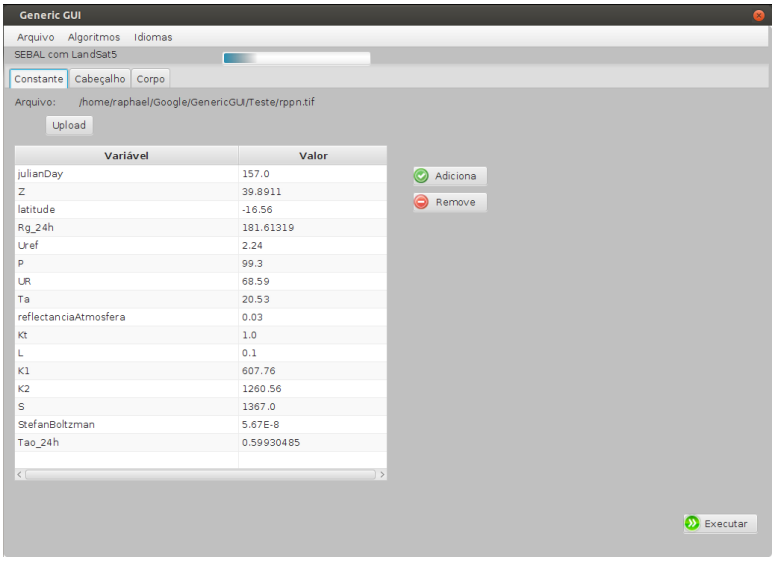


Figura 32: Parte da tela que representa o grupo de constantes (Θ).

Já a Figura 33 mostra a tabela na qual serão inseridas as equações que serão calculadas somente uma vez para todo o conjunto de dados, do grupo denominado $forVariables$

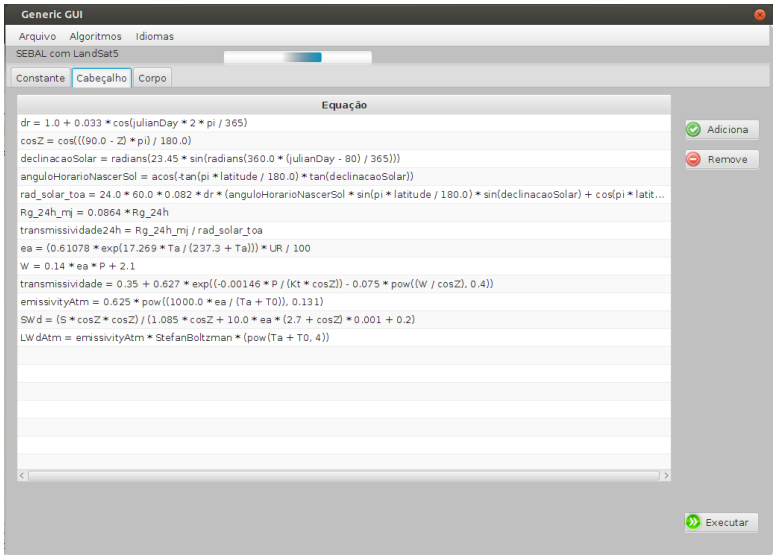


Figura 33: Parte da tela que representa o grupo de $forVariables$ (Ω).

E por fim a Figura 34 mostra as equações que serão processadas para cada pixel da imagem, e quais serão as variáveis que deseja-se obter como resposta.

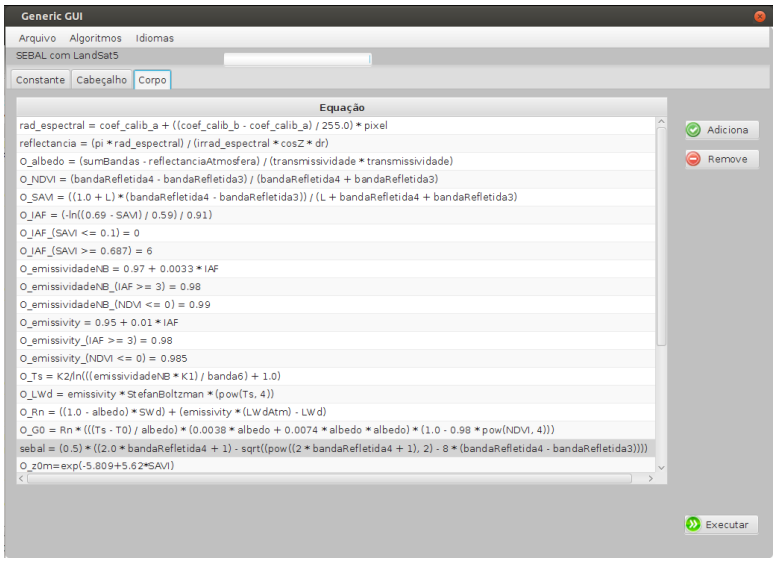


Figura 34: Parte da tela que representa o grupo de $forEachValue$ (Φ).

4.7 Conclusão

O conjunto de ferramentas que compõem o ambiente GeneSe permite o uso e consequentemente a difusão de modelos de estimativas de evapotranspiração, com o objetivo principal de facilitar e flexibilizar a utilização das técnicas citadas, mas sem enrijecê-las quanto a definição de quais fórmulas utilizar durante todo o processo.

Outra característica disponibilizada no ambiente é a utilização de dispositivos paralelos para os cálculos envolvidos nas imagens de satélite. Sendo o CUDA e OpenCL, as linguagens escolhidas para o paralelismo, e o Java para sequencial e como linguagem hospedeira. Os dispositivos escolhidos para serem utilizados pelo ambiente é o GPU e a CPU, sendo a CPU tanto para o cálculo paralelo quanto o sequencial.

Para flexibilizar o uso de CUDA e OpenCL foi realizado também um aprimoramento da framework JEDIGPU, sendo implementado na mesma o recurso de distribuir os cálculos entre vários dispositivos, se os mesmos encontrarem-se disponíveis na máquina utilizada. Além de automatizar diversos cálculos para a melhor utilização de cada linguagem.

Antes de desenvolver as técnicas de cálculo do balanço de energia de superfície no ambiente, foi necessário estudar cada algoritmo e automatizá-los, pois em todos haviam procedimentos que eram realizados de forma manual, dispendendo muito tempo do pesquisador para a obtenção dos resultados. Seguindo a etapa de automatização foi disponibilizado também uma flexibilização para cada algoritmo automatizado, sendo que o pesquisador pode definir de maneira simples como será empregada a técnica do algoritmo.

Com o ambiente desenvolvido realizou-se os testes com 85 imagens de satélite de quatro áreas distintas de Mato Grosso, sendo cada área com características da vegetação diferentes. Os testes demonstraram a facilidade no uso das técnicas; a flexibilização dos usos das equações nas etapas; a portabilidade do sistema entre máquinas distintas; a possibilidade de utilização de processamento paralelo, tanto em dispositivos GPU quanto em CPU e principalmente a diminuição do tempo de processamento de cada imagem testada. Considerando todos os algoritmos, todas as plataformas e todas as máquinas testadas, foram processadas uma quantidade equivalente a 10880 imagens. O processamento dessas imagens ao utilizar o algoritmo sequencial desenvolvido no ambiente seria de cinco dias aproximadamente, mesmo considerando que nesse processo houve a automatização de cada algoritmo. Já ao utilizar o processamento paralelo, por exemplo, o OpenCL GPU do ambiente, esse tempo diminui para aproximadamente sete horas, considerando que o equipamento utilizado possui somente 15% da quantidade de núcleos em dispositivos de última geração.

Os resultados mostram também que três algoritmos chegaram a um R^2 maior que 0,97, indicando que as plataformas utilizadas não influenciam nos resultados, contudo o S-SEBI atingiu cerca de 0,68. Isso se deve ao fato do S-SEBI procurar por regiões e não por pontos específicos. Logo, ao utilizar um dispositivo

paralelo o mesmo pode encontrar uma região diferente do que encontraria se fosse sequencial.

Os resultados da validação dos algoritmos com os dados disponíveis das torres micrometeorológicas, mostraram que os algoritmos possuem comportamentos distintos entre as regiões estudadas, sendo o SEBAL melhor para a região do Cambarazal (CAM), SSEB para a região da Fazenda Experimental (FE), e SEBTA para a região da Fazenda Miranda (FM). A única região que não obteve resultado satisfatório foi a região de Sinop, pois essa região teve problema na obtenção dos dados. Esse estudo mostrou a necessidade de se utilizar mais de um algoritmo para cada região, fazendo adaptações nas equações para uma melhor aproximação.

Este trabalho buscou utilizar o dispositivo de menor custo para processamento paralelo, e desenvolveu uma solução para minimizar o tempo de desenvolvimento/resposta para cálculos de imagens de satélite, utilizando arquitetura paralela. Para isso, o ambiente GeneSE torna transparente ao pesquisador a implementação de equações para essa arquitetura, ou seja, o ambiente tem o intuito de facilitar o processamento dos cálculos de forma paralela.

Portanto, o ambiente facilita e agiliza todo o processo de cálculo da evapotranspiração de quatro algoritmos de balanço de energia de superfície, permitindo ao pesquisador a realização de estudos sobre os impactos das equações, para que não tenha que investir tempo na implementação das técnicas citadas.

Capítulo 5

CONSIDERAÇÕES FINAIS

A água é um dos principais fatores para a determinação da meteorologia e do clima, sendo que estudar os processos que envolvem tal recurso é de fundamental importância, principalmente pela alta demanda na produção de alimentos, sendo essa produção extremamente vinculada a disponibilidade e utilização da água. Portanto, a realização de estimativas da evapotranspiração (maior fatia de água incidente na atmosfera) representa uma medida que auxilia o desenvolvimento dos processos de irrigação, de hidroelétricas, processos industriais, entre outros. Como visto, existem diversos modelos para realizar tal estimativa em área, tais como o SEBAL, SEBTA, SSEB e S-SEBI, sendo estes os utilizados neste trabalho.

Com o desenvolvimento do ambiente GeneSE foi possível realizar a automatização dos algoritmos utilizados, facilitando seu uso, dispensando a obtenção de conhecimentos por parte do pesquisador para a sua implementação. Desse modo, as estimativas obtidas da evapotranspiração pelo ambiente tornam-se mais confiáveis pela eliminação de possíveis erros. Portanto, novas perspectivas são concretizadas com a possibilidade do pesquisador se concentrar apenas na interpretação dos dados e na comparação dos resultados gerados pelos modelos. Principalmente pelo fato dos métodos possuírem características diferentes, possibilitando que o pesquisador ajuste os parâmetros de cada método diferentemente para cada tipo de área.

5.1 CONTRIBUIÇÕES

Podem-se elencar as seguintes contribuições deste trabalho:

- Abstração das técnicas e seus algoritmos em forma de equações, eliminando a necessidade de o pesquisador conhecer alguma linguagem de programação.

- Flexibilização da integração dos algoritmos através de equações definidas pelo pesquisador que facilita a realização de testes com equações diferentes.
- Automatização da sequência de processamento dos algoritmos SEBAL, SEBTA, SSEB e S-SEBI, de forma que o pesquisador não precisa saber como implementar as técnicas utilizadas, mas somente quais equações deseja-se utilizar para cada um.
- Automatização da criação, compilação e execução dos códigos relacionados com as equações.
- Diminuição de erros no processo de análise, de forma que o pesquisador fica isolado do processo de implementação, diminuindo assim a probabilidade de ocorrer erros nas técnicas utilizadas.
- Definir automaticamente as políticas para o melhor uso de GPU's com as linguagens CUDA e OpenCL. As políticas definidas buscam utilizar o potencial máximo das GPU's de forma automática, eliminando a necessidade de realizar todos os cálculos necessários para a execução de códigos paralelos de forma manual.
- Permitir a portabilidade entre máquinas com dispositivos paralelos (GPU e CPU) distintos, sendo assim o mesmo conjunto de equações pode ser executado em dispositivos paralelos distintos, sem a necessidade realizar alterações.
- Distribuição automática dos processos caso o hardware hospedeiro tenha mais de um dispositivo paralelo, portanto a paralelização pode ocorrer em um segundo nível, onde os processos além de serem divididos pelos núcleos do dispositivo, será dividido também entre dispositivos distintos.

5.2 TRABALHOS FUTUROS

Apesar das diversas contribuições científicas e tecnológicas já obtidas pelo ambiente GeneSE, entende-se que sua evolução ocorrerá com as seguintes abordagens:

Primeiramente é necessário incluir no ambiente perfis de validação das técnicas de evapotranspiração para regiões distintas. Tal funcionalidade permitirá identificar a existência de regras para aplicação das técnicas em determinados tipos de clima.

Para a manipulação das imagens, é importante melhorar o controle da qualidade das imagens de satélite, porque em alguns casos os dados não estão qualificados para serem utilizados. Por exemplo, a partir do uso de metadados já existentes nas imagens é possível verificar a qualidade das informações. Uma outra situação é a identificação de pixels com mesmo valor para bandas diferentes.

Já nos aspectos relacionados com a manipulação das imagens de satélites, é importante integração com o ambiente ArcGIS, sendo possível ao pesquisador a utilização dos recursos de tratamento de imagens do ArcGIS, juntamente com o processamento do GeneSE, além de permitir realizar todas as operações de forma transparente como sendo um único software.

Em relação ao uso das equações, um recurso a ser adicionado é o de permitir funções agregadas, que envolve funções como: somatório, produtório, média e desvio padrão. Outra característica é flexibilizar a criação de novas funções pelo pesquisador.

É necessário um maior aprofundamento nas técnicas de utilização das GPU's para complementar o ambiente, tais como a técnica de redução, que representa o agrupamento de dados em conjuntos cada vez menores; e a técnica de coalescência, que é uma técnica utilizada para otimizar o acesso à memória das GPU's.

Outro trabalho é a realização de teste em *clusters* a fim de testar a distribuição automática do ambiente com a linguagem OpenCL, pois como a mesma suporta CPU será possível comparar *clusters* de CPU e GPU.

REFERÊNCIAS

ALLEN, R.; BASTIAANSSEN, W.; WATERS, R.; TASUMI, M.; TREZZA, R. Surface energy balance algorithms for land (sebal). *Journal of Irrigation and Drainage Engineering*, v. 122, n. 2, p. 97–106, 2002. Citado 4 vezes nas páginas 11, 12, 13 e 17.

ALLEN, R.; IRMAK, A.; TREZZA, R.; HENDRICKX, J. M. H.; BASTIAANSSEN, W.; KJAERSGAARD, J. Satellite-based et estimation in agriculture using sebal and metric. *Hydrological Processes*, v. 25, p. 4011–4027, 2011. Citado 2 vezes nas páginas 16 e 59.

ALLEN, R. G.; TASUMI, M.; TREZZA, R. Satellite-based energy balance for mapping evapotranspiration with internalized calibration (metric)-model. *J. Irrig.. Drain. Engr.*, v. 133, n. 4, p. 380–394, 2007. Citado na página 10.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS 67 (Spring) Proceedings*, p. 483–485, 1967. Citado na página 27.

AUGUSTO, D. A.; BARBOSA, H. J. C. Accelerated parallel genetic programming tree evaluation with opencl. *Journal of Parallel and Distributed Computing*, 2012. Citado 2 vezes nas páginas II e 34.

BASTIAANSSEN, W. G. M. Sebal - based sensible and latent heat fluxes in the irrigated gediz basin turkey. *Journal of Hydrology*, v. 229, n. 0, p. 87–100, 2000. Citado 3 vezes nas páginas 10, 12 e 16.

BASTIAANSSEN, W. G. M.; MENENTI, M.; FEDDES, R. A.; HOLTSLAG, A. A. M. A. Remote sensing surface energy balance algorithm for land (sebal) 1. *Formulation Journal of Hydrology*, v. 212-213, n. 0, p. 198–212, 1998. Citado 2 vezes nas páginas 8 e 13.

BASTIAANSSEN, W. G. M.; PELGRUM, H.; WANG, J.; MA, Y.; MORENO, J. F.; ROERINK, G. F.; WAL, T. V. D. Remote sensing surface energy balance algorithm for land (sebal) 2. *Journal of hydrology*, v. 212, p. 213–229, 1998. Citado na página 8.

CHENG, W. L.; SHEHARYAR, A.; SADR, R.; BOUHALI, O. Application of gpu processing for brownian particle simulation. *Computer Physics Communications*, v. 186, n. 0, p. 39 – 47, 2015. ISSN 0010-4655. Citado na página 32.

CROW, T. S. *Evolution of the Graphical Processing Unit*. Dissertação (Mestrado) — University of Nevada-Reno, 2005. Citado na página 32.

ELLIOTT, G. A.; ANDERSON, J. H. Globally scheduled real-time multiprocessor systems with gpus. *Real-Time Systems*, v. 48, n. 1, p. 34–74, 2012. Citado na página 23.

FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 21, n. 9, p. 948–960, set. 1972. ISSN 0018-9340. Citado na página 22.

FOSTER, I. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0201575949. Citado 3 vezes nas páginas I, 27 e 28.

FURHT, B. Cloud computing fundamentals. In: FURHT, B.; ESCALANTE, A. (Ed.). *Handbook of Cloud Computing*. [S.l.]: Springer US, 2010. p. 3–19. ISBN 978-1-4419-6524-0. Citado na página 25.

GAO, Z. Q.; LIU, C. S.; GAO, W.; CHANG, N. B. A coupled remote sensing and the surface energy balance with topography algorithm (sebta) to estimate actual evapotranspiration over heterogeneous terrain. *Hydrology and Earth System Sciences*, v. 15, p. 119–139, 2011. Citado 2 vezes nas páginas 8 e 17.

GOMES, R. S. R. *Arquitetura de processamento paralelo com unidade de processamento gráfico de propósito geral para aplicação em séries temporais de dados ambientais*. master, 2012. Citado 3 vezes nas páginas 39, 43 e 59.

GOMES, R. S. R.; FIGUEIREDO, J. M. Por que um cientista se interessa por videogames? *Ciência Hoje*, v. 52, p. 24–27, 2014. ISSN 0101-8515. Citado na página 32.

GOMES, R. S. R.; FIGUEIREDO, J. M.; MARTINS, C. A.; OLIVEIRA, A. G.; NOGUEIRA, J. S. A framework for automating the configuration of opencl. *Environmental Modelling & Software*, v. 53, p. 81 – 86, 2014. ISSN 1364-8152. Citado na página 43.

GOVENDER, N.; WILKE, D. N.; KOK, S. Collision detection of convex polyhedra on the nvidia gpu architecture for the discrete element method. *Applied Mathematics and Computation*, n. 0, p. –, 2014. ISSN 0096-3003. Citado na página 32.

GOWDA, P. H.; CHAVEZ, J. L.; COLAIZZI, P. D.; EVETT, S. R.; HOWELL, T. A.; TOLK, J. A. Et mapping for agricultural water management. *Present status and challenges, Irrig. Sci*, v. 26, p. 223–237, 2008. Citado na página 4.

GOWDA, P. H.; SENAY, G. B.; HOWELL, T. A.; MAREK, T. H. Lysimetric evaluation of simplified surface energy balance approach in the texas high plains. *Applied Engineering in Agriculture*, v. 25, n. 5, p. 665–669, 2009. Citado 3 vezes nas páginas 8, 18 e 19.

GROUP, K. O. W. *The OpenCL specification, Version 1.1*. 2011. Available in <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>. Citado na página 34.

HWANG, K. *Advanced computer architecture: parallelism, scalability, programmability*. [S.l.]: McGraw-Hill, 1993. (McGraw-Hill series in electrical and computer engineering: Computer engineering). ISBN 9780070316225. Citado 3 vezes nas páginas 21, 30 e 31.

HWANG, K.; BRIGGS, F. A. *Computer architecture and parallel processing*. [S.l.]: McGraw-Hill, 1984. (McGraw-Hill computer communications series). ISBN 9780070315563. Citado 2 vezes nas páginas 21 e 27.

IQBAL, M. Front matter. In: *An Introduction to Solar Radiation*. [S.l.]: Academic Press, 1983. ISBN 978-0-12-373750-2. Citado na página 6.

JCUDA. *Java bindings for CUDA*. 2012. Available in <http://www.jcuda.org/>. Citado na página 39.

JOCL. *Java bindings for OpenCL*. 2012. Available in <http://www.jocl.org/>. Citado na página 39.

JOSELLI, M.; JUNIOR, J. R. da S.; CLUA, E. W.; MONTENEGRO, A.; LAGE, M.; PAGLIOSA, P. Neighborhood grid: A novel data structure for fluids animation with {GPU} computing. *Journal of Parallel and Distributed Computing*, v. 75, n. 0, p. 20 – 28, 2015. ISSN 0743-7315. Citado na página 32.

LUTSYSHYN, Y. Fast quantum monte carlo on a {GPU}. *Computer Physics Communications*, v. 187, n. 0, p. 162 – 174, 2015. ISSN 0010-4655. Citado na página 32.

MARKHAM, B. L.; HELDER, D. L. Forty-year calibrated record of earth-reflected radiance from landsat: A review. *Remote Sensing of Environment*, v. 122, n. 0, p. 30 – 40, 2012. ISSN 0034-4257. Landsat Legacy Special Issue. Citado 2 vezes nas páginas IV e 5.

MENDONÇA, J. C.; SOUZA, E. F.; BOUHID, A.; SILVA, B. B.; FERREIRA, F. J. Estimativa do fluxo de calor sensível utilizando o algoritmo sebal e imagens modis para a região norte fluminense. *Revista Brasileira de Meteorologia*, v. 27, n. 1, p. 85–94, 2012. Citado na página 13.

NASA. *The Landsat Program*. 2014. Disponível em: <http://landsat.gsfc.nasa.gov/>. Citado 2 vezes nas páginas IV e 5.

NASA. *MODIS Website*. 2014. Disponível em: <http://modis.gsfc.nasa.gov/>. Citado na página 7.

NVIDIA. *NVIDIA OpenCL Jump Start Guide*. 2009. Available in <http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf>. Citado na página 34.

NVIDIA. *OpenCL Programming for the CUDA Architecture*. 2009. Available in <http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingOverview.pdf>. Citado 2 vezes nas páginas 46 e 53.

NVIDIA. *NVIDIA CUDA C Programming Guide*. 2011. Available in <http://developer.download.nvidia.com/compute/\DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf>. Citado 3 vezes nas páginas 32, 33 e 47.

NVIDIA. *CUDA GPU Occupancy Calculator*. 2012. Available in <http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls>. Citado na página 47.

OWENS, J. D.; HOUSTON, M.; LUEBKE, D.; GREEN, S.; STONE, J. E.; PHILLIPS, J. C. GPU Computing. *Proceedings of the IEEE*, IEEE, v. 96, n. 5, p. 879–899, maio 2008. ISSN 0018-9219. Citado na página 32.

PAPAKONSTANTINOU, A.; GURURAJ, K.; STRATTON, J. A.; CHEN, D.; CONG, J.; HWU, W. M. W. Fcuda: Enabling efficient compilation of cuda kernels onto fpgas. In: *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*. [S.l.: s.n.], 2009. p. 35 –42. Citado na página 32.

PEREIRA, M. F. L. *Um Modelo de Reconstrução Tomográfica 3D para Amostras Agrícolas com Filtragem de Wiener em Processamento Paralelo*. 148 p. Tese (Doutorado), 2007. Citado 2 vezes nas páginas I e 29.

PURCELL, T. J.; BUCK, I.; MARK, W. R.; HANRAHAN, P. Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 21, n. 3, p. 703–712, jul. 2002. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/566654.566640>. Citado na página 32.

ROERINK, G. J.; SU, Z.; MENENTI, M. S-sebi: A simple remote sensing algorithm to estimate the surface energy balance. *Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere*, v. 25, n. 2, p. 147 – 157, 2000. ISSN 1464-1909. Disponível em: <http://www.sciencedirect.com/science/article\/pii/S1464190999001288>. Citado 2 vezes nas páginas 8 e 19.

SADLER, E. J.; BAUER, P. J.; BUSSCHER, W. J.; MILLEN, J. A. Site-specific analysis of a droughted corn crop: Ii. water use and stress. *Agronomy Journal*, v. 92, p. 403–410, 2000. Citado na página 19.

SANCHES, L.; VALENTINI, C. M. A.; JUNIOR, O. B. P.; NOGUEIRA, J. S.; VOURLITIS, G. L.; BIUDES, M. S.; SILVA, C. J.; BAMBI, P.; LOBO, F. A. Seasonal and interannual litter dynamics of a tropical semideciduous forest of the southern amazon basin, brazil. *Journal of geophysical researc*, v. 113, n. 113, p. 1–9, 2008. Citado na página 37.

SARKAR, V. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. Cambridge, MA, USA: MIT Press, 1989. ISBN 0262691302. Citado na página 26.

SEMCHEDINE, F.; MEDJKOUNE, L. B.; AıSSANI, D. Task assignment policies in distributed server systems: A survey. *Journal of Network and Computer Applications*, v. 24, p. 1123–1130, 2011. Citado 2 vezes nas páginas I e 22.

SENAY, G. B.; BUDDE, M.; VERDIN, J. P.; MELESSE, A. M. A coupled remote sensing and simplified surface energy balance approach (sseb) to estimate actual evapotranspiration from irrigated fields. *Sensors*, v. 7, p. 979–1000, 2007. Citado na página 8.

SENAY, G. B.; BUDDE, M. E.; VERDIN, J. P. Enhancing the simplified surface energy balance (sseb) approach for estimating landscape et: Validation with the {METRIC} model. *Agricultural Water Management*, v. 98, n. 4, p. 606 – 618, 2011. ISSN 0378-3774. Disponível em: <http://www.sciencedirect.com/science/article\/pii/S0378377410003355>. Citado na página 19.

SILVA, B. B. da; LOPES, G. M.; AZEVEDO, P. V. de. Determinação do albedo em áreas irrigadas com base em imagens landsat 5 - tm. *Revista Brasileira de Meteorologia*, v. 13, n. 2, p. 201–211, 2005. Citado 2 vezes nas páginas 11 e 12.

SU, H.; MCCABE, M.; WOOD, E.; SU, Z.; PRUEGER, J. Modeling evapotranspiration during smacex: Comparing two approaches for local-and regional-scale prediction. *Journal of hydrometeorology*, v. 6, n. 6, p. 910–922, 2005. Citado na página 18.

TOP500. *TOP 500 The List*. 2015. Disponível em: <http://www.top500.org/lists/2014/11/>. Citado na página 31.

TORKESTANI, J. A. A new approach to the job scheduling problem in computational grids. *Cluster Computing*, Springer Netherlands, p. 1–10, 2011. ISSN 1386-7857. 10.1007/s10586-011-0192-5. Citado na página 25.

VALENTINI, G. L.; LASSONDE, W.; KHAN, S. U.; MIN-ALLAH, N.; MADANI, S. A.; LI, J.; ZHANG, L.; WANG, L.; GHANI, N.; KOLODZIEJ, J.; LI, H.; ZOMAYA, A. Y.; XU, C. Z.; BALAJI, P.; VISHNU, A.; PINEL, F.; PECERO, J. E.; KLIAZOVICH, D.; BOUVRY, P. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, p. 0 – 0, 2011. Citado 2 vezes nas páginas 24 e 25.

VOURLITIS, G. L.; LOBO, F. A.; ZEIHLOFER, P.; NOGUEIRA, J. S. Temporal patterns of net co2 exchange for a tropical semi-deciduous forest of the southern amazon basin. *Journal of Geophysical Research*, v. 116, 2011. Citado na página 36.

VOURLITIS, G. L.; PRIANTE, N.; HAYASHI, M. M. S.; NOGUEIRA, J. D.; CASEIRO, F. T.; CAMPELO, J. H. Seasonal variations in the evapotranspiration of a transitional tropical forest of matogrosso, brazil. *Water Resources Research*, v. 38, n. 6, p. 1–11, 2002. Citado na página 37.

WEIGEL, M. Performance potential for simulating spin models on gpu. *Journal of Computational Physics*, v. 231, n. 8, p. 3064–3082, 2012. Citado na página 31.

WILLMOTT, C. J.; CKLESON, S. G.; DAVIS, R. E.; FEDDEMA, J. J.; KLINK, K. M.; LEGATES, D. R.; O'DONNELL, J.; ROWE, C. M. Statistics for the evaluation and comparison of models. *Journal of Geophysical Research*, v. 90, n. 5, p. 8995–9005, 1985. Citado na página 41.

WILLMOTT, C. J.; MATSSURA, K. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, v. 30, p. 79–92, 2005. Citado 2 vezes nas páginas 41 e 42.

YAMANAKA, A.; AOKI, T.; OGAWA, S.; TAKAKI, T. Gpu-accelerated phase-field simulation of dendritic solidification in a binary alloy. *Journal of Crystal Growth*, v. 318, n. 1, p. 40–45, 2011. Citado na página 31.

ZHANG, T.; ZHOU, J.; CARNEY, P. R.; JIANG, H. Towards real-time detection of seizures in awake rats with gpu-accelerated diffuse optical tomography. *Journal of Neuroscience Methods*, v. 240, n. 0, p. 28 – 36, 2015. ISSN 0165-0270. Citado na página 32.