Universidade Estadual de Campinas
Faculdade de Tecnologia

Luís Fernando Lopes Grim

ENSEMBLES DE ALTO DESEMPENHO DO ALGORITMO ONLINE SEQUENTIAL EXTREME LEARNING MACHINE PARA REGRESSÃO E PREVISÃO DE SÉRIES TEMPORAIS

Limeira, 2018

Luís Fernando Lopes Grim

ENSEMBLES DE ALTO DESEMPENHO DO ALGORITMO ONLINE SEQUENTIAL EXTREME LEARNING MACHINE PARA REGRESSÃO E PREVISÃO DE SÉRIES TEMPORAIS

Dissertação apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Tecnologia, na área de Sistemas de Informação e Comunicação.

Orientador: Prof. Dr. André Leon Sampaio Gradvohl

Este exemplar corresponde à versão final da Dissertação defendida pelo aluno Luís Fernando Lopes Grim e orientada pelo Prof. Dr. André Leon Sampaio Gradvohl.

Limeira, 2018

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.
ORCID: https://orcid.org/0000-0002-1221-409

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Faculdade de Tecnologia
Felipe de Souza Bueno - CRB 8/8577

G88e

Grim, Luís Fernando Lopes, 1987-
Ensembles de alto desempenho do algoritmo *Online Sequential Extreme Learning Machine* para regressão e previsão de séries temporais / Luís Fernando Lopes Grim. – Limeira, SP : [s.n.], 2018.

Orientador: André Leon Sampaio Gradvohl.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Computação de alto desempenho. 2. Aprendizado de máquina. 3. Análise de séries temporais. I. Gradvohl, André Leon Sampaio, 1973-. II. Universidade Estadual de Campinas. Faculdade de Tecnologia. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: High-performance ensembles of the Online Sequential Extreme Learning Machine algorithm for regression and time series forecasting
Palavras-chave em inglês:
High-performance computing
Machine learning
Time-series analysis
Área de concentração: Sistemas de Informação e Comunicação
Titulação: Mestre em Tecnologia
Banca examinadora:
André Leon Sampaio Gradvohl [Orientador]
Ivan Luiz Marques Ricarte
Elaine Parros Machado de Souza
Data de defesa: 22-08-2018
Programa de Pós-Graduação: Tecnologia

Universidade Estadual de Campinas
Faculdade de Tecnologia

Luís Fernando Lopes Grim

ENSEMBLES DE ALTO DESEMPENHO DO ALGORITMO ONLINE SEQUENTIAL EXTREME LEARNING MACHINE PARA REGRESSÃO E PREVISÃO DE SÉRIES TEMPORAIS

Banca Examinadora:

- Prof. Dr. André Leon Sampaio Gradvohl
  FT/UNICAMP
- Prof. Dr. Ivan Luiz Marques Ricarte
  FT/UNICAMP
- Prof.ª Dra. Elaine Parros Machado de Souza
  ICMC/USP

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Limeira, 22 de agosto de 2018

Agradecimentos

Agradeço primeiramente à DEUS por iluminar o meu caminho. À minha mãe por ser a pessoa que me motivou desde criança a seguir o caminho dos estudos. Ao meu pai por me passar as virtudes de um homem comprometido, honesto e de paz. À minha falecida avó, Rosalina Fialho, por me dar de presente meu primeiro computador e me induzir neste caminho. À minha irmã, simplesmente por ser minha Tata. À minha esposa por me acompanhar na minha jornada, dando o suporte necessário. Ao meu filho, por cada sorriso dado, que me motiva ainda mais. A todos os meus amigos que se envolveram ou me influenciaram de alguma forma com o mestrado, em especial ao meu amigo, o "colombiano", Andrés Bueno que prestou uma ajuda fundamental para a minha pesquisa.

Agradeço especialmente ao meu orientador, Prof. Dr. André Leon Sampaio Gradvohl, pelo acompanhamento desenvolvido, por confiar em mim mesmo nos momentos difíceis, e por todas as oportunidades vislumbradas relacionadas ao meu trabalho. À todos os professores da Faculdade de Tecnologia e da Unicamp com quem cursei as disciplinas desse programa de mestrado. Também aos funcionários da secretaria da pós-graduação da Faculdade de Tecnologia pelo suporte nos tramites burocráticos e auxílios para congressos.

Agradeço também ao Instituto Federal de São Paulo, pelo afastamento concedido e pelos incentivos para cursar os programas de pós-graduação, em especial ao pessoal do campus Hortolândia e do campus Piracicaba.

Resumo

As ferramentas baseadas em aprendizado de máquina têm sido utilizadas para previsão em séries temporais, devido à sua capacidade de identificar relações nos conjuntos de dados sem serem programadas explicitamente para isto. Algumas séries temporais podem ser caracterizadas como fluxos de dados, e consequentemente podem apresentar desvios de conceito, o que traz alguns desafios a mais para as técnicas tradicionais de aprendizado de máquina. A utilização de técnicas de aprendizado *online*, como os algoritmos e *ensembles* derivados do *Online Sequential Extreme Learning Machine* são adequados para previsão em fluxo de dados com desvios de conceito. No entanto, as previsões baseadas em fluxos de dados frequentemente possuem uma séria restrição relacionada ao tempo de execução dos algoritmos, devido à alta taxa de entrada das amostras. O objetivo deste trabalho foi verificar as acelerações no tempo de execução, proporcionadas pela aplicação de técnicas de computação de alto desempenho no algoritmo *Online Sequential Extreme Learning Machine* e em três *ensembles* que o utilizam como base, quando comparadas às respectivas abordagens convencionais. Para tanto, neste trabalho são propostas versões de alto desempenho implementadas em Linguagem C com a biblioteca Intel MKL e com o padrão MPI. A Intel MKL fornece funções que exploram os recursos *multithread* em processadores com vários núcleos, o que também expande o paralelismo para arquiteturas de multiprocessadores. O MPI permite paralelizar as tarefas com memória distribuída em vários processos, que podem ser alocados em um único nó computacional ou distribuídos por vários nós. Em resumo, a proposta deste trabalho consiste em uma paralelização de dois níveis, onde cada modelo do *ensemble* é alocado em um processo MPI e as funções internas de cada modelo são paralelizadas em um conjunto de *threads* por meio da biblioteca Intel MKL. Para os experimentos, foi utilizado um conjunto de dados sintético e outro real com desvios de conceito. Cada conjunto possui em torno de 175.000 instâncias contendo entre 6 e 10 atributos, e um fluxo *online* foi simulado com cerca de 170.000 instâncias. Os resultados experimentais mostraram que, em geral, os *ensembles* de alto desempenho melhoraram o tempo de execução, quando comparados com sua versão serial, com desempenho até 10 vezes mais rápido, mantendo a acurácia das previsões. Os testes foram realizados em três ambientes de alto desempenho distintos e também num ambiente convencional simulando um *desktop* ou um *notebook*.

Palavras-Chave - Computação de alto desempenho, Aprendizado de máquina, Análise de séries temporais.

Abstract

Tools based on machine learning have been used for time series forecasting because of their ability to identify relationships in data sets without being explicitly programmed for it. Some time series can be characterized as data streams, and consequently can present concept drifts, which brings some additional challenges to the traditional techniques of machine learning. The use of online learning techniques, such as algorithms and ensembles derived from the Online Sequential Extreme Learning Machine, are suitable for forecasting data streams with concept drifts. Nevertheless, data streams forecasting often have a serious constraint related to the execution time of the algorithms due to the high incoming samples rate. The objective of this work was to verify the accelerations in the execution time, provided by the adoption of high-performance computing techniques in the Online Sequential Extreme Learning Machine algorithm and in three ensembles that use it as a base, when compared to the respective conventional approaches. For this purpose, we proposed high-performance versions implemented in C programming language with the Intel MKL library and the MPI standard. Intel MKL provides functions that explore the multithread features in multicore CPUs, which expands the parallelism to multiprocessors architectures. MPI allows us to parallelize tasks with distributed memory on several processes, which can be allocated within a single computational node, or distributed over several nodes. In summary, our proposal consists of a two-level parallelization, where we allocated each ensemble model into an MPI process, and we parallelized the internal functions of each model in a set of threads through Intel MKL library. For the experiments, we used a synthetic and a real dataset with concept drifts. Each dataset has around 175,000 instances containing between 6 and 10 attributes, and an online data stream has been simulated with about 170,000 instances. Experimental results showed that, in general, high-performance ensembles improved execution time when compared with its serial version, performing up to 10-fold faster, maintaining the predictions' accuracy. The tests were performed in three distinct high-performance environments and also in a conventional environment simulating a desktop or a notebook.

Keywords – High-performance computing, Machine learning, Time-series analysis.

Lista de Figuras

Lista de Tabelas

Lista de Abreviações e Siglas

AddExp	*Additive Expert*
ADWIN	*Adaptive Windowing*
CUDA	*Compute Unified Device Architecture*
DDM	*Drift Detection Method*
DOER	*Dynamic and On-line Ensemble Regression*
DP	Desvio Padrão
EDDM	*Early Drift Detection Method*
ELM	*Extreme Learning Machine*
EOS	*Ensemble of Online Learners with Substitution of Models*
EOS-ELM	*Ensemble of Online Sequential Extreme Learning Machine*
FEDD	*Feature Extraction for Explicit Concept Drift Detection*
GPGPU	*General Purpose Computing on Graphics Processing Units*
GPU	*Graphics Processing Unit*
HP-ELM	*High-Performance Extreme Learning Machines*
HPI	Hasso Plattner Institut
HYP	Hyperplane
LaSCADo	Laboratório de Simulação e de Computação de Alto Desempenho
MAGMA	*Matrix Algebra on GPU and Multicore Architectures*
MP_{10}	Material Particulado Inalável
MPI	*Message Passing Interface*
MSE	*Mean Squared Error*
NO	Neurônios Ocultos
OAUE	*Online Accuracy Updated Ensemble*
OB	*Online Bagging*

OS-ELM	*Online Sequential Extreme Learning Machine*
PEOS-ELM	*Parallel Ensemble of Online Sequential Extreme Learning Machine*
POS-ELM	*Parallel Online Sequential Extreme Learning Machine*
RDMA	*Remote Direct Memory Access*
RMSE	*Root Mean Squared Error*
RNA	Rede Neural Artificial
SLFN	*Single Layer Feed-forward Networks*
SPMD	*Single Program Multiple Data*
SW	*Sliding Window*
TB	Tamanho do Bloco
TR	Tempo Real

Sumário

Capítulo 1

Introdução

É notória a presença cada vez maior dos sistemas computacionais em nosso cotidiano, que vêm evoluindo nas últimas décadas. Ao nosso redor são inúmeros dispositivos eletrônicos que estão produzindo dados constantemente. Dentro de casa temos *smartphones*, *smartTVs*, computadores portáteis e de mesa, entre outros. Nas ruas e avenidas podemos observar radares de trânsito, câmeras de segurança, parquímetros, sistemas de controle de acesso e segurança, sem contar os automóveis com sistemas de gerenciamento cada vez mais abrangentes e autônomos. Nas indústrias e empresas, nas atividades rurais, órgãos governamentais, ou seja, praticamente toda a sociedade utiliza tais sistemas.

O que talvez não seja tão perceptível ao nosso olhar é a enorme quantidade de dados gerados por tais sistemas. Via de regra, esses dados são gerados, processados e transmitidos com um objetivo específico, de modo que seja possível a extração de informações por meio de relações explícitas organizadas em seu conjunto. Porém, na enorme massa de dados que se tem disponível hoje em dia, também é possível extrair várias informações implícitas (ocultas) desses conjuntos através de algoritmos de Mineração de Dados e Aprendizado de Máquina (HAN; KAMBER, 2006).

Tais algoritmos analisam os conjuntos de dados em busca de padrões de comportamento, de novas relações, de classes distintas, da previsão de algum valor ou comportamento futuro (BRZEZINSKI, 2010), entre outros. Muitas áreas de conhecimento já se utilizam dos algoritmos de mineração de dados e aprendizado de máquina, como no sistema bancário, para prever e evitar fraudes de crédito e no mercado financeiro que os utiliza para previsão do preço de ações e bens (CAVALCANTE; OLIVEIRA, 2015). Para a política de vendas no varejo, esses algoritmos podem ser utilizados para a classificação de clientes, bem como na área de saúde em previsão de custos e diagnósticos assistidos por computador (BRZEZINSKI, 2010).

Além das áreas já citadas, vários setores ligados ao meio ambiente têm utilizado os algoritmos de mineração de dados e aprendizado de máquina na predição de enchentes em rios (YADAV et al., 2016) e também em predições relacionadas à qualidade do ar (SOUZA et al., 2015; BUENO; COELHO; BERTINI, 2017). Inclusive, a predição da qualidade do ar é hoje um tema em alta evidência devido à poluição do ar urbano e, especificamente, a ocorrência de episódios de poluição, ou seja, concentrações elevadas de poluentes que causam efeitos nocivos para a saúde e até mesmo mortes prematuras

entre grupos sensíveis (crianças, idosos e pessoas com doenças respiratórias e cardíacas) (CETESB, 2016).

Os conjuntos de dados das áreas mencionadas podem ser considerados como Séries Temporais quando correspondem a uma coleção de valores avaliados periodicamente. Tais conjuntos também podem ser considerados Fluxo de Dados (em inglês *Data Streams*), quando se tratam de uma sequência ordenada de instâncias que chegam com grande vazão (número de instâncias por unidade de tempo). Considerando essas características, os sistemas que operam em fluxos de dados não podem lidar com muitas instâncias na memória principal por um longo período (CAVALCANTE; OLIVEIRA, 2015), ou eles podem sofrer uma sobrecarga de memória. Devido a essa limitação, é preciso processamento em tempo real para lidar com fluxos de dados.

Além disso, os fluxos de dados possuem uma natureza dinâmica, e. g. as distribuições de dados subjacentes desses fluxos tendem a mudar ao longo do tempo. Esse fenômeno, conhecido na literatura como Desvio de Conceito (em inglês *Concept Drift*), consiste numa alteração da relação entre as entradas e a variável alvo a ser prevista, que pode ocorrer ao longo do tempo (GAMA et al., 2014).

A maior parte dos algoritmos tradicionais de mineração de dados trabalham *offline*, isto é, possuem todo o conjunto de dados armazenado e à disposição para acessá-lo quantas vezes necessário. Considerando que as características inerentes a fluxo de dados inviabilizam o armazenamento de todas as instâncias, cabe ressaltar a necessidade da utilização de algoritmos de aprendizado *online* e de tempo real para a tarefa de previsão em fluxo de dados. As abordagens de aprendizado *online* também podem lidar com desvios de conceito, devido à capacidade de aprender novos padrões de dados ao longo do tempo.

Algoritmos como o *Online Sequential Extreme Learning Machine* (OS-ELM) (LIANG et al., 2006), possuem a habilidade de atualizar o modelo de previsão de acordo com a chegada das novas amostras e são facilmente paralelizáveis por operarem em estruturas matriciais. Essas características tornam o OS-ELM capaz de lidar com desvios de conceito de forma implícita, além de possibilitar a aceleração do seu processamento. Além disso o OS-ELM pode operar em *Ensembles*.

Um *ensemble* de modelos de previsão é composto por uma combinação de modelos individuais, com o objetivo de criar um modelo de melhorado (HAN; KAMBER, 2006). Vários *ensembles* presentes na literatura podem operar com modelos OS-ELM (LAN; SOH; HUANG, 2009; SOARES; ARAÚJO, 2015a; BUENO; COELHO; BERTINI, 2017; KOLTER; MALOOF, 2005). Os *ensembles* de modelos OS-ELM tendem a apresentar melhorias na estabilidade e acurácia das previsões. No entanto, o uso dos *ensembles* também tende a aumentar o tempo de execução.

Sendo assim, neste trabalho propõe-se a aplicação de técnicas de Computação de Alto Desempenho em três *ensembles* com modelos OS-ELM, que os possibilitem executar em paralelo em processadores com vários núcleos e também em *clusters* com vários nós computacionais. A intenção é reduzir o tempo de execução sem prejudicar as tarefas de aprendizado necessárias e a acurácia das previsões.

1.1 Objetivos

O objetivo deste trabalho é verificar as acelerações proporcionadas pelas técnicas de computação de alto desempenho no tempo de execução do OS-ELM e de diferentes *ensembles* que o utilizam com base. Neste trabalho foram consideradas tarefas de previsão de séries temporais e de regressão em fluxos de dados dinâmicos, com desvios de conceito, simulados com aproximadamente 175.000 instancias contendo entre 6 e 10 atributos. As soluções que utilizam técnicas de alto desempenho foram comparadas com suas respectivas versões convencionais para avaliar ganhos ou perdas em diferentes ambientes de execução.

1.2 Objetivos Específicos

Para alcançar o objetivo principal, os seguintes objetivos específicos foram propostos:

- Caracterizar um conjunto de dados sintético, simulando um fluxo de dados com desvios de conceito, para ser utilizado como *benchmark* na execução dos algoritmos;
- Caracterizar um conjunto de dados ambientais, simulando um fluxo de dados real, com concentrações de Material Particulado Inalável (MP_{10}) no ar, para verificar o desempenho dos algoritmos com um conjunto real;
- Implementar versões paralelas do OS-ELM com Janelas Deslizantes em diferentes arquiteturas, para verificar as mais adequadas aos problemas propostos.
- Implementar *ensembles* seriais e paralelos do OS-ELM, para verificar seu comportamento em diferentes ambientes de execução.

1.3 Estrutura do texto

Esse texto está organizado nos seguintes capítulos:

- O Capítulo 2 apresenta um referencial teórico dos principais conceitos relacionados a fluxo de dados, desvio de conceito e programação paralela, e também descreve os algoritmos *Extreme Learning Machine* (ELM) (HUANG et al., 2006) e OS-ELM.
- O Capítulo 3 apresenta um levantamento bibliográfico dos principais trabalhos relacionados a esta dissertação de mestrado.
- O Capítulo 4 define as abordagens implementadas.
- A metodologia adotada, com os conjuntos de dados utilizados e as configurações iniciais dos algoritmos, bem como os resultados experimentais e as discussões são abordados no Capítulo 5.
- O Capítulo 6 aponta as conclusões e possíveis direções para trabalhos futuros.
- O texto ainda conta com 2 apêndices que tratam respectivamente de alguns resultados preliminares e de resultados complementares.

Capítulo 2

Referencial Teórico

Neste capítulo são abordados os principais conceitos relacionados a este trabalho e esclarecidos os conceitos chave necessários para os próximos capítulos.

As primeiras seções deste capítulo apresentam as definições de Séries Temporais, Fluxo de Dados e Desvio de Conceito, para depois serem apresentados os requisitos de mineração de dados em um ambiente dinâmico. Em seguida, é apresentada uma breve revisão do algoritmo ELM, de seu variante OS-ELM e também a definição de *Ensembles* de modelos de previsão. Posteriormente, na última seção, serão apresentados os conceitos referentes às técnicas de Computação de Alto Desempenho com Programação Paralela.

2.1 Séries Temporais

Um conjunto de dados de Séries Temporais consiste em sequências de valores ou eventos obtidos ao longo de repetidas medições de tempo, normalmente em intervalos de tempo iguais. Esses intervalos de tempos podem ser a cada segundo, a cada minuto, a cada dia ou outro período qualquer (HAN; KAMBER, 2006). Um conjunto de dados de séries temporais também é um conjunto de sequência. A diferença é a noção concreta de tempo que está presente nas séries temporais e que pode estar ausente num simples conjunto de sequência.

Vários processos dinâmicos podem ser modelados como uma série temporal. Exemplos desses processos são os preços de ações, as vendas mensais de uma empresa, as temperaturas de uma cidade (CAVALCANTE; OLIVEIRA, 2015) e os dados de poluição ambiental, entre outros. Uma série temporal envolvendo uma variável Y, representando, por exemplo, a quantidade de um determinado poluente no ar, pode ser vista como uma função do tempo t, isto é, $Y = F(t)$. Esta função pode ser ilustrada como um gráfico de séries temporais, como mostrado na Figura 2.1 (HAN; KAMBER, 2006).

Em aplicações de monitoramento e controle, os dados são frequentemente apresentados em forma de séries temporais. As duas tarefas de aprendizado mais típicas são a previsão de séries temporais (tarefa de regressão) ou a detecção de anomalias (tarefa de classificação) (GAMA et al., 2014).

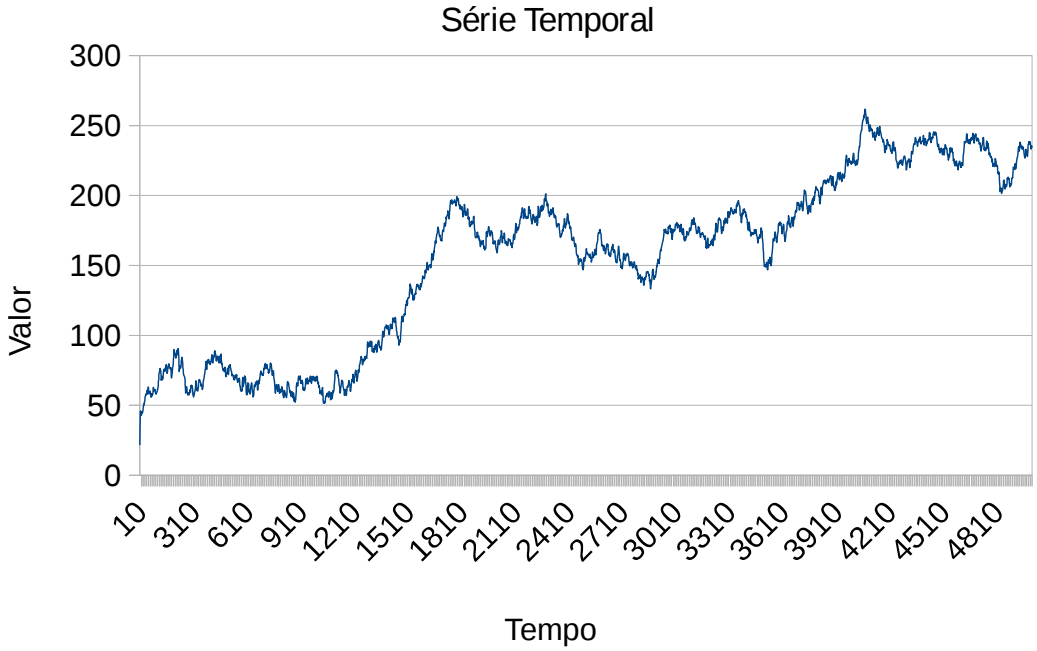

Figura 2.1: Exemplo de Série Temporal.

2.2 Fluxo de Dados

Segundo Brzezinski (2010), fluxo de dados é uma sequência ordenada de instâncias que chegam a uma vazão que não permite armazená-las permanentemente na memória. Os fluxos de dados também podem ser vistos de forma equivalente a um vetor de valores inteiros, categóricos ou gráficos, estruturados ou não-estruturados (PHRIDVIRAJ; GURURAO, 2014).

Exemplos típicos de fluxo de dados incluem dados científicos, de engenharia, e séries temporais como o fornecimento de energia, o tráfego de rede, as transações na bolsa de valores, os dados de telecomunicações, as sequências de cliques na *web*, os dados de sistema de vídeo-monitoramento e monitoramento climático ou ambiental (HAN; KAMBER, 2006). As séries temporais, que são uma sequência numérica indexada num intervalo de tempo, podem ser consideradas um tipo especial de fluxo de dados (CAVALCANTE; OLIVEIRA, 2015), quando são obtidas em altas taxas ou possuem uma vazão elevada.

De acordo com Han e Kamber (2006) e Brzezinski (2010), a maioria dos fluxos de dados possuem as seguintes características:

1. apresentam volumes de dados enormes ou possivelmente infinitos;
2. permitem um pequeno número de análises em cada instância;
3. exigem um tempo de resposta rápido (muitas vezes em tempo real);
4. possibilitam que a distribuição probabilística geradora das instâncias mude a qualquer momento.

Essas características podem se tornar limitações aos tradicionais algoritmos de mineração de dados que funcionam a partir de uma base de dados estática, com os dados

acessíveis a qualquer momento. A primeira característica torna evidente que seria impossível armazenar na memória principal todos os dados que chegam pelo fluxo, já que quantidade de memória disponível se tornaria insuficiente em algum momento.

Isso acarreta na segunda e na terceira características, uma vez que, se o tempo de resposta do algoritmo que for processar essas instâncias for maior do que o tempo de chegada entre as instâncias, a tendência seria de um crescimento constante da memória e a eventual extrapolação da sua capacidade. Além disso, um dos desafios em lidar com fluxo de dados é que a geração das instâncias pode ocorrer em intervalos de tempo irregulares (PHRIDVIRAJ; GURURAO, 2014), exceto em séries temporais.

Tais características podem levar à necessidade de se utilizar técnicas de sumarização dos dados, que produzem respostas aproximadas a partir de grandes conjuntos de dados. São algumas dessas técnicas: a Amostragem, o *Sketching*, os Histogramas, e as *Wavelets* (HAN; KAMBER, 2006; LAKSHMI; REDDY, 2010).

A quarta característica, isto é, a possível mudança da distribuição probabilística geradora das instâncias, é abordada de maneiras distintas a depender da aplicação. Muitas das primeiras abordagens de mineração em fluxo de dados ignoravam essa característica e atuavam como algoritmos de aprendizado de máquina estáticos. Porém, esse dinamismo da quarta característica implica que os padrões em fluxo de dados podem evoluir ao longo do tempo e introduz um grande desafio aos algoritmos tradicionais de aprendizado *offline*, que é a capacidade de manter permanentemente um modelo de decisão preciso (CAVALCANTE; MINKU; OLIVEIRA, 2016).

Essa situação fez com que novas pesquisas considerassem a quarta característica como ponto-chave e dessem foco ao "aprendizado de máquina evolutivo em fluxo de dados". A Tabela 2.1 compara o processamento de uma base comum e de um fluxo de dados.

Tabela 2.1: Processamento em bases comuns e em Fluxo de Dados, adaptado de PhridviRaj e GuruRao (2014)

Parâmetro	**Base Comum**	**Fluxo de Dados**
Acesso aos dados	Sequencial ou aleatório.	Sequencial.
Memória Disponível	Flexível.	Limitada.
Resultados computacionais	Preciso.	Aproximado.
Análises em cada instância	Flexível.	Limitada a uma ou poucas verificações
Algoritmos	O tempo de processamento não é restrição	O tempo de processamento é limitado pela velocidade do fluxo.
Velocidade dos dados	Pode ser ignorada.	A taxa de chegada dos dados pode ser maior do que taxa de processamento.
Modelagem dos dados	Persistente.	Modelado como fluxos de dados transitórios.
Esquema dos dados	Estático.	Dinâmico.

Como a natureza dos fluxo de dados é dinâmica, de acordo com a quarta característica, os padrões de dados evoluem ao longo do tempo, degradando a precisão de modelos de previsão baseados em fluxo de dados. Este fenômeno é chamado de Desvio de Conceito.

2.3 Desvio de Conceito

Como mencionado na seção anterior, a distribuição probabilística que gera as instâncias de um fluxo de dados pode mudar ao longo do tempo. Essas mudanças, dependendo da área de pesquisa, são referidas como evolução temporal, mudança covariável, não estacionaridade ou desvio de conceito (BRZEZINSKI, 2010).

Desvio de conceito, em aprendizado de máquinas e mineração de dados, refere-se às situações em que a relação entre os dados de entrada e a variável-alvo que o modelo está tentando prever muda ao longo do tempo de forma imprevista (BOSE et al., 2014), fazendo como que a precisão das previsões possa se degradar ao longo do tempo, e. g. critérios para a concessão de crédito podem variar inesperadamente conforme o nível de atividade econômica. Uma definição formal para desvio de conceito é apresentada a seguir.

Em aprendizado de máquina indutivo, o problema de aprendizagem supervisionada é definido da seguinte forma. Pretende-se prever uma variável alvo $y \in \mathbb{R}^1$ em tarefas de regressão, dado um conjunto de atributos de entrada de uma instância $X \in \mathbb{R}^p$.

Um exemplo é uma relação (X, y) em que X é um conjunto de leituras de sensores em p instantes de tempo e y é a leitura no instante de tempo seguinte. Na fase de treinamento, onde é realizada a construção dos modelos, tanto X quanto y são conhecidos. Porém, nos novos exemplos, nos quais o modelo preditivo é aplicado, X é conhecido, mas y não é conhecido no momento da predição (GAMA et al., 2014).

De acordo com as terminologias utilizadas nos parágrafos anteriores, um desvio de conceito pode ser descrito a partir da probabilidade posterior da variável alvo y em função das características de X (simbolicamente, $p(y|X)$), conforme o teorema de Bayes, descrito na Equação 2.1 a seguir:

$$p(y|X) = \frac{p(y)\, p(X|y)}{p(X)} \tag{2.1}$$

em que $p(X)$ corresponde à probabilidade das instâncias de entrada conforme suas características, $p(y)$ define a probabilidade marginal da variável alvo e $p(X|y)$ descreve a probabilidade de X dentro de um conjunto de resultados possíveis.

Nesse contexto, um desvio de conceito entre um instante de tempo t_0 e um instante de tempo t_1 é definido conforme a Equação 2.2 a seguir:

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y) \tag{2.2}$$

em que p_{t_0} denota a distribuição conjunta no tempo t_0 entre o conjunto de variáveis de entrada X e a variável alvo y (GAMA et al., 2014).

2.3.1 Tipos de Desvio de Conceito

Os desvios de conceito podem ocorrer quando as probabilidades marginais da variável alvo $p(y)$ mudam com o tempo; quando as probabilidades condicionais $p(X|y)$ mudam; e quando as probabilidades posteriores $p(y|X)$ mudam, afetando a predição do modelo (GAMA et al., 2014; KHAMASSI et al., 2016). Diante dessas possibilidades é possível distinguir dois tipos de desvio de conceito:

1. **Desvio de Conceito Real**: refere-se às mudanças na probabilidade *a posteriori* $p(y|X)$. Tais mudanças podem ocorrer com ou sem alteração da $p(X)$ (GAMA et al., 2014). Esse tipo de desvio afeta os limites de decisão.

2. **Desvio de Conceito Virtual**: refere-se a mudanças na distribuição dos dados de entrada (e. g. $P(X)$ varia) (GAMA et al., 2014) e na probabilidade condicional $p(X|y)$ (KHAMASSI et al., 2016) que não afetam a probabilidade *a posteriori* $p(y|X)$, e consequentemente os limites de decisão.

A Figura 2.2 ilustra alguns tipos de desvio de conceito. Essa figura mostra que apenas o desvio de conceito real altera a fronteira de decisão, e o modelo de decisão anterior torna-se obsoleto. Na prática, o desvio de conceito virtual, que altera as probabilidades marginais pode aparecer combinado com o desvio de conceito real. Nesses casos, a fronteira de decisão também é afetada (GAMA et al., 2014).

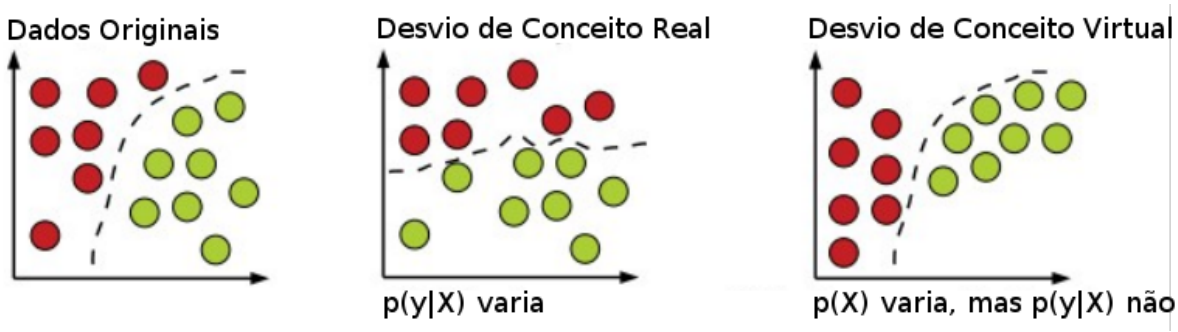

Figura 2.2: Desvio de Conceito Real e Virtual, adaptado de Gama et al. (2014).

Khamassi et al. (2016) ainda definem um terceiro tipo de desvio de conceito, intitulado de "Desvio de Conceito de Classe Anterior". Específico para tarefas de classificação, esse tipo refere-se às mudanças em $p(y)$, que pode significar um desequilíbrio de classe, surgimento de uma classe nova ou fusão de classes existentes.

2.3.2 Principais Características dos Desvios de Conceito

Vários autores também caracterizam os desvios de conceito de acordo com a sua ocorrência em função do tempo. Dongre e Malik (2014), Gama et al. (2014) e Bose et al. (2014) consideram as seguintes categorias que estão ilustradas na Figura 2.3:

- Desvio abrupto ou repentino: Ocorre de forma instantânea causando uma imediata mudança de conceito. Tem o poder de degradar rapidamente o modelo preditor, porém é de mais fácil detecção.

- Desvio incremental: Ocorre de forma lenta e crescente, causando erros pequenos que tendem a crescer constantemente. São mais difíceis de detectar.

- Desvio gradual: Apresenta uma variação de conceito num certo período de tempo, até que a troca efetiva seja feita.

- Desvio recorrente: Demonstra uma troca temporária de conceito, que volta a ser o mesmo após um período de tempo. Tal recorrência pode ser periódica ou aleatória.

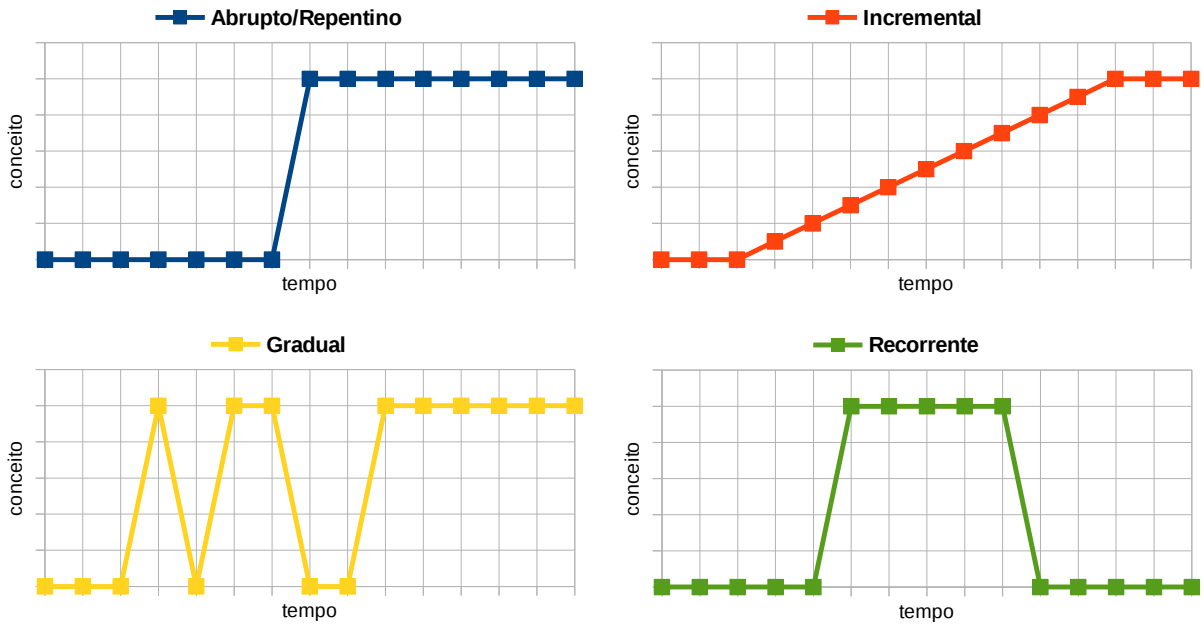

Figura 2.3: Principais características dos Desvios de Conceito, adaptado de Brzezinski (2010) e Gama et al. (2014).

2.4 Mineração em Fluxos com Desvios de Conceito

Com base nas discussões sobre fluxo de dados e desvio de conceito realizada nas seções anteriores, pode-se observar que há muito mais desafios para que um algoritmo de aprendizado de máquina consiga extrair ou aprender padrões em ambientes dinâmicos.

Basicamente, os algoritmos para previsão de fluxo de dados devem ser capazes de realizar uma tarefa de aprendizado sequencial, de acordo com a chegada dos dados, já que a totalidade dos dados nunca estará disponível. Isso pode ser resolvido adotando um algoritmo de aprendizado *online* (BRZEZINSKI, 2010). Da mesma forma, esses algoritmos devem ser capazes de reagir à ocorrência de desvios de conceito, o que pode ser solucionado utilizando parâmetros de esquecimento por meio de soluções ativas (e. g. utilizando detectores de desvios) ou passivas (e. g. sem utilizar detectores de desvios).

2.4.1 Algoritmos de aprendizado *online*

Em mineração de dados e aprendizado de máquina dois modos de aprendizagem podem ser identificados: aprendizagem *offline* e aprendizagem *online*. Em modelo de aprendizado indutivo *offline*, todos os dados de treinamento devem estar disponíveis no momento do treinamento do modelo. Somente depois de concluído o treinamento é que o modelo pode ser usado para realizar previsões (GAMA et al., 2014).

Em contraste, os algoritmos de aprendizado *online* processam os dados de forma sequencial à medida em que esses dados são apresentados ao algoritmo. Esses algoritmos são, portanto, adequados para fluxos de dados. Nesse caso, é posto em operação um modelo gerado sem que seja necessário possuir todo o conjunto de treinamento de antemão. Dessa forma, o modelo deve ser continuamente atualizado durante sua operação,

ou seja, à medida que os novos dados vão chegando, o conjunto de treinamento vai sendo aperfeiçoado (GAMA et al., 2014).

Um bom modelo de aprendizado *online* deve possuir uma série de requisitos, dentre os quais estão os seguintes (BRZEZINSKI, 2010):

- **Atuar de forma incremental**: ser capaz de ler instância após instância, de acordo com a sua chegada ou de ler blocos de instâncias, sumarizados ou não;
- **Atuar em poucos passos ou passo único**: devido à necessidade de processar cada instância em tempo hábil, não é possível fazer muitas análises sobre a mesma;
- **Utilizar tempo e memória limitados**: Cada instância deve ser processada em um tempo pequeno, independentemente do número de instâncias já processadas e deve consumir pouca memória.

2.4.2 Métodos para reagir à ocorrência de Desvios de Conceito

Há diversos métodos para lidar com a ocorrência de desvios de conceito. As revisões de literatura mais atuais a respeito do assunto estão nos trabalhos de Gama et al. (2014) e Khamassi et al. (2016). Além disso, em outros trabalhos recentes, que abordam modelos de previsão/regressão baseados na aprendizagem de máquinas, os autores categorizam os métodos de detecção de desvio de conceito, conforme segue (CAVALCANTE; MINKU; OLIVEIRA, 2016; SOARES; ARAÚJO, 2015a):

1. **Método Explícito:** Utiliza técnicas de detecção de desvios e o modelo de previsão é retreinado quando um Desvio de Conceito é identificado.
2. **Método Implícito:** Não utiliza técnicas de detecção de desvios e o modelo é atualizado continuamente conforme a chegada das instâncias.

O método Explícito monitora algumas estatísticas do fluxo de dados, a fim de detectar desvios no conceito e alarmar o modelo de aprendizado caso ele necessite ser refeito ou atualizado (DONGRE; MALIK, 2014). Uma vantagem da detecção de desvios explícita é que ela funciona como uma caixa branca, informando o usuário sobre a ocorrência de desvios de conceito. Bons exemplos são os algoritmos *Drift Detection Method* (DDM) (GAMA et al., 2004), um dos detectores de desvios mais conhecidos e citados da literatura, e sua variante, o *Early Drift Detection Method* (EDDM) (BAENA-GARCIA et al., 2006). Também há outros detectores mais recentes, como o *Feature Extraction for Explicit Concept Drift Detection* (FEDD), um detector de desvios explícito e *online* voltado para séries temporais proposto por Cavalcante, Minku e Oliveira (2016).

Já o método Implícito não realiza técnicas para detectar quando ocorre o início de um desvio de conceito. Esse método se baseia na utilização de algoritmos de aprendizado *online*, que aprendem constantemente com o ambiente, ajustando e construindo o conhecimento (SOARES; ARAÚJO, 2015a) independente da ocorrência de um desvio de conceito. As principais questões relativas ao método Implícito dizem respeito ao potencial consumo de recursos para atualizar o modelo a todo intervalo de tempo, mesmo quando

os dados recebidos pertencem ao mesmo conceito, bem como a potencial de sobrecarga do modelo devido ao seu retreino constante (CAVALCANTE; MINKU; OLIVEIRA, 2016). Algumas técnicas para mitigar esses efeitos são comumente adotadas conforme descrito por Cavalcante e Oliveira (2015) e Soares e Araújo (2015a):

- **Seleção de Instâncias (Janelas Deslizantes)**: um conjunto de amostras relevantes do conceito atual é selecionado para construir ou adaptar o modelo (SOARES; ARAÚJO, 2015a). Uma técnica comum é a de janelas deslizantes, que fornecem uma forma de limitar a quantidade de instâncias introduzidas para o modelo, eliminando assim os dados que vêm de um conceito antigo (DONGRE; MALIK, 2014). Uma questão importante em janelas deslizantes é adequar o tamanho das janelas. Janelas pequenas tendem a se adaptar mais rapidamente, porém podem levar a uma perda de acurácia do modelo em fases mais estáveis. Em contraponto, janelas grandes tendem a ser mais estáveis, mas falham ao não se adaptar de forma rápida o suficiente em mudanças no ambiente (SOARES; ARAÚJO, 2015a; DONGRE; MALIK, 2014). Diante desses problemas, já foram propostos mecanismos de janelas dinâmicas como o *Adaptive Windowing* (ADWIN) (BIFET; GAVALDA, 2007).

- **Ponderação de Instâncias**: As instâncias são ponderadas conforme a sua relevância para o conceito atual, e. g. Métodos Recursivos (SOARES; ARAÚJO, 2015a). Essa estratégia geralmente considera que amostras mais recentes podem conter informações mais relevantes para a previsão (CAVALCANTE; OLIVEIRA, 2015). A mesma técnica pode ser utilizada para ponderar as janelas deslizantes.

- ***Ensembles***: são conjuntos de modelos (componentes) cujas decisões são combinadas por uma regra, sendo que essa decisão geralmente é mais precisa do que a dada por um único componente (DONGRE; MALIK, 2014). São descritos com mais detalhes na Seção 2.7.

Após analisar os desafios impostos e os métodos adequados à mineração de dados em fluxo de dados com desvios de conceito, a próxima seção introduz o conceito de Redes Neurais Artificiais (RNA), do qual são derivados os algoritmos ELM e OS-ELM, apresentados nas seções subsequentes. Esses algoritmos podem ser adequados para a tarefa de previsão em séries temporais e em fluxo de dados com desvios de conceito, a partir da adoção de algumas estratégias e métodos abordados nesta seção.

2.5 Redes Neurais Artificiais

Uma Rede Neural Artificial é um conjunto de unidades de entrada e saída conectadas, em que cada conexão pode ser considerada um neurônio com um peso associado a ele. Durante a fase de aprendizado, a rede aprende os conceitos ajustando os pesos dos neurônios para que possa prever o valor futuro das instâncias de entrada (HAN; KAMBER, 2006).

Uma RNA funciona como um sistema de processamento de informações que compreende muitos elementos de processamento não-lineares e interconectados, na forma de camadas conectadas por pesos. Em geral, uma RNA possui três camadas: a camada de

entrada de dados na rede; a camada oculta para o processamento de dados; e a camada de saída para gerar os resultados em função da entrada fornecida (YADAV et al., 2016).

A Figura 2.4 ilustra uma Rede Neural de Camada Única do tipo *Feed-forward* (em inglês *Single Layer Feed-forward Networks* – SLFN).

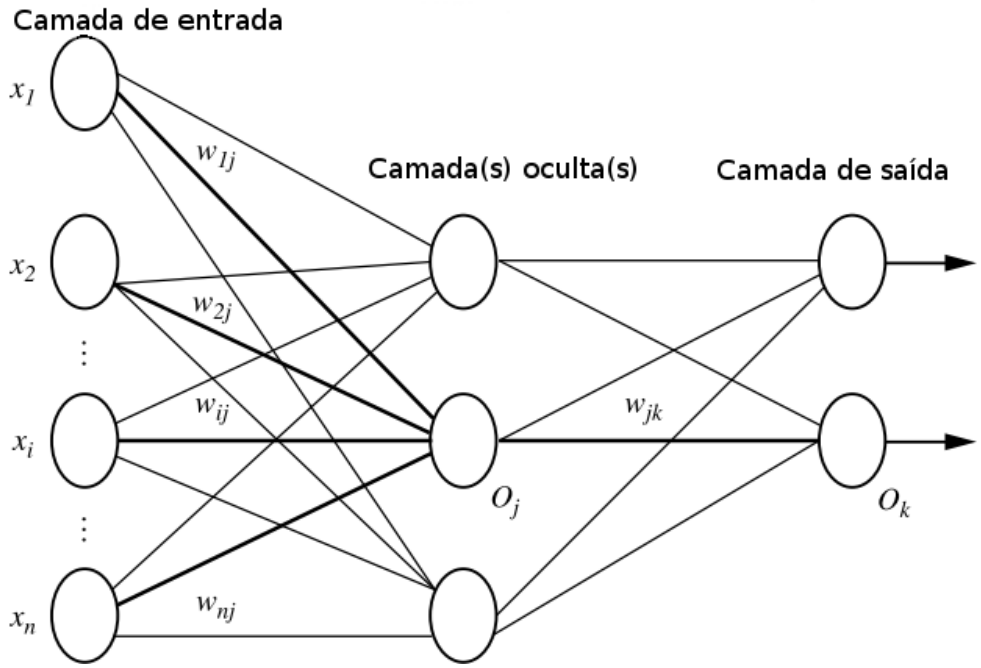

Figura 2.4: Rede Neural Artificial de camada única, adaptado de Han e Kamber (2006).

Na Figura 2.4, $[x_1, x_2, x_3, \ldots, x_n]$ são os atributos das instâncias de entrada da rede, $[w_{1j}, w_{2j}, w_{3j}, \ldots, w_{nj}]$ são os pesos da conexão de cada atributo em relação aos neurônios da camada oculta O_j e w_{jk} são os pesos de saída dos neurônios da camada oculta O_j em relação aos neurônios da camada de saída O_k.

A rede é nomeada *feed-forward*, pois nenhum dos ciclos volta para a camada de entrada ou para a camada de saída de uma camada anterior (HAN; KAMBER, 2006). Ela também é de camada única pois possui apenas uma camada oculta. Um pequeno número de neurônios na camada oculta pode afetar o processo de aprendizado, enquanto um grande número pode aumentar o tempo de computação e a rede pode sofrer um super-treino (YADAV et al., 2016), perdendo a capacidade de generalização do aprendizado.

Uma RNA pode lidar com um amplo escopo de tarefas, como agrupamento de dados, classificação, regressão e memória associativa, entre outras. As RNAs dedicadas a lidar com problemas de regressão são particularmente relevantes para o contexto deste trabalho, pois eles são conhecidos por conduzir a bons resultados em problemas de previsão de séries temporais (SOUZA et al., 2015).

Além disso, os algoritmos de RNAs são facilmente paralelizáveis com técnicas de Computação de Alto Desempenho. Essas técnicas podem ser usadas para acelerar o processamento das redes, diminuindo o tempo de execução dos algoritmos. Esses fatores contribuem para a utilidade de redes neurais para classificação e regressão em mineração de dados e aprendizado de máquina (HAN; KAMBER, 2006).

2.6 *Extreme Learning Machines*

Esta seção introduz o algoritmo *Extreme Learning Machine* (ELM) (HUANG et al., 2006) e sua variante *Online Sequential Extreme Learning Machine* (OS-ELM) (LIANG et al., 2006). Esses algoritmos consistem uma SLFN com neurônios ocultos, que não precisam ser treinados e recebem valores aleatórios.

O algoritmo ELM inicialmente proposto trabalha em modo *offline*, ou seja, necessita de todo o conjunto de treinamento e testes previamente para operar. Já sua variante, o OS-ELM, trabalha de modo *online* e incremental. O OS-ELM vem sendo relacionado em trabalhos ligados à previsão de fluxo de dados com desvio de conceito, devido ao bom desempenho em questões como tempo de processamento e precisão do modelo.

As abordagens adotadas nesses trabalhos utilizam mecanismos explícitos como detectores de desvios (CAVALCANTE; OLIVEIRA, 2015), ou mecanismos implícitos como adoção de *ensembles* (SOARES; ARAÚJO, 2015a).

2.6.1 *Extreme Learning Machine* padrão

O algoritmo ELM é uma eficiente estrutura de aprendizagem derivada de uma RNA do tipo SLFN (HUANG et al., 2006). Por utilizar pesos aleatórios entre a camada de entrada e a camada oculta, o ELM tem um tempo de treinamento relativamente rápido se comparado às RNAs convencionais, que precisam ajustar esses pesos iterativamente a cada mudança de camada.

Os parâmetros de entrada do algoritmo são: o conjunto de dados $\mathbf{D}$, o número de Neurônios Ocultos (NO) L e a função de ativação $g(x)$. Considerando que $\mathbf{D} = \{(\mathbf{x}_t, y_t)|t = 1, \ldots, T\}$ com T instâncias distintas, onde $\mathbf{x}_t = [x_{t_1}, x_{t_2}, \ldots, x_{t_r}]^T \in \mathbf{R}^r$ é um vetor com r atributos de uma instância e $\mathbf{y}_t = [y_{t_1}, y_{t_2}, \ldots, y_{t_m}]^T \in \mathbf{R}^m$ é a variável alvo, um ELM padrão com $L \leq T$ NOs e função de ativação $g(x)$ (como a função sigmóide: $g(x) = \frac{1}{(1+e^{(-x)})}$), que pode aproximar as T instâncias de $\mathbf{D}$ com erro zero, é matematicamente modelado conforme a Equação 2.3 (HUANG et al., 2006):

$$f_L(\mathbf{x}_t) = \sum_{j=1}^{L} \beta_j g(\mathbf{a}_j \cdot \mathbf{x}_t + b_j) = \mathbf{y}_t \qquad para\ t = 1, \ldots, T. \tag{2.3}$$

em que $\mathbf{a}_j = [a_{j_1}, a_{j_2}, \ldots, a_{j_r}]^T$ é o vetor com pesos aleatórios que conecta os r neurônios de entrada com o j-ésimo NO, $j = 1, \ldots, L$; $\beta_j = [\beta_1, \beta_2, \ldots, \beta_j]$ é o vetor de pesos que conecta o j-ésimo NO com os neurônios de saída; b_j é o limiar do j-ésimo NO; e $a_j \cdot x_t$ denota o produto interno de a_j e x_t.

As T equações anteriores podem ser escritas de forma compacta (HUANG et al., 2006) conforme as equações 2.4, 2.5 e 2.6 seguir:

$$\mathbf{H}\beta = \mathbf{Y}, \tag{2.4}$$

em que

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{a}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(\mathbf{a}_1 \cdot \mathbf{x}_T + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_T + b_L) \end{bmatrix}_{T \times L}, \tag{2.5}$$

$$\beta = \left[\beta_1^T, \cdots, \beta_L^T\right]_L \quad e \quad \mathbf{Y} = \left[\mathbf{y}_1^T, \cdots, \mathbf{y}_T^T\right]_T, \tag{2.6}$$

sendo $\mathbf{Y}$ a saída que contém as variáveis alvo. $\mathbf{H}$ é chamada de "matriz de saída da camada oculta da rede neural", em que a j-ésima coluna de $\mathbf{H}$ é a saída do j-ésimo NO em relação às entradas $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$ (HUANG et al., 2006); e a t-ésima linha de $\mathbf{H}$ é o vetor de saída da camada oculta com respeito a $\mathbf{x}_t$.

Como os pesos de entrada e os limiares são gerados aleatoriamente no ELM, a geração do modelo preditor a partir do conjunto de treinamento é baseada na solução do vetor β em função da matriz $\mathbf{H}$, gerada na camada oculta, e das saídas y conhecidas, conforme a Equação 2.4. Dessa forma, no conjunto de testes o modelo pode usar o β encontrado para fazer a predição e comparar o erro com a saída conhecida.

De acordo com Huang et al. (2006), quando o número de NOs for igual ao número de instâncias, $L = T$, a matriz $\mathbf{H}$ é quadrada e inversível se os pesos de entrada $\mathbf{a}_j$ e os limiares ocultos b_j são escolhidos de maneira aleatória. Sendo assim as SLFNs podem aproximar essas instâncias de treinamento com erro zero. Ocorre que, normalmente, o número de NOs é bem menor do que o número de instâncias de treinamento distintas, $L \ll T$, o que resulta numa matriz $\mathbf{H}$ não-quadrada, de forma que não pode existir $\mathbf{a}_j, b_j, \beta_j (j = 1, \ldots, L)$, tal que $\mathbf{H}\beta = \mathbf{Y}$. Tal solução para β pode ser determinada utilizando o Método dos Mínimos Quadrados. A menor solução de mínimos quadrados normais do sistema linear mencionado é mostrado na Equação 2.7 a seguir:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{Y}, \tag{2.7}$$

em que $\mathbf{H}^{\dagger}$ é a inversa generalizada de Moore-Penrose da matriz $\mathbf{H}$.

Existem várias maneiras de calcular a inversa generalizada de Moore-Penrose de uma matriz, como o Método de Projeção Ortogonal e a Decomposição de Valores Singulares (*Singular Value Decomposition* – SVD), entre outros. Neste trabalho será considerado o Método de Projeção Ortogonal, em que o algoritmo OS-ELM (LIANG et al., 2006) é baseado. Sendo assim, através do Método de Projeção Ortogonal, a matriz $\mathbf{H}^{\dagger}$ pode ser calculada conforme a Equação 2.8 a seguir:

$$\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T \tag{2.8}$$

quando $\mathbf{H}^T\mathbf{H}$ é não-singular. Substituindo a Equação 2.7 na Equação 2.8, então β pode ser descrita pela Equação 2.9 a seguir:

$$\beta = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y} \tag{2.9}$$

Para obter uma solução mais estável, evitando uma matriz $\mathbf{H}^T\mathbf{H}$ singular ou mal dimensionada, adicionou-se um parâmetro de corte ($\frac{1}{\lambda}$) a essa diagonal, como em Huang et al. (2012) e Krawczyk (2016). Assim, pode-se calcular β de acordo com a Equação 2.10.

$$\beta = (\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y} \tag{2.10}$$

em que $\mathbf{I}$ é uma matriz identidade com o tamanho de $\mathbf{H}$.

O modelo de aprendizado ELM é resumido no Algoritmo 1 a seguir.

Algoritmo 1: Modelo de aprendizado ELM

Entrada: Conjunto de dados $\mathbf{D} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; função de ativação $g(x)$; nº de neurônios ocultos (L) (onde $L \leq T$);
Saída: Erro de previsão; Tempo de execução;

```
início
    Aloque aleatoriamente os pesos de entrada a_j e os limiares b_j, j = 1, ..., L ;
    Calcule a matriz da camada oculta H com D conforme a Equação 2.5;
    Calcule o vetor β com os pesos de saída através da Equação 2.10;
    Calcule o erro de previsão e o tempo de execução do treinamento;
fim
```

Analisando o Algoritmo 1 é possível observar que o ELM ainda não cumpre os requisitos para mineração de fluxo de dados, pois o aprendizado ocorre apenas no início e o modelo necessita de todo o conjunto de dados para treinar. Apesar de o algoritmo não ser explicitamente de passo único, se trata de um rede neural de camada única que não precisa ser ajustada, o que consome menos tempo do que as redes neurais tradicionais.

2.6.2 *Online Sequential Extreme Learning Machine*

Quando se trabalha com fluxo de dados é impossível ter todo o conjunto de dados disponível de uma vez para a realização do treinamento. Nesse caso, as instâncias chegam sequencialmente, uma a uma, podendo chegar também bloco a bloco, em blocos que agregam várias instâncias. Nessas circunstâncias, o ELM padrão se torna ineficiente.

Sendo assim, o algoritmo OS-ELM foi proposto para lidar com a aprendizagem sequencial e promover um aprendizado *online* (LIANG et al., 2006). Os parâmetros de entrada do OS-ELM são os mesmos do ELM padrão, acrescidos de: número do conjunto de treinamento inicial N_0 e tamanho do bloco de dados da etapa *online* N_k. O processo de aprendizagem no OS-ELM é composto de duas etapas. Primeiro ocorre a etapa de inicialização, para que depois venha a etapa de aprendizagem sequencial.

Na etapa inicial, um conjunto de treinamento $\mathbf{D}_0 = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0} \subset \mathbf{D}$ (com $N_0 < T$) é utilizado para construir o modelo ELM inicial, em modo lote (*batch*). Na fase de aprendizagem sequencial apenas as novas instâncias que chegam pelo fluxo são usadas para atualizar o modelo. Uma vez concluído o passo, essas instâncias são descartadas.

A etapa de inicialização do OS-ELM é semelhante à do ELM padrão. O peso de saída β_0 é determinado conforme as equações 2.11, 2.12 e 2.13 a seguir (LIANG et al., 2006):

$$\beta_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1} \mathbf{H}_0^T \mathbf{Y}_0 \tag{2.11}$$

em que

$$\mathbf{H}_0 = \begin{bmatrix} g(\mathbf{a}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(\mathbf{a}_1 \cdot \mathbf{x}_{N_0} + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_{N_0} + b_L) \end{bmatrix}_{N_0 \times L}, \tag{2.12}$$

$$\beta_0 = \left[\beta_1^T, \cdots, \beta_L^T\right]_L \quad e \quad \mathbf{Y}_0 = \left[\mathbf{y}_1^T, \cdots, \mathbf{y}_T^T\right]_{N_0}, \tag{2.13}$$

sendo $\mathbf{Y}_0 = [y_1, y_2, ..., y_T]^{N_0}$ a saída de $\mathbf{D}_0$, e $\mathbf{H}_0$ a matriz oculta inicial. A fim de tornar $\mathbf{H}_0^T \mathbf{H}_0$ não-singular ($rank(\mathbf{H}_0) = L$), $N_0 \geq L$ é necessário (LIANG et al., 2006).

A Equação 2.11 pode ser reescrita como $\beta_0 = \mathbf{M}_0 \mathbf{H}_0^T \mathbf{Y}_0$, onde $\mathbf{M}_0$ é a matriz de covariância inicial, calculada pela Equação 2.14:

$$\mathbf{M}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1} \tag{2.14}$$

Na etapa de aprendizagem sequencial, o $(k+1)$-ésimo bloco é definido conforme a Equação 2.15 a seguir:

$$\mathbf{D}_{k+1} = (\mathbf{x}_t, \mathbf{y}_t)_{t=(\sum_{l=0}^{k} T_l)+1}^{t=\sum_{l=0}^{k+1} T_l} \tag{2.15}$$

onde $k \geq 0$, $\mathbf{D}_{k+1}$ representa o $(k+1)$-ésimo bloco de instâncias e T_{k+1} o número de instâncias no $(k+1)$-ésimo bloco.

Quando um novo bloco chega, o novo vetor de peso de saída β_{k+1} é calculado usando conceitos do algoritmo *Recursive Least Squared* (RLS) (LIANG et al., 2006), conforme as equações 2.16, 2.17, 2.18 e 2.19 a seguir:

$$\mathbf{H}_{k+1} = \begin{bmatrix} g(\mathbf{a}_1 \cdot \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1} + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1} + b_L) \\ \vdots & \ddots & \vdots \\ g(\mathbf{a}_1 \cdot \mathbf{x}_{\sum_{l=0}^{k+1} T_l} + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_{\sum_{l=0}^{k+1} T_l} + b_L) \end{bmatrix}_{T_{k+1} \times L}, \tag{2.16}$$

$$\mathbf{Y}_{k+1} = \left[\mathbf{y}^T_{(\sum_{l=0}^{k} T_l)+1}, \cdots, \mathbf{y}^T_{\sum_{l=0}^{k+1} T_l}\right]_{T_{k+1}}, \tag{2.17}$$

$$\mathbf{M}_{k+1} = \mathbf{M}_k - \mathbf{H}_{k+1}^T (1 + \mathbf{H}_{k+1} \mathbf{M}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{M}_k, \tag{2.18}$$

$$\beta_{k+1} = \beta_k + \mathbf{M}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{Y}_{k+1} - \mathbf{H}_{k+1} \beta_k). \tag{2.19}$$

Liang et al. (2006) fornecem uma explicação detalhada das equações 2.18 e 2.19. O modelo de aprendizado OS-ELM é descrito no Algoritmo 2.

Como se observa nesta seção, a atualização do OS-ELM utiliza várias operações matriciais que podem se tornar muito custosas em termos computacionais, dependendo do tamanho do bloco e da quantidade de neurônios utilizada. Vale lembrar também que a própria dinâmica dos fluxos de dados impõe uma restrição de tempo para a execução do

Algoritmo 2: Modelo de aprendizado OS-ELM

Entrada: Conjunto de dados $\mathbf{D} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; nº de instâncias para inicialização N_0; função de ativação $g(x)$; nº de neurônios ocultos (L);
Saída: Erro de previsão; Tempo de execução;

início
  Considere o conjunto inicial de treinamento $\mathbf{D}_0(\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0}$;
  Aloque aleatoriamente os pesos de entrada $\mathbf{a}_j$ e os limiares b_j, $j = 1, ..., L$;
  Calcule a matriz da camada oculta $\mathbf{H}_0$ com $\mathbf{D}_0$ conforme a Equação 2.12;
  Calcule o vetor β_0 com os pesos de saída através da Equação 2.11;
  para $k = N_0$; $k \leq T$; $k = k + 1$ **faça**
    Apresente o $(k+1)$-ésimo bloco $\mathbf{D}_{k+1}$ conforme a Equação 2.15;
    Calcule a matriz $\mathbf{H}_{k+1}$ utilizando $\mathbf{D}_{k+1}$ conforme a Equação 2.16;
    Defina $\mathbf{Y}_{k+1}$ conforme a Equação 2.17;
    Calcule $\mathbf{M}_{k+1}$ e β_{k+1} utilizando respectivamente as equações 2.18 e 2.19;
  fim
  Calcule o erro de previsão e o tempo de execução;
fim

algoritmo. Uma solução para mitigar esse problema pode ser o uso de técnicas de computação de alto desempenho, por meio da programação paralela, para dividir as tarefas de atualização do OS-ELM em vários núcleos de processamento.

2.7 *Ensembles* de Modelos de Previsão

Um *ensemble* de modelos de previsão é composto por uma combinação de modelos individuais. Um *ensemble* combina uma série de j modelos de aprendizado $M_1, M_2, \ldots, M_j$ com o objetivo de criar um modelo de previsão composto melhorado (HAN; KAMBER, 2006). A Figura 2.5 apresenta o esquema geral dos *ensembles*.

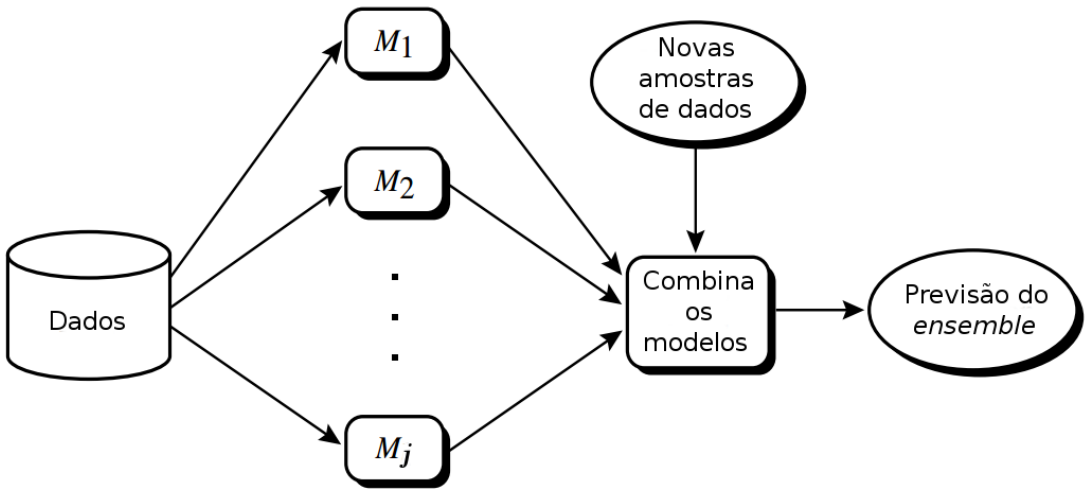

Figura 2.5: Estrutura generalizada de um *Ensemble*, adaptado de Han e Kamber (2006)

Normalmente, são usadas as estratégias de média aritmética ou média ponderada para a combinação dos modelos de um *ensemble* em tarefas de regressão. Com a média ponderada, normalmente os modelos mais precisos obtêm pesos maiores, tornando sua contribuição mais relevante na previsão final.

Os *ensembles* podem ser estáticos, nos quais seus modelos permanecem os mesmos durante toda a operação; ou dinâmicos, nos quais os modelos do *ensemble* são incluídos, substituídos ou excluídos por novos modelos de acordo com um critério pré-estabelecido.

A diversificação dos membros do *ensemble* é crucial para obter ganho de precisão. Os componentes podem diferir entre si pelos dados de treinamento, pelos seus atributos ou pelo modelo de aprendizado (BRZEZINSKI, 2010). *Ensembles* são considerados mecanismos úteis para lidar com desvio de conceito em fluxo de dados devido à natureza dinâmica dos conjuntos, nos quais os modelos individuais podem ser reciclados, novos modelos podem ser adicionados e modelos com baixa precisão podem ser descartados do conjunto (CAVALCANTE; OLIVEIRA, 2015).

No entanto, o custo computacional e o tempo de execução de um *ensemble* normalmente é proporcional ao seu tamanho. Também é possível possível mitigar esse efeito aplicando técnicas de alto desempenho para executar os modelos de forma paralela. A próxima seção discute algumas das técnicas mais comuns de computação de alto desempenho por meio da programação paralela.

2.8 Técnicas de Computação de Alto Desempenho – Programação Paralela

A programação paralela consiste na codificação em uma linguagem que permita indicar como diferentes porções de código podem ser executadas concorrentemente em vários núcleos de processamento (QUINN, 2003). Isso envolve a divisão de um problema em uma série de tarefas independentes, menores, de forma que seja possível utilizar múltiplos núcleos de processamento para executá-las "simultaneamente", com maior agilidade ou, muitas vezes, apenas em tempo computacional suficiente para o seu propósito. No entanto, nem todos os problemas são passíveis de paralelização.

Devido às limitações físicas, fatores econômicos e questões de escalabilidade, a tendência de crescimento da frequência do *clock* dos processadores tem diminuído nos últimos anos e os fabricantes têm utilizado a estratégia de lançar *chips* com mais núcleos de processamento. Como consequência, para alcançar o aproveitamento máximo dos processadores atuais e futuros, o uso de técnicas de programação paralela é imprescindível. Soma-se a isso o fato de a programação paralela poder ser utilizada em *clusters* computacionais com vários nós de processamento independentes.

Diante desse cenário, é possível distinguir duas arquiteturas para divisão de tarefas:

- Divisão de tarefas por nó computacional: típica para *clusters* computacionais com vários nós de processamento;
- Divisão de tarefas em um mesmo nó computacional: comum a processadores com vários núcleos (*multicore*) ou Unidades de Processamento Gráfico.

Para lidar com essas arquiteturas existem dois modelos básicos de programação paralela em processadores convencionais: i) utilizando memória compartilhada; ii) utilizando memória distribuída. Esses modelos serão discutidos na próxima seção. Também será abordado o modelo de programação em GPUs e modelos de programação híbridos.

2.8.1 Programação Paralela com Memória Compartilhada

Neste modelo de programação, o *hardware* é um conjunto de processadores, ou de núcleos de processamento, com acesso à mesma memória compartilhada. Como todos têm acesso aos mesmos endereços de memória, os processadores ou núcleos podem interagir e sincronizar suas tarefas entre si através de variáveis compartilhadas (QUINN, 2003). A Figura 2.6 ilustra o funcionamento desse modelo.

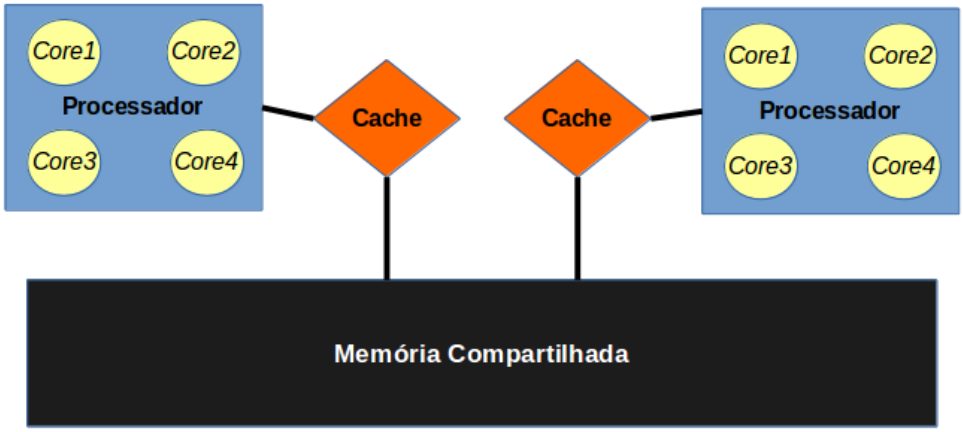

Figura 2.6: Programação Paralela com Memória Compartilhada.

Por mais que possa parecer, a programação de memória compartilhada não é uma invenção da era *multicore*. Os sistemas com vários processadores *single-core* tem origens na década de 60, porém as interfaces de programação portáteis apropriadas, especialmente a POSIX Threads (PThreads), foram desenvolvidos na década de 1990 (HAGER; WELLEIN, 2011).

A criação de *threads* através da PThreads não é um processo complexo. Porém, ela exige que o programador sincronize explicitamente os *threads*, através de Semáforos, Mutexes e outros mecanismos de sincronização (ANDRIJAUSKAS, 2013) para que não ocorra inconsistências nos dados na memória.

Hoje em dia, o padrão de programação de memória compartilhada dominante é a biblioteca OpenMP. Trata-se de um conjunto de diretivas que adota a metodologia de paralelismo *fork/join* (HAGER; WELLEIN, 2011; QUINN, 2003).

Apesar da fácil implementação fornecida pelo OpenMP, pesa contra o modelo com memória compartilhada o fato dela ser limitada à capacidade de adição de núcleos de processamento (ou processadores) num único sistema computacional, bem como o fato de o acesso à memória compartilhada poder ser um gargalo. Para situações em que isso ocorra, pode ser mais adequada a adoção de um modelo com memória distribuída.

2.8.2 Programação Paralela com Memória Distribuída

No modelo que adota memória distribuída, a comunicação entre os processadores é realizada através da troca de mensagens. Isso ocorre porque cada processador possui a sua própria memória. Sendo assim, não há maneira de um processador acessar diretamente o espaço de endereços de outro (HAGER; WELLEIN, 2011). A passagem de mensagens ocorre através de um canal de comunicação entre os processadores, que pode ser um barramento ou uma conexão de rede, conforme é ilustrado na Figura 2.7.

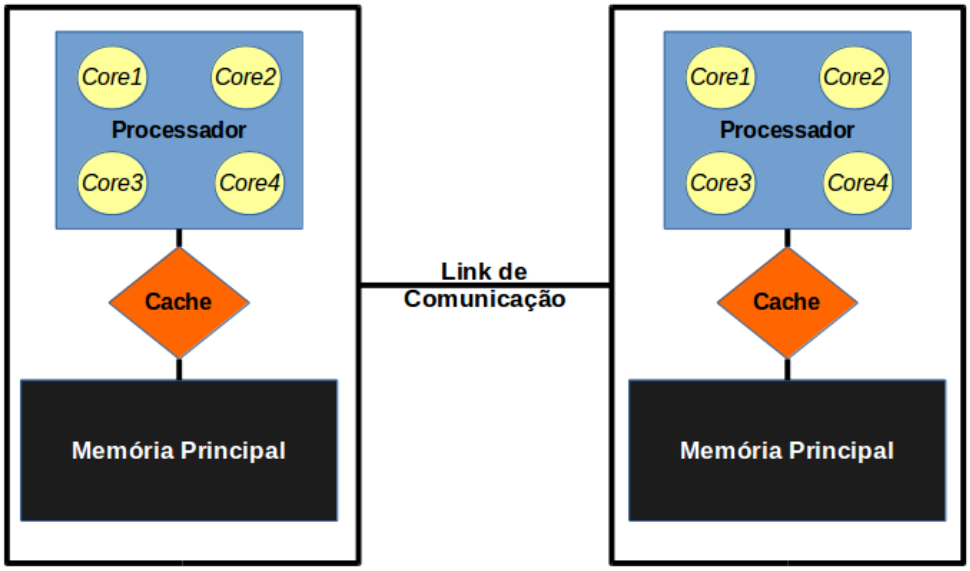

Figura 2.7: Programação Paralela com Memória Distribuída.

O padrão *Message Passing Interface* (MPI) é a especificação de passagem de mensagens mais popular que suporta programação paralela. Praticamente todos os computadores comerciais paralelos e várias linguagens de programação suportam o MPI (QUINN, 2003) que segue as seguintes regras (HAGER; WELLEIN, 2011):

- O mesmo programa é executado em todos os processos (*Single Program Multiple Data* – SPMD).
- O programa é escrito em uma linguagem sequencial como Fortran, C ou C++. A troca de dados, é feita através de chamadas de funções *send* e *receive*.
- Todas as variáveis em um processo são locais para aquele processo. Não há uma memória compartilhada entre os processadores ou núcleos de processamento.

Cabe ressaltar que, num programa que utiliza o MPI, as mensagens transportam dados entre processos e esses processos não precisam estar necessariamente executando em processadores separados. Eles podem ser executados em vários núcleos de processamento dentro de um mesmo nó computacional, em vários núcleos de processamento localizado em diferentes nós computacionais, ou mesmo dentro de um único núcleo de processamento compartilhando tempo e recursos computacionais.

Apesar dessa característica, quando se trata de um mesmo nó computacional, a manipulação de processos e a troca de mensagens entre eles por meio do MPI costuma consumir

mais tempo de processamento e memória do que a implementação por meio de *threads*. Diante dessa condição, num cenário como o ilustrado na Figura 2.7, pode ser interessante a utilização de um modelo híbrido. Antes de se explorar essa abordagem, a próxima seção trata da utilização de Unidades de Processamento Gráfico (em inglês *Graphics Processing Unit* – GPU) em programação paralela de propósito geral.

2.8.3 Programação Paralela em GPUs

As GPUs são *chips* com vários núcleos de processamento desenvolvidas originalmente para processamento gráfico. Elas foram concebidas inicialmente para aplicações de tratamento de imagens, vídeos e jogos digitais e não eram tão simples de se programar. Entretanto, o surgimento de plataformas de programação com suporte a linguagens de programação tornou viável o uso das GPUs para Computação de Propósito Geral (em inglês *General Purpose Computing on Graphics Processing Units* – GPGPU).

Uma dessas interfaces de programação de aplicações (API) é o *Compute Unified Device Architecture* (CUDA), que é fornecida pela empresa NVIDIA para executar aplicativos de uso geral sobre suas GPUs (KIRK; HWU, 2010). Devido ao surgimento dessa API, é possível lidar não só com aplicações gráficas, mas também com aplicações de uso geral nas GPUs (JO; KIM; BAE, 2014).

Ao programar com CUDA nem todo o código é executado na GPU. Normalmente a maior parte do código é executada de forma sequencial na CPU e apenas uma parte, menor, é executada na GPU. Geralmente, as funções de processamento mais intensivo ficam residentes na GPU. No modelo de programação com CUDA o código segue a lógica de execução CPU → GPU → CPU, conforme a Figura 2.8.

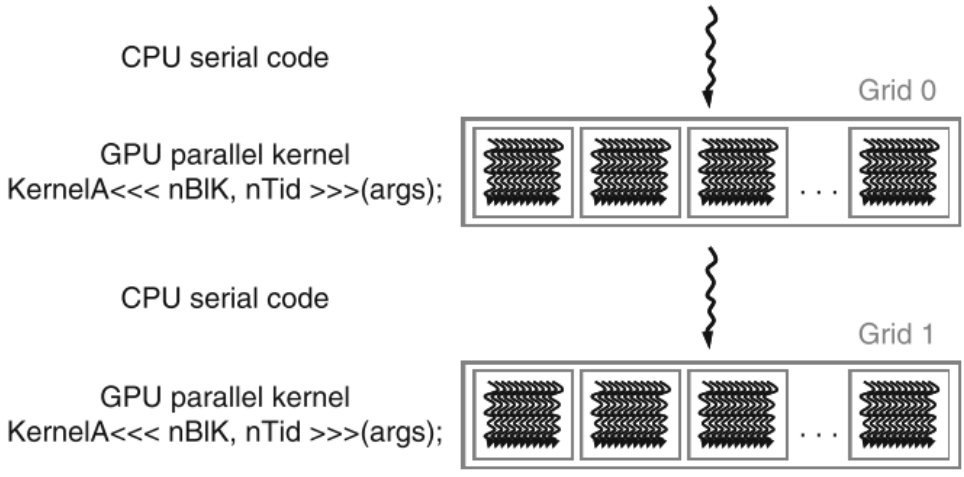

Figura 2.8: Sequência típica de código em CUDA. Fonte: Kirk e Hwu (2010).

A programação dos núcleos a serem executados na GPU são realizadas por meio do lançamento dos *kernels*, onde são colocadas as funções correspondentes. Segundo Karunadasa e Ranasinghe (2009), os *kernels* são implementados usando um modelo de memória compartilhada baseado em *threads*, sendo categorizados em "blocos de threads" e "*grids*".

Antes do lançamento dos *kernels*, é necessário chamar as funções do CUDA para alocação, e transferência de memória da CPU para a GPU, ou seja, internamente a memória

da GPU pode ser compartilhada, mas a GPU não acessa dados da memória principal da CPU, da mesma forma, a CPU não consegue acessar a memória da GPU. Tal abordagem aproxima-se de um Modelo Híbrido de programação, já que é necessária a passagem de dados entre memórias distintas, conforme a Figura 2.9.

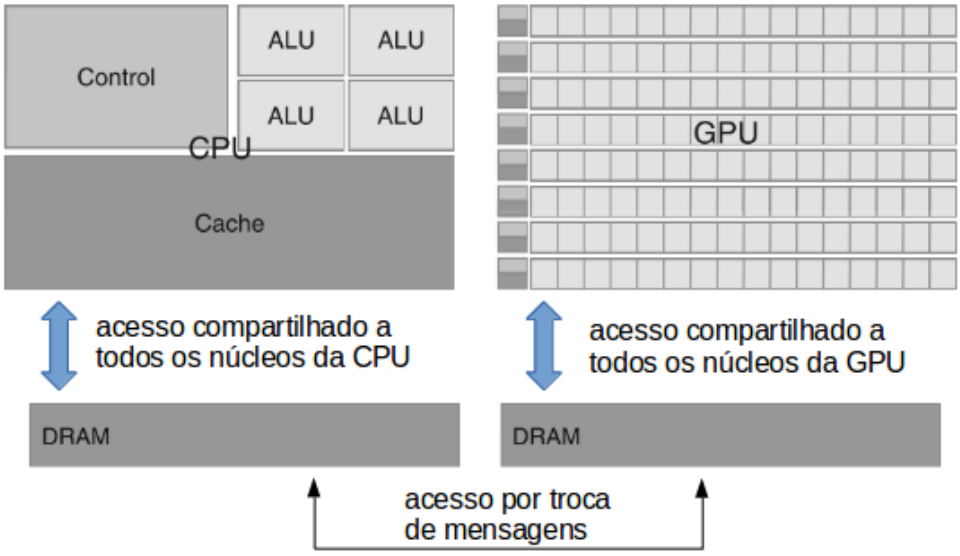

Figura 2.9: Comunicação entre CPU e GPU, adaptado de Kirk e Hwu (2010).

2.8.4 Modelos Híbridos de Programação Paralela

Esses modelos caracterizam-se pela combinação de dois ou mais modelos mencionados anteriormente, onde parte do programa pode executar em memória compartilhada e outra parte pode distribuir os dados entre memórias distintas. A Figura 2.10 ilustra um caso típico.

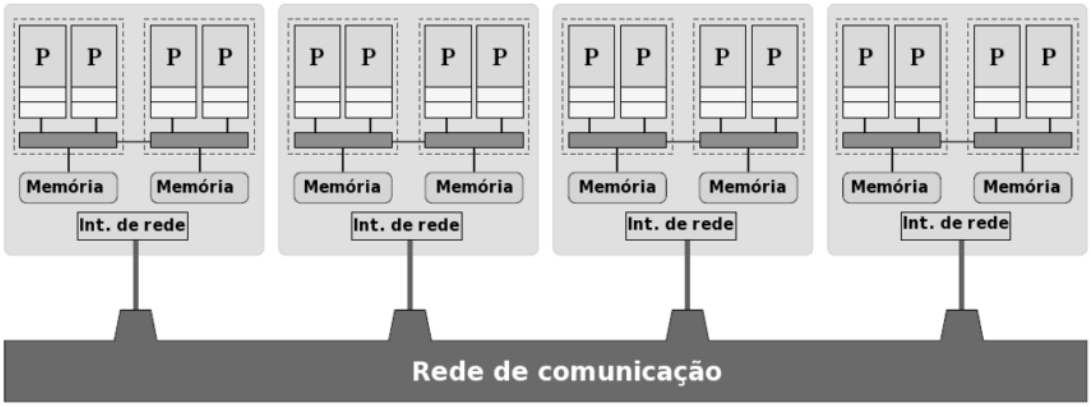

Figura 2.10: Modelo Híbrido de Programação Paralela. Fonte: Andrijauskas (2013).

O cenário da Figura 2.10 mostra um típico *cluster* com vários nós computacionais. Em cada nó computacional pode haver mais de um processador e estes processadores podem ter mais de um núcleo de processamento. Pode-se assumir também a existência de GPUs em conjunto com esses processadores.

Num cenário como esse é possível combinar os diferentes modelos de programação paralela. O MPI fornece a comunicação entre processos que estarão em nós diferentes, o OpenMP provê as características para que diversos núcleos de processamento em CPUs convencionais recebam partes dessa tarefa (ANDRIJAUSKAS, 2013) e o CUDA pode transferir parte do processamento para serem executados de forma paralela nas GPUs conforme pode ser observado em Karunadasa e Ranasinghe (2009).

2.9 Considerações do capítulo

Este capítulo definiu os conceitos teóricos mais importantes referidos a este trabalho. De forma geral, os desvios de conceito estão relacionados com o comportamento dinâmico dos fluxo de dados e os mesmos necessitam de processamento *online* e em tempo real.

Uma vez que o OS-ELM é adequado a este cenário e passível de paralelização, pode-se considerar as técnicas de programação paralela, combinando o OS-ELM com o uso de *ensembles*, para o desenvolvimento de soluções otimizadas para previsão de fluxo de dados. Sendo assim, o próximo capítulo apresentará brevemente os principais trabalhos que envolvem a aceleração dos algoritmos ELM e OS-ELM usando técnicas de programação paralela e sua aplicação com *ensembles*.

Capítulo 3

Levantamento Bibliográfico

Este capítulo apresenta um breve levantamento bibliográfico dos principais trabalhos que envolvem os conceitos abordados no Capítulo 2. Serão abordadas trabalhos que agregam os algoritmos OS-ELM em *ensembles*, com a finalidade de aumentar a precisão, e também trabalhos que utilizam técnicas de programação paralelas nos ELMs, com a finalidade de reduzir o tempo de execução mantendo a acurácia.

3.1 *Ensembles* para modelos OS-ELM

O *Ensemble of Online Sequential Extreme Learning Machine* (EOS-ELM), proposto por Lan, Soh e Huang (2009), consiste de várias redes neurais OS-ELM que compartilham o mesmo número de neurônios ocultos e a mesma função de ativação. Os parâmetros de entrada (pesos da camada de entrada e limiares) da rede neural para cada OS-ELM são gerados de forma aleatória e os pesos de saída são obtidos analiticamente com base nas instâncias de entrada, em comum para todos OS-ELMs, que chegam de forma sequencial.

A saída final do EOS-ELM é dada pelo cálculo da média das saídas de cada OS-ELM (LAN; SOH; HUANG, 2009). Apesar de soar estranho um *ensemble* com várias redes neurais que utilizam o mesmo modelo de aprendizado, ele se justifica porque os parâmetros de entrada, gerados aleatoriamente, tornam cada OS-ELM um modelo distinto. Sendo assim, os modelos OS-ELM que compõem o *ensemble* podem ter diferentes capacidades de adaptação às novas instâncias. Os experimento realizados no trabalho de Lan, Soh e Huang (2009) mostraram que o EOS-ELM forneceu resultados mais estáveis, diminuindo em até 5 vezes o desvio padrão do erro, e precisos, aumentando em até 2% a acurácia da previsão, se comparado a um único modelo OS-ELM.

O *Additive Expert* (AddExp), proposto por Kolter e Maloof (2005), é o *ensemble online* mais popular para tarefas de regressão (SOARES; ARAÚJO, 2015a). O AddExp agrega vários modelos preditivos (chamados de especialistas), onde cada modelo possui um peso associado. Os especialistas utilizam o mesmo modelo de aprendizado, mas cada um deles é criado em instantes de tempo diferentes (KOLTER; MALOOF, 2005). O AddExp pode ser utilizado com vários modelos de aprendizado *online*, inclusive com o OS-ELM, como feito no trabalho de Soares e Araújo (2015a).

Voltado para tarefas de classificação, o AddExp.D utiliza um voto ponderado que considera as saídas de todos os especialistas. Quando chega uma nova instância, a saída do algoritmo é determinada pela previsão do especialista com o maior peso. Os pesos dos especialistas com baixa precisão são diminuídos por um β constante multiplicativo. Se a previsão geral estiver incorreta, um novo especialista é adicionado ao *ensemble* e todos os especialistas são retreinados com a instância do instante de tempo corrente.

Voltado para tarefas de regressão, o AddExp.C funciona de forma similar ao AddExp.D. Porém, ao invés do voto ponderado, ele prevê a soma de todas as previsões dos especialistas, ponderadas pelo peso relativo de cada especialista. Outra diferença em relação ao AddExp.D é que, ao invés de adicionar um novo membro ao *ensemble* quando a previsão global estiver incorreta, no AddExp.C um novo membro é adicionado sempre que a perda global do algoritmo naquele instante de tempo for maior do que algum valor limite (KOLTER; MALOOF, 2005). O AddExp pode ser usado com qualquer algoritmo de aprendizado *online*, como o OS-ELM.

O *Ensemble of Online Learners with Substitution of Models* (EOS) (BUENO; COELHO; BERTINI, 2017) é um *ensemble* de modelos de aprendizado *online* que implementa janelas deslizantes e um esquema de atualização originalmente proposto por Street e Kim (2001), baseado em uma frequência constante. Nos experimentos, os autores aplicaram o EOS e outros *ensembles*, utilizando o ELM e o OS-ELM como modelos, em previsão de concentrações de Material Particulado Inalável (MP_{10}) no ar. Nesse cenário, também foram consideradas abordagem únicas, sem o uso de *ensembles*, do ELM e do OS-ELM.

Os resultados mostraram que o OS-ELM foi mais preciso que o ELM em tais cenários, e também que os *ensembles* com modelos OS-ELMs melhoraram a estabilidade dos resultados, sendo o EOS aquele que apresentou os melhores resultados. No entanto, de maneira geral, a maioria das abordagens OS-ELM e seus respectivos *ensembles* levaram mais tempo que as abordagens ELM para realizar as previsões.

Introduzido por Soares e Araújo (2015a), o *Dynamic and On-line Ensemble Regression* (DOER) oferece capacidade de adaptação rápida para a previsão *online* em ambientes não estacionários. O DOER é voltado para tarefas de regressão, com as seguintes propriedades:

- Inclusão e remoção dinâmica dos modelos de aprendizado, para manter apenas os modelos com maior precisão;
- Adaptação dinâmica dos pesos de cada modelo de aprendizado, com base em previsões e *online*;
- Adaptação *online* dos parâmetros dos modelos.

Os experimentos foram realizados em cenários que exigiam uma rápida capacidade de adaptação. O DOER foi comparado com quatro modelos de aprendizado derivados do OS-ELM e cinco *ensembles online*: EOS-ELM, AddExp, *Online Bagging* (OB) (OZA; RUSSELL, 2001), Learn^{++}.NSE (ELWELL; POLIKAR, 2011) e *Online Accuracy Updated Ensemble* (OAUE) (BRZEZINSKI; STEFANOWSKI, 2014). Os resultados mostraram que o DOER apresentou melhor precisão na maioria dos casos analisados. Apesar disso, o DOER variou entre o segundo e terceiro pior tempo de processamento em todos os casos analisados. Todos os *ensembles* utilizaram como modelo de base o algoritmo OS-ELM.

3.2 *Extreme Learning Machines* acelerados

Akusok et al. (2015) apresentaram uma abordagem completa para a utilização de uma ferramenta denominada *High-Performance Extreme Learning Machines* (HP-ELM), voltado para *Big Data*, que implementa o conhecimento de ponta em ELMs combinado com programação de alto desempenho. Segundo os autores, um ELM é um método simples que pode ser escrito em três linhas no programa MATLAB. Porém, o desempenho de tais ELMs é sub-otimizado (AKUSOK et al., 2015). Diferente da maioria dos trabalhos que utilizavam o MATLAB em seus experimentos levantados nesta dissertação, o HP-ELM foi desenvolvido em Python e requer as bibliotecas Numpy, Scipy, Numexpr and pyTables.

Existem duas versões do HP-ELM: um delas executa de forma serial em CPU; e a outra provê aceleração por meio da paralelização do código em GPUs NVIDIA compatíveis com o CUDA, GPUs AMD compatíveis com o OpenCL, ou cartão acelerador Xeon Phi (chamado arquitetura MIC). Para a versão acelerada também é necessária a biblioteca *Matrix Algebra on GPU and Multicore Architectures* (MAGMA) (TOMOV; DONGARRA; BABOULIN, 2010) para a arquitetura correspondente. As partes aceleradas são o cálculo das matrizes correlacionadas e o cálculo do β. Os resultados mostraram que a aceleração por GPU começa a ser vantajosa a partir de problemas que utilizam mais de mil neurônios ocultos (AKUSOK et al., 2015).

Krawczyk (2016) propôs uma abordagem que permite a aceleração do algoritmo OS-ELM em GPUs, para lidar com tarefas de classificação desbalanceada em fluxo de dados com desvios de conceito. A proposta combina a utilização do OS-ELM em conjunto com o ADWIN (BIFET; GAVALDA, 2007) e a aceleração dos cálculos matriciais da etapa *online* do OS-ELM, bem como os cálculos das matrizes iniciais da etapa *offline*, no caso de o ADWIN detectar um Desvio de Conceito e treinar um novo classificador.

Os experimentos realizados possuíam um número otimizado de neurônios ocultos, entre 100 e 1.000, usando a função de ativação sigmóide, com blocos de atualização de 2.500 instâncias. Todos os experimentos foram conduzidos em ambiente R (R Core Team, 2013), com o uso do RMOA, usando a biblioteca cuBLAS (NVIDIA, 2017) do NVIDIA CUDA. Os resultados mostraram que a simples delegação das operações matriciais para a GPU reduziu aproximadamente em 10 vezes o tempo de execução (KRAWCZYK, 2016).

No trabalho de Wang et al. (2015), foi proposta uma versão paralela do OS-ELM chamada de *Parallel Online Sequential Extreme Learning Machine* (POS-ELM) baseado na técnica MapReduce, desenvolvido em Java 1.6. O MapReduce (DEAN; GHEMAWAT, 2008), fornece uma estrutura simples, escalonável e tolerante a falhas, podendo ser utilizado para aprendizado em larga escala (HUANG et al., 2016). O POS-ELM foi avaliado em conjuntos de dados reais e sintéticos com o número máximo de 1.280.000 instâncias de treinamento, com o máximo de 128 atributos, com 25 neurônios ocultos e com função de ativação sigmoide, na plataforma Hadoop.

Os resultados experimentais mostraram que a precisão do conjunto de treinamento e do conjunto teste no POS-ELM estão no mesmo nível do OS-ELM e do ELM, e também que o POS-ELM tem boa escalabilidade em relação ao número de dados de treinamento e o número de atributos. Em comparação com o ELM original e o OS-ELM, em que a capacidade de processar dados em grande escala é limitada em uma única unidade de

processamento, o POS-ELM pôde lidar com dados de escala muito maior. Quanto maior o número de dados de treinamento, maior a aceleração do POS-ELM (WANG et al., 2015).

Seguindo a mesma linha de raciocínio, Huang et al. (2016) propôs o *Parallel Ensemble of Online Sequential Extreme Learning Machine* (PEOS-ELM) baseado em MapReduce para aprendizado em larga escala. O algoritmo PEOS-ELM também foi avaliado em conjuntos de dados reais e sintéticos com o número máximo de 5.120.000 instâncias de treinamento, com o máximo de 512 atributos, com 25 a 120 neurônios ocultos e com função de ativação sigmoide. O fator de aceleração desse algoritmo chegou a 40 vezes em um *cluster* com o máximo de 80 núcleos de processamento. A precisão do PEOS-ELM permaneceu no mesmo nível que o EOS-ELM original, executado em uma única máquina.

No trabalho de Van Heeswijk et al. (2011) é proposta uma abordagem que combina a paralelização do ELM em GPUs com a utilização de um *ensemble*. O autor utiliza as funções `culaGels` e `culaGesv` da biblioteca CULA Tools, a partir do MATLAB, para acelerar parte dos modelos ELMs em GPUs. Também é utilizado o *MATLAB's Parallel Computing Toolbox* que permite criar um conjunto de MATLAB *workers* para dividir os modelos do *ensemble* em vários núcleos da CPU. Cada um dos *workers* executa seu próprio *thread* e obtém sua própria GPU dedicada, que é usada para acelerar o ELM.

No caso de um *ensemble* com 100 ELMs e quatro *workers*, cada *worker* teria 25 ELMs (VAN HEESWIJK et al., 2011). Apesar de não ser o caso do trabalho em questão, os autores afirmam que o mesmo esquema poderia ser executado em vários computadores usando o *MATLAB's Distributed Computing Toolbox*. Os experimentos mostraram que o esquema proposto proporcionou uma aceleração de até 3,3 vezes sobre a implementação típica de um *ensemble* de ELMs não paralelizado (VAN HEESWIJK et al., 2011).

3.3 Considerações do capítulo

Este capítulo levantou alguns dos principais trabalhos que abordam o conceitos relacionados no Capítulo 2. De forma geral os *ensembles* melhoram a precisão e a estabilidade das previsões, mas aumentam ainda mais o custo computacional. Também foram levantadas soluções que utilizam processamento distribuído, por meio da programação paralela dos próprios modelos ELMs, ou mesmo dos *ensembles*.

As soluções paralelas se mostraram vantajosas para os parâmetros adotados, acelerando em até 40 vezes o tempo de execução. Com base nos resultados dos trabalhos levantados neste texto, o Capítulo 4 trata das abordagens implementadas neste trabalho.

Capítulo 4

Abordagens Implementadas

Este capítulo descreve a estrutura das abordagens implementadas neste trabalho em mais detalhes. As abordagens estudadas aqui foram baseadas em linguagem C, com e sem técnicas de alto desempenho, com exceção de uma versão do OS-ELM que foi mantida em MATLAB para ser usada como *benchmark*. Foram implementadas diferentes versões do OS-ELM com técnicas de paralelização em diferentes arquiteturas computacionais.

Também foram implementados três *ensembles* com características distintas, utilizando duas abordagens: uma versão convencional, que executa um modelo após o outro; e uma versão alternativa capaz de lançar os modelos simultaneamente em diferentes processos, usando o padrão MPI. Com o uso desse padrão é possível distribuir a execução dos modelos do *ensemble* em diferentes núcleos de processamento ou nós computacionais.

4.1 *Online Sequential Extreme Learning Machines* de Alto Desempenho com Janelas Deslizantes

Para adaptar o OS-ELM (HUANG, 2013) ao processamento de fluxo de dados, foi considerado um conjunto de dados em que um pedaço de tamanho N_0 é utilizado para o treinamento inicial e as instâncias restantes simulam um fluxo *online*, cujos valores seguintes serão previstos pelo OS-ELM. Inclusive, foi adotado um mecanismo que permite a operação do OS-ELM com Janelas Deslizantes (em inglês *Sliding Window* – SW).

Neste trabalho, as versões do OS-ELM em linguagem C ficaram restritas aos casos de regressão com a função de ativação sigmóide ($g(x) = \frac{1}{(1+e^{(-x)})}$). Usando bibliotecas *multithread*, é possível definir um número de *threads* N_{th} para paralelizar as funções do OS-ELM. O Algoritmo 3 descreve o OS-ELM com SW implementado nesse trabalho.

Além da versão em MATLAB, referenciada como OS-ELM, foram desenvolvidas três versões distintas em linguagem C: OS-ELM com OpenBLAS (OS-ELMob), OS-ELM com MKL (OS-ELMmkl) e OS-ELM com MAGMA (OS-ELMmgm). OpenBLAS, Intel MKL e MAGMA são bibliotecas que permitem executar as operações matriciais com as funções dos pacotes *Basic Linear Algebra Subprograms* (BLAS) e *Linear Algebra PACKage* (LAPACK). As funções do OpenBLAS (ZHANG; WANG; ZHANG, 2012) e do Intel MKL executam em ambientes *multithread*, o que permite o ganho de tempo com paralelismo em processadores *multicore* e também em arquiteturas de multiprocessadores. Por sua vez,

Algoritmo 3: OS-ELM com Janelas Deslizantes

Entrada: Nº de *threads* N_{th}; Fluxo de dados $\mathbf{D} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; nº de neurônios ocultos (L – em que $L \leq T_0 < T$); função de ativação $g(x)$; nº de instâncias para o treinamento inicial N_0; tamanho do bloco para a fase de aprendizado *online* e sequencial N_k; tamanho da janela m;

Saída: Erro de previsão do fluxo; Tempo de execução;

1 **início**
2 Considere $\mathbf{D}_0(\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0}$ para treinamento inicial;
3 Gere os pesos de entrada $\mathbf{a}_j$ e os limiares b_j aleatórios, $j = 1, \ldots, L$;
4 Calcule $\mathbf{H}_0$, de tamanho $N_0 \times L$, com $\mathbf{D}_0$, conforme a Eq. 2.12;
5 Calcule $\mathbf{M}_0$, de acordo com a Eq. 2.14;
6 Calcule o vetor β_0 através da Eq. 2.11;
7 **para** $k = N_0$; $k \leq T$; $k = k + N_k$ **faça**
8 Deslize a janela considerando $\mathbf{D}_{k+1} = (\mathbf{x}_t)_{t=k-(m-N_k)}^{k+N_k}$;
9 Calcule $\mathbf{H}_{k+1}$ usando $\mathbf{D}_{k+1}$ conforme a Eq. 2.16;
10 Calcule a previsão de $\mathbf{D}_{k+1}$ com o β_k previamente calculado;
11 Obtenha as saídas reais $\mathbf{Y}_{k+1} = (\mathbf{y}_t)_{t=k-(m-N_k)}^{k+N_k}$;
12 Atualize a $\mathbf{M}_{k+1}$ de acordo com a Eq. 2.18;
13 Atualize o β_{k+1} de acordo com a Eq. 2.19;
14 **fim**
15 Calcule o tempo de execução e o erro de previsão no fluxo;
16 **fim**

as funções do MAGMA exploram o paralelismo em arquiteturas heterogêneas/híbridas para sistemas *Multicore* + GPU.

Todas as versões em C utilizaram as respectivas funções `dgetrf()` e `dgetri()` para calcular a matriz de covariância inicial $\mathbf{M}_0$ (passo 5 do Algoritmo 3). As respectivas funções `dgesv()` foram usadas para encontrar β_0 (passo 6 do Algoritmo 3), onde também foi adicionando um parâmetro de corte $\frac{1}{\lambda}$, conforme a Equação 2.10, em que $\lambda = 50\epsilon$, e ϵ é a precisão da máquina para o tipo de dado *float*.

Na fase sequencial e *online*, foram utilizadas as respectivas funções `dgesv()` para encontrar a matriz inversa no cálculo de $\mathbf{M}_{k+1}$ (passo 12 do Algoritmo 3) ao invés das funções `dgetrf()` e `dgetri()`, a fim de melhorar o desempenho, como recomendado pelas documentações das bibliotecas. Também foram utilizadas outras funções do BLAS e do LAPACK, como `dgemm()`, `daxpy()`, `dlarnv()`, `dcopy()`, `dlaset()` e `dger()`, entre outras. Além disso, existem laços paralelizados com o padrão OpenMP.

O OS-ELMob e o OS-ELMmkl se diferenciam apenas no cabeçalho de inclusão das bibliotecas e na função que copia e gera a transposta das matrizes, chamada `cblas_domatcopy()` na OpenBLAS e `mkl_domatcopy()` na Intel MKL.

No OS-ELMmgm os passos 4 a 13 do Algoritmo 3 foram adaptados usando as funções correspondentes do MAGMA. Nessa versão, também foi necessário transferir os dados para a memória da GPU depois que eles foram carregados. A geração de pesos e limiares aleatórios (passo 3 do Algoritmo 3) usaram a função `LAPACKE_dlarnv` em todas as versões, já que o MAGMA não possui uma função `dlarnv()` correspondente.

4.2 *Ensembles* para modelos OS-ELM

A complexidade das estratégias utilizadas por um *ensemble* influenciam na sua previsão final, bem como no seu tempo de execução. Diante disso, baseando-se no levantamento bibliográfico, foram considerados neste trabalho três *ensembles* que incorporam as diferentes estratégias a seguir:

- EOS-ELM: estático, que usa a média aritmética como saída;
- EOS: dinâmico de critério fixo, que usa a média aritmética como saída; e
- DOER: dinâmico de critério variável, que usa a média ponderada como saída.

4.2.1 *Ensemble of Online Sequential Extreme Learning Machine*

O *Ensemble of Online Sequential Extreme Learning Machine* (EOS-ELM) (LAN; SOH; HUANG, 2009) é um *ensemble* estático que consiste em vários modelos do OS-ELM com os mesmos parâmetros. O Algoritmo 4 resume o EOS-ELM, onde consideramos o OS-ELM com janelas deslizantes como modelos.

Algoritmo 4: *Ensemble* EOS-ELM

Entrada: Nº de *threads* N_{th}; Fluxo de dados $\mathbf{D} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; modelo de aprendizado online f; nº de instâncias para o treinamento inicial N_0; tamanho do bloco para a fase de aprendizado *online* e sequencial N_k; tamanho da janela m; nº de modelos N_j;

Saída: Erro de previsão do fluxo; Tempo de execução;

início
  Atribua $\varepsilon \leftarrow \emptyset$; e o conjunto de treinamento inicial $\mathbf{D}_0 = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0}$;
  para $j = 1; j \leq N_j; j = j + 1$ **faça**
    $f_{N_j} \leftarrow$ obtenha um novo modelo treinado com $\mathbf{D}_0$;
    Set $\varepsilon \leftarrow \varepsilon \cup f_j$;
  fim
  para $k = N_0; k \leq T; k = k + N_k$ **faça**
    Deslize a janela considerando $\mathbf{D}_{k+1} = (\mathbf{x}_t)_{t=k-(m-N_k)}^{k+N_k}$;
    para $j = 1; j \leq N_j; j = j + 1$ **faça**
      Obtenha a previsão de saída do modelo j com $\mathbf{x}_t$;
    fim
    Obtenha a previsão de ε como: $\mathbf{y}_t = F(\mathbf{x}_t) = (\sum_j^{N_j} f_j(\mathbf{x}_t))/N_j$;
    Obtenha as saídas reais $\mathbf{Y}_{k+1} = (\mathbf{y}_t)_{t=k-(m-N_k)}^{k+N_k}$;
    para $j = 1; j \leq N_j; j = j + 1$ **faça**
      Atualize o modelo f_j com $(\mathbf{x}_t, \mathbf{y}_t)$;
    fim
  fim
  Calcule o tempo de execução e o erro de previsão do fluxo;
fim

O EOS-ELM inicia recebendo o conjunto de treinamento inicial D_0, que contém as amostras mais antigas do fluxo de dados. As etapas compreendidas entre o Passo 3 e o Passo 6 treinam todos os modelos do EOS-ELM com o conjunto de treinamento inicial.

Do Passo 7 ao Passo 17 está compreendida a etapa *online*. No Passo 8, a janela é deslocada para adicionar as novas amostras recebidas e remover as mais antigas, de acordo com o seu tamanho e o bloco adotado. Os passos 9, 10 e 11 calculam a previsão de cada modelo de maneira individual. No Passo 12, a saída conjunta é calculada usando a média aritmética das previsões dos modelos.

Após realizar a previsão, o *ensemble* aguarda a chegada das saídas reais no Passo 13, que utiliza para atualizar todos os modelos de previsão nos passos 14, 15 e 16. Após o término do fluxo de dados, o Passo 18 calcula o tempo de execução e o erro de previsão.

4.2.2 *Dynamic and On-line Ensemble Regression*

O *Dynamic and On-line Ensemble Regression* (DOER) (SOARES; ARAÚJO, 2015a) é um *ensemble* incremental e dinâmico, baseado em amostras e projetado para ambientes com desvios de conceito. O DOER oferece uma adaptação dinâmica aos pesos dos modelos e também um mecanismo dinâmico de inclusão ou substituição de modelos, com critério variável. Foram feitas algumas modificações na abordagem deste trabalho chamada *Simplified C DOER* (SCDOER), como segue:

- Ajuste para operar também com blocos de amostras;
- Cálculo do Erro Quadrático Médio (em inglês *Mean Squared Error* – MSE) para cada modelo de forma simplificada;
- Consideração do tamanho da janela igual a N_0 para o treinamento inicial e para a inclusão ou substituição de modelos. O Algoritmo 5 descreve o SCDOER.

Na fase de inicialização o DOER recebe o conjunto de treinamento inicial que contém as amostras mais antigas do fluxo de dados. O Passo 3 treina o primeiro modelo do DOER com o conjunto de treinamento inicial. O Passo 4 inicializa a vida do primeiro modelo, o que determina a quantidade de instantes de tempo que o modelo está ativo. O Passo 4 também inicializa o MSE e o peso do primeiro modelo, e o inclui ao *ensemble*.

A etapa *online* ocorre entre os passos 5 e 30. No Passo 6, a janela é deslocada de acordo com seu tamanho e o bloco adotado. Os passos 7, 8, 9 e 10 calculam a previsão de cada modelo de maneira individual e, de forma incremental, a vida de cada modelo, de acordo com o tamanho do bloco. O Passo 11 calcula a saída conjunta usando a média ponderada das saídas dos componentes.

Após realizar a previsão, o *ensemble* aguarda a chegada das saídas reais no Passo 12. O erro atual de cada modelo do *ensemble* é calculado no Passo 14, considerando o bloco atual, e acrescentado ao histórico de erros do modelo.

Com base no histórico de erros e na vida do modelo, o Passo 15 calcula o MSE de maneira simplificada, diferente do DOER original. O Passo 16 calcula os pesos de cada modelo de acordo com seu MSE e a mediana dos MSEs de todos os modelos. No Passo 17, os modelos são atualizados.

Algoritmo 5: *Ensemble* SCDOER

Entrada: Nº de *threads* N_{th}; Fluxo de dados $\mathbf{D} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; modelo de aprendizado online f; nº de instâncias para o treinamento inicial N_0; tamanho do bloco para a fase de aprendizado *online* e sequencial N_k; tamanho da janela m; nº máximo de modelos MAX_j; fator para adicionar ou substituir um modelo α;

Saída: Erro de previsão do fluxo; Tempo de execução;

1 **início**
2 Atribua $\varepsilon \leftarrow \emptyset$; $N_j = 1$; e o conjunto de treinamento inicial $\mathbf{D}_0 = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0}$;
3 $f_{N_j} \leftarrow$ obtenha um novo modelo treinado com $\mathbf{D}_0$;
4 Atribua $life_{N_j} = 0$; $MSE_{N_j} = 0$; $w_{N_j} = 1$; $\varepsilon \leftarrow \varepsilon \cup f_{N_j}$;
5 **para** $k = N_0$; $k \leq T$; $k = k + N_k$ **faça**
6 Deslize a janela considerando $\mathbf{D}_{k+1} = (\mathbf{x}_t)_{t=k-(m-N_k)}^{k+N_k}$;
7 **para** $j = 1$; $j \leq N_j$; $j = j + 1$ **faça**
8 Obtenha a previsão de saída do modelo j com $\mathbf{x}_t$;
9 Atribua $life_j = life_j + N_k$;
10 **fim**
11 Obtenha a previsão de ε como: $y_t = F(\mathbf{x}_t) = (\sum_{j=1}^{N_j} w_j f_j(\mathbf{x}_t)) / \sum_{j=1}^{N_j} w_j$;
12 Obtenha as saídas reais $\mathbf{Y}_{k+1} = (\mathbf{y}_t)_{t=k-(m-N_k)}^{k+N_k}$;
13 **para** $j = 1$; $j \leq N_j$; $j = j + 1$ **faça**
14 Obtenha o erro de previsão $e_j^{k+N_k} = (\mathbf{y}_t - f_j(\mathbf{x}_t))^2$;
15 Obtenha o MSE_j de acordo com $life_j$ e e_j;
16 Atribua $w_j = exp(MSE_j - med(\mathbf{MSE})/-(med(\mathbf{MSE})))$, em que $\mathbf{MSE}^{N_j} = [MSE_1, \cdots, MSE_{N_j}]$;
17 Atualize o modelo f_j com $(\mathbf{x}_t, \mathbf{y}_t)$;
18 **fim**
19 **se** $|(F(\mathbf{x}_t) - \mathbf{y}_t)/\mathbf{y}_t| > \alpha$ **então**
20 $f_0 \leftarrow$ obtenha um novo modelo treinado com $\mathbf{D}_{temp} = (\mathbf{x}_t, \mathbf{y}_t)_{t=k-(N_0-N_k)}^{k+N_k}$;
21 Atribua $life_0 = 0$; $MSE_0 = 0$; and $w_0 = 1$;
22 **se** $N_j < MAX_j$ **então**
23 Set $N_j = N_j + 1$;
24 Inclua o modelo f_0 ao ε: $f_{N_j} \leftarrow f_0$; $\varepsilon \leftarrow \varepsilon \cup f_{N_j}$;
25 **fim**
26 **senão**
27 Substitua o modelo com o pior MSE_j: $f_j \leftarrow f_0$;
28 **fim**
29 **fim**
30 **fim**
31 Calcule o tempo de execução e o erro de previsão do fluxo;
32 **fim**

O Passo 19 calcula o Erro Absoluto Relativo do bloco atual e, se ele for maior que o fator α predeterminado, é disparada a condição para adicionar o novo modelo ou substituir um modelo existente. Nesse caso, no Passo 20, um novo modelo é treinado com uma janela de tamanho N_0 e no Passo 21 são atribuídos o peso, a vida e o MSE do novo modelo.

Se o número atual de modelos no *ensemble* for menor que o limite definido (Passo 22), a quantidade de modelos do *ensemble* é incrementada (Passo 23) e o novo modelo será adicionado diretamente (Passo 24). Caso contrário (Passo 26), o modelo com pior MSE será substituído pelo novo modelo (Passo 27).

4.2.3 *Ensemble of Online Learners with Substitution of Models*

O *Ensemble of Online Learners with Substitution of Models* (EOS) (BUENO; COELHO; BERTINI, 2017) é um *ensemble* incremental que implementa um esquema de atualização proposto inicialmente por Street e Kim (2001), que contempla a inclusão e substituição de modelos a uma taxa fixa. O EOS que foi implementado neste trabalho funciona de maneira semelhante ao SCDOER, referenciado no Algoritmo 5, com exceção dos seguintes passos:

- Passo 11 do Algoritmo 5: O EOS não usa a média ponderada nas previsões, e sim a média aritmética, como no EOS-ELM (Passo 12 do Algoritmo 4);
- Passo 16 do Algoritmo 5: não necessário, pois utiliza média aritmética;
- Passo 19 do Algoritmo 5: nesta implementação, o EOS usa uma taxa fixa igual a N_0 para incluir ou substituir um modelo.

O EOS difere de outras abordagens no esquema de inclusão e substituição de modelos, já que incorpora novos modelos a uma taxa fixa, sem qualquer mecanismo implícito para determinar se um desvio de conceito ocorreu. Esta forma de operar ajuda na previsibilidade das inclusões ou substituições.

Cabe ressaltar que a tarefa de incluir ou substituir um novo modelo ao *ensemble* custa bem mais em termos computacionais do que atualização dos modelos. Ainda mais quando são utilizadas janelas maiores nesse processo, já que a complexidade das operações matriciais relacionadas estão diretamente ligadas ao tamanho da janela.

4.3 *Ensembles* de Alto Desempenho

Os *ensembles* mencionados antes foram implementados em duas versões: uma com o padrão MPI e outra serial. O MPI permite paralelizar as tarefas com memória distribuída em vários processos, que podem ser alocados em um único nó computacional ou distribuídos por vários nós computacionais.

Em resumo, a proposta deste trabalho consiste em uma paralelização de dois níveis, alocando cada modelo do *ensemble* em um processo MPI, e paralelizando as funções internas de cada modelo em um conjunto de *threads* de CPU ou em uma GPU. A Figura 4.1 mostra o esquema de funcionamento das versões sem MPI e com MPI.

Com esse conceito, nas versões de alto desempenho com MPI (hpEOS-ELM, hpEOS e hpSCDOER), cada modelo pode executar sua previsão sem esperar pelos outros. Em outras palavras, foram eliminados todos os laços ‘‘`para`’’ que percorrem os modelos dos

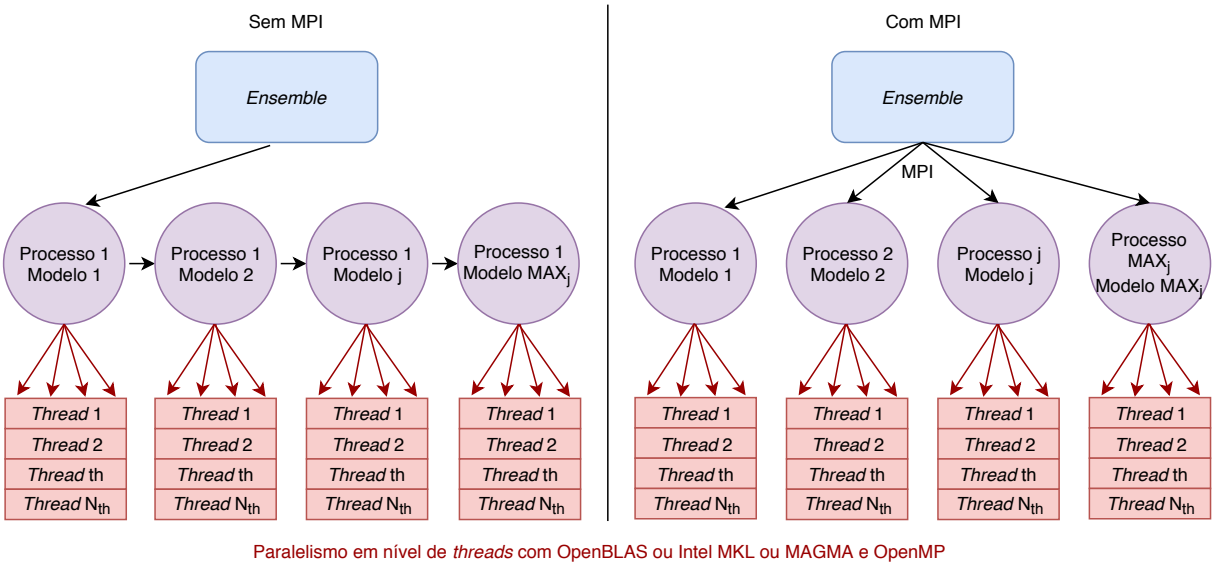

Figura 4.1: Esquema de funcionamento dos *Ensembles* sem MPI e com MPI.

ensembles, e os modelos foram lançados de forma paralela. No hpEOS-ELM, isso compreende os passos 4, 5, 10 e 15 do Algoritmo 4. No hpSCDOER e hpEOS isso compreende os passos 8, 9, 14, 15, 16 (hpSCDOER) e 17 do Algoritmo 5.

No entanto, como se tratam de *ensembles* preditivos *online*, eles precisam ser sincronizados antes de receber dados de cada amostra ou bloco de amostras. A sincronização também é necessária para fazer os *ensembles* aguardarem para receber as saídas reais de cada amostra ou bloco de amostras. Há também um processo mestre para executar algumas tarefas que exigem comunicações entre modelos. A Tabela 4.1 mostra, em forma cumulativa, as transferências de dados entre os modelos e o processo mestre, e as respectivas tarefas do processo mestre.

Tabela 4.1: Transferências de dados e tarefas relacionadas ao Processo Mestre dos *Ensembles* de Alto Desempenho.

Ensembles	**Transferências de dados**	**Tarefas do processo mestre**
Todos	Previsão de cada modelo;	Fazer a previsão do *ensemble*.
EOS e SCDOER	MSE de cada modelo; O pior modelo encontrado.	Localizar o pior modelo.
SCDOER	Peso de cada modelo; Erro rel. abs. do *ensemble*; MSE mediano do *ensemble*; Soma dos pesos dos modelos.	Calcular o MSE mediano de *ensemble*; Calcular o erro absoluto relativo do *ensemble*; Calcular a soma dos pesos dos modelos.

A Figura 4.2 ilustra um esquema geral demostrando como funciona o núcleo *online* dos *ensembles* de alto desempenho, considerando a distribuição dos modelos pelos processos MPI e as sincronizações.

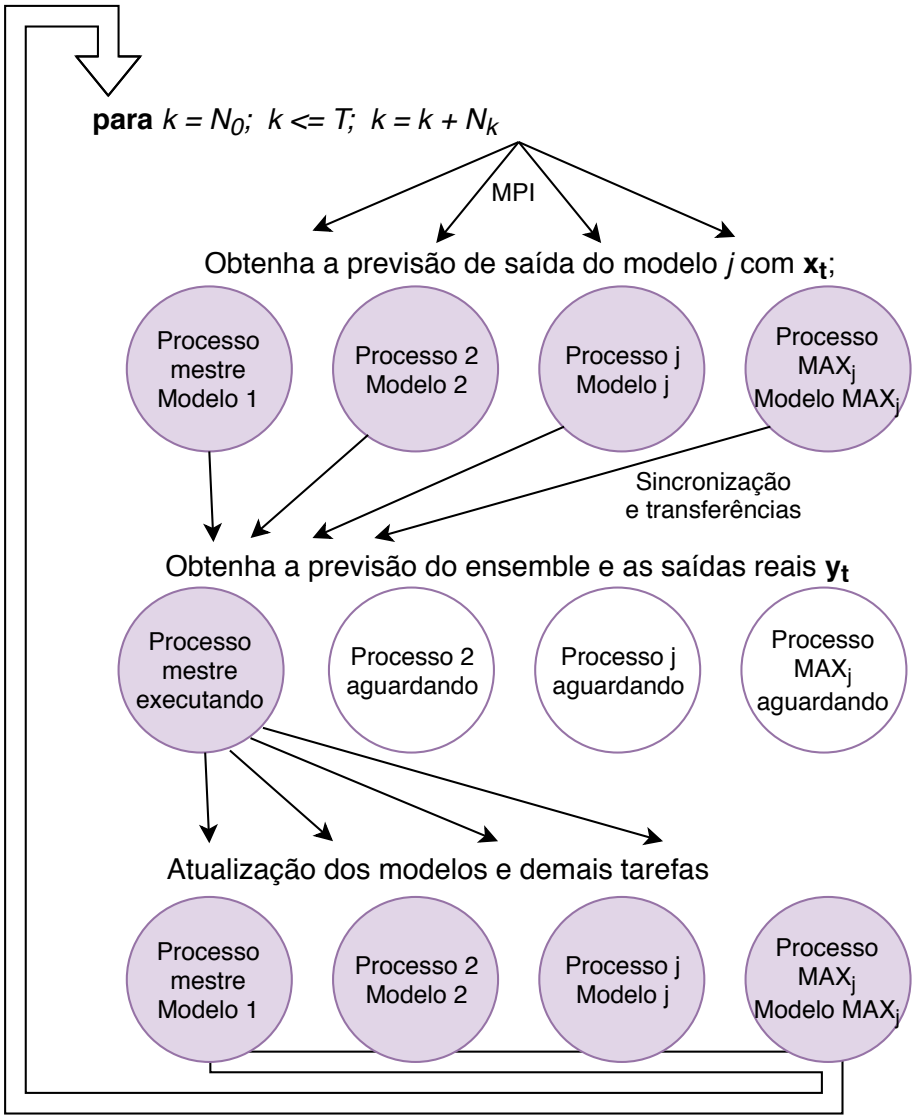

Figura 4.2: Esquema geral de sincronizações dos *Ensembles* de Alto Desempenho.

4.4 Considerações do capítulo

Considerando os algoritmos e estratégias descritos neste capítulo, foram desenvolvidas neste trabalho as seguintes configurações baseadas no OS-ELM:

- *Online Sequential Extreme Learning Machines* com Janela Deslizante (OS-ELM):
  - Versão de *benchmark* em MATLAB (OS-ELM);
  - Versão em C com a biblioteca OpenBLAS (OS-ELMob);
  - Versão em C com a biblioteca Intel MKL (OS-ELMmkl);
  - Versão em C com a biblioteca MAGMA (OS-ELMmgm).

Nas versões em C, o OS-ELMob e o OS-ELMmkl se diferenciam apenas no cabeçalho de inclusão das bibliotecas e na função que copia e gera a transposta das matrizes. O OS-ELMmgm, se diferencia bastante dos demais, já que o MAGMA possui suas próprias funções com diferentes parâmetros de entrada.

Também foram desenvolvidas e avaliadas neste trabalho as seguintes versões dos *ensembles* EOS-ELM, EOS e DOER:

- *Ensemble of Online Sequential Extreme Learning Machine* (EOS-ELM):
  - Versão serial (EOS-ELM);
  - Versão com o padrão MPI (hpEOS-ELM).
- *Ensemble of Online Learners with Substitution of Models* (EOS):
  - Versão serial (EOS);
  - Versão com o padrão MPI (hpEOS).
- *Simplified* C DOER (SCDOER):
  - Versão serial (SCDOER);
  - Versão com o padrão MPI (hpSCDOER).

Essas abordagens permitirão um estudo comparativo das versões quando avaliadas em cenários reais e artificiais. As versões seriais foram desenvolvidas para serem utilizadas no lugar das versões originais dos *ensembles*, que em sua maioria estão em MATLAB. Assim, o comparativo dos *ensembles* fica restrito à execução dos programas em C.

O objetivo dessa comparação é identificar se as versões em que foram aplicadas as técnicas de computação alto desempenho podem apresentar uma melhora significativa no tempo de execução, tanto dos modelos previsão, quanto dos *ensembles*. A novidade desta proposta e principal contribuição deste trabalho é a própria incorporação e comparação das técnicas de computação de alto desempenho nestas abordagens. Os resultados da execução desses algoritmos serão discutidos no próximo capítulo.

Capítulo 5

Metodologia Experimental e Resultados

Este capítulo apresenta a metodologia adotada para avaliar as abordagens propostas no Capítulo 4 em diferentes cenários, bem como os resultados e discussões. Primeiro, são caracterizados os conjuntos de dados utilizados e descritas as etapas de pré-processamento. Em seguida, são abordados os parâmetros iniciais adotados, comuns a todos os ambientes de execução. Posteriormente, são apresentados os resultados experimentais e as discussões.

5.1 Conjuntos de Dados e Etapas de Pré-processamento

O conjunto de dados sintético Hyperplane (HYP) (KOLTER; MALOOF, 2005) foi adotado porque ele é um *benchmark* para algoritmos que tratam fluxos de dados com desvio de conceito. Com o *Hyperplane Data Set Generator*, fornecido por Soares e Araújo (2015b), foi gerado um fluxo de dados com 175 mil instâncias. Uma descrição resumida para este conjunto de dados também pode ser encontrada em (SOARES; ARAÚJO, 2015a).

Em resumo, o conjunto de dados Hyperplane possui quatro conceitos distintos, e ocorrem mudanças abruptas de um conceito para o outro. Cada conceito contém 1/4 do total de instâncias, com 10 atributos de entrada no intervalo $[0, 1]$ e uma variável de saída no mesmo intervalo. Como este é um conjunto de dados sintético, já ajustado para os algoritmos de previsão, não foi utilizada nenhuma etapa de pré-processamento.

O conjunto de dados reais utilizado nos experimentos contém amostras de concentrações de MP_{10}, medidas em micrômetros por metro cúbico ($\mu m/m^3$), coletadas sequencialmente ao longo do tempo. Esse conjunto de dados, fornecido pelo sistema QUALAR (CETESB, 2017), contém amostras horárias coletadas de 01 de janeiro de 1998 a 23 de novembro de 2017, da estação automática de Cubatão – Vila Parisi. São 174.397 amostras, tendo zero como valor mínimo e 1.470 $\mu m/m^3$ como valor máximo.

Entre as estações monitoradas pela CETESB, a estação de Cubatão – Vila Parisi apresentou o maior número de amostras que excedem os padrões de qualidade do ar do Estado de São Paulo, no que se refere a MP_{10}. Cerca de 1.657 medições ultrapassaram os limites dentro do período estudado (CETESB, 2017). Esta estação também contém um dos históricos mais extensos e a base de dados apresenta desvios de conceito recorrentes.

No conjunto de dados de MP_{10} foram necessárias algumas etapas de pré-processamento. O software Amélia II (HONAKER; KING; BLACKWELL, 2011) foi utilizado para o pre-

enchimento das 8.011 amostras faltantes com o valor mais provável. Nos casos em que o Amélia II não resolveu, os valores foram preenchidos por meio de uma interpolação linear.

Foi feita uma análise de *outliers* em que foram normalizadas 964 amostras acima de 350 µm/m^3 (valor máximo da concentração de MP_{10} reportado pela CETESB para a estação de Cubatão – Vila Parisi). Também foi utilizado o filtro Hampel para a detectar e substituir os demais *outliers* presentes no conjunto de MP_{10}. Após essas etapas, todos os dados do conjunto MP_{10} foram normalizados entre $[0, 1]$ pela normalização min-max, como sugerido por Huang (2013).

Em se tratando de um fluxo de dados real, a normalização poderia ser realizada utilizando o mesmo parâmetro de corte apontado. O desafio maior seria para quanto à análise de *outliers*. Seria necessário atrasar o processamento do fluxo para que se pudesse aplicar o filtro Hampel. Ou poderia ser utilizado outro mecanismo de detecção que levasse em consideração apenas as amostras anteriores.

Na última etapa de pré-processamento do conjunto MP_{10}, foi elaborada uma série temporal composta pela amostra atual mais as amostras dos últimos cinco instantes de tempo $x_n = [s_{n-5}, \ldots, s_{n-1}, s_n]$, que foram usadas para prever a amostra do próximo instante $y = [s_{n+1}]$. Ou seja, as amostras dos últimos seis instantes foram utilizadas para prever o valor do próximo instante.

5.2 Configurações Iniciais

Baseando-se em resultados preliminares que testam todas as versões do modelo OS-ELM desenvolvidas, apresentados no Apêndice A, a versão OS-ELMmkl foi escolhida para ser testada em casos com 100 Neurônios Ocultos (NO) e com blocos de atualização de uma instância. Foram realizados testes com o OS-ELMmkl executando como modelo único e como componentes dos *ensembles* desenvolvidos.

Nos testes realizados, as 5.000 instâncias mais antigas foram utilizadas para o treinamento inicial e um fluxo de dados foi simulado com o restantes das instâncias mais recentes de cada conjunto de dados. Isso compreende 170.000 instâncias para o conjunto Hyperplane e 169.397 instâncias para o conjunto MP_{10}. O intuito foi simular um fluxo considerando em torno de 3% das instâncias para treinamento inicial e o restante como fluxo *online*. O fator para adicionar ou substituir um modelo (α) de todas as versões do SCDOER foi definido em $0,04$. Foram realizados testes com o OS-ELMmkl executando como modelo único e como componentes dos *ensembles* em todos os ambientes. Os *ensembles* foram testados com 2, 4 e 8 modelos.

Os experimentos foram repetidos por 20 vezes e conduzidos em diferentes ambientes de execução com configurações distintas, apresentadas aos longo das seções correspondentes. Para cada caso testado, o desempenho foi avaliado coletando as médias e os desvios-padrão dos erros de previsão e dos tempos de execução. Como erro de previsão, foi considerada a Raiz do Erro Quadrático Médio (em inglês *Root Mean Squared Error* – RMSE) entre as saídas reais e previstas do fluxo simulado. Em relação ao tempo, foram considerados os Tempos Reais (TR) de execução de todo o algoritmo, medidos em segundos.

5.3 Resultados Experimentais

Nesta seção, os resultados referentes aos RMSEs obtidos estão restritos ao primeiro ambiente testado (*cluster* IBM–Suse da FT/UNICAMP), já que não há variações significativas (maiores que 10^{-5}) em função do ambiente utilizado. Os resultados completos, com os RMSEs de todos os ambientes pode ser consultado no Apêndice B. Para todos os ambientes testados são apresentados os resultados dos TRs obtidos.

5.3.1 Cluster IBM–Suse da FT/UNICAMP

Os primeiros experimentos foram realizados no *cluster* IBM do Laboratório de Simulação e de Computação de Alto Desempenho (LaSCADo) da FT/UNICAMP. Esse *cluster* IBM é na verdade dividido em dois *clusters* que operam de forma independente. O *cluster* que foi utilizado usa o Sistema Operacional SUSE Linux Enterprise Server 11 e utiliza o seguinte conjunto de *hardware* (FT-UNICAMP, 2017):

- Máquina de *Login* (xcat02): Nó para efetuar login e submeter *jobs*
  - 1 nó 3550M3;
  - 4 núcleos Intel(R) Xeon(R) CPU E5620 @ 2.40GHz;
  - 12 GB de RAM;
- Máquinas de Processamento (idpx01, idpx02): Nós de execução dos *jobs*
  - 2 nós DX360M3;
  - 24 núcleos Intel(R) Xeon(R) CPU X5650 @ 2.67GHz em cada nó;
  - 50 GB de RAM em cada nó;
  - 1 GPU NVIDIA Tesla M2070 em cada nó;

Referente aos softwares, o *cluster* IBM–Suse já continha o Torque (PBS) 3.0.4 como sistema de filas, o compilador GNU gcc–c++ 4.3 e a biblioteca MPICH2 3.0.4 instalados (FT-UNICAMP, 2017). Também foi instalada a biblioteca Intel MKL 2018.2.199 no espaço do usuário, necessária para executar o OS-ELMmkl.

A Tabela 5.1 mostra os RMSEs médios e os Desvios Padrão (DP) do OS-ELMmkl ao variar a janela deslizante (SW) para cada conjunto de dados. Como os RMSEs não mostraram variações significativas (maiores que 10^{-5}) relacionados ao número de *threads*, são apresentados apenas os resultados obtidos com uma única *thread*. Os menores RMSEs de cada conjunto estão destacados em negrito.

Tabela 5.1: RMSEs para o modelo OS-ELMmkl no *cluster* IBM–Suse da FT.

SW	**Conjunto HYP**	**Conjunto $\mathbf{MP}_{10}$**
1	$0,14545 \pm 0,00001$	$\mathbf{0,10800 \pm 0,00004}$
50	$\mathbf{0,14400 \pm 0,00001}$	$0,10831 \pm 0,00004$
100	$0,14402 \pm 0,00001$	$0,10836 \pm 0,00005$

Com base na Tabela 5.1, foram selecionados os casos que apresentaram os RMSEs mais baixos, SW=50 para o conjunto de dados de Hyperplane e SW=1 para o conjunto de dados MP_{10}, para verificar o TR de execução com 1 , 2, 4 e 8 *threads*. Cabe ressaltar que os resultados da Tabela 5.1 também foram tomados como base para todos os ambientes de execução posteriores. A Tabela 5.2 mostra os TRs de execução com diferentes *threads*. Os menores TRs para cada conjunto estão destacados em negrito.

Tabela 5.2: TRs de execução do modelo OS-ELMmkl no *cluster* IBM–Suse da FT.

Conj.	**SW**	**Nº de *threads***			
		1	***2***	***4***	***8***
HYP	50	177,95 ± 38,59	148,89 ± 26,55	**124,01 ± 11,07**	**126,92 ± 11,39**
MP_{10}	1	**4,16 ± 0,18**	6,25 ± 1,31	6,20 ± 0,59	8,50 ± 0,95

A Tabela 5.2 mostra que o OS-ELMmkl executou mais rápido com 4 e 8 *threads* para o conjunto de dados do Hyperplane com SW=50, já que a variância obtida mostra um empate técnico. Para o conjunto de dados MP_{10} com SW=1, a execução mais rápida ocorreu com uma *thread*. Esse comportamento está relacionado à complexidade de várias operações matriciais no núcleo online do OS-ELMmkl, que está diretamente relacionado ao tamanho da SW. Com uma SW muito pequena, as operações matriciais no núcleo *online* do OS-ELMmkl são pouco complexas. Nesses casos não compensa aplicar paralelismo no nível de *threads* ao algoritmo OS-ELMmkl.

Os resultados da Tabela 5.2 foram tomados como base para atribuir 4 *threads* por modelo nos testes com *ensembles* relacionados ao conjunto de dados Hyperplane, que será executado com SW=50. Para os testes com *ensembles* relacionados ao conjunto MP_{10}, que será executado com SW=1, foi atribuído uma única *thread* por modelo. Estas atribuições são válidas para o *cluster* IBM-Suse. Para cada ambiente de execução será verificada a melhor opção quanto ao número de *threads*.

A Tabela 5.3 mostra os RMSEs para os *ensembles* em execução com 2, 4 e 8 modelos. Os resultados são restritos às versões seriais, porque não há variações significativas (maiores que 10^{-4}) entre as versões seriais e as de alto desempenho. Os menores RMSEs de cada conjunto estão destacados em negrito. As tabelas completas de cada ambiente, com todas as versões testadas, podem ser consultadas no Apêndice B.

Tabela 5.3: RMSEs para os *ensembles* no *cluster* IBM–Suse da FT.

Conj.	***Ensemble***	**Nº de modelos**		
		2	***4***	***8***
HYP	EOS-ELM	0,14396 ± 0,00001	0,14395 ± 0,00001	0,14394 ± 0,00000
	EOS	0,07225 ± 0,00007	0,08266 ± 0,00001	0,09276 ± 0,00001
	SCDOER	$\mathbf{0,06383 \pm 0,00044}$	0,06708 ± 0,00073	0,07056 ± 0,00040
MP_{10}	EOS-ELM	0,10792 ± 0,00003	0,10789 ± 0,00001	0,10787 ± 0,00002
	EOS	0,10816 ± 0,00003	0,10788 ± 0,00002	$\mathbf{0,10776 \pm 0,00002}$
	SCDOER	0,10931 ± 0,00004	0,10919 ± 0,00003	0,10910 ± 0,00002

Embora não seja o foco principal deste trabalho, a Tabela 5.3 mostra que os conjuntos de dados apresentam um comportamento diferente dependendo do número de modelos

nos *ensembles*. Com exceção do EOS-ELM no conjunto Hyperplane, o RMSE aumenta na medida em que aumenta o número de modelos, enquanto no conjunto MP_{10} o RMSE diminui na medida em que aumenta o número de modelos.

Também é possível observar na Tabela 5.3 uma diminuição significativa do RMSE para o conjunto Hyperplane nos *ensembles* EOS e SCDOER. Enquanto isso, a diminuição no RMSE para o conjunto MP_{10} é bastante discreta e está restrita aos *ensembles* EOS-ELM e EOS, aumentando no SCDOER. Ao comparar os dados da Tabela 5.3 com a Tabela 5.1 também é possível observar que, exceto nos casos de MP_{10} com SCDOER e com EOS com dois modelos, todos os outros casos relacionados aos *ensembles* apresentam diminuição do RMSE em relação ao modelo único OS-ELMmkl.

Finalmente, A Tabela 5.4, mostra o TR de execução para todas as versões dos *ensembles* e a Figura 5.1 mostra as acelerações dos *ensembles* de alto desempenho relacionadas às suas versões seriais. Nas versões com o sufixo "nd", os processos MPI foram distribuídos entre os dois nós computacionais. As demais versões são executadas em um único nó.

Os testes com os *ensembles* de alto desempenho para o conjunto Hyperplane dentro de um único nó computacional não foram executados, pois haveria concorrência de processos para o caso com 8 modelos, devido à limitação de 24 núcleos de processamento em cada nó para o *cluster* IBM–Suse da FT. Seriam necessários ao menos 32 de núcleos de processamento (8 processos MPI $\times$ 4 *threads* por modelo) para executar todos os casos sem concorrência dentro de um único nó computacional. As versões que apresentaram os melhores TRs de execução para cada *ensemble* estão destacadas em negrito.

Tabela 5.4: TRs de execução para os *ensembles* no *cluster* IBM–Suse da FT

Conj.	***Ensemble***	**Nº de modelos**		
		2	***4***	***8***
HYP	EOS-ELM	252.12 ± 23.78	500.04 ± 54.82	963.87 ± 93.51
	hpEOS-ELMnd	149.50 ± 5.62	156.40 ± 8.53	161.80 ± 8.00
	EOS	278.72 ± 13.49	536.95 ± 47.02	922.42 ± 128.56
	hpEOSnd	166.49 ± 5.20	183.26 ± 6.17	180.26 ± 10.20
	SCDOER	304.20 ± 13.16	463.77 ± 68.12	1004.07 ± 127.69
	hpSCDOERnd	200.10 ± 24.36	213.46 ± 4.61	236.15 ± 2.06
MP_{10}	EOS-ELM	8.77 ± 0.41	18.51 ± 0.64	38.51 ± 4.52
	hpEOS-ELM	4.66 ± 0.10	5.01 ± 0.10	9.79 ± 2.02
	hpEOS-ELMnd	22.86 ± 2.54	27.09 ± 3.11	29.58 ± 2.19
	EOS	22.30 ± 0.32	77.43 ± 14.33	166.56 ± 0.77
	hpEOS	16.88 ± 0.11	29.33 ± 0.36	57.04 ± 5.49
	hpEOSnd	39.85 ± 3.08	51.64 ± 5.27	63.41 ± 8.87
	SCDOER	818.90 ± 68.81	803.01 ± 5.79	818.98 ± 6.77
	hpSCDOER	824.30 ± 107.57	792.16 ± 6.73	818.70 ± 97.41
	hpSCDOERnd	825.24 ± 63.87	826.45 ± 62.21	824.17 ± 64.47

A Tabela 5.4 e a Figura 5.1 mostram que todos os *ensembles* de alto desempenho (versões "hp") apresentaram melhorias no TR de execução nos casos relacionados ao conjunto Hyperplane, executando perto de 6 vezes mais rápido no hpEOS-ELMnd com 8 modelos.

Para o conjunto de dados MP_{10}, todos os *ensembles* de alto desempenho distribuídos pelos dois nós computacionais (versões "nd") foram mais lentas que suas respectivas versões

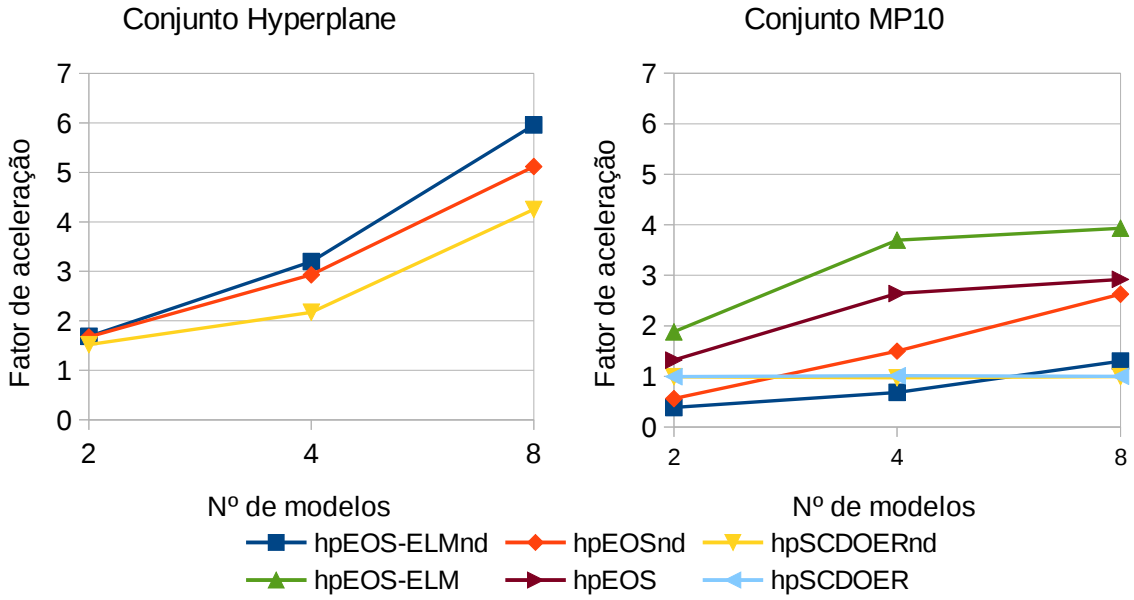

Figura 5.1: Aceleração dos *ensembles* de alto desempenho no *cluster* IBM–Suse da FT.

"hp" que executaram dentro de um único nó. Em alguns casos, as versões "nd" foram até mais lentas que as versões seriais.

Esses resultados ocorreram devido ao atraso da rede na comunicação entre os dois nós de computacionais. Como no conjunto MP_{10}, as operações no núcleo *online* dos modelos são pouco complexas. Por causa da SW de apenas uma instância, a latência da rede influencia muito mais nos resultados do que no conjunto Hyperplane.

Apesar do *cluster* IBM–Suse da FT possuir rede de alto desempenho Infiniband, a biblioteca MPI na versão 3.0.4 não dá suporte total à Infiniband, há apenas suporte parcial provido por terceiros. Cabe ressaltar, também, que nem todas as comunições entre os processos MPI passaram pela rede. Tomando como exemplo o caso com 8 modelos, seriam 4 modelos em cada nó computacional. Sendo assim, as comunicações entre os modelos que estão no mesmo não sofrem o atraso da rede.

Outra análise pertinente à Tabela 5.4 e à Figura 5.1 é que, no conjunto de dados Hyperplane, o hpEOS-ELMnd apresentou as maiores acelerações, seguido por hpEOSnd e hpSCDOERnd. No conjunto MP_{10} ocorreu o mesmo com as versões hpEOS-ELM, hpEOS e hpSCDOER. Esse comportamento está relacionado à dinâmica de operação e às tarefas e transferências do processo mestre de cada *ensemble*.

O EOS-ELM é um *ensemble* estático, ou seja, ele é executado com todos os mesmos modelos em paralelo desde o início e também apresenta a menor quantidade de tarefas e transferências para o processo mestre. O EOS e o SCDOER são *ensembles* incrementais e dinâmicos, isto é, eles começam a executar com um modelo e os outros modelos vão sendo adicionados de acordo com as regras de inclusão.

Na Tabela 5.4 e na Figura 5.1 também cabe observar que as versões hpSCDOER foram as que apresentaram as menores acelerações, especificamente para o conjunto MP_{10}. Esse comportamento se explica porque o SCDOER apresenta a maior quantidade de

tarefas e transferências relacionadas ao processo mestre, e também porque o SCDOER inclui/substitui os modelos com mais frequência que o EOS.

No SCDOER, as inclusões e substituições são determinadas pelo fator α, enquanto o EOS trabalha com uma frequência fixa. Cada inclusão ou substituição de modelo demora mais do que uma simples atualização, fazendo com que os processos dos outros modelos esperem pelo que está sendo incluído ou substituído, atrasando o *ensemble* como um todo.

Os resultados mostraram que os *ensembles* de alto desempenho foram mais rápidos, quando comparados com sua versão serial correspondente, na maioria dos casos. Em geral, o as versões "hp" do EOS-ELM apresentou as maiores acelerações devido à sua simplicidade e também ao fato de não ser incremental, ou seja, ele executa com todos os modelos em paralelo desde o início, diferente do EOS e do SCDOER, que são incrementais.

O atraso na comunicação da rede comprometeu a aceleração das versões "nd", principalmente quando executando com o conjunto MP_{10}, que possui operações de menor complexidade. Baseando-se na questão do atraso da rede, na Seção 5.3.2 são mostrados testes num ambiente com redes de alto desempenho Infiniband, com suporte de Acesso Remoto Direto à Memória (em inglês *Remote Direct Memory Access* – RDMA) .

5.3.2 Cluster Azure

Foi configurado um *cluster* computacional hospedado na nuvem Microsoft Azure, com 8 nós computacionais baseados em instâncias A8 de alto desempenho, totalmente compatível com RDMA/Infiniband. Cada nó computacional possui as seguintes configurações (Microsoft Azure, 2018):

- 1 vCPU Xeon E5-2670 @ 2.60GHz com 8 núcleos;
- 56 GB de RAM; Discos HDD;
- Sistema Operacional SUSE Linux Enterprise Server 12 SP3 HPC 64 bits;
- Biblioteca Intel MKL 2018.2.199;
- Biblioteca Intel MPI para S.O. Linux*, Versão 5.0 Update 3 Build 20150128;

Diferente dos outros ambientes de execução, neste foi utilizado o Intel MPI no lugar no MPICH, pois segundo a documentação da Azure é única biblioteca compatível com os *drivers* RDMA que já vêm pré-instalados nas instâncias de alto desempenho.

Como já foi comentado anteriormente, não será feira análise dos RMSEs obtidos neste e nos próximos ambientes de execução. A Tabela 5.5 apresenta os TRs de execução do modelo único OS-ELMmkl com diferentes *threads*. Os menores TRs para cada conjunto estão destacados em negrito.

A Tabela 5.5 não apresenta nenhuma novidade. Como no *cluster* IBM-Suse, fica evidente que o OS-ELMmkl executou mais rápido com 4 *threads* para o conjunto Hyperplane e com uma *thread* para o conjunto de dados MP_{10}. Da mesma forma, os resultados da Tabela 5.5 foram tomados como base para atribuir 4 *threads* por modelo nos testes com *ensembles* relacionados ao conjunto de dados Hyperplane, que será executado com SW=50.

Tabela 5.5: TRs de execução do modelo OS-ELMmkl no *cluster* Azure.

Conj.	**SW**	**Nº de *threads***			
		1	***2***	***4***	***8***
HYP	50	81,96 ± 0,30	73,66 ± 0,29	**61,56 ± 0,49**	101,64 ± 8,34
MP_{10}	1	**3,74 ± 0,07**	5,89 ± 0,08	6,19 ± 0,09	6,88 ± 0,09

Para os testes com *ensembles* relacionados ao conjunto MP_{10}, que será executado com SW=1, foi atribuído uma única *thread* por modelo.

A Tabela 5.6 mostra o TR de execução para as versões dos *ensembles* executadas no *cluster* Azure e a Figura 5.3 mostra as acelerações dos *ensembles* de alto desempenho relacionadas às suas versões seriais. Foram consideradas apenas as versões em que os processos MPI foram distribuídos entre os nós computacionais. Este ambiente foi criado propositalmente com 8 nós para alocar cada processo MPI em um nó distinto. Dessa forma, cada modelo do *ensemble* possui um nó de execução dedicado, podendo usar todos os seus núcleos de processamento para alocar suas *threads*. As versões que apresentaram os melhores TRs de execução para cada *ensemble* estão destacadas em negrito.

Tabela 5.6: TRs de execução para os *ensembles* no *cluster* Azure

Conj.	***Ensemble***	**Nº de modelos**		
		2	***4***	***8***
HYP	EOS-ELM	127,50 ± 0,54	254,20 ± 1,07	508,17 ± 1,28
	hpEOS-ELMnd	69,41 ± 0,43	74,63 ± 0,53	86,44 ± 1,01
	EOS	136,04 ± 0,70	277,99 ± 1,50	541,64 ± 1,57
	hpEOSnd	81,15 ± 0,47	92,59 ± 0,43	111,72 ± 0,83
	SCDOER	143,01 ± 0,71	281,10 ± 2,09	572,89 ± 2,07
	hpSCDOERnd	91,10 ± 0,63	111,79 ± 2,71	132,53 ± 0,77
MP_{10}	EOS-ELM	8,54 ± 0,08	17,84 ± 0,13	35,33 ± 0,30
	hpEOS-ELMnd	5,62 ± 0,09	6,62 ± 0,08	8,15 ± 0,13
	EOS	19,36 ± 0,09	54,92 ± 0,35	140,59 ± 0,51
	hpEOSnd	15,77 ± 0,08	27,29 ± 0,16	37,91 ± 0,28
	SCDOER	514,04 ± 3,13	524,44 ± 4,75	536,58 ± 2,95
	hpSCDOERnd	544,56 ± 7,39	540,48 ± 5,08	547,80 ± 8,01

Apesar de a Tabela 5.6 mostrar tempos de execução bem diferentes da Tabela 5.4, a Figura 5.2 mostra que as acelerações dos *ensembles* de alto desempenho no *cluster* Azure foram semelhantes às obtidas na Figura 5.1 (*cluster* IBM–Suse FT) para os casos relacionados ao conjunto Hyperplane. Cabe ressaltar, porém, que todas as comunicações inter-modelos passam pela rede no *cluster* Azure, o que não ocorre no *cluster* IBM-Suse.

Entretanto, é comparando as acelerações obtidas com o conjunto MP_{10} na Figura 5.2 com a Figura 5.1 que fica claro a melhoria proporcionada pelo ambiente totalmente compatível com a rede de alto desempenho RDMA/Infiniband. O hpEOS-ELMnd e o hpEOSnd obtiveram acelerações melhores no *cluster* Azure, quando comparadas às obtidas no *cluster* IBM-Suse da FT para o conjunto MP_{10}. Os casos com 8 modelos do hpEOS-ELMnd e do hpEOSnd no conjunto MP_{10} na Figura 5.2 aceleraram mais do que os respectivos hpEOS-ELM e hpEOS na Figura 5.1. Lembrando novamente que os referidos hpEOS-

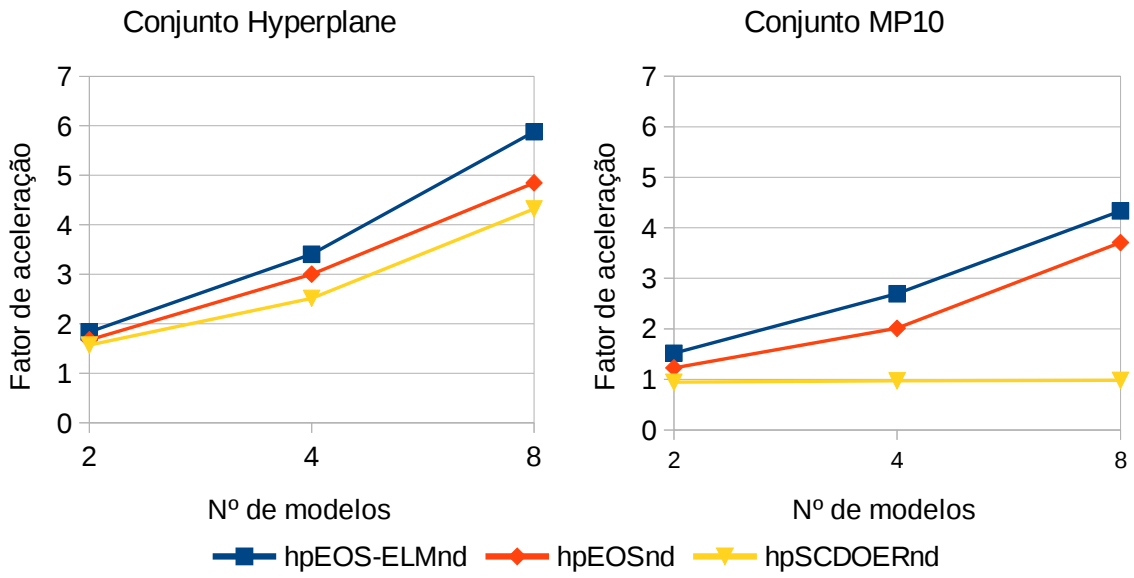

Figura 5.2: Aceleração dos *ensembles* de alto desempenho no *cluster* Azure.

ELMnd e hpEOSnd possuem um modelo em cada nó diferente, já o hpEOS-ELM e o hpEOS executam todos no mesmo nó.

Neste ambiente de execução, os resultados mostraram que, em todos os casos, os *ensembles* de alto desempenho configurados para alocar um modelo em cada nó foram mais rápidos quando comparados com a versão serial correspondente. Em alguns casos comparados, eles aceleraram mais do que versões de alto desempenho que executam dentro de um mesmo nó no *cluster* IBM-Suse da FT.

Isso mostra a importância de se ter um ambiente totalmente compatível com rede de alto desempenho RDMA/Infiniband para executar com os modelos distribuídos por vários nós computacionais.

5.3.3 HPI 1000 Core Cluster

O "1000 Core Cluster" é um dos equipamentos disponíveis no *Future SOC Lab* do *Hasso Plattner Institut* (HPI), sediado na Universität Potsdam. O HPI 1000 Core Cluster possui 25 nós Quanta QSSC-S4R e cada nó apresenta as seguintes configurações (Hasso Plattner Institut, 2018):

- 4 CPUs Intel Xeon E7-4870 @ 2.40GHz com 20 núcleos;
- 1024 GiB de RAM; 8 discos SSD com 450 GiB;
- 2 HBAs Intel Corporation 82599EB 10-Gigabit.

O nó principal executa o sistema operacional SUSE Linux Enterprise Server 11 SP4 e os demais nós executam Ubuntu 16.04.3 LTS. Todos os nós possuem o Slurm 17.02.6 *Workload Manager* e as bibliotecas Intel MKL 2018.2.199 e o MPICH 3.2 instaladas.

A Tabela 5.7 mostra os TRs de execução do OS-ELMmkl com diferentes *threads*. Os menores TRs para cada conjunto estão destacados em negrito.

Tabela 5.7: TRs de execução do modelo OS-ELMmkl no HPI 1000 Core Cluster.

Conj.	**SW**	**Nº de *threads***			
		1	***2***	***4***	***8***
HYP	50	142,69 ± 0,76	160,97 ± 1,72	**133,40 ± 1,04**	442,63 ± 19,97
MP_{10}	1	**4,89 ± 0,06**	7,98 ± 0,14	9,59 ± 0,28	9,67 ± 0,20

Como nos ambientes anteriores, a Tabela 5.7 mostra que o OS-ELMmkl executou mais rápido com 4 *threads* para o conjunto de dados do Hyperplane com SW=50. Para o conjunto de dados MP_{10} com SW=1, a execução mais rápida ocorreu com uma *thread*. O único comportamento fora do esperado, em relação aos outros ambientes, foi que o caso com 2 *threads* foi mais lento do que com 1 *thread* no conjunto Hyperplane. Como nos outros ambientes, foram atribuídos 4 *threads* nos testes com *ensembles* para o conjunto Hyperplane e 1 *thread* para o conjunto MP_{10}.

A Tabela 5.8, mostra o TR de execução para todas as versões dos *ensembles* e a Figura 5.3 mostra as acelerações dos *ensembles* de alto desempenho relacionadas às suas versões seriais. Nas versões com o sufixo "nd", cada processo MPI foi alocado em um nó computacional distinto. As outras versões são executadas em um único nó computacional. As versões que apresentaram os melhores TRs de execução para cada *ensemble* estão destacadas em negrito.

Tabela 5.8: TRs de execução para os *ensembles* no HPI 1000 Core Cluster.

Conj.	***Ensemble***	**Nº de modelos**		
		2	***4***	***8***
HYP	EOS-ELM	266,70 ± 2,92	535,73 ± 4,42	1069,36 ± 7,49
	hpEOS-ELM	126,49 ± 3,32	94,56 ± 4,80	104,94 ± 8,16
	hpEOS-ELMnd	165,73 ± 5,17	179,74 ± 5,07	200,00 ± 5,77
	EOS	291,25 ± 1,96	569,58 ± 2,71	1091,72 ± 5,83
	hpEOS	142,80 ± 5,29	119,93 ± 5,75	141,73 ± 3,76
	hpEOSnd	182,07 ± 4,66	205,70 ± 5,76	234,76 ± 3,95
	SCDOER	328,12 ± 2,14	639,30 ± 4,80	1254,89 ± 6,75
	hpSCDOER	177,66 ± 4,57	185,95 ± 4,46	203,58 ± 5,94
	hpSCDOERnd	230,66 ± 2,69	283,21 ± 5,73	340,72 ± 7,58
MP_{10}	EOS-ELM	10,44 ± 0,06	20,03 ± 0,21	38,44 ± 0,36
	hpEOS-ELM	6,07 ± 0,09	6,58 ± 0,07	7,07 ± 0,10
	hpEOS-ELMnd	23,92 ± 1,94	43,52 ± 5,10	65,44 ± 3,56
	EOS	23,79 ± 0,15	65,44 ± 0,71	166,30 ± 1,35
	hpEOS	18,56 ± 0,36	30,44 ± 0,15	39,41 ± 0,15
	hpEOSnd	42,67 ± 3,42	66,92 ± 2,31	96,25 ± 2,65
	SCDOER	848,90 ± 2,47	875,35 ± 4,77	883,20 ± 3,02
	hpSCDOER	844,10 ± 2,28	837,65 ± 1,18	847,81 ± 1,14
	hpSCDOERnd	893,33 ± 3,52	935,56 ± 7,07	989,76 ± 7,16

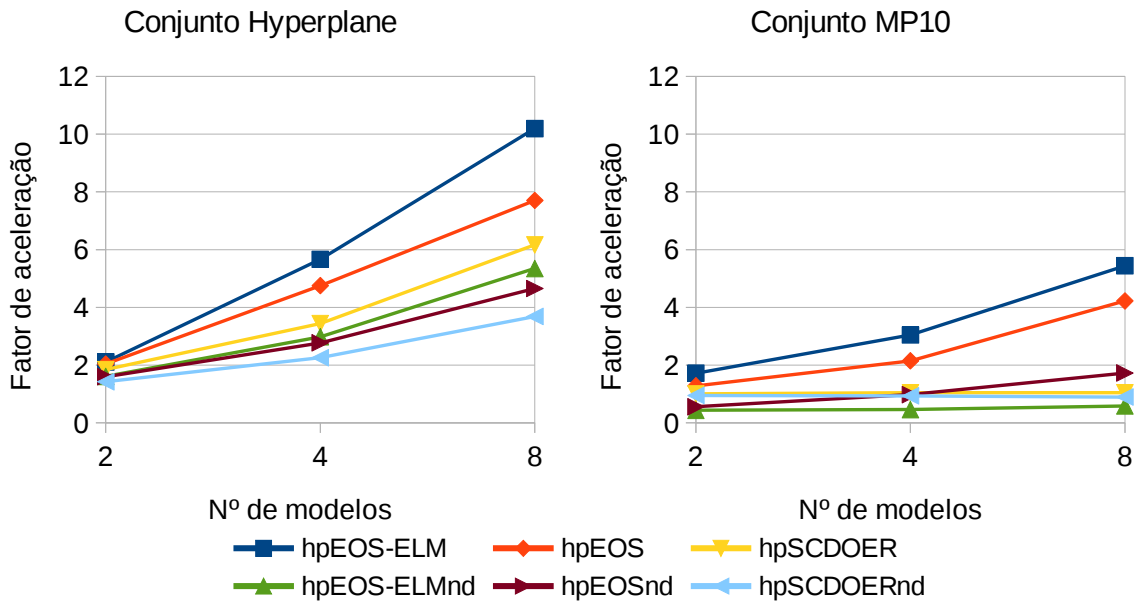

Figura 5.3: Aceleração dos *ensembles* de alto desempenho no HPI 1000 Core Cluster.

A Tabela 5.8 e a Figura 5.3 mostram que todos os *ensembles* de alto desempenho (versões "hp") apresentaram melhorias no TR de execução nos casos relacionados ao conjunto Hyperplane, executando até 10 vezes mais rápido no hpEOS-ELM com 8 modelos.

Para o conjunto de dados MP_{10}, todas as versões de alto desempenho distribuídas em diferentes nós de computação (versões "nd") foram mais lentas que suas versões seriais, exceto para o hpEOSnd com 8 modelos. Neste ambiente fica muito mais evidente o atraso na comunicação através da rede 10 Gigabit Ethernet entre os nós de computacionais. Em todos os casos, as versões "nd" foram mais lentas que suas respectivas versões "hp".

Neste ambiente de execução, os resultados mostraram que os *ensembles* de alto desempenho foram mais rápidos quando comparados com a sua versão serial correspondente na maioria dos casos. Foi também neste ambiente que as versões "hp" que executam dentro de um mesmo nó obtiveram as maiores acelerações. Também ficou mais evidente o impacto do atraso da rede nas versões "nd".

5.3.4 Máquina Virtual HPI XL

Os testes nesse ambiente foram feitos com a intenção de simular a execução em um *desktop* ou *notebook* convencional. Trata-se de uma máquina virtual de tamanho *Extra Large* (XL) hospedada no HPI Future SOC Lab com as seguintes configurações:

- 1 vCPU Core i7 9xx (Nehalem Class) @ 2,39 GHz com 8 núcleos;
- 8 GiB de RAM; Disco 80 GiB HDD;
- Sistema Operacional Ubuntu 16.04.2 LTS;
- Bibliotecas Intel MKL 2018.1.163 e MPICH 3.2;

A Tabela 5.9 mostra os TRs de execução do OS-ELMmkl com diferentes *threads*. Os menores TRs para cada conjunto estão destacados em negrito.

Tabela 5.9: TRs de execução do modelo OS-ELMmkl na máquina virtual HPI XL.

Conj.	**SW**	**Nº de threads**			
		1	***2***	***4***	***8***
HYP	50	161,09 ± 4,14	127,95 ± 3,00	**99,25 ± 3,87**	2636,9 ± 191,41
PM_{10}	1	**5,07 ± 0,05**	5,55 ± 0,40	5,64 ± 0,37	7,15 ± 0,46

Como no ambiente HPI 1000 Core Cluster, a Tabela 5.9 mostra que o OS-ELMmkl executou mais rápido com 4 *threads* para o conjunto de dados do Hyperplane e com uma *thread* para o conjunto de dados MP_{10}. Um dado que chama a atenção na Tabela 5.9 é a demora excessiva que levou a execução do conjunto Hyperplane com 8 *threads*. Como nos outros ambientes, seriam atribuídos 4 *threads* em todos os testes com *ensembles* para o conjunto Hyperplane e uma *thread* para o conjunto MP_{10}. Porém, devido à limitação do *hardware* virtual, foi adotada uma estratégia de redução de *threads* nos casos necessários.

A Tabela 5.10, mostra o TR de execução para todas as versões dos *ensembles* e a Figura 5.4 mostra as acelerações dos *ensembles* de alto desempenho relacionadas às suas versões seriais. Nas versões com o sufixo "tr" foi adotada a estratégia de redução de *threads* mencionada no parágrafo anterior. Essa estratégia foi adotada para os testes com o conjunto Hyperplane devido à limitação de 8 núcleos de processamento. Os casos que englobam *ensembles* com dois modelos, executam com 4 *threads* para cada modelo. Os casos com *ensembles* de 4 modelos, executam com 2 *threads* por modelo. E os casos com *ensembles* de 8 modelos executam com uma *thread* por modelo. Cabe ressaltar que para os *ensembles* seriais (EOS-ELM, EOS e SCDOER) não foi adotada a redução de *threads*, então todos eles executam com 4 *threads* por modelo para o conjunto Hyperplane. As versões que apresentaram os melhores TRs de execução estão destacadas em negrito.

Tabela 5.10: TRs de execução para os *ensembles* na máquina virtual HPI XL

Conj.	***Ensemble***	**Nº de modelos**		
		2	***4***	***8***
HYP	EOS-ELM	185,86 ± 2,59	370,82 ± 4,51	733,39 ± 7,11
	hpEOS-ELMtr	117,43 ± 2,53	153,50 ± 2,82	195,73 ± 5,01
	EOS	199,78 ± 2,75	405,34 ± 6,26	788,69 ± 9,47
	hpEOStr	131,15 ± 3,24	174,49 ± 2,93	223,39 ± 4,65
	SCDOER	252,56 ± 3,25	475,95 ± 5,90	916,47 ± 9,51
	hpSCDOERtr	170,53 ± 2,30	219,62 ± 4,35	294,70 ± 5,50
MP_{10}	EOS-ELM	10,54 ± 0,23	20,57 ± 0,99	39,25 ± 1,61
	hpEOS-ELM	5,92 ± 0,14	6,19 ± 0,32	6,78 ± 0,36
	EOS	24,84 ± 0,99	67,45 ± 2,12	173,11 ± 4,40
	hpEOS	19,12 ± 0,49	31,10 ± 0,43	43,39 ± 0,37
	SCDOER	907,74 ± 16,83	914,53 ± 21,55	944,13 ± 26,20
	hpSCDOER	933,16 ± 11,34	960,10 ± 5,20	1008,95 ± 4,55

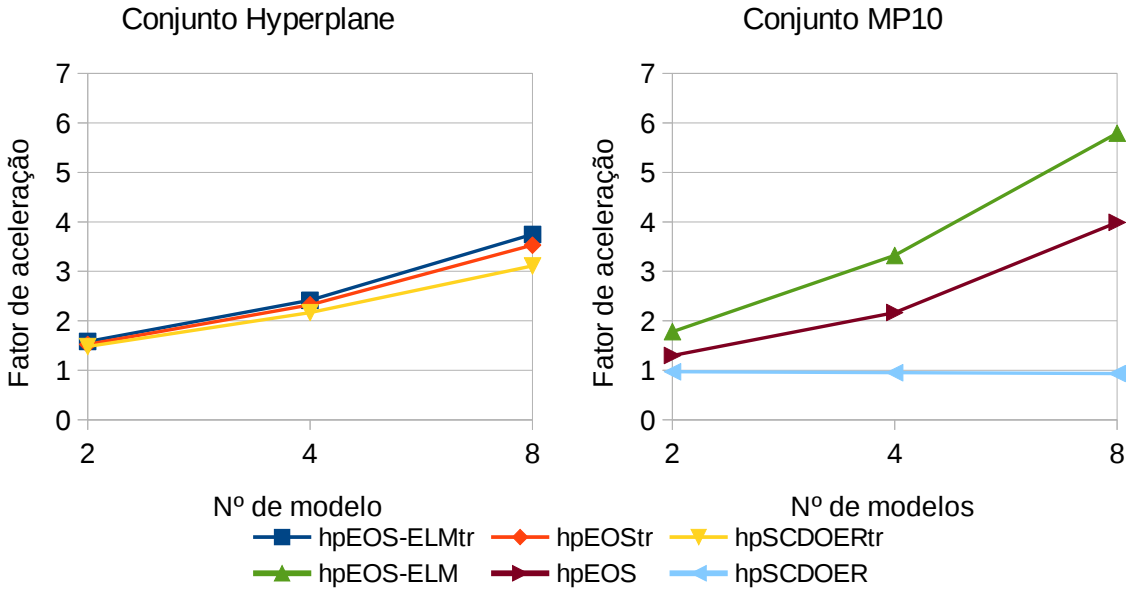

Figura 5.4: Aceleração dos *ensembles* de alto desempenho na máquina virtual HPI XL.

A Tabela 5.10 e a Figura 5.4 mostram que todos os *ensembles* de alto desempenho apresentaram melhorias no TR de execução nos casos relacionados ao conjunto Hyperplane, mesmo com a estratégia de redução de *threads*.

Para o conjunto de dados MP_{10}, o hpEOS-ELM e o hpEOS apresentaram aceleração em todos casos, apesar de o hpSCDOER ter se mostrado mais lento que o SCDOER. Em todos os ambientes de execução, o hpSCDOER apresentou pouca aceleração ou até mesmo piora no desempenho para o conjunto MP_{10}, como ocorreu nesse ambiente.

5.4 Considerações do Capítulo

Neste capítulo foram apresentados os conjuntos de dados, as configurações dos modelos e *ensembles* e resultados obtidos em diferentes ambientes de execução. Em resumo, seguem os resultados mais relevantes de acordo com os ambientes de execução:

- *Cluster* IBM-Suse da FT/UNICAMP: Primeiro ambiente a ser testado, apresentou as análises de RMSE, do comportamento de cada *ensemble* nas versões seriais e de alto desempenho e também o impacto da latência de rede com *ensembles* distribuídos, principalmente para o conjunto MP_{10}. Por não ser totalmente compatível com RDMA, em alguns dos casos as versões "nd" não apresentaram ganhos nos tempos de execução em relação às versões convencionais;

- *Cluster* Azure: Com modelos distribuídos, um em cada nó computacional nas versões de alto desempenho, os resultados mostraram que, num ambiente totalmente compatível com RDMA/Inifiniband, o impacto da latência da rede foi menor do que nos demais ambientes;

- HPI 1000 Core Cluster: Apresentou as maiores acelerações para as versões de alto desempenho quando alocadas dentro mesmo nó computacional. Porém, apresentou as menores acelerações para versões de alto desempenho quando os modelos são distribuídos em diferentes nós computacionais.

- Máquina virtual HPI XL: Com todos os *ensembles* tendo apenas um processador *multicore* para executar todos os modelos e adotando a estratégia de redução de *threads* para o conjunto Hyperplane, os resultados desse ambiente mostraram que nossas abordagens de alto desempenho também podem apresentar ganhos em ambientes convencionais.

Com o fechamento dos resultados e suas discussões, o próximo capítulo apresenta as conclusões deste trabalho e possíveis direções para trabalhos futuros.

Capítulo 6

Conclusões

A principal contribuição deste trabalho foi a incorporação de técnicas de computação de alto desempenho para acelerar a previsão de algoritmos adequados para fluxo de dados com desvios de conceito. Para tanto, este trabalho propôs a implementação dessas técnicas em uma série de *ensembles* com o OS-ELM como modelo de aprendizado. Foram implementados o EOS-ELM, o EOS e o SCDOER em linguagem C, com técnicas de alto desempenho, usando o padrão MPI. Também foram criadas versões seriais para efeitos de comparação.

Para executar como modelos desses *ensembles*, foram implementadas versões paralelas do modelo OS-ELM com as bibliotecas OpenBLAS, Intel MKL e MAGMA. As abordagens implementadas incorporam janelas deslizantes para atualizar os modelos *online* que compõem os *ensembles* ou atuam de forma única. Tais propostas foram avaliadas e comparadas com conjuntos de dados sintéticos e reais.

O conjunto de dados sintético Hyperplane foi adotado porque ele é um *benchmark* bem conhecido para algoritmos que tratam fluxos de dados com desvio de conceito. Também foi utilizado o conjunto de dados reais com amostras de concentrações de MP_{10}, fornecido pelo sistema QUALAR. Quatro ambientes de execução com configurações distintas foram utilizados para analisar o desempenho das abordagens propostas.

Os resultados obtidos mostraram que os *ensembles* de alto desempenho apresentaram acelerações no tempo de execução, quando comparados com a versão serial correspondente, na maioria dos casos. Em todos os ambientes, o hpEOS-ELM apresentou as maiores acelerações, executando até 10 vezes mais rápido que a versão serial no HPI 1000 Core Cluster. Isso ocorreu devido à sua simplicidade e também ao fato de não ser incremental, isto é, ele é executado com todos os modelos em paralelo desde o início, diferentemente do hpEOS e hpSCDOER, que são incrementais.

Quanto aos ambientes de execução, em geral, os *ensembles* de alto desempenho que concentraram sua execução dentro de um único nó computacional apresentaram as maiores acelerações no HPI 1000 Core Cluster. As versões "nd", distribuídas, obtiveram os melhores resultados no *cluster* Azure, totalmente compatível com rede RDMA/Infiniband. Também houve ganhos de tempo nos casos que executaram na máquina virtual do HPI, que simula um *desktop* ou *notebook* convencional.

Mesmo em cenários com maiores taxas de dados recebidos, e. g. 100 instâncias por segundo, todas as implementações abordadas neste trabalho, com os parâmetros adotados,

são adequadas para o processamento do fluxo de dados. Os piores casos (alguns *ensembles* com 8 modelos) levaram em torno de 1100 segundos para processar as 175.000 instâncias, ou seja, 0,006 segundos por instância, o que suportaria em torno de 167.

No caso que apresentou a melhor aceleração, (EOS-ELM com 8 modelos para o Conjunto Hyperplane no HPI 1000 Core Cluster) o tempo foi reduzido de 1069 segundos na versão serial para 104 segundos na versão de alto desempenho executada dentro do mesmo nó. Isso significa aumentar a capacidade de vazão do fluxo de 167 instâncias por segundo na versão serial para 1667 instâncias por segundo na versão de alto desempenho.

Diante dos resultados obtidos, os *ensembles* de alto desempenho propostos neste trabalho podem ser recomendados para qualquer ambiente de execução em casos que utilizem Janelas Deslizantes maiores do que 50 instâncias, já que esse é o parâmetro que determina a complexidade da maioria das operações no núcleo online do OS-ELM. Para os casos que utilizam janelas menores, depende da versão do *ensemble* e do ambiente de execução. As versões de alto desempenho do SCDOER e de alguns casos que distribuem o processamento em vários nós de processamento não mostraram desempenho satisfatório para janelas de tamanho 1.

6.1 Trabalho Futuros

Com base nos resultados obtidos, acredita-se que os conceitos e técnicas de alto desempenho apresentados neste trabalho também podem ser adequados para outros tipos de *ensembles* e modelos de aprendizagem *online*.

Dependendo do caso a ser aplicado, as abordagens apresentadas neste trabalho também poderiam ser executadas em processadores de baixo custo ou embutidos. Um desafio interessante seria a aplicação em *ensembles* compostos por modelos com janela deslizante adaptativa, o que poderia requerer um controle otimizado de *threads* durante a execução dos algoritmos.

Referências Bibliográficas

AKUSOK, A. et al. High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications. *IEEE Access*, v. 3, p. 1011–1025, 2015. ISSN 21693536.

ANDRIJAUSKAS, F. *Detecção de filamentos solares utilizando processamento paralelo em arquiteturas híbridas*. 79 f. Dissertação (Mestrado em Tecnologia) — Faculdade de Tecnologia. Universidade Estadual de Campinas, Limeira, São Paulo, 2013.

BAENA-GARCIA, M. et al. Early drift detection method. *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, p. 77–86, 2006. ISSN 0302-9743.

BIFET, A.; GAVALDA, R. Learning from Time-Changing Data with Adaptive Windowing. *Sdm*, v. 7, p. 2007, 2007.

BOSE, R. P. J. C. et al. Dealing with concept drifts in process mining. *IEEE Transactions on Neural Networks and Learning Systems*, v. 25, n. 1, p. 154–171, jan 2014. ISSN 2162237X.

BRZEZINSKI, D. *Mining Data Streams with Concept Drift*. 89 f. Dissertação (Ciência da Computação) — Poznań University of Technology, Poznań, 2010.

BRZEZINSKI, D.; STEFANOWSKI, J. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, Elsevier Inc., v. 265, p. 50–67, may 2014. ISSN 00200255.

BUENO, A.; COELHO, G. P.; BERTINI, J. R. Online Sequential Learning based on Extreme Learning Machines for Particulate Matter Forecasting. In: *Brazilian Conference on Intelligent Systems (BRACIS)*. Uberlandia: IEEE, 2017. p. 169–174. ISBN 9781538624074.

CAVALCANTE, R. C.; MINKU, L. L.; OLIVEIRA, A. L. I. FEDD: Feature Extraction for Explicit Concept Drift Detection in time series. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. Vancouver: IEEE, 2016. p. 740–747. ISBN 978-1-5090-0620-5.

CAVALCANTE, R. C.; OLIVEIRA, A. L. I. An approach to handle concept drift in financial time series based on Extreme Learning Machines and explicit Drift Detection. In: *Proceedings of the International Joint Conference on Neural Networks*. Killarney: IEEE, 2015. v. 2015-Septe, p. 1–8. ISBN 9781479919604.

CETESB. *Qualidade do ar no estado de Sao Paulo 2015*. São Paulo, 2016. 156 p. Disponível em: <http://ar.cetesb.sp.gov.br/publicacoes-relatorios/>. Acesso em: 30 de Junho de 2018.

CETESB. *Qualar | Qualidade do Ar - Sistema Ambiental Paulista - Governo de SP*. 2017. 1 p. Disponível em: <http://ar.cetesb.sp.gov.br/qualar/>. Acesso em: 30 de Junho de 2018.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, v. 51, n. 1, p. 107, 2008. ISSN 00010782. Disponível em: <http://dl.acm.org/citation.cfm?id=1327452.1327492{\%}5Cnhttp://portal.acm.org/citation.cfm?doid=1327452.1327>.

DONGRE, P. B.; MALIK, L. G. A review on real time data stream classification and adapting to various concept drift scenarios. *2014 IEEE International Advance Computing Conference (IACC)*, p. 533–537, 2014.

ELWELL, R.; POLIKAR, R. Incremental learning of concept drift in nonstationary environments. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, v. 22, n. 10, p. 1517–31, 2011. ISSN 1941-0093.

FT-UNICAMP. *Laboratório de Simulação e Computação de Alto Desempenho – LaSCADo*. 2017. 1 p. Disponível em: <http://www.ft.unicamp.br/lascado>. Acesso em: 30 de Junho de 2018.

GAMA, J. et al. Learning with drift detection. *Brazilian Symposium on Artificial Intelligence*, p. 286–295, 2004. ISSN 0302-9743.

GAMA, J. et al. A survey on concept drift adaptation. *Computing Surveys*, v. 46, n. 4, p. 1–37, 2014. ISSN 03600300.

HAGER, G.; WELLEIN, G. *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton: CRC Press, 2011. 349 p. ISBN 9781439811924.

HAN, J.; KAMBER, M. *Data Mining Concepts and Techniques*. São Francisco: Morgan Kaufmann, 2006. v. 2. 743 p. ISBN 978-1-55860-901-3.

Hasso Plattner Institut. *Equipment - Future SOC Lab*. 2018. 1 p. Disponível em: <https://hpi.de/en/research/future-soc-lab/equipment.html>. Acesso em: 30 de Junho de 2018.

HONAKER, J.; KING, G.; BLACKWELL, M. Amelia II: A program for missing data. *Journal of Statistical Software*, v. 45, n. 7, p. 1–47, 2011. Disponível em: <http://www.jstatsoft.org/v45/i07/>. Acesso em: 30 de Junho de 2018.

HUANG, G.-B. *MATLAB Codes of ELM Algorithm*. 2013. 1 p. Disponível em: <http://www.ntu.edu.sg/home/egbhuang/elm_random_hidden_nodes.html>. Acesso em: 30 de Junho de 2018.

HUANG, G.-B. et al. Extreme learning machine for regression and multiclass classification. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, v. 42, n. 2, p. 513–29, 2012. ISSN 1941-0492.

HUANG, G.-B. et al. Extreme learning machine: Theory and applications. *Neurocomputing*, v. 70, n. 1-3, p. 489–501, 2006. ISSN 09252312.

HUANG, S. et al. Parallel ensemble of online sequential extreme learning machine based on MapReduce. *Neurocomputing*, v. 174, p. 352–367, 2016. ISSN 18728286.

JO, Y.-y.; KIM, S.-w.; BAE, D.-h. GPU-based matrix multiplication methods for social networks analysis. In: *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems - RACS '14*. New York, New York, USA: ACM Press, 2014. p. 309–313. ISBN 9781450330602.

KARUNADASA, N. P.; RANASINGHE, D. N. Accelerating high performance applications with CUDA and MPI. *ICIIS 2009 - 4th International Conference on Industrial and Information Systems 2009, Conference Proceedings*, n. December, p. 331–336, 2009.

KHAMASSI, I. et al. Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, Springer Berlin Heidelberg, v. 7, n. 3, p. 1–23, oct 2016. ISSN 1868-6478.

KIRK, D. B.; HWU, W. M. W. *Programming Massively Parallel Processors A Hands-on Approach*. Amsterdam: Elsevier, 2010. 258 p. ISBN 978-0-12-381472-2.

KOLTER, J. Z.; MALOOF, M. A. Using additive expert ensembles to cope with concept drift. In: *Proceedings of the 22nd international conference on Machine learning - ICML '05*. New York, New York, USA: ACM Press, 2005. v. 119, n. 1990, p. 449–456. ISBN 1595931805.

KRAWCZYK, B. GPU-accelerated extreme learning machines for imbalanced data streams with Concept Drift. *Procedia Computer Science*, Elsevier Masson SAS, v. 80, p. 1692–1701, 2016. ISSN 18770509.

LAKSHMI, K. P.; REDDY, C. R. K. A survey on different trends in Data Streams. In: *ICNIT 2010 - 2010 International Conference on Networking and Information Technology*. Manila: IEEE, 2010. p. 451–455. ISBN 9781424475773.

LAN, Y.; SOH, Y. C.; HUANG, G. B. Ensemble of online sequential extreme learning machine. *Neurocomputing*, Elsevier, v. 72, n. 13-15, p. 3391–3395, 2009. ISSN 09252312.

LIANG, N.-Y. et al. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, v. 17, n. 6, p. 1411–1423, 2006. ISSN 1045-9227.

MATHWORKS. *MATLAB Multicore*. 2017. 1 p. Disponível em: <https://www.mathworks.com/discovery/matlab-multicore.html>. Acesso em: 30 de Junho de 2018.

Microsoft Azure. *Tamanhos de VM de computação de alto desempenho*. 2018. 1 p. Disponível em: <https://docs.microsoft.com/pt-br/azure/virtual-machines/windows/sizes-hpc>. Acesso em: 30 de Junho de 2018.

NVIDIA. *CUBLAS Library*. [S.l.], 2017. 180 p. Disponível em: <http://docs.nvidia.com/pdf/CUBLAS_Library.pdf>. Acesso em: 30 de Junho de 2018.

OZA, N. C.; RUSSELL, S. Experimental comparisons of online and batch versions of bagging and boosting. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*. New York, New York, USA: ACM Press, 2001. p. 359–364. ISBN 158113391X. ISSN 158113391X.

PHRIDVIRAJ, M.; GURURAO, C. Data Mining – Past, Present and Future – A Typical Survey on Data Streams. *Procedia Technology*, v. 12, p. 255–263, 2014. ISSN 22120173.

QUINN, M. J. *Parallel Programming in C with MPI and OpenMP*. Internatio. Columbus: McGraw-Hill Science/Engineering/Math; 1 edition (June 5, 2003), 2003. v. 1. 529 p. ISBN 007-282256-2.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2013. Disponível em: <http://www.R-project.org/>. Acesso em: 30 de Junho de 2018.

SOARES, S. G.; ARAÚJO, R. A dynamic and on-line ensemble regression for changing environments. *Expert Systems with Applications*, Elsevier Ltd, v. 42, n. 6, p. 2935–2948, 2015. ISSN 09574174.

SOARES, S. G.; ARAÚJO, R. An On-line Weighted Ensemble of Regressor Models to Handle Concept Drifts. *Engineering Applications of Artificial Intelligence*, v. 37, p. 392–406, 2015.

SOUZA, R. et al. Using Ensembles of Artificial Neural Networks to Improve PM10 Forecasts. *Chemical Engineering Transactions*, v. 43, p. 2161–2166, 2015. ISSN 22839216.

STREET, W. N.; KIM, Y. A streaming ensemble algorithm (SEA) for large-scale classification. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*. New York, New York, USA: ACM Press, 2001. v. 4, p. 377–382. ISBN 158113391X.

TOMOV, S.; DONGARRA, J.; BABOULIN, M. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, v. 36, n. 5-6, p. 232–240, jun. 2010. ISSN 0167-8191.

VAN HEESWIJK, M. et al. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, Elsevier, v. 74, n. 16, p. 2430–2437, 2011. ISSN 09252312.

WANG, B. et al. Parallel online sequential extreme learning machine based on MapReduce. *Neurocomputing*, Elsevier, v. 149, n. Part A, p. 224–232, 2015. ISSN 18728286. Disponível em: <http://dx.doi.org/10.1016/j.neucom.2014.03.076>.

YADAV, B. et al. Discharge forecasting using an Online Sequential Extreme Learning Machine (OS-ELM) model: A case study in Neckar River, Germany. *Measurement: Journal of the International Measurement Confederation*, Elsevier Ltd, v. 92, p. 433–445, 2016. ISSN 02632241.

ZHANG, X.; WANG, Q.; ZHANG, Y. Model-driven level 3 BLAS performance optimization on Loongson 3A processor. *Proceedings of the International Conference on Parallel and Distributed Systems – ICPADS*, p. 684–691, 2012. ISSN 15219097.

Apêndice A

Experimentos Preliminares

Os experimentos preliminares neste apêndice testam todas as versões do modelo OS-ELM: OS-ELM, OS-ELMob, OS-ELMmkl e OS-ELMmgm. Foram testados casos com 25, 50 e 100 Neurônios Ocultos (NO), valores dentro do intervalo utilizado por Krawczyk (KRAWCZYK, 2016), e blocos de atualização de 1, 10, 100, 1.000, 2.500 e 5.000 instâncias.

Quando os experimentos preliminares foram realizados, as versões desenvolvidas ainda não contavam com o recurso de janela deslizante, nem com o controle de *threads*. Para o conjunto de treinamento inicial, as 5.000 instâncias mais antigas foram utilizadas para o treinamento inicial e um fluxo de dados foi simulado com as 169.397 instâncias mais recentes do conjunto MP_{10}.

Para cada caso testado, o desempenho foi avaliado medindo os erros de previsão e os tempos de execução através das médias e desvios-padrão. Como medida de erro de previsão, foi considerada a Raiz do Erro Quadrático Médio (em inglês *Root Mean Squared Error* – RMSE) entre as saídas reais e previstas do fluxo simulado. Em relação ao tempo, foram considerados os Tempos Reais (TR) de execução de todo o algoritmo. Estes experimentos iniciais foram repetidos em 50 vezes e conduzidos em uma máquina virtual NC6 hospedada na nuvem Microsoft Azure com a seguinte configuração:

- 1 vCPU Xeon E5-2690 @ 2,60GHz com 6 núcleos; 56 GB de RAM; Disco HDD;
- 1 Tesla K80, com *clock* de 823,5 MHz e 11,5 GB de memória;
- Sistema Operacional CentOS 7.3 HPC 64 bits;
- Software MATLAB versão 9.3 (R2017b) *trial*;
- OpenBLAS 0.2.20 e Intel MKL 2018.0.1;
- MAGMA 2.2.0 compilado com CUDA 9.0 e OpenBLAS 0.2.20.

A.1 Resultados Preliminares com os Modelos OS-ELM

A Tabela A.1 mostra os RMSEs médios e o Desvio Padrão (DP) do fluxo simulado do conjunto MP_{10} ao variar o número de NOs e o Tamanho dos Blocos (TB) . Os casos com os menores RMSEs estão destacados em negrito.

Tabela A.1: RMSEs do conjunto MP_{10} para as versões do OS-ELM.

NO	TB	OS-ELM	OS-ELMob	OS-ELMmkl	OS-ELMmgm
25	1	0,10831	0,10830	0,10830	0,10830
	10	0,10832	0,10833	0,10831	0,10831
	100	0,10839	0,10831	0,10832	0,10832
	1.000	0,10855	0,10836	0,10836	0,10836
	2.500	0,10947	0,10840	0,10839	0,10839
	5.000	0,11109	0,10841	0,10841	0,10839
50	1	0,10819	0,10820	0,10819	0,10819
	10	0,10821	0,10820	0,10819	0,10820
	100	0,10828	0.10822	0,10822	0.10821
	1.000	0,10843	0,10826	0,10826	0,10826
	2.500	0,10938	0,10829	0,10830	0,10831
	5.000	0,11100	0,10830	0,10831	0,10832
100	**1**	0,10802	0,10799	0,10800	0,10800
	10	0,10801	0,10799	0,10800	0,10801
	100	0,10809	0,10801	0,10802	0,10803
	1.000	0,10824	0,10806	0,10807	0,10807
	2.500	0,10921	0,10811	0,10810	0,10811
	5.000	0,11086	0,10813	0,10814	0,10814

[1] O DP de todos os casos variou entre 2×10^{-5} e 6×10^{-5}.

O cenário apresentado na Tabela A.1 indica que, apesar da pequena diferença (10^{-5}), os casos com menores RMSEs ocorrem com 100 NOs e com TBs de 1 e 10 instâncias no OS-ELMob. Além disso, na maioria dos casos testados, o RMSE das versões em C (OS-ELMob, OS-ELMmkl e OS-ELMmgm) foram menores que a apresentada pelo OS-ELM (de 10^{-2} a 10^{-5}), sendo que as diferenças aumentavam conforme os TBs também aumentavam. Portanto, o uso das funções `dgesv()` e do parâmetro de corte, no treinamento inicial das versões em C, mostraram resultados mais estáveis.

A Tabela A.2 apresenta as médias e os DPs dos TRs, comparando o OS-ELM, o OS-ELMob e o OS-ELMmkl, que são versões *multithread* e o OS-ELMmgm, executado em arquitetura híbrida. Cabe observar que o MATLAB aplica cálculos *multithread* por padrão desde a versão 2008a (MATHWORKS, 2017) nas funções de álgebra linear e numéricas. Como já foi mencionado, no momento que estes experimentos foram realizados, as versões ainda não possuíam controle de *threads*. Portanto, todas as versões foram executadas com seus respectivos valores padrão, i. e., 6 *threads*, um para cada núcleo virtual.

A Tabela A.2 mostra que as versão OS-ELMmkl foi a mais rápida para TBs de até 100 instâncias. No entanto, para TBs com 1.000 e 2.500 instâncias, o OS-ELM (em MATLAB) foi o mais rápido.

Em contraste com o OS-ELMob e o OS-ELMmkl, a Tabela A.2 demonstra que o OS-ELMmgm, que executa em arquitetura híbrida com a biblioteca MAGMA, foi mais rápido apenas para o TB de 5.000 instâncias. Para TBs de 1, 10 e 100 e 1.000, o OS-ELMmgm foi o mais lento. Em TBs de 2.500, o OS-ELMmgm superou o OS-ELMob e o OS-ELMmkl. A Tabela A.2 também mostra que, nos casos com TBs de 1.000 e 2.500 instâncias nenhuma das versões em C foi mais rápida do que o OS-ELM.

Tabela A.2: TRs de execução do conjunto MP_{10} para as versões do OS-ELM

NO	TB	OS-ELM	OS-ELMob	OS-ELMmkl	OS-ELMmgm
25	1	4,20 ± 0,06	1,12 ± 0,01	**0,55 ± 0,02**	126,07 ± 1,17
	10	1,45 ± 0,04	0,42 ± 0,07	**0,26 ± 0,01**	21,26 ± 0,43
	100	0,93 ± 0,08	**0,75 ± 0,03**	0,75 ± 0,07	10,27 ± 1,57
	1.000	**3.66 ± 0.23**	8,86 ± 0,18	5,49 ± 0,21	12,28 ± 0,42
	2.500	**20,72 ± 0,80**	38,16 ± 0,22	26,50 ± 0,14	24,26 ± 0,56
	5.000	51,70 ± 0,22	124,21 ± 0,21	91,53 ± 0,26	**44,91 ± 0,64**
50	1	7,14 ± 0,46	2,51 ± 0,02	**0,99 ± 0,02**	131,62 ± 1,04
	10	2,56 ± 0,23	0,74 ± 0,03	**0,47 ± 0,01**	21,94 ± 0,46
	100	1,12 ± 0,12	0,98 ± 0,08	**0,88 ± 0,06**	10,08 ± 0,24
	1.000	**3,81 ± 0,14**	9,06 ± 0,14	5,49 ± 0,21	12,38 ± 0,37
	2.500	**20,82 ± 0,69**	38,41 ± 0,22	26,53 ± 0,21	24,37 ± 0,49
	5.000	52,59 ± 0,28	124,58 ± 0,32	92,37 ± 0,32	**45,23 ± 0,63**
100	**1**	18,93 ± 1,05	8,13 ± 0.05	**4,62 ± 0,11**	145,62 ± 1,23
	10	3,94 ± 0,44	1,81 ± 0,06	**1,61 ± 0,12**	23,31 ± 0,43
	100	1,42 ± 0,17	**1,30 ± 0,06**	1,42 ± 0,08	10,32 ± 0,29
	1.000	**4,15 ± 0,16**	9,48 ± 0,15	6,49 ± 0,23	12,46 ± 0,38
	2.500	**20,96 ± 0,59**	38,99 ± 0,21	27,49 ± 0,18	24,85 ± 0,52
	5.000	53.49 ± 0,23	125,98 ± 3,36	93,15 ± 0,37	**45,57 ± 0,56**

[1] Os valores de NO e TB em negrito apresentam as melhores precisões.
[2] Tempos em negrito indicam a versão mais rápida em cada caso.

Para visualizar melhor a aceleração das versões em C comparadas com o OS-ELM, a Figura A.1 apresenta as acelerações dos TRs obtidos.

Além disso, a Figura A.1 reforça que as versões OS-ELMob e OS-ELMmkl apresentaram desempenhos significantemente melhores para TBs de até 10 instâncias, que possuem mais iterações no laço **para** (passos 8 ao 14) do Algoritmo 3, e operações matriciais menos complexas. Com o OS-ELMmgm ocorre o oposto. Essa versão vai passando a funcionar melhor com TBs maiores, que possuem menos iterações no laço e operações matriciais mais complexas. No entanto, mesmo para o maior TB analisado, com 5.000 instâncias, a aceleração do OS-ELMmgm foi de apenas cerca de 1,2 vezes quando comparado com o OS-ELM.

A.2 Considerações deste capítulo

Os resultados mostraram que as nossas versões em C superaram o *benchmark* OS-ELM na maioria dos casos, quando comparados os TRs de execução. Em geral, o OS-ELMmkl apresentou os melhores desempenhos ao usar TBs de 1 e 10 instâncias, acelerando até 7,5 vezes comparado ao OS-ELM. O OS-ELMob foi ligeiramente melhor nos casos com TBs de 100, com 25 e 100 NOs.

Nos casos com as melhores precisões – 100 NOs com TBs de 1 e 10 instâncias – o OS-ELMmkl acelerou até 4,2 vezes comparado ao OS-ELM. Na maioria dos casos, com exceção aos com TBs de 1.000 e 2.500, pelo menos uma das abordagens em linguagem C

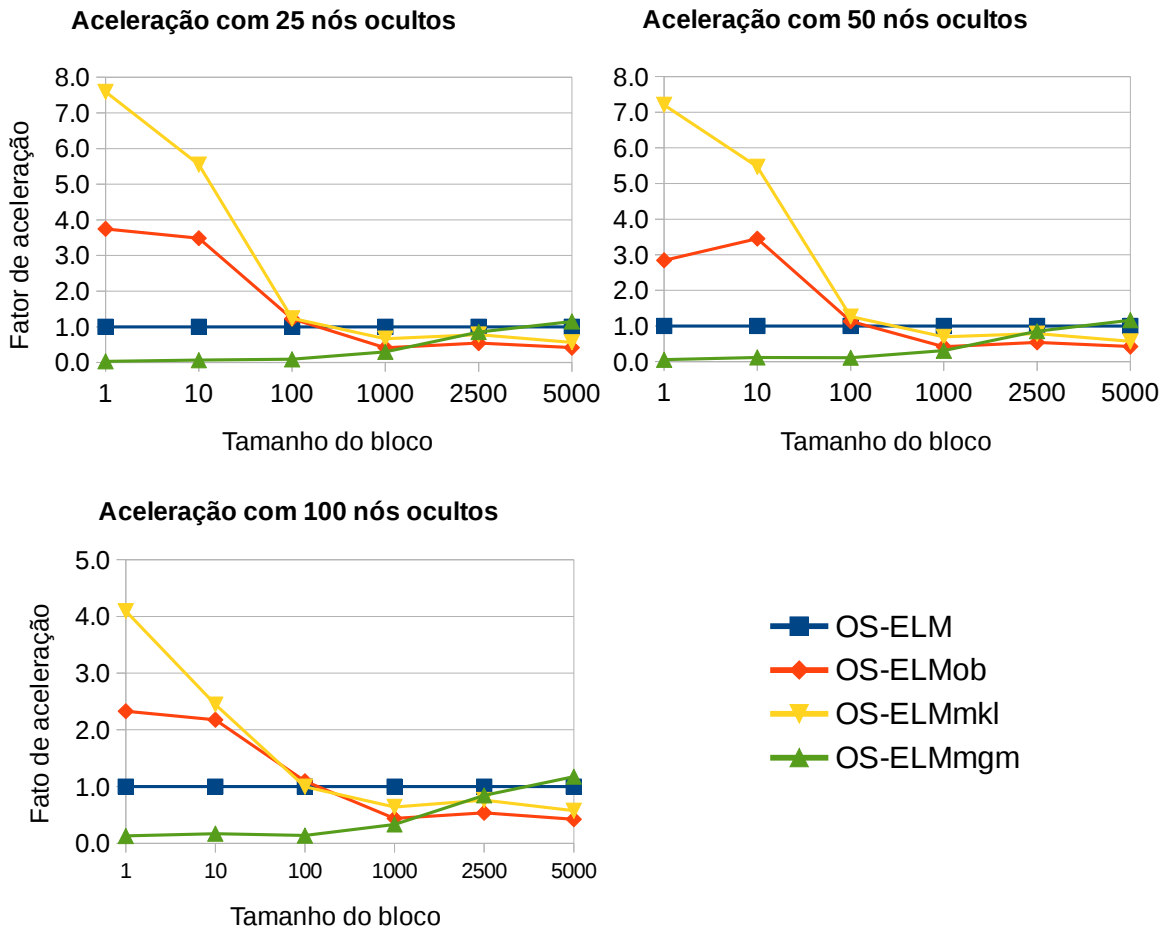

Figura A.1: Aceleração nos TRs do conjunto MP_{10} para as versões do OS-ELM.

apresentou melhor desempenho quanto aos tempos analisados. O mesmo é valido para os RMSEs, em que, na maioria dos casos, foram equivalentes ou menores.

Por outro lado, o OS-ELMmgm, usando arquitetura híbrida, apresentou os melhores desempenhos no tempo de processamento apenas com TBs de 5.000 instâncias. No entanto, esse ganho foi de apenas 1,2 vezes. Esse comportamento está relacionado à sobrecarga de comunicação entre as memórias de CPU e GPU nas funções híbridas do MAGMA. Para blocos menores (TBs até 100 instâncias), devido à grande quantidade de iterações do laço "`para`" (etapa 8 a 14 do Algoritmo 3), o número de chamadas de funções é maior e esse tempo de sobrecarga se torna considerável.

Apêndice B

Erros de previsão em todos ambientes

A Tabela B.1 mostra os RMSEs médios e DPs do OS-ELMmkl ao variar a janela deslizante (SW) para cada conjunto de dados em todos os ambientes de execução. Como os RMSEs não mostraram variações significativas (maiores que 10^{-5}) relacionados ao número de *threads*, são apresentados apenas os resultados obtidos com uma única *thread*.

Tabela B.1: RMSEs para o modelo OS-ELMmkl em todos os ambientes.

Ambiente	**SW**	**Conjunto HYP**	**Conjunto MP_{10}**
Cluster IBM-Suse	1	0.14545 ± 0.00001	0.10800 ± 0.00004
	50	0.14400 ± 0.00001	0.10831 ± 0.00004
	100	0.14402 ± 0.00001	0.10836 ± 0.00005
Cluster Azure	1	0.14545 ± 0.00001	0.10799 ± 0.00004
	50	0.14399 ± 0.00001	0.10832 ± 0.00005
	100	0.14402 ± 0.00001	0.10837 ± 0.00005
HPI 1000 Core Cluster	1	0.14545 ± 0.00001	0.10799 ± 0.00004
	50	0.14400 ± 0.00002	0.10830 ± 0.00005
	100	0.14402 ± 0.00002	0.10836 ± 0.00006
Máquina Virtual HPI XL	1	0.14545 ± 0.00001	0.10799 ± 0.00003
	50	0.14399 ± 0.00001	0.10829 ± 0.00004
	100	0.14402 ± 0.00002	0.10838 ± 0.00004

As tabelas B.2, B.3, B.4 e B.5 apresentam os RMSEs para os *ensembles* em execução com 2, 4 e 8 modelos. Estas tabelas apresentam com todas as versões testadas em seus respectivos ambientes.

Tabela B.2: RMSEs para os *ensembles* no *cluster* IBM–Suse da FT.

Conj.	***Ensemble***	**Nº de modelos**		
		2	*4*	*8*
HYP	EOS-ELM	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	hpEOS-ELMnd	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	EOS	0.07225 ± 0.00007	0.08266 ± 0.00001	0.09276 ± 0.00001
	hpEOSnd	0.07227 ± 0.00009	0.08264 ± 0.00008	0.09276 ± 0.00001
	SCDOER	0.06383 ± 0.00044	0.06708 ± 0.00073	0.07056 ± 0.00040
	hpSCDOERnd	0.06372 ± 0.00042	0.06716 ± 0.00050	0.07059 ± 0.00032
MP_{10}	EOS-ELM	0.10792 ± 0.00001	0.10788 ± 0.00002	0.10787 ± 0.00002
	hpEOS-ELM	0.10792 ± 0.00002	0.10789 ± 0.00002	0.10787 ± 0.00001
	hpEOS-ELMnd	0.10792 ± 0.00002	0.10789 ± 0.00002	0.10787 ± 0.00001
	EOS	0.10817 ± 0.00002	0.10788 ± 0.00001	0.10776 ± 0.00002
	hpEOS	0.10816 ± 0.00003	0.10788 ± 0.00002	0.10776 ± 0.00001
	hpEOSnd	0.10817 ± 0.00003	0.10787 ± 0.00002	0.10776 ± 0.00002
	SCDOER	0.10930 ± 0.00004	0.10918 ± 0.00003	0.10909 ± 0.00002
	hpSCDOER	0.10932 ± 0.00004	0.10918 ± 0.00002	0.10910 ± 0.00002
	hpSCDOERnd	0.10931 ± 0.00004	0.10919 ± 0.00002	0.10910 ± 0.00002

Tabela B.3: RMSEs para os *ensembles* no *cluster* Azure.

Conj.	***Ensemble***	**Nº de modelos**		
		2	*4*	*8*
HYP	EOS-ELM	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	hpEOS-ELMnd	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	EOS	0.07229 ± 0.00008	0.08266 ± 0.00001	0.09276 ± 0.00001
	hpEOSnd	0.07222 ± 0.00005	0.08264 ± 0.00008	0.09275 ± 0.00001
	SCDOER	0.06385 ± 0.00063	0.06687 ± 0.00050	0.07049 ± 0.00033
	hpSCDOERnd	0.06385 ± 0.00045	0.06694 ± 0.00074	0.07071 ± 0.00043
MP_{10}	EOS-ELM	0.10792 ± 0.00002	0.10789 ± 0.00001	0.10787 ± 0.00001
	hpEOS-ELMnd	0.10792 ± 0.00003	0.10790 ± 0.00002	0.10787 ± 0.00001
	EOS	0.10816 ± 0.00003	0.10788 ± 0.00002	0.10776 ± 0.00001
	hpEOSnd	0.10815 ± 0.00002	0.10788 ± 0.00002	0.10777 ± 0.00001
	SCDOER	0.10930 ± 0.00003	0.10919 ± 0.00002	0.10909 ± 0.00002
	hpSCDOERnd	0.10931 ± 0.00005	0.10918 ± 0.00002	0.10910 ± 0.00002

Tabela B.4: RMSEs para os *ensembles* no HPI 1000 Core Cluster.

Conj.	***Ensemble***	**Nº de modelos**		
		2	*4*	*8*
HYP	EOS-ELM	0.14396 ± 0.00001	0.14395 ± 0.00000	0.14394 ± 0.00000
	hpEOS-ELM	0.14396 ± 0.00001	0.14395 ± 0.00000	0.14394 ± 0.00000
	hpEOS-ELMnd	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	EOS	0.07227 ± 0.00009	0.08266 ± 0.00001	0.09276 ± 0.00001
	hpEOS	0.07225 ± 0.00007	0.08266 ± 0.00001	0.09276 ± 0.00001
	hpEOSnd	0.07227 ± 0.00009	0.08265 ± 0.00008	0.09276 ± 0.00001
	SCDOER	0.06377 ± 0.00051	0.06674 ± 0.00054	0.07065 ± 0.00035
	hpSCDOER	0.06352 ± 0.00039	0.06674 ± 0.00063	0.07066 ± 0.00041
	hpSCDOERnd	0.06377 ± 0.00046	0.06694 ± 0.00051	0.07056 ± 0.00039
MP_{10}	EOS-ELM	0.10792 ± 0.00003	0.10789 ± 0.00001	0.10787 ± 0.00002
	hpEOS-ELM	0.10792 ± 0.00003	0.10789 ± 0.00002	0.10787 ± 0.00001
	hpEOS-ELMnd	0.10793 ± 0.00002	0.10789 ± 0.00002	0.10787 ± 0.00001
	EOS	0.10816 ± 0.00003	0.10788 ± 0.00002	0.10776 ± 0.00002
	hpEOS	0.10816 ± 0.00002	0.10788 ± 0.00002	0.10777 ± 0.00002
	hpEOSnd	0.10817 ± 0.00002	0.10787 ± 0.00002	0.10776 ± 0.00001
	SCDOER	0.10931 ± 0.00004	0.10919 ± 0.00003	0.10910 ± 0.00002
	hpSCDOER	0.10931 ± 0.00003	0.10918 ± 0.00002	0.10909 ± 0.00002
	hpSCDOERnd	0.10931 ± 0.00003	0.10918 ± 0.00002	0.10910 ± 0.00002

Tabela B.5: RMSEs para os *ensembles* na máquina virtual HPI XL.

Conj.	***Ensemble***	**Nº de modelos**		
		2	*4*	*8*
HYP	EOS-ELM	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	hpEOS-ELMtr	0.14396 ± 0.00001	0.14395 ± 0.00001	0.14394 ± 0.00000
	EOS	0.07226 ± 0.00008	0.08266 ± 0.00001	0.09276 ± 0.00001
	hpEOStr	0.07228 ± 0.00009	0.08264 ± 0.00008	0.09276 ± 0.00000
	SCDOER	0.06371 ± 0.00049	0.06696 ± 0.00074	0.07067 ± 0.00038
	hpSCDOERtr	0.06376 ± 0.00059	0.06742 ± 0.00068	0.07054 ± 0.00043
MP_{10}	EOS-ELM	0.10792 ± 0.00003	0.10788 ± 0.00002	0.10787 ± 0.00002
	hpEOS-ELM	0.10792 ± 0.00002	0.10788 ± 0.00001	0.10787 ± 0.00001
	EOS	0.10816 ± 0.00003	0.10788 ± 0.00002	0.10777 ± 0.00002
	hpEOS	0.10816 ± 0.00002	0.10787 ± 0.00002	0.10776 ± 0.00002
	SCDOER	0.10932 ± 0.00003	0.10918 ± 0.00003	0.10909 ± 0.00002
	hpSCDOER	0.10930 ± 0.00003	0.10919 ± 0.00003	0.10910 ± 0.00002