UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RICARDO STADTLOBER SABEDRA

Implementação em FPGA de algoritmo para análise de ativos financeiros na bolsa de valores.

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Antonio Carlos Schneider Beck Filho

Porto Alegre
2020

“Share your knowledge.
It’s a way to achieve immortality.”
— Dalai Lama

AGRADECIMENTOS

Realizar uma graduação não é uma tarefa simples. Requer esforço, disciplina e sacrifícios. Felizmente a minha caminhada para conclusão do curso de engenharia da computação não foi solitária. Gostaria de agradecer a pessoas muito importantes, a quem sou eternamente grato por me ajudarem a completar esta fase de minha vida. Primeiramente aos meus pais Jorge Volmer Barreto Sabedra e Márcia Antônia Stadtlober que fizeram todo o possível para propiciar a mim o suporte e o ambiente necessário nessa etapa de minha vida. A minha irmã Marina Stadtlober Sabedra, que com sua gentileza e espírito positivo me motivaram imensamente neste processo. E ao meu orientador, mentor e grande amigo Prof. Dr. Antonio Carlos Schneider Beck Filho, por sempre acreditar em mim e ao esforço para me ajudar em todos os projetos que participamos juntos. Além desses, agradeço fortemente também os seguintes colegas:

Ao Paulo Guilherme Kipper, pela imensa ajuda, amizade, parceria e comprometimento para com ensinamentos importantíssimos na área de aceleradores de hardware.

Ao Daniel Maia Cunha, pelo motivação mútua, companheirismo nas discussões técnicas do projeto e os longos finais de semana que passamos testando soluções nos laboratórios do Campus do Vale.

E ao Filipe Bachini Lopes, pela ampla disponibilidade no compartilhamento de conhecimento e ferramentas que foram cruciais para o sucesso deste trabalho de graduação.

Todos esses me mostraram a grande e valorosa lição de que com a colaboração e o compartilhamento de conhecimento podemos ir muito mais longe.

RESUMO

Com a modernização e evolução tecnológica da bolsa de valores, estão sendo utilizados cada vez mais robôs para realizar negociações de compra e venda de ações. O fenômeno conhecido como *high-frequency trading* visa utilizar algoritmos que operam em altíssima frequência e velocidade para abrir e fechar posições no mercado em frações de segundos. Normalmente esses algoritmos e estratégias são implementados em *software* e, portanto, sua execução depende do uso de recursos da CPU de um computador de propósito geral, que podem não ser suficientes devido à natureza do problema. Este trabalho de graduação mostra a implementação de uma versão de um robô de *High-Frequency Trading* em *hardware*, e compara esta implementação com soluções desenvolvidas em *software*, realizando uma análise dos pontos positivos e negativos de cada implementação. Este trabalho mostra que a versão de *hardware* é muito superior ao *software*, apresentando um desempenho até 2800 vezes maior.

Palavras-chave: Acelerador de hardware. FPGA. HFT. bolsa de valores. performance. hardware heterogêneo.

ABSTRACT

With the modernization and technological evolution of the Stock Exchanges, trading robots for buying and selling stocks are being increasingly used over the years as time passes. This phenomenon is known as "High-frequency trading". It uses high-frequency algorithms to open and close deals in fractions of seconds. Usually, these algorithms and strategies are implemented in software, therefore its execution depends on the resources of a general propose computer CPU, which may not be enough due to the problem's nature. This undergraduate final work states a hardware implemented high-frequency trading robot and compares it with software developed solutions, analysing it's positive and negative points. This last course assignment concludes that the hardware version is highly superior compared to the software one, presenting a 2800 times greater performance.

Keywords: Harware accelerator, FPGA, HFT, stock exchanges, performance, heterogeneus hardware.

LISTA DE ABREVIATURAS E SIGLAS

HFT	high-frequency Trading
ECN	Eletronic Communication Network
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
RSI	Relative Strength Index
FPGA	Field Programmable Gate Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
CPU	Central Process Unit
DMA	Direct Memory Access
FIFO	First in First out
PC	Personal Computer
MACD	Moving Average Convergence / Divergence

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Segundo Irene Aldrige, 2010 (ALDRIGE, 2010), desde o seu surgimento, as bolsas de valores possuem um papel muito importante para vários âmbitos da sociedade em todo o mundo. Essa oferece para qualquer investidor: renda atrelada a um certo nível de risco e abertura para realização investimentos. Outro benefício importantíssimo da bolsa de valores (além de ser uma fonte de investimento), é o de possibilitar que empresas arrecadem fundos com um valor mais baixo que oferecidos por bancos ou financeiras. Promove-se, assim, o aquecimento de toda a cadeia produtiva e econômica dos países.
Esse impacto na economia aumentou muito nos últimos anos com a modernização dos sistemas de mercado. Contextualizando os problemas enfrentados por volta dos anos 80, existia um fator que causava muitas dificuldades para operação da bolsa de valores: o processo de compra e venda de ativos era, quase em sua totalidade, manual e, portanto, exposto a inúmeros erros humanos. A cadeia processual corriqueira para realizar uma ordem de investimento era:

- Contatava-se um representante comercial da bolsa de valores pessoalmente ou via algum envio de mensagem como fax ou telefone.
- O representante anotava o pedido e o preço que o cliente queria negociar.
- O vendedor se dirigia (ou gritava) para o corretor da bolsa a ação que ele estava vendendo ao preço acordado anteriormente.
- Realizava-se uma negociação entre as duas partes (vendedor e corretor).
- O vendedor retornava para relatar o valor de fechamento do processo ao cliente após a negociação, assim como as margens de lucro obtidas.

Este processo era realizado várias vezes ao dia. Além de muito suscetível a erros diversos, era também muito custoso para o investidor (cliente), pois cada etapa da requisição (validar preço do mercado, contatar corretor, negociar preço) estava atrelada a um certo valor.

Ao longo dos anos, o mercado começou a modernizar as suas operações a partir do uso de *Eletronic communication networks - ECNs*. Isto possibilitou uma grande descentralização e principalmente uma maior liquidez do mercado como um todo. Hoje em dia, o mercado financeiro moderno atua com operações que movimentam centenas de bilhões de dólares diariamente. Cada vez mais a bolsa de valores se torna acessível para novos investidores realizarem operações de compra e venda de ativos, devido ao aumento do

uso de tecnologia por parte da própria bolsa de valores e também por corretoras. Como há um aumento do número de investidores, o mercado vem se tornando mais agressivo e disputado.

Portanto, a análise de muitos ativos por parte dos investidores se torna ainda mais importante. Tomar decisões eficientes é essencial para obter lucro atualmente. Para isto, houve um crescimento exponencial de negociações realizadas por robôs em alta frequência (High-Frequency Trading - HFT).

Considerando estratégias para comércio de ativos em alta frequência, *HTF* podem ser divididas em quatro classificações principais, baseadas no tempo de espera na tomada de ação, para venda ou compra de um contrato:

- *Automated liquidity provision*: algoritmos quantitativos otimizados para execução mais rápida possível, por isso atuam com ordens de menos de um minuto.
- *Market microstructure trading*: algoritmos que realizam engenharia reversa de dados de um curto período no passado para realizar uma ação, normalmente atuando por volta dos 10 minutos.
- *Event trading*: são algoritmos específicos para eventos de curta duração sazonais do mercado, tempo de espera de 1 hora.
- *Deviations arbitrage*: utiliza arbitragem estatística e modelos matemáticos, atua em menos de 1 dia.

1.1 Motivação

Para todas essas estratégias apresentadas, além de eficiência, também é necessário desempenho: cada *milisegundo* e ou *nanosegundo* conta para obter lucro ou vantagem sobre outros investidores do mercado. É necessário que os algoritmos para tomada de decisões sejam executados cada vez mais rapidamente. Isso acontece devido ao fato de que as ordens enviadas para a bolsa de valores são computadas em ordem de chegada. Assim, dependendo do atraso entre o envio da requisição e a recepção da mesma no servidor, pode haver uma enorme variação de preço entre o valor esperado e o valor real da compra ou venda. Essa discrepância de valores tem como causa o funcionamento do mercado, pois cada negociação realizada altera o preço de compra e de venda. Neste cenário, aceleradores em hardware baseados em *FPGAs*; (*Field Programmable Gate Array*) têm sido amplamente utilizados para realizar esse tipo de negociação, por serem extrema-

mente velozes, mas também passíveis de reprogramação devido ao mercado estar sempre em constante mudança. Nos dias atuais, um grande número de investidores utiliza plataformas de corretoras para realizarem as suas ordens. Essas plataformas são utilizadas em sistemas operacionais comuns, portanto não otimizados para executar este tipo de tarefa o mais eficientemente possível. O problema principal é referente a velocidade do envio de informação da tomada de decisão. Porque quanto mais rápido for realizada a ordem, maiores as chances de que a ação de compra e venda será realizada no preço específico intencionado. Um exemplo ideal para especificar esse processo: Um investidor quer realizar uma operação de compra de uma ação a qual o preço está em 1,413 pontos com um montante de 10 milhões de reais. Devido ao *delay* de transmissão, ao ser computada pela bolsa de valores o preço desta ação estava 1,423. Ou seja, ao invés de o investidor possuir um lote de ações no valor de R$7.077.140,835 ele possui R$7.027.406,887. Foram perdidos de serem comprados 49 mil reais no processo. Portanto, o trabalho de graduação proposto trata de realizar a implementação e a validação em *FPGA* para algoritmos de análise de bandas de ações para compra e venda, visto que normalmente esses algoritmos possuem um alto custo computacional para serem calculados.

1.2 Solução proposta

Este trabalho de graduação contempla várias áreas do conhecimento: economia, funcionamento de *software* financeiros e a programação de um *hardware* específico para solução de um problema complexo. O projeto foi composto do uso de várias ferramentas, comunicações e implementações diferentes. Os pontos principais da solução são:

- Uso de uma ferramenta para captura da movimentação dos preços das posições ativas na bolsa.
- Implementação em *FPGA* com comunicação *PCI Express* para transmissão de dados.
- Implementar um algoritmo para análise da variação de preço e tomada de decisão, utilizando a linguagem de programação *C/C++*, com a versão implementada em *VHDL* para *FPGA*.
- Desenvolvimento de um modelo de comunicação entre a ferramenta de captura de movimentação de preços e o algoritmo implementado em *FPGA* para cálculos es-

Figura 1.1: Visão geral da arquitetura de comunicação *MetaTrader 5 - algortmo em C++*.

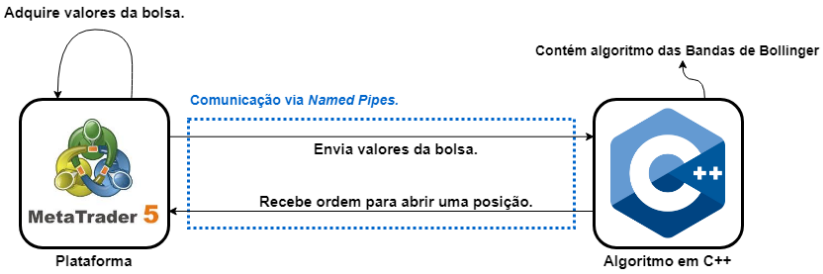
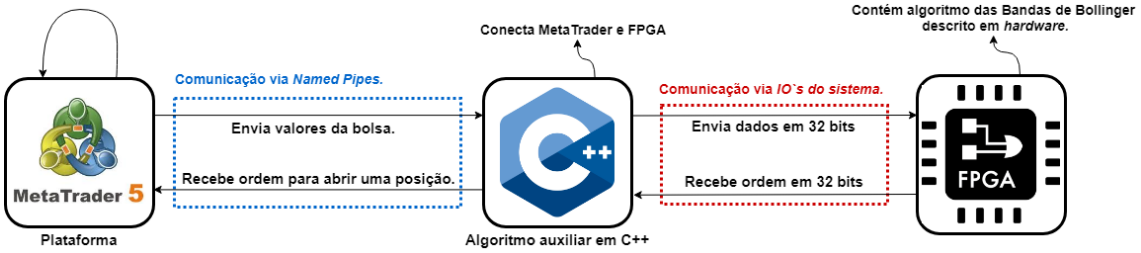


Figura 1.2: Visão geral da arquitetura de comunicação *MetaTrader 5 - FPGA*.



pecíficos utilizando o protocolo de comunicação *PCI Express*.

1.2.1 Visão geral da solução

A seguir serão apresentadas as Figuras 1.1 e 1.2 que compõem uma visão em alto nível da plataforma *MetaTrader 5* comunicando-se com o algoritmo em *C++* e a outra versão para comunicação com o *hardware* respectivamente. Todos os pontos principais como comunicação, desenvolvimento e arquitetura dessa solução serão apresentados nos próximos capítulos neste trabalho de conclusão.

1.3 Contribuições

As contribuições deste trabalho de graduação incluem:

- Uma avaliação de como o *High-Frequency Trading* afetou a bolsa de bolsa de valores.
- Comparação detalhada entre algoritmos de mesmo propósito implementados em *hardware* e em *software*.
- Explicação sobre os algoritmos clássicos utilizados utilizados no mercado de ações.
- Avaliação de performance entre soluções de *hardware* e *software*.

- Discussão sobre modelos de transmissão e recepção de dados.

2 BOLSA DE VALORES MODERNA E HFT

A bolsa de valores no mundo está sempre em constante atualização para satisfazer os seus usuários. Ao longo dos anos a bolsa sofreu muitas mudanças. Hoje, atua de uma maneira robusta e com o objetivo de propiciar maneiras para democratizar o acesso da população aos seus investimentos. O artigo (COSTA; ROSA; BONATO, 2018), coloca que hoje em dia, o mercado financeiro de ativos funciona na seguinte sequência:

1. O cliente gera suas próprias pesquisas e análises quantitativas, baseadas em suas análises de seus ativos;
2. Os clientes utilizam seus próprios algoritmos de distribuição e roteamento para executar as suas ordens de negociação na bolsa;
3. A bolsa informa o cliente diretamente sobre as suas negociações;
4. O corretor recebe informações sobre a negociação executada pelo cliente, e recebe pela liberação do acesso e concessão de privilégios, como melhor preço nas opções ou aberturas de contratos na bolsa.

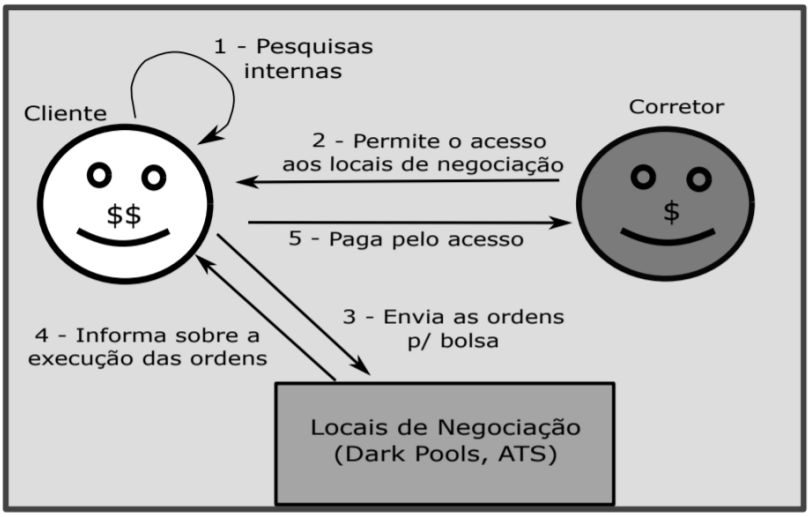
Na Figura 2.1 é apresentado um diagrama esquemático referente ao atual relacionamento do cliente com a bolsa de valores, e como o corretor atua neste meio. A descentralização das operações da bolsa mediante aos corretores possibilitou que o *HFT* se tornasse cada vez mais relevante na bolsa de valores. Principalmente pelo sistema da bolsa ter se tornado mais tecnológico e com menos intermediários no processo. Consequentemente diminuíram as taxas e encargos que os intermediários recebiam, possibilitando que um investidor abrisse mais ordens e realizasse mais negociações em um mesmo período de tempo.

2.1 HFT e suas classificações

High-Frequency Trading surgiu como uma maneira de obter mais ganhos no mercado de ações, devido ao fato de analisar dados e tomar decisões muito mais rapidamente que um ser humano. Em teoria, por mais que o nome seja "Negociação em alta frequência", podem ser classificados como robôs de *HFT* quaisquer tipos de algoritmo que realizem tomada de decisões: desde *nanossegundos* até dias.

Existem inúmeros modelos de algoritmos a serem utilizados para realizar a tomada de decisão, dependendo da preferência do usuário. Os modelos amplamente difundidos

Figura 2.1: Relacionamento entre cliente, bolsa e corretor no mercado moderno de ações.



Fonte: (COSTA; ROSA; BONATO, 2018).

no âmbito do *HFT*, utilizados pela maioria dos investidores, segundo (ALDRIGE, 2010), são os algoritmos de *Market microstructure trading*. Esses se baseiam no histórico dos dados anteriores para tomar a decisão no futuro reagindo a movimentações do mercado já conhecidas e que são similares a eventos do passado.

2.1.1 Algoritmos

Esta seção tem como objetivo principal apresentar os algoritmos clássicos de *Market microstructure trading* apresentados por (ALDRIGE, 2010) para *High-Frequency Trading*. Todas as estratégias que serão apresentadas são utilizadas como indicadores de aberturas de posições de compra ou venda. Esses algoritmos, por serem indicadores, são utilizados muitas vezes por robôs que trabalham com ativos. Mas também, como meio de visualização por investidores, sem que sejam abertas posições automáticas pelo robô. Ou seja, o indicador alerta o usuário, relatando um possível momento propicio para a abertura de uma posição. Abrir uma posição é o nome dado a ação de envio de uma ordem ao sistema da bolsa de valores indicando o desejo de realizar uma compra ou venda de um ativo. Os algoritmos que serão analisados são:

- MACD
- RSI
- Bollinger Bands

2.1.1.1 MACD

O indicador *MACD* ou *Moving Average Convergence Divergence* foi desenvolvido por Gerard Appel em 1979. Segundo (HALILBEGOVIC; SANEL, 2016), este indicador é representado graficamente por duas linhas, que são derivadas de três médias móveis. A primeira linha é chamada de linha *MACD* e a segunda é utilizada como uma linha sinalizadora. Ao utilizar esse indicador, a sua instanciação é normalmente seguida de alguns números: *MACD(12/26/9)*. Esses valores são referentes à variáveis das fórmulas que compõem o cálculo da formação das linhas. No caso do *MACD*, são:

- Média exponencial do últimos 12 dias referentes ao preço do ativo = Media_Exp_12
- Média exponencial do últimos 26 dias referentes ao preço do ativo = Media_Exp_26
- Média exponencial do últimos 9 dias referentes valor do *MACD* = Media_Exp_9_MACD

A fórmula para o cálculo do indicador é:

$$MACD = Media\_Exp\_12 - Media\_Exp\_26$$

E a linha sinalizadora é referente a:

$$Media\_Exp\_9\_MACD$$

Esses valores para cálculo do *MACD* são originários da década de 70, quando a rotina de trabalho era de 6 dias na semana. Mas, ainda são utilizados nos dias de hoje. Portanto, o valor dos últimos 12 dias representa 2 semanas de trabalho, e o número 26 é referente a todos os dias úteis do mês. O valor 9 está conectado com a representação de uma semana e meia. Portando, esses valores cobriam o mês inteiro de flutuações daquela época. Essa seleção de variáveis foi proposta pelo próprio criador do indicador: Gerard Appel.

A representação ilustrativa do indicador *MACD* pode ser visualizada na Figura 2.2. Pontos interessantes de serem visualizados nessa Figura, são referentes aos círculos verdes marcados com uma seta. Esses pontos são momentos propícios ao para aberturas de posições de compra ou venda.

Figura 2.2: Representação do funcionamento do indicador *MACD*.

Fonte: (HALILBEGOVIC; SANEL, 2016).

2.1.1.2 RSI

RSI ou *Relative Strength Index* foi desenvolvido por Welles Wilder Jr. em 1978. A ideia principal desta estratégia está em comparar o crescimento de preço de um ativo com a sua queda em um certo período de tempo. Segundo (MOROSAN, 2011), esse indicador também é utilizado por alguns investidores como medida de força para um ativo na bolsa, ao compará-lo ao mercado o qual esse está inserido.

As fórmulas referentes ao *RSI* são calculadas na seguinte ordem:

1. Calcula-se o valor de crescimento do ativo mapeado de *0 a N* sendo N qualquer número positivo. Diminuindo os preços de fechamento atual e passados:

$$Crescimento = MAP[Fechamento(hoje) - Fechamento(ontem)][0, N]$$

2. Calcula-se o valor de queda do ativo mapeado de *0 a N*, sendo N qualquer número positivo. relacionado os preços de fechamento:

$$Queda = MAP[Fechamento(ontem) - Fechamento(hoje)][0, N]$$

3. Realiza-se o cálculo das médias móveis exponenciais dos valores da queda e do

Figura 2.3: Representação do funcionamento do indicador *RSI*.

Fonte: (INVESTOPEDIA, 2019).

crescimento. Seguido pela divisão de ambas.

$$RS = EMA(crescimento)/EMA(queda)$$

4. Normalizam-se os resultados para se obter um valor entre 0 e 100:

$$RSI = 100 - 100 * 1/(1 + RS)$$

A partir desses cálculos, normalmente segue-se a seguinte regra para abertura de posições de compra e venda: caso o valor do oscilador *RSI* esteja abaixo de 30 pontos abre-se uma posição de compra. Se o mesmo está acima de 70 pontos abre-se uma posição de venda. A Figura 2.3, mostra o funcionamento e a visualização padrão do indicador *RSI*. Nela é possível observar os parâmetros de 70 e 30 pontos, que normalmente são utilizados para abertura posição em ativos.

2.1.1.3 Bandas de Bollinger

Neste trabalho, o modelo escolhido para ser implementado é chamado "Bandas de Bollinger". O modelo tem por objetivo indicar, em qualquer ponto no tempo, os valores máximos e mínimos dos preços de uma posição específica. Esse cálculo é realizado por meio de duas médias móveis, uma positiva e uma negativa sobre a média móvel dos valores de uma ação. É possível visualizar na Figura 2.4 como é o funcionamento deste

Figura 2.4: Envelope - Bandas de Bollinger.

Fonte: (METAQUOTES, 2009).

indicador, realizando os chamados "envelopes"em volta do preço corrente. Neste caso, a condição do envio de uma posição de compra ou de venda acontece por meio dos eventos descritos a seguir.

Caso a variação do preço corrente (linha vermelha) cruze a borda superior de baixo para cima, (linha verde, vide indicador 2 na Figura 2.5), é enviada uma ordem de venda. Quando o contrário ocorre (i.e. o valor em tempo real da variação de preço cruza a banda inferior), é colocada uma ordem de compra. Isso acontece porque esses envelopes são baseados no histórico das médias móveis do preço corrente. Assim, quando ocorrer um cruzamento, as probabilidades de acontecerem uma queda ou subida no preço são muito altas devido ao aumento ou diminuição do volume do mercado.

As fórmulas que compõem os cálculos dos envelopes podem ser visualizadas na Figura 2.6. Nesta Figura estão demostradas as fórmulas da média móvel utilizada para o cálculo do desvio padrão, e dos envelopes inferiores e superiores.

Baseado nessas fórmulas referentes à Fig. 2.6 é possível ter noção de quão custoso é o algoritmo, pois a cada mudança de valores (variação de preço na bolsa) deve ser calculada uma nova média, variância e desvio padrão para as bandas superiores e inferiores. Mais especificamente:

1. Para cálculo da média movel é realizado o somatório dos últimos X valores de preço e depois pela quantidade de números adquiridos.
2. No cálculo do desvio padrão é realizado um somatório em que, para cada nova variação de preço, esse valor é diminuído da média móvel, elevado ao quadrado,

Figura 2.5: Borda de subida - Bandas de Bollinger.

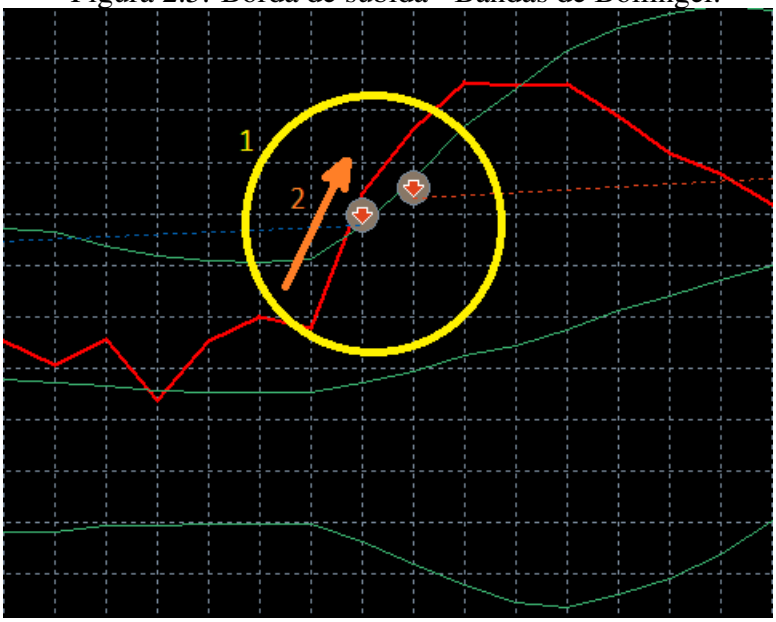


Figura 2.6: Fórmulas de cálculo para as Bandas de Bollinger.

Média móvel $\overline{X} = \frac{\sum_{j=1}^{N} X_j}{N}$

Desvio padrão $\sigma = \sqrt{\frac{\sum_{j=1}^{N}\left(X_j - \overline{X}\right)^2}{N}}$

Banda superior = $\overline{X} + 2\sigma$
Banda central = $\overline{X}$
Banda inferior = $\overline{X} - 2\sigma$

realizado uma divisão pela quantidade de números adquiridos e ao final realizado a raiz quadrada.

3. Após o cálculo do desvio padrão, calcula-se o valor das bandas superior, central e inferior. Sendo média móvel mais duas vezes o desvio padrão para a banda superior, média móvel simples para banda central e média móvel menos duas vezes o desvio padrão para a banda inferior.

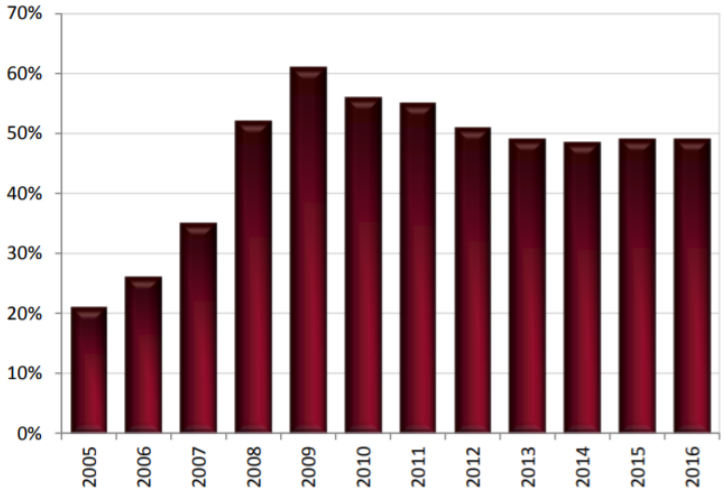
Também é necessário o armazenamento desses valores em filas do tipo *FIFO* (*First in, First out*) para compará-los com a variação de preço corrente, assim finalmente podendo decidir se deve ser realizada uma ação de compra, de venda ou nenhuma das anteriores. É importante ressaltar que cada uma dessas movimentações de preço possa ser verificadas concorrentemente, portanto é necessário uma fila para cada uma delas.

Este algoritmo foi escolhido para ser implementado, pois além de possuir uma performance relativamente eficiente em termos de ganhos para compra e venda de ativos, segundo (BABBAR, 2011), é um algoritmo que exige um processamento significativo, já que realiza um desvio padrão de médias móveis. O que leva a aumentar o tempo de execução do algoritmo, possivelmente diminuindo sua rentabilidade.

2.2 Impacto do HFT nas estruturas de mercado

Baseado no *report* (SUISSE, 2017) da empresa suíça *Credit Suisse* que realiza investimentos financeiros de grande porte na bolsa de valores, o *High-Frequency Trading* trouxe alterações muito significativas e modificou o mercado da bolsa de valores para sempre. *HFT* começou a se tornar relevante em meados de 2008 para 2009, impulsionado principalmente pela crise de 2009. A mudança de mentalidade aconteceu, pois após a crise, os investidores começaram a preferir uma liquidez maior de seus investimentos e não mais antigas estratégias de comprar e só vender as ações muitos anos depois. Na maneira mais antiga estariam totalmente suscetíveis à grandes volatilidades dos períodos de crise. O *HFT* conseguiu resolver esta necessidade de liquidez, visto que são trabalhados diversos montantes de valores, porém por um curto período de tempo. Assim, de certa forma o investidor é mais protegido de perder a valorização dos seus ativos. É possível ver na Figura 2.7 quanto o *trading* em alta frequência se tornou expressivo em 2009: mais de 70% de todo o mercado estadunidense era baseado em *trades* com *HFT*. Ao longo dos anos este modelo vem se estabilizando em uma média de 50% do mercado.

Figura 2.7: *HFT* em porcentagem de todo mercado de ativos dos Estados Unidos ao longo dos anos.



Fonte: (SUISSE, 2017).

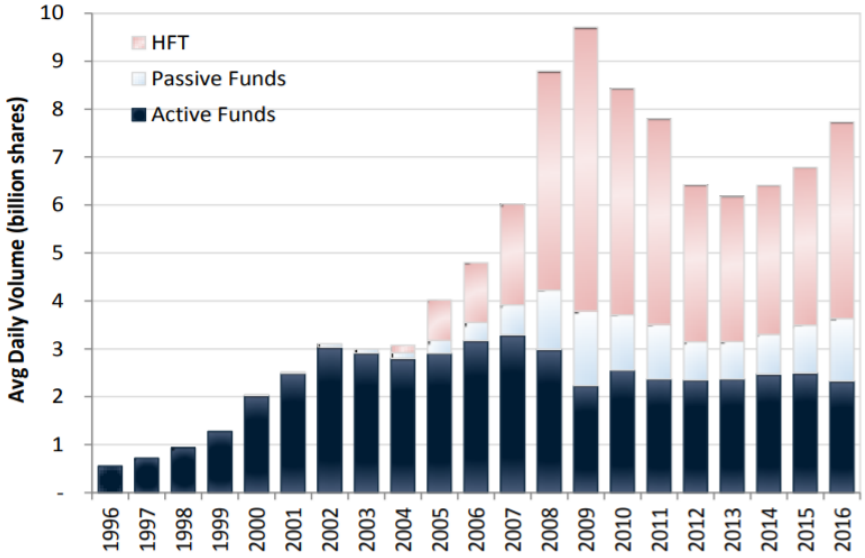
Segundo (SUISSE, 2017), o *HFT* trouxe para o mercado os seguintes impactos na bolsa, que serão explicados individualmente em subseções:

- Significativo aumento de volume no mercado, principalmente para ações de alta liquidez;
- Distorção nas distribuições de volume intra diário. Como o *HFT* começou a servir como um formador de mercado, o volume do mercado durante o dia começou a sair do poder de investidores com grandes concentrações de recursos financeiros;
- Tornou os preços do mercado mais eficientes. Isto acontece devido ao fato do *HFT* ser muito utilizado para arbitragem. Ou seja, quando a compra de uma opção em um ativo e a venda em outro é realizada, o lucro é obtido na diferença entre o preço da compra e venda.

2.2.1 Aumento de volume do mercado

É importante ressaltar que o volume do mercado representa o quanto de recursos financeiros está sendo movimentado em um período de tempo. Portanto, se o mercado está com um alto volume, isto é decorrente de grandes movimentações de ativos. O mesmo serve para um baixo volume. Um dado muito interessante que o artigo (SUISSE, 2017) coloca é o aumento do volume do mercado após o inicio da *mentalidade* de utilizar o *HFT*. Na Figura 2.8 é mostrada uma comparação entre fundos que utilizam *HFT*, fundos ativos controlados por gerenciadores empresariais de altos montantes e fundos passivos que são

Figura 2.8: Volume do *HFT* comparado com fundos passivos e ativos.



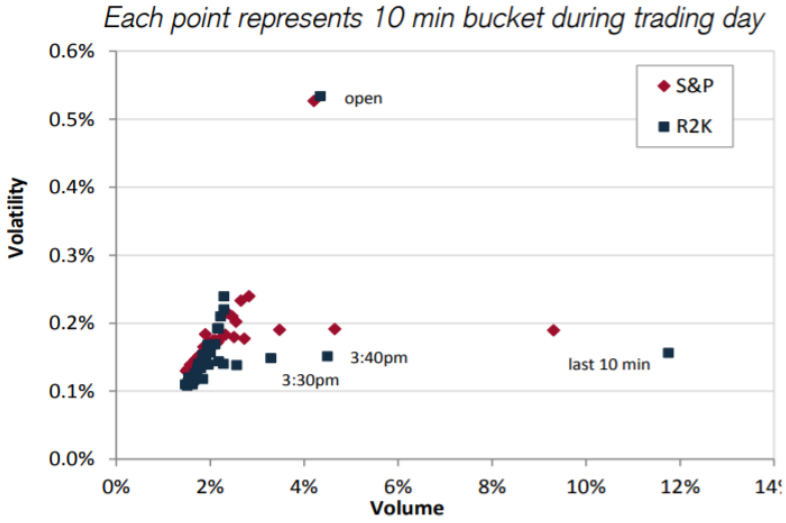
Fonte: (SUISSE, 2017).

oriundos de investidores. Essa comparação reflete a média de volume movimentado (em bilhões) no decorrer dos anos. É interessante observar que os fundos ativo e passivo ao longo dos anos, se mantêm praticamente estáveis enquanto o *HFT*, a partir de 2009, ocupa cada vez mais o mercado.

2.2.2 Distorção de volumes intra diários

O mercado de ações opera relacionado a horário de abertura e fechamento. Quando o mercado abre e começa a operar, chama-se o período *intra diário*. A junção do período de intra diário mais o resto do dia (quando o mercado está fechado), chama-se de período *diário*. Essas duas definições são muito importantes para o funcionamento do mercado e é sabido que a bolsa de valores é muito volátil a notícias e acontecimentos no mundo. Então quando o mercado fecha, existe um fenômeno especulativo sobre a abertura do mesmo no dia seguinte, baseado nas notícias relevantes que aconteceram enquanto a bolsa estava fechada. Por isso, muitas vezes quando o mercado abre, nos primeiros 15 minutos há uma grande volatilidade, referente a acontecimentos positivos ou negativos no mundo. O *HFT*, devido ao fato de possuir uma grande liquidez, acabou aumentando a volatilidade de ações previamente conhecidas como estáveis. Esse fato se reflete no mercado como um todo e flutuações semelhantes ao período de abertura acabam acontecendo em outros momentos do dia do período intra diário. Claramente não há uma volatilidade tão grande como o período de abertura, mas é suficiente para desestabilizar um pouco o mercado.

Figura 2.9: Representação da volatilidade em períodos intra diários do *HFT*.



Fonte: (SUISSE, 2017).

Para refletir este fenômeno, é possível observar a Figura 2.9, na qual são comparados a volatilidade e o volume de dois ativos diferentes. Neste caso o ponto mais importante é que na abertura do mercado há, conforme o esperado, uma grande volatilidade para pouco volume, mas devido ao *HFT*, em outros períodos do dia o mesmo acontece. Isso pode ser visto no gráfico, no horário referente às três horas da tarde.

2.2.3 Eficiência de preço

A arbitragem é uma estratégia para se trabalhar com ativos muito utilizada desde o início das atividades na bolsa. A ideia da arbitragem está em encontrar ineficiências e preços diferenciados para compra e venda de ativos. Um exemplo mais prático do uso deste método em *HFT*, segundo (ALDRIGE, 2010) é a chamada arbitragem triangular. Nesta, são realizadas conversões entre moedas para arrecadar lucros. Por exemplo, se o objetivo é averiguar uma possível abertura de compra para *EUR/CAD*, utiliza-se a fórmula:

$$EUR/CADideal = EUR/USD * USD/CAD$$

Caso o valor real do mercado da conversão *EUR/CAD* for menor do que o calculado na fórmula, a estratégia é abrir uma posição de compra em *EUR/USD*, ou vice-versa. Esses processos de arbitragem na bolsa de valores fizeram com que, segundo (SUISSE, 2017), os preços menos voláteis começassem a possuir mais *gaps* a partir de 2010. *Gaps* no mercado acontecem quando o preço de um ativo encontra-se diferente em duas bolsas

de valores no mundo. Como *HFT* é muito mais veloz do que qualquer ser humano, a chance de um robô encontrar um *gap* é muito maior. Esse fenômeno já era muito conhecido em ativos de pouca expressão, mas começaram a assustar alguns investidores com o passar dos anos. Hoje em dia as ações de baixa volatilidade, chamadas de *Largecaps*, sofreram um crescimento médio de 8 vezes o número de *gaps* encontrados, se comparado com o início do surgimento do *HFT*.

Todos os impactos apresentados demostram o quanto robôs de *trading* em alta frequência estão se tornando expressivos no mercado e todo o seu potencial de mudança do ambiente de *trading*. A tendência futura é o aumento do uso de métodos de *HFT* pelo público geral e grandes corporações. O fenômeno dos *gaps* de preço é um ponto muito interessante a ser estudado em trabalhos futuros, pois segundo (ALDRIGE, 2010), existem maneiras de utilizar o algoritmo das *bandas de bollinger* em conjunto com métodos de arbitragem. E, para uma aplicação eficiente desde conjunto, deve-se executar o algoritmo o mais rápido possível, devido ao tempo curto de permanência de um *gap*, assim conseguindo o máximo de lucro.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentada a fundamentação teórica da solução proposta, que incluem aceleração de hardware, ferramentas utilizadas, modelos de comunicação pesquisados.

3.1 Aceleração de hardware

Aceleradores de *hardware* estão sendo, com o passar o tempo, cada vez mais utilizados para atuarem em soluções desenvolvidas para o mercado. Isso se deve ao fato de que, com a evolução da tecnologia, tornam-se mais fáceis os meios de desenvolvimento e, principalmente, integração de soluções em projetos já existentes. Segundo (PATEL; HWU, 2008), um acelerador é uma subestrutura separada em uma arquitetura, referente a um mesmo *chip* desenvolvido, ou em fase de desenvolvimento. Este acelerador possui objetivos diferentes que o processador principal (de propósito geral), e esses objetivos são originários de uma classe especial de aplicações que se deseja executar mais eficientemente. No artigo (PATEL; HWU, 2008), é colocadas a informação que, costumeiramente, um acelerador de *hardware* provê ao projeto uma vantagem maior que 10 vezes se comparado com o custo ou performance anterior adição da subestrutura.

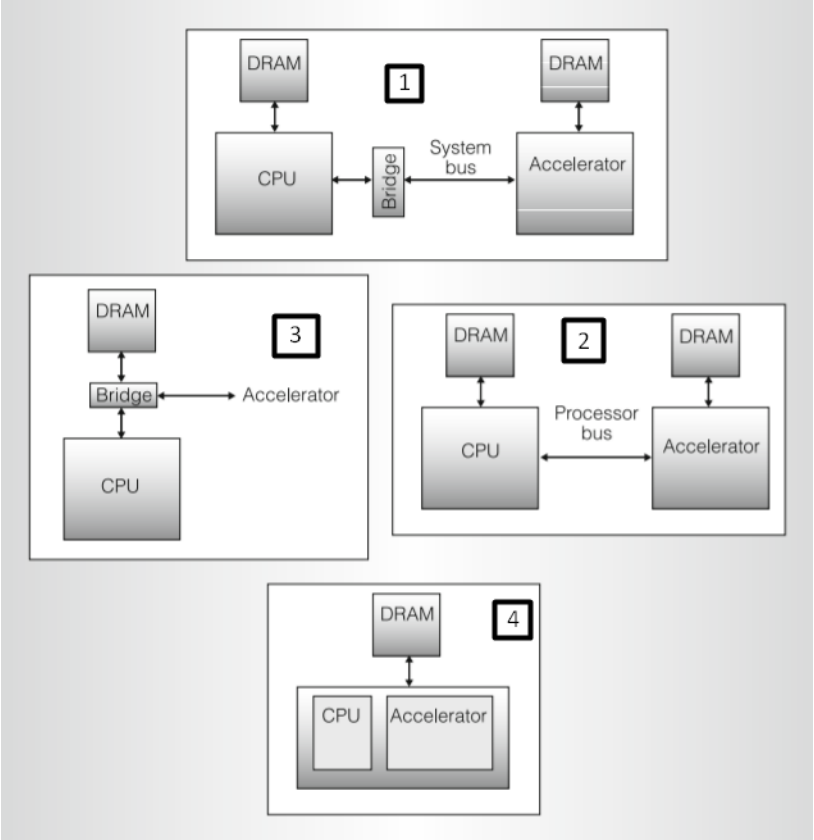
Alguns exemplos clássicos para uso de aceleradores são:

- Coprocessadores de unidades *floating-point*.
- Renderização de vídeo utilizando placas gráficas.
- Estimação de movimento em *codecs* de vídeo.

O artigo (PATEL; HWU, 2008), também apresenta uma série arquiteturas em alto nível para representação de como são utilizadas e desenvolvidas soluções com aceleradores de *hardware*. É possível ver na Figura 3.1 quatro modelos de estruturas:

1. Conectando-se o acelerador utilizando vias do sistema para comunicação.
2. Conectando-se o acelerador e realizando transmissão de dados através da via processador.
3. Utilizando o acelerador para aumentar a performance entre a comunicação da memória com a *CPU*.
4. Acoplando a *CPU* e o acelerador em uma mesma arquitetura compartilhada.

Figura 3.1: Modelos de arquiteturas para uso de aceleradores.



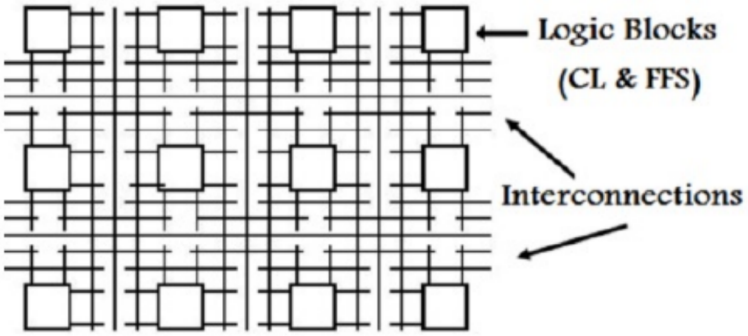
Fonte: (PATEL; HWU, 2008).

Estas estruturas representam os modelos mais clássicos e amplamente utilizados no mercado para se trabalhar com aceleradores de *hardware*. Na solução apresentada nessa monografia, o acelerador de *hardware* utiliza o primeiro modelo de arquitetura, a qual a placa realiza transmissões de dados através de vias do sistema e uma *bridge* de conexão. Nas próximas subseções serão apresentadas ferramentas, estruturas e arquiteturas cruciais que possibilitaram o desenvolvimento da solução deste trabalho.

3.1.1 FPGA

Segundo o artigo (SULAIMAN et al., 2009), *Field programable gate arrays*, mais conhecida como *FPGAs*, são *chips* de desenvolvimento que contém internamente: vetores de duas dimensões de blocos lógicos e *flip-flops* com conexões elétricas entre estes blocos. Devido a *FPGA* ser programável, é possível desenvolver rapidamente diferentes modelos de circuitos elétricos sem necessariamente mandar para fabricação. Ou seja, é um componente de hardware intermediário de desenvolvimento que pode ser usado para prototipação ou tarefas específicas cujo volume não justifica a produção em circuitos es-

Figura 3.2: Blocos lógicos e suas interconexões em FPGAs.



Fonte: (SULAIMAN et al., 2009).

pecíficos.

Na Fig. 3.2 é possível observar a representação de como é uma estrutura de blocos lógicos e suas interconexões dentro de uma placa *FPGA*.

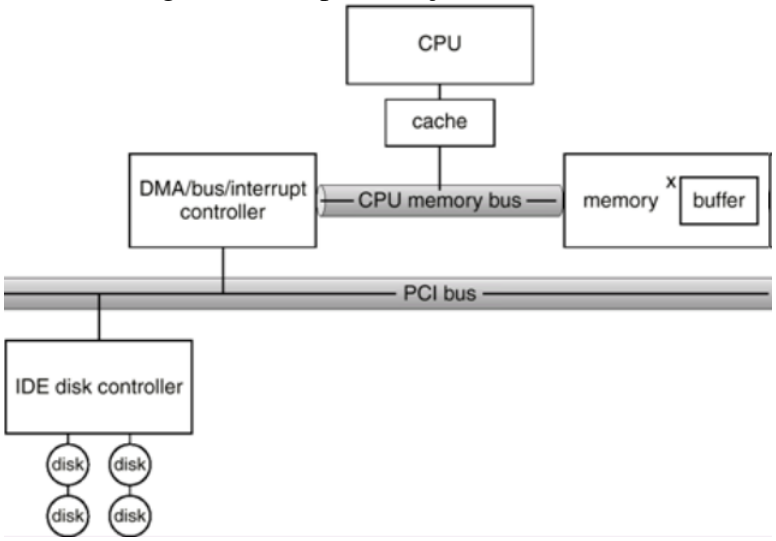
3.1.2 Direct Memory Access

O livro (INSTRUMENTS, 2001) coloca que *Direct memory access*, ou mais conhecido como *DMA*, é um mecanismo de *hardware* que permite acesso direto a memória por periféricos do sistema. A principal vantagem de uma *DMA* é realizar esse acesso sem envolver o processador do sistema. Esta solução propicia um aumento muito significante de vazão nas transmissões de dados. Não obstante, esse mecanismo é utilizado por diversas soluções de *hardware*, como o *Xillybus* e pelas ferramentas desenvolvidas pela *NETFPGA*. Ambos os tópicos, *Xillybus* e *NETFPGA*, serão apresentados posteriormente neste trabalho. Alguns pontos negativos sobre o *DMA* são referentes ao fato de que, para funcionar com alta vazão, é necessário o suporte de um controlador. Esse terá a função de realizar a sincronização da memória com todos os dispositivos comunicantes, realizar o controle de fluxo e ordem de pacotes. Este mecanismo é utilizado principalmente para a comunicação da *CPU* com algum periférico que utiliza a *PCI Express*, por exemplo, tendo o objetivo de que essa comunicação aconteça o mais rápido possível.

É possível observar na Figura 3.3 um exemplo de funcionamento em alto nível de um mecanismo *DMA* para transferência de dados de um disco rígido para a *CPU*. Esse segue o seguinte fluxo, retirado de (SILBERSCHATZ; GAGNE, 2002):

1. O *driver* do dispositivo recebe uma chamada para transferir dados de um disco rígido para o endereço *X* na memória RAM.

Figura 3.3: Representação de uma *DMA*.



Fonte: (SILBERSCHATZ; GAGNE, 2002).

2. O *driver* responde ao controlador do disco (IDE Disk Controller), para realizar a transferência de um certo número de bytes para o endereço *X*.
3. Controlador do disco inicia a transmissão de dados pelo *DMA*.
4. *DMA* envia os bytes recebidos, diretamente para a memória.
5. Após o término da transferência, o *DMA* envia uma interrupção para a *CPU*, indicando a finalização do processo.

3.1.3 FIFO

Uma *FIFO*, acrônimo em inglês de, *First In First Out*, é uma estrutura muito importante na área da computação, pois ela representa uma fila. Essa estrutura é normalmente implementada utilizando um vetor especial, em que, o dado lido deste vetor será: sempre a informação escrita à mais tempo na estrutura e quando realizada a leitura, o dado é excluído do vetor. Ou seja, quando se escreve um dado nesta fila, esse só será lido quando todos os outros dados previamente escritos tiverem sido lidos e excluídos da estrutura.

Em uma estrutura desde modelo só é permitida a escrita ou leitura de um valor por vez. Esse modelo de fila está sempre associado a uma largura, a qual é representada pelo tamanho dos dados que a compõem, e a uma profundidade, relacionada a quantos dados esta *FIFO* pode armazenar.

Nos dias de hoje, filas *FIFO* são utilizadas para diversos fins. Em aceleradores de

hardware os usos mais comuns são:

- Transmissão de dados via protocolos de comunicação.
- Armazenamento de dados de vídeo
- Alinhamento de dados para processamento posterior

Ao utilizar *FIFOs* em *FPGAs*, essas comumente são síncronas, ou seja, computam dados baseados na subida ou decida da borda do relógio interno da placa. Elas podem também, possuir um *clock* próprio para trabalhar com dados independentemente do *hardware* acoplado. Em *FPGA* sua implementação é geralmente baseada em memórias *BRAM*. *"Block RAM"* é uma memória pequena e rápida encontrada em *chips* de *FPGAs*, que pode ser acessada em qualquer ciclo ou via registradores, respeitando a borda do relógio.

Em uma *FPGA*, a *FIFO* normalmente segue uma representação padrão que pode ser visualizada na Figura 3.4. Nessa é possível observar os diferentes sinais que compõem a estrutura de uma *FIFO*, estes são:

- Relógio, utilizado na sincronização das entradas e saídas.
- Habilitar escrita, para sinalizar a fila que há um dado válido a ser escrito na próxima borda de subida do relógio.
- Escrita de dados, que possui as informações a serem escritas na *FIFO*.
- Fila cheia, a qual indica que a fila não possui mais espaço para armazenamento de dados.
- Habilitar leitura, que neste caso funciona como uma requisição de dados para a estrutura.
- Leitura de dado, sinal o qual a *FIFO* enviará informações na borda de subida do relógio após devida requisição pelo sinal de *habilitar leitura*.
- Fila vazia, indica se a estrutura não possui nenhum dado.

Apesar do funcionamento de uma *FIFO* ser simples, é muito importante que a sincronização dos relógios com a placa seja respeitada, e principalmente os sinais de fila vazia e fila cheia. Caso seja requisitada uma leitura quando a *FIFO* está vazia, ou uma escrita quando a mesma se encontra cheia, os dados de saída ou de entrada serão inconsistentes e imprevisíveis naquele momento.

Algumas variações de *FIFO* para *FPGA* contêm sinais muito úteis para aumentar a velocidade de vazão dos dados. Estas *flags* são chamadas de *Fila quase vazia* e *Fila quase cheia*. A utilização delas é bem interessante, pois possibilita que o usuário possa

Figura 3.4: Representação de uma *FIFO* em *FPGA*.

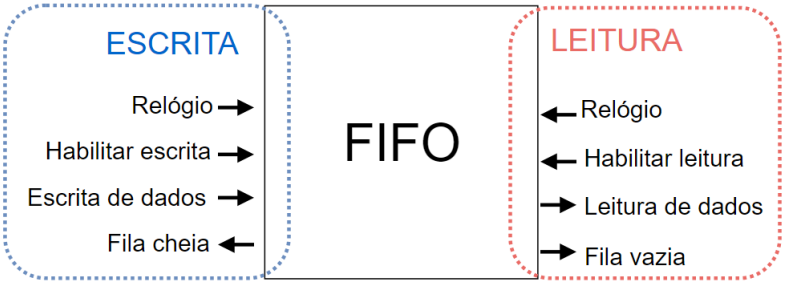
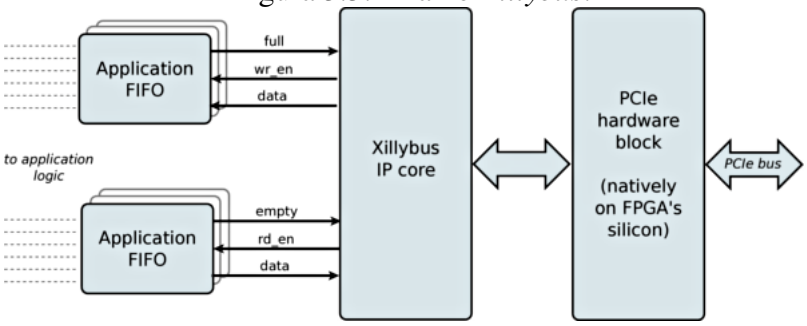


Figura 3.5: Fila no *Xillybus*.



Fonte: (XILLYBUS, 2019a).

trabalhar com rajadas de dados. Ou seja, em termos práticos o usuário, ao realizar leituras, pode analisar o sinal de *fila quase vazia* e, caso esta *flag* não esteja ligada, realizar várias leituras em sequência. Isso evita a necessidade de a cada processo de leitura, ter que se preocupar com o sinal de *fila vazia* e correr o risco de ler dados inconsistentes. O mesmo exemplo serve para o sinal de *Fila quase cheia* pois ele permite ao usuário escrever dados em rajada até que receba o sinal de que a *FIFO* está quase cheia. A não utilização destes sinais (ou seja, realizar a leitura e escrita somente após garantir que as *flags* vazia e cheia não estão ligadas) faz com que a vazão da *FIFO* seja de, no máximo, 50% do relógio.

Para o *Xillybus*, plataforma que será apresentada posteriormente, a *FIFO* representa uma peça crucial na solução criada, pois utilizam-se filas de entrada e saída para trocas de mensagens entre a *PCI Express* e a *FPGA*. Neste caso, a *FIFO* é apresentada a Figura 3.5.

Portanto, o *Xillybus* abstrai muito a comunicação entre as duas pontas, ou seja, facilita o meio de transmissão para o usuário. É possível ver na Figura 3.5 uma representação em alto nível do funcionamento das duas *FIFOs* que compõem o *Framework*. Nesse caso os sinais que o usuário interage são os mesmos já apresentados na explicação do funcionamento da *FIFO*.

3.2 Plataformas e ferramentas de desenvolvimento

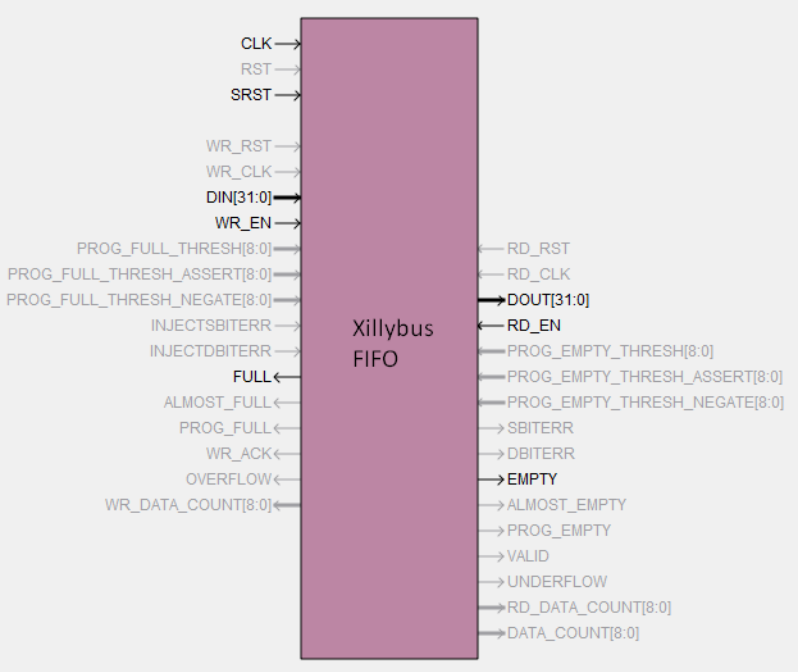
Nesta seção serão apresentadas as principais plataformas e ferramentas que propiciaram o desenvolvimento da solução proposta nesta monografia.

3.2.1 Xillybus

Segundo a própria documentação em (XILLYBUS, 2019b), o *Xillybus* é uma solução baseada em *DMA* criada com o objetivo de tornar fácil e intuitiva a comunicação de um computador de propósito geral, chamado de *host*, com uma placa *FPGA*. Essa interface de comunicação foi criada para funcionar em sistemas operacionais *Linux* e *Windows*. Isso propicia uma adesão maior de usuários na plataforma, já que ela funciona para os dois sistemas mais utilizados no mundo, por desenvolvedores. Mais especificamente, o *Xillybus* é um IP Core (*IP* é o acrônimo da sigla *Intelectual property*). IPs são muito utilizados no ramo de desenvolvimento de *hardware*, pois representam um bloco lógico que será acoplado em uma solução, com o objetivo de realizar uma função específica em *hardware*. Nessa área existem empresas especializadas em desenvolver *IP Cores* para fins diversos e portanto vender para usuários desenvolvedores. Portanto, o *Xillybus* é um *IP Core* utilizado para transmissão de dados, normalmente para comunicação *PCI Express*, mas segundo últimas documentações em (XILLYBUS, 2019b), também está sendo testado para transmissão via *USB 3.0*. Para realizar a conexão com a placa *FPGA* foram desenvolvidos diversos *drivers* e adaptações nas plataformas *ISE* e *Vivado* (utilizadas comumente para desenvolvimento de *hardware*).

O funcionamento do *IP* se baseia no uso de filas *FIFO*, apresentadas anteriormente neste trabalho, para transmissão de mensagens entre o *host* e a *FPGA*. Portanto, o usuário necessita somente realizar o carregamento do *IP* para a plataforma de desenvolvimento, escolher o *hardware* de sua preferência, e começar a utilizar o *Xillybus*. Na Figura 3.6 é possível ver a instanciação de uma fila *FIFO* de 32 bits de largura por 512 bits de comprimento. A Figura mostra alguns pontos principais da conFiguração do *IP Core* utilizado pela solução do *Xillybus*, para realizar a transmissão de dados. É importante ressaltar que os sinais em coloração mais escura representam as *flags* que o desenvolvedor pode utilizar. Os sinais com cor clara são abstratos ao usuário, a não ser que sejam realizadas modificações internas no *IP*. A Figura 3.6 foi retirada da plataforma de desenvolvimento de *hardware ISE* versão 14.7.

Figura 3.6: Instância do *Xillybus* na plataforma *ISE*.



As principais características do *Xillybus*, segundo (XILLYBUS, 2019b) são:

- Suportar as placas: Virtex-5T, Spartan-6T, Virtex-6T, all Series-7, Ultrascale e Ultrascale+, Zynq-7000 / Ultrascale+
- Realizar transporte de dados em *PCI Express*.
- Prover uma largura de banda bidirecional simultânea de até 6.6 giga bytes por segundo.

3.2.2 NETFPGA

A placa utilizada neste projeto foi a *NETFPGA-1G-CML*. Essa é desenvolvida e produzida pela empresa *Digilent*, para ser um dispositivo de baixo custo, mas com grande poder de processamento. As vantagens principais que esta placa possui são suas quatro portas para comunicação *Ethernet*, assim como entrada *PCI Express*. A entrada *PCIe* embarcada neste dispositivo é de segunda geração, e possui a largura do *link* de comunicação quatro vezes maior que placas *FPGAs* mais comuns, o que propicia, em tese, uma velocidade de transmissão de sinal de no mínimo 5 *Gb/s*, segundo (XILLINX, 2014).

Como esta placa *FPGA* possui quatro adaptadores de rede, ela é muito utilizada para diversos fins em telecomunicações como roteadores e *firewalls*. Outro ponto inte-

ressante é que ela foi desenvolvida para suportar uma arquitetura para interfaces de rede, criada na universidade de Stanford, chamada *Stanford NetFPGA architecture*. Esse grupo de desenvolvedores é responsável por diversas implementações e artigos, utilizando a *NETFPGA-1G-CML* focada para aplicações em redes de computadores. Abaixo, estão colocadas algumas das principais especificações da placa, segundo a documentação do fabricante, (DIGILENT, 2018):

- Xilinx Kintex-7 XC7K325T-1FFG676 FPGA
- Oscilador "Low-jitter"de 200 MHz
- Quatro adaptadores Ethernet
- PCI Express Gen 2 X4
- Memória RAM estática (450 MHz) X16 4.5 MB
- Memória RAM dinâmica DDR3 (800 MHz) X8 512 MB
- Entrada para cartão SD
- Entrada USB
- *Clock* de tempo real

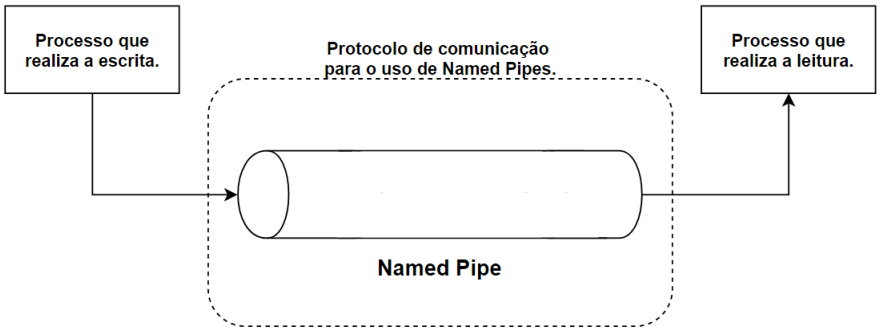
3.2.3 MetaTrader 5

A plataforma gratuita que originou a ideia da solução se chama *Meta Trader 5* e é um software desenvolvido pela empresa *MetaQuotes Software Corporation*. Seu objetivo é propiciar um ambiente de desenvolvimento para *software* de análise financeira, que permita ao usuário desenvolver com facilidade programas complexos para *trading* com diversos tipos de ativos, desde *bitcoin*, ações de empresas, sacas de café, entre muitos outros. Como muitos investidores não são familiarizados com linguagens de programação, a empresa baseou a ferramenta em uma linguagem criada especificamente para esse fim, muito similar ao *C++* chamada de *MQL5*. Essa linguagem abstrai alguns pontos mais complexos do *C++*, principalmente como a utilização de ponteiros.

3.3 Ferramentas e modelos de comunicação de dados

Transmissão de dados é um ponto crucial desde trabalho, pois a latência de comunicação representa uma enorme barreira a ser transposta, com o intuito de fazer com

Figura 3.7: Representação em alto nível de um *pipe*.



que a solução seja realmente viável. Portanto, nesta seção serão apresentados os modelos utilizados no projeto para realizar a comunicação de dados.

3.3.1 Named Pipes

Named pipes são uma ferramenta muito útil para comunicação entre processos. Inventado nos anos 2000, têm como objetivo propiciar um modelo de comunicação em alto nível de fácil implementação e com performance eficiente. Se comportam como filas *FIFO*, que recebem dados e os guardam até que haja uma requisição, para que sejam consumidos por outro processo que faz parte da transação. Os *named pipes* podem ser de uma via ou duas, dependendo da necessidade da transferência. A forma como os *pipes* se comportam é sempre seguindo o modelo de cliente/servidor, ou seja, são instanciados os chamados *pipe servers*, os quais se comunicarão com os *pipe clients*.

Segundo (KHAMBATTI, 2001), em baixo nível, os *pipes* na verdade liberam uma área da memória do processo para que esta seja acessada por outro. Também é importante ressaltar que esse modelo de comunicação é *Connection-oriented*, ou seja, é orientado a conexão. Portanto, para que a comunicação ocorra, ela deve ser transmitida por um "circuito virtual", sempre mantendo a confiabilidade e uma transmissão de dados sequencial. Na Figura 3.7 é possível observar uma representação em alto nível do funcionamento dos *Named Pipes*.

3.3.1.1 Criando Named pipes

A instanciação e utilização de *Named Pipes* é bem simples. Baseado na documentação da *Microsoft* em (MICROSOFT, 2018), para criar um *pipe*, somente é necessário realizar a chamada de um comando do sistema, no caso *CreateNamedPipe* com o cami-

nho:

```
\\NomeDoServer\pipe\NomeDoPipeServidor

\\.\pipe\NomeDoPipeCliente
```

Esses comandos criarão *Named pipes* que realizaram a função de servidor e cliente respectivamente. Após a criação das instâncias, os pipes se comportam como arquivos. Portanto para enviar mensagens é utilizando comandos comuns como *read* e *write* oriundos do próprio sistema operacional. É importante ressaltar que os *Named pipes* estão presentes tanto no sistema operacional *Windows*, como no *Linux*.

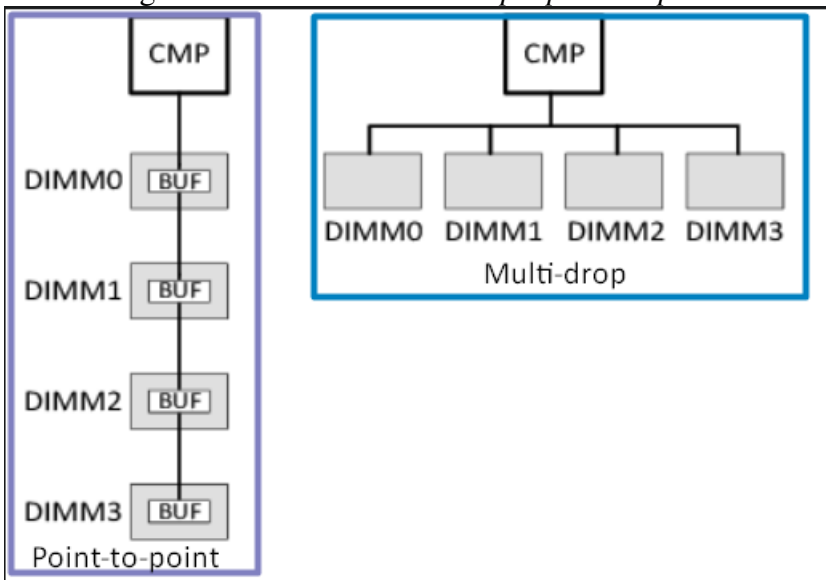
3.3.2 PCI Express

A *PCI Express*, também conhecida como *PCIe*, surgiu em 2010 com o intuito de realizar um grande avanço para aumentar consideravelmente a velocidade na comunicação entre periféricos. Sua antecessora, a *PCI* desenvolvida em 1990, não comportava mais os avanços da tecnologia e começou a ser necessária uma maior velocidade em meios de transmissão de dados. Segundo (INTEL, 2003) a *PCIe* tornou-se mais escalável, pois diferentemente da *PCI*, trata escalabilidade de duas formas. A primeira com um maior número de sinais lógicos para troca de mensagens e a segunda com o aumento da velocidade do *clock*, ou frequência da placa. Uma mudança significativa relacionada a *PCIe* é a diferença na maneira de troca de mensagens via periféricos. A sua antiga versão utilizava de uma comunicação paralela e *multi-drop*, este modelo que foi substituído por serial e *point-to-point*.

Para uma melhor visualização dos modelos *multi-drop* e *point-to-point*, veja Figura 3.8.

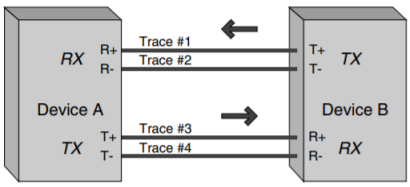
É possível notar a diferença entre os dois modos na Figura 3.8. No *multi-drop* todos os componentes são conectados na mesma via, assim recebendo a mesma mensagem a cada envio. Em *point-to-point* todos os pontos estão conectados por meio de *buffers*. Na representação da Figura 3.8, estão propostos modelos de conexão para memórias *dual in-line*, (*DIMM*), que foram utilizadas somente para ilustração da arquitetura. Portanto, a comunicação entre dois dispositivos via *PCIe* pode ser ilustrada pela seguinte representação na Figura 3.9.

Figura 3.8: Modelos *multi-drop* e *point-to-point*.



Fonte: (INTEL, 2003).

Figura 3.9: Representação da comunicação via PCIe.



Fonte: (INTEL, 2003).

4 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados alguns trabalhos relacionados que foram importantes para a concepção desde projeto. Esses artigos auxiliaram no projeto da solução proposta, assim como elevaram a velocidade de desenvolvimento, devido as informações relevantes que estão contidas neles.

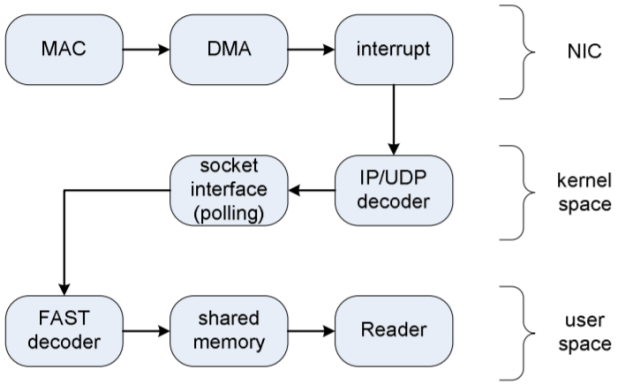
4.1 *High-Frequency Trading Acceleration using FPGAs*

O artigo (LEBER; GEIB; LITZ, 2011) apresenta uma solução que trabalha principalmente no pacote de mensagens que chegam da bolsa de valores com informações sobre os ativos. Portanto, neste artigo não são trabalhados modelos para tomada de decisão, mas como processar os pacotes de dados o mais rápido possível, para que a latência e o *parsing* dos dados sejam realizados no menor tempo. Diferentemente do trabalho proposto nesta monografia, nesse artigo a camada de comunicação de dados é realizada via interface *Ethernet* seguindo o protocolo de comunicação *UDP*. Essa decisão é coerente, pois como o objetivo desse trabalho está em aumentar a velocidade de recepção dos dados, utilizar a comunicação *UDP*, que não possui controle de fluxo, é muito interessante. Isso se deve ao fato que as placas *FPGA* trabalham eficientemente com um alto fluxo de dados e, assim, proveem uma grande vantagem em velocidade se forem comparadas a computadores de propósito geral. Outro ponto interessante desse trabalho é a criação de um compilador para o protocolo chamado *FAST*. Esse protocolo é utilizado para envio de dados de ativos por diversas bolsas de valores em todo o mundo. Portanto, segundo (LEBER; GEIB; LITZ, 2011), esse representa uma grande barreira para diminuição da latência de dados, pois como é um protocolo muito específico, sempre existe a necessidade de realizar a conversão dos pacotes por ele enviados. Ao decorrer do artigo, foram apresentadas três soluções para tentar reduzir a latência em aplicações de *HFT*.

A primeira foi chamada de *UDP Offloading*. É explicado no artigo que um dos fatores causadores de um aumento grande no tempo de envio de uma mensagem é a decodificação dos pacotes de rede no *kernel* do sistema operacional. É possível ver na Figura 4.1 o fluxograma do caminho percorrido normalmente por estas mensagens. A ideia é retirar a interface do *kernel* a decodificação dessas mensagens e realizar este processamento em *FPGA*.

A segunda solução foi a implementação em *hardware* de um algoritmo de pro-

Figura 4.1: Sistema de recepção de mensagens *UDP*.



Fonte: (LEBER; GEIB; LITZ, 2011).

cessamento dos pacotes recebidos no protocolo *FAST*, para atuar em conjunto com a a primeira solução. Com o objetivo de realizar o recebimento e decodificação dos pacotes recebidos em *hardware*.

A terceira e última implementação teve como objetivo criar uma interface de acesso direto à memória. Assim, após realizar o processamento de dados, o *hardware* possui a capacidade de escrever em uma memória compartilhada.

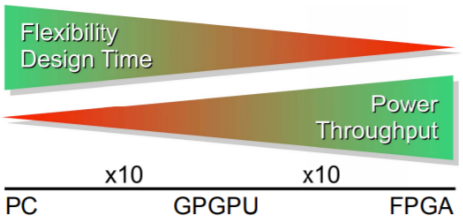
Todas estas soluções em conjunto proveram uma redução de quatro vezes no valor da latência inicial, representando um grande sucesso na abordagem para ser utilizado em soluções de *HFT*.

4.2 *Hardware Accelerators for Financial Mathematics - Methodology, Results and Benchmarking*

No artigo (SCHRYVER; MARXEN; SCHMIDT, 2011) foi realizado um estudo sobre qual a melhor abordagem, em *hardware*, para se trabalhar com dados matemáticos financeiros. Foram abordados computadores de propósito geral, implementação via *GPUs* e placas *FPGA*. A Figura 4.2 correlaciona as vantagens e desvantagens de utilizar cada um dos modelos de desenvolvimento. Nela é possível observar que um computador de propósito geral (*PC*) possui a maior flexibilidade de design, porém menor poder de vazão de dados. Totalmente oposto ao *FPGA* que possui uma alta vazão mas baixa flexibilidade no tempo de desenvolvimento.

Para realizar a comparação, o artigo coloca que seria importante desenvolver uma solução única para as três plataformas (*PC*, *GPU* e *FPGA*). Neste caso, contemplaria o cálculo de preço de um ativo no mercado utilizando um modelo matemático específico,

Figura 4.2: Comparação das diferentes abordagens para análise de dados financeiros.



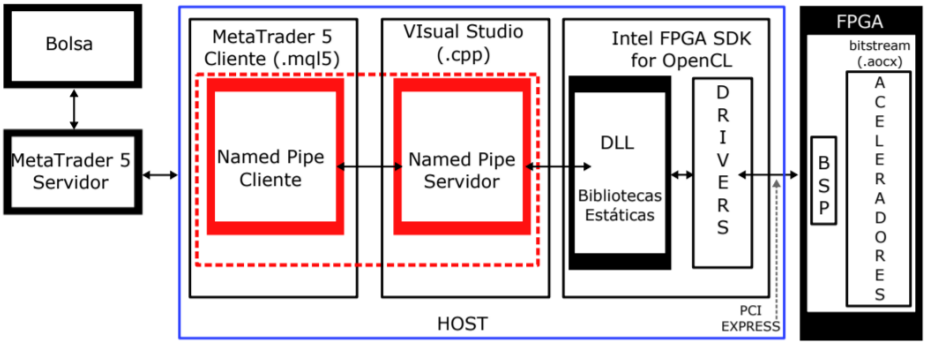
Fonte: (SCHRYVER; MARXEN; SCHMIDT, 2011).

chamado *Henson model*. Este algoritmo é extremamente custoso de se computar, portanto a escolha deste reflete somente em um meio para que, após realizados os cálculos, sejam claros os resultados de comparação entre as plataformas. Como a comparação de um mesmo método em diferentes ferramentas não é uma tarefa trivial, foi desenvolvido também um método de teste para realizar as comparações. Essa abordagem realiza uma combinação entre o problema e o modelo, pois mesmo que as implementações sejam diferentes, o resultado deve ser o mesmo em todos os casos. O artigo coloca que a implementação do algoritmo para as três plataformas está ainda em desenvolvimento.

4.3 NetFPGA: Processamento de Pacotes em Hardware

Em (GOULART et al., 2016), é apresentada uma solução para processamento de pacotes utilizando a placa da empresa *NETFPGA*, *NETFPGA 1G - CML*. Essa placa é a mesma utilizada neste trabalho. Esse artigo demonstrou-se muito útil para o desenvolvimento da solução aqui proposta, pois ele demostra passo-a-passo como implementar um projeto utilizando placa *NETFPGA 1G - CML*. No artigo, são utilizados uma série de ferramentas criadas por desenvolvedores da *NETFPGA* em código aberto, para realizar leitura e escrita de pacotes. Esse também apresenta uma explicação detalhada do funcionamento da placa *NETFPGA*, assim como seus componentes de *software* e de *hardware*, tendo como objetivo principal a implementação de um *firewall*. No início do projeto proposto nesta monografia, houve uma ponderação se seria interessante utilizar as plataformas desenvolvidas pela *NETFPGA* para realizar a comunicação via *PCI Express* ou em protocolo *Ethernet*. A decisão final foi baseada na facilidade em que a plataforma do *Xillybus* apresenta para os seus usuários desenvolvedores. A *NETFPGA*, em contrapartida, provê uma interface de rede completa, com múltiplas ferramentas e implementações; entretanto, é mais complexa. O *Xillybus* entrega uma solução de simples acesso e uso.

Figura 4.3: Visão geral do projeto proposto.



Fonte: (COSTA; ROSA; BONATO, 2018).

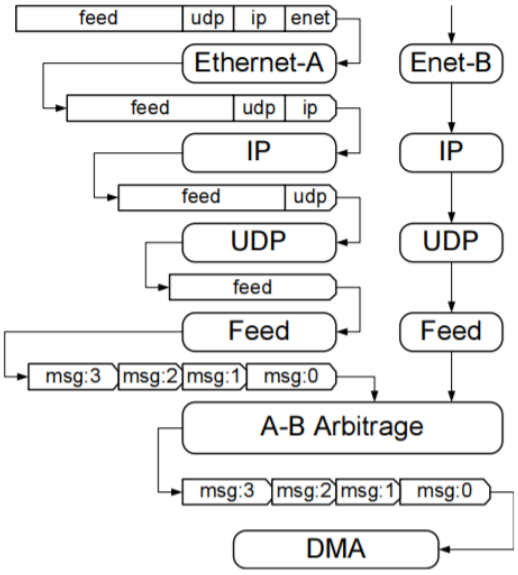
4.4 Integrando o MetaTrader5 com Aceleradores FPGA via OpenCL Named Pipes

O artigo (COSTA; ROSA; BONATO, 2018), foi o trabalho encontrado que possuiu mais semelhança e afinidade com a solução desta monografia. A proposta desse é justamente realizar a junção da plataforma do *Metatrader 5* com um algoritmo implementado em *FPGA*. Infelizmente nesse artigo consta somente uma solução parcial e, portanto, não possui resultados. De qualquer forma, esse ofereceu uma base muito interessante para o desenvolvimento da solução proposta neste trabalho. O modelo proposto pelo projeto do artigo pode ser visualizado na Figura 4.3.

No projeto apresentado no artigo, a comunicação com a *FPGA* é realizada através de um *driver* que se comunica via uma *DLL* interna com o programa desenvolvido em *C*. Um ponto importante é que a solução em (COSTA; ROSA; BONATO, 2018) utiliza *OpenCL*, que é uma ferramenta utilizada para desenvolvimentos de *hardware* heterogêneos como *FPGA* e *GPU*. Neste caso, também foi utilizada como solução de integração uma plataforma de desenvolvimento produzida pela empresa Intel.

Os resultados parciais obtidos referentes a taxa de transmissão, na etapa de comunicação entre a plataforma *Metatrader 5* e um programa em *C* via *Named Pipes*, foram muito similares com os desta monografia: cerca de 1500 *Mega bytes* por segundo, para a transferência de um dado de 1024 *Mega bytes*. Por fim, os autores de (COSTA; ROSA; BONATO, 2018) colocam que irão implementar em *FPGA* o algoritmo *Monte Carlo Black-Scholes Asian Options Pricing* que, segundo (COSTA; ROSA; BONATO, 2018), é um algoritmo que leva dias para completar cálculos em simulação.

Figura 4.4: Filtro progressivo de pacotes.



Fonte: (SUBRAMONI, 2010).

4.5 *FPGA accelerated low-latency market data feed processing*

O artigo (SUBRAMONI, 2010), apresenta uma solução muito similar ao (LEBER; GEIB; LITZ, 2011). A proposta é utilizar a *FPGA* para realizar o *parsing* da informação, descompressão dos dados e principalmente a filtragem dos pacotes. Esse último ponto apresentado é um fator que apresenta uma grande diferença se comparado com o artigo em (LEBER; GEIB; LITZ, 2011). Por fim, o artigo coloca que, após a realização da filtragem, os dados são colocados na memória de rápido acesso. O *parsing* foi desenvolvido com o objetivo de decodificar a informação oriunda do protocolo de comunicação *FAST*. A implementação realizada para a filtragem de pacotes é muito interessante: são utilizados dois pontos de transmissão de dados, chamados *streams*. Como a comunicação escolhida, *UDP*, é conhecida pela alta taxa de transferência, mas também pela grande perda de pacotes na transmissão, esses dois *streams* servem para capturar, filtrar e remontar pacotes incompletos. A Figura 4.4, mostra uma visualização em alto nível de como os *streams* atuam. A parte mais importante é a colocada como *A-B Arbitrage*, pois é esta peça que realiza todo o processo de remontagem de pacotes incompletos. Isso só é possível devido ao fato, conforme em (SUBRAMONI, 2010), que existe uma grande chance de, caso um pacote seja perdido ou incompleto em um *stream*, o mesmo esteja intacto em outra via.

Ao final do artigo é relatado um resultado muito satisfatório em termos de processamento de dados, o qual, o sistema proposto foi capaz de processar 3.5 milhões de

mensagens por segundo.

5 DESENVOLVIMENTO

Neste capítulo serão apresentados todos os passos para o desenvolvimento da solução, assim como explicações e fluxos de seu comportamento.

5.1 Evolução da solução

Os desenvolvimentos iniciais da solução proposta nesta monografia foram realizados na linguagem *MQL5*. O projeto possui extrema ligação com a bolsa de valores, portanto, foi necessária uma primeira ambientação com a plataforma *MetaTrader 5*. Primeiramente foram desenvolvidas diferentes versões e variações de código em *MQL5*, utilizando diversos indicadores de posições de compra e venda de ativos no mercado. Nesses testes foram utilizados indicadores como:

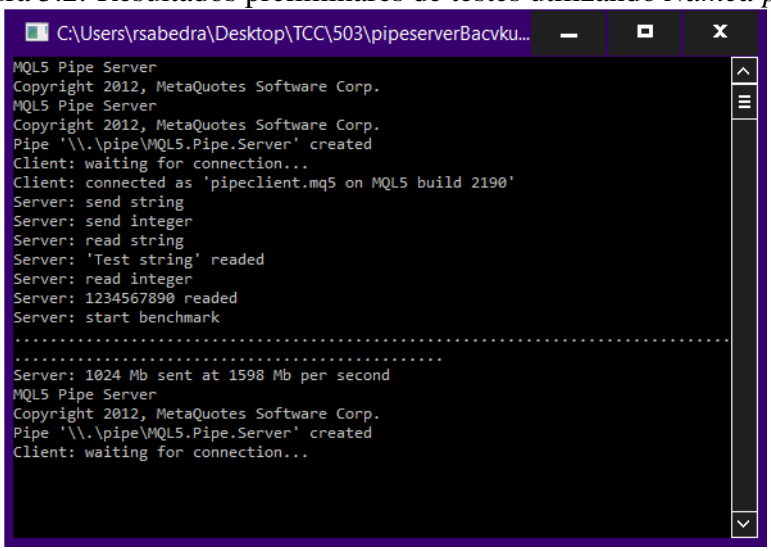
- MACD
- Boolinger Bands
- RSI

Para exemplificação dos testes realizados, é possível ver na Figura 5.1 um dos resultados obtidos, em dólar, utilizando o indicador *MACD* com diferentes parâmetros. Os principais são o ganho (*gain*), que é referente a quando o *software* lucra com o fechamento de uma ordem, e a perda (*loss*) que simboliza quanto o algoritmo aceita perder em cada ordem, seja de compra ou venda. A variável *Profit*, por sua vez, representa quanto foi a renda do indicador para uma determinada janela de tempo.

Figura 5.1: Resultados de testes de desempenho para recursos financeiros.

Profit	gain	loss
1582.40	200	4000
1382.40	200	5000
1286.00	100	5000
1286.00	100	4000
1286.00	100	3000
837.60	300	5000
665.00	100	1000
462.60	100	2000
431.00	300	4000
390.20	200	1000
240.60	300	1000
17.00	200	3000
-571.60	300	2000
-651.40	300	3000
-856.40	200	2000

Figura 5.2: Resultados preliminares de testes utilizando *Named pipes*.



A partir da ampla ambientação com a linguagem *MQL5* e a escolha de um indicador das *bandas de bollinger* para implementação em *hardware*, foram realizados estudos de como obter os valores utilizados na plataforma *MetaTrader 5* e exportá-los para outras aplicações. O melhor modelo encontrado foi utilizando *Named-pipes* pois eles possibilitaram uma comunicação de até 1.5Gb/s. Como os valores enviados para cálculo do indicador possuem menos de 20 *bytes*, a vazão deste meio de comunicação supre com eficácia as necessidades do projeto para comunicação entre processos. A Figura 5.2 evidencia esses primeiros resultados obtidos.

A partir deste ponto, foram escolhidos os dados que a plataforma *Meta Trader 5* deve adquirir para realizar abertura de posições no mercado de ações. Para tanto, as informações necessárias adquiridas foram:

- Nome do ativo.
- Data/hora.
- Preço de compra.
- Preço de venda.
- Valor da última ordem realizada, também chamada "preço de fechamento".

Relembrando as Figuras 1.2 e 1.1 apresentadas no início desta monografia, é importante ressaltar: mesmo que o software *Meta Trader 5* necessite de todos os itens anteriores para trabalhar com os ativos, na comunicação com a versão em C/C++ e descrita em *hardware*, somente serão transmitidos o preço de fechamento. Esse representa o único dado necessário para a realização dos cálculos referentes ao algoritmo das "Bandas de Bollinger".

5.1.1 Algoritmo em C/C++

Para fins de comparação de performance, foi desenvolvida uma versão do indicador das *Bandas de Bollinger* em *C/C++*.

Essa versão possui uma arquitetura e estruturas próprias diferentes do algoritmo implementado em *mql5*, analisado posteriormente neste trabalho de graduação. A implementação em *C/C++* recebe valores da ferramenta *MetaTrader 5*, calcula o momento certo de abrir uma posição de compra ou venda de um ativo e envia um valor representado como um inteiro de 32 bits como retorno para a plataforma *MetaTrader 5* realizar a abertura da ordem a ser executada. Esse inteiro pode possuir três valores:

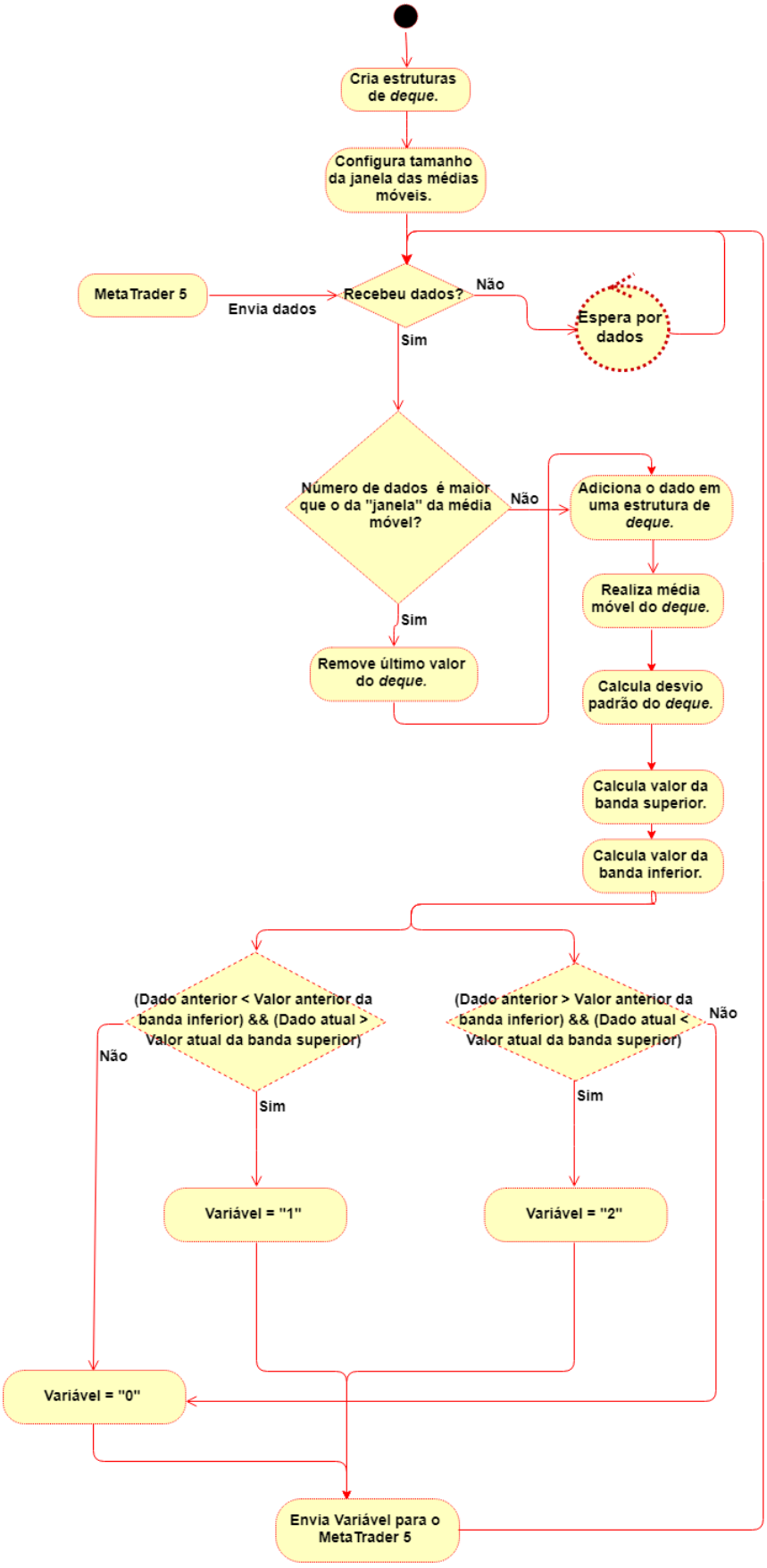
- Valor "0": representa que o programa não deve abrir nenhuma posição.
- Valor "1": indica a abertura de uma posição de compra.
- Valor "2": indica a abertura de uma posição de venda.

Mais detalhadamente, esse algoritmo em C++ possui classes para trabalhar com a mensagem recebida ou enviada via *Named pipes*. Para instanciar este objeto somente é necessário chamar a função *CPipeManager*: `CPipeManager nomeDoPipe;` Nesse algoritmo foi utilizado uma estrutura muito importante para a realização das médias móveis, crucial para o funcionamento do algoritmo das "Bandas de bollinger"chamada de *Deque*. Ela representa uma fila que pode ser iterada de ambos os lados. Esse tipo de contêiner é muito interessante para situações em que são necessários adicionar ou remover objetos em ambas as saídas da fila. No caso, o *deque* representou o principal ponto do cálculo das médias móveis, pois após obter o número de valores especificados na "janela"da média é realizado a remoção do último valor da fila e adição da nova variável recebida. Assim, realizando o comportamento esperado das médias. Para iterar valores nesta estrutura, são utilizadas as funções:

```
std::deque<double> nomeDoDeque;
nomeDoDeque.push_front(valor_1);
nomeDoDeque.pop_back();
```

O fluxo de comportamento do algoritmo pode ser visualizado na Figura 5.3. Nele está contido todo o comportamento do programa em alto nível.

Figura 5.3: Fluxo do *software implementado*.



5.1.2 Implementação em *hardware*

A partir da evolução dos testes com a plataforma *MetaTrader 5*, as validações na comunicação com o programa em C/C++ e o desenvolvimento do indicador das *Bandas de Bollinger*, deu-se início ao desenvolvimento e implementação da versão do indicador em *hardware*. O primeiro passo foi realizar uma versão em alto nível do projeto, antes de começar com a execução do mesmo. Assim, foi possível observar os pontos de maior dificuldade de desenvolvimento, também possuindo uma rápida visão geral do projeto e de sua escalabilidade. A Figura 5.4 representa a primeira versão realizada do diagrama de alto nível da solução em *FPGA*.

O *hardware* proposto computa valores de 32 bits em ponto fixo, sendo 20 bits de precisão. A solução se baseia na implementação de duas estruturas representativas de filas *FIFO* geradas via junção de vários registradores. Com a execução da solução, essa receberá dados referentes aos valores de preço dos ativos. Após realizar o registro desses valores em registradores, os números são somados até uma variável X de dados e ao final da soma realiza-se uma divisão pela quantidade de dados recebidos (X). Representa-se, assim, uma média dos valores registrados. O *hardware* foi projetado para atuar somente em um ciclo, ou seja é monociclo. Na Figura 5.4, para uma melhor ilustração, o *hardware* representado realiza uma média de quatro valores, diferentemente da versão implementada que inicialmente realizará uma média móvel de vinte valores. Outro ponto importante sobre o *hardware* é referente à aquisição de dados. Neste projeto, ao mesmo tempo que a *FPGA* recebe um valor da *CPU*, ele escreve o dado previamente calculado para o *PC*. Sendo assim, a entrada de saída de dados é simultânea. A Figura 5.5 mostra como foi projetado internamente o sistema de comunicação da *FPGA* com o *PC*. Nela é possível observar que foram geradas duas filas: uma para recepção de dados e outra para envio. Isso se conecta com os programas de envio e recepção gerados em *software*, um realiza o envio dos dados e outro recebe.

Figura 5.4: Representação da primeira versão de alto nível do *hardware* a ser implementado.

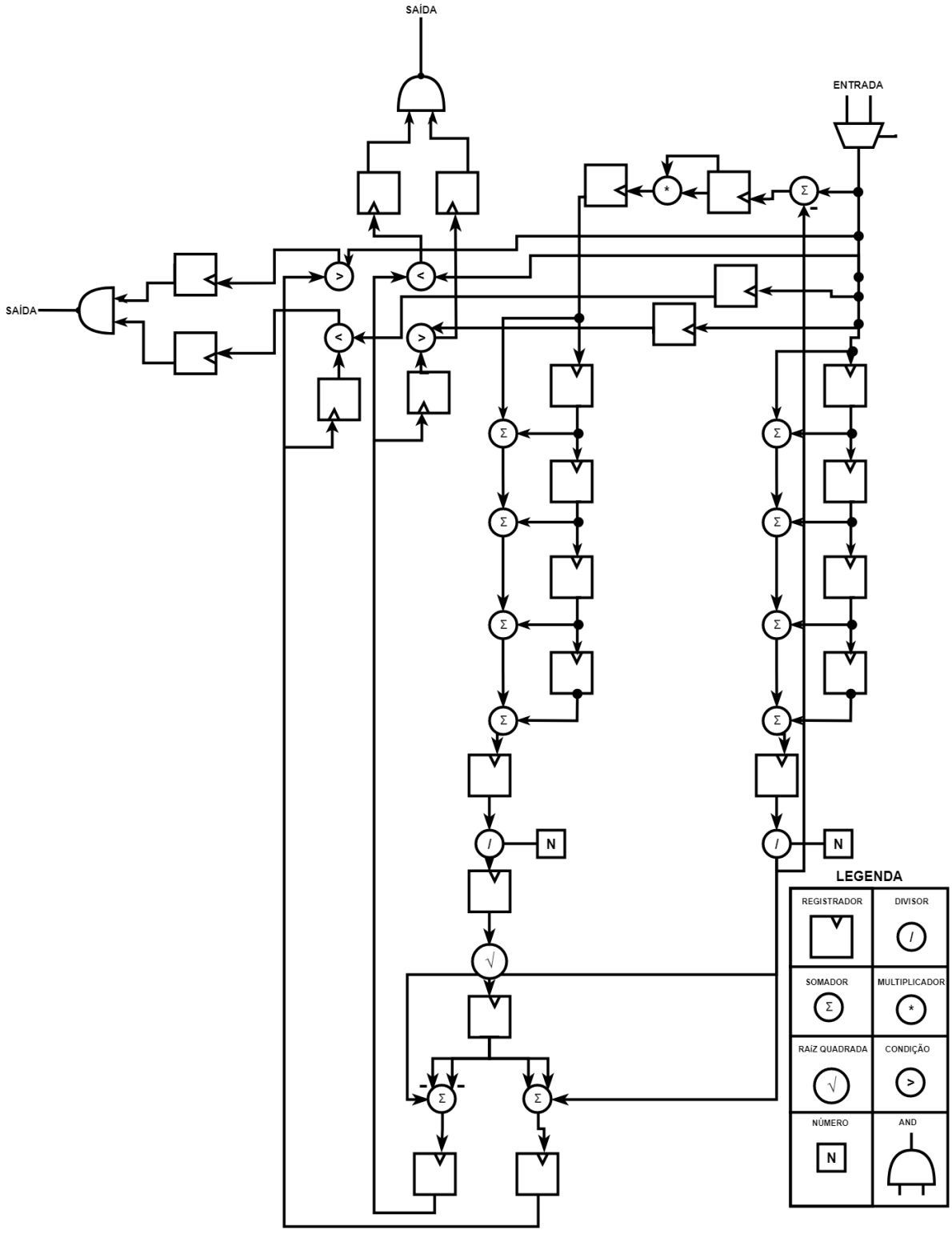
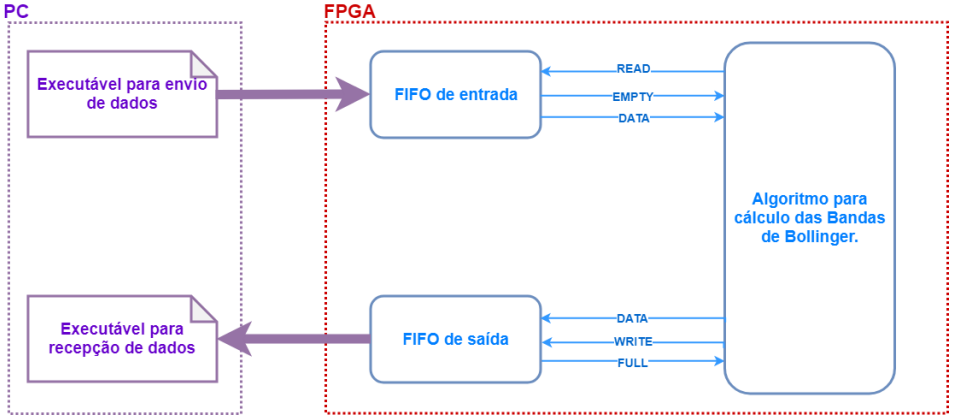


Figura 5.5: Representação em alto nível interna do *hardware*.



6 RESULTADOS

Neste capítulo serão apresentados e discutidos os resultados obtidos em termos de comparação de performance entre as soluções desenvolvidas.

6.1 Metodologia e análise de desempenho

Com o desenvolvimento finalizado de ambas as versões, deu-se início à obtenção de resultados e comparações entre as duas soluções. Vale ressaltar que, para o mercado financeiro, qualquer diminuição na velocidade de processamento possui um impacto extremamente positivo em possíveis ganhos com o algoritmo. Segundos dados da *Bovespa* em (B3, 2019), somente em *contratos de mini índice* são movimentados por dia mais de 170 bilhões de reais. Isso representa que, em um segundo, acontece uma movimentação média de 98 mil reais, ou seja, neste tipo de solução, qualquer lapso de tempo, por menor que seja, pode afetar na obtenção de lucros. Outro ponto importante para algoritmos financeiros é a quantidade de números que compõem as médias móveis. Esse valor é importante, pois conforme são aumentados os valores da média é ocasionada uma amortização cálculo das bandas. Ou seja, a tendência é que o algoritmo se torne menos volátil a variações repentinas de preços dos ativos do mercado. Portanto é interessante validar como ambas implementações geradas neste trabalho de graduação se comportam em termos de execução referente ao aumento da quantidade de números das médias, pois é uma prática muito útil e é utilizada diariamente em *HFT*.

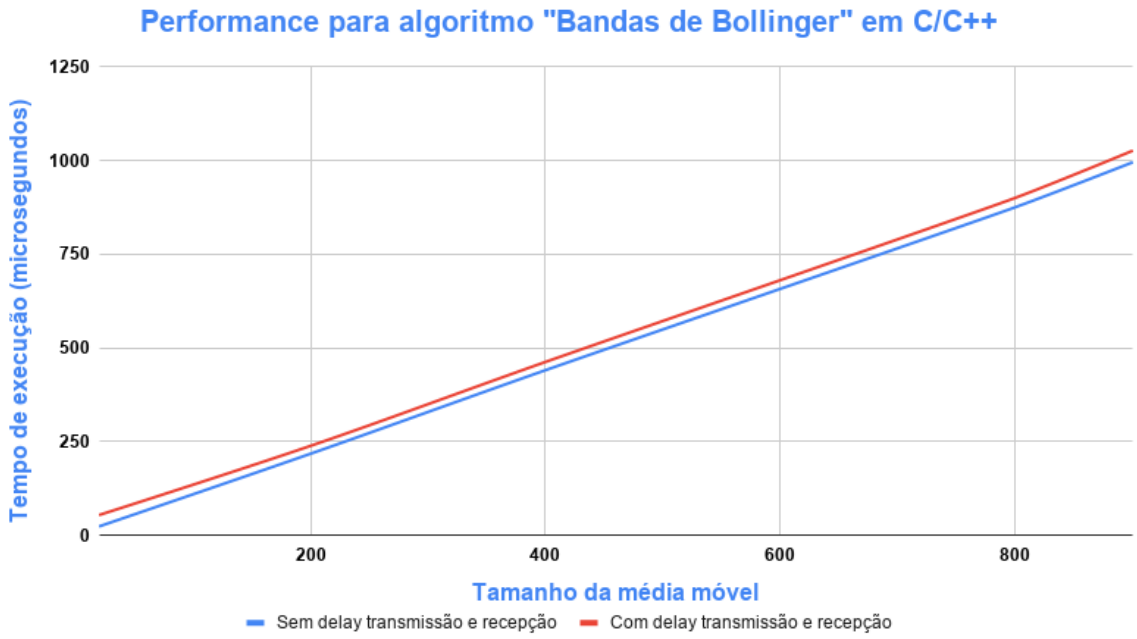
Os primeiros testes propostos possuem o objetivo de avaliar a performance para comparação entre a versão implementada em *C++* e a descrita em *hardware*. O ponto principal a ser analisado é o tempo para execução do algoritmo das *Bandas de Bollinger* em ambas as soluções.

6.1.1 Performance do algoritmo em C++

Os testes para a versão em *C++* foram realizados em um computador de propósito geral com as seguintes especificações:

- Sistema operacional: Windows 10 Home 64-bit (10.0, Build 18362).
- Processador: Intel(R) Core(TM) i7-5700HQ CPU @ 2.70GHz (8 CPUs).

Figura 6.1: Resultados obtidos para algoritmo em *C++*.



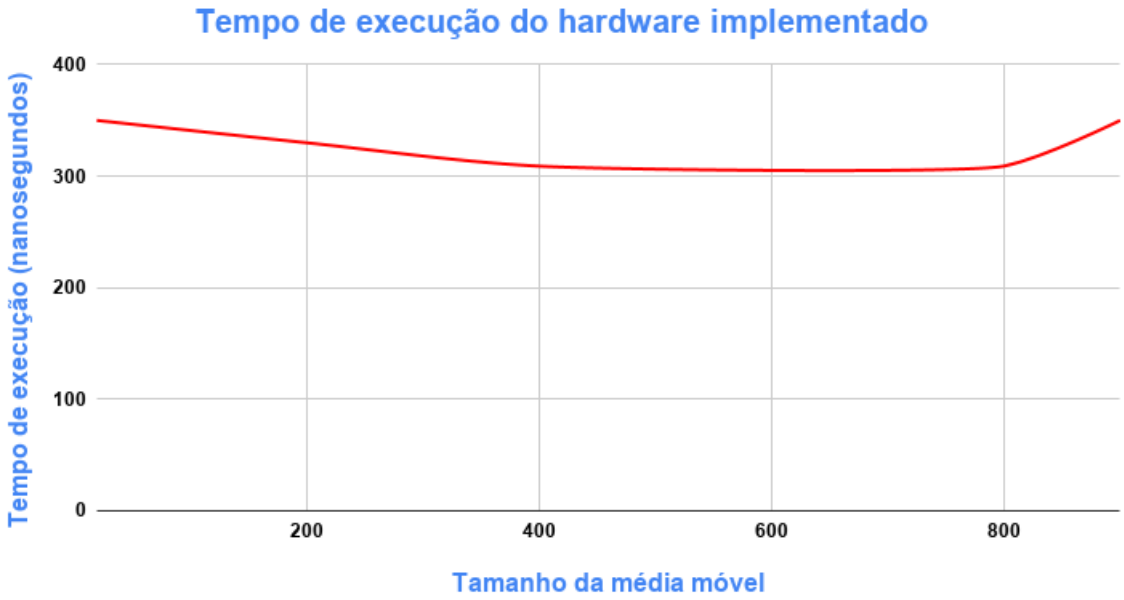
- Memória: 16384MB RAM.

Os primeiros resultados obtidos são baseados na velocidade de execução com e sem atrasos relacionados à recepção e transmissão de dados. Este resultado pode ser visualizado na Figura 6.1, na qual é possível perceber o tempo de execução de computador de propósito geral para realizar o cálculo proposto, e a magnitude o atraso no tempo de execução devido a *delays* de transmissão. Nesse caso o resultado obtido demonstra que o tempo adicional referente a transmissão permanece constante com o aumento do tamanho da média móvel.

6.1.2 Performance da implementação em *hardware*

Esta subseção apresenta os resultados obtidos relacionados a performance do projeto desenvolvido em *hardware*. A Figura 6.2 mostra um resultado muito satisfatório para o projeto. Nela consta os valores de tempo adquiridos com o aumento do tamanho da média móvel de 20 até 900 posições. Neste teste foram enviados valores para o cálculo do algoritmo somente via *PCI Express*, sem utilizar comunicação com a plataforma *metatrader* e, medido o tempo de resposta. Os resultados demonstram que mesmo aumentando o número de registradores para realizar o cálculo, o tempo de execução não escala linearmente, pelo contrário, se mantém praticamente constante. Isso acontece devido ao fato de

Figura 6.2: Resultados para analise de tempo de execução do *hardware*.

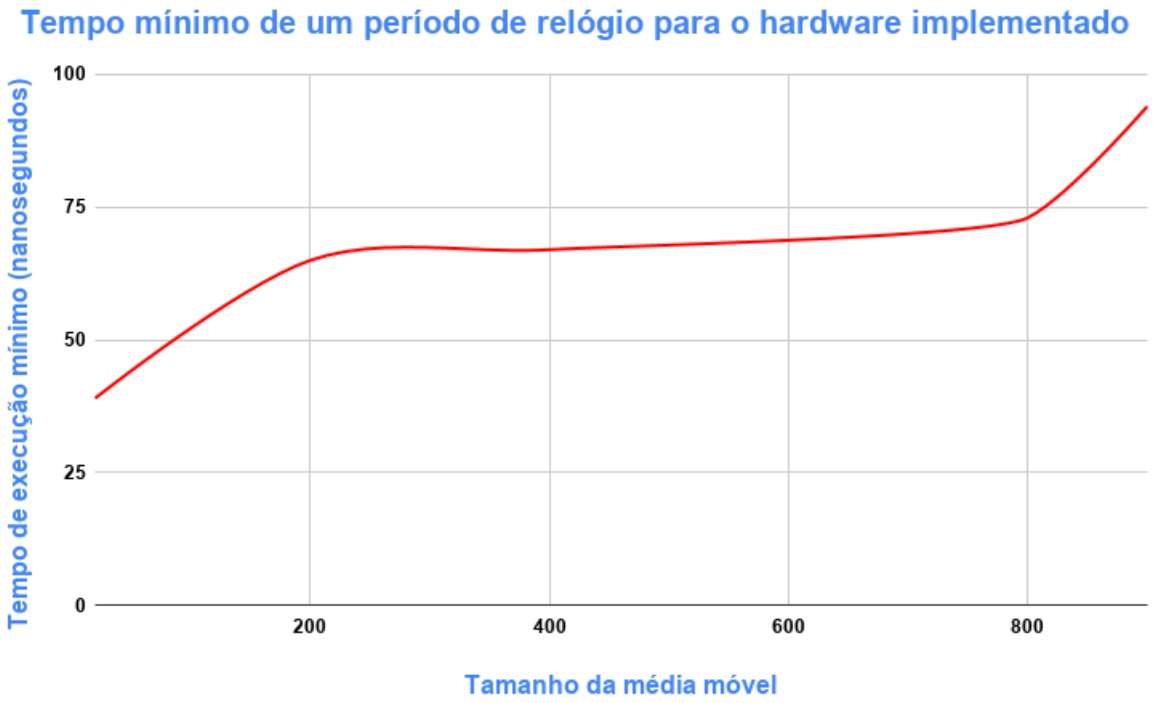


que o *hardware* foi desenvolvido utilizando lógica combinacional para que a realização dos cálculos ocorra em um período de relógio. Portanto, mesmo com o aumento do número de registradores e consequentemente a diminuição da frequência de operação, vide Figura 6.4, o tempo de execução é limitado pela velocidade de transmissão entre o sistema e a *FPGA* que se mantém por volta dos 350 nanossegundos.

Também mostra que a escala de tempo para se realizarem os cálculos é em nanossegundos, ou seja, muito menor que um computador de propósito geral. Para fins de comparação, com 900 valores para cálculo da média, segundo a Figura 6.1, o tempo de execução de um *PC* é próximo de 1 milisegundo. Para o mesmo cálculo em *hardware*, visualizado na Figura 6.2, o tempo necessário é de aproximadamente 350 nanossegundos. Portanto, isso demonstra que o *hardware* nesse caso é 2800 vezes mais rápido.

Os dados Figura 6.3, foram retirados do *software* de simulação *ISE 14.7*. Essas informações são calculadas quando a plataforma está gerando o arquivo de programação da *FPGA*. É interessante reparar o quanto os *delays* de transmissão afetam o tempo de cálculo do algoritmo em *hardware* se comparadas às Figuras 6.2 e 6.3. Segundo relatórios gerados pela a ferramenta de sintetização *ISE* sobre tempos de execução, o período de relógio ideal simulado para o *hardware* implementado gerar resultados é de no máximo 100 nanossegundos, se comparados todos os relatórios gerados para as versões com diferentes médias móveis. Mas em testes fora de simulação, os resultados de tempo de execução em 6.2 ficaram por volta dos 300 nanossegundos. Portanto, comparando os dados da Figura

Figura 6.3: Resultados para analise do período de relógio do *hardware*.



6.2 com o valor máximo do relatório gerado pela ferramenta (100 nanossegundos), é possível inferir que a diferença de ambos é ocasionada por atrasos de comunicação entre *PC* e *FPGA* na média de 200 nanossegundos.

Outro ponto interessante analisado foi a relação da frequência com a área utilizada na *FPGA* quando se aumentam o número dos registradores. É possível ver que ambas frequência e área são inversamente proporcionais. Antes de realizar os experimentos desta seção, era esperado que com a diminuição da frequência de *clock* do *hardware*, houvesse um grande aumento no tempo de execução do algoritmo implementado. Por mais que a Figura 6.3 demonstre a existência um aumento no tempo de execução, a Figura 6.2 mostra que este aumento não é significativo para realizar alterações temporais significativos na execução do *hardware*.

6.1.3 Comparação entre *hardware* e *software*

Por fim, foi realizada uma última e mais importante comparação. Esta relaciona o tempo de execução com atrasos de recepção e transmissão para a versão em *C++*, com a implementação em *hardware*. Esta comparação pode ser visualizada na Figura 6.5.

Figura 6.4: Resultados para analise de frequência e área do *hardware*.

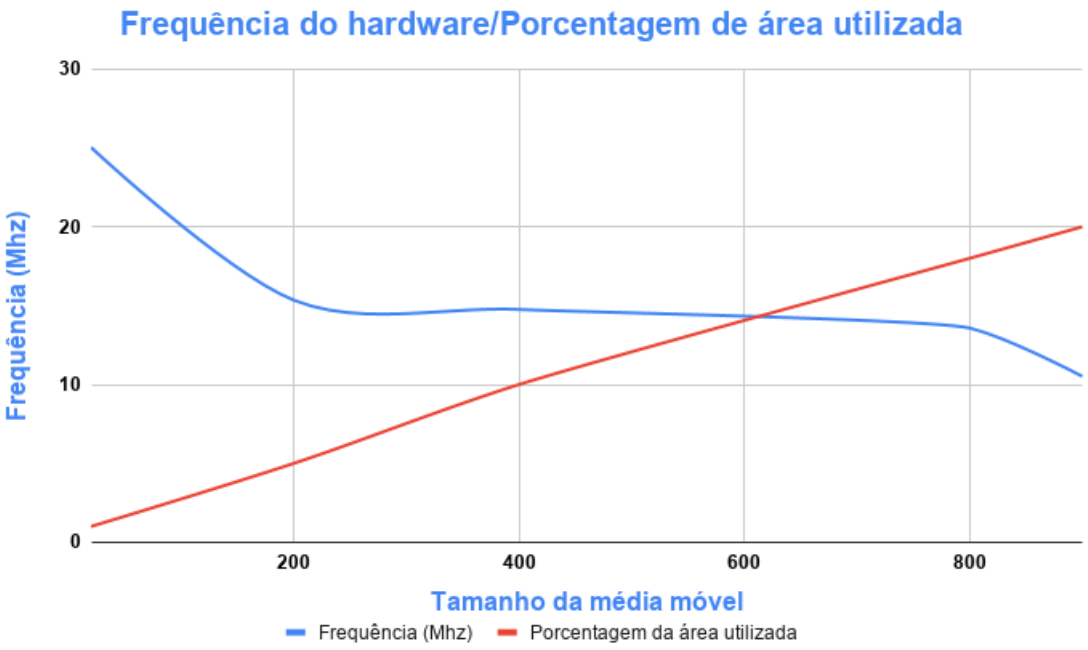


Figura 6.5: Comparação entre versões em *hardware* e *software*.

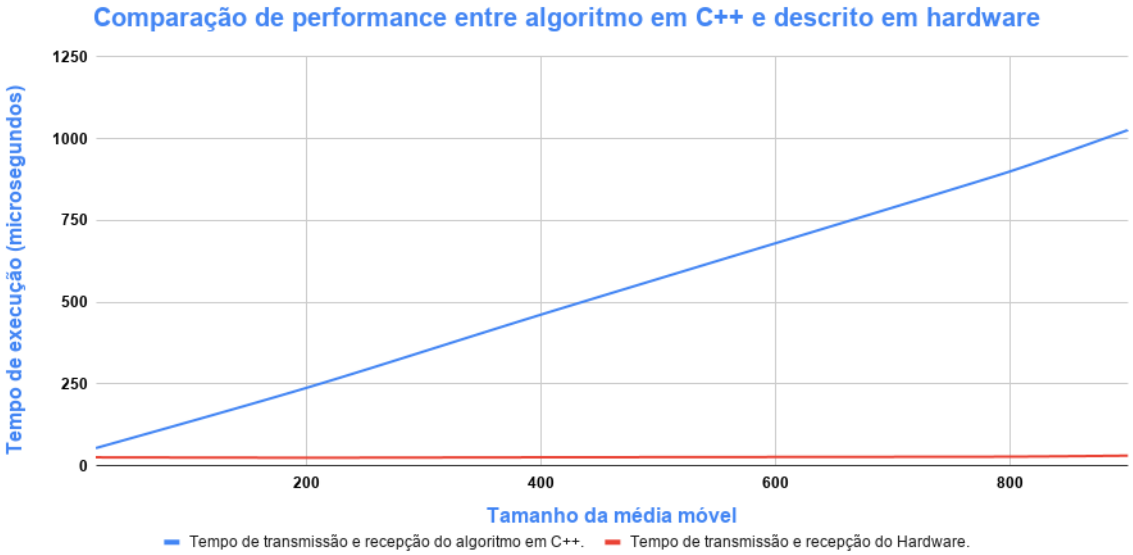
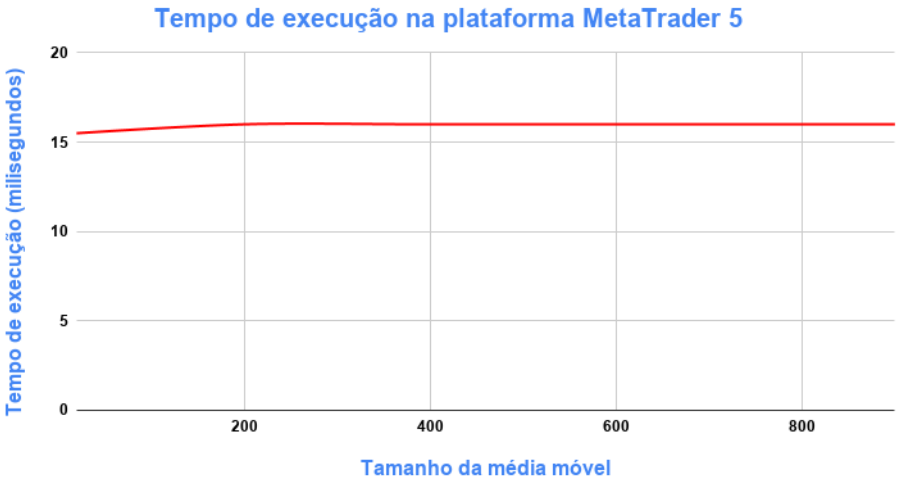


Figura 6.6: Tempo de execução para algoritmo em *MQL5*.

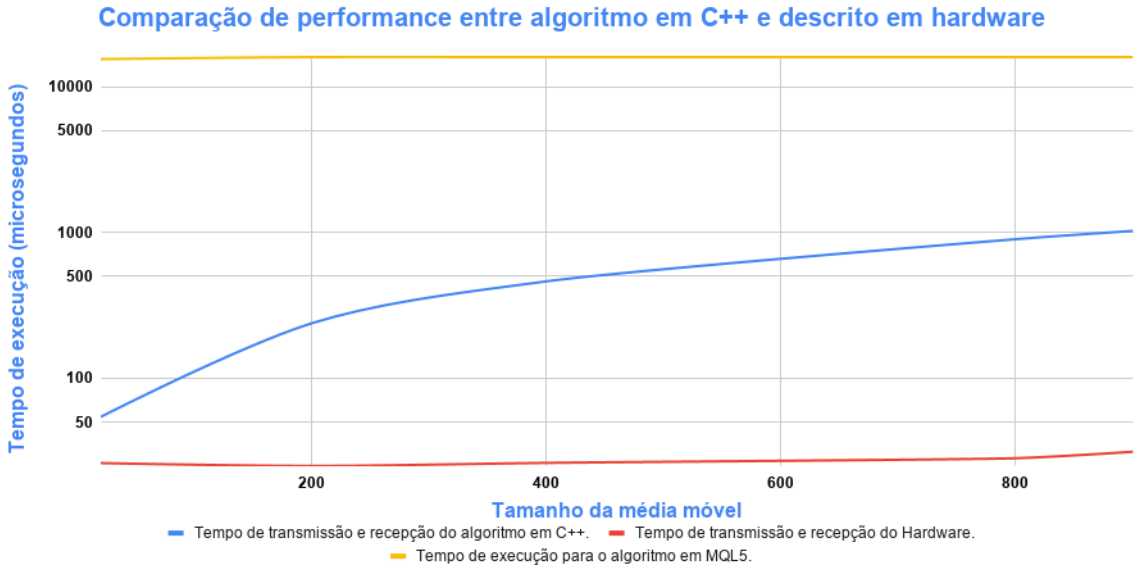


Essa Figura deixa claro quanto o desenvolvimento de um *hardware* dedicado é superior em relação a implementação de um *software* em um computador de propósito geral. Outro ponto interessante dessa comparação é que, devido a extrema velocidade do *hardware*, o atraso de comunicação com a plataforma *MetaTrader 5* foram os balizadores do tempo de execução em *FPGA*. A média do *delay* de comunicação com a plataforma é de 26 microsegundos. Se compararmos esse valor com o tempo de execução sem transmissão e recepção é possível perceber que a comunicação é muito mais significativa, sendo assim, esse foi em média o valor mínimo do tempo de execução do algoritmo em *hardware*. Vale ressaltar que o *hardware* permanece constante a medida que os valores computados pela média aumentam, enquanto a versão em *software* demonstra um aumento linear no tempo de execução, esse resultado é extremamente satisfatório para o desenvolvimento do projeto.

6.1.3.1 Comparação com MQL5

Para fins de comparação foi gerada também, uma versão do algoritmo das *Bandas de Bollinger* na linguagem do *MetaTrader 5*, *MQL5*. Para a obtenção dos dados de tempo de execução na ferramenta, foi utilizado um contador que retorna o tempo de execução de uma função dentro da plataforma em microsegundos. Infelizmente, o *timer* implementado na linguagem, não demonstrou precisão, nem estabilidade para obtenção de dados. Mesmo assim, a partir de uma média simples referente a execução do mesmo algoritmo 10 vezes para todos os tamanhos das médias móveis, foram obtidos os valores apresentados na Figura 6.6. Essa demonstra que a versão possui um tempo médio de execução de 16 milissegundos.

Figura 6.7: Comparação do tempo de execução de todas as versões implementadas das "bandas de bollinger".



A Figura 6.7 mostra uma combinação de todos os tempos de execução. É possível observar que a implementação mais rápida é a implementada em *hardware*.

7 CONCLUSÃO

High-frequency trading é uma área extremamente ampla, versátil e com um alto potencial agregado. É um modelo de computação que com absoluta certeza estará cada vez mais sendo utilizado e pesquisado ao redor do mundo. E portanto, a busca por estratégias mais performáticas e velozes para maior obtenção de lucro seguirá como tendência exponencial a medida que a tecnologia humana for evoluindo. Os resultados obtidos neste trabalho de graduação demostram o enorme potencial e vantagem que um *hardware* dedicado pode apresentar sobre um computador de propósito geral. Um dos pontos principais é referente a baixa variabilidade do tempo de execução em relação ao aumento do número de dados na média móvel do algoritmo, o que demonstra uma grande superioridade em relação ao *software*. Outro dado de grande relevância foi a demonstração que os métodos de transmissão e recepção se tornaram os balizadores do tempos de execução. Ou seja, a partir de um certo limiar a redução no tempo de execução via aceleradores de *hardware* passa a não se tornar mais expressivo, devido ao fato da comunicação entre plataformas não ser tão eficiente. Os principais pontos positivos e negativos encontrados na solução desenvolvida utilizando acelerador de *hardware* são:

Positivos:

- Extrema velocidade para realizar o cálculo proposto.
- Tempo de execução constante e estável mesmo com o aumento do número de variáveis do algoritmo.
- Possibilidade de reconFiguração.
- Não necessita dividir processamento com a *CPU*.

Negativos:

- Tempo de desenvolvimento.
- Dificuldade na integração *hardware*/*software*.
- Métodos de comunicação de dados pouco eficientes.

Por fim, um dos resultados mais importantes apresentados nesta monografia é referente a solução final permear várias áreas do conhecimento. Desde o início desse projeto um dos principais objetivos e motivação foi: apresentar uma solução multidisciplinar, diferenciada do *status quo* e que propusesse um avanço tecnológico para a área de estudo.

7.1 Trabalhos futuros

Nesta seção serão apresentados alguns pontos que estão no *roadmap* de desenvolvimento e pesquisa da solução.

- Utilizar área de memória da *FPGA* para escrita e leitura de vários dados simultâneos.
- Encontrar outros meios de realizar a comunicação do acelerador com a plataforma *MetaTrader 5*.
- Realizar estudos referentes a conectar a *FPGA* diretamente na rede para realizar a aquisição, *parsing* dos dados e tomada de decisão.
- Testar o método das *Bandas de Bollinger* para processos de arbitragem.

Todos esses pontos apresentados propiciam os mais diversos ganhos em termos de performance e ganhos de recursos financeiros para a solução.

REFERÊNCIAS

ALDRIGE, I. **High-Frequency Trading - A Practical Guide to Algorithmic Strategies and Trading Systems**. [S.l.]: John Wiley Sons, Inc., 2010.

B3. **Resumo das operações**. 2019. Available from Internet: <http://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/consultas/mercado-de-derivativos/resumo-das-operacoes/resumo-por-produto/>.

BABBAR, R. A study of performance of moving average, bollinger bands and relative strength index in selected stocks and stock indices. In: . [S.l.: s.n.], 2011.

COSTA, C. R.; ROSA, L. d. S.; BONATO, V. Integrando o metatrader5 com aceleradores fpga via opencl named pipes. In: **Workshop em Computação Heterogênea - WCH**. [S.l.]: SBC, 2018.

DIGILENT. **NetFPGA-1G-CML Reference Manual**. 2018. Available from Internet: <https://reference.digilentinc.com/reference/programmable-logic/netfpga-1g-cml/reference-manual>.

GOULART, P. et al. Netfpga: Processamento de pacotes em hardware. In: . [S.l.: s.n.], 2016.

HALILBEGOVIC; SANEL. Macd - analysis of weaknesses of the most powerful technical analysis tool. **Independent Journal of Management Production**, v. 7, p. 367–379, 06 2016.

INSTRUMENTS, T. **DMA Fundamentals on Various PC Platforms**. 2001. Available from Internet: <http://cires1.colorado.edu/jimenez-group/QAMSResources/Docs/DMAFundamentals.pdf>.

INTEL. **Introduction to PCI Express**. 2003. Available from Internet: <http://213.55.83.214:8181/Economics\%203/Computer\%20Science\%207/09386.pdf>.

INVESTOPEDIA. **Relative Strength Index – RSI**. 2019. Available from Internet: <https://www.investopedia.com/terms/r/rsi.asp>.

KHAMBATTI, M. Named pipes , sockets and other ipc. In: . [S.l.: s.n.], 2001.

LEBER, C.; GEIB, B.; LITZ, H. High frequency trading acceleration using fpgas. In: **2011 21st International Conference on Field Programmable Logic and Applications**. [S.l.: s.n.], 2011. p. 317–322.

METAQUOTES. **Explicação bandas de Bollinger e imagem**. 2009. Available from Internet: <https://www.mql5.com/en/forum/215>.

MICROSOFT. **Explicação Named-Pipes**. 2018. Available from Internet: <https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes>.

MOROSAN, A. The relative strength index revisited. In: . [S.l.: s.n.], 2011.

PATEL; HWU. Accelerator architectures. **IEEE Micro**, v. 28, n. 4, p. 4–12, July 2008. ISSN 1937-4143.

SCHRYVER, C. de; MARXEN, H.; SCHMIDT, D. Hardware accelerators for financial mathematics-methodology , results and benchmarking. In: . [S.l.: s.n.], 2011.

SILBERSCHATZ, G.; GAGNE. **Operating System Concepts**. [S.l.]: Wiley Publishing ©2012, 2002.

SUBRAMONI. Streaming, low-latency communication in on-line trading systems. In: . [S.l.: s.n.], 2010. p. 1–8.

SUISSE, C. **How HFT Has Altered Market Structure**. 2017. Available from Internet: <http://www.smallake.kr/wp-content/uploads/2017/10/HFT_world.pdf>.

SULAIMAN, N. et al. Design and implementation of fpga-based systems -a review. **Australian Journal of Basic and Applied Sciences**, v. 3, 10 2009.

XILLINX. **Understanding Performance of PCI Express Systems**. 2014. Available from Internet: <https://www.xilinx.com/support/documentation/white_papers/wp350.pdf>.

XILLYBUS. **Principle of operation**. 2019. Available from Internet: <http://xillybus.com/doc/xilinx-pcie-principle-of-operation>.

XILLYBUS. **Xillybus product brief**. 2019. Available from Internet: <http://xillybus.com/downloads/xillybus_product_brief.pdf>.