Universidade Federal de Pernambuco

Centro de Informática

Pós–graduação em Ciência da Computação

ALGORITMO DE PREFETCHING DE DADOS TEMPORIZADO PARA SISTEMAS MULTIPROCESSADORES BASEADOS EM NOC

Maria Cireno Ribeiro Silveira

DISSERTAÇÃO DE MESTRADO

Recife

2015

MARIA CIRENO RIBEIRO SILVEIRA

ALGORITMO DE PREFETCHING DE DADOS TEMPORIZADO PARA SISTEMAS MULTIPROCESSADORES BASEADOS EM NOC

Trabalho apresentado ao Programa de Pós–graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof^a. Dr^a. Edna Natividade da Silva Barros

Coorientador: Prof. Dr. André Aziz Camilo de Araújo (UFRPE)

Recife

2015

Catalogação na fonte
Bibliotecária Joana D'Arc Leão Salvador CRB4-532

S587a Silveira, Maria Cireno Ribeiro.
Algoritmo de prefetching de dados temporizado para sistemas multiprocessadores baseados em NOC / Maria Cireno Ribeiro Silveira. – Recife: O Autor, 2015.
111 f.: fig., tab., graf., quadros.

Orientadora: Edna Natividade da Silva Barros.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIN, Ciência da Computação, 2015.
Inclui referências e apêndices.

1. Engenharia de computador. 2. Arquitetura de computador. 3. Multiprocessador. I. Barros, Edna Natividade da Silva (Orientadora). II. Titulo.

621.39 CDD (22. ed.) UFPE-MEI 2015-078

Dissertação de Mestrado apresentada por **Maria Cireno Ribeiro Silveira** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Algoritmo de Prefetching de dados temporizado para Sistemas Multiprocessadores baseados em NoC**", orientada pela **Profa. Edna Natividade da Silva Barros** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Manoel Eusebio de Lima
Centro de Informática/UFPE

_____

Prof. Elmar Uwe Kurt Melcher
Departamento de Sistemas e Computação / UFCG

_____

Profa. Edna Natividade da Silva Barros
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 9 de março de 2015.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

AGRADECIMENTOS

Foi necessário muita dedicação e trabalho para que finalmente chegasse a conclusão do trabalho de mestrado. Nada disso seria possível sem o apoio das pessoas presentes em minha vida. Qualquer agradecimento que eu faça aqui não será suficiente para transmitir toda a minha gratidão.

Gostaria de agradecer primeiramente aos meus pais, Regina e Helder, pelo amor incondicional e pela dedicação na minha criação, vocês são responsáveis por tudo que sou. À minha irmã, Rebeca, pelo companheirismo e amizade de uma vida inteira.

Ao meu namorado Caio, pelo apoio, amor e motivação que me deram coragem para atingir meus objetivos. À sua família, por me acolher e me proporcionar vários momentos de alegria.

À minha família por ser a base da minha formação, especialmente às minhas avós, Denise e Helena, pelo carinho e compreensão. Aos meus amigos, irmãos que a vida me proporcionou, por me completarem.

Gostaria também de agradecer a Professora Edna, pela orientação acadêmica e amizade de longa data. E ao meu co-orientador e Professor André e sua esposa Fran, pela paciência, pelos ensinamentos, pelas conversas e pelo encorajamento constante.

Aos meus companheiros do grupo da pós-graduação e do trabalho, pelos conselhos e momentos de descontração no dia-a-dia.

Por fim, agradeço a todas as pessoas que me apoiaram de alguma forma e que me ajudaram a conquistar mais esse objetivo. Obrigada!

RESUMO

O prefetching é uma técnica considerada eficiente para mitigar um problema já conhecido em sistemas computacionais: a diferença entre o desempenho do processador e do acesso à memória. O objetivo do prefetching é aproximar o dado do processador retirando-o da memória e carregando na cache local. Uma vez que o dado seja requisitado pelo processador, ele já estará disponível na cache, reduzindo a taxa de perdas e a penalidade do sistema. Para sistemas multiprocessadores baseados em NoCs a eficiência do prefetching é ainda mais crítica em relação ao desempenho, uma vez que o tempo de acesso ao dado varia dependendo da distância entre processador e memória e do tráfego da rede.

Este trabalho propõe um algoritmo de prefetching de dados temporizado, que tem como objetivo minimizar a penalidade dos núcleos através uma solução de prefetching baseada em predição de tempo para sistemas multiprocessadores baseados em NoC. O algoritmo utiliza um processo pró-ativo iniciado pelo servidor para realizar requisições de prefetching baseado no histórico de perdas de cache e informações da NoC. Nos experimentos realizados para 16 núcleos, o algoritmo proposto reduziu a penalidade dos processadores em 53,6% em comparação com o prefetching baseado em eventos (faltas na cache), sendo a maior redução de 29% da penalidade.

Palavras-chave: Multiprocessador. Manycore. Prefetching. NoC - Network-on-Chip. Coerência de cache.

ABSTRACT

The prefetching technique is an effective approach to mitigate a well-known problem in multi-core processors: the gap between computing and data access performance. The goal of prefetching is to approximate data to the CPU by retrieving the data from the memory and loading it in the cache. When the data is requested by the CPU, it is already available in the cache, reducing the miss rate and penalty. In multiprocessor NoC-based systems the prefetching efficiency is even more critical to system performance, since the access time depends of the distance between the requesting processor and the memory and also of the network traffic.

This work proposes a temporized data prefetching algorithm that aims to minimize the penalty of the cores through one prefetching solution based on time prediction for multiprocessor NoC-based systems. The algorithm utilizes a proactive process initiated by the server to request prefetching data based on cache miss history and NoC's information. In the experiments for 16 cores, the proposed algorithm has successfully reduced the processors penalty in 53,6% compared to the event-based prefetching and the best case was a penalty reduction of 29%.

Keywords: Multiprocessor. Manycore. Prefetching. NoC - Network-on-Chip. Cache coherence.

LISTA DE FIGURAS

1.1 Classificação de estratégias de prefetching de acordo com Byna et al. . . 20

2.1 NoC em topologia Mesh 2-D de tamanho 3x3 e seus elementos. 25
2.2 Exemplos de séries temporais com diferentes comportamentos. 29

3.1 Diagrama de blocos da abordagem "When to Prefetch" de acordo com Byna et al. 31
3.2 Linha do tempo de uma requisição de prefetching de acordo com Sun et al. 37

4.1 Localização do componente de prefetching em uma arquitetura baseada em diretório. 44
4.2 Arquitetura interna de componente de prefetching. 45
4.3 Máquina de estados do algoritmo de Prefetch Temporizado. 46
4.4 Chegada do dado com atraso em relação à requisição do processador. . . 49
4.5 Chegada do dado muito adiantado em relação à requisição do processador. 49
4.6 Chegada do dado logo antes da requisição do processador. 50
4.7 Ocorrências de faltas de cache definindo uma série temporal. 52
4.8 Arquitetura de um processador dual core hipotético. 54
4.9 Exemplo de solicitação de leitura da CPU do núcleo 0 de um endereço hipotético. 56
4.10 Exemplo da execução de uma transação de prefetching em um processador dual core. 57
4.11 Exemplo de atualização das estatísticas dos controladores de prefetching. 57

5.1 Visão geral da plataforma Infinity instanciada com 16 núcleos. 60

5.2 Exemplo de funcionamento da tabela MLDT. 61
5.3 Exemplo do componente de Prefetching modificado. 64
5.4 Estrutura interna do subsistema de Controle de Prefetching Temporizado. 65
5.5 Diagrama de dados do Algoritmo de Prefetching Temporizado. 66

LISTA DE TABELAS

6.1 Tabela com resultados da penalidade para aplicação Radix. 74
6.2 Tabela com resultados do número de transações na para aplicação Radix. 75
6.3 Tabela com resultados do percentual de precisão do prefetching para a aplicação Radix. 76
6.4 Tabela com percentual de transações de prefetching atrasadas para a aplicação Radix. 77
6.5 Tabela com percentual de poluição na cache local provocada pelo prefetching para a aplicação Radix. 78
6.6 Tabela com resultados da penalidade para aplicação Sha. 80
6.7 Tabela com número de transações na rede para a aplicação Sha. 81
6.8 Percentual de precisão do prefetching para a aplicação Sha. 83
6.9 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Sha. 83
6.10 Percentual de transações de prefetching atrasadas para a aplicação Sha. . 83
6.11 Tabela com resultados da penalidade para aplicação Lu. 84
6.12 Tabela com número de transações na rede para a aplicação Lu. 84
6.13 Percentual de precisão do prefetching para a aplicação Lu. 86
6.14 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Lu. 87
6.15 Percentual de transações de prefetching atrasadas para a aplicação Lu. . 87
6.16 Tabela com resultados da penalidade para aplicação Basicmath. 87
6.17 Tabela com número de transações na rede para a aplicação Basicmath. . 87
6.18 Percentual de precisão do prefetching para a aplicação Basicmath. 89

6.19 Percentual de transações de prefetching atrasadas para a aplicação Basicmath. 90
6.20 Tabela com resultados da penalidade para aplicação Susan Corners. . . . 90
6.21 Tabela com número de transações na rede para a aplicação Susan Corners. 91
6.22 Percentual de precisão do prefetching para a aplicação Susan Corners. . . 92
6.23 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Susan Corners. 93
6.24 Percentual de transações de prefetching atrasadas para a aplicação Susan Corners. 93
6.25 Melhoria da penalidade média normalizada em relação à abordagem NO-PREFETCHING para as aplicações Lu, Basicmath, Sha e Susan Corners. 94
6.26 Quantidade de transações na rede normalizada em relação à abordagem NO-PREFETCHING para as aplicações Lu, Basicmath, Sha e Susan Corners. 95
6.27 Melhoria da precisão do prefetching em valores percentuais em relação à abordagem EVENT para as aplicações Lu, Basicmath, Sha e Susan Corners. 97
6.28 Comparativo da porcentagem de poluição causada pelo prefetching em relação à abordagem EVENT para as aplicações Lu, Sha e Susan Corners. 99
6.29 Comparativo da porcentagem de transações de prefetching atrasadas em relação à abordagem EVENT para as aplicações Lu, Basicmath, Sha e Susan Corners. 100
6.30 Resumo dos resultados mais significativos de redução da penalidade. . . . 100

7.1 Configurações da plataforma testadas com benchmark sintético. 107

LISTA DE QUADROS

2.1 Quadro comparativo de vantagens e desvantagens de sistemas com NoC. 24

2.2 Quadro com os tipos de séries temporais e métodos de previsão adequados para cada tipo. 28

3.1 Quadro comparativo dos trabalhos relacionados. 41

6.1 Configuração da Plataforma utilizada nos experimentos. 71

7.1 Quadro comparativo dos trabalhos relacionados com o trabalho proposto. 103

LISTA DE GRÁFICOS

6.1 Penalidade média dos processadores para a aplicação Radix. 74
6.2 Número de transações na rede para a aplicação Radix. 75
6.3 Percentual de precisão do prefetching para a aplicação Radix. 76
6.4 Percentual de transações de prefetching atrasadas para a aplicação Radix. 76
6.5 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Radix. 77
6.6 Penalidade média para a aplicação Radix com agressividade 4, variando o tamanho da janela de previsão. 78
6.7 Número de transações na rede para a aplicação Radix com agressividade 4, variando o tamanho da janela de previsão. 79
6.8 Penalidade média dos processadores para a aplicação Sha. 80
6.9 Número de transações na rede para a aplicação Sha. 81
6.10 Percentual de precisão do prefetching para a aplicação Sha. 81
6.11 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Sha. 82
6.12 Percentual de transações de prefetching atrasadas para a aplicação Sha. . 82
6.13 Penalidade média dos processadores para a aplicação Lu. 84
6.14 Número de transações na rede para a aplicação Lu. 85
6.15 Percentual de precisão do prefetching para a aplicação Lu. 85
6.16 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Lu. 86
6.17 Percentual de transações de prefetching atrasadas para a aplicação Lu. . 86
6.18 Penalidade média dos processadores para a aplicação Basicmath. 88

6.19 Número de transações na rede para a aplicação Basicmath. 88
6.20 Percentual de precisão do prefetching para a aplicação Basicmath. 89
6.21 Percentual de transações de prefetching atrasadas para a aplicação Basicmath. 89
6.22 Penalidade média dos processadores para a aplicação Susan Corners. . . . 90
6.23 Número de transações na rede para a aplicação Susan Corners. 91
6.24 Percentual de precisão do prefetching para a aplicação Susan Corners. . . 91
6.25 Percentual de poluição na cache L1 provocada pelo prefetching para a aplicação Susan Corners. 92
6.26 Percentual de transações de prefetching atrasadas para a aplicação Susan Corners. 92
6.27 Penalidade média dos processadores adicionando as abordagens CONTROLLED e CONTROLLED+TIMED em relação as aplicações Lu, Basicmath, Sha e Susan Corners. 95
6.28 Número de transações de prefetching na rede adicionando as abordagens CONTROLLED e CONTROLLED+TIMED em relação as aplicações Lu, Basicmath, Sha e Susan Corners. 96
6.29 Precisão do prefetching em relação às aplicações Lu, Basicmath, Sha e Susan Corners. 97
6.30 Poluição causada pelo prefetching em relação às aplicações Lu, Sha e Susan Corners. 98
6.31 Porcentagem de transações de prefetching atrasadas para as aplicações Lu, Basicmath, Sha e Susan Corners. 99
7.1 Gráfico da penalidade média para a plataforma L1-1-WAY. 108
7.2 Gráfico da penalidade média para a plataforma L1-2-WAY. 108
7.3 Gráfico da penalidade média para a plataforma L1-4-WAY. 109
7.4 Gráfico da penalidade média para a plataforma L1-8-WAY. 109
7.5 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-1. . . . 109
7.6 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-2. . . . 110

7.7 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-4. . . . 110
7.8 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-8. . . . 110
7.9 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-16. . . 111
7.10 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-32. . . 111

LISTA DE ALGORITMOS

1 Prefetching Temporizado 67
2 Algoritmo de Holt 68

LISTA DE ACRÔNIMOS

IP *Intellectual Property* (Propriedade Intelectual)

NOC *Network-on-Chip* (Redes Intrachip)

SOC *System-on-Chip* (Sistema em um chip)

NUMA *Non-Uniform Memory Access* (Acesso à memória não uniforme)

DPS *Data Push Server* (Servidor de envio de dados)

MLDT *Multi-Level Difference Table* (Tabela de diferenças multinível)

RPT *Reference Prediction Table* (Tabela de predição referência)

PDM *Pattern Detection Manager* (Gerenciador de detecção de padrão)

PSS *Prefetch Strategy Selector* (Seletor de estrátégia de prefetching)

DMA *Direct Memory Access* (Acesso direto à memória)

UART *Universal asynchronous recerveir/transmitter* (Receptor/Transmissor assíncrono universal)

APIC *Advanced Programmable Interrupt Controller* (Controlador de interrupções avançado programável)

SUMÁRIO

Capítulo 1—Introdução 19

Capítulo 2—Conceitos Básicos e Fundamentos 23

2.1 NoC - Network-on-Chip 23
2.2 Prefetching de dados 25
2.3 Séries Temporais 27

Capítulo 3—Estado da Arte e Trabalhos Relacionados 30

3.1 Prefetching baseado em eventos 31
3.2 Outras classificações 33
3.3 Prefetching baseado em predição 35
3.4 Prefetching em Sistemas baseados em NoC 38

Capítulo 4—Proposta de Algoritmo de Prefetching de dados temporizado 42

4.1 Funcionamento do Algoritmo de Prefetching 46
4.2 Cálculo de previsão de tempo 48
4.2.1 Predição do tempo de falta 51
4.2.2 Cálculo do delta 53

Capítulo 5—Implementação 59

5.1 Plataforma Infinity 59

5.2 Implementação do Algoritmo de Prefetching 62
5.3 Parâmetros de previsão da Série temporal 66

Capítulo 6—Experimentos e Resultados 69

6.1 Configuração da Plataforma Infinity 70
6.2 Benchmarks 71
6.3 Resultados 71
6.3.1 Estatísticas Importantes 72
6.3.2 Análise dos Resultados 73
6.3.3 Prefetching temporizado com controle de agressividade 93

Capítulo 7—Conclusão 101

Referências 104

Apêndice 107

CAPÍTULO 1

INTRODUÇÃO

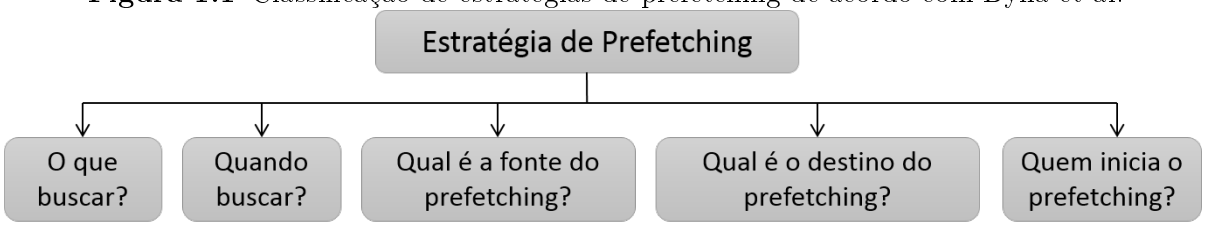
Um dos fatores limitantes para o aumento do desempenho dos sistemas é o tempo de acesso à memória. Existe uma defasagem entre o desempenho de processadores e de memórias RAM. O principal motivo do surgimento dessa diferença foi a divisão da indústria de semicondutores entre o desenvolvimento de memória e de processadores, consequentemente, cada uma das áreas focou em diferentes tecnologias: enquanto a tecnologia das memórias foi voltada para aumentar a capacidade, a de processadores se voltou para o aumento da velocidade (Carvalho, 2002). Assim, ao longo dos anos, essa diferença de desempenho vem aumentando exponencialmente e o quando o processador precisa acessar um dado da memória ele é forçado a esperar a latência do dado ser buscado na memória, esse tempo de processamento que é "sacrificado" pelo processador é chamado de penalidade.

As memórias cache foram inseridas nesse cenário para que os dados acessados recentemente pelo processador possam ser acessados em menos tempo. Porém, quando o processador requisita um dado que não se encontra na cache, caracteriza-se uma falta na cache e este dado precisa ser buscado no próximo nível de memória, penalizando o sistema. Uma das técnicas desenvolvidas para mitigar o problema de faltas na cache e reduzir a penalidade associada a tais faltas é o *prefetching*. O prefetching de dados é uma técnica que tem como objetivo aproximar os dados do processador, seja copiando o dado de um nível de memória para um nível mais acima (onde acima é mais próximo do processador) ou copiando o dado na cache local. Com isto, quando o processador requisitar um novo dado, ele já estará presente na cache e a penalidade associada a buscar o dado na memória será reduzida. O prefetching de dados utiliza-se da localidade espacial de dados, ou seja, da propriedade que diz que se um dado foi acessado recentemente, existe

grande chance de um dado próximo a ele ser também acessado.

Existem diferentes abordagens de prefetching de dados. Segundo Byna (Byna, Chen, and Sun, 2008), as estratégias de prefetching de dados podem atacar cinco diferentes problemas. A figura 1.1 apresenta a classificação proposta por Byna.

Figura 1.1 Classificação de estratégias de prefetching de acordo com Byna et al.



O problema de **"o que buscar?"** refere-se ao problema do prefetching de prever qual endereço de bloco deve ser carregado na cache local. A técnica mais utilizada para abordar este problema é o prefetching baseado em histórico, que procura encontrar um padrão nos últimos acessos para prever o próximo endereço de bloco que será requisitado pelo processador. Já o problema de **"quando buscar?"** é comumente solucionado por técnicas de prefetching baseadas em evento, que dependem da ocorrência de uma falta na cache, do acesso a um bloco de prefetching ou de cada acesso à memória para realizar o prefetching (Chen and Baer, 1995) (Brown, Wang, Chrysos, Wang, and Shen, 2002) (Kamruzzaman, Swanson, and Tullsen, 2011) (Flores, Aragon, and Acacio, 2010). Por outro lado, existem as técnicas que também utilizam o histórico de requisições do processador para executar o prefetching baseado em predição, como a técnica proposta por Sun (Sun, Byna, and Chen, 2007).

Sun (Sun, Byna, and Chen, 2007) propôs um servidor de envio de dados que tem como objetivo utilizar o padrão de acesso aos dados para prever não só o endereço do próximo dado a ser requisitado, mas também escolher o momento de enviar o dado para a cache. Utilizando-se da combinação da abordagem de **o que** e **quando** buscar, Sun obteve uma redução média de 71% na taxa de faltas de cache. Porém, é uma técnica para arquiteturas baseadas em barramento e centralizada, portanto, não escalável. Uma vez que a quantidade de processadores aumente em grande escala, o desempenho desta

técnica provavelmente degradaria.

No âmbito de prefetching de dados para sistemas baseados em NoC, Nachiappan (Nachiappanan, Sivasubramaniam, Mishra, Mutlu, Kandemir, and Das, 2012) apresentou um mecanismo de prefetching de priorização de transações da rede. Neste trabalho, as aplicações são avaliadas em relação ao seu potencial de utilização do prefetching utilizando um mecanismo de *feedback* do desempenho do prefetching (Srinath, Mutlu, Kim, and Patt, 2007). Este mecanismo avalia se a aplicação está sendo beneficiada pelo prefetching e se está causando pouca interferência nas outras aplicações e utiliza desta medida para aumentar ou diminuir a prioridade das transações de prefetching trafegando na NoC. Os resultados indicaram uma melhoria de 9% no desempenho do sistema.

As NoCs (*Network-on-Chip*) são, juntamente com o barramento, as interconexões mais utilizadas nas arquiteturas de multiprocessadores atuais. Em arquiteturas baseadas em barramento, em que a memória é compartilhada entre os núcleos, o tempo de acesso à memória é constante e, portanto, qualquer técnica de prefetching para sistemas baseados em barramento é mais simples. Por outro lado, estas arquiteturas apresentam problemas de desempenho e escalabilidade, como por exemplo, a limitação da quantidade de núcleo.

Como alternativa, os sistemas podem ser interconectados via NoCs. Nestas arquiteturas é utilizada uma rede para interconectar os processadores, assim, a limitação associada ao aumento do número de processadores é minimizada. Embora seja uma solução para a escalabilidade, as NoCs inserem um problema no tempo de acesso à memória, pois não garantem o mesmo tempo de acesso dos núcleos da rede aos endereços de memória compartilhada. Fatores como a distância entre a localização do dado e do processador e congestionamento da rede influenciam no tempo de acesso, por isso, estas arquiteturas são classificadas como NUMA (*Non-Uniform Memory Access*), ou seja, arquiteturas onde o tempo de acesso à memória não é uniforme e, portanto, variável. Sendo assim, além de variável, o tempo de acesso à memória pode ser bastante longo. Por isto, o prefetching para sistemas baseados em NoC assume grande importância no papel de minimizar a penalidade do sistema.

Nesse cenário, este trabalho propõe uma abordagem de prefetching de dados baseado

em predição de tempo. O objetivo é a redução da penalidade média dos processadores do sistema. A abordagem é voltada para sistemas multiprocessadores baseados em NoC e utiliza de previsão de tempo para determinar **quando** executar o prefetching. O algoritmo proposto utiliza um método de previsão de séries temporais chamado "Método de suavização exponencial de Holt" para, a partir do histórico de faltas na cache, prever quando o processador vai requisitar um novo dado e carregar o dado previsto na cache local daquele processador.

Uma vez que o tempo de acesso a memória é não uniforme, o algoritmo de prefetching proposto neste trabalho também utiliza de dados estatísticos da rede, como congestionamento e distância entre fonte e destino, para estimar a latência do dado ser carregado na cache após a requisição de prefetching ser enviada. O objetivo do mecanismo de prefetching é garantir que o dado esteja disponível na cache local no momento em que o processador requisitar o dado, minimizando a penalidade do sistema. Porém, caso este dado seja carregado na cache muito antes do processador utilizá-lo, ele pode substituir outro bloco da cache que ainda será utilizado, causando poluição na cache. Assim, o desempenho do prefetching é bastante sensível à estimativa de tempo calculada.

Para todas as aplicações utilizadas para validar o algoritmo foi possível encontrar uma configuração do algoritmo que reduziu a penalidade em relação à abordagem de prefetching baseada em evento. A maior redução de penalidade média obtida em relação à abordagem baseada em evento foi de 29%, enquanto em relação a não utilização de prefetching foi de 31%.

Este trabalho é estruturado da seguinte forma: no Capítulo 2 alguns conceitos básicos são explicados para facilitar o entendimento geral do trabalho. Em seguida, o Capítulo 3 apresenta técnicas relacionadas à técnica proposta e o estado da arte de prefetching de dados para sistemas baseados em NoC. No Capítulo 4, a técnica proposta neste trabalho e sua formalização são apresentadas e, no Capítulo 5, é explicado como a técnica proposta foi implementada. Por fim, o Capítulo 6 apresenta os experimentos realizados para validação da proposta e os resultados obtidos. O Capítulo 7 conclui o trabalho e propõe trabalhos futuros.

CAPÍTULO 2

CONCEITOS BÁSICOS E FUNDAMENTOS

Nesse capítulo serão explanados conceitos básicos para o entendimento geral do trabalho. A proposta é um algoritmo de prefetching de dados basedo em predição de tempo para sistemas multiprocessadores com comunicação via NoC. Por isto, conceitos como o de NoC, prefetching de dados e tópicos auxiliares serão apresentados a seguir.

2.1 NOC - NETWORK-ON-CHIP

Com o aumento da capacidade computacional dos sistemas em um chip (SoC - System-on-Chip) surge a necessidade de uma nova alternativa de comunicação que não limite a escalabilidade do sistema. As NoCs - Networks-on-Chip apresentam um conjunto de soluções de comunicação on-chip para suprir à necessidade de comunicação em larga escala ocupando pouco espaço do chip (Bjerregaard and Mahadevan, 2006).

Em sistemas com mais de um processador a comunicação via barramento se torna um limitador de escalabilidade, uma vez que a quantidade de informação trocada aumenta com o número de processadores e só existe uma via de comunicação. Neste caso, o barramento fica cada vez mais sobrecarregado, aumentando o tempo de espera de cada processador para que a informação seja enviada. Por outro lado, em sistemas onde os processadores são conectados por uma NoC, a comunicação entre processadores acontece ponto-a-ponto, permitindo que dois ou mais processadores enviem informações para a rede ao mesmo tempo. Dessa forma, o aumento do número de processadores não possui impacto negativo no desempenho. Bjerregaard (Bjerregaard and Mahadevan, 2006) faz um comparativo de algumas vantagens e desvantagens entre a utilização de NoC ou barramento. Essas informações podem ser observadas no Quadro 2.1.

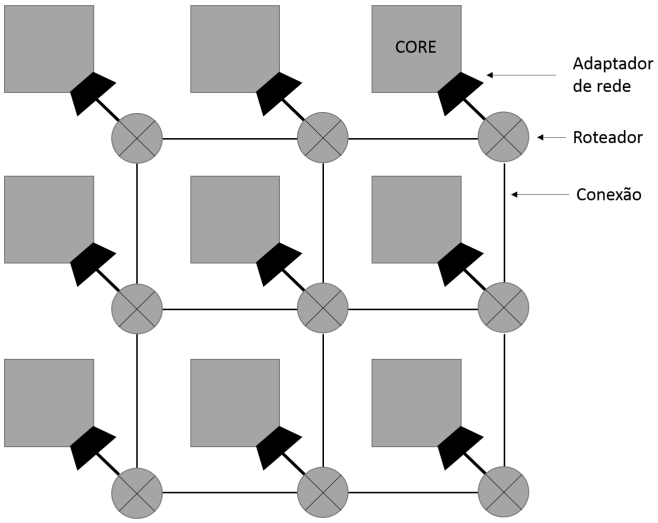
Quadro 2.1 Quadro comparativo de vantagens e desvantagens de sistemas com NoC.

Barramento	**NoC**
− Cada unidade de processamento anexada ao barramento adiciona capacitância prejudicando o desempenho elétrico	+ Aumento do número de processadores não prejudica tanto o desempenho individual de cada processador
− Temporização do barramento é mais complexa	+ Utilização das vias podem ocorrer ao mesmo tempo
− A arbitragem se torna um gargalo à medida que cresce o número de processadores	+ Decisões de roteamento podem ser distribuídas
− Para cada tamanho de rede o barramento tem que mudar	+ O mesmo roteador por ser utilizado para qualquer tamanho de rede
− Testar o barramento é mais lento e complexo	+ Mais rápido e fácil de testar
− Largura de banda é limitada e dividida entre as unidades anexadas	+ Largura de Banda aumenta com tamanho da rede
+ Uma vez que o árbitro tenha garantido controle, a latência é conhecida e pequena	− Congestionamento interno pode aumentar a latência da comunicação
+ Compatível com a maioria dos IPs	− Em geral, os IPs precisam de adaptação e sincronização
+ Os conceitos são simples e já bem difundidos	− Reeducação para novos paradigmas de programação com comunicação explícita entre processos concorrentes

Os elementos básicos de uma NoC são: adaptadores de rede, nós de roteamento ou roteadores e conexões. Os adaptadores implementam a interface de comunicação da rede com os IPs com o objetivo de separar a computação dos cores da comunicação da rede, tornando-as independentes. Os roteadores são os elementos responsáveis por tomar as decisões de roteamento da rede. De acordo com o protocolo de roteamento escolhido, eles redirecionam os pacotes de dados por diferentes caminhos, traçando a estratégia de roteamento. As conexões são as vias de comunicação que conectam os roteadores com seus vizinhos, formando a malha da rede. A malha formada pode ter variadas topologias, sendo a topologia Mesh a mais utilizada. A figura 2.1 mostra um exemplo de uma NoC de topologia Mesh de duas dimensões (2D), tamanho 3x3 e seus elementos.

Uma vez que a tendência dos sistemas é migrar para uma comunicação via NoC, surge a necessidade de desenvolver novas técnicas que se adaptem a este novo conceito. Uma destas técnicas é o prefetching de dados que será explicado na próxima seção 2.2

Figura 2.1 NoC em topologia Mesh 2-D de tamanho 3x3 e seus elementos.



2.2 PREFETCHING DE DADOS

Um problema conhecido nos sistemas computacionais modernos é o tempo de acesso à memória e o processamento que é penalizado devido à esse tempo. Em NoCs, este problema fica ainda mais evidente e complexo, uma vez que o tempo de acesso pode ser variável, pois depende de fatores como o congestionamento da rede e distância entre o processador e a memória que está sendo acessada. Com o objetivo de mitigar este problema novas soluções de hierarquia de memória foram sugeridas, dentre eles a utilização de memórias cache em um ou mais níveis. As memórias caches são memórias menores e mais rápidas que armazenam cópias de alguns blocos da memória principal para diminuir o tempo de acesso do processador à esses dados.

Quando um dado requisitado pelo processador não se encontra na cache ocorre uma falta na cache, que adiciona ao processador uma penalidade referente ao tempo de buscar o dado na memória principal ou no próximo nível de memória. A penalidade é o tempo de processamento desperdiçado para realizar uma leitura ou escrita do dado na memória.

Buscando diminuir a taxa de faltas na cache e a penalidade do processador, diferentes técnicas e arquiteturas foram desenvolvidas ao longo dos anos, uma dessas técnicas é o *Prefetching* ou pré-busca. O prefetching de dados tem como objetivo buscar dados que

ainda serão requisitados para mais perto do processador, seja para a cache local ou de um nível de memória para outro. Para isto, o mecanismo de prefetching tenta prever qual o próximo dado que será requisitado pelo processador e em que momento ele deve ser carregado na cache local do processador para que esteja disponível no momento da requisição.

Existem diferentes formas de se implementar um Prefetch de dados e vários parâmetros a serem explorados. Segundo a taxonomia proposta por Byna (Byna, Chen, and Sun, 2008) existem cinco principais problemas para se abordar em um mecanismo de prefetch:

- **O que buscar?** Qual o próximo dado que deve ser carregado na memória.
- **Quando buscar?** Quando deve ser iniciado o prefetch: sempre que a memória for acessada, na falta de cache etc.
- **Qual é a fonte do prefetching?** Qual a origem de dados do prefetching: memória, cache etc.
- **Qual é o destino do prefetching?** Para onde vão os dados do prefetch: cache privada, cache compartilhada etc.
- **Quem inicia o prefetching?** A partir do processador, a partir da memória etc.

Para fazer um prefetching eficiente deve-se considerar os seguintes parâmetros: estratégia de busca, ou seja, como será realizada a previsão de qual dado deve ser buscado; agressividade do prefetching, que é a quantidade de dados que serão buscados de uma só vez; o gatilho do prefetch, que define o momento em que a transação de prefetching será enviada; em que níveis de memória o prefetch vai atuar e que componente será responsável por realizar o prefetching.

Embora o Prefetch de dados para sistemas que se comunicam via barramento seja uma solução considerada consolidada, a adaptação dessas técnicas para sistemas com muitos processadores, que se comunicam via NoC, ainda é um problema com vários desafios a serem resolvidos. Um dos principais agravantes desse tipo de sistema é que o tempo necessário para os dados percorrerem a rede pode ser não-determinístico, o que

torna desconhecido o intervalo de tempo entre o envio de uma transação de prefetching e o momento em que o dado é disponibilizado, dificultando ainda mais a operação de prefetching, que deveria garantir que o bloco de dados estivesse disponível na cache no momento da requisição da CPU.

Quando a latência do envio de transações é não-determinístico resta a opção de realizar uma estimativa do tempo em que o dado estará disponível. Este trabalho tem como objetivo o desenvolvimento de uma técnica de prefetching, que além da estimativa da latência de transferência de dados via NoC, considera o suposto momento que a CPU irá requisitar um novo bloco da cache para realizar o prefetching. Para isto, é utilizada uma técnica de previsão de séries temporais. A próxima seção 2.3 apresenta os conceitos básicos de séries temporais.

2.3 SÉRIES TEMPORAIS

Uma série temporal é uma forma de representar um conjunto de observações discretas ordenadas no tempo. Por exemplo, o consumo de energia elétrica mensal de uma residência, ou a produção industrial de uma indústria por ano. Por ser um indicador importante em várias áreas, diferentes técnicas de previsão de séries temporais foram estudadas e hoje são de grande importância na ciência, engenharia e negócios.

Formalmente, uma série temporal pode ser expressa pela equação 2.1. Onde t é um índice temporal discreto e N é o número de observações.

$$Z_t = \{Z_t \, \epsilon \, \Re \, | t = 1, 2, 3...N\} \tag{2.1}$$

Existem três componentes principais em uma série temporal: aleatoriedade, tendência e sazonalidade. A aleatoriedade está presente em todas as séries e representa a variação dos valores em torno da tendência. A sazonalidade é uma componente presente na série temporal quando ela possui um comportamento que tende a se repetir a cada p períodos de tempo. Alguns autores também colocam uma componente cíclica, que é uma sazonalidade repetitiva. A tendência é a componente de crescimento ou descrescimento da série, esta

pode ser linear, exponencial ou amortecida. A Figura 2.2 mostra exemplos de séries com diferentes comportamentos (Ehlers, 2009).

A série (a) é o número total de passageiros em linhas aéreas internacionais nos EUA por mês, entre os anos de 1949 e 1960. Ela possui uma componente de tendência com crescimento linear e um padrão sazonal ao longo dos anos. Em (b) tem-se o número anual de linces capturados em armadilhas no Canadá entre os anos de 1821 e 1934. Esta série obedece um padrão constante com flutuações aleatória e também possui uma tendência cíclica a cada 10 anos. Já a série (c), que represeta as medições anuais de vazão do Rio Nilo em Ashwan entre 1871 e 1970, possui um padrão constante com flutuações aleatórias, mas não possui componente de tendência nem sazonalidade. Finalmente, a série (d) representa o consumo de gás no Reino Unido entre o primeiro trimestre de 1960 e o quarto trimestre de 1986. Nesta série pode-se observar uma tendência de crescimento linear superposto por flutuações aleatórias.

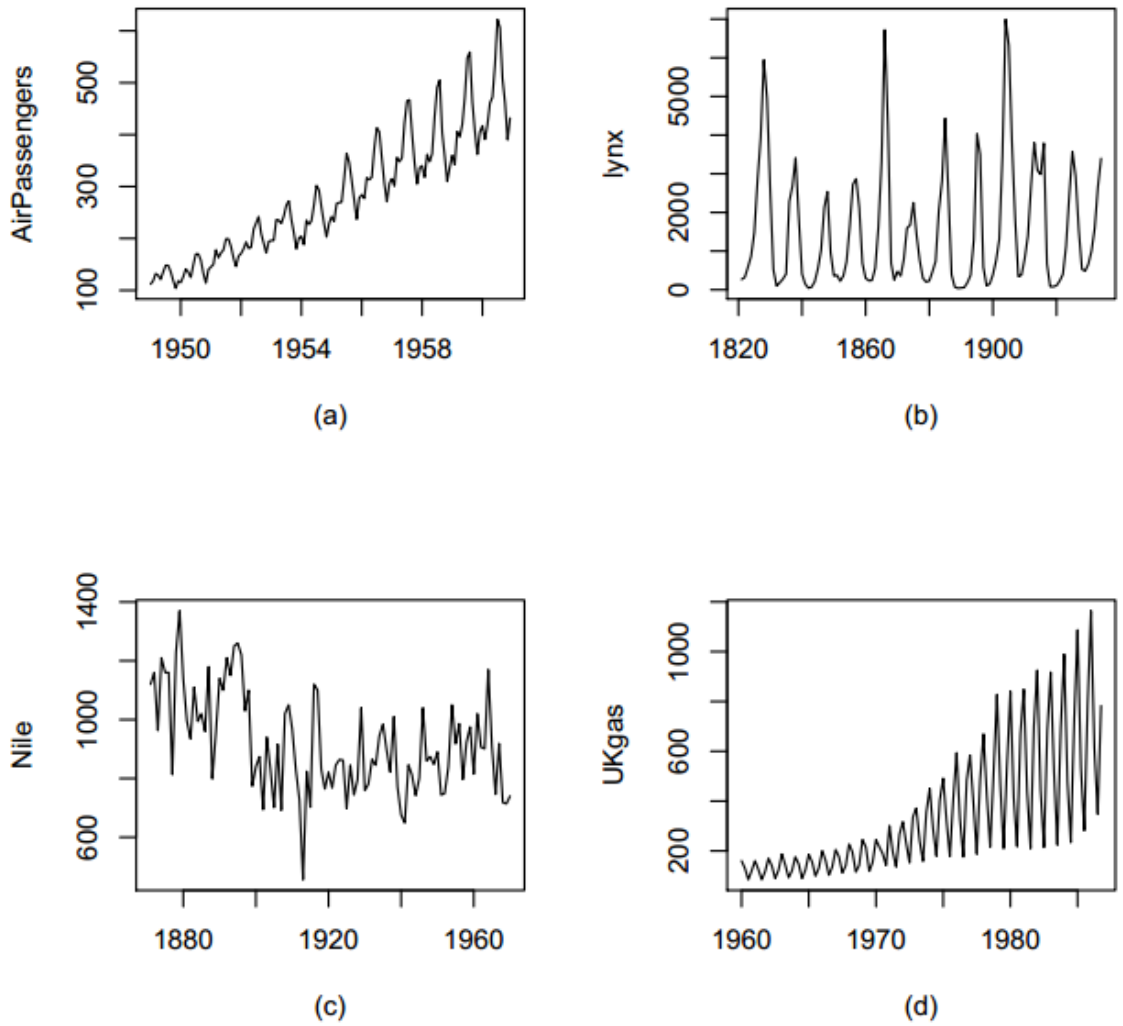
Existem vários métodos de previsão de séries temporais e cada um se aplica da melhor forma para cada tipo de série. O Quadro 2.2 mostra uma classificação para diferentes tipos de séries e as técnicas de previsão indicadas para cada tipo, segundo (Ribeiro, Goldschmidt, and Choren, 2009). O método de interesse para este trabalho é o Método de previsão com suavização exponencial de Holt que é explicado em detalhes no Capítulo 4.

Quadro 2.2 Quadro com os tipos de séries temporais e métodos de previsão adequados para cada tipo.

Série Temporal	**Método de Previsão**
Valor constante superposto a flutuações aleatórias	Método de previsão de media móvel e Método de previsão com suavização exponencial
Processo linear superposto a flutuações aleatórias	Método de previsão com suavização exponencial de Holt
Valor constante superposto a variações sazonais e flutuações aleatórias	Método de previsão com suavização exponencial de Holt-Winters

Os principais métodos de previsão incluem dois parâmetros associados a uma série temporal: janela de previsão e horizonte de previsão. O horizonte de previsão representa

Figura 2.2 Exemplos de séries temporais com diferentes comportamentos.
Fonte: Ehlers, 2009.



o período coberto de previsão, onde $Z_t(h)$ é o último ponto da série que pode ser previsto e h é o horizonte de previsão. A janela de previsão é o conjunto de pontos pertencentes à série $(Z_t(k), Z_t(k+1), \ldots, Z_t(k+j))$, onde j é o tamanho da janela de previsão, que estão sendo observados como base para a previsão dos próximos valores, ou seja, é o histórico de valores anteriores ao horizonte de previsão utilizados para criar um padrão de comportamento para a série.

No Capítulo 4 veremos como o conceito de Série Temporal foi aplicado no trabalho e de que forma sua modelagem influencia nos resultados.

CAPÍTULO 3

ESTADO DA ARTE E TRABALHOS RELACIONADOS

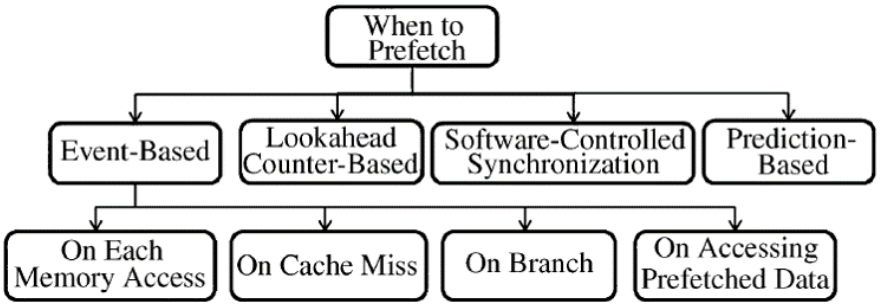
O maior desafio de mecanismos de Prefetching de dados que consideram o problema de **quando buscar** é garantir que o dado seja carregado no seu destino antes de ser requisitado pelo processador, evitando uma falta de cache. A escolha do momento de enviar o dado, considerando o tempo de processamento do prefetching, a latência (tempo que leva para o bloco ser copiado da memória para a cache) e quando o dado será requisitado possui um impacto significante na eficiência do prefetching e, por isso, devem ser mensurados ou estimados com a maior precisão possível.

Como apresentado no Capítulo 2 os mecanismos de prefetching podem focar no problema de "o que buscar" e de "quando buscar". Alguns trabalhos (Sun, Byna, and Chen, 2007) fazem uma combinação dessas estratégias buscando melhorar a eficiência total da solução. Outros mecanismos apresentam independência entre "o que" e "quando" buscar, lidando separadamente com a escolha do algoritmo de predição do próximo endereço e o momento de enviar a transação de prefetching. Ao longo dos anos foram apresentadas diversas abordagens diferentes (Sun, Byna, and Chen, 2007) (Liu, Dimitri, and Kaeli, 1999) (Chen and Baer, 1995) (T and A., 1991) (Caragea, Tzannes, Keceli, and Vishkin, 2010) focando em quando se deve realizar o prefetching. Byna (Byna, Chen, and Sun, 2008) classificou estas abordagens como mostrado na figura 3.1, pode-se observar que a decisão de "when to prefetc" (quando realizar o prefetching) é dividida em quatro abordagens: *Event based* (baseado em eventos), *Lookahead counter based* (baseado em contador de instruções antecipado), *Software controlled synchronization* (sincronização controlada por software) e *Prediction based* (baseado em predição). O prefetching baseado em eventos, por sua vez, é subdividido em quatro categorias: *On each memory access* (em cada

acesso à memória), *On cache miss* (na falta da cache), *On branch* (em uma sequência de instruções) e *On accessing prefetched data* (no acesso a dados de prefetching).

Neste capítulo vamos explicar a classificação proposta por Byna (Byna, Chen, and Sun, 2008) e apresentar algumas técnicas que se enquadram nos diferentes tipos. O foco maior será dado para as técnicas baseadas em predição (*prediction based*), que incorporam um certo grau de pró-atividade e independência no prefetching, uma vez que estas utilizam o histórico de acesso à cache para prever quando enviar uma transação de prefetching, ao invés de esperar pela ocorrência de um evento, como as técnicas baseadas em evento (*event based*). A escolha de dar mais atenção às técnicas baseadas em predição é devido ao fato do algoritmo de prefetching proposto neste trabalho se enquadrar nesta classificação.

Figura 3.1 Diagrama de blocos da abordagem "When to Prefetch" de acordo com Byna et al.
Fonte: Byna, 2008.



3.1 PREFETCHING BASEADO EM EVENTOS

Estratégias baseadas em evento se iniciam quando um evento ocorre: uma falta de dado na cache, o primeiro acesso a um dado que foi carregado pelo prefetch, um acesso à memória ou até o início de uma sequência de instruções. Este tipo de abordagem é o mais difundido por ser mais simples de implementar.

Byna (Byna, Chen, and Sun, 2008) apresenta quatro subclassificações de prefetching baseadas em evento. A busca iniciada pelo **acesso a memória** também é chamado de estratégia **"sempre buscar"**, pois independente se ocorreu uma falta ou um acerto de

cache, novos dados são buscados pelo mecanismo de prefetching e carregados na cache. Uma desvantagem deste tipo de prefetching é a poluição gerada na cache que pode levar à substituição de dados que ainda serão requisitados pelo processador por novos dados buscados pelo prefetching, causando uma falta na cache forçada.

A estratégia de realizar uma busca quando ocorre **uma falta de cache** é comumente utilizada devido à sua simplicidade (Brown, Wang, Chrysos, Wang, and Shen, 2002) (Kamruzzaman, Swanson, and Tullsen, 2011) (Flores, Aragon, and Acacio, 2010). Neste mecanismo, toda vez que é detectada uma falta de cache o algoritmo de prefetching calcula o endereço do próximo dado a ser buscado e faz a requisição para que seja carregado na cache. Caso ocorra outra falta, um novo dado será buscado e assim em diante. Enquanto não houver falta na cache, novos dados não serão carregados.

Outras abordagens de prefetching de dados estão associadas ao prefetching de instruções. Liu (Liu, Dimitri, and Kaeli, 1999) apresenta como exemplo de prefetching **no acesso de desvios** uma técnica que associa referências de dados com instruções. Para isto, quando uma instrução de desvio é encontrada, o mecanismo de predição prediz qual caminho o desvio deve tomar e realiza o prefetching dos dados referenciados nas instruções seguintes ao desvio. A vantagem desta técnica é o aproveitamento da previsão do comportamento dos desvios. Como essa previsão é bastante precisa, as referências de dados tem grande chance de obedecer o mesmo padrão. Porém, ainda existe uma quantidade significante de dados que vão fugir do padrão de acesso, degradando a eficiência do prefetching.

Por último, tem-se o evento de **acesso a um dado fruto de prefetching**, nesse caso, os dados carregados na cache pelo mecanismo de prefetching são identificados e quando seus blocos são acessados pela primeira vez o mecanismo é disparado para executar uma nova busca. Esta técnica é quase tão eficiente quanto a estratégia de **sempre buscar**, enquanto o prefetching na ocorrência de falta de cache possui quase a metade da eficiência (Liu, Dimitri, and Kaeli, 1999) da estratégia de **sempre buscar**.

Estas abordagens deram resultados positivos em sistemas com um único processador. Porém, com o aumento do número de processadores (multiprocessador e multicore)

nos sistemas computacionais e a utilização de NoCs na comunicação desses sistemas, a penalidade associada ao acesso à memória quando um dado requisitado pela CPU não se encontra na cache local aumenta consideravelmente. Então, surge a necessidade de mecanismos de prefetching que antecipem a necessidade dos processadores e realizem pró-ativamente requisições de prefetching, com o objetivo de diminuir a penalidade no acesso a dados da memória.

3.2 OUTRAS CLASSIFICAÇÕES

Na classificação proposta por Byna (Byna, Chen, and Sun, 2008) ainda existem mais duas abordagens de prefetching que focam no problema de quando buscar: prefetching controlado por software e prefetching baseado em contador de instruções antecipado.

Em (Chen and Baer, 1995), foram propostas três variações de prefetching baseado em hardware que aborda o problema de **quando** realizar o prefetching. A idéia básica do esquema de prefetching proposto é acompanhar o padrão de acesso dos *loops* a endereços de dados e armazená-los numa Tabela de Predição de Referências (RPT - *Reference Prediction Table*) que será utilizada como referência para realizar o prefetching. A RPT funciona como uma cache de instruções que guarda as informações de acesso de instruções LOAD e STORE dos *loops*: o endereço da instrução, o último endereço de dado que foi referenciado por aquela instrução, a diferença entre os últimos dois endereços (*stride*) que foram gerados e o estado da referência. O estado da referência pode ser:

- Inicial: setado na primeira entrada da referência no RPT ou após uma predição incorreta no estado estável;
- Transitório: corresponde ao caso quando o sistema não tem certeza se a última previsão foi boa ou não;
- Estável: indica que a previsão deve ficar estabilizada por um tempo;
- Sem predição: o prefetching para essa entrada é desabilitado.

As três variações de abordagem propostas por (Chen and Baer, 1995) se diferenciam pelo momento de realizar o prefetching, são elas:

1. Básica: o prefetching é realizado uma iteração antes para que o dado esteja disponível na cache na próxima iteração.

2. Lookahead: utiliza um contador de instruções para antecipar iterações baseado na latência de acesso à memória.

3. Correlacionada: realiza uma detecção de padrão nos loops.

A predição Básica (1) utiliza o histórico armazenado na RPT para realizar a requisição de prefetching da próxima iteração. Quando o contador do programa decodifica uma instrução LOAD/STORE, a RPT é consultada para verificar se existe uma entrada correspondente à instrução, caso não exista ela é adicionada. Caso a referência exista e seja possível prever (está no estado inicial, transitório ou estável) a referência da próxima iteração a ela, o prefetching do endereço calculado pela soma do último endereço acessado mais o *stride* é realizado. O problema desta abordagem é que ela não se preocupa com a latência do acesso à memória. Então caso o corpo do *loop* seja pequeno é provável que o prefetching do dado da iteração anterior chegue atrasado no seu próximo acesso, por outro lado caso o corpo do *loop* seja muito grande existe uma chance que o bloco substitua (ou seja substituído) por outros blocos antes que o dado seja utilizado.

A abordagem que utiliza o contador de instruções antecipado (*Lookahead*) tem o objetivo de melhorar a abordagem básica. O tempo ideal para realizar uma requisição de prefetching é δ ciclos na frente do uso do dado, onde δ é a latência de acesso ao próximo nível na hierarquia de memória. Dessa forma, o prefetching *Lookahead* utiliza um contador de instruções adiantado em relação ao contador real em δ ciclos e que acessa a RPT para gerar as requisições de prefetching.

Como solução mais complexa foi apresentada a abordagem correlacionada. A idéia chave da predição baseada em referência correlacionada é acompanhar não só os acesso adjacentes internos ao loop, mas também os acessos quando muda o nível do loop. Para

isto, foi incluído na RPT um registrador que armazena o retorno dos últimos desvios e a RPT foi estendida utilizando campos separados para computar os acessos correlacionados.

Embora as três abordagens tenham apresentado uma diminuição significativa na penalidade do acesso a dados, a abordagem utilizando o contador de instruções antecipado (*Lookahead*) obteve melhor desempenho, obtendo reduções na penalidade de 16% a até 97%. A diferença entre a abordagem básica para *Lookahead* foi de até 40% a mais de penalidade, já a abordagem correlacionada obteve valores de penalidade em torno de 2% a mais de penalidade.

Apesar de não ser o foco deste trabalho, na taxonomia proposta por Byna (Byna, Chen, and Sun, 2008) também são apresentadas abordagens de prefetching que buscam resolver o problema de **quando** buscar utilizando prefetching controlado por software. Um desses trabalhos foi desenvolvido por Mowry (T and A., 1991). Nessa abordagem é necessário que o desenvolvedor ou compilador insira funções de prefetching em referências de dados que podem causar faltas na cache. Assim, quando o software é executado, um hardware específico faz as requisições de prefetching para os dados que foram inseridos pelo desenvolvedor ou compilador.

Para aplicações com padrão de acesso a dados regular, a abordagem proposta por Mowry se mostrou bastante eficiente, aumentando o desempenho de 100% a 150%. Já para aplicações com padrão de acesso mais complexo, o prefetching foi menos eficiente, chegando a no máximo 30% de melhora no desempenho.

Na próxima seção são apresentadas técnicas de prefetching baseadas em predição.

3.3 PREFETCHING BASEADO EM PREDIÇÃO

Seguindo a linha de prefetching baseado em predição, Sun (Sun, Byna, and Chen, 2007) propôs uma técnica para realizar requisições de prefetching iniciadas por um servidor de prefetching, chamado *Data Push Server (DPS)*, em ambiente multiprocessador. Os autores atacaram dois tópicos críticos **"o que"** e **"quando"** buscar, criando um servidor que prediz o próximo endereço de dado usando o histórico de acesso e também prediz o momento para enviar o dado para que ele esteja disponível a tempo.

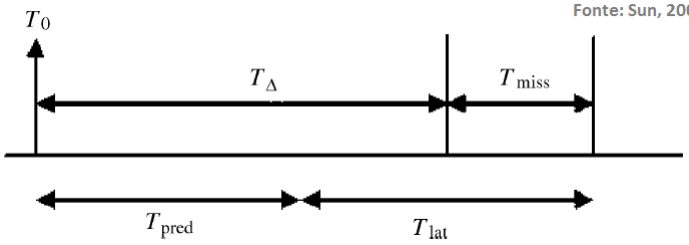
Na abordagem proposta por Sun, o DPS possui três componentes primários: gerenciador de detecção de padrão, mecanismo de prefetching e mecanismo de gerenciamento. O gerenciador de detecção de padrão (PDM) coleta o histórico de acesso à dados e as informações de tempo e espaço desses acessos, onde o tempo é quando ocorreu o acesso e de espaço são os avanços entre acessos consecutivos. Em seguida, o PDM classifica os padrões de acesso entre: contínuo, descontínuo e uma combinação entre contínuo e descontínuo. Já o mecanismo de prefetching é responsável por prever futuros acessos e quando eles irão ocorrer. Ele possui três subcomponentes:

- Seletor de estratégia de prefetching (PSS) - seleciona adaptativamente um método de previsão de futuros acessos baseado na informação de padrão.
- Preditor de prefetching - prediz "o que" buscar utilizando a técnica *MLDT (Multi-level difference table)* (Sun, Byna, and Chen, 2007).
- Gerador de requisições - decide "quando" buscar enviando a transação de prefetching para que o dado cheque em tempo.

A decisão do Gerador de requisições de "quando" buscar depende de três fatores: T_{pred} é o tempo para prever futuros acessos, T_{lat} representa a latência the iniciar e transferir um dado da sua fonte para seu destino e T_{Δ} é o intervalo entre o tempo atual e o tempo da próxima referência a um dado que causaria uma falta, caso não houvesse prefetching. A Figura 3.2 mostra uma linha do tempo de como esses fatores se relacionam, onde T_{miss} é a penalidade causada por uma falta de cache. Nessa figura tem-se quatro possíveis cenários:

1. $(T_{pred} + T_{lat}) > (T_{\Delta} + T_{miss})$
2. $(T_{pred} + T_{lat}) > T_{\Delta}$ e $(T_{pred} + T_{lat}) < T_{miss}$
3. $(T_{pred} + T_{lat}) = T_{\Delta}$
4. $(T_{pred} + T_{lat}) < T_{\Delta}$

Figura 3.2 Linha do tempo de uma requisição de prefetching de acordo com Sun et al.



No cenário 1 o prefetching não possui nenhum efeito positivo, pois o processador já pagou toda a penalidade referente à uma falta na cache quando não se tem prefetching. Já no cenário 2, existe um ganho parcial de desempenho em relação à penalidade total de uma falta na cache. O cenário 3 é o ideal quando se realiza um prefetching, pois o dado ficou disponível exatamente no momento que o processador requisitou o dado. No cenário 4 existem duas alternativas, caso o bloco substitua outro bloco que seria utilizado pelo processador, é criado o efeito negativo da poluição de cache que pode gerar mais faltas na cache. Caso o bloco seja alocado em um espaço vazio, não existem efeitos negativos no prefetching.

Finalmente, o último componente do DPS é o mecanismo de gerenciamento, este componente reúne as informações do PDM e do mecanismo de prefetching e envia a requisição de prefetching do dado para levá-lo da memória para a unidade de processamento que precisará do dado.

Os resultados apresentaram uma redução média de 71% na taxa de faltas de cache. Neste caso, a boa eficiência do prefetching é uma consequência da combinação do acerto da previsão de acesso aos dados e da precisão do momento de enviar o dado, que é feito por um gerador de requisições. A abordagem DPS considera três fatores para garantir que o dado seja carregado na cache a tempo: o tempo necessário para realizar os cálculos de predição, o tempo estimado de quando o processador irá requisitar o dado e a latência da transferência do dado até que seja copiado na cache.

Pode-se observar que os intervalos de tempo para sistemas baseados em barramento ou com um processador citados anteriormente são conhecidos, com a exceção do tempo para o dado ser requisitado pelo processador, que pode ser estimado. No caso de multi-

processadores baseados em NoC a latência de acesso ao dado não é conhecida.

Embora as abordagens apresentadas tenham sido de grande importância para a evolução das técnicas de prefetching, sua utilização em sistemas multiprocessadores baseados em NoC não é viável, uma vez que o tempo de transferência dos dados, por exemplo, é não-determinístico. Neste tipo de cenário a decisão de quando realizar o prefetch se torna bem mais complexa, pois outros fatores devem ser levados em consideração como: o congestionamento da rede, a distância entre processadores e o algoritmo de roteamento utilizado. Estes fatores possuem impacto direto na eficiência do prefetching e devem ser considerados pela técnica de prefetching.

Na próxima seção vamos apresentar o estado da arte das técnicas de prefetching desenvolvidas para sistemas manycore.

3.4 PREFETCHING EM SISTEMAS BASEADOS EM NOC

Com o aumento da capacidade computacional dos sistemas surge a necessidade do aumento do número de processadores em um único chip. Com isso é criado um gargalo de comunicação via barramento que vem sendo solucionado pela utilização de redes on-chip (NoC - Network-on-Chip). Em geral, estes sistemas já são desenvolvidos com suporte à escalabilidade, por isso, surgem novos paradigmas de desenvolvimento e programação. O prefetching é ainda um problema pouco explorado na área de sistemas baseados em NoC. Exlorando essa necessidade, Aziz (Aziz, 2014) desenvolveu um sistema de prefetching com controle adaptativo da agressividade do prefetching para sistemas baseados em NoC.

O trabalho proposto por Aziz utiliza estatísticas do sistema, como penalidade e taxa de faltas na cache, para controlar dinamicamente o número de blocos buscados no prefetching, ou seja, a agressividade do prefetching de dados. O prefetching realizado por Aziz utiliza uma MLDT para prever o próximo endereço a ser buscado e realiza o prefetching baseado em evento: falta na cache. Os componentes de prefetching são distribuídos nos nós de processamento da NoC, permitindo a escalabilidade do sistema sem aumentar significativamente a complexidade do prefetching. Os resultados mostraram uma diminuição média da penalidade de 20% para aplicações single core e de 23% para aplicações

multithread utilizando quatro núcleos. Para uma configuração com dezesses núcleos, a diminuição de penalidade foi de 4%.

Seguindo a linha de prefetching baseado em NoCs, Nachiappan desenvolveu um mecanismo de priorização de prefecthing. No trabalho proposto por Nachiappan (Nachiappanan, Sivasubramaniam, Mishra, Mutlu, Kandemir, and Das, 2012), foi criado um sistema de prefetching que realiza a priorização de transações de prefetching na NoC dependendo das aplicações e do efeito que o prefetching possui para cada aplicação. O mecanismo de prefetching foi baseado em (Srinath, Mutlu, Kim, and Patt, 2007), que realiza a predição do endereço utilizando o histórico de acessos e executa o prefetching baseado nos eventos de falta na cache e acesso a um dado fruto de prefetching.

Primeiramente, os prefetchings das aplicações são classificados baseados no seu potencial de utilização para aquela aplicação e na sua propensão de causar interferência negativa nas outras aplicações. Idealmente, a prioridade maior será das transações de prefetching da aplicação que mais se beneficia do prefetching e que menos prejudica às outras aplicações. Enquanto as transações de prefetching das aplicações que se beneficiam pouco do prefetching e que ainda degradam o desempenho das outras aplicações serão menos priorizadas. Esse grau de benefício do prefetching para as aplicações e dos malefícios às outras aplicações é baseada em duas métricas: precisão do prefetching e contagem do prefetching, onde a precisão é o número de acertos na cache causados por transações de prefetching em relação ao número total de transações, e a contagem é o número de transações de prefetching injetadas na rede. No caso, a prioridade é maior para as transações de prefetching da aplicação onde o prefetching é mais preciso e que envia menos transações, pois menos transações indica que aquele prefetching congestiona menos a rede, tendo menor interferência nas outras transações de prefetching.

A classificação do efeito do prefetching na aplicação é feita em quatro categorias de acordo com as duas métricas mencionadas. Da categoria de maior prioridade para a de menor tem-se:

- Alto grau de precisão e baixo número de transações na rede.
- Alto grau de precisão e alto número de transações na rede.

- Baixo grau de precisão e baixo número de transações na rede.
- Baixo grau de precisão e alto número de transações na rede.

A precisão do prefetching e o número de transações do prefetching podem ser medidas dinamicamente ou estaticamente. Estaticamente, um programa de perfil categoriza as aplicações, na versão dinâmica, um hardware coleta as informações e no final de um ciclo classifica as aplicações. Essa informação é utilizada no próximo ciclo.

Os resultados indicaram uma melhoria de 9% no desempenho do sistema utilizando o mecanismo de prefetching com priorização dinâmica em relação a um prefetching sem priorização de transações.

Este trabalho é um exemplo de mecanismo de prefetching que considera a influência de utilização de NoCs na comunicação de sistemas multiprocessadores. Por ser baseado em NoCs e ser uma solução de prefetching distribuída é um trabalho que pode ser classificado como escalável.

Para fazer uma análise dos trabalhos relacionados foi construído um quadro comparativo que classifica os trabalhos de acordo com a classificação de **quando buscar** proposta por Byna e mais três características que este trabalho propõe para comparação:

- Implementação - se o trabalho foi desenvolvido em hardware ou software.
- Interconexão - se o trabalho foi desenvolvido para interconexão via barramento ou NoC.
- Escalabilidade - se o sistema é distribuído ou centralizado, pois sistemas centralizados tendem a ser não escaláveis devido ao processamento ficar restrito a um único local.

O Quadro 3.1 apresenta a classificação dos trabalhos de acordo com essas cinco variáveis. Foi escolhido um trabalho de cada abordagem **quando buscar**: baseado em evento, baseado em predição, baseado em contador de instruções antecipado e sincronização controlada por software. De acordo com essa classificação é possível identificar os trabalhos de Sun (Sun, Byna, and Chen, 2007) e Aziz (Aziz, 2014) como principais estratégias de

comparação com a solução proposta neste trabalho. Sun por se tratar de um prefetching baseado em predição, embora seja desenvolvido para um sistema com interconexão de barramento, e Aziz por se tratar de uma solução baseada em NoC, embora o prefetching seja baseado em eventos (falta na cache).

Quadro 3.1 Quadro comparativo dos trabalhos relacionados.

Trabalho	**Quando buscar?**	**Implementação**	**Interconexão**	**Escalabilidade**
Chen, 1995	Baseado em contador de instruções antecipado	Hardware	Barramento	Centralizado
Mowry, 1991	Sincronização controlada por software	Software	Barramento	N/A
Sun, 2007	Baseado em predição	Hardware	Barramento	Centralizado
Nachiappan, 2012	Baseado em evento	Hardware	NoC	Distribuído
Aziz, 2014	Baseado em evento (falta na cache)	Hardware	NoC	Distribuído

Como pode ser observado no Quadro 3.1, as abordagens que utilizam predição baseada em tempo não são escaláveis, enquanto as abordagens escaláveis são baseados em ocorrências de evento. Como sistemas baseados em NoC são ainda mais sensíveis ao desempenho do prefetching, devido à não-uniformidade do tempo de acesso à memória, surge uma oportunidade de utilizar predição de tempo para tentar melhorar o desempenho do prefetching. Dessa forma, foi proposto este trabalho de um algoritmo de prefetching de dados temporizado, que será explicado em seguida no Capítulo 4.

CAPÍTULO 4

PROPOSTA DE ALGORITMO DE PREFETCHING DE DADOS TEMPORIZADO

Técnicas de prefetching para sistemas de multiprocessadores possuem diferenças significativas em relação àquelas desenvolvidas para sistemas com um único processador. Primeiramente, os paradigmas de programação são diferentes e provavelmente necessitarão de informações adicionais sobre o acesso aos dados. Em segundo lugar, as hierarquias de memória de sistemas multiprocessadores são, em geral, mais complexas, possuem latências maiores e a fonte e o destino do prefetching podem ser diferentes. Isso porque várias arquiteturas utilizam a memória compartilhada distribuída para evitar que uma mesma memória fique sobrecarregada com várias requisições. Em terceiro lugar, e mais importante, o desempenho desses sistemas são mais sensíveis ao prefetching, uma vez que várias requisições de prefetching podem sobrecarregar a comunicação e causar congestionamento na rede, aumentando consideravelmente a latência de acesso aos dados.

No Capítulo 3 foram apresentadas algumas técnicas de prefetching propostas tanto para sistemas multiprocessadores quanto para com um único processador. Várias técnicas desenvolvidas para multiprocessadores que se comunicam por barramento foram estendidas de técnicas para um único processador, pois parâmetros como largura de banda e latência são conhecidos, o que permite a adaptação dos algoritmos sem grandes dificuldades. Porém, para sistemas multiprocessador baseados em NoC que possuem escalabilidade para suportar uma quantidade muito grande de nós de processamento, a complexidade do hardware necessário para implementar um prefetching eficiente e que atenda a mudança estrutural e de paradigmas é maior. Como consequência, tem-se a proposição de novos algoritmos de prefetching de dados mais adequados para uma arquitetura baseada em NoC.

Frente a essas dificuldades, foi proposto por Aziz (Aziz, 2014) um controle de agressividade de prefetch para sistemas multicores com arquitetura baseada em NoC. Para tal, foi desenvolvida uma plataforma multiprocessador onde o algoritmo foi validado e que será detalhada no Capítulo 5. Porém, o mecanismo de prefetching de dados proposto por Aziz é baseado em eventos. No caso em questão, o evento é uma falta de cache: quando ocorre uma falta o módulo de prefetching executa operações para prever o próximo endereço e realizar a busca dos dados, com o objetivo de minimizar a taxa de faltas e reduzir a penalidade do sistema.

Buscando a melhoria da eficiência do prefetching e, consequentemente, do desempenho da plataforma, este trabalho propõe um algoritmo de prefetching de dados baseado em predição de tempo para atuar na plataforma, agindo de forma pró-ativa na operação de prefetching, ao invés de apenas esperar a ocorrência do evento de falta de cache. Essa proposta será melhor detalhada na Seção 4.1.

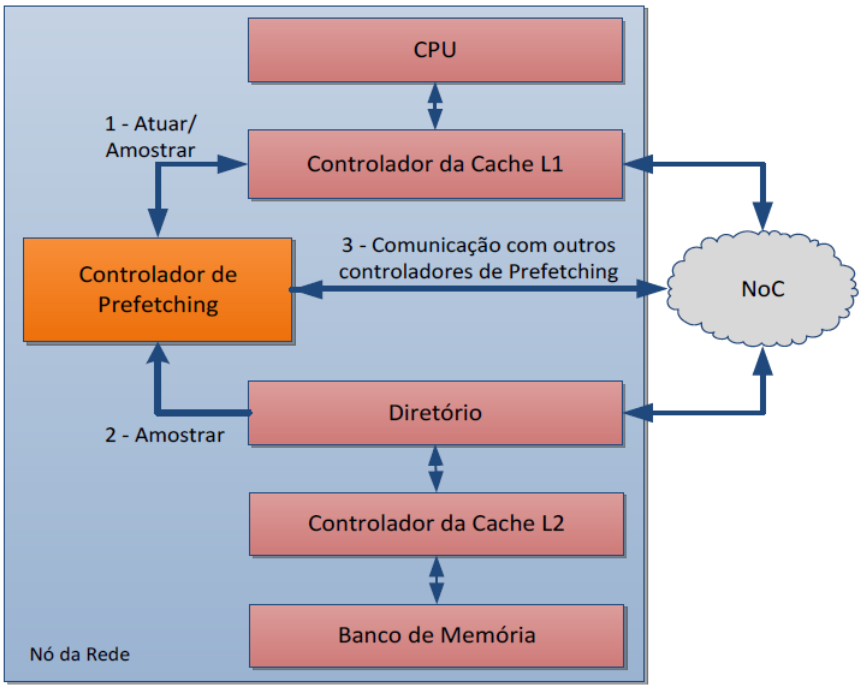
A Figura 4.1 mostra a estrutura de cada nó da plataforma proposto por (Aziz, 2014) e da localização do controlador de prefetching dentro do nó. A troca de informações em torno do controlador ilustra como ele se comunica com os componentes fa plataforma e por quais funções é responsável:

1. Amostrar dados e atuar na cache;
2. Amostrar dados do diretório;
3. Trocar informações com outros componentes da rede.

O controlador de prefetching amostra do controlador de cache a penalidade de cada requisição de acesso à memória realizada pelo processador. Essa é uma das métricas que será utilizada pelo algoritmo desenvolvido para estimar o tempo de prefetching do nó. A atuação do controlador de prefetching é a requisição dos blocos da cache. Os dados amostrados do diretório pelo controlador de prefetching são referentes às requisições de blocos de cache feitas pelos controladores de caches L1 (local ou remoto). Essas informações são utilizadas para poder fazer previsões de quais endereços os controladores de prefetching irão precisar em um futuro próximo, e encaminhar o resultado para os

Figura 4.1 Localização do componente de prefetching em uma arquitetura baseada em diretório.

Fonte: Aziz, 2014.



controladores de prefetching que requisitarão esses blocos aos controladores de cache. Desta forma, cada controlador de prefetching realiza previsões de endereços na faixa de endereços cujo diretório associado a ele é responsável. Portanto, um controlador de prefetching pode enviar previsões de endereços para todos os controladores de cache e receber previsões de todos os controladores (local ou remoto).

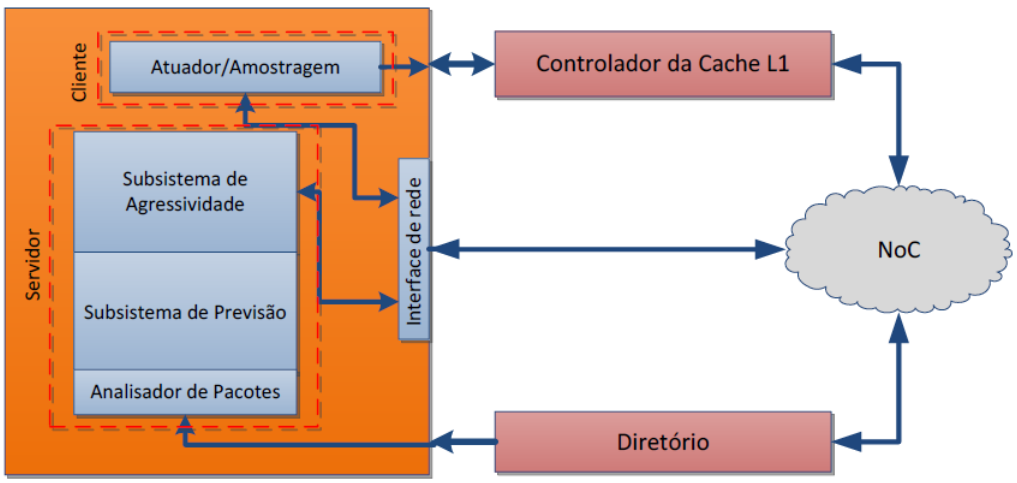
O trabalho proposto foi inspirado no controle de agressividade desenvolvido por Aziz (Aziz, 2014) e, para validação do algoritmo serão explorados os casos de prefetching com agressividade fixa e também a atuação do algoritmo trabalhando em conjunto com o controlador de agressividade proposto por Aziz. O objetivo é mostrar um comparativo entre o prefetching baseado em evento e o prefetching baseado em previsão de tempo proposto neste trabalho, com e sem a influência do controlador de agressividade. Além do módulo de controle de agressividade ser independente (pode-se utilizá-lo ou utilizar agressividade fixa) da atuação do algoritmo de prefetching proposto, o módulo de previsão de endereços pode implementar diferentes técnicas de previsão de endereço. Porém é importante

observar que o módulo de previsão de endereços é essencial para o funcionamento e desempenho do prefetching, pois a precisão do prefetching depende diretamente dele. Isto porque não basta que o dado esteja disponível em tempo para a CPU utilizá-lo, mas que também seja o dado correto.

A Figura 4.2 ilustra o componente de prefetching da plataforma "Infinity". O componente de prefetching é formado por duas partes principais: o servidor e o cliente; onde o cliente é responsável por realizar ações junto ao controlador de cache e o servidor por realizar ações junto ao diretório. O prefetching cliente realiza amostragem das penalidades da cache em relação a cada um dos diretórios com os quais ela se relacionou e, de tempos em tempos (10.000 ns), envia essa informação a cada um dos prefetching servidores. Além disso, o prefetching cliente recebe de cada prefetching servidor as previsões sobre quais blocos ele deve buscar, atuando na cache como se fosse um processador e, com isso, adiantando a busca de dados.

Figura 4.2 Arquitetura interna de componente de prefetching.

Fonte: Aziz, 2014.



Já a parte servidor do controlador pode ser dividida em dois subsistemas e um módulo auxiliar:

- Módulo Analisador de Pacote: responsável por identificar, dentre todas as transações que chegam ao diretório, as transações referentes às faltas de blocos na cache

e repassar os endereços faltantes para o subsistema de previsão de endereço.

- Subsistema de Previsão de Endereço: responsável pelo cálculo do próximo endereço que a cache irá demandar do diretório.

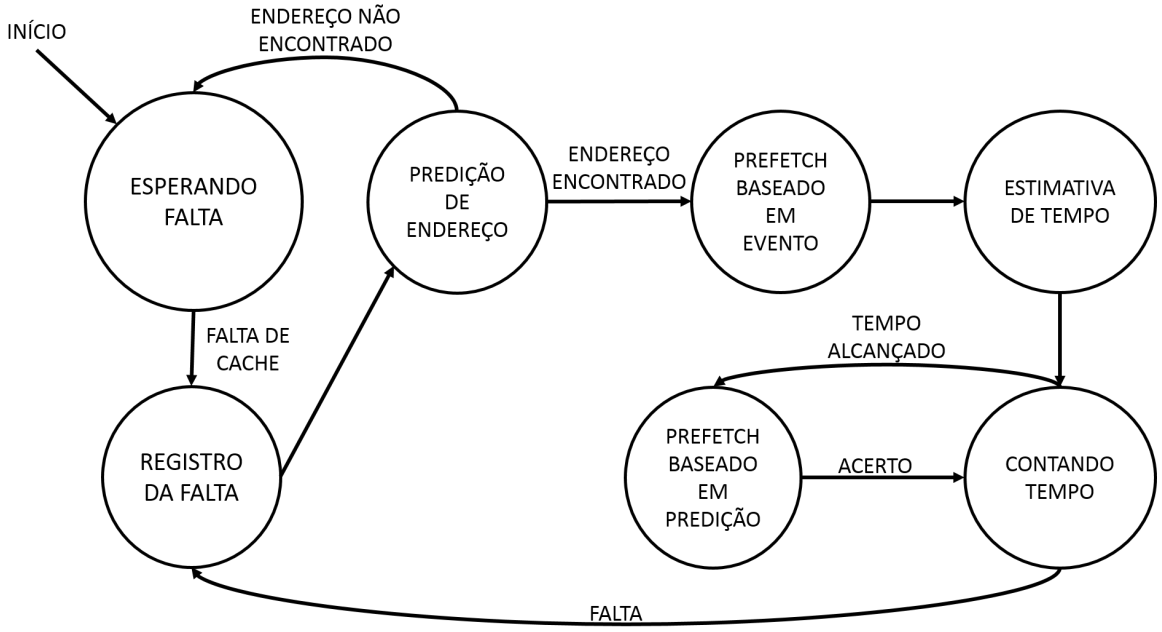
- Subsistema de Controle de Agressividade: responsável pelo cálculo da quantidade de blocos que serão pré-buscados a partir do endereço previsto.

No Capítulo 5 será explicado como são implementados os subsistemas do controlador de prefetching e que modificações foram feitas no componente de prefetching proposto por Aziz para incorporar o algoritmo de prefetching temporizado à plataforma já existente.

4.1 FUNCIONAMENTO DO ALGORITMO DE PREFETCHING

O algoritmo proposto neste trabalho aborda o problema de "quando realizar uma busca antecipada" usando uma combinação de prefetching baseado em evento (falta de cache) com prefetching baseado em predição de tempo. A Figura 4.3 apresenta a máquina de estados do algoritmo proposto.

Figura 4.3 Máquina de estados do algoritmo de Prefetch Temporizado.



O estado inicial do algoritmo é a espera por uma ocorrência de falta de cache, quando a falta de cache é detectada, ela é registrada sendo salvo seu endereço, o instante em que ocorreu e qual o processador requisitante. Em seguida, esses dados são utilizados pelo preditor de endereço para encontrar um padrão de acesso à memória. Caso o padrão seja encontrado e, consequentemente, o próximo endereço a ser buscado esteja definido, é realizado o prefetch baseado em evento (falta de cache) para adiantar a necessidade do processador antes de se iniciar os cálculos de previsão do prefetch temporizado.

Existem vários fatores a serem considerados quando se tenta prever em que momento o dado será requisitado e, mais ainda, quando deve se iniciar a busca para que o dado esteja disponível na cache a tempo de evitar uma falta. Para fazer a estimativa de quando o processador irá requisitar o dado, o algoritmo utiliza um método de previsão de séries temporais alimentado pelo histórico de faltas de cache. Este método será apresentado com maiores detalhes na Subseção 4.2.1.

Após a previsão de tempo o módulo de prefetch temporizado passa a ter controle sobre as requisições de prefetching. Até que ocorra uma nova falta de cache, o módulo espera pelo tempo previsto e requisita uma nova busca. Quanto mais requisições forem feitas sem que ocorra uma falta, melhor será, desde que não cause congestionamento na rede. Quando uma falta acontece, o algoritmo retorna para o seu estado inicial e retoma o processo de predição.

Embora uma busca seja realizada quando ocorre uma falta de cache a contribuição principal do algoritmo é o prefetch pró-ativo realizado pela predição de tempo que visa não só a diminuição da taxa de faltas, mas principalmente, a diminuição da penalidade total do sistema. A penalidade total do sistema é calculada pelo somatório da penalidade de cada processador. Define-se como penalidade o tempo total que o processador fica aguardando a cópia dos dados na cache, incluindo a resolução da coerência de cache. Então, reduzindo a penalidade, o sistema ganha em desempenho.

Somado ao desafio de prever quando será a próxima falta de cache está a complexidade de estimar o tempo necessário para que o dado fique disponível para o processador. O objetivo principal é disponibilizar o dado em uma janela específica de tempo que

se posiciona imediatamente antes da requisição do processador, pois se o dado chegar muito cedo pode causar a poluição da cache, ou seja, pode substituir um outro dado que ainda será utilizado, forçando uma falta. Para realizar a estimativa de tempo existem vários fatores a considerar: a distância [1] entre o nó fonte e o nó destino, o algoritmo de roteamento, o tempo de resolução da coerência, o congestionamento da rede e o tempo de processamento do próprio algoritmo de prefetch.

Um fato importante a ser observado é que o prefetching só tem sentido se o próximo endereço de busca tiver sido calculado, então, embora o controle de prefetching temporizado seja independente do resultado do algoritmo de previsão de endereços, ele depende diretamente do resultado do endereço previsto para realizar as requisições de prefetching. Na plataforma Infinity é utilizada uma técnica chamada MLDT – *Multi-level Difference Table* (Sun, Byna, and Chen, 2007) para realizar o cálculo de endereço do prefetching.

Nas próximas seções será explicado como ocorre o cálculo da previsão de tempo e, mais importante, como é escolhido o momento de requisitar o dado.

4.2 CÁLCULO DE PREVISÃO DE TEMPO

O processo de requisição do prefetching tem início no nó servidor. Para cada nó cliente o servidor possui um submódulo de prefetching que estima o tempo de prefetching para aquele cliente, utilizando estatísticas pertencentes àquele cliente. Esses parâmetros são obtidos através do histórico de faltas daquele cliente, do status da rede e da distância entre o cliente e o servidor. O objetivo do processo de prefetching é usar toda a informação coletada para determinar o momento certo de requisitar o dado.

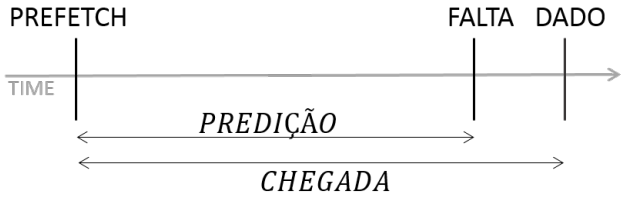
Usando um contador interno, o algoritmo de prefetching contabiliza o tempo até que chegue o momento previsto para que o dado seja enviado pela rede. Três possíveis cenários podem ser identificados dependendo do momento escolhido para o prefetching enviar a mensagem de busca. As Figuras 4.4, 4.5 e 4.6 ilustram estas situações.

Na Figura 4.4 a requisição de pré-busca do dado foi enviada tardiamente em relação

[1] A distância em uma NoC é a medida pela quantidade de vias que o pacote passa enquanto é roteado pela rede para ir da sua fonte para seu destino.

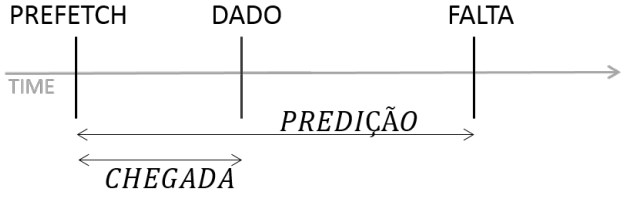
ao momento em que era necessário, consequentemente, o processador requisitou o dado antes que ele ficasse disponível na cache. Nesse caso, como o dado já foi requisitado pelo prefetching, a CPU vai aguardar até que ele esteja disponível e vai registrar um acerto na cache, porém o algoritmo de prefetching temporizado vai considerar um insucesso e voltará para o estado inicial para registrar a falta. Mesmo com a cache contabilizando um acerto, vai haver um aumento da penalidade em relação ao caso onde o dado já estaria disponível no momento em que a CPU o requisitasse.

Figura 4.4 Chegada do dado com atraso em relação à requisição do processador.



Outra situação que afeta o desempenho do prefetching é ilustrada na Figura 4.5. O intervalo entre a chegada do dado e a requisição do processador pode variar, mas caso esse intervalo seja muito grande, ou seja, o dado chegue muito antes do processador requisitá-lo, corre o risco dele substituir um outro dado que ainda vai ser requisitado pelo processador, causando poluição na cache. Embora o prefetching do dado em questão seja bem sucedido e vá reduzir a penalidade e taxa de falta da cache em relação àquele dado, ele pode influenciar negativamente o processamento de outro dado, pois irá substituí-lo e ocorrerá uma nova falta na cache.

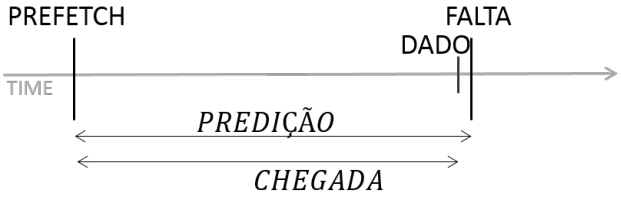
Figura 4.5 Chegada do dado muito adiantado em relação à requisição do processador.



Para ser considerado um prefetching bem sucedido que contribua em todos os sentidos com o desempenho do sistema (penalidade e taxa de faltas), o dado tem que estar disponível para o processador momentos antes de ser requisitado. O maior desafio para esse

caso é estimar o momento adequado para buscar o dado, considerando todos os fatores que podem influenciar na latência de acesso. A Figura 4.6 mostra essa situação.

Figura 4.6 Chegada do dado logo antes da requisição do processador.



Diante destes cenários fica clara a importância do resultado do cálculo da previsão de tempo ser o mais aproximado possível do tempo real para garantir um bom desempenho do prefetching. O cálculo de previsão proposto neste trabalho leva em consideração três fatores: o tempo de processamento do algoritmo, a latência de resolução da coerência de cache e o tempo de transferência do dado. Desses três apenas um é possível ser calculado precisamente (tempo de processamento) e, portanto, é previamente conhecido e os outros dois são não-determinísticos (a latência de resolução da coerência de cache e o tempo de transferência do dado), por isso, são estimados. De forma geral, o momento de realizar a busca do dado propriamente dita é calculada como mostrado na equação 4.1, onde $T_{prefetch}$ é o tempo no qual a mensagem de busca do dado é enviada, T_{falta} é o momento previsto para quando o processador irá requisitar o dado, ou seja, quando aconteceria a próxima falta de cache e δ é o tempo gasto pelo dado para chegar na cache a partir do momento que o algoritmo inicia o prefetching.

$$T_{prefetch} = T_{falta} - \delta \tag{4.1}$$

É importante notar que existe a possibilidade de, em alguns casos, o δ calculado ser maior do que o tempo previsto para ocorrer a falta, neste caso, tomou-se a decisão de realizar o prefetching mesmo assim. Esta decisão foi tomada para casos em que a agressividade do prefetching é maior que um, ou seja, que mais de um dado contíguo vai ser buscado, com o objetivo de aproveitar a localidade espacial - conceito que diz que após uma posição da memória ser acessada, existe uma maior probabilidade de que, em

um breve espaço de tempo, posições contíguas a esta também sejam acessadas.

Nas próximas subseções será detalhado como ocorre o cálculo de cada um dos fatores da equação 4.1.

4.2.1 Predição do tempo de falta

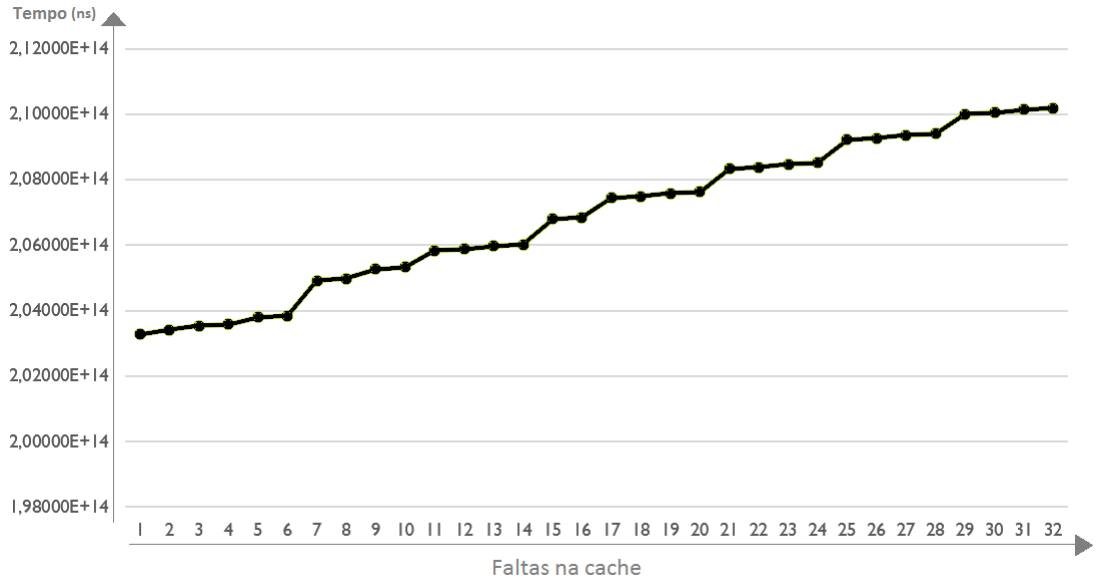
Para que seja possível que o dado esteja disponível na cache local do processador antes de ser requisitado é necessário prever quando a requisição irá acontecer. Como os eventos de ocorrência das faltas de cache podem ser identificados como informações discretas distribuídas no tempo, é possível classificar estes eventos como pontos de uma Série Temporal, conceito já explicado na Seção 2.3 do Capítulo 2.

O algoritmo de prefetching monitora as faltas de cache, fazendo o registro de quando ocorreram e qual foi o processador requisitante. Essas informações alimentam um módulo de previsão de tempo que implementa um método de previsão de séries temporais para estimar quando será a próxima ocorrência. Para o trabalho em questão foi escolhido o Método de Suavização Exponencial de Holt (Gardner, 1985). Porém o método é independente do controlador de prefetching e pode ser substituído por outro, desde que a interface de comunicação com o componente seja a mesma.

O Método de Holt é uma solução bem aceita e bastante utilizada em diversas aplicações, principalmente na indústria e mercado financeiro. Ela é direcionada para prever futuros pontos de uma série temporal que possui um comportamento linear sobreposto por flutuações aleatórias. A Figura 4.7 mostra um exemplo de uma série temporal que segue o padrão mencionado. O gráfico em questão é justamente uma série gerada a partir do histórico de faltas de cache em uma execução qualquer da plataforma.

Como explicado anteriormente, a série temporal é alimentada pelo histórico de ocorrências de falta de cache, sendo armazenado o tempo de ocorrência que vai compor a janela da série temporal. O parâmetro de tamanho de janela é livre (pode-se armazenar quantas faltas for conveniente), assim como o horizonte de previsão (pode-se prever um valor tão a frente quanto se deseje, embora para este trabalho apenas o próximo ponto da série interesse).

Figura 4.7 Ocorrências de faltas de cache definindo uma série temporal.



Por um lado, quanto maior for a janela mais precisa é a previsão, por outro lado, caso o controlador de prefetching tenha que esperar por muitas ocorrências para iniciar o fluxo do algoritmo, além de ser adicionado um overhead de processamento, o algoritmo vai demorar muito para ir para o estado de prefetching proativo.

O tamanho da janela é importante portanto, para a estabilização da previsão e, uma vez que a relação entre os pontos da série temporal e as faltas de cache é direta (pois os pontos da série são os momentos em que ocorreram o registro das falta), o algoritmo aguarda as primeiras 'x' faltas para prever o próximo ponto, onde 'x' é o tamanho da janela de previsão.

O algoritmo de Holt usa mais dois outros fatores para prever os pontos futuros da série temporal: uma componente de nível e uma de tendência. As equações 4.2, 4.3 e 4.4 são usadas para calcular a estimativa do ponto futuro. Onde F_{t+n} é o ponto a ser previsto, x_t é o ponto da série já conhecido, L_t é a componente de nível, T_t é a componente de tendência e n é um instante de observação, onde $n = 1, 2, ..., h$ e h é o horizonte de previsão.

$$F_{t+n} = L_t + nT_t \tag{4.2}$$

$$L_t = \alpha x_t + (1 - \alpha)(L_{t-1} + T_{t-1}) \tag{4.3}$$

$$T_t = \beta(L_t - L_{t-1}) + (1 - \beta)T_{t-1} \tag{4.4}$$

Pode-se observar que nas equações 4.3 e 4.4 tem-se duas constantes: α e β. O α ($0 < \alpha < 1$) é a constante de suavização do nível e β ($0 < \beta < 1$) é a constante de suavização da tendência. Esses valores podem ser variados para ajuste da previsão. Quando se aumenta o valor de α, o fator que multiplica a componente de nível diminui, ou seja, aquela componente passa a ter menos influência no cálculo da previsão. O mesmo ocorre com β em relação à componente de tendência.

Após computar a ocorrência das faltas de cache, o algoritmo pode prever quando ocorrerá a próxima falta e usar essa informação para realizar o prefetching, tentando disponibilizar o dado na cache antes do processador requisitá-lo.

4.2.2 Cálculo do delta

Como apresentado na equação 4.1, existe um intervalo de tempo δ a ser calculado, que deve ser descontado do momento previsto para o processador requisitar o dado, de forma que o início do prefetching se dê com antecedência suficiente para que o dado esteja disponível a tempo.

Em sistemas baseados em NoC onde os processadores são conectados ponto-a-ponto e a memória é distribuída entre os nós de processamento, um dos maiores desafios do prefetching é disponibilizar o dado em tempo na cache local do processador requisitante, onde em tempo significa antes da requisição do processador, mas não tão antes que comprometa outros blocos da memória. Caso o dado não chegue a tempo, além da penalidade adicionada no processador, a rede fica sobrecarregada com mensagens de menor prioridade, aumentando a latência de outros dados que estão trafegando.

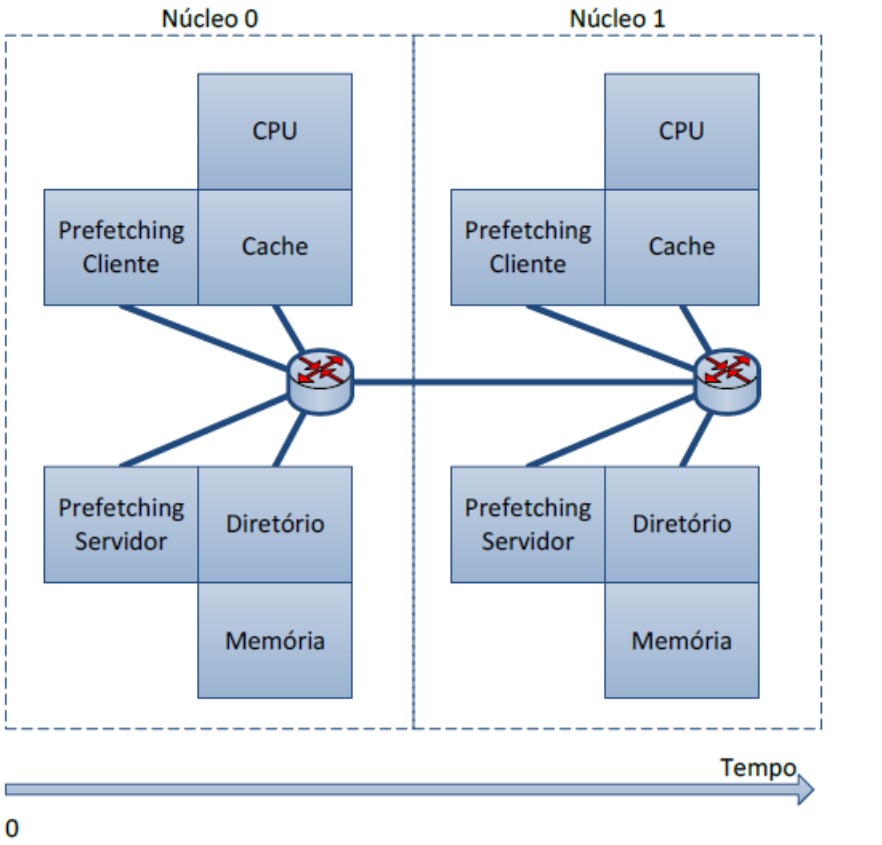
Para evitar esses efeitos negativos o algoritmo calcula uma aproximação da latência de transmissão do dado utilizando as estatísticas da rede, a distância entre os nós de

processamento servidor e o tempo de processamento. A equação 4.5 mostra o cálculo do fator δ. Seus fatores serão explicados em detalhes a seguir.

$$\delta = L_{coerencia} + T_{deslocamento} + T_{processamento} \tag{4.5}$$

O tempo de processamento tanto do subsistema de prefetching, quanto do algoritmo de Holt são facilmente mensuráveis por serem fixos, estes tempos são representados na equação 4.5 pelo fator $T_{processamento}$. Como o algoritmo se inicia no estado de espera de uma falta de cache e é a partir dela que é previsto quando aconteceria a próxima, todo o tempo de processamento tem que ser considerado como o tempo contado para o momento do dado ficar disponível. Por ser um fator determinístico, este é o menor dos desafios do cálculo do fator δ.

Figura 4.8 Arquitetura de um processador dual core hipotético.
Fonte: Aziz, 2014.



O tempo $L_{coerencia}$ é o tempo gasto para resolver a coerência de cache entre os nós de processamento. Ele depende do congestionamento da rede e do tempo que cada nó leva para processar e responder as mensagens de coerência. Embora o tempo de processamento das mensagens também seja fixo e mensurável, o tempo da troca das mensagens é não determinístico. Para se entender como ele é estimado é necessário entender como funciona a coerência da plataforma proposta por Aziz. Para isto, Aziz utilizou um processador dual core mostrado na Figura 4.8, com dois nós: 0 e 1, cada um com apenas um nível de cache. A imagem também mostra uma linha do tempo que será utilizada para contextualizar a coerência de cache da plataforma.

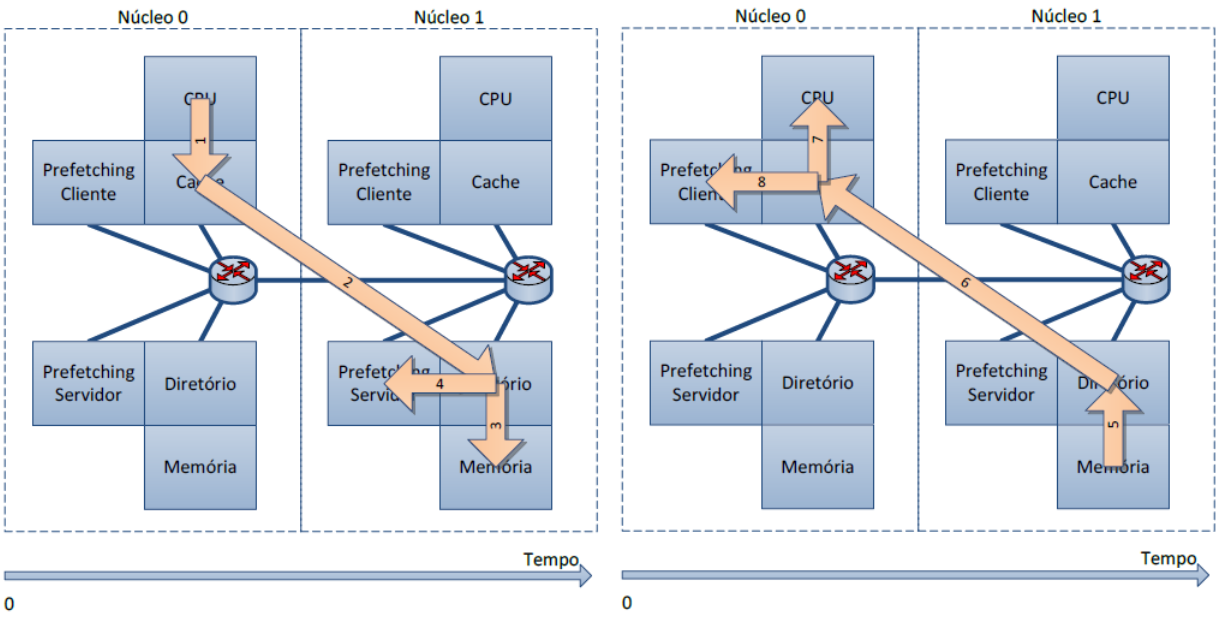
A seguir será dado um exemplo de como ocorre o funcionamento da coerência de cache e do prefetching na plataforma. Começando pela Figura 4.9, temos um exemplo de uma leitura da CPU 0 de um endereço hipotético que não está na cache local (1). Nesse caso, a cache solicita o dado ao diretório responsável por aquela faixa de endereços (2), no caso, é o diretório do nó 1. O diretório do nó 1 verifica que o dado está presente na memória e faz uma solicitação à mesma (3), além disso ele informa ao prefetching servidor que houve uma falta daquele endereço no núcleo 0 (4).

Fazendo o retorno da coerência, a memória responde ao diretório com o dado requisitado (5) e o diretório o encaminha para a cache requisitante (6). Por fim, o dado é repassado à CPU 0 (7) pela cache local e depois o prefetching cliente é informado do tempo gasto para realizar essa transação de coerência (8), esse tempo é a penalidade.

Quando o algoritmo de previsão de endereços do prefetching já tem informações de faltas de cache suficientes ele é capaz de calcular uma previsão do próximo endereço, assim, o controlador de prefetching pode começar a realizar requisições de prefetching para aquele núcleo, como mostrado na Figura 4.10.

É importante observar que cada controlador de prefetching guarda as informações de faltas, de penalidade e de endereços para cada um dos demais nós da plataforma, assim o cálculo de previsão de endereços e o controle de agressividade é realizado separadamente para cada nó cliente. O mesmo acontece com o subsistema de prefetching proposto neste trabalho, cada nó guarda estatísticas diferentes em relação a cada um dos outros nós, o

Figura 4.9 Exemplo de solicitação de leitura da CPU do núcleo 0 de um endereço hipotético.
Fonte: Aziz, 2014.



que resulta em uma previsão de tempo para cada.

Na Figura 4.10 as mensagens se iniciam no prefetching servidor. Como ele já possui informações suficientes, inicia o prefetching enviando uma mensagem ao prefetching cliente com o valor do endereço e da agressividade do prefetching (1). O prefetching cliente envia uma solicitação à cache do endereço do bloco previsto (2) e este inicia uma transação de coerência de (3) a (6) para buscar o dado na memória. A última mensagem (7) é referente à cache informando ao prefetching cliente que a requisição de prefetch foi concluída. Caso a agressividade seja maior do que 1 (um) endereço, o prefetching cliente faz outra solicitação à cache e esse processo se repete pelo número de endereços a ser solicitado.

A última das ações que fecha o ciclo de coerência de cache e prefetching é o envio de estatísticas por parte do prefetching cliente para os demais componentes de prefetching servidor dos outros núcleos. O objetivo de envio destas estatísticas é deixar atualizada a informação de penalidade para cada prefetching servidor, permitindo que o cálculo do fator δ seja mais preciso. Essas informações são enviadas em períodos constantes de tempo chamados de ciclos de amostragem.

Figura 4.10 Exemplo da execução de uma transação de prefetching em um processador dual core.

Fonte: Aziz, 2014.

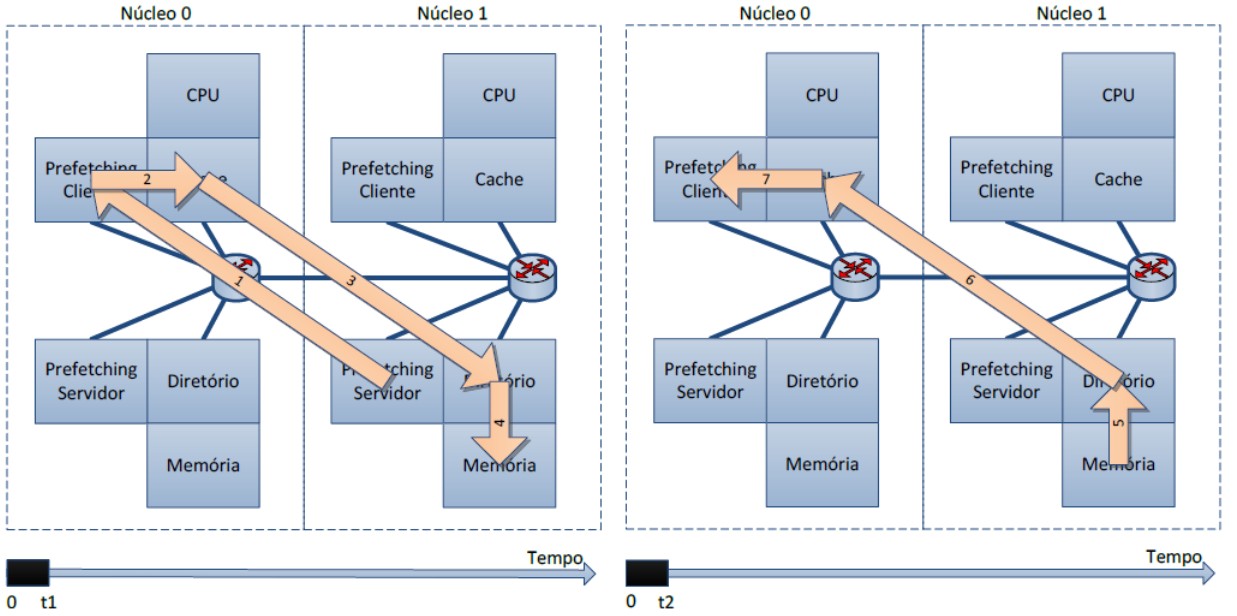
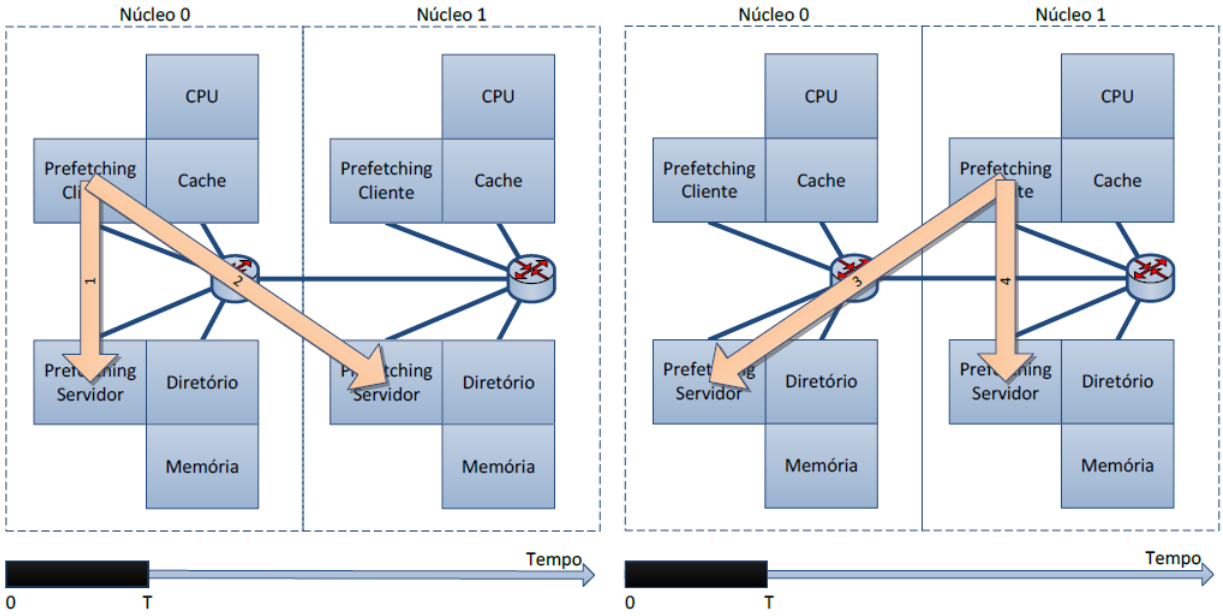


Figura 4.11 Exemplo de atualização das estatísticas dos controladores de prefetching.

Fonte: Aziz, 2014.



Assim, podemos definir o tempo $L_{coerencia}$ como o tempo gasto pela troca de mensagens entre os núcleos para resolver a coerência de cache, que é a penalidade média medida pelas

estatísticas do prefetching cliente; e $T_{deslocamento}$ o tempo gasto para o dado ser carregado na cache local do núcleo após o envio da mensagem de requisição de prefetching. O tempo que cada pacote leva para trafegar na rede é medido por contadores presentes nos próprios pacotes que são incrementados à medida que o pacote passa pelos roteadores da NoC.

Como estes tempos dependem do congestionamento da rede, não há como saber exatamente quanto tempo o próximo pacote vai levar para trafegar pela rede, logo eles são estimados utilizando a média dos valores medidos nos ciclos de amostragem, fazendo uma estimativa aproximada da situação da rede no último ciclo. Dessa forma, $L_{coerencia}$ consiste na média das penalidades e $T_{deslocamento}$ na média dos últimos tempos que as mensagens levaram para percorrer aquele caminho.

Para que um pacote percorra a rede da sua fonte para seu destino é necessário que os roteadores da rede implementem um algoritmo de roteamento que decida por que porta o pacote deve ser encaminhado após entrar no roteador. Neste trabalho, o algoritmo de roteamento utilizado foi o XY (Chawade, Gaikwad, and Patrikar, 2012). Para este algoritmo, pacotes com mesma fonte e destino sempre irão seguir o mesmo caminho, primeiramente percorrendo as vias horizontalmente, depois verticalmente. Uma das vantagens desse algoritmo é que a única coisa impedindo que dois pacotes que saiam da mesma origem levem o mesmo tempo para chegar ao mesmo destino é o congestionamento da rede.

Com todos os componentes calculados, tem-se uma estimativa do tempo em que deve ser solicitado o prefetching para que o dado chegue antes da ocorrência de outra falta na cache.

Neste capítulo foi apresentada a proposta do Algoritmo de Prefetching de dados Temporizado que teve como principal inspiração o trabalho desenvolvido por Aziz. No Capítulo 5 será explicado como foi feita a Implementação do trabalho em função da plataforma já existente.

CAPÍTULO 5

IMPLEMENTAÇÃO

A implementação da abordagem proposta foi feita utilizando a Plataforma Infinity como forma de continuação do trabalho de Aziz (Aziz, 2014). Nas próximas Seções serão discutidas as características da plataforma e também será explicado como foi implementado o algoritmo proposto neste trabalho.

5.1 PLATAFORMA INFINITY

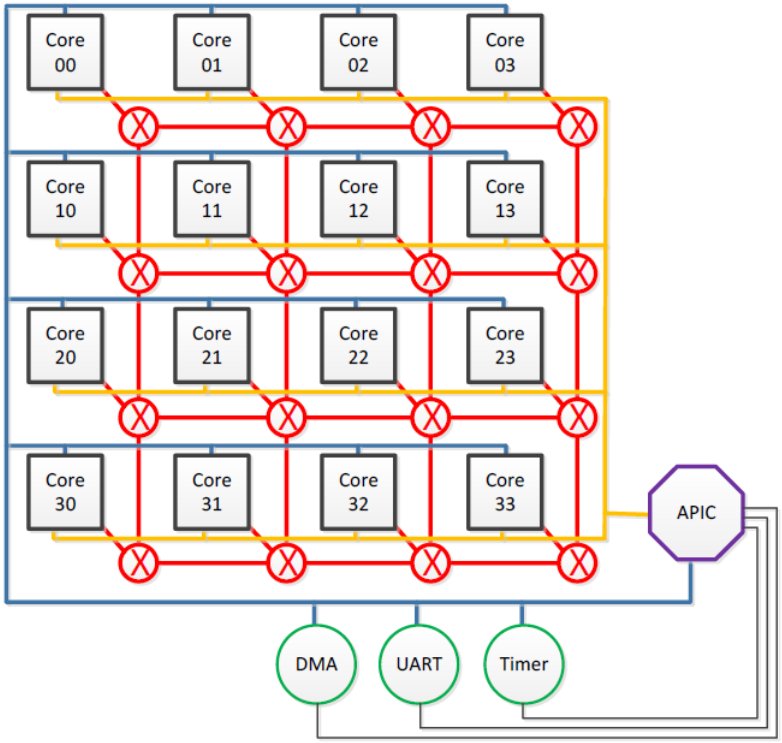
A plataforma "Infinity" foi desenvolvida na linguaguem SystemC com o objetivo de se tornar uma plataforma de validação para sistemas multiprocessadores. A idéia do nome "Infinity" foi pensado devido à escalabilidade da plataforma, característica que permite suporte a vários processadores. A arquitetura geral da plataforma pode ser observada na Figura 5.1, que mostra a plataforma em uma configuração de 16 nós de processamento.

Cada nó de processamento é composto de: processador, controlador de interrupção, barramento de periféricos e memória (cache local, diretório, cache de segundo nível e memória principal).

- A escolha da arquitetura do processador é independente para a Infinity, embora nesse trabalho tenha sido utilizada uma versão estendida do Sparc V8 ArchC (Azevedo, Bartholomeu, Araujo, Araujo, and E.Barros, 2005), com modificações nas instruções de LOAD e STORE. Essa modificação foi necessária, pois a versão do Sparc utilizada não possuía suporte a acessos a memória com tempo de acesso não uniforme (NUMA).

- Embora os nós de processamento sejam conectados pela NoC, a plataforma possui um barramento de interrupções e um barramento de periféricos dedicado, que

Figura 5.1 Visão geral da plataforma Infinity instanciada com 16 núcleos.
Fonte: Aziz, 2014.



atendem a todos os núcleos.

- A hierarquia de memória é composta por dois níveis de cache. A coerência de cache é baseada em diretório e a memória principal possui endereçamento global. O primeiro nível da cache é privado enquanto o segundo nível é compartilhado. A memória principal e a cache de segundo nível são distribuídas nos nós de processamento onde cada um possui um diretório que é responsável por controlar as requisições de memória e da cache de segundo nível.

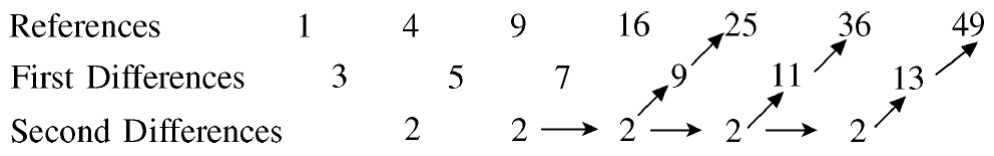
Para se comunicar com outros pontos da rede, cada nó possui uma interface local com um roteador. Os roteadores que fazem a interconexão dos nós da rede possuem um *buffer* para guardar os pacotes que esperam na fila em cada uma das interfaces de conexão: local, esquerda, direita, acima e abaixo. Cada roteador é conectado a outros quatro roteadores em uma estrutura de matriz e podem ser utilizados diferentes algoritmos de roteamento. Neste trabalho utiliza-se o algoritmo de roteamento XY.

Embora a plataforma possua diversos componentes, este trabalho foca no componente de prefetching. Na plataforma original o prefetching é baseado em faltas na cache, ou seja, o componente de prefetching só realiza as requisições de prefetching após a ocorrência de uma falta e caso o preditor de endereços tenha encontrado um padrão de acesso baseado no histórico das faltas de cache mais recentes. Neste trabalho é proposta uma nova abordagem de prefetching que permite que as requisições de prefetching sejam realizadas de forma pró-ativa, baseando-se na predição do tempo de futuras requisições dos processadores. Dessa forma, o componente de prefetching tenta disponibilizar o dado na cache antes mesmo do processador requisitá-lo. Assim, como na versão original da plataforma, também depende-se da previsão de endereço do bloco que, em ambos os casos, é realizado pelo algoritmo MLDT (Sun, Byna, and Chen, 2007).

A técnica MLDT utiliza a diferença dos endereços em um ou mais níveis para encontrar um padrão de acesso. Ela calcula a diferença dos endereços e, caso não encontre um padrão, calcula a diferença dos valores encontrados, e continua dessa forma recursivamente até encontrar uma diferença que se repita. Quando a tabela encontra um padrão ela retorna da recursão e utiliza o valor encontrado para calcular o próximo endereço previsto. A Figura 5.2 mostra um exemplo da aplicação da técnica MLDT em uma sequência de endereços, na qual foi encontrado o padrão no segundo nível de diferenças.

Figura 5.2 Exemplo de funcionamento da tabela MLDT.

Fonte: Sun, 2007.



Para a plataforma Infinity foi utilizada uma MLDT de dois níveis e histórico de 16 endereços. Ela recebe os endereços que provocaram uma falta na cache e prevê o próximo endereço de bloco para o componente realizar o prefetching.

Como apresentado no Capítulo 4, além do módulo de previsão de endereços, o componente de prefetching ainda possui o módulo de controle de agressividade e o analisador de pacotes. O analisador de pacotes é responsável por identificar as transações referentes às

faltas de blocos na cache e repassar os endereços faltantes para o módulo de previsão de endereço. O módulo de controle de agressividade foi a principal contribuição do trabalho de Aziz (Aziz, 2014) e é responsável por ajustar a agressividade do prefetching analisando a penalidade do processador, ou seja, escolhe quantos dados serão pré-buscados ao realizar o prefetching a partir das estatísticas de penalidade. A plataforma também permite que a agressividade do prefetching seja fixa. Neste trabalho foram exploradas as duas situações: realizando prefetching com agressividade fixa e utilizando o controle de agressividade proposto por Aziz.

5.2 IMPLEMENTAÇÃO DO ALGORITMO DE PREFETCHING

Nesta seção serão apresentados detalhes da implementação do algoritmo de prefetching que foi integrado à plataforma. Em adição aos módulos do componente do prefetching mostrados na Figura 4.2 foi proposto neste trabalho um algoritmo de controle de prefetching temporizado, que é baseado em predição de tempo.

O algoritmo de prefetching proposto funciona como um subsistema que assume o controle do prefetching quando o módulo de previsão de endereços estabiliza e encontra um padrão de acesso. O papel principal do algoritmo é enviar requisições de prefetching de tempos em tempos, antecipando a necessidade do processador. Para tal, ele realiza uma estimativa de tempo de quando o processador irá requisitar o dado. O cálculo de estimativa de tempo foi explicado previamente no Capítulo 4, na Seção 4.2.

Para implementação do módulo de prefetching temporal foi necessária a modificação do fluxo de prefetching. Originalmente, quando ocorria uma falta de cache e a MLDT encontrava o próximo endereço de bloco a ser buscado, era iniciado o processo de requisição do prefetching e, após a busca dos endereços previstos (pode ser mais de um dependendo da agressividade) o sistema de prefetching voltava a aguardar a ocorrência de outra falta na cache.

Já para este trabalho, após a busca dos endereços previstos o módulo de controle temporizado assume as operações de prefetching até que ocorra uma nova falta na cache. Enquanto o subsistema de controle temporizado atua no prefetching pode ocorrer

nenhuma, uma ou mais requisições, dependendo do momento em que volte a ocorrer uma falta de cache.

Para implementação do subsistema foram desenvolvidos os seguintes módulos:

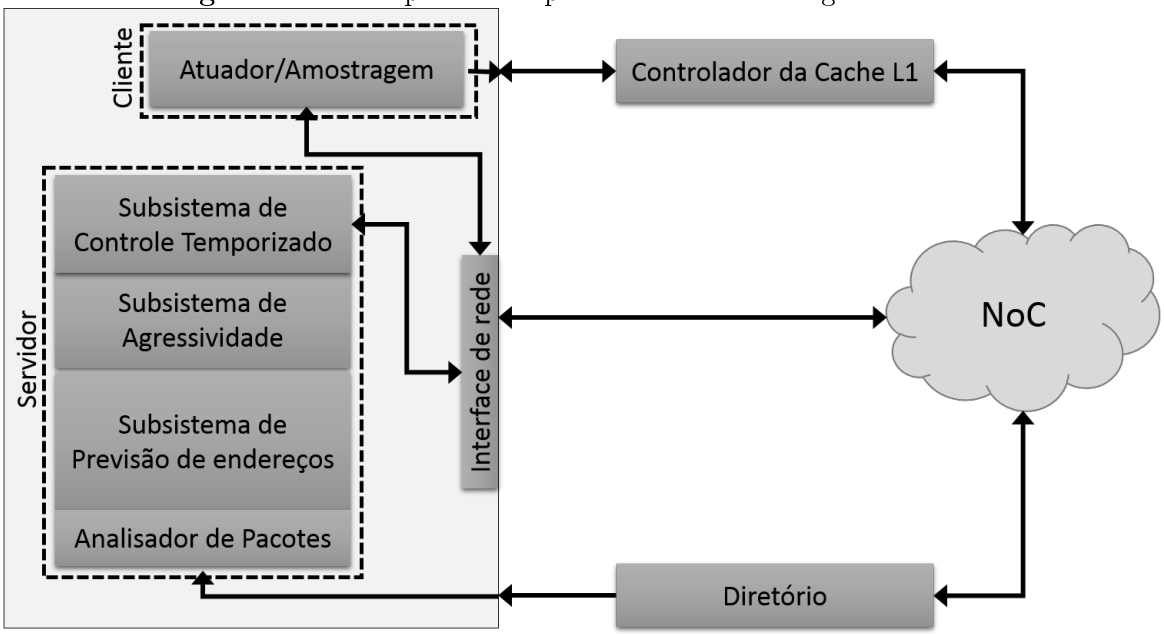
- Módulo de previsão de série temporal - módulo que implementa o algoritmo de previsão de séries temporais (Método de suavização exponencial de Holt) para prever quando irá ocorrer a próxima falta na cache.
- Módulo de controle de prefetching temporizado - realiza o controle das requisições de prefetching e só é interrompido quando ocorre uma falta na cache.
- Módulo de cálculo do δ - utiliza as estatísticas coletadas nos ciclos de amostragem para estimar o tempo que o dado levará para ficar disponível para o processador após a requisição de prefetching.

Relembrando a noção de prefetching cliente-servidor apresentada no Capítulo 4, um prefetching servidor tem que atender a demanda de todos os outros prefetching cliente. Sendo assim, ele precisa ter previsões de endereço diferentes para cada cliente, ou seja, uma instância do subsistema de previsão de endereços para cada. O mesmo ocorre com o subsistema de controle temporizado, para cada cliente tem-se uma previsão diferente de tempo e controladores diferentes para requisitar os prefetchings.

Logo, temos um módulo de previsão de falta na cache, um controlador e um módulo do cálculo do δ para cada núcleo dentro de cada componente de prefetching da plataforma. A Figura 5.3 detalha a arquitetura do componente de prefetching proposto neste trabalho. Pode-se observar que foi adicionado o subsistema de controle temporizado que é justamente o algoritmo de predição de tempo para realização de prefetching.

Com a adição desse novo subsistema tem-se um novo módulo controlando as requisições de prefetching. A Figura 5.4 mostra a estrutura interna do subsistema. A seta pontilhada ilustrada na figura representa um sinal de controle que é enviado ao módulo de controle do prefetching quando o subsistema de previsão de endereços estabiliza, passando o controle das requisições de prefetching para o controle temporizado.

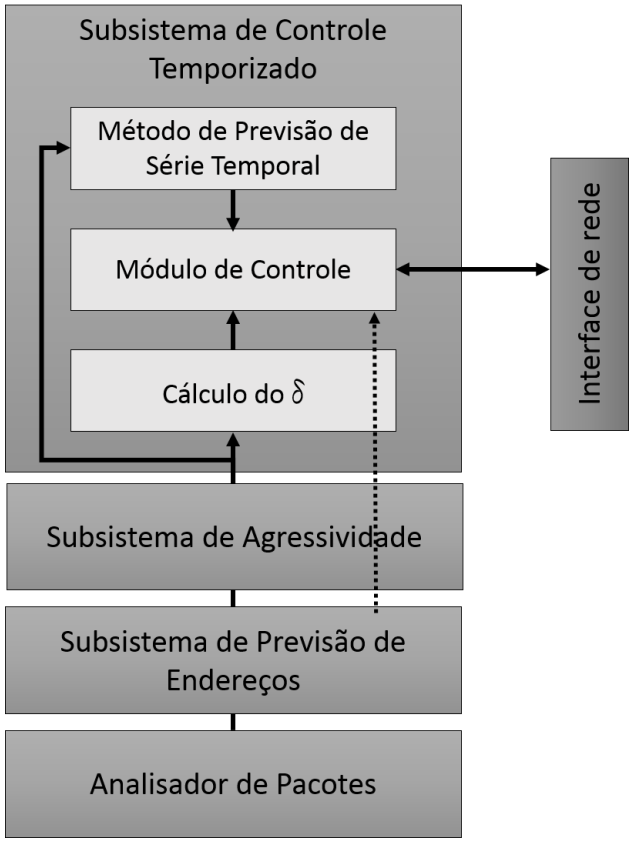
Figura 5.3 Exemplo do componente de Prefetching modificado.



Também é possível observar na Figura 5.3 que as informações da falta da cache recebidas pelo Analisador de pacotes é repassada para os módulos de previsão de séries temporais e de cálculo do δ. Após o processamento dessas informações estes módulos disponibilizam para o controle temporizado de prefetching as informações de tempo necessárias para que ele realize o controle das requisições de prefetching. Para tal, o controlador possui um contador interno que ao atingir o tempo previsto para realizar a requisição de prefetching (calculado pela diferença do tempo previsto para a próxima falta de cache com o δ) envia uma mensagem de prefetching e é zerado para recomeçar um novo ciclo de prefetching. Este processo só é interrompido caso haja uma falta na cache. Neste caso, o componente de prefetching volta a registrar as informações da falta na cache e realiza um prefetching baseado em evento. Caso o subsistema de previsão de endereços consiga calcular o próximo endereço de bloco baseado nessa nova informação de falta na cache, o processo de controle temporizado recomeça, caso contrário, o componente de prefetching fica aguardando uma nova falta na cache. A máquina de estados 4.3 do módulo de controle foi apresentada na seção 4.1.

O diagrama apresentado na Figura 5.5 mostra o fluxo de dados do algoritmo. Cada transação representa o conjunto de dados que é utilizado para alimentar um processo da

Figura 5.4 Estrutura interna do subsistema de Controle de Prefetching Temporizado.



plataforma.

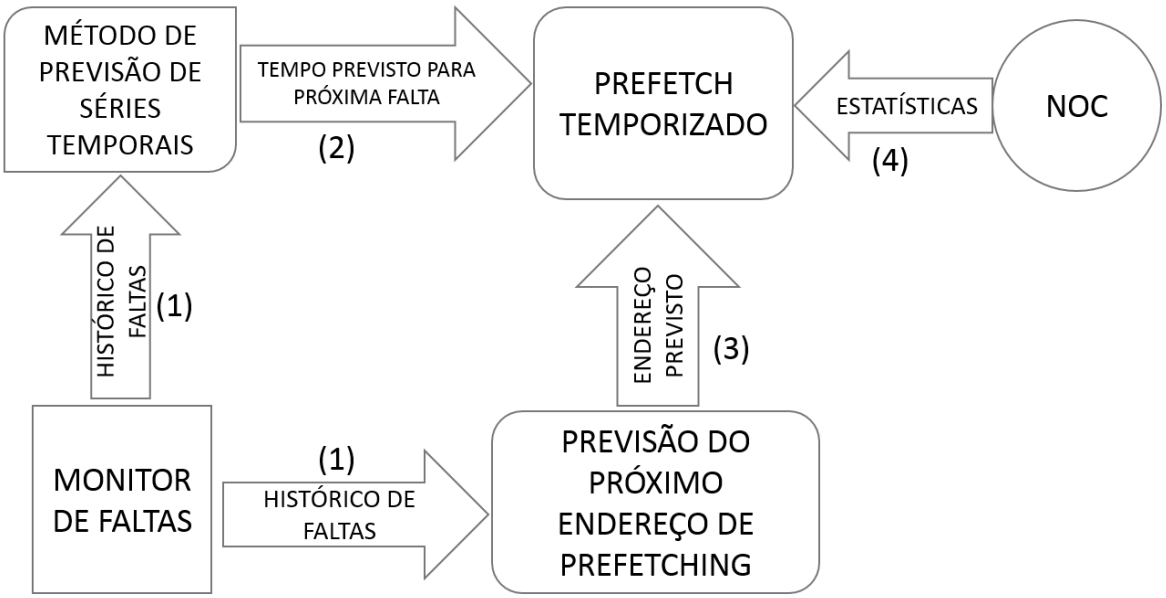
1. O histórico de faltas de cache é utilizado para alimentar o Método de previsão de séries temporais que vai usar os tempos de ocorrência das faltas para prever quando aconteceria a próxima falta; e o algoritmo de previsão de endereços do prefetching que utiliza os endereços das faltas para prever o próximo bloco a ser buscado.

2. O resultado do processamento do método de previsão temporal é o tempo previsto para a próxima falta da cache. Este é utilizado pelo Algoritmo de prefetching temporizado no cálculo de previsão de tempo para decidir quando será iniciado o prefetching.

3. Como resultado de processamento do algoritmo de previsão de endereços tem-se o próximo bloco a ser buscado que também é utilizado pelo prefetching para saber o endereço a ser buscado.

4. Finalmente, as estatísticas levantadas nos ciclos de amostragem de penalidade dos processadores e de tempo de transferência dos dados pela rede são providas pelo prefetching cliente de outros núcleos da NoC e utilizadas pelo algoritmo de prefetching também no cálculo de previsão do tempo.

Figura 5.5 Diagrama de dados do Algoritmo de Prefetching Temporizado.



O Algoritmo 1 apresenta o pseudo-código do algoritmo de prefetching baseado em predição de tempo proposto neste trabalho.

5.3 PARÂMETROS DE PREVISÃO DA SÉRIE TEMPORAL

Como mencionado no Capítulo 4, para implementação do método de previsão de séries temporais foi escolhido o algoritmo de Holt. Este é um método de suavização exponencial apropriado para séries temporais que seguem uma função linear, sobreposta por variações aleatórias. Este mesmo comportamento é observado nas amostras de tempo das faltas de cache, como mostrado na Figura 4.7 no Capítulo 4.

O algoritmo de Holt foi implementado em SystemC e recebe como entrada as ocorrências de falta de cache e as usa para compor a série temporal. A série temporal foi

Algoritmo 1: Funcionamento do algoritmo de prefetching baseado em predição de tempo.

```
1:  while True do
2:    Calcula δ atualizado.
3:    if Ocorreu uma falta na cache. then
4:      if O próximo endereço a ser pré-buscado foi calculado. then
5:        Faz prefetching baseado em evento.
6:        if Existe previsão de quando vai ocorrer a próxima falta na cache. then
7:          T_falta = Previsão de em quanto tempo vai ocorrer a próxima falta na
            cache.
8:          Espera T_falta − δ.
9:          while Não ocorrer nova falta. do
10:           Atualiza endereço do novo dado a ser pré-buscado.
11:           Faz prefetching baseado em tempo.
12:           Espera T_falta.
13:         end while
14:       end if
15:     else
16:       Armazena informações da falta.
17:     end if
18:   end if
19: end while
```

implementada como um buffer do tamanho da janela de previsão que vai sendo atualizado com os tempos de ocorrências das faltas a medida em que novas faltas de cache vão ocorrendo, assim, os registros mais antigos vão sendo descartados. Essa atualização constante das informações de falta na cache faz sentido porque o padrão de acesso à dados das aplicações pode mudar ao longo do tempo de execução da aplicação. O resultado fornecido pelo algoritmo de Holt é a previsão do próximo ponto da série temporal estimando quando irá acontecer a próxima falta de cache. Seus parâmetros são o tamanho da janela de previsão, α, β e o horizonte de previsão.

Para o tamanho da janela foram testados os valores: 4, 8, 16, 32, 48, 64 e 128; para garantir os diferentes padrões de acesso das aplicações fossem explorados. Os experimentos realizados mostram a variação dos resultados para todos esses casos como será mostrado no Capítulo 6. Esse parâmetro não foi fixado, pois sua variação influencia diretamente no desempenho do algoritmo, uma vez que um tamanho pequeno de janela pode minimizar

a penalidade média dos processadores para uma dada aplicação e um tamanho maior pode minimizar para uma aplicação diferente, dependendo do padrão de acesso à dados de cada aplicação.

O valor escolhido para o horizonte de previsão foi o tamanho da janela de previsão mais um, pois o interesse do trabalho é calcular, a partir do histórico de faltas na cache, quando vai ocorrer a próxima falta na cache. Então é suficiente que o valor a ser previsto seja o dado seguinte à janela de previsão, que é justamente a próxima (suposta) ocorrência de uma falta na cache.

Os parâmetros α e β devem ser escolhidos para minimizar ou maximizar a influência das componentes de nível e de tendência na previsão. Neste trabalho, foi selecionado o valor de 0,4 tanto para α quanto para β.

O Algoritmo 2 apresenta o pseudo-código do método de suavização exponencial de Holt, utilizado para previsão de séries temporais.

Algoritmo 2: Funcionamento do algoritmo de Holt para previsão de séries temporais.

Require: α e β
Ensure: T_{falta}
1: **while** Número de elementos da série for menor que o tamanho da janela. **do**
2: Atualiza a janela com nova ocorrência de falta na cache.
3: **end while**
4: $L_1 = 0$
5: $T_1 = 0$
6: **for** $t = 2$ **to** $t =$ janela de previsão **do**
7: $L_t = \alpha x_t + (1 - \alpha)(L_{t-1} + T_{t-1})$
8: $T_t = \beta(L_t - L_{t-1}) + (1 - \beta)T_{t-1}$
9: **end for**
10: $T_{falta} = L_t + nT_t$

CAPÍTULO 6

EXPERIMENTOS E RESULTADOS

Este Capítulo descreve os experimentos realizados para validação do algoritmo proposto e os resultados obtidos a partir destes experimentos.

Em primeiro lugar, temos a plataforma "Infinity" que por ser um ambiente de estudo pode ser configurada de diversas formas, porém uma exploração da Infinity em todas as suas variações se torna inviável em tempo de projeto. Afim de solucionar este problema, foi criado um benchmark sintético simplificado para explorar a configuração da plataforma que se adequa melhor ao trabalho proposto. A configuração escolhida é apresentada na Seção 6.1 e os testes realizados para a escolha das configurações da plataforma estão anexados a este trabalho no Apêndice 7. Após a escolha da plataforma, íniciaram-se os experimentos utilizando benchmarks comerciais, apresentados neste capítulo na Seção 6.2.

Os experimentos compreenderam três abordagens principais: NO-PREFETCHING, EVENT e TIMED. A abordagem NO-PREFETCHING, como o próprio nome já indica, não possui influência do prefetching, então quando uma falta de cache ocorre o dado é buscado na memória e carregado na cache, mas os próximos endereços que seriam previstos não são requisitados.

O prefetching aqui referenciado como EVENT é a abordagem baseada em faltas na cache onde, quando ocorre uma falta na cache, o bloco é buscado na memória e além disso os 'n' próximos endereços previstos são requisitados, onde 'n' é a agressividade do prefetching. Para estes experimentos a agressividade do prefetching foi variada entre 4, 8 e 16.

Por fim, a abordagem TIMED é a nova abordagem desenvolvida neste trabalho que, diferentemente da abordagem EVENT, não é baseada em eventos, mas em previsão de

tempo. O funcionamento desta técnica ocorre da seguinte forma: após um número de faltas na cache (tamanho da janela de previsão), necessárias para estabilizar a previsão de séries temporais e o subsistema de previsão de endereços, o algoritmo realiza requisições de prefetching de forma pró-ativa objetivando antecipar os dados que serão requisitados pelos processadores. Para esta abordagem, além de variar a agressividade, foi variada a janela de previsão da série temporal utilizada pelo método de suavização exponencial de Holt. A janela assumiu os valores de 4, 8, 16, 32, 48, 64 e 128 pontos.

Para avaliar os resultados é preciso que as variações dos algoritmos EVENT e TIMED sejam comparadas com a mesma agressividade visando manter a consistência da comparação. Assim, foram gerados gráficos para cada aplicação e cada agressividade, variando no gráfico a abordagem (NO-PREFETCHING, EVENT e TIMED) e o tamanho da janela.

6.1 CONFIGURAÇÃO DA PLATAFORMA INFINITY

Como mencionado anteriormente, para a escolha da configuração da plataforma foi criado um benchmark sintético que foi executado com diferentes configurações da plataforma e foi selecionada a configuração que melhor responde às alterações propostas pelo algoritmo de prefetching.

A aplicação sintética realiza o envio de arrays de tamanho fixo de uma cache privada para a outra, sempre armazenados no mesmo endereço. Esses arrays sofrem modificações nos bancos de memória locais e depois são encaminhados para a cache seguinte. Os resultados deste experimento estão no Apêndice 7. Ao final dos testes foi selecionada a plataforma que possuiu menor penalidade para a abordagem TIMED quando comparados com EVENT ou NO-PREFETCHING. A configuração da plataforma selecionada se encontra no Quadro 6.1.

Outra especificação importante sobre a plataforma é que a Infinity não possui nenhum sistema operacional multitarefa portado para ela, o que limitou a quantidade de aplicações selecionadas para validar trabalho, pois algumas aplicações não puderam ser portadas para a plataforma. A seguir, serão demonstrados os resultados dos experimentos

Quadro 6.1 Configuração da Plataforma utilizada nos experimentos.

Processador	SPARC V8 Estendido, 1GHz, 32 bits, IPC=1
Memória	64MB de RAM; Endereçamento Global
Cache L1	Privada, 4 palavras por bloco, mapeamento direto, 1 KB
Cache L2	64KB por banco; Compartilhada, 8 palavras por bloco, 16-way
NoC	2D Mesh (4x4), 1GB/s e full duplex, buffer de tamanho 10, roteamento XY

realizados e as aplicações utilizadas.

6.2 BENCHMARKS

Para validação do algoritmo de prefetching de dados baseado em predição de tempo proposto neste trabalho, algumas aplicações provenientes de Benchmarks usados amplamente foram simuladas na plataforma utilizando a configuração descrita na Tabela 6.1. As cinco aplicações multithread utilizadas pertencem aos benchmarks PARSEC e SPLASH-2. As aplicações foram divididas em threads onde cada thread foi alocada a um núcleo da rede. As aplicações selecionadas foram: BASICMATH, SHA, LU, SUSAN CORNERS e RADIX.

A Seção 6.3 a seguir descreve os resultados obtidos nas simulações.

6.3 RESULTADOS

Nesta Seção serão detalhadas as estatísticas que foram medidas nos experimentos, o que elas representam e porque elas são importantes para avaliar o desempenho do algoritmo proposto. Além disso, serão apresentados os gráficos dos resultados dos experimentos em relação à cada uma dessas estatísticas.

Os experimentos englobam todos os benchmarks comerciais citados na Seção 6.2, variando as abordagens de prefetching (NO-PREFETCHING, EVENT e TIMED), a agressividade do prefetching (4, 8 e 16) e o tamanho da janela de previsão (4, 8, 16, 32,

48, 64 e 128).

6.3.1 Estatísticas Importantes

Nos experimentos realizados os benchmarks comerciais foram avaliados na plataforma em relação a cinco fatores: penalidade média dos processadores, o número de transações de rede executadas, a poluição de cache gerada pelo prefetching, a precisão do prefetching e o número de transações de prefetching com atraso.

A penalidade foi explicada na Seção 4.2 onde define-se penalidade como o tempo que o processador fica aguardando a coerência de cache ser resolvida e o dado ser copiado na cache local.

O número de transações na rede representa o número de pacotes que foram enviados para a rede. Este número é importante porque indica se o número de transações de controle e de prefetching aumentou ou diminuiu em relação às outras abordagens. O aumento exagerado do número de transações não é um bom indicativo, pois um grande número de pacotes trafegando significa mais congestionamento na rede o que pode aumentar a latência de transferência de dados.

A poluição de cache ocorre quando acontece uma falta na cache de um bloco que foi substituído por uma transação de prefetching não utilizada pelo processador. Então, o percentual de poluição gerada pelo prefetching é a razão entre o número de faltas provocadas pela poluição e o número total de faltas na cache, como mostrado na Equação 6.1.

$$\text{Percentual de Poluição} = \frac{\text{Número de faltas por causa da Poluição}}{\text{Número total de faltas}} \tag{6.1}$$

A precisão do prefetching é dada pelo número de blocos do prefetching carregados na cache e que foram utilizadas pelo processador. Sendo assim, o percentual de precisão do prefetching é a razão entre o número de blocos utilizados pelo processador que foram requisitados pelo prefetching e o número total de transações completas. A Equação 6.2 descreve este percentual.

$$\text{Percentual de Precisão} = \frac{\text{Número de blocos de prefetching utilizados pelo processador}}{\text{Número total de transações completas}} \tag{6.2}$$

Por fim, tem-se o percentual de atraso das transações de prefetching. Uma transação é considerada atrasada quando ela chega após um pedido do processador pelo mesmo bloco ou quando o bloco que ele irá utilizar está sendo utilizado por outra transação de coerência de cache. Assim, calcula-se o percentual de atraso como mostrado na Equação 6.3.

$$\text{Percentual de Atrasos} = \frac{\text{Número de transações atrasadas}}{\text{Número total de transações}} \tag{6.3}$$

Em seguida são apresentados e analisados os resultados obtidos nos experimentos para as métricas apresentadas.

6.3.2 Análise dos Resultados

Para facilitar a compreensão dos gráficos, todos os valores de penalidade, taxa de faltas e número de transações foram normalizados em relação aos valores obtidos no caso de NO-PREFETCHING. As demais métricas do prefetching: precisão do prefetching, atraso nas transações de prefetching e poluição gerada na cache são expressas em valores percentuais.

O Gráfico 6.1 apresenta os resultados obtidos com o algoritmo proposto neste trabalho em relação à penalidade média dos processadores, quando executando a aplicação Radix para agressividade fixa em 4, 8 e 16 endereços. Pode-se observar que para esta aplicação o prefetching pode afetar negativamente a penalidade dos processadores, pois os resultados para a abordagem NO-PREFETCHING para agressividade 4 foram melhores em relação à abordagem de prefetching baseada em evento (EVENT) em 40,65%. Por outro lado, a abordagem TIMED utilizando a janela de previsão com tamanho 48 obteve sucesso em reduzir a penalidade em relação à abordagem NO-PREFETCHING em 0,55% e EVENT em 29,29%. A Tabela 6.1 descreve os resultados da aplicação Radix onde os números ne-

gativos representam piora na penalidade, ou seja, uma penalidade média por processador maior, e os números positivos representam melhora na penalidade, ou seja, diminuição da mesma.

Gráfico 6.1 Penalidade média dos processadores para a aplicação Radix.

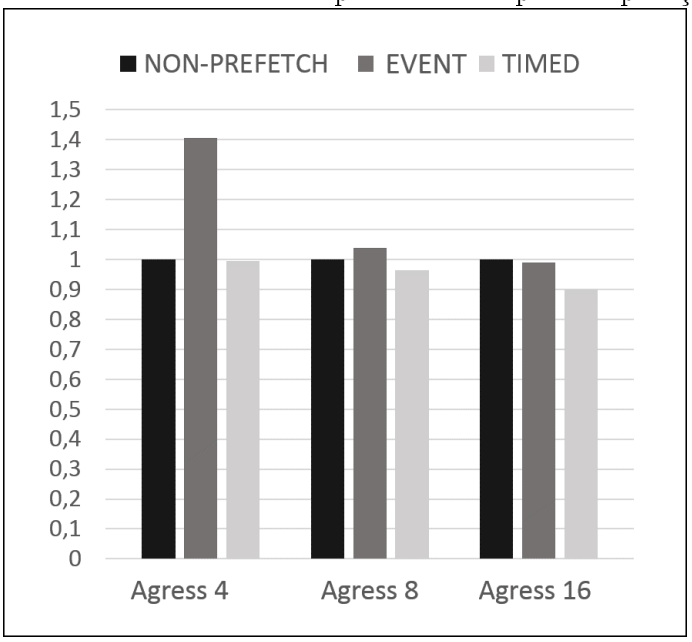


Tabela 6.1 Tabela com resultados da penalidade para aplicação Radix.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	-40,65%	-3,97%	0,97%
Timed x No-Prefetching	0,55%	3,51%	10,11%
Timed x Event	29,29%	7,20%	9,23%

O Gráfico 6.2 ilustra o número de transações na rede para a aplicação Radix também utilizando o mecanismo de prefetching com agressividade fixa em 4, 8 e 16. A Tabela 6.2 demonstra os números que representam o aumento ou diminuição do número de transações. Pode-se observar pela tabela que tanto a abordagem EVENT quanto a TIMED resultaram em um aumento no número de transações em relação à abordagem NO-PREFETCHING. Ainda assim, a abordagem TIMED obteve redução do número de transações em relação à abordagem EVENT, sendo mais significativa para agressividade 4, onde o número de transações foi reduzido em 34,25%.

Gráfico 6.2 Número de transações na rede para a aplicação Radix.

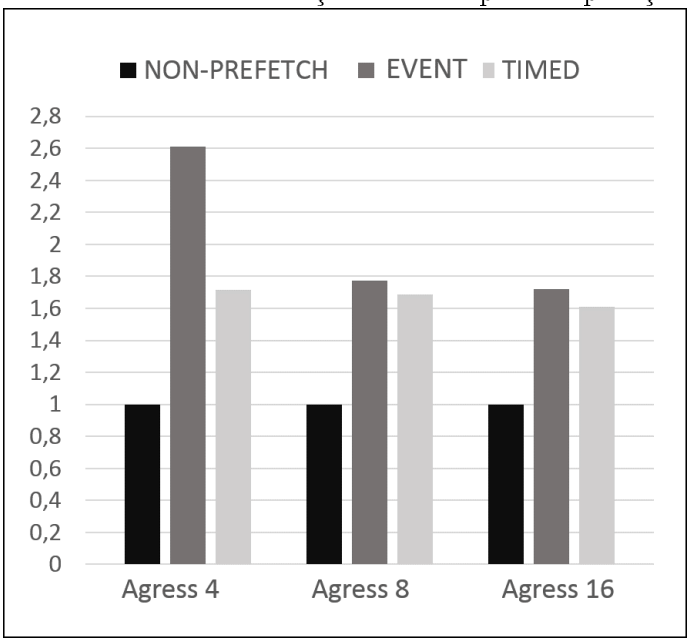


Tabela 6.2 Tabela com resultados do número de transações na para aplicação Radix.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	161,23%	77,37%	71,89%
Timed x No-Prefetching	71,76%	68,74%	61,04%
Timed x Event	-34,25%	-4,87%	-6,31%

No Capítulo 5 foi mencionada a influência que o tamanho da janela de previsão possui no desempenho do mecanismo de prefetching proposto neste trabalho. Por isso, os gráficos de Poluição, Precisão e de Transações atrasadas do prefetching apresentam não só a comparação com a abordagem EVENT, mas também a comparação considerando diferentes tamanhos de janela da abordagem TIMED. O Gráfico 6.3 apresenta a precisão do prefetching e a Tabela 6.3 detalha os valores percentuais obtidos em comparação com a abordagem EVENT.

O Gráfico 6.4 e a Tabela 6.4 apresentam o percentual de transações de prefetching atrasadas para a aplicação Radix em comparação com a abordagem EVENT. Pode-se observar que o tamanho da janela de previsão utilizado para estimar quando ocorrerá a próxima falta na cache influencia diretamente no percentual de transações atrasadas. Isto porque a precisão da estimativa é um dos fatores determinantes para o momento de

Gráfico 6.3 Percentual de precisão do prefetching para a aplicação Radix.

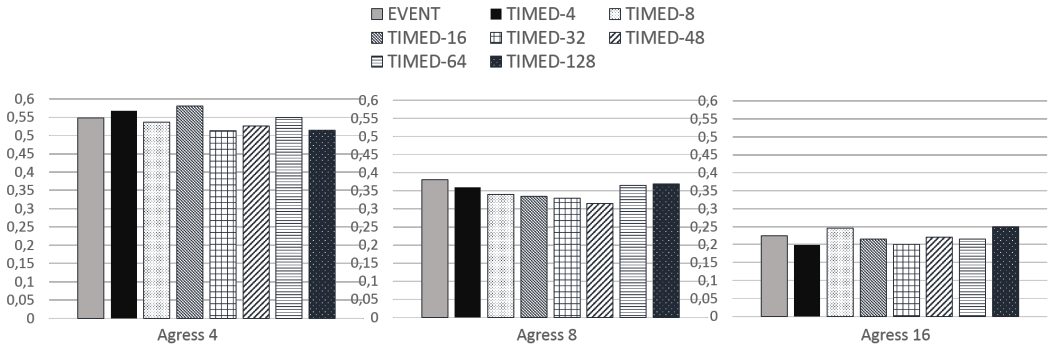
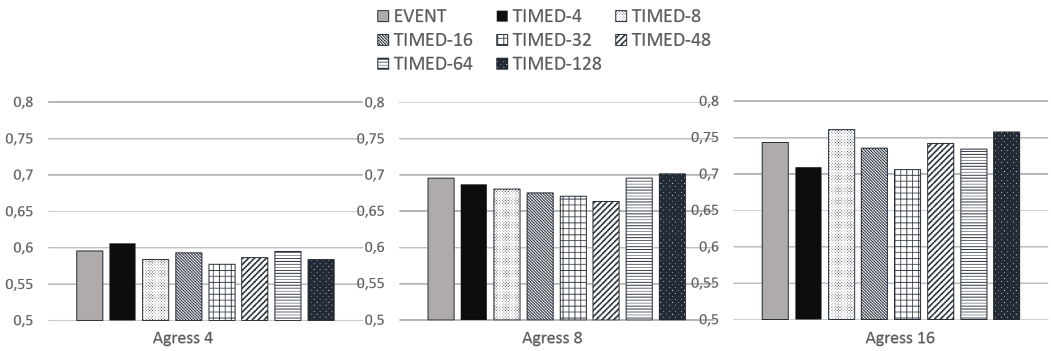


Tabela 6.3 Tabela com resultados do percentual de precisão do prefetching para a aplicação Radix.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	54,87%	38,05%	22,48%
Timed (janela de previsão = 4)	56,7%	35,95%	19,94%
Timed (janela de previsão = 8)	53,59%	33,95%	24,53%
Timed (janela de previsão = 16)	58,12%	33,48%	21,50%
Timed (janela de previsão = 32)	51,36%	32,92%	20,02%
Timed (janela de previsão = 48)	52,59%	31,49%	21,99%
Timed (janela de previsão = 64)	54,89%	36,44%	21,58%
Timed (janela de previsão = 128)	51,37%	36,96%	24,92%

enviar a requisição de prefectching, caso ela erre, mesmo que seja por pouco, a transação de prefetching chegará atrasada no seu destino.

Gráfico 6.4 Percentual de transações de prefetching atrasadas para a aplicação Radix.



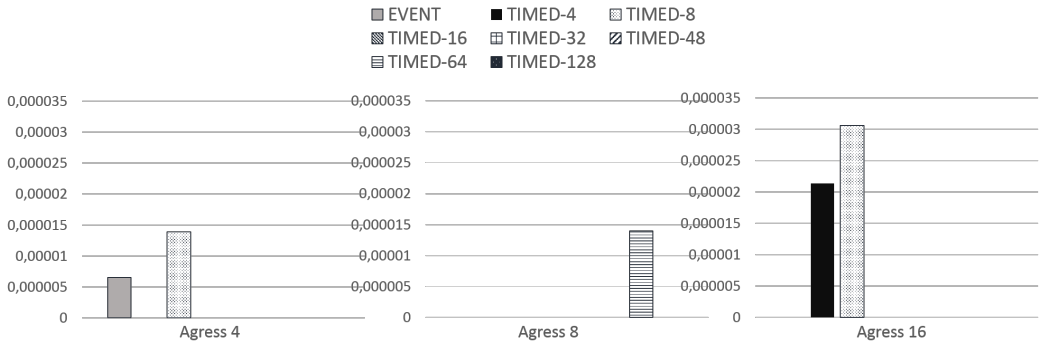
Finalmente, tem-se o resultado da poluição de cache causada pelo prefetching para a aplicação Radix. O Gráfico 6.5 e a Tabela 6.5 apresentam estes números. Pode-se observar que alguns valores estão zerados, isto pode ocorrer quando o número de transações de

Tabela 6.4 Tabela com percentual de transações de prefetching atrasadas para a aplicação Radix.

Abordagens	Agress 4	Agress 8	Agress 16
Event	59,54%	69,53%	74,33%
Timed (janela de previsão = 4)	60,60%	68,74%	70,90%
Timed (janela de previsão = 8)	58,37%	68,04%	76,07%
Timed (janela de previsão = 16)	59,27%	67,56%	73,55%
Timed (janela de previsão = 32)	57,71%	67,08%	70,56%
Timed (janela de previsão = 48)	58,66%	66,35%	74,20%
Timed (janela de previsão = 64)	59,48%	69,56%	73,38%
Timed (janela de previsão = 128)	58,41%	70,17%	75,73%

prefetching não é muito alto e quando os dados pré-buscados estão ficando disponíveis na cache pouco antes da requisição do processador. Embora para alguns tamanhos de janela exista uma taxa de poluição, os valores são tão baixos que não chegam a afetar o desempenho do prefetching.

Gráfico 6.5 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Radix.

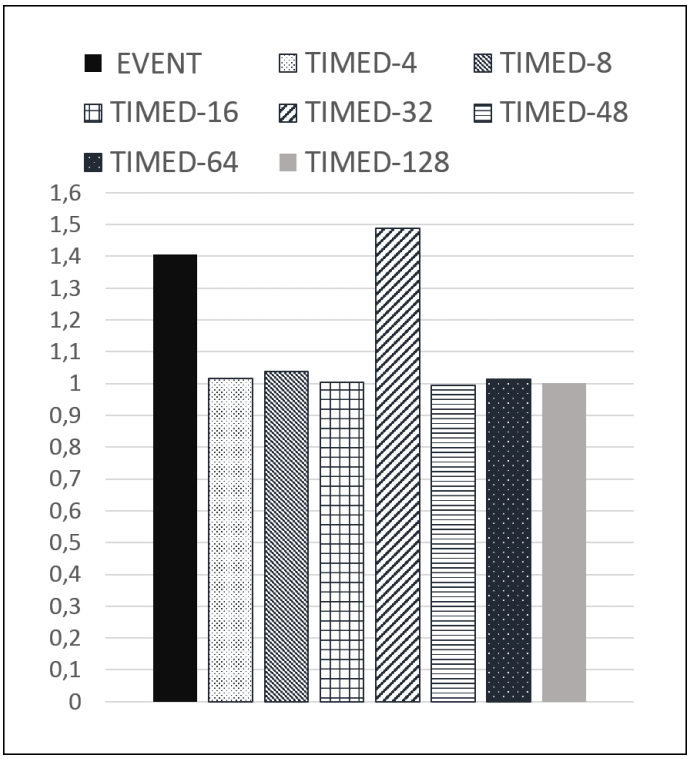


Para mostrar a importância do tamanho da janela, o Gráfico 6.6 ilustra a penalidade média dos processadores em relação à variação do tamanho da janela de previsão do algoritmo de prefetching temporizado com agressividade 4. Já o Gráfico 6.7 apresenta o número de transações também em relação à variação do tamanho da janela. Pelo gráfico da penalidade é possível observar que a penalidade para janela de tamanho 48 foi um pouco menor que os demais. Este fato pode ser explicado pela combinação dos resultados do número de transações da rede com a precisão, poluição e número de transações atrasadas do prefetching. Isto porque o número de transações injetadas na rede para a

Tabela 6.5 Tabela com percentual de poluição na cache local provocada pelo prefetching para a aplicação Radix.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	0,0007%	0%	0%
Timed (janela de previsão = 4)	0%	0%	0,0021%
Timed (janela de previsão = 8)	0,0014%	0%	0,0031%
Timed (janela de previsão = 16)	0%	0%	0%
Timed (janela de previsão = 32)	0%	0%	0%
Timed (janela de previsão = 48)	0%	0%	0%
Timed (janela de previsão = 64)	0%	0,0014%	0%
Timed (janela de previsão = 128)	0%	0%	0%

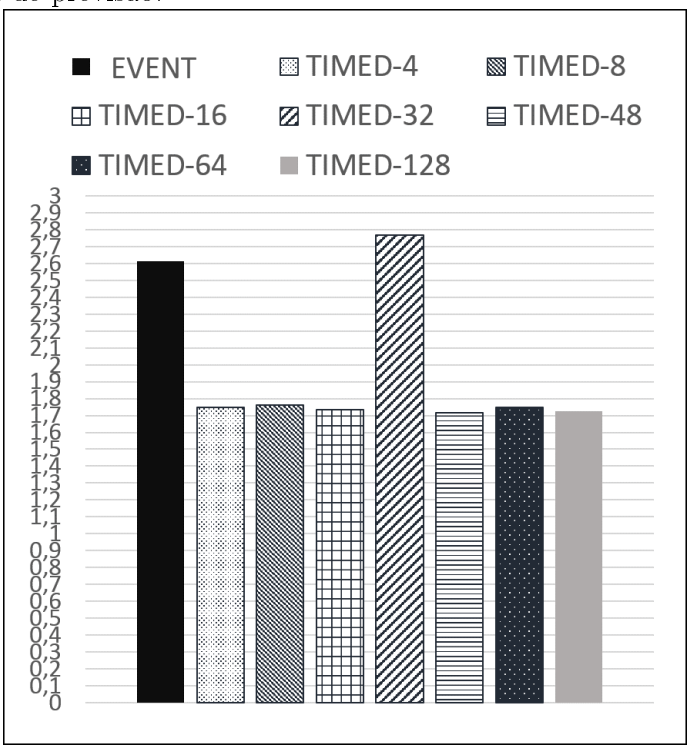
Gráfico 6.6 Penalidade média para a aplicação Radix com agressividade 4, variando o tamanho da janela de previsão.



janela de tamanho 48 foi o menor dentre às outras opções de tamanho de janela, o que não causa tanto congestionamento na rede, lembrando que o congestionamento aumenta com a quantidade de pacotes trafegando e, consequentemente, a latência de transmissão de dados. Associado a isto, a precisão do prefetching e o atraso das transações foi intermediário em relação aos outros tamanhos de janela. Um dado que chama atenção é a

penalidade para a janela de tamanho 32, que destoa dos demais resultados, isto porque houve uma taxa de poluição na cache de 0,0014%.

Gráfico 6.7 Número de transações na rede para a aplicação Radix com agressividade 4, variando o tamanho da janela de previsão.



Para agressividade 4 e 16, executando a aplicação Radix, o melhor desempenho do algoritmo foi para a janela de previsão de tamanho 48, enquanto que para a agressividade 8, a janela de previsão que apresentou melhores resultados foi a de tamanho 32.

Daqui em diante, a penalidade e o número de transações do algoritmo proposto sempre será da janela que obteve melhor resultado para aquela agressividade.

A próxima aplicação a ser analisada é a SHA, o Gráfico 6.8 e a Tabela 6.6 apresentam o resultado da execução do prefetching temporizado na aplicação para valores de agressividade iguais a 4, 8 e 16, comparando com o resultado da aplicação com a abordagem NO-PREFETCHING e EVENT. Assim como a aplicação Radix, para agressividade 4 e 16 a melhor janela de previsão foi a de tamanho 48, obtendo uma redução na penalidade de 17,36% e 24,70% em relação ao NO-PREFETCHING para a agressividade 4 e 16, respectivamente. Em relação à abordagem EVENT, a melhoria foi de 4,12% e 8,77% para

agressividade 4 e 16, respectivamente. Para agressividade 8, o desempenho do algoritmo foi melhor para a janela de tamanho 32 com uma melhoria de 22,31% e 6,29% em relação às abordagens NO-PREFETCHING e EVENT, respectivamente.

Gráfico 6.8 Penalidade média dos processadores para a aplicação Sha.

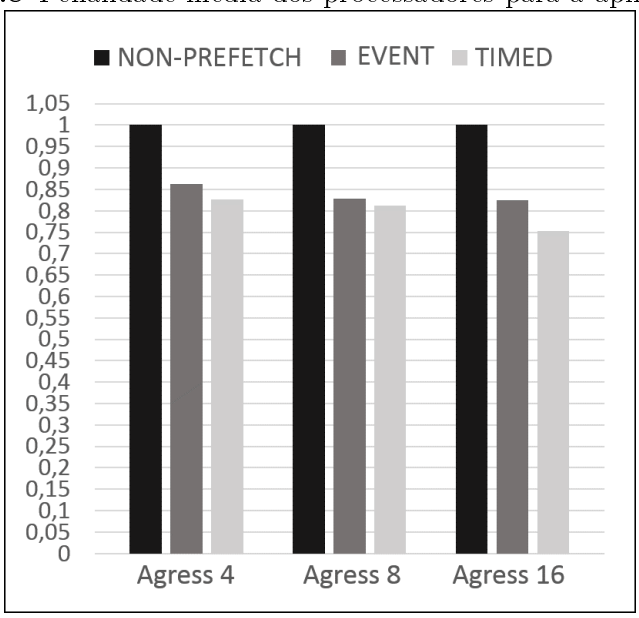


Tabela 6.6 Tabela com resultados da penalidade para aplicação Sha.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	13,81%	17,09%	17,47%
Timed x No-Prefetching	17,36%	22,31%	24,70%
Timed x Event	4,12%	6,29%	8,77%

Em relação ao número de transações na rede, para a aplicação Sha, pode-se observar no Gráfico 6.9 e na Tabela 6.7 que a abordagem NO-PREFETCHING envia bem menos transações para a rede, ainda assim, a abordagem TIMED enviou um número menor de transações de prefetching para a rede do que a abordagem EVENT. Isto acontece porque uma transação de prefetching insere menos pacotes na rede do que a resolução de coerência de cache quando ocorre uma falta na cache. Como para realizar o prefetching na abordagem EVENT tem que ocorrer uma falta na cache, mais transações são enviadas para a rede.

Os Gráficos 6.10, 6.11 e 6.12 apresentam a precisão do prefetching, a poluição gerada

Gráfico 6.9 Número de transações na rede para a aplicação Sha.

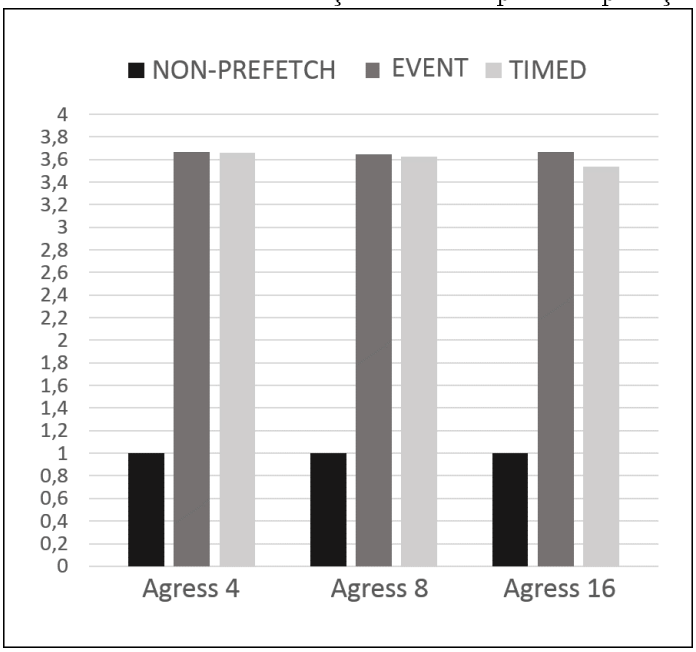
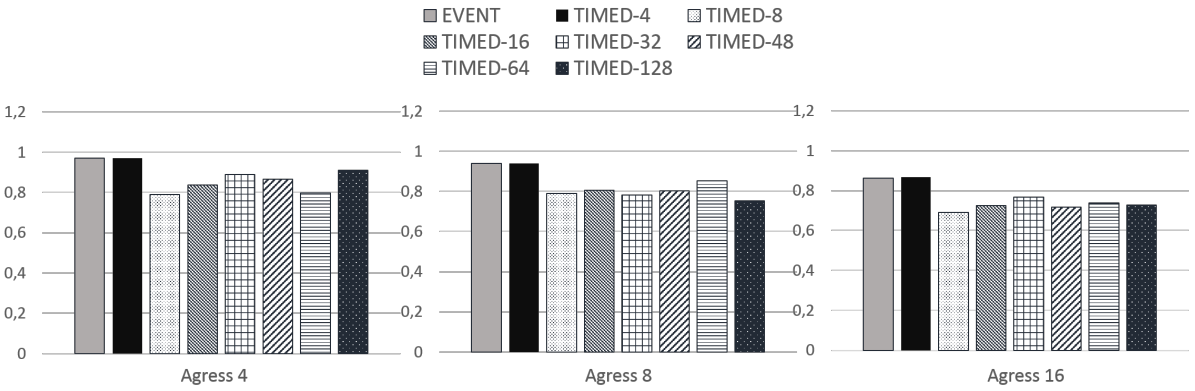


Tabela 6.7 Tabela com número de transações na rede para a aplicação Sha.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	-266,26%	-264,38%	-266,35%
Timed x No-Prefetching	-266,21%	-257,52%	-253,97%
Timed x Event	0,01%	1,88%	3,38%

na cache pelas transações de prefetching e o percentual de transações de prefetching que chegaram atrasadas na cache para a aplicação Sha, respectivamente.

Gráfico 6.10 Percentual de precisão do prefetching para a aplicação Sha.



As Tabelas 6.8, 6.9 e 6.10 apresentam os valores percentuais para precisão, poluição

Gráfico 6.11 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Sha.

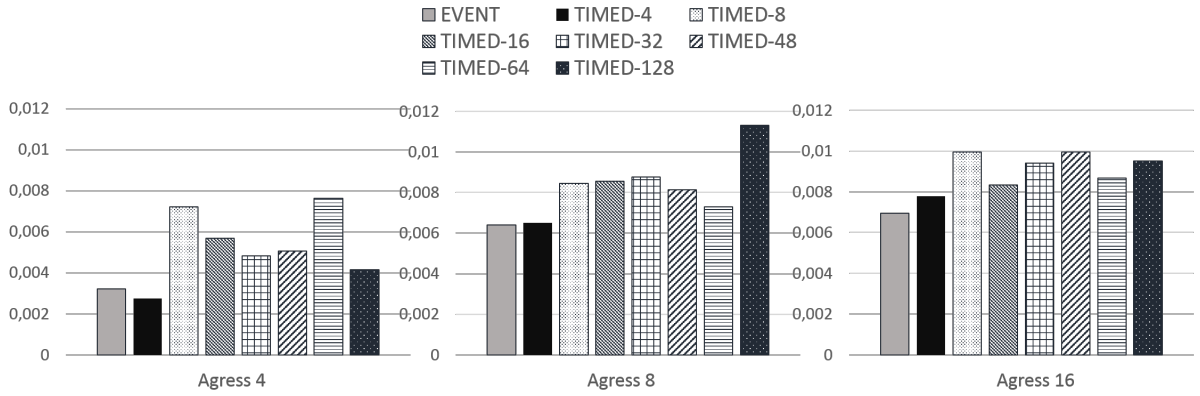
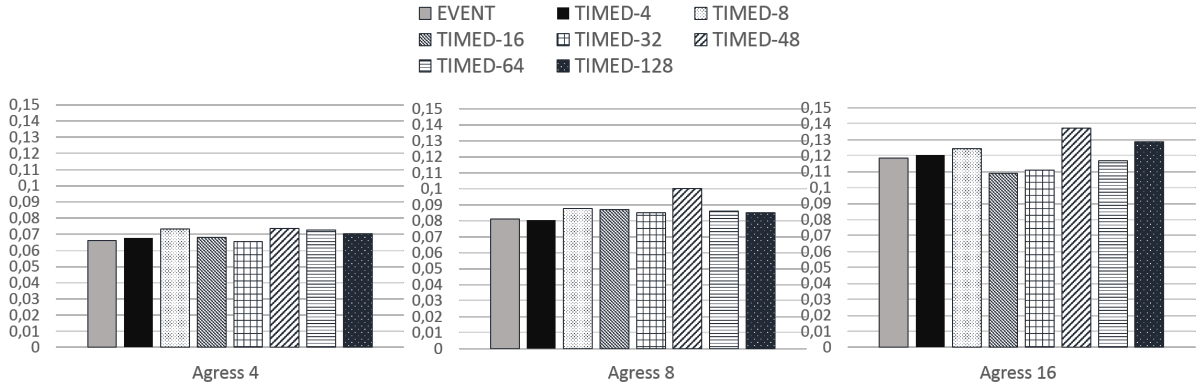


Gráfico 6.12 Percentual de transações de prefetching atrasadas para a aplicação Sha.



e transações atrasadas na execução da aplicação Sha, respectivamente. Pode-se observar nesses gráficos que os tamanhos de janela com maior precisão são os que obtiveram percentual de poluição na cache baixo e menos transações de prefetching atrasadas. Enquanto aqueles que causaram muita poluição na cache e tiveram um número elevado de transações atrasadas, tiveram suas precisões degradadas.

Ao realizar a análise da aplicação Lu apresentada no Gráfico 6.13, é possível notar que para agressividade 4 os ganhos em melhora de penalidade não são tão significativos, mas à medida que a agressividade aumenta, a diferença entre a penalidade da abordagem EVENT em comparação com a abordagem TIMED começa a aumentar chegando a 10,45% para agressividade 16, utilizando a janela de previsão com tamanho 48.

A tabela 6.11 mostra em valores percentuais a diminuição da penalidade amostrada

Tabela 6.8 Percentual de precisão do prefetching para a aplicação Sha.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	97,03%	94%	86,37%
Timed (janela de previsão = 4)	96,99%	94,17%	86,82%
Timed (janela de previsão = 8)	79,11%	79,09%	69,11%
Timed (janela de previsão = 16)	83,50%	80,62%	72,38%
Timed (janela de previsão = 32)	88,99%	78,36%	76,68%
Timed (janela de previsão = 48)	86,70%	80,29%	71,78%
Timed (janela de previsão = 64)	79,90%	85,48%	73,94%
Timed (janela de previsão = 128)	91,09%	75,30%	72,70%

Tabela 6.9 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Sha.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	0,32%	0,64%	0,70%
Timed (janela de previsão = 4)	0,28%	0,65%	0,78%
Timed (janela de previsão = 8)	0,72%	0,85%	1%
Timed (janela de previsão = 16)	0,57%	0,86%	0,83%
Timed (janela de previsão = 32)	0,48%	0,88%	0,94%
Timed (janela de previsão = 48)	0,51%	0,81%	1%
Timed (janela de previsão = 64)	0,76%	0,73%	0,87%
Timed (janela de previsão = 128)	0,42%	1,13%	0,95%

Tabela 6.10 Percentual de transações de prefetching atrasadas para a aplicação Sha.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	6,61%	8,10%	11,83%
Timed (janela de previsão = 4)	6,77%	8,03%	12,02%
Timed (janela de previsão = 8)	7,31%	8,76%	12,43%
Timed (janela de previsão = 16)	6,79%	8,69%	10,88%
Timed (janela de previsão = 32)	6,54%	8,50%	11,10%
Timed (janela de previsão = 48)	7,35%	10,01%	13,71%
Timed (janela de previsão = 64)	7,26%	8,59%	11,67%
Timed (janela de previsão = 128)	7,04%	8,51%	12,85%

para abordagem TIMED em relação à abordagem EVENT e NO-PREFETCHING, bem como a diminuição da penalidade da abordagem EVENT em relação à NO-PREFETCHING.

Para o número de transações na rede também foi identificada diminuição do mesmo para a abordagem TIMED em relação a EVENT. Mas, assim como em todos os outros casos, utilizar um mecanismo de prefetching aumenta significativamente o número de

Gráfico 6.13 Penalidade média dos processadores para a aplicação Lu.

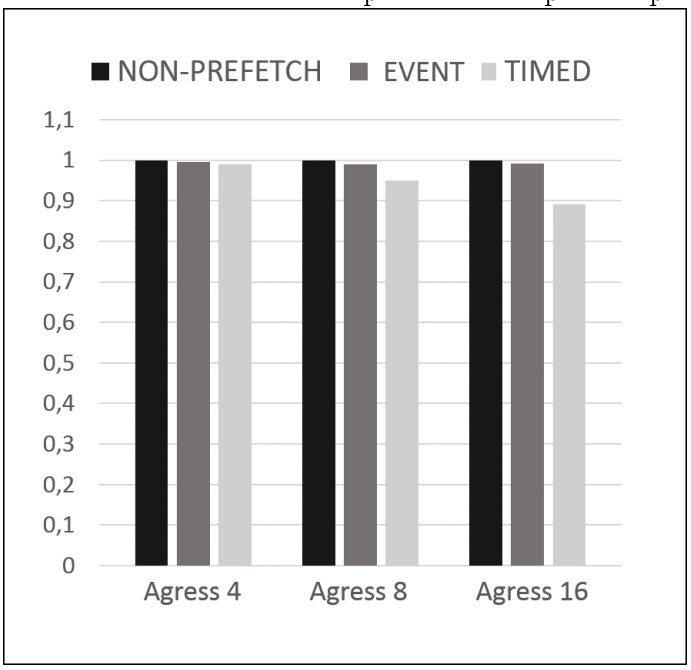


Tabela 6.11 Tabela com resultados da penalidade para aplicação Lu.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	0,46%	1,06%	0,89%
Timed x No-Prefetching	0,97%	4,99%	10,88%
Timed x Event	0,51%	3,97%	10,07%

transações quando comparado a não realizar prefetching. Isto pode ser observado no Gráfico 6.14 e confirmado na Tabela 6.12.

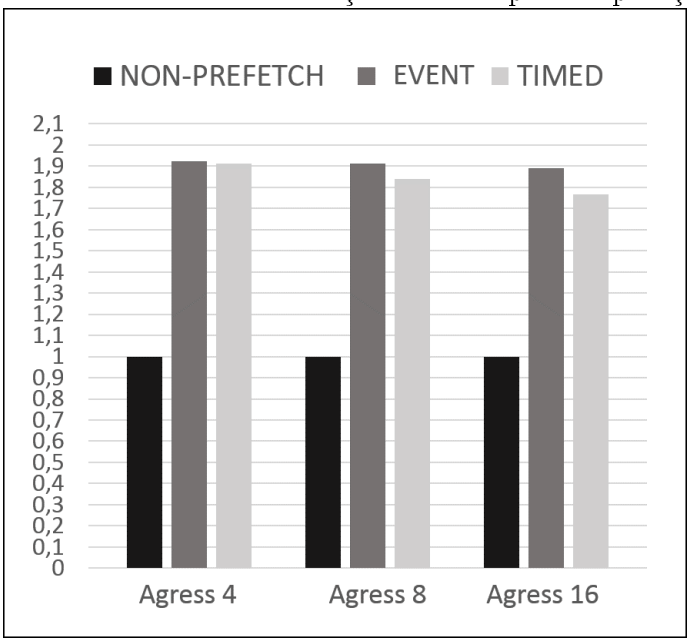
Tabela 6.12 Tabela com número de transações na rede para a aplicação Lu.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	-92,19%	-91,913%	-89,22%
Timed x No-Prefetching	-91,38%	-84,09%	-76,83%
Timed x Event	0,42%	3,68%	6,55%

Os Gráficos 6.15, 6.16 e 6.17 apresentam a precisão do prefetching, a poluição gerada na cache pelas transações de prefetching e o percentual de transações de prefetching que chegaram atrasadas na cache para a aplicação Lu, respectivamente.

As tabelas 6.13, 6.14 e 6.15 apresentam os valores percentuais para precisão, poluição e transações atrasadas na execução da aplicação Lu, respectivamente.

Gráfico 6.14 Número de transações na rede para a aplicação Lu.



Para a aplicação Basicmath também houve redução na penalidade média dos processadores utilizando o algoritmo de prefetching proposto neste trabalho em relação ao prefetching baseado em eventos. Neste caso, a maior redução da penalidade foi de 4,51% quando executando o prefetching com agressividade fixa em 16 e janela de tamanho 8. O Gráfico 6.18 e a Tabela 6.16 apresentam estes dados.

Como esperado, a redução do número de transações também foi maior para o mesmo caso em que a penalidade foi menor, prefetching TIMED com agressividade 16 e janela

Gráfico 6.15 Percentual de precisão do prefetching para a aplicação Lu.

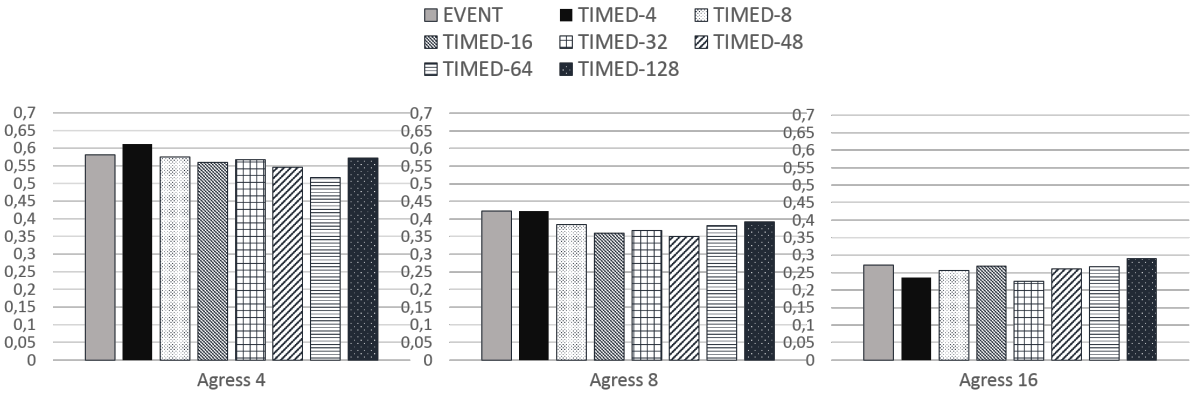


Gráfico 6.16 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Lu.

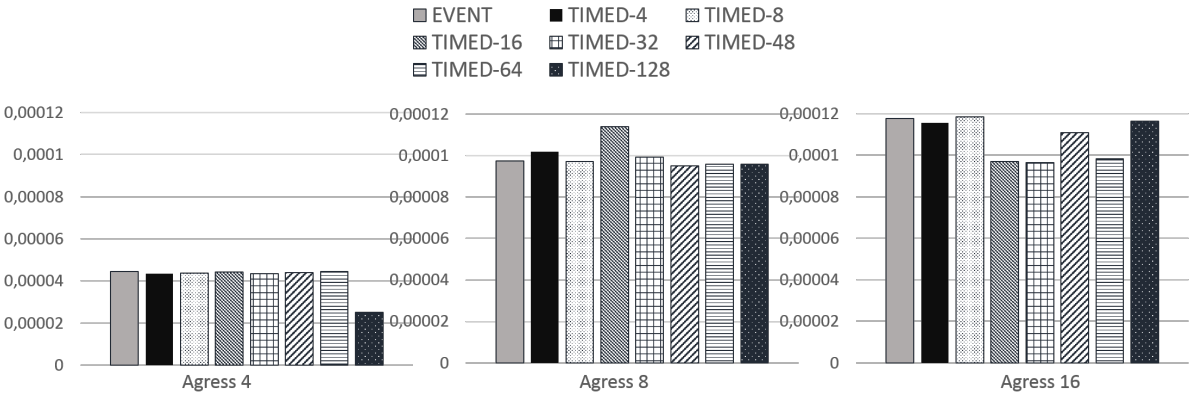


Gráfico 6.17 Percentual de transações de prefetching atrasadas para a aplicação Lu.

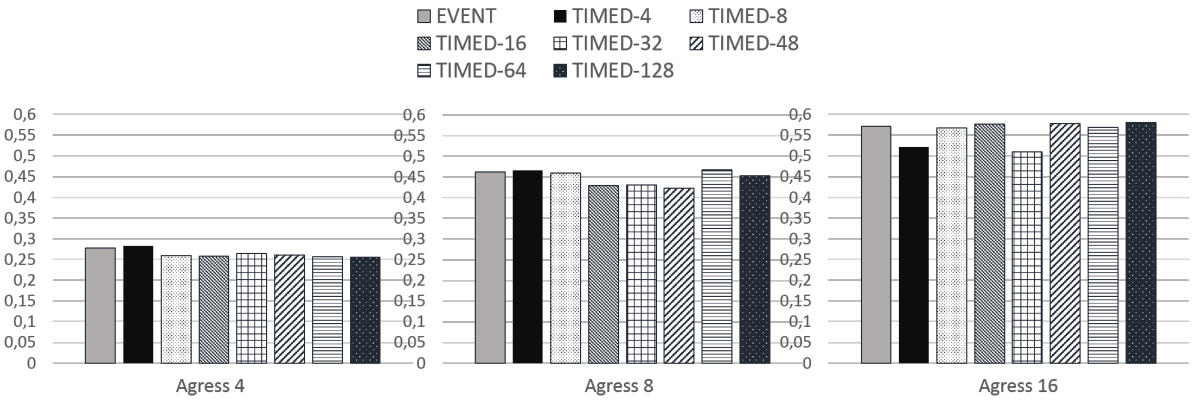


Tabela 6.13 Percentual de precisão do prefetching para a aplicação Lu.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	58,13%	42,34%	27,16%
Timed (janela de previsão = 4)	61,23%	42,21%	23,58%
Timed (janela de previsão = 8)	57,51%	38,41%	25,62%
Timed (janela de previsão = 16)	55,98%	35,98%	26,77%
Timed (janela de previsão = 32)	56,71%	36,79%	22,58%
Timed (janela de previsão = 48)	54,53%	35,04%	26%
Timed (janela de previsão = 64)	51,66%	38,10%	26,73%
Timed (janela de previsão = 128)	57,16%	39,29%	28,96%

de previsão de tamanho 8. O número de transações diminuiu em 3,65% como pode ser observado no Gráfico 6.19 e na Tabela 6.17.

Os Gráficos 6.20 e 6.21 apresentam a precisão do prefetching e o percentual de tran-

Tabela 6.14 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Lu.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	0,0045%	0,0098%	0,0118%
Timed (janela de previsão = 4)	0,0043%	0,0102%	0,0116%
Timed (janela de previsão = 8)	0,0044%	0,0097%	0,0118%
Timed (janela de previsão = 16)	0,0044%	0,0114%	0,0097%
Timed (janela de previsão = 32)	0,0044%	0,0099%	0,0097%
Timed (janela de previsão = 48)	0,0044%	0,0095%	0,0111%
Timed (janela de previsão = 64)	0,0045%	0,0096%	0,0098%
Timed (janela de previsão = 128)	0,0096%	1,0096%	0,0116%

Tabela 6.15 Percentual de transações de prefetching atrasadas para a aplicação Lu.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	27,81%	46,29%	57,17%
Timed (janela de previsão = 4)	28,24%	46,56%	52,20%
Timed (janela de previsão = 8)	25,89%	45,99%	56,69%
Timed (janela de previsão = 16)	25,84 %	42,88%	57,72%
Timed (janela de previsão = 32)	26,46%	43,02%	50,97%
Timed (janela de previsão = 48)	26,03%	42,31%	57,84%
Timed (janela de previsão = 64)	25,71%	46,76%	56,92%
Timed (janela de previsão = 128)	25,52%	45,31%	58,11%

Tabela 6.16 Tabela com resultados da penalidade para aplicação Basicmath.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	6,11%	6,7%	5,15%
Timed x No-Prefetching	6,87%	6,89%	9,43%
Timed x Event	0,81%	0,20%	4,51%

Tabela 6.17 Tabela com número de transações na rede para a aplicação Basicmath.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	-68,77%	-69,09%	-70,46%
Timed x No-Prefetching	-67,48%	-67,51%	-64,24%
Timed x Event	0,76%	0,94%	3,65%

sações de prefetching que chegaram atrasadas na cache para a aplicação Basicmath. Essa aplicação não apresentou poluição em nenhuma abordagem nem para agressividade 4 nem para agressividade 8. Apenas para a agressividade 16 a abordagem TIMED com janela de previsão tamanho 128 apresentou 0,03% de poluição gerada na cache pelo prefetching.

Gráfico 6.18 Penalidade média dos processadores para a aplicação Basicmath.

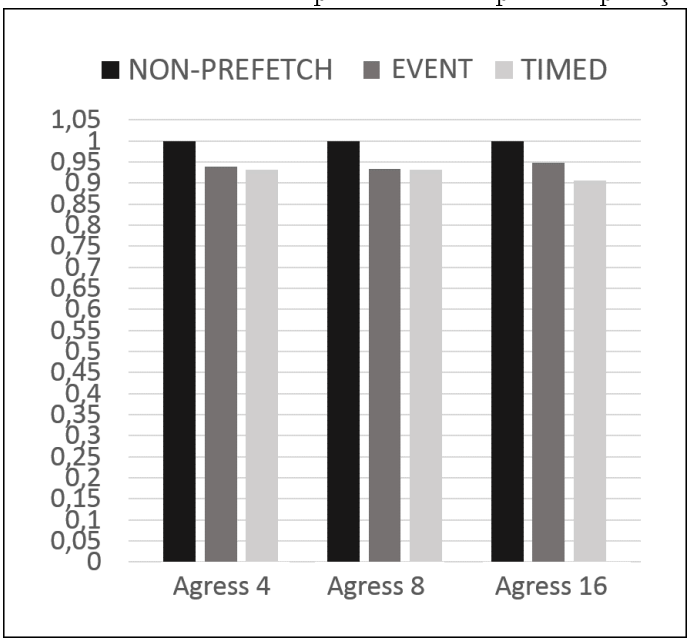
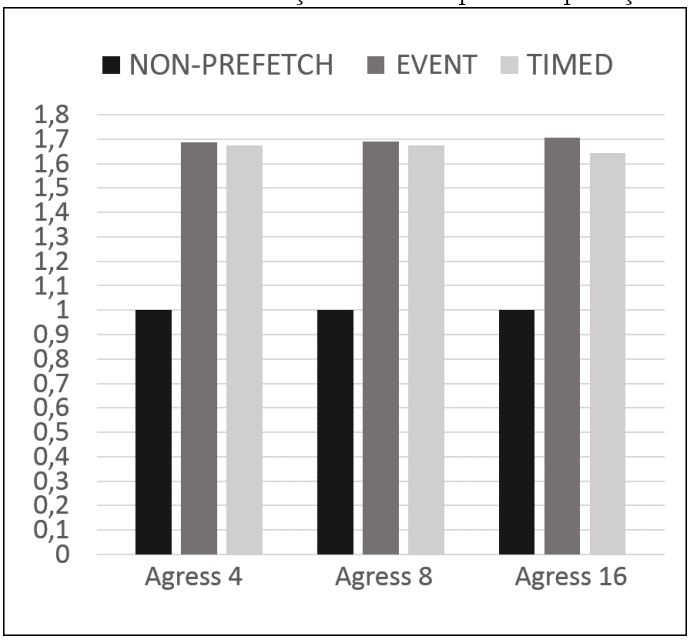


Gráfico 6.19 Número de transações na rede para a aplicação Basicmath.



As Tabelas 6.18 e 6.19 apresentam os valores percentuais para precisão e transações atrasadas na execução da aplicação Basicmath.

A aplicação Susan Corners foi a que apresentou uma melhoria menos significativa quando executada com o algoritmo de prefetching temporizado (TIMED), ainda assim,

Gráfico 6.20 Percentual de precisão do prefetching para a aplicação Basicmath.

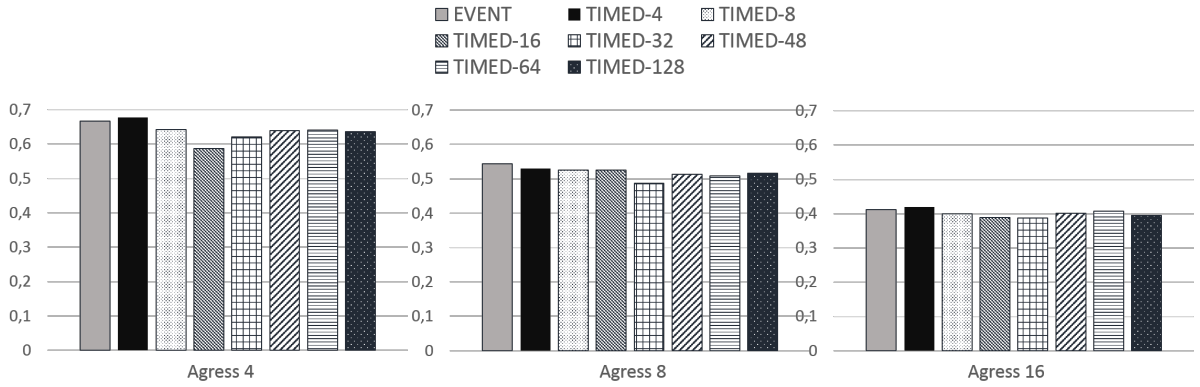


Gráfico 6.21 Percentual de transações de prefetching atrasadas para a aplicação Basicmath.

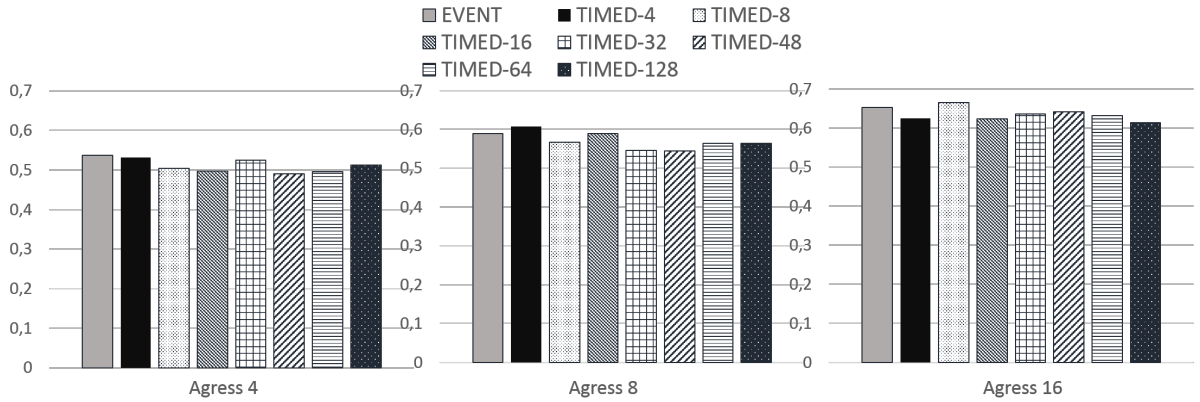


Tabela 6.18 Percentual de precisão do prefetching para a aplicação Basicmath.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	66,72%	54,38%	41,23%
Timed (janela de previsão = 4)	67,74%	52,95%	41,91%
Timed (janela de previsão = 8)	64,24%	52,48%	39,97%
Timed (janela de previsão = 16)	58,67%	52,58%	38,82%
Timed (janela de previsão = 32)	62,01%	48,68%	38,64%
Timed (janela de previsão = 48)	63,84%	51,40%	40,05%
Timed (janela de previsão = 64)	64,06%	50,94%	40,75%
Timed (janela de previsão = 128)	63,56%	51,56%	39,42%

a penalidade média foi menor que a abordagem NO-PREFETCHING e o que a abordagem EVENT. O Gráfico 6.22 apresenta essa informação. No melhor caso, a redução da penalidade foi de 2,82% como pode ser verificado na Tabela 6.21.

Embora a penalidade média da abordagem TIMED tenha sofrido uma redução em

Tabela 6.19 Percentual de transações de prefetching atrasadas para a aplicação Basicmath.

Abordagens	Agress 4	Agress 8	Agress 16
Event	53,75%	58,85%	65,23%
Timed (janela de previsão = 4)	53,21%	60,73%	62,45%
Timed (janela de previsão = 8)	50,49%	56,64%	66,49%
Timed (janela de previsão = 16)	49,78%	58,89%	62,35%
Timed (janela de previsão = 32)	52,43%	54,62%	63,57%
Timed (janela de previsão = 48)	49,03%	54,42%	64,19%
Timed (janela de previsão = 64)	49,65%	56,35%	63,17%
Timed (janela de previsão = 128)	51,22%	56,33%	61,39%

Gráfico 6.22 Penalidade média dos processadores para a aplicação Susan Corners.

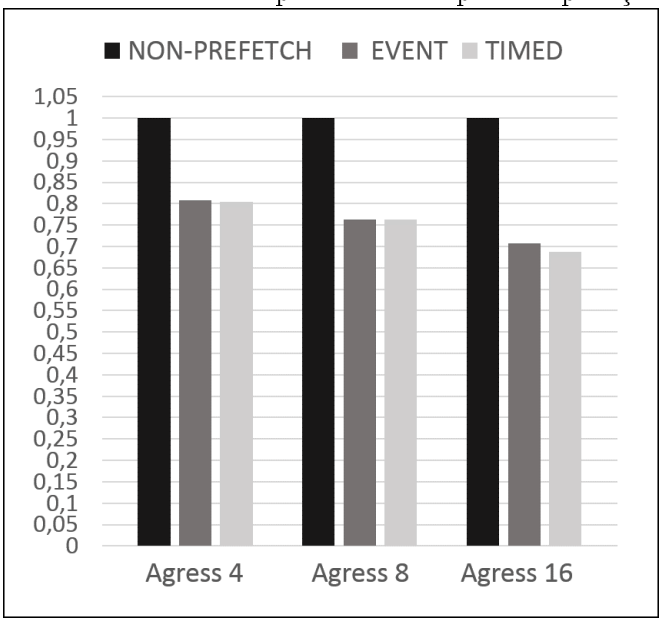


Tabela 6.20 Tabela com resultados da penalidade para aplicação Susan Corners.

Abordagens	Agress 4	Agress 8	Agress 16
Event x No-Prefetching	19,18%	23,79%	29,22%
Timed x No-Prefetching	19,49%	24,49%	31,21%
Timed x Event	0,38%	0,92%	2,82%

relação às abordagens EVENT e NO-PREFETCHING, o mesmo não aconteceu com o número de transações, que embora muito próximo ao número de transações do EVENT, foi sempre superior.

Os Gráficos 6.24, 6.25 e 6.26 apresentam a precisão do prefetching, a poluição gerada na cache pelas transações de prefetching e o percentual de transações de prefetching que

Gráfico 6.23 Número de transações na rede para a aplicação Susan Corners.

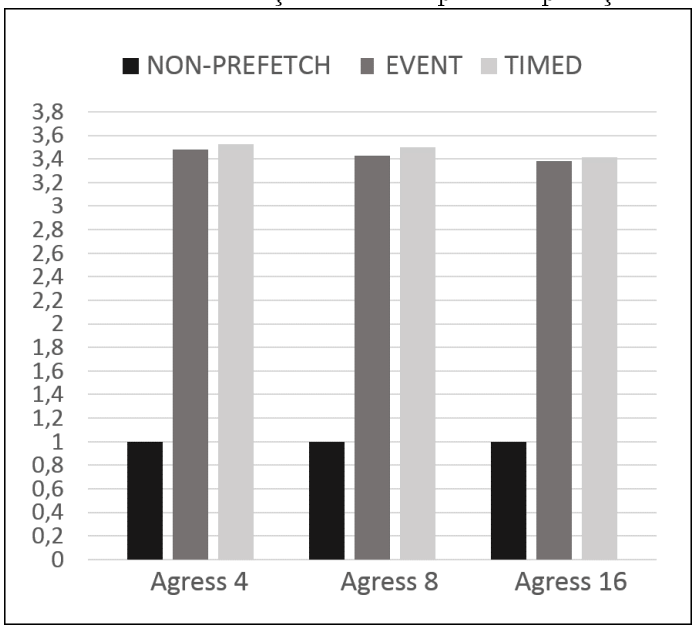
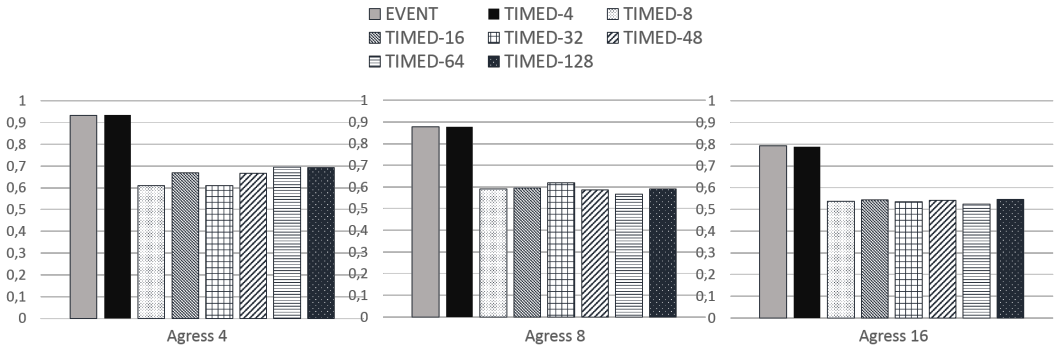


Tabela 6.21 Tabela com número de transações na rede para a aplicação Susan Corners.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event x No-Prefetching	-248,06%	-242,75%	-237,97%
Timed x No-Prefetching	-252,63%	-243,19%	-241,40%
Timed x Event	-1,31%	-0,13%	-1,02%

chegaram atrasadas na cache para a aplicação Susan Corners, respectivamente.

Gráfico 6.24 Percentual de precisão do prefetching para a aplicação Susan Corners.



As Tabelas 6.22, 6.23 e 6.24 apresentam os valores percentuais para precisão, poluição e transações atrasadas na execução da aplicação Susan Corners, respectivamente. O alto grau de poluição na cache somado à taxa de transações atrasadas, explica porque a

Gráfico 6.25 Percentual de poluição na cache L1 provocada pelo prefetching para a aplicação Susan Corners.

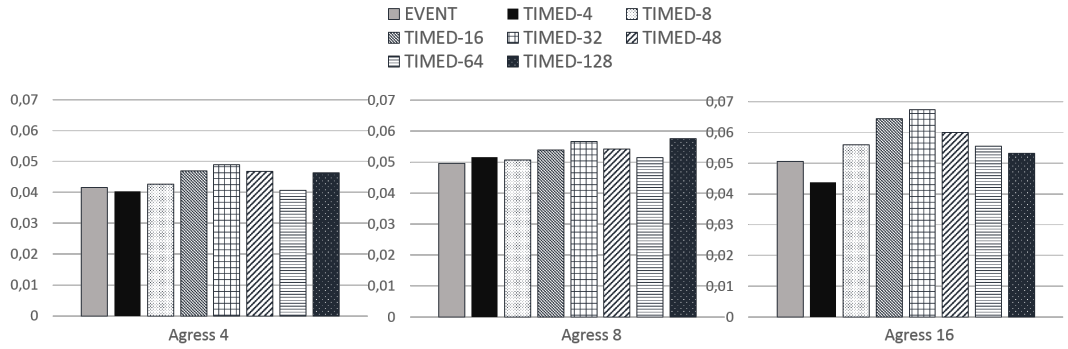
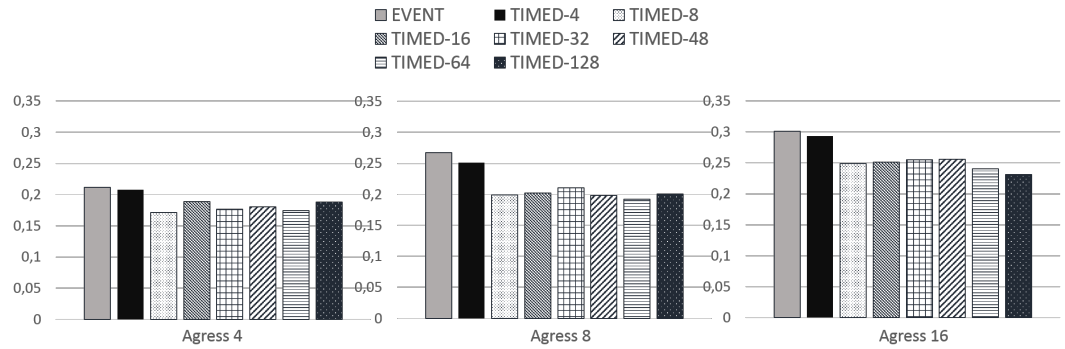


Gráfico 6.26 Percentual de transações de prefetching atrasadas para a aplicação Susan Corners.



penalidade média não teve tanta diferença da abordagem EVENT.

Tabela 6.22 Percentual de precisão do prefetching para a aplicação Susan Corners.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	93,31%	87,81%	79,33%
Timed (janela de previsão = 4)	87,72%	94,17%	78,93%
Timed (janela de previsão = 8)	59,19%	79,09%	53,66%
Timed (janela de previsão = 16)	59,45%	80,62%	54,35%
Timed (janela de previsão = 32)	61,89%	78,36%	53,55%
Timed (janela de previsão = 48)	58,64%	80,29%	54,25%
Timed (janela de previsão = 64)	56,58%	85,48%	52,34%
Timed (janela de previsão = 128)	59,02%	75,30%	54,71%

Foi possível observar que a abordagem TIMED obteve sucesso em reduzir a penalidade média dos processadores quando se varia o tamanho da janela de previsão. O tamanho da janela de previsão que quantitativamente obteve mais sucesso foi a janela de tamanho 48, que diminuiu a penalidade em 11 de 15 casos (73,33% dos casos).

Tabela 6.23 Percentual de poluição na cache local provocada pelo prefetching para a aplicação Susan Corners.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	4,16%	0,64%	5,06%
Timed (janela de previsão = 4)	4,04%	0,65%	4,38%
Timed (janela de previsão = 8)	4,26%	0,85%	5,60%
Timed (janela de previsão = 16)	4,7%	0,86%	6,45%
Timed (janela de previsão = 32)	4,9%	0,88%	6,75%
Timed (janela de previsão = 48)	4,68%	0,81%	6%
Timed (janela de previsão = 64)	4,07%	0,73%	5,55%
Timed (janela de previsão = 128)	4,63%	1,13%	5,32%

Tabela 6.24 Percentual de transações de prefetching atrasadas para a aplicação Susan Corners.

Abordagens	**Agress 4**	**Agress 8**	**Agress 16**
Event	21,15%	26,74%	30,12%
Timed (janela de previsão = 4)	20,81%	25,10%	29,37%
Timed (janela de previsão = 8)	17,16%	19,93%	24,92%
Timed (janela de previsão = 16)	18,90%	20,19%	25,10%
Timed (janela de previsão = 32)	17,68%	21,07%	25,47%
Timed (janela de previsão = 48)	18,07%	19,85%	25,56%
Timed (janela de previsão = 64)	17,45%	19,20%	24,02%
Timed (janela de previsão = 128)	18,80%	20,09%	23,10%

6.3.3 Prefetching temporizado com controle de agressividade

Nesta seção serão apresentados resultados dos experimentos utilizando a combinação do prefetching baseado em predição de tempo com o controlador de agressividade proposto por Aziz (Aziz, 2014).

O Gráfico 6.27 mostra um comparativo da penalidade média dos processadores para as aplicações Lu (1), Basicmath (2), Sha (3) e Susan Corners (4), para as abordagens NO-PREFETCHING, EVENT, CONTROLLED, TIMED e CONTROLLED+TIMED. NO-PREFETCHING é uma abordagem sem a utilização de nenhum mecanismo de prefetching; EVENT é a abordagem de prefetching baseada em evento com agressividade fixa em 4; CONTROLLED é a abordagem baseada em evento com o controle de agressividade proposto por Aziz (Aziz, 2014); TIMED é a abordagem proposta neste trabalho com prefetching baseado em predição de tempo com agressividade fixa em 16; e CON-

TROLLED+TIMED é uma abordagem que faz a combinação do prefetching baseado em predição de tempo utilizando controle de agressividade.

Já a Tabela 6.25 apresenta a melhoria de cada uma das abordagens mencionadas em relação à abordagem NO-PREFETCHING. Todos os resultados desta seção foram obtidos usando a janela de previsão com tamanhos 8 e 48, quando utilizada a abordagem TIMED.

Tabela 6.25 Melhoria da penalidade média normalizada em relação à abordagem NO-PREFETCHING para as aplicações Lu, Basicmath, Sha e Susan Corners.

Abordagens	**Lu**	**Basicmath**	**Sha**	**Susan Corners**
Event	0,89%	5,15%	17,47%	29,22%
Timed	10,88%	9,43%	24,70%	31,21%
Controlled	1,37%	7,32%	16,04%	20,18%
Controlled+Timed	-1,46%	16,30%	17,3%	22,11%

Pelo Gráfico 6.27 pode-se observar que para diferentes aplicações o impacto das abordagens varia. Nem sempre a combinação da predição baseada em tempo com o controlador de agressividade vai obter a maior redução de penalidade. Para a aplicação Lu (1), por exemplo, a combinação aumentou a penalidade do sistema, já na aplicação Basicmath (2) é possível observar que a combinação CONTROLLED+TIMED foi a que obteve maior redução da penalidade.

Para o número de transações na rede é apresentado o Gráfico 6.28, também para as aplicações Lu (1), Basicmath (2), Sha (3) e Susan Corners (4). Como o número de transações na rede quando utilizando alguma abordagem de prefetching foi sempre superior em relação ao NO-PREFETCHING, a Tabela 6.26 apresenta a porcentagem de transações a mais na rede para as abordagens EVENT, CONTROLLED, TIMED e CONTROLLED+TIMED, em relação à abordagem NO-PREFETCHING.

Em relação à análise das estatísticas de prefetching: precisão, poluição e transações atrasadas; foi realizado o comparativo das abordagems TIMED, CONTROLLED e CONTROLLED+TIMED em relação à abordagem EVENT. O Gráfico 6.29 mostra a precisão do prefetching para as aplicações Lu (1), Basicmath (2), Sha (3) e Susan Corners (4).

Pode-se observar que a predição baseada em tempo (TIMED) piorou a precisão do

Gráfico 6.27 Penalidade média dos processadores adicionando as abordagens CONTROLLED e CONTROLLED+TIMED em relação as aplicações Lu, Basicmath, Sha e Susan Corners.

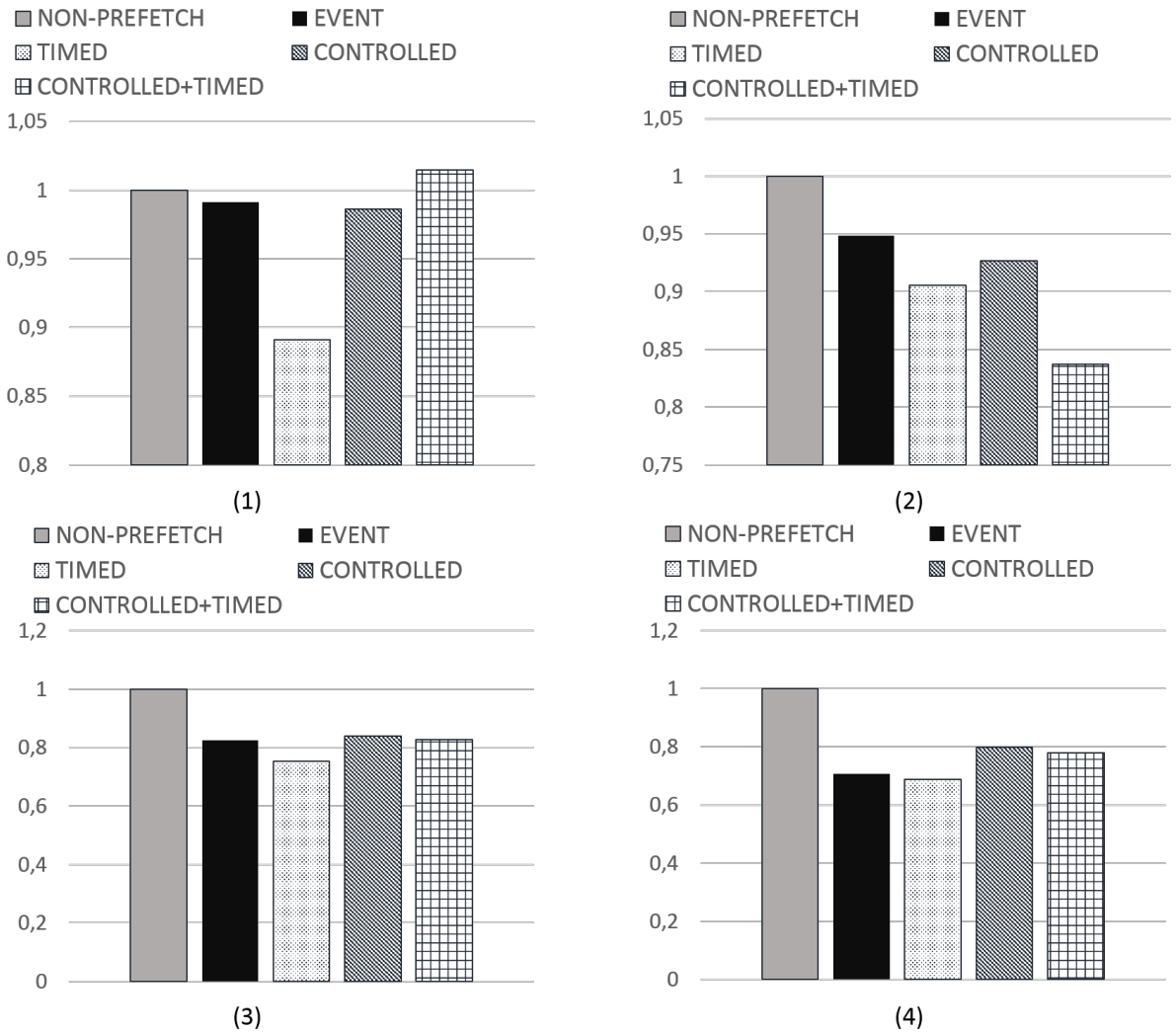
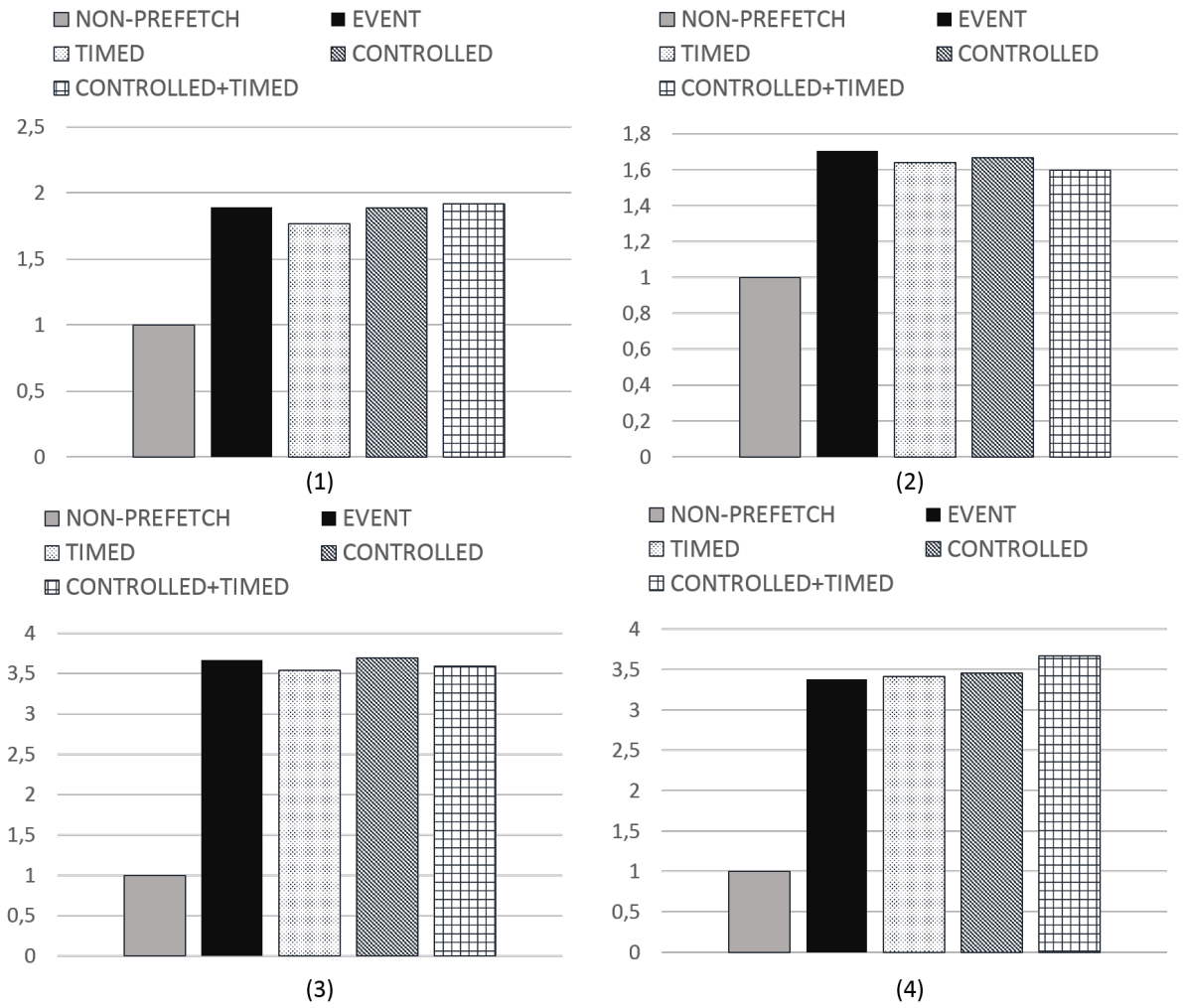


Tabela 6.26 Quantidade de transações na rede normalizada em relação à abordagem NO-PREFETCHING para as aplicações Lu, Basicmath, Sha e Susan Corners.

Abordagens	**Lu**	**Basicmath**	**Sha**	**Susan Corners**
Event	89,22%	70,46%	266,35%	237,97%
Timed	76,83%	64,24%	253,97%	241,40%
Controlled	88,67%	66,74%	269,04%	245,96%
Controlled+Timed	91,48%	59,79%	258,92%	266,97%

prefetching em relação às abordagens que utilizam a predição baseada em evento (EVENT e CONTROLLED), porém melhorou a precisão em relação à abordagem com a combinação CONTROLLED+TIMED para as aplicações (2) e (4). Este fato é explicado pelo

Gráfico 6.28 Número de transações de prefetching na rede adicionando as abordagens CONTROLLED e CONTROLLED+TIMED em relação as aplicações Lu, Basicmath, Sha e Susan Corners.



aumento da poluição na cache e de transações atrasadas em relação à abordagem de prefetching baseada em evento. Para melhoria desses valores é necessário um ajuste do tamanho da janela de previsão. Outro fator que contribui para a degradação do desempenho do prefetching é que a predição de tempo não está levando em consideração o tempo de processamento do controle de agressividade, esta modificação tem que ser realizada no algoritmo para que a combinação do TIMED com o CONTROLLED seja positiva para o sistema. A Tabela 6.27 apresenta os valores percentuais da precisão do prefetching em relação à abordagem EVENT.

Gráfico 6.29 Precisão do prefetching em relação às aplicações Lu, Basicmath, Sha e Susan Corners.

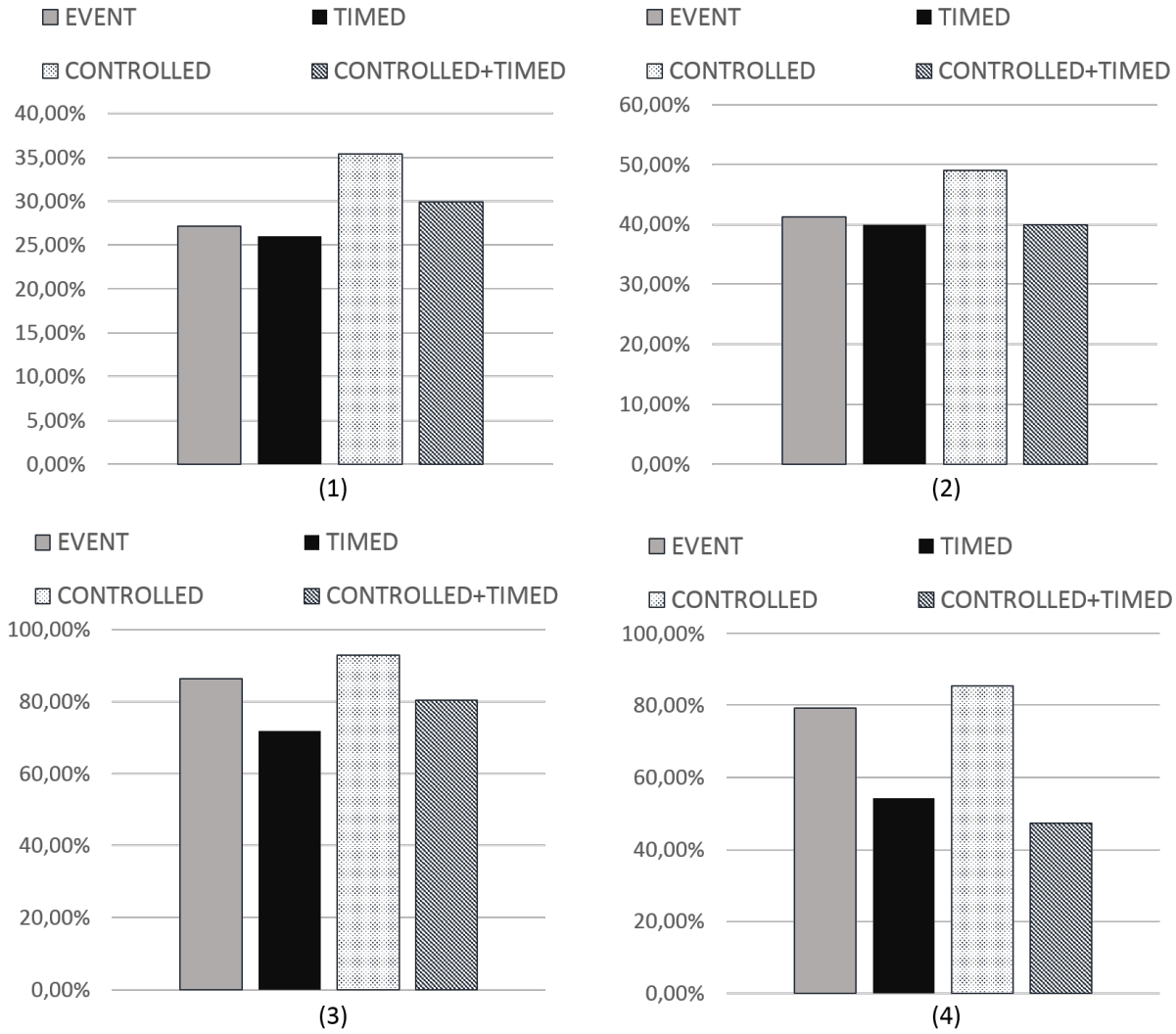


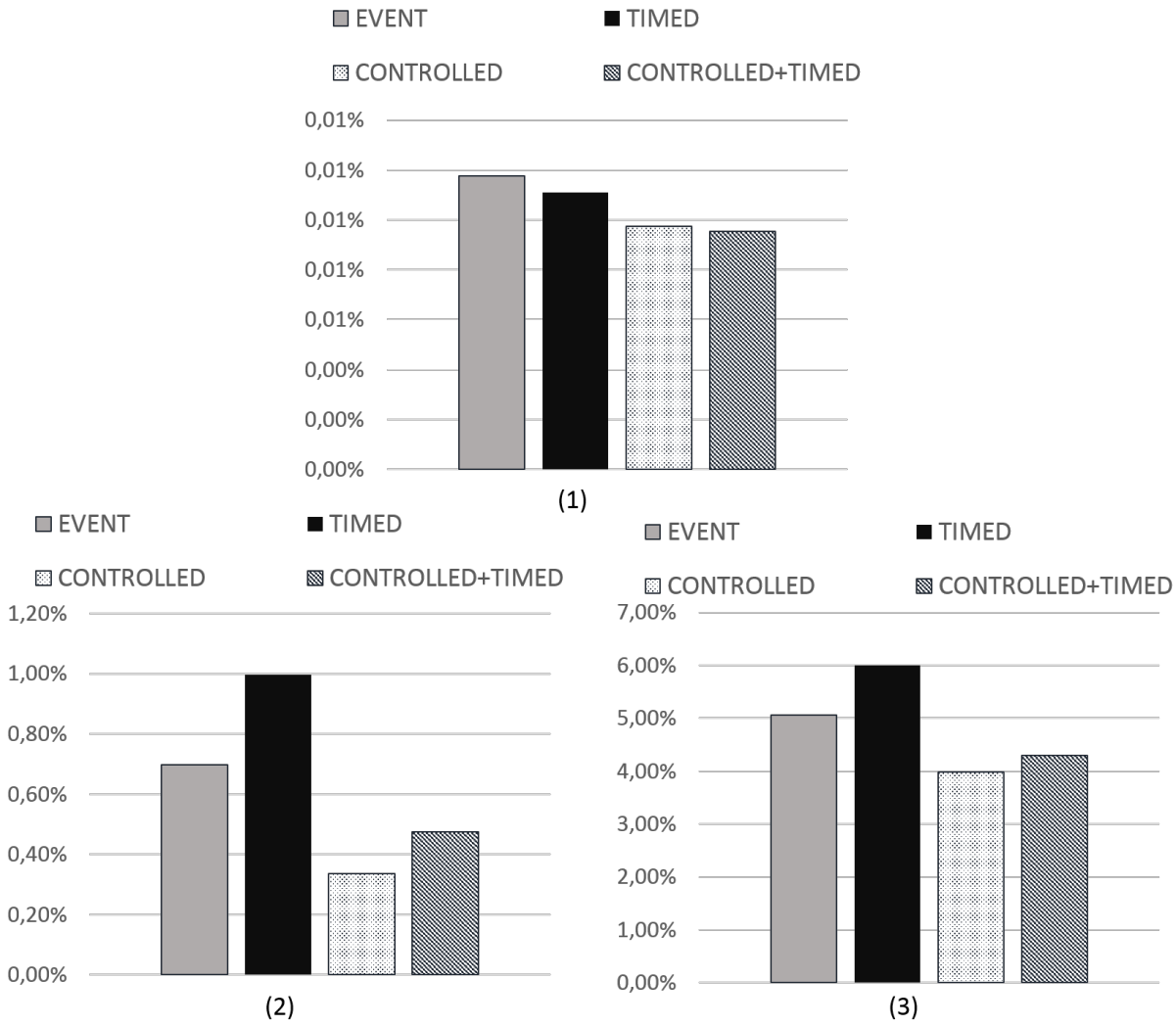
Tabela 6.27 Melhoria da precisão do prefetching em valores percentuais em relação à abordagem EVENT para as aplicações Lu, Basicmath, Sha e Susan Corners.

Abordagens	**Lu**	**Basicmath**	**Sha**	**Susan Corners**
Timed	-4,25%	-3,06%	-16,89%	-31,62%
Controlled	30,26%	18,87%	7,67%	7,77%
Controlled+Timed	10,10%	-2,96%	-6,94%	-40,29%

Para a aplicação Basicmath não foi gerada poluição da cache pelo prefetching, isto pode ocorrer quando o número de transações de prefetching não é muito alto e a precisão do prefetching é elevada. O Gráfico 6.30 apresenta os percentuais de poluição causados

pelo prefetching para as aplicações Lu (1), Sha (2) e Susan Corners (3).

Gráfico 6.30 Poluição causada pelo prefetching em relação às aplicações Lu, Sha e Susan Corners.



Como observado anteriormente, a poluição é maior quando a abordagem TIMED é combinada com a CONTROLLED, pelos motivos já mencionados, para as aplicações (2) e (3). Enquanto que na aplicação (1), o prefetching baseado em tempo sempre reduziu a poluição na cache em relação ao prefetching baseado em evento.

Em relação à porcentagem de transações de prefetching atrasadas, pode-se observar no Gráfico 6.31 que para a aplicação (3) a combinação CONTROLLED+TIMED obteve maior quantidade de transações atrasadas do que a abordagem TIMED, o que refletiu

Tabela 6.28 Comparativo da porcentagem de poluição causada pelo prefetching em relação à abordagem EVENT para as aplicações Lu, Sha e Susan Corners.

Abordagens	**Lu**	**Sha**	**Susan Corners**
Timed	-5,78%	43,16%	18,68%
Controlled	-17,31%	-51,64%	-21,20%
Controlled+Timed	-19,03%	-31,86%	-15,06%

na precisão do prefetching e na penalidade. Enquanto para as aplicações (1), (2) e (4), a utilização da abordagem baseada em predição de tempo obteve menos transações atrasadas do que a baseada em evento quando utilizando o controle de agressividade.

Gráfico 6.31 Porcentagem de transações de prefetching atrasadas para as aplicações Lu, Basicmath, Sha e Susan Corners.

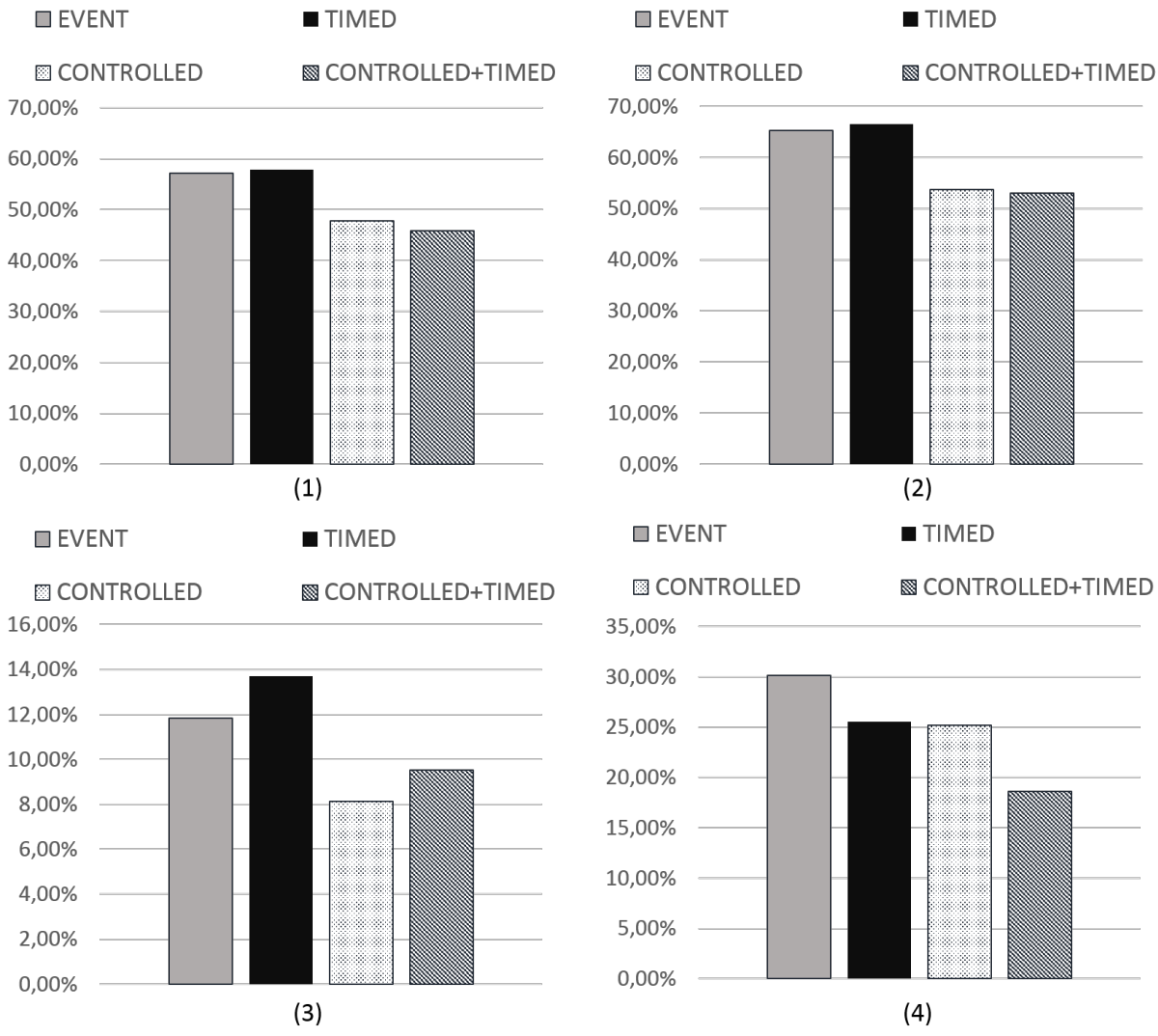


Tabela 6.29 Comparativo da porcentagem de transações de prefetching atrasadas em relação à abordagem EVENT para as aplicações Lu, Basicmath, Sha e Susan Corners.

Abordagens	**Lu**	**Basicmath**	**Sha**	**Susan Corners**
Timed	1,18%	1,94%	15,84%	-15,12%
Controlled	-16,31%	-17,75%	-31,43%	-16,37%
Controlled+Timed	-19,74%	-18,70%	-19,60%	-38,27%

Por fim, o Quadro 6.30 apresenta um resumo dos resultados mais significativos obtidos nos experimentos.

Tabela 6.30 Resumo dos resultados mais significativos de redução da penalidade.

Comparação	**Redução da penalidade**
TIMED x EVENT	29%
TIMED x NO-PREFETCHING	31%
CONTROLLED + TIMED x NO-PREFETCHING	22,96%
CONTROLLED + TIMED x CONTROLLED	9,69%
CONTROLLED + TIMED x EVENT	11,76%
CONTROLLED + TIMED x TIMED	7,58%

CAPÍTULO 7

CONCLUSÃO

Neste trabalho foi apresentado um Algoritmo de Prefetching de Dados Baseado em Predição de Tempo voltado para sistemas multiprocessadores baseados em NoC. O objetivo do algoritmo é antecipar a necessidade dos processadores em relação a quais dados eles irão requisitar e carregar tais dados na cache local no momento adequado, reduzindo assim a penalidade média dos processadores.

O algoritmo utiliza o método de previsão de suavização exponencial de Holt para séries temporais para realizar a previsão de momento em que ocorreria uma suposta falta de cache, causada pela requisição do processador por um bloco não presente na cache. Essa previsão é realizada utilizando o histórico de faltas na cache.

Porém, antecipar o momento de requisição do processador não é suficiente para garantir que o dado estará disponível na cache em tempo. Para isto, o algoritmo calcula uma estimativa da latência de transferência de uma requisição de prefetching do nó fonte para o nó destino através da rede. Este cálculo é feito utilizando a penalidade média do ciclo anterior de amostragem de estatísticas da rede. Além disso o algoritmo também considera o tempo necessário para a coerência de cache ser resolvida e o tempo de processamento total utilizado para calcular a previsão.

Os experimentos foram realizados na Plataforma Infinity, desenvolvida por Aziz (Aziz, 2014), utilizando aplicações pertencentes aos benchmarks comerciais PARSEC e SPLASH-2. As aplicações utilizadas foram: BASICMATH, SHA, LU, SUSANCORNERS e RADIX. Nos experimentos a abordagem proposta neste trabalho (TIMED) foi comparada com uma abordagem de prefetching baseada no evento de falta de cache (EVENT) e com uma abordagem que não utiliza nenhum algoritmo de prefetching (NO-PREFETCHING).

Nos experimentos realizados para 16 núcleos, o algoritmo proposto reduziu a penali-

dade dos processadores em 53,6% dos casos em comparação com o prefetching baseado em evento (EVENT) para todos os tamanhos de janela testados, sendo a maior redução de 29% da penalidade. Além disso, para todas as aplicações testadas, existe algum tamanho de janela de previsão que melhora a penalidade do sistema quando utilizando o algoritmo proposto em comparação ao algoritmo baseado em evento e à abordagem NO-PREFETCHING. Em comparação com a abordagem NO-PREFETCHING, o algoritmo baseado em predição de tempo reduziu a penalidade média dos processadores em 71,4% dos casos para todos os tamanhos de janela testados, onde o melhor caso reduziu a penalidade em 31%.

Também foram realizados experimentos comparativos em relação ao trabalho proposto por Aziz e em combinação com o controle de agressividade desenvolvido por ele. Foi utilizado tamanho de janela 64 para realizar esses experimentos e, quando a agressividade é fixa, seu valor foi 4. A combinação do controle de agressividade com a predição baseada em tempo (CONTROLLED+TIMED) obteve resultado mais significativo na redução da penalidade em 22,96% em relação à abordagem NO-PREFETCHING. E, quando comparando a abordagem CONTROLLED+TIMED em relação à abordagem com controle de agressividade e prefetching baseado em evento (CONTROLLED), o melhor caso foi uma redução da penalidade em 3,2%.

Resgatando o Quadro 3.1 comparativo entre os trabalhos relacionados do Capítulo 3 foi criado um novo quadro, Quadro 7.1, que inclui a estratégia proposta neste trabalho e também a estratégia proposta por Aziz e as classifica de acordo com as variáveis apresentadas. Para efeitos de identificação, no quadro o trabalho aqui proposto foi chamado de “Cireno, 2015”.

Após a análise dos resultados dos experimentos realizada no Capítulo 6, fica clara a dependência do desempenho do algoritmo para diferentes aplicações em relação ao tamanho da janela de previsão utilizada para previsão de tempo. Por isso, foi identificado como trabalho futuro uma otimização do algoritmo utilizando um controle adaptativo do tamanho da janela de previsão, que através do monitoramento das estatísticas do prefetching pode aumentar ou diminuir o tamanho da janela, aproveitando ao máximo

Quadro 7.1 Quadro comparativo dos trabalhos relacionados com o trabalho proposto.

Trabalho	**Quando buscar?**	**Implementação**	**Interconexão**	**Escalabilidade**
Chen, 1995	Baseado em contador de instruções antecipado	Hardware	Barramento	Centralizado
Mowry, 1991	Sincronização controlada por software	Software	Barramento	N/A
Sun, 2007	Baseado em predição	Hardware	Barramento	Centralizado
Nachiappan, 2012	Baseado em evento	Hardware	NoC	Distribuído
Aziz, 2014	Baseado em evento (falta na cache)	Hardware	NoC	Distribuído
Cireno, 2015	Baseado em predição	Hardware	NoC	Distribuído

a tendência da aplicação. Embora seja uma solução mais complexa do que a proposta neste trabalho, seu desempenho em relação à diminuição da penalidade pode ser muito maior. Outros parâmetros que podem ser variados dinamicamente para tentar otimizar a previsão de tempo são as constantes de suavização de nível e tendência da série temporal, α e β, respectivamente.

Como solução mais simples, outra sugestão seria a adição de um parâmetro de controle no algoritmo que pode optar por persistir na tendência da estimativa da série temporal por mais tentativas, mesmo que ocorra uma falta na cache, ao invés de retornar ao estado inicial. O objetivo seria não deixar um erro pontual de estimativa ser uma grande influência negativa no fluxo do algoritmo.

Finalmente, também ficou pendente uma análise da relação de desempenho versus o consumo de energia do sistema, principalmente quando a escalabilidade for explorada e o número de processadores aumentar significativamente. Uma proposta de trabaho futuro seria explorar como a potência do sistema responde ao aumentar ou diminuir o número de processadores, bem como o desempenho do prefetching e a penalidade dos processadores.

REFERÊNCIAS

Azevedo, R., Bartholomeu, S. R. M., Araujo, G., Araujo, C., and E.Barros (2005). The archc architecture description language and tools. *Int. J. Parallel Program*, **33**(5), 453–484.

Aziz, A. (2014). *Controle de Agressividade de Prefetch em uma Arquitetura Multicore com Coerencia de Cache Baseada em Diretorio e NoC*. Ph.D. thesis, Universidade Federal de Pernambuco.

Bjerregaard, T. and Mahadevan, S. (2006). A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, **38**(1).

Brown, J. A., Wang, H., Chrysos, G., Wang, P. H., and Shen, J. P. (2002). Speculative precomputation on chip multiprocessors. *6th Workshop on Multithreaded Execution, Architecture, and Compilation*, pages 1–8.

Byna, S., Chen, Y., and Sun, X.-H. (2008). Taxonomy of data prefetching for multicore processors.

Caragea, G. C., Tzannes, A., Keceli, F., and Vishkin, U. (2010). Resource-aware compiler prefetching for many-cores. *ISPDC '10 Proceedings of the 2010 Ninth International Symposium on Parallel and Distributed Computing*, pages 133–140.

Carvalho, C. (2002). The gap between processor and memory speeds. *3rd Internal Conference on Computer Architecture*.

Chawade, S. D., Gaikwad, M. A., and Patrikar, R. M. (2012). Review of xy routing

algorithm for network-on-chip architecture. *International Journal of Computer Applications*, **43**(21), 975 – 8887.

Chen, T. and Baer, J. (1995). Effective hardware-based data prefetching for high performance processors. *IEEE Transactions on Computers*, pages 609–623.

Ehlers, R. S. (2009). *Analise de Series Temporais*. 5 edition.

Flores, A., Aragon, J. L., and Acacio, M. E. (2010). Energy-efficient hardware prefetching for cmps using heterogeneous interconnects. *18th Euromicro International Conferenceon Parallel, Distributed and Network-Based Processing (PDP).*, pages 147–154.

Gardner, E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting*, **4**(1), 1–28.

Kamruzzaman, M., Swanson, S., and Tullsen, D. M. (2011). Inter-core prefetching for multicore processors using migrating helper threads. *ASPLOS - Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, page 393âĂŞ404.

Liu, Y., Dimitri, M., and Kaeli, D. R. (1999). Branch-directed and pointer-based data cache prefetching. *Journal of Systems Architecture: the EUROMICRO Journal*, **45**(12-13), 1047–1073.

Nachiappanan, N. C., Sivasubramaniam, A., Mishra, A. K., Mutlu, O., Kandemir, M., and Das, C. R. (2012). Application-aware prefetch prioritization in on-chip networks. *International Conference on Parallel Architectures and Compilation Techniques*, pages 19–23.

Ribeiro, C. V., Goldschmidt, R. R., and Choren, R. (2009). *Metodos para Previsao de Series Temporais e suas Tendencias de Desenvolvimento*. Monografias em sistemas e computacao, Instituto Militar de Engenharia.

Srinath, S., Mutlu, O., Kim, H., and Patt, Y. N. (2007). Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers. *IEEE*

13th International Symposium on High Performance Computer Architecture, 2007, pages 63–74.

Sun, X.-H., Byna, S., and Chen, Y. (2007). Serverbased data push architecture for multi-processor environments. *Journal of Computer Science and Technology - JCST*, **22**(5), 641–652.

T, M. and A., G. (1991). Tolerating latency through software-controlled prefetching in shared-memory multiprocessors. *Journal of Parallel and Distributed Computing*, **12**(2), 87–106.

APÊNDICE

Este apêndice apresenta o resultado de experimentos realizados na plataforma utilizando um benchmark sintético. O objetivo desses experimentos foi avaliar diferentes configurações da plataforma para escolher a que melhor se adequa ao algoritmo proposto. Foram testadas diversas configurações da plataforma até se observar que as alterações que possuíam impacto nos resultados eram aquelas que alteravam parâmetros da Cache L1. Por fim, as opções de configurações da plataforma foram reduzidas às mostradas na Tabela 7.1. A plataforma escolhida foi a com configuração "L1-1-Way", por apresentar menor penalidade para a versão TIMED. Esses experimentos foram realizados para agressividade 4, 8 e 16 e tamanho da janela de previsão de 32.

Na Tabela 7.1 podemos observar três parâmetros variando em relação à Cache L1. O **L1-BLOCK-WIDTH** é o número de palavras por bloco na cache L1. **L1-ENTRY-WIDTH** é o número de entradas da cache e **L1-CACHE-WAYS** é a associatividade da cache.

Tabela 7.1 Configurações da plataforma testadas com benchmark sintético.

CONFIGURAÇÃO	L1-BLOCK-WIDTH	L1-ENTRY-WIDTH	L1-CACHE-WAYS
L1-1-Way	4	8	1
L1-2-Way	4	7	2
L1-4-Way	4	6	4
L1-8-Way	4	5	8
L1-Block-Size-1	1	9	4
L1-Block-Size-2	2	8	4
L1-Block-Size-4	4	7	4
L1-Block-Size-8	8	6	4
L1-Block-Size-16	16	5	4
L1-Block-Size-32	32	4	4

A aplicação do benchmark sintético realiza o envio de arrays de tamanho fixo de uma cache privada para outra, no caso em questão, foram realizados experimentos para arrays de tamanho 4, 8 e 16. Os arrays sempre são armazenados no mesmo endereço e são transferidos para os banco de memória locais, onde sofrem modificações antes de seguir para a cache seguinte.

Gráfico 7.1 Gráfico da penalidade média para a plataforma L1-1-WAY.

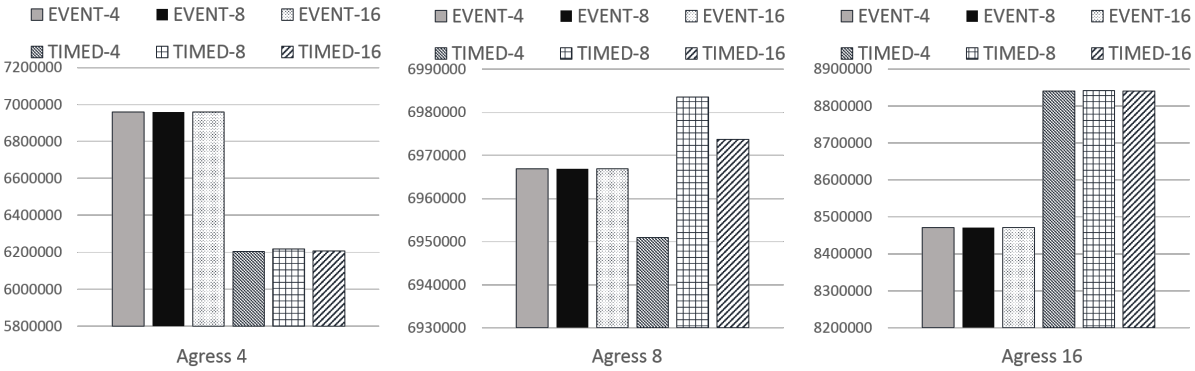


Gráfico 7.2 Gráfico da penalidade média para a plataforma L1-2-WAY.

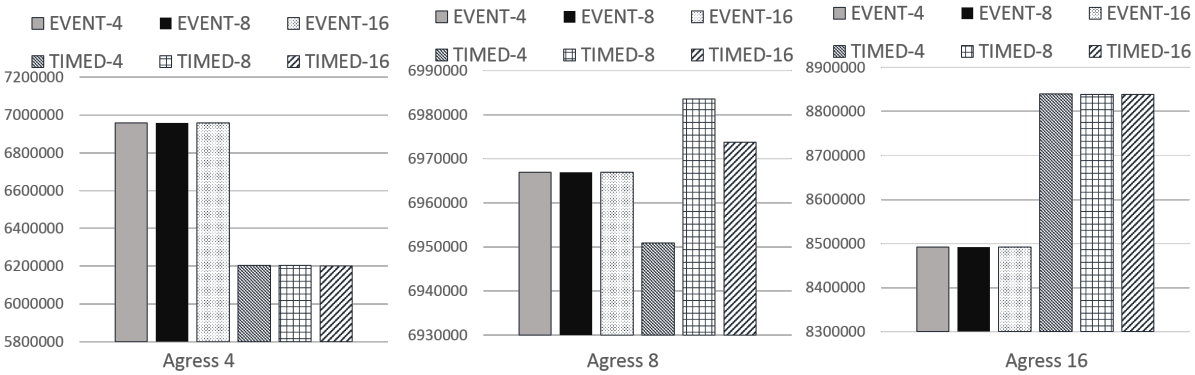


Gráfico 7.3 Gráfico da penalidade média para a plataforma L1-4-WAY.

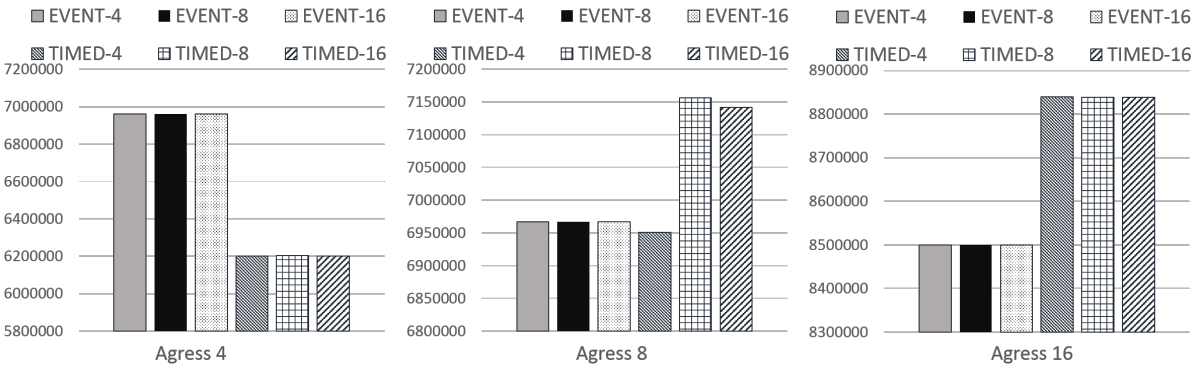


Gráfico 7.4 Gráfico da penalidade média para a plataforma L1-8-WAY.

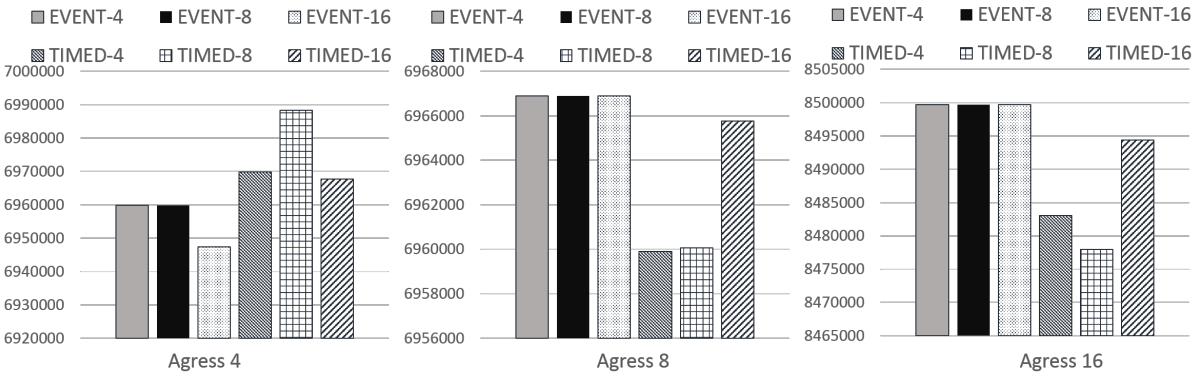


Gráfico 7.5 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-1.

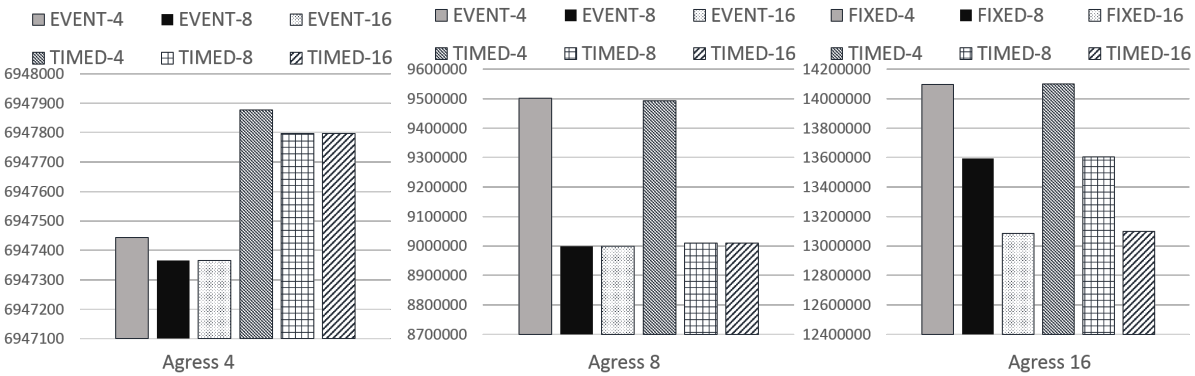


Gráfico 7.6 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-2.

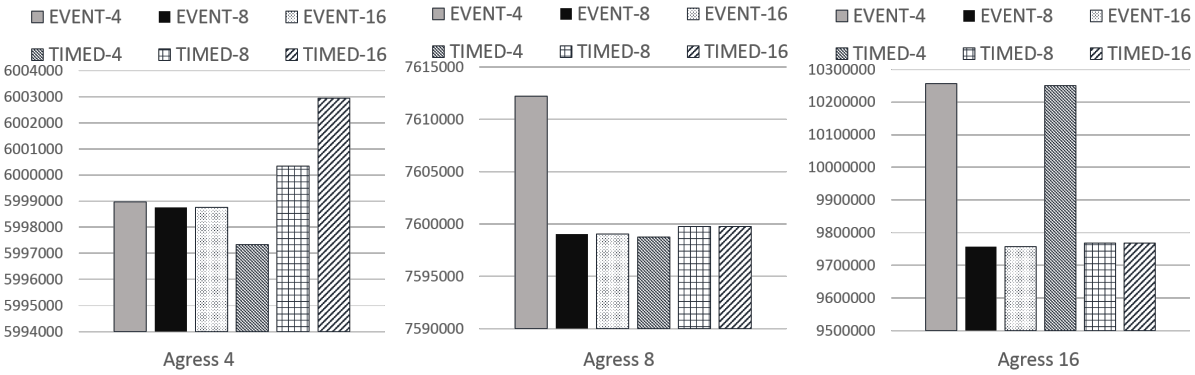


Gráfico 7.7 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-4.

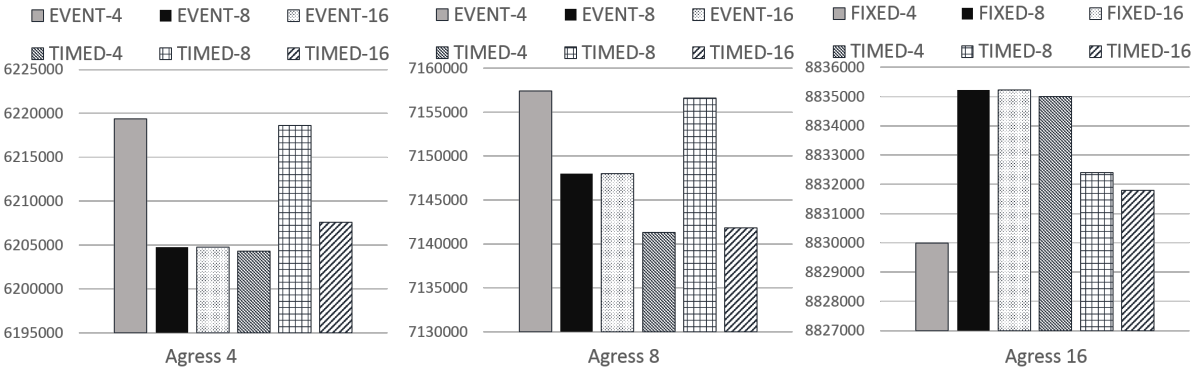


Gráfico 7.8 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-8.

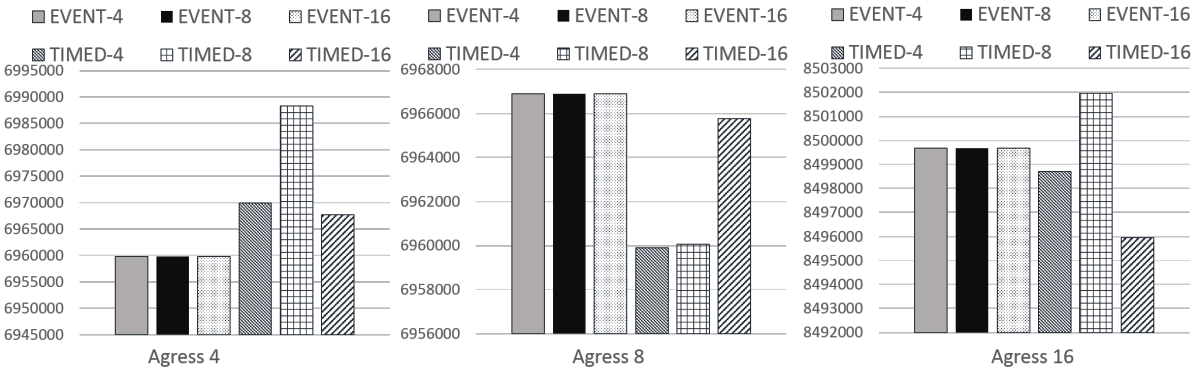


Gráfico 7.9 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-16.

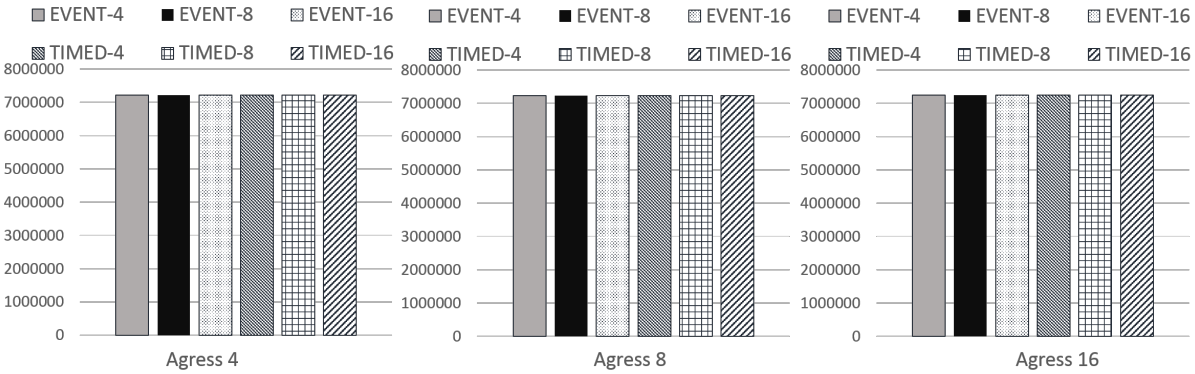


Gráfico 7.10 Gráfico da penalidade média para a plataforma L1-BLOCK-SIZE-32.