FILIPPE MONTEFUSCO SCOTON

Power Laws na modelagem de caches de microprocessadores

São Paulo
2011

FILIPPE MONTEFUSCO SCOTON

Power Laws na modelagem de caches de microprocessadores

Dissertação de Mestrado apresentada
à Escola Politécnica da Universidade
de São Paulo para obtenção do título
de Mestre em Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais

Orientador: Professor Doutor
Mario Donato Marino

São Paulo
2011

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 09 de agosto de 2011.

Assinatura do autor _____

Assinatura do orientador _____

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Catálogo da Publicação
Escola Politécnica da Universidade de São Paulo

FICHA CATALOGRÁFICA

Scoton, Filipe Montefusco

**Power Laws na modelagem de caches de microprocessado - res / F.M. Scoton. -- ed.rev. -- São Paulo, 2011.
112 p.**

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Arquitetura de computadores 2. Engenharia de computação 3. Simulação 4. Geometria e modelagem computacional 5. Métodos gráficos (Estatística) 6. Modelagem de dados I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II. t.

Nome: SCOTON, Filipe Montefusco

Título: Power Laws na modelagem de caches de microprocessadores

Dissertação de Mestrado apresentada
à Escola Politécnica da Universidade
de São Paulo para obtenção do título
de Mestre em Engenharia Elétrica.

Aprovado em: 10/06/2011

Banca Examinadora

Prof. Dr. _____ Instituição: _____
Julgamento: _____ Assinatura: _____

Prof. Dr. _____ Instituição: _____
Julgamento: _____ Assinatura: _____

Prof. Dr. _____ Instituição: _____
Julgamento: _____ Assinatura: _____

Dedico este trabalho à minha mãe, com muito amor, admiração e gratidão por toda sua compreensão, carinho, presença, orações e apoio ao longo destes três anos; e ao meu pai, onde ele estiver, cujas boas lembranças, sentimentos e lições sempre levarei comigo. Espero poder orgulhá-los com este trabalho.

Agradecimentos

Ao Professor Doutor Mario Donato Marino, pela orientação, atenção, apoio e pela cobrança positiva ao longo do mestrado, me mantendo focado e motivado a fazer um trabalho melhor e mais rico. Também agradeço a compreensão da dedicação dividida entre o mestrado e o trabalho, que por vezes resultou em respostas lentas às comunicações.

À minha família, amigos, namorada e pessoas queridas, que sabiamente demonstraram paciência e apoio, perdendo certas vezes minha pouca presença como amigo ou uma presença pouco participativa como filho, irmão e neto. O apoio de todos, cada um à sua maneira, foi fundamental ao longo deste período.

Ao Professor Doutor Jorge Kinoshita pela ajuda no desenvolvimento de parte do tema do trabalho.

Ao *Laboratory of Architecture and High Performance Computing* – LAHPC – e seus integrantes, por colocar à disposição área do laboratório e equipamentos, bem como a preocupação em manter estes equipamentos funcionando apesar dos eventuais problemas de infraestrutura ao longo destes três anos.

À Escola Politécnica da Universidade de São Paulo, pela oportunidade de realização do curso de mestrado.

Aos colegas e professores que desde o início do curso participaram direta ou indiretamente da minha vida acadêmica, cada opinião e conhecimento trocado foram de extrema valia.

"Transportai um punhado de terra todos os dias e farás uma montanha."

(Confúcio).

RESUMO

SCOTON, F.M. Power Laws na modelagem de caches de microprocessadores. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2011.

Power Laws são leis estatísticas que permeiam os mais variados campos do conhecimento humano tais como Biologia, Sociologia, Geografia, Linguística, Astronomia, entre outros, e que têm como característica mais importante a disparidade entre os elementos causadores, ou seja, alguns poucos elementos são responsáveis pela grande maioria dos efeitos. Exemplos famosos são o Princípio de Pareto, a Lei de Zipf e o modelo de Incêndios Florestais. O Princípio de Pareto diz que 80% da riqueza de uma nação está nas mãos de apenas 20% da população; em outras palavras, uma relação causa e efeito chamada 80-20. A Lei de Zipf enuncia que o comportamento da frequência versus o *ranking* de ocorrência é dado por uma curva hiperbólica com um comportamento semelhante a $1/x$. O modelo de Incêndios Florestais modela o comportamento do crescimento de árvores em uma floresta entre sucessivas queimadas que causam destruição de agrupamentos de árvores. As *Power Laws* demonstram que uma porcentagem pequena de uma distribuição tem uma alta frequência de ocorrência, enquanto o restante dos casos que aparecem tem uma frequência baixa, o que levaria a uma reta decrescente em uma escala logarítmica. A partir de simulações utilizando o conjunto de *benchmarks* SPEC-CPU2000, este estudo procura investigar como essas leis estatísticas podem ser utilizadas para entender e melhorar o desempenho de *caches* baseados em diferentes políticas de substituição de linhas de *cache*. O estudo sobre a possibilidade de uma nova política de substituição composta por um *cache* Pareto, bem como um novo mecanismo de chaveamento do comportamento de algoritmos adaptativos de substituição de linhas de *cache*, chamado de *Forest Fire Switching Mechanism*, ambos baseados em *Power Laws*, são propostos a fim de se obter ganhos de desempenho na execução de aplicações.

Palavras-chave: Pareto, Zipf, Power Law, microprocessador, cache, simulação, trace, métodos estatísticos

ABSTRACT

SCOTON, F.M. Power Laws on the modeling of caches of microprocessors. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2011.

Power Laws are statistical laws that permeate the most varied fields of human knowledge such as Biology, Sociology, Geography, Linguistics, Astronomy, among others, and have as most important characteristic the disparity between the cause events, in other words, some few elements are responsible for most of the effects. Famous examples are the Pareto Principle, the Zipf's Law and the Forest Fire model. The Pareto Principle says that 80% of a nation's wealth is in the hands of just 20% of the population; in other words, a cause and effect relationship called 80-20. Zipf's Law states that the behavior of frequency versus ranking of occurrence is given by a hyperbolic curve with a behavior similar to $1/x$. The Forest Fire model represents the behavior of trees growing in a forest between successive fires that cause the destruction of clusters of trees. The Power Laws demonstrate that a small percentage of a distribution has a high frequency of occurrence, while the rest of the cases that appear have a low frequency, which would take to a decreasing line in a logarithmic scale. Based on simulations using the SPEC-CPU2000 benchmarks, this work seeks to investigate how these distributions can be used in order to understand and improve the performance of caches based on different cache line replacement policies. The study about the possibility of a new replacement policy composed by a Pareto cache, and a new switching mechanism of the behavior of cache line replacement adaptive algorithms, called Forest Fire Switching Mechanism, both based on Power Laws, are proposed in order to obtain performance gains on the execution of applications.

Keywords: Pareto, Zipf, Power Law, microprocessor, cache, simulation, trace, statistical methods

LISTA DE ILUSTRAÇÕES

Figura 1 - A figura 1-a representa um CMP com dois cores e caches L1 e L2 privados. A figura 1-b representa um CMP com dois cores com caches L1 privados e um cache L2 compartilhado entre ambos. Observa-se que existem dados em duplicidade (amarelo) nos caches L2 privados, enquanto esse dado tem uma única ocorrência no cache L2 compartilhado.	24
Figura 2 - Exemplo de prefetching com sobreposição de atividades. Figura adaptada de (VAN DER PAAS, 2002).	26
Figura 3 - Organização de um victim cache. Figura adaptada de (JOUPII, 1990).	28
Figura 4 - Porcentagem de homens adultos e estatura em centímetros. Figura extraída de (NEWMAN, 2005).	34
Figura 5 - 5-a - Porcentagem de cidades dos EUA e população destas. 5-b - Apresenta a mesma figura em escala logarítmica. Figuras extraídas de (NEWMAN, 2005).	35
Figura 6 - No modelo de incêndios florestais, um raio atinge uma posição aleatória do terreno onde há uma árvore, iniciando desta forma um incêndio que queimará todo o agrupamento de árvores adjacentes. Figura extraída de (NEWMAN, 2005).	41
Figura 7 - Exemplo de uma curva de distribuição cumulativa do tamanho de incêndios em uma simulação do modelo de incêndios florestais. Figura extraída de (NEWMAN, 2005). ...	43
Figura 8 - Gráficos de entropia do ADEA. Figura 8-a: apresenta o gráfico sem ajuste, com o valor de entropia diminuindo a partir de um limiar igual a 0,38. Figura 8-b: apresenta o gráfico de entropia com o ajuste assumindo um limiar igual a 0,38, fazendo com que a entropia volte a crescer após o limiar. Figuras extraídas de (KOBAYASHI, 2010).	46
Figura 9 - Exemplo de organização do processador incluindo a inserção de um cache Pareto entre os caches L1D e L2. Antes de consultar o L2 haverá consulta ao cache Pareto, de tamanho reduzido e baixa latência de acesso.	53
Figura 10 - Funcionamento simplificado de um cache modelado como uma floresta. Na sequência, o cache possui posições ocupadas em verde e espaços vazios em marrom. O acesso a um novo endereço faz com que o cache o armazene em uma posição vazia. Após três acessos, todas as posições são ocupadas. No quarto acesso (bloco azul) não há espaço no cache, logo a política de substituição escolhe um bloco para ser retirado do cache dando espaço para o armazenamento do novo endereço.	55
Figura 11 - Diferentes padrões de acesso à memória. Operações sobre diferentes working sets representam mudanças de estado do cache e na forma como este é preenchido. A figura 11-a	

mostra o acesso a um array disposto contigualmente na memória principal. A figura 11-b apresenta acessos a variáveis dispostas aleatoriamente na memória. A figura 11-c mostra o acesso a blocos dispostos aleatoriamente, tais como em uma lista ligada.	57
Figura 12 - Funcionamento do mecanismo de chaveamento da política de substituição de linhas de cache. No início a política vigente é a RRF-decay do ADEA. Quando uma instrução de call é executada, há o chaveamento da RRF-decay para a FR-decay. A política vigente se mantém até que seja executada uma instrução de return, quando então haverá outro chaveamento, alterando a política vigente para a RRF-decay.	59
Figura 13 - Curvas de comportamento de acesso ao cache L1 de dados para os benchmarks CRAFTY e GCC.....	69
Figura 14 - Curvas de comportamento de acesso ao cache L1 de dados para os benchmarks APPLU e MESA.....	69
Figura 15 - Curvas de comportamento de acesso ao cache L1 de instruções para os benchmarks CRAFTY e GCC.	70
Figura 16 - Curvas de comportamento de acesso ao cache L1 de instruções para os benchmarks APPLU e MESA.....	70
Figura 17 - Curvas de comportamento de acesso ao cache L2 para os benchmarks CRAFTY e GCC.....	71
Figura 18 - Curvas de comportamento de acesso ao cache L2 para os benchmarks APPLU e MESA.	72
Figura 19 - Tempo de execução em ciclos. Resultados normalizados em relação ao Baseline.	76
Figura 20 - Miss Rate para um cache L2 de 512 kB. Resultados normalizados em relação ao LRU.	79
Figura 21 - Miss Rate para um cache L2 de 1024 kB. Resultados normalizados em relação ao LRU.	79
Figura 22 - Miss Rate para um cache L2 de 2048 kB. Resultados normalizados em relação ao LRU.	79
Figura 23 - Miss Rate para um cache L2 de 2048 kB e associatividade 16. Resultados normalizados em relação ao LRU.	80
Figura 24 - IPC para um cache L2 de 1024 kB. Resultados em valor absoluto.....	82
Figura 25 - Miss Rate para um cache L2 de 1024 kB. Resultados normalizados em relação ao LRU.	83

LISTA DE TABELAS

Tabela 1 - Parâmetros de simulação. A coluna “Parâmetro Pareto” se refere aos parâmetros utilizados na geração dos traces e na análise de desempenho do cache Pareto. A coluna “Parâmetro FFSM” se refere aos parâmetros utilizados na análise de desempenho da implementação do Forest Fire Switching Mechanism e dos algoritmos de substituição de linhas de cache.....	62
Tabela 2 - Tabela de benchmarks do conjunto SPEC-CPU2000. Traduzida de (SPEC, 2009).	63
Tabela 3 - Tabela de coeficientes de inclinação obtidos com a regressão linear das curvas de comportamento de acesso aos caches.....	74

SUMÁRIO

FICHA CATALOGRÁFICA	III
1INTRODUÇÃO.....	15
1.1 MOTIVAÇÃO.....	16
1.2 OBJETIVOS	17
1.3 METODOLOGIA	18
1.4 CONTRIBUIÇÕES	19
1.5 ESTRUTURA DO DOCUMENTO	19
2ESTADO DA ARTE	21
2.1 MECANISMOS DE SUBSTITUIÇÃO DE LINHAS DE CACHE.....	21
2.2 COMPARTILHAMENTO DE CACHES	23
2.3 PREFETCHING	25
2.4 VICTIM CACHE.....	27
2.5 COMPILADORES	29
3MÉTODOS ESTATÍSTICOS E ANALÍTICOS	31
3.1 MODELOS ANALÍTICOS	31
3.2 MODELOS ESTATÍSTICOS	32
3.3 ENTROPIA DA INFORMAÇÃO	32
3.4 POWER LAWS – LEIS DE POTÊNCIA.....	34
3.4.1 O PRINCÍPIO DE PARETO	37
3.4.2 A LEI DE ZIPF	37
3.4.3 O MODELO DE INCÊNDIOS FLORESTAIS	40
4O ALGORITMO ADEA.....	44
4.1 INFRAESTRUTURA DO ADEA	44
4.2 FREQUENCY-RECENCY-BASED DECAY FUNCTION – FR-DECAY	50
4.3 RECENCY-RATHER-FREQUENCY-BASED DECAY FUNCTION – RRF-DECAY	51
5PROPOSTA DE CRIAÇÃO DO CACHE PARETO	52
6FOREST FIRE SWITCHING MECHANISM - FFSM	54
6.1 O CACHE MODELADO COMO UMA FLORESTA	54
6.2 CHAVEAMENTOS DE CONTEXTO DO CACHE.....	55
6.3 APLICAÇÃO DO FFSM.....	58
7METODOLOGIA DE ESTUDO.....	60
7.1 AMBIENTE DE SIMULAÇÃO.....	61

7.2	PARÂMETROS DE SIMULAÇÃO	61
7.3	BENCHMARKS	62
7.4	TRACES	64
7.5	GERAÇÃO DE TRACES	65
7.6	GERAÇÃO DE HISTOGRAMAS E CURVAS DE COMPORTAMENTO	65
7.7	ANÁLISE COMPARATIVA DE ALGORITMOS DE SUBSTITUIÇÃO DE LINHAS DE CACHE	66
8	RESULTADOS E ANÁLISES	67
8.1	ANÁLISE DOS TRACES	67
8.2	CURVAS DE COMPORTAMENTO DE ACESSO	68
8.2.1	CACHE L1 DE DADOS	68
8.2.2	CACHE L1 DE INSTRUÇÕES	69
8.2.3	CACHE L2	71
8.2.4	CONCLUSÕES SOBRE AS CURVAS DE COMPORTAMENTO	72
8.3	RESULTADOS DA REGRESSÃO LINEAR	74
8.4	ANÁLISE DE DESEMPENHO DO CACHE PARETO	75
8.5	FOREST FIRE SWITCHING MECHANISM - FFSM	77
8.5.1	FFSM PARA OS ALGORITMOS DE QURESHI	82
9	CONCLUSÕES E TRABALHOS FUTUROS	85
	REFERÊNCIAS BIBLIOGRÁFICAS	87
	GLOSSÁRIO	95
	APÊNDICE	97
	A.SIMULTANEOUS MULTITHREADING – SMT	97
	B.CHIP MULTIPROCESSORS – CMPs	99
	C.INFRAESTRUTURA DO ADEA	101
	D.CURVAS DE COMPORTAMENTO DE ACESSO	104
	a.CACHE L1 DE DADOS	104
	b.CACHE L1 DE INSTRUÇÕES	105
	c.CACHE L2	107
	E.GRÁFICOS	109
	F.CONCEITOS BÁSICOS SOBRE CACHES	110

1 INTRODUÇÃO

Segundo a famosa Lei de Moore (PATTERSON; HENNESSY, 1990), o poder de processamento de sistemas computacionais deve dobrar a cada 18 meses, exigindo sempre mais dos projetistas.

O desenvolvimento nessa área tem sido cada vez mais difícil, muitas vezes limitado pela própria tecnologia (OLUKOTUN; HAMMOND, 2005). Avanços na microeletrônica e o barateamento de processos de fabricação de deposição de silício têm criado arquiteturas cada vez mais complexas. Isso tem feito com que os processadores atuais sejam cada vez menores, consumam menos, sejam mais densos em termos de transistores por área e núcleos de processamento (*cores*) por *die* e com uma quantidade de memória interna cada vez maior e mais rápida (OLUKOTUN; HAMMOND, 2005). Essa memória interna, conhecida como *cache*, de menor capacidade e latência que a memória principal, permite o aumento do desempenho do processador na execução de aplicações (PATTERSON; HENNESSY, 1990).

Nos processadores atuais, *caches*¹ mais complexos e com capacidade de armazenamento cada vez maior têm sido implementados pela indústria (AMD, 2009), (INTEL, 2009), (IBM, 2009), (SUN, 2009)² e estudados pela comunidade científica. Modelos capazes de se adaptar em tempo de execução (QURESHI et al., 2007), novas políticas de gerenciamento (TYSON et al., 1995), modelos analíticos (GUO; SOLIHIN, 2006), (AGARWAL; HOROWITZ; HENNESSY, 1986) e estatísticos (BERG; ZEFFER; HAGERSTEN, 2006), além de políticas de substituição baseadas no conceito de Entropia da Informação (KOBAYASHI, 2010) são apenas alguns exemplos de linhas de pesquisa na área.

Novos modelos para *caches* podem ser propostos com base em estudos nas áreas de Ciência e Engenharia da Computação ou fora dela, buscando alternativas capazes de fornecer subsídios para novos estudos e avanços na tecnologia atual.

¹ Conceitos básicos sobre *caches* podem ser encontrados no Apêndice F.

² A empresa Sun Microsystems, referenciada neste texto, foi adquirida pela Oracle Corporation, tendo seu processo de aquisição finalizado no dia 27 de janeiro de 2010.

1.1 MOTIVAÇÃO

A hierarquia de memória tem se mantido a mesma ao longo dos últimos anos, adaptando-se às variações no número de *cores* e mudanças de contexto de execução. Como mencionado anteriormente, com o avanço das tecnologias e processos de fabricação, cada vez mais têm surgido novos tipos de memória com latências menores e maior capacidade de armazenamento. Nos processadores, isso implica em *caches* mais rápidos (menor latência) e maiores.

Novos métodos de ganho de desempenho na hierarquia de memória têm se tornado cada vez mais difíceis de serem desenvolvidos. Alterações dos modelos já consolidados permitem pequenos ganhos de desempenho em processadores *single-core*, enquanto o *multithreading* e a capacidade de colocar mais CPUs dentro de um mesmo *die* criam novas demandas por acessos mais eficientes à memória. Tendo em vista esta necessidade de maior eficiência dos acessos, cabe o estudo das políticas de substituição nos *caches* desses multiprocessadores.

Sistemas multiprocessados (PATTERSON; HENNESSY, 1990), CMPs (*chip multi-processors*) (OLUKOTUN et al., 1996) e processadores *multithreaded* (SMT – *Simultaneous Multithreading*) (TULLSEN; EGGERS; LEVY, 1995) fazem uso da execução paralela de instruções. Isso permite que diferentes instruções sejam executadas sobre diversos conjuntos de dados (MIMD), ou que as mesmas instruções sejam executadas sobre dados diferentes (SIMD).

Pesquisas envolvendo o compartilhamento de *caches* em CMPs (KOTERA et al., 2008), (KUMAR; ZYUBAN; TULLSEN, 2005), (AGGARWAL et al., 2007), (ZEBCHUCK; MAKINENI; NEWELL, 2008), (ZHANG; ASANOVIC, 2005), (LEVERICH et al., 2007), (XIE; LOH, 2009), (QURESHI; PATT, 2006), (VARADARAJAN et al., 2006) e SMTs (TULLSEN; EGGERS; LEVY, 1995) apresentam o problema e buscam determinar a melhor maneira de se adaptar o *cache* às diferentes demandas por recursos de acesso à memória. Tal adaptação envolve a porção do *cache* disponível a cada *core* ou a cada *thread* e a política de substituição empregada.

Avaliações estatísticas do funcionamento dos *caches* poderiam indicar como os dados mais relevantes são mantidos nos mesmos, bem como indicar sua tendência de comportamento. Essa avaliação permitiria determinar quais endereços possuem maior probabilidade de serem acessados novamente ao longo da

execução de uma aplicação (BERG; ZEFFER; HAGERSTEN, 2006). Com a informação sobre o comportamento dos acessos em mãos, poder-se-ia criar um mecanismo que, de forma antecipada, mantivesse as informações contidas nesses endereços mais próximas do processador, diminuindo a probabilidade de um *cache miss* (falha de *cache*), o que acarretaria um novo acesso e portanto uma maior latência.

Dentro das análises estatísticas e distribuições de densidade de probabilidade de ocorrência de eventos, uma classe de funções chama a atenção de forma especial, são as chamadas funções do tipo *Power Laws* (Leis de Potência) (NEWMAN, 2005). Tais distribuições têm como característica fundamental a disparidade, de forma que alguns dos seus eventos têm origem em um número muito reduzido de elementos causadores.

Diferente do que ocorre em uma curva normal (ou Gaussiana), em que os valores da distribuição giram em torno de um valor médio com a característica de maior probabilidade de ocorrência, os eventos em uma função do tipo *Power Law* não possuem valor médio, de forma que alguns poucos eventos têm uma probabilidade de ocorrência várias ordens de grandeza maior do que a maior parte do universo de eventos.

As *Power Laws* têm sido estudadas devido não somente à sua peculiaridade, mas também devido ao fato de serem observadas em áreas muito distintas do conhecimento humano, como Biologia (WILLIS; YULE, 1922), Sociologia (ZANETTE; MANRUBIA, 2001), Geografia (MOURA; RIBEIRO, 2006), Ciência da Computação (CROVELLA; BESTAVROS, 1996), Economia (ADAMIC; HUBERMAN, 2000), Astronomia (LU; HAMILTON, 1991), entre outras (NEWMAN, 2005).

1.2 OBJETIVOS

O presente trabalho tem como objetivo estudar o comportamento dos acessos a endereços de memória nos *caches* de microprocessadores a fim de se determinar se esse comportamento é regido por uma função do tipo *Power Law*. Através desse estudo, pretende-se realizar uma verificação do modelo de comportamento de acessos e uma análise matemática da execução de uma dada aplicação sob a ótica de uma *Power Law*.

Este trabalho propõe a utilização de um *cache* Pareto baseado no conceito de *Power Laws* de que uma pequena fração dos acessos é de grande relevância para a execução da aplicação como um todo. Desta forma, seria possível prever o comportamento da aplicação, armazenando as instruções e dados mais relevantes no *cache* Pareto.

Além disso, o trabalho propõe um mecanismo de chaveamento do comportamento de algoritmos de substituição de linhas de *cache*, tornando-os mais aptos a realizar substituições mais eficientes de linhas de *cache*, de acordo com o padrão de acessos à memória. Tal mecanismo recebe o nome de *Forest Fire Switching Mechanism* – FFSM – e como o *cache* Pareto, também se baseia no conceito de *Power Laws*.

1.3 METODOLOGIA

A metodologia deste trabalho consistiu basicamente na realização de uma revisão bibliográfica sobre o tema em questão, o estudo acerca do estado da arte da tecnologia foco, a reunião de material sobre métodos analíticos e estatísticos, bem como sobre Entropia da Informação e *Power Laws*, o problema de desempenho dos *caches* de microprocessadores e possíveis arquiteturas que solucionem tal problema. Em maiores detalhes apresenta-se a ordem de execução do trabalho:

- Estudo sobre *Power Laws* e como determinar se uma massa de dados empíricos segue ou não tais leis;
- Através do uso de simuladores com parâmetros de processadores reais, gerar *traces* (rastros) de execução de conjuntos de *benchmarks* a fim de se obter dados sobre os acessos aos *caches*;
- Verificar o comportamento dos acessos a partir dos *traces*, determinar qual o comportamento de cada aplicação e testar se cada curva segue uma função do tipo *Power Law*;
- Propor e avaliar o desempenho de um mecanismo de chaveamento do comportamento de algoritmos de substituição de linhas de *cache* (*Forest Fire Switching Mechanism*), bem como a criação de um *cache* Pareto, ambos baseados em *Power Laws*.

1.4 CONTRIBUIÇÕES

Este trabalho possui como contribuições:

- Apresentar um estudo sobre o comportamento de acessos a *caches* de microprocessadores baseado no conceito de *Power Laws*, utilizando métodos de regressão linear para a aproximação das curvas de comportamento a retas médias, analisando os coeficientes de inclinação;
- Aplicar o método proposto em (CLAUSET; SHALIZI; NEWMAN, 2007) aos dados extraídos a partir da geração dos *traces* de acessos, a fim de se verificar se o comportamento dos acessos segue ou não uma *Power Law* segundo (CLAUSET; SHALIZI; NEWMAN, 2007);
- Propor uma nova política de substituição através do uso de um *cache* Pareto baseado em *Power Laws*.
- Estender o estudo sobre as funções decaimento do *Adapted Discrete-based Entropy Algorithm* – ADEA – (KOBAYASHI, 2010), além de realizar novas comparações entre as diversas implementações do ADEA e algoritmos reconhecidos pela comunidade científica, como o *LRU Insertion Policy* - LIP - e o *Bimodal Insertion Policy* - BIP - (QURESHI et al., 2007).
- Propor a criação do *Forest Fire Switching Mechanism* – FFSM, mecanismo baseado em *Power Laws* e que é capaz de chavear entre diferentes comportamentos de um algoritmo adaptativo de substituição de linhas de *cache*. Os resultados deste mecanismo foram publicados no artigo (SCOTON; MARINO; KOBAYASHI, 2010).

1.5 ESTRUTURA DO DOCUMENTO

Este trabalho é apresentado dentro da seguinte estrutura:

No capítulo 2 são apresentados resumos sobre o estado da arte da tecnologia tema do trabalho. O capítulo 3 detalha métodos analíticos e estatísticos para o tratamento de dados, bem como a descrição conceitual da Entropia da Informação e *Power Laws*.

O capítulo 4 resume alguns dos aspectos mais relevantes do *Adapted Discrete-based Entropy Algorithm* – ADEA – (KOBAYASHI, 2010) e suas funções decaimento.

A proposta de criação de um *cache* Pareto é apresentada no capítulo 5.

No capítulo 6 é apresentado um mecanismo de chaveamento do comportamento de algoritmos adaptativos de substituição de linhas de *cache*, bem como um exemplo de sua aplicação alternando entre as funções decaimento do ADEA.

O capítulo 7 detalha a metodologia de abordagem do problema e apresenta dois modelos de implementação de *caches* baseados nos estudos realizados, com o objetivo de melhorar o desempenho destes na execução de aplicações.

Os resultados obtidos são apresentados no capítulo 8, no qual também é realizada uma análise seguida de discussão sobre as curvas de frequência versus *ranking*. Ainda no capítulo 8 são discutidas melhorias de desempenho ocasionadas pela inserção do *cache* Pareto além da aplicação do mecanismo de chaveamento entre funções decaimento do ADEA.

No capítulo 9, são apresentadas as conclusões obtidas e trabalhos futuros. Na sequência são listadas as referências bibliográficas, glossário de termos e siglas, além de um apêndice com mais detalhes sobre SMTs e CMPs, bem como gráficos e curvas suprimidos do texto principal.

2 ESTADO DA ARTE

O capítulo apresenta o estado da arte e técnicas de ganho de desempenho para *caches* de microprocessadores. Assume-se que o leitor saiba os conceitos de *multithreading* e *multicore*, ou neste contexto, respectivamente representados por SMT e CMP. Contudo, para contextualizar o leitor, são descritos nos apêndices A e B respectivamente, os conceitos básicos envolvidos nesses dois tipos de arquitetura.

2.1 MECANISMOS DE SUBSTITUIÇÃO DE LINHAS DE CACHE

Um dos aspectos mais importantes que afetam o desempenho de *caches* é a política de substituição de linhas (QURESHI et al., 2007). Dentre eles, os mais famosos são o *Least Recently Used* – LRU, que descarta a linha de *cache* que foi menos recentemente utilizada, privilegiando linhas mais recentes; e o *Most Recently Used* (MRU), que ao contrário do LRU descarta a linha mais recentemente utilizada. Diversos trabalhos já foram publicados nessa área, como (GUO; SOLIHIN, 2006), que propõe um modelo analítico de previsão do desempenho de políticas de *caches*.

O estudo realizado por Zhang, Fan e Liu (2008) apresenta uma nova política de substituição de linhas de *caches* L2 compartilhados chamada de *Light Pollution Replacement* – LPR – que visa substituir dados poluidores do *cache*, armazenados durante a execução de um caminho errado em uma previsão de desvio mal-sucedida.

Etsion e Feitelson (2006) definem políticas de inserção de *caches* capazes de identificar quais linhas de *cache* poderiam ser reutilizadas através da observação de uma pequena janela dos acessos mais recentes à memória.

Na tentativa de adotar diferentes políticas de substituição, Rajan e Ramaswamy (2007) propõem a divisão lógica de um *cache* L2 em dois componentes, um *Shepherd Cache*, com substituição FIFO e um *Main Cache* com uma emulação da substituição ótima. O *Shepherd Cache* possui o duplo papel de armazenar linhas de *cache* e guiar as decisões de substituição no *Main Cache*. Os autores de (ZEBCHUCK; MAKINENI; NEWELL, 2008) propõem mudanças no modelo *Shepherd Cache* (RAJAN; RAMASWAMY, 2007), diminuindo o *overhead* de geração de metadados.

O autor Qureshi explora em seu estudo (QURESHI et al., 2007) as políticas de substituição em *caches* para cargas de trabalho que fazem uso intensivo da memória e que possuem um *working set*³ maior que o tamanho disponível do *cache*. Nestes casos o algoritmo LRU estaria sujeito ao *thrashing*, degradando o desempenho do *cache*. O autor propõe modificações na política de inserção de modo que pelo menos uma fração do *working set* seja mantido no *cache*, contribuindo com *cache hits*. O estudo de Qureshi et al. (2007) apresenta portanto variações do algoritmo LRU tais como o *LRU Insertion Policy* – LIP, que insere novas linhas na posição LRU e somente as promove para a posição MRU após um novo acesso, descartando desta forma rapidamente endereços que serão acessados apenas uma vez. Outro algoritmo apresentado é o *Bimodal Insertion Policy* – BIP, uma variação do LIP que, a partir de uma frequência determinada por um parâmetro chamado bimodal, altera o comportamento do LIP inserindo a nova linha na posição MRU. Por fim, Qureshi et al. (2007) apresenta o algoritmo *Dynamic Insertion Policy* – DIP, capaz de escolher em tempo de execução entre o BIP e o LRU, dependendo da política que incorrer em menos *misses*.

O estudo de Xie e Loh (2009) por sua vez propõe uma nova abordagem que combina inserção dinâmica e políticas de promoção a fim de obter os benefícios do particionamento de *caches*, inserção adaptativa, e roubo de capacidade, tudo em um mesmo mecanismo.

O estudo de Kim, Burger e Keckler (2002) demonstra que uma aplicação interessante para CMPs é a execução de instâncias similares do mesmo programa. Os autores identificam que esse tipo de modelo de execução é bastante comum e que cada instância utiliza dados muito similares. Ainda segundo esse estudo, em hierarquias convencionais de *cache*, cada instância iria armazenar seus próprios dados independentemente. Com isso, o trabalho propõe uma arquitetura de *caches* chamada *Mergeable*, capaz de detectar similaridades entre dados e juntar (*merge*) blocos de *caches*, resultando em economia de espaço de armazenamento no *cache* e, conseqüentemente, diminuição dos acessos *off-chip* (acessos realizados fora do *chip*) e do consumo de energia, além do aumento significativo de desempenho na execução de aplicações.

³ Mais sobre a definição de *working set* pode ser encontrada no estudo de Denning (1968).

Em Somogyi et al. (2009), os autores propõem uma nova classe de mecanismo de predição de *dead blocks*⁴ baseados no que chamam de *bursts*⁵ de acessos aos blocos de um *cache*. Segundo esse estudo, os *bursts* são mais previsíveis que referências individuais, uma vez que escondem as irregularidades destas referências. Utilizado no *cache* L1, o melhor mecanismo de predição foi capaz de identificar, segundo o artigo, 96% dos *dead blocks*. A partir do uso de tal mecanismo, o trabalho propõe formas de eliminação antecipada dos *dead blocks*, aumentando a eficiência do *cache*, destacando-se entre elas a otimização da política de substituição.

Os trabalhos de Biswas et al. (2009) e Hardavellas et al. (2009) estudam os *caches* NUCA – *Non-Uniform Cache Architecture*. Biswas et al. (2009) explora essa não-uniformidade para prover acessos mais rápidos a linhas que estejam armazenadas em porções do *cache* mais próximas do processador. O mesmo trabalho propõe um projeto físico do *cache* e uma política capaz de migrar dados importantes para regiões dentro de um mesmo nível de *cache* que sejam mais próximas do processador. Hardavellas et al. (2009) por sua vez propõe um projeto de *cache* capaz de movimentar blocos para regiões mais adequadas dentro deste, através da classificação de padrões de acesso ao *cache*. O R-NUCA – *Reactive NUCA* – proposto é capaz de cooperar com o sistema operacional a fim de suportar o posicionamento, migração e replicação dos blocos sem o *overhead* de um mecanismo explícito de coerência para o último nível do *cache*. Segundo Hardavellas et al. (2009), ganhos da ordem de 14% foram obtidos em relação a projetos de *caches* similares.

2.2 COMPARTILHAMENTO DE CACHES

Em arquiteturas multiprocessadas, os *caches* podem estar organizados de diversas formas, sendo as mais comuns na forma de *caches* privados (*private*) e compartilhados (*shared*).

Os *caches* privados são exclusivos a um processador ou um *core* dentro de um CMP, oferecendo como vantagem simplicidade de implementação e a maior

⁴ Uma linha de *cache* que é removida do *cache* antes de ser novamente acessada é chamada de *dead block*.

⁵ Um *cache burst* tem início quando um bloco se torna MRU e finaliza quando este se torna não-MRU.

capacidade de armazenamento do *cache*, uma vez que não há contenção do recurso por mais de um *core*.

O *cache* compartilhado, geralmente em níveis mais altos como L2 e L3, permite o compartilhamento de um grande *cache* entre os diversos *cores*. O uso do *cache* pode ser mais eficiente, pois os dados não mais estão armazenados em duplicidade em diferentes *caches* do mesmo nível, melhorando a localidade através da eliminação de *misses* em um dos *caches* (SMITH, 1982).

Figura 1 - A figura 1-a representa um CMP com dois *cores* e *caches* L1 e L2 privados. A figura 1-b representa um CMP com dois *cores* com *caches* L1 privados e um *cache* L2 compartilhado entre ambos. Observa-se que existem dados em duplicidade (amarelo) nos *caches* L2 privados, enquanto esse dado tem uma única ocorrência no *cache* L2 compartilhado.

Implementações como o *Smart Cache* (INTEL SMART CACHE, 2009) trabalham com esse modelo de forma bastante eficiente, ao compartilhar o *cache* L2 de forma totalmente dinâmica, podendo se adaptar às cargas de cada processador, significando que um único *core* pode alocar até 100% da área do *cache* L2 dinamicamente se necessário. O compartilhamento de dados passa a ser mais eficiente, diminuindo o número de *misses* e a própria latência. Além disso, mecanismos adicionais permitem que em processadores *multithreaded* seja possível ter um *core* escrevendo um dado no *cache* enquanto outro esteja lendo outro dado no mesmo momento, sem com isso bloquear o *cache* para um ou outro *core* (INTEL SMART CACHE, 2009).

Os trabalhos de Kotera et al. (2008), Kumar (KUMAR; ZYUBAN; TULLSEN, 2005), Aggarwal et al. (2007) e Zebchuck (ZEBCHUCK; MAKINENI; NEWELL, 2008) fazem uso de modelos de *caches* compartilhados em suas análises, enquanto Zhang (ZHANG; ASANOVIC, 2005) e Leverich et al. (2007) comparam modelos privados e compartilhados. Os dois últimos trabalhos concluem que o

particionamento de *caches* compartilhados permite o aumento da taxa de *hits* (LEVERICH et al, 2007), diminuindo o tráfego fora do *chip* (ZHANG; ASANOVIC, 2005).

2.3 PREFETCHING

Chen e Baer (1992) mencionam que a técnica de *prefetching* em *hardware* ou *software* pode eliminar a penalidade de uma falha de *cache* ao gerar pedidos de *prefetch* ao sistema de memória para trazer dados para dentro do *cache* antes que estes sejam de fato utilizados. Tal técnica explora a sobreposição de cálculos de forma antecipada a uma falha de *cache*.

O *prefetching* reduz a latência de memória ao trazer dados de forma antecipada ao invés de trazê-los sob demanda para um nível da hierarquia de memória que esteja mais próximo ao processador (VAN DER PAAS, 2002). A técnica pode ser baseada em *hardware*, *software* ou uma combinação dos dois. As principais vantagens de uma abordagem baseada no *prefetching* por *hardware* é que cada *prefetch* é manipulado dinamicamente sem a intervenção do compilador, sendo mantida a compatibilidade do código. Entretanto, para a implementação em *hardware*, recursos extras são necessários e dados não desejados podem ser buscados. Em contraste, uma abordagem baseada em *software* reside em padrões de acesso a dados detectados por programas estáticos de análise e permite que o *prefetching* seja feito de forma seletiva (VAN DER PAAS, 2002) e (HUR; LIN, 2006). O problema desta abordagem está principalmente relacionado ao *overhead* de execução devido a instruções extras de *prefetching*.

Figura 2 - Exemplo de *prefetching* com sobreposição de atividades. Figura adaptada de (VAN DER PAAS, 2002).

A técnica de *prefetching* em geral considera os seguintes pontos (CHEN; BAER, 1992):

- Selecionar quais acessos poderão sofrer *cache miss*, pois estes acessos serão alvo de uma busca antecipada;
- Selecionar onde a instrução será posicionada, sendo o *prefetch* executado suficientemente antes da necessidade de uma instrução;
- Saber o endereço de memória a ser utilizado na instrução de *prefetch*.

Chen e Baer (1992) avaliam a técnica de *prefetching* juntamente com a de *caches* não-bloqueantes e propõem um modelo híbrido. Qureshi et al. (2007) menciona o problema de poluição do *cache* devido a *prefetchings* de baixa acurácia. Leverich et al. (2007) cita possíveis implementações de mecanismos de *prefetching* feitos por *software* e por *hardware*.

Os estudos de Zebchuck (ZEBCHUCK; SAFI, MOSHOVOS, 2007) e Cantin (CANTIN; LIPASTI; SMITH, 2006) propõem um *framework* para otimizações em que um dos seus componentes é um mecanismo de *prefetching* chamado *Stealth Prefetching*. Esse mecanismo rastreia quais blocos em uma determinada região

sofreram buscas anteriores e utiliza essas informações para realizar posteriormente *prefetches* desses blocos.

O conceito de *stream buffers* é sugerido por Naz et al. (2004). Os *stream buffers* nada mais são do que pequenas memórias com o objetivo de armazenar blocos buscados através de *prefetching*, ao invés de estes blocos serem armazenados no *cache*, evitando possível poluição.

O trabalho de Chou (2007) propõe um *prefetcher* baseado em correlação chamado *epoch-based*, que armazena sua tabela de correlação na memória principal e explora o conceito de *epochs* (período de cálculo *on-chip* adicionado ao acesso *off-chip*) para esconder a longa latência de acesso a essa tabela.

Diaz e Cintra (2009) criam uma nova classe de *prefetchers* baseada no encadeamento de várias sequências (*streams*) de endereços que tenham sofrido *miss*. Dessa forma o mecanismo de *prefetching* não mais se limita a uma busca profunda por uma instrução de acesso à memória que tenha sofrido um *miss*, mas pode de forma adaptativa, buscar outras instruções mais próximas no tempo, atingindo melhor desempenho.

O mesmo conceito de *streams* é utilizado por Liu et al. (2008), que propõe um modelo chamado STeMS – *Spatio-Temporal Memory Streaming*. O modelo é capaz de gravar e reproduzir a sequência temporal de acessos a uma região e utiliza relacionamentos espaciais dentro de cada região para reconstruir dinamicamente toda uma ordem prevista de *misses*, obtendo ganhos significativos de desempenho contra outros modelos de previsão.

2.4 VICTIM CACHE

Os *victim caches* foram originalmente propostos por Jouppi (1990) como uma abordagem que tinha como objetivo a redução de conflitos de *cache misses* em *caches* com mapeamento direto, sem contudo afetar seu tempo de acesso rápido. O *victim cache* é um *cache* totalmente associativo (*full associative*), de tamanho pequeno e que reside entre um nível de *cache* e outro ou entre um nível de *cache* e a memória principal. Em um *cache miss*, antes de ir ao próximo nível, o *victim cache* é verificado. Caso o endereço seja encontrado no *victim cache*, o dado desejado é retornado à CPU e também promovido ao nível principal de *cache*, havendo uma

permutação entre o dado no *cache* e o dado no *victim cache*. No caso de um *miss* no *victim cache*, o próximo nível da hierarquia de memória é acessado e o dado encontrado preenche a linha no nível principal de *cache* enquanto o dado atual é movido para o *victim*. Neste caso a entrada no *victim cache* é descartada e, se tiver sofrido alteração, é escrita no próximo nível da memória (NAZ et al., 2004).

Figura 3 - Organização de um *victim cache*. Figura adaptada de (JOUPII, 1990).

O *victim cache* armazena temporariamente dados retirados do *cache* e, devido à sua total associatividade, pode simultaneamente armazenar blocos que conflitariam em *caches* com mapeamento direto. Se o número de blocos conflitantes for suficientemente pequeno para caber no *victim cache*, tanto a taxa de falhas com relação ao próximo nível da hierarquia de memória quanto o tempo médio de acesso serão melhorados devido à penalidade menor de acesso a dados no *victim cache* (JOUPII, 1990).

Zhang e Asanovic (2005) fazem uso de vários *victim caches* em um CMP. Os autores apresentam uma política de gerenciamento de *cache* chamada *victim replication*, ao combinar as vantagens de *caches* privados e de *caches* compartilhados. O *victim replication* (ZHANG; ASANOVIC, 2005) é uma variação dos *caches* compartilhados que tenta manter cópias de *victim caches* primários locais dentro do *slice* local do *cache* L2. *Hits* nas cópias replicadas reduzem a latência efetiva do *cache* L2 compartilhado, enquanto mantém os benefícios de uma capacidade de dados maior com o compartilhamento de *cache*. Entre outros resultados alcançados pelos autores, destaca-se a diminuição da latência média de acesso à memória dentro do *chip* por meio da replicação das linhas de *cache* dentro do mesmo nível, local aos processadores ativamente acessando a linha.

Beckmann, Marty e Wood (2006) apresentam um modelo de controle de replicação dinâmico, mais eficiente que o *victim replication*, que é baseado em regras estáticas de replicação. As regras estáticas de replicação não podem se adaptar dinamicamente ao comportamento de diferentes cargas de trabalho. No modelo proposto neste trabalho (BECKMANN; MARTY; WOOD, 2006), pode-se dinamicamente estimar o custo (*misses* adicionais) e o benefício (latência de *hit* reduzida) da replicação, ajustando seu nível para reduzir o tempo médio de acesso. Como resultado deste trabalho, foram obtidas melhorias de desempenho na execução de aplicações comerciais com grandes *working sets*, frente aos modelos baseados em regras estáticas.

2.5 COMPILADORES

Segundo o estudo realizado por Hwu (1995), avanços na tecnologia de *hardware* têm permitido aos processadores executar uma grande quantidade de instruções de forma paralela, aumentando a velocidade de execução de aplicações. Assim como em outras formas de processamento paralelo, o desempenho dos multiprocessadores pode variar de acordo com a qualidade de *software*, onde os compiladores desempenham um papel crucial (HWU, 1995). Este mesmo trabalho apresenta os componentes críticos da tecnologia que lida com as dificuldades da compilação de programas, estudando compiladores capazes de extrair grandes níveis de paralelismo em nível de instrução.

Cheong e Veidenbaum (1990) discutem em seu trabalho a necessidade de se encontrar alternativas para estratégias de coerência de *caches* baseadas em *hardware* para multiprocessadores de larga escala. Esse mesmo trabalho propõe três diferentes estratégias baseadas em *software*, consistindo de uma abordagem de invalidação simples, um esquema de invalidação seletivo e um esquema baseado em controle de versão. Segundo (CHEONG; VEIDENBAUM, 1990), essas estratégias são cabíveis a multiprocessadores com memória compartilhada, longas redes de interconexões e um grande número de processadores. Por fim, os autores apresentam uma comparação de desempenho entre os três esquemas.

Os artigos de Mowry (1998) e Gornish (GORNISH; GRANSTON; VEIDENBAUM, 1990) apresentam técnicas de *prefetching* controladas por *software*, de modo que o compilador insira automaticamente instruções de *prefetching* no código. Os resultados de simulações mostram ganhos significativos na execução de aplicações paralelas, podendo chegar a um fator de 2 (MOWRY, 1998).

Segundo Calder et al. (1998), técnicas de compiladores têm sido utilizadas para melhorar o desempenho do *cache* de instruções. Isso ocorre através do mapeamento de código com localidade temporal para diferentes blocos de *cache* dentro do espaço de endereço virtual, eliminando conflitos de *cache*. Calder et al. (1998) apresenta uma abordagem direcionada pelo compilador que cria endereços de posicionamento para a pilha, variáveis locais e globais, objetos de *heap* e constantes, com o objetivo de reduzir a taxa de *misses* em *caches* de dados, seguindo o mesmo tipo de técnica mencionada para o *cache* de instruções, ou seja, fazendo uso da localidade temporal. O trabalho de Chilimbi e Shaham (2006) estende o estudo apresentado por Calder et al. (1998) ao isolar e combinar sítios (*sites*) de alocação de instâncias de objetos no mesmo bloco de *cache*. Em ambos os trabalhos, há diminuição no tempo de execução de aplicações, melhorando o desempenho.

3 MÉTODOS ESTATÍSTICOS E ANALÍTICOS

O capítulo discute a respeito de métodos estatísticos e analíticos, *traces* e modelagem matemática de *caches*. Além disso, apresenta os principais conceitos relacionados à Entropia da Informação e *Power Laws*.

3.1 MODELOS ANALÍTICOS

No estudo realizado por Agarwal, Horowitz e Hennessy (1986), um modelo analítico de *cache* é proposto com a ideia de utilizar parâmetros extraídos de *traces* com endereços de programas, dessa forma seria possível fornecer de maneira eficiente estimativas do desempenho de *caches*, além de demonstrar os efeitos da variação de parâmetros dos mesmos. Através da representação de fatores que afetam o desempenho de *caches*, os autores desenvolveram um modelo analítico que fornece as taxas de *misses* para um dado *trace*. O trabalho tinha como objetivo encontrar uma forma de obter resultados com menor custo computacional e acurácia próxima a simulações com *traces*.

Guo e Solihin (2006) estuda as políticas de substituição em *caches* com o objetivo de criar um modelo analítico capaz de prever o desempenho dessas políticas. O modelo é baseado na teoria da probabilidade e utiliza processos de Markov para calcular cada probabilidade de *miss* em um acesso ao *cache*. A partir de sequências circulares de *profiling*, o modelo fornece como saída previsões quanto às taxas de *misses* de uma aplicação utilizando diferentes políticas de substituição. Esse mesmo modelo mostra-se bastante acurado, com erros médios de predição de 1,41% e tempo de execução menor que 0,1 segundo, bem menor que o tempo de simulações baseadas em *traces*.

Os trabalhos (MADAN; BALASUBRAMONIAN, 2007), (MURALIMANO HAR; BALASUBRAMONIAN, 2007) e (MURALIMANO HAR; BALASUBRAMONIAN; JOUPPI, 2007) utilizam a ferramenta CACTI (2009), que faz uso de um modelo analítico para estimar o tempo de acesso, *layout* e consumo de energia de *caches* a partir de um conjunto de parâmetros, como capacidade de armazenamento do *cache*, tamanho do bloco, associatividade, processo de fabricação, número de portas e número de bancos (*banks*) independentes.

3.2 MODELOS ESTATÍSTICOS

Berg, Zeffer e Hagersten (2006) propõem um modelo matemático do sistema de memória capaz de prever as taxas de *misses* em *caches* baseado em dados extraídos por amostragem. A acurácia do método foi avaliada através da comparação com simulações de aplicações, sendo que os resultados produzidos pelos algoritmos se mostraram consistentes com aqueles obtidos com um simulador tradicional.

Petoumenos et al. (2006) apresenta e avalia um modelo estatístico de *caches* compartilhados e uma forma de gerenciamento das suas linhas em CMPs. O modelo é alimentado com informações sobre o número de referências à memória entre dois acessos consecutivos à mesma linha de *cache* realizados por cada *thread*. Desta forma, esse modelo é capaz de descrever o comportamento dessas *threads* compartilhadas bem como dizer quais destas poderiam ser comprimidas em um espaço menor do *cache* sem danos perceptíveis.

A patente de Salsburg (ZITEL CO., 1993) descreve um método e estrutura capazes de criar um modelo estatístico que quantifica a localidade dos dados e assim seleciona os elementos que devem ser armazenados em *cache* ou aqueles que não devem ser armazenados. O modelo é utilizado para prever os efeitos de inclusão ou remoção de um elemento do conjunto de todos os elementos que serão armazenados em *cache*. Por fim os autores descrevem que o método e estrutura da invenção podem ser úteis para a análise de sistemas mensuráveis e produção de relatórios de desempenho, além de poder ser utilizado como um esquema de gerenciamento de *caches*, de forma dinâmica e em tempo-real, com o objetivo de otimização de desempenho.

3.3 ENTROPIA DA INFORMAÇÃO

Em 1951, o matemático Claude E. Shannon definiu a base para a Teoria da Informação (SHANNON, 1951). A Entropia se refere à medida da informação contida dentro de uma mensagem, que por sua vez representa uma sequência de caracteres que pertencem a uma determinada linguagem. De acordo com o conceito de Entropia da Informação, cada ocorrência de um caractere pode ser interpretada como um evento desconexo. Isto significa que a ocorrência de um determinado

caractere não influencia a ocorrência de qualquer outro caractere dentro da mesma mensagem; entretanto, a observação de quais caracteres ocorreram previamente influencia as chances de se prever o próximo caractere a aparecer na mensagem, o que aumenta as chances de se prever futuras características e pode ser entendido como conhecimento ou medida da informação já contida no trecho de mensagem transmitido. Tal conhecimento ou porção de informação foi nomeado por Shannon como entropia da mensagem.

Para estimar a entropia de uma determinada mensagem, a ocorrência de caracteres pode ser modelada na forma de variáveis aleatórias, cada uma com certa probabilidade associada. A partir dessas distribuições de probabilidade ($p(x_i)$), a incerteza de cada caractere ($u(x_i)$) pode ser avaliada a fim de se estimar a entropia de toda uma mensagem. Para cada x_i que pertence a uma linguagem, tal medida de incerteza ($u(x_i)$) pode ser definida pela relação a seguir:

$$u(x_i) = \log_b \frac{1}{p(x_i)} = -\log_b p(x_i), \text{ em que } b \in \{2, e, 10\} \quad (\text{I})$$

A entropia discreta de um caractere $h(x_i)$ e a entropia de uma sequência de caracteres $H(X = x_i)$ podem ser então estimadas pela relação:

$$h(x_i) = u(x_i) * p(x_i), i \in \{1, \dots, n\}$$

$$H(X = x_i) = \sum_{i=1}^n p(x_i) * u(x_i) \quad (\text{II})$$

Um aspecto característico da Entropia da Informação é que cada variável aleatória da sequência tem a mesma probabilidade de ocorrência. Nesta situação em particular, a entropia de tal sequência seria máxima. Uma vez que todas as variáveis possuem a mesma probabilidade de ocorrência, a incerteza possui seu nível mais elevado.

3.4 POWER LAWS – LEIS DE POTÊNCIA

Segundo o estudo de Newman (2005), muitos dos objetos de medidas realizadas por cientistas possuem um valor típico ou escala, ao redor do qual as medidas são centradas. Um exemplo simples disso, citado no mesmo artigo, é o das estaturas de seres humanos. A maioria dos humanos adultos possui cerca de 180 cm. Apesar de variações quanto a esse valor, especialmente no que diz respeito ao sexo, não é costumeiro ver adultos com 10 ou 500 cm de altura.

A figura 4 mostra um histograma das estaturas em centímetros de homens adultos nos Estados Unidos da América - EUA, com medições realizadas entre 1959 e 1962. Nela pode-se observar uma distribuição bastante concentrada, do tipo normal – Gaussiana - com pico ao redor de 180 cm. Dessa forma, pode-se concluir que o valor típico de estatura de um homem adulto é de cerca de 180 cm.

Figura 4 - Porcentagem de homens adultos e estatura em centímetros. Figura extraída de (NEWMAN, 2005).

O artigo de Newman (2005) cita ainda a proporção de estatura entre a pessoa mais alta e a mais baixa, avaliando as medidas de pessoas presentes no *Guinness Book of Records* – Livro dos Recordes. No livro, a pessoa mais alta do mundo mede 272 cm, enquanto a mais baixa mede 57 cm, fazendo com que a razão entre uma e outra medida seja de 4,8. Essa razão, com valor de 4,8, pode ser considerada como uma razão de valor pequeno, uma vez que existem variações de muitas ordens de grandeza. Um exemplo clássico dessa disparidade de valores é, segundo (NEWMAN, 2005), o tamanho de cidades nos EUA.

A maior população dentre todas as cidades dos Estados Unidos se encontra em New York City, com cerca de 8 milhões de habitantes segundo censo realizado na cidade em 2000. A cidade com a menor população, segundo o *Guinness Book*, é a cidade de Duffield no estado da Virginia, com uma população de 52 habitantes. A razão entre uma medida e outra nesse caso é de pelo menos 150.000, claramente diferente da proporção apresentada para estaturas de homens adultos (4,8), o que demonstra grande disparidade dos valores máximo e mínimo no que diz respeito ao tamanho de uma cidade dos EUA.

Outra observação interessante é apresentada por Newman em seu trabalho (2005) através de um histograma do tamanho das cidades, mostrado na figura 5-a. A curva apresenta uma longa extensão na sua porção direita, significando que enquanto uma grande porção da distribuição ocorre para cidades similarmente pequenas – a maioria das cidades dos EUA têm populações pequenas – existe um número pequeno de cidades com uma população muito maior do que o valor típico, o que produz essa extensão. O formato dessa distribuição é muito diferente do apresentado anteriormente para estaturas de pessoas, em que havia extensões laterais de tamanho semelhante ao redor do valor típico – ou de maior porcentagem dentre os homens adultos.

Figura 5 - 5-a - Porcentagem de cidades dos EUA e população destas. 5-b - Apresenta a mesma figura em escala logarítmica. Figuras extraídas de (NEWMAN, 2005).

Considerando-se um país como os EUA, que possui cerca de 300 milhões de habitantes, não poderiam existir mais que 40 cidades neste país com o tamanho de

New York City ou, em outra análise, suas 2.700 cidades não poderiam possuir uma população média maior que 110 mil habitantes ($3 \times 10^8 / 2700$).

Tomando-se como hipótese o valor médio de 110 mil habitantes por cidade e seguindo o exemplo de estaturas de seres humanos, a curva do tamanho das cidades dos EUA deveria ser uma normal com valor de pico próximo a 110 mil. No entanto, avaliando-se novamente a figura 5-a, pode-se perceber que a curva não reflete essa hipótese, visto que a maior parcela das cidades possui população de até 25 mil habitantes e sua maior extensão segue apenas uma das laterais⁶.

Newman (2005) ainda apresenta outra figura, representada em 5-b, na qual o histograma do tamanho das cidades é plotado novamente, porém em escala logarítmica.

Quando apresentado dessa forma, é possível perceber a proximidade dos pontos a uma linha reta. Ainda segundo Newman (2005), essa observação parece ter sido realizada pela primeira vez por Auerbach (1913), no entanto sendo usualmente atribuída a Zipf (1932, 1949).

Para avaliar o significado dessa observação, seja $p(x)dx$, a fração de cidades com população entre x e $x + dx$.

Sendo o histograma uma linha reta em uma escala logarítmica, seguindo a relação de linearização observada na figura 5, então:

$$\ln p(x) = -\alpha \ln x + c, \text{ onde } \alpha \text{ e } c \text{ são constantes.}$$

Tomando-se a exponencial dos dois lados, isso equivale a:

$$p(x) = Cx^{-\alpha}, \text{ com } C = e^c. \quad (\text{III})$$

Distribuições da forma (III) são chamadas *Power Laws*, em que a constante α é chamada expoente da *Power Law*.

O estudo de Newman (2005) também menciona que distribuições do tipo *Power Law* ocorrem em uma vasta e variada gama de fenômenos. Exemplos apresentados no artigo são: o tamanho de terremotos (GUTENBERG; RICHTER, 1944), de crateras da Lua (NEUKUM; IVANOV, 1994), de labaredas solares (LU; HAMILTON, 1991) e de guerras (ROBERTS; TURCOTTE, 1998), a frequência do uso de palavras em uma língua humana (ZIPF, 1932; 1949) e de ocorrência de nomes na maioria das culturas (ZANETTE. MANRUBIA, 2001), o número de artigos escritos por cientistas (LOTKA, 1926), de citações a estes artigos (PRICE, 1965), de

⁶ Na figura 5-a não é possível a observação de cidades com menos de 25 mil habitantes devido à pequena porcentagem destas e da escala de plotagem do gráfico.

hits em páginas *web* (ADAMIC; HUBERMAN, 2000), as vendas de livros (KOHLI; SAH, 2003), de gravações musicais (COX; FELTON; CHUNG, 1995) e de quase todos os outros produtos considerados como *commodities* (KOHLI; SAH, 2003), o número de espécies em um *taxa*⁷ biológico (WILLIS; YULE, 1922) e a distribuição de riqueza em um país (PARETO, 1896).

3.4.1 O PRINCÍPIO DE PARETO

A observação original do Princípio de Pareto foi realizada sobre a conexão entre ganhos monetários e riquezas. Vilfredo Pareto, economista que viveu no início do século XX, percebeu que 80% da riqueza da Itália estava concentrada nas mãos de apenas 20% da população (PARETO, 1896). Posteriormente, Pareto realizou pesquisas sobre uma variedade de outros países e descobriu que uma distribuição similar era aplicável.

Um gráfico que deu à desigualdade uma forma bastante visível e compreensível foi abordado no Relatório do Programa de Desenvolvimento das Nações Unidas de 1993, que mostrou que a distribuição global de importações era bastante diferenciada, com 20% da população do mundo controlando 82,7% dos ganhos monetários globais (PNUD, 1993).

3.4.2 A LEI DE ZIPF

A Lei de Zipf enuncia que para um determinado corpo de texto baseado em linguagem natural, a frequência de cada palavra é inversamente proporcional à sua posição na tabela de frequências (ZIPF, 1932, 1949). Sendo assim, a palavra mais frequente irá ocorrer aproximadamente duas vezes mais que a segunda palavra mais frequente no texto, que por sua vez ocorrerá duas vezes mais que a quarta palavra mais frequente, e assim por diante.

O trabalho original de Zipf (1932) usa como exemplo um corpo de texto, onde a palavra “*the*” é a de maior frequência, contribuindo por si só com 7% de todas as ocorrências de palavras (69.971 de aproximadamente 1 milhão). De acordo com a

⁷ *Taxa*, plural de *táxon*, que por sua vez denomina uma unidade taxonômica, essencialmente associada a um sistema de classificação, podendo estar em qualquer nível dentro deste sistema.

Lei de Zipf, a segunda palavra mais frequente “of” contabiliza aproximadamente 3,5% do total de palavras (36.411 ocorrências), seguida por “and” (28.852). Seguindo essa linha, percebe-se que somente 135 vocábulos são necessários para escrever metade do corpo de texto (ZIPF, 1932).

A Lei de Zipf é mais facilmente observada plotando-se os dados em um gráfico com escala logarítmica, com os eixos sendo $\log(\text{frequência})$ versus $\log(\text{posição na tabela de frequências} - \textit{ranking})$. Por exemplo, “the” como descrito anteriormente apareceria em $x = \log(1), y = \log(69971)$. Dessa forma, os dados seguem a Lei de Zipf à medida que os pontos plotados parecem se dispor ao longo de um segmento linear.

Formalmente seja⁸:

- N o número de elementos;
- k a posição na tabela de frequências;
- s o valor do expoente caracterizando a distribuição.

Então, a Lei de Zipf, de acordo com suas formulações encontradas em (ZIPF, 1932, 1949), prevê que de uma população com N elementos, a frequência de elementos de posição k , $f(k; s, N)$, é:

$$f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^N (1/n^s)} \quad (\text{IV})$$

No exemplo de frequência de palavras na língua inglesa, N representa o vocabulário - número de palavras na língua inglesa utilizadas no texto – e, se utilizada a versão clássica da Lei de Zipf, o expoente s é igual a 1. Tal frequência de elementos $f(k; s, N)$ será então a fração de vezes que a k -ésima palavra mais frequente ocorre. É visível⁸ que a distribuição é normalizada, isto é, as frequências previsíveis somam 1:

$$\sum_{k=1}^N f(k; s, N) = 1 \quad (\text{V})$$

⁸ Descrições formais mais elaboradas podem ser encontradas em (CLAUSET; SHALIZI; NEWMAN, 2007), (MITZENMACHER, 2004), (LI, 1992) e (ADAMIC; HUBERMAN, 2000).

A distribuição também pode ser escrita:

$$f(k; s, N) = \frac{1}{k^s H_{\{N, s\}}}, \text{ em que } H_{\{N, s\}} \text{ é o } n\text{-ésimo número harmônico}$$

generalizado.

O caso mais simples da Lei de Zipf é uma função do tipo $1/f$, citado em (SHANNON, 1951), que estuda a previsão de palavras na língua inglesa. Dado um conjunto de frequências distribuídas de acordo com a Lei de Zipf, ordenadas da mais comum para a menos comum, a segunda frequência mais comum ocorrerá metade do número de vezes da mais comum. A terceira frequência ocorrerá $1/3$ das vezes da primeira. A n -ésima frequência mais comum ocorrerá $1/n$ vezes a primeira frequência mais comum. Entretanto, essa relação não é precisa, pois as ocorrências devem seguir um número inteiro de vezes, ou seja, uma palavra não pode ocorrer 2,5 vezes. Mas considerando-se grandes amostras e uma boa aproximação, muitos fenômenos naturais seguem a Lei de Zipf. Matematicamente, é impossível para a versão clássica da lei ocorrer de forma precisa se existem infinitas palavras em uma linguagem, uma vez que a soma de todas as frequências relativas no denominador é igual à série harmônica, portanto:

$$\sum_{n=1}^{\infty} \frac{1}{n} = \infty$$

(VI)

No inglês, as frequências das palavras podem ser modeladas razoavelmente bem com a Lei de Zipf com um s próximo de 1 (SHANNON, 1951). Enquanto o expoente s exceder 1, é possível para tal distribuição servir para infinitas palavras, uma vez que $s > 1$, então:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} < \infty, \text{ em que } \zeta \text{ representa a função zeta de Riemann. (VII)}$$

Estudos têm mostrado que a Lei de Zipf regula *traces* de *web caches* (BRESLAU et al., 1999). Neste trabalho, os autores concluem que esses *traces* não seguem a Lei de Zipf com precisão, no entanto possuem um comportamento assintótico muito semelhante, sugerindo que propriedades observadas de taxas de acerto e localidade temporal são inerentes a acessos *web* observados em *proxies*.

3.4.3 O MODELO DE INCÊNDIOS FLORESTAIS

Em seu estudo, Newman (2005) cita mecanismos para geração de distribuições que seguem *Power Laws*. Um destes exemplos de mecanismos recebe o nome de Criticidade Auto-Organizada (*Self-organized criticality*).

Segundo Newman (2005), alguns sistemas são capazes de desenvolver distribuições que seguem *Power Laws* em pontos críticos do seu espaço de parâmetros devido à divergência de uma determinada escala⁹, como o tamanho médio de um agrupamento em um modelo de percolação¹⁰, o que leva tais sistemas a serem livres de escala. Contudo, Newman também afirma que isso não fornece uma explicação plausível para a origem de *Power Laws* em sistemas reais uma vez que, com parâmetros do mundo real, dificilmente seria possível ter tais pontos em que a divergência ocorreria.

Apesar de tal dificuldade, Newman menciona em seu artigo (2005) que alguns sistemas dinâmicos conseguem se arranjar de tal forma que sempre se acomodam sobre o ponto crítico, independente do estado de início. Diz-se de tais sistemas que se auto-organizam para o ponto crítico, ou em outras palavras, que apresentam uma criticidade auto-organizada. Um exemplo de tal tipo de sistema é o modelo de incêndios florestais de Drossel e Schwabl (1992), que por sua vez é baseado em um modelo de percolação.

Drossel e Schwabl (1992) imaginam um terreno vazio como uma grade, ou seja, um quadrado subdividido em diversos outros pequenos quadrados de tamanhos iguais. Neste terreno, árvores podem nascer em qualquer lugar devido a processos naturais como a dispersão biológica, tais como a dispersão de sementes feita por animais, pelo vento ou pela água. No modelo, árvores aparecem instantaneamente e aleatoriamente a uma taxa constante, sempre ocupando um quadrado da grade, sendo que apenas uma árvore pode ocupar um quadrado. Desta

⁹ Um exemplo de divergência é citado em (NEWMAN, 2005), no qual um ímã possui um comprimento de correlação que mede o tamanho típico de domínios magnéticos. Sob determinadas circunstâncias esta escala baseada em comprimento pode divergir, levando o sistema a não ter qualquer tipo de escala, portanto livre de escalas. Isto significa que o comportamento de tal sistema não muda caso sua escala seja multiplicada por um fator comum. O ponto preciso no qual o sistema apresenta tal divergência é chamado de ponto crítico.

¹⁰ O modelo de percolação descreve o comportamento de agrupamentos conectados em um grafo gerado por um processo aleatório. Mais detalhes sobre o modelo de percolação podem ser encontrados em (NEWMAN, 2005), na seção que discorre sobre transições de fase e fenômenos críticos (*Phase transitions and critical phenomena*).

forma, quadrados ocupados representam árvores no terreno e quadrados vazios representam espaços vazios de terra sem árvores.

Assim, tem-se um terreno similar a uma floresta, com árvores que nascem em locais aleatórios porém a uma taxa constante, ou seja, o intervalo de tempo entre o aparecimento de árvores é sempre o mesmo. Transpondo a ideia para o modelo, tem-se uma grade com quadrados sendo preenchidos aleatoriamente e a uma taxa constante.

De tempos em tempos, um incêndio tem início a partir de um quadrado na grade, ocasionado por exemplo, por um raio (descarga elétrica) que atinge a floresta. Caso atinja uma árvore, ou seja, um quadrado ocupado, irá queimá-la juntamente com cada uma das árvores diretamente à sua volta. Este evento é ilustrado na figura 6. O incêndio se espalha de forma que o fogo se alastra para árvores adjacentes até que todo o agrupamento de árvores seja queimado. No entanto, o fogo jamais pode cruzar a barreira formada por espaços vazios no terreno, ou seja, quadrados em branco. Caso não haja árvore no quadrado atingido pelo raio, ou seja, um quadrado vazio, nada acontece e o incêndio não tem início. Após um incêndio, árvores podem crescer novamente nos quadrados deixados vagos pelas árvores queimadas, de forma que o processo continua indefinidamente.

Figura 6 - No modelo de incêndios florestais, um raio atinge uma posição aleatória do terreno onde há uma árvore, iniciando desta forma um incêndio que queimará todo o agrupamento de árvores adjacentes. Figura extraída de (NEWMAN, 2005).

Começando com uma grade vazia, árvores começarão a crescer aleatoriamente, logo presume-se que ocuparão a grade de forma esparsa, com o mínimo de agrupamento entre árvores. Sendo assim, quando raios atingirem a floresta, terão grandes chances de acertarem áreas vazias. Ainda no início, caso estes raios atinjam um quadrado ocupado por uma árvore, queimarão não somente

ela como as árvores adjacentes. No entanto o agrupamento, caso haja um, será pequeno e localizado, pois o sistema ainda se apresenta em um estado muito abaixo do limiar de percolação e portanto pouco ocupado por árvores. Logo, no início, os incêndios não terão grandes efeitos destrutivos sobre a floresta como um todo.

À medida que o tempo passa mais e mais árvores crescerão ocupando quadrados da grade, até que em um determinado ponto tem-se a percolação, ou seja, atinge-se o ponto crítico. Neste ponto, tem-se a formação de ao menos um agrupamento de árvores de tal maneira que seu tamanho é limitado apenas pelo tamanho da grade, e quando qualquer árvore deste agrupamento for atingida por um raio, todo o agrupamento será queimado, ocasionando um incêndio de grande capacidade destrutiva para a floresta.

O agrupamento desaparece de forma que o sistema volta a um estado bem abaixo do ponto crítico, porém com o tempo, à medida que mais árvores aparecem, o sistema vai presumivelmente atingir a percolação novamente, e desta forma o cenário volta se repetir. O resultado final é que o sistema oscila ao redor do ponto crítico, inicialmente indo acima do limiar de percolação com o aparecimento de árvores e depois sendo levado a um ponto abaixo com os incêndios. No seu artigo, Newman (2005) menciona que no limite e para sistemas de grande tamanho, tais flutuações no número de árvores se tornam pequenas se comparadas ao tamanho do sistema como um todo e, através de uma aproximação, o sistema se manterá ao redor do limiar indefinidamente. Sendo assim, com o tempo espera-se que o modelo de incêndios florestais se auto-organize para um estado em que tenha uma distribuição do tipo Power Law dos tamanhos dos agrupamentos, ou neste contexto do tamanho dos incêndios.

A figura 7 mostra a distribuição cumulativa dos tamanhos dos incêndios no modelo de incêndios florestais e, como se pode ver, ela segue de perto uma *Power Law*, com o expoente da distribuição sendo bem pequeno neste caso. A melhor estimativa fornece um valor de $\alpha = 1,19 \pm 0,01$, significando que a distribuição tem uma média infinita no limite de um sistema de grande tamanho. Entretanto, para todos os sistemas reais a média é finita: a distribuição é cortada na cauda mais longa devido ao fato de os incêndios jamais poderem ser maiores que a área da grade como um todo.

Figura 7 - Exemplo de uma curva de distribuição cumulativa do tamanho de incêndios em uma simulação do modelo de incêndios florestais. Figura extraída de (NEWMAN, 2005).

Isso significa que nas florestas haverá uma grande quantidade de incêndios ocasionados por raios e que causarão pouca destruição, queimando árvores isoladas ou pequenos agrupamentos. Em contrapartida, haverá nestas florestas uma pequena quantidade de incêndios que causarão grande destruição, queimando boa parte da floresta por meio de grandes agrupamentos de árvores. Obviamente, florestas reais são muito mais complexas que as modeladas no modelo de incêndios florestais e os autores de (NEWMAN, 2005) e (DROSSEL; SCHWABL, 1992) não sugerem o modelo como representações fiéis do mundo real, no entanto este modelo fornece um guia de eventos e processos que podem de fato ocorrer em florestas.

Newman (2005) ainda cita que modelos de criticidade auto-organizada podem também ser utilizados para descrever terremotos, labaredas solares, evolução biológica, avalanches e muitos outros fenômenos. Apesar de não ser uma lei universal e não implicar diretamente na geração de distribuições do tipo *Power Law*, o modelo de criticidade auto-organizada é certamente um conceito intrigante e que tem diversas aplicações em sistemas naturais ou sistemas criados por seres humanos.

4 O ALGORITMO ADEA

O conceito de Entropia da Informação enunciado por Claude E. Shannon foi usado como inspiração para um algoritmo de substituição de linhas de *caches* para microprocessadores chamado ADEA – *Adapted Discrete-based Entropy Algorithm*, também chamado de Entropy (KOBAYASHI, 2010). Utilizando a mesma abordagem para descrever uma sequência de caracteres em uma linguagem arbitrária, seria possível calcular a entropia do *working set* de um programa modelando-o como uma sequência de endereços. Tratando cada endereço como uma variável aleatória, poder-se-ia estimar a entropia discreta de uma determinada variável como caracteres em uma linguagem. Em (KOBAYASHI, 2010) o autor propõe uma heurística de substituição que captura a localidade referencial de programas ao estimar as chances de reuso de linhas de *cache* baseadas na sua entropia discreta. O algoritmo ADEA armazena a recência e a frequência escolhendo quais linhas devem ser retiradas do *cache*. O maior benefício é explorar a localidade temporal de programas e atingir taxas de *miss* mais baixas se comparado a outros algoritmos de substituição tais como o LRU e os algoritmos LIP e BIP (QURESHI et al., 2007).

4.1 INFRAESTRUTURA DO ADEA

A fim de estimar a entropia discreta de um endereço que é referenciado por um programa, uma política de *cache* deve ser capaz de saber qual a probabilidade de tal referência ocorrer. Uma vez que não é possível saber *a priori* esta probabilidade, um novo mecanismo deve ser projetado para se estimar tal probabilidade e suportar a operação do ADEA.

Considere o *working set* de um programa como uma sequência de endereços $X = (x_1, x_2, \dots, x_n)$, na qual cada endereço x_i é uma variável aleatória que pode assumir qualquer valor no espaço de endereços do computador. Cada ocorrência de endereços é considerada independente de qualquer outra ocorrência no espaço de endereços. Sendo assim, a probabilidade da ocorrência de um determinado endereço não irá influenciar a probabilidade de qualquer outro endereço.

Levando em consideração o número de vezes que cada endereço é referenciado durante a execução de um programa, a política de substituição do

ADEA calcula a probabilidade de cada endereço que foi referenciado. Ao se iniciar o algoritmo, todos os possíveis endereços têm a mesma probabilidade de ocorrência e são definidos como 0 (zero). Isto significa que, antes de qualquer referência ao endereço, a incerteza daquele endereço é considerada máxima.

O ADEA requer uma probabilidade $p(x_i)$ para cada endereço a fim de considerar seus valores de entropia. Considerando-se um *cache* associativo sem perda de generalidade, define-se a probabilidade $p(x_i)$ como a taxa entre o número de vezes que o endereço foi referenciado e o número total de referências ao *cache set* que mapeia aquele endereço.

Uma situação indesejada pode ocorrer quando a primeira requisição a um endereço é feita por um programa em execução. Durante a fase de inicialização de um programa, a probabilidade do primeiro endereço é estimada em 100% já que é a primeira vez que é referenciado. Caso esta probabilidade fosse aplicada à equação de entropia discreta anteriormente mostrada, retornaria 0 (zero). Aquele valor seria um resultado inconsistente e tornaria o ADEA também inconsistente. Além disso, a mesma situação poderia ocorrer se, hipoteticamente, um programa requisitasse o mesmo endereço recorrentemente levando sua probabilidade a 100%. Para contornar esta situação indesejada, Kobayashi (2010) propõe o uso de um parâmetro de limiar que evita tais inconsistências como mostrado na equação a seguir:

$$\begin{cases} h(x_i) = u(x_i) * p(x_i), & \text{se } p(x_i) < \text{limiar} \\ h(x_i) = 1 - [u(x_i) * p(x_i)], & \text{se } p(x_i) > \text{limiar} \end{cases}$$

(VIII)

O ADEA calcula a incerteza $u(x_i)$ enquanto a entropia discreta calcula $h(x_i)$. Diferentemente do conceito de Entropia da Informação, a curva de entropia discreta declina após a probabilidade $p(x_i)$ atingir valores mais elevados que um limiar. No caso de probabilidades distribuídas entre possíveis valores de variáveis aleatórias, $h(x_i)$ declina após o limiar e acima. A figura 8 mostra a evolução do ADEA com e sem o ajuste da equação, no exemplo, o limiar é definido em 0,38.

Figura 8 – Gráficos de entropia do ADEA. Figura 8-a: apresenta o gráfico sem ajuste, com o valor de entropia diminuindo a partir de um limiar igual a 0,38. Figura 8-b: apresenta o gráfico de entropia com o ajuste assumindo um limiar igual a 0,38, fazendo com que a entropia volte a crescer após o limiar. Figuras extraídas de (KOBAYASHI, 2010).

A fim de se obter o melhor entendimento sobre o ADEA, algumas das peculiaridades dos algoritmos MRU (*Most Recently Used*) e do LRU (*Least Recently Used*) devem ser discutidas. Um *cache set* com a política de substituição baseada no algoritmo LRU é usualmente tratado como uma cadeia de duas extremidades, a extremidade MRU e a extremidade LRU. A extremidade MRU armazena as linhas que chegam enquanto a extremidade LRU armazena as linhas que serão removidas. No LRU, se uma linha específica é referenciada apenas uma vez, irá gradualmente se mover da extremidade MRU para a extremidade LRU até que seja retirada do *cache*. No entanto, se esta linha for referenciada enquanto estiver no *cache* ela será promovida para a extremidade MRU.

Inserção e substituição no ADEA ocorrem na mesma posição da cadeia do *cache set*. As linhas de *cache* com o menor valor de entropia entre linhas atualmente no *cache set* serão substituídas e serão sempre armazenadas na posição de inserção/substituição. Linhas com valores mais elevados de entropia serão armazenadas na outra extremidade da cadeia do *cache set*. Esta pode ser chamada região protegida do *cache set* onde substituições não ocorrerão. Uma linha de *cache* só é migrada ou promovida para aquela posição protegida se tiver seu valor de entropia elevado, o que vai acontecer se ela for mais vezes referenciada; na verdade, esta é a maior dissimilaridade entre os algoritmos ADEA e LRU. Enquanto o LRU imediatamente insere as novas linhas na posição mais protegida da cadeia de *cache set*, o ADEA irá aguardar por mais referências à mesma linha para promovê-la, baseando-se na probabilidade de futuras referências àquela linha de

cache. Em outras palavras, o ADEA apresenta uma inércia maior que o LRU ao promover linhas de *cache* para regiões protegidas, pois o valor de entropia aumenta apenas com novas referências.

Por outro lado, quando comparado ao LRU, o ADEA leva mais tempo para decidir remover uma linha, ou seja, necessita de mais acessos ao *cache* para ajustar os parâmetros que levarão tal linha à remoção. Poderão existir cenários de execução nos quais uma linha de *cache* intensivamente acessada seja movida a uma posição protegida do *cache set* e após um tempo não receba qualquer outra referência. As novas linhas apresentam uma entropia mais baixa que esta primeira linha por um período de tempo. O ADEA não move imediatamente a primeira linha mais referenciada à posição de substituição, o que poderia causar o indesejável efeito de confinamento (*pinning*) daquela linha indefinidamente no *cache set*, desperdiçando portanto espaço valioso. Para evitar tal situação, o ADEA possui duas funções decaimento de entropia que gradualmente diminuem o valor da entropia de linhas baseado no tempo que estão armazenadas no *cache set*. Estas funções decaimento diferenciam o algoritmo ADEA de outras abordagens que empregam a frequência de acessos como proposto por Dybdahl, Stenström e Natvig (2006). Isto significa que linhas com uma alta probabilidade não terão necessariamente uma prioridade mais alta sobre outras linhas com baixa probabilidade devido ao acesso frequente que ocorreu no passado distante. Novas referências devem ser levadas em consideração pelas funções decaimento.

De uma forma diferente, no algoritmo LIP (*LRU Insertion Policy*) proposto por Qureshi et al. (2007), novas linhas são armazenadas na extremidade LRU da cadeia e serão promovidas para a extremidade MRU apenas se tiverem uma nova referência antes de serem removidas. Ao fazer isso, o algoritmo tenta evitar o armazenamento de linhas de *cache* que serão referenciadas apenas uma vez, privilegiando aquelas que possuem uma maior frequência de ocorrência. Desta forma, inserção e remoção sempre ocorrerão na mesma posição da cadeia do *cache set*. Por outro lado, o algoritmo BIP (*Bimodal Insertion Policy*) se comporta de forma semelhante ao LIP, exceto pelo fato de que uma vez em um determinado intervalo de tempo vai inserir novas linhas na extremidade MRU. Este período de tempo entre os dois comportamentos é definido pelo parâmetro bimodal. Caso o parâmetro seja 1/32, por exemplo, significa que uma vez a cada 32 inserções de linhas de *cache* na extremidade LRU da cadeia de *cache set*, haverá um comportamento diferente e a

nova linha será inserida na extremidade MRU, e o ciclo de uma a cada 32 inserções será repetido.

Uma possível contrapartida do ADEA é possuir uma linha de *cache* indefinidamente confinada, resultando no desperdício de espaço. Esta situação de confinamento pode acontecer após um acesso intenso a uma determinada linha mesmo tendo havido um grande número de acessos a outras linhas de *cache*. Assim, existe a motivação de se possuir um mecanismo que diminua o valor da entropia discreta de um endereço intensivamente acessado, mas cujos acessos ocorreram no passado durante a execução, *i.e.* a fim de balancear frequência e recência, foram propostas duas diferentes funções decaimento para o ADEA:

Frequency-recency-based decay function – FR-decay: baseia-se na frequência e recência para remover linhas de *cache*;

Recency-rather-frequency-based function – RRF-decay: prioriza recência ao invés da frequência.

Nem todos os programas poderão se beneficiar das funções decaimento propostas. Uma função decaimento pode causar a remoção de linhas de *cache* ainda que estas linhas possam beneficiar tais programas. A análise de tal comportamento permite entender que para determinadas situações o confinamento de linhas de *cache* poderá melhorar o desempenho. Considerando o ADEA padrão (sem funções decaimento) como linha de base, esta configuração vai permitir que linhas de *cache* sejam substituídas com base apenas em seus valores absolutos de entropia discreta, o que significa que quanto mais alta a frequência de acessos maior a entropia discreta e o tempo no *cache*. Apenas para reforçar o conceito, o ADEA se baseia apenas na frequência para remover linhas de *cache*. Para melhor entender esta configuração, a equação II mostra como a entropia discreta é calculada.

Em geral, o ADEA designa um valor de entropia discreta para a linha de *cache* que foi mais intensivamente acessada. Neste caso, a linha seria colocada na porção mais protegida do *cache* e não seria movida de lá a não ser que uma nova linha mostrasse frequências mais elevadas de referências. Assim $p(x_i)$ é a probabilidade comum de x_i calculada pela taxa de frequência de referências de x_i e o número de referências ao *cache set* que mapeia tal bloco. A variável $u(x_i)$ é a incerteza como calculada pela equação I.

Analisando primeiramente a versão padrão do ADEA, com a substituição estritamente baseada na frequência, as linhas mais intensivamente utilizadas serão armazenadas na porção protegida do *cache* enquanto as linhas menos intensivamente acessadas irão competir pelas posições menos protegidas do *cache*. É natural comparar tal configuração do ADEA com o algoritmo MRU. Caso o programa referencie uma sequência de linhas distintas, por exemplo, A, B, C, D, E, no mesmo *cache set*, cada uma delas terá a mesma frequência de referências. O ADEA apenas leva em consideração o número de referências e não o momento em que a linha foi referenciada no tempo. Nesta situação particular, cada nova linha referenciada terá o mesmo valor de entropia discreta e será mantida na mesma posição de substituição. A justificativa para isso é que o valor de entropia não é maior que a entropia de outras linhas de *cache*. Portanto, quando novas linhas são referenciadas, a linha mais recentemente armazenada será removida para abrir espaço no *cache*. Desta forma, o ADEA irá se comportar de forma semelhante ao MRU.

Ter linhas de *cache* confinadas no *cache* pode beneficiar alguns programas, mas na maioria dos casos irá tornar o *cache* menos eficiente. Uma linha armazenada no *cache* por um longo período de tempo certamente tem grande probabilidade de não mais ser referenciada. De fato, Qureshi et al. (2007) afirma que 60% das linhas armazenadas em *caches* LRU são removidas sem nunca terem uma segunda referência enquanto armazenadas, e esta afirmação representa o algoritmo LIP com perfeição, uma vez que seu princípio é de que uma linha deva ser inserida na posição LRU já que provavelmente não será referenciada novamente, levando à substituição mais rapidamente. Para o algoritmo ADEA, a fim de evitar o desperdício de espaço no *cache* é razoável empregar uma função decaimento que minimize as chances de ter linhas de *cache* confinadas.

Quando comparado a abordagens baseadas em LRU tais como LIP e BIP, o ADEA não apenas estima o reuso baseado em frequência, mas também na recência, *i.e.* baseando-se no quão distante ela ocorreu no tempo de execução, caracterizado pelas funções decaimento do ADEA.

4.2 FREQUENCY-RECENCY-BASED DECAY FUNCTION – FR-DECAY

Frequency-recency-based decay function – FR-decay – emprega uma combinação de frequência e recência de acessos para calcular o valor discreto de entropia. Foi introduzido o parâmetro t_s – *time stride* – que acompanha quanto tempo no passado da execução houve referência à linha de *cache* relativa ao último acesso ao *cache set*. Para as linhas de *cache* mapeadas no *cache set*, define-se $t_s = 0$ quando o tempo de acesso à linha de *cache* coincide com o acesso ao *cache set*. À medida que a execução do programa continua, o t_s para as linhas já armazenadas tende a aumentar. Usando este conceito de recência pode-se fazer decair o valor discreto de entropia das linhas armazenadas. Relembrando o exemplo mencionado anteriormente, uma linha de *cache* intensivamente acessada e que é armazenada na porção mais protegida do *cache* terá seu valor discreto de entropia diminuindo como uma inversão do logaritmo do tempo tal como mostrado na equação IX. Foi escolhido um decaimento que segue uma função de logaritmo ao invés de uma função linear, do contrário este decaimento aconteceria mais rápido do que apropriado. Esta escolha é discutida em mais detalhes no trabalho (KOBAYASHI, 2010).

$$\begin{aligned}
 t_s &= LastCacheSetAccess - LastBlockAccess(x_i); \\
 decay &= 1/(\log_{10}(t_s + 1) + 1); \\
 h(x_i) &= \begin{cases} u(x_i) * p(x_i) * decay & \text{se } p(x_i) < limiar \\ 1 - (u(x_i) * p(x_i) * decay) & \text{se } p(x_i) \geq limiar \end{cases}
 \end{aligned}
 \tag{IX}$$

Ao empregar a função FR-decay, o ADEA se comporta de forma diferente do MRU em uma situação na qual uma sequência de blocos de linhas distintas é referenciada. Já que os blocos possuem a mesma frequência, o componente recência é empregado; depois linhas mais antigas na sequência têm suas entropias diminuídas, enquanto novas linhas, ainda que com a mesma frequência e probabilidade, têm seus valores de entropia elevados. Desta forma, aquelas novas linhas serão armazenadas na porção mais protegida do *cache*, que é exatamente o oposto da versão linha de base.

4.3 RECENCY-RATHER-FREQUENCY-BASED DECAY FUNCTION – RRF-DECAY

Recency-rather-frequency-based decay function – RRF-decay – criada para cobrir situações nas quais uma linha muito acessada é removida do *cache* devido à função decaimento e depois volta a ser referenciada pelo programa. Caso os contadores de frequência não sejam zerados no momento em que a linha for removida, um *offset* extra será adicionado, levando a linha diretamente para a porção protegida do *cache* e carregando o histórico de acessos juntamente com a nova referência. Isto pode levar a efeitos indesejados tais como competição por espaço no *cache* e remoção de outras linhas com menos, porém mais recentes acessos.

Para evitar tal situação, o RRF-decay verifica se o último acesso que substituiu uma linha foi feito em um tempo muito distante na execução. Caso sim, seus valores de entropia e contadores são zerados. Ao empregar esta nova heurística nas substituições, evita-se que uma linha acessada muito intensivamente retorne como privilegiada e consiga uma posição protegida no *cache*. A equação X mostra a heurística adicional:

$$\begin{aligned}
 t_s &= LastCacheSetAccess - LastBlockAccess(x_i); \\
 decay &= 1/(\log_{10}(t_s + 1) + 1); \\
 h(x_i) &= \begin{cases} u(x_i) * p(x_i) * decay & \text{se } p(x_i) < limiar \\ 1 - (u(x_i) * p(x_i) * decay) & \text{se } p(x_i) \geq limiar \end{cases} \\
 \text{Na substituição: } p(x_i) &= 0; \text{ se } t_s > 1000000
 \end{aligned}$$

(X)

A estratégia do RRF-decay vai zerar os contadores de linhas mais antigas que foram removidas. Caso as linhas removidas sejam referenciadas novamente, serão carregadas no *cache* como se fosse a primeira vez e não serão levadas à porção mais protegida do *cache*. Presumivelmente, deverão ocorrer situações nas quais programas apresentem longos padrões de localidade temporal e tal função decaimento vai impactar negativamente o desempenho geral. Em tais longos padrões temporais, um bloco de *cache* pode ser referenciado novamente após ciclos de instruções muito longos. Para tais casos, tal função decaimento vai certamente diminuir o desempenho e aumentar as taxas de *misses* ao diminuir as chances de manter uma linha reutilizável no *cache* por um longo período de tempo.

5 PROPOSTA DE CRIAÇÃO DO CACHE PARETO

Este trabalho propõe a criação de um *cache* Pareto baseado nos princípios das *Power Laws*, de pequena capacidade de armazenamento e baixa latência.

Segundo as *Power Laws*, dada uma aplicação a ser executada pelo processador, uma pequena parcela dos seus endereços sofrerá uma grande quantidade de acessos ao longo da sua execução. Essa diferença no número de acessos tende a ser muito discrepante, podendo indicar que apenas 20% do total de endereços representa 80% do total de acessos (Princípio de Pareto). Partindo desta ideia, pode-se assumir que uma pequena porcentagem da aplicação sofre uma quantidade de acessos muito maior que o restante, logo deve ser mantida em uma espécie de *buffer* de baixa latência.

O *cache* Pareto é inserido entre dois níveis da hierarquia de memória, por exemplo, entre L1 e L2. No *cache* Pareto estariam os conteúdos dos endereços mais acessados pelo processador no *cache* L2. Dessa forma, quando há um *miss* no L1, o conteúdo do endereço é buscado no *cache* Pareto e, se não fosse encontrado, é buscado no L2. Se fosse encontrado no *cache* Pareto, seria copiado para o L1 e utilizado pelo processador. Tratando-se de um endereço muito acessado, é interessante tê-lo dentro do L1 para acessos futuros, evitando uma nova consulta ao *cache* Pareto.

No entanto, o *cache* Pareto não tem o mesmo comportamento de um *cache* intermediário (entre L1 e L2). No exemplo anterior, se o conteúdo do endereço requisitado não é encontrado no *cache* Pareto e é encontrado no L2, este conteúdo é então copiado do L2 para o L1 e não para o *cache* Pareto, pois a política de preenchimento do *cache* Pareto não é a mesma dos *caches*, mas baseado nas *Power Laws*. Desta forma (ainda seguindo o mesmo exemplo), na próxima janela de tempo em que houvesse uma avaliação dos endereços mais acessados no L2, estes seriam copiados para o *cache* Pareto independente do conteúdo do *cache* L1.

O *cache* Pareto se diferencia de um *victim cache* (JOUPII, 1990) essencialmente pela forma como se dá seu preenchimento. Enquanto o *victim cache* é preenchido com endereços removidos do *cache*, o *cache* Pareto armazenará os endereços mais acessados, o que tende a mudar ao longo da execução da aplicação, mas não à medida que endereços são removidos do *cache* como se daria com o *victim cache*.

Figura 9 - Exemplo de organização do processador incluindo a inserção de um *cache* Pareto entre os *caches* L1D e L2. Antes de consultar o L2 haverá consulta ao *cache* Pareto, de tamanho reduzido e baixa latência de acesso.

Este modelo é avaliado com mais detalhes no capítulo 8, no qual são apresentados resultados comparativos de desempenho de implementações do *cache* Pareto.

6 FOREST FIRE SWITCHING MECHANISM - FFSM

Este trabalho propõe a criação de um mecanismo de chaveamento que permite ajustar o comportamento do algoritmo de substituição de linhas de *cache* de modo que este possa, de forma mais eficiente, realizar substituições de endereços mais velhos por outros mais novos.

6.1 O CACHE MODELADO COMO UMA FLORESTA

Como mencionado anteriormente, o modelo de incêndios florestais apresenta um terreno modelado como uma grade, um quadrado subdividido em quadrados menores e iguais em tamanho.

Imaginando um *cache* como uma grade, cada pequeno quadrado representa uma linha de *cache* contendo blocos de endereços. Desta forma, quadrados vazios representam espaços vazios no *cache*, enquanto quadrados ocupados por árvores representariam linhas de *cache* que são trazidas pelo processador de outro nível da hierarquia de memória.

O surgimento de árvores na floresta modelada seria semelhante ao acesso a endereços de memória realizado pelo processador, que traz blocos de endereços e os armazena no *cache*. No início, tem-se um *cache* vazio que, com o passar do tempo de execução de uma determinada aplicação, é preenchido pelas instruções e dados acessados pelo processador.

Com o passar do tempo é comum que haja a necessidade de substituir uma linha de *cache* por outra mais nova, uma vez que o *cache* possui tamanho finito e muitas ordens de grandeza menor que a memória principal. Tal substituição remove do *cache* a linha antiga dando lugar à nova linha trazida, por exemplo, da memória principal. Esta remoção da linha de *cache* a ser substituída pode ser comparada à destruição provocada por um raio no modelo de incêndios florestais. Desta forma, raios podem ser comparados a substituições ocasionadas pela política de substituição de linhas do *cache*, em que a substituição de uma linha de *cache* por

outra seria equivalente à destruição de uma árvore com o surgimento de outra na mesma posição.

Figura 10 - Funcionamento simplificado de um *cache* modelado como uma floresta. Na sequência, o *cache* possui posições ocupadas em verde e espaços vazios em marrom. O acesso a um novo endereço faz com que o *cache* o armazene em uma posição vazia. Após três acessos, todas as posições são ocupadas. No quarto acesso (bloco azul) não há espaço no *cache*, logo a política de substituição escolhe um bloco para ser retirado do *cache* dando espaço para o armazenamento do novo endereço.

6.2 CHAVEAMENTOS DE CONTEXTO DO CACHE

Assumindo que um programa em execução possa ser representado por uma sequência de contextos do *cache*, instruções tais como *calls* e *returns* podem representar mudanças entre diferentes contextos. Para cada *call*, há a possível mudança do estado do *cache*, uma vez que padrões de acesso à memória tendem a mudar, o que é complementado pela instrução *return* correspondente, que fará com que haja outra possível mudança no padrão de acessos. Sendo assim, o algoritmo

de substituição do *cache* deve se adaptar a determinados contextos a fim de melhorar a localidade, reduzindo o número de *misses* sempre que um chaveamento ocorrer.

A associação entre raios (descarga elétrica de nuvens) e *calls* e *returns* de programas representa como a ideia de Incêndios Florestais modelados como *Power Laws* pode ajudar um mecanismo adaptativo de substituição do *cache* a melhorar seu desempenho. Cada região de um programa possui uma recência de dados adequada. Com base no modelo de incêndios florestais é possível construir um mecanismo que pode ajustar o algoritmo de substituição para melhor se adequar a uma região específica do programa, melhorando a recência de dados.

O mecanismo chamado *Forest Fire Switching Mechanism* – FFSM - é baseado no Modelo de Incêndios Florestais, que como anteriormente mencionado pode ser descrito por uma *Power Law*. Neste trabalho, grandes incêndios florestais significam grandes chaveamentos de contexto de programas, como *calls* e *returns*, que serão refletidos na forma como a memória *cache* se comporta. O FFSM vai sinalizar quando grandes chaveamentos de contexto de dados ocorrerão, ajudando o algoritmo de substituição a realizar melhores substituições de linhas de *cache*.

A melhoria de desempenho será alcançada ao se entender o que causa os raios que irão destruir várias árvores, ou em outras palavras, os chaveamentos de contexto que farão com que várias linhas sejam removidas a fim de dar espaço a novas linhas. O modelo explora estas mudanças, ajudando o algoritmo de substituição a selecionar quais árvores, ou neste caso quais linhas devem dar lugar a outras novas, baixando a taxa de *misses* e melhorando o desempenho geral.

De acordo com o Modelo de Incêndios Florestais descrito por uma *Power Law*, espera-se que uma grande quantidade de incêndios cause pouca destruição e que grandes incêndios sejam menos frequentes. Transpondo esta ideia para o *cache* modelado de acordo com os princípios do Modelo de Incêndios Florestais, é de se esperar que haja uma grande quantidade de substituições de linhas de *cache* com pouco impacto no estado do *cache* como um todo, porém a cada chaveamento de contexto provocado por um *call* ou *return*, uma mudança significativa no padrão de acessos ocorrerá, dando origem a um ocasional incêndio mais destrutivo, ou seja, que causará a substituição de uma quantidade maior de linhas de *cache*, alterando seu estado como um todo.

Figura 11 - Diferentes padrões de acesso à memória. Operações sobre diferentes *working sets* representam mudanças de estado do *cache* e na forma como este é preenchido. A figura 11-a mostra o acesso a um *array* disposto contiguamente na memória principal. A figura 11-b apresenta acessos a variáveis dispostas aleatoriamente na memória. A figura 11-c mostra o acesso a blocos dispostos aleatoriamente, tais como em uma lista ligada.

6.3 APLICAÇÃO DO FFSM

O conceito do FFSM pode ser aplicado de forma generalizada a algoritmos adaptativos de substituição de linhas de *cache*, como uma forma de sinalização de uma mudança significativa no padrão de acessos à memória, que demanda do algoritmo uma adaptação do seu comportamento. Neste trabalho, o algoritmo ADEA (KOBAYASHI, 2010) foi escolhido devido a suas diferentes funções decaimento, capazes de fazer com que o ADEA se comporte de maneiras diferentes. Embora não tenha sido concebido para ser um algoritmo adaptativo, suas diferentes funções poderiam ser combinadas na forma de um único algoritmo, capaz de chavear seu comportamento em determinadas situações para se adequar ao padrão de acessos do programa em execução.

Neste trabalho, a implementação proposta para o FFSM basicamente chaveia entre o RRF-decay e o FR-decay do ADEA, usando instruções de *call* e *return* para fazê-lo. Desta forma, o algoritmo de substituição tem seu comportamento chaveado para melhor se adaptar a tais mudanças. O *return* do *call* correspondente leva a uma mudança similar, uma vez que o estado tende a mudar novamente devido ao padrão diferente de acessos à memória. Este trabalho propõe portanto a mudança do RRF-decay para o FR-decay e vice-versa como os dois principais mecanismos de incêndio florestal.

Foram feitas duas implementações para lidar com as funções decaimento do ADEA:

RRF-FR: para esta implementação, toda vez que houver uma instrução de *call* haverá mudança da função decaimento RRF-decay para a FR-decay, voltando para RRF-decay quando uma instrução de *return* for executada;

FR-RRF: comporta-se de forma exatamente oposta à primeira, mudando de FR-decay para RRF-decay com um *call* e novamente para FR-decay com uma instrução de *return*.

Figura 12 - Funcionamento do mecanismo de chaveamento da política de substituição de linhas de *cache*. No início a política vigente é a RRF-decay do ADEA. Quando uma instrução de *call* é executada, há o chaveamento da RRF-decay para a FR-decay. A política vigente se mantém até que seja executada uma instrução de *return*, quando então haverá outro chaveamento, alterando a política vigente para a RRF-decay.

Resultados preliminares deste modelo foram publicados em (SCOTON; MARINO; KOBAYASHI, 2010). Resultados adicionais são discutidos no capítulo 8.

7 METODOLOGIA DE ESTUDO

Por meio da leitura de publicações, estudou-se o estado da arte das tecnologias atuais de *caches*, bem como problemas de desempenho e possíveis abordagens.

Estudos realizados sobre modelagens analíticas e estatísticas de *caches* também serviram de base para a consolidação do conhecimento e a escrita deste texto.

A análise mais concentrada em *Power Laws* visa entender os conceitos e pesquisar métodos úteis para determinar se uma massa de dados empíricos segue ou não uma função deste tipo. O artigo de Clauset, Shalizi e Newman (2007) propõe uma metodologia com tal finalidade.

A discussão acerca do ADEA tem por objetivo o entendimento dos conceitos básicos deste algoritmo, utilizado como referência de aplicação do FFSM.

Através do uso do simulador *SimpleScalar* (2009) e seu artigo introdutório (AUSTIN; LARSON; ERNST, 2002), foram gerados *traces* de execução para um conjunto de aplicativos que compõem o SPEC CPU2000 (2009). Com isso pretendia-se obter a sequência de acessos aos endereços nos *caches*.

A partir de *scripts* que têm como entrada os *traces* anteriormente mencionados, foram gerados histogramas que contêm a lista de endereços acessados e a frequência de acessos, ou seja, quantas vezes houve acesso a cada endereço durante a execução da aplicação. Quando os dados contidos nesses histogramas são representados em escala logarítmica, pode-se avaliar o comportamento dos acessos aos *caches* e se eles seguem ou não uma *Power Law*. A escala logarítmica tem a capacidade de linearizar funções exponenciais e se aplicada a uma *Power Law*, dará origem a uma reta decrescente.

Uma implementação simplificada de um *cache* Pareto foi feita e avaliações de desempenho realizadas e comparadas com um modelo padrão, baseado em configurações típicas, com o objetivo de se medir potenciais ganhos de desempenho proporcionados pelo *cache* Pareto.

Ainda utilizando-se o *SimpleScalar*, neste trabalho foram geradas análises de comparação de desempenho entre algoritmos de substituição de linhas de *cache*, com o objetivo de avaliar os ganhos proporcionados pelo mecanismo de chaveamento de comportamento de tais algoritmos. Esta análise propiciou um

estudo mais aprofundado do próprio ADEA e de suas funções decaimento, complementando o trabalho iniciado por Kobayashi (2010) que pela primeira vez apresentou tal algoritmo.

7.1 AMBIENTE DE SIMULAÇÃO

Para o ambiente de simulação foi utilizado o simulador *SimpleScalar* (2009). O conjunto de ferramentas *SimpleScalar* se baseia em uma infraestrutura de *software* utilizada para construir modelos para análise de desempenho de programas e modelagem detalhada de microarquiteturas (SIMPLESCALAR LLC, 2009). Através do *SimpleScalar*, pode-se construir modelos de *caches* de microprocessadores que simulam a execução de uma arquitetura real. O conjunto de ferramentas inclui uma variedade de simuladores, indo desde rápidos simuladores funcionais até processadores com agendamento dinâmico, capazes de suportar *caches* não-bloqueantes, execução especulativa e predição de desvios (SIMPLESCALAR LLC, 2009).

Esse simulador foi escolhido entre outros devido à sua facilidade e agilidade na criação de modelos de *caches*. O *SimpleScalar* tem sido muito utilizado pela comunidade científica (FRIENDS OF SIMPLESCALAR LLC, 2009), mostrando-se uma ferramenta muito útil e confiável.

7.2 PARÂMETROS DE SIMULAÇÃO

Para as simulações executadas com o objetivo de gerar *traces* de execução, foi considerado um processador contendo um único *core* com configurações típicas de *caches*, baseadas em processadores reais e no trabalho de Kirman et al. (2006): 32 kB e associatividade 4 para o *cache* L1 de dados e instruções; 2 MB e associatividade 8 para o *cache* L2; mecanismo de substituição baseado no algoritmo LRU. Os detalhes dos parâmetros utilizados são apresentados na Tabela 1, coluna “Parâmetro Pareto”.

Para as simulações executadas com o objetivo de análise de desempenho do *Forest Fire Switching Mechanism* e de algoritmos de substituição de linhas de *cache*, foram consideradas configurações mais variadas, especialmente no que diz respeito

ao tamanho do *cache* L2 e sua associatividade. Neste caso, foi considerado um processador contendo um único *core* com configurações típicas de *caches*: 16 kB e associatividade 4 para o *cache* L1 de dados e instruções e mecanismo de substituição baseado no algoritmo LRU para o L1; para o *cache* L2, foram utilizados tamanhos de 512 kB, 1 MB e 2 MB, com associatividade 8 e 16; para os algoritmos de substituição foram utilizados o LRU, LIP, BIP, ADEA e suas funções decaimento, além das novas variações contendo o mecanismo de chaveamento proposto. Os detalhes dos parâmetros utilizados são apresentados na Tabela 1, coluna “Parâmetro FFSM”.

Tabela 1 - Parâmetros de simulação. A coluna “Parâmetro Pareto” se refere aos parâmetros utilizados na geração dos *traces* e na análise de desempenho do *cache* Pareto. A coluna “Parâmetro FFSM” se refere aos parâmetros utilizados na análise de desempenho da implementação do *Forest Fire Switching Mechanism* e dos algoritmos de substituição de linhas de *cache*.

Componente	Parâmetro Pareto	Parâmetro FFSM
Processador	Out-of-order	Out-of-order
L1D e L1I – tamanho	32 kB +32 kB	16 kB +16 kB
L1 – associatividade	4	4
L1 – latência	2 ciclos	2 ciclos
L2 – tamanho	2 MB	512 kB, 1 MB e 2 MB
L2 – associatividade	8	8 e 16
L2 – latência	8 ciclos	8 ciclos

7.3 BENCHMARKS

O SPEC (*The Standard Performance Evaluation Corporation*) (2009) é uma organização sem fins lucrativos formada para estabelecer e manter um conjunto padronizado de *benchmarks* que possam ser aplicados à nova geração de computadores de alto desempenho. A suíte SPEC é um dos conjuntos de *benchmarks* mais utilizados pela comunidade científica, sendo amplamente reconhecido tanto no meio acadêmico quanto no mercado (SPEC, 2009).

Neste trabalho foram executadas simulações com as aplicações AMMP, APPLU, APSI, ART, BZIP, CRAFTY, EON, EQUAKE, FMA3D, GAP, GCC, GZIP, LUCAS, MESA, MGRID, PARSER, SWIM, TWOLF e VPR, todas componentes do conjunto de *benchmarks* SPEC-CPU2000 (2009). O conjunto de *benchmarks* é comercializado e por isso nem todas as aplicações que compõem o conjunto foram utilizadas, uma vez que para este trabalho possuía-se acesso apenas a uma parte das aplicações, listadas na Tabela 2.

A tabela a seguir enumera e descreve de forma simplificada cada um dos *benchmarks* do SPEC-CPU2000 que foram utilizados.

Tabela 2 - Tabela de *benchmarks* do conjunto SPEC-CPU2000. Traduzida de (SPEC, 2009).

Benchmark	Tipo	Área de aplicação / Descrição
AMMP	FP	Química computacional
APPLU	FP	Equações parciais diferenciais parabólicas / elípticas
APSI	FP	Meteorologia: distribuição de poluentes
ART	FP	Reconhecimento de imagens / Redes neurais
BZIP	INT	Compressão
CRAFTY	INT	Jogo de xadrez
EON	INT	Visualização computacional
EQUAKE	FP	Simulação de propagação de ondas sísmicas
FMA3D	FP	Simulação de colisão de elementos finitos
GAP	INT	Teoria de grupos, interpretador
GCC	INT	Compilador de linguagem C
GZIP	INT	Compressão
LUCAS	FP	Teoria dos números, teste de primalidade
MESA	FP	Biblioteca de gráficos 3D
MGRID	FP	Solucionador multi-grid: Campo potencial 3D
PARSER	INT	Processamento de palavras
SWIM	FP	Modelagem de água rasa
TWOLF	INT	Simulador de localização e rota
VPR	INT	Posicionamento e roteamento de circuitos FPGA

7.4 TRACES

Simulações realizadas utilizando *traces* (*trace-driven simulations*) estão entre as técnicas mais comuns para a obtenção do perfil do desempenho de *caches* (AGARWAL; HOROWITZ; HENNESSY, 1986).

Smith em seu trabalho (1982) discute inúmeros aspectos relativos a memórias *cache* e detalhes de projeto destes, tais como algoritmos de busca, *prefetching*, tamanho e número dos *sets*, tamanho das linhas, políticas de substituição, consistência, entre outros. Nesse mesmo artigo, cada parâmetro é variado e as simulações são refeitas para se obter o impacto de cada variação no desempenho do *cache*.

Os *caches* são estudados por Goodman (1983) sob a perspectiva de reduzir o tráfego de informações entre processador e memória. Os autores de (NAZ et al., 2004) focam no *cache* de dados ao avaliar o impacto de *split*¹¹ e *victim caches* no primeiro nível da hierarquia de memória, valendo-se de simulações direcionadas por *traces*.

Chen e Baer utilizam em (1992) a geração de *traces* combinada com simulação *on-the-fly* para o estudo da redução de latência através de *caches* não-bloqueantes e de *prefetching*. O objetivo dessa combinação é reduzir o *overhead* da geração dos *traces* para cada configuração. No trabalho de Strauss, Shen e Torrellas (2006), os autores estudam mecanismos de coerência de *caches* e fazem uso de mais de uma técnica de simulação (*trace-driven* e *execution-driven*), porém de forma separada, utilizando simuladores e conjuntos de *benchmarks* distintos para cada uma.

Estudos acerca do comportamento dos acessos ao *cache* L2 são realizados por Kotera et al. (2008), no qual são utilizados *traces* para se obter informações sobre a localidade dos acessos por meio da técnica de *stack distance profiling*¹² (CHANDRA et al., 2005), a fim de obter um modelo de previsão de mudanças nesta distribuição e que seja baseado na Lei de Zipf (1932, 1949).

¹¹ O *split cache* provê uma arquitetura de suporte para distinguir referências de memória que exibem localidade espacial e temporal, além de mapeá-las a *caches* separados a fim de implementar configurações explorando diferentes parâmetros de *cache* de forma seletiva e eficiente (NAZ et al., 2004).

¹² A técnica de *stack distance profiling* utiliza contadores de distância para contar o número de acessos a linhas na pilha LRU. Linhas mais próximas à extremidade MRU, ou seja, mais recentemente acessadas, possuem distâncias menores (*smaller stack distances*). A partir dos valores destes contadores gera-se um histograma capaz de ajudar a distinguir entre diferentes comportamentos de acessos ao *cache*.

7.5 GERAÇÃO DE TRACES

Cada simulação envolveu a execução de uma amostra de 100 milhões de instruções (amostragem – *sampling*), sendo que a geração de *traces* começa a partir da instrução de número 2 milhões. Esse número de instruções foi escolhido devido ao tempo de simulação reduzido (da ordem de minutos), o que viabiliza intervalos de tempo menores entre cada teste. O avanço de 2 milhões de instruções (*warm-up*) tem como objetivo eliminar os primeiros *misses* relacionados ao início da execução da aplicação (*cold misses*), que podem distorcer as informações sobre a execução da aplicação como um todo. As técnicas de *warm-up* e de *sampling* são práticas comuns, podendo ser encontrados exemplos das mesmas no trabalho de Zhang e Asanovic (2005).

Através de modificações simples no simulador, foi possível obter os *traces* de execução de cada aplicação testada. Desta forma, a cada acesso ao *cache*, imprimia-se o endereço acessado, de forma a obter-se a listagem de todos os acessos realizados durante a simulação.

7.6 GERAÇÃO DE HISTOGRAMAS E CURVAS DE COMPORTAMENTO

Com os arquivos de saída contendo os *traces* de execução das aplicações, foram gerados histogramas a partir de pequenos *scripts* criados utilizando-se a linguagem PERL. Esses histogramas continham uma listagem de todos os endereços acessados e o número de vezes em que apareciam na execução de cada aplicação (frequência de ocorrência), sendo ordenados do mais para o menos frequente.

A partir desses histogramas, foi utilizado o *software* Gnuplot (2009) para traçar curvas em escala logarítmica - $\log(\text{frequência})$ versus $\log(\text{ranking})^{13}$ – permitindo uma primeira avaliação visual do comportamento dos acessos aos *caches* na execução de cada aplicação.

Com base nos histogramas, foi utilizado o conjunto de ferramentas R (2009) para realizar a regressão linear dos dados, transformando as curvas de comportamento em retas médias, o que permite a comparação com *Power Laws*.

¹³ O *ranking* significa a posição na lista de endereços e suas frequências de ocorrência.

7.7 ANÁLISE COMPARATIVA DE ALGORITMOS DE SUBSTITUIÇÃO DE LINHAS DE CACHE

Foram realizadas simulações das primeiras 20 bilhões de instruções de cada *benchmark* para as diferentes configurações de *caches* e os diferentes algoritmos de substituição de linhas de *cache* utilizados no trabalho. O número de instruções foi escolhido para que se pudesse obter amostras mais significativas, compreendendo diferentes trechos de execução dos *benchmarks*.

Utilizando-se como base a implementação do LRU para o *SimpleScalar* e a infraestrutura do próprio simulador, neste trabalho foram reimplementados os algoritmos LIP e BIP no *SimpleScalar* tais como propostos no estudo de Qureshi et al. (2007). Com base na implementação do ADEA, também para o *SimpleScalar*, foi implementado o FFSM com o objetivo de chavear entre as funções decaimento do algoritmo.

Como resultado das simulações, foram utilizadas as estatísticas de taxa de *misses* (*miss rate*) no *cache* L2, bem como instruções por ciclo ou IPC (*instructions per cycle*). Estas estatísticas são geradas automaticamente pelo *SimpleScalar* ao final de cada simulação e são bons indicadores quanto ao que se espera de ganho de desempenho proporcionado por uma política de substituição do *cache*, ou seja, diminuição do número total de *misses*, portanto uma taxa mais baixa, e aumento de número de instruções executadas por ciclo, portanto mais eficiência na execução de aplicações.

8 RESULTADOS E ANÁLISES

O capítulo descreve a forma como foram analisados os dados empíricos, a geração de curvas de comportamento de acessos aos *caches* e as regressões lineares às quais foram submetidos os dados de cada uma dessas curvas. Adicionalmente, apresenta análise comparativa de desempenho entre os algoritmos de substituição de linhas de *cache* e a implementação do FFSM para o ADEA e para alguns dos algoritmos de Qureshi, além de um comparativo, também de desempenho, visando observar os potenciais ganhos da inserção de um *cache* Pareto.

8.1 ANÁLISE DOS TRACES

Para cada simulação foram gerados dados empíricos a partir dos *traces* extraídos durante a execução de cada uma das aplicações escolhidas. Esses *traces* apresentam todos os endereços acessados em cada um dos *caches*. A partir da execução de um programa que tem como entrada os *traces*, tem-se como saída um histograma, contendo cada um dos endereços acessados e o número de vezes em que aparecem no *trace* gerado. Essa lista de endereços e frequências de ocorrência aparece de forma ordenada, do endereço mais frequente para o menos frequente.

Os histogramas criados são os dados já tratados mas que não serão de fato aqueles utilizados para as modelagens e curvas de comportamento. Uma nova tabela contendo os *rankings* de ordenação - posição na tabela - no lugar dos endereços é criada para a geração da curva frequência versus *ranking* em escala logarítmica (utilizando-se o Gnuplot). As curvas aproximadas a retas, devido ao uso da escala logarítmica, permitem a comparação visual com uma *Power Law*, que neste caso se apresentaria na forma de uma reta decrescente.

Também com essa tabela é possível utilizar o aplicativo R para determinar os coeficientes resultantes da regressão linear e com isso traçar a reta média do comportamento de acessos aos *caches* de cada aplicação.

8.2 CURVAS DE COMPORTAMENTO DE ACESSO

A partir das curvas de comportamento de acesso traçadas, observa-se uma variedade de comportamentos de cada aplicação utilizada.

Foram realizadas avaliações para *traces* de acessos aos *cache* L1 de dados, L1 de instruções e L2. As seções a seguir detalham as curvas referentes aos acessos a cada um desses *caches*. Cada seção apresenta apenas 4 dos 12 *benchmarks* utilizados, as demais curvas podem ser encontradas no apêndice D.

8.2.1 CACHE L1 DE DADOS

Para o *cache* L1 de dados, as aplicações AMMP, CRAFTY, EON, GCC, TWOLF e VPR apresentam comportamento mais próximo de uma *Power Law*, assemelhando-se mais a uma reta decrescente. As demais aplicações – APPLU, ART, EQUAKE, FMA3D, MESA e MGRID – possuem patamares como característica marcante, o que significa que uma grande quantidade de endereços possui a mesma frequência de ocorrência, ou seja, são acessados a mesma quantidade de vezes ao longo da execução de cada aplicação. Portanto, pode-se dizer para ambos os casos que existe grande localidade para uma pequena quantidade de endereços de cada uma das aplicações, ou seja, existem poucos endereços que possuem grande probabilidade de serem acessados de tempos em tempos – grande frequência de ocorrência – e uma grande quantidade de endereços que ocorre poucas vezes.

As figuras a seguir mostram o comportamento descrito para acessos ao *cache* L1 de dados:

Figura 13 – Curvas de comportamento de acesso ao *cache* L1 de dados para os *benchmarks* CRAFTY e GCC.

Figura 14 - Curvas de comportamento de acesso ao *cache* L1 de dados para os *benchmarks* APPLU e MESA.

As figuras relativas aos demais *benchmarks* podem ser encontradas no final do texto, apêndice D.a.

8.2.2 CACHE L1 DE INSTRUÇÕES

Para o *cache* L1 de instruções, as aplicações apresentam comportamento mais distante de uma reta. Com exceção de CRAFTY e GCC, todas as demais aplicações apresentam patamares nos valores mais à esquerda do gráfico (onde as frequências de ocorrência são mais elevadas). Além disso, as curvas de AMMP, APPLU, ART, EON, EQUAKE, FMA3D, MESA, MGRID, TWOLF e VPR possuem outra característica que é a queda abrupta da curva, com baixa concentração de pontos. Isso significa que existem poucos endereços com frequências de ocorrência muito baixas - de algumas unidades. Portanto, pode-se dizer que existe grande localidade para cada uma das aplicações, ou seja, existem muitos endereços que possuem

grande probabilidade de serem acessados de tempos em tempos – grande frequência de ocorrência. Esse comportamento é diferente do apresentado no *cache* L1 de dados, em que poucos endereços apresentavam localidade.

As figuras a seguir mostram o comportamento descrito para acessos ao *cache* L1 de instruções:

Figura 15 - Curvas de comportamento de acesso ao *cache* L1 de instruções para os *benchmarks* CRAFTY e GCC.

Figura 16 - Curvas de comportamento de acesso ao *cache* L1 de instruções para os *benchmarks* APPLU e MESA.

As figuras relativas aos demais *benchmarks* podem ser encontradas no apêndice D.b.

Observando-se as figuras do *cache* L1, tanto para dados quanto para instruções, é possível diferenciar o tipo de *cache* com uma análise visual. O *cache* de instruções apresenta muitos endereços com grandes frequências de ocorrência (grande localidade), enquanto o *cache* de dados apresenta uma quantidade proporcionalmente menor de endereços com tal característica, o que faz com que

sua curva seja mais inclinada e com uma concentração menor de pontos na parte superior esquerda do gráfico.

8.2.3 CACHE L2

Para o *cache* L2, apenas CRAFTY, GCC e VPR podem ser aproximados a retas de forma razoável. Por outro lado, as aplicações APPLU, ART, EON, EQUAKE, FMA3D, MESA e MGRID apresentam comportamentos bastante diferenciados, possuindo grandes patamares e muitas vezes com diferenças muito significativas entre um e outro no que diz respeito à frequência (ordens de grandeza).

O gráfico de TWOLF apresenta uma série de pequenos patamares, mostrando um comportamento mais intermediário entre os dois conjuntos anteriormente citados. AMMP por sua vez, possui um gráfico diferente de todos os demais, assemelhando-se mais aos gráficos anteriormente apresentados nos acessos ao *cache* L1 de instruções.

Figura 17 - Curvas de comportamento de acesso ao *cache* L2 para os *benchmarks* CRAFTY e GCC.

Figura 18 - Curvas de comportamento de acesso ao *cache* L2 para os *benchmarks* APPLU e MESA.

As figuras relativas aos demais *benchmarks* podem ser encontradas no apêndice D.c.

Para os *benchmarks* CRAFTY e GCC, as curvas dos três *caches* indicam que existem muitos endereços responsáveis pela maioria dos acessos a dados ao longo da execução de cada uma das aplicações. Por outro lado, no que diz respeito a endereços contendo instruções, esse comportamento muda, indicando que existem poucas instruções com baixa localidade. Para os demais *benchmarks* é necessário realizar uma avaliação mais criteriosa, pois a quantidade de instruções executadas nas simulações e as massas de dados de entrada de cada *benchmark* podem ser muito pequenas, o que leva à geração de gráficos de difícil interpretação. Esta avaliação será feita adiante no trabalho, na avaliação de desempenho do *cache* Pareto.

8.2.4 CONCLUSÕES SOBRE AS CURVAS DE COMPORTAMENTO

Analisando-se as curvas, é possível perceber que há pouca concentração de pontos nas frequências mais elevadas – e que consequentemente estão nas primeiras posições do *ranking*. Essa disparidade indica que poucos endereços são responsáveis pela grande maioria dos acessos aos *caches* ao longo da execução das aplicações.

Como mencionado no capítulo 3, curvas com valores de tangente próximos a 1 indicam semelhança com a curva da Lei de Zipf, o que significa que o segundo endereço mais acessado no *cache* possui frequência de acessos próxima à metade

da frequência do endereço mais acessado, e o terceiro no *ranking* possui 1/3 dos acessos do primeiro. Desta forma, a curva segue a razão $1/x$, sendo x o *ranking*. Essa mesma avaliação mostra, segundo o Princípio de Pareto, que 20% do total de endereços acessados durante a execução de uma dada aplicação seria responsável por mais de 80% de todos os acessos feitos ao *cache* do processador (neste caso o *cache* estudado – L1 de dados, L1 de instruções ou L2).

No entanto, percebe-se a partir da análise dos gráficos e dos resultados dos *traces* que a inclinação das curvas é maior que 45 graus (tangente maior que 1) em vários casos. Isso significa que essa relação de discrepância é ainda maior, ou seja, menos de 20% do total de endereços acessados sendo responsável por mais de 80% do total de acessos. Essa disparidade é notada também nos gráficos que apresentam patamares.

Conclui-se com isso que os endereços que possuem maior frequência de acesso também possuem maior probabilidade de serem acessados ao longo da execução da aplicação, logo devem estar dentro do *cache* a maior parte do tempo. Essa conclusão sugere que, dentro do processador, uma pequena parcela dos endereços deva ser privilegiada de acordo com a sua probabilidade de ocorrer novamente, armazenadas no *cache* ou dentro de um *buffer* auxiliar de baixa latência como o *cache* Pareto. Desta forma, garante-se que a latência média de acesso a esses endereços seja reduzida, o que por sua vez reduz a latência média ao longo da execução de toda a aplicação, melhorando o desempenho do processador.

Armazenados no *cache* Pareto, esses endereços com alta probabilidade de acesso, ainda que sejam removidos do *cache* após um determinado tempo (devido à política de substituição LRU), estão armazenados em uma memória de baixa latência. Sendo assim, quando forem acessados novamente, não haverá penalidade de espera pela busca do dado na memória, já que este dado está próximo ao processador, armazenado no *cache* Pareto. Além disso, devido ao tamanho reduzido do *cache* Pareto, sua latência de acesso é menor que a dos *caches*, logo mesmo que o endereço acessado esteja no *cache*, a latência de acesso será menor acessando-o no *cache* Pareto.

Ao longo da execução de uma dada aplicação, é esperado que haja acessos recorrentes a um mesmo conjunto de endereços. Isso poderia ser explicado por um *loop* de execução, ou pela operação sobre um conjunto único de dados. Estas são possíveis explicações para o comportamento de curvas que apresentam patamares,

que significam que um variado conjunto de endereços possui a mesma frequência de ocorrência.

8.3 RESULTADOS DA REGRESSÃO LINEAR

A partir dos recursos de regressão linear oferecidos pelo aplicativo R (2009), foram geradas as retas médias oriundas dos *traces* de cada aplicação.

Para a análise deste trabalho, serão consideradas as inclinações das retas. Uma tabela contendo cada coeficiente de inclinação é apresentada a seguir. Coeficientes maiores que 1 indicam inclinação maior que 45 graus. Coeficientes próximos a 1 indicam proximidade do comportamento descrito pela Lei de Zipf e do Princípio de Pareto.

Tabela 3 - Tabela de coeficientes de inclinação obtidos com a regressão linear das curvas de comportamento de acesso aos *caches*.

Benchmarks	Coeficientes		
	L1 - dados	L1 - instruções	L2
AMMP	0.596992	0.229303	0.398525
APPLU	0.7723702	0.227420	1.668665
ART	0.5040789	0.215882	0.338869
CRAFTY	0.8957684	0.2621380	0.7838870
EON	0.4148143	0.2647070	0.507281
EQUAKE	0.967227	0.172642	1.59648
FMA3D	0.465352	0.232875	0.256358
GCC	0.6825538	0.3343869	0.5874221
MESA	0.493643	0.183445	1.83470
MGRID	0.6583848	0.370587	2.978104
TWOLF	0.419513	0.202986	1.54490
VPR	0.9947012	0.210069	1.305425

A partir dos coeficientes, apenas os *benchmarks* APPLU, CRAFTY, EQUAKE e VPR, avaliados para acessos ao *cache* L1 de dados, e CRAFTY para o *cache* L2, possuem valores próximos a 1, com margem de 25%, sugerindo que são as únicas

curvas próximas à Lei de Zipf. Todos os coeficientes relacionados ao *cache* L1 são menores que 1 e vários dos coeficientes relacionados ao *cache* L2 são maiores que 1.

A inclinação da reta média obtida com o método de regressão linear está de acordo com as expectativas ao mostrar que a maior concentração de pontos está nas frequências mais baixas, e consequentemente nos *rankings* mais à direita do gráfico. Portanto, existe uma tendência de perda de inclinação da reta, inerente ao método de regressão linear adotado e à grande concentração de endereços com baixa frequência de ocorrência.

Os valores obtidos para o L2, porém, são pouco conclusivos. Isso pode ser devido à quantidade relativamente pequena de instruções utilizadas nas simulações, e que não seria suficiente para gerar uma quantidade satisfatória de dados para analisar esse *cache*. Além disso, o tamanho das entradas dos *benchmarks* utilizados também poderia influenciar o resultado, uma vez que entradas muito pequenas não preencheriam todos os endereços disponíveis no L2, logo haveria uma quantidade pequena de *misses* nesse *cache*, o que também poderia distorcer a análise.

Confirmando aquilo que foi discutido anteriormente, para o *cache* L1, um *cache* Pareto baseado em *Power Laws*, possibilitaria ganhos de desempenho ao proporcionar uma latência média de acessos reduzida. Para o L2, contudo, isso não se confirma, visto que os coeficientes de inclinação variam muito de *benchmark* para *benchmark*.

8.4 ANÁLISE DE DESEMPENHO DO CACHE PARETO

A partir do modelo teórico proposto anteriormente, foi implementado um *cache* Pareto para armazenar os endereços mais acessados do *cache* L1 de dados (DL1_Pareto) e dos endereços mais acessados do *cache* L2 (DL2_Pareto.). A implementação com o *cache* Pareto DL1_Pareto armazena os 64 endereços mais acessados do *cache* L1 de dados. Da mesma forma, a implementação com o *cache* Pareto DL2_Pareto armazena os 64 endereços mais acessados do *cache* L2. Feitas no *SimpleScalar*, as duas implementações foram comparadas à implementação padrão do simulador, que por sua vez é denominada nos testes como *Baseline*.

Para as simulações, foram utilizados os mesmos *benchmarks* utilizados na análise dos *traces* apresentada anteriormente. Cada um dos *benchmarks* teve 100 milhões de instruções executadas, sendo que as primeiras 2 milhões são avançadas e portanto não figuram nas estatísticas geradas ao término da simulação.

A métrica avaliada neste caso é o tempo de execução em ciclos, uma vez que a latência mais baixa do *cache* Pareto afeta diretamente tal métrica. Quanto mais acessos ao *cache* Pareto em busca dos endereços mais acessados, do *cache* L2 por exemplo, menor será o tempo de execução em ciclos uma vez que a latência associada ao acesso será menor, uma vez que a latência do *cache* Pareto é menor que dos demais *caches* do processador.

A figura 19 apresenta os tempo de execução em ciclos de cada *benchmark*, simulado nas duas implementações e com resultados normalizados em relação à versão sem *cache* Pareto (*Baseline*). Quanto menor o tempo de execução, melhor o desempenho e maior a utilização do *cache* Pareto. A partir da figura, é possível comprovar alguns dos aspectos citados na análise dos coeficientes gerados a partir da regressão linear das curvas apresentadas nas seções anteriores.

Figura 19 – Tempo de execução em ciclos. Resultados normalizados em relação ao *Baseline*.

Na análise de coeficientes, o *cache* L1 se mostra como um bom candidato a ter seus endereços mais acessados dentro do *cache* Pareto, uma vez que possui coeficientes menores que 1 e portanto uma concentração de acessos a uma parcela menor de endereços. Isso se torna mais evidente com os *benchmarks* EQUAKE,

MESA e TWOLF, nos quais se observa reduções próximas a 10% do tempo de execução, bem como do *benchmark* VPR, com uma redução próxima a 6%.

Por outro lado, o *cache* L2 apresenta ganhos poucos significativos, como esperado a partir da análise das curvas e coeficientes. Este ganho chega a pouco mais de 4% de redução do tempo de execução no caso do *benchmark* FMA3D, porém menor que 2% no caso dos demais *benchmarks*. Este comportamento pode ser explicado pelo tamanho do *cache* L2 bem como da massa de dados utilizada nas simulações de cada *benchmark*, de modo que existem poucas substituições no *cache* L2 e, possivelmente, boa parte do *working set* se mantém dentro do L1, o que faz com que o *cache* Pareto também seja pouco acessado, uma vez que neste caso ele se encontra entre o L1 de dados e o L2 e portanto será acessado apenas quando o *cache* L2 também for acessado, respondendo mais rapidamente no caso de endereços com grande frequência de acessos.

Em outras palavras, com as configurações de *cache* simuladas e os *benchmarks* utilizados, observa-se que, em termos de número de acessos, o *cache* L1 tem maior potencial de ser beneficiado pela inserção do *cache* Pareto se comparado ao *cache* L2, no entanto, como a diferença de latência entre os dois é muito pequena (entre 1 e 2 ciclos para o *cache* L1), o impacto no resultado final somente é significativo para uma grande quantidade de instruções executadas (da ordem de centenas de milhões). O oposto ocorre com o *cache* L2, no qual a diferença de latência em relação ao *cache* Pareto é mais significativa (superior a 7 ciclos de *clock*), porém a quantidade de acessos ao *cache* Pareto é menor, portanto o impacto no tempo total de execução acaba sendo menor que 5%, indicando que seria necessário executar *benchmarks* com entradas maiores ou com um número de instruções executadas da ordem de bilhões ou dezenas de bilhões.

8.5 FOREST FIRE SWITCHING MECHANISM - FFSM

O simulador *SimpleScalar* foi utilizado para modelar e avaliar o desempenho de cada algoritmo de substituição de linhas de *cache* proposto, bem como dos algoritmos apresentados como referência. Ratificando, todas as estruturas auxiliares do ADEA bem como do FFSM foram inseridas no módulo de *cache* do *SimpleScalar* para suportar a operação do ADEA tal como demonstrado por Kobayashi (2010). O

mecanismo de chaveamento foi feito utilizando-se como partida a implementação do ADEA. Foram simulados tamanhos padrão do *cache* L2 enquanto o *cache* L1 (tanto dados quanto instruções) permaneceu o mesmo, focando no comportamento do *cache* de maior associatividade no processador modelado.

Usando a mesma metodologia empregada em (SCOTON; MARINO; KOBAYASHI, 2010) foi utilizado um processador com execução fora de ordem (*OOO-core*) por seu maior paralelismo intrínseco, que por sua vez leva a um padrão de preenchimento do *cache* mais difícil de se prever e portanto mais interessante para o objetivo do trabalho. Independentemente, assim como o algoritmo ADEA, o FFSM poderia também ser usado em processadores com execução ordenada de instruções (*in-order*). Para comparação, o LRU foi assumido como linha de base para todas as simulações de comparação de desempenho, desta forma cada um dos resultados dos diferentes algoritmos (LIP, BIP, ADEA e variações) aparece normalizado com base no resultado do LRU para o mesmo *benchmark*.

Os resultados mostrados nas figuras a seguir foram obtidos simulando-se 20 bilhões de instruções de 19 programas do SPEC CPU2000, com as três configurações do ADEA apresentadas por Kobayashi (2010) e duas novas implementações com o FFSM. Os números apresentados nos gráficos cobrem os resultados de taxas de *misses* (*miss rate*) relativos ao LRU, que é a linha de base para todos os resultados de desempenho. Para se ter uma referência de desempenho mais atual que o LRU, neste trabalho foram portados os algoritmos LIP e BIP (QURESHI et al., 2007) para o *SimpleScalar*. Como trabalho futuro, pretende-se desenvolver uma implementação de FFSM para estes algoritmos, portanto para os resultados aqui exibidos, os algoritmos não apresentam o mecanismo proposto. Sendo assim, são apresentados os desempenhos dos algoritmos LIP, BIP, ADEA com funções decaimento e ADEA com o FFSM, todos com resultados normalizados em relação aos resultados obtidos com o LRU, também simulado.

Figura 20 - Miss Rate para um cache L2 de 512 kB. Resultados normalizados em relação ao LRU.

Figura 21 - Miss Rate para um cache L2 de 1024 kB. Resultados normalizados em relação ao LRU.

Figura 22 - Miss Rate para um cache L2 de 2048 kB. Resultados normalizados em relação ao LRU.

Com o mecanismo de chaveamento e para um *cache* L2 de 1024 kB, o ADEA modificado (RRF-FR e FR-RRF) foi capaz de melhorar os resultados em cerca de 6% (resultados calculados com base em uma média harmônica) se comparado ao ADEA sem qualquer função decaimento, como era esperado de acordo com o trabalho de Kobayashi (2010). Resultados melhores podem ser vistos nos *benchmarks* APPLU, APSI, CRAFTY, EQUAKE, LUCAS, MGRID, SWIM e VPR. Resultados semelhantes em termos de desempenho podem ser vistos no caso dos *benchmarks* EON, FMA3D, GAP, GZIP e PARSE. Os demais *benchmarks* apresentaram resultados cerca de 1% melhores com o ADEA padrão, ainda que comparados às versões com funções decaimento, como observado por Kobayashi (2010).

Além dos resultados apresentados, a figura 23 apresenta resultados para um *cache* L2 de 2048 kB e associatividade 16, a fim de se observar o impacto da mudança do valor de associatividade. Para isso, foram realizadas simulações com apenas uma parcela dos *benchmarks*, escolhidos de acordo com o potencial de se observar diferenças mais significativas entre os valores de *miss rate* dos algoritmos, advindo da análise dos demais resultados.

Figura 23 – Miss Rate para um *cache* L2 de 2048 kB e associatividade 16. Resultados normalizados em relação ao LRU.

Nota-se que alguns *benchmarks* são prejudicados pelo aumento do tamanho do *cache* L2. No *benchmark* GCC, para um *cache* L2 de 512 kB, o RRF-FR e o FR-RRF, juntamente com o ADEA-FR-decay e o ADEA-RRF-decay mostraram resultados melhores que o ADEA padrão e os demais algoritmos de referência. No

entanto, com o aumento do *cache* para 1024 kB e 2048 kB estes resultados sofreram mudanças, de forma que o ADEA e suas variações passaram a ter resultados inferiores aos demais, sendo que no *cache* de 2048 kB as funções decaimento e consequentemente os algoritmos com o FFSM apresentaram resultados até 7 vezes piores que o LRU (caso do *benchmark* GCC). Apesar do aumento do tamanho do *cache* L2 nas simulações, os conjuntos de dados de entrada para os *benchmarks* não foram alterados, continuando os mesmos para todas as simulações realizadas. Desta forma, pode-se presumir que ao aumentar-se a associatividade do *cache* e o seu tamanho, com um conjunto de dados menor que o *cache*, não se obtém benefícios utilizando-se o ADEA ou alguma de suas variações para alguns *benchmarks* como o GCC.

Como mencionado anteriormente, o FFSM se adapta entre o FR-decay e o RRF-decay, indo de um algoritmo para o outro através do disparo por *calls* e *returns*. Comparado ao FR-decay e ao RRF-decay, o mecanismo de chaveamento foi sempre semelhante ou entre o melhor e o pior resultado entre os dois. O trabalho de Kobayashi (2010) apresenta as funções decaimento implementadas isoladamente, desta forma há diferença de desempenho entre ambas para diferentes *benchmarks*, pois cada função decaimento é melhor adaptada a um determinado tipo de aplicação. Com o chaveamento, é possível obter ganhos das duas funções em uma mesma aplicação, de forma que no pior caso ter-se-ia uma quantidade pequena de chaveamentos, com resultado comparável ao pior resultado do ADEA com decaimento, porém não inferior a este. Este é um resultado esperado, uma vez que ambas as implementações do mecanismo estão chaveando entre as duas funções decaimento, sendo assim tem-se um algoritmo mais adaptado à diversidade de aplicações. É interessante notar que para alguns *benchmarks*, os pares de resultados similares se apresentam de tal forma que podemos assumir que alguns programas possuem muitos *calls* ou gastam mais instruções dentro de *calls*. Isto pode ser observado com os *benchmarks* CRAFTY, GCC, MESA, MGRID, SWIM, TWOLF e VPR que possuem pares de resultados que indicam que tais programas apresentam muitas instruções dentro de *calls*, por exemplo: o chaveamento de FR-decay para RRF-decay possui quase os mesmos resultados que o modelo RRF-decay.

A melhoria de desempenho que vem do Modelo de Incêndios Florestais implementado pelo FFSM, ajuda a entender como os chaveamentos de contexto afetam o comportamento do algoritmo de substituição de linhas de *cache*, uma vez que para o ADEA frequência e recência são os principais aspectos de cada função decaimento.

Além da taxa de *misses* (*miss rate*), foram avaliadas estatísticas do número de instruções executadas por ciclo, ou IPC (*instructions per cycle*), apresentadas na figura para um *cache* L2 de 1024 kB. Para melhor visualização e diferentemente dos gráficos de *miss rate*, os resultados de IPC são apresentados em seus valores absolutos, não sendo portanto normalizados com relação ao LRU.

Figura 24 - IPC para um *cache* L2 de 1024 kB. Resultados em valor absoluto.

Observando-se o gráfico da figura percebe-se que existe pouca variação do IPC entre os algoritmos para os diversos *benchmarks* simulados (menor que 5%). Por isso e pela falta de relação imediata entre os resultados de IPC e de *miss rate*, as demais figuras contendo o IPC para os caches L2 de 512 kB e 2048 kB são apresentadas no apêndice E.

8.5.1 FFSM PARA OS ALGORITMOS DE QURESHI

Com o objetivo de estender o estudo do FFSM para outros algoritmos que não o ADEA, foi realizada uma modelagem do mecanismo chaveando entre o

algoritmo BIP – *Bimodal Insertion Policy* (QURESHI et al., 2007) e o LRU. No trabalho original de Qureshi, são apresentados três algoritmos, sendo o principal deles o DIP – *Dynamic Insertion Policy*. O comportamento dinâmico do DIP escolhe, em tempo de execução, qual algoritmo de substituição de linhas de *cache* é mais adequado para manter o melhor desempenho, escolhendo entre os algoritmos BIP e LRU. Desta forma, para o FFSM, os algoritmos BIP e LRU também foram os escolhidos, assim poder-se-ia ter um algoritmo dinâmico mas com critérios diferentes do DIP.

Para este estudo do FFSM são criados dois novos modelos:

- CALLBIP: muda de LRU para BIP toda vez que há uma instrução de *call*, chaveando novamente para LRU quando há uma instrução de *return*;
- CALLLRU: muda de BIP para LRU toda vez que há uma instrução de *call*, chaveando novamente para BIP quando há uma instrução de *return*.

Os resultados são apresentados na figura 25.

Figura 25 - Miss Rate para um cache L2 de 1024 kB. Resultados normalizados em relação ao LRU.

A partir da figura 25 é possível notar pares de resultados semelhantes, indicando como os algoritmos se comportam no que diz respeito à quantidade de instruções *call* e *return*. Por exemplo, para o *benchmark* AMMP, os resultados do BIP e do CALLBIP são muito próximos e com desempenho melhor que do par LRU e CALLLRU, que por sua vez também possuem entre si resultados muito próximos. Tais resultados sugerem o chaveamento do BIP para o LRU através de um *call* no

modelo CALLLRU, mantendo-se neste estado durante a maior parte do tempo de execução. Com isso seu comportamento se mostrou muito semelhante ao do algoritmo LRU.

De forma oposta, para o *benchmark* APSI, os pares de resultados se mostram outros, dando uma nova visão ao comportamento dos algoritmos frente a variações nos tipos de instruções dos *benchmarks*. Para o *benchmark* APSI, os algoritmos LRU e CALLBIP possuem o melhor desempenho, o que sugere que existem poucas instruções de *call*, o que por sua vez faz com que haja poucos chaveamentos do FFSM. Desta forma, o CALLBIP tende a se comportar mais próximo ao LRU e o CALLLRU mais próximo ao BIP, o que pode ser observado pelos resultados na figura 25.

Os *benchmarks* BZIP, PARSER e VPR são exemplos nos quais o BIP possui desempenho inferior ao do LRU. Além disso, para os mesmos *benchmarks*, ambos os modelos que se utilizam do FFSM mostram-se melhores que o BIP. Isso significa que existe um balanço mais equivalente do tempo de execução no qual a política vigente é a LRU ou BIP. Este balanço é devido à quantidade de *calls* em conjunto com o tempo dentro de cada *call*, levando os dois modelos baseados no FFSM a melhorar o desempenho do BIP original, aproximando-se mais do melhor desempenho obtido com o LRU. Desta forma conclui-se que a aplicação do FFSM a outros algoritmos que não apenas o ADEA é justificada, uma vez que o objetivo final é a utilização de um algoritmo de substituição de linhas de *cache* mais bem adaptado à variedade de aplicações que podem ser executadas por um microprocessador.

9 CONCLUSÕES E TRABALHOS FUTUROS

O trabalho se propõe a estudar as *Power Laws* como ferramental estatístico para a melhoria de desempenho de *caches* de microprocessadores, ao avaliar a possibilidade de utilização deste conhecimento nos acessos aos *caches*.

Através do uso de um simulador, foram gerados *traces* de execução nos quais se obteve listas de endereços acessados em cada um dos *caches*. A partir dos *traces*, foram geradas curvas de frequência versus *ranking* e, utilizando-se um modelo de regressão linear, tais curvas foram aproximadas a retas. Cada uma dessas curvas foi analisada sob o ponto de vista de *Power Law* e comparada com leis correlatas, como a Lei de Zipf e o Princípio de Pareto, a fim de se verificar se seguiriam ou não tais leis empíricas.

A análise das curvas de comportamento de acessos aos *caches* mostrou que não há aderência à Lei de Zipf e ao Princípio de Pareto (exemplos de *Power Laws*), pois a inclinação da reta média é diferente se comparada aos dois casos. Para os casos em que a inclinação é maior que 1, é possível concluir que o *working set* das aplicações avaliadas corresponde a menos de 20% do total da aplicação armazenada na memória e que essa pequena parcela do total de endereços corresponde a mais de 80% do total de acessos aos *caches* durante a execução da aplicação. Quanto maior a inclinação dessas retas, menor a parcela do *working set* no total da aplicação.

No trabalho de Kotera et al. (2008) foram estudados os comportamentos de acesso aos *caches*, em especial a localidade destes acessos, utilizando-se a técnica de *stack distance profiling*. As curvas de distribuição obtidas - *stack distance distributions* - foram comparadas à Lei de Zipf. O objetivo dos autores era o de se atingir um particionamento eficiente do *cache* L2 através de um modelo baseado na Lei de Zipf, capaz de prever as mudanças em tais distribuições, redimensionando a porção do *cache* disponível a cada *thread* em um CMP.

Este trabalho se diferencia do trabalho de Kotera et al. (2008) por não fazer uso da técnica de *stack distance profiling*, o que não restringe o estudo à localidade de acessos. A avaliação é feita diretamente com os endereços acessados, tomando-se o *ranking* desses endereços e sua frequência de ocorrência. Kotera et al. (2008) se concentra no compartilhamento eficiente do *cache* L2 em um CMP através de estimativas quanto ao tamanho do *working set* da aplicação.

Como parte do trabalho é apresentado um “pequeno” *cache* Pareto, baseado nos princípios das *Power Laws*, independente do número de *cores* e criado a fim de manter próximos ao processador os endereços que são de fato mais acessados por este, objetivando melhorar o desempenho do algoritmo de substituição. O trabalho apresenta resultados de avaliações de desempenho de configurações de *cache* contendo o *cache* Pareto e comparando-as com uma versão padrão. A introdução do *cache* Pareto apresenta ganhos de desempenho, com diminuição no número total de ciclos durante a simulação. A criação de um mecanismo de *prefetching* baseado em *Power Laws* e com o objetivo de buscar endereços antecipadamente é uma proposta para um trabalho futuro.

Foi proposto também o *Forest Fire Switching Mechanism* - FFSM, com uma implementação específica para o algoritmo ADEA e suas diferentes funções decaimento. Analisando o comportamento dos programas do SPEC CPU2000 através das simulações, pôde-se observar que tais modelos podem ser melhorados com uma mudança na forma como o algoritmo se comporta sempre que uma mudança de contexto ocorre. Pretende-se aplicar o modelo a outros algoritmos adaptativos como trabalho futuro. Para o ADEA conclui-se que não se deve priorizar recência ou frequência, mas chavear entre ambas dependendo do impacto do chaveamento de contexto para um determinado programa. Uma modelagem inicial foi realizada com o algoritmo BIP do autor Qureshi. Como trabalho futuro pretende-se utilizar o FFSM para chavear entre o LIP e o BIP, entre o LIP e o LRU, ou alguma outra variação utilizando o algoritmo DIP.

O modelo proposto pode ajudar a entender como identificar características importantes de um programa durante sua execução, sem pistas dadas por seu código-fonte. O estudo sobre como o compilador pode ajudar a identificar estas características e deixar tais pistas é assunto para um trabalho futuro. Outra ideia é verificar a intensidade de diferentes tipos de *calls*, chaveando entre diferentes comportamentos de acordo com esta intensidade, e não em todas as ocorrências de *calls* como foi implementado.

REFERÊNCIAS BIBLIOGRÁFICAS¹⁴

ADAMIC, L.A. **Zipf, power-law, Pareto - a ranking tutorial**. Disponível em: <<http://www.hpl.hp.com/research/idl/papers/ranking>>. Acesso em: 14 Jul. 2009.

ADAMIC, L.A.; HUBERMAN, B.A. The nature of markets in the World Wide Web. **Quarterly Journal of Electronic Commerce**. v. 1, p. 512, 2000.

AGARWAL, A.; HOROWITZ, M.; HENNESSY, J. An analytical cache model. **Computer Systems Lab. Rep.** Stanford Univ. Stanford, Calif., TR. 86-304, Set. 1986.

AGGARWAL, N.; RANGANATHAN, P.; JOUPPI, N.P.; SMITH, J.E. Configurable isolation: building high availability systems with commodity multi-core processors. In: **Proceedings of the 34th Annual International Symposium on Computer Architecture** (San Diego, California, USA, Jun. 09-13, 2007). ISCA '07. ACM, New York, NY, 470-481.

AMD – Advanced Micro Devices. Disponível em: <www.amd.com>. Acesso em: 21 Jul. 2009.

AUERBACH, F. Das Gesetz der Bevölkerungskonzentration. **Petermanns Geographische Mitteilungen**. v. 59, p. 74–76, 1913.

AUSTIN, T.; LARSON, E.; ERNST, D. SimpleScalar: an infrastructure for computer system modeling computer. **IEEE Computer Magazine**. v. 35, n. 2, p. 59-67, Feb. 2002, doi:10.1109/2.982917

BECKMANN, B.M.; MARTY, M.R.; WOOD, D.A. ASR: adaptive selective replication for CMP caches. In: **Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture** (Dec 09-13, 2006). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, Dec. 2006, p. 443-454.

BERG, E.; ZEFFER, H.; HAGERSTEN, E. A. Statistical Multiprocessor Cache Model. In: **Proceedings of International Symposium on Performance Analysis of Systems and Software**, Mar. 2006.

BISWAS, S. et al. Multi-execution: multicore caching for data-similar executions. **SIGARCH Comput. Archit. News**. v. 37, n. 3, p. 164-173, Jun. 2009.

BRESLAU, L.; CAO, P.; PHILIPS, G.; SHENKER, S. Web caching and Zipf-like distributions: Evidence and implications. In: **Proceedings of the IEEE INFOCOM**. IEEE Computer Society, Piscataway - NJ, Mar. 1999, p. 126-134.

BUCK, I. GPU computing: programming a massively parallel processor. In: **Proceedings of the International Symposium on Code Generation and Optimization** (Mar. 11-14, 2007). Code Generation and Optimization. IEEE Computer Society, Washington, DC, Mar 2007, v. 17.

¹⁴ De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

CACTI. **HP Labs: CACTI**. Disponível em: <<http://www.hpl.hp.com/research/cacti/>>. Acesso em: 14 Jul. 2009.

CALDER, B.; KRINTZ, C.; JOHN, S.; AUSTIN, T. Cache-conscious data placement. **SIGPLAN Not.** v. 33, n. 11, Nov. 1998, p.139-149.

CANTIN, J.F.; LIPASTI, M.H.; SMITH, J.E. Stealth prefetching. In: **Proceedings of the 12th international Conference on Architectural Support For Programming Languages and Operating Systems**, (San Jose, California, USA, Oct. 21-25, 2006). ASPLOS-XII. ACM, New York, NY, p. 274-282.

CHANDRA, D.; GUO, F.; KIM, S.; SOLIHIN, Y. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In: **Proceedings of the 11th international Symposium on High-Performance Computer Architecture** (Feb. 12-16, 2005). HPCA. IEEE Computer Society, Washington, DC, p. 340-351.

CHEN, T.; BAER, J. Reducing memory latency via non-blocking and prefetching caches. In: **Proceedings of the Fifth international Conference on Architectural Support For Programming Languages and Operating Systems** (Boston, Massachusetts, United States, Oct. 12-15, 1992). R. L. Wexelblat, Ed. ASPLOS-V. ACM, New York, NY, p. 51-61.

CHEONG, H.; VEIDENBAUM, A.V. Compiler-directed cache management in multi-processors. **Computer**. v. 23, n. 6, p. 39-47, Jun. 1990.

CHILIMBI, T.M.; SHAHAM, R. Cache-conscious coallocation of hot data streams. **SIGPLAN Not.** v. 41, n. 6, Jun. 2006, p. 252-262.

CHOU, Y. Low-Cost Epoch-Based Correlation Prefetching for Commercial Applications. In: **Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec 01-05, 2007). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 301-313.

CLAUSET, A.; SHALIZI, C.R.; NEWMAN, M.E.J. Power-law distributions in empirical data. **ArXiv e-print 0706.1062** v1, Jun. 2007.

COX, R.A.K.; FELTON, J.M.; CHUNG, K.C. The concentration of commercial success in popular music: an analysis of the distribution of gold records. **Journal of Cultural Economics**. v. 19, n. 4, p. 333-340, 1995.

CROVELLA, M.E.; BESTAVROS, A. Self-similarity in World Wide Web traffic: evidence and possible causes. In: B. E. Gaither and D. A. Reed (eds.), **Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems**. Association of Computing Machinery, New York, 1996, p. 148-159.

DENNING, P.J. The working set model for program behavior. **Communications of the ACM**, v.11, n. 5 May. 1968, p. 323-333.

DIAZ, P.; CINTRA, M. Stream chaining: exploiting multiple levels of correlation in data prefetching. In: **Proceedings of the 36th Annual international Symposium on Computer Architecture** (Austin, TX, USA, Jun. 20-24, 2009). ISCA '09. ACM, New York, NY, p. 81-92.

DROSSEL, B.; SCHWABL, F. Self-organized critical forestfire model. **Phys. Rev. Lett.** n. 69, Sep. 1992, p. 1629-1632.

DYBDAHL, H.; STENSTRÖM, P.; NATVIG, L. An LRU-based replacement algorithm augmented with frequency of access in shared chip multiprocessor caches. In: **MEDEA '06: Proceedings of the 2006 workshop on Memory performance**. New York, NY, USA: ACM, 2006, p. 45-52.

ETSION, Y.; FEITELSON, D.G. Cache insertion policies to reduce bus traffic and cache conflicts. **Technical Report** 2006-4, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Feb. 2006.

FRIENDS OF SIMPLESCALAR LLC. Disponível em: <www.simplescalar.com/friends.html>. Acesso em: 21 Jul. 2009.

GNU PLOT. Disponível em: <www.gnuplot.info>. Acesso em: 21 Jul. 2009.

GOODMAN, J.R. Using cache memory to reduce processor-memory traffic. In: **The 10th Annual Symposium on Computer Architecture**. Jun. 1983, p. 124-131.

GORNISH, E.H.; GRANSTON, E.D.; VEIDENBAUM, A.V. Compiler-directed data prefetching in multiprocessors with memory hierarchies. In: **Proceedings of the 4th international Conference on Supercomputing** (Amsterdam, The Netherlands, Jun. 11-15, 1990). ICS '90. ACM, New York, NY, p. 354-368.

GUO, F.; SOLIHIN, Y. An analytical model for cache replacement policy performance. In: **Proceedings of the Joint international Conference on Measurement and Modeling of Computer Systems** (Saint Malo, France, Jun. 26-30, 2006). SIGMETRICS '06/Performance '06. ACM, New York, NY, p. 228-239.

GUTENBERG, B.; RICHTER, R.F. Frequency of earthquakes in California. **Bulletin of the Seismological Society of America**. v. 34, p. 185-188, 1944.

HARDAVELLAS, N.; FERDMAN, M.; FALSAFI, B.; AILAMAKI, A. Reactive NUCA: near-optimal block placement and replication in distributed caches. **SIGARCH Comput. Archit. News**. v. 37, n. 3, Jun. 2009, p. 184-195.

HUR, I.; LIN, C. Memory Prefetching Using Adaptive Stream Detection. In: **Proceedings of the 39th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec. 09-13, 2006). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 397-408.

HWU, W. et al. Compiler Technology for Future Microprocessors. In: **Proceedings of the IEEE**. v. 83, n. 12, p. 1625-1640, Dec. 1995.

IBM - INDUSTRIAL BUSINESS MACHINES. Disponível em: <www.ibm.com>. Acesso em: 21 Jul. 2009.

INTEL. Disponível em: <www.intel.com>. Acesso em: 21 Jul. 2009.

INTEL SMART CACHE. Intel. Disponível em: <<http://www.intel.com/technology/product/demos/cache/demo.htm>>. Acesso em: 14 Jul. 2009.

JOUPPI, N.P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In: **Proceedings of the 17th Annual international Symposium on Computer Architecture** (Seattle, Washington, United States, May. 28-31, 1990). ISCA '90. ACM, New York, NY, p. 364-373.

KIM, C.; BURGER, D.; KECKLER, S.W. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. **SIGPLAN Not.** v. 37, n. 10, Oct. 2002, p. 211-222.

KIRMAN, N. et al. Leveraging optical technology in future bus-based Chip Multiprocessors. In: **Proceedings of the 39th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec. 09-13, 2006). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 492-503.

KOBAYASHI, J.M. Entropy: algoritmo de substituição de linhas de cache baseado em entropia da informação. Dissertação de Mestrado. Escola Politécnica da Universidade de São Paulo. São Paulo, 2010.

KOHLI, R.; SAH, R. Market shares: some power law results and observations. **Working paper 04.01**, Harris School of Public Policy, University of Chicago, 2003.

KONGETIRA, P.; AINGARAN, K.; OLUKOTUN, K. Niagara: a 32-way multithreaded SPARC processor. **IEEE Micro.** v. 25, n. 2, Mar. 2005, p. 21-29.

KOTERA, I.; EGAWA, R.; TAKIZAWA, H.; KOBAYASHI, H. Modeling of cache access behavior based on Zipf's law. In: **Proceedings of the 9th Workshop on Memory Performance: Dealing with Applications, Systems and Architecture** (Toronto, Canada, Oct. 26-26, 2008). MEDEA '08, v. 310. ACM, New York, NY, p. 9-15.

KUMAR, R.; ZYUBAN, V.; TULLSEN, D. M. Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling. In: **Proceedings of the 32nd annual international symposium on Computer Architecture**. Madison, Wisconsin, USA. Jun. 04-08, 2005, p.408-419.

LEVERICH, J. et al. Comparing memory systems for chip multiprocessors. In: **Proceedings of the 34th Annual international Symposium on Computer Architecture** (San Diego, California, USA, Jun. 09-13, 2007). ISCA '07. ACM, New York, NY, p. 358-368.

LI, W. Random texts exhibit Zipf's-law-like word frequency distribution. **IEEE Transactions on Information Theory**, v. 38, n. 6, p. 1842-1845, 1992.

LIU, H.; FERDMAN, M.; HUH, J.; BURGER, D. Cache bursts: a new approach for eliminating dead blocks and increasing cache efficiency. In: **Proceedings of the 2008 41st IEEE/ACM international Symposium on Microarchitecture**. Nov. 08-12, 2008. International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 222-233.

LOTKA, A. J. The frequency distribution of scientific production. **J. Wash. Acad. Sci.** v. 16, n. 12, p. 317-323, 1926.

LU, E.T.; HAMILTON, R.J. Avalanches of the distribution of solar flares. **Astrophysical Journal, Part 2 - Letters** (ISSN 0004-637X), v. 380, Oct. 1991, p. L89-L92.

MADAN, N.; BALASUBRAMONIAN, R. Leveraging 3D technology for improved reliability. In: **Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture**. (Dec. 01-05, 2007). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 223-235.

MITZENMACHER, M. A brief history of generative models for power law and lognormal distributions. **Internet Math.** v. 1, n. 2, p. 226-251, 2004.

MOURA, N.; RIBEIRO, M. Zipf law for brazilian cities. **PHYSICA A.** v. 367, p.441, 2006.

MOWRY, T.C. Tolerating latency in multiprocessors through compiler-inserted pre-fetching. **ACM Trans. Comput. Syst.** v. 16, n. 1, Feb. 1998, p. 55-92.

MURALIMANO HAR, N.; BALASUBRAMONIAN, R. Interconnect design considerations for large NUCA caches. In: **Proceedings of the 34th Annual international Symposium on Computer Architecture** (San Diego, California, USA, Jun. 09-13, 2007). ISCA '07. ACM, New York, NY, p. 369-380.

MURALIMANO HAR, N.; BALASUBRAMONIAN, R.; JOUPPI, N. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In: **Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture**. (Dec. 01-05, 2007). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 3-14.

NEUKUM, G.; IVANOV, B.A. Crater size distributions and impact probabilities on Earth from lunar, terrestrial planet, and asteroid cratering data. In: **Gehrels T. (ed.) Hazards due to comets and asteroids**, University of Arizona Press, Tucson, AZ, 1994, p. 359-416.

NAZ, A.; REZAEI, M.; KAVI, K.; SWEANY, P. Improving data cache performance with integrated use of split caches, victim cache and stream buffers. In: **Proceedings of the 2004 Workshop on Memory Performance: Dealing with Applications, Systems and Architecture**. (Antibes Juan-les-Pins, France, Sep. 29–Oct. 03, 2004). MEDEA '04. ACM, New York, NY, p. 41-48.

NEWMAN, M.E.J. Power laws, Pareto distributions and Zipf's law. **Contemp. Phys.** n. 46, p. 323-351, 2005.

OLUKOTUN, K. et al. The case for a single-chip multiprocessor. In: **Proceedings of the Seventh international Conference on Architectural Support For Programming Languages and Operating Systems** (Cambridge, Massachusetts, United States, Oct. 01-04, 1996). ASPLOS-VII. ACM, New York, NY, p. 2-11.

OLUKOTUN, K.; HAMMOND, L. The future of microprocessors. **Queue**, v. 3, n. 7, Sep. 2005, p. 26-29.

PARETO, V. **Cours d'economie politique**. Librairie Droz. Geneva, 1896.

PETOUMENOS, P. et al. STATSHARE: A Statistical Model for Managing Cache Sharing via Decay. In: **Second Annual Workshop on Modeling, Benchmarking and Simulation** (MoBS 2006), 2006.

PATTERSON, D.A.; HENNESSY, J.L. **Computer architecture: a quantitative approach**. San Mateo, CA. Morgan Kaufmann, 1990.

PNUD. Programa das Nações Unidas para o Desenvolvimento. Cooperation au developpement Cap Vert / PNUD, 1993. **Relatório PNUD**.

PRICE, D.J.S. Networks of scientific papers. **Science**, v. 149, p. 510-515, 1965.

QURESHI, M.K. et al. Adaptive insertion policies for high performance caching. In: **Proceedings of the 34th Annual international Symposium on Computer Architecture** (San Diego, California, USA, Jun. 09-13, 2007). ISCA '07. ACM, New York, NY, p. 381-391.

QURESHI, M.K.; PATT, Y.N. Utility-based cache partitioning: a low-overhead, high-performance, runtime mechanism to partition shared caches. In: **Proceedings of the 39th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec. 09-13, 2006). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 423-432.

R - R STATISTICS. Disponível em: <openwetware.org/wiki/R_Statistics>. Acesso em: Jul. de 2009.

RAJAN, K.; RAMASWAMY, G. Emulating optimal replacement with a shepherd cache. In: **Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec. 01-05, 2007). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 445-454.

ROBERTS, D.C.; TURCOTTE, D.L. Fractality and self-organized criticality of wars. **Fractals**. n. 6, p. 351-357, 1998.

SCOTON, F.M.; MARINO, M.D.; KOBAYASHI, J.M. Forest Fires: improving a Cache Replacement Algorithm. In: **International Workshop on Unique Chips and Systems**, Dec. 2010.

SHANNON, C.E. Prediction and entropy of printed english. **Bell Systems Tech. Journal**, n. 30, p. 50-64, 1951.

SIMPLESCALAR LLC. Disponível em: <www.simplescalar.com>. Acesso em: 14 Jul. 2009.

SMITH, A.J. Cache Memories. **ACM Comput. Surv.**, v. 14, n. 3, Sep. 1982, p. 473-530.

SOMOGYI, S.; WENISCH, T.F.; AILAMAKI, A.; FALSAFI, B. Spatio-temporal memory streaming. **SIGARCH Comput. Archit. News**, v. 37, n. 3, Jun. 2009, p. 69-80.

SPEC – Standard Performance Evaluation Corporation. Disponível em: <www.spec.org>. Acesso em: 21 Jul. 2009.

STRAUSS, K.; SHEN, X.; TORRELLAS, J. Flexible snooping: adaptive forwarding and filtering of snoops in embedded-ring multiprocessors. In: **Proceedings of the 33rd Annual international Symposium on Computer Architecture**. (Jun. 17-21, 2006). International Symposium on Computer Architecture. IEEE Computer Society, Washington, DC, p. 327-338.

SUN - SUN MICROSYSTEMS. Disponível em: <www.sun.com>. Acesso em: 21 Jul. 2009.

TULLSEN, D.M.; EGGERS, S.J.; LEVY, H.M. Simultaneous multithreading: maximizing on-chip parallelism. In: **Proceedings of the 22nd annual international symposium on Computer Architecture**, p.392-403, Jun. 22-24, 1995, S. Margherita Ligure, Italy.

TYSON, G.; FARRENS, M.; MATTHEWS, J.; PLESZKUN, A.R. A modified approach to data cache management. In: **Proceedings of the 28th Annual international Symposium on Microarchitecture** (Ann Arbor, Michigan, United States, Nov. 29-Dec. 01, 1995). International Symposium on Microarchitecture. IEEE Computer Society Press, Los Alamitos, CA, p. 93-103.

VAN DER PAAS, R. **Memory Hierarchy in Cache-based Systems**, Edition Nov. 2002. Disponível em: <<http://www.sun.com/blueprints>>. Acesso em: 14 Jul. 2009.

VARADARAJAN, K. et al. Molecular caches: a caching structure for dynamic creation of application-specific heterogeneous cache regions. In: **Proceedings of the 39th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec. 09-13, 2006). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 433-442.

WILLIS, J.C.; YULE, G.U. Some statistics of evolution and geographical distribution in plants and animals, and their significance. **Nature**, n. 109, p. 177-179, 1922.

XIE, Y.; LOH, G.H. PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches. In: **Proceedings of the 36th Annual International Symposium on Computer Architecture** (Austin, TX, USA, Jun. 20-24, 2009). ISCA '09. ACM, New York, NY, p. 174-183.

ZANETTE, D.H.; MANRUBIA, S.C. Vertical transmission of culture and the distribution of family names. **Physica A.**, n. 295, p. 1-8, 2001.

ZEBCHUCK, J.; MAKINENI, S.; NEWELL, D. Re-examining cache replacement policies. **ICCD 2008**. 671-678, Oct. 2008

ZEBCHUCK, J.; SAFI, E.; MOSHOVOS, A.A Framework for coarse-grain optimizations in the on-chip memory hierarchy. In: **Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture** (Dec. 01-05, 2007). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, p. 314-327.

ZHANG, M.; ASANOVIC, K. Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors. In: **Proceedings of the 32nd Annual International Symposium on Computer Architecture** (Jun. 04-08, 2005). International Symposium on Computer Architecture. IEEE Computer Society, Washington, DC, p. 336-345.

ZHANG, J.; FAN, X.; LIU, S. A pollution alleviative L2 cache replacement policy for chip multiprocessor architecture. In: **Proceedings of the 2008 international Conference on Networking, Architecture, and Storage** (Jun. 12-14, 2008). NAS. IEEE Computer Society, Washington, DC, p. 310-316.

ZIPF, G. K. **Human Behaviour and the principle of least effort**. Addison-Wesley, Reading, MA, 1949.

ZIPF, G.K. **Selective studies and the principle of relative frequency in language**. Cambridge, Massachusetts: MIT, 1932.

ZITEL CO. (United States). Michael A. Salsburg. Method and structure for evaluating and enhancing the performance of cache memory systems, **Patent US 5452440**, Jul. 16, 1993.

GLOSSÁRIO

Benchmark	Execução de um programa com o objetivo de avaliação de desempenho
Buffer	Memória temporária
Cache	Área de armazenamento temporário
Cache controller	Controlador de <i>cache</i>
Cache hit	Acerto de <i>cache</i>
Cache miss	Falha de <i>cache</i>
CMP	<i>Chip Multiprocessor</i>
Commodities	Algo que apesar da demanda é oferecido pelo mercado sem grande diferenciação
Core	Núcleo de processamento
CPU	<i>Central Processing Unit</i>
Dead block	Bloco que não será mais utilizado
Die	Pedaço retangular de um <i>wafer</i> de semicondutor
DRAM	<i>Dynamic Random Access Memory</i>
Framework	Estrutura básica conceitual usada para resolver problemas complexos
Full associative	Totalmente associativo
GPU	<i>Graphics Processing Unit</i>
Heap	Estrutura de programação em forma de árvore
Hit	Acerto
Layout	Traçado
Loop	Laço
Lookup table	Estrutura em forma de <i>array</i>
LRU	<i>Least Recently Used</i>
Memory bank	Banco de memória
MHA	Miss Handling Architecture
Merge	Juntar, fundir
MIMD	<i>Multiple Instruction Multiple Data</i>
Miss	Falha
Multicore	Dotado de vários <i>cores</i>
Multithreading	Suporte para execução de múltiplas <i>threads</i>
NUCA	<i>Non-Uniform Cache Access</i>
Off-chip	Fora do <i>chip</i>

On-chip	Dentro do <i>chip</i>
On-the-fly	Em tempo de execução
Overhead	Tempo de computação em excesso
Power Law	Lei de Potência
Prefetching	Pré-carregamento
Private	Privado
Profiling	Investigação do comportamento de um programa
Proxies	Servidor intermediário de rede
Ranking	Posição em uma lista ordenada
Sampling	Amostragem
Script	Roteiro
Set	Conjunto
Shared	Compartilhado
SIMD	<i>Single Instruction Multiple Data</i>
Sites	Localidades
Slice	Fatia
SMT	<i>Simultaneous multithreading</i>
SRAM	<i>Static Random Access Memory</i>
Thrashing	Situação degenerativa do sistema
Thread	Sequência de instruções que pode ser executada em paralelo com outras <i>threads</i>
Trace	Rastro
Wafer	Fatia de material semiconductor
Warm-up	Aquecimento
Wire delay	Atraso de interconexão
Working set	Área de trabalho

APÊNDICE

A. SIMULTANEOUS MULTITHREADING – SMT

O *multithreading* é uma técnica que permite que várias *threads* compartilhem as unidades funcionais de um único processador com sobreposição (PATTERSON; HENNESSY, 1990).

O *multithreading* simultâneo (SMT - *simultaneous multithreading*) é uma variação do *multithreading* que permite que *threads* independentes sejam direcionadas a diferentes unidades funcionais a cada ciclo (TULLSEN; EGGERS; LEVY, 1995), aproveitando de forma mais eficiente os recursos de emissão múltipla dos processadores modernos.

Figura A-1 - Duas abordagens diferentes para a utilização de *slots* de emissão de um processador escalar. A dimensão horizontal representa a capacidade de emissão de instruções em cada ciclo de *clock*. A dimensão vertical representa uma sequência de ciclos de *clock*. As cores pretas, vermelhas, azuis e cinza significam diferentes *threads* sendo executadas nos processadores. Figura extraída de (PATTERSON; HENNESSY, 1990).

Com este tipo de técnica, instruções de mais de uma *thread* podem estar sendo executadas em um *pipeline* qualquer em um dado momento. Isto é feito sem grandes mudanças na arquitetura do processador, sendo necessário basicamente: habilidade de emissão de instruções de múltiplas *threads* em um mesmo ciclo e um banco de registradores maior, a fim de armazenar dados de múltiplas *threads*.

O objetivo da técnica é o de aumentar substancialmente a utilização do processador em face das longas latências de memória e paralelismo limitado disponível em cada *thread* (TULLSEN; EGGERS; LEVY, 1995). Ainda segundo (TULLSEN; EGGERS; LEVY, 1995), o SMT combina a múltipla emissão por instruções (*multiple-issue-per-instruction*) dos processadores superescalares modernos com a habilidade de esconder a latência (*latency-hiding*) das arquiteturas *multithread*.

O grande motivador do SMT reside no fato dos processadores modernos de emissão múltipla frequentemente terem disponível um maior paralelismo das suas unidades funcionais do que aquele que com uma única *thread* poderia conter efetivamente (TULLSEN; EGGERS; LEVY, 1995). Além disso, com a mudança de nome dos registradores e o escalonamento dinâmico, várias instruções de *threads* independentes podem ser emitidas sem considerar as dependências entre elas, podendo a resolução das dependências ser tratada pelo recurso de programação dinâmica.

O SMT também herda inúmeros desafios de projeto como atingir grande largura de banda no banco de registradores, suportar grandes demandas de acesso à memória, atender a várias requisições de encaminhamento e agendar instruções para unidades funcionais de forma eficiente (TULLSEN; EGGERS; LEVY, 1995). As GPUs (*Graphics Processing Unit*) são exemplos extremos de SMTs ao lidar com mais de 1000 *threads* simultaneamente (BUCK, 2007).

B. CHIP MULTIPROCESSORS – CMPs

Segundo (OLUKOTUN et al., 1996), modelos de execução superescalar têm limitações quanto à sua amplitude devido à dificuldade de se implementar filas de emissão complexas e bancos de registradores multi-portas. Isso motiva a necessidade de uma microarquitetura descentralizada para manter o crescimento do desempenho dos microprocessadores. Da perspectiva das aplicações, a microarquitetura que melhor funciona depende da quantidade e das características do paralelismo nas aplicações.

(OLUKOTUN; HAMMOND, 2005) cita outros motivadores como redução de volume, aumentando o desempenho por unidade de volume, e redução de consumo, ao fazer com que todos os processadores dentro do mesmo *die* compartilhem a mesma conexão com o resto do sistema, reduzindo a quantidade de infraestrutura de comunicação de alta velocidade necessária.

Em sistemas nos quais a vazão de informações é mais importante do que a velocidade em si, ou seja, em que a latência deixa de ser um fator crítico, pode-se fazer uso de CMPs com *clocks* reduzidos no lugar de processadores únicos com *clocks* mais elevados. Exemplos disso são o processador Niagara (KONGETIRA; AINGARAN; OLUKOTUN, 2005) e as GPUs (BUCK, 2007). Dessa forma, cada requisição é processada mais lentamente, porém, em um sistema com diversas aplicações e diversos CMPs, é possível a execução simultânea de cada aplicação, garantindo uma vazão maior de informações. A redução no *clock* também implica menores tensões de alimentação, o que leva a uma economia de energia.

Figura B-2 - Diferentes implementações de CMPs. Figura extraída de (OLUKOTUN; HAMMOND, 2005).

C. INFRAESTRUTURA DO ADEA

O *cache controller* será responsável por calcular e armazenar a probabilidade de ocorrência à medida que a execução continua. Adicionalmente, para cada acesso a um endereço de *cache*, independente do fato de ser um *hit* ou *miss*, seu contador e o contador do *set* são atualizados. Todo acesso resultará em incrementos para uma entrada existente ou a criação de uma nova entrada, com seu contador sendo definido como 1.

O ADEA requer uma tabela de histórico (*history table*) de todos os endereços que estão no *set* como uma estrutura auxiliar. A tabela de histórico mantém atualizado o número de vezes que cada endereço foi referenciado, não importando se está no *cache* ou se foi recentemente retirado deste. Esta tabela de histórico é implementada como uma *lookup table* para rápidos resultados de busca e contém a probabilidade de cada endereço que está sendo utilizado, representada pela taxa entre o número de referências ao endereço e o número de referências ao *set*. Além disso, a tabela de histórico mantém um bit de controle adicional para indicar quais endereços tem sido recentemente referenciados e quando é necessário atualizá-la.

Além da tabela de histórico e do contador para o *cache set*, é necessária uma estrutura que aponte para as linhas de *cache* no *cache set* indicando a melhor candidata a ser substituída. Devido à sua constante mudança a cada acesso ao *cache*, foi empregada uma *register stream* que estará dentro do *cache controller*. O tamanho da *register stream* será proporcional ao número de *sets* do *cache* onde a política será aplicada.

Figura C-3 - ADEA e suas estruturas auxiliares. Figura extraída de (KOBAYASHI, 2010).

Durante a execução de programas, se a maioria das referências de *cache* forem para um determinado endereço, apenas o *cache set* correspondente terá seu contador atualizado. Na prática, se dois endereços diferentes apontarem para dois *cache sets* diferentes que tenham o mesmo número de referências, ambos poderão ter diferentes valores de probabilidade de serem utilizados novamente ou de serem reutilizados no futuro, uma vez que o número de referências para cada *cache set* que armazena uma linha é diferente.

Devido ao critério projetado para estimar a entropia discreta e a probabilidade, haverá situações durante a execução tais como todos os endereços no *cache* tendo a mesma probabilidade. Para imaginar esta situação particular, vamos assumir que um *cache set* com associatividade 4 (*4-way associative*) tenha armazenado quatro diferentes linhas de *cache* nos primeiros quatro acessos. Cada endereço terá 25% de chances de ser referenciado novamente de acordo com o critério para se estimar a probabilidade.

A fim de ilustrar a operação do ADEA, assume-se uma sequência arbitrária de endereços que foram referenciados por um programa: {A, B, A, A, B, C, D, E, B, B}. O algoritmo LRU vai gerenciar tais endereços assim como descrito na figura C-4-a. De forma diferente, o ADEA vai substituir linhas como mostrado na figura C-4-b. Ambos os algoritmos irão reter o último endereço B, enquanto o LRU irá substituir o endereço A e o ADEA irá retirar o endereço D. Este é um exemplo simples de como

o ADEA irá fornecer mais chances de privilegiar endereços que tenham recebido mais referências baseando-se na adaptação do conceito de Entropia da Informação para prever acessos futuros. Da mesma forma, poderá haver outras sequências em que o comportamento irá variar e diferentes resultados poderão ser observados.

Figura C-4 - Funcionamento de algoritmos de substituição de linhas de *cache*. Figura C-4-a: Funcionamento do algoritmo LRU. Figura C-4-b: Funcionamento do algoritmo ADEA. Figuras extraídas de (KOBAYASHI, 2010).

D. CURVAS DE COMPORTAMENTO DE ACESSO

a. CACHE L1 DE DADOS

Acessos ao *cache* L1 de dados:

Figura D-5 - Curvas de comportamento de acesso ao *cache* L1 de dados para os *benchmarks* AMMP, ART, FMA3D, EON, EQUAKE e MGRID.

Figura D-6 - Curvas de comportamento de acesso ao *cache* L1 de dados para os *benchmarks* TWOLF e VPR.

b. CACHE L1 DE INSTRUÇÕES

Acessos ao *cache* L1 de instruções:

Figura D-7 - Curvas de comportamento de acesso ao *cache* L1 de instruções para os *benchmarks* AMMP e ART.

Figura D-8 - Curvas de comportamento de acesso ao cache L1 de instruções para os benchmarks AMMP, ART, FMA3D, EON, EQUAKE, MGRID, TWOLF e VPR.

c. CACHE L2

Acessos ao *cache* L2:

Figura D-9 - Curvas de comportamento de acesso ao *cache* L2 para os *benchmarks* AMMP, ART, FMA3D, EON, EQUAKE e MGRID.

Figura D-10 - Curvas de comportamento de acesso ao *cache* L2 para os *benchmarks* TWOLF e VPR.

E. GRÁFICOS

Gráficos de IPC (*instructions per cycle*) para *caches* de 512 kB e 2048 kB.

Figura E-11 - IPC para um *cache* L2 de 512 kB. Resultados em valor absoluto.

Figura E-12 - IPC para um *cache* L2 de 2048 kB. Resultados em valor absoluto.

Figura E-13 - IPC para um *cache* L2 de 1024 kB e associatividade 16. Resultados em valor absoluto.

F. CONCEITOS BÁSICOS SOBRE CACHES

O tempo de acesso à memória é um dos principais pontos de gargalo no desempenho de aplicações, por isso, uma aplicação pode passar uma quantidade considerável de tempo esperando por dados. Este impacto negativo faz com que, ainda que haja uma atualização no *clock* do processador, a aplicação pode não obter os benefícios esperados, tendo seu desempenho limitado.

Uma maneira de contornar este problema é a inserção de um pequeno *buffer* de memória de alta velocidade entre o processador e a memória. Tal *buffer* é comumente chamado de memória *cache*, ou simplesmente *cache*.

Segundo (PATTERSON; HENNESSY, 1990), *cache* é o nome dado ao primeiro nível da hierarquia de memória encontrado assim que o endereço deixa a CPU. Quando a CPU encontra na memória *cache* um item de dados que foi solicitado, chamamos de acerto de *cache* (*cache hit*). Quando a CPU não encontra um item de dados de que precisa na memória *cache*, ocorre uma falha de *cache* (*cache miss*). Uma coleção de tamanho fixo de dados contendo a palavra solicitada, chamada bloco, é recuperada da memória principal e inserida no *cache*. A localidade temporal diz que provavelmente esta palavra será utilizada novamente no futuro próximo, portanto é útil inseri-la no *cache*, onde ela pode ser acessada com mais rapidez. Devido à localidade espacial, há uma alta probabilidade de que os outros dados no bloco sejam necessários em breve. Considerando-se que o princípio da localidade é aplicável a muitos níveis e que tirar proveito da localidade para melhorar o desempenho é uma solução popular, o termo *cache* é aplicado sempre que se emprega a “bufferização” com o objetivo de reutilizar itens que ocorrem comumente. Exemplos incluem *caches* de arquivos, *caches* de nomes e assim por diante.

O tempo necessário para o *cache miss* depende da latência e da largura de banda da memória. A latência determina o tempo para recuperar a primeira palavra do bloco, a largura de banda por sua vez determina o tempo para se recuperar o restante desse bloco. Um *cache miss* é tratado por *hardware* e faz os processadores com execução em ordem (*in-order*) efetuarem uma pausa (*stall*), até que os dados fiquem disponíveis.

De modo semelhante, nem todos os objetos referidos por um programa precisam residir na memória principal. Se o computador possuir memória virtual, alguns objetos poderão residir em disco. Normalmente, o espaço de endereços é

dividido em blocos de tamanho fixo chamado páginas. Em qualquer instante, cada página reside na memória principal ou em disco. Quando a CPU se refere a um item dentro de uma página que não está presente no *cache* ou na memória principal, ocorre uma falha de página (*page miss*), e a página inteira é movida do disco para a memória principal. Tendo em vista que as falhas de página demoram muito, elas são tratadas em *software*, e a CPU não sofre uma parada. Em geral, a CPU passa para alguma outra tarefa enquanto ocorre o acesso ao disco. A memória *cache* e a memória principal mantêm o mesmo relacionamento que existe entre a memória principal e o disco (PATTERSON; HENNESSY, 1990). Este problema é equivalente ao gerenciamento de memória e disco em sistemas operacionais.

A figura mostra uma organização com múltiplos níveis de *cache*. Destaca-se na figura que, da esquerda para a direita, o tamanho dos *caches* aumenta e a velocidade destes diminui.

Figura F-14: Organização com múltiplos níveis de *cache*. Da esquerda para a direita, distanciando-se do processador, as conexões entre cada nível se tornam mais lentas e a largura de banda menor; por outro lado, o tamanho dos *caches* torna-se maior, bem como o tamanho da memória principal.

Nos microprocessadores atuais, os *caches* não só variam em tamanho mas também na sua organização interna, ou seja, nas funcionalidades específicas para os quais podem ser utilizados dentro do processador. As organizações mais populares são:

- *Cache* de instruções: utilizado para o armazenamento de instruções, reduzindo o custo de ir até a memória buscar novas instruções para serem executadas. Informações sobre predição de desvios (*branch prediction*) também podem ser armazenadas no *cache* de instruções;

- *Cache* de dados: contém os dados da aplicação para utilização pelo processador. Os dados são carregados da memória para o *cache* e posteriormente para os registradores para que a instrução com esses valores possa operar nesses dados. O conteúdo destes registradores é então copiado novamente para o *cache* de dados e eventualmente para a memória principal.
- TLB (*Translation Lookaside Buffer*): possui entradas que mapeiam endereços virtuais em endereços físicos. Essa tradução possui um custo muito elevado, por isso é vantajoso armazenar este mapeamento próximo ao processador.