Arquitetura de Computadores 2
Trabalho 2 - Hierarquia de Memória

Thiago Reis Porto e Mathaus Huber

Configuração da hierarquia:

Processador: i7 2,93 GHz
Número de Registradores: 16 por núcleo do processador(4x16)
Número de níveis na hierarquia de memória: 4
Número de níveis de cache: 4
Modo de endereçamento: Byte
Tamanho de Palavra: 32 bits
Tamanho da Memória Principal: 8GB
Método de Gravação: Write-Back

1 Introdução

Neste trabalho será utilizada a arquitetura de CPU baseada em um i7 da Intel, com uma visão “moderna” de hierarquia de memória, modelada com objetivo de economia de energia e alto desempenho. Com análises baseadas em resultados do simulador SimpleScalar. A hierarquia possui 4 níveis de cache L1, L2, e L3 no chip. Uma L4 externa compartilhada com a GPU. Porém, por questão de simplicidade, para as análises serão considerados apenas um core. Além disso, o SimpleScalar simula até dois níveis de Cache, o que limita a qualidade das análises neste trabalho. Para realização das simulações foi utilizado o benchmark do compilador GCC-1 na ferramenta Sim-cache disponível no SimpleScalar.

Memórias:

1. 16 registradores por núcleo (64 total), cada registrador é um conjunto de Flip-flops para armazenamento temporário para uso da CPU.
2. Quatro níveis de memórias cache, 3 internas no chip do processador(SRAM) e uma externa fora do chip que pode ser compartilhada entre CPU e GPU(DRAM).
3. Memória Principal(DRAM).
4. Armazenamento em massa requer memórias não voláteis, esta arquitetura sugere uma memória flash SSD.

2 Processador

Baseado na linha iX de processadores Core da Intel. Com 2,93 GHz de frequência, 4 núcleos físicos e 16 registradores por núcleo. Cada núcleo também possui suas próprias caches L1 e L2, com uma única L3 compartilhada. Também, em nossa arquitetura há uma cache L4 externa compartilhada com a GPU.

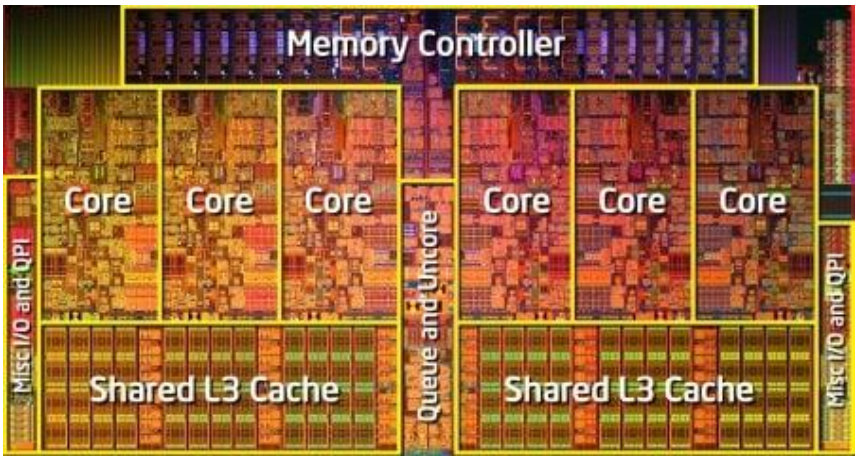

Figura 1: exemplo de mapa do die de uma arquitetura i7.
Fonte:https://adrenaline.com.br/files/upload/reviews/2010/core_i7_980x/diemap.jpg

	Core i7 940	Core i7 950	Core i7 965 XE	Core i7 975 XE	Core i7 980X
Geração	Bloomfield	Bloomfield	Bloomfield	Bloomfield	Gulftown
Socket	LGA1366	LGA1366	LGA1366	LGA1366	LGA1366
Litografia	45nm	45nm	45nm	45nm	32nm
Área Die	263mm²	263mm²	263mm²	263mm²	248mm²
Quantidade Transistores	731 Million	731 Million	731 Million	731 Million	1.17 Billion
Qtde. Núcleos Físicos	4	4	4	4	6
Qtde. Threads	8	8	8	8	12
Clock	2.93Ghz	3.06Ghz	3.20Ghz	3.33Ghz	3.33Ghz
Turbo Boost	Até 3.20Ghz	Até 3.33Ghz	Até 3.46Ghz	Até 3.60Ghz	Até 3.60Ghz
Multiplicador CPU	22X	23X	24X	25X	25X
Clock Base	133 Mhz	133 Mhz	133 Mhz	133 Mhz	133 Mhz
Clock Uncore	2133 Mhz	2133 Mhz	2666 Mhz	2666 Mhz	2666 Mhz
Memória	Triple-Channel DDR3-1066	Triple-Channel DDR3-1066	Triple-Channel DDR3-1066	Triple-Channel DDR3-1066	Triple-Channel DDR3-1066
QPI	4.8 GT/s	4.8 GT/s	6.4 GT/s	6.4 GT/s	6.4 GT/s
Cache L1	4 x 32KB Data + 4 x 32KB Instruction	4 x 32KB Data + 4 x 32KB Instruction	4 x 32KB Data + 4 x 32KB Instruction	4 x 32KB Data + 4 x 32KB Instruction	6 x 32KB Data + 6 x 32KB Instruction
Cache L2	4 x 256KB	4 x 256KB	4 x 256KB	4 x 256KB	6 x 256KB
Cache L3	8MB Shared	8MB Shared	8MB Shared	8MB Shared	12MB Shared
Conjunto de Instruções (ISA)	MMX, EMT64, SSE-SSSE4.2, VT-x	MMX, EMT64, SSE-SSSE4.2, VT-x	MMX, EMT64, SSE-SSSE4.2, VT-x	MMX, EMT64, SSE-SSSE4.2, VT-x	MMX, EMT64, SSE-SSSE4.2, VT-x, AES
TDP	130W	130W	130W	130W	130W
Preço de Lançamento	$284 USD	$284 USD	$562 USD	$999 USD	$999 USD

Figura 2: tabela com as diferenças de algumas arquiteturas i7.

Fonte:https://adrenaline.com.br/files/upload/reviews/2010/core_i7_980x/tabela.jpg

3 Cache

3.1 Tamanhos

É importante frisar, que a quantidade de memória cache é um fator que impacta diretamente na performance geral justamente por diminuir a quantidade de requisições da CPU para a memória RAM. Quanto mais espaço tiver no cache do processador, menores serão as chances de ele ter de solicitar algum dado da RAM.

Foram selecionados tamanhos de cache similares aos das arquiteturas reais. Através de simulação no Sim-cache, é possível ver na figura 3 que como esperado o aumento de tamanho da cache diminui as faltas, porém isso aumenta o custo de acesso/hardware, área e manutenção da mesma. Como as memórias mais próximas da CPU devem ser as mais rápidas e de fácil manutenção, elas possuem tamanhos menores que as de níveis mais distantes. Com a cache externa L4 sendo a maior, e mais distante do processador. Esta cache é compartilhada com a GPU.

L1: 2 x 32kB(Uma para instruções e outra para dados, por núcleo)
L2: 1 x 256kB(por núcleo)
L3: 1 x 8MB
L4: 1 x 128MB

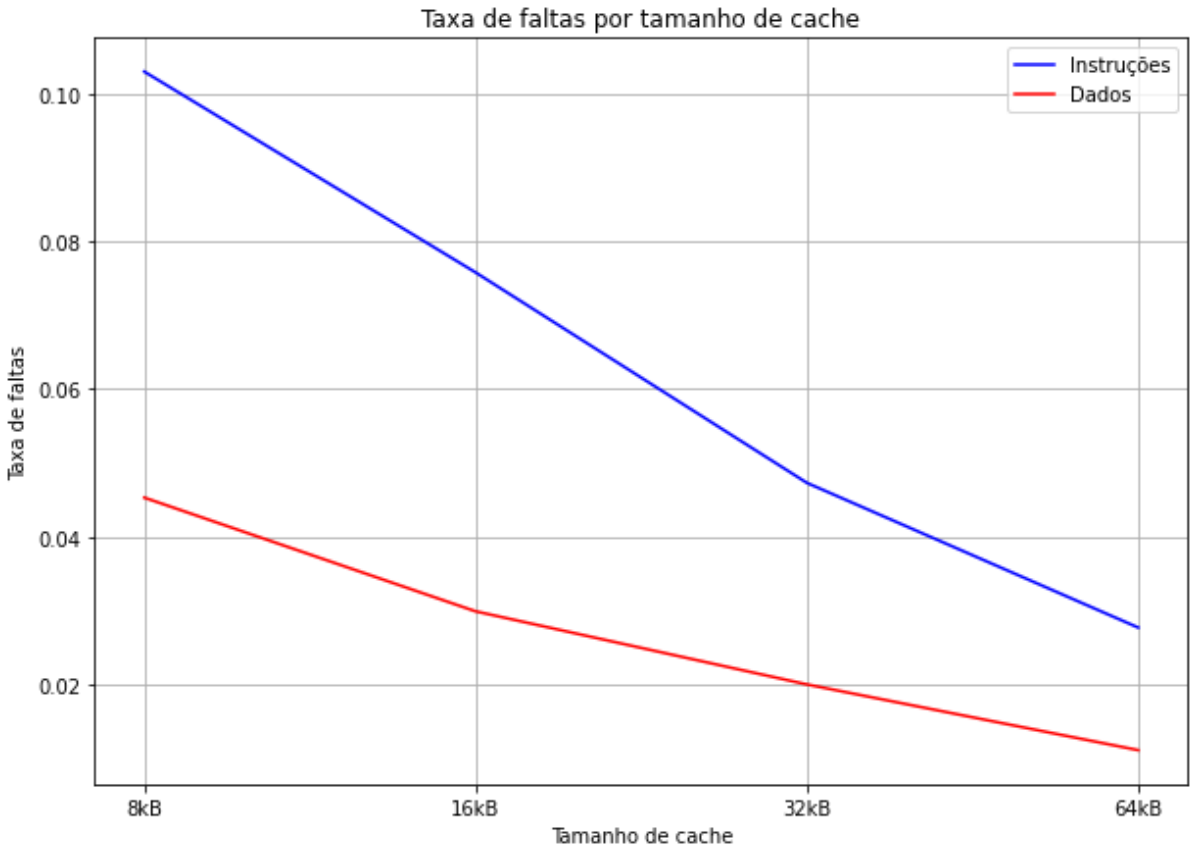

Figura 3: taxa de faltas comparando tamanhos de cache. Com 45079001 instruções executadas, 16725083 loads/stores. Com tamanho de bloco de 32 bits e mapeamento direto.

3.2 Caches Separadas vs Unificadas

L1 é o único nível que será separado entre instruções e dados(abordagem Harvard), o motivo para isso é o fato de ser o nível mais próximo da CPU e por possuir tamanho menor. Visando reduzir a competição entre dados e instruções neste nível. Facilitando especialmente o prefetching de instruções(mas também de dados se possível). O restante dos níveis são consideravelmente maiores e utilizam caches unificadas(Von Neumann). Visto que, em geral, caches unificadas reduzem o número total de faltas, e podem aproveitar de maneira mais eficiente o tamanho da cache, compartilhando o espaço com dados e instruções.

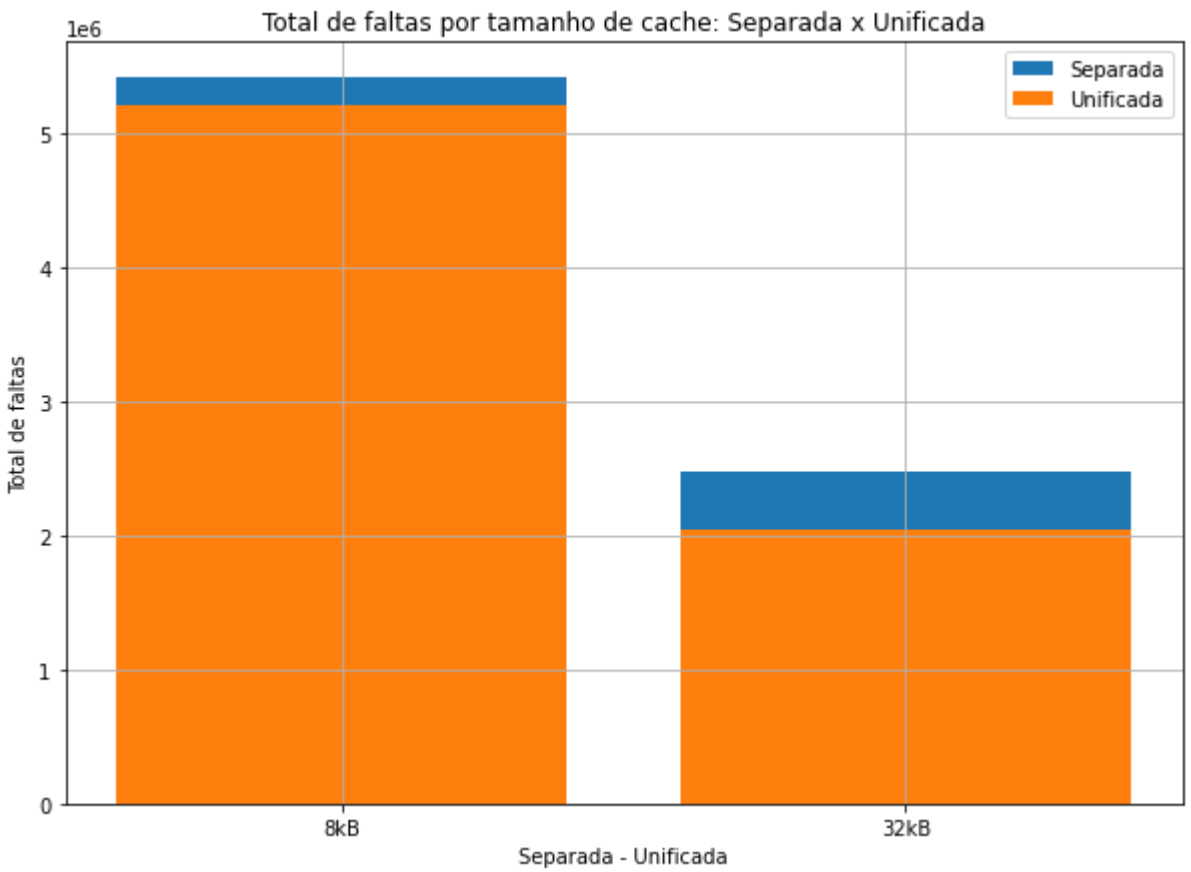

Figura 4: comparação de total de faltas com tamanhos de cache em 8kB e 32kB. Com 45079001 instruções executadas, 16725083 loads/stores. Com tamanho de bloco de 32 bits e mapeamento direto.

3.4 Associatividade

Mapeamento de cache, se refere ao modo de como e onde serão armazenados os blocos de memória. Onde o mapeamento geralmente utiliza a equação: “endereço de bloco MOD número de linhas da cache”, e podem ser divididos em Direto, conjunto-associativo e totalmente associativo. No mapeamento Direto os blocos só podem ser mapeados para uma única linha da cache, pode ser interpretado como cada linha sendo um conjunto. Em conjunto-associativo cada linha possui N conjuntos, diminuindo o número de conflitos quando blocos são mapeados para a mesma linha da cache. Em totalmente associativa é como se a cache possuísse apenas um conjunto, os blocos podem ser mapeados para qualquer linha da cache.

Quando menor associatividade, requer menos bits de controle e é mais fácil de encontrar os blocos mapeados. Isso, porém, aumenta o número de blocos mapeados para os mesmos conjuntos. Aumentando a associatividade reduz os conflitos por mapeamento, em troca de um maior gasto para controle e busca dos blocos mapeados(e.g. a tag que identifica o bloco aumenta com associatividade). Os extremos geralmente devem ser evitados enquanto mapeamento direto gera muitos conflitos, totalmente associativo extrapola o gasto de hardware e de energia. O ideal é encontrar um equilíbrio que evite o “overhead” de custo, e também reduza o número de conflitos. Conforme a simulação da Figura 5, o mapeamento com 4 conjuntos mostrou bom resultado e mostra que em algum momento diminuir os conjuntos não reduz o número de conflitos para instruções no GCC-1. Poderíamos testar mapeamentos-associativos com mais de conjuntos, sem aumentar tanto o custo como uma 8-Way. Mas optamos pelo mapeamento 4-Way que já possui bom balanceamento custo/desempenho.

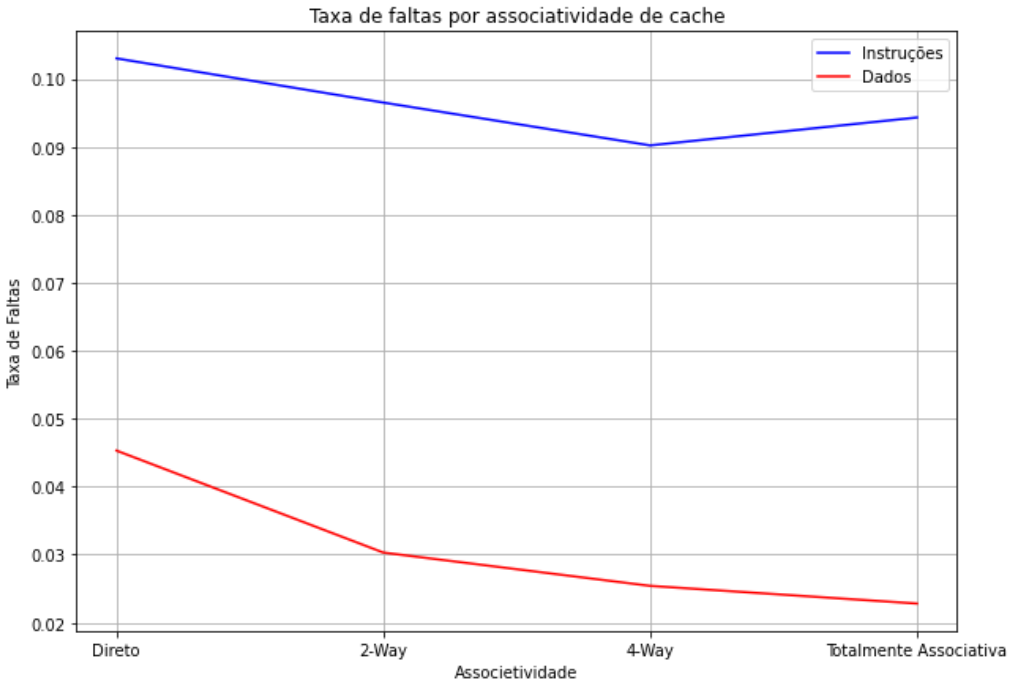

Figura 5: comparação de taxa de faltas com tamanho de cache de 8kB, considerando o modo de mapeamento. Com 45079001 instruções executadas, 16725083 loads/stores. Com tamanho de bloco de 32 bits.

3.5 Método de escrita na cache

Os principais métodos de gravação são o Write-Through(WT) e Write-Back(WB). WT mantém a hierarquia consistente, pois os dados são sempre escritos na memória principal. WB irá escrever em um nível inferior de memória apenas quando o dado for substituído no nível atual. Optamos por WB, pois apesar de possuir uma implementação mais complicada, seu desempenho e custo energético é mais eficiente, em geral, que com o WT. O que acontece no WB é justamente que, quando os dados são atualizados, eles são gravados apenas no cache. Os dados

modificados são gravados no armazenamento de backend apenas quando os dados são removidos do cache. Este modo tem alta velocidade de gravação de dados, mas os dados serão perdidos se ocorrer uma falha de energia antes que os dados atualizados sejam gravados no armazenamento. Portanto, a única vantagem que teríamos ao utilizarmos, ao invés disso, o WT é que ele torna a implementação extremamente simples e nenhum protocolo de coerência de cache complicado seria necessário. Na prática, um processador moderno com múltiplos núcleos como o proposto neste trabalho, utilizaria implementações mais complexas que um simples WT ou WB. Combinando alguns elementos de ambos e geralmente recorrendo ao uso de múltiplos buffers, de modo que o desempenho e custo em energia seja eficiente.

3.6 Política de Substituição

A política de substituição, se refere em como a cache decide qual bloco deve substituir quando um bloco precisa ser inserido em uma linha cheia. No mapeamento direto é irrelevante, pois os blocos sempre são mapeados para o mesmo conjunto então. Em caso de conflito simplesmente ocorreu uma substituição, mas em mapeamento com maior associatividade é necessário um método para substituição. LRU(Least Recently Used) substitui os blocos segundo o comportamento temporal, buscando retirar os blocos menos utilizados e há mais tempo na cache. Esse método é bastante eficiente e possui boa predição na substituição, mas possui um alto custo e complicada implementação. É necessário bits extras para determinar qual bloco deve ser substituído. Outras alternativas mais baratas podem ser boas opções, entre elas estão: Random e FIFO. Em Random é utilizado um gerador de números pseudo-aleatórios, e sorteia o bloco que deve ser substituído. Não possui um custo tão alto de implementação e seu comportamento estocástico funciona bem em caches grandes. FIFO(First In First Ou) possui o comportamento de fila, que substitui sempre o primeiro bloco inserido na linha. Dependendo da aplicação pode funcionar muito bem, mas, em geral, embora seja mais barato possui pior desempenho em comparação ao LRU e Random, Na figura 6 mostra uma simulação comparando os métodos, onde LRU e Random aparecem com os melhores números. Optamos por LRU para os níveis L1 e L2, e Random para L3 e L4, considerando o tamanho destas caches e buscando novamente um equilíbrio custo/desempenho.

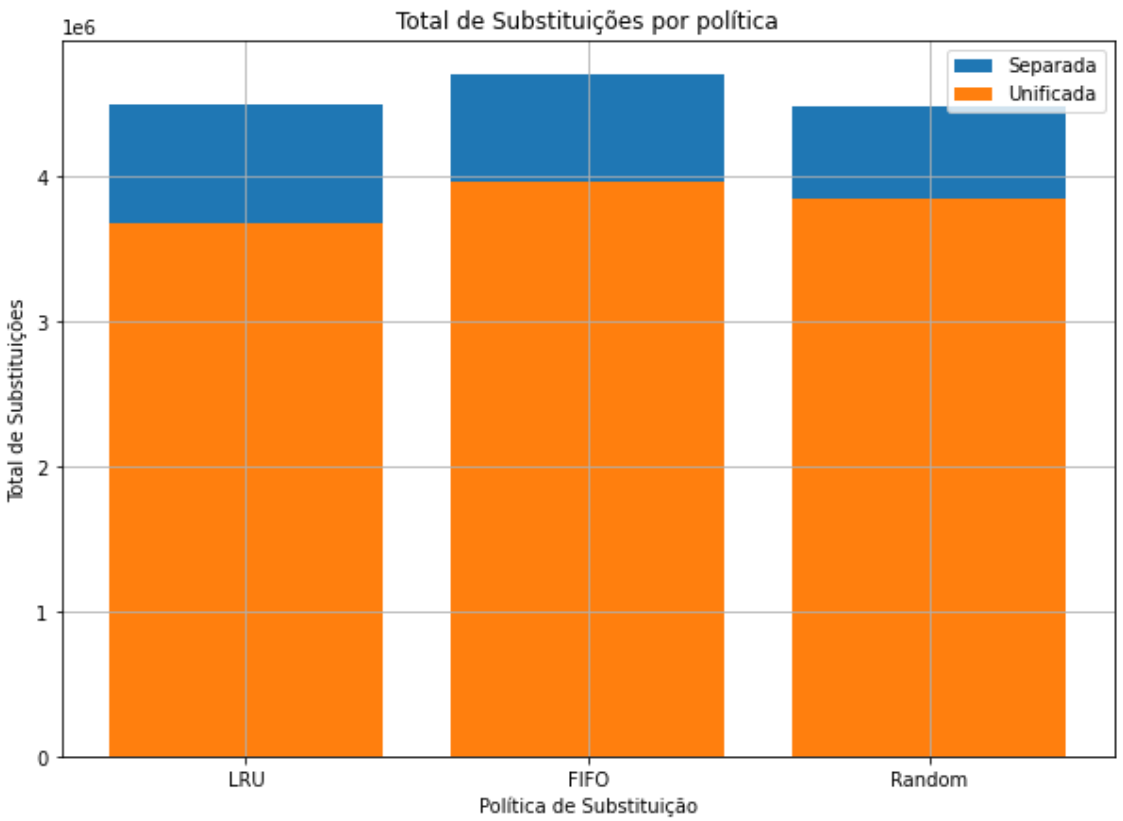

Figura 6: comparação do total de substituições com tamanho de cache de 16Kb Unificada e Separada de 8Kb somando total das instruções e dados, considerando a política de substituição. Com 45079001 instruções executadas, 16725083 loads/stores. Com tamanho de bloco de 32 bits e mapeamento 4-Way.

3.7 Tamanhos de Bloco

O tamanho de bloco da cache tem uma relação com a exploração espacial do conteúdo de memória. De modo trivial, conforme o tamanho do bloco aumenta, mais informações são carregadas pelo bloco, de modo que pode evitar faltas. Por exemplo, se a CPU precisar de um dado próximo localmente de outro dado de memória, que já foi carregado na cache anteriormente. Quando estes dados estão no mesmo bloco, ocorre uma falta, invés de duas, e isto pode ser melhor aproveitado conforme o tamanho do bloco cresce. Então aumentando o tamanho do bloco pode reduzir o número de faltas, consequentemente melhorando o desempenho. Entretanto, aumentar o tamanho do bloco sem aumentar o tamanho da capacidade da cache, implica em uma redução das linhas de cache. Eventualmente se o número das linhas for muito baixo acaba em gerar muitos conflitos, pois muitos blocos são mapeados para as mesmas linhas. Essa diminuição das linhas também reduz a exploração temporal, então o tamanho do bloco é de extrema importância para o desempenho da cache. Também é importante considerar como as instruções e dados se comportam em relação ao tamanho da cache. As instruções, pelo motivo de possuírem uma natureza sequencial, e dos algoritmos normalmente conterem muitos laços de repetição,

costumam aproveitar muito bem a exploração espacial dos blocos de cache. Isso não se mantém da mesma forma para os dados, o que é demonstrado pela figura 7. Entretanto, até mesmo os dados podem aproveitar melhor o tamanho do bloco de cache, pois durante o desenvolvimento do software, blocos de cache podem ser considerados nas estruturas de dados. Por exemplo, através de alinhamento de variáveis, e otimizações como SoA(Struct of Arrays) onde estruturas mantêm arrays dos dados internos, ao contrário da comum AoS(Array of Struct). AoS é muito utilizado em Orientação a Objetos onde uma classe possui vários atributos e métodos singulares a uma instância. Quando há necessidade de vários objetos de uma classe, é normalmente criado um conjunto(array) de instâncias do objeto. Se for necessariíssimo atualizar apenas um atributo particular de várias destas instâncias, irá gerar várias faltas na cache. Em SoA como os dados estão agrupados na mesma estrutura, potencialmente ocorre uma falta na cache durante a atualização dos valores. Algumas otimizações simples são geralmente feitas pelo próprio compilador, mas de fato boas otimizações dependem muito mais do programador. Esse tipo de abordagem é fundamental para aplicações de alto desempenho e em tempo real, como em desenvolvimento de jogos onde essas otimizações têm sido chamada de: Design Orientado a Dados. Com tudo isso a considerar, somado a simulação da figura 7, optamos por um tamanho de bloco razoável para as caches(tanto para dados, quanto instruções), utilizando uma 4N(quatro palavras por bloco).

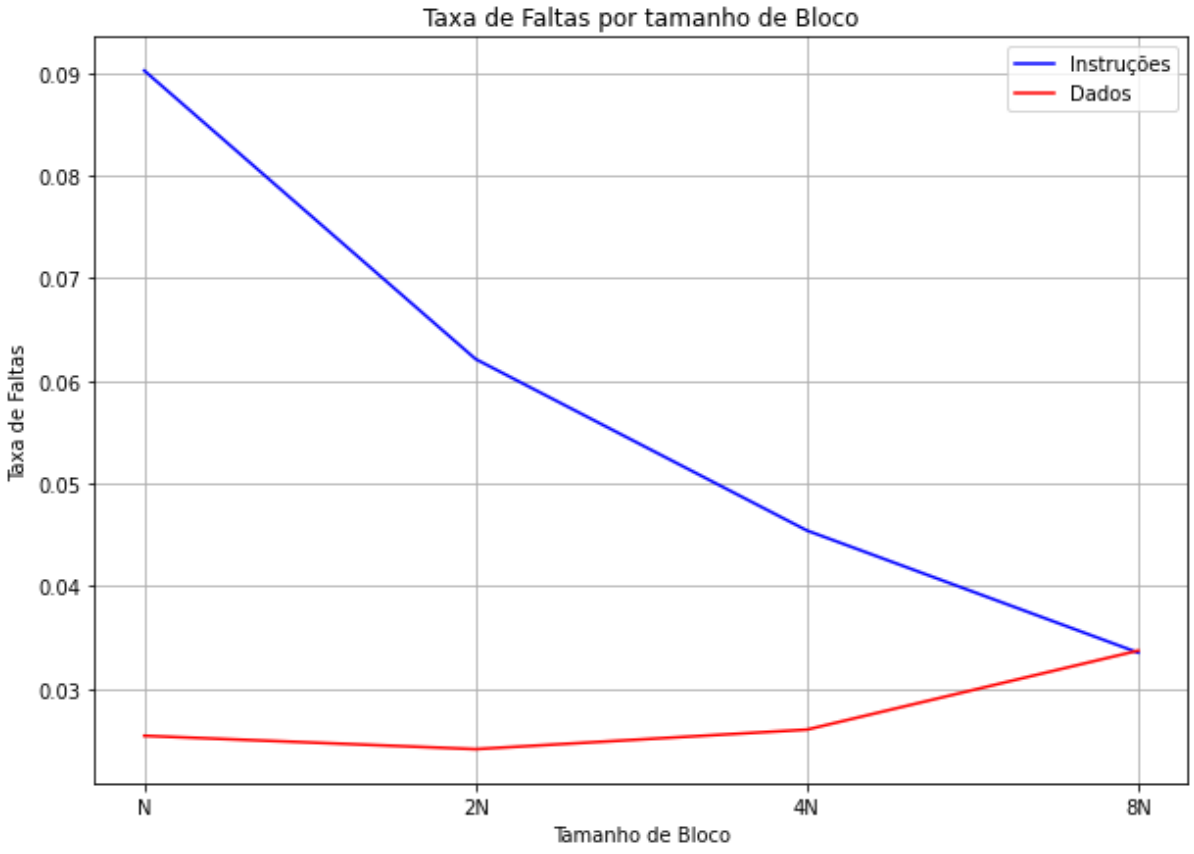

Figura 7: comparação de taxa de faltas com relação ao tamanho do bloco em uma cache de 8kB com mapeamento 4-Way, onde N é o tamanho da palavra de memória. Com 45079001 instruções executadas, 16725083 loads/stores.

```
class Circulo {
    Ponto pos;
    Cor cor;
    double raio;
};
vector<Circulo> circulos;
```

```
class Circulos {
   vector<Ponto> posicoes;
   vector<Cor> cores;
   vector<double> raios;
};
```

Figura 8: AoS(esquerda) e SoA(direita).

3.8 Desempenho da cache em níveis

Devido às limitações do SimpleScalar as análises serão realizadas em pares, assim como alguns valores são assumidos como tempo de acesso e CPI.

Processador:

Frequência: 2.93 GHz = 0.34 ns(p)
CPI = 1
Penalidade de Falta (Ciclos):10

L1(Para facilitar o cálculo, foi unificada na simulação):

Instruções 32kB, Dados 32kB / 64kB
Tempo de Acesso: 10 ns (SRAM)
Taxa de Faltas: 0.0114

L2:

Tamanho: 256kB
Tempo de Acesso: 10 ns
Taxa de Faltas: 0.0109

L3:

Tamanho: 8MB
Tempo de Acesso: 50 ns (DRAM)
Taxa de Faltas: 0.0927

L4:

Tamanho: 128MB
Tempo de Acesso: 50 ns (DRAM)
Taxa de Faltas: 0.0801

Para melhorar o desempenho seria necessário: reduzir a taxa de faltas; reduzir a penalidade nas faltas; reduzir o tempo de acerto. Existem algumas formas de

reduzir a taxa de faltas, tentaremos elucidar algumas formas para melhorar a taxa de falta na cache. Começando pela busca antecipada por hardware, que é uma estratégia que necessita de sistema de memória com muita vazão sobrando e que possa ser utilizada sem aumentar a penalidade, uma das desvantagens de se utilizar essa estratégia é que pode causar poluição da cache, sobrando blocos/palavras inúteis. Também pode ser feita, uma busca antecipada por software, onde os dados são buscados antecipadamente, é o caso que acontece no MIPS-4 onde ele carrega dados antecipadamente para cache, esse tipo de estratégia também causa poluição na cache e executar instruções de busca antecipada custa tempo, por isso, deve ser levado em consideração se o custo da busca antecipada é menor que os ganhos pelos acertos, para termos uma estratégia vantajosa. Outra forma é a otimização por compilador, onde foi abordado no tópico 3.7, a utilização de SoA e AoS.

De forma a reduzir a penalidade nas faltas, poderíamos melhorar a hierarquia das caches, tentando melhorar a relação entre associatividade, tamanho e inclusão, o que já foi proposto dentro do escopo do projeto, onde almejamos diminuir ao máximo a penalidade nas faltas.

Para reduzir o tempo de acerto, seria necessário utilizar caches simples e pequenas, não traduzindo o endereço virtual para o físico, e utilizarmos escritas em pipeline.

4 Armazenamento em Massa

O intuito da nossa hierarquia de memória é justamente fornecer um maior desempenho, pensando dessa forma, optamos pela utilização de memórias flash, em específico um SSD modelo A400 de 480GB, que é dez vezes mais rápido do que um disco rígido tradicional. Algumas das vantagens de se utilizar esse tipo de memória é justamente o baixo consumo de energia, a ocupação mínima de espaço, alta resistência, durabilidade e segurança. A tendência é que, no futuro, fabricantes de computadores substituam os discos rígidos por unidades flash, o que ainda não é viável do ponto de vista de retorno lucrativo, pois a sua fabricação se tornaria cara para as empresas.

5 Tecnologias de Memória

Iremos mostrar algumas das tecnologias de memórias existentes nos computadores modernos, suas aplicações, vantagens e desvantagens, porém, almejando focar sempre nas tecnologias usadas dentro do escopo do projeto.

Existem uma gama de tecnologias de memória disponíveis hoje em dia, porém, todas elas são basicamente divididas em dois tipos: Voláteis e Não voláteis.

Do tipo de memórias não voláteis, temos: ROM, PROM, EPROM, EEPROM, Flash RAM; Por outro lado, abordando as memórias voláteis, temos: RAM estática e RAM dinâmica.

Em relação às memórias não-voláteis, caracterizam-se por não perder os dados gravados ao cessar-se a alimentação elétrica. A exceção é a memória Flash, basicamente utilizada dentro do escopo do nosso projeto, que é na verdade constantemente alimentada por uma bateria, por isso sendo considerada não-volátil. Como já dito anteriormente, optamos pelo uso de um SSD, ou seja, uma memória flash, porque, embora seja mais cara, traz um desempenho maior.

Quando se trata de aplicação desse tipo de memória, temos que, a memória flash é largamente usada para conter códigos de controle ou sistemas operacionais dedicados, como a BIOS de um computador, o programa de controle de um telefone celular, câmeras digitais, receptores de satélite domésticos, controladores embutidos, adaptadores de vídeo e outros dispositivos. O uso da memória Flash nestes dispositivos permite a atualização dos programas de controle de maneira fácil, o que traz extrema vantagem de custo e tecnologia.

Já, quando se trata de memórias voláteis são as que requerem energia para manter a informação armazenada, são fabricadas em duas tecnologias: dinâmica e estática. A RAM estática ou SRAM são muito mais rápidas que as RAM's dinâmicas, e por isto mesmo mais caras, elas são usadas na memória cache de um computador, as desvantagens justamente estão relacionadas ao custo, por isso deve ser levado em conta o objetivo final, se é ter desempenho ou economia. No caso da RAM dinâmica, elas são o tipo mais comum de RAM para uso em computadores pessoais ou de maior porte. A DRAM é dinâmica porque, ao contrário da SRAM, precisa restabelecer o conteúdo de suas células de armazenamento periodicamente. Devido a limitações de custo, todos os computadores, exceto aqueles topo de linha, têm utilizado memória DRAM como principal, sua vantagem, sendo o oposto da RAM estática, está no custo, sendo obviamente, a sua maior desvantagem o seu desempenho.

6 Descrição da Hierarquia

A base da pirâmide representa dispositivos de armazenamento de massa, de baixo custo por byte, no nosso caso, como é a hierarquia de uma memória hipotética, não nos preocupamos com o custo monetário e em hardware, e optamos por colocar uma memória mais rápida, visando trazer mais desempenho.

No topo, considerada as mais rápidas, temos os registradores, que são memórias individuais agrupadas para armazenar uma palavra em binário. São de alto desempenho, usadas pela CPU ou por outros dispositivos para armazenamento temporário de dados. Alguns registradores da CPU participam do processamento e

podem ser acessados por programas, o que torna o tipo de memória mais rápido e consequentemente o mais caro.

Logo após, temos a nossa memória cache, onde o único core do nosso processador possui caches L1 e L2, e uma L3 compartilhada, utilizando também uma L4 externa ao chip, entre a CPU e a memória principal. O que caracteriza essas memórias é a velocidade, pois são memórias muito rápidas, com capacidade de alguns KB a poucos MB.

Quase na base da pirâmide temos a memória RAM, que é responsável pelo armazenamento de informações necessárias para a execução de aplicativos em uso e para o funcionamento do próprio sistema operacional. Esse tipo de memória, inclusive, facilita o trabalho do processador que pode acessar os dados essenciais mais rapidamente.

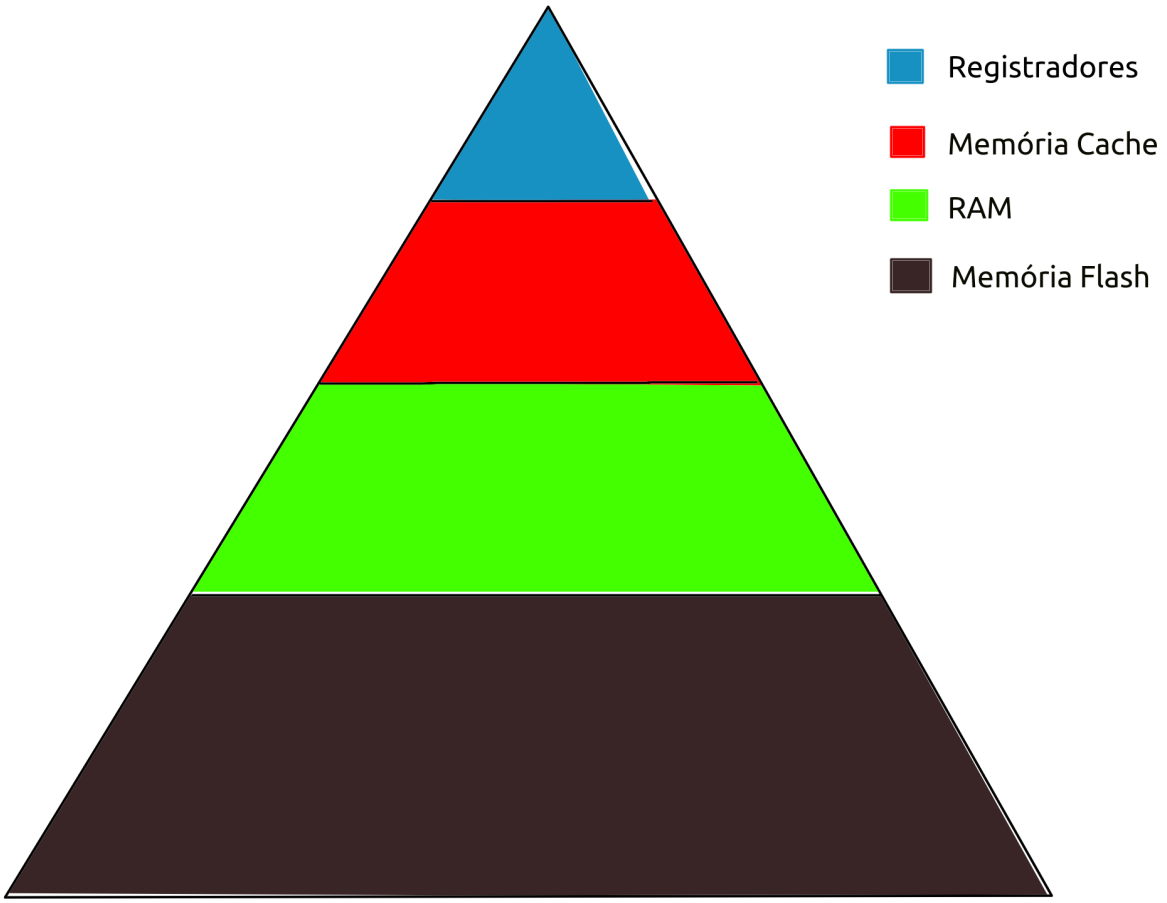

Figura 9: Pirâmide da hierarquia de memória descrita no projeto.

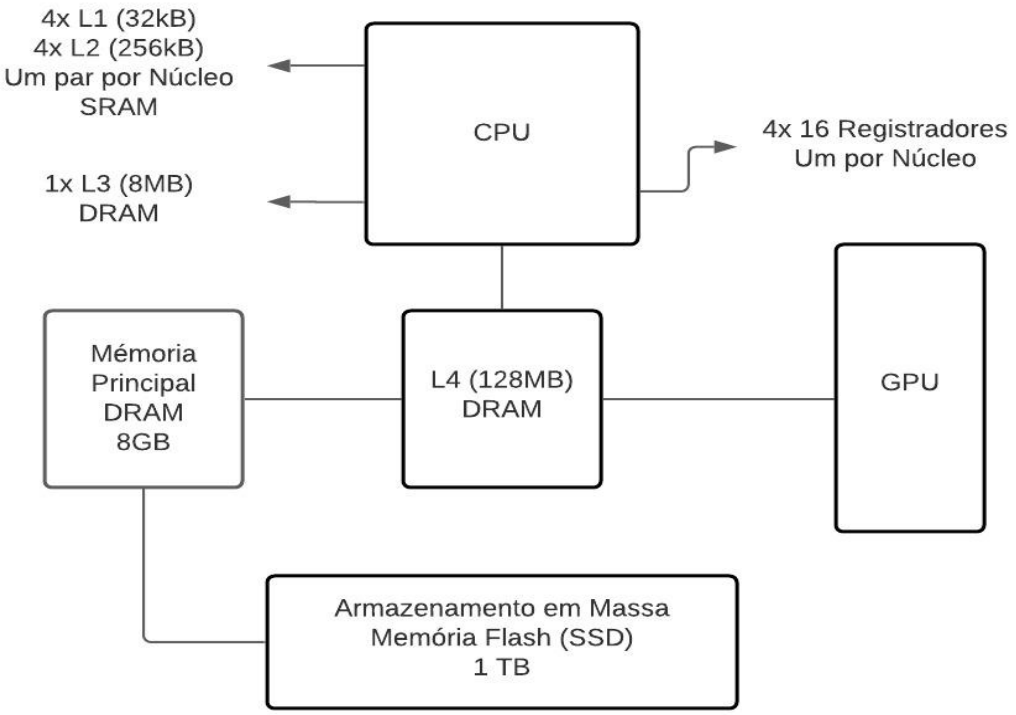

Figura 10: Diagrama da hierarquia de memória descrita no projeto.

7 Conclusão

Buscamos trazer a referência de um artigo exploratório de pesquisa, para a implementação de uma hierarquia de memória, justificando todas as opções escolhidas dentro do escopo de tecnologias viáveis. Além disso, almejamos elucidar, a cada tópico proposto dentro do escopo do trabalho, a justificativa de cada técnica e abordagem utilizada na implementação dessa hierarquia de memória e definição de uma cache específica para o melhor desempenho.

Visando a ideia principal do trabalho, que buscava justamente abordar os conteúdos passados durante a disciplina até o presente momento, tentamos trazer um questionamento mais atual das arquiteturas de computadores, mostrando uma hierarquia de memória voltada para processadores modernos, considerando, em específico, os processadores iX da intel.

8 Referências de Consulta

PATTERSON, D.; HENESSY, J. L.; **Organização e projeto de computadores: a interface hardware/software.** 3ª Edição. São Paulo: Elsevier, 2005, 484 p.

STALLINGS, W. **Arquitetura e Organização de Computadores.** 8ª edição. São Paulo: Pearson, 2010. 640 p.

TANENBAUM, A. S. **Organização Estruturada de Computadores.** 5ª edição. São Paulo: Pearson, 2007. 464 p.

MURDOCCA, M. J.; HEURING, V. P. **Introdução à Arquitetura de Computadores.** Rio de Janeiro: Campus, 2000. 512p.

LOURENÇO, A. C.; CRUZ, E. C.; FERREIRA, S.; JÚNIOR, S. **Circuitos Digitais.** 3ed. São Paulo: Érica, 1999. 321p.

DE ROSE, C. A. F.; NAVAUX, P. O. **Arquiteturas Paralelas.** Porto Alegre: Sagra Luzzatto, 2003. 152p.

MONTEIRO, M. A. **Introdução à Organização de Computadores**. 5ª edição. Rio de Janeiro: LTC, 2007. 720 p.

S. Tavarageri and P. Sadayappan, **"A Compiler Analysis to Determine Useful Cache Size for Energy Efficiency,"** 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, 2013, pp. 923-930, doi: 10.1109/IPDPSW.2013.268.