UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Análise de Desempenho em Memórias Cache

Marcelo Pinheiro Montanher - 7236527

Pedro Peçanha Martins Góes - 7547138

1. Introdução ao Simulador Escolhido

O simulador escolhido para a realização dos experimentos foi o Amnésia, desenvolvido no próprio ICMC como ferramenta de aprendizado. A entrada do programa consiste em dois arquivos diferentes: um documento .xml que define os atributos do processador e arquitetura de memória que será utilizada e um arquivo texto com a sequência de acessos à memória a serem simulados.

No arquivo de definição da arquitetura da memória é possível definir todos os aspectos relevantes para a execução deste trabalho. Para o processador deve-se configurar o tamanho da palavra em bytes. Para cada nível da memória cache deve-se configurar o seu tipo (unificada ou separada), seu tamanho em palavras, o número de palavras por bloco, o número de blocos por conjunto (i.e. a associatividade), o número de ciclos para cada acesso de leitura, o número de ciclos para cada acesso de escrita, o tempo de cada ciclo, a política de substituição (FIFO ou LRU) e a política de atualização (write-through ou write-back). Para a memória principal deve-se configurar seu tamanho em palavras, o número de palavras por bloco, o número de ciclos para cada acesso de leitura, o número de ciclos para cada acesso de escrita e o tempo de cada ciclo.

No arquivo de trace, temos as diferentes sequências de acesso que serão realizadas na memória. Usamos para isso uma sequência de caracteres com dois elementos: o primeiro indica o tipo de operação que será realizada na memória (0 para leitura, 1 para gravação, 2 para busca de instrução, 3 para registro de acesso desconhecido e 4 para operação de cache flush) e o segundo indica o endereço hexadecimal onde deve ser realizada a operação.

O programa é de fácil uso, gerando automaticamente o relatório com informações da execução do trace em uma tela dedicada. Seu modo de execução passo a passo facilita a análise e estudo das interações, algo essencial para o correto entendimento de memória cache.

2. Arquivos de *Trace*

Para gerar os arquivos de *Trace* foi desenvolvido o seguinte programa em java:

```
public class TraceGenerator {

   private static final int NUMBER_OPERATIONS=1000;      // Numero de acessos a memoria
   private static final int NUMBER_WORDS=256000;         // Numero de palavras da memoria

   private static final int TYPE=3;    // 0-> aleatorio
                               // 1-> localidade temporal
                               // 2-> localidade espacial
                               // 3-> temporal e espacial

   public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException {
      PrintWriter writer = new PrintWriter("trace.txt", "UTF-8");
      Random rand = new Random();

      int pos=0;
      switch(TYPE){
         case 0: // aleatorio
            for(int i=0; i<NUMBER_OPERATIONS; i++){
               pos = rand.nextInt(NUMBER_WORDS);
               writer.println("2 "+Integer.toHexString(pos));
            }
            break;
         case 1: // temporal
            for(int i=0; i<NUMBER_OPERATIONS; i++){
                // Iremos garantir a localidade temporal apenas em 75% dos casos
                if(rand.nextFloat()<0.75){
                   // O numero é multiplicado por 4 para garantir apenas a localidade temporal e não espacial
                   // Usamos 0-20 pois é um intervalo pequeno onde os números irão se repetir frequentemente
                   pos = rand.nextInt(20) * 4;
                   writer.println("2 "+Integer.toHexString(pos));
               }else{
                   pos = rand.nextInt(NUMBER_WORDS);
                   writer.println("2 "+Integer.toHexString(pos));
               }
            }
            break;
         case 2: // espacial
            for(int i=0; i<NUMBER_OPERATIONS; i++){
               // Usamos o diff para garantir que a variação entre as interações será pequena e controlada
               int diff = rand.nextInt(30);
               diff -= 10;     // Diff terá um valor de -10 a 10
               pos += diff;
               // Caso o número seja negativo, devemos gerar outro
               // Assim como devemos gerar outro caso ele esteja fora do tamanho da memória
               while (pos<0 || pos>=NUMBER_WORDS){
                  pos -= diff;
                  diff = rand.nextInt(30);
                  diff -= 10;
                  pos += diff;
               }
```

```
            writer.println("2 "+Integer.toHexString(pos));
        }
        break;
    case 3:
        for(int i=0; i<NUMBER_OPERATIONS; i++){
            // Para 50% dos casos, iremos garantir a localidade temporal
            if(rand.nextFloat()<0.50){
                pos = rand.nextInt(20)*4;
                writer.println("2 "+Integer.toHexString(pos));

            // Para os outros 50%, iremos garantir a localidade espacial
            }else{
                int diff = rand.nextInt(30);
                diff -= 10;     // Diff terá um valor de -10 a 10
                pos += diff;
                while(pos<0 || pos>=NUMBER_WORDS){
                    pos -= diff;
                    diff = rand.nextInt(30);
                    diff -= 10;
                    pos += diff;
                }
                writer.println("2 "+Integer.toHexString(pos));
            }
        }
        break;
    }
    writer.close();
}
```

Foram gerados arquivos com 1000 acessos à memória cada, conforme a lógica definida acima para os 4 casos diferentes: Aleatório, Localidade Temporal, Localidade Espacial e Localidade Temporal e Espacial. Observe a lógica empregada para gerar cada um dos dois tipos:

- **Localidade Temporal:** Escolhemos um intervalo relativamente pequeno (no caso, 0 a 20) e mantivemos todas as gerações randômicas dentro deste intervalo. Dado o grande número de interações que escolhemos, as chances de valores neste intervalo se repetirem com grande frequência são altas, o que garante a localidade temporal. No caso, introduzimos um condição que afeta 75% dos casos, com o objetivo de aproximar o trace de um cenário real.

- **Localidade Espacial:** Definimos uma variável responsável pela normalização dos valores de leitura ao longo das interações. Com isso, podemos controlar a variação e mantê-la em valores bem baixos (escolhemos 30 como variação máxima), o que garante a localidade espacial.

- **Ambos:** Para um trace com ambas as localidades embutidas, definimos que 50% dos casos teriam localidade espacial e os outros 50% teriam a localidade espacial. Atingimos este objetivo usando uma condição no início de cada interação.

3. Configurações das Arquiteturas

Como base vamos trabalhar com uma memoria principal de 1 MB (256.000 palavras) e a seguinte arquitetura da cache L1:

Tamanho da Cache	64 palavras
Tamanho do Bloco	4 palavras
Função de Mapeamento	Direto
Algoritmo de Substituição	FIFO
Número e Tipos de Cache	1 cache unificada

Resultados obtidos:

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0	11000
Localidade Temporal	0.448	6520
Localidade Espacial	0.4	7000
Localidade Temporal e Espacial	0.747	3530

3.1. Alterando o Tamanho da Cache

- **128 Palavras:**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0.001	10990
Localidade Temporal	0.608	4920
Localidade Espacial	0.402	6980
Localidade Temporal e Espacial	0.972	1280

- **256 Palavras**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0.001	10990
Localidade Temporal	0.668	4320
Localidade Espacial	0.402	6980
Localidade Temporal e Espacial	0.972	1280

3.2. Alterando o Tamanho do Bloco

- **8 Palavras:**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0	11000
Localidade Temporal	0.434	6660
Localidade Espacial	0.554	5460
Localidade Temporal e Espacial	0.779	3210

- **16 Palavras**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0	11000
Localidade Temporal	0.425	6750
Localidade Espacial	0.715	3850
Localidade Temporal e Espacial	0.829	2710

3.3. Alterando a Função de Mapeamento

- **Associatividade 4:**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0	11000
Localidade Temporal	0.383	7170
Localidade Espacial	0.402	6980
Localidade Temporal e Espacial	0.703	3970

- **Associatividade 8**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0	11000
Localidade Temporal	0.365	7350
Localidade Espacial	0.402	6980
Localidade Temporal e Espacial	0.711	3890

3.4. Alterando o Algoritmo de Substituição

Como não existe política de substituição em mapeamento direto, vamos utilizar para o teste seguinte associatividade 8.

- **LRU:**

Tipo de Trace	Taxa de acerto	Tempo total
Aleatório	0	11000
Localidade Temporal	0.394	7060
Localidade Espacial	0.402	6980
Localidade Temporal e Espacial	0.719	3810

3.5. Alterando o Número de Caches

- **2 Caches (L2 com 256 palavras e 8 palavras por bloco):**

Tipo de Trace	Taxa de acerto (L1)	Taxa de acerto (L2)	Tempo total
Aleatório	0	0	11990
Localidade Temporal	0.448	0.416	4772
Localidade Espacial	0.4	0.263	6020
Temporal e Espacial	0.747	0.944	1393

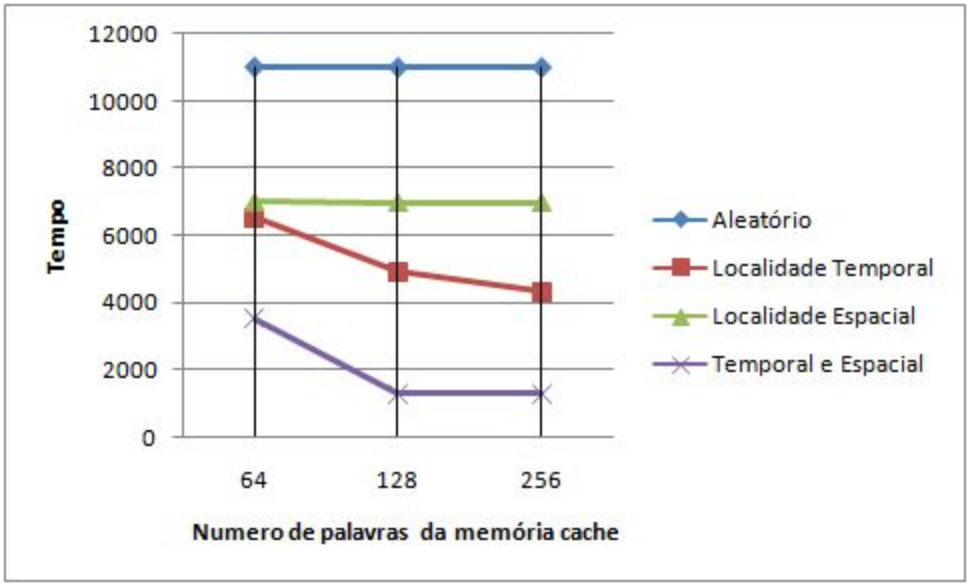
- **3 Caches (L3 com 1024 palavras e 16 palavras por bloco):**

Tipo de Trace	Taxa de acerto (L1)	Taxa de acerto (L2)	Taxa de acerto (L3)	Tempo total
Aleatório	0	0	0.002	12969
Temporal	0.448	0.416	0.152	4604
Espacial	0.4	0.263	0.364	4852
Temporal e Espacial	0.747	0.944	0.5	1337

4. Resultados e Análise

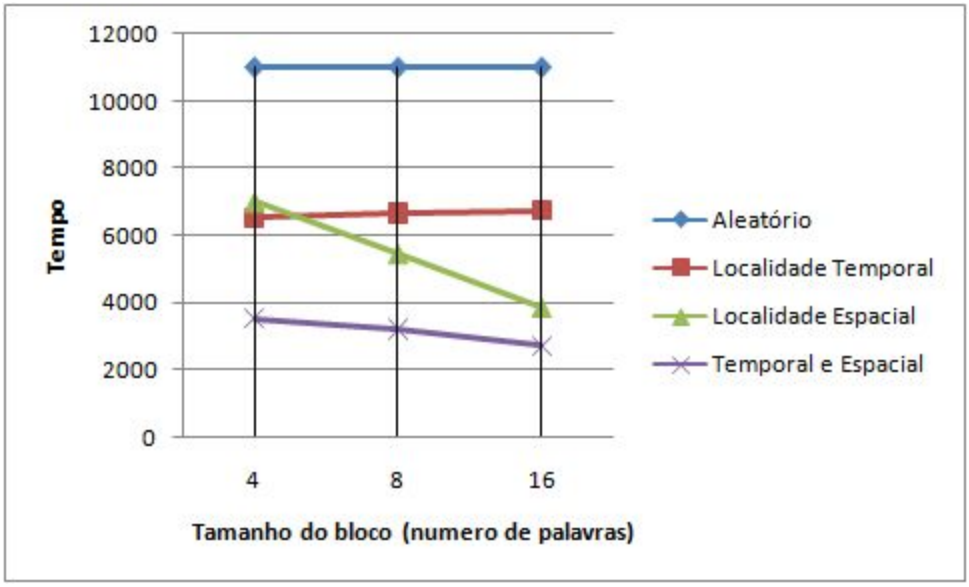
3.1 Alterando o Tamanho da Cache

Como era de se esperar, quanto maior o tamanho da cache, menor o tempo necessário para o processamento dos dados. Pode-se notar facilmente este melhor desempenho no caso de localidade temporal, porém para o caso de acesso aleatório e o caso de localidade espacial, não foi observado uma melhoria no rendimento. Por que? Aparentemente, o tamanho da cache foi muito pequeno em relação ao tamanho da memória principal. Quando realizados acessos aleatórios, dificilmente um dado que já estava na cache chegou a ser requisitado (nota-se isso em todos os casos de testes feitos) e, a respeito da localidade espacial, o tamanho de bloco utilizado foi pequeno; logo, o fato de apresentar localidade espacial só sofreria mudanças claras em seu desempenho caso o tamanho do bloco fosse alterado, como será visto a seguir.



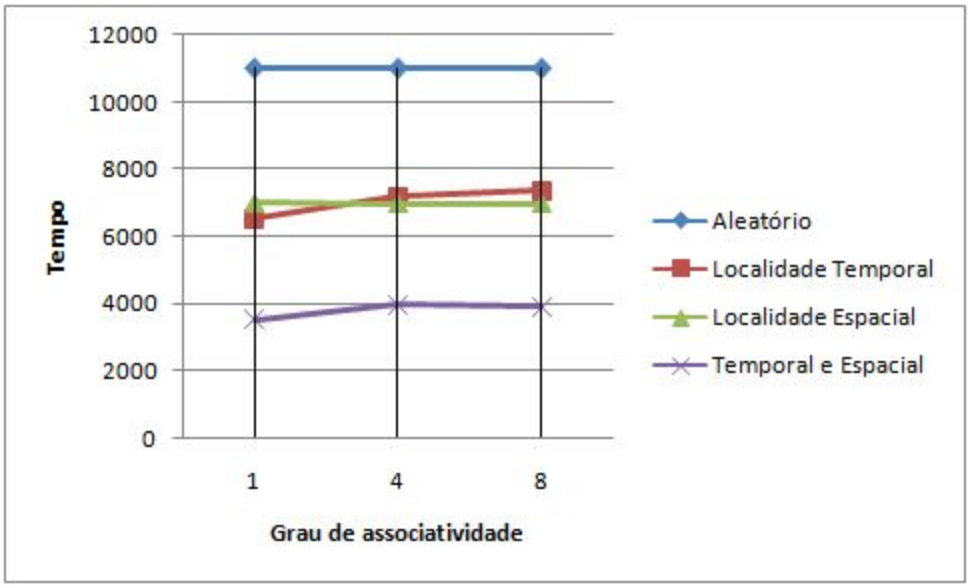
3.2 Alterando o Tamanho do Bloco

Como era de se esperar, o desempenho para o caso de localidade espacial foi o mais afetado com o aumento do tamanho de palavras do bloco. Isso ocorre pois quanto maior o bloco, mais palavras adjacentes àquela requisitada serão trazidas para a memória cache; assim, a localidade espacial oferece maiores chances de acesso a palavras próximas na memória e o aumento do desempenho representou exatamente isso. No caso de localidade temporal, o tamanho do bloco não deixou o sistema mais eficiente, pelo contrário, aumentou o tempo de processamento. Isso ocorreu pois, na localidade temporal, os dados mais freqüentemente utilizados não estão localizados próximos uns dos outros na memória.



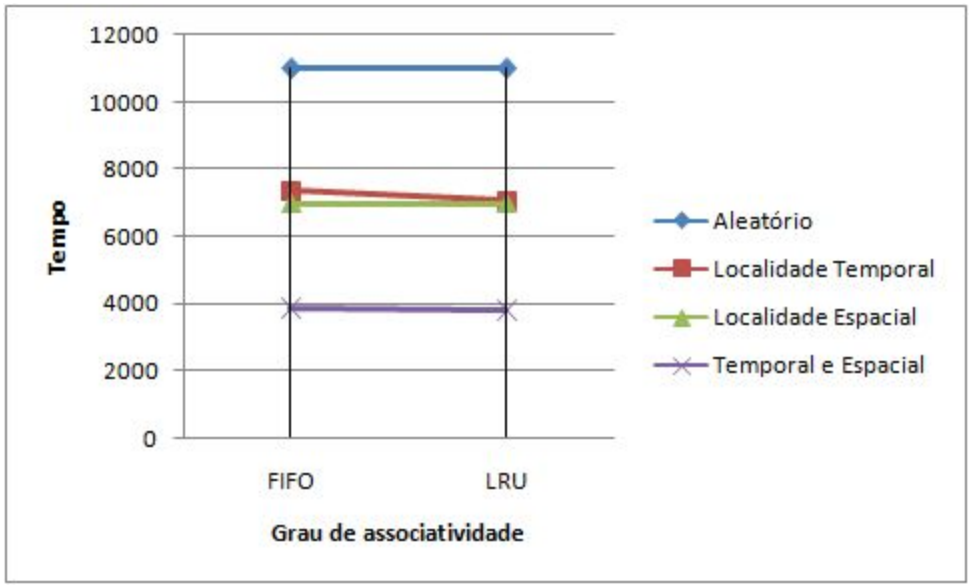
3.3 Alterando a Função de Mapeamento

Aumentar o grau de associatividade não trouxe nenhum benefício para os casos estudados. Porém isso não significa que o mapeamento direto é a melhor opção para todos os casos. Vale ressaltar que os testes foram realizados numa configuração específica de memória, portanto aumentar o grau de associatividade não foi eficiente para essa configuração específica da memória.



É possível que o gargalo desse teste foi o tamanho reduzido de memória cache, pois, ao aumentar o grau de associatividade, foi reduzido o número de conjuntos e aumentando o número de acessos ao mesmo ponto da memória cache. Como o tamanho da memória principal era muito grande, o índice de substituição manteve-se elevado, prejudicando o desempenho do teste no caso da localidade temporal.

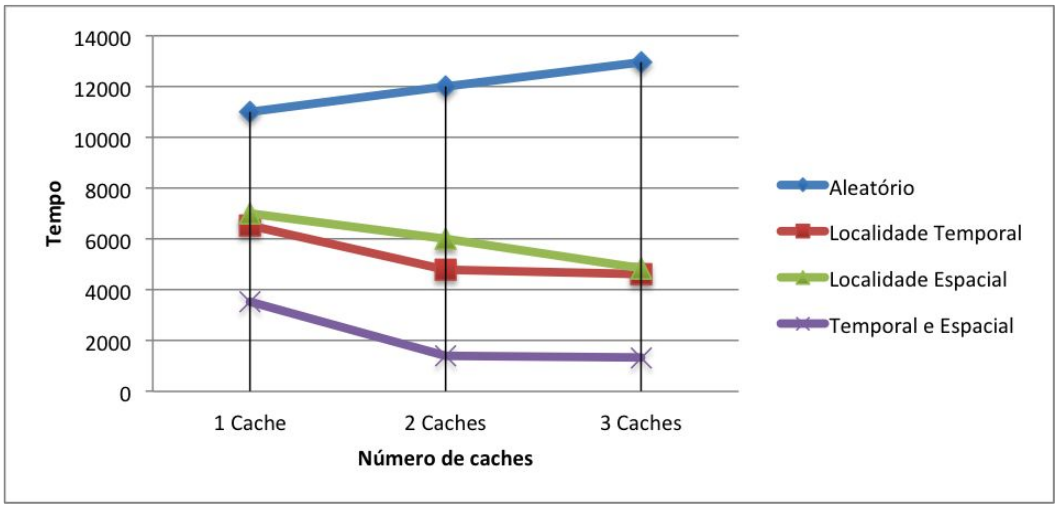
3.4. Alterando o Algoritmo de Substituição



Ao alterar o algoritmo de substituição, não percebemos nenhuma melhora ou piora significativa no desempenho de nenhum dos casos. Isso ocorreu pois o tamanho da memória cache era relativamente pequeno e, a utilização do *Least Recent Used* quanto do *First In First Out* representavam o mesmo resultado pois o bloco não tinha tamanho suficiente para formar histórico que diferenciasse o menos usado do último que foi adicionado na cache.

3.5. Alterando o Número de Caches

Ao testarmos outras variações de cache, tivemos dois resultados distintos: o caso de acesso aleatório teve um desempenho inferior enquanto os três casos (localidade temporal, localidade espacial e localidade temporal e espacial) tiveram um desempenho melhor. O desempenho inferior do caso aleatório é explicado devido ao maior número de níveis necessários para chegar a memória principal; quanto maior o número de caches para chegar ao processador, pior será o desempenho do caso de acesso aleatório. No caso da localidade temporal, o melhoria no desempenho ocorreu devido principalmente ao uso frequente dos dados; quando um bloco estava no nível superior, ele tinha altas chances de ser usado novamente. No caso da localidade espacial, o uso de blocos com diferentes tamanhos selecionou os dados de uma maneira conjunta, ou seja, como estavam adjacentes uns aos outros e eram transferidos em bloco, tinham altas chances de serem acessados com maior frequência no mesmo intervalo de tempo.



5. Conclusão

Como pode ser visto, esse trabalho foi concluído e analisado em todas suas instâncias. Foram criadas 11 arquiteturas, sendo que para cada uma delas foi rodado 4 casos de trace, totalizando mais de 40 execuções no Amnésia.

Algo curioso que a equipe notou foi a baixa performance do Amnésia, que para a execução de um simples arquivo de trace com 1.000 comandos levou às vezes mais de cinco minutos para completar a operação. Também percebeu-se a falta de alguns artifícios úteis no programa, como arrastar arquivos para rápida importação, salvar a última pasta aberta e, principalmente, permitir a execução de arquivos que não estejam na mesma pasta do executável.

Durante a análise, o grupo conseguiu gerar raciocínios lógicos para explicar a performance para cada uma das variações testadas. Estes raciocínios serviram para compreender melhor o funcionamento da memória cache e serviram como forma de estudo para as avaliações do curso. Usamos uma planilha de Excel para a geração de gráficos de maneira uniforme, dando ao documento uma padronização mesmo sendo criado por diversos autores.