INSTITUTO METRÓPOLE DIGITAL
Disciplina: Arquitetura de computadores (DIM0127)
Professora: Mônica Magalhães

Implementação de simulação da cache de sistemas multi-core com três níveis de memória

Lucas Gabriel Matias Paiva

Marco Antonio Fernandes

Natal, 04 de dezembro de 2019

INTRODUÇÃO

O constante aumento da necessidade de mais poder computacional levou à implementação de diversas soluções, métodos e técnicas para permitir maior aproveitamento dos recursos disponíveis, adicionar mais possibilidades de uso e especializar ou generalizar os sistemas em projeto. Frente à isso, surge a ideia de posicionar diversos centros de processamento em um único chip, o chamado multi-core.

A base da utilidade de um sistema de processadores com mais de um núcleo surge da incapacidade dos sistemas de refrigeração usuais em manter sob controle a temperatura de cores muito poderosos, e a dificuldade (e consequente custo) de gerar processadores com clocks cada vez maiores. Além disso, um problema gigantesco é o gargalo de Von Neumann, ou seja, a incapacidade dos sistemas de memória acompanharem a velocidade dos sistemas processadores.

Assim, o uso de sistemas descentralizados permite a aplicação de memórias e barramentos distintos entre os centros e, consequentemente, diminui o tempo ocioso do processador.

Neste trabalho, objetivamos simular o comportamento dos sistemas de memória cache de processadores multi-core, utilizando o sistema de write-through, em que, ao ser processado um dado presente na cache de algum dos cores, atualiza-se a informação em todas as memórias superiores. Assim, ao alterar-se um dado na cache L1, ocorre a atualização correspondente na L2 e na memória principal.

Os conceitos e limitações usados nesse projeto se enquadram, assim, diretamente com os estilos de hierarquização e atualização de memória estudados durante a disciplina.

METODOLOGIA

A implementação feita para essa simulação foi composta em Java, utilizando-se primariamente do paradigma de orientação a objetos, isto é, classes/objetos feitas(os) para representar os componentes de processamento de memória do sistema Os dados são recebidos por meio da leitura de um arquivo, que, então, preenche a memória principal e se inicia a execução.

Dessa forma, temos as seguintes classes e seus respectivos conteúdos e métodos:

- Cache

  A classe cache foi feita para representar uma memória interna, sendo composta de um array de inteiros onde os dados são armazenados, e métodos que permitem a leitura e escrita de dados. Os dados são armazenados de forma sequencial, sendo uma variável utilizada para armazenar qual a última posição atualizada. Isso permite maior agilidade e simplicidade na atualização e recuperação do dado que está sendo manipulado.

- SystemMemory

  Classe com estrutura muito parecida com a Cache, usada para representar a memória principal, possui mesmos métodos e se difere apenas na apresentação de código e no tamanho padrão da memória (200).

- Core

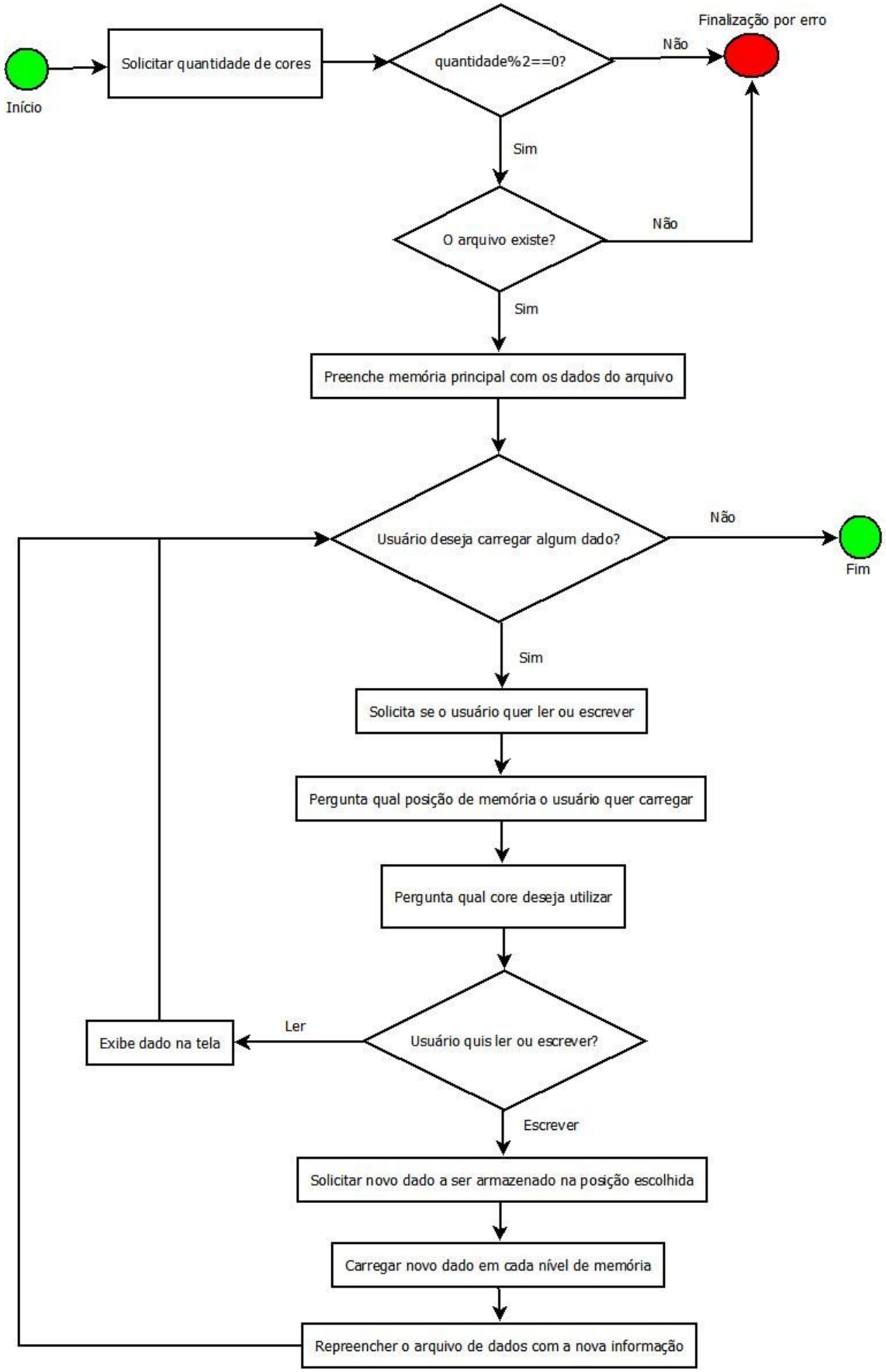
  Unidade que representa um único core isolado, é composto de uma instância de Cache(L1) e métodos que permitem o processamento ou leitura de dados em uma posição qualquer da memória principal. Os dados, ao serem atualizados no core, são atualizados em todas as instâncias superiores (cache do processador e SystemMemory, até chegar ao arquivo) (write-through).

- Processador

  Classe que representa um processador dual-core. Basicamente, é composto de duas instâncias de Core e uma de Cache (L2), possuindo métodos para acesso e atualização dos mesmos.

A execução do projeto se concentra, assim, com a informação de quantos cores o usuário deseja simular, seguido do recebimento de um arquivo de dados (padrão). A quantidade de cores, por motivos de projeto, deve ser múltiplo de 2, considerando que cada centro de processamento é composto de 2 núcleos. A partir daí, recebe-se do usuário a posição de memória que deve ser utilizada para o processamento, o core que deseja utilizar, e, em seguida, a intenção de alterar ou simplesmente realizar a leitura da mesma. A alteração no core implica em gerar alterações no processador e, em seguida, na memória principal.

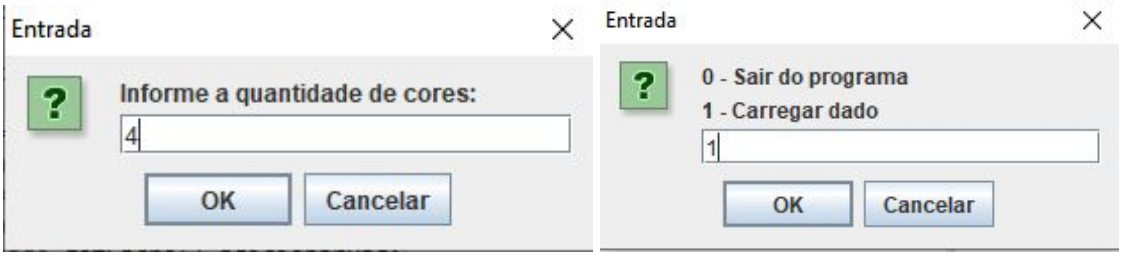
FLUXOGRAMA DO PROJETO



RESULTADOS

O projeto descrito não apresenta geração de dados de forma direta, apenas a capacidade de ler e escrever em um arquivo e em memórias diversas. Isso dificulta a discussão orientada a resultados obtidos, restando apenas descrever que o acesso a posições diversas de memórias permite a alteração e leitura após alteração dos mesmos com a correta modificação e representação dos dados.

Assim, uma execução padrão do programa se daria da seguinte forma (conforme fluxograma disposto à frente):

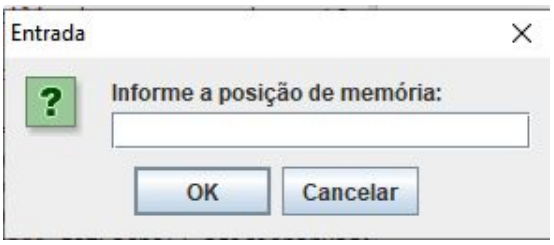


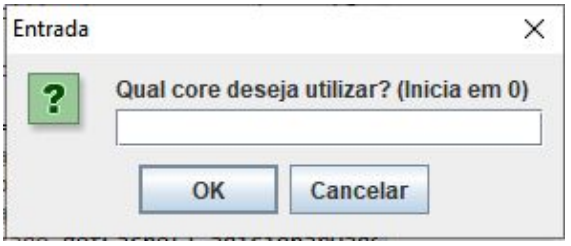
Imagens 1 e 2: entrada exemplo de como acessar o programa.

As opções constroem a forma como o sistema de processadores será criado, sendo necessário que a quantidade de cores seja um múltiplo de 2 maior que 0, ou ocorrerá uma mensagem de erro, e uma inserção de 0 na segunda etapa (quando ocorre o loop de execução), levaria à saída normal do programa.

Imagens 3 e 4: saída por número inválido de cores e saída regular.

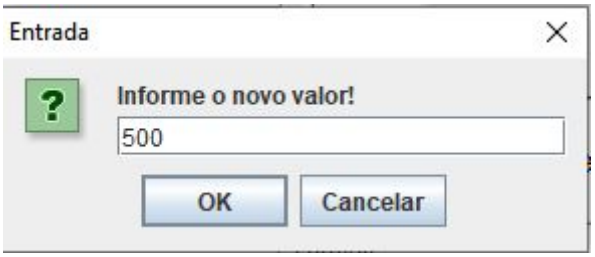
A continuação da execução regular, por meio da opção de carregar dados, habilita o menu de seleção dos cores e da posição de memória que deseja-se executar:





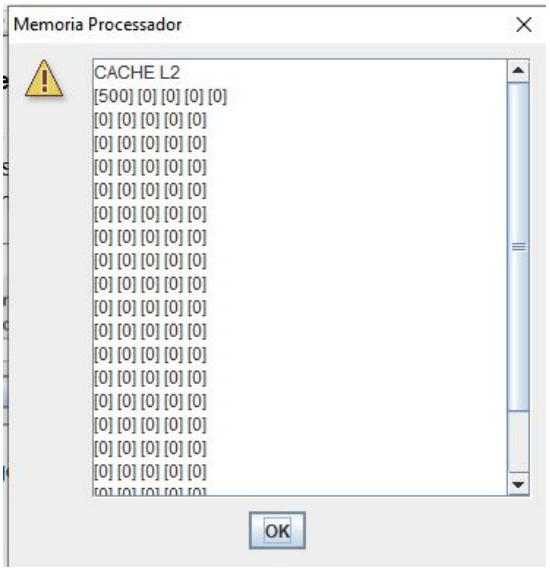
Imagens 5 e 6: Entradas de posições de memórias e core utilizado.

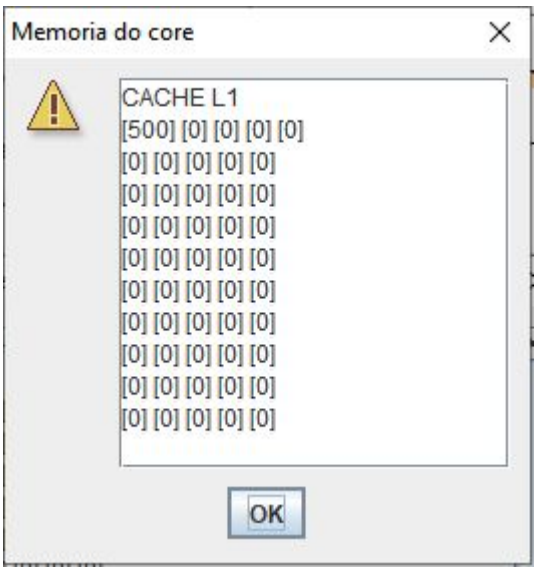
Após a inserção da posição e core, entra-se no menu de leitura ou escrita. A escolha da opção 2 permite a inserção de um valor novo na posição escolhida anteriormente.



Imagens 7 e 8: Entrada do menu de inscrição e inserção de valor.

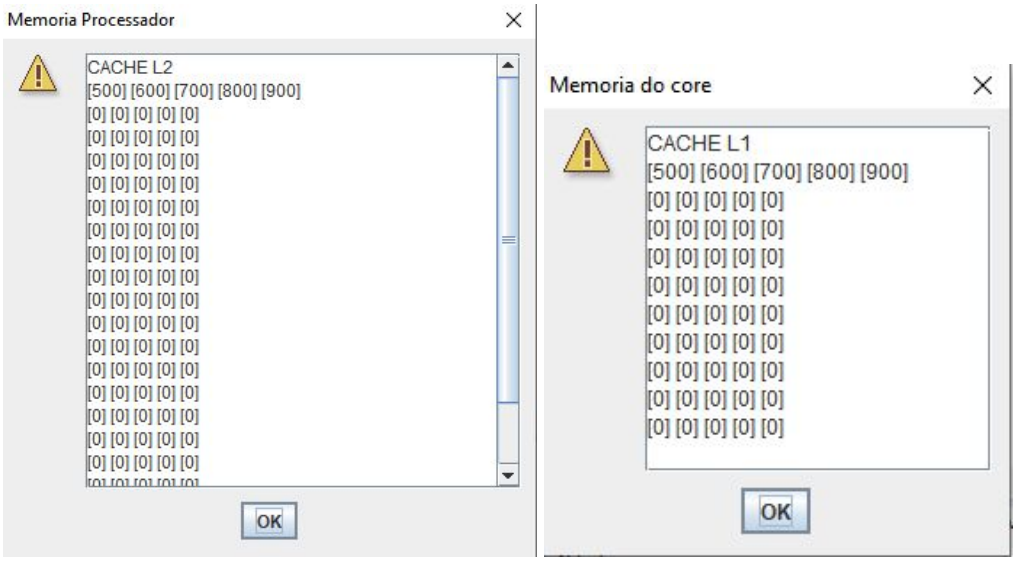
Após a inserção, ocorre a impressão de ambas as caches para visualização de resultados.





Imagens 9 e 10: Caches após inserção.

Inserções posteriores alocam as posições seguintes da cache, como demonstrado a seguir:



Tal qual esperado da hierarquia das memórias.

CONCLUSÃO

Assim, frente ao comportamento e implementação geral do algoritmo, podemos inferir a capacidade do sistema implementado em gerar simulações fidedignas de um processador com diversos cores, permitindo correta leitura, escrita e inserção em todos os níveis de cache na posição adequada a estes, como esperado de um sistema deste tipo.

DESCRIÇÃO DA ORGANIZAÇÃO

Dentro do diretório onde encontra-se o projeto (em anexo), têm-se duas seções relevantes ao usuário, o diretório “dados” contém o arquivo “arquivodados”, onde pode-se colocar as informações a serem carregadas na memória simulada, até um limite de 200 informações (tamanho padrão da memória principal).

O outro diretório relevante é o “src”, onde encontram-se os arquivos de código do projeto, incluindo, na raiz, o arquivo main.java, que concentra os métodos primários de declaração dos objetos e variáveis, além de pacotes contendo os métodos que regem a simulação.

No diretório/pacote “entidades” temos os arquivos que contêm o código relativo às classes descritas anteriormente na seção de metodologia. Em

“utilities” se encontra uma pequena função estática responsável pela leitura e escrita no “arquivodados”.

COMPILAÇÃO E EXECUÇÃO

O projeto acompanha um executável .jar, por motivos de projeto, o executável só é garantido de executar normalmente em sistemas Windows, o comportamento em outros sistemas operacionais é desconhecido e podem ocorrer problemas de acesso.

Para esses casos, a compilação pode se dar por meio de uma IDE com suporte a Java (Eclipse, VSCode, NetBeans, etc) ou, na ausência de uma destas, acessando-se a pasta “src”, pode-se compilar pelo terminal com o comando “javac Main.java”.

O código se encontra disponível em https://github.com/lucasgmpaiva/projeto_arquitetura.