Simulador de Caches Multiníveis para o Processador RISC-V

Nikolas Cavalheiro Gonçalves da Silva
Samuel Willian de Souza Moraes

Orientadora: Nahri Balesdent Moreano

Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)

Resumo

No ensino de Arquitetura e Organização de Computadores, o uso de ferramentas de simulação é fundamental para o aprendizado, pois permite visualizar e compreender o funcionamento de sistemas complexos. Nesse contexto, a ferramenta *Educacional Multilevel Caches Simulator* (EMCS) é um recurso valioso, especialmente após sua integração ao RARS, um simulador do conjunto de instruções RISC-V. Essa mudança permite combinar as funcionalidades do EMCS na simulação de memórias cache multiníveis com a capacidade de executar programas RISC-V. Com o EMCS integrado ao RARS, os alunos podem explorar diferentes configurações de dois níveis de memórias cache, visualizar os acessos realizados às memórias durante a execução de um programa RISC-V, observando o comportamento das mesmas, e analisar os resultados de desempenho, por meio de uma interface gráfica intuitiva e interativa que enriquece a experiência de aprendizado.

Abstract

In teaching Computer Architecture and Organization, the use of simulation tools is essential for learning, as it allows visualizing and understanding how complex systems work. In this context, the Educational Multilevel Caches Simulator (EMCS) tool is a valuable resource, especially after its integration with RARS, a RISC-V instruction set simulator. This change allows combining the EMCS functionalities in simulating multilevel cache memories with the ability to execute RISC-V programs. With EMCS integrated with RARS, students can explore different configurations of two levels of cache memories, visualize the accesses made to the memories during the execution of a RISC-V program, observing their behavior, and analyze the performance results, through an intuitive and interactive graphical interface that enriches the learning experience.

1. Introdução

O desempenho da hierarquia de memórias é um aspecto crucial para o desempenho geral de um computador. Nesse contexto, compreender o funcionamento dessa hierarquia torna-se indispensável para estudantes da área de Computação. Ferramentas que simulam componentes de hardware, como simuladores de memórias cache, apresentam-se como instrumentos valiosos no ensino de Arquitetura e Organização de Computadores. Uma revisão detalhada e análise comparativa de diversos simuladores de cache é apresentada em [1].

A arquitetura de processador MIPS [4] por tempos foi a padrão na literatura acadêmica nas disciplinas de Arquitetura e Organização de Computadores e para ela foram desenvolvidos diversos softwares com o intuito de estudar e compreender o seu funcionamento. Dentre essas ferramentas está o MARS (*MIPS Assembler and Runtime Simulator*) [7], que foi utilizada no desenvolvimento das ferramentas ECS (*Educational Cache Simulator*) e EMCS (*Educational Multilevel Caches Simulator*) [3]. Com o avanço da tecnologia e o surgimento da arquitetura RISC-V [5, 6], este processador passou a ser usado na literatura acadêmica, em substituição ao processador MIPS. Assim, surgiram diversas ferramentas para esse processador, dentre elas o RARS (*RISC-V Assembler and Runtime Simulator*) [2], correspondente ao MARS, porém para o novo conjunto de instruções.

Seguindo essa evolução, esse trabalho trata do desenvolvimento de uma nova versão do simulador EMCS, para uso acoplado à ferramenta RARS e assim permitir a simulação de memórias cache multiníveis durante a execução de um programa RISC-V. Tal desenvolvimento envolve a migração do EMCS e também a atualização e melhoria de suas funções. A Figura 1 ilustra o principal objetivo deste trabalho.

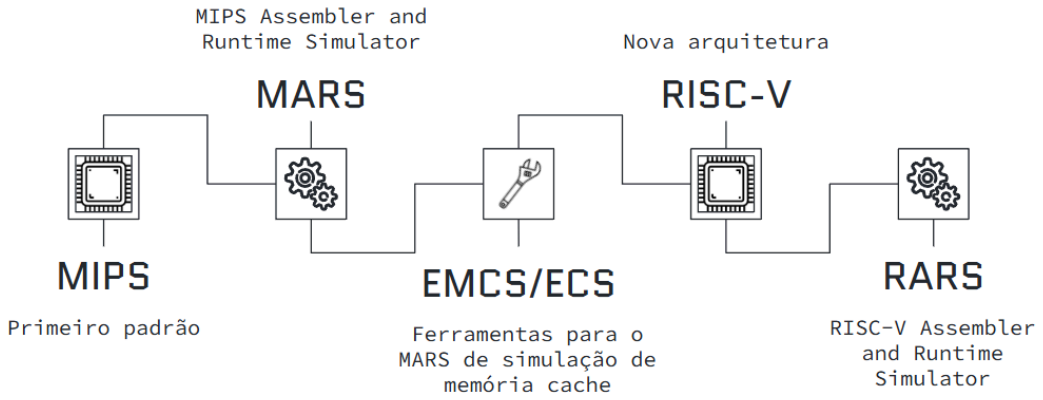

Figura 1. Migração do EMCS da ferramenta MARS para a RARS

Este texto está dividido em sete seções. A Seção 2 apresenta a ferramenta EMCS e descreve suas funcionalidades. Na Seção 3 é demonstrado como foi o processo de migração da ferramenta do MARS para o RARS, com reestruturações e correções no código. As modificações de otimização, melhorias na estrutura e apresentação do código estão descritas na Seção 4. As Seções 5 e 6 ilustram o uso do EMCS para a realização de experimentos, com a simulação de um programa RISC-V e a simulação de um *trace* de endereços de memória, respectivamente. Por fim, a Seção 7 apresenta a conclusão do trabalho e sugere trabalhos futuros.

2. *Educational Multilevel Caches Simulator* (EMCS)

O EMCS é uma ferramenta educacional acoplada ao simulador RARS, projetada para simular o comportamento de memórias cache multiníveis, apresentando, de forma didática e visual, o seu funcionamento. Essa ferramenta permite que estudantes acompanhem o processo de acessos às memórias cache, tornando o aprendizado mais prático e interativo.

A ferramenta oferece diversas funcionalidades para simulação de dois níveis de caches de dados, de instruções ou unificada, permitindo a configuração detalhada de cada cache. Além disso, exibe cada acesso realizado às caches, proporcionando um acompanhamento detalhado de acertos, falhas e substituições de blocos. Os acessos à memória simulados pelo EMCS podem ser oriundos da execução de um programa RISC-V no RARS, com o EMCS atuando como um simulador *emulation-driven*.

Também é possível simular acessos à memória oriundos de um *trace* de endereços de memória, fornecido pelo usuário, e nesse caso o EMCS funciona como um simulador *memory trace-driven*. O *trace* é fornecido em um arquivo texto, com um acesso por linha, contendo o tipo da operação (0, 1 ou 2, para leitura de instrução, leitura de dado e escrita de dado, respectivamente) e o endereço acessado.

O EMCS possui duas abas que oferecem as seguintes funcionalidades:

- *Settings*: Permite a configuração dos dois níveis de cache a serem simulados, incluindo a definição do número de blocos, associatividade, tamanho do bloco e política de substituição, de cada cache individualmente. Também é possível definir os tempos de acesso a cada nível de cache e à memória principal.
- *Access & Performance*: Mostra medidas de desempenho de cada nível de cache, como o número de acessos, acertos, falhas e substituições de blocos, além da taxa de falhas e o tempo médio de acesso à memória. Exibe também uma tabela com informações sobre todos os acessos realizados. E permite a carga e simulação de um *trace* de endereços de memória.

Uma maneira alternativa de utilizar a ferramenta EMCS é através da sua execução autônoma (*stand-alone*), com o RARS sendo executado apenas em *background*, sem o seu ambiente de desenvolvimento. Para isso, é necessário utilizar o seguinte comando em um terminal, na pasta em que o programa executável rars_1.6_EMCS.jar se encontra:

```
java -cp rars1_6_emcs.jar rars.tools.EducationalMultilevelCachesSimulator
```

3. Migração do EMCS para o Simulador RARS

Com o objetivo de migrar a ferramenta EMCS para o simulador RARS, foi realizada uma análise detalhada do seu código, buscando identificar pontos relevantes para o processo. Durante essa avaliação, constatou-se que a ferramenta ECS compartilhava módulos com a EMCS. Considerando que o foco do projeto era exclusivamente a EMCS, decidiu-se separar esses módulos e refatorá-los para que contivessem apenas os elementos essenciais ao funcionamento da ferramenta.

Nesse processo, foi necessário reestruturar as classes relacionadas ao gerenciamento da memória cache, tais como *Cache*, *Controller*, *BlockStatistic*, *SetStatistic*, *StatisticCache* e *TipoCache*. Todas essas classes foram consolidadas em uma única classe denominada *Cache*, com o intuito de simplificar a compreensão do código e reduzir sua extensão. Métodos utilizados exclusivamente pela ECS foram descartados, enquanto que os métodos compartilhados entre as ferramentas foram refatorados.

Na implementação inicial do EMCS, a classe *Cache* tinha como função apenas armazenar as configurações da cache. O *Controller* também mantinha informações sobre a cache, mas incluía métodos *get* e *set* que não eram utilizados, o que tornou sua permanência dispensável. A classe *TipoCache* possuia três inteiros que definiam o tipo de cache a ser utilizado (Dados, Instrução ou Unificada), enquanto a *StatisticCache* era a responsável pelo gerenciamento dos dados da memória, atuando efetivamente como a cache e realizando os cálculos necessários. Essa última utilizava as classes *SetStatistic*, que definia o conjunto de blocos (*array*), e *BlockStatistic*, que representava os blocos da cache. Para otimizar o código e evitar classes com funções limitadas ou redundantes, optou-se por unificar essas funcionalidades na classe *StatisticCache*, realocando apenas os métodos indispensáveis e eliminando elementos desnecessários.

Além da separação dos códigos compartilhados, foram realizadas modificações estruturais para que o código da ferramenta reconhecesse as novas dependências introduzidas pela migração para o simulador RARS. Grande parte dessas mudanças envolveu a substituição de referências ao pacote MARS pelo pacote RARS, processo que se revelou relativamente simples devido às similaridades estruturais entre os dois simuladores no que tange ao acoplamento de novas ferramentas aos simuladores dos processadores. A Figura 2 mostra a modificação feita na importação de pacotes de todas as classes do EMCS.

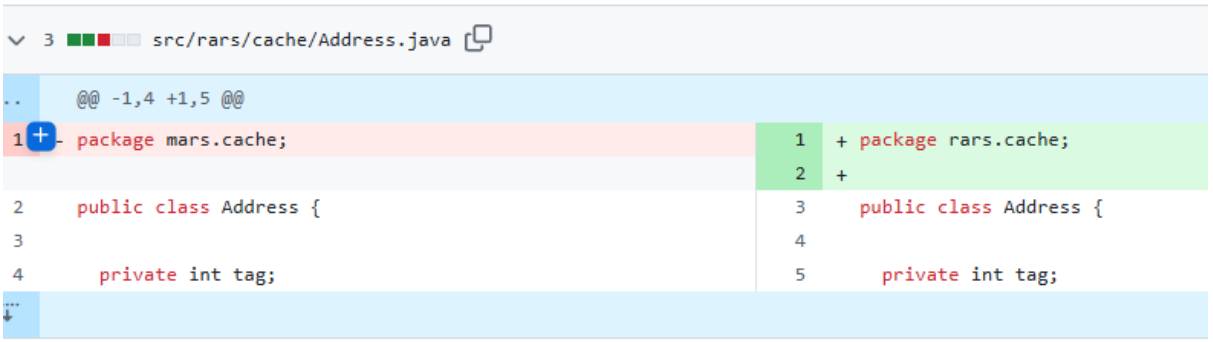

Figura 2: Mudança da importação do pacote do simulador do processador

Na ferramenta MARS, toda nova ferramenta implementada para ser acoplada precisava herdar da classe abstrata *Abstract Tool*, que serve como base para a criação de ferramentas. Essa estrutura também é utilizada no RARS, onde a classe equivalente mantém a mesma funcionalidade.

4. Modificações no EMCS

Com o objetivo de aprimorar a qualidade, a legibilidade do código da ferramenta, a facilidade de manutenção, o desempenho e a facilidade de uso da mesma, foi identificada a necessidade de realizar melhorias e simplificações em várias de suas partes. A versão do EMCS destinada ao MARS apresentava trechos com baixa otimização, códigos redundantes e fragmentos de código morto (que nunca eram executados), o que comprometia sua compreensão. Contudo, considerando o nível de complexidade da ferramenta e para evitar riscos que pudessem comprometer sua funcionalidade, decidiu-se

priorizar as alterações mais relevantes, sem implementar todas as melhorias identificadas. Assim, as seguintes modificações foram realizadas:

- Melhoria na experiência do usuário através de janelas de diálogo do tipo *pop-up* mais descritivas;
- Melhoria na organização e legibilidade do código ao adicionar estruturas de dados mais condizentes com o contexto de simulação de caches, como por exemplo as estruturas *Cache, CacheLevel*, *CacheCounters* e *CacheSettings*;
- Remoção de código morto e redução da duplicação de código utilizando abstrações de conceito, como por exemplo as estruturas *CaseApply*, *CacheCounters* e *CacheSettings*;
- Eliminação da dependência da ferramenta EMCS com a ferramenta ECS;
- Refatoração do código das abas de *Settings* e *Access & Performance* da ferramenta.

Visando melhorar a usabilidade da ferramenta, foi implementada uma funcionalidade que facilita a seleção de um arquivo de *trace* de endereços de memória a ser simulado. Originalmente, ao buscar um novo arquivo *trace*, o usuário era obrigado a localizar novamente o diretório de origem. A nova funcionalidade, implementada na classe *TraceFile*, ajusta o comportamento da ferramenta para que ela retorne automaticamente ao último diretório acessado, tornando o processo mais eficiente e intuitivo.

Além disso, foram adicionadas mensagens de aviso durante a carga e simulação de um *trace* de endereços de memória. Originalmente no EMCS, os usuários não tinham indicadores claros de que a ferramenta estava em processamento, o que frequentemente levava à impressão de que o sistema havia congelado. Com as modificações, ao carregar um arquivo *trace*, o botão "*Run Trace*" é ativado e surge uma janela, mostrada na Figura 3, informando como prosseguir com a execução do *trace*.

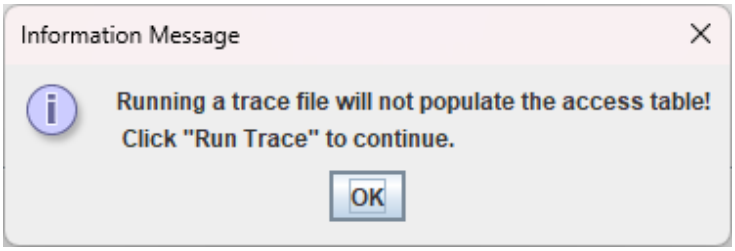

Figura 3: Pop-up de instrução de início do *trace*

Ao acionar o botão "*Run Trace*", ele é automaticamente desativado durante a simulação do *trace*, evitando cliques repetidos que poderiam resultar em resultados de contagem de acessos incorretos, por incluírem execuções consecutivas. Além disso, surge uma janela, mostrada na Figura 4, que notifica o usuário de que o processamento do *trace* está em andamento. Essa janela é automaticamente fechada ao final da simulação do *trace*, proporcionando uma melhor compreensão sobre o estado da ferramenta.

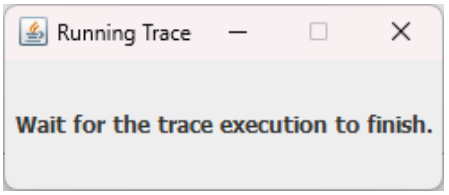

Figura 4: Pop-up de aviso de *trace* em execução

Foi desenvolvida uma nova classe denominada *ComputingTrace*, responsável pela janela que é exibida ao iniciar a execução de um arquivo *trace*. Durante o processo de execução do método *RunTraceFileAP*, a janela é instanciada no início e, ao final da execução, é descartada por meio de um comando *dispose*. Além disso, dentro do mesmo método, o botão "*Run Trace*" é programado para ser desativado no início da execução e reativado ao término, garantindo que não ocorram múltiplas interações simultâneas que poderiam comprometer os resultados.

Com o intuito de facilitar a manutenção e futuras análises do código, foram realizadas alterações significativas para eliminar fragmentos não alcançáveis, conhecidos como código morto. Entre as melhorias, destaca-se a remoção de trechos comentados e métodos e variáveis não utilizados, presentes em diversas classes. Essas modificações foram implementadas para evitar possíveis confusões e simplificar futuras alterações por outros pesquisadores na ferramenta, promovendo um ambiente de desenvolvimento mais claro.

O código da ferramenta apresentava duas classes para a mesma interface, *InterfaceEMCS* e *InterfaceEMCSV2*, sendo que a primeira não era utilizada. Os trechos de código que faziam referência a essa classe não eram executados durante o funcionamento da ferramenta. Após análise, optou-se por remover todas as referências relacionadas e, consequentemente, eliminar a primeira classe correspondente à interface. Na classe *MultilevelCacheTableAccess* também foram identificados e removidos trechos de código comentados, que não contribuíam para o funcionamento da ferramenta e poderiam gerar confusão em futuras atualizações.

Durante o desenvolvimento das interfaces da ferramenta, foi constatada a presença de trechos de código repetidos. Para promover a padronização e reduzir o tamanho do código, foram criadas as classes *DefaultButton* e *DefaultCacheCountersComponent*. Essas classes encapsulam funcionalidades frequentemente utilizadas, permitindo que, sempre que um desses componentes fosse necessário, uma única instanciação fosse suficiente para suprir suas necessidades. Essa abordagem simplifica a implementação, melhora a legibilidade e favorece a manutenção do código.

Originalmente, os cálculos das métricas de desempenho das caches eram realizados com truncamento das casas decimais. Considerando que essa prática não oferece a precisão necessária para a comparação dos resultados obtidos, foram realizadas alterações nos módulos *AccessPerformanceEMCS*, *Utils* e *CacheCountersComponent*. Essas alterações introduziram o arredondamento desses resultados para duas casas decimais, garantindo maior padronização e controle sobre os valores apresentados.

Por fim, como as informações a serem mostradas na tabela de acessos na aba *Access & Performance* mudam de acordo com a configuração das caches simuladas, o método responsável por definir essas informações foi reestruturado para retornar um objeto do tipo *caseApply*, representado na Figura 5, associado a um *enum* previamente definido. Consequentemente, os métodos que utilizavam as variáveis relacionadas foram devidamente ajustados para acomodar suas novas implementações, garantindo maior clareza e eficiência no processamento.

```
17  + import rars.EMCS.Utils.CacheLevel;
18  +
19      /**
20       *
21       * @author dawson
22       */
23      public class MultilevelCacheTableAccess {
24  +       private enum CaseApply {
25  +           STYLE_1,
26  +           STYLE_2,
27  +           STYLE_3,
28  +           STYLE_4;
29  +       }
30
31          private JTable mlcTableAccess;
32          private MultilevelCacheComponent mlc;
33  +       private CaseApply l1CaseApply;
34  +       private CaseApply l2CaseApply;
35          private Address l1Address;
36          private Address l2Address;
37          private List<String> tableColumnName;
```

Figura 5: Definição e uso do *enum CaseApply*

5. Simulação de um Programa RISC-V

Como o principal objetivo do EMCS é o seu uso educacional, esta seção e a próxima apresentam experimentos que utilizam a ferramenta para a simulação de memórias cache multiníveis, visando proporcionar ao estudante uma compreensão mais aprofundada dos conceitos sobre a organização e o funcionamento da hierarquia de memórias e o desempenho da mesma.

Em um primeiro experimento, a ferramenta EMCS é usada para a simulação de dois níveis de cache, durante a execução de um programa RISC-V. Isto é, os acessos à memória simulados pelo EMCS são oriundos da execução de um programa RISC-V no RARS. Assim, o EMCS atua como um simulador *emulation-driven*.

O programa utilizado neste experimento é uma implementação em linguagem de montagem RISC-V do algoritmo de ordenação *BubbleSort*, já familiar aos estudantes da área de Computação, usado aqui para ordenar um vetor de 64 elementos. Após iniciar o RARS (com o EMCS integrado), o programa RISC-V do BubbleSort deve ser carregado e montado. O simulador EMCS é iniciado selecionando-se a opção *Educational Multilevel Caches Simulator* do menu *Tools* do RARS. Assim, a janela do EMCS é aberta, apresentando as suas duas abas, *Settings* e *Access & Performance*, como mostrado na Figura 5. Para que os acessos à memória realizados pelo programa *BubbleSort* durante a sua execução sejam usados pelo EMCS na simulação das memórias cache, é necessário selecionar o botão *Connect to Program* na janela do EMCS.

O próximo passo é selecionar a organização das caches L1 e L2 que se deseja simular. Para fins de experimentação, será utilizada a configuração ilustrada na Figura 6. A configuração escolhida é aplicada selecionando-se o botão *Apply*.

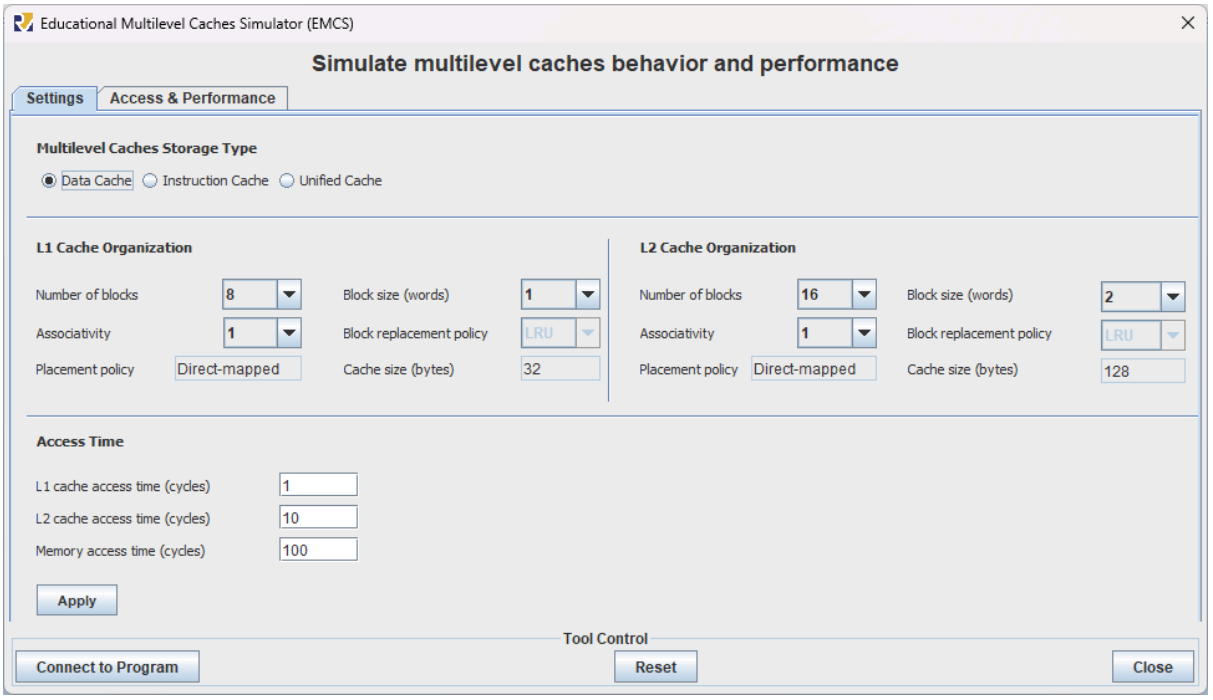

Figura 6: Configuração das memórias cache L1 e L2 a serem simuladas

O programa *BubbleSort* pode então ser executado, através dos botões *Run* ou *Step* do RARS. Essa execução deve ser acompanhada na aba *Access & Performance* do EMCS (Figura 6), onde os resultados da simulação das caches serão visualizados. Essa aba apresenta, na forma de uma tabela, as informações sobre todos os acessos à memória realizados durante a execução do programa, indicando o endereço de memória acessado, a posição acessada na cache L1, se houve acerto ou falha, e se for o caso, as mesmas informações para a cache L2. O EMCS permite que essa tabela seja exportada como uma planilha no formato CSV, caso o aluno deseje analisar as informações.

A aba *Access & Performance* também apresenta os resultados de desempenho da simulação. A Figura 7 mostra os resultados da simulação do BubbleSort para as caches da organização indicada na Figura 6. Os resultados incluem medidas de desempenho da

cache L1 (número de acessos, de acertos, de erros e de substituições, taxa de falhas e número de falhas por instrução), da cache L2 (número de acessos, de acertos, de erros e de substituições, taxa de falhas local e global e número de falhas por instrução) e da hierarquia de memórias como um todo (tempo efetivo de acesso à memória - *Average Memory Access Time*, AMAT).

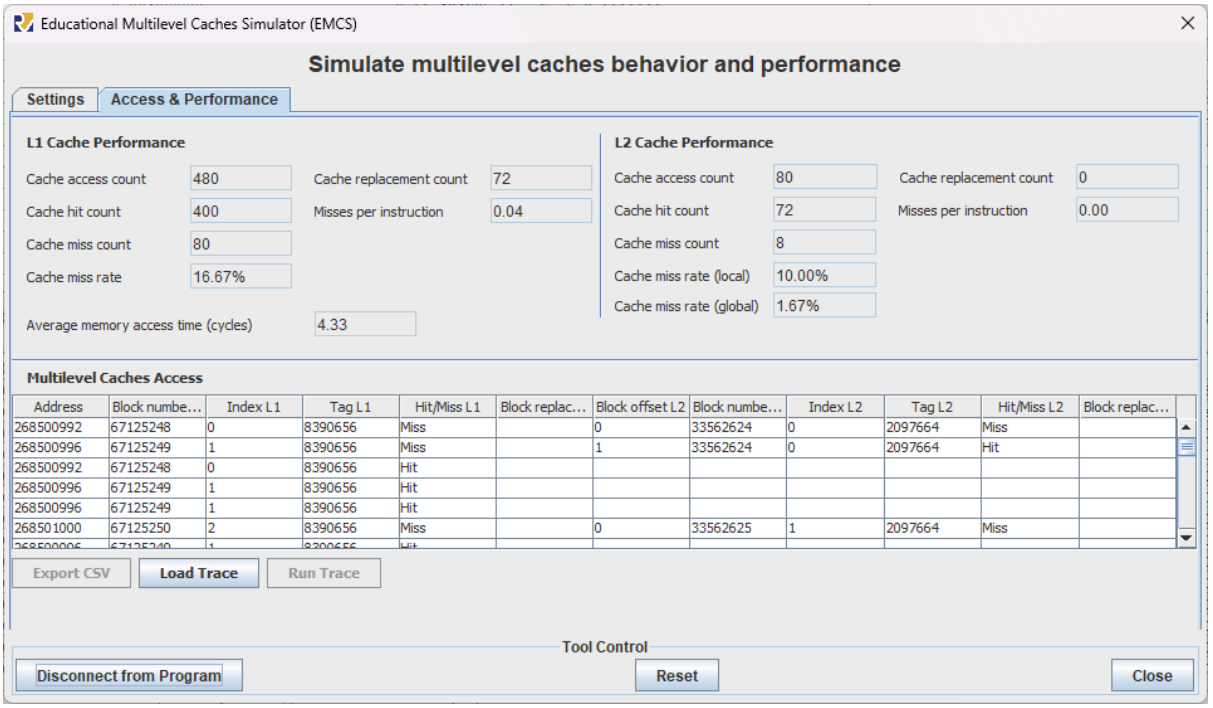

Figura 7: Acessos às caches L1 e L2 e resultados de desempenho da simulação do programa *BubbleSort*

Seguindo os mesmos passos, o aluno pode simular o mesmo programa *BubbleSort* utilizando diferentes configurações de caches. Para isso, basta selecionar o botão *Reset* no EMCS, reconfigurar as caches e executar novamente o programa. Isso permitirá que o aluno compare e analise os diferentes resultados de desempenho obtidos e tire conclusões sobre diversas questões, tais como:

- Qual a diferença entre a taxa de falhas e a razão falhas por instrução?
- Qual é o efeito do tamanho do bloco da cache L1 na taxa de falhas dessa cache? E da cache L2?
- Qual é o efeito da associatividade das caches nas taxas de falhas?
- Qual é o efeito do número total de blocos das caches nas taxas de falhas?
- O programa apresenta localidade temporal e/ou espacial?
- As falhas na cache L1 são predominantemente compulsórias, de capacidade ou de conflito? E na cache L2?

A mesma abordagem analítica pode ser aplicada a outros programas, possibilitando um estudo mais abrangente sobre o comportamento de diferentes aplicações em relação à hierarquia de memórias.

6. Simulação de um *Trace* de Endereços

Em um segundo experimento, a ferramenta EMCS é usada para a simulação de dois níveis de cache, durante o processamento de um *trace* de endereços de memória. Isto é, os acessos à memória simulados pelo EMCS são oriundos de um *trace* de endereços, fornecido através de um arquivo. Dessa forma, o EMCS atua como um simulador *trace-driven*.

Os endereços contidos em um trace correspondem à sequência de acessos à memória realizados durante a execução de um programa. Esses acessos podem ser coletados utilizando outras ferramentas ou mecanismos, como por exemplo outros simuladores ou através da instrumentação do código. Assim, a utilização de arquivos de *trace* possibilita ao EMCS realizar simulações correspondentes à execução de programas mais extensos, desenvolvidos em outras linguagens ou ambientes.

O *trace* de endereços é fornecido ao EMCS em um arquivo texto, com um acesso por linha, contendo o tipo da operação (0, 1 ou 2, para leitura de instrução, leitura de dado e escrita de dado, respectivamente) e o endereço de memória acessado, como exemplificado na Figura 8.

```
1  1860
1  2108
1  1924
1  2112
2  4
```

Figura 8. Exemplo de um arquivo de trace de endereços de memória.

Neste experimento é utilizado um *trace* de memória, usado em [3], contendo 910.237 acessos a dados, correspondentes à execução do programa *Susan Corners* do *benchmark* MiBench. Novamente, inicia-se selecionado a organização da hierarquia de memórias que se deseja simular, na aba *Settings* do EMCS. A Figura 9 mostra a configuração escolhida para esse experimento.

Em seguida, na aba *Access & Performance* do EMCS, mostrada na Figura 9, usando o botão *Load Trace*, seleciona-se o arquivo com o *trace* de endereços desejado. Selecionando o botão *Run Trace,* a ferramenta começará a simular o *trace* de endereços, porém não atualizará os resultados de desempenho a cada iteração (isto é, a cada endereço acessado simulado). Ao invés disso, os resultados de desempenho são mostrados apenas após o processamento do arquivo de *trace* por inteiro. Além disso, na execução no EMCS no modo *trace-driven*, a tabela de acessos não é preenchida, o que permite a simulação de *traces* muito grandes, com milhões de acessos. Quando a execução termina, os resultados de desempenho são exibidos, como ilustrado na Figura 10.

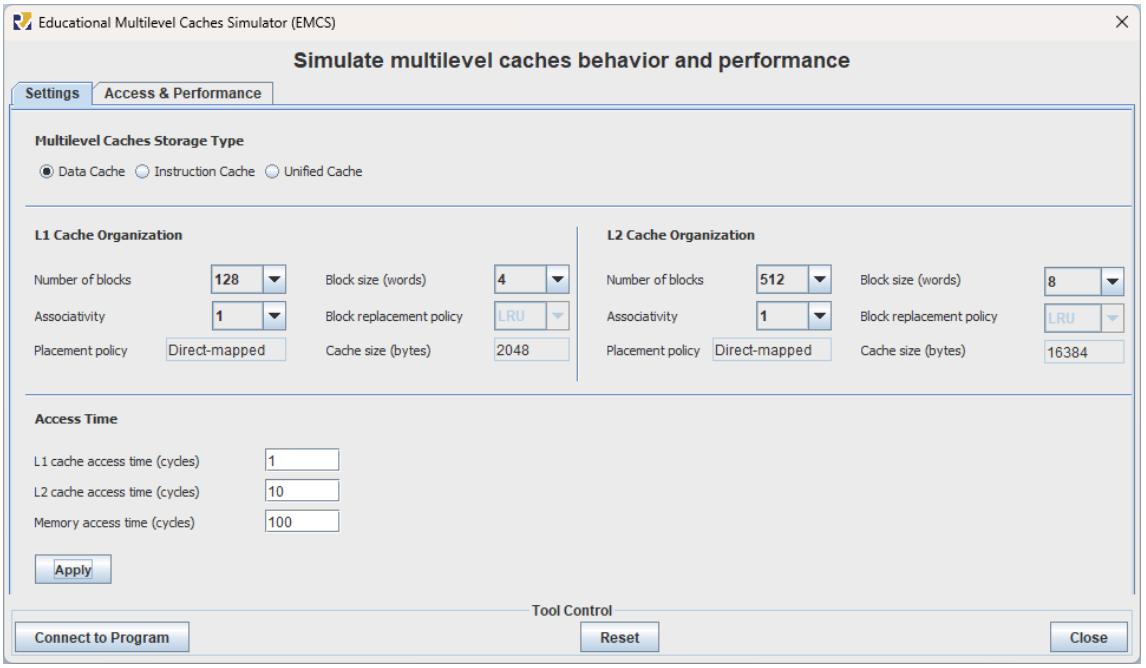

Figura 9: Configuração das memórias cache L1 e L2 a serem simuladas

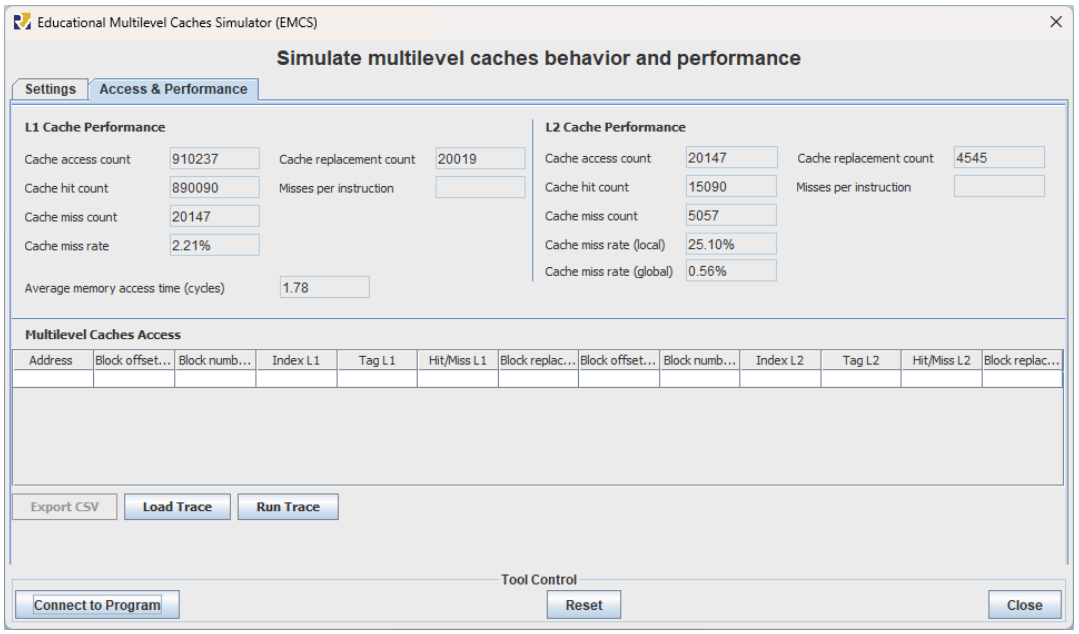

Figura 10: Resultados de desempenho da simulação do *trace* de endereços do programa *Susan Corners*

Avaliando os resultados de desempenho obtidos, o aluno pode analisar as seguintes questões, dentre outras:

- Como as taxas de falhas local e global da cache L2 são calculadas?
- Como o tempo efetivo de acesso à memória (*Average Memory Access Time* - AMAT) é calculado?
- Qual seria o AMAT se houvesse apenas a cache L1, sem a cache L2?

- Como pode ser realizada uma comparação entre uma organização com caches unificadas e uma com caches *split* (separadas para instruções e dados)?

A mesma análise pode ser realizada com outros *traces* e/ou configurações de caches, permitindo uma avaliação de desempenho mais detalhada da hierarquia de memórias.

7. Conclusão

Com os esforços empenhados neste projeto, foi possível migrar com sucesso a ferramenta EMCS do simulador MARS para o RARS, torná-la independente da ferramenta ECS que a acompanhava e garantir o seu funcionamento adequado. O EMCS final tem uma estrutura mais organizada e polida para futuros estudos e oferece uma experiência mais intuitiva para o usuário. Além disso, exemplos de experimentos que podem ser realizados com a mesma são fornecidos, mostrando como utilizar a ferramenta e analisar os resultados produzidos por ela.

Como trabalhos futuros, novas melhorias podem ser implementadas, como por exemplo:

- Modificações para melhorar o desempenho no preenchimento da tabela de acessos na aba *Access & Performance*, pois este apresenta um certo atraso no processamento;
- Usando as ferramentas ECS e EMCS já reestruturadas e integradas ao simulador RARS, desenvolver uma versão do mesmo que contemple ambas as ferramentas, para utilização dos discentes sem que eles precisem usar duas instâncias separadas do software;
- Como os processadores atuais já possuem cache L3, seria interessante adicionar mais um nível de memória cache à hierarquia de memórias simulada pelo EMCS, permitindo estudos da realidade dos processadores atuais. Com a adição de algumas estruturas de dados ao longo do desenvolvimento deste trabalho, essa adição ficou facilitada, entretanto a interface visual precisa ser refeita;
- Documentação e avaliação do código para submeter a ferramenta EMCS para a coleção de ferramentas do RARS.

8. Bibliografia

1. Brais, H., Kalayappan, R. e Panda, P. R. A Survey of Cache Simulators. ACM Computing Surveys, 53(1):19:1-19:32, 2020.
2. Landers, B. RARS – RISC-V Assembler and Runtime Simulator, https://github.com/TheThirdOne/rars, acessado em outubro de 2024.
3. Lima, D. P., Moreano, N. ECS e EMCS: Simuladores de Caches para o Apoio Pedagógico no Ensino de Arquitetura de Computadores, Anais do XXIX Workshop sobre Educação em Computação, 2021.
4. Patterson, D. A., Hennessy, J. L. Computer Organization and Design – The Hardware/Software Interface – MIPS Edition. Elsevier, 6a. edição, 2020.
5. Patterson, D. A., Hennessy, J. L. Computer Organization and Design – The Hardware/Software Interface – RISC-V Edition. Elsevier, 2a. edição, 2021.
6. RISC-V International. RISC-V, https://riscv.org/, acessado em outubro de 2024.
7. Sanderson, P., Vollmar, K. MARS – MIPS Assembler and Runtime Simulator, https://github.com/dpetersanderson/MARS/, acessado em outubro de 2024.