Experimentos com um Simulador de Caches Multiníveis

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)

Neste texto ilustramos a utilização de um simulador de caches multiníveis de uso educacional, o EMCS (*Educational Multilevel Caches Simulator*) [2] para a realização de experimentos de simulação de memórias caches, demonstrando como os estudantes podem estudar os conceitos de hierarquia de memórias através das simulações realizadas. Exemplificamos, com dois roteiros de estudos, o uso do EMCS através da simulação de um programa RISC-V e da simulação de um trace de endereços de memória, nas Seções 1 e 2 respectivamente.

1. Experimento de Simulação de um Programa RISC-V

No primeiro experimento, usamos a ferramenta EMCS para a simulação de uma hierarquia de memórias com dois níveis de cache, seguidas pela memória principal, 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, utilizamos o EMCS como um simulador *emulation-driven*.

O programa que utilizamos neste experimento é uma implementação em linguagem de montagem RISC-V do algoritmo de ordenação *BubbleSort*, já familiar aos estudantes da área de Computação, que usamos aqui para ordenar um vetor de 64 elementos. Após iniciar o RARS (com o EMCS integrado), devemos carregar o programa RISC-V do BubbleSort e montá-lo. O código fonte desse programa é apresentado no Figura 1.

Iniciamos o simulador EMCS selecionando 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 2. Para que os acessos à memória realizados pelo programa *BubbleSort* durante a sua execução sejam usados pelo EMCS na simulação das memórias cache, precisamos selecionar o botão *Connect to Program* na janela do EMCS.

```
                                    #---------------------------------------
            .text                   # Área de código
                                    #---------------------------------------
main:                               # Inicialização
            addi  s0, zero, 64      # n = 64
            la    s1, vetor         # s1 = endereço inicial de vetor na memória
            addi  s2, zero, 1       # trocou = 1
            addi  s3, s0, -1        # limite = n-1
while:      slt   t0, zero, s3      # limite > 0 ?
            beq   t0, zero, fora_while # Se limite <= 0, desvia p/ fora do while
            beq   s2, zero, fora_while # Se trocou == false, desvia p/ fora do while
            add   s2, zero, zero    # trocou = 0
            add   s4, zero, zero    # i = 0
for:        slt   t0, s4, s3        # i < limite ?
            beq   t0, zero, fora_for   # Se i >= limite, desvia para fora do for
            add   t1, s4, s4        # t1 = i * 4
            add   t1, t1, t1
            add   t1, s1, t1        # t1 = endereço de vetor[i] na memória
            lw    t2, 0 (t1)        # t2 = valor de vetor[i] lido da memória
            lw    t3, 4 (t1)        # t3 = valor de vetor[i+1] lido da memória
            slt   t0, t3, t2        # vetor[i] < vetor[i+1] ?
            beq   t0, zero, fora_if    # Se vetor[i] <= vetor[i+1], desvia p/ fora
            do if
                                    # Troca vetor[i] e vetor[i+1]
            sw    t3, 0 (t1)        # vetor[i] = t3
            sw    t2, 4 (t1)        # vetor[i+1] = t2
            addi  s2, zero, 1       # trocou = 1
fora_if:    addi  s4, s4, 1         # i++
            j     for               # Desvia para início do for
fora_for:   addi  s3, s3, -1        # limite--
            j     while             # Desvia para início do while
fora_while: addi  a7, zero, 10      # Chamada ao sistema: encerrar programa
            ecall
                                    #---------------------------------------
            .data                   # Área de dados
                                    #---------------------------------------
                                    # Variáveis e estruturas de dados do programa
vetor:      .word 63 62 61 60 59 58 57 56 55 54 53 52 51 40 49 48 47 46 45 44 43 42
41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14
13 12 11 10 9 8 7 6 5 4 3 2 1 0 # Vetor a ser ordenado
```

Figura 1: Programa BubbleSort em linguagem de montagem RISC-V

Em seguida, selecionamos a organização das caches L1 e L2 que desejamos simular. Para fins de experimentação, utilizamos a configuração descrita na Tabela 1 e ilustrada na Figura 2. A configuração escolhida é aplicada ao selecionarmos o botão *Apply*.

Executamos então o programa *BubbleSort*, através dos botões *Run* ou *Step* do RARS. Podemos acompanhar essa execução na aba *Access & Performance* do EMCS, onde visualizamos os resultados da simulação das caches. Essa aba, mostrada na Figura 3, 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 o aluno exporte essa tabela como uma planilha no formato CSV, caso ele deseje analisar as informações.

	Cache L1	Cache L2
Número total de blocos	8	16
Tamanho do bloco	1	2
Associatividade	1	1
Política de substituição	–	–
Tempo de acesso (ciclos)	1	10
	Memória principal	
Tempo de acesso (ciclos)	100	

Tabela 1: Configuração da hierarquia de memórias utilizada no experimento de simulação do programa *BubbleSort*

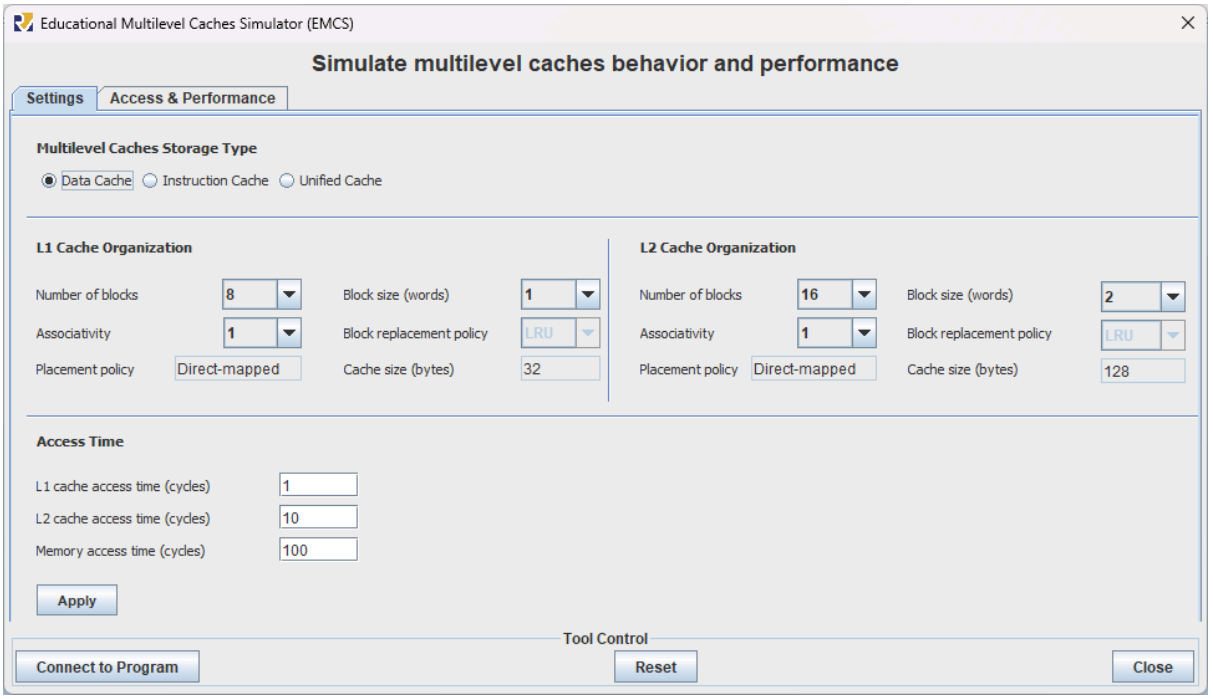

Figura 2: Configuração das memórias cache L1 e L2

A aba *Access & Performance* também apresenta os resultados de desempenho da simulação. A Figura 3 mostra os resultados da simulação do BubbleSort para as caches da organização indicada na Tabela 1. 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).

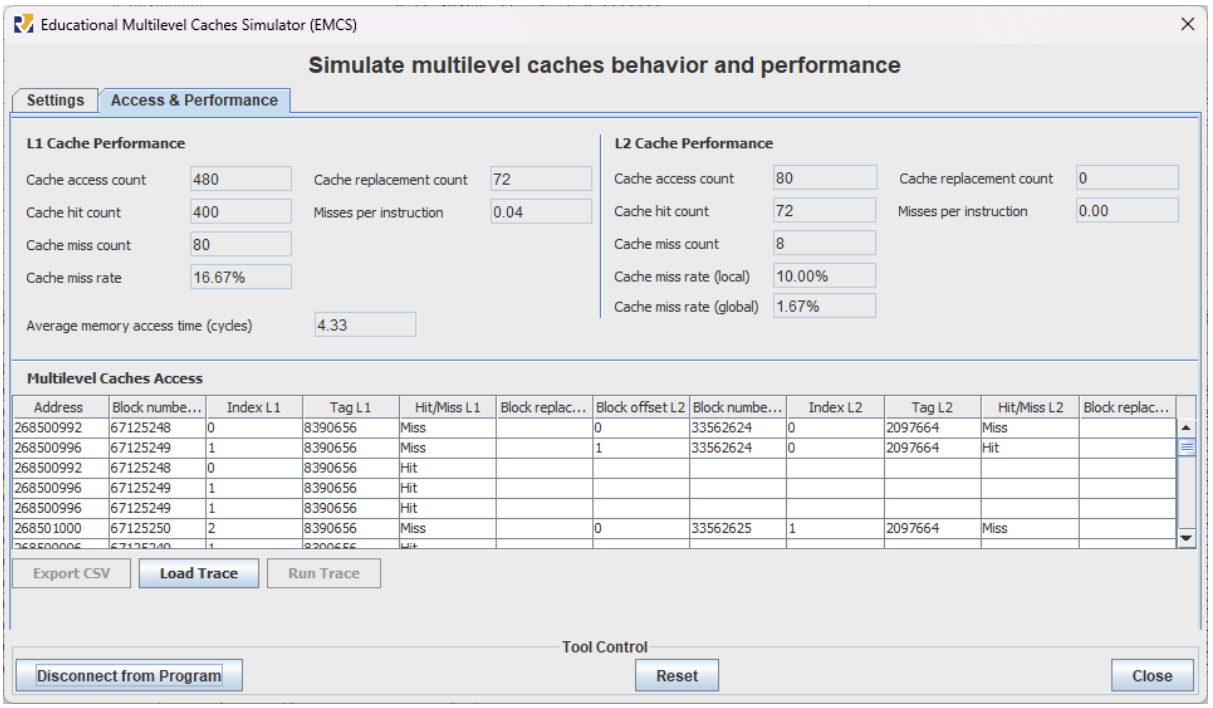

Figura 3: Acessos às caches L1 e L2 e resultados de desempenho da simulação do programa *BubbleSort*

Obtemos os seguintes resultados na simulação:

- Número total de acessos na L1: 480
- Número de falhas na L1: 80
- Número de substituições na L1: 72
- Taxa de falhas da L1: 16,67%
- Falhas por instrução da L1: 0,04
- Número total de acessos na L2: 80
- Número de falhas na L2: 8
- Número de substituições na L2: 0
- Taxa de falhas local da L2: 10,00%
- Taxa de falhas global da L2: 1,67%
- Falhas por instrução da L2: 0,00
- Tempo efetivo de acesso à memória (*Average Memory Access Time* - AMAT): 4,33 ciclos

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

1. Qual a diferença entre a taxa de falhas e a razão falhas por instrução?

2. Qual é o efeito do tamanho do bloco da cache L1 na taxa de falhas dessa cache? E da cache L2?
3. Qual é o efeito da associatividade das caches nas taxas de falhas?
4. Qual é o efeito do número total de blocos das caches nas taxas de falhas?
5. O programa apresenta localidade temporal e/ou espacial?
6. As falhas na cache L1 são predominantemente compulsórias, de capacidade ou de conflito? E na cache L2?

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

2. Experimento de Simulação de um *Trace* de Endereços

Em um segundo experimento, usamos a ferramenta EMCS para a simulação de uma hierarquia de memórias com dois níveis de cache, seguidas pela memória principal, 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, utilizamos o EMCS 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. Podemos coletar esses acessos utilizando outras ferramentas ou mecanismos, como por exemplo outros simuladores ou através da instrumentação do código. Assim, com a utilização de arquivos de *trace* é possível usarmos o EMCS para 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 4.

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

Figura 4. Exemplo de um arquivo de trace de endereços de memória.

Neste experimento utilizamos um *trace* de memória, usado em [1], contendo 910.237 acessos a dados, correspondentes à execução do programa *Susan Corners* do *benchmark* MiBench. Novamente, iniciamos selecionando a organização

da hierarquia de memórias que desejamos simular, na aba *Settings* do EMCS. A Tabela 2 e a Figura 5 mostram a configuração escolhida para esse experimento.

	Cache L1	**Cache L2**
Número total de blocos	128	512
Tamanho do bloco	4	8
Associatividade	1	1
Política de substituição	–	–
Tempo de acesso (ciclos)	1	10
	Memória principal	
Tempo de acesso (ciclos)	100	

Tabela 2: Configuração da hierarquia de memórias utilizada no experimento de simulação do *trace* de endereços do programa *Susan Corners*

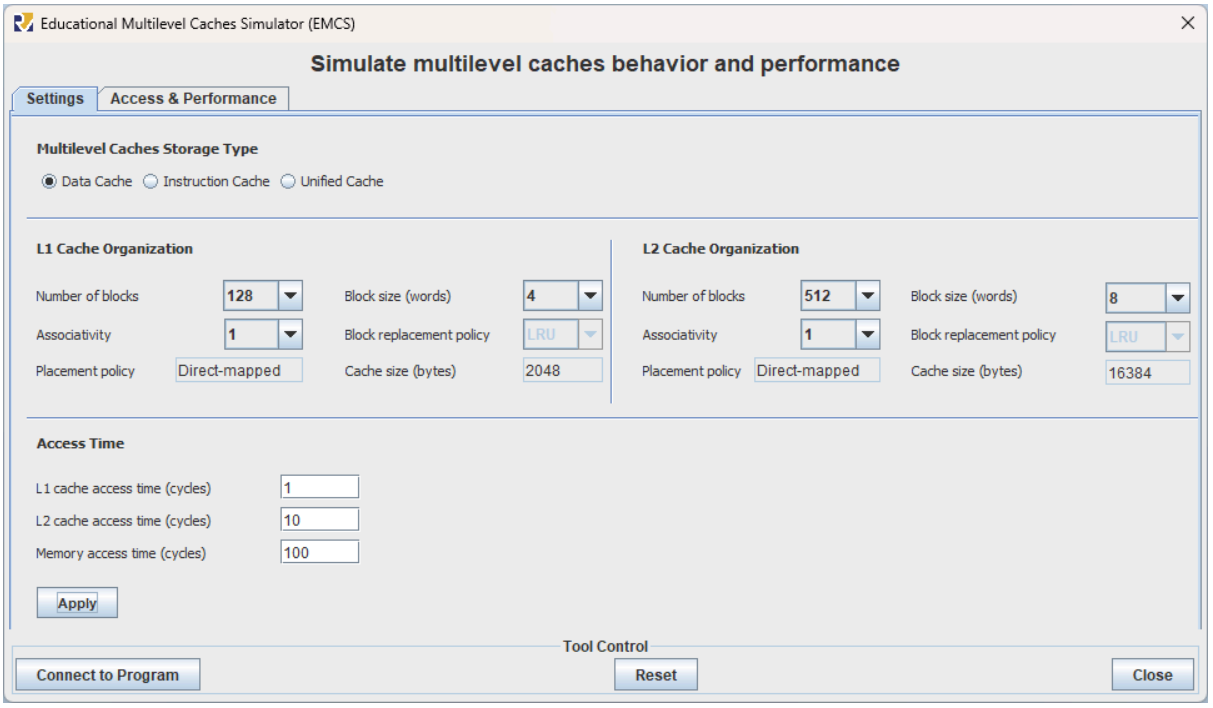

Figura 5: Configuração das memórias cache L1 e L2 a serem simuladas

Em seguida, na aba *Access & Performance* do EMCS, usamos o botão *Load Trace*, para selecionar o arquivo com o *trace* de endereços desejado. Após a carga do *trace*, aparece uma janela com uma mensagem de confirmação, como mostra a Figura 6. 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.

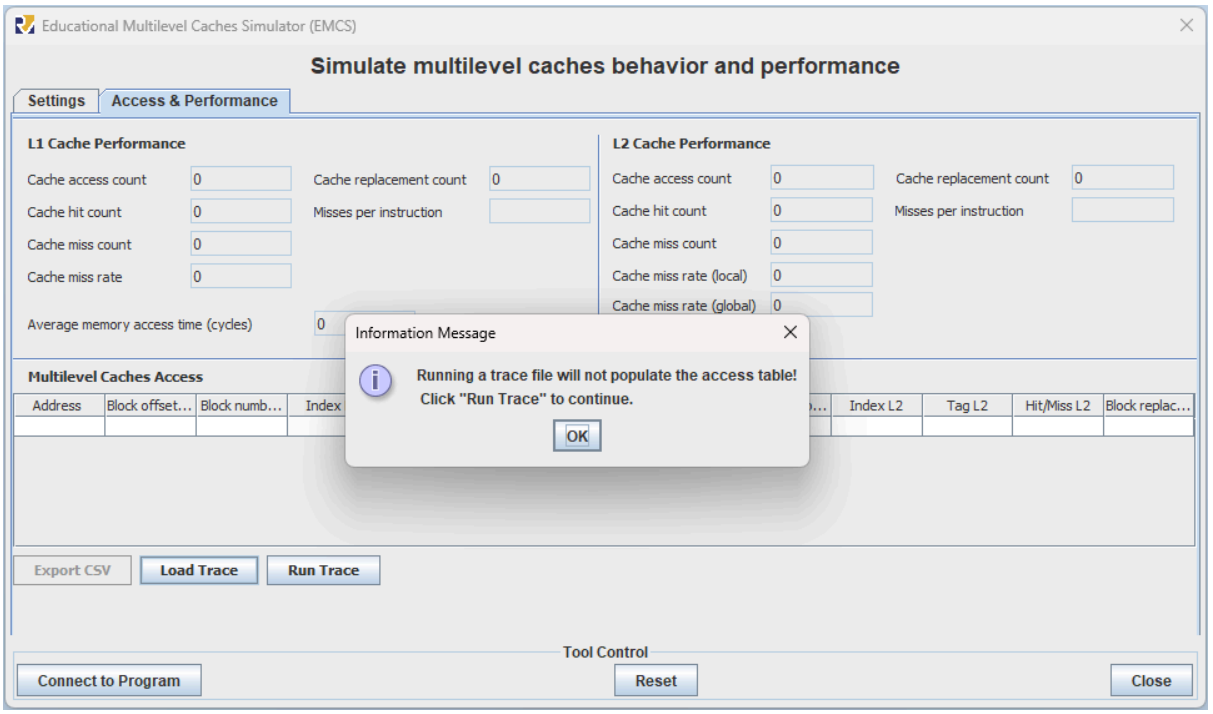

Figura 6: Mensagem de confirmação após a carga do *trace* de endereços

Ao selecionarmos o botão *Run Trace,* a ferramenta começa a simular o *trace* de endereços, porém não atualiza 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. Quando a execução termina, os resultados de desempenho são exibidos, como mostrado na Figura 7.

Como realizamos a simulação de um trace de endereços e não de um programa RISC-V, o EMCS não possui a informação de quantas instruções foram executadas. Portanto, ele não calcula as razões *Falhas por instrução* das caches L1 e L2. Obtemos os seguintes resultados de desempenho na simulação:

- Número total de acessos na L1: 910.237
- Número de falhas na L1: 20.147
- Número de substituições na L1: 20.019
- Taxa de falhas da L1: 2,21%
- Número total de acessos na L2: 20.147
- Número de falhas na L2: 5.057
- Número de substituições na L2: 4.545
- Taxa de falhas local da L2: 25,10%
- Taxa de falhas global da L2: 0,56%
- Tempo efetivo de acesso à memória (*Average Memory Access Time* - AMAT): 1,78 ciclos

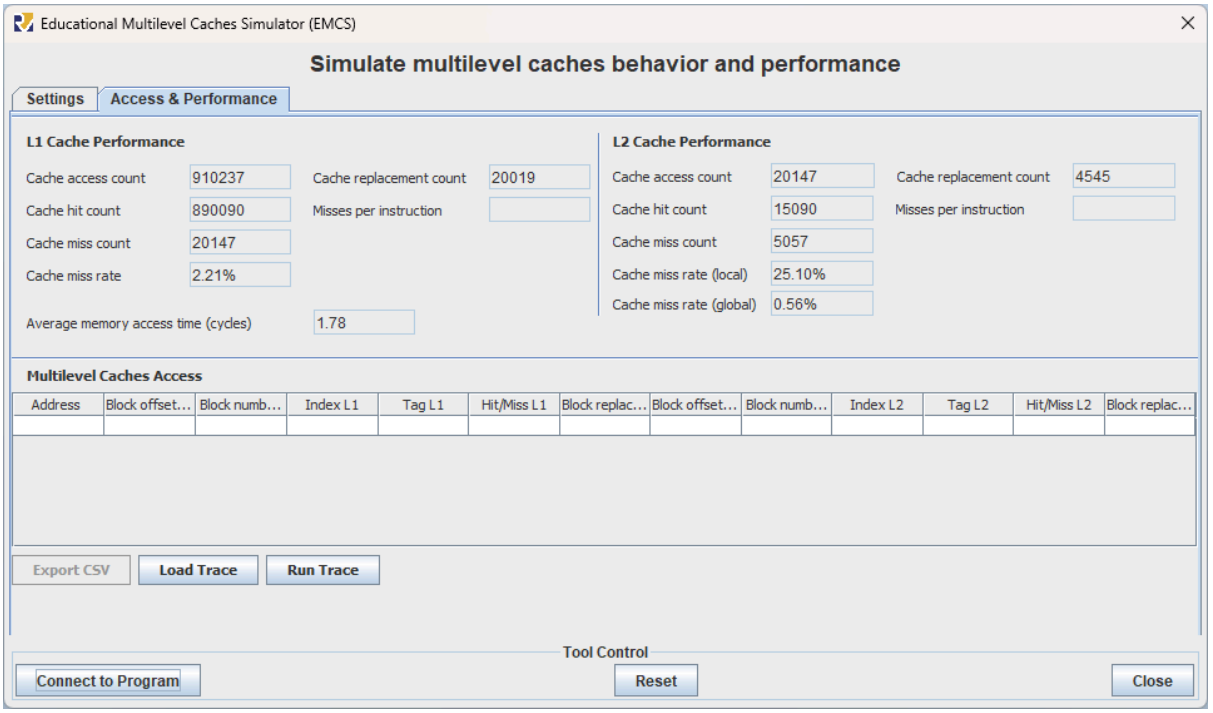

Figura 7: Resultados de desempenho da simulação do *trace* de endereços do programa *Susan Corners*

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

1. Como as taxas de falhas local e global da cache L2 são calculadas?
2. Como o tempo efetivo de acesso à memória (*Average Memory Access Time* - AMAT) é calculado?
3. Qual seria o AMAT se houvesse apenas a cache L1, sem a cache L2?
4. Como pode ser realizada uma comparação entre uma organização com caches unificadas e uma com caches *split* (separadas para instruções e dados)?

O aluno pode realizar a mesma análise com outros *traces* e/ou configurações de caches, permitindo uma avaliação de desempenho mais detalhada da hierarquia de memórias.

3. Conclusão

A ferramenta EMCS se destaca como um instrumento de ensino versátil para o aprendizado sobre hierarquia de memórias e memórias cache nas disciplinas de nas disciplinas de Arquitetura e Organização de Computadores. Os estudantes podem utilizá-lo tanto em atividades práticas em laboratório quanto em estudos individuais, explorando diversos cenários através de simulações.

4. Bibliografia

1. Lima, D. P. e 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.
2. Silva, Nikolas Cavalheiro Gonçalves e Moraes, Samuel Willian de Souza, Simulador de Caches Multiníveis para o Processador RISC-V, Trabalho de Conclusão de Curso, Faculdade de Computação, 2025.