PES University Electronic City Campus

Hosur Rd, Konappana Agrahara, Electronic City, Bengaluru, Karnataka
560100

A Project Report on

"CUDA PROGRAM AND CACHE CONFIGURATION"

Submitted by:

- **MADHU L (PES2UG20EC802)**
- **ADITHYA NAIR (PES2UG20EC008)**

Under the guidance of:

PROF. MAHESH AWATI
Associate Professor,
Department of Electronics and Communication Engineering
PES UNIVERSITY
Electronic City, Bangalore-560100

2022 - 2023

ASSIGNMENT 1: CUDA PROGRAM

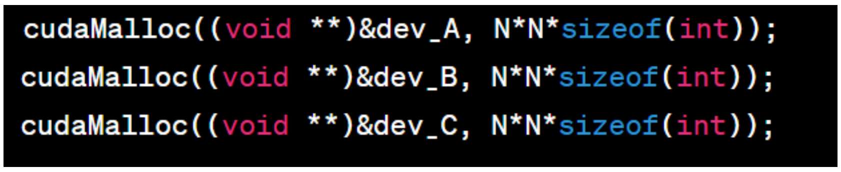
N X N MATRIX MULTIPLICATION

CODE:

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>

#define N 2

__global__ void matrixMult(int* A, int* B, int* C, int n)
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    int Pvalue = 0;
    for (int k = 0; k < n; k++)
    {
        Pvalue += A[i * n + k] * B[k * n + j];
    }
    C[i * n + j] = Pvalue;
}

int main()
{
    int A[N * N] = { 6, 7,8, 2 };
    int B[N * N] = { 5, 6, 7, 8 };
    int C[N * N] = { 0, 0, 0, 0 };
    int* dev_A = NULL, * dev_B = NULL, * dev_C = NULL;

    cudaError_t err = cudaMalloc((void**)&dev_A, N * N *
sizeof(int));
    if (err != cudaSuccess)
    {
        printf("Error allocating memory for dev_A: %s\n",
cudaGetErrorString(err));
        return -1;
    }

    err = cudaMalloc((void**)&dev_B, N * N * sizeof(int));
    if (err != cudaSuccess)
    {
        printf("Error allocating memory for dev_B: %s\n",
cudaGetErrorString(err));
```

```
        cudaFree(dev_A);
        return -1;
    }

    err = cudaMalloc((void**)&dev_C, N * N * sizeof(int));
    if (err != cudaSuccess)
    {
        printf("Error allocating memory for dev_C: %s\n",
cudaGetErrorString(err));
        cudaFree(dev_A);
        cudaFree(dev_B);
        return -1;
    }

    err = cudaMemcpy(dev_A, A, N * N * sizeof(int),
cudaMemcpyHostToDevice);
    if (err != cudaSuccess)
    {
        printf("Error copying A to device: %s\n",
cudaGetErrorString(err));
        cudaFree(dev_A);
        cudaFree(dev_B);
        cudaFree(dev_C);
        return -1;
    }

    err = cudaMemcpy(dev_B, B, N * N * sizeof(int),
cudaMemcpyHostToDevice);
    if (err != cudaSuccess)
    {
        printf("Error copying B to device: %s\n",
cudaGetErrorString(err));
        cudaFree(dev_A);
        cudaFree(dev_B);
        cudaFree(dev_C);
        return -1;
    }

    dim3 threadsPerBlock(N, N);
    matrixMult << <1, threadsPerBlock >> > (dev_A, dev_B, dev_C,
N);

    err = cudaMemcpy(C, dev_C, N * N * sizeof(int),
cudaMemcpyDeviceToHost);
    if (err != cudaSuccess)
    {
        printf("Error copying C from device: %s\n",
cudaGetErrorString(err));
        cudaFree(dev_A);
        cudaFree(dev_B);
        cudaFree(dev_C);
        return -1;
    }

    cudaFree(dev_A);
    cudaFree(dev_B);
    cudaFree(dev_C);

    printf("Matrix A:\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", A[i * N + j]);
        }
        printf("\n");
    }

    printf("Matrix B:\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", B[i * N + j]);
        }
        printf("\n");
    }
    printf("Resultant Matrix C:\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", C[i * N + j]);
        }
        printf("\n");
    }

    return 0;
}
```

NOTE:



```
cudaMalloc((void **)&dev_A, N*N*sizeof(int));
cudaMalloc((void **)&dev_B, N*N*sizeof(int));
cudaMalloc((void **)&dev_C, N*N*sizeof(int));
```

ALLOCATE MEMORY ON GPU FOR dev_A,dev_B,dev_C using cudaMalloc

```
cudaMemcpy(dev_A, A, N*N*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dev_B, B, N*N*sizeof(int), cudaMemcpyHostToDevice);
```

COPIES THE MATRICES FROM THE HOST TO GPU USING cudaMemcpy( ).

```
dim3 threadsPerBlock(N,N);
matrixMult<<<1,threadsPerBlock>>>(dev_A, dev_B, dev_C);
```

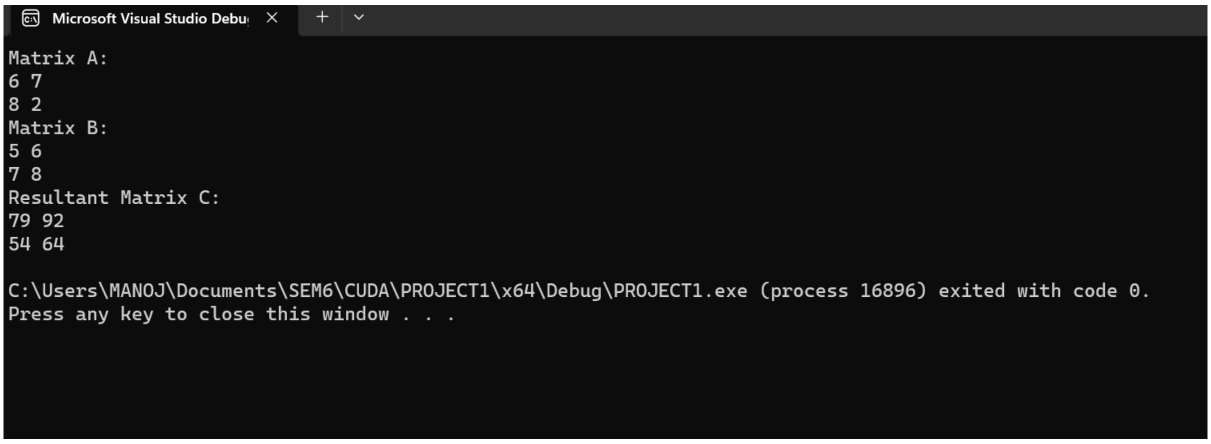
DEFINES THE NUMBER OF THREADS PER BLOCK AS 2D BLOCK WITH DIM NxN

```
cudaMemcpy(C, dev_C, N*N*sizeof(int), cudaMemcpyDeviceToHost);
```

COPIES MATRIX C FROM GPU TO HOST CPU USING cudaMemcpy( ).

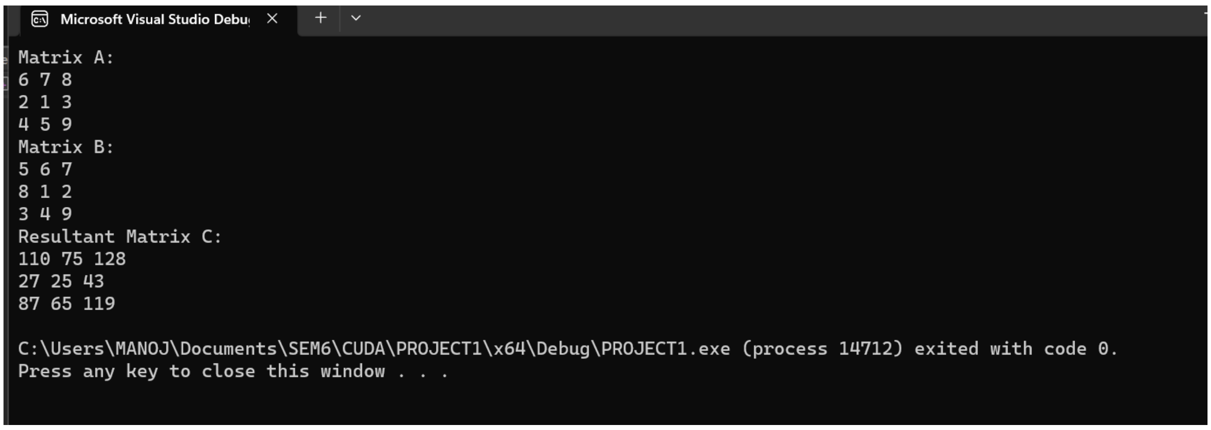
OUTPUT:

1.



```
Matrix A:
6 7
8 2
Matrix B:
5 6
7 8
Resultant Matrix C:
79 92
54 64

C:\Users\MANOJ\Documents\SEM6\CUDA\PROJECT1\x64\Debug\PROJECT1.exe (process 16896) exited with code 0.
Press any key to close this window . . .
```

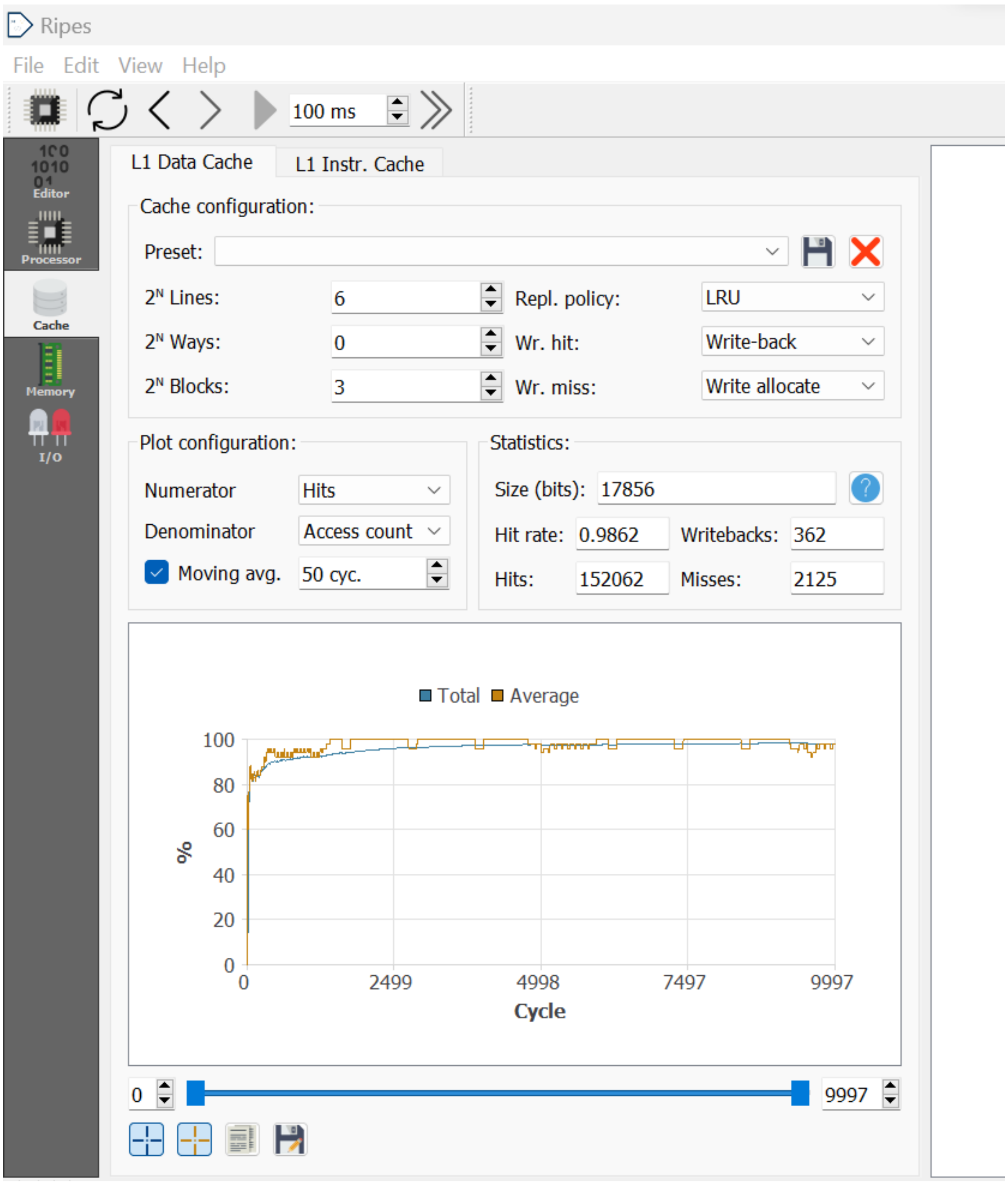
2.



```
Matrix A:
6 7 8
2 1 3
4 5 9
Matrix B:
5 6 7
8 1 2
3 4 9
Resultant Matrix C:
110 75 128
27 25 43
87 65 119

C:\Users\MANOJ\Documents\SEM6\CUDA\PROJECT1\x64\Debug\PROJECT1.exe (process 14712) exited with code 0.
Press any key to close this window . . .
```

ASSIGNMENT 2: CACHE CONFIGURATION

1. Configure a 64-entry 8-word direct mapped cache.



Cache configuration:

Preset:

2^N Lines: 6
2^N Ways: 0
2^N Blocks: 3
Repl. policy: LRU
Wr. hit: Write-back
Wr. miss: Write allocate

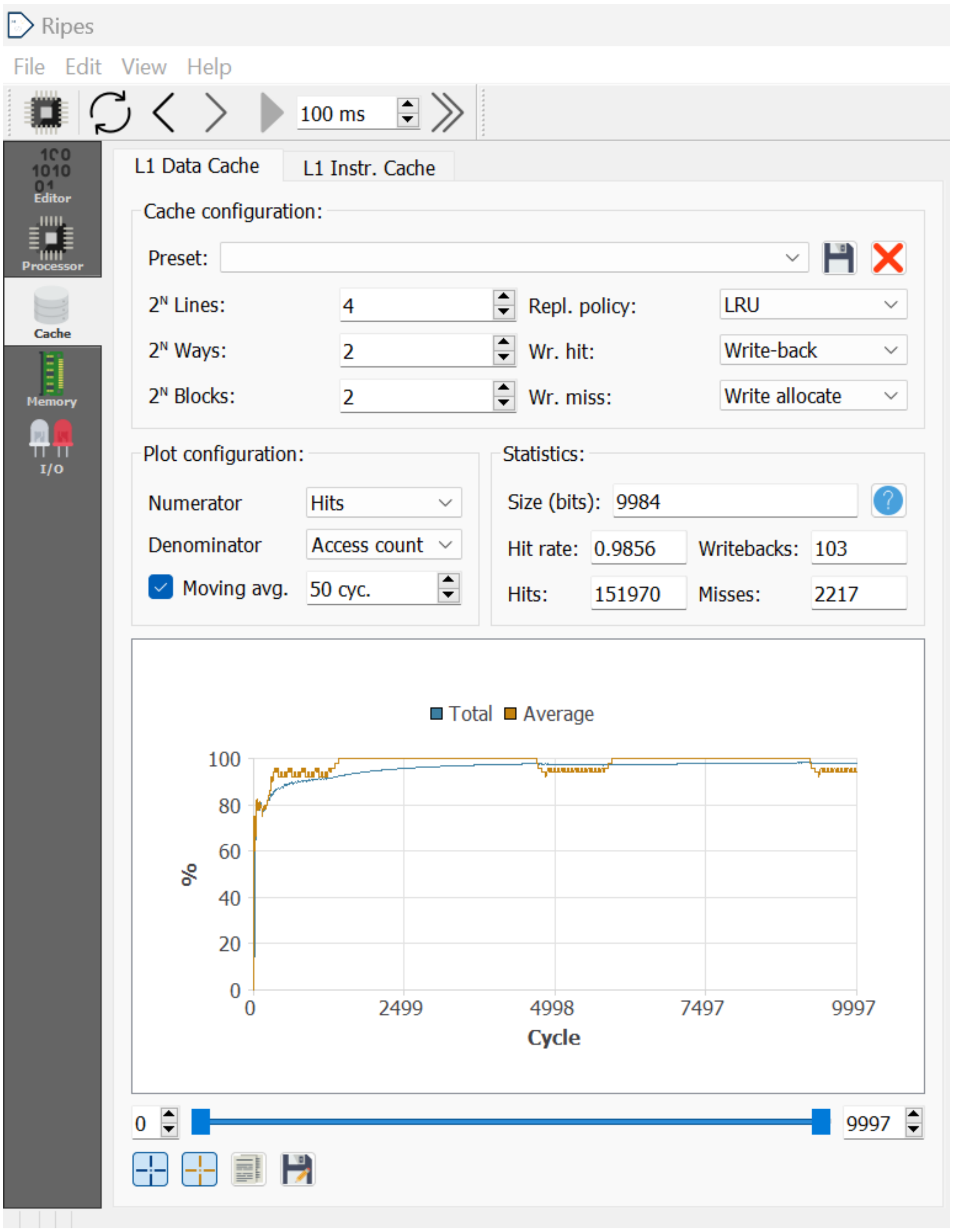
Plot configuration:

Numerator: Hits
Denominator: Access count
✓ Moving avg. 50 cyc.

Statistics:

Size (bits): 17856
Hit rate: 0.9862
Writebacks: 362
Hits: 152062
Misses: 2125

2. Configure a 16-entry 4-word 4-way set-associative cache.



Cache configuration:

Preset:

2^N Lines: 4
Repl. policy: LRU
2^N Ways: 2
Wr. hit: Write-back
2^N Blocks: 2
Wr. miss: Write allocate

Plot configuration:

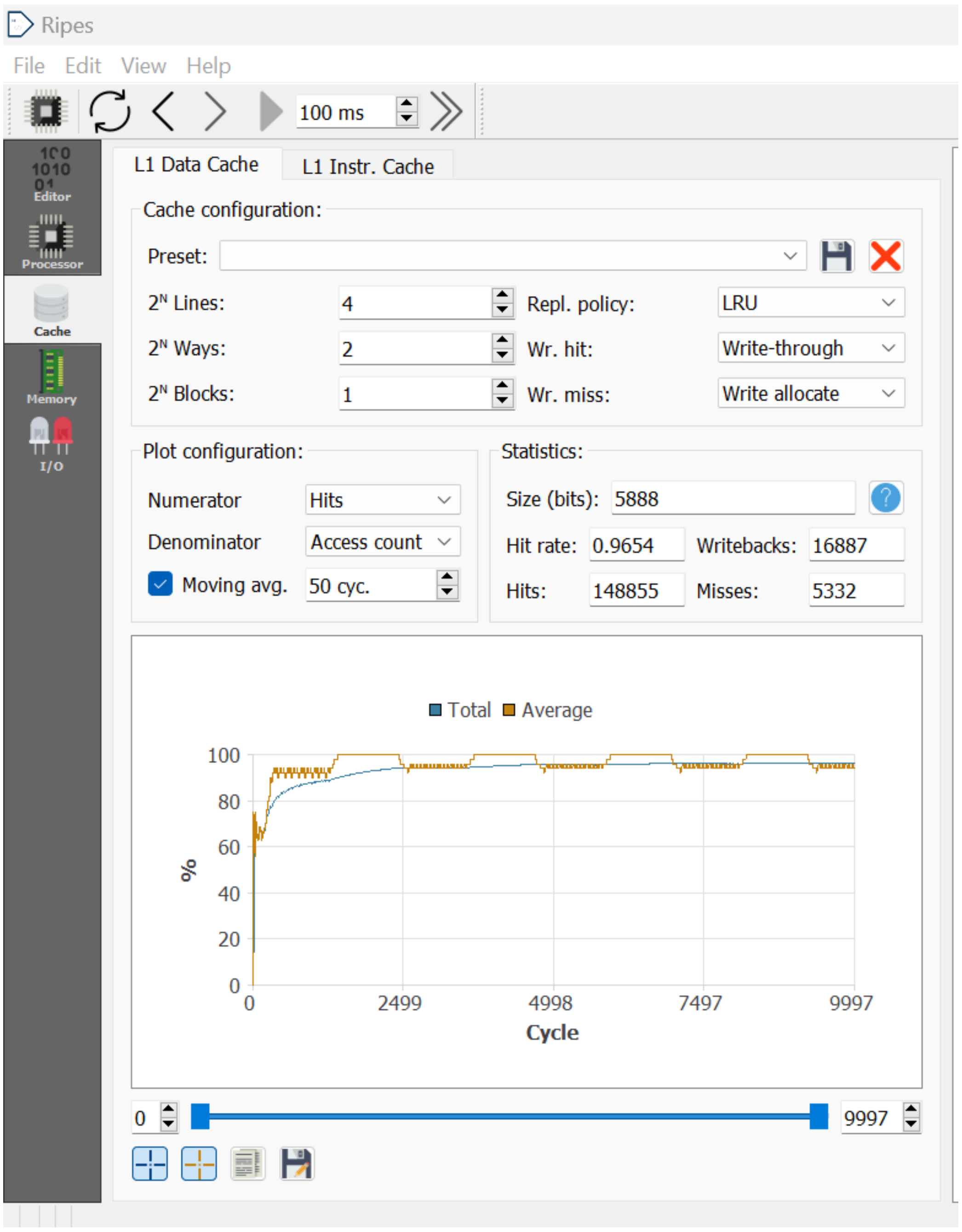
Numerator: Hits
Denominator: Access count
✓ Moving avg.: 50 cyc.

Statistics:

Size (bits): 9984
Hit rate: 0.9856
Writebacks: 103
Hits: 151970
Misses: 2217

0 — 9997

3. Configure a 16-entry 2-word 4-way set-associative cache with write-through.



Cache configuration:

Preset:			
2^N Lines:	4	Repl. policy:	LRU
2^N Ways:	2	Wr. hit:	Write-through
2^N Blocks:	1	Wr. miss:	Write allocate

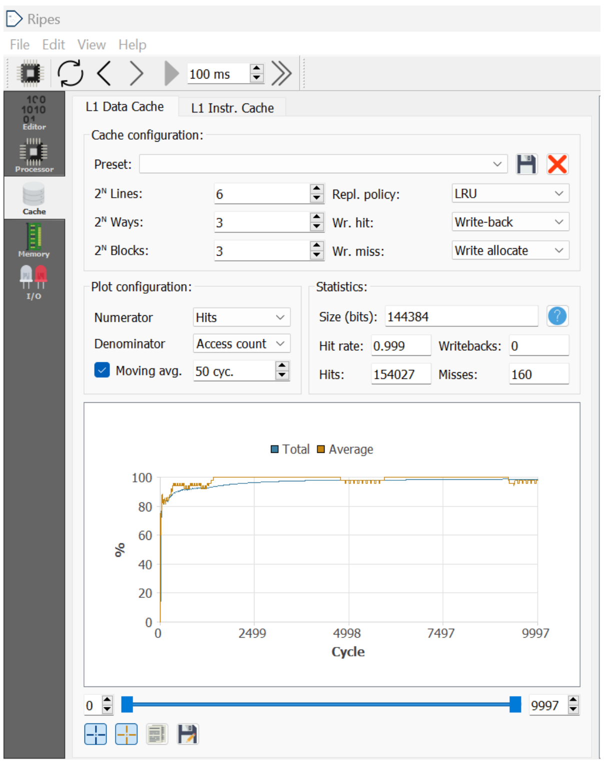
Plot configuration:

Numerator	Hits
Denominator	Access count
✓ Moving avg.	50 cyc.

Statistics:

Size (bits):	5888		
Hit rate:	0.9654	Writebacks:	16887
Hits:	148855	Misses:	5332

4. Configure a 64-entry 8-word fully associative cache with Least Recently Used replacement policy and report the numbers. Change the replacement policy to Random and report the numbers for the same cache.
   1. LRU

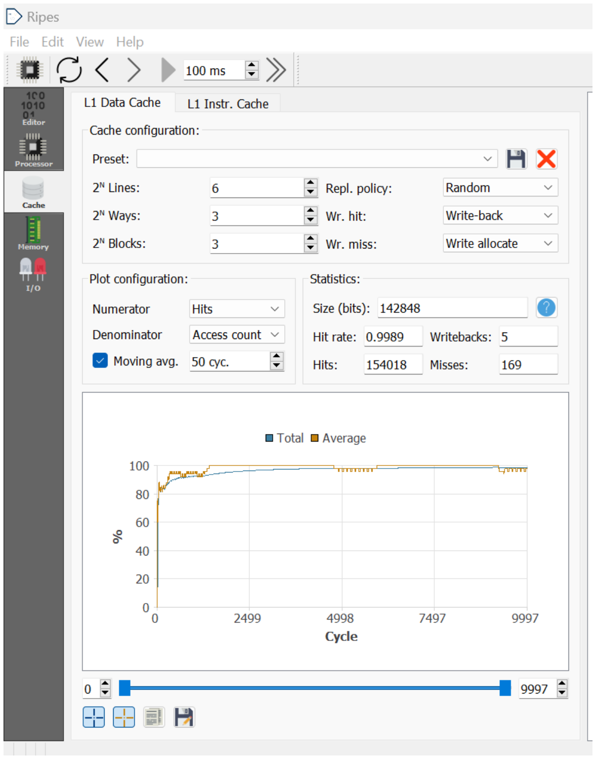


Cache configuration:
Preset:
2^N Lines: 6 Repl. policy: LRU
2^N Ways: 3 Wr. hit: Write-back
2^N Blocks: 3 Wr. miss: Write allocate

Plot configuration:
Numerator: Hits
Denominator: Access count
Moving avg. 50 cyc.

Statistics:
Size (bits): 144384
Hit rate: 0.999 Writebacks: 0
Hits: 154027 Misses: 160

2.RANDOM

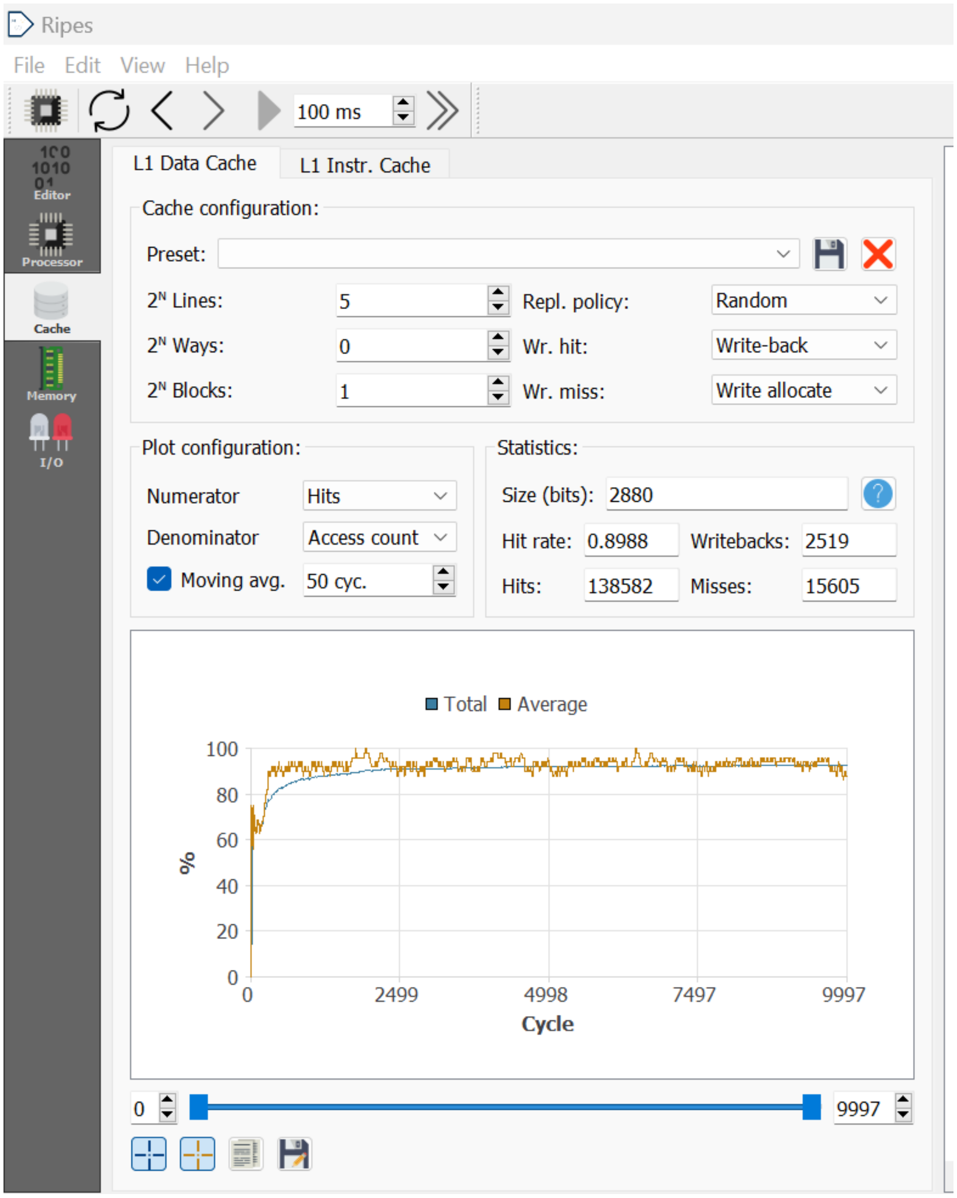


Cache configuration:
Preset:
2^N Lines: 6 Repl. policy: Random
2^N Ways: 3 Wr. hit: Write-back
2^N Blocks: 3 Wr. miss: Write allocate

Plot configuration:
Numerator: Hits
Denominator: Access count
Moving avg. 50 cyc.

Statistics:
Size (bits): 142848
Hit rate: 0.9989 Writebacks: 5
Hits: 154018 Misses: 169

5. Configure a 32-entry 2-word direct-mapped cache and plot a graph using the plot configuration with numerator as Hits and denominator as Access count. Explain why the number of hits increases and decreases back down before increasing again.



Cache configuration:

Preset:

2^N Lines:	5	Repl. policy:	Random
2^N Ways:	0	Wr. hit:	Write-back
2^N Blocks:	1	Wr. miss:	Write allocate

Plot configuration:

Numerator	Hits
Denominator	Access count
✓ Moving avg.	50 cyc.

Statistics:

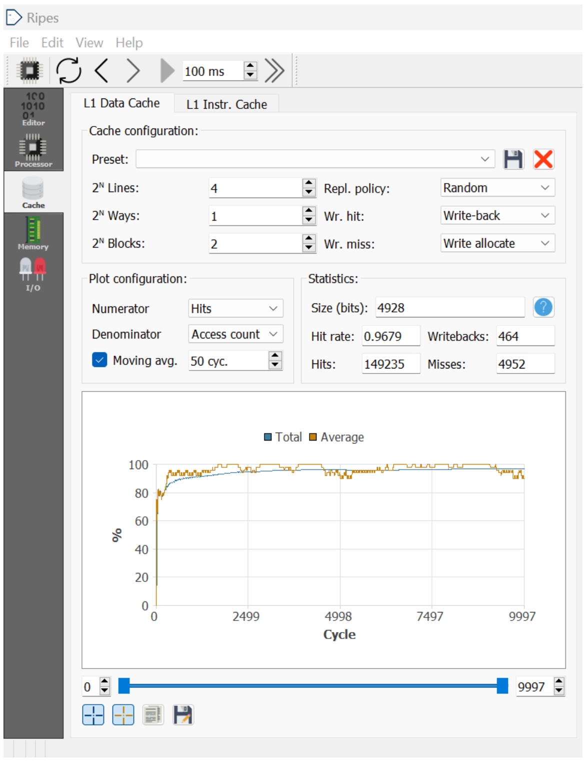
Size (bits):	2880		
Hit rate:	0.8988	Writebacks:	2519
Hits:	138582	Misses:	15605

0 — 9997

In a 32-entry 2-word direct-mapped cache, the number of hits increases and decreases back down before increasing again because of cache conflicts. When a program accesses memory blocks that map to the same cache line, the cache can only hold one of the blocks at a time, so it will evict one and fetch the other, resulting in a cache miss. As the program continues to access these memory blocks, the cache will alternate between holding one block and the other, resulting in alternating hits and misses on the cache line. Therefore, the number of hits may increase up to a maximum of 16 per cache line (since there are only 16 cache lines available for each set of two memory blocks that map to the same cache line), but will eventually decrease due to evictions and cache misses, before increasing again when the program accesses the memory blocks that were evicted earlier.

6. Configure a 16-entry 4-word 2-way set-associative cache with write-back and write allocate and report the numbers. For the same configuration of cache, use write-through and no write allocate and report the numbers. Explain the differences between the two cache configurations.

1. WITH WRITE BACK AND WRITE ALLOCATE



Cache configuration:

Preset:

2ᴺ Lines: 4
2ᴺ Ways: 1
2ᴺ Blocks: 2
Repl. policy: Random
Wr. hit: Write-back
Wr. miss: Write allocate

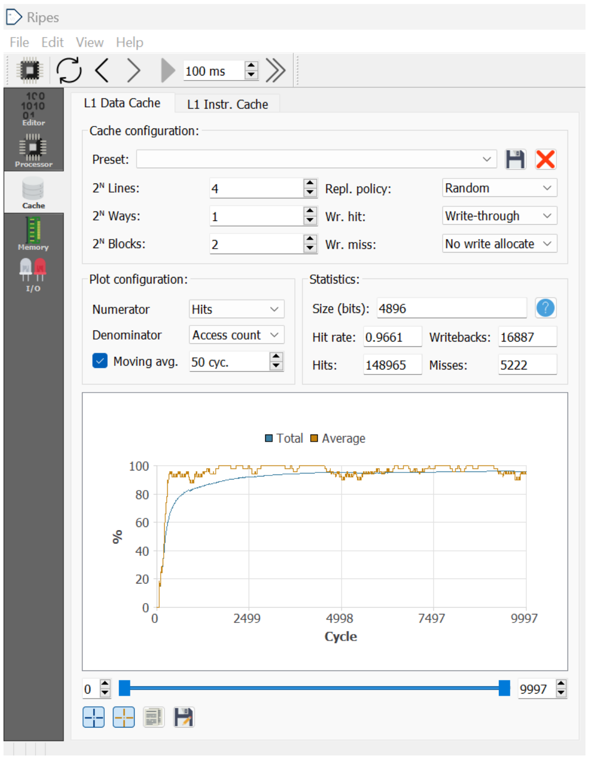
Plot configuration:

Numerator: Hits
Denominator: Access count
Moving avg.: 50 cyc.

Statistics:

Size (bits): 4928
Hit rate: 0.9679
Writebacks: 464
Hits: 149235
Misses: 4952

2.WITH WRITE THROUGH AND NO WRITE ALLOCATE



Cache configuration:

Preset:

2ᴺ Lines: 4
2ᴺ Ways: 1
2ᴺ Blocks: 2
Repl. policy: Random
Wr. hit: Write-through
Wr. miss: No write allocate

Plot configuration:

Numerator: Hits
Denominator: Access count
Moving avg.: 50 cyc.

Statistics:

Size (bits): 4896
Hit rate: 0.9661
Writebacks: 16887
Hits: 148965
Misses: 5222

Cache with write-back and write-allocate policies, write operations are first written to the cache and marked as "dirty", but not immediately written to main memory. When a cache line containing dirty data is evicted from the cache, the dirty data is then written back to main memory. Write-allocate means that when a write operation misses in the cache, the entire cache block is brought into the cache before the write is performed.

In a Cache with write-through and no-write-allocate policies, write operations are immediately written both to the cache and main memory. No-write-allocate means that when a write operation misses in the cache, the write is directly performed in main memory without bringing the block into the cache.