# [Microprocessor Applications]
# Lab 5: Cache Optimization

Chester Sungchung Park (박성정)

SoC Design Lab, Konkuk University

Webpage: http://soclab.konkuk.ac.kr

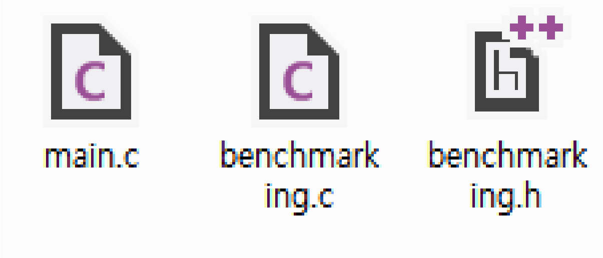
# Outline

- ❑ Running C applications
- ❑ Optimizing C applications

# Running C Applications

❑ Create a new application project

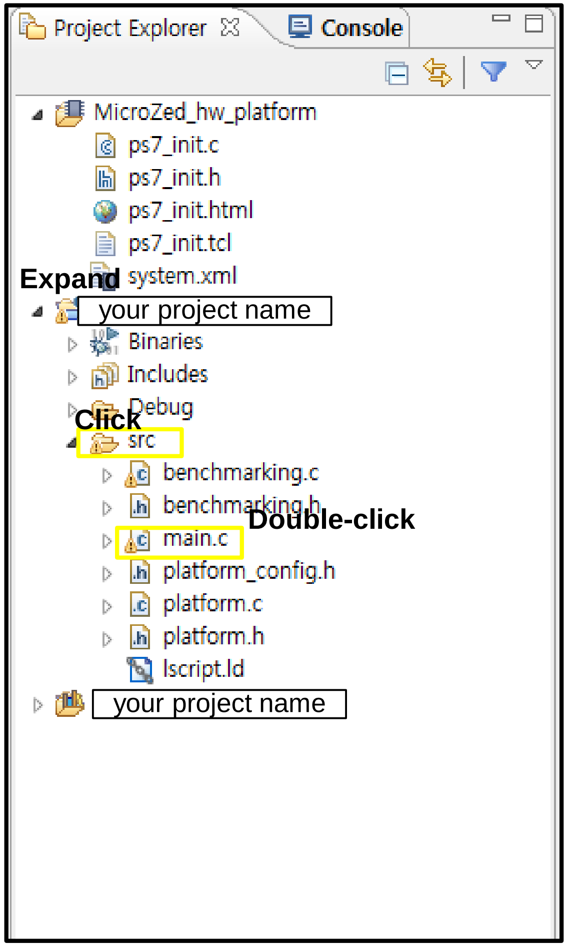
- Follow pp. 3~7 of the following lab workbook: **Lab_MP2022_2_work_r1.pdf**
  - ✓ Type the project name
  - ✓ Add the source files attached below to the project.
    - main.c, benchmarking.c, benchmarking.h



# Running C Applications

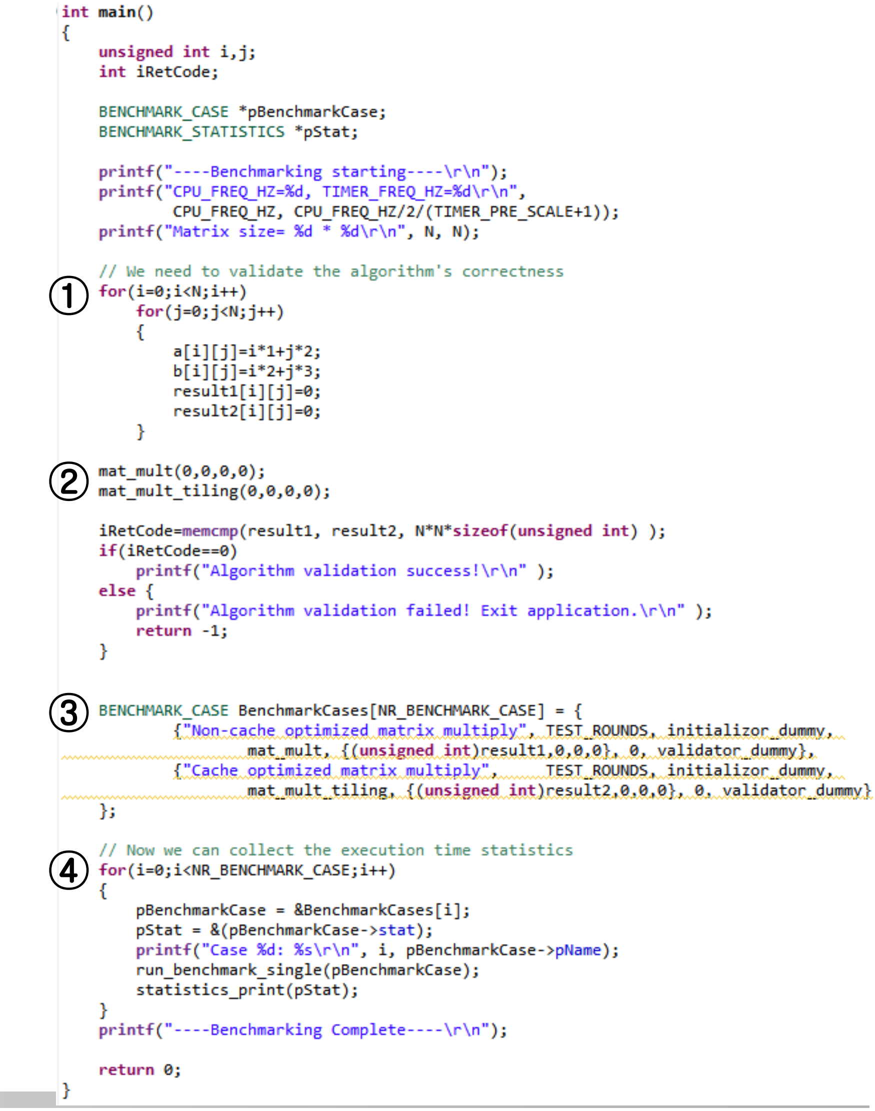
❑ Check the source codes

- Expand ***'(your project name)'*** to see all of the source files that are added to the project by clicking the ***'src'*** icon.
- Double-click each of the file names to open it.



# Running C Applications

- ❑ Review the source codes: ***'main()'***
  1. ① Initializes the arguments of the functions
  2. ② Calls the functions to compare the results
  3. ③ Set the benchmark to compare the execution times
  4. ④ Run the benchmark and measure the execution times

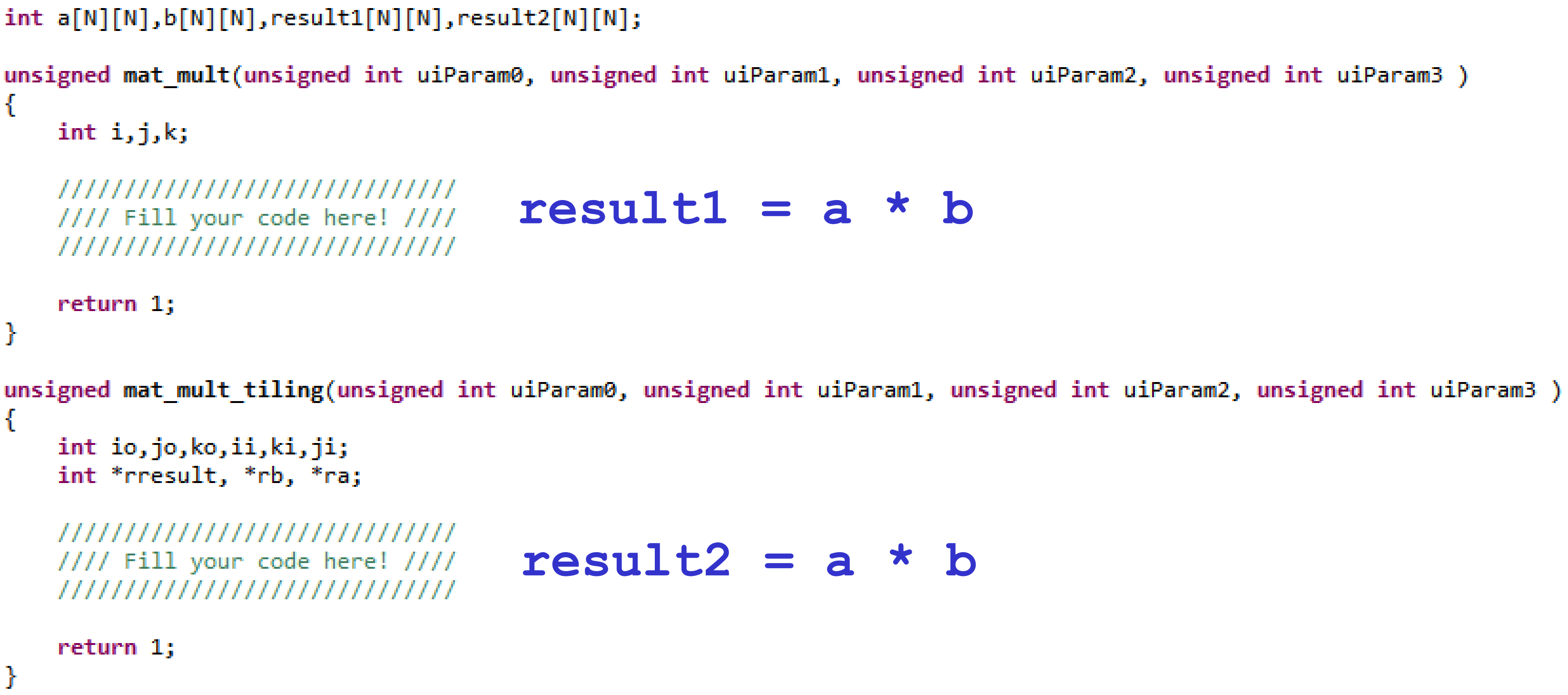


```c
int main()
{
    unsigned int i,j;
    int iRetCode;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n",
            CPU_FREQ_HZ, CPU_FREQ_HZ/2/(TIMER_PRE_SCALE+1));
    printf("Matrix size= %d * %d\r\n", N, N);

    // We need to validate the algorithm's correctness
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        {
            a[i][j]=i*1+j*2;
            b[i][j]=i*2+j*3;
            result1[i][j]=0;
            result2[i][j]=0;
        }

    mat_mult(0,0,0,0);
    mat_mult_tiling(0,0,0,0);

    iRetCode=memcmp(result1, result2, N*N*sizeof(unsigned int) );
    if(iRetCode==0)
        printf("Algorithm validation success!\r\n" );
    else {
        printf("Algorithm validation failed! Exit application.\r\n" );
        return -1;
    }


    BENCHMARK_CASE BenchmarkCases[NR_BENCHMARK_CASE] = {
            {"Non-cache optimized matrix multiply", TEST_ROUNDS, initializor_dummy,
                    mat_mult, {(unsigned int)result1,0,0,0}, 0, validator_dummy},
            {"Cache optimized matrix multiply",     TEST_ROUNDS, initializor_dummy,
                    mat_mult_tiling, {(unsigned int)result2,0,0,0}, 0, validator_dummy}
    };

    // Now we can collect the execution time statistics
    for(i=0;i<NR_BENCHMARK_CASE;i++)
    {
        pBenchmarkCase = &BenchmarkCases[i];
        pStat = &(pBenchmarkCase->stat);
        printf("Case %d: %s\r\n", i, pBenchmarkCase->pName);
        run_benchmark_single(pBenchmarkCase);
        statistics_print(pStat);
    }
    printf("----Benchmarking Complete----\r\n");

    return 0;
}
```

# Running C Applications

❑ Complete the two functions below

- **`mat_mul()`**: matrix multiplication **without** tiling
- **`mat_mul_tiling()`**: matrix multiplication **with** tiling



```
int a[N][N],b[N][N],result1[N][N],result2[N][N];

unsigned mat_mult(unsigned int uiParam0, unsigned int uiParam1, unsigned int uiParam2, unsigned int uiParam3 )
{
    int i,j,k;

    ////////////////////////////////
    //// Fill your code here! ////
    ////////////////////////////////

    return 1;
}

unsigned mat_mult_tiling(unsigned int uiParam0, unsigned int uiParam1, unsigned int uiParam2, unsigned int uiParam3 )
{
    int io,jo,ko,ii,ki,ji;
    int *rresult, *rb, *ra;

    ////////////////////////////////
    //// Fill your code here! ////
    ////////////////////////////////

    return 1;
}
```

**result1 = a * b**

**result2 = a * b**

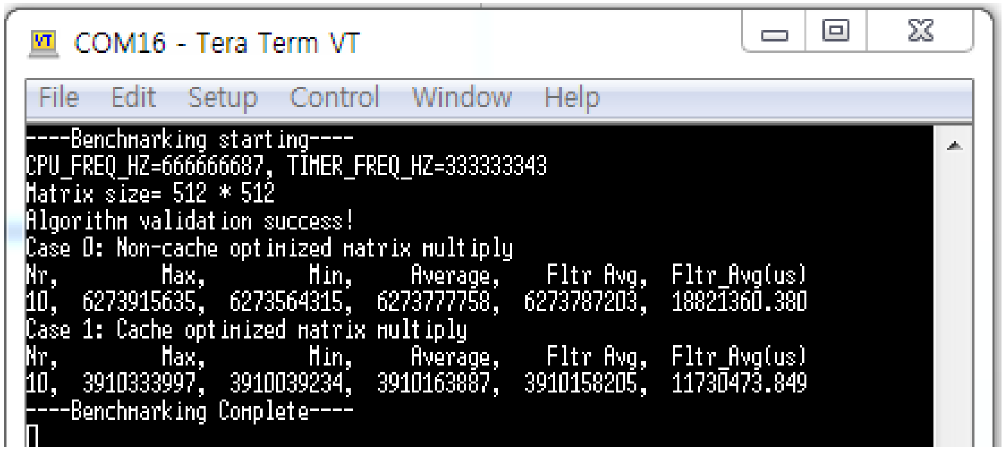
# Running C Applications

❑ Repeat the previous steps

- Follow to pp. 30~33 of the following lab workbook: **Lab_MP2022_1_work.pdf**

❑ Run the application

- Check the output on ***'Tera Term'***
  - ✓ You should see the execution times of both the functions.

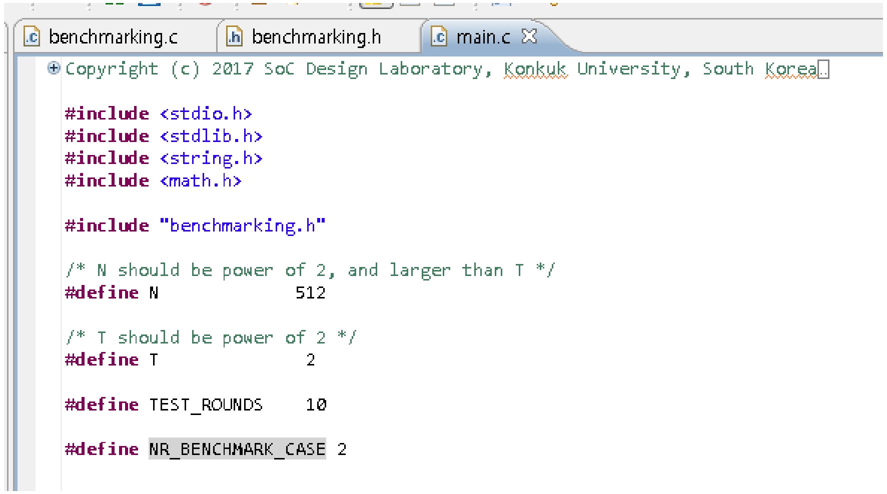


```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
Matrix size= 512 * 512
Algorithm validation success!
Case 0: Non-cache optimized matrix multiply
Nr,          Max,          Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,  6273915635,  6273564315,  6273777758,  6273787203,  18821360.380
Case 1: Cache optimized matrix multiply
Nr,          Max,          Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,  3910333997,  3910039234,  3910163887,  3910158205,  11730473.849
----Benchmarking Complete----
```

- Nr: Function execution count.
- Max: The longest time in the function execution count. (unit: cycles)
- Min: The shortest time in the function execution count. (unit: cycles)
- Average: Average time except Max and Min. (unit: cycles)
- Fltr_Avg: Average / TIMER_FREQ_HZ (unit: usecs)

# Optimizing C Application

- Optimize the tile size (`T`) to minimize the execution time
  - `T`: 2, 4, 8, 16



```
Copyright (c) 2017 SoC Design Laboratory, Konkuk University, South Korea

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "benchmarking.h"

/* N should be power of 2, and larger than T */
#define N             512

/* T should be power of 2 */
#define T              2

#define TEST_ROUNDS    10

#define NR_BENCHMARK_CASE 2
```

- Figure out how the tile size affects the execution time

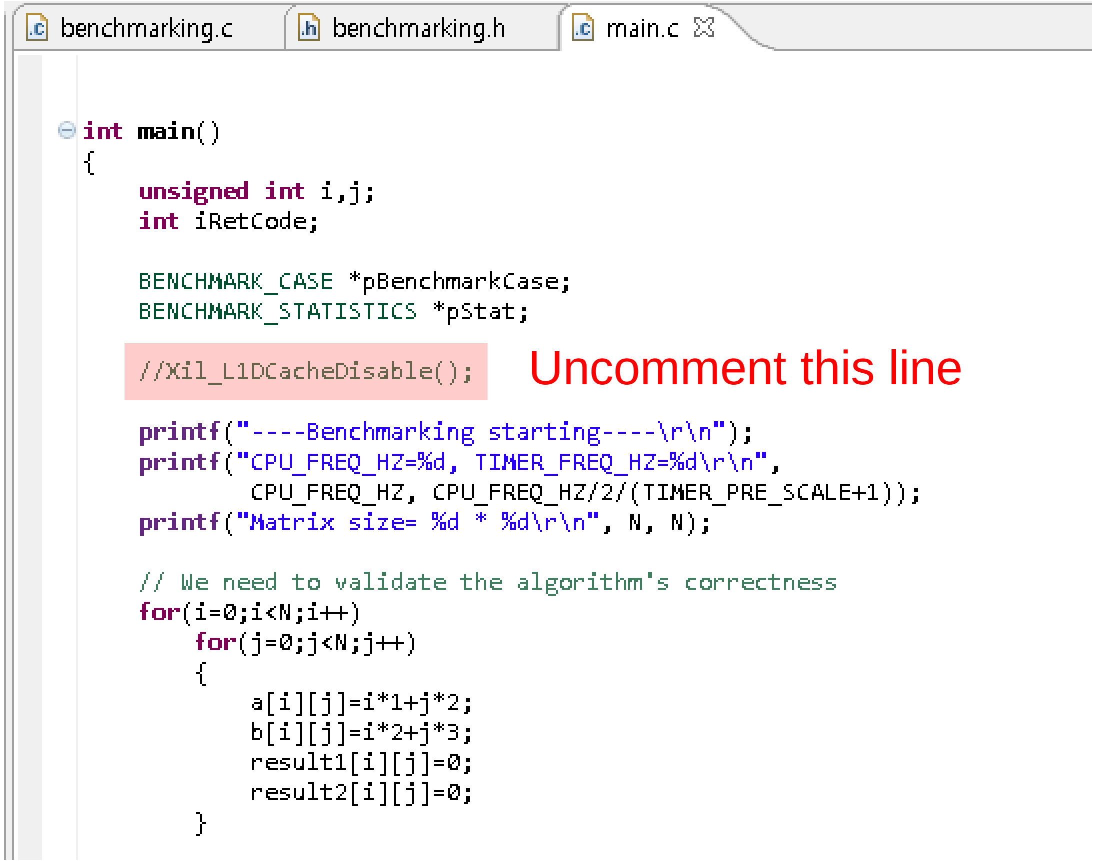
# Optimizing C Application

❑ Repeat with the optimization level set to –O2

- Follow to pp. 23-24 of the following lab workbook: **Lab_MP2022_2_work_r1.pdf**
- Review the disassembly to figure out how it reduces the execution time

# Optimizing C Application

- ❑ Repeat with L1 data cache disabled



```
int main()
{
    unsigned int i,j;
    int iRetCode;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    //Xil_L1DCacheDisable();

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n",
            CPU_FREQ_HZ, CPU_FREQ_HZ/2/(TIMER_PRE_SCALE+1));
    printf("Matrix size= %d * %d\r\n", N, N);

    // We need to validate the algorithm's correctness
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        {
            a[i][j]=i*1+j*2;
            b[i][j]=i*2+j*3;
            result1[i][j]=0;
            result2[i][j]=0;
        }
```

Uncomment this line