CS 475/575 -- Spring Quarter 2020

Project #7B

Autocorrelation using CPU OpenMP, CPU SIMD, and GPU {OpenCL or CUDA}

110 Points

Due: June 8 -- 23:59:59

```
rabbit ~/cs575/pro7 1017$ make
g++ -o CL-pro7 CL-pro7.cpp /usr/local/apps/cuda/cuda-10.1/lib64/libOpenCL.so.1.1
 -lm -fopenmp -Wno-write-strings
./CL-pro7
Size   32768    Local Size  32   48844.220 MultsPerSecond
rabbit ~/cs575/pro7 1017$ make
g++ -o CL-pro7 CL-pro7.cpp /usr/local/apps/cuda/cuda-10.1/lib64/libOpenCL.so.1.1
 -lm -fopenmp -Wno-write-strings
./CL-pro7
Size   32768    Local Size  32   48850.780 MultsPerSecond
rabbit ~/cs575/pro7 1017$ make
g++ -fopenmp -lm -o OpenMP-pro7 OpenMP-pro7.cpp
./OpenMP-pro7
 1298.14 avgPerformance per second
rabbit ~/cs575/pro7 1017$ ls
CL          CL-pro7.cpp  OpenMP-pro7.cpp    signal.txt     SIMD-Results.txt
CL-pro7     Makefile     resultsOMP.txt     SIMD-pro7
CL-pro7.cl  OpenMP-pro7  resultsOpenCL.txt  SIMD-pro7.cpp
rabbit ~/cs575/pro7 1018$
```


Liang Zhao

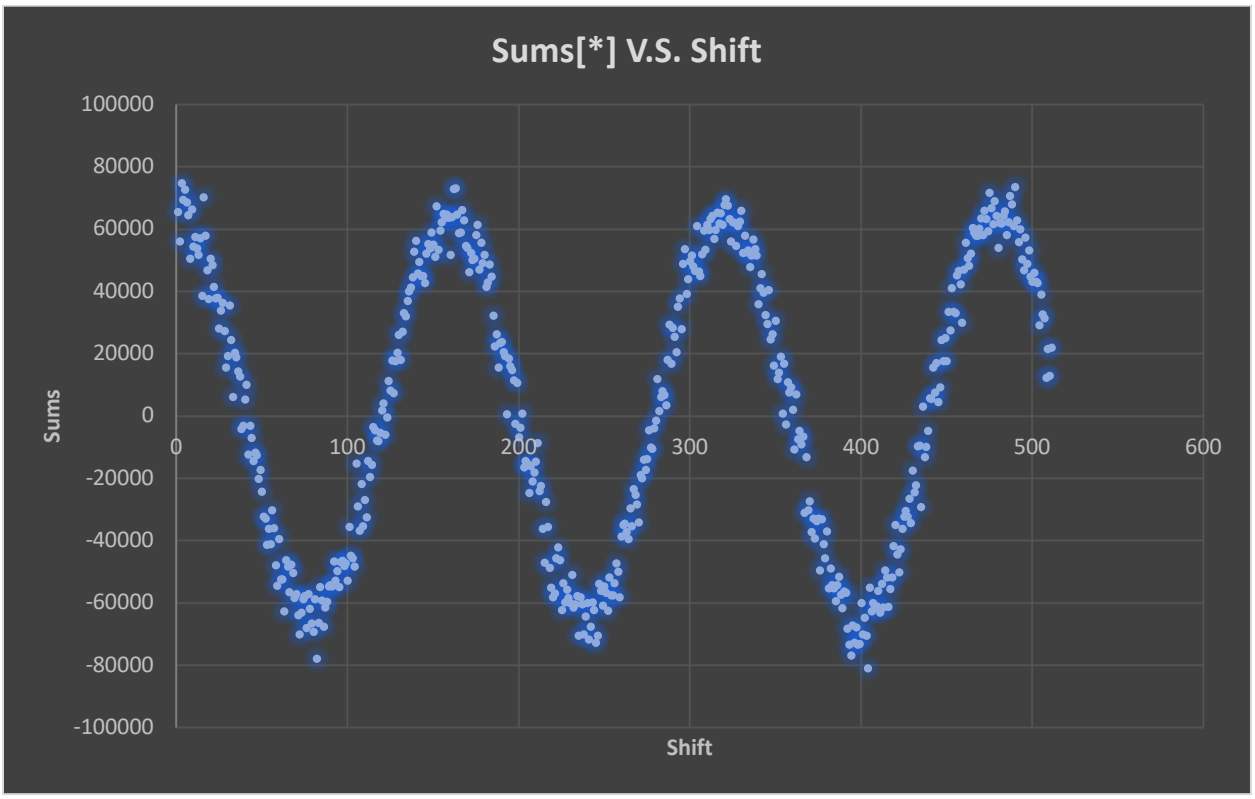
933-667-879

zhaolia@oregonstate.edu

What machines you ran this on:

My project runs on rabbit.engr.oregonstate.edu.

Show the Sums{1] ... Sums[512] vs. shift scatterplot:

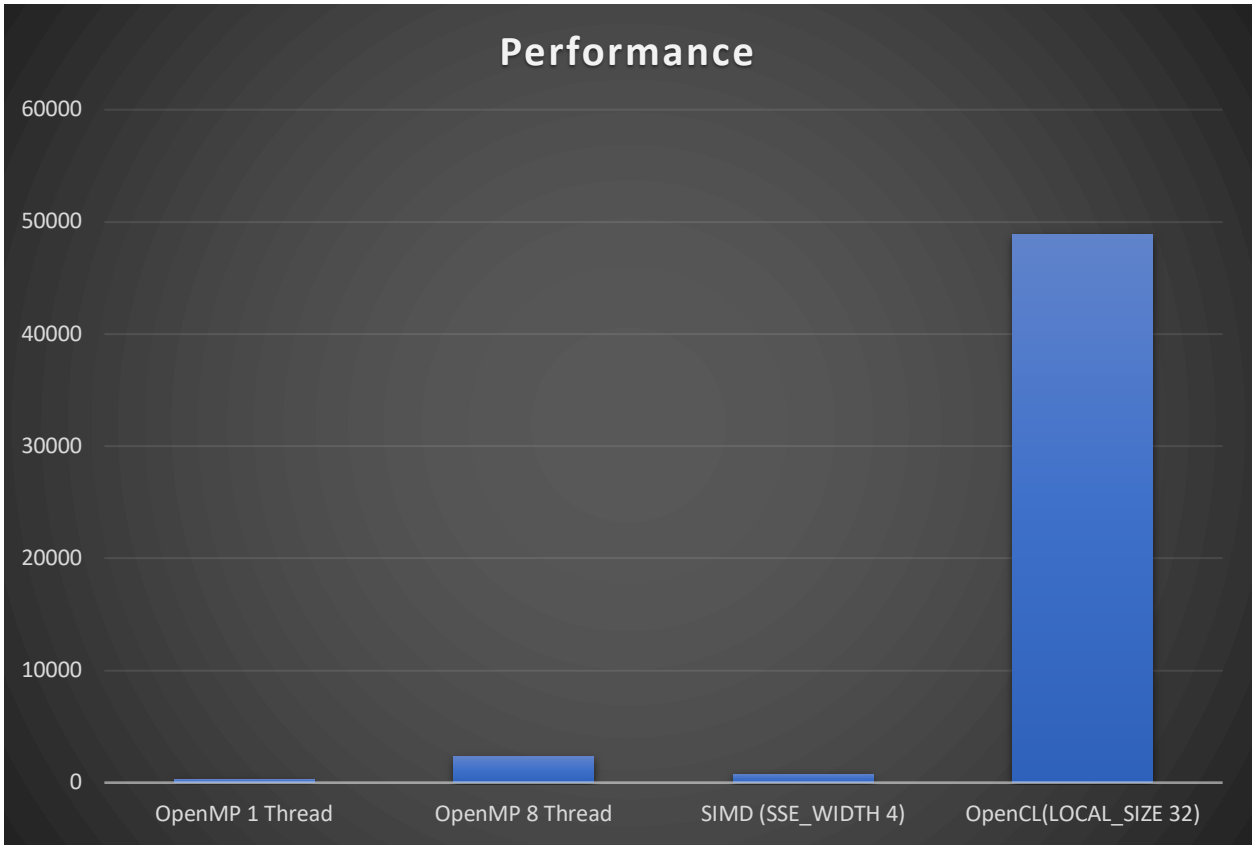


State what the hidden sine-wave period is, i.e., at what multiples of *shift* are you seeing maxima in the graph?

From the figure, the sine-wave period is about 160.

What patterns are you seeing in the performance bar chart? Which of the four tests runs fastest, next fastest, etc.? By a little, or by a lot?

	OpenMP 1 Thread	OpenMP 8 Thread	SIMD (SSE_WIDTH 4)	OpenCL(LOCAL_SIZE 32)
Performance	333.17	2342.59	838.94	48858.335



The results of this performance graph are surprising to me. The GPU OpenCL test run absolutely destroyed the CPU test runs at 48858 Megatrials per second. The CPU test runs were a little closer together with the next highest performance being OpenMP with 8 threads at 2342 Megatrials per second followed by SIMD at 838 Megatrials per second. Fortunately, 1 thread of OpenMP also ran the test with Megatrials 333 per second,lol.

Why do you think the performances work this way?

OpenCL beats other CPU tests because the GPU has many cores, which can share the work and overwhelmingly defeat other opponents. OpenCL does show that powerful GPUs have become powerful and how much destruction can be generated on simple tasks such as multiplication and addition. I use OSU's Rabbit GPU instead of DGX GPU, the results may be different, but I still believe it will destroy other test runs. The next closest run is through OpenMP (8 threads), then SIMD, then OpenMP (1 thread). The performance of the first three rounds was somewhat close, but it finally met my expectations. OpenMP performs eight floating points at a time, while SIMD only executes four floating points at a time. OpenMP (1 thread) is the worst-performing operation because it uses only one thread at a time, so it cannot effectively split the work.

Main Code:

```
omp_set_num_threads(NUMT);

double maxPerformance = 0.;
double avgPerformance = 0.;

for (int t = 0; t < NUMTRIES; t++)
{
    // start time
    double time0 = omp_get_wtime();

    #pragma omp parallel for default(none) shared(Size, A, Sums)
    for (int shift = 0; shift < Size; shift++)
    {
        float sum = 0.;
        for (int i = 0; i < Size; i++)
        {
            sum += A[i] * A[i + shift];
        }

        Sums[shift] = sum;
    }

    // end time
    double time1 = omp_get_wtime();

    double avgPerformance = (double)(Size*Size)  / (time1 - time0) / 1000000.;


    if (avgPerformance > maxPerformance)

        maxPerformance = avgPerformance;


    //Write Data
    fp = fopen( "resultsOMP.txt", "w" );
    if( fp == NULL )
    {
        fprintf( stderr, "Cannot open file 'resultsOMP.txt' for writing\n" );
        exit( 1 );
    }
    for( int i = 0; i < 512; i++ )
    {
        fprintf( fp, "%4d\t%f\n",i, Sums[ i ] );
    }
    fclose( fp );
```

```
    }



    printf( "%8.2lf avgPerformance per second\n", maxPerformance );
```

```
double maxMegaMultsSimd = 0.;

    //Computing SimdMulSum performance

    for( int t = 0; t < NUMTRIES; t++ )
    {
        double time0 = omp_get_wtime( );

        for( int shift = 0; shift < Size; shift++ )
        {
            Sums[ shift ] = SimdMulSum( &A[ 0 ], &A[ 0 + shift ], Size );
        }


        double time1 = omp_get_wtime( );


        double megaMults = (double)(Size*Size)/(time1-time0)/1000000.;

        if( megaMults > maxMegaMultsSimd )
            maxMegaMultsSimd = megaMults;


        //Write Results
        fp = fopen( "SIMD-Results.txt", "w" );
        if( fp == NULL )
        {
            fprintf( stderr, "Cannot open file 'resultsSIMD.txt' for writing\n" );
            exit( 1 );
        }
        for( int i = 0; i < 512; i++ )
        {
            fprintf( fp, "%4d\t%f\n",i, Sums[ i ] );
        }
        fclose( fp );

    }
```

```
    printf( "Peak Performance with SIMD    = %8.2lf MegaMults/Sec\n", maxMegaMultsSimd
);

    delete [ ] A;
    delete [ ] Sums;

    return 0;


}



float SimdMulSum( float *a, float *b, int len )
{
    float sum[4] = { 0., 0., 0., 0. };
    int limit = ( len/SSE_WIDTH ) * SSE_WIDTH;
    register float *pa = a;
    register float *pb = b;

    __m128 ss = _mm_loadu_ps( &sum[0] );
    for( int i = 0; i < limit; i += SSE_WIDTH )
    {
        ss = _mm_add_ps( ss, _mm_mul_ps( _mm_loadu_ps( pa ), _mm_loadu_ps( pb ) ) );
        pa += SSE_WIDTH;
        pb += SSE_WIDTH;
    }
    _mm_storeu_ps( &sum[0], ss );

    for( int i = limit; i < len; i++ )
    {
        sum[0] += a[i] * b[i];
    }

    return sum[0] + sum[1] + sum[2] + sum[3];
}
```

```
// 11. enqueue the kernel object for execution:

    size_t globalWorkSize[3] = { Size, 1, 1 };
    size_t localWorkSize[3]  = { LOCAL_SIZE,   1, 1 };

    Wait( cmdQueue );
```

```
    double time0 = omp_get_wtime( );

    time0 = omp_get_wtime( );

    status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize,
localWorkSize, 0, NULL, NULL );
    if( status != CL_SUCCESS )
        fprintf( stderr, "clEnqueueNDRangeKernel failed: %d\n", status );

    Wait( cmdQueue );
    double time1 = omp_get_wtime( );

    // 12. read the results buffer back from the device to the host:

    status = clEnqueueReadBuffer( cmdQueue, dSums, CL_TRUE, 0, SumsSize, hSums, 0,
NULL, NULL );
    if( status != CL_SUCCESS )
            fprintf( stderr, "clEnqueueReadBuffer failed\n" );

    // pro7-adding main point

    //Write Result
    fp = fopen( "resultsOpenCL.txt", "w" );
    if( fp == NULL )
    {
        fprintf( stderr, "Cannot open file 'resultsGPU.txt' for writing\n" );
        exit( 1 );
    }
    for( int i = 0; i < 512; i++ )
    {
        fprintf( fp, "%4d\t%f\n", i,hSums[ i ] );
    }
    fclose( fp );


    fprintf( stderr, "Size%8d\tLocal Size%4d\t%10.3lf MultsPerSecond\n",
        Size, LOCAL_SIZE,  (double)(Size*Size)/(time1-time0)/1000000. );

#ifdef WIN32
    Sleep( 2000 );
#endif
```