**CS 475/575 -- Spring Quarter 2020**

**Project #4**

**Vectorized Array Multiplication/Reduction using SSE**

**60 Points**

**Due: May 11**

**A close up of a newspaper

Description automatically generated**

**Liang Zhao**

**933-667-879**

**zhaolia@oregonstate.edu**

Hi there, my program run on OSU Linux server: access.engr.oregonstate.edu. I used a bash script to automatically adjust the number of Array-Sizes and SimdPerformance, NonSimdPerformance , SpeedUp, and save the results in a csv file. Let us first look at the results of the experiment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Array-Sizes** | **SimdPerformance** | **NonSimdPerformance** | **SpeedUp** |
| 1000 | 3381.86 | 956.14 | 3.35 |
| 2000 | 3591.11 | 962.13 | 3.73 |
| 4000 | 3658.4 | 969.3 | 3.77 |
| 8000 | 3345.64 | 972.04 | 3.65 |
| 10000 | 3386.66 | 972.55 | 3.48 |
| 16000 | 3394.22 | 973.56 | 3.49 |
| 32000 | 3139.17 | 971.41 | 3.23 |
| 64000 | 2704.91 | 966.31 | 2.8 |
| 128000 | 2673.91 | 969.34 | 2.76 |
| 256000 | 2735.28 | 968.21 | 2.83 |
| 512000 | 2726.67 | 957.53 | 2.85 |
| 1024000 | 2439.85 | 930.75 | 2.62 |
| 2048000 | 1342 | 821.9 | 1.63 |
| 4096000 | 1360.11 | 816 | 1.67 |
| 5000000 | 1282.86 | 837.58 | 1.62 |
| 6000000 | 1410.74 | 836.2 | 1.69 |
| 8000000 | 1356.04 | 844.96 | 1.62 |

And the graph of SIMD/non-SIMD speedup versus array size:

Let's zoom in on the data of 1-128000 Array-Sizes. We can see subtle changes in the speedup data.

We can see that the maximum speedup is 3.77, which the array size is 4000. I later improved g ++ 's compile command and added "-O3". The performance of the program was improved. Speedup increased during the process of size 1-4000, and gradually decreased to 1.62 after 4000. I also tested that when the array size is close to infinity, the speedup is around 1.5. This may be the limit data of speedup. If the data size is too small, the benefit of SIMD will be shaded by the additional work for SIMD. And the temporal coherence is violated when the size of data set is too much. I think data in cache line is possibly only used once and then be replaced.

I also did the extra credit. The core code is like this:

    omp\_set\_num\_threads(NUMT);

    double maxMegaMultsSimd = 0.;

    double maxMegaMultsNonSimd = 0.;

    //Computing SimdMulSum performance

    for( int t = 0; t < NUMTRIES; t++ )

    {

        double time0 = omp\_get\_wtime( );

        //SimdMulSum(a\_array, b\_array, ARRAYSIZE);

        #pragma omp parallel

        {

            int e = omp\_get\_thread\_num( ) \* (ARRAYSIZE/NUMT);

            SimdMulSum(&a\_array[e], &b\_array[e], (ARRAYSIZE/NUMT));

        }

        double time1 = omp\_get\_wtime( );

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

        if( megaMults > maxMegaMultsSimd )

            maxMegaMultsSimd = megaMults;

    }

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SpeedUP |  | 1 Thread | 2 Threads | 4 Threads | 8 Threads | 16 Threads |
| Array-Sizes | 1000 | 2.07 | 1.93 | 2.12 | 1.53 | 1.5 |
|  | 2000 | 2.4 | 3.23 | 3.55 | 3.12 | 2.68 |
|  | 4000 | 2.56 | 4.03 | 5.46 | 5.13 | 4.52 |
|  | 8000 | 2.7 | 4.8 | 7.41 | 8.92 | 9.48 |
|  | 10000 | 2.73 | 4.72 | 7.78 | 10 | 6.56 |
|  | 16000 | 2.78 | 5.15 | 8.94 | 11.99 | 8.26 |
|  | 32000 | 2.8 | 5.41 | 10.02 | 14.67 | 9.9 |
|  | 64000 | 2.82 | 5.33 | 9.39 | 17.61 | 11.21 |
|  | 128000 | 2.83 | 5.58 | 10.44 | 19.88 | 21.05 |
|  | 256000 | 2.85 | 5.64 | 10.84 | 21.31 | 21.93 |
|  | 512000 | 2.74 | 5.61 | 11.01 | 21.94 | 22.1 |
|  | 1024000 | 2.71 | 5.67 | 11.5 | 22.22 | 22.82 |
|  | 2048000 | 2.74 | 5.57 | 11.13 | 21.76 | 22.91 |
|  | 4096000 | 2.73 | 5.49 | 9.58 | 11.92 | 15.9 |
|  | 5000000 | 2.68 | 5.15 | 10.05 | 11.07 | 14.54 |
|  | 6000000 | 2.7 | 5.03 | 9.86 | 10.88 | 13.98 |
|  | 8000000 | 2.67 | 4.71 | 9.73 | 11.63 | 13.68 |
|  |  |  |  |  |  |  |

Data set size is critical to performance. If the size of the data set is too small, the advantages of SIMD will be masked by the extra work of SIMD. If the data set is too large, it will destroy temporal coherence and reduce performance.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <xmmintrin.h>

#ifndef NUMT

#define NUMT      4

#endif

#ifndef ARRAYSIZE

#define ARRAYSIZE 1000000

#endif

#ifndef NUMTRIES

#define NUMTRIES 100

#endif

#ifndef SSE\_WIDTH

#define SSE\_WIDTH 4

#endif

float Ranf( unsigned int \*seedp,  float low, float high );

float SimdMulSum( float \*a, float \*b, int len );

float NonSimdMulSum( float \*a, float \*b, int len);

    float a\_array[ARRAYSIZE];

    float b\_array[ARRAYSIZE];

int main(int argc, char \*\*argv)

{

//Verify OpenMP support

#ifndef \_OPENMP

    fprintf( stderr, "No OpenMP support!\n" );

    return 1;

#endif

    //float \* a\_array = (float \*)malloc(sizeof(float) \* length);

    //Inialize the arrays:

    unsigned int seed = omp\_get\_wtime( );

    for(int i=0;i<ARRAYSIZE;i++)

    {

        a\_array[i] = Ranf(&seed, 0.f, 1000.f );

        b\_array[i] = Ranf(&seed, 0.f, 1000.f );

    }

    omp\_set\_num\_threads(NUMT);

    double maxMegaMultsSimd = 0.;

    double maxMegaMultsNonSimd = 0.;

    //Computing SimdMulSum performance

    for( int t = 0; t < NUMTRIES; t++ )

    {

        double time0 = omp\_get\_wtime( );

        //SimdMulSum(a\_array, b\_array, ARRAYSIZE);

        #pragma omp parallel

        {

            int e = omp\_get\_thread\_num( ) \* (ARRAYSIZE/NUMT);

            SimdMulSum(&a\_array[e], &b\_array[e], (ARRAYSIZE/NUMT));

        }

        double time1 = omp\_get\_wtime( );

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

        if( megaMults > maxMegaMultsSimd )

            maxMegaMultsSimd = megaMults;

    }

    //Computing NonSimdMulSum performance

    for( int t = 0; t < NUMTRIES; t++ )

    {

        double time0 = omp\_get\_wtime( );

        NonSimdMulSum(a\_array, b\_array, ARRAYSIZE);

        double time1 = omp\_get\_wtime( );

        double megaMultsNonSimd = (double)ARRAYSIZE/(time1-time0)/1000000.;

        if( megaMultsNonSimd > maxMegaMultsNonSimd )

            maxMegaMultsNonSimd = megaMultsNonSimd;

    }

    //speedup

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

    printf( "Peak Performance without SIMD   = %8.2lf MegaMults/Sec\n", maxMegaMultsNonSimd );

    double speedup = maxMegaMultsSimd/maxMegaMultsNonSimd;

    printf("SpeedUp = %8.2lf\n", speedup);

}

float Ranf(unsigned int \*seedp, float low, float high) {

    float r = (float) rand\_r(seedp);

    return (low + r \* (high - low) / (float)RAND\_MAX);

}

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];

}

float NonSimdMulSum( float \*a, float \*b, int len)

{

    float sum = 0.;

    for(int i =0; i< len; i++)

    {

        sum += a[i] \* b[i];

    }

    return sum;

}