COMP2006 Computer Organization
Introduction to AVX Intrinsics

In the last lab, we know that the size of the matrix affects the performance of the DGEMM function to calculates the multiplication of matrices. Today, we learn how to use the *AVX (Advanced Vector Extensions) Intrinsics* to increase the performance of our programs.

AVX Intrinsics

AVX is a set of instructions for doing Single Instruction Multiple Data (SIMD) operations on Intel Architecture CPUs. SIMD instructions allow processing of multiple pieces of data in a single step, speeding up throughput for many tasks, from video encoding and decoding to image processing to data analysis to physics simulations.

Intel C++ Compiler version 11.1 or later, *Microsoft Visual Studio C++ 2010 with SP1* and later, *GNU Compiler Collection (GCC)* version 4.4 or later support Intel *AVX intrinsics* using the **<immintrin.h>** header.

AVX intrinsics provides the following data types:

Type	Meaning
__m256	256-bit as 8 single-precision floating-point values
__m256d	256-bit as 4 double-precision floating-point values
__m256i	256-bit as 8 integers (32-bit each), 32 bytes (8-bit each), 16 words (16-bit each), etc.
__m128	128-bit as 4 single-precision floating-point
__m128d	128-bit as 2 double-precision floating-point values

In this lab, we focus on the data type **__m256d** with the following functions:

- __m256d **_mm256_load_pd** *(double const * mem_addr)*
  - Moves packaged single-precision floating-point values from aligned memory location to a destination vector (a 256-bite destination operand).

- void **_mm256_store_pd** *(double * mem_addr, __m256d a)*
  - Stores 256 bits from __mm256d **a** into memory. **mem_addr** must be aligned on a 32-byte boundary or a general-protection exception may be generated.

- __m256d **_mm256_add_pd** *(__m256d a, __m256d b)*
  - Sums the elements of two vector variables (**a** and **b**).

- __m256d **_mm256_sub_pd** *(__m256d a, __m256d b)*
  - Subtracts the elements of one vector variable b from another vector variable a.

- __m256d **_mm256_mul_pd** *(__m256d a, __m256d b)*
    - Multiplies the elements of two vector variables (**a** and **b**).

- __m256d **_mm256_div_pd** *(__m256d a, __m256d b)*
    - Divides the elements of one vector variable a by another vector variable b.

- __m256d **_mm256_sqrt_pd** *(__m256d a)*
    - Returns a __m256d vector where each of the elements are set equal to the square root of a.

- __m256d **_mm256_broadcast_sd** *(double const * mem_addr)*
    - Loads one double-precision floating-point value into all elements of a vector variable.

Load and Store Operations

The following example program shows how to use the AVX intrinsics functions to load the values from an array to the __mm256d vector variable and store the values to another array.

```c
#include <immintrin.h>
#include <stdio.h>

int main()
{
    int i;
    double A[] = {1.2, 2.4, 3.6, 4.8};
    double B[4] = {0.0};

    __m256d vA = _mm256_load_pd(A);

    _mm256_store_pd(B, vA);

    for (i = 0; i < 4; i++)
        printf("B[%d] = %.1lf\t", i, B[i]);

    printf("\n");
    return 0;
}
```

To compile the program with *AVX Intrinsics* functions using GCC, we need to use **-mavx** option. Let's download **avx1.c** from our course webpage. Then, compile it and run it.

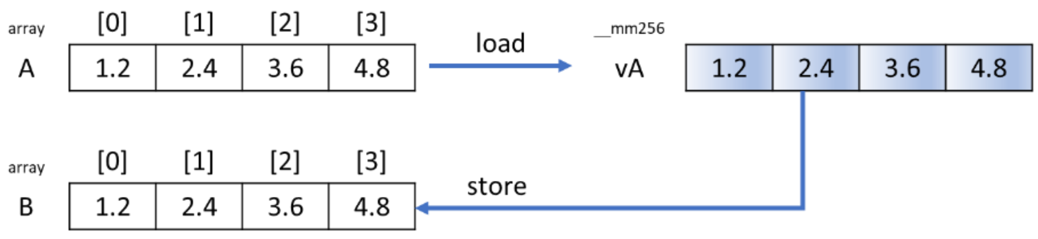
Compile: ***gcc -O3 -mavx avx1.c -o avx1***

Run on Windows: ***avx1***

Run on Macintosh: ***./avx1***

The following is the output:

```
B[0] = 1.2       B[1] = 2.4       B[2] = 3.6       B[3] = 4.8
```



Add Operations

The following example program shows how to use the *AVX Intrinsics* functions to load the values from two arrays, add them and store them to the destination.

```
#include <immintrin.h>
#include <stdio.h>

int main() {
   int i;
   double A[] = {1.2, 2.4, 3.6, 4.8};
   double B[] = {0.1, 1.2, 2.3, 3.4};
   double C[4] = {0.0};

   __m256d vA = _mm256_load_pd(A);
   __m256d vB = _mm256_load_pd(B);

   __m256d vC = _mm256_add_pd(vA, vB);

   _mm256_store_pd(C, vC);

   for(i=0; i<4; i++)
      printf("C[%d] = %.1lf\t", i, C[i]);
   printf("\n");
   return 0;
}
```

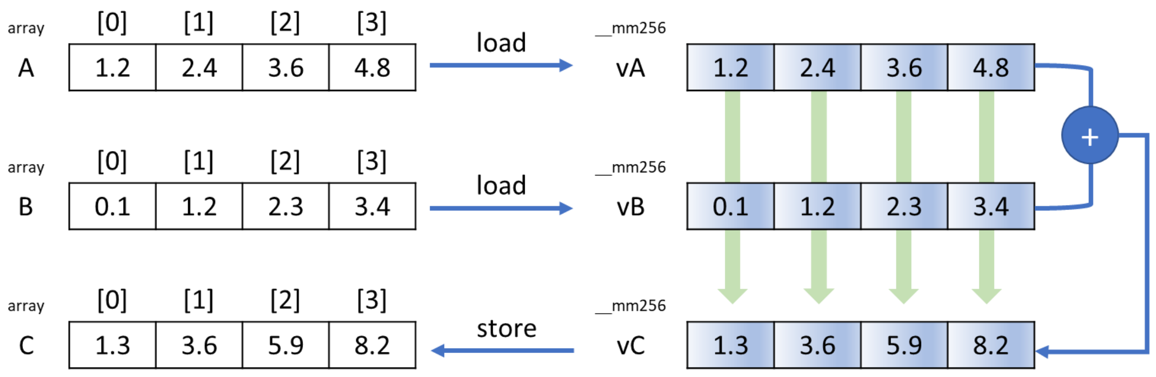
Download avx2.c from our course webpage. Then, compile it and run it.

Compile: ***gcc -O3 -mavx avx2.c -o avx2*** *Run on Windows:* ***avx2*** *Run on Macintosh:* ***./avx2***

You should see the output as follows:

```
C[0] = 1.3      C[1] = 3.6      C[2] = 5.9      C[3] = 8.2
```

What has the program done with the *AVX Intrinsics*? The program summed 4 pair doubles in a single instruction.



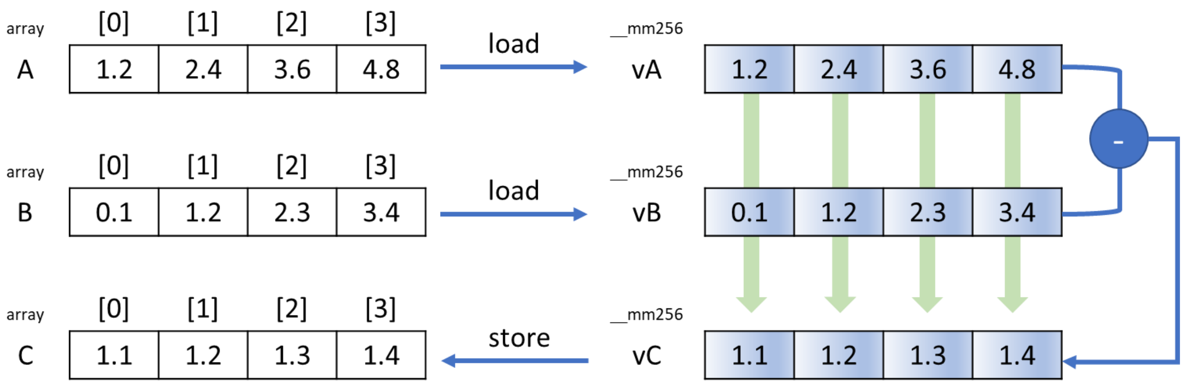
Let's change the line:

```
  __m256d vC = _mm256_add_pd(vA, vB);
```

To:

```
  __m256d vC = _mm256_sub_pd(vA, vB);
```

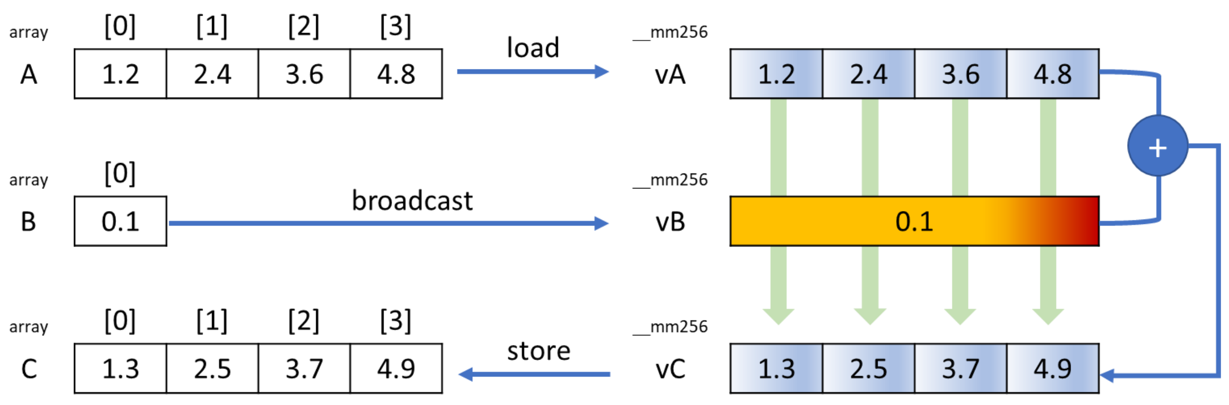
Then, we compile and run the program again. Now, we get the difference of A and B.



Broadcast operation

In the previous example program, we use the **_mm256_load_pd** function to load the values from an array to a destination vector. Let’s try another function **_mm256_broadcast_sd** and see what the difference is if we modify the program code as follows:

```
#include <immintrin.h>
#include <stdio.h>

int main()
{
    int i;
    double A[] = {1.2, 2.4, 3.6, 4.8};
    double B[] = {0.1};
    double C[4] = {0.0};

    __m256d vA = _mm256_load_pd(A);

    __m256d vB = _mm256_broadcast_sd(B);

    __m256d vC = _mm256_add_pd(vA, vB);

    _mm256_store_pd(C, vC);

    for (i = 0; i < 4; i++)
        printf("C[%d] = %.1lf\t", i, C[i]);
    printf("\n");
    return 0;
}
```



Let’s download avx3.c from our course webpage. Then, compile it and run it.

Compile: ***gcc -O3 -mavx avx2.c -o avx2*** *Run on Windows:* ***avx2*** *Run on Macintosh:* ***./avx2***

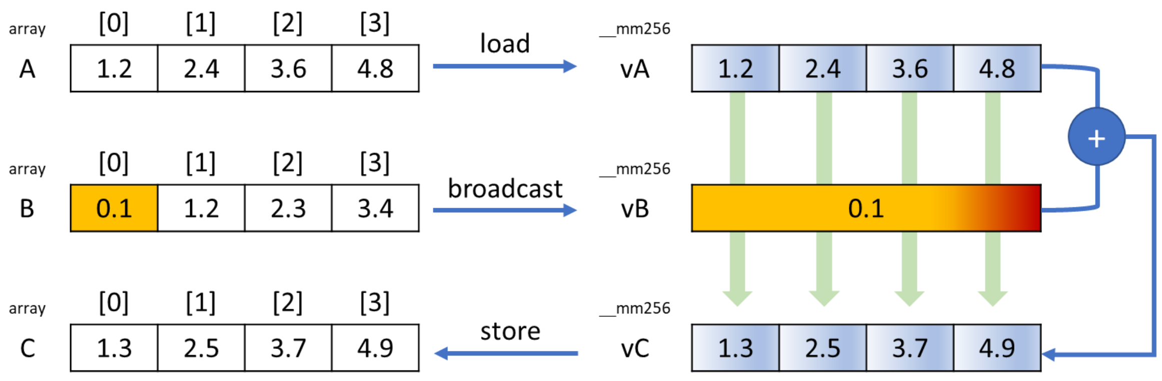
You should see the output as follows:

```
C[0] = 1.3      C[1] = 2.5      C[2] = 3.7      C[3] = 4.9
```

So, how about we have multiple elements in array *B* and we use **_mm256_broadcast_sd** to load the value to the vector.

```
#include <immintrin.h>
#include <stdio.h>

int main() {
   int i;
   double A[] = {1.2, 2.4, 3.6, 4.8};
   double B[] = {0.1, 1.2, 2.3, 3.4};
   double C[4] = {0.0};

   __m256d vA = _mm256_load_pd(A);

   __m256d vB = _mm256_broadcast_sd(B);

   __m256d vC = _mm256_add_pd(vA, vB);

   _mm256_store_pd(C, vC);

   for(i=0; i<4; i++)
      printf("C[%d] = %.1lf\t", i, C[i]);
   printf("\n");
   return 0;
}
```

We will get the same result because the **_mm256_broadcast_sd** function ignores other elements of array *B*.



Exercise

Set A: *{3.3, 1.5, 4.4, 2.2, 9.5, 1.4, 5.4, 4.3, 3.8, 5.8, 8, 9.3, 8.5, 0.8, 5.3, 4.3, 5.6, 4.5, 1.2, 2.1}*

Set B: *{9.6, 4.7, 8.7, 8.4, 5.3, 1.3, 7.8, 6.1, 7.5, 0.1, 5.4, 3.4, 6.3, 7.4, 2.9, 0.2, 1.7, 5.9, 1.2, 7.4}*

Given two sets of values above. Reference avx2.c and avx3.c and write C programs with AVX Intrinsics to complete the followings respectively:

1. Multiply two sets of values and print the result to the console.
   *i.e.: A[0]*B[0], A[1]*B[1], so on and so forth.*

2. Divide two sets of values and print the result to the console.
   i.e.: A[0]/B[0], A[1]/B[1], so on and so forth.

3. Divide the values of the set A by two and print the result to the console.
   i.e.: A[0]/2, A[1]/2, so on and so forth.

Copy your codes and paste them on the worksheet and submit it to Moodle.