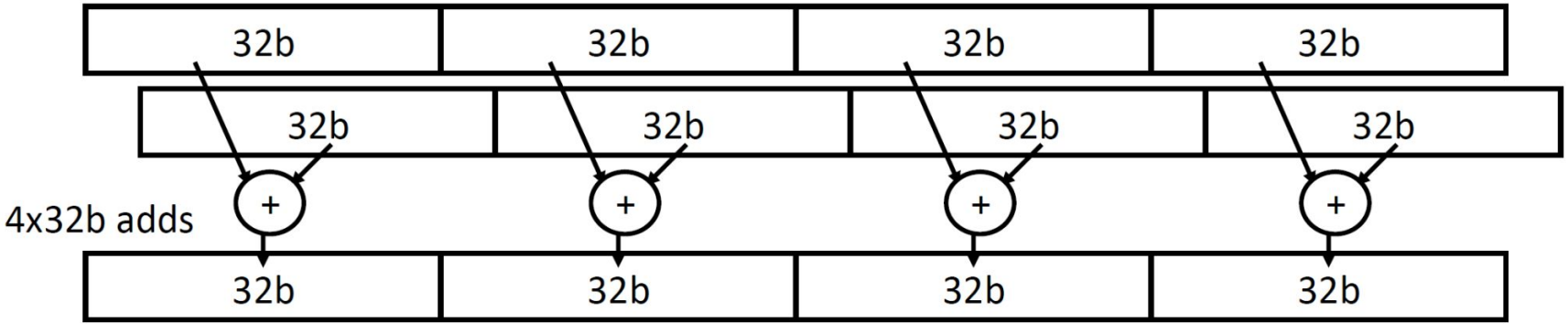
Lab 8

CS61C

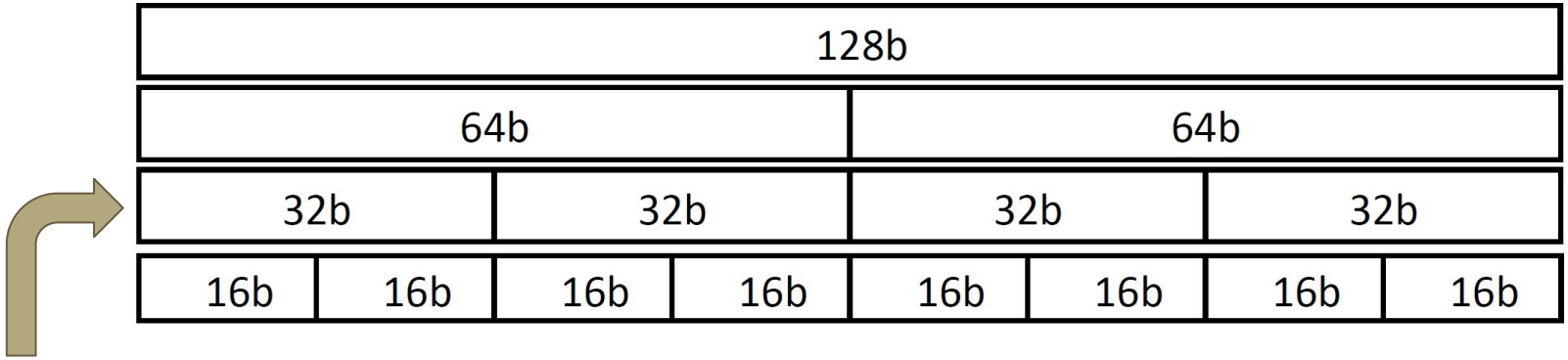
What is SIMD?

- Single Instruction, Multiple Data
- Data are packed together then same operations can be done in parallel



What is SIMD?

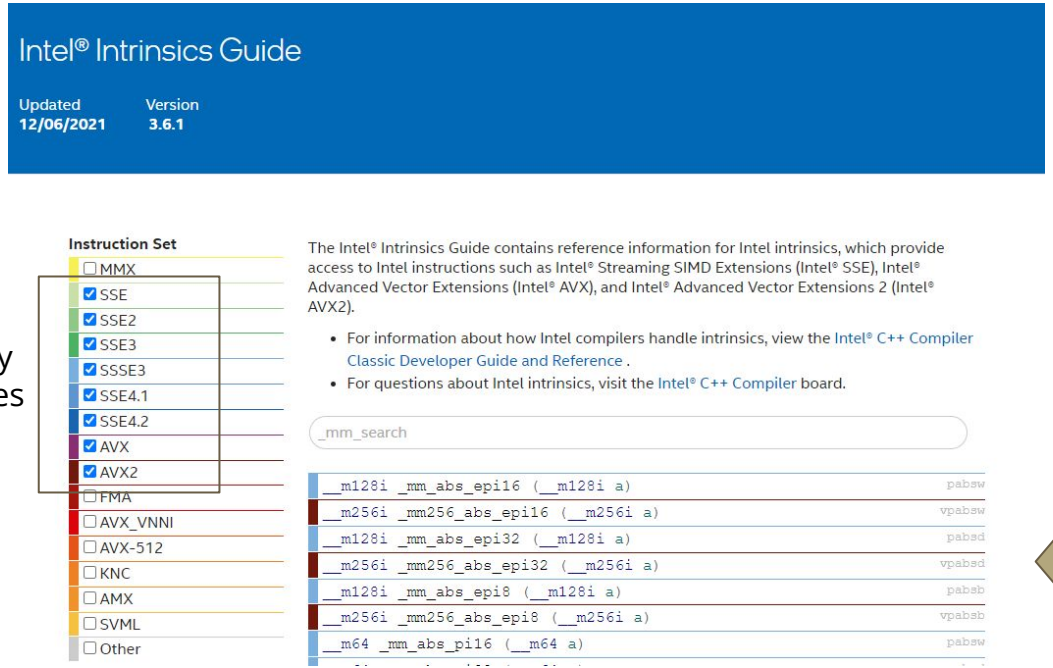
- Single Instruction, Multiple Data
- Data are packed together then same operations can be done in parallel



4 32-bit integers combined to form 128-bits

SIMD Functions

- For this class, we'll use Intel's SIMD instructions



Supported by hive machines

List of all possible functions that you can use for SIMD operations

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

Some SIMD Functions

- `__m128i _mm_setzero_si128()`
  - returns a 128-bit zero vector
- `__m128i _mm_loadu_si128(__m128i *p)`
  - returns 128-bit vector stored at pointer p
- `__m128i _mm_add_epi32(__m128i a, __m128i b)`
  - returns vector ($a_0 + b_0$, $a_1 + b_1$, $a_2 + b_2$, $a_3 + b_3$)
- `void _mm_storeu_si128(__m128i *p, __m128i a)`
  - stores 128-bit vector a into pointer p

Review sum() and sum_unrolled()

SIMD Functions recap

- `__m128i _mm_setzero_si128()`
  - returns a 128-bit zero vector
- `__m128i _mm_loadu_si128(__m128i *p)`
  - returns 128-bit vector stored at pointer p
- `__m128i _mm_add_epi32(__m128i a, __m128i b)`
  - returns vector ($a_0 + b_0$, $a_1 + b_1$, $a_2 + b_2$, $a_3 + b_3$)
- `void _mm_storeu_si128(__m128i *p, __m128i a)`
  - stores 128-bit vector a into pointer p

Sample SIMD code

```c
#include <x86intrin.h>

int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3, 1, 4, 1}

// Load array elements 4-7 into a temporary vector register
tmp = _mm_loadu_si128((__m128i *) (arr + 4));
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}

// Create temporary array to hold values from sum_vec
// Store the vector into an array to access the individual values
int tmp_arr[4];
_mm_storeu_si128((__m128i *) tmp_arr, sum_vec);
// Collect values from sum_vec in a single integer
int sum = tmp_arr[0] + tmp_arr[1] + tmp_arr[2] + tmp_arr[3];
```

Sample SIMD code

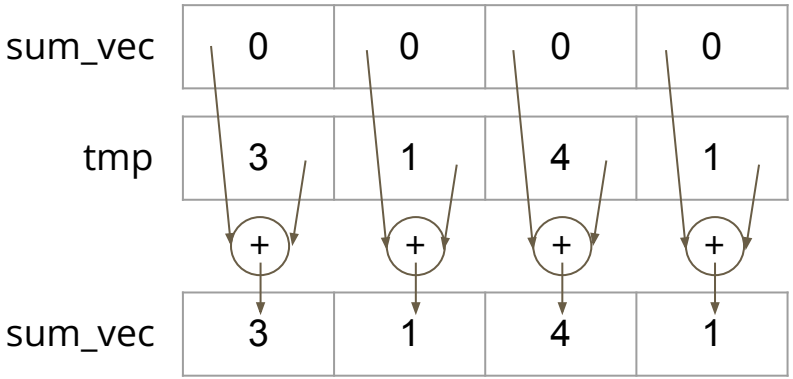
```
1  #include <x86intrin.h>
2
3  int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};
4  // Initialize sum vector to {0, 0, 0, 0}
5  __m128i sum_vec = _mm_setzero_si128();
6
7  // Load array elements 0-3 into a temporary vector register
8  __m128i tmp = _mm_loadu_si128((__m128i *) arr);
9  // Add to existing sum vector
10 sum_vec = _mm_add_epi32(sum_vec, tmp);
11 // sum_vec = {3, 1, 4, 1}
12
13 // Load array elements 4-7 into a temporary vector register
14 tmp = _mm_loadu_si128((__m128i *) (arr + 4));
15 // Add to existing sum vector
16 sum_vec = _mm_add_epi32(sum_vec, tmp);
17 // sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}
18
19 // Create temporary array to hold values from sum_vec
20 // Store the vector into an array to access the individual values
21 int tmp_arr[4];
22 _mm_storeu_si128((__m128i *) tmp_arr, sum_vec);
23 // Collect values from sum_vec in a single integer
24 int sum = tmp_arr[0] + tmp_arr[1] + tmp_arr[2] + tmp_arr[3];
```

C library to use the SIMD functions

Initial array that will be “SIMDized”

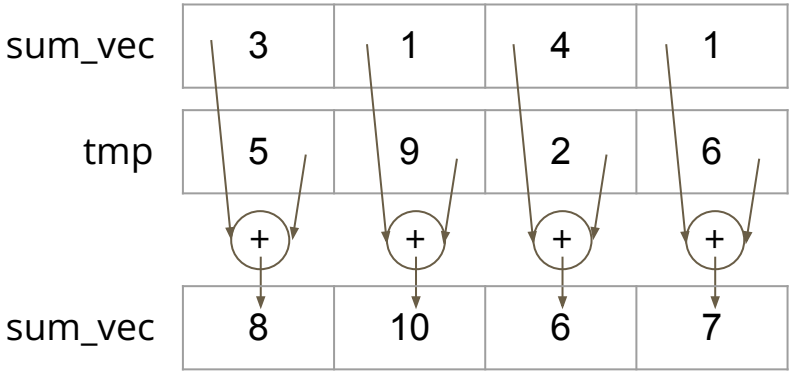
Sample SIMD code

```
1  #include <x86intrin.h>
2
3  int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};
4  // Initialize sum vector to {0, 0, 0, 0}
5  __m128i sum_vec = _mm_setzero_si128();
6
7  // Load array elements 0-3 into a temporary vector register
8  __m128i tmp = _mm_loadu_si128((__m128i *) arr);
9  // Add to existing sum vector
10 sum_vec = _mm_add_epi32(sum_vec, tmp);
11 // sum_vec = {3, 1, 4, 1}
12
13 // Load array elements 4-7 into a temporary vector register
14 tmp = _mm_loadu_si128((__m128i *) (arr + 4));
15 // Add to existing sum vector
16 sum_vec = _mm_add_epi32(sum_vec, tmp);
17 // sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}
18
19 // Create temporary array to hold values from sum_vec
20 // Store the vector into an array to access the individual values
21 int tmp_arr[4];
22 _mm_storeu_si128((__m128i *) tmp_arr, sum_vec);
23 // Collect values from sum_vec in a single integer
24 int sum = tmp_arr[0] + tmp_arr[1] + tmp_arr[2] + tmp_arr[3];
```



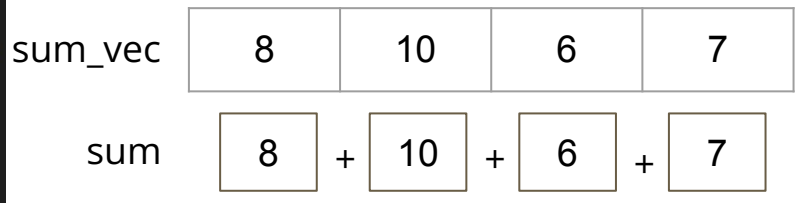
Sample SIMD code

```
#include <x86intrin.h>

int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};
// Initialize sum vector to {0, 0, 0, 0}
__m128i sum_vec = _mm_setzero_si128();

// Load array elements 0-3 into a temporary vector register
__m128i tmp = _mm_loadu_si128((__m128i *) arr);
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3, 1, 4, 1}

// Load array elements 4-7 into a temporary vector register
tmp = _mm_loadu_si128((__m128i *) (arr + 4));
// Add to existing sum vector
sum_vec = _mm_add_epi32(sum_vec, tmp);
// sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}

// Create temporary array to hold values from sum_vec
// Store the vector into an array to access the individual values
int tmp_arr[4];
_mm_storeu_si128((__m128i *) tmp_arr, sum_vec);
// Collect values from sum_vec in a single integer
int sum = tmp_arr[0] + tmp_arr[1] + tmp_arr[2] + tmp_arr[3];
```



Sample SIMD code

```
1  #include <x86intrin.h>
2
3  int arr[8] = {3, 1, 4, 1, 5, 9, 2, 6};
4  // Initialize sum vector to {0, 0, 0, 0}
5  __m128i sum_vec = _mm_setzero_si128();
6
7  // Load array elements 0-3 into a temporary vector register
8  __m128i tmp = _mm_loadu_si128((__m128i *) arr);
9  // Add to existing sum vector
10 sum_vec = _mm_add_epi32(sum_vec, tmp);
11 // sum_vec = {3, 1, 4, 1}
12
13 // Load array elements 4-7 into a temporary vector register
14 tmp = _mm_loadu_si128((__m128i *) (arr + 4));
15 // Add to existing sum vector
16 sum_vec = _mm_add_epi32(sum_vec, tmp);
17 // sum_vec = {3 + 5, 1 + 9, 4 + 2, 1 + 6}
18
19 // Create temporary array to hold values from sum_vec
20 // Store the vector into an array to access the individual values
21 int tmp_arr[4];
22 _mm_storeu_si128((__m128i *) tmp_arr, sum_vec);
23 // Collect values from sum_vec in a single integer
24 int sum = tmp_arr[0] + tmp_arr[1] + tmp_arr[2] + tmp_arr[3];
```



SIMD Functions

- `__m128i _mm_and_si128(__m128i a, __m128i b)`
  - returns vector (a & b), where & represents the bitwise AND operator
- `__m128i _mm_cmpgt_epi32(__m128i a, __m128i b)`
  - returns the vector ($a_n > b_n$? 0xffffffff : 0x0 for n from 0 to 3). AKA a 32-bit all-1s mask if $a_n > b_n$ and a 32-bit all-0s mask otherwise

a	0x3	0x4	0x5	0x6
b	0x0	0x5	0x6	0x3
_mm_cmpgt_epi32()	0xFFFFFFFF	0x0	0x0	0xFFFFFFFF

Ex3 and Ex4 time