SSSE3

Supplemental Streaming SIMD Extensions 3

Sybrandt, Bowman, Jarvis

What is the SSSE3?

- **SSSE3 was first introduced with Xeon 3000 (“Woodcrest”, 2006)**
- **Two main goals:**
  - **Improve upon SSE3 by improving...**
  - **Accelerate the computation of packed integers**

(Was called SSE4 during the development of the Core microarchitecture)

CPUs with SSSE3

- AMD
  - Bobcat
  - Bulldozer
  - Piledriver
- Intel
  - Xeon 3000 (“Woodcrest”), 5100, 5300
  - Core 2 Duo, Extreme, Quad
  - Core i3, i5, i7
  - Pentium Dual Core
  - Celeron
  - Atom

The Instructions

- SSSE3 has a total of 32 instructions
  - Only 16 of them are new
  - 6 of the new 16 instructions are variants
  - That is a whopping total of 10 (practically) new instructions!!!

Some Instructions

- Packed Sign
  - Flips the sign of register A, if register B is negative
  - $A = [- a_0\ a_1\ \ldots], B = [- b_0\ b_1\ \ldots] \to R = [a_0\ \ldots]$

- Packed Absolute Value
  - Fills register A with the absolute value of bytes in B
  - $A = [abs(b_0)\ abs(b_1)\ \ldots]$

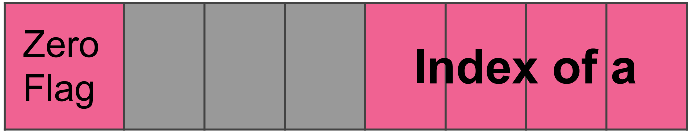
More Instructions

- Multiply and Add
  - Multiples bytes in A with the respective byte in B
  - Then adds the byte pairs
  - $R = [(A_0B_0 + A_1B_1)\ (A_2B_2 + A_3B_3)\ \ldots]$
- Packed Horizontal Add/Subtract
  - Adds/Subtracts bytes
  - $A = [A_0 + A_1, A_2 + A_3, \ldots]$ $B = [B_0 + B_1, B_2 + B_3, \ldots]$
  - Then concatenates the results

Shuffle

- _mm_shuffle_epi8(__m128i **a**, __m128i **b**)
    - **a** - initial data
    - **b** - selection array
    - **r** is the resulting data from **a** according to **b**.
- Pseudocode:
    - ```
      for (i = 0; i < 16; i++){
       if (b[i] & 0x80){
        r[i] =  0;
       }
       else {
        r[i] = a[b[i] & 0x0F];
       }
      }
      ```

Visualization of a byte in **b**.


      ```

Shuffle Visualization

- Example: Every other byte.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
a:																
b:	01	03	05	07	09	0B	0D	0F	80	80	80	80	80	80	80	80
r:																

Shuffle Visualization

- Example: AES shift.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
a:																
b:	00	01	02	03	05	06	07	04	0A	0B	08	09	0F	0C	0D	0E
r:																

Concatenation

- _mm_alignr_epi8(__m128i **a**, __m128i **b**, int **n**)
  - concatenates **a** and **b**
  - shifts the resulting 256 bits by **n** bytes
  - **r** is the resulting 128 bits
- Pseudocode
  - ```
    t1[255:128] = a;
    t1[127:0] = b;
    t1[255:0] = t1[255:0] >> (8 * n); // unsigned shift
    r[127:0] = t1[127:0];
    ```
    ```

Concatenation Visualization

a = `0xAAAAAAAAAAAAAAAA`

b = `0xBBBBBBBBBBBBBBBBB`

n = 5

Concatenation Visualization

a = 0xAAAAAAAAAAAAAAAA

b = 0xBBBBBBBBBBBBBBBBB

n = 5

temp = 0xAAAAAAAAAAAAAAAA BBBBBBBBBBBBBBBBB

Concatenation Visualization

a = 0xAAAAAAAAAAAAAAAA

b = 0xBBBBBBBBBBBBBBBBB

n = 5

temp = 0xAAAAAAAAAAAAAAAA BBBBBBBBBBBBBBBBB

temp >> n = 0x00000AAAAAAAAAAA AAAAABBBBBBBBBBBB

Concatenation Visualization

a = 0xAAAAAAAAAAAAAAAA

b = 0xBBBBBBBBBBBBBBBBB

n = 5

temp = 0xAAAAAAAAAAAAAAAA BBBBBBBBBBBBBBBBB

temp >> n = 0x00000AAAAAAAAAAA AAAAABBBBBBBBBBBB

r = 0xAAAAABBBBBBBBBBBB

Packed Multiply High with Round and Scale (mulhrs)

- _mm_mulhrs_epi16(__m128i **a**, __m128i **b**);
    - **a** and **b** are treated as arrays signed numbers between −1 and 1.
    - Each pair of numbers is multiplied and rounded, with the results stored in **r**.
- Bit Encoding:

Sign	1/2	1/4	1/8	1/16	1/32	1/64	1/128	...	...	...	...	...	...	...	...

- Examples:
    - 0x4000 = $0100\ 0000\ 0000\ 0000\ 0000_2$ = 0.5
    - 0xd000 = $1101\ 0000\ 0000\ 0000\ 0000_2$ = -0.375
    - 0x8000 = $1000\ 0000\ 0000\ 0000\ 0000_2$ = -1
    - 0x7FFF = $0111\ 1111\ 1111\ 1111\ 1111_2$ = 0.99993896484375 ≈1

Packed Multiply High with Round and Scale (mulhrs)

- **Pseudocode**
  - ```
    for (i = 0; i < 8; i++) {
    r[i] =  (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
    }
    ```
- **Example Problem:**
  - **a** = 0x4000 = 0.5
  - **b** = 0xD000 = -0.375
    ```

Packed Multiply High with Round and Scale (mulhrs)

Specifics Unimportant

- **Pseudocode**
  - ```
    for (i = 0; i < 8; i++) {
    r[i] =  (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
    }
    ```
- **Example Problem:**
  - **a** = 0x4000 = 0.5
  - **b** = 0xD000 = -0.375
  - **a** * **b** = 0xE800 0000
    ```

Packed Multiply High with Round and Scale (mulhrs)

Discard Least Significant Bits

- **Pseudocode**
  - ```
    for (i = 0; i < 8; i++) {
    r[i] =  (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
    }
    ```
- **Example Problem:**
  - **a** = 0x4000 = 0.5
  - **b** = 0xD000 = -0.375
  - **a** * **b** >> 14 = 0x1 D000
    ```

Packed Multiply High with Round and Scale (mulhrs)

Round

- **Pseudocode**
  - ```
    for (i = 0; i < 8; i++) {
    r[i] =  (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
    }
    ```
- **Example Problem:**
  - **a** = 0x4000 = 0.5
  - **b** = 0xD000 = -0.375
  - (**a** * **b** >> 14) +1 = 0x1 D001
    ```

Packed Multiply High with Round and Scale (mulhrs)

Discard
Last Bit

- **Pseudocode**
  - ```
    for (i = 0; i < 8; i++) {
    r[i] =  (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
    }
    ```
- **Example Problem:**
  - **a** = 0x4000 = 0.5
  - **b** = 0xD000 = -0.375
  - ((**a** * **b** >> 14) +1)>>1 = 0xE800
    ```

Packed Multiply High with Round and Scale (mulhrs)

- **Pseudocode**
  - ```
    for (i = 0; i < 8; i++) {
    r[i] =  (( (int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
    }
    ```
- **Example Problem:**
  - **a** = 0x4000 = 0.5
  - **b** = 0xD000 = -0.375
  - **r** = 0xE800 = -0.1875
    ```

Experimental Overview

- 3 instructions from SSE3 to focus on
  - _mm_shuffle_epi8()
  - _mm_alignr_epi8()
  - _mm_mulhrs_epi16()
- Timing using StopWatch class
  - average over 1,000,000 iterations
- Compared software and hardware implementations
- All tests conducted in release mode x64 builds
- Randomized data input between each iteration

Results: _mm_shuffle_epi8

- Vastly improved by SSSE3 implementation
- Nearly 14x faster when implemented through SSSE3
  - Avg 83 ns for each software iteration
  - Avg 6 ns for each hardware iteration

```
//software implementation
__m128i SSSE3Helper::shufflePseudo(unsigned char (&a)[16], unsigned char (&b)[16]){
      unsigned char r[16];

      for (int i = 0; i<16; i++){
            if (b[i] & 0x80)r[i] = 0;
            else r[i] = a[b[i] & 0x0F];
      }

      return _mm_loadu_si128((__m128i*)&r[0]);
};
```

Results: _mm_alignr_epi8

- Nearly 12x faster when implemented through SSSE3
  - Avg 24 ns for each software iteration
  - Avg 2 ns for each hardware iteration

```
//software implementation
__m128i SSSE3Helper::alignrPseudo(unsigned char (&a)[16], unsigned char (&b)[16], int n){

	unsigned char t1[32];//an array of 32 bytes, totaling 256 bits

	for (int i = 0;i<16;i++){
		t1[i+16] = a[i];//put a in the second half of the array
		t1[i] = b[i];//put b in the first half of the array
	}

	for (int i = 0;i<32;i++){
		t1[i] = t1[i+n];//shift entire bytes down the array.
	}
	return _mm_loadu_si128((__m128i*)&t1[0]);
};
```

Results: _mm_mulhrs_epi16

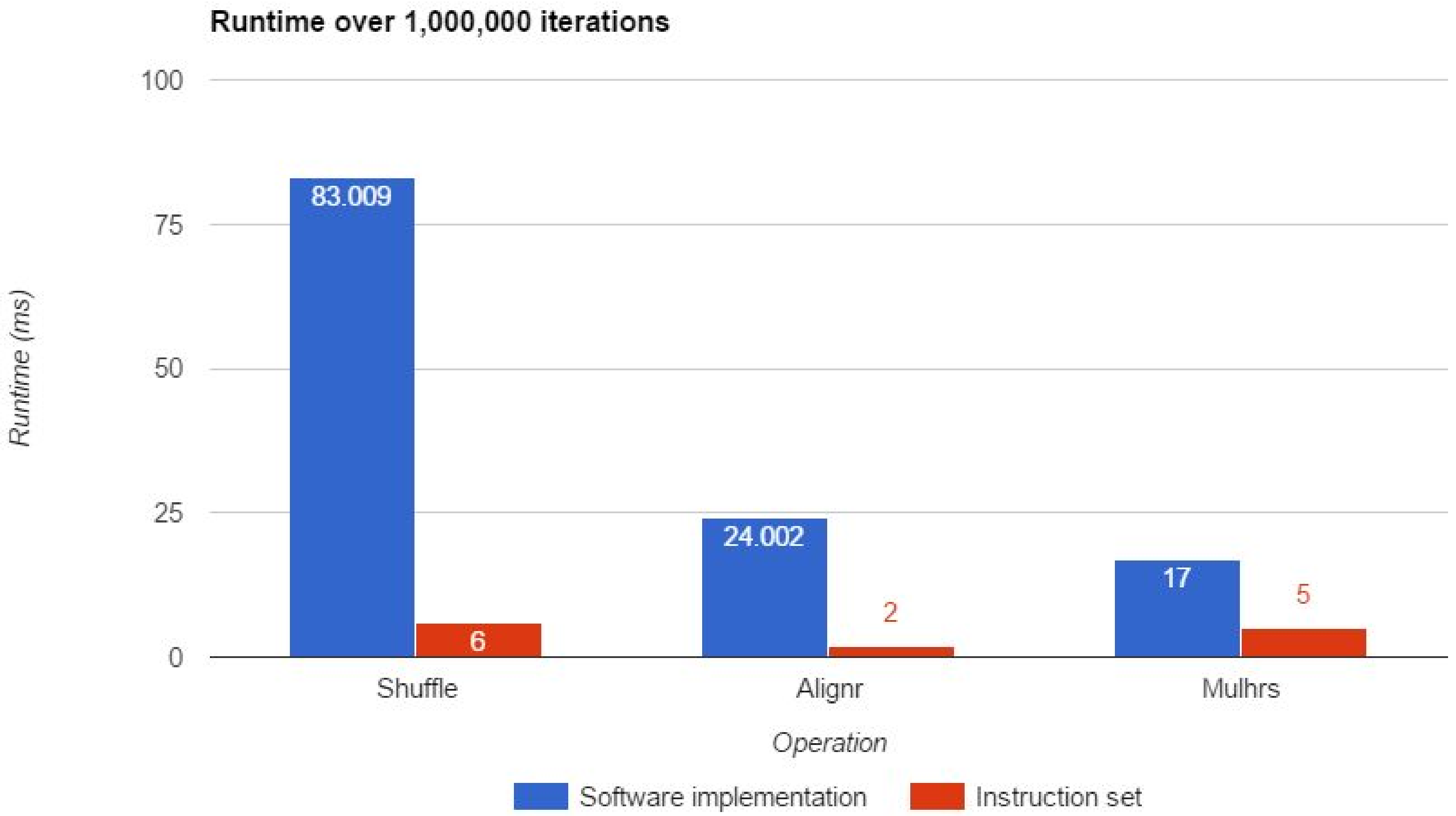
- About 3x faster when implemented through SSSE3
  - Avg 17 ns for each software iteration
  - Avg 5 ns for each hardware iteration

```
//software implementation
__m128i SSSE3Helper::mulhrsPseudo( signed short (&a)[8], signed short (&b)[8]){

	signed short r[8];

	for (int i = 0; i < 8; i++) {
		r[i] =  (( (int)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
	}

	return _mm_loadu_si128((__m128i*)&r[0]);
};
```


Runtime over 1,000,000 iterations
100
75
50
25
0
Runtime (ms)
83.009
6
24.002
2
17
5
Shuffle
Alignr
Mulhrs
Operation
Software implementation
Instruction set

Use Case: Employee Schedule

- Create work schedule patters for a list of employees and managers
  - Regular week work schedule template, holiday work schedule template
- Run employee list and manager list through shuffle using mask
  - Mask off last 13 bytes of manager shuffle, mask off first 3 bytes of employee shuffle
- Concatenate using alignr to build a composite schedule including 3 managers and 13 employees
  - Shift right 3 bytes using alignr and truncate to a 128 bit string

Use Case: Employee Schedule

- Manager list - sequential IDs starting with 0xa1
- Employee list - sequential IDs starting with 0x01
- Compomposite shuffled list contains 13 employees and 3 managers

0x000000130fb1
0xa1 '¡'
0xa2 '¢'
0xa3 '£'
0xa4 '¤'
0xa5 '¥'
0xa6 '¦'
0xa7 '§'
0xa8 '¨'
0xa9 '©'
0xaa 'ª'
0xab '«'
0xac '¬'
0xad '-'
0xae '®'
0xaf '¯'
0xb0 '°'

0x000000130fb1
0x01 '\x1'
0x02 '\x2'
0x03 '\x3'
0x04 '\x4'
0x05 '\x5'
0x06 '\x6'
0x07 '\a'
0x08 '\b'
0x09 '\t'
0x0a '\n'
0x0b '\v'
0x0c '\f'
0x0d '\r'
0x0e '\xe'
0x0f '\xf'
0x10 '\x10'

r	{m128i_i8=0
m128i_i8	0x000000130
[0x00000000]	0x0d '\r'
[0x00000001]	0x0c '\f'
[0x00000002]	0x0c '\f'
[0x00000003]	0x04 '\x4'
[0x00000004]	0x01 '\x1'
[0x00000005]	0x0d '\r'
[0x00000006]	0x10 '\x10'
[0x00000007]	0x02 '\x2'
[0x00000008]	0x0a '\n'
[0x00000009]	0x0e '\xe'
[0x0000000a]	0x0e '\xe'
[0x0000000b]	0x07 '\a'
[0x0000000c]	0x0c '\f'
[0x0000000d]	0xa3 '£'
[0x0000000e]	0xaa 'ª'
[0x0000000f]	0xad '-'

Results: Employee Schedule

- 1.2x faster when implemented through SSSE3
  - Avg 345 ns for each software iteration
  - Avg 287 ns for each hardware iteration

- Times include the whole process of schedule generation, start to finish
  - Employee and Manager list creation, shuffle, concatenation, and result

References

[1] Software.intel.com, "Overview: Supplemental Streaming SIMD Extensions 3 (SSSE3) | Intel® Developer Zone", 2016. [Online]. Available: https://software.intel.com/en-us/node/524210. [Accessed: 04- Feb- 2016].

[2] M. Cornea, "Intel® AVX-512 Instructions and Their Use in the Implementation of Math Functions", 2015.

[3] *Intel® Architecture Instruction Set Extensions Programming Reference*, 1st ed. 2015.

[4] Wikipedia, "SSSE3", 2016. [Online]. Available: https://en.wikipedia.org/wiki/SSSE3. [Accessed: 04- Feb- 2016].

[5] L. Wenyan, "Android* - Using the Intel® SSSE3 Instruction Set to Accelerate DNN Algorithm in Local Speech Recognition | Intel® Developer Zone", *Software.intel.com*, 2015. [Online]. Available: https://software.intel.com/en-us/android/articles/using-the-intel-ssse3-instruction-set-to-accelerate-dnn-algorithm-in-local-speech. [Accessed: 04- Feb- 2016].