SSE

Review

- We've seen SSE intrinsics and datatypes (int, float, double)
- We've seen float, double, and integer operations
- Now we'll discuss comparison operations

Comparison

- SSE has several comparison functions
  - Numeric
  - String
- First, we look at ones for numeric data

Compare

- Compare four floats:
- v3 = _mm_cmpeq_ps(v1,v2)
  - v3[0] = 0xffffffff if v1[0]==v2[0], 0 otherwise
  - Same for other three slots
- We also have cmpge (≥), cmpgt (>), cmple (≤), cmplt (<), cmpne (≠)
- This gives us an easy way to set a register to all one's: v = _mm_cmpeq_ps(v,v);

Compare

- We also have integer comparisons:
- _mm_cmpeq_epi{8,16,32,64}
  - Equality
  - Sets slots to all-ones or all-zeros
- _mm_cmpgt_epi{8,16,32,64}
  - Greater-than test
  - Same idea: Set slots to all-ones or all-zeros
- _mm_cmplt_epi{8,16,32,64}
  - less-than test
  - Same idea: Set slots to all-ones or all-zeros

Strings

- What about strings? Or arbitrary chunks of data?
- Two kinds of buffers: Explicit-length vs. implicit length
- Explicit = We know string length beforehand
  - Typically, software stores length of string separate from string data
  - Ex: C++ string type
- Implicit: We don't know string length without looking for sentinel
  - C strings: \0 (null) byte
- SSE supports both forms

String Ops

- Four functions
- Obtain an index:
  - int _mm_cmpestri(__m128i a, int la, __m128i b, int lb, int flags)
    - Explicit length
  - int _mm_cmpistri(__m128i a, __m128i b, int flags)
    - Implicit length: CPU scans for null byte
- Obtain a mask:
  - __m128i _mm_cmpestrm(__m128i a, int la, __m128i b, int lb, int flags)
  - __m128i _mm_cmpistrm(__m128i a, __m128i b, int flags)
- We'll tackle the mask ones first

Flags

- Flags is a bitwise-or of several values
- First thing: Need to choose size for comparison
- Choose one of:
  - _SIDD_UBYTE_OPS: unsigned bytes
  - _SIDD_SBYTE_OPS: signed bytes
  - _SIDD_UWORD_OPS: unsigned shorts
  - _SIDD_SWORD_OPS: signed shorts
- For character strings, usually use _SIDD_UBYTE_OPS
  - We'll assume that for the following discussion

Result

- Choose whether result should be returned as a bit mask or byte mask
  - If flags is OR'd with _SIDD_UNIT_MASK:
    - If input size is byte: Output 16 bytes, each one 0 or 255
    - If input size is short: Return 8 shorts, each one 0 or 65535
  - If flags is OR'd with _SIDD_BIT_MASK:
    - If input size is a byte: Result is 16 bits, with upper bits all zero
    - If input size is short: Result is 8 bits, with upper bits all zero
- Following slides refer to output "slot"
  - Will be either a single bit or a single byte or a single short, depending on flags

Operation

- Next, choose a comparison operation and bitwise-OR that into flags
- Choose one of
  - _SIDD_CMP_EQUAL_ANY,
  - _SIDD_CMP_RANGES,
  - _SIDD_CMP_EQUAL_EACH,
  - _SIDD_CMP_EQUAL_ORDERED

Flags

- Easiest one to understand: Flags was OR'd with _SIDD_CMP_EQUAL_EACH
- Then: Slot i of output is true iff a[i] == b[i]
- This is a "string equal" operation
- Note: If i is past end of both strings, slot i gets true
  - If i is only past end of one string, slot i gets false

Flags

- Suppose instead flags included bitwise OR of _SIDD_CMP_EQUAL_ANY
- Slot i of output is true iff b[i] is found somewhere in a
- Bytes past end of either string don't count as equal

Flags

- Suppose flags included bitwise OR of _SIDD_CMP_EQUAL_ORDERED
  - Slot 0 of output is true iff a appears at start of b
  - Slot 1 of output is true iff a appears at b[1:]
  - Slot 2 of output is true iff a appears at b[2:]
  - ...
- For all slots past end, set slot to false

Flags

- Suppose flags included bitwise OR of _SIDD_CMP_RANGES
- Consider a to be a list of character pairs
- Return true for each slot i where b[i] is within any of the ranges defined by the character pairs in a
- This will make more sense when we see an example...

Other Flags

- There are some other tweaks that can be made
  - If flags includes _STDD_NEGATIVE_POLARITY: Invert results
  - If flags includes _SIDD_MASKED_NEGATIVE_POLARITY: Invert results for indices 0,1,2,...,len(b)

Wow

- Wow! That's a lot to absorb
- How can we use instructions like this?
- Let's see some examples

Example

- String equality
- Example program:

```
alignas(16) char aa[] = "pack my box with";
alignas(16) char bb[] = "get thy box\0\0\0\0";
int la = strlen(aa);
int lb = strlen(bb);
__m128i a = _mm_load_si128((__m128i*)aa);
__m128i b = _mm_load_si128((__m128i*)bb);
__m128i res = _mm_cmpestrm(a,la,b,lb,
    _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_UNIT_MASK );
alignas(16) char rr[16];
_mm_store_si128((__m128i*)rr,res);
```

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_UNIT_MASK
- Result:

```
a=pack my box with
b=get thy box
  FFFFFFTTTTTFFFFF
```

- Notice: true wherever strings are equal
  - The ‘T’ represents “this slot of output contains eight one bits”
  - A ‘F’ means “this slot of output contains eight zero bits”

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_UNIT_MASK
- Result:

```
a=pack my bag wit
b=get zat box
  FFFFFFFTTFFFFFFT
```

- Notice that when we're past end of string, compares as true

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_BIT_MASK
- Result:

```
a=pack my box with
b=get thy box
c0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

- Here, the output is not just 0 or 255 in each slot
  - Take low 16 bits of result: 0xc0, 0x7, 0, 0 (little endian)
  - That's 0x7c0 = 0b0000 0111 1100 0000
  - Convert to binary (msb corresponds to end of strings):

```
0000011111000000
htiw xob ym kcap
     xob yht teg
```

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_ANY | _SIDD_UNIT_MASK
- Result:

```
a=pack my bag wit
b=get zat box
  101101111000000
```

- Since a contains 'g', 't', ' ', 'b', those output slots are true
- Since a does not contain 'e', 'z', 'o', or 'x', those slots are false
- Slots corresponding to locations past end of either string are false

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_ORDERED | _SIDD_UNIT_MASK
- Result:

```
a=my bag
b=pack my bag, sir
  0000010000000000
```

- Notice the ‘1’ appears where ‘a’ first appears in b

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_ORDERED | _SIDD_UNIT_MASK
- Result:

```
a=my bag
b=my bag! my bag!!
  1000000010000000
```

- Since two matches, two 1's

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_ORDERED | _SIDD_UNIT_MASK
- Result:

```
a=foofoo
b=foofoofoofoo
  1001001000000000
```

- Overlapping matches are detected as separate matches

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_ORDERED | _SIDD_UNIT_MASK
- Result:

```
a=my bag!
b=my bag is mine
  0000000000000000
```

- Since the ! does not appear in b, no match

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_RANGES | _SIDD_UNIT_MASK
- Result:

```
a=am
b=pack my box with
  0111010010000101
```

- Returns true for each slot where character from b is in {a,b,c,...,l,m}

Result

- Using flags = _SIDD_UBYTE_OPS | _SIDD_CMP_RANGES | _SIDD_UNIT_MASK
- Result:

```
a=aemmwz
b=pack my box with
  0110011010101000
```

- Returns true where any character of b is in {a,b,c,d,e} or {m} or {w,x,y,z}

Index

- What about the index functions?
  - int _mm_cmpestri(__m128i a, int la, __m128i b, int lb, int flags)
  - int _mm_cmpistri(__m128i a, __m128i b, int flags)
- Operation is similar to mask, but returns an index
  - If flags includes _SIDD_LEAST_SIGNIFICANT: Return index of first place where comparison succeeded
  - If flags includes _SIDD_MOST_SIGNIFICANT: Return index of last place where comparison succeeded

Example

- Using int res = _mm_cmpestri(... , _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_LEAST_SIGNIFICANT);
- Result:

```
a=crack a boxcar
b=pack my box with
res=7
```

- Match starts at character 7 (the space character before ‘box’)

Example

- Using int res = _mm_cmpestri(... , _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_MOST_SIGNIFICANT);
- Result:

```
a=crack a boxcar
b=pack my box with
res=10
```

- Last match is at index 10

Example

- Using int res = _mm_cmpestri(... , _SIDD_UBYTE_OPS | _SIDD_CMP_EQUAL_EACH | _SIDD_MOST_SIGNIFICANT);
- Result:

```
a=crackerjack
b=pack my box with
res=16
```

- 16 because no match (we're using EQUAL_EACH here)

Uses

- EQUAL_EACH is like strcmp()
- EQUAL_ANY is like strspn()
- EQUAL_ORDERED is like strstr()
- EQUAL_RANGES doesn't have a direct C equivalent

Assignment

- Often, sound recordings contain noise in areas that should be silent
- Write a program to fix this using SSE:
  - It should take two command line arguments: The filename of a WAV file and a silence threshold
  - Read the WAV file. Any values within “threshold” of silence should be squelched to silence
    - For u8: samples are in range 0...255, so silence == 127
    - For s16, samples are in range -32768...32767, so silence == 0
  - Write output to the file “silenced.wav”
- Your program should work with u8 and s16 WAV’s
- Example code: silence.cpp

Sources

- https://software.intel.com/en-us/articles/intel-software-development-emulator#faq
- http://www.tomshardware.com/reviews/intel-drops-pentium-brand,1832-2.html
- https://www.mathworks.com/matlabcentral/answers/93455-what-is-the-sse2-instruction-set-how-can-i-check-to-see-if-my-processor-supports-it?requestedDomain=www.mathworks.com
- http://softpixel.com/cwright/programming/simd/ssse3.php
- https://software.intel.com/sites/default/files/m/8/b/8/D9156103.pdf
- https://msdn.microsoft.com/en-us/library/y08s279d(v=vs.100).aspx
- http://www.linuxjournal.com/content/introduction-gcc-compiler-intrinsics-vector-processing?page=0,2
- http://en.cppreference.com/w/cpp/language/alignas
- https://software.intel.com/sites/landingpage/IntrinsicsGuide/
- MSDN SSE reference
- https://wiki.multimedia.cx/index.php/YUV4MPEG2
- http://ok-cleek.com/blogs/?p=20540
- http://www.liranuna.com/sse-intrinsics-optimizations-in-popular-compilers/
- https://www.cilkplus.org/tutorial-pragma-simd
- A. Ortiz. “Teaching the SIMD Execution Model: Assembling a Few Parallel Programming Skills.” Proceedings of 2003 ACM SIGCSE.
- Nasm documentation.
- Greggo. x86 - SSE Compare Packed Unsigned Bytes. https://stackoverflow.com/questions/16204663/sse-compare-packed-unsigned-bytes
- John D. Cook. Converting color to grayscale. https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org