• HW0 grades & HW2 out.

• HW1 peer-reviews out today.

• Get Cython installed & working for HW2.
  • Test repo posted - please give feedback!
Peer Reviews

Added my name #2

raahilsha wants to merge 1 commit into harvard-cs205:master from raahilsha:HW0a

Files changed 1

Showing 1 changed file with 1 addition and 0 deletions.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ My name is Raahil Sha

thouis added a note 28 days ago

Hi Raahil, just testing the PR commenting feature.

Add a line note
SIMD Extensions
(MMX, SSE, AVX, ...)
SIMD

- “Single Instruction, Multiple Data”
  - Perform the same operation on several values at once.
  - In modern CPUs, special vector registers and instructions.
Why SIMD?
Why SIMD?

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp
Why SIMD?

• More transistors, same clock speed.
• Increased demand for graphics and sound.
• SIMD hardware is simple - Cores are complex.
• SIMD is efficient - Cores can be...
  • N parallel operations on N values = same time as for 1 value.
History

- MMX (1997) 8 64-bit registers (integer only)
- SSE (1999) 8 128-bit registers
- SSE2 (2001) 16 128-bit registers
  - ???
- AVX (2008) 16 256-bit registers
- AVX-512 (2015) 32 512-bit registers
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp
Performance

~8x
How?

- Let’s concentrate on AVX:
  - 256 bits:
    - 4x 64-bit doubles
    - 8x 32-bit floats
    - (or 16-bit floats, and with AVX 2, integer types)

<table>
<thead>
<tr>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
</tr>
</thead>
</table>
__m256 test(__m256 a, __m256 b) {
    // compute (10 * a + b) for 8 values simultaneously

    // load the value 10.0 into all 8 values
    __m256 const_10 = _mm256_set1_ps(10.0);

    // multiply a * 10
    __m256 a_10 = _mm256_mul_ps(a, const_10);

    // add b and return
    return _mm256_add_ps(a_10, b);
}
AVX example

__m256 test(__m256 a, __m256 b) {
    // compute (10 * a + b) for 8 values simultaneously

    // load the value 10.0 into all 8 values
    __m256 const_10 = _mm256_set1_ps(10.0);

    // multiply a * 10
    __m256 a_10 = _mm256_mul_ps(a, const_10);

    // add b and return
    return _mm256_add_ps(a_10, b);
}
AVX example

__m256 test(__m256 a, __m256 b) {
    // compute (10 * a + b) for 8 values simultaneously
    // load the value 10.0 into all 8 values
    __m256 const_10 = _mm256_set1_ps(10.0);

    // multiply a * 10
    __m256 a_10 = __mm256_mul_ps(a, const_10);

    // add b and return
    return _mm256_add_ps(a_10, b);
}

ps = “packed single”
AVX instructions

A useful subset:

_mm256_add_ps         _mm256_sqrt_ps
_mm256_sub_ps         _mm256_and_ps
_mm256_div_ps         _mm256_max_ps
_mm256_mul_ps         _mm256_min_ps
_mm256_ceil_ps        _mm256_set_ps
_mm256_floor_ps       _mm256_set1_ps
_mm256_cmp_ps         _mm256_movemask_ps
AVX instructions

A useful subset:

```
add      sqrt
sub      and
div      max
mul      min
ceil     set
floor    set1

cmp      movemask
```
Creating

- set1(val)
- set(val7, val6, val5, ..., val0)
  - val0 ends up at LSB, val7 at MSB
Comparing

- \( \text{comp}(a, b, \text{OP}) \)
  - \( \text{OP} = \_\text{CMP}_\text{EQ}_\text{OQ} \)
    - \( \text{EQ} = \text{equal} \)
    - \( \text{O (or U)} = \text{what about comparing to NaN} \)
    - \( \text{Q (or S)} = \text{quiet or signaling} \)
Results of comparison

- True = all 1s = 0xFFFFFFFF = -NaN
- False = all 0s = 0x00000000 = 0.0
Using Comparisons

• `and(val1, val2) = (val1 & val2)`

  • bitwise AND

  • If all -1.0 and 0.0, masks where True (-NaN)

• `movemask(val) -> integer`

  • extract high bits

  • high bit = sign, so gives bitmask of True/False
AVX-512

- 32 x 512-bit registers (twice as many as AVX)
- 16 single-precision floats
- Putting pressure on GPUs from below…
Coming Up

- CPU SIMD —> GPUs

- Review on Wednesday, after you have a chance to look at HW2.

- Friday: Intro to Odyssey (Havard cluster)