ECE 574 – Cluster Computing
Lecture 7

Vince Weaver

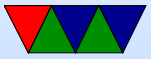
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

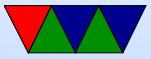
14 February 2017

Announcements

- Too many snow days
- Will post a video of this lecture

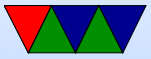


Homework #4 Review

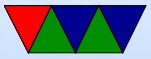


General Comments

- Comment your code!
- Don't ignore compiler warnings!
- Issues I saw:
  - Make sure stay in bounds of array
    We are iterating 1 to x-1 not 0 to x
  - You need to saturate to 255 in combine function too
    sqrt(255*255+255*255) is greater than 255.
    If you wrap around in 8-bits your results will be off.
  - Check errors on PAPI!

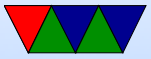


Not the best interface, but it will let you know if adding improper events.



General C array Comment

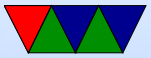
- Why a linear array and not multidimensional?
  C doesn't do dynamically sized multi-dimensional well
- It's all a fiction anyway. You always get a linear array. The multiply/add we are doing is what the compiler does behind the scenes.
- You might think the whole multiply/add stuff would kill performance, but the compiler can often reduces it to just one CPU instruction on x86 (the magical load effective address `lea` instruction



Butterfinger Results

Butterfinger was a pet guinea pig from long ago.

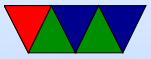
```
time ./sobel ./butterfinger.jpg
output_width=320, output_height=320, output_components=3
SOBELX  L3 CACHE MISSES: 1554   CYCLES 9436089
SOBELY  L3 CACHE MISSES: 0      CYCLES 9362614
COMBINE L3 CACHE MISSES: 3      CYCLES 6574264

real    0m0.048s user    0m0.024s sys     0m0.004s
```



- Why 0 cache misses for SOBELY?
  Cache. 320*320*3=307k
  IN, SOBEL_X, SOBEL_Y, COMBINED, so 300k*4 = 1.2MB or so

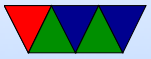
- Haswell-EP has 20MB of L3 cache

- Reading causes misses to read input in, rest are writing out so while not necessarily hits, with write allocate cache do not seem to be accounted for as misses



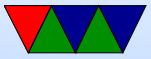
Brief Cache Overview

- Haswell-EP caches
  memory – 200+ cycles best case 20MB of L3, 20MB, 64B/line (30-60 cycles?)
  256kB per-core L2, 64B/line, 8-way (12-cycles)
  32kB per-core L2, 64B/line, 8-way (4 cycles)
- Chunks of fast memory close to CPU
- Multiple levels
- Memory broken up into cacheline sized chunks (64-byte



on HSW-EP)

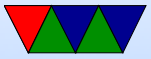
- When access an address, all 64-B brought in even if not need rest
- When cache full, something is kicked out to make room (usually oldest)
- Want to take advantage of spatial and temporal locality
- With butterfinger all fits in L3 cache



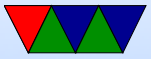
Cake, Straight implementation of pseudo-code

Yes, boring image, but just happened to be a high-res image I had around.

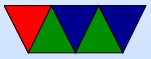
```
time ./sobel ./IMG_1733.JPG
output_width=3888, output_height=2592, output_components=3
SOBELX  L3 CACHE MISSES: 713,241        CYCLES 1,554,165,544
SOBELY  L3 CACHE MISSES: 697,464        CYCLES 1,539,635,869
COMBINE L3 CACHE MISSES: 1,252,635      CYCLES 1,182,455,505

real    0m1.601s user    0m1.476s sys     0m0.072s
```



```
perf report

  61.65%  sobel    sobel                [.] generic_convolve
  23.45%  sobel    sobel                [.] main
   1.11%  sobel    [kernel.kallsyms]    [k] clear_page_c_e
   0.51%  sobel    libjpeg.so.62.2.0    [.] jpeg_fill_bit_buffer
   0.51%  sobel    [kernel.kallsyms]    [k] page_fault

perf annotate
                                  sum += filter[0][2]*(input_image->p
  0.61        movslq %r11d,%r11
  0.66        movzbl (%rcx,%r11,1),%esi
           convert():
                  return (y*xsize*depth)+(x*depth)+color;
 42.22        lea    (%r9,%rbx,1),%r11d
           generic_convolve():
```



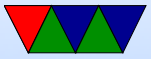
- converts down to one instruction
- skid? probably the memory load previously is the problem (`lea is just a shift/add`)
- 3888*2952=10M*3, 30MB (larger than L3)



Loop Order Optimization

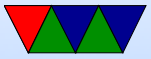
- How is an array laid out in memory?
  Row-major (C) vs Column-major (Fortran)
- Default with loop x then y, are actually walking columns.
  Worst case.
- Switch order of loops, things get a lot better.

```
time ./sobel_improved ./IMG_1733.JPG
output_width=3888, output_height=2592, output_components=3
SOBELX  L3 CACHE MISSES: 21,246 CYCLES 882,000,608
SOBELY  L3 CACHE MISSES: 19,556 CYCLES 881,998,207
COMBINE L3 CACHE MISSES: 1,241,446      CYCLES 1,183,759,970

real    0m1.181s user    0m1.112s sys     0m0.052s
```



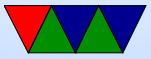
Loop Unrolling

- Loop unrolling. Unroll the color loop (explicitly do the three things 0, 1, 2 and put the values in.
- Can have benefits. Change all occurences of "color" to be a constant, which can be optimized.
- Remove branches, which can be slow or mispredicted.
- More code for out-of-order processor to work with and try to do in parallel.
- Downsides: too large: no longer fit in instruction cache or loop stream detector.



Other Optimizations

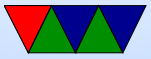
- Other optimizations, often are things the compiler does for you with -O2.
- Hoisting (move things out of loop that only need to be done once)
- Simplification. Lots of things.
- Take a compiler class.



Convert to one single Loop

No need to iterate X and Y and Color, just walk through output linearly. Really you have three pointers of input (line above, current line, below).

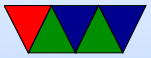
```
time ./sobel_improved ./IMG_1733.JPG
output_width=3888, output_height=2592, output_components=3
SOBELX  L3 CACHE MISSES: 15,703 CYCLES 411,148,087
SOBELY  L3 CACHE MISSES: 15,334 CYCLES 411,284,853
COMBINE L3 CACHE MISSES: 1,245,842      CYCLES 1,186,204,125

real    0m0.924s user    0m0.848s sys     0m0.044s
```



Same for Combine

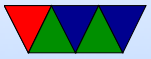
No need to offset, just start at beginning of x and y and write to output, doing the combine operation.

```
time ./sobel_improved ./IMG_1733.JPG output_width=3888, output_height=2592, out$
L3 CACHE MISSES: 16,188 CYCLES 410,983,833
L3 CACHE MISSES: 14,850 CYCLES 411,059,831
L3 CACHE MISSES: 36,652 CYCLES 496,394,104

real    0m0.690s
user    0m0.628s
sys     0m0.040s
```



ISRA= interprocedural scalar replacement of aggregates,

```
  39.71%  sobel_improved  sobel_improved      [.] generic_convolve.isra.0
  24.51%  sobel_improved  sobel_improved      [.] main
   2.41%  sobel_improved  [kernel.kallsyms]   [k] clear_page_c_e
   1.23%  sobel_improved  libjpeg.so.62.2.0   [.] jpeg_fill_bit_buffer
   1.02%  sobel_improved  libjpeg.so.62.2.0   [.] 0x0000000000039356
   0.83%  sobel_improved  [kernel.kallsyms]   [k] page_fault
```

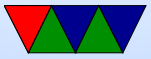


SIMD (SSE/AVX)

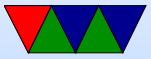
- SIMD = Single Instruction, multiple data
  One instruction (say add) can add multiple values at once

- On intel chips SSE, SSE2, etc. Up to AVX/AVX2 on newer systems

- 256-bit wide registers. So sixteen 16-bit values (can do integer), Four 64-bit doubles, etc.

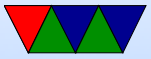


- Large number of these registers, xmm0 (128bit) ymm0 (256bit) zmm0 (512bit on newer machines)
- One way is to program in assembly language with some obscure opcodes: an example `PMADDWD` 16-bit integer parallel 128-bit multiply and add
- On recent gcc and other compilers there are "intrinsics" to use in C, for example you can use `_mm_madd_epi16()` to do a `PMADDWD` instruction



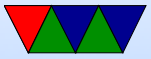
Initial SIMD try

```
9 values from the three input pointers (16-bit)
A B C X D E F X G H I X X X X X
The sobel filter values (16-bit)
 1 2 3 0 4 5 6 0 7 8 9 0 0 0 0 0
Multiply and add all in parallel
A1+B2 C3+0 D4+E5 F6+0 G7+H8 I9+00 0+0 0+0
Rearrange and then do a "horizontal add"
              A1+B2+G7+H8 C3+I9 D4+E5 F6+0
Another Horizontal Add
              0 0  A1+B2+G7+H8+C3+I9  D4+E5+F6
Another Horizontal Add
              0 0 0 A1+B2+G7+H8+C3+I9+D4+E5+F6
Convert to 16-bit result, saturate, and be done

The 18 ops (9mul/9add) turned into 4 ops
```

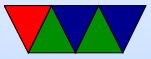


Problems

- Math is very fast, handfull of instructions
- Problem is getting memory from 3 pointers with 3-byte offsets into registers
- This is a “scatter/gather” problem found often with SIMD (and GPU)
- There are instructions to try to gather the values together, but not really suited for this
- Once you do it manually performance is actually worse than regular code



- Challenge: if picture not multiple of 16-bytes



Improved SIMD – Can we do better?

With many problems: re-think outside the serial box

```
Load full 16 bytes of pixel info from the three pointers,
multiply by the 9 values in sobel filter, shifting right by 3
   A * RGB RGB RGB RGB RGB RGB R
   B *     RGB RGB RGB RGB RGB R
   C *         RGB RGB RGB RGB R
   D * RGB RGB RGB RGB RGB RGB R
   E *     RGB RGB RGB RGB RGB R
   F *         RGB RGB RGB RGB R
   G * RGB RGB RGB RGB RGB RGB R
   H *     RGB RGB RGB RGB RGB R
+  I *         RGB RGB RGB RGB R
==================================
               RGB RGB RGB RGB R 13 values of result
Use compare instruction to saturate in parallel
Store out the 13 bytes at once

So (18*13) operations reduced to (~20) I think. Still haven't tried this yet
```