Image resizing with bilinear interpolation

SENG 440 Summer 2020

Ryan Woodward, V00857268, rnw@uvic.ca
Grant Hames-Morgan, V00857826, hamesg@uvic.ca

Image resizing is a very common operation for modern devices which must be able to quickly and efficiently scale images while maintaining quality. There are many scaling algorithms for images that determine which pixels are added/removed and their neighbouring pixels affect its colour value. Our project focuses on bilinear interpolation, which iterates over all pixels in an image and, for each colour channel (i.e RGBA), uses 3 neighbouring pixels to determine the new value using a series of floating point arithmetic operations. The algorithm is entirely nested loops with nested function calls to perform repeated floating point calculations, which makes it a great candidate for optimization towards a specific processor.

We began by simplifying the problem and deciding to process only grayscale bitmap-like (BMP) images. BMP is actually quite a complex format, with many header files, colour tables, support for compression, and multiple colour channels. To simplify, helper functions were written to convert colour BMP files to a `*.simple` grayscale-only binary format that could be more easily processed by the image resizing ARM code. This new format has the form `<w uint32_t><h uint32_t><pixels uint8_t>`

Due to the repeated nature of the algorithm, we recognized 2 different types of cache that could provide an optimization.

1. Floating point math done per-pixel could be cached in a table using ARM Neon Intrinsic `vtbl4_u8()`[1]. Consider the pixel and it's 3 neighbours as A, B, C:

   ```
   pixel_with_neighbours = pixel << 24 | A << 16 | B << 8 | C
   vtbl4_u8(uint32_t pixel_w_neighbours, uint8_t result_pixel)
   ```

   This still needs research, but could avoid floating point operations.

2. Reducing CPU Load/Store cache misses through better spatial locality. Pixels that are neighbours are not necessarily "together" in memory. By identifying areas of code that load/store pixels we may be able to improve spatial locality.

   Pre-generating a cache could potentially reduce the time spent going into memory. Our bilinear interpolation is currently being analyzed in order to bring pre-calculated values into a table. This table could then be brought into the cache from memory, and potentially decrease the amount of cache misses. The idea is that the table will attempt to capitalize CPU locality references by having its memory location held.

Our program has been written and tested. It is attached as a ZIP to this email. We have tested it on x86 and ARM (via UVic intranet). Snippets of the bilinear interpolation code is below.

```
void scale(SIMPLE_Image* src, SIMPLE_Image* dst, float scalex, float scaley) {
  int newWidth = (int)src->width * scalex;
  int newHeight = (int)src->height * scaley;
  int x, y;
  for (y = 0; y < dst->height; y++) {
    for (x = 0; x < dst->width; x++) {
      float gx = x / (float)(newWidth) * (src->width - 1);
      float gy = y / (float)(newHeight) * (src->height - 1);
      int gxi = (int)gx;
      int gyi = (int)gy;
      uint8_t c00 = getpixel(src, gxi, gyi);
      uint8_t c10 = getpixel(src, gxi + 1, gyi);
      uint8_t c01 = getpixel(src, gxi, gyi + 1);
      uint8_t c11 = getpixel(src, gxi + 1, gyi + 1);
      uint8_t result = (uint8_t)blerp(c00, c10, c01, c11, gx - gxi, gy - gyi);
      putpixel(dst, x, y, result);
    }
  }
}

float blerp(float c00, float c10, float c01, float c11, float tx, float ty) {
  return lerp(lerp(c00, c10, tx), lerp(c01, c11, tx), ty);
}


float lerp(float s, float e, float t) {
  return s + (e - s) * t;
}
```

The bulk of the work is being performed in a double for loop as seen in the above figure. As such we intend to focus on the software pipelining optimization technique by changing the loop to leverage instruction level parallelism. We intend to identify and test whether the rearrangement of code within these loops will give us a performance boost. One concern is the complication of the code considering each pixel needs bordering pixels in order to interpolate.

Also notice several function calls could be inlined as an optimization. There is potential for `lerp` to be hand-inlined using macros with `#define`. Similarly, `getpixel` and `putpixel` are simple lookups into the pixel array, and can be hand-inlined using macros. Whether or not these changes will provide a performance boost has yet to be determined.

Research into prefetch logic of the CPU is also being investigated. CPU cores often contain built in prefetch logic invocable with software instructions. The idea is to place software prefetch instructions in the interrupt handler. This could potentially address the issue of

pre-emptive task switching hampering efficient interrupt processing, which leads to cache misses. The prefetch would attempt to stay several steps ahead of the CPU by prefetching sequential cache lines that are based upon the current line. This technique differs from cache hit prefetch techniques whereby prefetches are triggered by cache hit accesses. Our prefetching technique will occur on cache misses. The cache miss is handled as expected, however the next cache line is immediately pre fetched. Using this method it is possible to avoid cache misses if the data is prefetched in time and placed directly into the cache. This could be incredibly difficult as we're unaware of how prefetch logic in interrupt processing works with our code.

References:

1. Neon Intrinsic Vector Table Lookup: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics?search=vtbl4_u8
2. https://developer.arm.com/docs/den0024/a/caches/cache-policies
3. https://researcher.watson.ibm.com/researcher/files/jp-INOUEHRS/IEEEBigdata2017_RabbitCT.pdf
4. https://scholarworks.rit.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=3686&context=theses:A
5. https://developer.arm.com/docs/ddi0344/h/level-1-memory-system/cache-organization/cache-miss-handling
6. https://community.arm.com/developer/tools-software/tools/f/armds-forum/1127/bilinear-interpolation-optimiziation-using-arm