Clown-free Gradienting

Philipp Czerner

"The Generic Image Library (GIL) is a C++ library that abstracts image representations from algorithms and allows writing code that can work on a variety of images with ***performance similar to hand-writing*** *for a specific image type."*

— Boost GIL documentation, 2007 (emphasis mine)

"Huh?"

— Me, last week

Overview

optimisation	time	speedup	difficulty	generality
baseline	1			
-O3, remove boost	0.004518	220.324	😎	😃
manual loop unrolling	0.005079	0.889	😐	🙂
manual vectorisation	0.002920	1.547	😯	☺️
parallelise	0.001264	2.311	😂	😊
intelligent initialisation	0.000986	1.281	😵	😭
OpenMP	0.000959	1.029	😴	😇
total	0.000959	1042.840		

Source Code

- The source code will be released together with this presentation
- Not just the code, but comments and documentation too!
  - More comments than code, actually...
- Check them out if you want to get an in-depth look, together with lots of explanations

Baseline

```
typedef typename channel_type<DstView>::type dst_channel_t;
for (int y = 0; y < src.height(); ++y) {
  typename SrcView::x_iterator src_it = src.row_begin(y);
  typename DstView::x_iterator dst_it = dst.row_begin(y);
  for (int x = 1; x < src.width() - 1; ++x) {
    static_transform(src_it[x - 1], src_it[x + 1], dst_it[x],
        halfdiff_cast_channels<dst_channel_t>());
  }
}
```

- Not the whole code
- Various pieces in other places

Enable compiler optimisations, get rid of boost ($\times$220.324)

```
for (int y = 0; y < height; ++y) {
  for (int x = 0; x+1 < width; ++x) {
    uint8_t* p = source + 3 * (width*y + x);
    int8_t*  q = target +      width*y + x ;
    int u_l = (p[-3]*4915 + p[-2]*9667 + p[-1]*1802 + 8192) >> 14;
    int u_r = (p[ 3]*4915 + p[ 4]*9667 + p[ 5]*1802 + 8192) >> 14;
    *q = (int8_t)((u_r - u_l) / 2);
  }
}
```

- Replace abstractions with straight C code
- Much easier to read, reason about and optimise
- Compiler can break through abstractions, if you compile with `-O3`
  - Not possible for submission

Enable compiler optimisations, get rid of boost ($\times$220.324)

- Move my code into separate compilation unit
  - Compile times improve by a factor of $\sim$340, from 3m26s to 0.6s!

```
void compute_gradient_fast(
    uint8_t* __restrict__ source,
    int8_t*  __restrict__ target,
    int width,
    int height
) { ... }
```

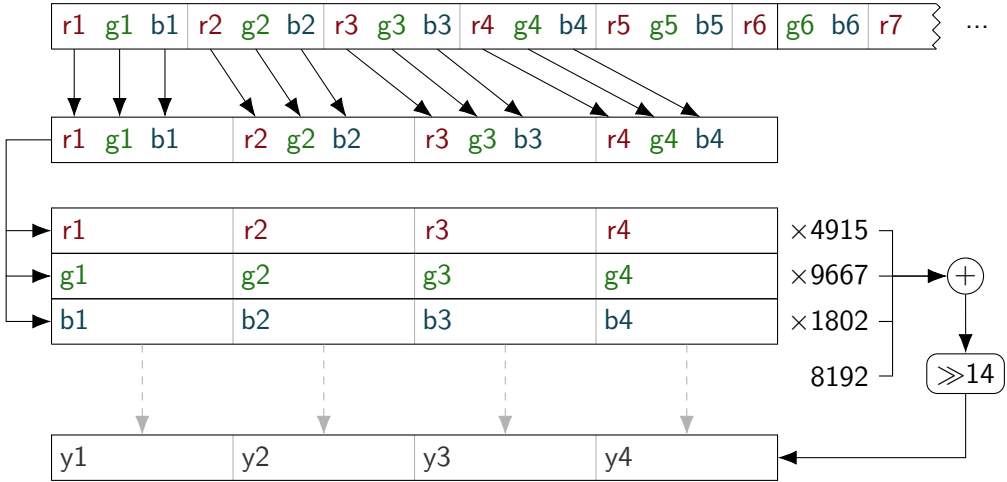
- Important: Tell the compiler that the pointers do not alias
- Enables automatic vectorisation

Manual loop unrolling ($\times 0.889$)

```
for (x = 1; x+5 < width; x += 4) {
    uint8_t* p = source + 3 * (width*y + x);
    int8_t*  q = target +      width*y + x;
    int c2 = (p[ 3]*4915 + p[ 4]*9667 + p[ 5]*1802 + 8192) >> 14;
    int c3 = (p[ 6]*4915 + p[ 7]*9667 + p[ 8]*1802 + 8192) >> 14;
    int c4 = (p[ 9]*4915 + p[10]*9667 + p[11]*1802 + 8192) >> 14;
    int c5 = (p[12]*4915 + p[13]*9667 + p[14]*1802 + 8192) >> 14;
    q[0] = (c2 - c0) / 2;
    q[1] = (c3 - c1) / 2;
    q[2] = (c4 - c2) / 2;
    q[3] = (c5 - c3) / 2;
    c0 = c4;
    c1 = c5;
}
```

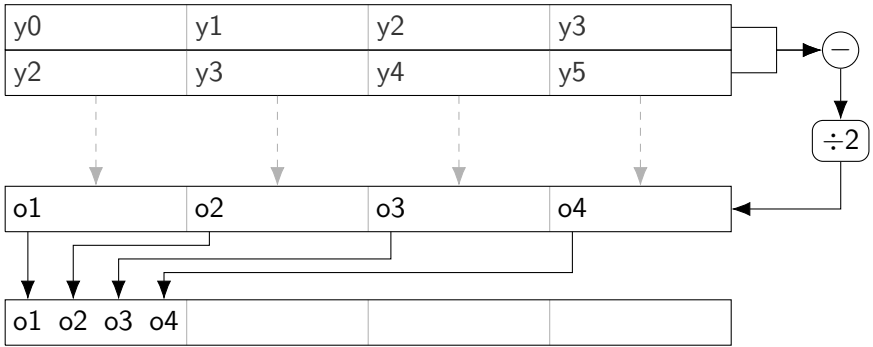
- Straightforward transformation
- Makes things worse
- Compiler already vectorised the previous code, unrolling does not help

Manual vectorisation ($\times 1.547$, compared to previous best)



Manual vectorisation ($\times 1.547$, compared to previous best)

Combine values of multiple iterations:



Then do it three times more, for o5 to o16. In total, 3×16 bytes read and 16 bytes written.

Parallelise ($\times 2.311$), intelligent initialisation ($\times 1.281$)

- First simple parallelisation
  - On my 4-core machine, best speedup with 3 threads
- For intelligent initialisation, model costs of creating threads and compute optimal strategy
  - Automatically chooses best number of threads
  - Distributed thread creation
  - Assign work accordingly
  - Uses machine dependent parameters
  - Not generally useful!

OpenMP ($\times 1.029$)

```
#pragma omp parallel for num_threads(num_threads_)
for (u64 y = 0; y < height; ++y) {
    compute_gradient_fast(source, target, width, y);
}
```

- Very simple to use!
- Even faster than my crazy initialisation, have to investigate...
- Have to manually select best number of threads

Questions?

For later, `noclowns@nicze.de` or just talk to me.