Fusing GPU kernels within a novel single-source C++ API

PEGPUM 2014

Ralph Potter[1,2], Paul Keir[1], Jan Lucas[3], Maurico Alvarez-Mesa[3], Ben Juurlink[3], Andrew Richards[1]

[1] Codeplay Software Ltd., Edinburgh
[2] University of Bath
[3] Technische Universität Berlin

Visit us at
www.codeplay.com

2nd Floor
45 York Place
Edinburgh
EH1 3HP
United Kingdom

Overview

- Why is fusion relevant for low-power computing?
- The Offload3 C++ parallel programming model
- Fusion of image processing primitives within Offload3

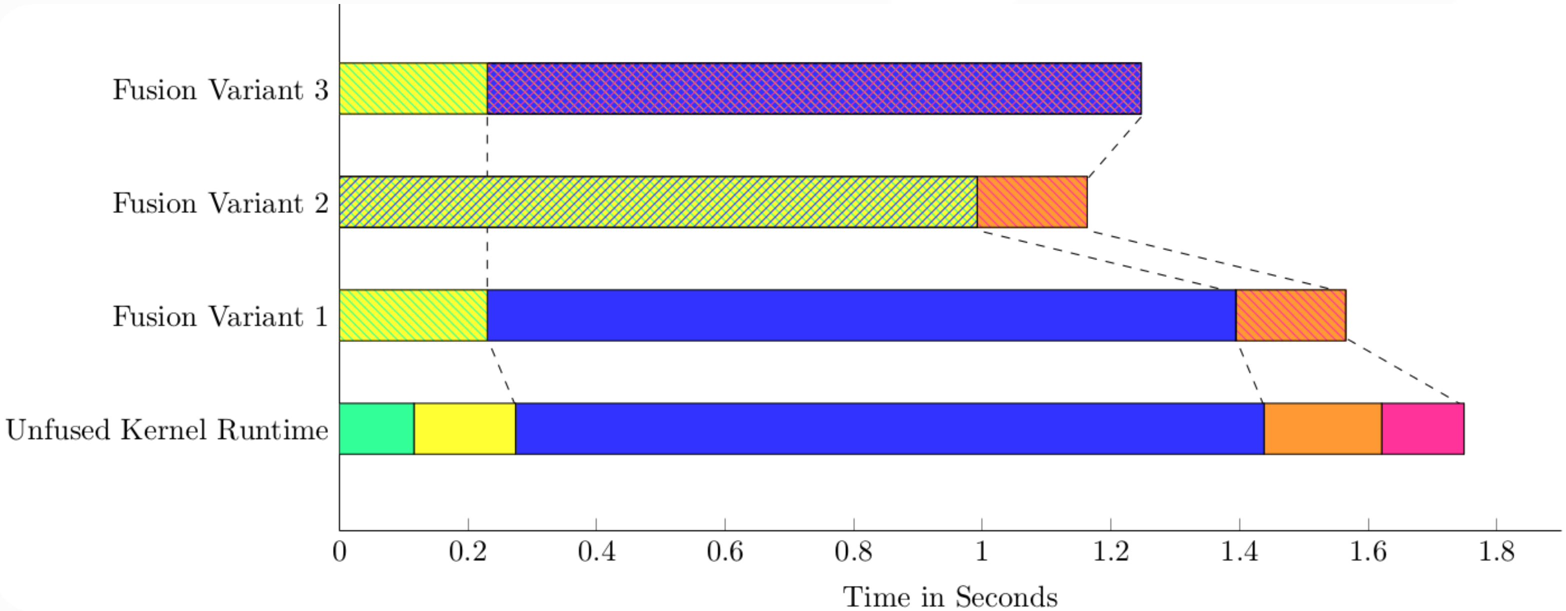
Motivation

- Some developers find the existing runtime APIs for GPU compute challenging
- Exploration of higher level models on Offload3
- Can we reduce unnecessary memory accesses through adopting different ways of expressing our computation?

Why Fusion Matters?

- Reusable and recombinable kernels often produce unnecessary loads and stores
- Data transfers are a major source of power consumption
- Loads and stores can be combined into a single fused kernel
- Hand-fused kernels possible, but reusability is poor

Preparatory Benchmarking of Hand-Fused Kernels



Principles of the Offload3 Programming Model

- C++11 compiler and framework for parallel computation targeting OpenCL-SPIR
- Call graph duplication avoids the need for additional decoration, simplifies porting and enables code reuse
- Generic algorithms through template meta-programming
- Provide a foundation for higher-level programming models

Offload3 Example

```
#include <Offload.h>

class MyObject {
  void myFunction() {}; // myFunction() declaration on the host CPU.
};

int main(int argc, char** argv) {
  ol3::queue q;
  MyObject obj; // MyObject instance declared on the host CPU.

  // Command groups wrap one or more kernel invocations.
  ol3::command_group(q, [&]() {
    // parallel_for executes the lambda parameter on the GPU.
    ol3::parallel_for(ol3::range(4, 4, 4),
                      ol3::kernel_lambda<class MyTask>([=](ol3::item_id id)
    {
      // Kernel code goes here...
      obj.myFunction(); // obj.myFunction() running on the GPU.
    }));
  });
}
```

Image Processing Primitives

- We borrow extensively from functional programming
- Image processing primitives implemented as function objects

```
struct blend {
  rgb operator()(const rgb& a, const rgb& b, float alpha) {
    return a * alpha + b * (1 - alpha);
  }
};
```

- We provide a C++ interface for fusing multiple primitives into kernels

Pipeline Composition

```
float saturation = 0.5f; // Saturation declared on the host

// Here multiple primitives (rgb_to_hsv, desaturate, hsv_to_rgb) are
// fused into a single function object.
placeholder<0> _1;
auto pipeline = bind<hsv_to_rgb>(
                  bind<desaturate>(
                    bind<rgb_to_hsv>(_1), // Unbound input parameter
                    saturation // Saturation captured
                  )
                );
```

- Bind mimics std::bind, allows partial application of functions
- Captures parameters, but allows for placeholders
- Builds an expression tree

Pipeline Execution

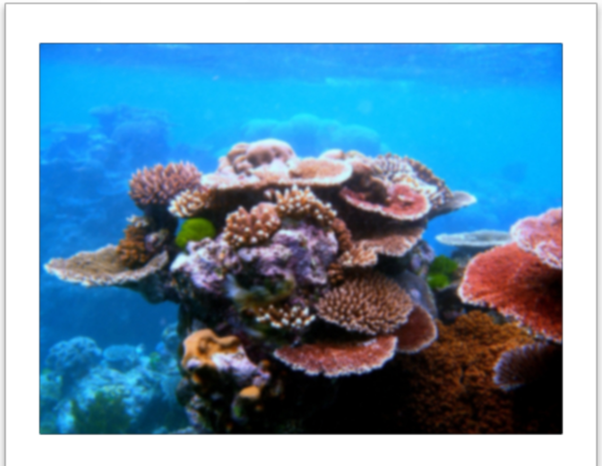
```
auto pipeline = ...

// Execute the pipeline in parallel for each pixel in the image.
ol3::parallel_for(ol3::range(2048, 2048)),
                  ol3::kernel_lambda<class MyPipe>([=](ol3::item_id id)

  // Read from an image. A unique value per thread.
  rgb colour = ...

  pipeline(colour); // Compiler detected that we use the objects
                    // function operator, so it duplicated it to the GPU
                    // _1 in the pipeline is replaced by colour variable
}));
```

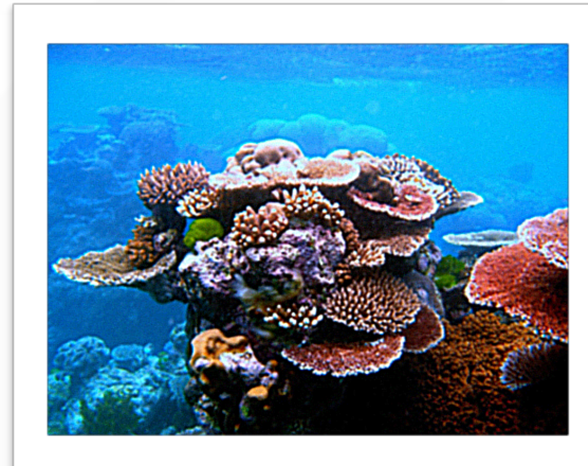
- Copies the expression tree to the GPU and lazily evaluates it

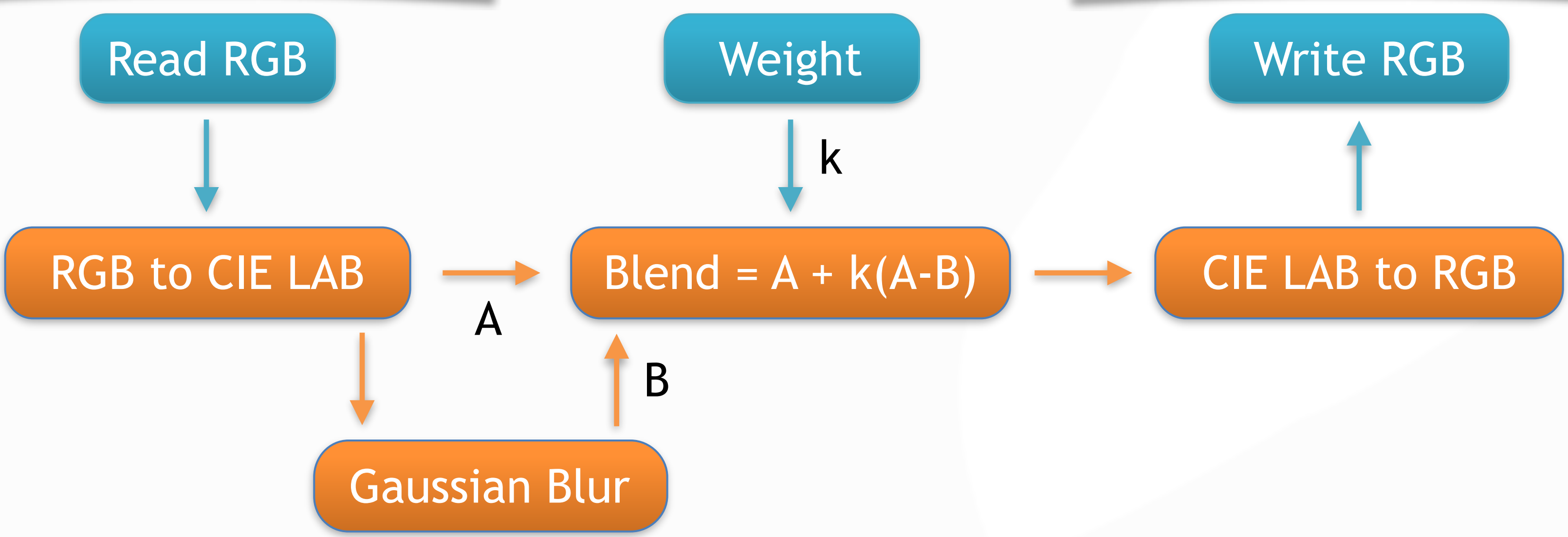
Convolution

- Stencils (and tiling data into shared/local memory)
- Convolution primitives (blurs, edge detection)
- Halo size inference from the expression tree
- Guiding workgroup grid sizes from the above

Unsharp Mask

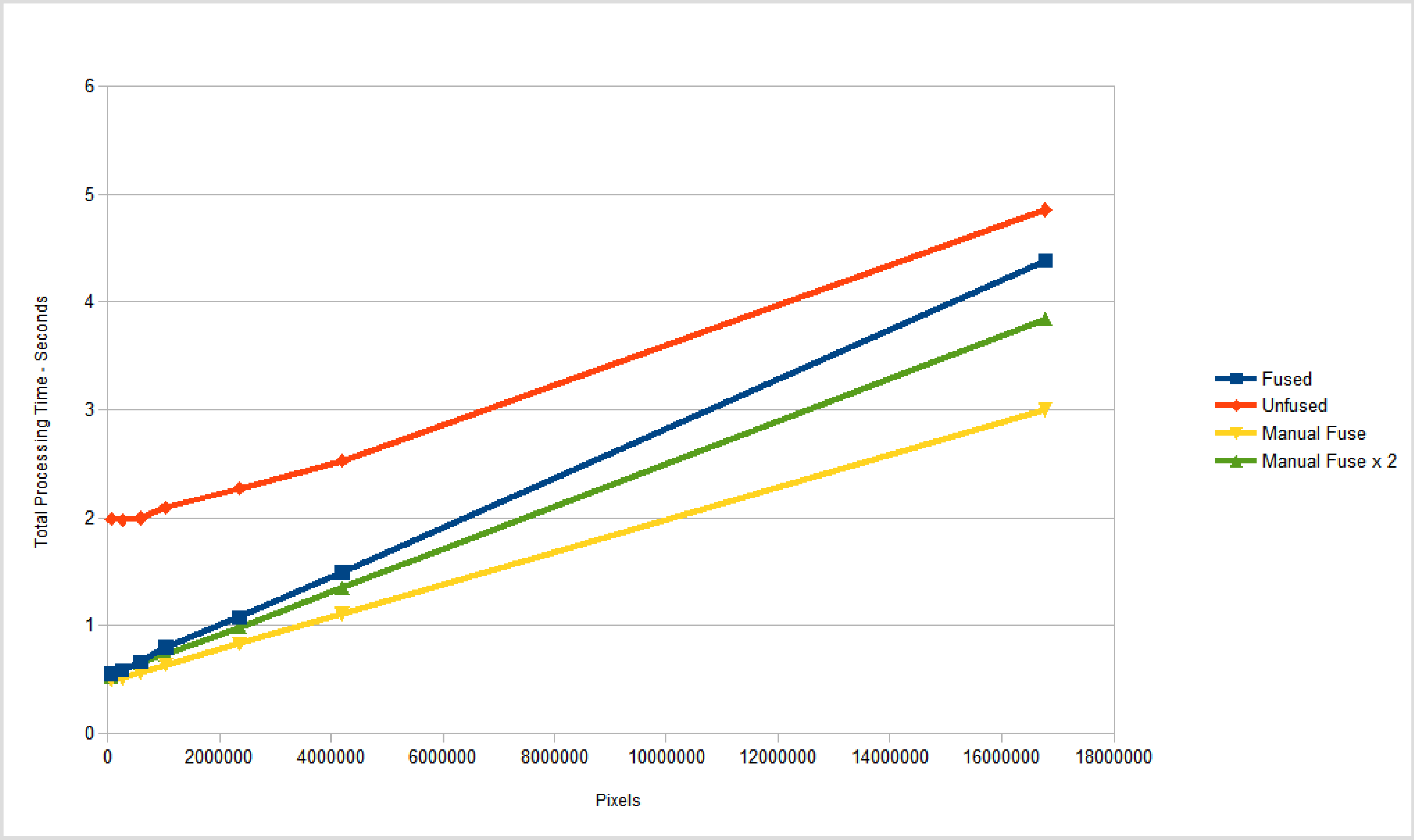






Results

- We timed our kernel on various sizes of square image (although the results are consistent for non-square)
- Benchmarked:
  - Primitives as individual kernels
  - Kernel fused by hand
  - Kernel fused through the bind expression
  - Kernel fused by hand, with additional common expression work
- Caveat: no optimisation, pre-release OpenCL-SPIR tool and driver chain



6
5
4
3
2
1
0
Total Processing Time - Seconds
0
2000000
4000000
6000000
8000000
10000000
12000000
14000000
16000000
18000000
Pixels
Fused
Unfused
Manual Fuse
Manual Fuse x 2

Future Work

- Power Measurement
- Fusion API Improvements
    - Generate Boost proto expression from custom bind expression
    - Fuse, unfuse, query, cut, join expression trees
    - Enable and disable filter subcomponents
    - Automatically fuse or unfuse an expression
    - Unsharp mask: purity of bind expressions can lose sharing, resolve common subexpressions
- New forms of primitives (morphological such as erosion, histogram operations)

Questions?

ralph@codeplay.com

paul@codeplay.com

Visit us at
www.codeplay.com

2^{nd} Floor
45 York Place
Edinburgh
EH1 3HP
United Kingdom