Automated OpenCL GPU kernel fusion for Stan Math

Tadej Ciglarič (presenter)*, Rok Češnovar, Erik Štrumbelj

* University of Ljubljana Faculty of Computer and Information Science
Stan

• State-of-the-art software for Bayesian statistics.

• Probabilistic programming language + Math library with auto-differentiation + Inference algorithms.

• Some operations have an OpenCL implementation.
Overview
GPU development in the Stan Math library

- Hundreds of possible operations and distributions to implement for GPUs.
- Sequence of basic kernels: simple to develop, poor performance.
- Specialized kernels: good performance, slow development.
Kernel fusion

• Execution of multiple operations in a single kernel.

• Speedup: kernel launch overhead, memory transfers between registers and global memory.

• Can be automated.

• Data fusion.

• Parallel fusion.
Lazy evaluation:
- Operations are C++ objects,
- expression is evaluated when assigned to result matrix.

Curiously Recurring Template Pattern:

```cpp
template <typename T_a, typename T_b>
class addition_ : public binary_operation<addition_<T_a, T_b>, T_a, T_b> {
  public:
    addition_(T_a&& a, T_b&& b)
      : binary_operation<addition_<T_a, T_b>, T_a, T_b>(
        std::forward<T_a>(a), std::forward<T_b>(b), "+") {}

};
```

```cpp
template <typename T_a, typename T_b, typename = require_all_valid_expressions_t<T_a, T_b>>
inline addition_<as_operation_cl_t<T_a>, as_operation_cl_t<T_b>> operator+(T_a&& a, T_b&& b) {
  return {as_operation_cl(std::forward<T_a>(a)),
           as_operation_cl(std::forward<T_b>(b))};
}
```
Implementation: operation types

Example:

```cpp
matrix_cl<double> a, b;
double c;
matrix_cl<double> d = c * (a + b);
```

**a + b**

`addition_<load_<matrix_cl<double>&>, load_<matrix_cl<double>>&>`

**c * (a + b)**

`elewise_multiplication_<scalar_<double>, addition_<load_<matrix_cl<double>&>, load_<matrix_cl<double>&>>>`

Assignment of an expression to a matrix generates, compiles and executes a kernel.
Implementation: generating kernel code

Operation objects generate code for their operation:

_addition:

double var4 = var2 + var3;

_load:

var5_global[i + var5_rows * j] = var4;
kernel void calculate(__global double var1,
   __global double* var2_global, int var2_rows, int var2_view,
   __global double* var3_global, int var3_rows, int var3_view
   __global double* var6_global, int var6_rows, int var6_view){
    int i = get_global_id(0);
    int j = get_global_id(1);
    double var2 = 0;
    if (!((!contains_nonzero(var2_view, LOWER) && j < i) ||
            (!contains_nonzero(var2_view, UPPER) && j > i))) {
        var2 = var2_global[i + var2_rows * j];
    }
    double var3 = 0;
    if (!((!contains_nonzero(var3_view, LOWER) && j < i) ||
            (!contains_nonzero(var3_view, UPPER) && j > i))) {
        var3 = var3_global[i + var1_rows * j];
    }
    double var4 = var2 + var3;
    double var5 = var1 * var4;
    var6_global[i + var6_rows * j] = var5;
}
Adding a new operation

• New class for the operation (derived from `operation_cl` or `operation_cl_lhs`).

• Must define:
  • Scalar,
  • generate,
  • view.

• Optional: `generate_lhs`, `rows`, `cols`.

• A function that constructs the object.
Empirical validation

• Comparison with a sequence of basic kernels.
• Comparison with a hand crafted kernel.
• Comparison with VexCL, a similar library.
• On NVIDIA GeForce GTX 1070 and AMD Radeon VII.
Comparison with a sequence of basic kernels

- Single operation kernel is comparable.
- Sequence is much faster.
- Matrix multiplication is slow, so speedups are negligible.
- We also avoid memory reallocations, which are slow on NVIDIA GPU.
Comparison with a hand crafted kernel

- On Bayesian linear regression.
- Comparable performance.
- Much simpler to use.
Comparison with VexCL

• Transposition and colwise sum are much faster.
• Rowwise sum is slightly slower.
• Other operations and multi-operation kernels are comparable.
• Also supports general tensors and multiple OpenCL devices.
Conclusion

- Performance is comparable to hand crafted kernels.
- As simple to use as calling premade kernels.
- Our work is similar to VexCL and Tensorflow XLA.