Lift Tutorial: Applications

Writing an Application

General Steps

- Determine input parameters
- Initialise input data
  - If testing, initialise comparison data
- Craft or translate the algorithm of interest
- Determine what memory to use running the resulting kernel
- Create an OpenCL kernel from your algorithm

Data Input to Lift Algorithms

- Lift takes in arrays as input parameters
  - Single values can be passed in in array form or used as global values

```
val liftLambda = fun(
    ArrayType(Float, SizeVar("N")),
    ArrayType(Float, weights.length),
    ...
)
```

- Single entry point for arrays into functions
  - Multiple arrays can be zipped together (but must be the same size!)

```
fun(neighbourhood =>
{
  ...

  $ Zip(weights, neighbourhood)
}
```

Initialising Data in Scala

- Create arrays of data to pass into Lift algorithms in Scala

```
val stencilValues = Array.tabulate(nx,ny,nz) { (i,j,k) => (i + j + k + 1).toFloat }
```

- Our examples are all in unit tests, which include data to compare against - often from the same algorithm in Scala

```
assertEquals(dotProductScala(leftInputData, rightInputData), output.sum, 0.0)
```

Developing an Algorithm

The goal is not for Lift to be programmed in directly.

However, functionality for new types of algorithms must be added in and tested. In doing so, there are a few things to keep in mind:

- Lift allows multiple inputs, but there is only one data entry point to the main algorithm (can contain tuples)
- The algorithm itself must eventually map values back to global memory
- The result will be returned in a single array (however, this array can also contain tuples)

Simple Example: 1D Jacobi Stencil

```
val jacobi1Dstencil =
 fun(
    ArrayType(Float, N),
    (input) =>
      Map(Reduce(add, 0.0f)) o
        Slide(3,1) o
          Pad(1,1,clamp) $ input
    )
```

Memory Options

- There are a range of different OpenCL memories that can be targeted in an algorithm
  - MapGlb - use global memory
  - MapWrg - use workgroup memory
  - MapLcl - use local memory
- Can parallelise up to 3 dimensions in OpenCL

```
MapGlb(2)(MapGlb(1)(MapGlb(0)(
          fun(neighbours => {
```

Creating an OpenCL kernel

- To compile your Lift kernel to OpenCL, run [opencl.executor]Compile(<*kernel*>)
  - This kernel can then be saved as a string or file

```
Compile(lambda)
```

- To execute the kernel straight away (compiling will happen behind the scenes), run [opencl.executor]Execute(<*options*>)[Array[*type*](*lambda*, *..inputs..*)

```
val (output, runTime) = Execute(inputData.length)[Array[Float]](stencilLambda, inputData, stencilWeights)
```

Detailed Example: Matrix-Matrix Multiplication

Matrix-Matrix Multiplication Overview

- Widely used algorithm in mathematics, physics, engineering, etc
- Range of possible optimisations can be used, the performance of which varies across architectures
- Lift makes it easy to test these optimisations out without having to rewrite the original code

Matrix-Matrix Multiplication Example Code

```
val f = fun(
     ArrayTypeWSWC(ArrayTypeWSWC(Float, K), M),
     ArrayTypeWSWC(ArrayTypeWSWC(Float, K), N),
     (matA, matB) => {
       MapGlb(fun( Arow =>
         MapSeq(fun( Bcol =>
           toGlobal(MapSeq(id)) o ReduceSeq(add, 0.0f) o MapSeq(mult) $ Zip(Arow, Bcol)
         )) $ matB
       )) $ matA
     })
```

Matrix-Matrix Multiplication Using Rewrites

$$
\begin{aligned}
&\mathbf{A} * \mathbf{B} = \\
&\quad Map(\overrightarrow{rowA} \mapsto \\
&\quad\quad Map(\overrightarrow{colB} \mapsto \\
&\quad\quad\quad DotProduct(\overrightarrow{rowA}, \overrightarrow{colB}) \\
&\quad\quad) \circ Transpose()\ \$\ \mathbf{B} \\
&\quad)\ \$\ \mathbf{A}
\end{aligned}
$$

80 rewrites

$$
\begin{aligned}
&(p239, p36 \mapsto \\
&\quad Join() \circ Map((p179 \mapsto \\
&\quad\quad Transpose() \circ Join() \circ Map((p70 \mapsto \\
&\quad\quad\quad Transpose() \circ Join() \circ Map((p20 \mapsto \\
&\quad\quad\quad\quad Transpose() \circ Map((p65 \mapsto \\
&\quad\quad\quad\quad\quad Transpose()(p65) \\
&\quad\quad\quad\quad)) \circ Transpose()(p20) \\
&\quad\quad\quad)) \circ Transpose() \circ Reduce((p75, p0 \mapsto \\
&\quad\quad\quad\quad Map((p164 \mapsto \\
&\quad\quad\quad\quad\quad Join() \circ Map((p81 \mapsto \\
&\quad\quad\quad\quad\quad\quad Reduce((p136, p90 \mapsto \\
&\quad\quad\quad\quad\quad\quad\quad Map((p163 \mapsto \\
&\quad\quad\quad\quad\quad\quad\quad\quad Get(0)(p163) + Get(1)(p163) * Get(1)(p90) \\
&\quad\quad\quad\quad\quad\quad\quad)) \circ Zip(2)(p136, Get(0)(p90)) \\
&\quad\quad\quad\quad\quad\quad))(Get(0)(p81), Zip(2)(Transpose() \circ Get(1)(p164), Get(1)(p81))) \\
&\quad\quad\quad\quad\quad)) \circ Zip(2)(Get(0)(p164), Get(1)(p0)) \\
&\quad\quad\quad\quad)) \circ Zip(2)(p75, Split(blockFactor) \circ Transpose() \circ Get(0)(p0)) \\
&\quad\quad\quad))(Zip(2)(Split(sizeK) \circ Transpose()(p179), p70)) \\
&\quad\quad)) \circ Transpose() \circ Map((p4 \mapsto \\
&\quad\quad\quad Split(sizeN) \circ Transpose()(p4) \\
&\quad\quad)) \circ Split(sizeK)(p36) \\
&\quad)) \circ Split(sizeM)(p239)
\end{aligned}
$$

Detailed Example: Convolutional Neural Network

Convolutional Neural Network Overview

Convolutional Neural Network Example Code

```
Compile(lambda)
```

Detailed Example: Acoustics Simulation

Acoustics Simulation Overview

- Partial Differential Equations can be discritised into stencils to model physical simulations like 3D wave models
- A simple acoustics simulation shows how this can be done in Lift

Acoustics Simulation Example Code

```
val acousticLambda = fun(
      ArrayTypeWSWC(ArrayTypeWSWC(ArrayTypeWSWC(ArrayTypeWSWC(Float,1),m), n), o),
      ArrayTypeWSWC(ArrayTypeWSWC(ArrayTypeWSWC(Float, m ), n ), o ),
      ArrayTypeWSWC(ArrayTypeWSWC(ArrayTypeWSWC(Float, StencilUtilities.weights3D(0)(0).length), StencilUtilities.weights3D(0).length),
StencilUtilities.weights3D.length),
      ArrayTypeWSWC(ArrayTypeWSWC(ArrayTypeWSWC(Float, StencilUtilities.weightsMiddle3D(0)(0).length), StencilUtilities.weightsMiddle3D(0).length),
StencilUtilities.weightsMiddle3D.length),
      (roomt, roomtminus1, weights, weightsMiddle) => {
        MapGlb(2)(MapGlb(1)(MapGlb(0)((fun((m) => {

          val cf = toPrivate( fun(x => getCoeff(x,RoomConstants.cf(0), RoomConstants.cf(1))) ) $ Get(m,2)
          val cf2 = toPrivate( fun(x => getCoeff(x,RoomConstants.cf2(0), RoomConstants.cf2(1))) ) $ Get(m,2)
          val valueMask = toPrivate(MapSeq(BoundaryUtilities.idIF)) $ Get(m,2)

          toGlobal(MapSeq(multTuple)) $
                  Zip(MapSeq(addTuple) $
                          Zip(MapSeq(subtractTuple) $ Zip(
                                                          MapSeq(fun(x => mult(x,RoomConstants.l2))) o
                                                          ReduceSeq(add, 0.0f) o Join() o MapSeq(ReduceSeq(add, id $ 0.0f) o MapSeq(multTuple)) o
                                                            Map(\(tuple => Zip(tuple._0, tuple._1))) $ Zip(Join() $ Get(m, 1), Join() $ weights),
                                                          (MapSeq(multTuple)) $ Zip(
                                                        ReduceSeq(add, 0.0f) $ Get(m, 0), cf2)),
                            (MapSeq(multTuple)) $
                                   Zip(MapSeq(fun(x => (subtract(2.0f,x) ))) o MapSeq(fun(y => mult(y,RoomConstants.l2))) $ valueMask, ReduceSeq(add, 0.0f)
o Join() o MapSeq(ReduceSeq(add, id $ 0.0f) o MapSeq(multTuple)) o Map(\(tuple => Zip(tuple._0, tuple._1))) $ Zip(Join() $
                                             Get(m, 1), Join() $ weightsMiddle))), cf)
        }))))) $ Zip3D(roomt, (Slide3D(StencilUtilities.slidesize, StencilUtilities.slidestep) o PadConstant3D(1,1,1,0.0f) $ roomtminus1),
Array3DFromUserFunGenerator(getNumNeighbours, ArrayTypeWSWC(ArrayTypeWSWC(ArrayTypeWSWC(Int, m), n), o)))
      })
```