OSL: A Compiler and Runtime for Data Parallel Python

Yunsup Lee Bryan Catanzaro and Yong-Jin Kwon

CUDA Programming Model

- What we LIKE
  - The programming model encourages the programmer to write scalable code
  - High performance is possible
- What we DISLIKE
  - Hard to debug kernel code
  - Well, a lot of people use “printf” to debug...
  - Global synchronization is hard to program
  - Exploiting the GPU’s memory hierarchy efficiently is hard
  - C/C++ is not highly productive

OSL: Highly Productive Parallel Programming Language

- Python is highly productive
  - Don’t have to worry about types or declarations
  - Batteries included
- We want to bring Python’s productivity to GPU programming
  - Allow arbitrary Python code to be integrated with high performance subset of Python code
  - Compile this subset of Python code to GPU
  - Use Python as an emulator for debugging

OSL Parallel primitives

- stencil (array, filters, arguments)
  - Similar to “map”
  - array: explained on the next slide
  - filters: a list of
    - filter(array, element, [dim], [position], arguments)
    - it has access to the peers’ elements
    - arguments: a list of any type
- catreduce (labels, array, redop, arguments)
  - Similar to “reduce”
  - labels, array: explained on the next slide
  - Reduction operator
    - redop(element, element)
    - Commutative, associative
    - arguments: a list of any type

Map vs. Stencil

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter1</td>
<td>filter2</td>
<td>filter1</td>
</tr>
<tr>
<td>B = map(filter1, A)</td>
<td>C = map(filter2, B)</td>
<td></td>
</tr>
</tbody>
</table>

def filter1(element):
    def filter2(element):
        filters = [filter1, filter2]
        arguments = [...]
        C = stencil(A, filters, arguments)

        def filter1(array, element, [height, width], [y, x], arguments):
            redop = element
            arguments = [...]
            result = catreduce(labels, A, redop, arguments)

            def redop(element, element):
                results = [
                    redop(element, element, arguments):
                ]