PyPlover: A System for GPU-enabled Serverless Instances

Motivation
- Serverless GPUs greatly reduce the amount of set-up needed on the user side
- Scalable:
  - Serverless provider automatically load balances
- Resource requirements can be met more exactly
- Saves cost needed for set-up

Core Challenges
- How would a user interact with our system?
  - Serialization of input
  - How much flexibility should we give to the user?
- What structure would the internal execution pipeline of our system take?
- Would the tradeoff for the overhead of function-as-a-service be worth the ease of use and cost efficiency?

Execution Structure
- **Init**: Loads in specified kernel functions and returns a list of kernel function pointers
- **Init_func**: Creates the initial state object and mallocs the required memory in GPU
- **Run**: Takes in as input the return values from init and init_func
  - Runs the kernel functions implemented by the user

Results

<table>
<thead>
<tr>
<th></th>
<th>SRK/OpenLambda</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Kernel</td>
<td>0.0015s</td>
<td>0.079s</td>
</tr>
<tr>
<td>Load Constants</td>
<td>0.0620s</td>
<td></td>
</tr>
<tr>
<td>Run Kernel</td>
<td>0.0208s</td>
<td></td>
</tr>
<tr>
<td>SRK/OpenLambda Overhead</td>
<td>2.8307s</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2.915s</td>
<td></td>
</tr>
</tbody>
</table>

Future Work/Lessons
- We need more sophisticated options to test on for results
  - Simple vecAdd test is not sufficient
  - Is overhead cost negligible?
  - Much of CUDA code is generated
  - Working on results for a CNN written in CUDA
- Attach PyPlover to the backend of a developed ML framework
  - Attempt was made on the backend of TVM
  - Much more complex than we originally thought
  - Cannot just replace CUDA calls with PyPlover
- Use a more efficient serverless provider
  - i.e Droplet
  - The overhead for SRK/OpenLambda is not ideal