Homework 4: MNIST inference with CNN as a PLURAL program

1. Revisit HW3 and measure (memory) space (shared and private), making the following considerations:
   a. The space in shared memory reflects the global data structures that you define in your code.
   b. In HW3 we implied that everything resides in shared memory. Namely, data and parameters are pre-loaded in their entirety from disk files into shared memory (and we don’t count file loading time), all intermediate data structures are pre-declared, and all output is generated in shared memory rather than stored to disk files. Likewise, we do not assume reading data (nor parameters) from communication links, and we don’t count that time. In other words, we sacrifice memory for speed. The question is, how much space do we need?
   c. As implied above, two ways are typically employed to reduce space needs:
      i. First, data may be read in parallel with computation. Sometimes it may slow down processing, and it usually incurs more energy. For streaming (processing a stream of input data sets) it is a must—when one data set is processed (e.g., classified) the next one is being read in.
      ii. Second, space may be reused. For instance, once Layer $k$ has completed and its input is no longer needed, that memory area may be reused for storing the output of layer $k+1$. This space saving is typically achieved by either union data type (redundant definition of same memory area) or by dynamic memory allocation with malloc() and free().
   d. For simplicity, let’s assume that HW3 (and this HW4) employ only global static data types, without any memory reuse. All you need to report is the memory use table (size of each data structure in bytes and the total size in bytes). Recall that we use single precision floating point (4 bytes) exclusively.
   e. In addition, identify the total memory size accessed by each task. This can indicate the size of private memory or the data cache memory of each core. We ignore re-use of cache memory by swapping (capacity miss) etc. Report the table of memory used by each task and the maximum (at most one task executes per core, and each core must accommodate the largest task because we prefer homogeneous allocation of tasks to cores).
   f. Similar to the case of shared memory, in private memories and caches we can reduce the size of needed memory by swapping, but this is less efficient because simple cores (e.g., ARM, MIPS, RISC-V) typically do not facilitate cache pre-fetch and swap time (by software control as in IBM CELL or by cache miss) is incurred exclusive of computation. This argument motivates us to employ cache-aware programming, namely decompose the computation into chunks of data that fit into the cache.

2. Repeat the MNIST inference machine as a Convolutional Neural Network (CNN).
3. Use the following CNN structure

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Layer 4</th>
<th>Layer 5</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>Max Pooling</td>
<td>Convolution</td>
<td>Max Pooling</td>
<td>F-C</td>
<td>Softmax</td>
</tr>
<tr>
<td>Activation: ReLU</td>
<td>Inputs: 32</td>
<td>Activation: ReLU</td>
<td>Inputs: 64</td>
<td>Activation: ReLU</td>
<td>Neurons: 1024</td>
</tr>
<tr>
<td>Image size: 28</td>
<td>Input size: 28</td>
<td>Input size: 14</td>
<td>Input size: 7x7x64</td>
<td>Neurons: 10</td>
<td></td>
</tr>
<tr>
<td>Zero padding: 2</td>
<td>2x2 Max pooling</td>
<td>Zero padding: 2</td>
<td>2x2 Max pooling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter size 5</td>
<td>Stride 2</td>
<td>Filter size 5</td>
<td>Stride 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td># of filters: 32 (outputs)</td>
<td>32</td>
<td># of filters: 64 (outputs)</td>
<td>64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Short Explanation of the Convolutional Layers**

Each convolutional layer receives a feature map as input. For Layer 1, the input image is the only input feature map. Its output comprises 32 different feature maps, where each map is the result of a 5x5 convolution applied to the input image, with ReLU activation on the result.

The max pooling Layer 2 simply reduces the size of each of the 32 output feature maps of Layer 1 from 28x28 to 14x14 by selecting the maximum value in each 2x2 neighborhood. The neighborhoods are NOT overlapped. For instance, the first element (coordinates [0,0]) of a result feature map is max[0:1,0:1] of an input feature map, and the second element (coordinates [0,1]) is max[0:1,2:3].

Layer 3 is again a convolutional layer, similar to Layer 1. However, it receives 32 input feature maps (each 14x14 in size) and outputs 64 feature maps, one for each of its 64 filters. Every output feature map is the result of convolution kernels applied on each of the 32 input feature maps. The following image demonstrates 32 kernels, applied to the same region in the input feature maps, added to produce the value of a single pixel in one output feature map.

Overall, Layer 3 has a total of 32x64 kernels, each 5x5 in size (+1 for bias).

4. The parameter (weight) files and an example code for reading them are posted on the course web page.

5. Evaluate and report performance (T(p), SUP(p) and Eff(p)) and energy for 1,2,4,8,16,32,64,128,256,512,1024 cores, as done in HW3. Also report memory space, similar to item 1, revisiting HW3.

6. Compare performance, energy and space to the fully connected NN you have implemented and analyzed in HW3. Provide a short explanation of the results.
7. Submit by 30 January 2017, using email to ran@ee with subject line “048874-F2016-HW4”:
   a. Your code
   b. Performance, energy and space report and comparison to the FC-NN report of HW3.