Discussion on the Project of Making the MATLAB Implementation Competitive with Tensorflow

Last updated: May 10, 2021

Part One: Two accumarray Implementations I

- The first code considers a simple loop as follows

```
for(mwSize i = 0; i < m; i++)
vTPp[int(subsp[i]) - 1] += valp[i];
```

- However, an issue is that some threads may try to update the same address
- See our example before

$$(P_\phi^m)^T \boldsymbol{v}^i = [\ v_1 \quad v_2 + v_5 \quad v_6 \quad v_3 \quad v_4 + v_7 \quad v_8\ ]^T \quad (1)$$

Part One: Two accumarray Implementations II

- We need to specify that the update is an atomic operation:

```
for(mwSize i = 0; i < m; i++)
#pragma omp atomic
vTPp[int(subsp[i]) - 1] += valp[i];
```

- Notice that we do accumarray on multiple instances in one call

Part One: Two `accumarray` Implementations III

- Recall that in the earlier discussion we prepared indices in different ranges: for given indices

$$\begin{bmatrix} 1 & 2 & 4 & 5 & 2 & 3 & 5 & 6 \end{bmatrix}^T \tag{2}$$

We can apply *MATLAB*'s `accumarray` on the vector

$$\begin{bmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^l \end{bmatrix}, \tag{3}$$

Part One: Two accumarray Implementations IV

by giving the following indices as the input.

$$\begin{bmatrix} (2) \\ (2) + a^m_{\text{pad}} b^m_{\text{pad}} d^m \mathbb{1}_{h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}}} \\ (2) + 2a^m_{\text{pad}} b^m_{\text{pad}} d^m \mathbb{1}_{h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}}} \\ \vdots \\ (2) + (I-1) a^m_{\text{pad}} b^m_{\text{pad}} d^m \mathbb{1}_{h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}}} \end{bmatrix}, \quad (4)$$

where

$a^m_{\text{pad}} b^m_{\text{pad}} d^m$ is the size of $\text{pad}(Z^{m,i})$

Part One: Two `accumarray` Implementations V

and

$h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}}$ is the size of $\phi(\text{pad}(Z^{m,i}))$ and $\boldsymbol{v}_i$.

- Then we can do a two-level loop, where the first one is on instances
- Then we can parallelize the outer loop without needing atomic operations
- This is the second implementation

Part One: Two `accumarray` Implementations VI

- Comparisons done by our TAs sometime ago on a clean machine are as follows.
- Average of 10 runs on the full set of mnist

  1-level loop: 36.68 seconds

  2-level loop: 14.55 seconds
- Clearly the use of a 2-level loop is much better
- It's unclear why this happens, but atomic operations might be a reason.
- We add atomic in the 2-level loop, and the running time is increased to 36.75 seconds

Part Two: Outer Sum I

- The outer sum computes the P_ϕ matrix for all instances in the batch as a combined matrix
- However, P_ϕ is the same for all instances
- We can therefore calculate the indices for different instances by adding the appropriate offset
- The new for loop is as follows:

Part Two: Outer Sum II

```
for (mwSize i = 0; i < inst; i++)
{
    auto inst_arrayp = arrayp + i * m;
    for (mwSize j = 0; j < m; j++)
    {
        auto at = int(idxp[j]) - 1 + i * sz;
        vTPp[at] += inst_arrayp[j];
    }
}
```

- Note that P_{pool} is dependent on the instance, and cannot be computed by the same method

Part Three: Comparison with Tensorflow I

- Some reported that MATLAB is slower
- But some reported the opposite
- A possible reason is that some instructions may not be supported by older CPU architecture. On some workstation machines (linux5 to linux15), due to hardware limitations, Tensorflow was not installed to use optimized instructions such as `AVX2` and `FMA`.
- Our TAs did the following command

```
python3 -c 'import tensorflow as tf;tf.ones(1)'
```

Part Three: Comparison with Tensorflow II

- The following statement

  `This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA`

  was displayed on linux1 to linux4. While the last two instructions `AVX2` and `FMA` were not available on linux5 to linux15.

Part Three: Comparison with Tensorflow III

- For current department workstation machines, TensorFlow is more powerful on linux1 to linux4 compared with linux5 to linux15.
- You can run `lscpu` on the command line and check the "Flags" at the end where you can see which instructions are supported by the CPU.