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

Last updated: June 1, 2020

Matrix Expansion and `accumarray` I

- From project 3, we know their complexity is relatively smaller than matrix-matrix products
- However, they are among the bottlenecks
- From project 4, we provide a MATLAB-C interface for matrix expansion
- Some simply apply it and check the running time reduction
- But we did mention that you should try to reduce the time of other parts, in particular, `accumarray`

Matrix Expansion and accumarray II

- Therefore, those who apply only the matrix expansion code get lower points because others have paid more efforts on this project.

The `accumarray` Implementation I

- Most of you figured out that the code is extremely simple

```
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

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

The `accumarray` Implementation 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];
```

- Some are excellent to figure this out
- On the other hand, we do `accumarray` on multiple instances in one call

The `accumarray` Implementation 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

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

The `accumarray` Implementation IV

by giving the following indices as the input.

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

where

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

The `accumarray` Implementation 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
- Some are good to try such an approach
- Our TAs have conducted a comparison on a clean machine

The `accumarray` Implementation VI

- 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

Change of SimpleNN I

- You might notice that recently simpleNN MATLAB code was updated a few times
- The changes were for the second part of project 6
- Unfortunately the running time of SG part was affected
- Due to some unsuitable changes, SG code in some versions becomes slower
- This is fine as we don't evaluate you on how close your timing result is to Tensorflow's.

Change of SimpleNN II

- We check on what you really have done, in particular, the respective improvement of matrix expansion and `accumarray`
- If we think from the viewpoint of a regular course, the tool used for a HW shouldn't be constantly changed
- But ours is not a regular one. For a research oriented course, this is what it should be – we constantly research and improve the tool

Change of SimpleNN III

- I want to take this chance to say again that to take a course like ours, the mindset may need to be different
- By the way, for project 6, please git pull the latest code