Enable Operator Level Parallelism under Subgraph

- Credit to Zhennan for this proposal

Problem

Although data parallel is used in MXNet, its performance is not good enough for the less computational intensive operators in the inference stage, especially for the small batchsize. This phenomena widely exists in many popular models, such as googlenet, wide deep and inception v3. For example in the wide deep model, 26 embedding OPs are executed in sequence and each one only consumes very little computing resource. So, the model level performance is sub-optimal since the long execution path in the low-parallelism operators.

Goals/Usecases

The primary goal is to improve the performance by paralleling inefficient and independent OPs. In this proposal, only the situation that OPs comes to/from one OP is covered and other hierarchical patterns will be considered in the future. The change in this proposal will grantee that the modification is transparent to users and does not change existing scripts, models.

The approach can work for all backends by sharing the same subgraph path. But in practice some adjusts in the interfaces and implementations are still needed, like the difference in the hardware mapping where the CPU can assign the OP to different cores and GPU needs to multiple stream. Thus, in the first step, the CPU and MKLDNN backend are enabled.

Proposed Approach

Take the wide deep model for example, after split, data flow is divided to 26 and each of them will be handled by single embedding OP. In ordinary process, these 26 embedding OPs will be executed one by one when running inference, and data parallel will be used in its kernel function. Now we replace the 26 OPS using one parallel OP which can handle inference in OP level parallel.
As Fig. 2 shown, we implement the whole workflow based on subgraph API. `SgParallelOpSelector` inherited from `SubgraphSelector` is used to find the parallel structure, and `SgParallelOpProperty` inherited from `SubgraphProperty` is to connect its input/output entry.

The key block in Fig. 2 is `Filter` which is used check whether the finding parallel structure meet the metrics. It must guarantee that the operator is thread safe; otherwise, it may fails during simultaneous execution by multiple threads. From MKL-DNN 1.0 all MKLDNN operators will be thread safe and can be executed in parallel. But now, we need to maintain a whitelist for thread safe operators. There are some other conditions which used to fine tune the performance such as `paralleled Node number >= threshold` will cause performance drop. Environment variable may be add by user to add/remove whitelists in future release.

The main body of parallel op forward function is accelerate by OMP multithread as Figure3. When do the inference, several operators run parallel. Such as in the wide deep model, 26 embedding forward function are called simultaneously. By this parallel in OP level, performance is improved a lot.

```c
#pragma omp parallel for num_threads(omp_threads)
for(int i = 0; i < parallel_op_size; i++) {
    originOp[i].Forward<xpu>(attrs, ctx, each_input, req, each_output);
}
```

To get the best performance, we need to support nested OMP and fine tune the parameters. In current version, we just simplify it by disable nested OMP. Environment variable may be added to support fine tune the performance in future release.
This method is different from setting environment MXNET_CPU_WORKER_NTHREADS. Using our method, we just do parallelism for special OP, while MXNET_CPU_WORKER_NTHREADS is for all OPs.

**Addition of New APIs**

No new APIs were added or modified.

**Backward compatibility**

We add a pass for backend, which have no backward compatibility issue when deactivate. When inactive, we may consider the compatibility for different passes.

**Performance**

In wide and deep model, we replace 26 embedding Ops with one parallel_op as Fig. 1. When we do inference on SKX-8180 1 socket with batch size 1 and OMP thread 28, performance as Table 1 shows. Parallel OP has a 3.7X speedup.

<table>
<thead>
<tr>
<th>OP</th>
<th>Time cost (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>embedding</td>
<td>51240.051</td>
</tr>
<tr>
<td>SgParallel_op</td>
<td>13763.959</td>
</tr>
</tbody>
</table>

Table 1 performance for Embedding and SgParallel_OP

MKLDNN OPs will be supported since version 1.0 that will make Intel MKL-DNN primitives stateless and thread safe: the same primitive can be executed in multiple independent threads as long as different threads use different scratchpads. So we can accelerate more models such inception and googlenet.

**Test Plan**

Tests need to cover 2 parts. First one is the graph conversion test. We need to ensure that:

<table>
<thead>
<tr>
<th>Step</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All OPs are partitioned into one or more subgraphs according to executing mode.</td>
</tr>
<tr>
<td>2</td>
<td>Desired patterns can be captured and desired paralleled OPs will be created.</td>
</tr>
</tbody>
</table>

Another one is the unit test for OPs in parallel OP whitelist. All these OPs should be thread-safe. The test should cover all supported OPs and make sure they can provide the accurate result.

**Milestones**

1. Support structure as Fig.1.
2. Support structure as Fig.4. In this Fig, all OPs to be replaced has output to OP.

![Figure 4. Replace Ops come to one OP](image)

3. Support structure as Fig.5. In this Fig, all OPs to be replaced has input from OP X.
4. Support all MKL-DNN OP, all OPs which support parallel add to whitelist.

References

https://cwiki.apache.org/confluence/display/MXNET/Parallel+Inference+in+MXNet
https://cwiki.apache.org/confluence/display/MXNET/MXNet+Graph+Optimization+and+Quantization+based+on+subgraph+and+MKL-DNN
https://github.com/intel/mkl-dnn/tree/rfc-api-changes-v1.0/doc/rfc-api-v1.0