AutoMCL Documentation

Haoyu Chi

2021/6/2

v1.0

1 Introduction

```
We implements AutoMCL based on TVM(version 0.6.0) tools in python.
```

```
TVM: TVM is an open deep learning compiler stack for CPUs, GPUs, and specialized
accelerators. It aims to close the gap between the productivity-focused deep
learning frameworks, and the performance or efficiency-oriented hardware
backends.

AutoMCL: We introduce several optimization strategies, combining analytic ideal
cache models with machine learning models trained with real hardware measures,
and integrate them into a unified auto-tuning framework, called AutoMCL, to
improve the performance of DL compilers on both the operation level and the end-
to-end model inference.
```

2 Install

step0. Requirements

```
gcc >= 4.8
CMake >= 3.5
python >= 3.5
llvm >= 4.0.0
xgboost == 0.90
sklearn <= 0.23   (recommend:0.20.3)
pandas
numpy
```

TVM Installation and Source Code Replace(Recommend)

step1. TVM Installation(https://tvm.apache.org/docs/install/index.html)

You can also choose to clone the source repo from github. It is important to clone the submodules along, with --recursive option.

```
# Path:~/
git clone --recursive https://github.com/dmlc/tvm  (select version==0.6.0)
git clone --recursive https://github.com/CharlieCurry/incubator-tvm.git
```

step2. Build the Shared Library

```
sudo apt-get update
sudo apt-get install -y python-dev python-setuptools libtinfo-dev zlib1g-dev
build-essential
```

step3. Create build file, copy and modify config configuration file

```
cd tvm
mkdir build
cp cmake/config.cmake build
```

step4. Create build file, copy and modify config configuration file. For example, Change set(USE_CUDA OFF) to set(USE_CUDA ON) to enable CUDA backend.

```
set(USE_CUDA OFF)     --->set(USE_CUDA ON)
set(USE_LLVM OFF)     --->set(USE_LLVM ON)
```

step5. AutoMCL requires LLVM for for CPU codegen. We highly recommend you to build with the LLVM support on. We can then build tvm and related libraries.

```
cd build
cmake -DCMAKE_BUILD_TYPE=Debug ..
DCMAKE_BUILD_TYPE=Debug
make -j4
```

step6. Add environment variables

```
vim ~/.bashrc
export TVM_PATH=/~/tvm
export
PYTHONPATH=$TVM_PATH/python:$TVM_PATH/topi/python:$TVM_PATH/nnvm/python:${PYTHON
PATH}
source ~/.bashrc
```

step7. Replace Source Code

Method1 manual replacement

`AutoMCL_EasyReplace/AutoConfig     --> tvm/python/tvm/autotvm/tuner`

`AutoMCL_EasyReplace/InitConfigTask --> tvm/python/tvm/autotvm/task`

`AutoMCL_EasyReplace/OptSchedule    --> tvm/topi/python/topi/x86`

`AutoMCL_EasyReplace/AS_OS          --> tvm/topi/python/topi/x86`

Method2 setup script

```
#Setup.sh
echo Setup Start.
mv $TVM_PATH/python/tvm/autotvm/tuner $TVM_PATH/python/tvm/autotvm/tuner_tvm
cp -r AutoConfig $TVM_PATH/python/tvm/autotvm/tuner
mv $TVM_PATH/python/tvm/autotvm/task $TVM_PATH/python/tvm/autotvm/task_tvm
cp -r InitConfigTask $TVM_PATH/python/tvm/autotvm/task
mv $TVM_PATH/topi/python/topi/x86 $TVM_PATH/topi/python/topi/x86_tvm
cp -r AS_OS $TVM_PATH/topi/python/topi/x86
echo Setup Done.
```

3 Experiments

The maximum number of trials for the whole tuning and the early stopping are set respectively as 10, 000 and 400 for most of the experiments. The only exception is the end-to-end evaluation of CNNs, where we set the two numbers respectively as 500 and 300.

1) Task Tuning

The DL compiler TVM provides two default computes for the matrix multiplication operator, namely `DNMM` and `RPMM` and one default compute `CONV` for general 2D-convolution. We introduce another four alternative computes TMM, TTMM, DPMM, LPMM for matrix multiplication and two alternative computes `Im2colDNMM332` and `Im2colRPMMV` for convolution by converting convolution to matrix multiplication in an `im2col` manner. We manually write schedule template for each new compute and improve the default schedule templates for `DNMM`, `RPMM`, `CONV` respectively as `DNMM332` (single-level tiling to double-level tiling), `RPMMV` (adding missing vectorization for some loop) and `CONVOpt` (loop reordering).

All dense and conv2d operators and schedules can be found in: `AutoMCL_Repository/UserTest/op/dense_template`, `AutoMCL_Repository/UserTest/op/conv2d_template`, the relevant python files are easy to use. For example, using `python dnmm332.py 64 2048 1000` are respectively represent the matrix multiplication `DNMM332` operator's three dimension `M=64,K=2048,N=1000`. Matmul takes two matrices `A_{M×K}` and `B_{K×N}` as input and computes their product matrix `C_{M×N}`.

2) End2End Tuning

Evaluation of AutoMCL on the operation and the end-to-end level. Now we evaluate the performance of AutoMCL, which integrates all the optimization strategies introduced in Paper Section 3, on optimizing *matmul* and *conv2d* for both fully connected neural networks (`FCNNs`) and typical convolutional neural networks (`CNNs`) ResNet-50 , Inception-v3, and VGG16.

(Instructions and more details will be shown in relevant package README file)

All the end-to-end CNNs experiments can be found in:
`AutoMCL_Repository/PaperData/e2e_conv2d(10_e2e@intel1)`,
`AutoMCL_Repository/PaperData/e2e_conv2d(e2e20210410@mechrev)`,
`AutoMCL_Repository/PaperData/e2e_conv2d(10_e2e@intel1)`,
`AutoMCL_Repository/PaperData/e2e_conv2d(10_e2e@intel1)`

All the end-to-end FCNNs experiments can be found in:
`AutoMCL_Repository/PaperData/e2e_dense(20210128mxnet@AMD)`,
`AutoMCL_Repository/PaperData/e2e_dense(20210128mxnet@intel2)`

New configuration space exploiting strategies. AutoMCL's performance model (`REG`) replace AutoTVM's exploration module (`SA+RANK`)

```
# AutoMCL XGBTuner: if you want to test REG module in AutoMCL, you should switch
loss_type="regg", optimizer="reg";
REGXGBtuner = autotvm.tuner.XGBTuner(tsk, loss_type="regg", optimizer="reg")
```

The relavant experiments can be found in:
`AutoMCL_Repository/PaperData/RegVSRank(6_end2endE@intel1)`

```
# AutoTVM XGBTuner
SAXGBtuner = autotvm.tuner.XGBTuner(tsk, loss_type="rank", optimizer="sa")
```

tuneMCL.
We introduce several optimization strategies, combining analytic ideal cache models with machine learning models trained with real hardware measures, and integrate them into a unified auto-tuning framework, called AutoMCL, to improve the performance of DL compilers on both the operation level and the end-to-end model inference.

```
# AutoMCL Tuner
XGBtuner.tuneMCL(n_trial=n_trial,
                        early_stopping=early_stopping,
                        measure_option=measure_option,
                        callbacks=[autotvm.callback.progress_bar(n_trial),
                                   autotvm.callback.log_to_file(tmp_log_file)],
                        initConfig=True, useFilter=True, useRecommend=False,
                        sch="conv2d", dtype="float32", L2=256*1024,
cacheline=64,
                        Vl=128/8):
'''
       tuneMCL:
           We introduce several optimization strategies, combining analytic
ideal cache models with machine learning models trained with real hardware
measures, and integrate them into a unified auto-tuning framework, called
AutoMCL, to improve the performance of DL compilers on both the operation level
and the end-to-end model inference.
        :param Vl:  int
                Vl be the length of vectorization(B)
        :param cacheline: int
                the cache line size(B)
        :param L2: int
                cache size(B)
        :param n_trial:  int
                Maximum number of configs to try (measure on real hardware)
        :param measure_option: dict
            The options for how to measure generated code.
            You should use the return value ot autotvm.measure_option for this
argument.
        :param early_stopping: int, optional
            Early stop the tuning when not finding better configs in this number
of trials
        :param callbacks: List of callable
            A list of callback functions. The signature of callback function is
```

```
            (Tuner, List of MeasureInput, List of MeasureResult)
            with no return value. These callback functions will be called on
            every measurement pair. See autotvm/tuner/callback.py for some
examples.
        :param initConfig: boolean
            Select whether to use the initConfig knob
        :param useFilter: boolean
            Select whether to use the filter knob
        :param useRecommend: boolean
            Select whether to use the recommend knob
        :param sch: string
            support 'tmm', 'ttmm', 'dnmm', 'dnmm332', 'dpmm', 'lpmm', 'rpmm',
'rpmmv', 'conv2d','convim2colrpmm','convim2coldnmm' and 'autoschedule'
        :param dtype: string
            float32 or float64
'''
[NOTES]
1.if you want to tuning FCNNs by AutoMCL, you should set options like
"initConfig=True, useFilter=True, useRecommend=False, sch="dense",
dtype="float32", L2=256 * 1024, cacheline=64, Vl=128 / 8" or "initConfig=True,
useFilter=True, useRecommend=False, sch="autoschedule", dtype="float32", L2=256
* 1024, cacheline=64, Vl=128 / 8";

2.if you want to tuning CNNs by AutoMCL, you should set options like
"initConfig=True, useFilter=True, useRecommend=False, sch="autoschedule",
dtype="float32", L2=256 * 1024, cacheline=64, Vl=128 / 8" or "initConfig=True,
useFilter=True, useRecommend=False, sch="conv2d", dtype="float32", L2=256 *
1024, cacheline=64, Vl=128 / 8";

3.if you want to test init-filter module in AutoMCL, you should switch options
"initConfig=True, useFilter=True" ;

#More details will be shown in next chapter——New Developer Features VS TVM
```

For example, Set like `NOTES-2`, More details will be shown `AutoMCL_Repository/UserTest/e2e/tune_model_e2e.py`:

Instructions: `python tune_model_e2e.py 1    # 1 means batch size for CNNs`

Tuning Log Info:

```
Extract tasks...
Tuning...
n_trial= 500
early_stopping= 300

Current/Best:    0.00/   0.00 GFLOPS | Progress: (0/500) | 0.00 sschedule
template is: conv2d
(('TENSOR', (1, 512, 14, 14), 'float32'), ('TENSOR', (512, 512, 3, 3),
'float32'), (1, 1), (1, 1), (1, 1), 'NCHW', 'float32')
space len: 800
parse space start
parse space end,parse cost time is: 0.010681629180908203
configs space before filter: 800
x          1
fx        14
y          1
```

```
fy       512
k          1
fk       512
index    800
dtype: int64

**********************filterConfigSpace start...**********************

conv2d filer ...
if gemm: Tm,Tn,Tk = fx,fy,fk;if conv2d: ow_bn,oc_bn,ic_bn = fx,fy,fk
fx: [1, 2, 7, 14]
fy: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
fk: [1, 2, 4, 8, 16, 32, 64, 128]
**********************filterConfigSpace end.**************************
Step2.filter by cache and vec limitation configs space is:  520

Current/Best:   29.00/  61.30 GFLOPS | Progress: (12/500) | 23.12 s | case
next_batch_filter
Current/Best:    8.97/  61.30 GFLOPS | Progress: (24/500) | 54.16 s | case
next_batch_filter
Current/Best:   38.22/  65.71 GFLOPS | Progress: (36/500) | 77.53 s | case
next_batch_filter
 ......
```

4 New Developer Featrues VS TVM(v0.6.0)

4.1 Operators

4.1.1 dense

```
DNMM:TVM (tvm/topi/python/topi/x86/dense.py _declaration_dense_nopack)
RPMM:TVM (tvm/topi/python/topi/x86/dense.py _declaration_dense_pack)
DNMM332: Modified based on DNMM
RPMMV: operator same with RPMM, but schedule is different
LPMM: added
DPMM: added
TMM: added
TTMM: added
```

4.1.2 conv2d

```
CONV:TVM (tvm/topi/python/topi/x86/conv2d.py _declaration_conv_impl)
CONVOpt: operator same with CONV, but schedule is different
```

4.1.3 conv2d-im2col

```
CONVIm2colRPMMV: added
CONVIm2colDNMM332: added
```

4.2 Schedule

4.2.1 dense

```
DNMM:TVM (tvm/topi/python/topi/x86/dense.py _schedule_dense_nopack_template)
RPMM:TVM (tvm/topi/python/topi/x86/dense.py _schedule_dense_pack_template)
DNMM332: Modified based on DNMM
RPMMV: Modified based on RPMM
LPMM: added
DPMM: added
TMM: added
TTMM: added
```

4.2.2 conv2d

```
CONV1x1: TVM (tvm/topi/python/topi/x86/conv2d_avx_1x1.py _schedule_conv)
CONV:TVM (tvm/topi/python/topi/x86/conv2d_avx_common.py _schedule_conv)
CONVOpt:  Modified based on CONV
```

4.2.3 conv2d-im2col

```
CONVIm2colRPMMV: added
CONVIm2colDNMM332: added
```

4.3 InitConfig

AutoMCL/task/space.py

```
def get_oracle(M,pThread)
def get_gap8s(n)
```

AutoMCL/tuner/model_based_tuner.py

```
def initConfigSpace(self,schedule,n_parallel,isInitConfig,task_args)
def initConfigSpace_Mini(M, N, K, pThread)
def initConfigSpace_PRO(M, N, K, pThread)
```

4.4 FilterConfig

AutoMCL/tuner/model_based_tuner.py

```
def filterConfigSpace(self, schedule, dtype, L2, cacheline, Vl, useFilter,
dataframe, task_args)
```

4.5 REGTuner

AutoMCL/tuner/tuner.py

```
def tuneMCL(self, n_trial, measure_option, early_stopping=None, callbacks=(),
initConfig=False, useFilter=False, useRecommend=False,
               sch="dnmm", dtype="float32", L2 = 256 * 1024, cacheline=64,
Vl=128 / 8)
```

AutoMCL/tuner/xgboost_cost_model.py

```
loss_type == 'regg':
            self.xgb_params = {
                'max_depth': 8,
                'gamma': 0.0001,
                'min_child_weight': 1,
                'subsample': 1.0,
                'eta': 0.3,
                'lambda': 1.00,
                'alpha': 0,
                'objective': 'reg:gamma',
            }
def fit(self, xs, ys, plan_size)
```

AutoMCL/tuner/reg_model_optimizer.py

```
class RegOptimizer(ModelOptimizer):
    def find_maximums(self, model, num, exclusive)
    def top_num(self, points, scores, num, exclusive)
    def top_num_expand(self, points, scores, num, exclusive)
```

More

TVM Docs: https://tvm.apache.org/docs/

Contact Me: chihaoyu@cigit.ac.cn or 2209832406@qq.com

AutoMCL_Repository: