TensorTuner: Auto-tuning TensorFlow Threading Model for CPU Backend

Niranjan Hasabnis
May 4th, 2018.

Question

- How easy it is to get the best performance out of TensorFlow's Eigen and MKL CPU backends?

- What does THE bible say about this.
  1. Build from source – compiler version, compiler options (O3)
  2. Instruction set – AVX2, SSE, etc
  3. Use MKL-DNN
  4. Setting threading params: inter_op, intra_op, OMP/KMP settings

Implementing the advice

Advice	Will following the advice deliver improved performance
Build from source – compiler version, compiler options (O3)	Yes, use better compiler, level O3, etc.
Instruction set – AVX2, SSE, etc	Yes, use -march
Use Intel's MKL	Yes
Setting threading params: inter_op, intra_op, OMP/KMP settings	???

What's advice on setting threading model params

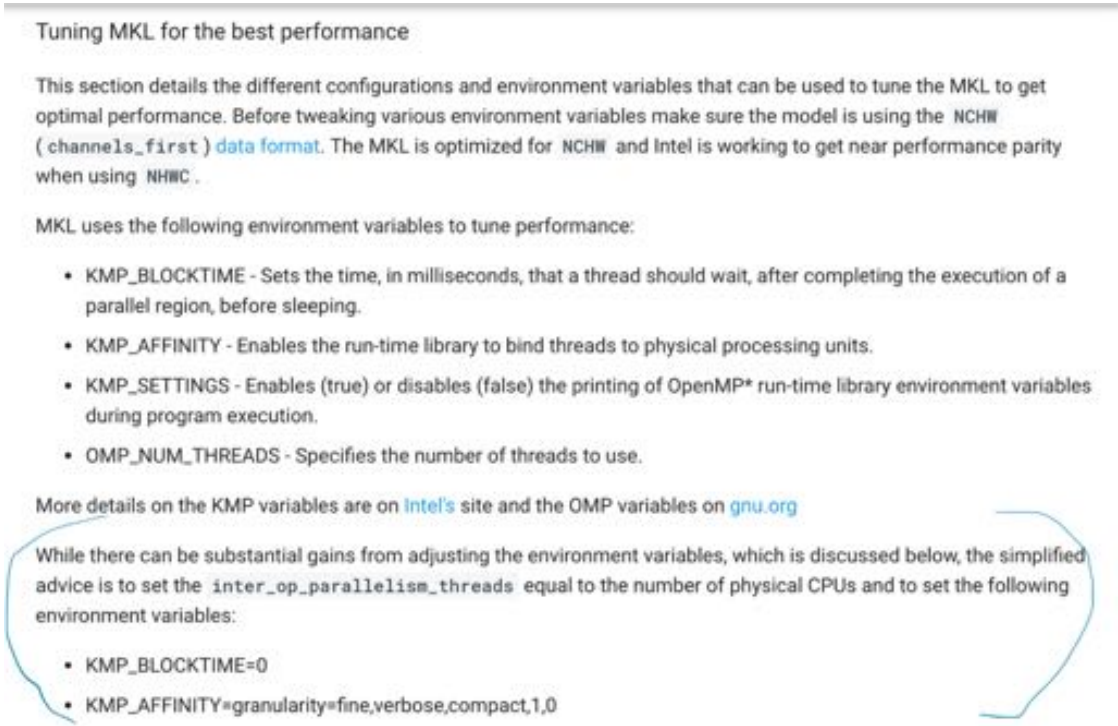
The two configurations listed below are used to optimize CPU performance by adjusting the thread pools.

- `intra_op_parallelism_threads` : Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.
- `inter_op_parallelism_threads` : All ready nodes are scheduled in this pool.

These configurations are set via the `tf.ConfigProto` and passed to `tf.Session` in the `config` attribute as shown in the snippet below. For both configuration options, if they are unset or set to 0, will default to the number of logical CPU cores. Testing has shown that the default is effective for systems ranging from one CPU with 4 cores to multiple CPUs with 70+ combined logical cores. A common alternative optimization is to set the number of threads in both pools equal to the number of physical cores rather than logical cores.

"Testing has show that the default is effective for systems ranging from one CPU with 4 cores to multiple CPUs with 70+ combined logical cores. A common alternative … "

And advice on setting MKL threading model params?



Tuning MKL for the best performance

This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the `NCHW` (`channels_first`) data format. The MKL is optimized for `NCHW` and Intel is working to get near performance parity when using `NHWC`.

MKL uses the following environment variables to tune performance:

- KMP_BLOCKTIME - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- KMP_AFFINITY - Enables the run-time library to bind threads to physical processing units.
- KMP_SETTINGS - Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- OMP_NUM_THREADS - Specifies the number of threads to use.

More details on the KMP variables are on Intel's site and the OMP variables on gnu.org

While there can be substantial gains from adjusting the environment variables, which is discussed below, the simplified advice is to set the `inter_op_parallelism_threads` equal to the number of physical CPUs and to set the following environment variables:

- KMP_BLOCKTIME=0
- KMP_AFFINITY=granularity=fine,verbose,compact,1,0

There are models and hardware platforms that benefit from different settings.

What do we say about tuning MKL?

Settings on Intel® Xeon® Scalable processor (2 Sockets, 28 Cores each) that were used for benchmarking.

	Data format	Intra threads	Inter threads	OMP_NUM_THREAD	KMP_BLOCKTIME
Vgg16	NCHW	56	1	56	1
Inception v3	NCHW	56	2	56	1
Resnet50	NCHW	56	2	56	1

Please note: The parameter settings were carefully tuned to gain maximum performance for the specific platform.

Fact of the matter is…

- *Getting the best performance from TensorFlow's Eigen and MKL CPU backends is not easy!*
- Manual tuning is
  - **Time-consuming** (may be ok for *offline* tuning, not for *online*)
    - exponentially-growing search space makes offline tuning expensive
  - **May not guarantee optimal settings** (if exhaustive sweep is not performed)
- *An automated tool that suggests optimal settings for threading model and that is efficient (Online-tuning for OOB) is desirable*.

That’s where TensorTuner comes in

- Auto-tuning HPC applications is a well-researched area.

Problem formulation

- Getting best performance from TensorFlow's CPU backend is a function maximization problem.
- Performance depends on:
  - Model along with its hyper-parameters and input datasets
  - Configuration of the machine used for execution: Hardware config (micro-arch, cache sizes, etc), Software config (TF version, compiler version, Python version, etc)
- Precise formulation requires assuming certain params are constants.

Problem formulation

- For this problem, we assume that:
  - Model along with its hyper-parameters, input dataset is defined
  - Hardware and software configurations are defined
  - Represent this as: C
- Then performance $\boldsymbol{f}$ can be defined as:

$$\boldsymbol{s} = \boldsymbol{f}_{C}(\boldsymbol{\Sigma})$$

- Where
  - $\boldsymbol{s}$ is performance score (imgs/sec)
  - $\boldsymbol{\Sigma}$ is the set of params that we want to tune: $\{\boldsymbol{p}_1, \boldsymbol{p}_2, .., \boldsymbol{p}_n\}$, where $\boldsymbol{p}$ is a parameter.

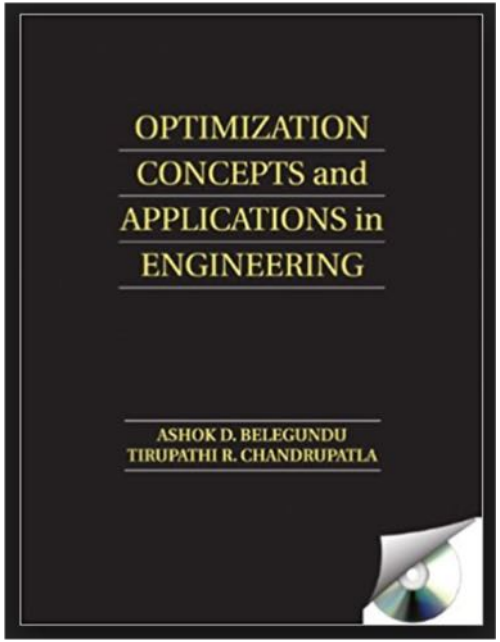
Problem formulation

- With $\boldsymbol{\Sigma}$ is the set of params, set of instantiations of $\boldsymbol{\Sigma}$ (represented using $\boldsymbol{\tau}$) represent param search space.
- Instance $\boldsymbol{t}$ of $\boldsymbol{\tau}$ looks like $\{(\boldsymbol{p}_{1,}\, \boldsymbol{v}_1), \ldots, (\boldsymbol{p}_{n,}\, \boldsymbol{v}_n)\}$, where $\boldsymbol{v}$ is a param's value.
- Then the problem of tuning Tensorflow's threading model for best performance can be defined as:

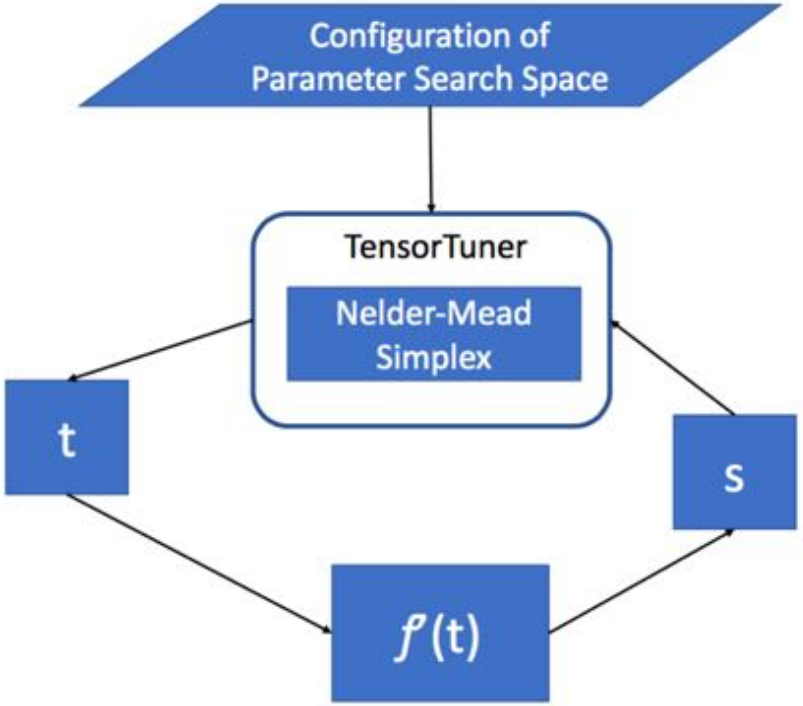
$$find \quad t \in \tau \mid f(t) > f(t') \, \forall \, t' \in \tau \wedge t \neq t'$$

Algorithm for function maximization

- Algorithms
  - **Gradient-based optimizers**
    - Uses gradient of the objective functions to determine search direction
    - Limitations: difficulty handling non-differentiable functions, discontinuous functions
  - **Gradient-free optimizers**
    - Ability to handle problems that cannot be solved using gradient-based optimizers
    - Designed as global optimizers
    - E.g., Nelder-Mead Simplex, Simulated Annealing, Particle Swarm optimizations, Genetic algorithms
- TensorTuner uses Nelder-Mead Simplex algorithm. It has been shown to be effective even today.



Design


Configuration of
Parameter Search Space
TensorTuner
Nelder-Mead
Simplex
t
s
f(t)

Implementation

- Uses open-source Active Harmony tool from UoW (https://dyninst.org/harmony).
- Tool comes with support for Nelder-Mead algorithm
- Wrote wrapper shell script to accept param configs and invoke target script

Evaluation: Criteria

1. Tuning Quality
   - Measures ability of the algorithm to find global optimum setting
   - Measured as $\boldsymbol{f}(\boldsymbol{t}_{\text{suggested}})$
   - Compare with $\boldsymbol{f}(\boldsymbol{t}_{\text{best-known}})$: comes from our blog, running Eigen with default settings suggested by Google
2. Tuning Efficiency
   - Measures ability of the algorithm to converge to global optimum quickly
   - Measured as % of the param space explored before getting to $\boldsymbol{t}_{\text{suggested}}$

Evaluation: setup

- Xeon 8180, Cent OS, GCC-6.3, Python-2.7.5
- Eigen backend: TF-1.7 wheel
- MKL backend: built wheel from TF master (sometime in March)
- Tensorflow tf_cnn_benchmarks

Backend	Model	Batch Size	Data Format
MKL CPU	ResNet-50	128	NCHW
MKL CPU	Inception3	64	NCHW
MKL CPU	VGG16	128	NCHW
MKL CPU	VGG11	128	NCHW
MKL CPU	GoogLeNet	96	NCHW
MKL CPU	AlexNet	256	NCHW
Eigen CPU	ResNet-50	128	NHWC
Eigen CPU	Inception3	64	NHWC
Eigen CPU	VGG16	128	NHWC
Eigen CPU	VGG11	128	NHWC
Eigen CPU	GoogLeNet	96	NHWC
Eigen CPU	AlexNet	256	NHWC

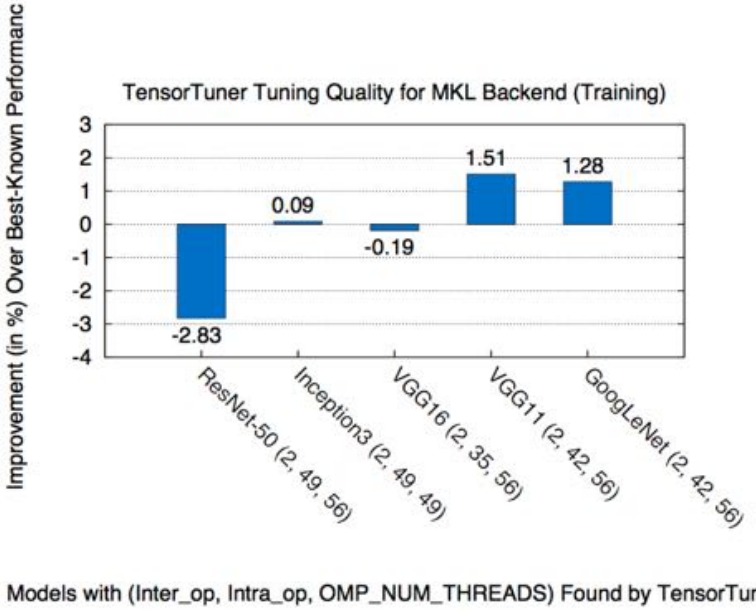
Evaluation: param search space configuration

Backend	inter_op	intra_op	OMP_NUM_THREADS
MKL CPU	[1, 4, 1]	[14, 56, 7]	[14, 56, 7]
Eigen CPU	[1, 4, 1]	[14, 56, 7]	-

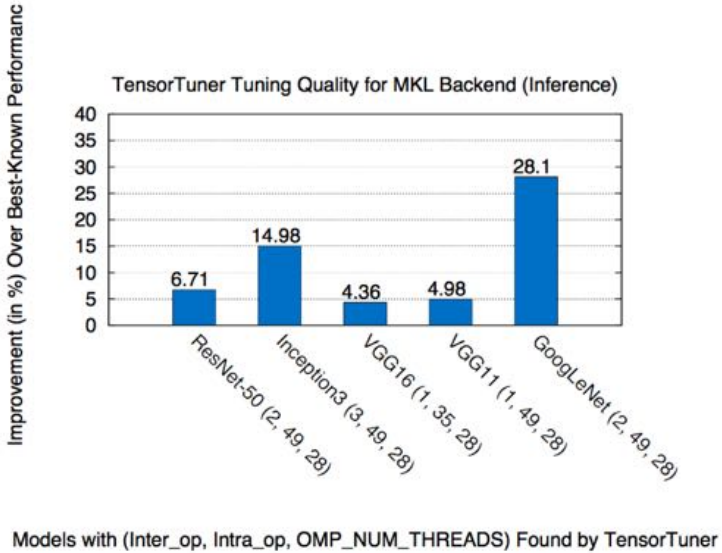
Figure 3: Lower bound, upper bound and step size used for parameter search

- Constraints are not necessary. NM can handle unconstrained searches.

Evaluation: Tuning quality for MKL backend



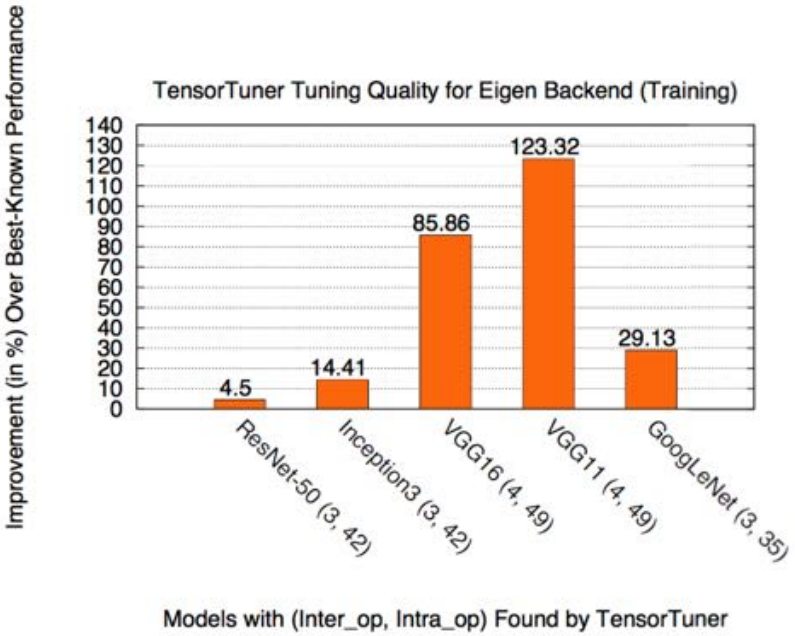
Models with (Inter_op, Intra_op, OMP_NUM_THREADS) Found by TensorTuner

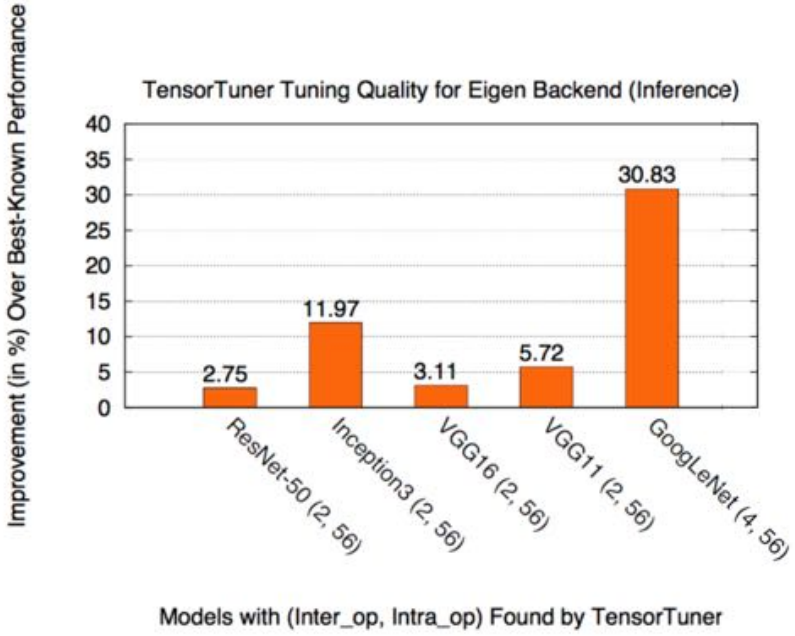


Models with (Inter_op, Intra_op, OMP_NUM_THREADS) Found by TensorTuner

With $t_{suggested}$, we get 2% to 28% improvement in TF performance with MKL backend for Inference.

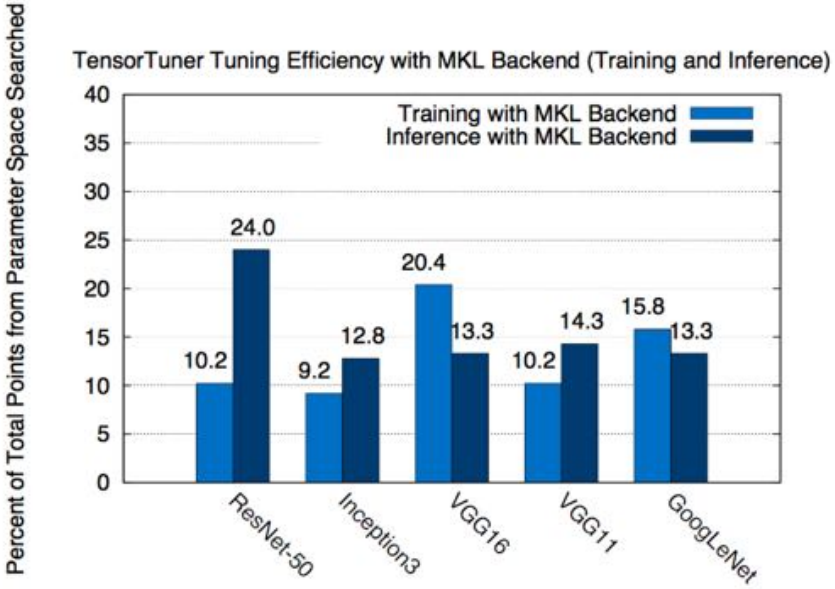
Evaluation: tuning quality with Eigen backend

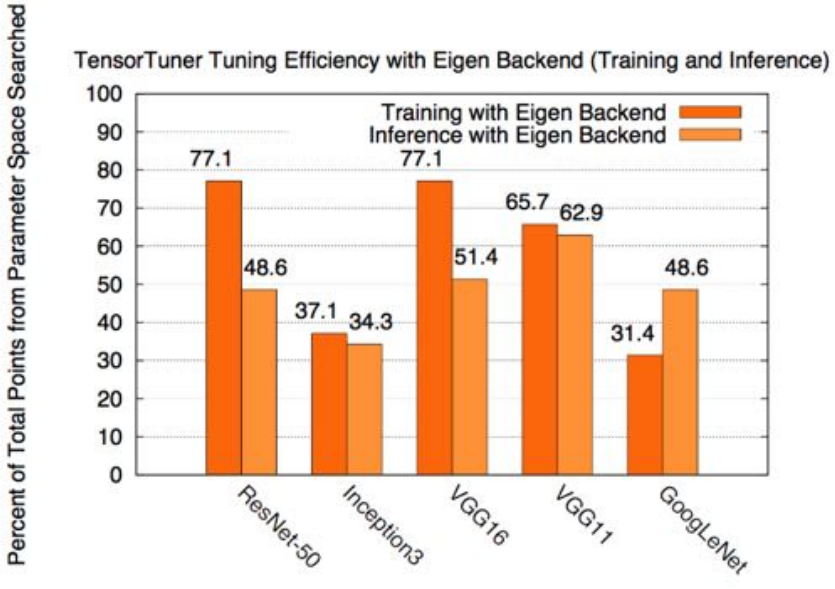




With $t_{suggested}$, we get 2% to 123% improvement in TF performance with Eigen backend (Training), and 2% to 30% improvement for Inference.

Evaluation: Tuning efficiency





Conclusion

- Manual param tuning is time-consuming, inefficient and does not work for OOB cases.
- An automated tool that tunes TF's threading model efficiently is possible.
- Settings suggested by TensorTuner improves Eigen and MKL performance from 2% to 123%.