Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions

Jiazheng Yuan

Problem

1.Newly invented operator might suffer from performance cost if no available function

call is present in the framework library

2.Even if there is, might not be optimal for the user’s network and dataset,

Existing functions might not tuned for all combination of data sizes

3.Computation graph are too abstract to capture all detail

Solution:Tensor Comprehension

(1) a language close to the mathematics of deep learning,

(2) a polyhedral Just-In-Time compiler to convert a mathematical description of a deep learning DAG into a CUDA kernel with delegated memory management and synchronization, also providing optimizations such as operator fusion and specialization for specific sizes

(3) a compilation cache populated by an autotuner.

Current approaches

1.Halide, a image processing library, requiring users to do scheduling by

themselves, often too hard for most users, 2d only

2. Active library which generate code on demand, which is still hard to cover all cases.

3.Current deep learning Compilers is on its way,such as XLA and Latte, performance level is still not met on GPU.

Advantage

1. Concise and expressive
2. Specializing a polyhedral intermediate representation and its compilation algorithms to the domain of deep learning
3. general enough to be integrated into other ML frameworks
4. An end-to-end compilation flow capable of lowering tensor comprehensions to efficient GPU code
5. TC closely matches an algorithmic notation

Syntax

Actual TC code:

```
def mv(float(M,K) A, float(K) x) → (C) {
  C(i)   = 0
  C(i) += A(i,k) * x(k)
}
```

Equivalent c-style pseudo-code:

```
tensor C({M}).zero(); // 0-filled single-dim tensor
parallel for (int i = 0; i < M; i++)
  reduction for (int k = 0; k < K; k++)
    C(i) += A(i,k) * x(k);
```

Rule:1.index implicitly defined. Bound inferred

2.Index only on right will be reduced

3.order of evaluation point in iteration space does not matter

Syntax

Initialization: append ! to operator to initialize

```
def maxpool2x2(float(B,C,H,W) in) → (out) {
  out(b,c,i,j) max=! in(b,c, 2 * i + kw, 2 * j + kh)
    where kw in 0:2, kh in 0:2
}
```

Range of variable must be specified using kw, in case they cannot be inferred, in this case, by specifying kw,

It can be infered that 0=<2*i & 2*i + 2 < H. 0=<2*j & 2*j + 2 < W. Thus each out value is taking the max of four values

Range Inferring

```
def sgemm(float a, float b,
          float(N,M) A, float(M,K) B) → (C) {
  C(i,j)  = b * C(i,j)              # initialization
  C(i,j) += a * A(i,k) * B(k,j)     # accumulation
}
```

1.Making variable as large as possible without going out of bound, if element appeared twice, take the less range(this is taken as a example for compilation process). Loops are implicit, and optimizations are performed automatically

```
def conv1d(float(M) I, float(N) K) → (O)
  O(i) = K(x) * I(i + x)
}
```

2.Find the range of expression containing only one variable, then deduce ones containing more variables using the already obtained range. 0=<X<N, 0< i <= M-N .

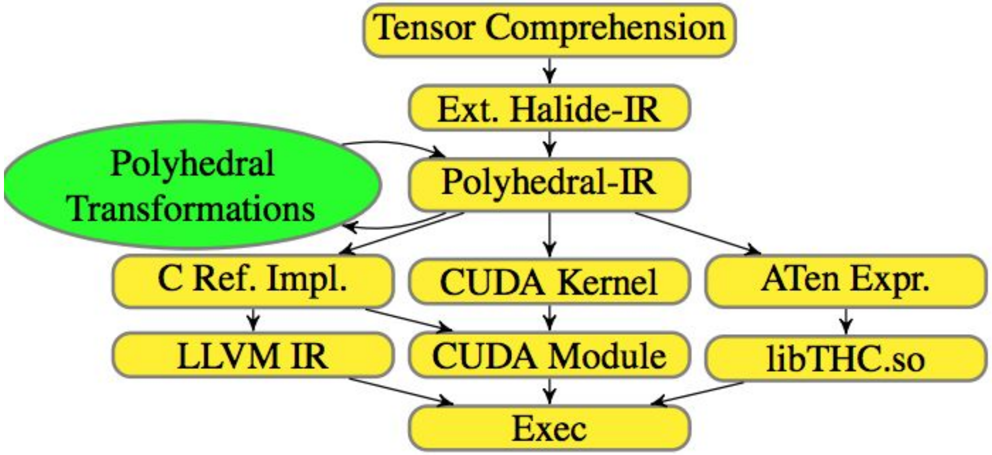
Integration

1. Translating tensor of other framework to TC’s own(**DLPack support, a header only library providing meta info only for converting tensor to TC’S**)

2. Overriding operators to generate TC.

3. Currently support Pytorch and caffe2

Workflow description



Can also generate a readable naive cuda for reference

Polyhedral JIT Compilation

schedule trees structure:

1.A band node :defines a partial execution order through one or multiple piecewise affine functions

2.Filter node: binding its subtree to a subset of the iteration domain

3.Context nodes provide additional information on the variables and parameters

4.extension nodes introduce auxiliary computations that are not part of the original iteration domain

5. Sequence node: specify execution order if necessary

Example code to be compiled

```
def sgemm(float a, float b,
          float(N,M) A, float(M,K) B) → (C) {
  C(i,j)  = b * C(i,j)              # initialization
  C(i,j) += a * A(i,k) * B(k,j)     # accumulation
}
```

Compilation steps

A canonical schedule tree for a TC is defined by an outer Sequence node, followed by Filter nodes for each TC statement. Inside each filtered branch, Band nodes define an identity schedule with as many one-dimensional schedule functions as loop iterators for the statement

$$
\text{Domain}\left[\begin{array}{ll}
\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\
\{\mathtt{T}(i,j,k) & \mid 0 \le i < N \\
 & \quad \wedge 0 \le j < K \wedge 0 \le k < M\}
\end{array}\right.
$$

$$
\begin{array}{l}
\text{Sequence} \\
\quad \text{Filter}\{\mathtt{S}(i,j)\} \\
\qquad \text{Band}\{\mathtt{S}(i,j) \to (i,j)\} \\
\quad \text{Filter}\{\mathtt{T}(i,j,k)\} \\
\qquad \text{Band}\{\mathtt{T}(i,j,k) \to (i,j,k)\}
\end{array}
$$

(a) canonical `sgemm`

Autotuning

Command for tuning

tensor_comprehensions.autotune(*tc*, *entry_point*, **inputs*, *starting_options=None*, *tuner_config=<tensor_comprehensions.tclib.TunerConfig object>*, *cache_filename=None*, *load_from_cache=False*, *store_to_cache=False*)

1.either through *starting_options* or (*load_from_cache* and *cache_filename*)

2. Different method of *tensor_comprehensions.tclib.MappingOptions* can achieve different strategy

3.How to choose starting mapping options? Don't., better use default: i.e:*makeConvolutionMappingOptions()*

Fused

This tree features an outer band node with i and j loops that became common to both statements, which corresponds to loop fusion. The sequence node ensures that instances of S are executed before respective instances of T enabling proper initialization. The second band is only applicable to T and corresponds to the innermost (reduction) loop k

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & \mid 0 \le i < N \\ & \quad \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad \text{Sequence} \\
\quad\quad \text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad \text{Band}\{\mathtt{S}(i,j) \to (i,j)\} \\
\quad\quad \text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad \text{Band}\{\mathtt{T}(i,j,k) \to (i,j,k)\}
\end{array}
$$

(a) canonical sgemm

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & \mid 0 \le i < N \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad \text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \to (i,j)\} \\ \{\mathtt{T}(i,j,k) & \to (i,j)\}\end{array}\right. \\
\quad\quad \text{Sequence} \\
\quad\quad\quad \text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad \text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad \text{Band}\{\mathtt{T}(i,j,k) \to (k)\}
\end{array}
$$

(b) fused

After this step, loop of initialization and execution are fused

Transform command:

.scheduleFusionStrategy(<choice of Max, Preserve3Coincident, Min>)

Loop Tiling

it converts a permutable schedule band into a chain of two bands with the outer band containing tile loops and the inner band containing point loops with fixed trip count

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & \mid 0 \le i < N \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad\text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \to (i,j)\} \\ \{\mathtt{T}(i,j,k) & \to (i,j)\}\end{array}\right. \\
\quad\quad\text{Sequence} \\
\quad\quad\quad\text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad\text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (k)\}
\end{array}
$$

(b) fused

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & \mid 0 \le i < N \\ & \quad \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad\text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \to (32\lfloor i/32 \rfloor, 32\lfloor j/32 \rfloor)\} \\ \{\mathtt{T}(i,j,k) & \to (32\lfloor i/32 \rfloor, 32\lfloor j/32 \rfloor)\}\end{array}\right. \\
\quad\quad\text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \to (i \bmod 32, j \bmod 32)\} \\ \{\mathtt{T}(i,j,k) & \to (i \bmod 32, j \bmod 32)\}\end{array}\right. \\
\quad\quad\quad\text{Sequence} \\
\quad\quad\quad\quad\text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad\quad\text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (k)\}
\end{array}
$$

(c) fused and tiled

Comparing to step(b), tiling is applied which is an additional optimization

Transform command

.tile(<list of positive integers>)

C style equivalent

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\texttt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\texttt{T}(i,j,k) & \mid 0 \le i < N \\ & \quad \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad \text{Band}\left[\begin{array}{ll}\{\texttt{S}(i,j) & \to (32\lfloor i/32 \rfloor, 32\lfloor j/32 \rfloor)\} \\ \{\texttt{T}(i,j,k) & \to (32\lfloor i/32 \rfloor, 32\lfloor j/32 \rfloor)\}\end{array}\right. \\
\quad\quad \text{Band}\left[\begin{array}{ll}\{\texttt{S}(i,j) & \to (i \bmod 32, j \bmod 32)\} \\ \{\texttt{T}(i,j,k) & \to (i \bmod 32, j \bmod 32)\}\end{array}\right. \\
\quad\quad\quad \text{Sequence} \\
\quad\quad\quad\quad \text{Filter}\{\texttt{S}(i,j)\} \\
\quad\quad\quad\quad \text{Filter}\{\texttt{T}(i,j,k)\} \\
\quad\quad\quad\quad\quad \text{Band}\{\texttt{T}(i,j,k) \to (k)\}
\end{array}
$$

(c) fused and tiled

```
for(int i = 0;i / 32 * 32< N; i +=32){
    for(int j = 0; j /32 * 32 < K;j+=32){
        for(int ii = i; ii < min(i + 32,N); ii+=1){


                for(int jj = j; jj < min(j + 32,K);jj++){
                   initialization();

                    for(int k = 0; k<M;k++){
                        accumulation();
                    }
                }
        }
    }
}
```

Sunk parallel loop point

imperfectly nested tiling is implemented by first tiling all bands in isolation and then sinking parallel point loops in the tree

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{l|l} \{\mathtt{S}(i,j) & 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & 0 \le i < N \\ & \wedge 0 \le j < K \wedge 0 \le k < M\} \end{array}\right. \\
\quad \text{Band}\left[\begin{array}{ll} \{\mathtt{S}(i,j) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\} \\ \{\mathtt{T}(i,j,k) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\} \end{array}\right. \\
\quad\quad \text{Band}\left[\begin{array}{ll} \{\mathtt{S}(i,j) & \to (i \bmod 32, j \bmod 32)\} \\ \{\mathtt{T}(i,j,k) & \to (i \bmod 32, j \bmod 32)\} \end{array}\right. \\
\quad\quad\quad \text{Sequence} \\
\quad\quad\quad\quad \text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad\quad \text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad\quad \text{Band}\{\mathtt{T}(i,j,k) \to (k)\}
\end{array}
$$

(c) fused and tiled

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{l|l} \{\mathtt{S}(i,j) & 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & 0 \le i < N \wedge 0 \le j < K \wedge 0 \le k < M\} \end{array}\right. \\
\quad \text{Band}\left[\begin{array}{ll} \{\mathtt{S}(i,j) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\} \\ \{\mathtt{T}(i,j,k) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\} \end{array}\right. \\
\quad\quad \text{Sequence} \\
\quad\quad\quad \text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad\quad \text{Band}\{\mathtt{S}(i,j) \to (i \bmod 32, j \bmod 32)\} \\
\quad\quad\quad \text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad \text{Band}\{\mathtt{T}(i,j,k) \to (32\lfloor k/32\rfloor)\} \\
\quad\quad\quad\quad\quad \text{Band}\{\mathtt{T}(i,j,k) \to (k \bmod 32)\} \\
\quad\quad\quad\quad\quad\quad \text{Band}\{\mathtt{T}(i,j,k0 \to (i \bmod 32, j \bmod 32)\}
\end{array}
$$

(d) fused, tiled and sunk

Comparing to step(c), access on dimension K is also tiled, thus achieved imperfect loop tiling(not all computations Are in inner loop)

Transform command:
.scheduleFusionStrategy(<choice of Max, Preserve3Coincident, Min>)

C style equivalent

$$\text{Domain}\left[\begin{array}{l|l}\{\mathtt{S}(i,j) & 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & 0 \le i < N \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right.$$

$$\text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \rightarrow (32\lfloor i/32 \rfloor, 32\lfloor j/32 \rfloor)\} \\ \{\mathtt{T}(i,j,k) & \rightarrow (32\lfloor i/32 \rfloor, 32\lfloor j/32 \rfloor)\}\end{array}\right.$$

Sequence

Filter$\{\mathtt{S}(i,j)\}$

Band$\{\mathtt{S}(i,j) \rightarrow (i \bmod 32, j \bmod 32)\}$

Filter$\{\mathtt{T}(i,j,k)\}$

Band$\{\mathtt{T}(i,j,k) \rightarrow (32\lfloor k/32 \rfloor)\}$

Band$\{\mathtt{T}(i,j,k) \rightarrow (k \bmod 32)\}$

Band$\{\mathtt{T}(i,j,k0 \rightarrow (i \bmod 32, j \bmod 32)\}$

(d) fused, tiled and sunk

```
for(int i = 0;i / 32 * 32< N; i +=32){
    for(int j = 0; j /32 * 32 < K;j+=32){
        for(int ii = i; ii < min(i + 32,N); ii+=1){


                for(int jj = j; jj < min(j + 32,K);jj++){
                   initialization();

                }
        }
        for(int k = 0; k /32 * 32 < M;k+=32){
            for(int kk = k; kk < min(k + 32,M); kk+=1){
                for(int ii = i; ii < min(i + 32,N); ii+=1){

                    for(int jj = j; jj < min(j + 32,K);jj++){
                        accumulation();

                    }
                }


            }
        }
    }
}
```

Gpu Mapping

Mapping to GPU, adding parameters. Comparing to step(d), this becomes code on GPU, GPU parameter

Added, such as tx,ty,bx,by, corresponding to block and thread id

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & \mid 0 \le i < N \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad\text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\} \\ \{\mathtt{T}(i,j,k) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\}\end{array}\right. \\
\quad\quad\text{Sequence} \\
\quad\quad\quad\text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad\quad\text{Band}\{\mathtt{S}(i,j) \to (i \bmod 32, j \bmod 32)\} \\
\quad\quad\quad\text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (32\lfloor k/32\rfloor)\} \\
\quad\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (k \bmod 32)\} \\
\quad\quad\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k0 \to (i \bmod 32, j \bmod 32)\}
\end{array}
$$

(d) fused, tiled and sunk

$$
\begin{array}{l}
\text{Domain}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid 0 \le i < N \wedge 0 \le j < K\} \\ \{\mathtt{T}(i,j,k) & \mid 0 \le i < N \wedge 0 \le j < K \wedge 0 \le k < M\}\end{array}\right. \\
\quad\text{Context}\{0 \le b_x, b_y < 32 \wedge 0 \le t_x, t_y < 16\} \\
\quad\text{Filter}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \mid i - 32b_x - 31 \le 32 \times 16\lfloor i/32/16\rfloor \le i - 32b_x \wedge \\ & \quad j - 32b_y - 31 \le 32 \times 16\lfloor j/32/16\rfloor \le j - 32b_y\} \\ \{\mathtt{T}(i,j,k) & \mid i - 32b_x - 31 \le 32 \times 16\lfloor i/32/16\rfloor \le i - 32b_x \wedge \\ & \quad j - 32b_y - 31 \le 32 \times 16\lfloor j/32/16\rfloor \le j - 32b_y\}\end{array}\right. \\
\quad\quad\text{Band}\left[\begin{array}{ll}\{\mathtt{S}(i,j) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\} \\ \{\mathtt{T}(i,j,k) & \to (32\lfloor i/32\rfloor, 32\lfloor j/32\rfloor)\}\end{array}\right. \\
\quad\quad\quad\text{Sequence} \\
\quad\quad\quad\quad\text{Filter}\{\mathtt{S}(i,j)\} \\
\quad\quad\quad\quad\quad\text{Filter}\ \{\mathtt{S}(i,j) \mid (t_x - i) = 0 \bmod 16 \wedge (t_y - j) = 0 \bmod 16\} \\
\quad\quad\quad\quad\quad\quad\text{Band}\{\mathtt{S}(i,j) \to (i \bmod 32, j \bmod 32)\} \\
\quad\quad\quad\quad\text{Filter}\{\mathtt{T}(i,j,k)\} \\
\quad\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (32\lfloor k/32\rfloor)\} \\
\quad\quad\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (k \bmod 32)\} \\
\quad\quad\quad\quad\quad\quad\quad\text{Filter}\ \begin{array}{l}\{\mathtt{T}(i,j,k) \mid (t_x - i) = 0 \bmod 16 \wedge \\ \quad\quad\quad\quad (t_y - j) = 0 \bmod 16\}\end{array} \\
\quad\quad\quad\quad\quad\quad\quad\quad\text{Band}\{\mathtt{T}(i,j,k) \to (i \bmod 32, j \bmod 32)\}
\end{array}
$$

(e) fused, tiled, sunk and mapped

This series of steps is autotuning, but user can also specifies dependency graph to tell what is allowed and what is not to be optimized.

Transform command:

.mapToBlocks(<list of 1..3 positive integers>) .mapToThreads(<list of 1..3 positive integers>)

Autotuning and Caching

balance out the cost of JIT compilation, caching and autotuning is used:

- Caching:each entry key is a tuple (TC, input,shapes, target, architecture),Entry is the fastest know version. There could be pre-populated reference implementation to avoid unpredictably long compilation time.

Autotuning and Caching

3 parts to set up

1. a set of starting configurations that worked well for similar TCs, and a few predefined strategies

2. the tuning space dimensions and admissible values for ranges;

3. the type of search—currently a genetic algorithm or random.

Autotuning and Caching

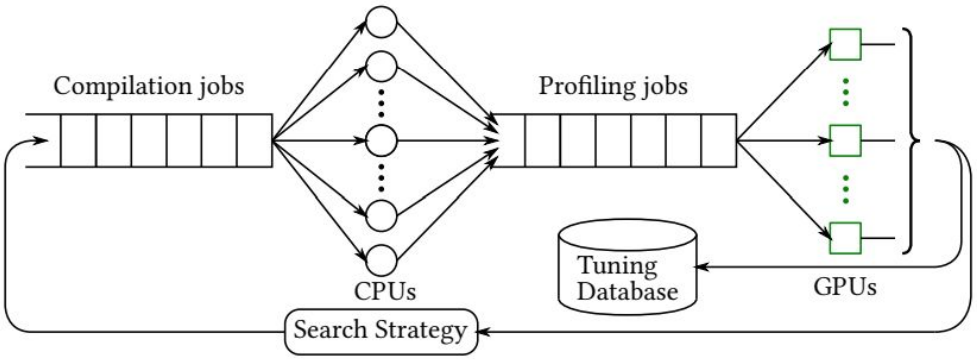


Figure 4: Multithreaded autotuning pipeline for kernels

Tuning process: Multiple candidates compiled profiled and run, result used to generate the candidate for next phase,

A database to save all performance data for all versions, useful when additional technique implemented such as Bayesian hyperparameter search

Parameters for tuning:Block size,grid size, fusion strategy, shared memory etc.

Performance

Common and Research Kernels

	(B, M, K, N)	$(nil, 128, 32, 256)$			$(nil, 128, 1024, 1024)$			$(nil, 128, 4096, 16384)$				$(500, 72, 26, 26)$		
		p0	p50	p90	p0	p50	p90	p0	p50	p90		p0	p50	p90
tmm	Caffe2 + CUBLAS	33	35	36	127	134	136	3,527	3,578	3,666	tbmm	340	347	350
tmm	ATen + CUBLAS	35	35	36	120	123	125	3,457	3,574	3,705	tbmm	342	348	353
tmm	TC (manual)	32	33	35	441	446	469	24,452	24,583	24,656	tbmm	166	170	172
tmm	TC (autotuned)	28	29	30	309	313	316	14,701	14,750	14,768	tbmm	96	101	110
tmm	Caffe2 + CUBLAS	29	30	31	107	108	109	2,404	2,431	3,068	tbmm	189	192	197
tmm	ATen + CUBLAS	26	27	27	104	106	108	2,395	2,409	3,043	tbmm	188	190	191
tmm	TC (manual)	21	22	23	188	194	210	8,378	8,402	8,411	tbmm	91	92	93
tmm	TC (autotuned)	24	25	26	107	110	111	8,130	8,177	8,251	tbmm	51	53	54

	(N, G, F, C, W, H)	$(32, 32, 16, 16, 14, 14)$			$(32, 32, 32, 32, 7, 7)$			$(32, 32, 4, 4, 56, 56)$			$(32, 32, 8, 8, 28, 28)$		
		p0	p50	p90	p0	p50	p90	p0	p50	p90	p0	p50	p90
gconv	Caffe2 + CUDNN	1,672	1,734	1,764	1,687	1,777	1,802	4,078	4,179	4,206	3,000	3,051	3,075
gconv	ATen	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
gconv	TC (manual)	6,690	6,752	6,805	3,759	3,789	3,805	2,866	2,930	2,959	3,939	4,009	4,045
gconv	TC (autotuned)	666	670	673	1,212	1,215	1,216	1,125	1,144	1,159	847	863	870
gconv	Caffe2 + CUDNN	1,308	1,343	1,388	1,316	1,338	1,350	4,073	4,106	4,119	1,993	2,021	2,036
gconv	ATen	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
gconv	TC (manual)	3,316	3,339	3,345	2,327	2,348	2,363	1,683	1,691	1,694	1,845	1,870	1,883
gconv	TC (autotuned)	319	321	322	691	705	714	464	481	504	371	377	379

Wall-clock execution of kernels (in µs). Each kernel ran 1000 times. The top half of each table is Tesla M40 (Maxwell) and the bottom half is Tesla P100 (Pascal); N/A indicates the framework lacked an implementation

Performance

Production Models

	1LUT			2LUT		
	p0	p50	p90	p0	p50	p90
Caffe2 + CUBLAS	78	80	82	188	193	207
ATen + CUBLAS	N/A	N/A	N/A	N/A	N/A	N/A
TC (manual)	38	39	40	47	49	52
TC (autotuned)	38	39	40	47	49	52
Caffe2 + CUBLAS	63	64	66	122	125	128
ATen + CUBLAS	N/A	N/A	N/A	N/A	N/A	N/A
TC (manual)	21	22	23	30	31	32
TC (autotuned)	21	22	23	30	30	31

	MLP1			C3			MLP3		
	p0	p50	p90	p0	p50	p90	p0	p50	p90
Caffe2 + CUBLAS	123	125	135	146	159	164	124	128	142
ATen + CUBLAS	109	110	112	128	142	148	188	192	213
TC (manual)	150	157	159	344	349	351	67	68	70
TC (autotuned)	123	125	131	219	224	227	56	57	59
Caffe2 + CUBLAS	123	133	134	107	113	115	129	131	133
ATen + CUBLAS	98	98	99	105	110	112	164	167	168
TC (manual)	91	92	93	275	279	281	48	48	49
TC (autotuned)	79	80	80	117	128	129	45	46	46

Wall-clock execution of kernels (in µs). Each kernel ran 1000 times. The top half of each table is Tesla M40 (Maxwell) and the bottom half is Tesla P100 (Pascal); N/A indicates the framework lacked an implementation

Performance

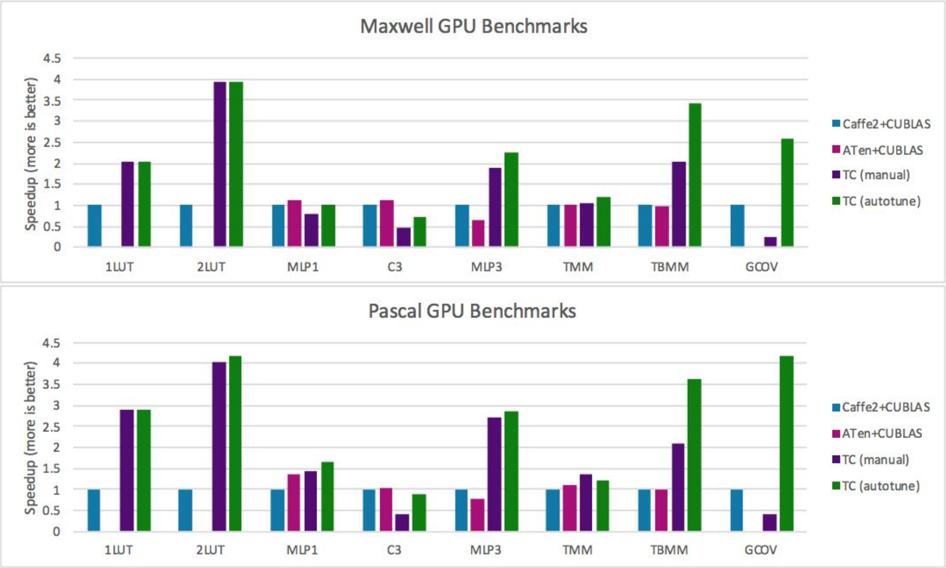
Evaluation:

Except on transposed matrix

Multiplication kernel with large

Enough size, TC performs well

Comparing to the reference.



(Graph representation of the chart on last page)

Conclusion

1. High performance achieved in a variation of kernels

2.Still has space for improvement ,i.e:register tiling.

3.Claims to be concise,expressive, and easy to understand. It’s hard to compare.

4.Quite new framework,needs time to see how well it develops.