TVM at Facebook

Lots of contributors at FB and elsewhere

Why TVM?

- Performance matters a lot
- Heterogenous computing environment
- High variety of workloads
- Ever-increasing set of primitives (over 500 `aten` kernels)
- Interpreter methods not delivering generalized performance

TVM for Speech Synthesis

- WaveRNN-style model architecture
- Autoregressive sampling net running at faster than real-time
- Compute split between GRU units and FC layers
- 24kHz sampling frequency requires **40us sampling net runtime**
- First PyTorch model used a **3,400us sampling net runtime**

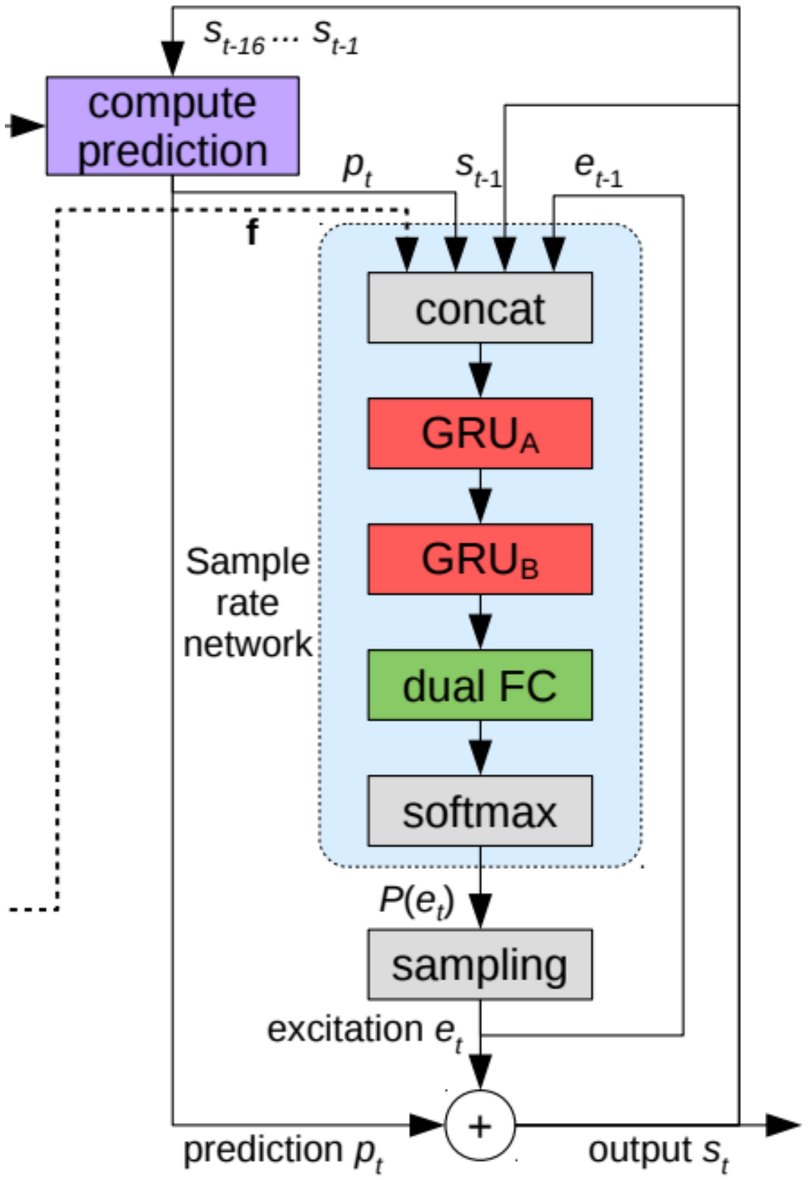


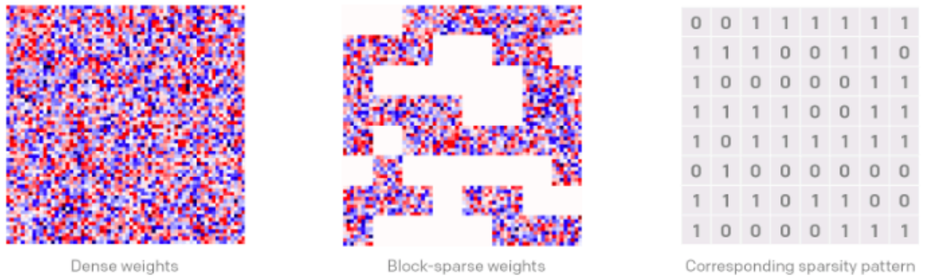
Image from LPCNet

Exit, Pursued By A Bear

- **3400us (baseline), 40us (target)**
- **85x speedup**
- **Uh oh**

Enter, TVM and model co-design

- PyTorch operator overhead makes interpreter infeasible
- Reduce FLOPs with block-sparsified weight matrices
  - not a new idea, cf WaveRNN, Sparse Transformers, etc
- Reduce precision with int8/float16
  - very helpful to maintain model in core-private L1 dcaches
- Use rational approximations for transcendentals (exp, tanh, erf, etc)
  - very general technique, allows clean vectorization
- Related work in Gibiansky (2017), Gray (2019), et al.



Dense weights | Block-sparse weights | Corresponding sparsity pattern

Visualization of dense (left) and block-sparse (center) weight matrices, where white indicates a weight of zero.

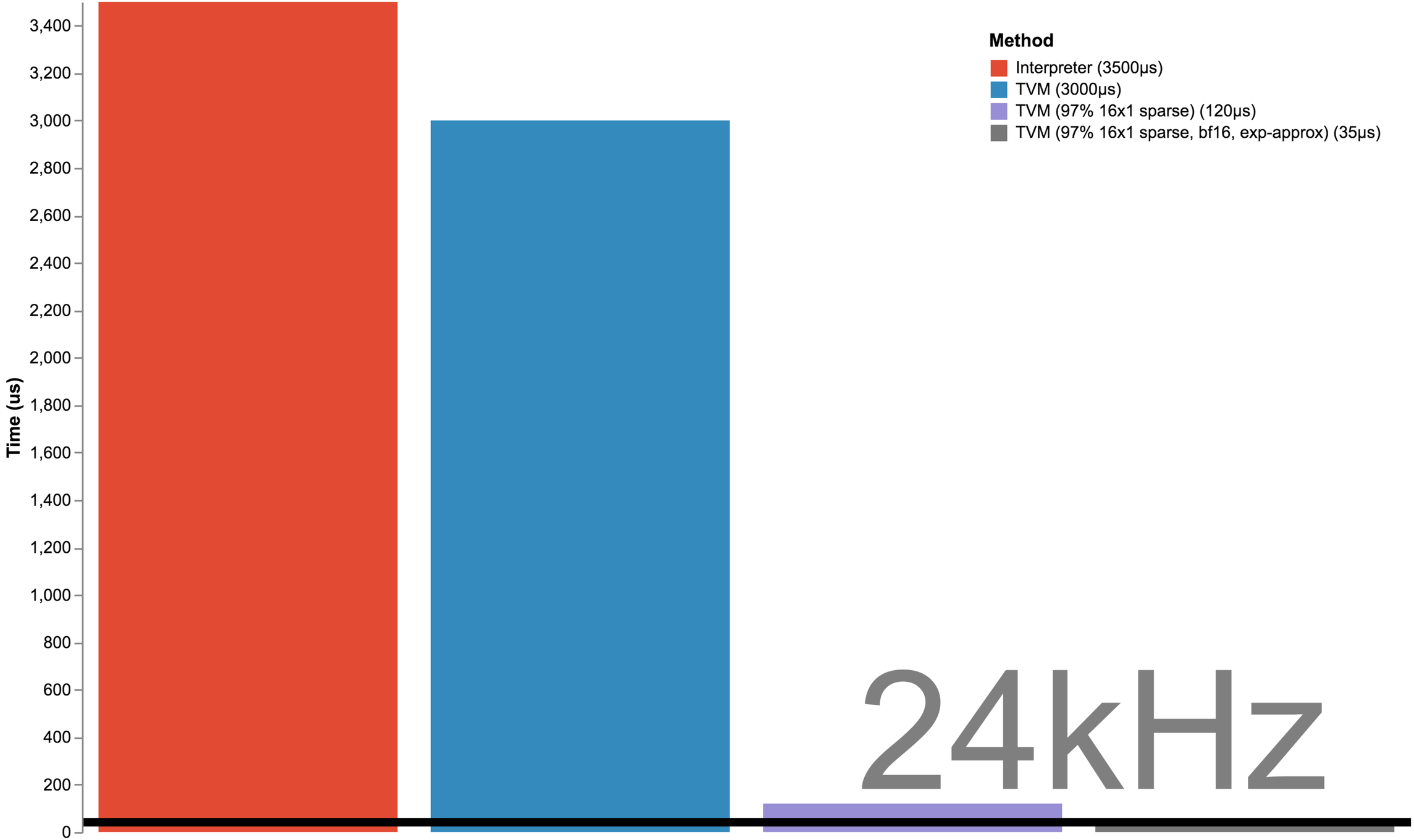
Image from OpenAI

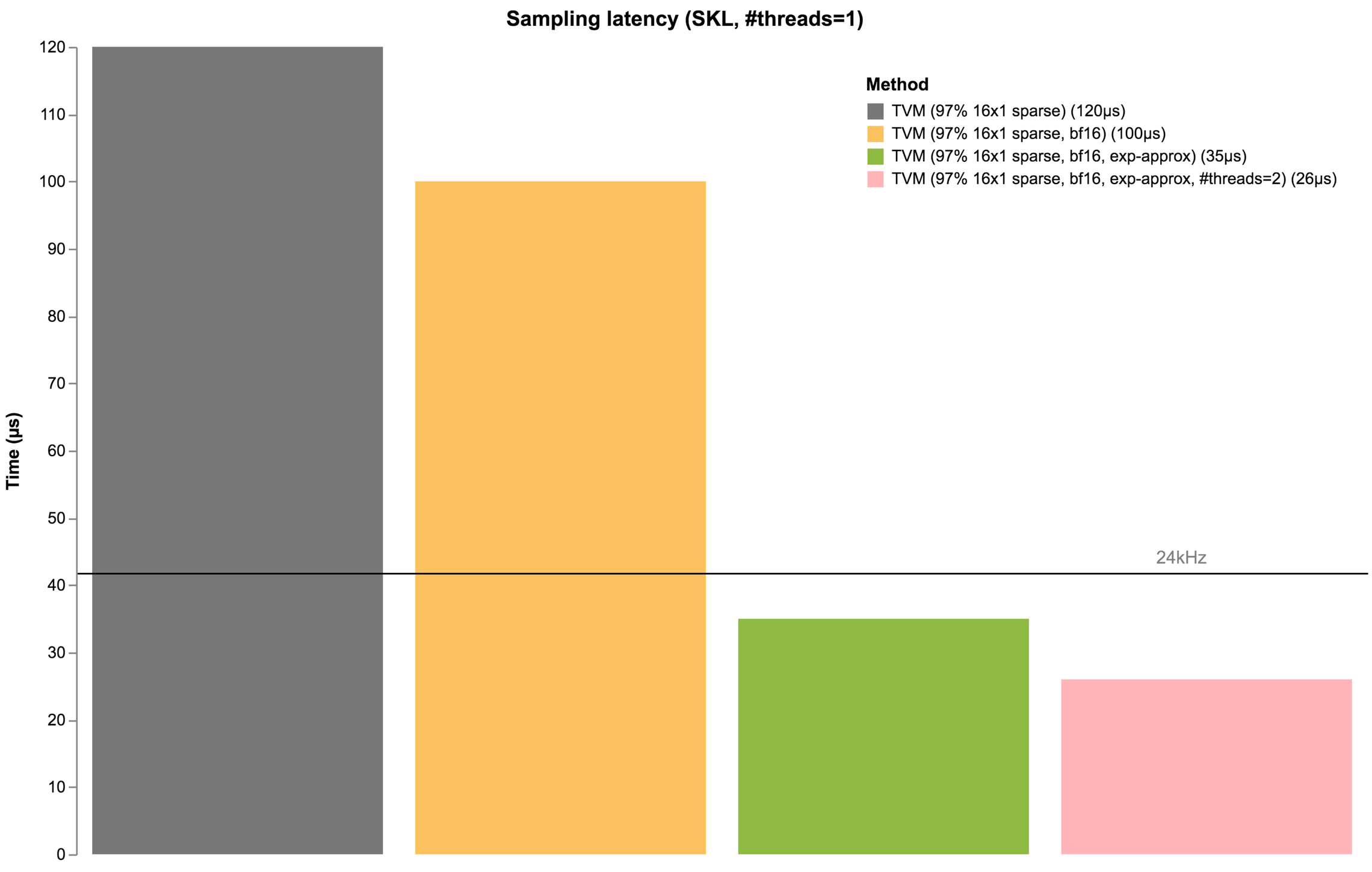
```
def approx_exp(x):
    x = relay.minimum(relay.maximum(x, C(-88.0)), C(88.0))
    x = C(127.0) + x * C(1.44268504)

    i = relay.cast(x, "int32")
    xf = relay.cast(i, "float32")
    x = x - xf
    Y = C(0.99992522) + x * (C(0.69583354) + x \
        * (C(0.22606716) + x * C(0.078024523)))
    exponent = relay.left_shift(i, relay.expr.const(23, "int32"))
    exponent = relay.reinterpret(exponent, "float32")
    return exponent * Y
```

TVM specifics

- Add `relay.nn.sparse_dense` for block-sparse matrix multiplication (~50 lines of TVM IR)
- Add `relay.reinterpret` to implement rational approximations in user space (~10 lines of Relay IR)
- A few days of work
- TVM sampling model running in **30us on single server CPU core**
- Beat hand-written, highly optimized baselines (https://github.com/mozilla/LPCNet) by ~40%
- Bonus: **Real-time on mobile CPUs for free**

Sampling latency (SKL, #threads=1)

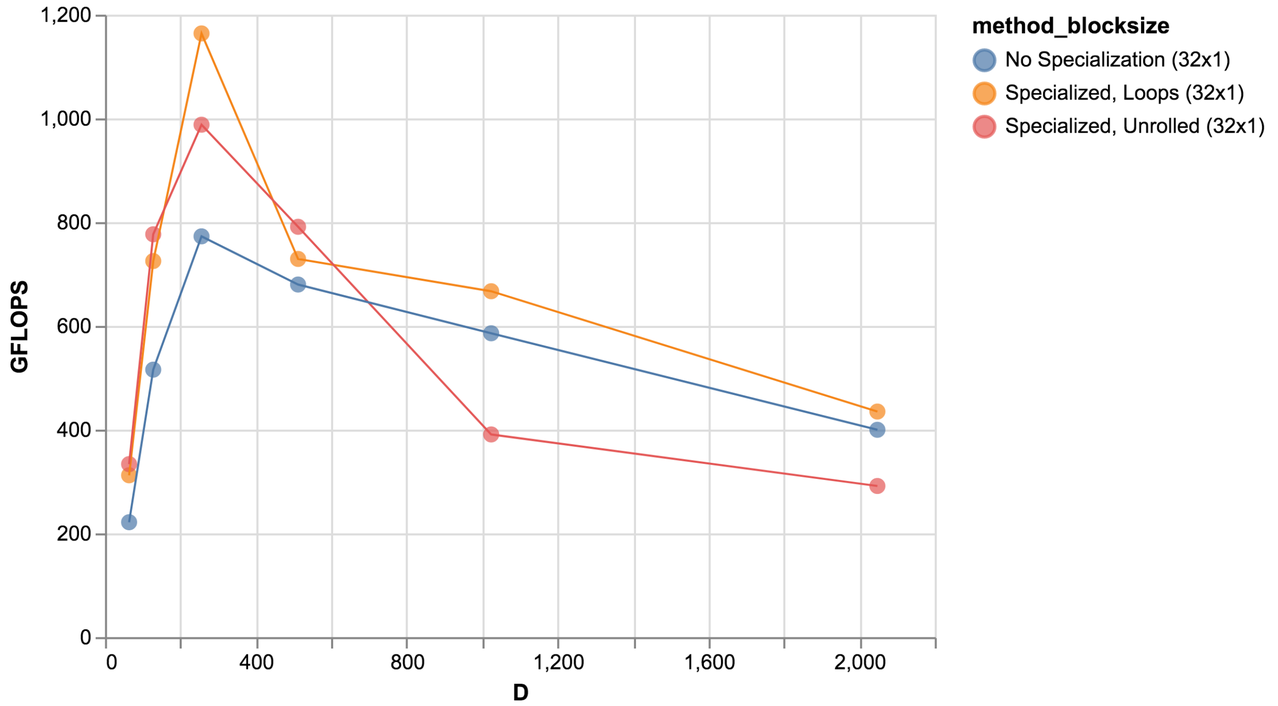



Sampling latency (SKL, #threads=1)
Time (µs)
120
110
100
90
80
70
60
50
40
30
20
10
0
Method
TVM (97% 16x1 sparse) (120µs)
TVM (97% 16x1 sparse, bf16) (100µs)
TVM (97% 16x1 sparse, bf16, exp-approx) (35µs)
TVM (97% 16x1 sparse, bf16, exp-approx, #threads=2) (26µs)
24kHz

Structured and Unstructured Sparsity

- Lots of ‘free’ wins from exploring sparsity in modern ML models
- Can often prune models to 80%+ sparsity(with retraining)
- Massive speedups combined with specialized code-generation techniques (TVM, Xbyak, etc)
- Interesting new tradeoffs
  - how `const` are parameters?
  - structure specialization trades off `icache`/ `dcache`
- also available today in FBGEMM

```
def tvm_bsr_codegen(ins, outs):
    ib = tvm.ir_builder.create()
    load_count = 0

    for nb in range(0, N, R):
        acc = tvm.const(0, vecty)
        for jj in range(indptr[nb // R], indptr[nb // R + 1]):
            j = indices[jj]
            x_k = ins[0].vload([0, C * j], 'float32').astype(vecty)
            w_nk_vec = ins[1].vload([jj, 0, 0], vecty)
            acc += x_k * w_nk_vec
            load_count += 1

        ib.emit(outs[0].vstore([0, nb], acc))
    return ib.get()
```



PyTorch and TVM

- Lots of opportunity in PyTorch
- Graph optimization
  - Existing fusion infrastructure fairly limited (CUDA-only, injective-only)
- Kernel synthesis
  - Dynamic shapes, stride specialization
- Impedance mismatch with PyTorch JIT IR and Relay IR
- Watch this space :)

```
import torch
import torch_tvm

torch_tvm.enable()

# The following function will be compiled with TVM
@torch.jit.script
def my_func(a, b, c):
    return a * b + c
```

```
input = torch.randn(N, D).type(torch.float32)
index = torch.randint(N, size=(K,)).type(torch.int64)
index = index[:, None].expand(index.shape[0], input.shape[1])
out = torch.gather(input, 0, index)

input_ph = torch_placeholder(input, dynamic=True)
index_ph = torch_placeholder(index, dynamic=True)

out_ph = tvm_gather(input=input_ph, dim=0, index=index_ph)
```

```
func = tvm.build(s, [input_ph, index_ph, out_ph],
                 binds={
                     out_ph:
                         tvm.decl_buffer(
                             out_ph.shape,
                             out_ph.dtype,
                             strides=[tvm.var(), 1]),
                     input_ph:
                         tvm.decl_buffer(
                             input_ph.shape,
                             input_ph.dtype,
                             strides=[tvm.var(), 1]),
                     index_ph:
                         tvm.decl_buffer(
                             index_ph.shape,
                             index_ph.dtype,
                             strides=[tvm.var(), 0])},
                 )
```

Big thanks to the community