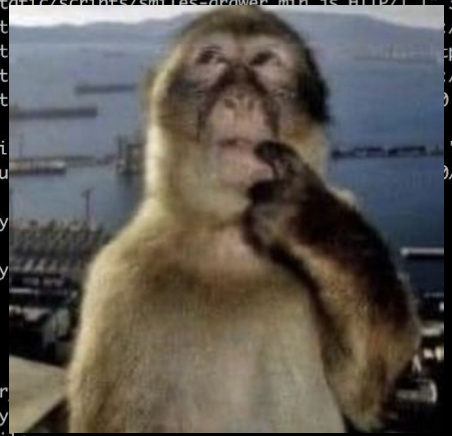
PyTorch perf optimization

Yahya Jabary

I planned a PyTorch perf optimization talk,
then realized I don't know how computers work.

I planned a PyTorch perf optimization talk,
then realized I don't know how computers work.
> Send help.

```
(flame) nbosc-ml:flame nbosc$ python predict-ws.py
[05/Jun/2018:09:24:49] ENGINE Listening for SIGTERM.
[05/Jun/2018:09:24:49] ENGINE Listening for SIGHUP.
[05/Jun/2018:09:24:49] ENGINE Listening for SIGUSR1.
[05/Jun/2018:09:24:49] ENGINE Bus STARTING
[05/Jun/2018:09:24:49] ENGINE Started monitor thread 'Autoreloader'.
[05/Jun/2018:09:24:49] ENGINE Serving on http://0.0.0.0:8080
[05/Jun/2018:09:24:49] ENGINE Bus STARTED
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET / HTTP/1.1" 200 3162 "" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko/20100101 Firefox/60.0"
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /static/scripts/jquery-3.3.1.min.js HTTP/1.1" 304 - "http://localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /static/css/bootstrap.min.css HTTP/1.1" 304 - "http://localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Ge
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /static/scripts/smiles-drawer.min.js HTTP/1.1" 304 - "http://localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:6
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /st                                    //localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko/2
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /st                                    p://localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /st                                    //localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko/2
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /st                                    ) 11068 "http://localhost:8080/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:
"
127.0.0.1 - - [05/Jun/2018:09:24:56] "GET /di                                    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko/20100101 Firefox/60.
127.0.0.1 - - [05/Jun/2018:09:25:02] "POST /u                                    0/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko/20100101 Firefox/
Segmentation fault: 11
(flame) nbosc-ml:flame nbosc$ python flame.py
flame :  True new endpoint MyModel created
(flame) nbosc-ml:flame nbosc$ python flame.py
/Users/nbosc/flame/flame//models
DIPL2
MyModel
CACO2
flame :  True 3 models found in the repositor
(flame) nbosc-ml:flame nbosc$ python flame.py
flame build :  False wrong training series file
(flame) nbosc-ml:flame nbosc$ python flame.py -c build -e MyModel -f ../mols/
caco2.sdf     dipl2.sdf     minicaco.sdf
(flame) nbosc-ml:flame nbosc$ python flame.py -c build -e MyModel -f ../mols/minicaco.sdf
Segmentation fault: 11
(flame) nbosc-ml:flame nbosc$ python flame.py -c predict -e MyModel -v 0 -f ../mols/minicaco.sdf
Segmentation fault: 11
(flame) nbosc-ml:flame nbosc$
```



What is happening

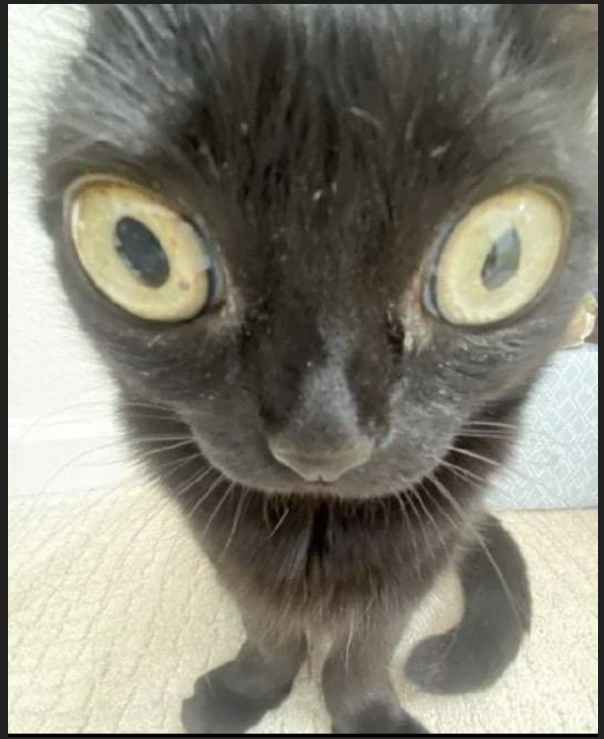
Python Layer (CPython Interpreter)
- Executes the high-level PyTorch code
- Performs basic Python optimizations like constant folding
- Handles memory management through garbage collection
- Provides bytecode optimization

PyTorch Frontend Layer
- Builds computational graphs
- Performs automatic differentiation (autograd)
- Provides JIT compilation through `torch.compile`
- Handles operator fusion for pointwise operations
- Manages memory pinning and asynchronous GPU transfers

PyTorch Backend Layer
- Executes kernel operations
- Performs automatic kernel fusion through `TorchInductor`
- Handles OpenMP parallelization for CPU operations
- Manages memory allocation and deallocation
- Provides backend-specific optimizations through delegates

Hardware Abstraction Layer
- GPU drivers handle device-specific optimizations
- CPU instruction set optimizations (SIMD, AVX)
- Memory hierarchy management (cache, DRAM)
- Hardware-specific threading and parallelization

Hardware Layer
- Physical execution of computations
- Direct memory access (DMA) transfers
- Cache coherency protocols
- Hardware-level parallelization



Levels of Abstraction

Python Layer (CPython Interpreter)
- Executes the high-level PyTorch code
- Performs basic Python optimizations like constant folding
- Handles memory management through garbage collection
- Provides bytecode optimization

Levels of Abstraction

PyTorch Frontend Layer
- Builds computational graphs
- Performs automatic differentiation (autograd)
- Provides JIT compilation through `torch.compile`
- Handles operator fusion for pointwise operations
- Manages memory pinning and asynchronous GPU transfers

Levels of Abstraction

PyTorch Backend Layer
- Executes kernel operations
- Performs automatic kernel fusion through `TorchInductor`
- Handles OpenMP parallelization for CPU operations
- Manages memory allocation and deallocation
- Provides backend-specific optimizations through delegates

Levels of Abstraction

Hardware Abstraction Layer
- GPU drivers handle device-specific optimizations
- CPU instruction set optimizations (SIMD, AVX)
- Memory hierarchy management (cache, DRAM)
- Hardware-specific threading and parallelization

Levels of Abstraction

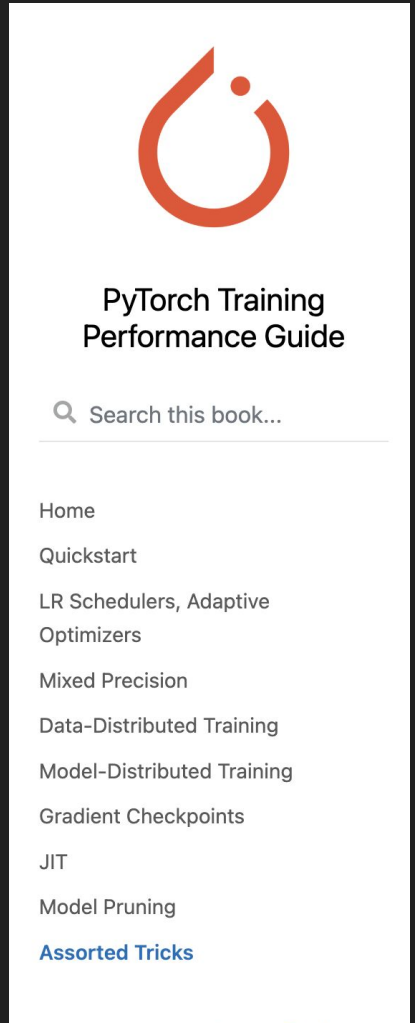
Hardware Layer
- Physical execution of computations
- Direct memory access (DMA) transfers
- Cache coherency protocols
- Hardware-level parallelization

PyTorch Performance Tuning

- LR Schedulers, Adaptive Optimizers
- Mixed Precision
- Data-Distributed Training
- Model-Distributed Training
- Gradient Checkpoints
- JIT
- Model Pruning
- ...

https://residentmario.github.io/pytorch-training-performance-guide/intro.html

https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html



“you cannot effectively optimize a system which you do not understand”

https://drewdevault.com/2020/02/21/Thoughts-on-performance.html

Thoughts on performance & optimization

February 21, 2020 on Drew DeVault's blog

The idea that programmers ought to or ought not to be called “software engineers” is a contentious one. How you approach optimization and performance is one metric which can definitely push my evaluation of a developer towards the engineering side. Unfortunately, I think that a huge number of software developers today, even senior ones, are approaching this problem poorly.

Centrally, I believe that you cannot effectively optimize a system which you do not understand. Say, for example, that you’re searching for a member of a linked list, which is an O(n) operation. You know this can be improved by switching from a linked list to a sorted array and using a binary search. So, you spend a few hours refactoring, commit the implementation, and… the application is no faster. What you failed to consider is that the lists are populated from data received over the network, whose latency and bandwidth constraints make the process much slower than any difference made by the kind of list you’re using. If you’re not optimizing your bottleneck, then you’re wasting your time.

This example seems fairly obvious, and I’m sure you, our esteemed reader, would not have made this mistake. In practice, however, the situation is usually more subtle. Thinking about your code really hard, making assumptions, and then acting on them is not the most effective way to make performance improvements. Instead, we apply the scientific method: we think really hard, *form a hypothesis*, make predictions, test them, and then apply our conclusions.

To implement this process, we need to describe our performance in factual terms. All software requires a certain amount of resources — CPU time, RAM, disk space, network utilization, and so on. These can also be described over time, and evolve as the program takes on different kinds of work. For example, we could model our program’s memory use as bytes allocated over time, and perhaps we can correlate this with different stages of work — “when the program starts task C, the rate of memory allocation increases by 5MiB per second”. We identify bottlenecks — “this program’s primary bottleneck is disk I/O”. When we hit performance problems, then we know that we need to upgrade to SSDs, or predict what reads will be needed later and prep them in advance, cache data in RAM, etc.

Good optimizations are based on factual evidence that the program is not operating within its constraints in certain respects, then improving on those particular problems. You should always conduct this analysis before trying to solve your problems. I generally recommend conducting this analysis in advance, so that you can predict performance issues before they occur, and plan for them accordingly. For example, if you know that your disk utilization grows by 2 GiB per day, and you’re on a 500 GiB hard drive, you’ve got about 8 months to plan your next upgrade, and you shouldn’t be surprised by an ENOSPC when the time comes.

For CPU bound tasks, this is also where a general understanding of the performance characteristics of various data structures and algorithms is useful. When you know you’re working on something which *will become* the application’s bottleneck, you would be wise to consider algorithms which can be implemented efficiently. However, it’s equally important to re-prioritize performance when you’re not working on your bottlenecks, and instead consider factors like simplicity and conciseness more seriously.

Much of this will probably seem obvious to many readers. Even so, I think the general wisdom described here is institutional, so it’s worth writing down. I also want to call out some specific behaviors that I see in software today which I think don’t take this well enough into account.

I opened by stating that I believe that you cannot effectively optimize a system which you do not understand. There are two groups of people I want to speak to with this in mind: library authors (especially the standard library), and application programmers. There are some feelings among library authors that libraries should be fairly opaque, and present high-level abstractions over their particular choices of algorithms, data structures, and so on. I think this represents a fundamental lack of trust with the programmer downstream. Rather than write idiot-proof abstractions, I think it’s better

Conclusion

Yes - There are many great PyTorch optimization feats. / libs.

No – I don’t recommend any of them,
unless you have time for extensive profiling.

There is no one-size-fits-all solution to performance.
There are a lot of unknown-unknowns.

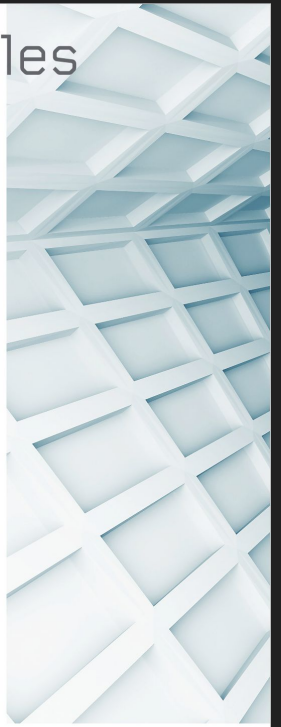
Benchmark in reproducible environments.
Minimize noise.

DOI:10.1145/3213770

Performance measurements often go wrong, reporting surface-level results that are more marketing than science.

BY JOHN OUSTERHOUT

Always Measure One Level Deeper



PERFORMANCE MEASUREMENT IS one of the most important parts of software development. In academic research a thorough performance evaluation is considered essential for many publications to prove the value of a new idea. In industry, performance evaluation is necessary to maintain a high level of performance across the lifetime of a product. For example, cloud services promise to maintain particular performance levels; service providers must thus be able to detect when performance drops below acceptable levels and quickly identify and fix the problem.

A good performance evaluation provides a deep understanding of a system's behavior, quantifying not only the overall behavior but also its internal mechanisms and policies. It explains why a system behaves the way it does, what limits that behavior, and what problems must be addressed in order to

key insights

- Performance measurement is less straightforward than it might seem; it is easy to believe results that are incorrect or misleading and overlook important system behaviors.
- The key to good performance measurement is to make many more measurements besides the ones you think will be important; it is crucial to understand not just the system's performance but also why it performs that way.
- Performance measurement done well results in new discoveries about the system being measured and new intuition about system behavior for the person doing the measuring.

Thanks for listening!

Demo: Profiling GIL-Free Python

https://github.com/sueszli/nogil/