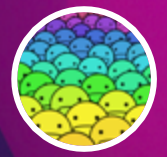
AI编译器-系列之PyTorch

Dispatch 机制



ZOMI

Talk Overview

1. **PyTroch 2.0 新特性**
   - 2.0 新特性回顾
   - PyTorch 2.0 安装与新特性使用
   - PyTorch 2.0 对厂商的启发和思考
2. **TorchDynamo 解读**
   - TorchDynamo 特性
   - TorchDynamo 实现方案
3. **AOTAutograd 解读**
   - AOTAutograd 效果
   - AOTAutograd 实现方案
   - Torch dispatch 机制和原理
4. **TorchInductor 新特性**
   - Triton 使用解读
   - Triton 深度剖析

Talk Overview

1. Torch dispatch 机制和原理
   - 什么是 Dispatch
   - 为什么需要 Dispatch
   - 注册和分发机制
   - Dispatch key 的表示和计算
   - Dispatch table 注册

什么是 Dispatch

dispatch 英[dɪˈspætʃ] 美[dɪˈspætʃ]

vt. 派遣; 调遣; 派出; 发出，发送(邮件、包裹、信息); 迅速处理; 迅速办妥; 迅速完成; 杀死;

n. 派遣; 调遣; 发送; (军事人员或政府官员之间的)急件，快信; (驻外国记者发给报刊的)新闻报道，电讯;

[例句] The Italian government was preparing to dispatch 4,000 soldiers to search the island.

意大利政府正准备派遣4,000名士兵搜索该岛。

[其他] 第三人称单数：dispatches 复数：dispatches 现在分词：dispatching 过去式：dispatched 过去分词：dispatched

进行更多翻译 扫码下载百度翻译APP

fanyi.baidu.com

什么是 Dispatch

为什么需要 Dispatch

```
if inputs.content == GPU:
    then xxxx
elif inputs.content == CPU:
    then xxxx
```

- **device**：针对一个算子，有 CPU、GPU、NPU、TPU、FPGA 等不同设备
- **layout**：张量有普通张量、稀疏张量，不同张量有不同布局
- **dtype**：张量数据类型有float64、float32、float16、int8，甚至有 bf16、hf32 等不同数据类型

为什么需要 Dispatch

- **device**：针对一个算子，有 CPU、GPU、NPU、TPU、FPGA 等不同设备
- **layout**：张量有普通张量、稀疏张量，不同张量有不同布局
- **dtype**：张量数据类型有float64、float32、float16、int8，甚至有 bf16、hf32 等不同数据类型

- 需要一个 Dispatcher，让它来统一管理分派工作

注册和分发

- （常见架构）注册 + 分发 -> （设计模式）注册器模式 + 工厂模式

注册和分发

```
2   def add(z1, z2):
3       if z1.tag != z2.tag:
4           raise ValueError
5
6       if z1.tag == "rectangular":
7           return rectangular_add(z1, z2)
8       elif z1.tag == "polar":
9           return polar_add(z1, z2)
10      else:
11          raise ValueError
```

注册和分发

- 采用注册 + 分发的机制，实际上会存在一张支持 register 和 get 操作的虚拟表格 Vtable

Key	Value
rectangular	rectangular_add()
polar	polar_add()
…	…

注册和分发

- 每次有新的方法时，通过 register 注册到表里，然后我们的接口通过 get 取到对应的实现函数，此时接口就不再是 manager 了，它只需要从表中取出对应的函数并执行，而向表中注册的事情，交由编写具体的新方法的人来负责：

```
14    def add(z1, z2):
15        if z1.tag != z2.tag:
16            raise ValueError
17
18        method = get(z1.tag)
19        return method(z1, z2)
```

Virtual tables in C++

1. Dispatch tables are allocated per operator, whereas vtables are allocated per class.
2. the computation of dispatch key considers all arguments to the operator (multiple dispatch) as well as thread-local state (TLS).
3. Finally, the dispatcher supports boxing and unboxing as part of the calling convention for operators.

Virtual tables in C++

c++ vatable	PyTorch vtable	解释
每个类有一个 vtable	每个 op 有一个 vtable	pytorch 中，扩展一个已有的 op，只需要提供一个新的 vtable
只有 *this 指针重要	不仅考虑 tensor，还有其他的 meta 信息	
	支持 boxing 和 unboxing	

Dispatch Key

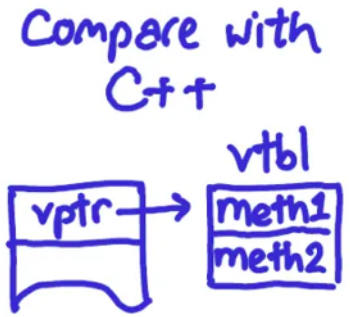
- 当一个执行一个如 *torch::add 的 operator* 时，dispatcher 首先会找到这个 operator 对应的 *dispatch key*，然后再根据这个 dispatch key 去找到对应的 kernel 函数。

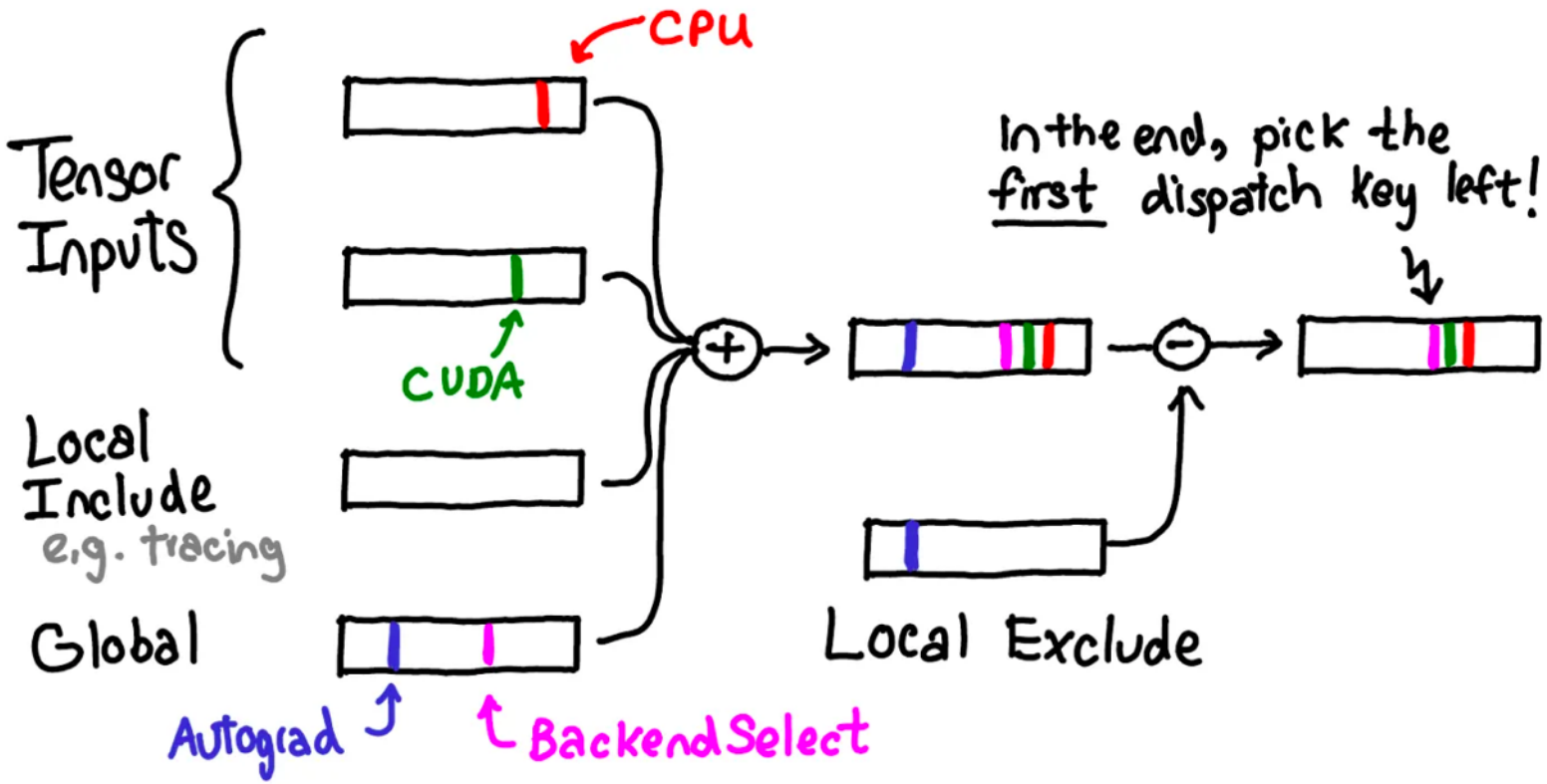
Dispatch Key

- how exactly do we compute the dispatch key which we use to index into the dispatch table?

Dispatcher: What should I call?

DispatchKeySet (a bitset)

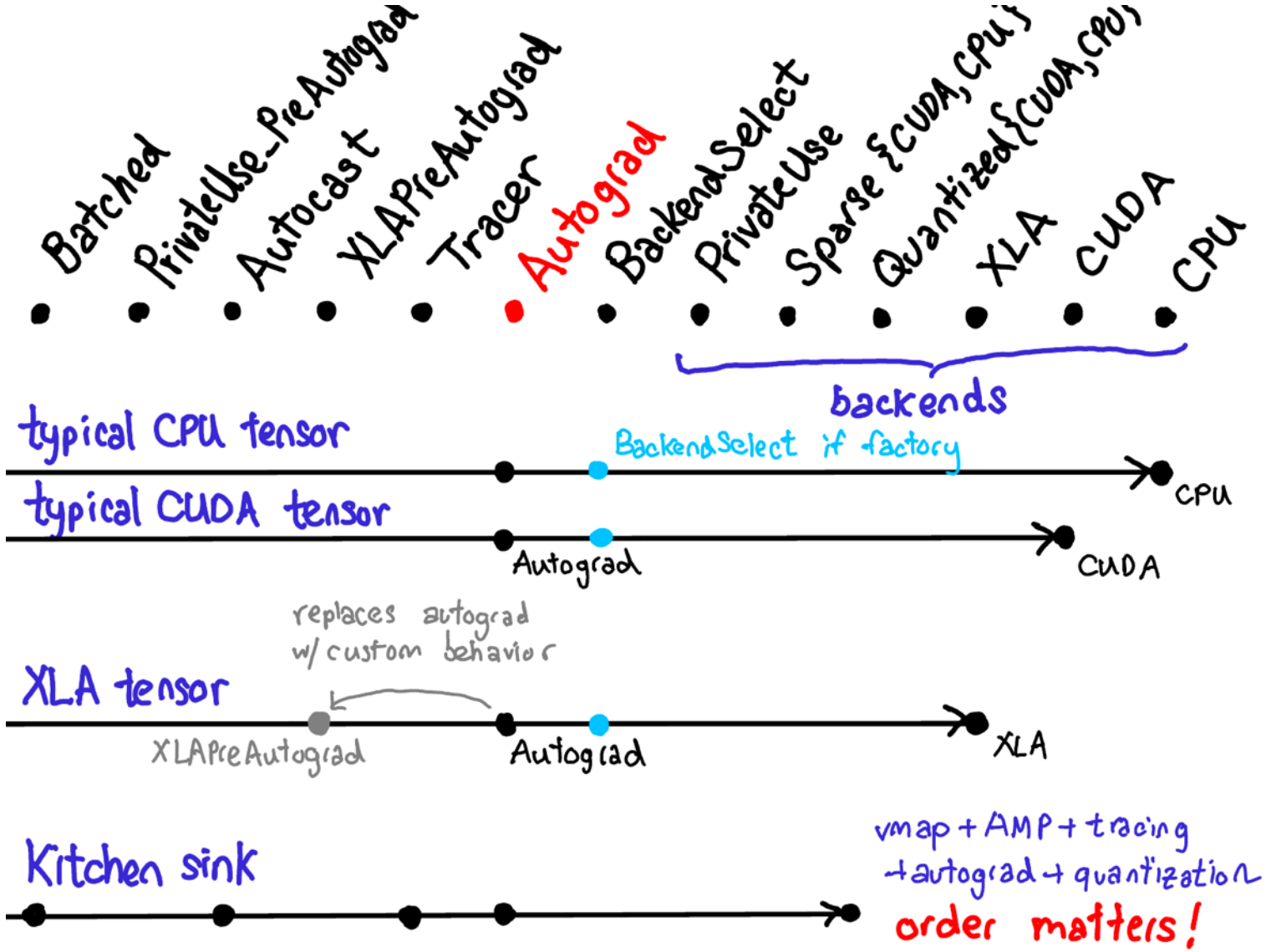




Dispatch Key

- Dispatcher 并不是只针对backends环境来做调遣，换句话说，dispatch key 并不总是对应一种实打实的backends硬件环境(cpu, gpu)，它对应的是一种更抽象的概念。

- diapatch key 的计算是通过一个 dispatch key set 的结构来实现的， dispatch key set 可以理解为一个 64bit 的数组，每一个 bit 都代表了一个 dispatch key，然后从左到右有优先级关系，同样一个算子，可能有针对不同 dispatch key 的实现，针对这些散落在 Pytorch各处的注册实现，然后也可能会将某些 key 排除的情况，这个数组通过最终将关于同一个算子的所有可用实现都结合到一起，调用其中优先级最高的 dispatch key 对应的 kernel 实现。



函数指针是如何进入virtual table的?

- 通过registeration api实现的： https://pytorch.org/tutorials/advanced/dispatcher.html

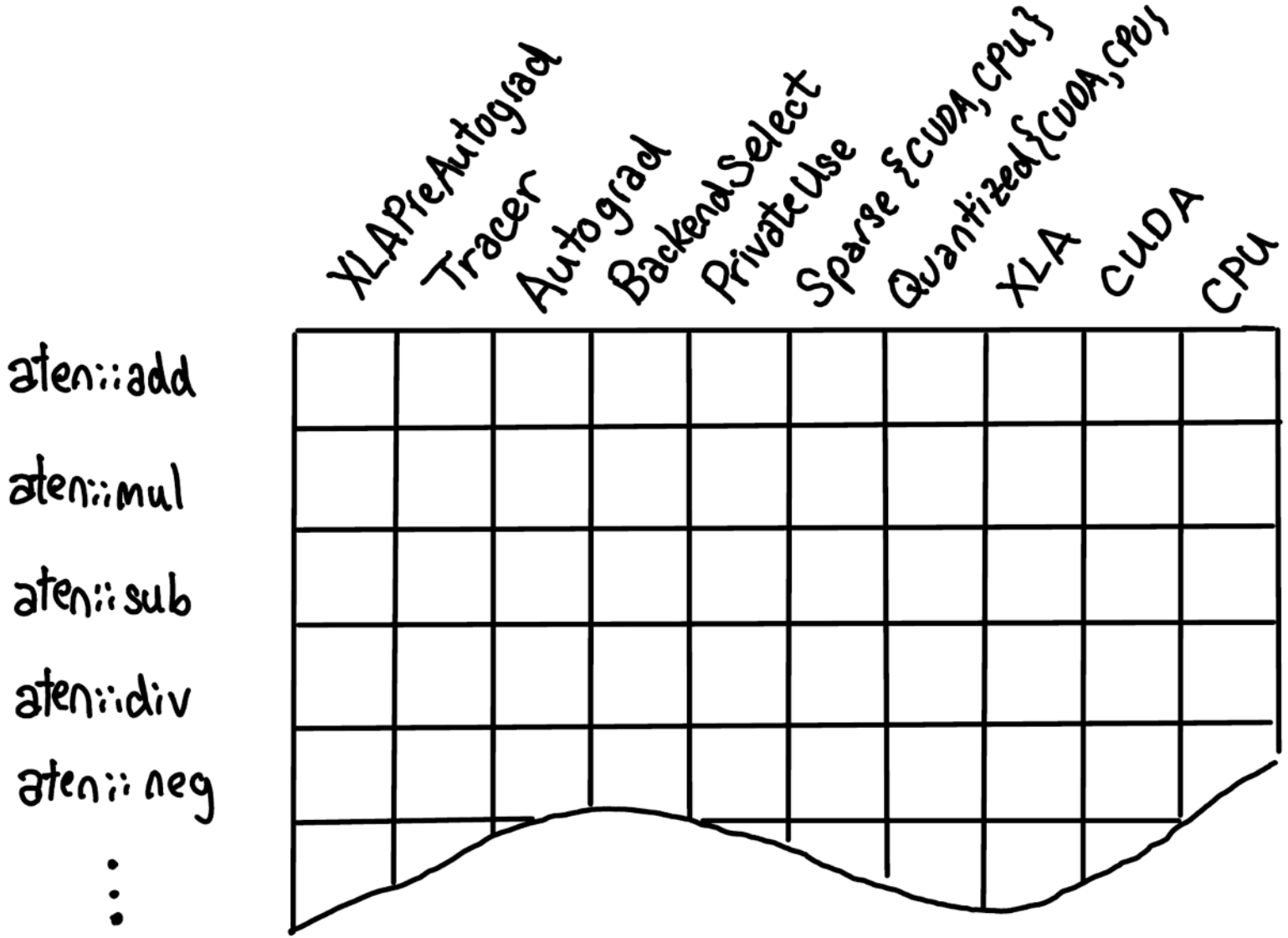
- 与registeration api交互方式：

1. 定义schema(m.def)
2. 实现register(m.impl)
3. fallback(m.fallback)

算子注册

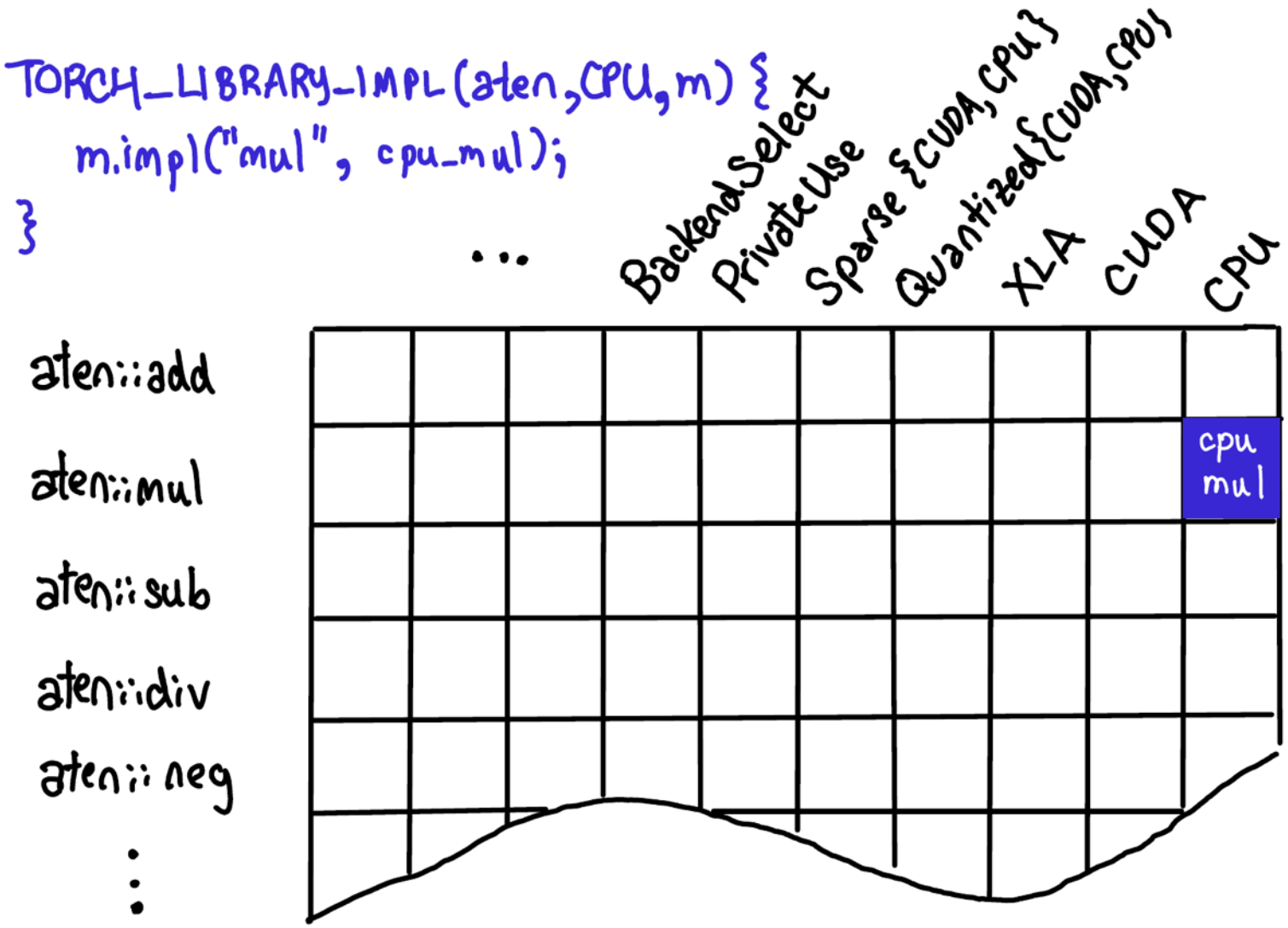
```
TORCH_LIBRARY(aten, m) {
    m.def("neg(Tensor self) -> Tensor");
}

TORCH_LIBRARY_IMPL(aten, CPU, m) {
    m.impl("neg", neg_cpu);
}

TORCH_LIBRARY_IMPL(_, Tracer, m) {
    m.fallback(generic_tracing);
}
```

dispatch key


XLAPreAutograd
Tracer
Autograd
BackendSelect
PrivateUse
Sparse {CUDA,CPU}
Quantized{CUDA,CPU}
XLA
CUDA
CPU
aten::add
aten::mul
aten::sub
aten::div
aten::neg

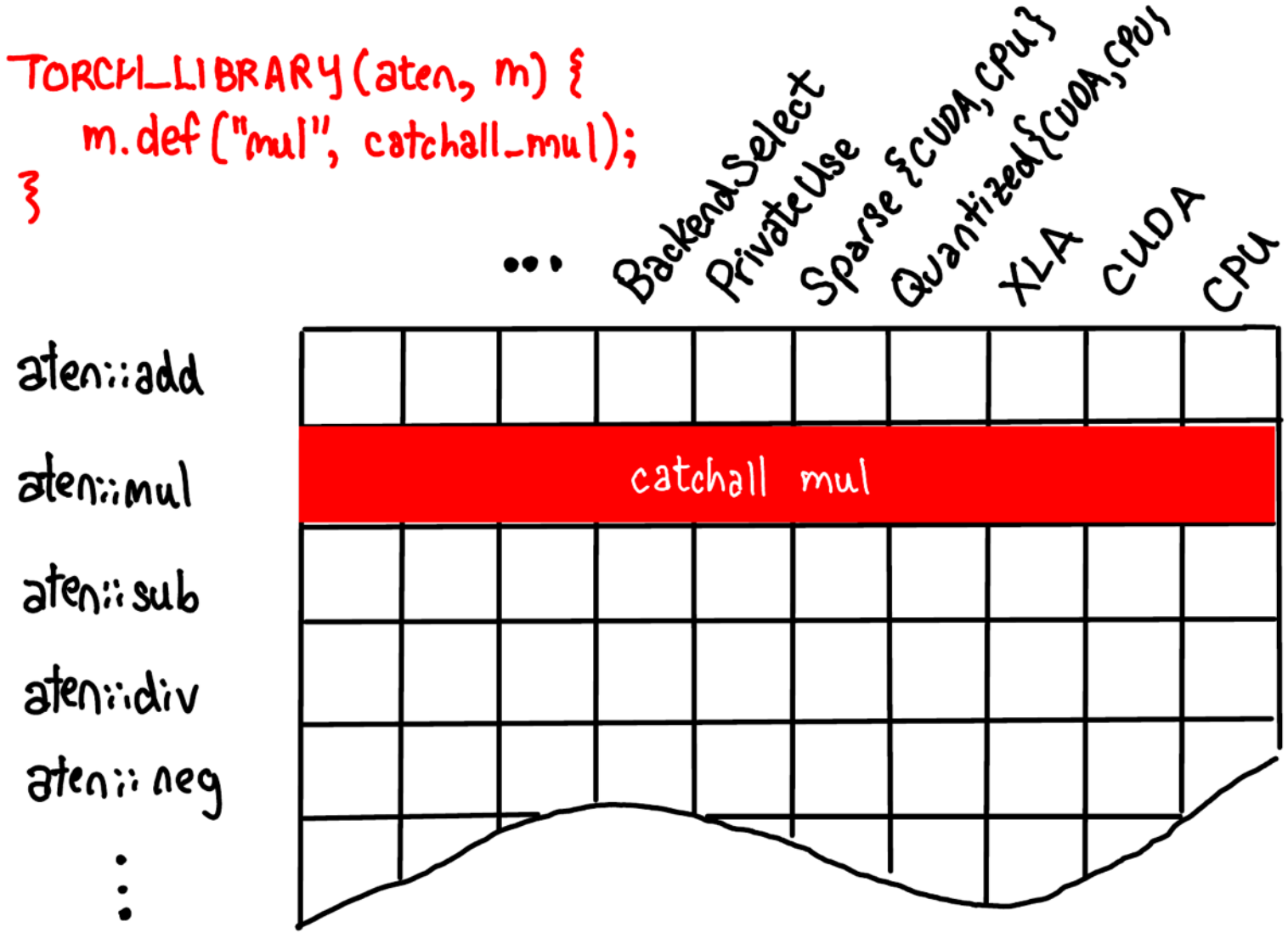
算子注册

```
1
2   TORCH_LIBRARY_IMPL(aten, CPU, m) {
3     m.impl("mul", cpu_mul);
4   }
5
```



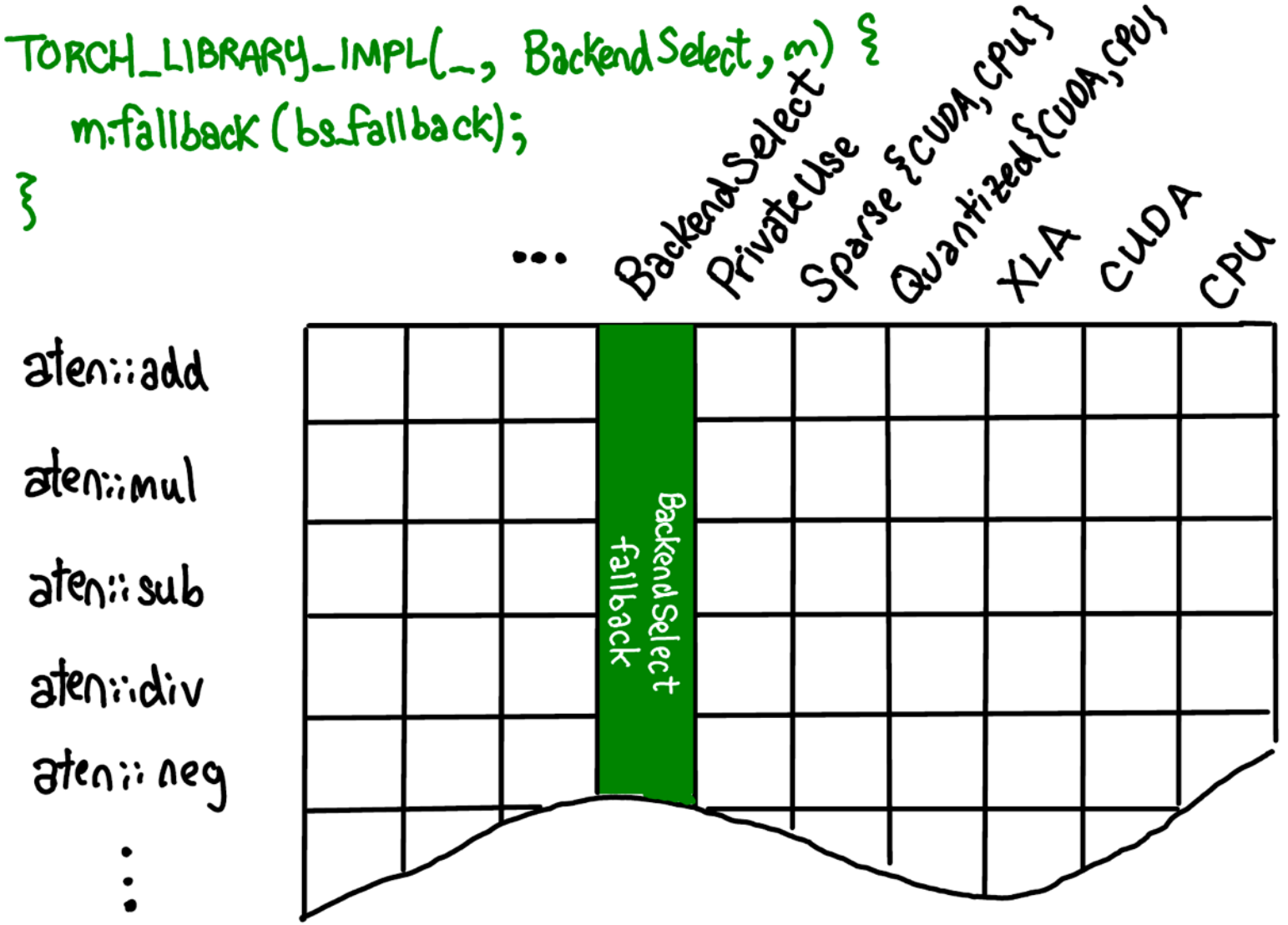
算子注册

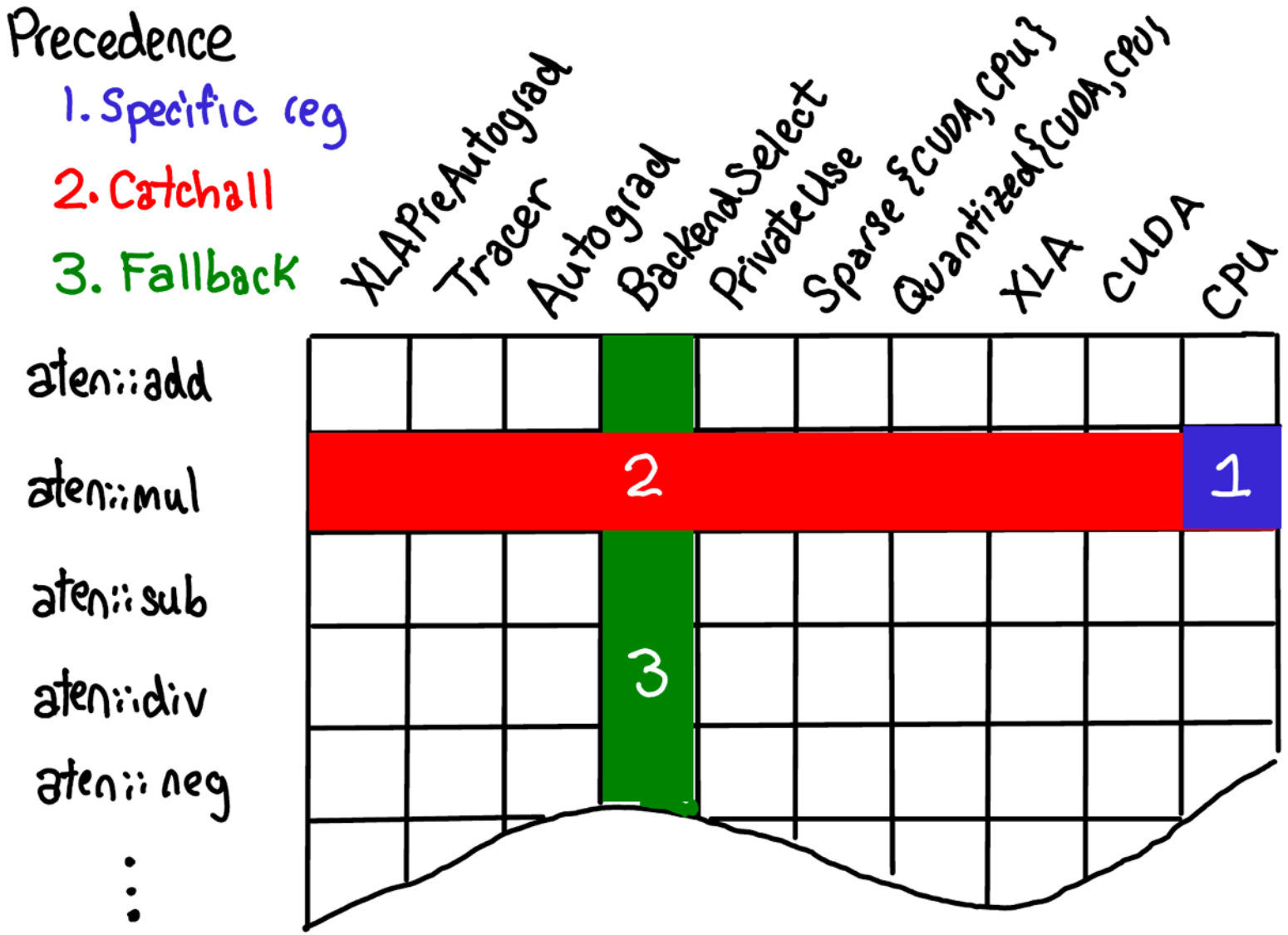
```
2   TORCH_LIBRARY(myops, m) {
3     m.def("myadd", catchall_mul);
4   }
```



算子注册

```
2   TORCH_LIBRARY_IMPL(_, AutocastCPU, m) {
3     m.fallback(makeFallthrough());
4   }
```





Users should be able to

- Compute efficient per-sample gradients 8
- Train 10 copies of Resnet18 efficiently at the same time 9
- Compute the FLOPS 31
- Parallelize it across arbitrary number of devices using arbitrary horizontal/vertical parallelism 56
- Represent the weights in some arbitrary more efficient representation
  - Low rank approximation
  - Butterfly sparsity 8
  - Factorized compression 1
  - Taichi sparsity data structure 4
  - 8-bit quantized format
  - Diagonal Tensors
  - Linear Operator Tensors
  - Arbitrary Einsum Tensors
- Pass MaskedTensors in as inputs
- Keep the tensors on SSD until they're needed, and then load them SSD when they're used 13
- Execute in some arbitrary lazy execution manner (like LazyTensor)
- Trace out the operations occurring in the graph (i.e. AOTAutograd)
- A billion more things we haven't thought of yet.

引用

1. https://pytorch.org/tutorials/advanced/dispatcher.html
2. https://pytorch.org/tutorials/advanced/extend_dispatcher.html
3. http://blog.ezyang.com/2020/09/lets-talk-about-the-pytorch-dispatcher/
4. https://developer.aliyun.com/article/911380
5. https://dev-discuss.pytorch.org/t/what-and-why-is-torch-dispatch/557
6. https://dev-discuss.pytorch.org/t/tracing-with-primitives-update-0/577
7. https://zhuanlan.zhihu.com/p/376495783
8. https://zhuanlan.zhihu.com/p/386876377
9. https://blog.csdn.net/Chris_zhangrx/article/details/119489853
10. https://www.cnblogs.com/ijpq/p/16292722.html