PyTorch CONFERENCE EUROPE 2026

Combo Kernels: Horizontal fusion optimization in torch.compile

Karthick Panner Selvam, Elias Ellison
Meta Superintelligence Labs

Simple torch.compile code

```
import torch

@torch.compile
def fn(t1, t2, t3, t4):
    return t1.sum(1), t2.max(1), t3.min(1), t4.sum(1)

t1 = torch.randn(1024, 512, device='cuda')
t2 = torch.randn(1024, 512, device='cuda')
t3 = torch.randn(1024, 512, device='cuda')
t4 = torch.randn(1024, 512, device='cuda')

out = fn(t1, t2, t3, t4)
```

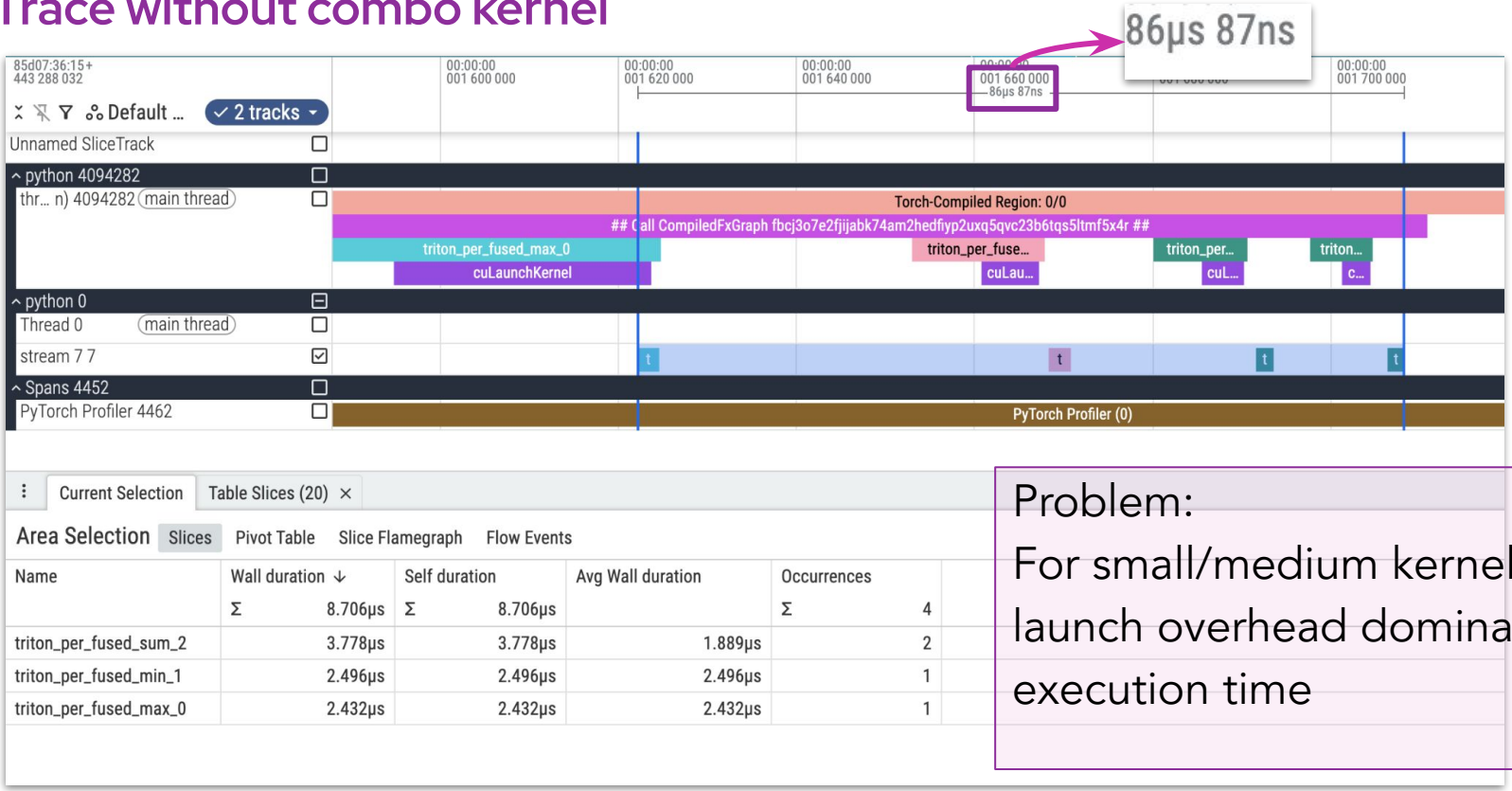
Simple torch.compile code

```
import torch

@torch.compile
def fn(t1, t2, t3, t4):
    return t1.sum(1), t2.max(1), t3.min(1), t4.sum(1)

t1 = torch.randn(1024, 512, device='cuda')
t2 = torch.randn(1024, 512, device='cuda')
t3 = torch.randn(1024, 512, device='cuda')
t4 = torch.randn(1024, 512, device='cuda')

out = fn(t1, t2, t3, t4)
```

It will launch four independent kernels

- triton_per_fused_max_0
- triton_per_fused_min_1
- triton_per_fused_sum_2
- triton_per_fused_sum_2

Trace without combo kernel



Problem:
For small/medium kernels, launch overhead dominates execution time

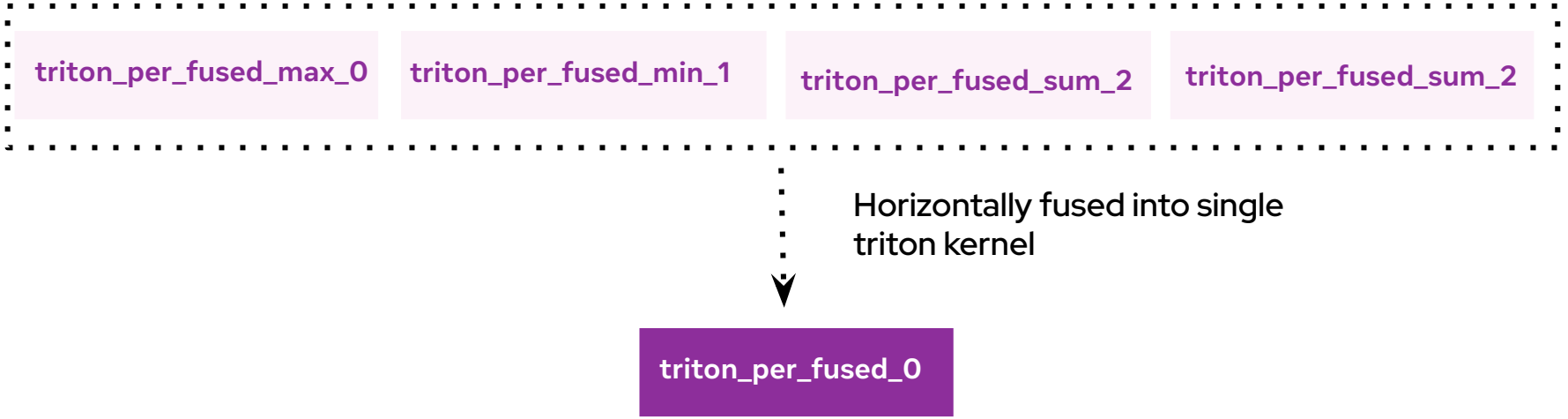
Solution

Merge N independent kernels into 1 Triton kernel that routes thread blocks to sub-kernels via `tl.program_id(0)`

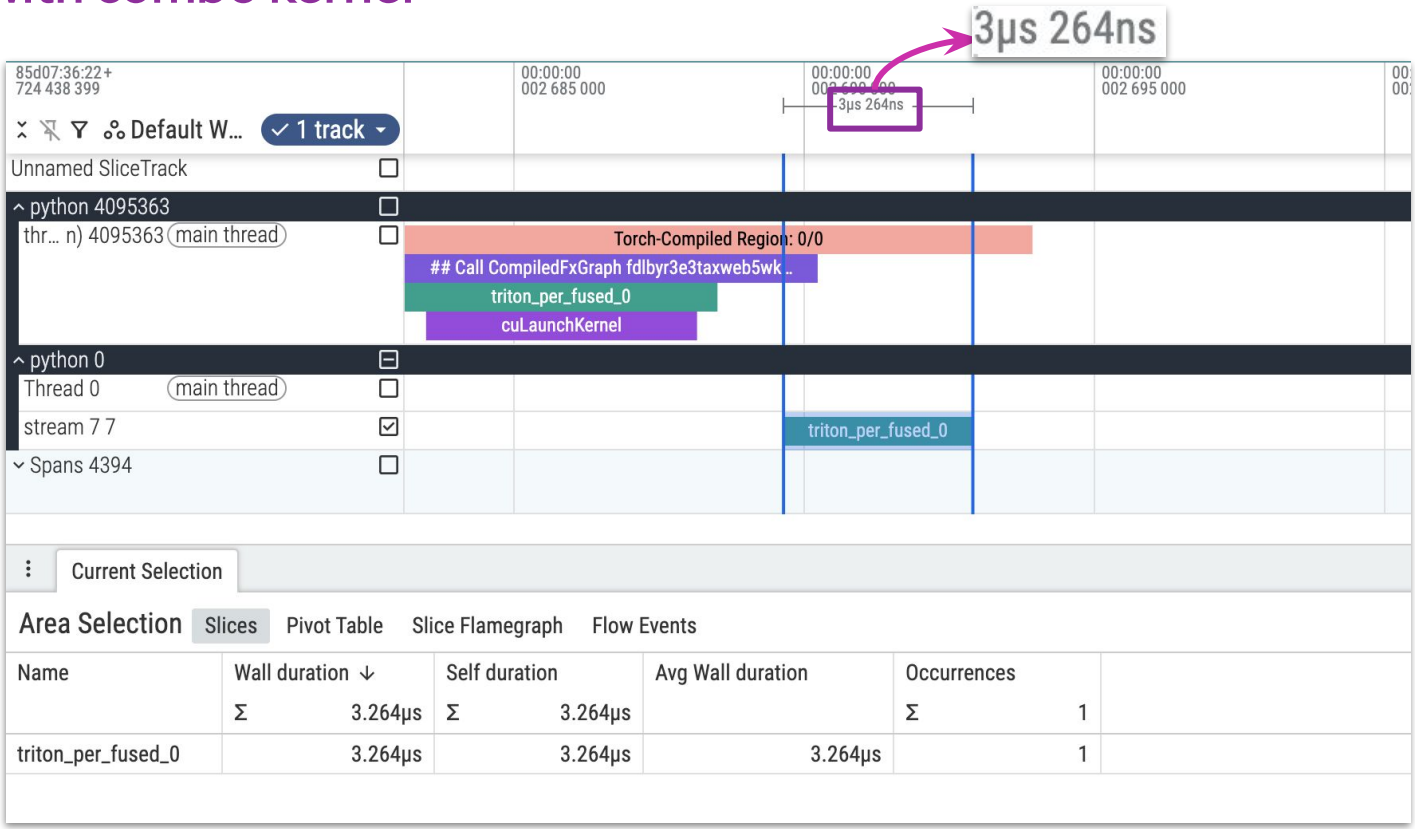
```
# Config: torch._inductor.config
combo_kernels = True                          # master toggle
combo_kernel_per_subkernel_blocks = True  # independent block sizes
benchmark_combo_kernel = True                 # only allow ones with perf gains
```

- Experimental feature, off by default
- Works with pointwise, reductions, and persistent reductions

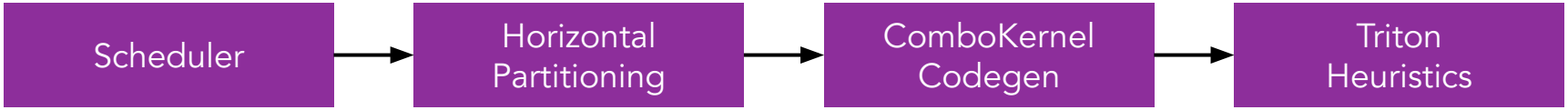
Simple torch.compile code with combo_kernel



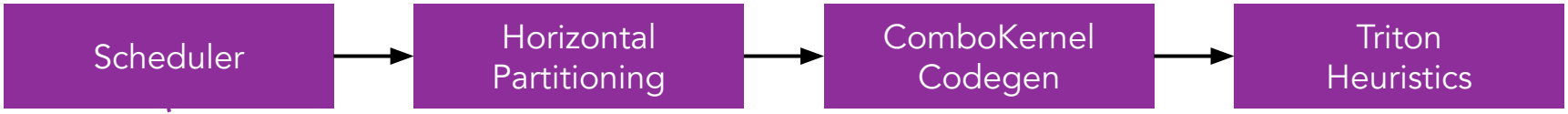
Trace - with combo kernel



Architecture Overview



Architecture Overview

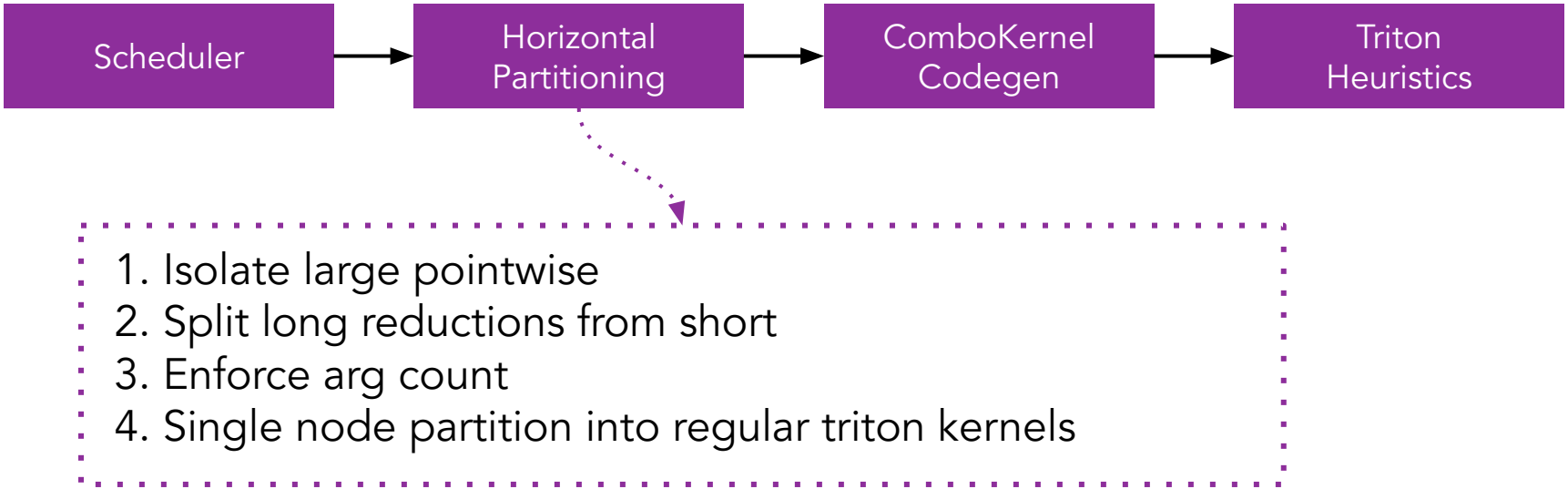


1. `topological_sort_nodes()` → find parallel nodes
2. `combinable_nodes()` → filter eligible nodes
3. `speedup_by_combo_kernel()` → benchmark gate default

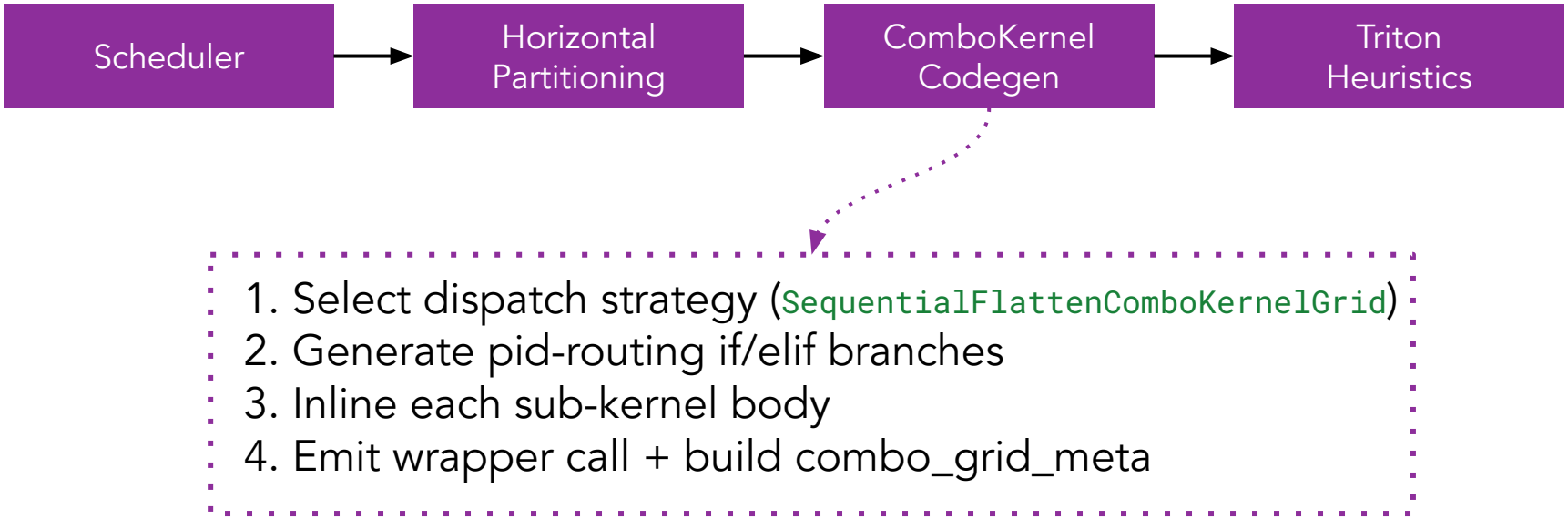
`benchmark_combo_kernel`=False (**optional**: ideal goal is to make combo kernels always profitable without needing the benchmarking pass.)

4. create `ForeachKernelSchedulerNode`

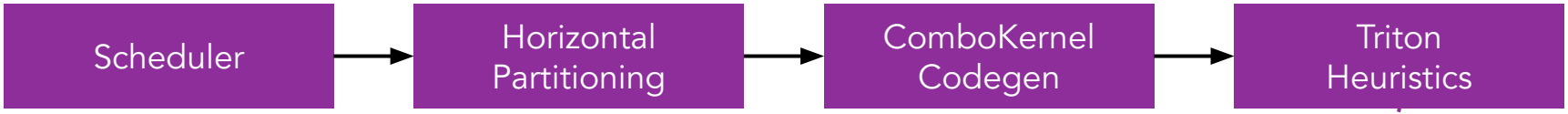
Architecture Overview



Architecture Overview



Architecture Overview



1. Each sub-kernel calls its own heuristic (pointwise / reduction / persistent_reduction)
2. Renames: XBLOCK→XBLOCK_0, XBLOCK_1, …
3 `ComboKernelGrid` computes launch grid from `combo_grid_meta`
4. `grid = (sum(x_blocks_i * y_blocks_i), 1, 1)`
5. Individual sub kernel autotuning + merging of configs

Generated triton code

```python
@triton_heuristics.persistent_reduction(
    size_hints={'x': 1024, 'r0_': 512},
    reduction_hint=ReductionHint.INNER,
    filename=__file__,
    triton_meta={'signature': {'in_ptr0': '*fp32', 'in_ptr1': '*fp32', 'in_ptr2': '*fp32', '
    inductor_meta={'grid_type': 'SequentialFlattenComboKernelGrid', 'combo_grid_meta': {'num
)
@triton.jit
def triton_per_fused_0(in_ptr0, .., XBLOCK_0, XBLOCK_1, XBLOCK_2, XBLOCK_3):
    pid = tl.program_id(0)
    x_blocks_0 = tl.cdiv(1024, XBLOCK_0)
    num_blocks_0 = x_blocks_0
    x_blocks_1 = tl.cdiv(1024, XBLOCK_1)
    num_blocks_1 = num_blocks_0 + x_blocks_1
    x_blocks_2 = tl.cdiv(1024, XBLOCK_2)
    num_blocks_2 = num_blocks_1 + x_blocks_2
    x_blocks_3 = tl.cdiv(1024, XBLOCK_3)
    num_blocks_3 = num_blocks_2 + x_blocks_3
    if pid < num_blocks_0:
        ... # sub-kernel 1
    elif pid < num_blocks_1:
        ... # sub-kernel 2
    elif pid < num_blocks_2:
        ... # sub-kernel 3
    elif pid < num_blocks_3:
        ... # sub-kernel 4
    else:
        pass
```

Combo Kernel
Grid type

Per-subkernel
config

pid-routing if/elif branches

Inline each
sub-kernel body

Full generated triton code

Benchmark results on inductor CI perf

On CI perf h100, combo kernels deliver geomean speedups of +7.3% for HuggingFace and +5.97% for TorchBench.

WIP / Active development

1. Fixing memory regression (only fuse if it doesn't increase peak memory)
2. Improving Coordinate descent tuning fo combo kernels
3. Fusing pointwise with reductions

pytorch.org

Get started!

github.com/pytorch

PyTorch CONFERENCE EUROPE 2026