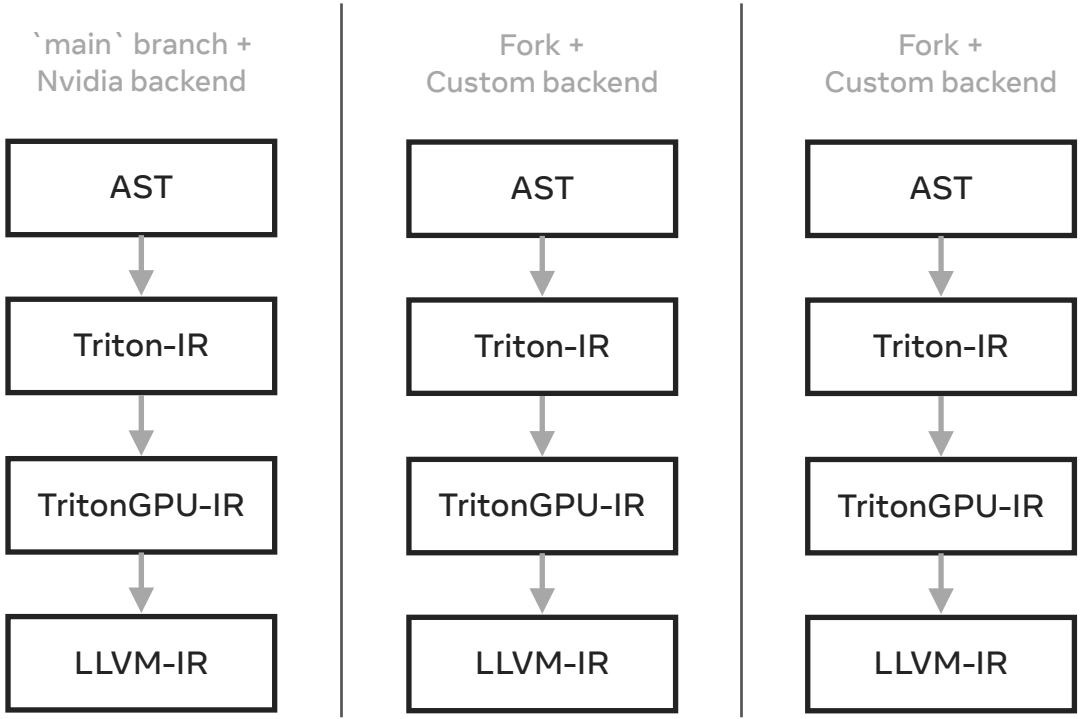
Triton:
Today & Beyond

Philippe Tillet

Retrospective

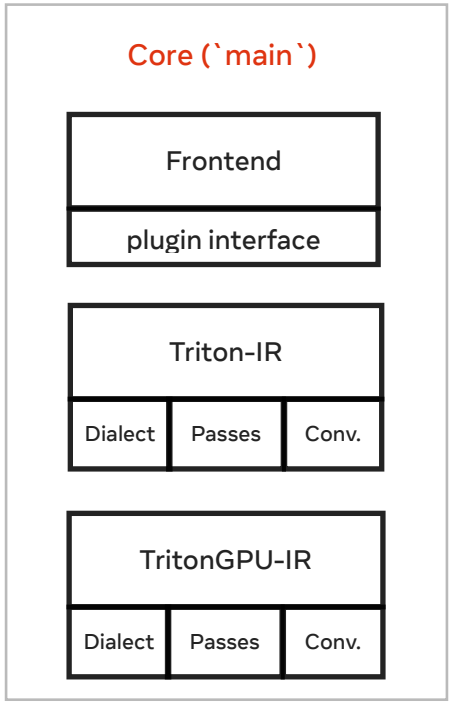
Portability

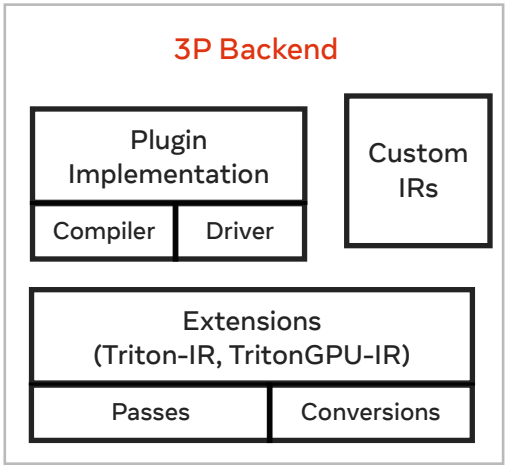
Triton codebase was not scalable a year ago



Portability

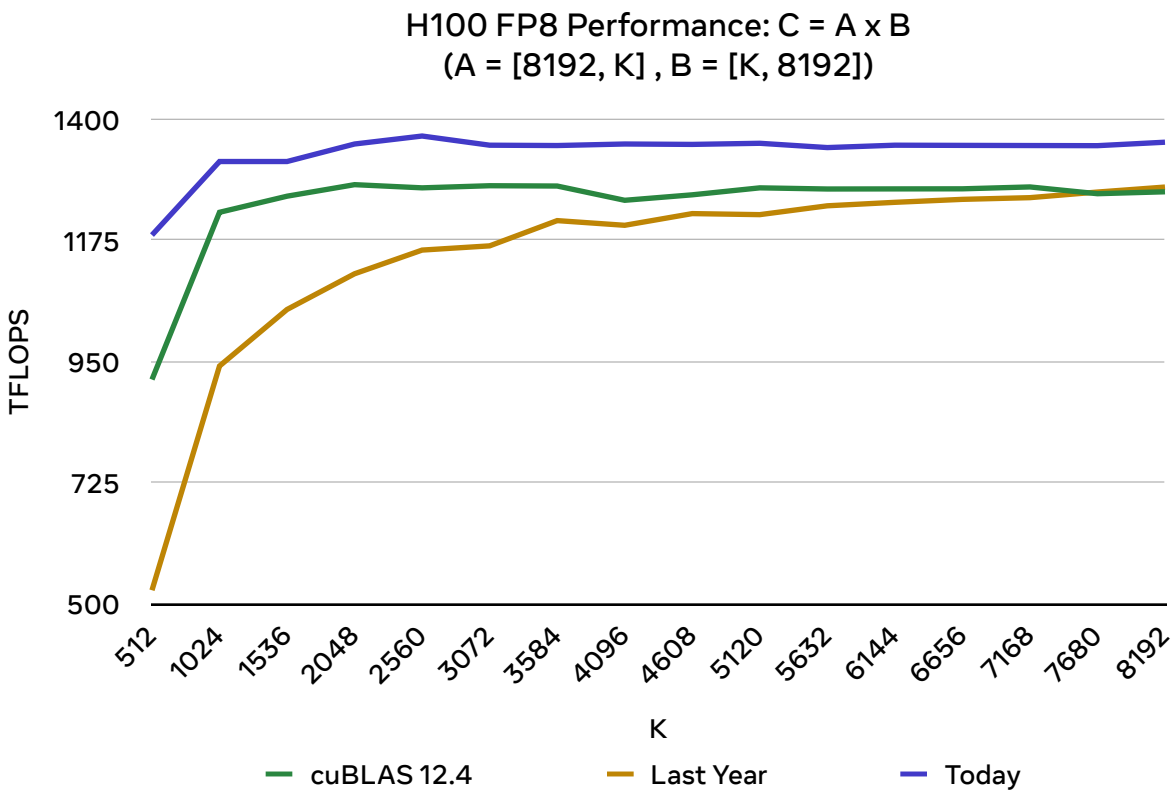
We are now sharing a core infrastructure





Caveat: Today, TritonGPU-IR is often forked as a custom IR

Performance



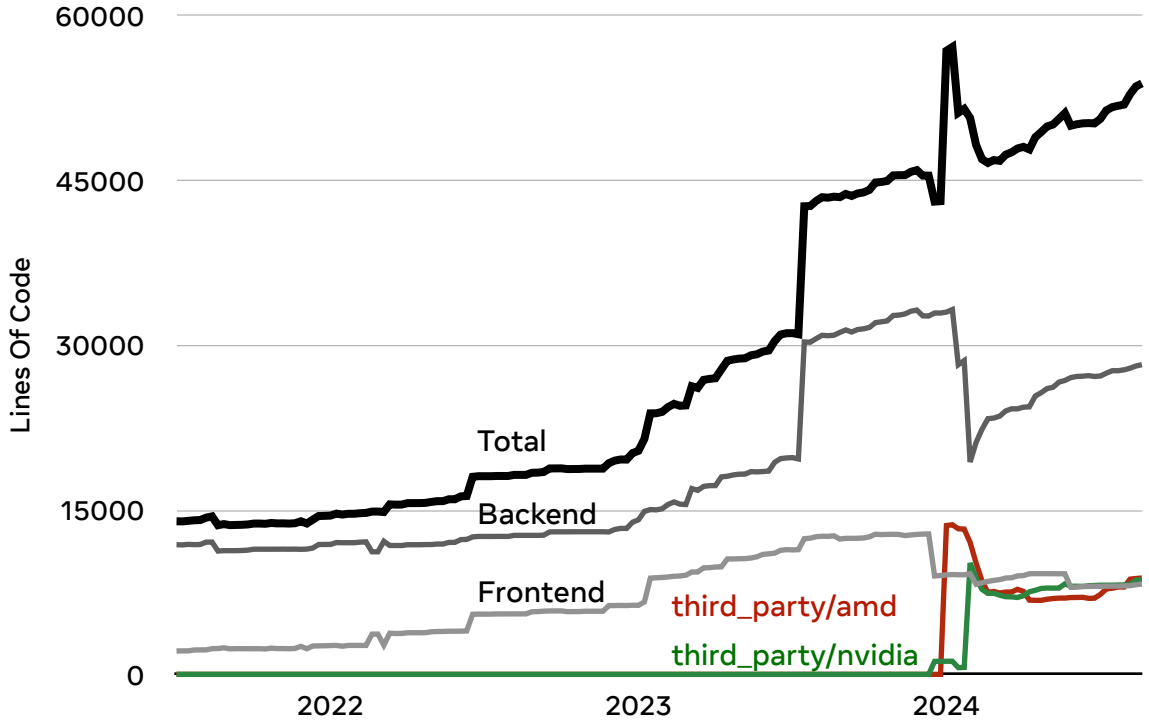
Tools

Developed major tools to improve user productivity

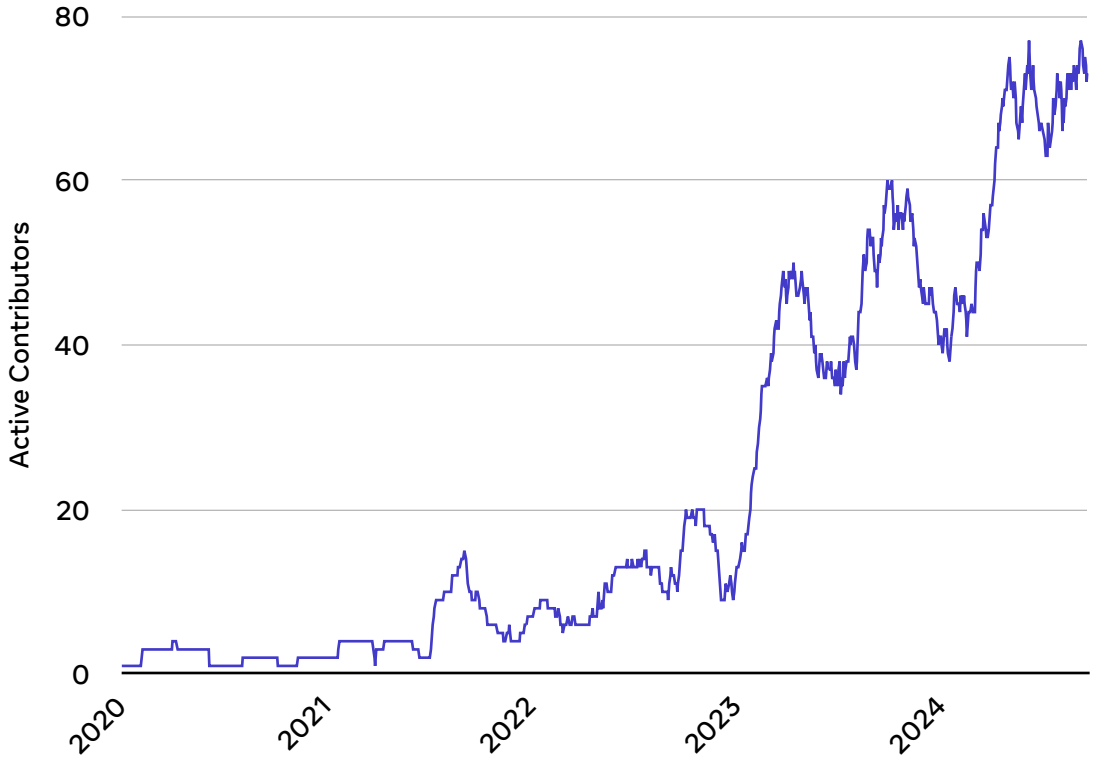
- **Interpreter:** a numpy-based interpreter for Triton kernels; can be used to:
  - Run programs line by line sequentially
  - Set breakpoints and investigate intermediate values
  - Anecdotically, this has saved me so much time

- **Proton:** a CUPTI-based profiler for Triton; can be used to:
  - Trace kernels that were called
  - Locate bottlenecks in programs
  - Analyze timings and compute roofline
  - Flag performance regressions

Code Complexity



Community



Looking forward

Portability

TritonGPU-IR cannot be easily re-used across backends

- **Linear Layouts:** a new framework for unifying layouts within and across backends:
  - Blocked / DotOperand / MMA / Slice / etc. -> Linear
  - NvidiaMMAv2 / NvidiaMMAv3 / AmdMFMA / etc/ -> Linear

- **Structured Memory Accesses:** we still do not have portable abstractions for specialized memory units (e.g., TMAs):
  - This is a problem even within vendors (A100 vs H100)
  - This could lead to severe fragmentation of e.g., fastest matrix-multiplication / attention codes across vendors

Performance

Good performance in benchmarks is not always meaningful:

- **Performance cliffs are getting worse:**
  - Register spilling are still as bad
  - Software pipelining is becoming more complex
- **Triton can create unnecessary shared memory round-trips:**
  - This is quite glaring for `tl.sort`
  - Linear layouts may help here
- **Start exploring new paradigms:**
  - Warp specialization

Tools

Triton is still missing a set of potentially very useful tools:

- **Performance warnings:** could mitigate cliffs
  - The compiler knows when it can't prove alignment, and why
  - The compiler knows when it can't pipeline, and why
  - The compiler knows when it can't vectorize, and why
  - …
- **Runtime sanitizers:** frontend could instrument code
  - UBSan: detect integer overflows, divisions by zero
  - ASan: detect out-of-bounds accesses (might require TMAs)

Code Complexity

We need to make our code-base more modular and scalable

- **Software pipeliner refactoring:**
  - Make it possible to grow possible schedules without growing the core pipelining infrastructure

Community

Support for new users is virtually non-existent:

- **Developers don't have the bandwidth to answer questions:**
  - Slack workspace is primarily intended for technical chatter
  - Pull requests are reviewed (but our bar is high)
- **We need your help!**
  - We encourage joining CUDA MODE on discord (#triton)
  - We encourage users to submit PRs that substantially improve documentation

Thank you!