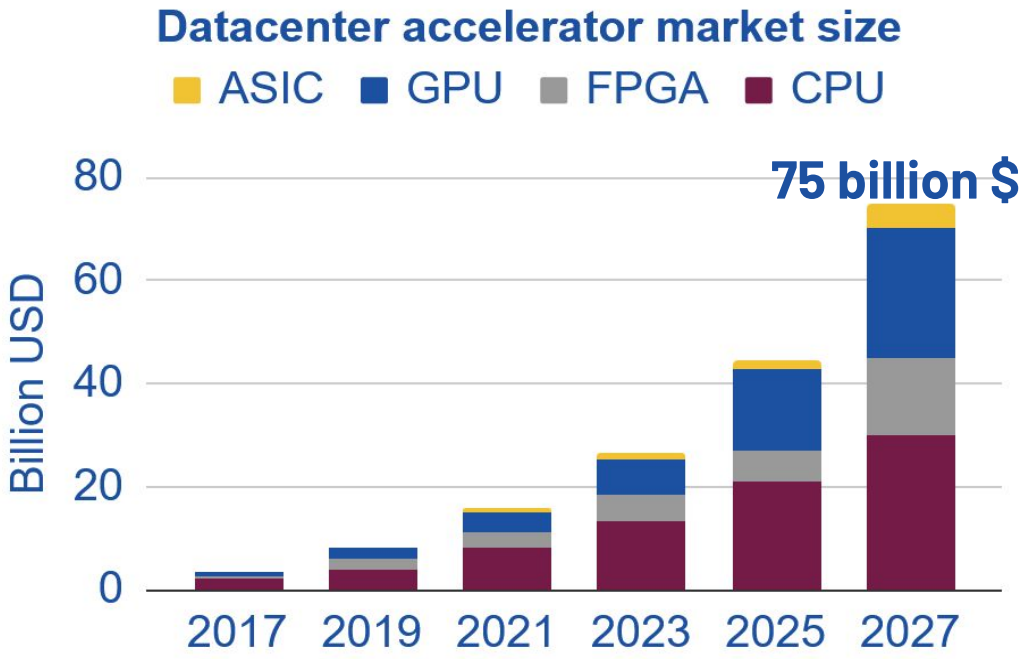
SCALE: a Cross-Vendor extension of the CUDA Programming Model for GPUs

Manos Pavlidakis

SPECTRAL COMPUTE

The need for High Performance at Low Energy Consumption

- The use of **accelerators increases**
- Accelerator **heterogeneity** increases [1, 2]
  - Different applications have different needs
  - Inference CPU, ASIC
  - Training GPU, FPGA
- **New** accelerator **APIs** are **continually emerging** [3]
  - E.g. NVIDIA→CUDA, AMD→RoCM, intel → oneAPI



https://www.grandviewresearch.com/industry-analysis/data-center-accelerator-market-report

[1] DOE ASCR Basic Research Needs Workshop 2018, Extreme Heterogeneity
[2] HPCA 2018, Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective
[3] ASPLOS 2020, AvA: Accelerated Virtualization of Accelerators

Problem: CUDA Dominates the GPGPU Ecosystem

- **CUDA** is the **most widely-used** GPGPU programming language

- CUDA is awesome:
  - **Early Entry** and **Maturity:** almost 20 years in the market
  - **Extensive ecosystem**: wide range of libraries, debugging tools, profilers
  - **Broad Adoption** in ML and AI: TensorFlow, PyTorch
  - **Vast community** of developers and extensive **documentation**

CUDA can only target NVIDIA hardware leading to vendor lock-in !

Challenge: Make GPU Programs as Portable as CPU ones

- There are **two solutions** to run the **same app** on **heterogeneous accelerators**

  1st: **Use the accelerator API:** Maintaining one app version per accelerator

  - N versions, where N equals to the num of accelerators

  2nd: **Use cross-platform** solutions (e.g., SYCL, HIP, Kokkos)

  - At least two versions of the code

Automatic Source2Source Translation Does Not Work

- Translators as **HIPify** and **SYCLomatic** transform CUDA to HIP or SYCL
- Source to source translation **has** serious **issues**
  - **Incomplete** code **conversion**
  - **NVIDIA CUDA** and **Clang-CUDA** are subtly **different languages** → The "dialect problem"
  - **No PTX Support:** Inline PTX blocks require manual porting
  - **Macros** are challenging for source translators

Once done: now you have two codebases to maintain !

SCALE: Ahead-Of-Time Compilation of CUDA for AMD GPUs

- Goal: **Expand CUDA to** support **multiple accelerators** (now AMD)
  - Make **GPU code as CPU**: write once, recompile for different hw

- **SCALE**
  - Avoids maintaining **multiple versions** of **code**
  - Converts **PTX Assembly** to **GPU machine code**
  - **Enhances CUDA** programming model
  - Overcomes the **CUDA dialect** issue
  - **Maps CUDA** features to **other architectures**

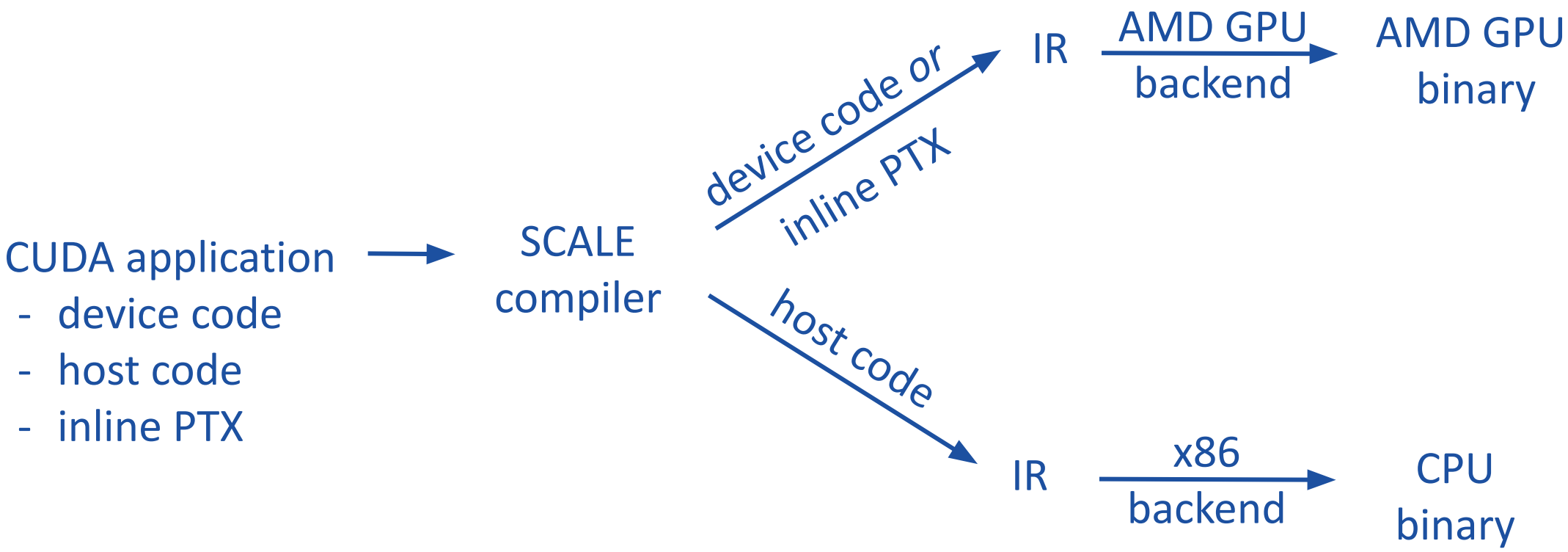
Why SCALE ?

Approach	Offers a Single Codebase	Resolves CUDA Dialect Issue	NVIDIA Proprietary Free
Cross-platform (HIP, SYCL, Kokkos)	-	✓	✓
Clang CUDA	✓	-	✓
Intercept CUDA calls (ZLUDA)	✓	✓	-
SCALE	✓	✓	✓

Outline

- Motivation and overview
- **SCALE's Clang compiler**
  - Enhance CUDA: Inline PTX + Warnings + Extensions
  - Resolve CUDA Dialect Issue
  - Cross-Platform Compatibility
- Evaluation
- Conclusions

SCALE Compiler Process



Inline PTX Assembly with Optimal Performance

- PTX lop3: does **any 3-input bitwise** operations
- SCALE converts lop3 to optimal IR

```
constexpr uin32_t Op = (0xF0 & 0xCC) ^ (~0xAA);
asm(
   "lop3.b32 %0, %0, %1, %2, %3;"
   : "+r"(x)
   : "r"(y), "r"(z), "n"(OP)
);
```

- LLVM IR

```
%0 = and i32 %y, %x
%1 = xor i32 %0, %z
%2 = xor i32 %1, -1
```

- AMD GPU Machine Code

```
v_and_b32_e32 v3, v4, v3
v_xnor_b32_e32 v2, v5, v3
```

Richer Compiler Warnings & Language Extensions

- SCALE adopts clang's **stricter warning** behavior
  - Projects using *-Werror* may fail to compile
  - *[[nodiscard]]* throughs warnings when return codes are ignored

- SCALE enhances CUDA with **optional language extensions**
  - *clang::loop_unroll offers explicit control over loop unrolling*
  - *__builtin_provable(bool X)* used in static analysis

Resolve the CUDA Dialect Issue

Dialect Differences Between clang and nvcc

There is no formal CUDA spec, and clang and nvcc speak slightly different dialects of the language.

This section is painful; hopefully you can skip this section and live your life blissfully unaware.

- The is **no formal** CUDA **specification** → **NO** "**standard**"
- The "**standard**" is **defined** by NVIDIA's **nvcc** behavior
- Many programs **fail** to **compile** with **clang**
  - E.g.: The following code is **accepted** by **nvcc**, but **rejected** by **clang:**

```
struct Foo {
    const int x = 2;
}
template<typename T>
__device__ void bar(Foo& o, T y) {
     o.x = 7;  // Invalid write to a const field
}
```

- SCALE offers an **nvcc mode** on its **clang compiler**

Match different Warp size of NVIDIA and AMD GPUs

- **NVIDIA** warp size is **32**, whereas **64** in (some) **AMD** GPUs
  - This difference affects thread scheduling, synchronization, etc.
- Many **CUDA applications assume** NVIDIA's **32-thread warps**
  - Different warp size leads to incorrect behaviors of operations like ballot and shuffle
- SCALE uses two techniques:
  1. **warp32 emulation:** Splits a 64-lane warps into two 32-lanes
  2. **cudaLaneMask_t:**
     - Extends warp-level operations to 64-lanes
     - Throughs warnings when wrong warp size is used

Convert NVIDIA's Compute Capability to AMD's

- **Compute capability** (cc) system enables **hw specific features**
- **Returning AMD cc to CUDA** apps results in **errors**
  - **NVIDIA** uses ***sm_60, sm_80***
  - **AMD** uses ***gfx1030***
- **SCALE** maps NVIDIA compute capability to a AMD
  - Using by **default sm_86** to ensure compatibility
  - Providing a **CUDA** installation **directory** per **AMD target** for more flexibility

Execute CUDA Libraries to AMD GPUs

- CUDA **Runtime** and **Driver** libraries
  - SCALE re-implements CUDA Runtime and Driver APIs using AMD system calls

- **CUDA-X libraries** such as cuBLAS and cuFFT
  - SCALE maps CUDA-X libraries to their functionality to AMD's ROCm equivalents
  - E.g. *cublasCreate(...)* → *rocblas_create_handle(...)*

Outline

- Motivation and overview
- SCALE's Clang compiler
  - Enhance CUDA: Inline PTX + Warnings + Extensions
  - Resolve CUDA Dialect Issue
  - Cross-Platform Compatibility: Warp size and Compute Capability system
- **Evaluation**
- Conclusions

Testbed

- Five different AMD GPUs microarchitectures
  - gfx900 (Vega 10, GCN 5.0)
  - gfx1030 (Navi 21, RDNA 2.0)
  - gfx1100 (Navi 31, RDNA 3.0)
  - and the **datacenter** grade gfx942 (MI300X, CDNA3) , gfx94a (MI210, CDNA3)

- Real world applications
  - AMGX, FLAMEGPU2, ALIEN, GOMC, GPU JPEG2K, XGBoost, Fais → **NO** AMD support
  - Rodinia suite, GPUJPEG, hashcat, LLaMA C++, Thrust, stdgpu → **HIP** for AMD

CUDA Coverage

- SCALE currently supports
  - 13# open-source CUDA projects
  - 5# AMD GPU architectures

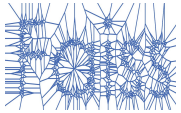
- To run all of those SCALE covers
  - 88% of the CUDA 12.6 runtime API
  - 70%of the CUDA driver API
  - 80% of the CUDA math API
  - 100% cuSOLVER, cuBLAS, and cuFFT

Run on AMD **only** with **SCALE**

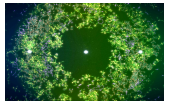
amgX

FLAMEGPU2

Faiss



alien



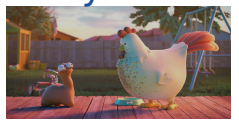
GOMC

gpujpeg2k

XGBoost

Multiple code bases:
HIP for AMD and CUDA for NVIDIA

cycles



hashcat

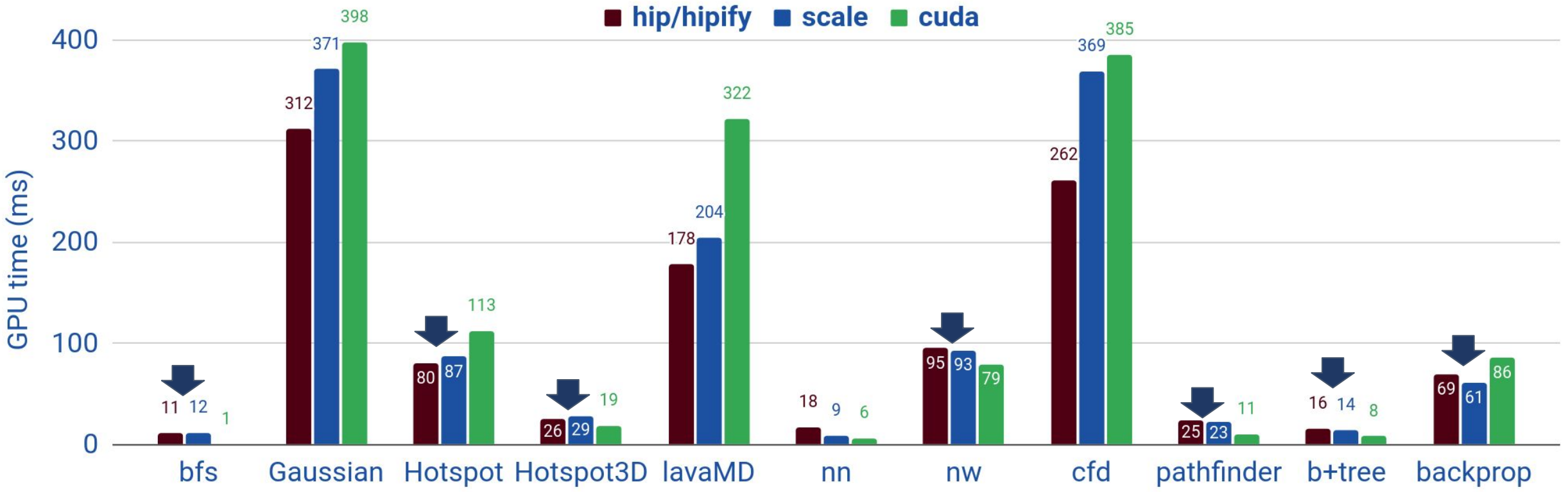
stdgpu

GPUJPEG

thrust

llama

SCALE offers near ROCm (native) performance

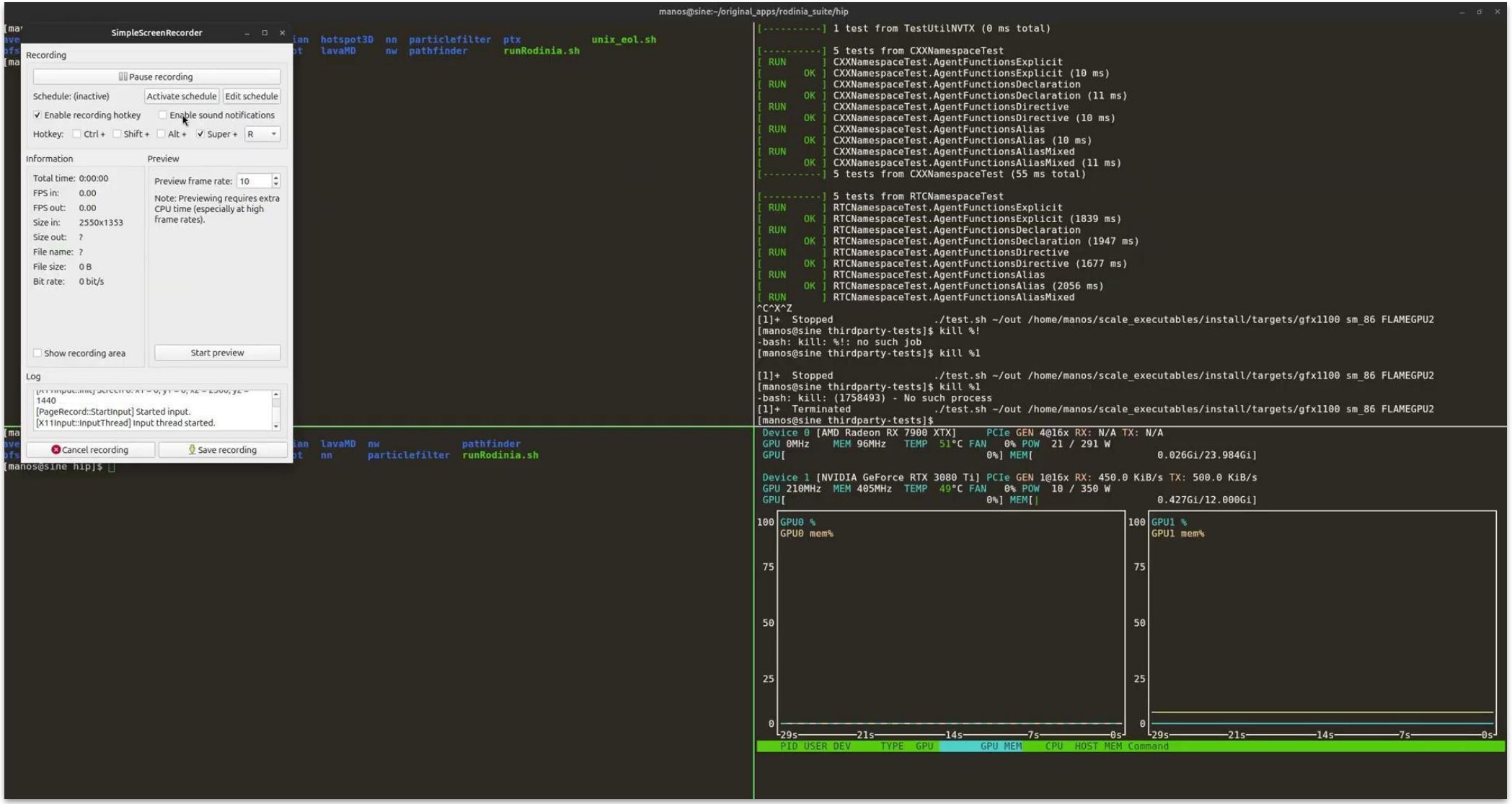


- Currently we **focus** in **coverage** and **not** in **optimizations**

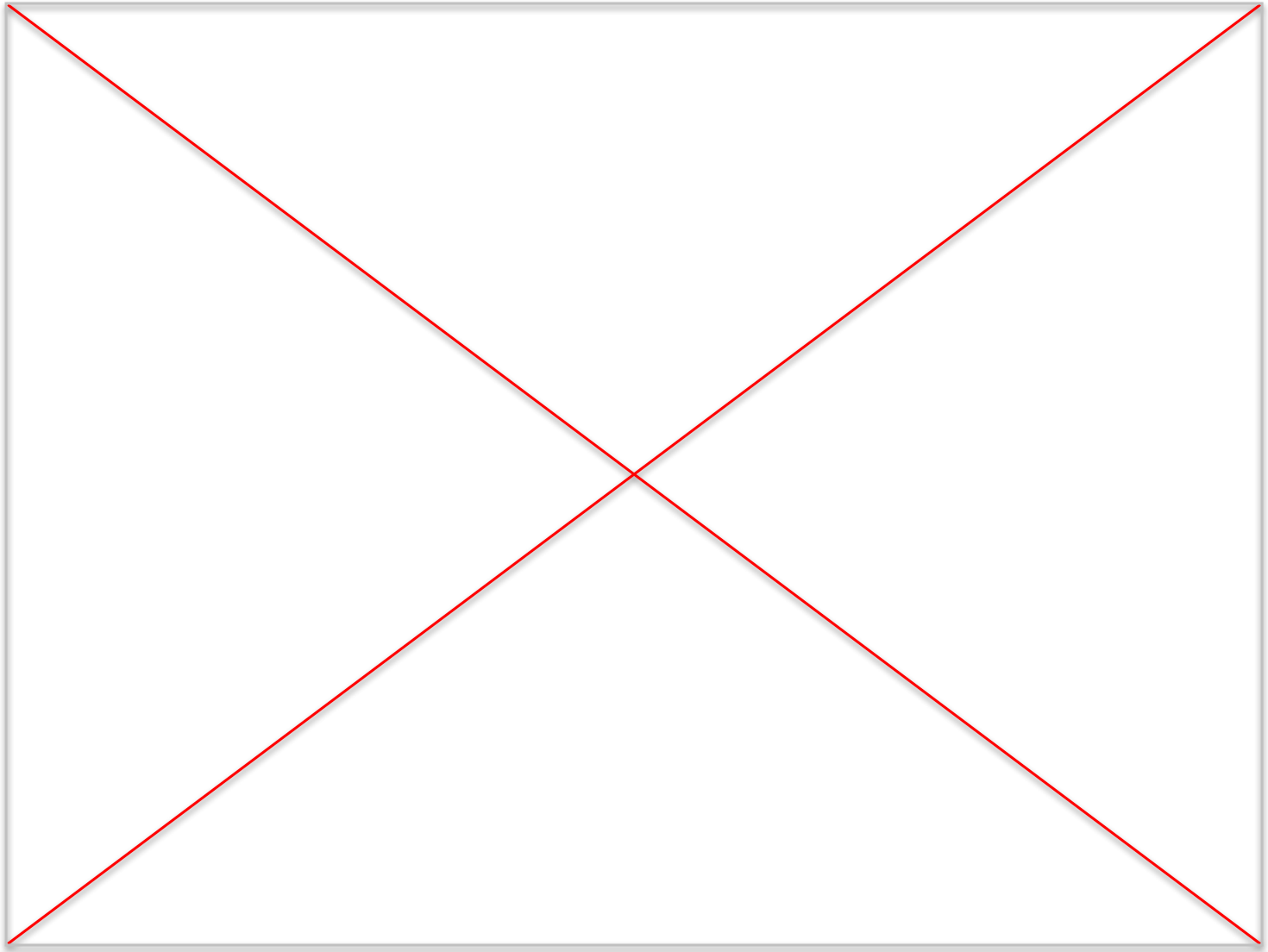
Summary

- **SCALE** enables **seamless execution** of **CUDA** apps to **AMD** GPUs using
  - A **clang** based **compiler:** eliminates the need for multiple code based
  - An **nvcc** mode to its compiler to resolve the *"**CUDA dialect issue**"*
  - Language **extensions** to enhance **GPU programming**
- We demonstrate SCALE's capabilities using
  - **13# real world** frameworks
  - **5# AMD** architectures

Demo: run CUDA apps with SCALE, HIP, and CUDA



Demo: HIPify issues



SCALE: a Cross-Vendor extension of the CUDA Programming Model for GPUs

Thank you

Manos Pavlidakis

manos@spectralcompute.co.uk

SPECTRAL COMPUTE