GPGPUプログラミングのデバッグを楽にしたい

Zin (Sho Ikeda)

アジェンダ

- GPGPUは楽しい
- CPUとGPUで共通で使えるコードを書く
- 今回試した環境

GPGPUは楽しい

- 大量の並列演算器による高速な処理
  - CPUの数倍のコアによる大量並列処理
- グラフィックスや画像処理に向いている
  - 計算結果を見るのが楽しい

GPGPUは楽しい、ただし…

- GPGPUプログラミングは難しい
  - GPUアーキテクチャの理解 (Data-Level Parallelism, Memory address space, etc…)
  - GPU APIの理解 (DX, OpenCL, Vulkan, etc…)
  - プログラミング言語の理解
  - デバッグが難しい(ツールの不足, GUIのクラッシュ, NaN NaN Nan!, etc…)

GPGPUのデバッグを簡単にできないか？

CPUとGPUで共通で使えるコードを書く

- CPUのデバッグ環境をGPUコードでも使えるようにする
  - CPU側で実装ロジックが正しい事を検証してからGPU実行する
  - なるべくGPU環境でデバッグしない
- CPUによる同時実行、フォールバック実行
  - ただし、コードは(おそらく)GPU向けに最適化されている

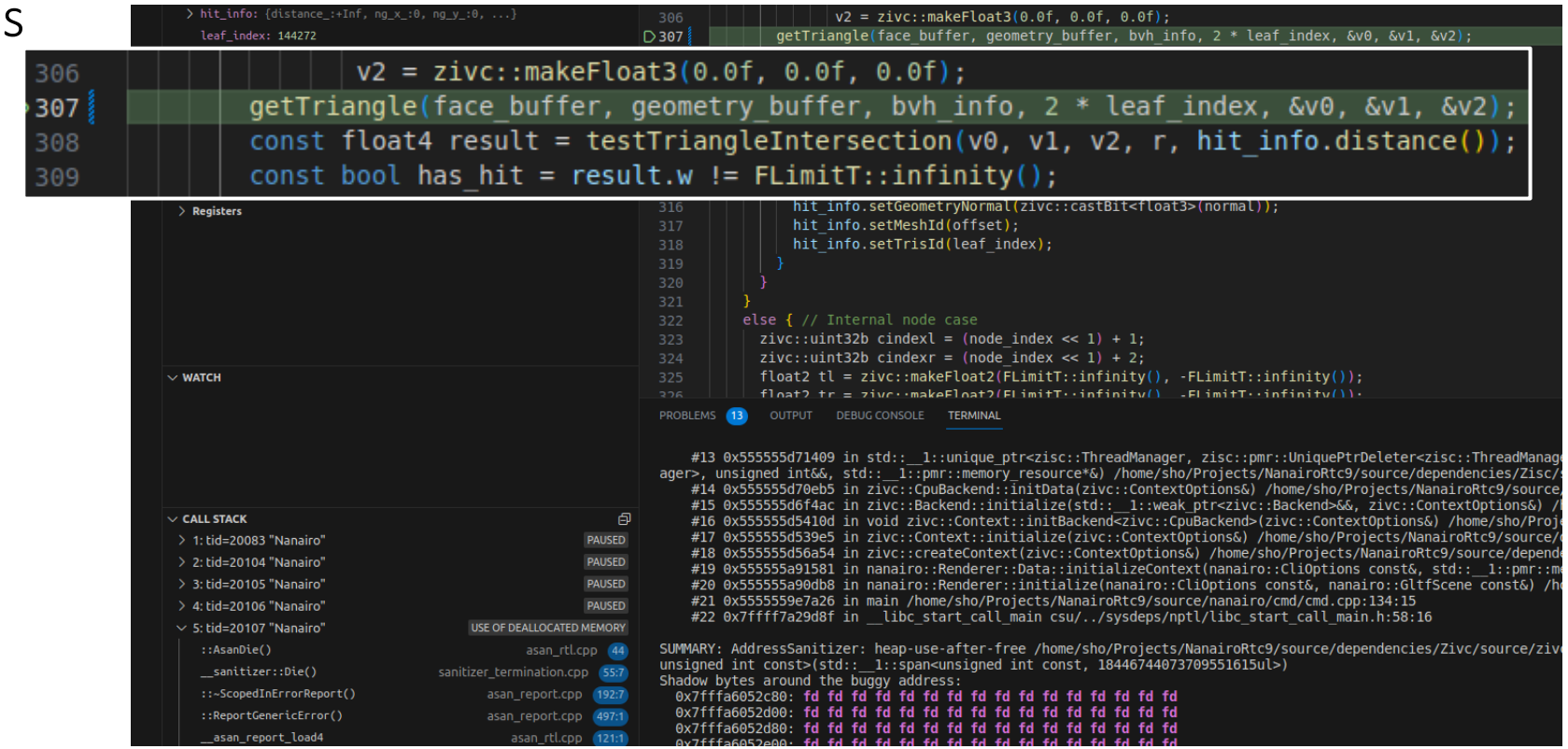
本セッションはC++環境を想定

- C++17やC++20が使える
- Visual Studioやgdbなどのデバッガーが使える
- GPU側の言語はCuda, HIP, OpenCL C++など
- たぶんRustにも適用できる

CPUのデバッグ環境

- ブレイクポイント, ステップ実行, etc...

- Sanitizer (Address, UndefinedBehaviour, etc...)
  - AddressSanitizer: visual studio, clang, gccでサポート
  - Out of bounds access
  - Use after free
  - Division by zero
  - Mis-aligned access



CPUとGPUコードを共通化する1

- 特定の処理(関数)を共通化する
  - 乱数生成、レイの生成、BRDF、ライトサンプリングなど
- 比較的共通化しやすい
- 共通化できていない部分のGPUコードはデバッグしづらい

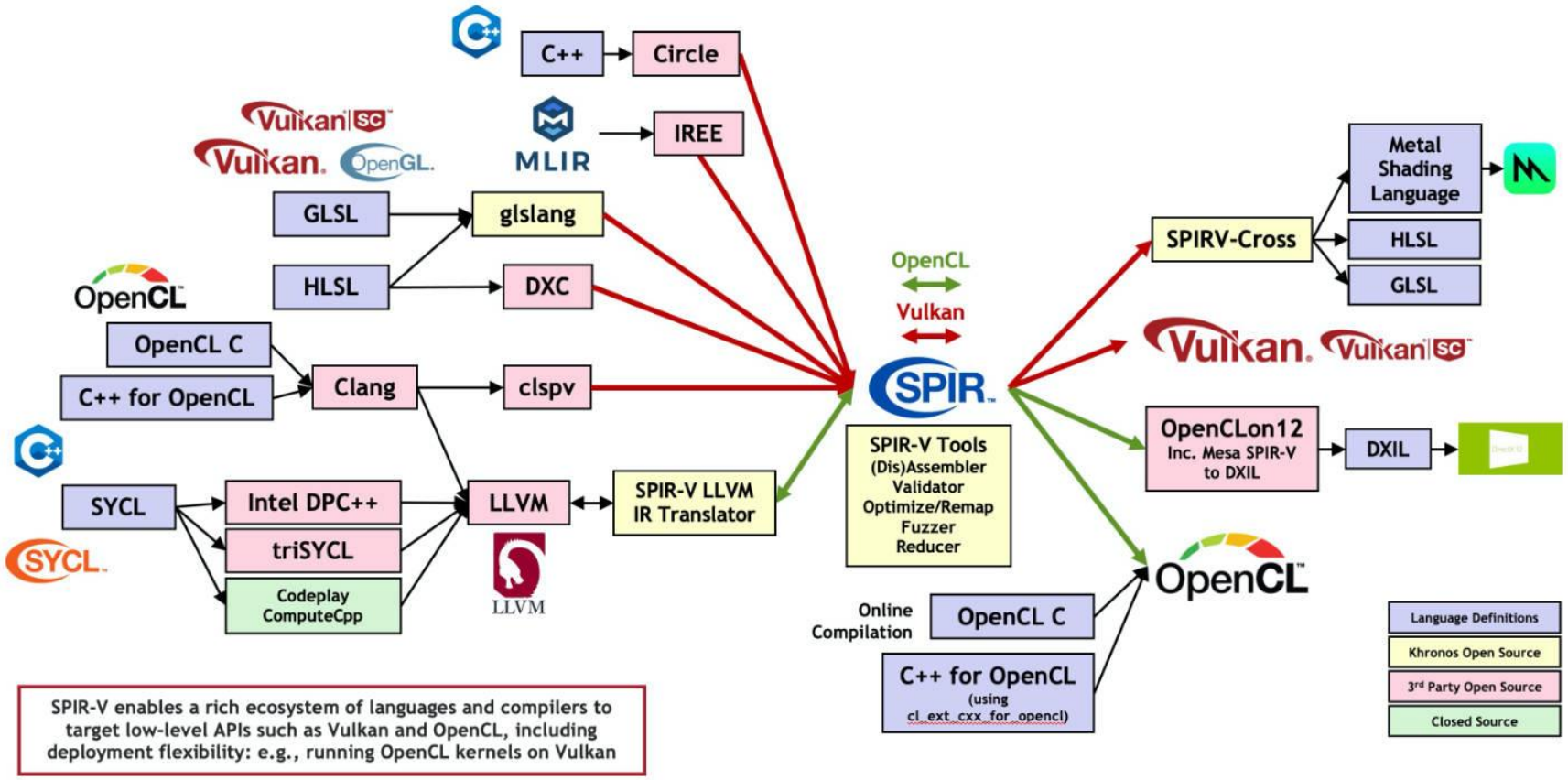
CPUとGPUコードを共通化する2

- カーネル全体をCPUとGPUで実行できるようにする
- カーネル全体のロジックをCPUで検証できる
- CPU側の実装が難しい
  - GPUアーキテクチャをCPU側でサポートする
    - thread_localを用いたwork-item情報の保持
    - coroutineを用いたwork-groupの実装
  - Device, Buffer, Kernelなどの抽象化

	Device	Buffer	Kernel
CPU	C++	配列	C++ function
GPU	Vulkan device	Vulkan buffer	OpenCL C++

今回試した環境

- C++20, Vulkan, OpenCL C++ (clspv)
- Linux+AMD, Windows+NV, cpu



ありがとうございました