HetGPU: The pursuit of making binary compatibility towards GPUs

Yiwei Yang

Background & Motivation

- GPU ecosystems fragmented across NVIDIA, AMD, Intel, Tenstorrent
- High-level frameworks (OpenCL, SYCL, HIP) only solve source-level portability
- No “compile once, run anywhere” for binaries → limits heterogeneous scheduling and live migration

Key Challenges

1. **Execution Model Divergence**: SIMT (NVIDIA/AMD) vs. MIMD (Tenstorrent)

2. **ISA Differences**: PTX/SASS vs. GCN/RDNA vs. RISC-V Vector

3. **Memory & Consistency**: Hardware shared memory vs. explicit DMA

4. **State Capture & Migration**: Abstracting registers, program counters, shared state across ISAs

System Overview

- **Portable IR (ptx subset)**: Virtual GPU instruction set
- **Compiler Toolchain**: CUDA C++ → LLVM IR → ptx
- **Runtime & Abstraction Layer**: JIT translation, unified API, cross-device scheduling
- **State Management**: Barrier-based cooperative checkpoint and restore

hetIR Design Highlights

- **SPMD Model**: Threads in a block treated independently—no built-in warp size
- **Explicit Sync & Predication**: `barrier()`, `set_predicate()`, predicated blocks
- **Virtualized Special Ops**: `VOTE_ANY`, `SHUFFLE`, atomics, etc.
- **Unified Memory Ops**: `LD_GLOBAL/ST_GLOBAL`, `LD_SHARED/ST_SHARED`

Compiler Frontend & Backends

- **Frontend**: Clang/LLVM with custom hetIR intrinsics
- **Backend Outputs**:
  - **PTX** → NVIDIA JIT (cuModuleLoadDataEx)
  - **SPIR-V** → AMD/OpenCL & Intel/Level Zero
  - **Metalium** → Tenstorrent TT-MLIR → assembler

Runtime System

- Device detection and on-demand JIT with kernel caching
- Unified APIs: `gpuMalloc`, `gpuMemcpy`, streams, events
- SIMT→MIMD mapping strategies on Tenstorrent:
    - **Single-Core Vector Mode** (warp emulation)
    - **Multi-Core Partitioning** (split block across cores)

Checkpointing & Live Migration

- **Cooperative Checkpoint** at hetIR barriers via `pause_flag`
- **State Snapshot**: per-thread registers, program counters, shared memory
- **Segmented Restart**: split kernel by barriers, resume next segment on target GPU
- **Data Transfer**: host-mediated or peer-to-peer copy

Preliminary Evaluation

- **Functional Portability**: 10+ kernels validated across NVIDIA, AMD, Intel, Tenstorrent
- **Performance Overhead**:
  - Compute-bound < 10% slowdown
  - Memory-bound < 5%
  - JIT latency 10–200 ms on first launch
- **Live Migration Demo**: 30 s job with 2.2 s total downtime

Gemini integration

- **WorkFlow**: MILR -> TOSA to Linalg -> Linalg Gemmini Dialect -> Gemmini->spike

- **Progress:**
  - Setup up the workflow in HetGPU
  - 25/100 tests passes, mostly gemm related

- **Next Week Job**:
  - Finish the remaining job.
  - Get Multi-NPU framework planned.
  - Target kodiak.

Conclusion & Future Work

- Achieved true “compile once, run anywhere” for GPU binaries
- Bridges SIMT and MIMD, supports heterogeneous live migration
- **Next Steps**:
  - Architecture-aware optimizations (e.g., Tensor Core support)
  - Leverage unified memory / pre-copy to minimize downtime