Open 3D Alliance: RISC-V

An open invitation to collaborate on 3D Graphics Hardware and Software
for mobile, embedded, and innovative purposes

With thanks to Pixilica, GoWin, and Western Digital

August 26, 2019

Why collaborate?

- 3D is hard. It's also not the same as HPC
- NVIDIA, AMD, Imagination - cannot meet "unusual" needs
- Working together on flexible standards, everyone wins
- Without collaboration: 10-20 man-years development
- With collaboration: cross-verification (avoids mistakes)

What is the goal?

- You get to decide! No, really!
- Outlined here: some ideas and cost/time-saving approaches
- Two new platforms: 3D "Embedded", 3D "UNIX"
- Flexible optional extensions (Transcendentals, Vectors, Texturisation, Pixel/Z-Buffers - all optional)
- Good software support absolutely essential (basically, that means Vulkan)

Libre RISC-V Team

- Small team, sponsored by Purism and the NLNet Foundation
- Therefore, focus is on efficiency: leap-frogging ahead without requiring huge resources.
- OpenGL API? Gallium3D / Vulkan is better
- Gallium3D turns out to be a single-threaded interpreter (Vulkan is compiled, and can be parallelised)
- Independent teams have provided OpenGL to Vulkan adaptors
- Same approach on hardware: seek highest bang-per-buck Save design time, save implementation time

What (optional) things are needed?

- Vectorisation. (SIMD? RVV? Other?)
- Transcendentals (SIN, COS, EXP, LOG)
- Texture opcodes, Pixel/Z-Buffers
- Pixel conversion (YUV/RGB etc.)
- Optional accuracy (embedded space needs less accuracy)
- Options give implementors flexibility. No imposition: imposition risks fragmentation (however, collaboration does need some hard easily-logically-justifiable rules)

What is essential (not really optional)

- The software, basically. Anything other than Vulkan is a 10+ man-year effort
- Two new 3D "platforms". Vulkan compliance has implications for hardware, and, with the API being public, interoperability (and Khronos Compliance - which is Trademarked) is critical.
- Respecting that standards are hard to get right (and that consequences of mistakes are severe: no opportunity for corrections after a freeze)
- Respecting that, for collaboration and interoperability, some things go into a standard that you might not "need"
- Mutually respectful open and fully transparent collaboration. No NDAs, no "closed forums". We need the help of experts (such as Mitch Alsup) in this highly technical specialist area.

Why Two new Platforms?

- Unique pragmatic consequences of "Hybrid" CPU/GPU
- Embedded - no traps need be raised. Interoperability is impossible, software toolchain collaboration is incidental).
- UNIX - illegal instruction traps mandatory: software interoperability is mandatory and essential.
- 3D Embedded - failure to allow implementors the freedom to reduce FP accuracy automatically results in product failure (too many gates, too much power, equals end-user rejection).
- 3D UNIX - likewise. Also: failure to comply with Khronos Specifications (then use "Vulkan") is a Trademark violation.
- Solution: allow software to select FP accuracy level **at runtime**. (UNIX Platform: IEEE754. 3D UNIX: Vulkan).
- HW: slow for IEEE754, fast for 3D. Product now competitive!

What has our team done already?

- Decided to go the "Hybrid" Route (Separate GPUs requires a full-blown RPC/IPC mechanism to transfer all 3D API calls to and from userspace memory to GPU memory... and back).
- Developed Simple-V (a "Parallelising" API)
  (Simple-V is very hard to describe, because it is unique: there is no common Computer Science terminology)
- Started on Kazan (a Vulkan SPIR-V to LLVM compiler)
- Started work on a highly flexible IEEE754 FPU
- Started work on a "Precise" CDC 6600 style OoO Engine, with help from Mitch Alsup, the designer of the M68000
- Variable-issue, predicated SIMD backend, Vector front-end "precise" exceptions, branch shadowing, much more
- All Libre-licensed and developed publicly and transparently.

Why Simple-V? Why not RVV?

- RVV is designed exclusively for supercomputing
  (RVV simply has not been designed with 3D in mind).
- Like SIMD, RVV uses dedicated opcodes
  (google "SIMD considered harmful")
- 98% of FP opcodes are duplicated in RVV. Large portion
  of BitManip opcodes duplicated in predicate Masks
- OP32 space is extremely precious: 48 and 64 bit opcode space
  comes with an inherent I-Cache power consumption penalty
- Simple-V "prefixes" scalar opcodes (all of them)
  No need for any new "vector" opcodes (at all).
  Can therefore use the RVV major opcode for 3D
- SV augments "scalar" opcodes. Implications: it is relatively
  straightforward to convert an *existing design* to SV.
  SV "slots in" between instruction decode and the ALU.

Simple-V "Prefixing"

- SV "Prefix" does exactly that: takes RVC and OP32 opcodes and "prefixes" them with predication and a "vector" tag
- Three prefix types: SV P32 (prefixed RVC), P48 and P64
- Prefixed RVC takes 3 "Custom" OP32 opcodes.
  P48 takes standard OP32 scalar opcodes and "prefixes" them
  P64 adds additional vector context on top of P48
- "Prefixing" is a bit like SIMD. Vectors may be specified of length 2 to 4, elements may be "packed" into registers, opcode element widths over-ridden.
- Convenient, but not very space-efficient (and VBLOCK is)

VBLOCK Format

- Again: hard to describe. It is a bit like VLIW (only not really) A "block" of instructions is "prefixed" with register "tags" which give extra context to scalar instructions within the block
- Sub-blocks include: Vector Length, Swizzling, Vector/Width overrides, and predication. All this is added to scalar opcodes! **There are NO vector opcodes** (and no need for any)
- In the "context", it goes like this: "if a register is used by a scalar opcode, and the register is listed in the "context", SV mode is "activated"
- "Activation" results in a hardware-level "for-loop" issuing **multiple** contiguous scalar operations (instead of just one).
- Implementors are free to implement the "loop" in any fashion they see fit. SIMD, Multi-issue, single-execution: anything.

Other Standard Proposals

- Ztrans and Ztrig* - Transcendentals and Trigonometrics (optional so that Embedded implementors have some leeway)
- ISAMUX / ISANS - stops arguments over OP32 space (also allows clean "paging" of new opcodes into e.g. RVC)
- MV.SWIZZLE and MV.X - RV does not have a MV opcode.
- Zfacc - dynamic FP accuracy. Needed for "fast" Vulkan native and to switch between fast 3D accuracy and IEEE754 modes.
- These - and more - need your input! 3D is hard!
- The key strategic premise: these are required as **public** standards, because the **software** is to be public.
- This is **not** understood by the RISC-V Foundation. ("custom" status not appropriate for high-profile mass-volume end-user APIs such as Vulkan).

Summary

- 3D is hard (and pure Vectorisation gets you 25% of commercially-acceptable performance).
- Layered optional extensions are going to be key to acceptance by a wide variety of 3D Alliance Members.
- With a custom specialised SPIR-V (Vulkan) Compiler being an absolutely critical strategic requirement, RVV and its associated compiler (still not developed) is of marginal value (no clear benefits, extra cost)
- Question everything! Your input, and a willingness to take active responsibility for tasks that your Company is critically dependent on, are extremely important.
- Public and transparent Collaboration is key. There is simply too much to do.

The end

Thank you

- http://lists.libre-riscv.org/pipermail/libre-riscv-dev/
- http://libre-riscv.org/simple_v_extension/abridged_spec/
- https://libre-riscv.org/ztrans_proposal/
- https://libre-riscv.org/simple_v_extension/specification/mv.x/