LLVM + Gallium3D: Mixing a Compiler With a Graphics Framework

Stéphane Marchesin
<marchesin@icps.u-strasbg.fr>

What problems are we solving?

- Shader optimizations are really needed
  - All Mesa drivers are spanked really bad in benchmarks that involve shaders
- Advanced optimizations
  - Complex
    - – Intermediate language definition must be flexible
      - Must support all architectures
      - Must support all optimizations
  - Keep optimizations driver-independent
    - – Share as much as possible between the drivers

What other problems are we solving?

- Target multiple architectures
  - CPUs
    - Current Mesa/Gallium3D has some code generation paths for SSE
      - Bug prone (currently buggy both in Mesa & Gallium3D)
      - Drop that burden onto another library
  - GPUs
    - GPU-specific optimizations must be supported
- Not interested in reinventing the wheel

What hardware problems are we solving?

- In the nvidia case
  - All cards
    - Input/output register allocation
    - Loop unrolling
    - Function inlining

What other hardware problems are we solving?

- In the nvidia case
  - nv30/nv40
    - Implements most of the ARB shader specs
      - Full instruction set
    - Vec4 architecture
    - Free abs, neg, postmultiplication
    - All swizzles are supported
    - (VP) Merged instructions
  - nv50
    - More RISC-like
    - Scalar architecture
    - Texture access limitations

What's LLVM?

- A compiling, optimizing and code generation framework
- Targets many CPU architectures
- Good at optimization
  - llvm-gcc is on par with gcc performance-wise
- Widely used in the field
  - In particular, by Apple for shader optimization in OSX

Why LLVM?

- Open source
- Modular
  - Unlike GCC
  - Technically it is a library
    - Interface to generate LLVM intermediate representation, optimize and output machine code
- Targets
  - Numerous existing CPU targets
  - Easy to retarget

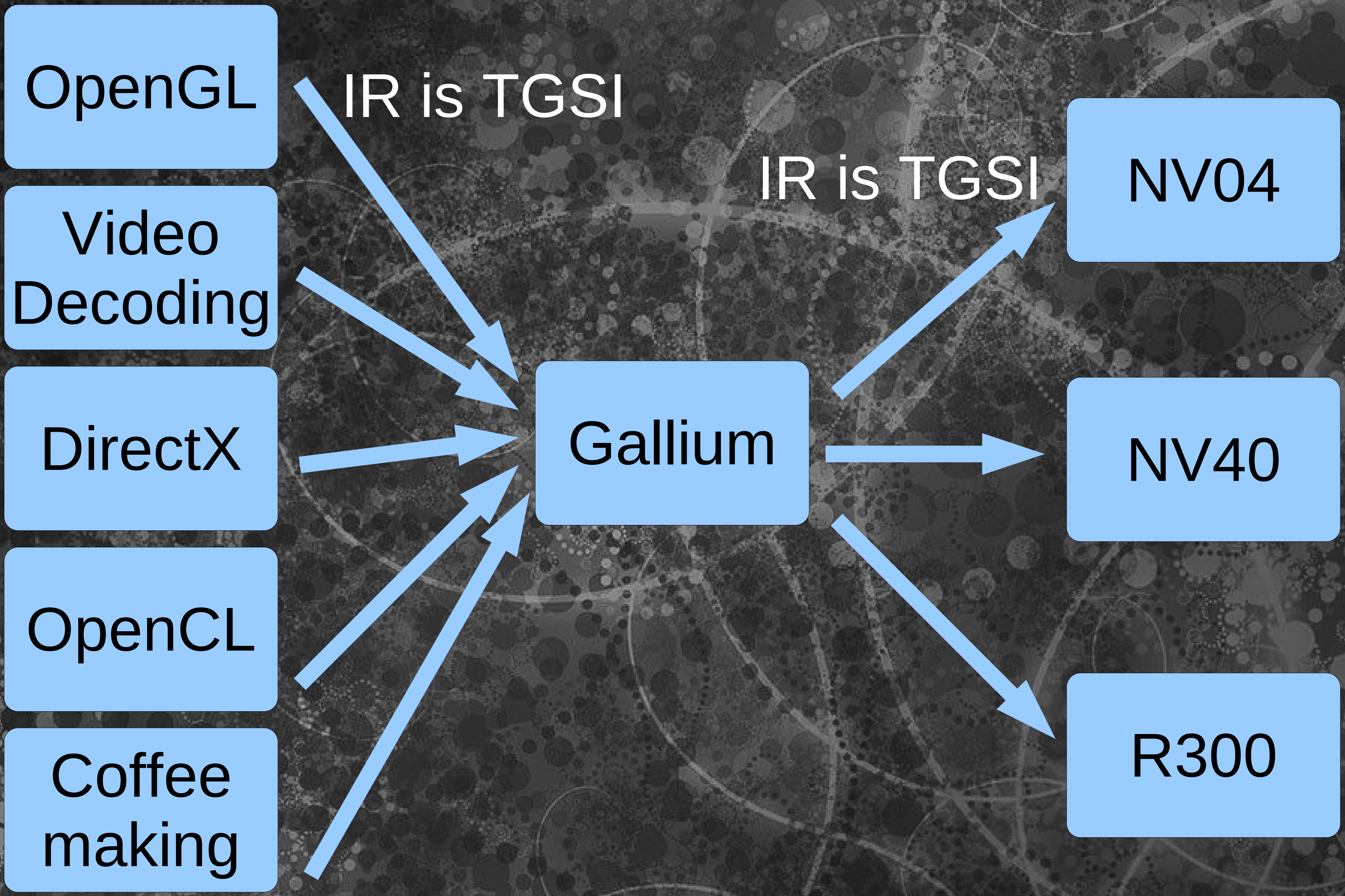
What will we get from LLVM?

- An intermediate representation (IR) language
  - SSA-like
- LLVM code generators for the CPU
  - SSE
  - Altivec
  - ...
- Optimization passes
  - All common optimizations

What's Gallium3D?

- A driver framework
- Everything is a shader internally
- Shaders should be optimized

Do you have a nice diagram?



OpenGL

Video Decoding

DirectX

OpenCL

Coffee making

IR is TGSI

Gallium

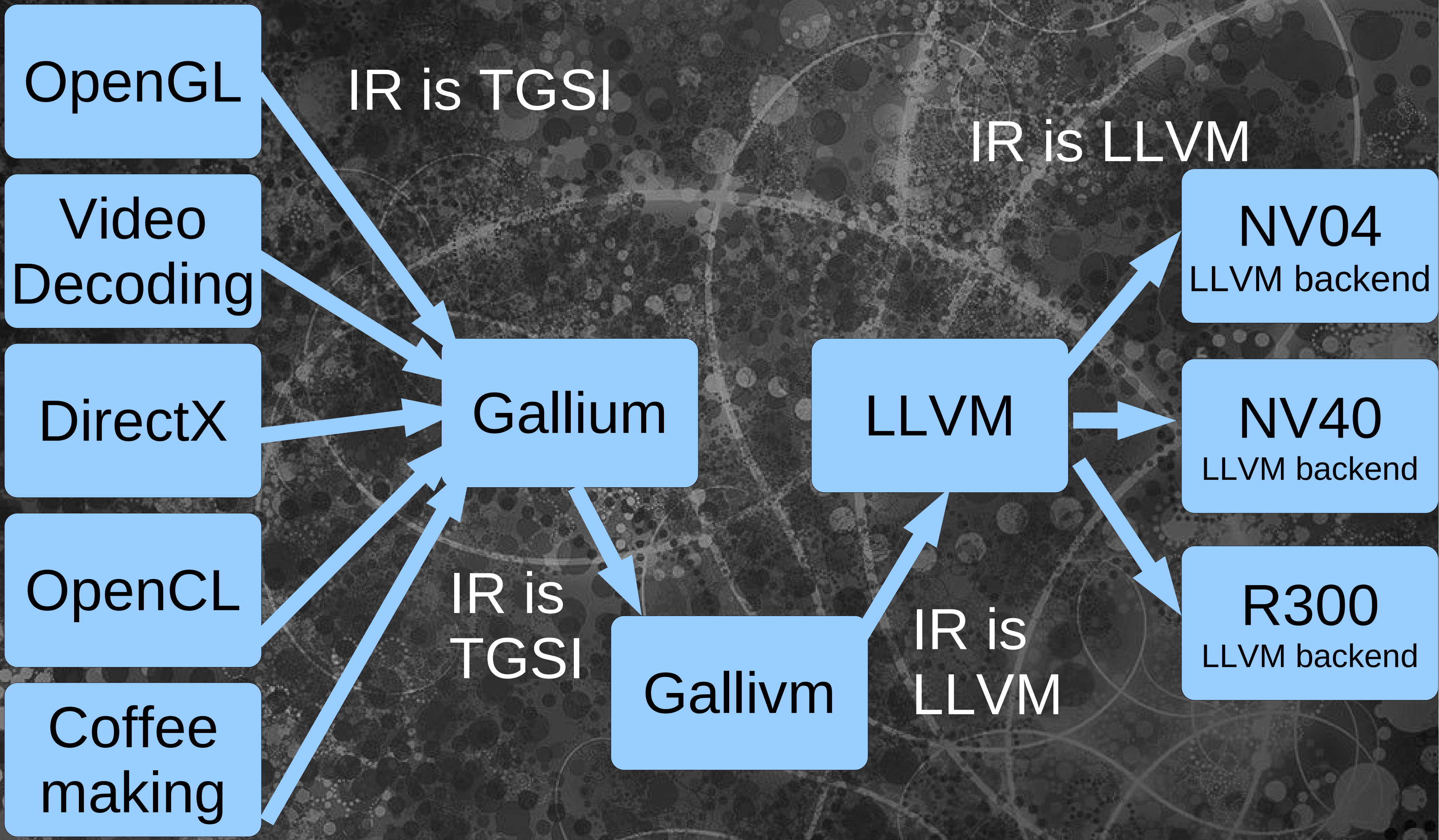
IR is TGSI

NV04

NV40

R300

Where do you plug LLVM in?



OpenGL

Video Decoding

DirectX

OpenCL

Coffee making

IR is TGSI

Gallium

IR is TGSI

Gallivm

IR is LLVM

LLVM

IR is LLVM

NV04 LLVM backend

NV40 LLVM backend

R300 LLVM backend

How does it work?

- The previous diagram is for shaders only
  - Rest is unchanged
- Gallivm outputs LLVM intermediate representation from TGSI intermediate representation
- LLVM optimizes it
- Each driver has an LLVM backend that describes its GPU

What about shaderful GPUs?

- Write an LLVM backend that describes your GPU:
  - Instruction set
  - Registers
  - Constraints
- (maybe) Add some specific optimization passes
  - Merge instructions into VLIWs

What about shaderful GPUs?

- LLVM's tablegen language
  - Describe your architecture high level-ish
  - .td files
  - Tablegen example:

```
def MUL : bin_instruction<0x02, "mul", fmul, (outs
VR128:$dst),   (ins VR128:$src1, VR128:$src2),
[(set VR128:$dst), (fmul VR128:$src1, VR128:$src2)] ,1>;
```

  - Compiles into C++ files

What about fixed pipe GPUs?

- Fixed pipe cards have shaders, but:
  - With a restricted instruction set
  - With a limited number of instructions (think 2-4)
  - These instructions describe
    - Texture combining
    - Fog
    - Constant color
    - ...

What about fixed pipe GPUs?

- Write an LLVM backend that describes fixed pipe:
  - Texture combining
  - Fog
  - ...
- Let LLVM's rewrite engine figure out how to make fixed pipe instructions from a shader
  - Works for the simple shaders
  - Of course will not get GLSL running on old GPUs

What do we have today?

- TGSI to LLVM IR translation
- Partial LLVM code generators for our GPUs
  - NV40
  - NV50
  - R300
  - Fixed pipe (NV04)
- Code generation to the CPU for vertex shaders

What do we have today?

- Target GPUs and CPUs with a single infrastructure
- Reuse CPU code generators from LLVM
  - Vectorised (SSE/Altivec)
  - Scalar (x86,amd64, PPC, ARM, MIPS...)
- Get access to a wide range of optimizations
  - Existing LLVM optimization passes

Do we live happily everafter?

- Not yet
- Finish the LLVM backends
- Iron out code generation problems
- Add new LLVM optimizations passes
- Changes in LLVM itself
  - More intermediate level instructions in LLVM (especially vector ones)
  - Straightforward support for VLIW

And now, do we live happily everafter?

«They lived happily everafter, and had many optimized little programs»