Developer Toolchain for

Paul T. Robinson
Sony Computer Entertainment
LLVM Dev Meeting, 7 Nov 2013

Agenda

- ***PlayStation®4 – Info for game teams***
- Why Clang?
- Special Considerations
- Hacking on Clang/LLVM
- Now and the Future

PlayStation®4

- "Next Gen" PlayStation® console
  - Powerful game machine
  - Modern graphics features
  - PC based architecture
  - Lightning fast memory
  - New networking and interface features

CPU

- AMD x86-64 Jaguar
- Low power consumption, low heat
- 8 cores, 8 HW threads
- 2MiB L2 cache per 4 core group
- 32KiB L1 I-cache and D-cache per core
- Atomics, threads, fibers, ULTs (user-level threads)

GPU

- DX 11.1+ feature set with SCE custom features
  - Fine-grained cache control
  - Performance counters
  - Extra debugging support
- Asynchronous compute architecture
  - Carefully balanced for maximum graphics power plus compute tasks
- 800MHz clock, 1.843 TFLOPS
- Greatly expanded shader pipeline compared to PS3™
  - Geometry and tessellation shaders
  - More direct exposure to shader stages than DirectX

RAM

- 8GB 256 bit GDDR5
- GDDR5 is very high end graphics memory only found on PC graphics cards
- Fully unified address space
- 176 GB/s total bandwidth

Toolchain at a Glance

- Windows 7 (and later), 64-bit
- Tools are fully integrated into Visual Studio®
  - 2010 , 2012, and later
  - Simple wizard-based project creation
- CPU compiler, shader compiler, linker, debugger
- SN-DBS (distributed build system)
- CPU and GPU performance analyzers
  - Real time and static analysis
- Various supporting binary utilities

CPU Compiler

- Compiler – LLVM with the Clang front end
  - Highly conformant C and C++ front end
  - Great C++ 11 support
  - Excellent diagnostic messages
  - Fast compilation
  - Excellent code generation
- Updates to newer versions will be regular
  - Driven by SCE and the open source community
- Comprehensive set of intrinsics

CPU Compiler

- Compatibility
  - Microsoft
    - Some attributes and pragmas have different syntax
  - GCC
    - Largely compatible by default
    - Many, but not all attributes are supported
  - PS3™ and PS Vita
    - Some compatibility due to GCC compatibility
- Beware of relying on undefined behavior in other compilers!

Linker

- Fast, mature linker
  - Comparable to GNU gold
- Fine-grained dead-stripping/de-duplication
  - Fragments based on symbol relocations
  - Operates on code and data
  - Does not need separate section per function/variable
- LTO support “on the list”
  - Positive results from evaluation, see our lightning talk

Debugger

- Debug your PS4™ code as you would your PC code
- Mirrors the Visual Studio® multi-threaded debugging feature set
- Advanced feature support
  - Core dump debugging
  - Parallel call stacks and watches
  - Thread-specific break points and trace points

Agenda

- PlayStation®4 – Info for game teams
- ***Why Clang?***
- Special Considerations
- Hacking on Clang/LLVM
- Now and the Future

Why Switch to Clang?

- SCE traditionally provided customized GCC-based toolchains
- SN Systems (Bristol UK) had a popular toolchain
  - Compiler (SNC) came from Apogee
  - EDG front end, proprietary optimizer/back end
  - Specifically designed for RISC (MIPS, PPC, ARM)
- SCE bought SN; now we have two PS3™ toolchains
- PS4™ is coming up; SNC can't target x86…

Why Switch to Clang?

- SCE Worldwide Studios working with LLVM since 2008
- Four choices to evaluate (circa 2010):
  - Straight gcc
  - Hybrid llvm-gcc
  - EDG front end (from SNC) + LLVM
  - Clang/LLVM (2.8)
- Technical evaluation not conclusive

Why Switch to Clang?

- EDG had a couple strikes against it
  - Home-brew “glue” – EDG didn’t want it
  - ARM experience (LLVM DevMtg 2010) similar
  - Debug info problematic
- Non-technical considerations mattered a lot
  - Clang not a clear winner, but on a trajectory
  - LLVM community considered more “nimble” than GCC
  - Clang+LLVM “joined at the hip” so less effort to stay current
- Subsequent experience validated this decision

Agenda

- PlayStation®4 – Info for game teams
- Why Clang?
- ***Special Considerations***
- Hacking on Clang/LLVM
- Now and the Future

What We Provide

- Toolchain, SDK, samples
- Developer Services organization
  - Front-line support in multiple time zones
- End-user documentation
  - Compiler Reference; Transition Guide; ABI Overview; Intrinsics Reference
- Testing
  - Conformance, regression, functional

Our Licensees

- Many studios, with large development teams
  - SCE Worldwide Studios (“first party”)
  - Non-SCE (“third party”)
- Massive real-time graphics-intensive 3D simulations
  - And they call them games…
  - Vectors (LOTS of vectors) and not just GPU stuff
  - Piles of shaders (GPU kernels)
  - Data build (assets) much bigger than code

How They Build

- Optimization is always on (-O2 minimum)
  - Hard real-time frame rate deadlines are unforgiving
  - Assets (data) typically consume lots of memory
  - Using -O0/-O1, game *will not work*
  - Guess how well debugging goes
- Unity builds (#include *.cpp)
  - Improve optimization/inlining
  - Reduce data and debug-info size
  - LTO might replace this

And it looks like this…

- Short clip from inFAMOUS: Second Son
- Sample game play shows CPU managing lots of objects (sparks, debris etc)
- Complete trailer: http://www.youtube.com/watch?v=o-B40rzJHOY
- Longer section of game play: http://www.youtube.com/watch?v=Uibnf_Q_51s

Agenda

- PlayStation®4 – Info for game teams
- Why Clang?
- Special Considerations
- ***Hacking on Clang/LLVM***
- Now and the Future

Distributed Compiler/RTL Teams

- SCE Technology Platform
  - SN Systems (Bristol UK) – overall toolchain
  - U.S. R&D (San Mateo CA) – Clang/LLVM
  - SN Systems (Campbell CA) – SNC
  - SCEI (Tokyo) – RTL
- SCE Worldwide Studios
  - Tools & Technology (everywhere)

Driver Changes

- Changed various defaults, including:
  - -target and -mcpu (it's a cross-compiler)
  - -std=gnu++11 (only one C++ dialect, so far)
  - -fno-exceptions -fno-rtti
  - -fPIC -fstack-protector-strong
  - -fno-omit-frame-pointer -momit-leaf-frame-pointer
- Customized target
  - Header/lib search paths
  - Run our proprietary linker

Clang Changes

- Pragmas for compatibility or custom features
- Beefed up Windows hosting
  - Windows backslash separators
  - Non-ASCII characters in path/file names
- Intrinsic function documentation (coming)
  - Derived from our user manual
- Hack to reduce debug-info size
  - Suppress unused methods in classes
  - Not anything upstream would want, sorry

LLVM Changes

- Added X86 instruction subsets
  - Mostly superseded by upstream implementations
  - We did contribute TBM
- Backend tweaks
  - Relocation changes to enable spiffy dead-stripping
  - FastISel fiddling to get better debug-line info

Working with the LLVM Community

- Initial development hampered by secrecy
  - Blanket secrecy policy included compiler project
  - No advice/feedback from the group consciousness
  - Sitting on features and bugs and fixes applicable upstream
- Clear value to working openly with upstream
  - File a bug; somebody else might fix it
  - Send a patch; people will review/advise
  - Bigger features can get better design review
  - Putting private changes upstream reduces merge pain

Working with the LLVM Community

- Policy evolved over time
  - Pre-announcement: Nothing that reveals target details
  - Post-announcement: Nothing that reveals still-secret details
  - Details relevant to compiler essentially all public
- Workflow evolved to include sending fixes upstream
  - Sometimes doing that first!
- Lots of backlog to work through
  - Upstream review often means revising our patches
  - Adds work now, but eliminates future merge pain

Night of the Living Merge

- Merging once per upstream release was a nightmare
  - We stuck our fingers in everywhere
  - The 3.0 merge work took three months
- Actively progressing toward “living on trunk”
  - Automation will catch merge issues immediately
  - Fix-upstream-first will cost less in the long run
- Formal releases still based on upstream releases
  - Trading merge pain for branch pain… should be much less

Agenda

- PlayStation®4 – Info for game teams
- Why Clang?
- Special Considerations
- Hacking on Clang/LLVM
- ***Now and the Future***

Game Developers Love It!

Quotes from 3rd-party studios (not SCE):

"Clang for PS4™ is a ***huge improvement over GCC*** for PS3™. The ***same codebase*** (more or less) on the same hardware ***went from ~25 minutes to ~1.5 minutes.*** Clang's ***improved warning and error messages*** also pointed us to some very questionable legacy stuff."
--Steven Houchard, Gearbox

Game Developers Love It!

"Toolchain is really nice, ***link time is ~10 seconds, versus 2-4 minutes*** on PC."

--Sammy Fatnassi, Eidos Montreal

"The quality of diagnostics is also incredible! It's ***as pretentious as Google Search*** when it comes to ***correcting typos*** for us and that's a good thing."

--Jean-François Marquis, Ubisoft

…except when debugging

- Debugging optimized code is terribly painful
  - We meet studios' low expectations
- Alternate approach: Un-optimize just this function
  - Nearly every compiler allows function-level control
  - Except not Clang/LLVM
  - Most-requested feature by an order of magnitude
- Need function-level IR attribute to control optimization
  - Must work with normal or LTO builds
  - "optnone" in place, need help to implement semantics

More for the Wish List

- Hooks for developer support
  - Which optimization caused your problem?
  - Auto-bugpoint with MetaRenamer for bug reports
- Various goodies
  - Static analyzer – pretty much works today
  - Profiling – going through some changes upstream
  - Sanitizers – needs runtime support

Q & A