CPU Toolchain Launch Postmortem

Greg Bedwell
- x86-64 AMD “Jaguar” 8-core CPU
- 1.84 TFLOPS AMD Radeon™ based GPU
- 8GB GDDR5 RAM
Developer Toolchain for PS4

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

http://llvm.org/devmtg/2013-11/
postmortem noun

“an analysis or discussion of an event after it is over”

Now that we have successfully launched PlayStation®4 it is a good time to look back on our initial period of development up to that point

http://www.merriam-webster.com/dictionary/postmortem
First, some history...
SN Systems Ltd. was founded in 1988 to provide development tools for the games industry.

1990 Psy-Q

16-bit home systems
Psy-Q included a version of GCC that was highly customized for the needs of game developers.

1994 Psy-Q

PlayStation®
Continued to provide GCC but started researching a proprietary compiler technology – “SNC”
Provided SNC as part of the ProDG suite of tools although GCC was also available

2004 ProDG

PSP®
(PlayStation®Portable)

Provided SNC as part of the ProDG suite of tools although GCC was also available.

2006 ProDG

PlayStation®3
CPU Compiler is SNC

2011

PlayStation®Vita
CPU Compiler is Clang

2013

PlayStation®4
The CPU compiler project is a global effort
Builds and build systems
(and test systems)
We wanted to make use of all of our pre-existing test suites and test systems, which are shared across all targets.
<table>
<thead>
<tr>
<th>Build Configuration</th>
<th>Optimized</th>
<th>Assertions</th>
<th>Debug Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MinSizeRel</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Release</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>RelWithDebInfo</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Optimized</td>
<td>Assertions</td>
<td>Debug Info</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Debug</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MinSizeRel</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Release</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>RelWithDebInfo</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Clang configuration effect on game build time

Too slow for continuous integration testing!

Useful for continuous integration testing but still too much overhead over the pure release configuration of clang.
<table>
<thead>
<tr>
<th></th>
<th>Optimized</th>
<th>Assertions</th>
<th>Debug Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Checking</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Release</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

The same set of build configurations as we use for SNC

“Debug” for debugging

“Release” for releasing

“Checking” (Release+Asserts) for continuous integration testing
<table>
<thead>
<tr>
<th></th>
<th>Optimized</th>
<th>Assertions</th>
<th>Debug Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Checking</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Release</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
No executable size overhead for enabling debug data.
<table>
<thead>
<tr>
<th></th>
<th>Optimized</th>
<th>Assertions</th>
<th>Debug Info</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Debug</strong></td>
<td>❌</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td><strong>Checking</strong></td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td><strong>Release</strong></td>
<td>✅</td>
<td>❌</td>
<td>✅</td>
</tr>
<tr>
<td></td>
<td>Optimized</td>
<td>Assertions</td>
<td>Debug Info</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Debug</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Checking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Release</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

Suppress Windows crash dialog box for Checking and Release builds.
Improving test coverage
Game code is usually **BIG**

For a new platform, the amount of code that exists is *small*
Users value **CORRECTNESS** over all else
Why write *tests*

*when I can write a test generator?*
Why write tests when I can write a test generator?
Generate random C++ tests using SIMD language extensions and intrinsics to increase test coverage.
Random input → Vector Operation → Vector Operation → Vector Operation → Vector Operation → Vector Operation → Result
Random input → Result
Random input → Result
Undefined behaviour makes runtime-behaviour random testing hard!
Solution: Use a ‘safe’ wrapper to make all undefined behaviour defined for the purpose of the test.
Reducing optimization bugs
bugpoint

Windows
Clang integration
An alternative approach
SNC’s max_opts

*Not representative of actual visuals
SNC’s max_opts
SNC’s max_opts
SNC’s max_opts
SNC's max_opt
SNC's max_opts
SNC optimizer

SSA Form
Rule Based

Every transformation is guarded by a specific check
SNC keeps an internal counter of the number of transformations it performs.

```c
if (/* conditions match */ &&
    Opt_Enabled(permuteConvertedToOpsCalartToVector))
{
    Trace_Opt(permuteConvertedToOpsCalartToVector);
    /* Perform optimization */
}
```

Allows the user to specify a limit on the command line after which no further transformations can be performed.
SNC’s max_opts

"Autochop" harness -O2
SNC’s max_opts

“Autochop” harness

-00
SNC’s max_opt

“Autochop” harness

-00

-02
SNC’s max_opts

“Autochop” harness

-O0

-O2
SNC’s max_opts

“Autochop” harness

-00

-02
SNC’s max_opts

“Autochop” harness

We’ve found the game source file with the bad transformation
SNC’s \texttt{max\_opts} \quad \texttt{-Xmax\_opts=2048} \quad \text{“Autochop” harness}
SNC's `max_opts`

```
-Xmax_opts=1024
```
“Autochop” harness

SNC’s max_opts

-Xmax_opts=512
“Autochop” harness

SNC’s max_opts

-Xmax_opts=988
SNC’s max_opts

“Autochop” harness

-Xmax_opts=989

Now we’ve found the specific transformation causing the miscompile
A compiler trace shows us the culprit optimization.
Narrowed down to a single difference in IR

Just a single line change in the IR
A question for the community:

Would this work in LLVM/Clang?
(even if just at pass level)
The release process
End-user documentation is lacking

Release notes aimed at Clang/LLVM developers, not users
We plan to contribute our documentation improvements to the community.
We plan to contribute our documentation improvements to the community.

```
clide <x86intrin.h>
_mm256_mm256_round_ps(__m256 v, const int m);
```

**SYNOPSIS**

- __m256_mm256_round_ps(__m256 v, const int m);

**DESCRIPTION**

Rounds the values stored in a packed 256-bit vector [8 x float] as specified by the byte operand. The source values are rounded to integer values and returned as floating point values.

**PARAMETERS**

- **v**
  - A 256-bit vector of [8 x float] values.

- **m**
  - An immediate byte operand specifying how the rounding is to be performed.

  - Bits [7:4] are reserved.
    - Bit [3] is a precision exception value:
      - 0: A normal PE exception is used
      - 1: The PE field is not updated
    - Bit [2] is a rounding control source:
      - 0: MXCSR:RC
      - 1: Use the RC field value
    - Bit [1:0] contain the rounding control definition:
      - 00: Nearest
      - 01: Downward (toward negative infinity)
      - 10: Upward (toward positive infinity)
      - 11: Truncated

**RETURNS**

- A 256-bit vector of [8 x float] containing the rounded values.
Forward compatibility
ABI

v3.x

v4.x

System libraries

PlayStation
Maintaining a stable ABI is a MUST (including maintaining existing ABI bugs)
We have created a full IA64 ABI test suite
We hope to **contribute** our test suite to the **community**

(some logistics still to be worked out)
Developer reaction
Paul T. Robinson  
Sony Computer Entertainment  
LLVM Dev Meeting, 7 Nov 2013

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

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
But...
Most requested feature by an order of magnitude and already supported by all the other major compilers
This is the most common use-case:

```c
void CriticalToPerformance() {
    ....
}
```

```c
void MaybeHasABugInIt() {
    ....
}
```

```c
void AlsoCriticalToPerformance() {
    ....
}
```

Game runs too slowly at `-O0`, but is very hard to debug at `-O2`


Solution: use a pragma to selectively disable optimization on a small set of functions to be debugged.

We proposed this on the mailing lists, but it is a major change and we got a limited response.
#pragma optimize

void CriticalToPerformance() {
}

__attribute__((optnone))
void MaybeHasABugInIt() {
}

void AlsoCriticalToPerformance() {
}
Many of our users abstract this away behind a compiler-independent interface. Function attribute does not fit this model!

We still need a range-based solution
In summary
Our initial experience with Clang and LLVM has been very positive.

Thanks to all of you who helped make Clang and LLVM great!
There are still improvements that can be made...

We will be working alongside you to make them