PHP and HHVM

- HHVM has played a valuable role in the evolution of PHP
- Grew from an initial PHP to C++ compiler
- Released in 2011 as the Hip Hop Virtual Machine
- A just-in-time compiler generating machine code from bytecode
- Saw incentive in achieving parity and delivering compatible PHP 5 results
- Together with HHVM, Facebook also released Hack, which is a programming language specifically tailored for HHVM
- In 2017, the HHVM team announced a parting of ways recognizing the growth of PHP7 and continuing their own development in support of Hack
The HHVM JIT is broken up into four components:
- Type specialization
- Profile-guided optimizations
- Side exits
- Region-based compilation

Type specialization relates to obtaining type information either dynamically or statically and the granularity by which they are specialized.

Profile-guided optimizations are a class of optimizations that can be either enabled or enhanced by the use of runtime information.

Side-exits are uncommon traps which give the JIT the ability to leave a compilation unit and transfer control elsewhere.

Code generation in HHVM is based on regions (region-based compilation). These are not restricted to entire methods, single basic blocks, or straightline traces.
The goal is to provide a benchmark suite, testing something representative of real-world situations.

The main suite configures and runs nginx, siege, and PHP5/PHP7/HVVM over FastCGI, over a TCP socket. Configuration is as close to identical as possible.

The script will run 300 warmup requests, then as many requests as possible in 1 minute. Statistics are only collected for the second set of data.

Generates reliable synthetic benchmarks for purposes of profiling popular PHP5 compatible apps such as Drupal and MediaWiki using either the PHP runtime or HHVM.
Siege

- Siege is an open source regression test and benchmark utility. It can stress test a single URL with a user defined number of simulated users, or it can read many URLs into memory and stress them simultaneously.
- One can run either dynamic or cached tests
- It runs its test from only one server
- Runs from the command line, relatively painless setup and install
- The program reports the total number of hits recorded, bytes transferred, response time, concurrency, and return status.
- A transaction is characterized by the server opening a socket for the client, handling a request, serving data over the wire and closing the socket upon completion.
Optimization focus

- Initial focus on OS and tools
- Added PGO
- Enabled USE_LOWPTR:
  - This allows us to compact objects with members that are guaranteed to point to low memory.
  - Some other optimizations are tied to enabling USE_LOWPTR as well.
- Enabled Hardware CRC through our compiler and via CMake flag
- Set-up pre-run optimizations
  - Flush caches
  - Give the kernel the ability to reuse TCP ports which may be in a TIME_WAIT state
  - Disable ASLR, i.e., kernel randomization of va space
  - Set cpufreq to performance
  - Etc.
Optimization focus

- Look at JEMalloc and current configuration options
- Evaluate 64KB pages with enabled Transparent Huge Page support
- Investigate memset/memcpy optimizations
- Evaluate Cache performance
- Consider branching and address range limitations
Profiling

- Enabled TC-Print for annotated perf collection
- Used perf to capture key performance counters
- Made use of flamegraphs in combination with perf
Observations

- PHP-Zend lacked support for Aarch64 GCC Global variables. Where GCC Global variables help is that they can be substituted for arguments to the handler by storing them in global registers.
- Limitations on immediate moves into a register, three instructions versus one for example
  - ARMv8
    - movz x0, #0xe8c8
    - movk x0, #0x92e3, lsl #16
    - movk x0, #0x3ff, lsl #32
  - x86
    - mov $0x7f5caa293f08, %rax
Observations

● Limitations on conditional branch address ranges
  ○ Unconditional simple relative branches can branch backward or forward up to 128MB from the current program counter location.
  ○ Conditional simple relative branches, where a condition code is appended, have a smaller range of ±1MB.
  ○ That can be costly when the JIT is trying to move code blocks. In such a case, preventing the code splitting between hot/cold makes sense.
  ○ With x86, when hot/cold code splitting occurs, that relevant short-displacement conditional branch instructions are changed to larger displacement conditional branch instructions

● Cache topology and size
Discussion
Thank you

- skelley@amperecomputing.com