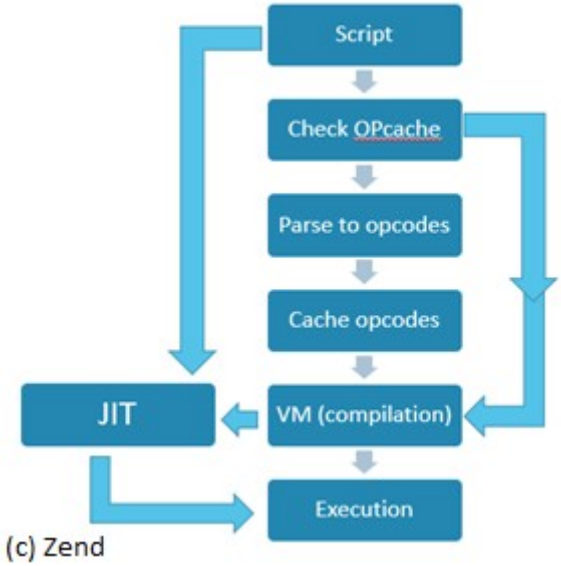
بررسی عمیق PHP JIT

یکی از مهمترین ویژگی‌های جدید در PHP 8.0، کامپایلر JIT است. JIT می‌تواند با کامپایل کردن و ذخیره کردن کامل یا بخش‌هایی از اسکریپت PHP، به **کد ماشین CPU**، عملکرد را بسیار بهبود بخشد و به طور مستقیم کد ماشین را اجرا کند، بطوریکه Zend VM و سربار ناشی از عملیات و فرایندهای آنرا دور می‌زند و نادیده می‌گیرد.



JIT ترکیبی از مفسران traditional و کامپایلرهای **AoT** است. این مدل ترکیبی/Hybrid، مزایا و معایب هر دو مفسر و کامپایلر را به ارمغان می‌آورد.
پیاده‌سازی PHP JIT تنها با تلاش‌های شگفت‌انگیز Dmitry Stogov در چند سال اخیر، ارزش بحث، اجرا و آزمایشات را پیدا کرده است.

این مقاله در مورد benchmark ها، نحوه‌ی کار JIT، امکانات و گزینه‌های پیکربندی در php.ini می باشد.

اغلب اپلیکیشن‌های تحت PHP، درخواست‌های HTTP را گرفته، داده‌ها را از database بازیابی و پردازش کرده، و نتیجه‌ی را گزارش می‌کنند. معمولاً محدودیت‌ها و bottleneck های مهم، مربوط به عملیات I/O هستند که شامل خواندن/نوشتن داده‌ها از disk و پاسخ به درخواست‌های شبکه است.
PHP 8.0، ویژگی JIT را به عنوان گام بعدی برای بهبود عملکرد اپلیکیشن‌های PHP معرفی می‌کند، اما مانع مهمی را هم در debugging می‌افزاید، زیرا برخی از قسمت‌های اپلیکیشن ممکن است بصورت **کد ماشین CPU** ذخیره شوند که debugger های استاندارد PHP نمی‌توانند با آن کار کنند! **این JIT Pull-request** در مخزن PHP 8.0، بالغ بر 50 هزار خط جدید را به کد موتور PHP افزوده است بطوریکه خود توسعه‌دهندگان اصلی PHP، جدای از کسانی که با JIT کار می‌کنند، ممکن است به خوبی در آن تبحر نداشته باشند...

PHP VM

کد PHP، زمانی که پردازش می‌شود (Tokenize، Parse، تولید AST، تولید OpCodes)، روی ماشین مجازی Zend اجرا می‌شود. این ماشین مجازی Zend همانند Java و JavaScript، بخش ارتباطات سخت‌افزاری اپلیکیشن را ساده‌سازی و شبیه سازی می‌کند، که «اجرای» سورس کدهای PHP را بدون کامپایل کردن فراهم می‌سازد.
اکستنشن OpCache می‌تواند به ذخیره‌ی OpCodeها در Shared memory کمک کند، تا PHP مراحل تکراری قطعه‌بندی/تجزیه/ایجاد OpCode را دور زده و نادیده بگیرد.
PHP چندین عملیات بهینه‌سازی مانند حذف کدهای dead در سطح OpCode را انجام میدهد، اما انجام بهینه‌سازی‌ها فراتر از سطح ماشین مجازی Zend امکان‌پذیر نیست، زیرا در آن نقطه، ماشین مجازی Zend کد را تفسیر و کامپایل میکند.

سپردن به دیگر اپلیکیشن ها

اکستنشن GD همان نامی باشد که به نظر آشنا می‌آید. اگر قرار بود PHP تصاویر را در همان سطح Vector یا Bitmap دستکاری و استفاده کند، به دلیل وجود لایه‌ی ماشین مجازی در PHP، بسیار کند عمل می‌کرد. اکستنشن GD که باینری‌های کامپایل شده را به کار می‌گیرد، می‌تواند از instruction های پیشرفته‌ی CPU برای اجرای همین فعالیت‌ها و اقدامات بهره ببرد.
PHP 7.4 یک Foreign Functions Interface را معرفی کرده که یک رابط یکپارچه برای به کارگیری اپلیکیشن‌های مختلف را بدون نیاز به تغییر و توسعه اکستنشن PHP فراهم می‌کند. از اینرو یکپارچه سازی زبان‌های سنتی کامپایلی، مانند C و Rust به همراه PHP، به لطف FFI امکان پذیر است.

کامپایل کردن کد PHP

گام بعدی نزدیک شدن به سطح CPU، نادیده‌گرفتن و حذف ماشین مجازی Zend است، و این همان کاری است که JIT انجام می‌دهد!
کامپایلر JIT یک ویژگی است که زبان JavaScript آن را چندین سال قبل، به طور موفقیت آمیز با موتور Chromium V8 به کار گرفته بود. همچنین، دیگر زبان‌ها نیز به هر طریقی که شد JIT را به کار گرفته و پیاده‌سازی کردند. بزرگ‌ترین مزیت JIT این است که سورس کد نیاز به pre-compile کردن ندارد، اما با Shared cache ای که حاوی Machine code کامپایل‌شده باشد، زبان می‌تواند سورس کد را trigger کرده تا با Machine code کامپایل‌شده اجرا شود، یا برای بعد کامپایل شود، و یا بدون JIT اجرا شود.

LLVM

LLVM یک toolkit کامپایلر محبوب است که امروزه برای توسعه‌ی کامپایلرها به اکثر زبان‌های AoT کمک می‌کند. بسترهای مورد هدف LLVM شامل معماری‌های x86-64، x86 و چندین نوع دیگر، از جمله پردازنده‌های گرافیکی، web assemblers، ARM و غیره است.

نکته اینکه PHP استفاده از LLVM را مدنظر داشته است، اما به دلیل سرعت نامطلوب کامپایلر، چندان برایش مفید نبوده است...

DynASM

DymASM از پروژه Lua JIT، برای JIT پی اج پی بسیار سریع بوده است. با اینکه پشتیبانی DymASM از instruction های CPU در مقایسه با LLVM محدودتر است، اما از instruction های x86 و x86-64 پشتیبانی می‌کند که برای زبانهای برنامه‌نویسی server-side، مانند PHP رایج است.
نکته اینکه موتور JIT ای که در PHP 8.0 پیاده‌سازی و استفاده شده، DynASM می باشد.

PHP JIT چگونه کار می‌کند

PHP JIT در بخشی از OpCache پیاده‌سازی شده است. این بخش، JIT را از موتور PHP جدا کرده است.
در خود JIT، سه کامپوننت ذخیره‌سازی کد، بررسی کد و invoke کد وجود دارد که این‌ها کد را با ماشین مجازی استفاده می کنند. و یا حتی مستقیم از machine code ذخیره‌شده در بافر(buffer) استفاده می‌کند.

Buffer

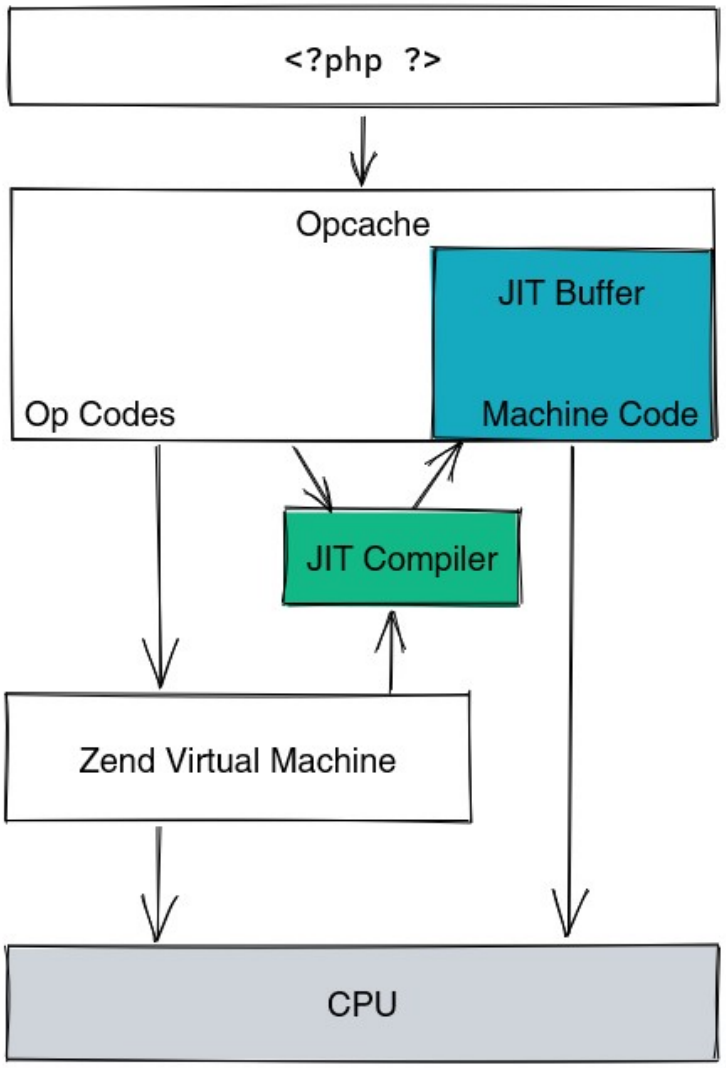
JIT Buffer حافظه ای است که machine code کامپایل‌شده، داخل آن ذخیره شده است. PHP، گزینه‌هایی را برای تنظیم ظرفیت این حافظه (opcache.jit_buffer_size) برای JIT Buffer فراهم کرده است.

Triggers

Triggerها مسئول ایجاد و به کارگیری machine code کامپایل‌شده در زمانی هستند که با یک کد structure برخورد می‌کنند. این Triggerها می‌توانند یک loop ،function call entry و یا غیره باشند.

Tracer

JIT Tracer، کد را پیش از اجرا، حین اجرا و بعد از اجرا مورد بررسی قرار می‌دهد و مشخص می‌کند که کدام کد «Hot» است و کدام structure را می‌توان با JIT کامپایل کرد.
Tracer همچنین می‌تواند کد را در هنگام اجرا هم کامپایل کند، که این امر نیز قابل تنظیم است.



Tracing JIT و Function JIT

PHP 8.0 دو حالت برای عملیات JIT اضافه کرده است. این حالات نیز قابل customize است، اما برجسته‌ترین نوع کارآیی و قابلیت JIT با عنوان function و tracing مشخص می‌شود.

Function JIT

حالت Function JIT در مقایسه با حالت Tracing JIT، نسبتاً ساده است. این حالت، کل تابع را بدون Tracing برای کد structure استفاده شده مانند loopها در یک تابع کامپایل می‌کند. این حالت هنوز از سیستم profiling برای عملکردهای تکراری استفاده می‌کند و بعد شروع به کامپایل JIT یا اجرای machine code کامپایل شده می‌کند. حال پیش از اجرای برنامه، در حین آن، و یا بعد از آن.

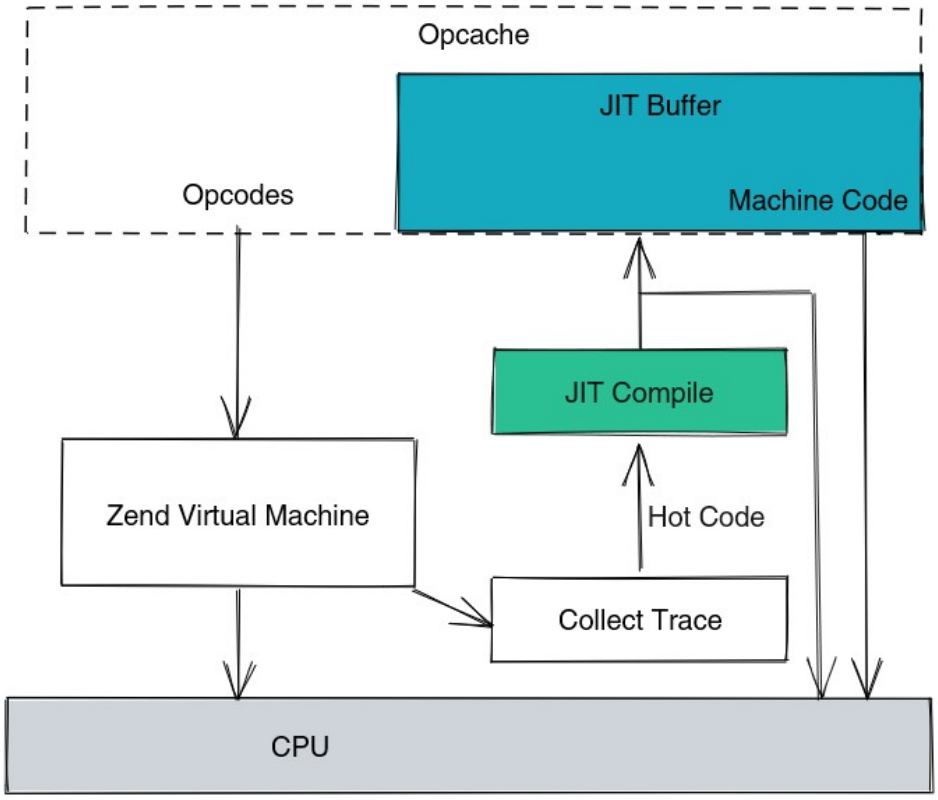
Tracing JIT

Tracing JIT که به صورت پیش فرض در PHP 8.0 انتخاب شده است، سعی می‌کند که قسمت های مورد استفاده و مکرر کد را شناسایی کرده و به طور selectively آن ساختارها را برای بالانس و تعادل بخشد. نکته اینکه همه‌ی زبان‌های برنامه‌نویسی از کامپایلرهای Tracing JIT پشتیبانی نمی‌کنند، اما PHP درست پس از اولین انتشار، از Tracing JIT پشتیبانی کرده و آن را به طور پیش فرض انتخاب کرده است.

امکانات و گزینه‌های تنظیم زیادی وجود دارد که باعث تغییراتی در نحوه‌ی تعیین ساختار کد «Hot»، مانند تعداد function callها، تعداد تکرارهای loop، ساختار و غیره می‌شوند.

پروفایلینگ و بهینه‌سازی

JIT می‌تواند کد را از زمان آغاز اجرای کد بررسی، پروفایل‌سازی و بهینه‌سازی کند. و کنترل دقیقی را بر triggerها داشته باشد.



کدهای JIT-friendly

JIT در زمانی که دستورالعمل‌ها و رجیسترهای CPU را offload میکند، مزایای زیادی را به ارمغان می‌آورد. PHP یک زبان weakly typed است که تشخیص نوع متغیر را برای موتور PHP دشوار می‌سازد پس نیازمند تجزیه و تحلیل بیشتری در چرخه‌ی عمر یک متغیر می باشد، زیرا نوع یک متغیر ممکن است در هر لحظه تغییر کند!
اما اسکریپت حاوی عبارت strict_types=1 و توابع نوع scalar می‌تواند به JIT برای تشخیص و استفاده از instructionها و رجیسترهای CPU در هر زمان کمک کند. برای مثال، یک تابع ساده (که هیچ side-effect ای ندارد) به همراه عبارت strict_types=1 و تعیین نوع پارامتر و تعیین نوع برگشتی، یک مورد بی‌عیب و نقص می باشد:

```
declare(strict_types=1);

function sum(float $a, float $b): float
{
   return $a + $b;
}
```

اما زمانی که PHP نتواند type ها را تشخصی دهد، ممکن است قادر نباشد که از بهینه‌سازی‌های JIT بهترین استفاده را ببرد!
در واقع برخی از بهبودها در PHP 7، از این بهینه‌سازی‌ها نشأت می‌گیرد که می‌تواند کد dead را حذف کرده و reference counting را بهبود بخشد. این بدان معنا است که کدهای strictly typed، فرصت‌های بیشتری برای PHP فراهم می‌کند تا کد را در سطح OpCache و در سطح JIT بهینه‌سازی کند.
برنامه‌هایی که I/O – bound هستند، از جمله آنهایی که زیاد از پایگاه داده، DNS queries، عملیات‌های write/read فایل، sockets ،FTP و غیره استفاده می‌کنند، ممکن است تفاوت جدی را مشاهده نکنند، زیرا معمولاً عملیات‌های I/O، خودشان محدودیت‌ها و bottleneck های چنین برنامه‌هایی هستند.

پیکربندی/تنظیم JIT

به طور پیش‌فرض، همیشه JIT فعال است اما با محدود کردن اندازه‌ی buffer، غیرفعال می‌شود.
PHP JIT: پایه و اساس
برای بررسی JIT در PHP 8.0 و امکانات و گزینه‌های پیکربندی، لینک **PHP 8.0: JIT** را ببینید.
این مقاله در مورد معیارها، نحوه‌ی کار JIT و امکانات و گزینه‌های پیکربندی ایده‌آل است.
ساده‌ترین تنظیم این است که اندازه‌ی بافر برای JIT تنظیم شود و JIT از تنظیمات پیش‌فرض و معقول استفاده خواهد کرد.

```
opcache.enable=1
opcache.enable_cli=1
opcache.jit_buffer_size=256M
```

در اینجا 256MB برای بافر JIT تخصیص داده می‌شود و JIT بر روی اپلیکیشن‌های CLI نیز فعال می‌شود.
دستور opcache.jit امکان اصلاح دقیق عملکرد و تابع‌پذیری JIT را امکان‌پذیر می‌کند.

```
opcache.jit=tracing
```

عبارت opcode.jit تا حدی یک پیکربندی پیچیده است. این دستور مقادیر disable، on، off، trace، function و مقدار 4 رقمی (نه از bitmask) از 4 پارامتر flags مختلف را به ترتیب می‌پذیرد.

- Disable: به طور کامل ویژگی JIT را در زمان آغاز از کار می‌اندازد و در زمان اجرا نمی‌تواند فعال شود.
- Off: از کار می‌اندازد اما فعالسازی JIT در زمان اجرا امکان‌پذیر است.
- On: حالت tracing را فعال می‌کند.
- Tracing: یک نام مستعار برای **پیکربندی دقیق** 1254.
- Function: یک نام مستعار برای **پیکربندی دقیق** 1205.

PHP JIT، نام‌های Tracing یا Function را به عنوان یک پیکربندی آسان می‌پذیرد که ترکیب از پیکربندی را نشان می دهد.
علاوه بر نام‌های مستعار Tracing و Function، عبارت opcache.jit مقدار پیکربندی 4 رقمی را نیز می‌پذیرد و همچنین می‌تواند رفتار JIT را پیکربندی کند.
مقدار پیکربندی 4 رقمی، به شکل CRTO است که در آن، هر موقعیت و جایگاه، به یک مقدار تک رقمی اجازه‌ی flag تعیین‌شده توسط یک حرف را می‌دهد.

JIT Flags

عبارت opcache.jit مقدار 4 رقمی را برای کنترل رفتار JIT، به شکل CRTO پذیرفته و همچنین مقادیر زیر را برای جایگاه‌های C، R، T و O می‌پذیرد.

CPU-specific Optimization Flags:

- 0: بهینه‌سازی CPU را غیرفعال می‌کند.
- 1: استفاده از AVX را در صورتی فعال می‌کند که CPU از آن پشتیبانی کند.

تخصیص**R**egister :

- 0: تخصیص register را انجام نمی‌دهد.
- 1: تخصیص block-local register را انجام می‌دهد.
- 2: تخصیص global register را انجام می‌دهد.

Trigger:

- 0: کامپایل تمامی توابع بر روی script load.
- 1: کامپایل تمامی توابع بر روی اجرای اول.
- 2: پروفایل کردن درخواست اول و سپس کامپایل جذاب‌ترین توابع.
- 3: پروفایل کردن در زمان اجرا و کامپایل توابع hot.
- 4: در حال حاضر بدون استفاده.

- 5: استفاده از tracing JIT. پروفایل کردن در زمان اجرا و کامپایل trace ها برای بخش‌های کد hot.

سطح **O**ptimization:

- 0: بدون JIT.
- 1: Minimal JIT (با نام VM handlers استاندارد)
- 2: VM handlers برخط
- 3: استفاده از type inference
- 4: استفاده از call graph.
- 5: بهینه‌سازی اسکریپت کامل.

نکته اینکه گزینه‌ی چهارم در زیرشاخه‌ی Triggers (T=4) به نسخه‌ی نهایی پیاده‌سازی JIT نرسیده است. Trigger JIT در توابع از طریق قسمت کامنت @jit DocBlock این مورد را اعلام کرده است. *این مورد اکنون بدون استفاده است.*

هم Tracing و هم Function در پیکربندی JIT از مجموعه دستورالعمل‌های CPU و تخصیص register بهره می‌برند تا از بیشتر قابلیت‌های CPU استفاده کنند (C=1, R=2).

opcache.jit=function

function یک نام مستعار برای T=0، R=2، C=1 و O=5 است.
تفاوت در مقدار function این است که دنبال کامپایل کردن اسکریپت به سریع‌ترین زمان ممکن است، و اسکریپت را کاملاً کامپایل می‌کند. این امر یک رویکرد بسیار جسورانه و خودسرانه است، و بسیار مشابه با pre-load فایل‌های PHP برای Opcache، به همراه ویژگی pre-load در PHP 7.4 است.

opcache.jit=tracing

tracing یک نام مستعار برای T=5، R=2، C=1 و O=4 است.
با فعالسازی JIT ،tracing می‌تواند دقیق‌تر عمل کند و بخش‌های کد را در یک تابع برای کامپایل کردن انتخاب کند. نامزدهای ایده‌آل، ساختارهای looping و توابعی خواهد بود که اغلب فراخوانده می‌شوند.
این همان پیکربندی پیش‌فرض است که می‌تواند تعادل و بالانس بیشتری را بین مزایای عملکردی و سربار کامپایل کردن فراهم کند.
کارآمدی و قابلیت JIT Tracing (T=2, 3, or 5) امکان تنظیم بیشتری را فراهم می‌کند چرا که تعدادی فراخوانی برای یک تابع انجام می‌گیرد تا به عنوان hot نشانه گذاری شود و سپس نهایتاً در JIT کامپایل می‌شود.

دستور	توضیح	مقدار پیش‌فرض
opcache.jit_hot_loop	پس از تعدادی تکرار، loop به عنوان hot در نظر گرفته می‌شود.	64

opcache.jit_hot_func	پس از تعدادی فراخوانی، یک تابع به عنوان hot در نظر گرفته می‌شود.	127
opcache.jit_hot_return	پس از تعدادی بازگشت، یک بازگشت به عنوان hot در نظر گرفته می‌شود.	8
opcache.jit_hot_side_exit	پس از تعدادی خروج، خروجی کناری به عنوان hot در نظر گرفته می‌شود.	8

مقادیر پیش‌فرض ممکن است تقریباً برای تمامی برنامه‌ها، مناسب‌ترین مقدار باشند، و کاهش آنها منجر به کامپایل‌شدن بیشتر ساختارهای کد شود، چرا که آستانه را نیز کاهش می‌دهند.

پیکربندی JIT ایده‌آل

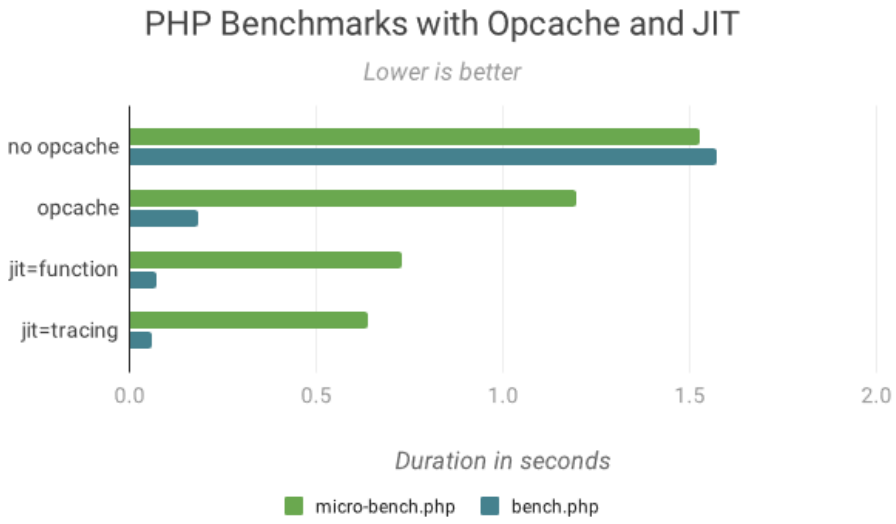
کد JIT کامپایل‌شده‌ی بیشتر لزوماً به معنای سریع‌تر شدن اپلیکیشن نمی‌باشد (همانطور که در بنچمارک‌های وب اپلیکیشن در زیر دیده می‌شود). سربار کامپایل به همراه بافر کوچک‌تر می‌تواند سرعت اپلیکیشن‌ها را کندتر کند، که دلیل آن زمان صرف شده در مراحل کامپایل JIT است.
مقدار تنظیم opcache.jit بهتر است دست نخورده باقی بماند (مقدار پیش‌فرض همان tracing است) چرا که در حال حاضر تعادل و بالانس خوبی از کارآیی CPU و حافظه فراهم می‌کند و پیگیری اینکه کدام ساختار کد باید کامپایل شود را انجام می‌دهد.
JIT هیچ مزایای عملکردی و معناداری را از اپلیکیشن‌های به شدت I/O-bound کسب نخواهد کرد. امروزه اکثر وب اپلیکیشن‌ها در واقع به شدت I/O-bound هستند، که در آن JIT تفاوتی را رقم نمی‌زند، چه برسد که بخواهد تفاوت مثبتی را ایجاد کند.
برای اندازه‌ی بافر، توجه داشته باشید که حافظه‌ی کمی نداشته باشید، که می‌تواند کد کامپایل‌شده‌ی JIT را از بین برده و باعث کامپایل مجدد و مکرر می‌شود. البته حافظه‌ی بسیار بزرگ هم می‌تواند زیاده روی باشد. مقدار 50 تا 100 درصدی حافظه‌ی اشتراکی و فعلی Opcache برای opcode ممکن است مقدار ایده‌آل برای تنظیم opcache.jit_buffer_size باشد.

بنچمارک‌های JIT

تمامی تست‌ها و آزمایش‌های زیر بر روی سیستم x86-64 با 8 هسته‌ی 16 thread ای انجام شدند. با این حال، این آزمایش‌ها هرگز از اعداد صحیحی که نیاز به رجیسترهای 64 بیتی دارند، استفاده نمی‌کنند تا تست‌ها را به CPUهای x64 بیشتر مرتبط و نزدیک کنند.

بنچمارک PHP Script

سورس PHP شامل دو اسکریپت بنچمارک است که کارآمدی PHP را آزمایش می‌کند. فایل‌های micro_bench.php و bench.php در شاخه‌ی PHP 8.0 قرار داده شده تا مورد آزمایش قرار گیرد.

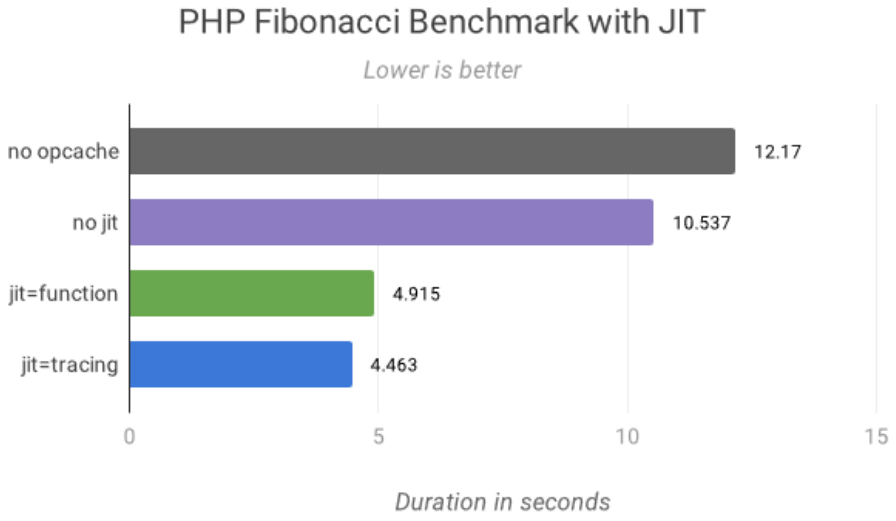


اولین آزمایش با غیرفعال بودن کامل OpCache انجام شد و آزمایش دوم با غیرفعال JIT و فعال بودن OpCache انجام شد.
هر دو حالت JIT دستاوردهای عملکردی قابل‌توجهی را به ارمغان می‌آورند و با حالت tracing کمی این دستاوردها بیشتر است.

این بنچمارک به سختی برنامه‌ی کاربردی واقعی PHP را نشان می‌دهد. فراخوانی‌های مکرر برای تابع یکسان، برای JIT مزایایی را به ارمغان می‌آورد.

بنچمارک PHP Fibonacci

تابع ساده‌ی فیبوناچی(Fibonacci)، چهل‌ودومین عدد از دنباله فیبوناچی را محاسبه می‌کند.

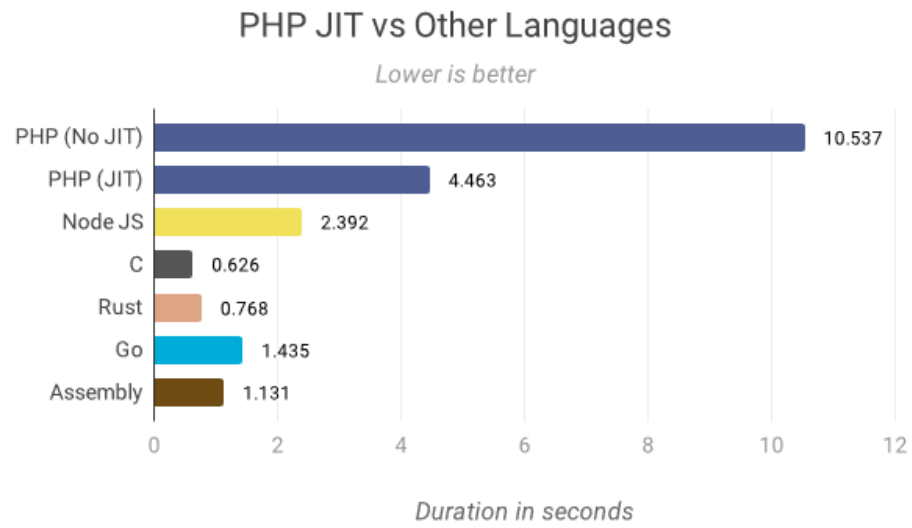


دنباله‌ی فیبوناچی تماماً در مورد فراخوانی تابع مکرر است و داستان کاملی از اپلیکیشن واقعی PHP ارائه نمی‌کند، مگر اینکه یک برنامه‌ی ماشین‌حساب فیبوناچی باشد.

فیبوناچی: PHP در برابر دیگر زبان‌ها

همان تست و آزمایش فیبوناچی(42) در کنار دیگر زبان‌های کامپایل‌شده (از جمله Rust ،GO و C) و NodeJS قرار داده شده است، که ویژگی JIT را نیز دارند.

ویژگی JIT از PHP 8.0 تلاش نمی‌کند که بهینه‌سازی‌هایی انجام دهد که دیگر زبان‌های کامپایل‌شده AoT می‌توانند آن را انجام دهند. اما JIT پیشرفت عملکرد قابل‌توجهی را به همراه دارد و هنوز راه زیادی برای بهترشدن در پیش دارد.

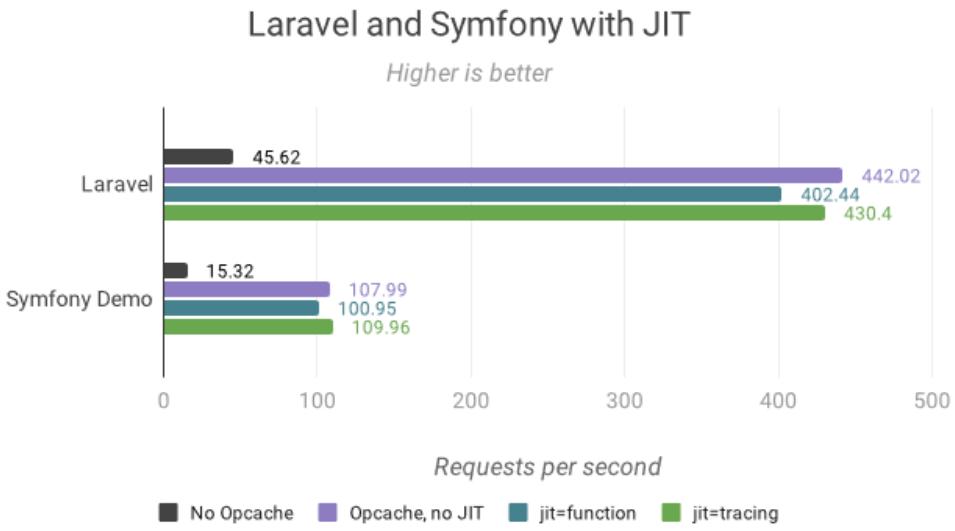


بنچمارک Web Application

پیش‌بینی تاثیر JIT دشوار است، زیرا JIT به شدت به حجم کاری بستگی دارد. اکثر نمونه‌های زیر، نمونه‌های hello-world از framework های وب است که لزوماً کاربرد واقعی را به دلیل پلاگین‌های مختلف و ذخیره‌سازی سیستم‌های درگیر نشان نمی‌دهد.

اپلیکیشن‌هایی که از ارتباط database استفاده می‌کنند، به احتمال زیاد در جستجوی database، بزرگترین محدودیت‌ها و تنگناها را خواهند داشت. در یک تست سرور وب که درخواست به ازای هر ثانیه اندازه‌گیری می‌شود، سربارهای HTTP ،TLS و FPM ممکن است نسبت بر تفاوت‌های عملکردی JIT غلبه کند.

فریم ورک Laravel (8.4.4) و Symfony (نسخه‌ی 1.6.3 با استفاده از اجزا و مولفه‌های نسخه‌ی 5.1.8) به همراه طرح کلی اپلیکیشن‌ها، بر روی همان سخت‌افزار و به همان سناریوی سابق در بنچمارک‌های قبلی مورد آزمایش و تست قرار گرفت. هر دو اپلیکیشن‌ها با وب سرور داخلی PHP مورد استفاده قرار گرفتند و با استفاده از Apache Bench (ab) مورد ارزیابی قرار گرفتند که 100 درخواست داشتند.



هر دو اپلیکیشن مزایای قابل‌توجهی را به دست نیاوردند، و در فریم ورک Larval، عملکرد حدود 2 درصد بدتر از JIT بود که به احتمال زیاد، ناشی از سربار کامپایل‌کردن بوده است که تاثیری نداشته است.

بنچمارک هر اپلیکیشن

برای داشتن مزایای عملکردی واقعی، هر اپلیکیشن باید تحت یک بنچمارک یا معیاری قرار بگیرد تا مشخص شود که در صورت استفاده از JIT، می‌تواند مزایای قابل توجهی را به دست آورد یا خیر.
اپلیکیشن‌های تحت CLI، و به خصوص اپلیکیشن‌های CPU intensive، به احتمال زیاد بهبود عملکرد قابل‌توجهی را به دست خواهند آورد.
برای اپلیکیشن‌های شبکه‌ای و file – intensive همانند Composer و PHPUnit، به احتمال زیاد شاهد افزایش عملکرد نخواهیم بود چرا که از بهبود کد ماشین سود و مزایایی را به دست نمی‌آورند. برای کسب نتایج بهتر، ظرفیت SSD/RAM و پهنای باند بیشتری را چاشنی کار کنید.

دیدگاه‌های پایانی

JIT یک گام بزرگ در سریع‌تر کردن اجرا و عملکرد PHP است و از قابلیت‌های زیرساختی سخت‌افزاری بهره می‌برد. JIT حاصل سال‌ها تلاش است و در حال حاضر پیشرفت های قابل‌توجهی را در محاسبات حجیم به نمایش می‌گذارد.
هنوز راهی برای پیشرفت و بهبود JIT وجود دارد و به احتمال زیاد، بهتر هم عمل خواهد کرد.
از تلاش‌ها و کارهای شگفت‌انگیز Dmitry Stogov و Nikita Popov در خصوص JIT، بسیار متشکریم. نیکیتا پوپوف با افتخار بخش اول این مقاله(انگلیسی) در خصوص نحوه‌ی کار JIT را بررسی و ویرایش کرده است. از او متشکرم.

مترجم: یوشا آل ایوب

منبع: https://php.watch/articles/jit-in-depth