来る新 JIT エンジンについて知った気になる

nsfisis (いまむら)

PHP カンファレンス小田原 2024

自己紹介

nsfisis (いまむら) 

@ デジタルサーカス株式会社

普通の PHP プログラム

PHP スクリプト

opcode (VM で実行)

JIT コンパイルとは

- Just In Time (ちょうど間に合って)
- 広義 : 実行時にコンパイルする
- 狭義 : 実行時に機械語へコンパイルする
- PHP 8.0 で導入

PHP + JIT コンパイル

PHP スクリプト

opcode (VM で実行)

機械語 (CPU で実行)

JIT コンパイルの難点

- コンパイルに時間を割けない
  - コンパイルに 1 分かかるので実行は待ってというわけにはいかない
  - コンパイルすればするほど速くなるわけではない
- すべての情報が事前に確定しない
  - 動的言語。変数が未定義かも、型が想定と違うかも
  - `$a + $b` が常に int と int の足し算とは限らない

Tracing JIT

何度も繰り返し実行される部分、何度も通る分岐などを特定し、そこだけピンポイントで JIT コンパイルする

INI で `opcache.jit=tracing` (デフォルト) にするとこのモードになる

Tracing JIT

- コンパイルに時間を割けない
  - 繰り返し実行される箇所だけコンパイルする
  - これまで何度も実行されたのだから、これからも何度も実行されるはず
- すべての情報が事前に確定しない
  - 実際の実行パターンを元に、そのケースに最適化させた機械語を生成する
  - `$a + $b` が int と int だったときに限定した機械語を生成し、想定と違っていれば従来の opcode を VM で動かす

PHP 8.4 での変更

PHP 8.4 では、JIT コンパイルの仕組みに大きな変更が入る

PHP 8.4 での変更

IR (Intermediate Representation、中間表現) の導入

PHP スクリプト

opcode (VM で実行)

IR (中間表現)

機械語 (CPU で実行)

IR 導入のモチベーション

- opcode から直接の変換だと制限が大きい
  - 最適化しづらい
    - opcode の表現に制約を受ける
  - CPU アーキテクチャごとに別々の実装を抱える
- PHP と密結合
  - 誰もメンテナンスできない

IR の特徴

- より強力な最適化
  - SSA + CFG から Sea-of-Nodes へ
- PHP に依存しない

実際どうなった？

TODO: ベンチマークのスクリーンショットを引用

実際どうなった？

- 生成された機械語の速度 : 0-5
- Tracing JIT のコンパイルにかかる時間 : ほぼ同等
- Function JIT のコンパイルにかかる時間 : 4 倍ほど遅い

実際どうなった？

8.3.4 と master を比較

- `ext/opcache/jit` の変更量
  - 5.9 万行追加、4.9 万行削除
- アーキテクチャごとのコード生成部
  - x86: 1.1 万行追加、1.6 万行削除
  - arm64: 0.65 万行追加、1.1 万行削除

知った気になる

知った気になる

8.4 で変わる JIT

opcode と機械語の間に中間表現 (IR) が導入され、
より最適化がかけられるように