BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Překlad podmnožiny jazyka PHP do C++

Stanislav Nechutný*

Abstrakt

Tato práce se zaměřuje na návrh a tvorbu nástroje pro automatizovaný překlad funkcí napsaných v podmnožině jazyka PHP do C++. Vygenerovaný zdrojový kód je možno zkompilovat jako rozšíření PHP a zavést stejným způsobem jako například MySQL, PDO, GD apod. Ve výsledku je tedy možno zavolat tyto funkce z PHP, jako by se jednalo o původní interpretovanou funkci. Předpokladem je však rozdíl v rychlosti vykonávání, protože odpadá analýza zdrojových kódů, jejich interpretace, či režie způsobená správou paměti. Vytvořený nástroj provádí převod zdrojového kódu do abstraktního syntaktického stromu, staticky jej analyzuje pro určení datových typů proměnných, a následně provádí generování C++ kódu. Výsledné zrychlení pak záleží na charakteristice překládaného kódu a použití je komplikované kvůli implementaci podmnožiny PHP.

Klíčová slova: PHP — C++ — Překlad — Rozšíření

Přiložené materiály: Zdrojový kód

* xnechu01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Současné webové aplikace napsané v PHP obsahují desítky knihoven, které programátoři nepotřebují upravovat a chtějí je jen použít. Tyto knihovny někdy provádějí výpočetně náročné operace a jelikož jsou interpretovány, tak dochází k zpomalení v důsledku použitých mechanismů. Příkladem je správa paměti, která může způsobovat výrazné zpomalení. Na tuto skutečnost poukázala optimalizace nástroje Composer, které bylo dosaženo vypnutím automatického uvolňování nepoužívané paměti, a dosáhla 30% - 90% zrychlení (v závislosti na množství prováděných operací)[1]. Dlouhá doba načítání webové stránky je jedním z hlavních důvodů pro opuštění návštěvníkem. Škálování takových náročných systémů pak nemusí být ekonomické řešení. Tento nástroj však umožňuje snížit vytížení serverů s nutností minimální úpravy aplikace.

Překlad zdrojového kódu z dynamicky typovaného jazyka do silně typovaného je problematický hned z několika důvodů. Prvním je detekce datových typů ze zdrojového kódu. Dále tu máme různé obtížně přeložitelné konstrukce, jako například potlačení chybového výpisu, nebo operátor porovnání.

Překlad probíhá následovně: Nejprve je třeba provést syntaktickou analázu zdrojových kódů a převést je do abstraktního syntaktického stromu pro následnou analýzu. Na základě této struktury můžeme provést analýzu možných hodnot, graf volání pro detekci datových typů. Toto řešení však nepokrývá veškeré možnosti, takže v projektu jsem také experimentoval s generováním kódu pro více variant a následnou volbou vykonaného kódu za běhu na základě obdržených hodnot.

V současné době zde není přímo konkurenční řešení. Můžeme porovnat tento projekt s HPHPc od společnosti Facebook, které však bylo zamýšleno pro kompilaci celé PHP aplikace do jednoho velkého binárního souboru. Toto je rozdílný koncept, jelikož HPHPc vyžadovalo překompilování celé aplikace i v případě drobné změny a následnou distribuci až několik GB velkého souboru na servery. Představené řešení je však zaměřeno pouze na kompilaci knihoven, které nejsou měněny často a hlavní aplikace je stále interpretována. Projekt HPHPc byl nahrazen projektem HHVM[1], který je JIT kompilátor (překládající jen použité části kódu) spouštějící bytecode ve virtálním

[1] http://hhvm.com/

stroji, čímž řeší zmíněné nedostatky HPHPc[2].

Dále existuje projekt PHC[2], který také kompiluje PHP aplikaci do spustitelného binárního souboru a má experimentální podporu pro generování rozšíření PHP. Bohužel projekt není již přes 6 let udržován a nepodařilo se mi tuto funkcionalitu otestovat. Problémem byla kompilace programu, která nebyla možná ani na Debian stable s PHP 5.3 a to i přes různé úpravy kódu a instalaci starších verzí knihoven. Ani pokus kontaktovat dva vývojáře z tohoto projektu pro konzultaci mého řešení se bohužel nezdařil.

Většina zbylých řešení ve výsledku pouze zabalí PHP skript do datové části spustitelného souboru a přibalí PHP interpret. Nejedná se tedy de facto o překladač, ale o pouhé zabalení skriptu k interpretru.

Zde zmiňované řešení používá funkce z PHP pro převod zdrojového kódu na tokeny a přidává další vrstvu abstrakce pro přesnější rozlišení tokenů. Pro parsování zdrojových kódů je použit vlastní parser s rekurzivním sestupem a zásobníková precedenční analýza. Vlastní řešení bylo zvoleno pro snadnost úpravy, získání veškerých potřebných údajů v požadovaném formátu a vyzkoušení si teorie v praxi.

Abstraktní syntaktický strom z precedenční analýzy je použit pro zjištění použitých proměnných a operací s nimi prováděných. Dvěma průchody stromu jsou na základě prováděných operací a volaných funkcí přiřazeny proměnným datové typy. Současně s parsováním a precedenční analýzou probíhá také předávání informací generátorům, které mají na starosti výsledný C++ kód.

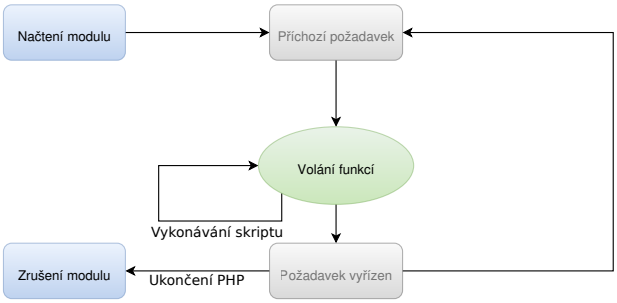
Po úspěšném rozparsování a zanalyzování zdrojového kódu je provedeno generování kódu na základě zjištěných informací,

Nástroj, jehož návrh a implemetace je v tomto článku popisována, je možností, jak dosahnout snadno rychlejší odezvy PHP aplikací. V současné době je praktické použití komplikované implementací pouhé podmnožiny jazyka PHP a zejména absence objektové části je pro překlad moderních knihoven problematická. To však vzhledem k pokračujícímu vývoji nemusí být překážkou nastálo.

2. PHP

Některé knihovny jsou náročné na výpočet a přesto jsou implementovány v PHP, pro jeho jednoduchost na psaní a velký rozsah funkcí, kdy se není třeba zaměřovat na správu paměti, čtení ze souboru apod. Cílem je ponechat tuto jednoduchost a přinést zrychlení.

[2] http://www.phpcompiler.org/



Obrázek 1. Životní cyklus PHP rozšíření.

Příkladem může být rozpoznání obličeje[3], nebo generování PDF z HTML - mPdf.

Optimalizace přepisováním částí systému do C/C++ není zcela běžná, ale používá se u velkých projektů - třeba Yahoo provádí přepisování náročných částí aplikace do C++[3]. Z open source knihoven pak může být příkladem framework Phalcon[4], či Ice, které jsou kompletně napsané v jazyce C. Zpřístupňují MVC model s databázovou vrstvou, šablonami a dalšími funkcemi, jako klasické frameworky implementované v PHP. Dle testů[4] webová aplikace využívající výhod této optimalizace je schopna odbavit násobky příchozích požadavků oproti implementacím v PHP. Je tedy zřejmé, že tato optimalizace má smysl a právě cílem této práce je zjednodušit tvorbu C++ knihoven pro PHP.

Životní cyklus PHP rozšíření sestává z několika stavů.[5] Při startu PHP jsou načteny sdílené knihovny definované v konfiguračním souboru direktivou `extension=` a zavolána funkce vracející informace o rozšíření. Tyto informace obsahují verzi PHP API, callbacky a funkce, třídy, rozhraní a konstanty, které rozšíření poskytuje.[6]

Po inicializaci PHP jsou zavolány inicializační funkce rozšíření, které umožňují provést alokaci paměti, otevření deskriptorů k souborům apod. V tomto kroku je například vhodné, aby rozšíření pro logování otevřelo soubor, do kterého bude zapisovat, jelikož bude pro všechny požadavky stejný a je zbytečné jej otevírat a zavírat při každém požadavku.

PHP proces čeká na příchozí požadavek k odbavení. Při jeho přijetí nastaví potřebné proměnné a spustí interpretaci skriptu. Spolu s nastavením proměnných je zavolán další callback z rozšíření, který může připravit k načtení již konkrétní hodnoty pro daný požadavek - může tím být například vlastní implementace session.

Následně dochází k případným voláním funkcí při vykonávání skriptu obsluhujícího požadavek. Po dokončení požadavku je zavolán další callback, který

[3] https://github.com/mauricesvay/php-facedetection

[4] https://phalconphp.com/en/

umožňuje provést uvolnění paměti a další operace pro funkcionalitu rozšíření. Z důvodu optimalizace nedochází po obsloužení požadavku k ukončení procesu PHP, ale proces čeká na další příchozí požadavek. Pokud dojde k vypnutí webového serveru, tak je zavolán další callback, který by měl provést úklid zbylých zdrojů - protiklad k prvně volanému callbacku. Celý tento proces znázorňuje diagram na obrázku č. 1.

Z PHP rozšíření není možné ovlivňovat seznam vložených souborů pro funkce `include_once` a `require_once`. Pokud bude přeložen zdrojový kód vkládající část pomocí jedné z těchto funkcí, tak v době vykonávání bude možné provést opět jedno vložení souboru, který byl vložen v přeloženém kódu.

Velmi problematickým je pak překlad konstrukcí jako `$$varName`, které v jazyce PHP zpřístupní proměnnou, jejíž název je uložen v proměnné `$varName`. Vzhledem k tomu, že hodnota proměnné může být předána jako argument funkce, či načtena ze souboru a nemusí být v době překladu známa, se jedná o velmi problematickou konstrukci, která není v současné implementaci podporována.

3. Výběr podmnožiny PHP

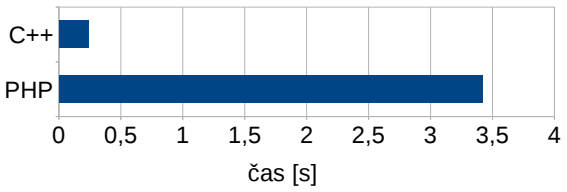
Při návrhu nástroje bylo třeba vybrání vhodné podmnožiny PHP k implemetaci. Vzhledem k rozsahu nebylo možné za omezený čas implementovat podporu kompletní gramatiky jazyka. Vybraná podmnožina by měla umožnit tvorbu funkcí, volání funkcí a řízení toku programu. Tedy implementovat konstrukce pro podmínky, cykly a zaměřit se zejména na podporu výrazů, která je klíčová. Při transformaci výrazů mezi těmito jazyky je třeba řešit odlišnosti v prioritě a chování některých operátorů.

V současné době neimplementovanou, ale v budoucnu plánovanou funkcionalitou je podpora objektového programování - tj. podpora pro třídy, dědičnost, rozhraní, jmenné prostory, mixins apod.

```
1  function hello($name, array $arg = [])
2  {
3    foreach($arg AS $a)
4    {
5      if(rand() AND 1 ** "15")
6      {
7        echo "Hello ".$name." -- ".$a;
8      }
9    }
10   return str_replace("a","b",$arg);
11 }
```

Kód 1. Podporované výrazy PHP

Nástroj provádí jako jednu ze svých funkcí analýzu výrazů a kódu pro detekci datových typů. Použití



Obrázek 2. Porovnání rychlosti Bubble sortu.

C++ třídy s přetíženými operátory povádějící převod hodnoty do požadovaného typu pro veškeré proměnné by mohlo mít nepříznivý dopad na rychlost, a proto, pokud je to možné, jsou použity primitivní datové typy `long`, `double`, `std::string`, `boolean` a `std::vector` / `std::map`.

Kód 1 ukazuje některé podporované konstrukce překladače pro převod z PHP do jazyka C++, které je možné použít v zdrojovém kódu - možnost definovat výchozí hodnotu argumentů, podpora pro mocninu, vyhodnocování výrazů, či volání vestavěných funkcí.

4. Experimenty a implementace

Před zahájením návrhu a implemetace jsem provedl několik testů pro změření rozdílů doby vykonávání. Na obrázku č. 2 můžeme vidět graf ukazující dobu potřebnou pro seřazení stejného pole o 50 000 náhodných číslech algoritmem Bubble sort. Zdrojové kódy obou variant jsou přiloženy u zdrojového kódu projektu. Výrazný rozdíl časů - 3.42s a 0.24s v prospěch pro C++ naznačuje, že je možné dosáhnout výrazné optimalizace. V tomto případě zrychlení 14.24x. Obdobných výsledků hovořících v prospěch C++ verze zavedené jako rozšíření dosáhly další experimenty. Uvedu dále například součet všech čísel od 1 do 100 000 000 000, kdy čísla jsou ještě více rozdílná - 73s PHP a 0.35s C++. Tyto testy byly prováděny na počítači s OS Fedora 21 64bit, Intel i7 4702MQ, ReiserFS na SSD. Překladč GCC 4.9.1 s volbou optimalizace -O3. Měření bylo provedeno 50x a vypočten matematický průměr. Žádná z naměřených hodnot nevybočovala výrazněji z průměru.

Výsledné rychlení bude velmi záležet na charakteristice překládaného kódu. V případě, že se bude jednat o pouhá volání vestavěných či knihovních funkcí, které jsou již zkompilované, tak zrychlení bude maximálně v řádu jednotek procent, či dokonce nulové. Mnohem zajímavějších výsledků je však možno dosahnout v případě funkcí, které provádějí ve větší míře operace s proměnnými, viz výše uvedený Bubble sort.

Rozšíření je možné implementovat v C prostřednictvím rozhraní definovaného společností Zend, která stojí za referenčním interpretrem PHP. Dokumentace i s návody je dostupná na jejím webu. Po průzkumu

možností jsem zvolil knihovnu PHP-CPP[5] od společnosti Copernica, která umožňuje pohodlnější tvorbu PHP rozšíření přidáním abstrakce a zpřístupněním funkcí z PHP v prostředí jazyka C++.

Tato knihovna implementuje třídu `Php::Value`, která slouží pro předávání argumentů z PHP do funkcí implementovaných v jazyce C++ a zpět. Pro samotné použití v generovaném kódu místo nativních typů není vhodná, jelikož její provázání s PHP interpretem má vysokou režii.

Pro tvorbu programu byl zvolen jazyk PHP, který poskytuje prostředky pro rychlý vývoj s možnostmi provádět rychlé změny v kódu. Dalším důvodem pro PHP je vestavěný tokenizer pro zdrojový jazyk. Ten bylo třeba rozšířit o podrobnější kategorizaci tokenů, o což se stará vlastní lexikální analyzátor.

Pro tvorbu abstraktního syntaktického stromu z tokenů je použit vlastní parser s rekurzivním sestupem. V průběhu parsování jsou již předávány generátorům kódu informace o konstrukcích, které mají generovat a prováděna derivace výrazů precedenční analýzou.

Před samotným generováním C++ kódu dochází k detekci datových typů. Tato detekce je provedena ve dvou krocích. Při prvním průchodu stromem jsou určeny požadované vstupní typy hodnot do uzlů stromu a výsledná hodnota na základě operátorů. Například při operaci dělení je zřejmé, že výsledkem bude typ float, či u konkatenace řetězec a jeho operandy budou také řetězec (u jiných typů se provede přetypování). Obdobně jsou pak určeny typy hodnot - čísla, řetězce, či návratové hodnoty vestavěných funkcí.

Druhým průchodem tímto stromem s již částečnými informacemi o typech jsou doplněny zbylé typy v závislosti na operacích - pokud je prováděno sčítání dvou hodnot typu int, tak výsledkem bude int. V případě, že je jedna z nich typu double, tak již výsledek bude také typu double. V případě přiřazení do proměnné je tento typ uložen k proměnné s adresou značící místo použití typu.

Pokud se nepodaří detekovat s jistotou typ proměnné, tak je raději zvolen benevoletnější, který umožní uložit všechny možné hodnoty - tedy pro číslo raději double, než int. V případě úplné neznalosti typu je použita instance třídy `Php::Value` z knihovny PHP-CPP, která má větší režii, ale umožňuje práci s všemi datovými typy[7].

Při generování C++ kódu jsou řešeny rozdíly mezi C++ a PHP. Průchodem bloků kódu jsou získány všechny použité proměnné a definovány na začátku funkce. Důvodem je rozdílný obor platnosti proměnných. Dále je v instanci třídy reprezentující proměnnou uloženo od jaké části kódu byla proměnná definována a kde byla zrušena konstrukcí `unset`, aby došlo k nahrazení konstrukce `isset` za správnou hodnotu.

Další generátor je specializovaný pro generování výrazů. Při generování řeší konverzi výsledků podvýrazů na datový typ potřebný pro další operace prostřednictvím funkcí `php2cpp::to_string` a `php2cpp::to_float` implementovaných v C++. Dalším úkolem je konverze operátorů chybějících v C++. Pro operátor mocniny je volána funkce `pow` z matematické knihovny a rozdílné priority operátorů jsou řešeny závorkováním.

Volání vestavěných, či uživatelem definovaných funkcí je řešeno pomocí metod z knihovny PHP-CPP pro provázání s interpretrem.

5. Závěr

Konverze PHP knihoven do C++ je možná a použití dává smysl při konverzi knihoven, či jiných neměnných částí. Tento postup je vhodný i při vývoji, kdy testování trvá kratší dobu a je možno měnit kód aplikace.

V případě jednoduššího kódu je výsledný kód výrazně rychlejší, než původní. Pro složitější konstrukce je komplikované určit přesně datové typy a tak nemusí být zrychlení natolik výrazné, aby se vyplatilo s přihlédnutím k možnému zavlečení chyby.

Jsou dostupné zdrojové kódy[6] pod open source licencí Apache 2.0 umožňující vyzkoušet si překlad a v případě úspěchu nasadit, či se zapojit do vývoje.

V budoucnu je plánována podpora tříd, propracovanější detekce datových typů (nyní je občas použit double i když by stačil long), omezení zbytečných volání `to_string` a `to_float`. Dále je plánováno experimentování s další detekcí datových typů, kdy slibné výsledky by mohlo poskytnout sledování hodnot proměnných v interpretované verzi při provozu. Při dostatečně velkém množství vzorků by bylo možné vygenerovat optimalizovanou verzi + obecnou. Při zavolání knihovny by se poté zvolila varianta dle typů argumentů.

Literatura

[1] Fabien Potencier. Performance impact of the php garbage collector. blogpost (english), Dec 2014. http://blog.blackfire.io/performance-impact-of-the-php-garbage-collector.html.

[2] Drew Paroski. Speeding up php-based development with hhvm. article (english), Nov 2012. https://www.facebook.com/notes/

[5] http://www.php-cpp.com/

[6] https://www.github.com/nechutny/BP

facebook-engineering/speeding-up-php-based-development-with-hiphop-vm/10151170460698920.

[3] David Peterson. Rasmus lerdorf: Php frameworks? think again. article (english), Aug 2008. http://www.sitepoint.com/rasmus-lerdorf-php-frameworks-think-again/.

[4] Kenji Suzuki. Php framework benchmark. github (english), Nov 2015. https://github.com/kenjis/php-framework-benchmark.

[5] Life cycle of an extension. article (english). http://php.net/manual/en/internals2.structure.lifecycle.php.

[6] The zend_module structure. article (english). http://php.net/manual/en/internals2.structure.modstruct.php.

[7] Emiel Bruijntjes. Working with variables. comment (english), May 2015. http://www.php-cpp.com/documentation/variables#comment-2021818775.