Statement-Level Parallelism

Dibakar Gope and Mikko H. Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
gope@wisc.edu, mikko@engr.wisc.edu

Abstract

The dynamic scripting language PHP has become enormously popular in preparing web pages in the server-side. A program written in PHP either uses an interpreter or a Just-In-Time (JIT) complier to execute its program statements. Recent works in JIT compiler for PHP have demonstrated substantial improvement in PHP runtime in web server environment. However, in absence of sophisticated analyses, compiler optimizations for JIT-compiled PHP is forced to be conservative, resulting in sub-optimal code. In this abstract, we propose two ideas—statement-level parallelism (SLP) and same-target decoupled control-flow—to accelerate the execution of PHP intermediate code directly. The idea is to exploit the available parallelism present among the independent PHP script statements using SLP. The decoupled control-flow allows same-target separable branches to compute their predicates in parallel and the branch that resolves the earliest among those to dictate the execution time of the entire group of branches.

Now we provide an overview of the two ideas.

1. Statement-Level Parallelism

The Zend Engine acts as an interpreter for the PHP scripts. It generates the intermediate code and passes to the executor which executes these operations one at a time from the opcode stack as illustrated by the interpreter execution loop in Figure 1. The sequentialization in the Zend execution engine essentially fails to extract the available parallelism present among the independent PHP script statements.

For a dynamically-typed language, such as PHP, access to a variable requires a lookup in a symbol table to resolve its location in memory. In general, each of the PHP script statements is interpreted to multiple intermediate code operations. Furthermore, the execution of these operations requires invoking functions which carry out the actual operation. The majority of these functions meant for various operations in PHP involve traversing through long chains of branches including indirect branches, resulting in poor single-threaded performance and under-utilization of processor resources. At the end of their execution, these functions also trigger the execution of the next intermediate code operation in the sequence.

Figure 2 demonstrates an example code from the *Customization* phase of the E-commerce application in SPECweb2005. Each of the key-value pair assignments involved in populating the various fields of the variable *array* requires to execute

```
void execute_ex (.....)
while (1) {
    .....
   if (( ret = OPLINE->handler(execute_data TSRMLS_CC)) > 0) {
      switch ( ret ) {
          .....
        case 3:
          execute_data = EG( current_execute_data );
          break;
          .....
      }
   }
}
```

Figure 1: Main Interpreter Execution Loop in Zend VM.

```
$item = array ();
 $total_price = $_SESSION['backend_price'];
foreach ( $configchoices as $choice){
   if (empty($_POST[$choice[0]])) $_POST[$choice[0]] = $choice [2];
   if ($_POST[$choice[0]] == $choice [2]) $total_price += $choice [3];
   $item[$choice [0]][] = array (
   'name' => $choice [1],
   'id' => $choice [2],
   'price' => $choice [3],
   'currency' => $choice [4],
   'selected' => ($_POST[$choice[0]] == $choice [2])
   );
}
```

Figure 2: Example Code Illustrating SLP.

ZEND_FETCH_R and ZEND_ASSIGN intermediate code operations. While the execution of those operations cannot exploit high instruction-level parallelism due to the reasons stated above, however there is an abundance of statement-level parallelism (SLP) in populating the various fields of such a customization choice. Since there are no true dependences among the key-value pair assignments, they can be executed in parallel.

This proposal advocates a light-weight hardware mechanism[1] to dynamically track those true data dependences among PHP statements, execute independent statements in parallel and thus extract SLP in PHP scripts.

We analyze the data-flow and the control-flow landscape in couple of PHP benchmark suites, such as *SPECWeb2005*, *Shootout*, *RUBBoS*, *RUBiS*, *Eveactive_1.0*, *phpSQLiteAdmin-0.2*, *tigerPhpNewsSystem_1.0_beta_build39* in addition to the

[1] L. Chen, S. Dropshoy, and D. H. Albonesi. Dynamic Data Dependence Tracking and its Application to Branch Prediction, HPCA 2002.

microbenchmarks from *phpbench*. Microbenchmarks from *phpbench* and *Shootout* have moderate to high SLP, whereas the applications from the *SPECweb2005*, *RUBBoS*, *RUBiS* and *tigerPhpNewsSystem_1.0_beta_build39* have substantial presence of SLP.

2. Same-Target Decoupled Control-Flow

Same-target decoupled control-flow attempts to exploit parallelism among *same-target separable* branches. *Same-target separable* branches in a group have two characteristics:

1) Each of those branches has the same taken target address.

2) The predicate computations of those branches do not depend on each other. In other words, each of those branches is not data dependent on the preceding branches in the original sequential program. The predicate computation (backward slice) of one branch is entirely separable from the predicate computation of others in the group, conceptually parallelizing their predicate computation.

Total separability thus allows same-target branches to compute their predicates in parallel and the branch that resolves the earliest among those to dictate the execution time of the entire group of branches.

The banking and E-commerce applications from the SPECweb2005 benchmark suite illustrate plenty of scenarios with *same-target separable* branches.

Many of those applications, such as checking, billing, shipping and login information, transferring money to a payee's account and so forth, require validating the contents entered for various fields. However the predicate computation for those fields do not depend on each other. As a result, the contents entered for those fields can be checked simultaneously. As soon as the information entered for one of the fields is found to be invalid or inaccurate, the validation check for the remaining fields occurring in parallel can be skipped. *Same-target separable* branches thus can benefit from the uneven time spent in resolving the various predicates. For example, the complexity of validating the content of an email field or performing a regular expression match against it is much higher than validating a portion of an address field (for example, content entered for the *State* field in an address). Figure 3 illustrates an example code from the *quick_pay* script of the banking application in SPECweb2005. Here in this example, all the conditions inside the *foreach* loop have the same taken target address. Furthermore, their predicate computations do not have any data dependence on each other. As a result, all those conditions inside the *foreach* loop before making a payment can be checked simultaneously and failure to meet one of those conditions such as *no amount entered* can trigger other condition checks to skip their execution.

Similar control-flow characteristics with *same-target separable* branches are also observed in majority of the applications from the *RUBiS* and *RUBBoS* benchmark suites.

There has been prior research work in dynamic reconvergence prediction[2]. Our proposal advocates to extend such a light-weight hardware mechanism to dynamically track those *same-target separable* branches in PHP scripts.

```
$userid=$_SESSION['userid'];
if(empty($_POST['payee']) || ! is_array ($_POST['payee'])){
    exit ();
}else{
    $payee=$_POST['payee'];
}
if(empty($_POST['date']) || ! is_array ($_POST['date'])){
    exit ();
}else{
    $date=$_POST['date'];
}
if(empty($_POST['amount']) || ! is_array ($_POST['amount'])){
    exit ();
}else{
    $amount=$_POST['amount'];
}
foreach ($payee as $index => $id){
    if(empty($id)) continue;
    if(empty($date[$index])) continue;
    $date_array= split_date ($date[$index]);
    if(! $date_array ) continue;
    $today=localtime(time (), TRUE);
    if(compare_date($date_array [0], $date_array [1], $date_array [2],
    $today['tm_year']+1900, $today['tm_mon']+1, $today['tm_mday'])<0) continue;
    if(empty($amount[$index])) continue;
    $a=(float )$amount[$index];
    if($a<=0.0) continue;
    $request=BACKEND_CMD_QUICK_PAY.'&'.$userid.'&'.$id.'&'.$date[$index].'&
     list ($r, $errno) = backend_get_array( $request );
    if($errno) continue;
     .....
}
```

Figure 3: Example Code Illustrating *Same-Target Separable* Branches.

[2] J. D. Collins, D. M. Tullsen, and H. Wang. Control Flow Optimization via Dynamic Reconvergence Prediction, MICRO 2004.