Implementation of Additional Optimizations for VPO on the Power Platform

Chungsoo Lim, Gary A. Smith, Won So
{clim, gasmith, wso}@ncsu.edu
http://www4.ncsu.edu/~gasmith/csc791a_proj

Spring 2005 CSC791A Project Final Report
06 May 2005

Solved Issues

This section contains a detailed analysis of the issues solved during the course of our project. Each of our three assigned tasks was investigated individually by team members. The team met to discuss high-level issues involving cooperation of optimizations. We also shared insights regarding how our individual implementations operate. Our group met at regular intervals to evaluate team progress and ensure that the workload breakdown was fair to all team members. As tasks were completed, members of the team transitioned between roles to collaborate on the remaining tasks until completion.

Common Subexpression Elimination (CSE)

Chungsoo investigated the local CSE compiler pass. Later, Chungsoo and Gary collaborated on the global version of this optimization. VPO code for CSE is located in powerpc-vpo/vpo/lib/cse.c.

Chungsoo verified the correctness of local CSE using two approaches. The first approach was to read the corresponding code and try to find errors. The second was to create example test codes consisting of a single basic block. Chungsoo compiled tests, compared RTLs and assembly codes, and evaluated results. He found evidence to support that local CSE functioned correctly for the test codes.

Three warning messages were present when global CSE was invoked. Although the messages were only warnings from the compiler's standpoint, they were the result of global CSE failing to fully optimize the code. The following list summarizes warnings and their solutions.

1. **vpo (common_subexpression_elimination): basic block not examined**
   Gary found that this warning message is emitted because basic blocks were originally left unexamined in the implementation. Control flow was traversed by the CSE optimization starting at the top block. All successors were then recursively examined until the entire control flow tree was evaluated.

   Some basic blocks, comments for example, were not touched during this process. In addition, dead code (i.e., a control flow tree whose root has no predecessor)

would not be evaluated using the original traversal scheme. As a result, blocks marked undone at the beginning of CSE continued to be in this state at the end of the algorithm.

After the active control flow path has been evaluated, Gary's amended CSE source code (see the CSC791A comment in cse.c – common_subexpression_elimination function) walks through all basic blocks to find items in the undone state. As a result, this warning is no longer emitted by VPO after our changes are incorporated in the compiler package. This warning was produced by nearly all of the cases in powerpc-vpo/tests. After Gary's changes, the warning was eliminated.

2. **vpo (add_link): not enough links available**
   Gary investigated and resolved this problem. Links are needed by CSE when associating producer and consumer instructions. Given that a single producer can feed many consumers, a large number of links may be needed by the CSE optimization pass. The number of links available for this purpose is specified by the MAXLINKS macro in the VPO implementation. When the optimization attempts to add a link and none are available, the "not enough links available" warning is emitted. The CSE compiler pass will complete; however, optimization opportunities are missed because all of the available links are consumed. The default value of MAXLINKS for powerpc-vpo is 6.

   This warning was produced by the following cases in powerpc-vpo/tests: ctests {010, 011, 012, 050, 051, 052, 060, 905, 905b, 905c, 905d, 905e, 908, 910, 917, 918}, eqntott, espresso, misc {othello, od, matmult, fm-part, diag08, diag03, diag01, compact}, and others {t12, t13, t8, t9}. The solution to this problem was to specify MAXLINKS=15 in vpo/lib/Makefile. The warning message was eliminated from all test cases as a result of this change.

   Note: Segmentation faults were originally produced for eqntott and espresso when this change was implemented. After many hours of investigation, Gary learned that custom debugging code added to the VPO implementation was the cause for the segmentation faults. Once this debugging code was switched off, all tests in powerpc-vpo/tests were successful. Although this fix appears trivial, the problem was difficult to diagnose. The segmentation faults were being emitted from various locations through the compiler, such as the show_rtl and rins functions, that were not directly related to the debugging output. Furthermore, the abnormal behavior continued after calls to the functions causing segmentation faults were eliminated. In some cases, the presence of basic fprintf calls was causing a problem.

   The effort invested in finding the source of the segmentation faults was valuable. When the "vpo (cse_basic_block): cannot recalculate =r[2];" (item 3 below) message was eliminated by Chungsoo, problems were encountered with the

espresso test. Gary suggested that Chungsoo switch off debugging output. Subsequently, all tests functioned correctly.

3. **vpo (cse_basic_block): cannot recalculate =r[2];**
   This warning message is produced because the current vpo implementation does not fully support power-pc RTLs. When this message was encountered, it was difficult to track down where this message was generated. Chungsoo tried to reproduce this message with simple test code. Fortunately, test013.c in the powerpc-vpo/tests/ctest directory can reproduce the warning message. This code contains only one printf statement for printing out strings. First of all, the RTL that caused the message was identified. The RTL is as follows:

   r[3] = TC[L0_TC_LBL_temp];=r[2];

   Note that unlike any other RTLs, there are two semicolons. Chungsoo sought advice from Professor Muller, who suggested that the corresponding assembly line may be helpful. The matching assembly line is as follows:

   Lwz 3, L0_TC_LBL_temp (2)

   L0_TC_LBL_temp is an offset to a memory location and r[2] is used as a base register in indirect addressing mode.

   Extensive code examination was performed to determine how existing code could be modified to support such an RTL. After hours of painful examination, two problems were identified.

   1) Argument to rplfast( )
   rplfast( ) is called by csc_basic_block( ), which performs local common sub-expression elimination. This function replaces an item with the fastest known version. The function is called twice for load instructions (for both source and destination) or once for other instructions (for destination). Two of the arguments to the function are the pointers that specify the beginning and the end of a string that is a part of an RTL. For example, if the function is called for a source operand (left hand side of an equation), the pointers specify the beginning and the end of the source.

   Thorough examination of the code revealed that the function was called twice (destination only) for the problematic RTL. This behavior resulted from the presence of two semicolons in the RTL. If only part of a source is used as an input to rplfast( ), the entire source cannot be replaced by a less expensive alternative. Furthermore, an optimization opportunity may be missed in this scenario.

   The pointer that defines the end of the string is moved forward from the first semicolon to the second semicolon in order to solve this problem and make the

pointers specify the entire source. After this change, the warning message was eliminated. In the VPO implementation, yyparse( ) checks the validity of RTLs. A yyparse() return value of 0 indicates success and a return value of 1 indicates that a warning is present. The warning message was originally generated when the RTL was removed by rplfast( ) and yyparse( ) returned 1. By setting the pointers correctly, the RTL is not removed and yyparse( ) returns 0.

2) gtds( )
The warning message was eliminated for all RTLs. However, a problem still existed in the updated implementation. For instance, a given RTL would be eligible for elimination but would remain after CSE. Chungsoo inspected equivalent classes and found that the specific equivalent class created by the RTL was incorrect. For example, the equivalent class might appear as follows:

r[3] = TC[L0_TC_LBL_temp]

The part after the first semicolon, r[2], in the problematic RTL is missing in the equivalent class. To pinpoint this problem, the update_equivalence_table( ) function was examined. This function uses source and destination information provided by gtds( ). gtds( ) emits source and destination portion of an RTL as string pointers.

Chungsoo discovered that gtds( ) and rplfast() had the same pointer problem. gtds() is called twice for the problematic RTL because of the presence of two semicolons. When this function is called for the first time, TC[L0_TC_LBL_temp] is emitted as a source. When called the second time, r[2] is emitted as a source. This behavior causes the equivalent class to be incorrect.

To fix the problem, Chungsoo made gtds( ) return the entire source instead of two separate ones. After fixing this problem, a correct equivalent class is created for the troublesome RTL and thus, the RTL can be eliminated by CSE. Chungsoo also made sure that register's use counter in the source portion of the RTL was updated correctly.

This warning was produced by the following cases in powerpc-vpo/tests: ctests {013, 014, 015, 016, 017, 018, 902}, diff, espresso, jpeg, misc {subpuzzle}, msim, yacc. By fixing the two problems described above, the warning message is completely eliminated and more optimization opportunities for CSE are provided.

Evaluation order determination (EOD)

Won investigated the Evaluation order determination optimization. Evaluation order determination (EOD) is an optimization, which reorders evaluations so that ones which cost much are evaluated first. There is no benefit with this optimization by itself but it effectively helps following optimization such as register allocation reducing the live ranges of registers.

EOD code within VPO is located in powerpc-vpo/vpo/lib/eod.c. The EOD pass is composed of two processes: The first is constructing links between RTLs with ruage() function and compute costs of RTLs. The second is reordering RTLs so that the operations evaluating the same value aggregate together then the evaluations with higher costs are located prior to the ones with lower costs.

Test results by enabling EOD with "-VV" option shows that the EOD routine in given power-vpo has some problems. For most benchmarks, compilation does not finish. Won investigated the EOD routine and fixed existing problems. Steps for attacking problems and corresponding fixes are described as follows:

1) Basic test [3/28 ~ 4/3]
By writing the custom function[1] which emits RTLs before and after performing this optimization, Won examined the problems in current EOD implementation. Testing with simple operations such as series of summations and multiplications does not expose the problem and the changes in RTLs before and after EOD look correct.

2) Naïve approach to resolve the infinite looping problem [4/3 ~ 4/10]
By testing with the files in the benchmark, which end up with infinite looping such as JPEG, 2 problems were identified in the code and resolved. The first problem was the wrong link (Every RTL has a link array which stores pointers for producer RTLs. i.e. use-def chain) constructed by ruage() function in the 1st process. Some RTLs have wrong links to a future instruction. This problem is fixed by adding a condition before calling add_link() function. The second problem happened while reordering instructions. The while loop calling reoder() could not escape from it because it was looping with the same RTL. This problem might be cause by a error in the reorder() function. It was fixed by adding one condition inside this loop so that it can avoid traversing the same item infinitely. However, the compiled binary was suffering segmentation fault in some cases though the infinite looping problem was fixed.

3) Analytical approach to resolve the infinite looping problem [4/11 ~ 4/23]
Following Dr. Mueller's suggestion, Won tried to run VPO for Sparc machines (sparc-vpo). Though VPO on a Sparc is fully operational[2], comparing RTLs from sparc-vpo and ones from power-vpo gave an insight. Since VPO is designed to operate RTLs regardless of platforms, a problem may not occur due to an algorithmic error but some architectural differences of Sparc and Power platforms. Won found that rusage() function were working improperly after examining differences in RTLs. Rusage() function creates links (i.e. set up dependences) which are not explicitly shown in RTLs. These links include a link between function call and an operation using its return value, and a link between an operation setting a condition register and its consumer operation. In power-vpo, the link between a function call and an RTL using its return values was not created properly. The

[1] The –D option does not print out any output yet with some trials.
[2] We are not successful in building given power-vpo code for Sparc machines. Instead we downloaded original vpo code from UVA and compiled on Sparc machines. With a little modification, sparc-vpo works but it is not fully functional. Some errors are found while compiling given benchmarks even with –VA option.

reason was that the condition for finding a call instruction – the macro cnts() – worked only for a Sparc RTL. Won modified the condition – the macro is_call() – so that the link is created correctly between a function call and a dependent instruction. (See WSO_FIX in rusage().)
The previous fixed code turned out to be obsolete because it only eliminates the links referencing the future instructions but does not create right links. Therefore, old changes were reverted. With a new fix, the infinite looping problem was completely removed. However, some benchmarks still experienced segmentation faults during execution or diff fails during verification.

4) Attack for segmentation faults [4/24 ~ 5/1]
Won began with testing with a simple benchmark, wc and cb in misc, which causes a diff fail and a segmentation fault respectively. With extensive comparison of RTLs, Won found that the problem occurred only when reordering a operation which assigns grp2 (i.e. "r[2] = …"), which always immediately follows a call operation (i.e. "ST=…"). This assignment of grp2 seems to exist for recovering a data frame pointer after a function call. If this operation is moved down below, the generated assembly code is incorrect. Therefore, one more check routine is added into move_down() function which is recursively called by reoder() function. (See WSO_FIX in move_down().) The routine checks if the operation asked to be moved down is the assignment of gpr2 and prevents it from being reordered if it is. With this fix, all benchmarks pass compilation, execution and verification. The table below summarizes the changes of benchmark results before and after these fixes.

CNF = "compilation does not finish"

Benchmarks	-VA	-VV w/o fix	-VV with fix in 3)	-VV with fix in 3) & 4)
avdhoot	OK	ta26.c: CNF	OK	OK
Ctest	OK*	test905.c: CNF	OK	OK
Diff	OK	diffreg.c: CNF	diff fail	OK
eqntott	OK	cannon.c: CNF	OK	OK
espresso	OK	cmpl.c: CNF	Verification error	OK
Jpeg	OK	jddeflts.c: CNF	seg. fault	OK
Misc	OK	cb: CNF	cb, othello: seg. fault wc: diff fail	OK
Msim	OK	OK	OK	OK
Others	OK	OK	OK	OK
Yacc	OK	Y1.c: CNF	Verification error	OK

* test008.c and test115.c are experiencing diff fails with -VA. We ignored these.

Live Variable Analysis

Gary was responsible for the liveness analysis optimization. Existing VPO source code was evaluated to discover the status of existing functionality. The source code for live_variable_analysis is located in powerpc-vpo/vpo/lib/dataflow.c.

This analysis pass clears old dataflow accounting information in basic blocks, examines each RTL, sets use/def vectors, and performs an iterative liveness analysis algorithm.

Gary learned how to emit RTL and assembly files from the VPO compiler suite. Although debugging output was available from the VPO compiler tool, Gary created custom functions to output liveness information. For instance, writing custom routines to output control flow information, RTL instructions, and liveness data was more efficient for our testing purposes than utilizing the existing code in the VPO package. Test codes were authored and utilized to ensure that liveness analysis is functional.

When examining the results of live variable analysis, it was learned that no variables other than the stack pointer and global data pointer were live across basic blocks. The compiler would perform loads immediately prior to value use and then stores as soon as each operation was completed. As a result, registers were not live across basic blocks. This problem is eliminated when CSE is enabled during compilation.

Open Issues

We found during testing that all cases who executed successfully for –VA (No Optimization) compilation also functioned successfully when compiled with –VC (CSE) and –VV (EOD). We were curious about the interaction between CSE and EOD; therefore, we experimented with using a combination of the optimizations simultaneously. We found that test cases using –VVC (CSE and EOD) compilation executed correctly. However, one case failed during testing. The test case eqntott yields output file differences when compiled with –VVC. This difference can be investigated through an in-depth examination of the test case and optimizations to learn why differences exist. Since this test case executes successfully when the CSE and EOD optimizations are invoked independently, we hypothesize that the interaction between the two optimizations might be a cause for this behavior. Future work might seek to solve this problem by isolating the portion of the eqntott code yielding this difference and evaluating how CSE and EOD affect the associated calculations.

Disclaimer

This paper and associated software changes are intended for **informational purposes only**. Therefore, any use of the information presented in this student work is at your own risk. Chungsoo Lim, Gary A. Smith, and Won So provide **no warranties of any kind** surrounding the use of this material.