# Path Profile Guided Partial Redundancy Elimination Using Speculation *

Rajiv Gupta†
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

David A. Berson  Jesse Z. Fang
Microcomputer Research Lab
Intel Corporation
Santa Clara, CA 95052

**Abstract**

*While programs contain a large number of paths, a very small fraction of these paths are typically exercised during program execution. Thus, optimization algorithms should be designed to trade off the performance of less frequently executed paths in favor of more frequently executed paths. However, traditional formulations to code optimizations are incapable of performing such a trade-off. We present a path profile guided partial redundancy elimination algorithm that uses speculation to enable the removal of redundancy along more frequently executed paths at the expense of introducing additional expression evaluations along less frequently executed paths. We describe cost-benefit data flow analysis that uses path profiling information to determine the profitability of using speculation. The cost of enabling speculation of an expression at a conditional is determined by identifying paths along which an additional evaluation of the expression is introduced. The benefit of enabling speculation is determined by identifying paths along which additional redundancy elimination is enabled by speculation. The results of this analysis are incorporated in a speculative expression hoisting framework for partial redundancy elimination.*

## 1 Introduction

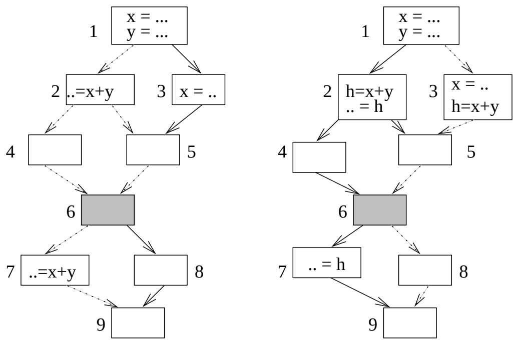
Existing approaches to partial redundancy elimination (PRE) remove redundancy along some paths without adding additional instructions along any path. Implicit in this formulation is the assumption that all paths through the program are equally important. However, in practice, it has been observed that some paths through a program are more frequently executed than others and many paths are never executed [1]. In a study of SPEC95 integer benchmarks that we report in this paper we found that 65% of the functions in these benchmarks contained no more than five paths with non-zero execution frequency. Thus, an optimizer should exploit path execution frequency information to reduce redundancy along more frequently executed paths even if doing so introduces some additional instructions along other less frequently paths. While instruction scheduling algorithms have long exploited this observation, code optimization algorithms do not take advantage of it. We present PRE algorithms that trade-off the performance of less frequently executed paths in order to obtain improved performance for more frequently executed paths.

Consider the example in Figure 1. In the first flow graph shown in Figure 1a, the evaluation of the expression $x + y$ in node 7 is partially redundant. Along paths that visit node 2 prior to reaching node 7 (i.e., paths $P1$ and $P2$) the expression is evaluated twice. A traditional PRE algorithm will not be able to remove this redundancy because it will not allow the expression evaluation in node 7 to be hoisted above node 6. Now consider the second flow graph in Figure 1b in which the expression evaluation has been hoisted above node 6 and placed at node 3 using speculation (i.e., unconditional execution of an expression that is otherwise executed conditionally). Speculation has enabled the removal of redundancy along paths $P1$ and $P2$. At the same time an additional evaluation of $x+y$ has been introduced along path $P5$. If the profile information indicates that the total number of times paths $P1$ and $P2$ are executed is expected to be greater than the number of times path $P5$ is executed, then the benefit derived from speculation at



†Supported in part by NSF PYI Award CCR-9157371, NSF grant CCR-9402226, Intel Corporation, and Hewlett Packard.

node 6 is greater than the cost of allowing speculation. For the profile information shown in Figure 1c, it is clear that speculation is beneficial since the expected number of evaluations of $x+y$ is reduced by 110 $(Freq(P1)+Freq(P2)-Freq(P5) = 100+100-90)$.




(a) (b)

| Path | Freq. |
|---|---|
| P1: 1-2-4-6-7-9 | 100 |
| P2: 1-2-5-6-7-9 | 100 |
| P3: 1-2-5-6-8-9 | 10 |
| P4: 1-3-5-6-7-9 | 60 |
| P5: 1-3-5-6-8-9 | 90 |
| P6: 1-2-4-6-8-9 | 20 |

(c)

Benefit(P1) = Freq(P1) = 100

Benefit(P2) = Freq(P2) = 100

Cost(P1) = Cost(P2) = Freq(P5) = 90

(d)

Figure 1: Path Profiles and Speculation.

In general it is safe to perform speculation for instructions that cannot cause exceptions. However, application of the same optimization to instructions that can result in exceptions (eg., a floating point divide) may alter the semantics of the program. This is because if this instruction is introduced along a path where it was not executed in the original program, then an exception may be incorrectly reported for an execution that follows that path. However, modern architectures provide solution to this problem [12]. Exceptions are suppressed initially if they occur during the execution of a speculated instruction. However, if later execution reaches the point at which the instruction originally resided, the exception is reported. The task of reporting is carried out by special instructions called sentinels [12]. For architectures that do not provide such advanced features, our algorithm should only be applied to non-exception causing instructions.

We present a general approach which reduces redundant expression evaluations along more frequently executed paths at the expense of introducing additional expression evaluations along less frequently executed paths. This approach first performs data flow analysis for computing cost and benefit information at all conditionals. Based upon this information paths along which redundancy elimination should be performed are selected and speculation at appropriate conditionals along the path is enabled. A speculative expression hoisting framework is then developed that uses the cost-benefit information with respect to an expression to enable hoisting of that expression past selected conditionals.

There are two types of profiling that can be used: *edge profiling* which gives the number of times each edge in the program flow graph is traversed and *path profiling* which gives the number of times various acyclic paths in a program are traversed. It is easy to see that edge profiles can be computed from path profiles and different path profiles can give rise to the same edge profile. Thus, it is not possible to accurately compute the frequency of a path using edge profiles and hence the degree of redundancy present in a program cannot be accurately measured using edge profiles. In fact from the edge profiles we can only assert that a path has an execution frequency of any where between zero and the minimum of the edge frequencies along the path. For example, even if each edge along path $P1$ in Figure 1a has a non-zero execution count, it is possible that path $P1$ has a zero execution count. This is because each of the edges along path $P1$ also belongs to other paths in the flow graph. If path $P1$ has a zero execution count, then there is no benefit to applying PRE for eliminating redundancy along this path. In fact the application of PRE may be harmful since it increases register pressure due the creation of a live range for the auxiliary variable $h$ introduced by the PRE transformation. Selective optimization can be guided more effectively using path profiles than edge profiles since path profiles more accurately measure the benefit of optimization. Therefore in this work we rely on path profiles to estimate the amount of redundancy accurately and then guide the application of PRE through these measures. Later in this paper we show that using path profiles it is possible to achieve greater degree of optimization and avoid unnecessary increase in register pressure in comparison to algorithms based upon edge profiles.

In section 2 we present data flow analysis for computing cost-benefit information. Section 3 describes the algorithms for selecting conditionals at which speculation is enabled and speculative expression hoisting to carry out PRE. In section 4 we discuss compile-time expense of cost-benefit analysis and show how it can be reduced by sacrificing the precision of such analysis. In section 5 related work is dicussed. Concluding remarks are given in section 6.

## 2 Cost-Benefit Analysis of Speculation

Speculation is the process of hoisting expressions such that an expression whose execution is controlled by a conditional prior to speculation, is executed irrespective of the outcome of that conditional after speculation. Cost-benefit analysis identifies opportunities for useful speculation. We first discuss the cost-benefit analysis for acyclic graphs and later present extensions for handling programs with loops.

The algorithms in this paper use the control flow graph representation of the program. We assume that each node in the control flow graph contains a single statement. This assumption simplifies the discussion of the data flow equations. We also assume that a node has been placed along each critical edge (i.e., an edge that connects a node with multiple successors to a node with multiple predecessors) to create code placement points along the critical edges.

To perform cost-benefit analysis for hoisting an expression $exp$ above a conditional node $n$, we categorize the program subpaths that either originate or terminate at $n$ as follows:

**Available subpaths** which are subpaths from the *start* node to $n$ along which an evaluation of $exp$ is encountered and is not killed by a redefinition of any of its operands prior to reaching $n$.

**Unavailable subpaths** which are subpaths from the *start* node to $n$ along which $exp$ is not available at $n$. These subpaths include those along which either an evaluation of $exp$ is not encountered or if $exp$ is encountered, it is killed by a redefinition of one of its operands prior to reaching $n$.

**Anticipable subpaths** which are subpaths from $n$ to the *end* node along which $exp$ is evaluated prior to any redefinition of the operands of $exp$.

**Unanticipable subpaths** which are subpaths from $n$ to the *end* node along which $exp$ is not anticipable at $n$. These subpaths include those along which either $exp$ is not evaluated or if $exp$ is evaluated, it is done after a redefinition of one of its operands.

The paths that can potentially *benefit* from hoisting of $exp$ above a conditional node $n$ are the paths that are obtained by concatenating *available* subpaths with *anticipable* subpaths at $n$. This is because along these paths an evaluation of $exp$ prior to reaching $n$ causes an evaluation of $exp$ performed after $n$ to become redundant. Given a path along which redundancy exists, in general, multiple conditionals may be present between the two evaluations of an expression. Thus, only when speculation is performed at multiple conditionals the redundancy is eliminated. The measure of the benefit of removing the redundancy along the path is simply the execution frequency of the path.

The paths that incur a *cost* due to hoisting of $exp$ above a conditional node $n$ are the paths that are obtained by concatenating *unavailable* subpaths with *unanticipable* subpaths at $n$. This is because along these paths an additional evaluation of $exp$ prior to reaching $n$ is introduced. The measure of the cost of removing redundancy along a path is measured by summing up the execution frequencies of paths that incur a cost due to speculation at each of the conditionals where the speculation must be enabled for removing the redundancy.

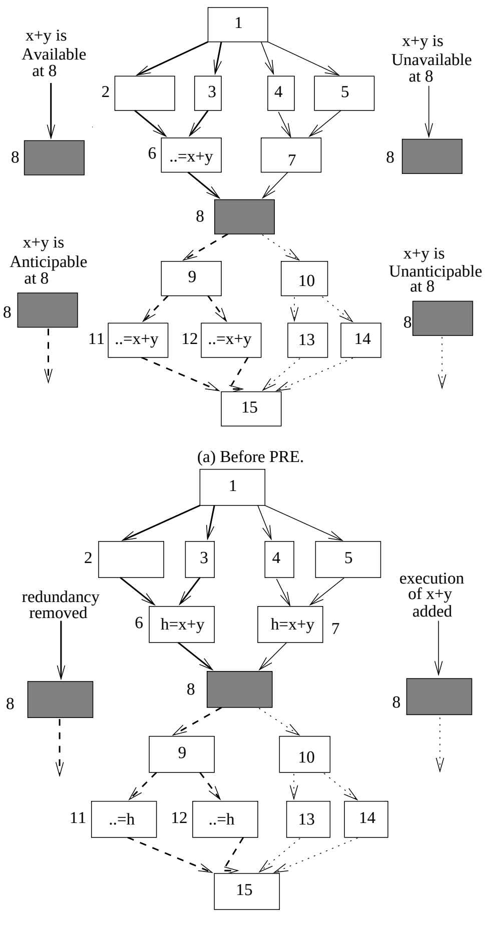



Figure 2: Cost-Benefit Analysis.

Consider the example in Figure 2a. In this example of the four subpaths from 1 to 8, $x+y$ is *available* along two subpaths (1-2-6-8 and 1-3-6-8) and *unavailable* along two subpaths (1-4-7-8 and 1-5-7-8). Of the four subpaths from 8 to 15, $exp$ is anticipable along two subpaths (8-9-11-15 and 8-9-12-15) and *unanticipable* along two subpaths (8-10-13-15 and 8-10-14-15). Thus the paths that *benefit* from the hoisting of $x+y$ above 8 include 1-2-6-8-9-11-15, 1-2-6-8-9-12-15, 1-3-6-8-9-11-15, and 1-3-6-8-9-12-15 (see Figure 2b). The paths that incur a *cost* include 1-4-7-8-10-13-15, 1-4-7-8-10-14-15, 1-5-7-8-10-13-15, and 1-5-7-8-10-14-15 (see Figure 2b).

In the remainder of this section we show how to express the identification of paths that contribute to the cost or benefit as a data flow problem. The precise definitions of cost and benefit are given next.

*Definition 1:* Given a path $p$ that passes through a conditional node $n$ in an acyclic path, the hoisting of an expression $exp$ above $n$ **benefits** path $p$ if and only if $exp$ is *available* and *anticipable* at $n$'s entry along $p$. We denote the set of paths through $n$ that benefit from hoisting of $exp$ above $n$ as $BenefitPaths_{exp}(n)$.

*Definition 2:* Given a path $p$ that passes through a conditional node $n$ in an acyclic path, the hoisting of an expression $exp$ above $n$ **costs** path $p$ if and only if $exp$ is *unavailable* and *unanticipable* at $n$'s entry along $p$. We denote the set of paths through $n$ that incur a cost due to hoisting of $exp$ above $n$ as $CostPaths_{exp}(n)$.

*Definition 3:* Given a path $p$ along which redundancy due to evaluation of expression $exp$ exists and the set of conditionals $Spec_{exp}(p)$ along $p$ at which speculation should be enabled to remove the redundancy, the cost of removing $exp$'s redundancy $Cost_{exp}(n)$ and the benefit of removing $exp$'s redundancy $Benefit_{exp}(n)$ along path $p$ are given by:

$$
\begin{aligned}
&Spec_{exp}(p) = \\
&\quad \{n \ st \ n \ is \ a \ conditional \ and \ p \in CostPaths_{exp}(n)\} \\
&Cost_{exp}(p) = \sum_{p' \in P(p)} Freq(p'), \\
&\qquad \text{where } P(p) = \bigcup_{n \in Spec_{exp}(p)} CostPaths_{exp}(n) \\
&Benefit_{exp}(p) = Freq(p).
\end{aligned}
$$

It should be noted that if an expression $exp$ is available at a conditional node along each path leading to that node's entry or the expression is anticipable at the conditional node along all paths starting from the node's entry, then the set $CostPaths_{exp}$ at the node is empty since no additional executions of the expression will be introduced to enable hoisting above the node.

## 2.1 Handling Acyclic Graphs

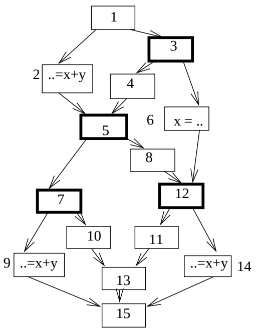
To implement the computation of cost and benefit information, rather than simply computing data flow values at various program points, we compute the set of paths along which the data flow value holds. The set of paths is represented by a bit vector in which each bit corresponds to a unique path from the entry to the exit of the acyclic flow graph with non-zero execution frequency according to the path profiles. To facilitate the computation of sets of paths, with each node $n$ in the flow graph, we associate a bit vector $OnPaths(n)$ where each bit corresponds to a unique path and is set to 1 if the node belongs to that path; otherwise it is set to 0.

To perform the *availability* analysis we associate with each node the following one bit variables: (a) $PRES_{exp}(n)$ which is 1(0) if $n$ preserves(kills) $exp$; and (b) $DGEN_{exp}(n)$ which is 1 if $n$ computes $exp$ which is then available at $n$'s exit; otherwise it is 0. The availability of an expression is computed using *forward* data flow analysis. At each node we compute the subset of paths through the acyclic flow graph along which the expression $exp$ is available. This set of paths is denoted as $N-AVPATHS_{exp}(n)(X-AVPATHS_{exp}(n))$ in the data flow equations given in Figure 3a. The set of paths in $N-AVPATHS_{exp}(n)$ is computed by unioning the set of paths along which expression is available at each of $n$'s predecessors. In order to ensure that we only consider those paths that pass through $n$, we intersect the result with $OnPaths(n)$. If $exp$ is available at $n$'s exit along all paths because $exp$ is in $DGEN_{exp}(n)$ then $X-AVPATHS_{exp}(n)$ is $OnPaths(n)$ and if $exp$ is not available at $n$'s exit then $X-AVPATHS_{exp}(n)$ is null (i.e., $\vec{0}$). Finally if $exp$ is available at $n$'s exit but not generated by $n$ then $X-AVPATHS_{exp}(n)$ is same as $N-AVPATHS_{exp}(n)$.

To perform the *anticipability* analysis we associate with each node the following one bit variables: (a) $PRES_{exp}(n)$ which is 1(0) if $n$ preserves(kills) $exp$; and (b) $UGEN_{exp}(n)$ which is 1 if $n$ computes $exp$ which is then anticipable at $n$'s entry; otherwise it is 0. The anticipability of an expression computed using *backward* data flow analysis. At each node we compute the subset of paths through the acyclic flow graph along which the expression is anticipable. This set of paths is denoted as $N-ANPATHS_{exp}(n)(X-ANPATHS_{exp}(n))$ in the data flow equations given in Figure 3b. The set of paths in $X-ANPATHS_{exp}(n)$ is computed by unioning the set of paths along which expression is anticipable at each of $n$'s successors. In order to ensure that we only consider those paths that pass through $n$, we intersect the result with $OnPaths(n)$. If $exp$ is anticipable at $n$'s entry along all paths because $exp$ is in $UGEN_{exp}(n)$ then $N-ANPATHS_{exp}(n)$ is $OnPaths(n)$ and if $exp$ is not anticipable at $n$'s entry then $N-ANPATHS_{exp}(n)$

is null (i.e., $\vec{0}$). Finally if $exp$ is anticipable at $n$'s entry but not generated by $n$, then $N-ANPATHS_{exp}(n)$ is same as $X-AVPATHS_{exp}(n)$.

Consider the example in Figure 1. The results of the above analysis are shown in the table in Figure 1d. Our analysis will determine that the paths that benefit from hoisting expression $x+y$ above node 6 are $P1$ and $P2$ and hence the total benefit of speculation is 200. Furthermore in order to achieve this benefit, the path that incurs a cost is $P5$ whose frequency is 90. Thus, there is an overall benefit to allow speculation at node 6. Another example of this analysis is shown in Figure 4. Later in section 3 we will show how this information is used to enable speculation at 5, 7 and 12 while disabling speculation at node 3. This results in benefit along paths $P1$ and $P4$ whose frequencies add up to 150. The paths that incur a cost include paths $P6$, $P7$ and $P9$ whose frequencies add up to 91. Notice that if speculation is enabled also at node 3, the cost incurred will increase by 50 (the frequency of path $P10$) while no additional benefit will result.




(a)

| Freq. | Path |
|---|---|
| 100 | P1: 1-2-5-7-9-15 |
| 1 | P2: 1-2-5-7-10-13-15 |
| 1 | P3: 1-2-5-8-12-11-13-15 |
| 50 | P4: 1-2-5-8-12-14-15 |
| 50 | P5: 1-3-4-5-7-9-15 |
| 40 | P6: 1-3-4-5-7-10-13-15 |
| 50 | P7: 1-3-4-5-8-12-11-13-15 |
| 1 | P8: 1-3-4-5-8-12-14-15 |
| 1 | P9: 1-3-6-12-11-13-15 |
| 50 | P10: 1-3-6-12-14-15 |

(b)

CostPaths(3) = {P6,P7,P9,P10}
BenefitPaths(3) = {}

CostPaths(5) = {P6,P7}
BenefitPaths(5) = {P1,P4}

CostPaths(7) = {P6}
BenefitPaths(7) = {P1}

CostPaths(12) = {P7,P9}
BenefitPaths(12) = {P4}

Cost(P1) = Freq(P6+P7) = 90

Cost(P4) = Freq(P6+P7+P9) = 91

(c)

Figure 4: Example of Cost-Benefit Analysis.

## 2.2 Handling Loops

In the presence of loops, the paths considered only include those paths that do not cross loop boundaries. The program is viewed as a collection of acyclic subgraphs and only paths within these subgraphs are profiled. The application of our optimization described in the preceding section will perform speculation within an acyclic graph if it is beneficial with respect to the profiles for that acyclic graph. However, in order to achieve beneficial speculative movement of expressions out of loops, simple extension is needed in the treatment of the loop header node. Traditional PRE is able to move invariant expressions out of loops while speculative motion out of loops enables movement of *conditionally invariant* expressions out of loops as long as the expressions are invariant along more frequently executed paths through the loop.

In Figure 5a the dashed lines indicate the edges that are ignored during initial cost-benefit analysis. Notice that the exclusion of these edges will result in acyclic graphs corresponding to the code before the loop, after the loop, and the loop body. The cost and benefit of speculatively moving a conditionally invariant expression out of the loop is associated with the loop header node. The paths that contribute to the cost of moving a conditionally invariant expression out of the loop include paths in the loop along which expression evaluation must remain in the loop ($\overline{ANPATHS_{exp}(header)}$), the paths prior to the loop along which evaluations of the expression must be placed ($\overline{AVPATHS_{exp}(pre-header)}$), and the paths following the loop along which expression is not anticipable ($\overline{ANPATHS_{exp}(post-exit)}$). The paths that contribute to the benefit include the paths in the loop from which the evaluation of the expression is removed ($ANPATHS_{exp}(header)$), the paths prior to the loop along which expression is available ($AVPATHS_{exp}(pre-header)$) and the paths following the loop along which an expression evaluation becomes unnecessary ($ANPATHS_{exp}(post-exit)$).

$$CostPaths_{exp}(header) = \overline{ANPATHS_{exp}(header)} \vee \overline{AVPATHS_{exp}(pre-header)} \vee \overline{ANPATHS_{exp}(post-exit)}$$
$$BenefitPaths_{exp}(header) = ANPATHS_{exp}(header) \vee AVPATHS_{exp}(pre-header) \vee ANPATHS_{exp}(post-exit)$$

The example in Figures 5b-c illustrates this analysis. The analysis also indicates that enabling speculation at the loop header is beneficial (i.e., $Cost_{x+y}(5) < Benefit_{x+y}(5)$) and hence the conditionally invariant expression $x+y$ must be moved out of the loop.

It should be noted that the cost-benefit analysis of individual acyclic graphs is not performed separately. Instead the analysis over the entire program is performed simultaneously with respect to a given *expression*. The edges that cross loop boundaries are simply ignored during this analysis phase. Our approach performs PRE one expression at a time. As pointed out by Chow et al. [6] this approach has many advantages over the traditional approach that considers all expressions simultaneously. It lowers the maximum storage requirement of optimizing all expressions and allows exploitation of nesting relationship in expression trees to speed up the optimization of large expressions. It

$$\mathbf{N-AVPATHS_{exp}(n)} = \begin{cases} \vec{\mathbf{0}} & \textbf{if } \mathbf{n = entry} \\ \mathbf{OnPaths(n)} \wedge \bigvee_{\mathbf{m \in Pred(n)}} \mathbf{X-AVPATHS_{exp}(m)} & \textbf{otherwise} \end{cases}$$

$$\mathbf{X-AVPATHS_{exp}(n)} = \begin{cases} \mathbf{OnPaths(n)} & \textbf{if } \mathbf{DGEN_{exp}(n) = 1} \\ \mathbf{N-AVPATHS_{exp}(n)} & \textbf{elseif } \mathbf{PRES_{exp}(n) = 1} \\ \vec{\mathbf{0}} & \textbf{otherwise} \end{cases}$$

$$\mathbf{X-ANPATHS_{exp}(n)} = \begin{cases} \vec{\mathbf{0}} & \textbf{if } \mathbf{n = exit} \\ \mathbf{OnPaths(n)} \wedge \bigvee_{\mathbf{m \in Succ(n)}} \mathbf{N-ANPATHS_{exp}(m)} & \textbf{otherwise} \end{cases}$$

$$\mathbf{N-ANPATHS_{exp}(n)} = \begin{cases} \mathbf{OnPaths(n)} & \textbf{if } \mathbf{UGEN_{exp}(n) = 1} \\ \mathbf{X-ANPATHS_{exp}(n)} & \textbf{elseif } \mathbf{PRES_{exp}(n) = 1} \\ \vec{\mathbf{0}} & \textbf{otherwise} \end{cases}$$

$$\forall \mathbf{n} \textbf{ st } \mathbf{n} \textbf{ is a conditional node}:$$

$$\mathbf{COSTPATHS_{exp}(n)} = \overline{\mathbf{N-AVPATHS}}_{\mathbf{exp}}\mathbf{(n)} \wedge \overline{\mathbf{N-ANPATHS}}_{\mathbf{exp}}\mathbf{(n)}$$

$$\mathbf{BENEFITPATHS_{exp}(n)} = \mathbf{N-AVPATHS_{exp}(n)} \wedge \mathbf{N-ANPATHS_{exp}(n)}$$

$$\forall \mathbf{p} \textbf{ st } \mathbf{p} \in \mathbf{BENEFITPATHS_{exp}(n)} \textbf{ for some conditional } \mathbf{n}:$$

$$\mathbf{SPEC_{exp}(p)} = \{\mathbf{n} \textbf{ st } \mathbf{n} \textbf{ is a conditional and } \mathbf{p} \in \mathbf{COSTPATHS_{exp}(n)}\}$$

$$\mathbf{COST_{exp}(p)} = \sum_{\mathbf{p}' \in \bigcup_{\mathbf{n \in SPEC_{exp}(p)}} \mathbf{COSTPATHS_{exp}(n)}} \mathbf{FREQ(p')}$$

$$\mathbf{BENEFIT_{exp}(p)} = \mathbf{FREQ(p)}$$

Figure 3: Cost-Benefit Data flow Analysis.

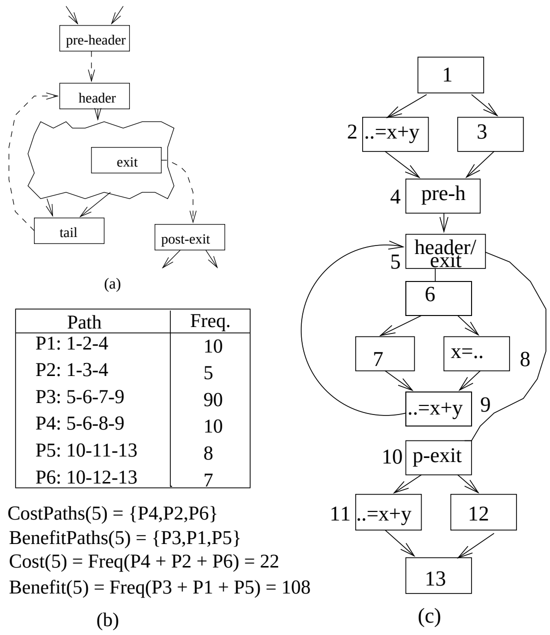



Figure 5: Cost-Benefit Analysis for Loops.

also allows the optimizer to focus on more important expressions while ignoring others and provide new opportunities for code reshaping.

## 3 Speculative Hoisting Framework

A number of solutions to PRE have been proposed [7, 13, 11]. In this section we present an integration of speculation with the expression hoisting framework proposed by Steffen [13] to perform PRE. The modifications to Steffen's framework that are necessary to incorporate speculation are described in this section. Before this modified code motion can be used we must identify the conditional nodes at which speculation is to be enabled.

### 3.1 Enabling Speculation

To determine the conditional nodes in an acyclic subgraph at which speculation should be enabled we first identify all paths that contain some redundancy ($RedPaths_{exp}$). These paths can be easily obtained by unioning the $BenefitPaths_{exp}$ sets of all conditionals. The enabling of speculation is carried out in three phases by the algorithm in Figure 6. The algorithm sets the variable $SpecCond_{exp}(c)$ for conditional $c$ to true or false depending upon whether or not speculation is to be enabled or disabled at $c$.

In the first phase we discard paths from $RedPaths_{exp}$ along which the removal of redundancy can never result in an overall benefit. To do so we examine each conditional node $n$ and compute the maximum benefit that can result by speculation at that node and the minimum cost that will accompany speculation at $n$. The maximum benefit is computed by summing the frequencies of paths in $BenefitPaths_{exp}(n)$ while the minimum cost is computed by summing the frequencies of paths in $CostPaths_{exp}(n)$. If the maximum benefit is less than the minimum cost then speculation should never be enabled at $n$. Thus, each path in $RedPaths_{exp}$ which requires speculation at $n$ for removal of its redundancy is removed from consideration.

**Initialization:**
Let $C$ be the set of program conditional nodes.
$\forall c \in C$, $SpecCond_{exp}(c) = false$
$RedPaths_{exp} = \bigcup_{c \in C} BenefitPaths_{exp}(c)$
**Phase 1:**
**for each** conditional $c \in C$ **do**
  $MaxBenefit = \sum_{p \in BenefitPaths_{exp}(c)} Freq(p)$
  $MinCost = \sum_{p \in CostPaths_{exp}(c)} Freq(p)$
  **if** $MaxBenefit < MinCost$ **then**
    **for each** path $p \in RedPaths_{exp}$ **do**
      **if** $c \in Spec_{exp}(p)$ **then** $RedPaths_{exp} = RedPaths_{exp} - \{p\}$
    **endfor**
  **endif**
**endfor**
**Phase 2:**
**for each** $p \in RedPaths_{exp}$ **do**
  **if** $Benefit_{exp}(p) > Cost_{exp}(p)$ **then**
    **for each** $c \in Spec_{exp}(p)$ **do** $SpecCond_{exp}(c) = true$
    $RedPaths_{exp} = RedPaths_{exp} - \{p\}$
  **endif**
**endfor**
**Phase 3:**
$change = true$;
**while** $change$ **do**
  **for each** path $p \in RedPaths_{exp}$ **do**
    $AddCostPaths_{exp}(p) = \bigcup_{\overline{SpecCond_{exp}(c)} \wedge c \in Spec_{exp}(p)} CostPaths_{exp}(c)$
    $AddCost_{exp}(p) = \sum_{p' \in AddCostPaths_{exp}(p)} Freq(p')$
    **if** $Benefit_{exp}(p) > AddCost_{exp}(p)$ **then**
      $change = true$;
      **for each** $c \in Spec_{exp}(p)$ **do** $SpecCond_{exp}(c) = true$
      $RedPaths_{exp} = RedPaths_{exp} - \{p\}$
    **endif**
  **endfor**
  $change = change \wedge RedPaths_{exp} \neq \phi$
**endwhile**

Figure 6: Enabling Speculation in Acyclic Graphs.

In the second phase we identify paths along which removal of redundancy using speculation is beneficial irrespective of whether or not the speculation performed to remove this redundancy enables removal of redundancy along any other path. We consider each path in $RedPaths_{exp}$ and determine if speculation must be enabled at appropriate conditionals along this path to exploit this benefit. Speculation is enabled at conditionals along path $p$ that are in the set $Spec_{exp}(p)$ if the benefit of removing redundancy along the path (i.e., $Benefit_{exp}(p)$) is greater than the cost (i.e., $Cost_{exp}(p)$).

In the third phase we identify paths along which removal of redundancy is only beneficial if it is carried out in conjunction with some other paths. This situation arises because sometimes the cumulative benefit of removing redundancy along two paths is greater than the cumulative cost while for one of the paths individually, the benefit of removing redundancy may be lower than the cost of removing redundancy. The cause for such a situation is the presence of common conditionals in the $Spec$ sets of the two paths. Finally it should be noted that at the end of the algorithm the set $RedPaths_{exp}$ need not be empty. In fact it will contain all those paths along which redundancy was detected but not removed because our algorithm considered the removal of this redundancy not to be profitable (i.e., benefit exceeds the additional cost).

As we know application of PRE may increase register pressure since the value of the expression that is optimized may now have to be kept in a register for long periods of time. The above algorithm can be easily adapted to be sensitive to register pressure. If the benefit of removing redundancy along a path is low, and speculation at additional conditional nodes must be enabled to remove the redundancy, then it might be better not to remove redundancy along this path. This will avoid increase in register pressure that would have resulted from enabling speculation at additional conditional nodes.

Let us consider the example in Figure 1. In this example there are two paths $P1$ and $P2$ which can benefit from speculation above node 6. Since $Benefit_{x+y}(P1)$ is equal to 100 and $Cost_{x+y}(P1)$ is 90, in the first phase of enabling speculation, we enable speculation at node 6 to allow removal of redundancy along path $P1$. The same condition also holds for path $P2$. However, since speculation has already been enabled at node 6, no additional actions are required to remove redundancy along path $P2$.

In the example in Figure 4a there are two paths along which redundancy exists, paths $P1$ and $P4$. Since $Benefit_{x+y}(P1)$ is equal to 100 while $Cost_{x+y}(P1)$ is 90 (sum of the frequencies of paths $P6$ and $P7$), in the first phase of the algorithm speculation is enabled at conditionals in $Spec_{exp}(P1)$, that is, nodes 5 and 7. On the other hand, since $Benefit_{x+y}(P4)$ is equal to 50 while $Cost_{x+y}(P4)$ is 91 (sum of the frequencies of paths $P6$, $P7$, and $P9$), removal of redundancy is not considered to be profitable during the first phase. During the second phase we determine that since speculation has already been enabled at node 5, the additional cost of removing redundancy along path $P4$ is equal to 1 (the frequency of path $P9$ that incurs a cost when speculation is enabled at node 12). The additional cost of 1 is less than the benefit of removing redundancy along path $P4$ which is 50. Thus, speculation at 12 is also enabled and all redundancy will be removed in this example. The total benefit is 150 while the total cost incurred is 91. Finally it should be noted that while speculation at conditional nodes 5, 7 and 12 was enabled, it was not enabled for conditional node 3. This is because there is no additional benefit that will result from hoisting above 3 while there is a cost associated with such hoisting.

The algorithm that we have presented is a heuristic that is guaranteed to enable speculation in such a way that the overall benefit of speculation is more than the cost of enabling speculation. However, the algorithm cannot always detect situations in which enabling speculation is beneficial. For the example in Figure 4a if the frequency of path $P6$ was 60 instead of 40, then in the first phase speculation will not be enabled for path $P1$. Hence the algorithm will not remove redundancy along paths $P1$ and $P4$. However, if we were to enable speculation at conditional nodes 5, 7 and 12, we would observe that the overall benefit is 150 while the cost is 111. This situation can be detected if instead of considering paths $P1$ and $P4$ individually, as is done by our algorithm, we consider them together. This would reveal that while individually their costs are greater than their benefits, taken together their cost is lower than the benefit. In general we will have to consider each combination of conditional nodes (in pairs, triples etc.) to determine whether their cumulative benefit is greater than their cumulative cost. Such an algorithm will in general have exponential complexity hence we use the heuristic presented in Figure 6. However, in practice if the number of remaining paths in $RedPaths_{exp}$ is small, it may be entirely possible to consider combinations of paths to look for detecting additional profitable optimization opportunities.

After the application of the above algorithm to acyclic subgraphs we examine loop header nodes to decide whether speculative hoisting of an expression out of the loop should be enabled. If the value of $Cost_{exp}(header)$ is less than the value of $Benefit_{exp}(header)$, then speculation is enabled at the *header* node. Moreover speculation at other conditionals along the paths in $ANPATHS_{exp}(header)$ and $ANPATHS_{exp}(post-exit)$ is also enabled to allow hoisting of $exp$ along these paths. For the example in Figure 5c, according to the above rules speculation at nodes 5, 6 and 10 is enabled.

### 3.2 Expression Hoisting

The original analysis in [11] consists of a backward data flow analysis phase followed by a forward data flow analysis phase. Backward data flow is used to identify all *down-safe* points, that is, points to which expression evaluations can be safely hoisted. Forward data flow analysis identifies the *earliest* points at which expression evaluations can be placed. Finally, the expression evaluations are placed at points that are both earliest and down-safe. We modify the *down-safety* analysis to take advantage of speculation past the conditional nodes where speculation has been enabled. The *earliestness analysis* remains unchanged. Next we discuss the modification to down-safety analysis.

The equations for computing down-safety of an expression $exp$, denoted by $DS_{exp}$, are given below.

$$
\begin{aligned}
DS_{exp}(n) = \; & if\ n = exit\ \vee\ N-AVPATHS_{exp}(n) = \vec{0} \\
& then\ 0 \\
& elseif\ N-AVPATHS_{exp}(n) \neq \vec{0}\ then \\
& \quad Used_{exp}(n) \vee \\
& \quad (Pres_{exp}(n) \wedge \bigwedge_{m \in Succ(n)} DS_{exp}(m)) \vee \\
& \quad SpecCond_{exp}(n)
\end{aligned}
$$

Node $n$ is down-safe if one of the following conditions is true: (i) $exp$ is computed in $n$ (i.e., $Used_{exp}(n) = 1$); (ii) $exp$ is preserved by $n$ (i.e., $Pres_{exp}(n) = 1$) and $exp$ is anticipatable at $n$'s exit along each path from $n$ to the end of the program; or (iii) $n$ is a conditional node at which speculation of $exp$ has been enabled, that is, $SpecCond_{exp}(n)$ is true. The first two conditions are used in Steffen's original expression hoisting framework while the third condition has been included to enable useful speculation. Finally notice that hoisting of an expression above node $n$ is suppressed if there is no path along which the expression is available at $n$ (i.e., $N-AVPATHS_{exp}(n) = \vec{0}$ which is computed during cost-benefit analysis). This is because further hoisting will simply lengthen the expression's live range.

The application of the above analysis to the examples in Figures 4 and 5 is shown in Figure 7. In the first example by enabling speculation at conditional nodes 5, 7 and 12, the expression is allowed to propagate above these conditionals during the down-safety phase. The propagation cannot proceed above node 6 due a definition of $x$ in 6. Even though node 4 is found to be down-safe, propagation is not allowed to proceed above 3 because speculation is not enabled at 3. Therefore the final placement of the expression evaluations is found to be nodes 2, 4 and 6. In the second example, speculation is enabled at node 5 (loop header) and hence the conditional nodes 6 and 10 (post-exit). The final resulting placement is able to reduce the evaluations of $x+y$ along the frequently executed path $P3$ (5-6-7-9) through the loop body.

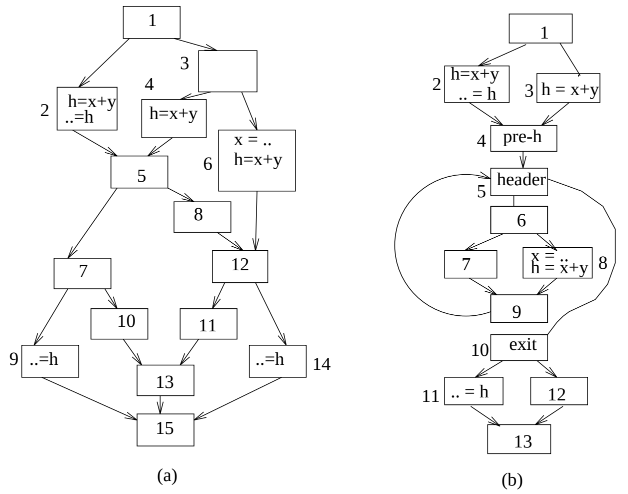


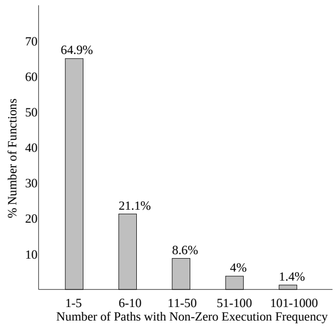

Figure 7: Speculative PRE.

## 4 Compile-time Cost

An important component of the cost of the analysis described in the preceding sections depends upon the number of paths which are being considered during cost-benefit analysis. In general the number of static paths through a program can be extremely large. However, in practice the number of paths that need to be considered by the cost-benefit analysis is quite small. First only the paths with non-zero execution counts need to be considered. Second only the paths through a given function are considered at any one time.

Figure 8 shows the summary of the characteristics of path profiles for seven SPEC95 integer benchmarks (`compress`, `go`, `li`, `ijpeg`, `m88ksim`, `perl`, `gcc`, `vortex`). The bar graph shows that in 65% of the functions that were executed no more than 5 paths with non-zero frequency were found and only 1.4% of functions had over 100 paths. Moreover, no function had greater than 1000 paths with non-zero execution count.

An approach for further reducing the number of paths being considered in the analysis is to include enough paths with non-zero frequency such that these paths account for the majority of the execution time of the program. The table in Figure 8 shows how the number of functions that contain up to 5, 10, 50, 100, and 1000 paths with non-zero frequency changes as we consider enough paths to account for 100% down to 75% of the program execution time. As we can see the number of functions that require at most 5 paths increases substantially (from 1694 to 2304) while the number of functions that require over hundred paths reduces significantly (from 35 to 1). The final graph shows the maximum number of paths that we ever have to consider in a function from among all the functions. Again this maximum value reduces sharply (from 1000 to 103) as the paths considered account for less than 100% of the program execution time.




| Number of Paths | Number of Functions | | | | | |
|---|---|---|---|---|---|---|
| | 100% | 95% | 90% | 85% | 80% | 75% |
| 1-5 | 1694 | 2022 | 2133 | 2235 | 2304 | 2371 |
| 6-10 | 550 | 295 | 279 | 228 | 195 | 162 |
| 11-50 | 225 | 257 | 181 | 138 | 109 | 75 |
| 51 - 100 | 105 | 25 | 15 | 7 | 0 | 0 |
| 101-1000 | 35 | 10 | 1 | 1 | 1 | 1 |

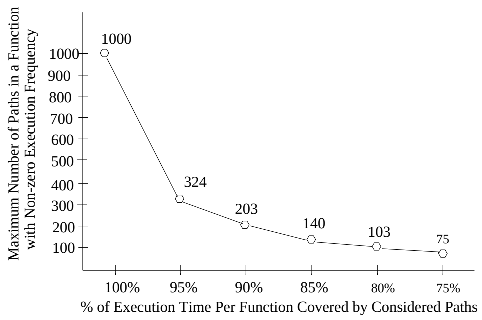



Figure 8: Characteristics of Path Profiles for SPEC95.

Ignoring the low frequency paths as proposed above will result in approximate cost-benefit information. Our objective is to compute conservative estimates of cost by ensuring that the estimated cost is no less than the actual cost while the estimated benefit is no more than the actual benefit.

Consider the cost-benefit analysis for node 7 in the flow graph shown in Figure 9. The frequencies of three paths along which benefit is observed and one path along which cost is observed are given in the figure. Let us assume that we ignore the two paths with a low execution frequency of 10 during the analysis. Along both these paths benefit would be observed if precise analysis were carried out and this benefit will not be observed by conservative cost-benefit analysis. As a result while the actual benefit is 120, conservative benefit is found to be 100. Similarly in computing the cost we will assume that along the two paths which were ignored by the analysis additional evaluations of the expression must be placed. Thus, while the actual cost is 25, conservative cost is estimated to be 45.

The equations for computing the conservative estimates of cost and benefit are given below. In these equations we assume that paths $HighPaths$ and $LowPaths$ represent the high frequency and low frequency paths. The $CostPaths_{exp}$ information is computed only for $HighPaths$. Thus, in computing the cost at node $n$ we obtain a conservative estimate by assuming that evaluations of $exp$ will be placed along all paths in $LowPaths$ that contain $n$. In computing the conservative estimate of the benefit we use $BenefitPaths_{exp}(n)$ information computed only for $HighPaths$, that is, it is assumed that none of the paths in $LowPaths$ benefit.

$$Cost_{exp}(n) \leq EstCost_{exp}(n) = \sum_{p \in HighPaths} CostPaths_{exp}(n)(p) \times Freq(p) + \sum_{\substack{p \in \text{LowPaths} \\ \wedge n \in p}}^{Total} Freq(p)$$

$$Benefit_{exp}(n) \geq EstBenefit_{exp}(n) = \sum_{p \in HighPaths} BenefitPaths_{exp}(n)(p) \times Freq(p)$$

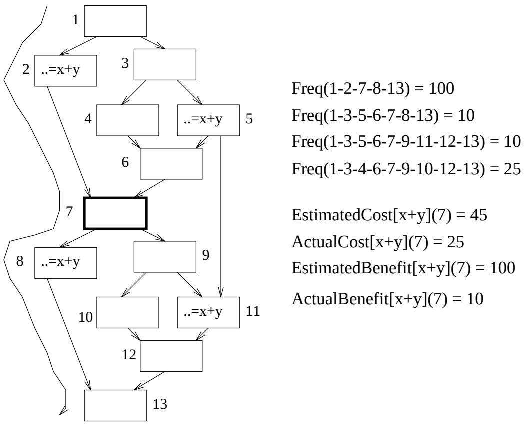



Figure 9: Conservative Cost-Benefit Analysis.

## 5 Related Work

In previous work by Fang [7] the use of speculation during partial redundancy elimination was also proposed. However, the algorithms in [7] apply to a single-entry-single-exit acyclic region and no cost-benefit analysis to guide speculation is presented. Independently of our work Horspool and Ho [10] have recently proposed a profile driven PRE algorithm based upon edge profiles. However, our algorithm is superior to theirs because it is more powerful, flexible and efficient.

Our algorithm is more powerful because any optimization that is achievable using the node profiles by Horspool and Ho will also be achieved by our algorithm. However, our algorithm achieves optimization in situations where the node profile based algorithm either fails to discover any optimization opportunity or performs optimization to a lesser degree. We illustrate the superiority of our algorithm using the example in Figure 10a. In this example there are two paths, path $P1$ (1-2-3-5-7-8-15) and path $P4$ (1-9-11-12-13-8-15), along which redundancy exists. Removal of redundancy along path $P1$ requires enabling speculation at node 5 and therefore it results in introduction of redundancy along path $P2$ (1-2-4-5-6-15). Similarly removal of redundancy along path $P4$ requires enabling speculation at node 12 and therefore it results in introduction of redundancy along path $P3$ (1-9-10-12-14-15). Consider the treatment of this example by our path profiling based algorithm for the two sample profiles given in Figures 10b and 10c. For both profiles the benefit of enabling speculation at node 5 (i.e., $Freq(P1) = 100$) outweights the cost (i.e., $Freq(P2) = 50$). On the other hand for both profiles the benefit of enabling speculation at node 12 (i.e., $Freq(P4) = 25$) is less than the cost (i.e., $Freq(P3) = 50 \; or \; 75$). Thus our algorithm will enable speculation at 5 and disable it at node 12 resulting in the PRE transformation shown in Figure 10d. The overall benefit of this transformation is 50 (=100-50) for both profiles.

Now let us consider the treatment of this example by the algorithm of Horspool and Ho. The basic approach of this algorithm is to identify points in the program at which placement of evaluations of $x + y$ will cause it be become available at node 8 along all paths and hence removal of $x + y$ from node 8 will become possible. In this example the identified nodes will be nodes 4 and 10. The benefit of applying the transformation is equal to the frequency of node 8 ($Freq(8)$) while the cost is equal to the sum of frequencies of nodes 4 and 10 ($Freq(4) + Freq(10)$. The execution frequency can be computed from the frequency of edges entering the node.

For the first sample profile of Figure 10b, the benefit is 125 while the cost is 100 and therefore the program will be transformed as shown in Figure 10e. The overall benefit of this transformation is 25 which is less than the benefit of 50 achieved by our algorithm based upon path profiles. For the second sample profile of Figure 10c, both the benefit and cost are 125 and the PRE transformation will not be applied by Horspool and Ho while our algorithm achieves a benefit of 50. Thus, our algorithm is more powerful, since unlike Horspool and Ho's algorithm, it evaluates the merit of removing redundancy for each relevant path individually. Horspool and Ho cannot perform such evaluation since they do not perform path based analysis that can take advantage of path profile information. They instead either enable speculation at both 5 and 12 or at neither of the two conditionals.

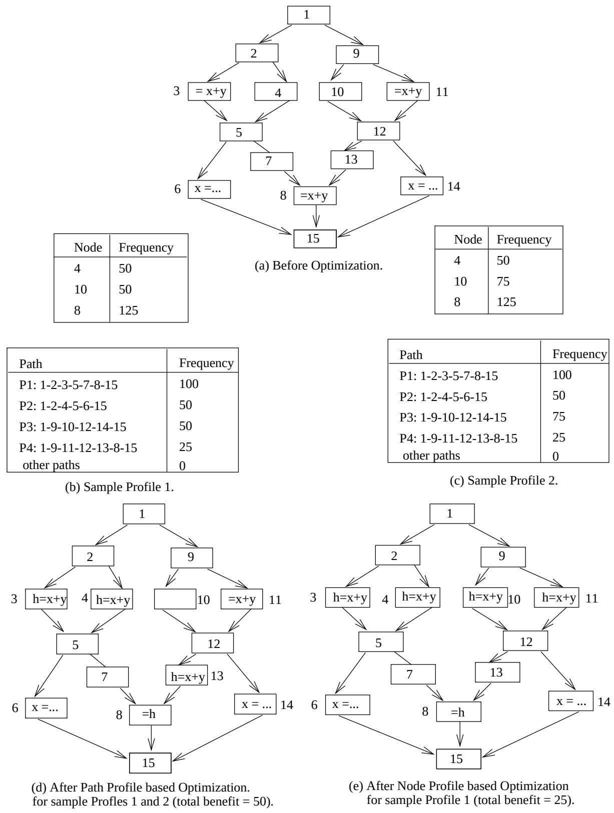



Figure 10: Node Profiles vs Path Profiles.

Our approach provides additional flexibility since it allows control over the compile-time cost of cost-benefit analysis as shown in section 4. It is also possible to ignore paths along which optimization yields very little benefit in order to avoid increase in register pressure. On the other hand it is not possible to trade-off the precision of Horspool and Ho's cost-benefit analysis with its complexity since the cost-benefit analysis is integrated with the algorithm for determining the correct placement points.

| Number of Blocks/Paths | Number(Percentage) of Functions | |
|---|---|---|
| | Total Blocks | Paths Executed |
| 1-5 | 385(14.8%) | 1694(64.9%) |
| 6-10 | 453(17.4%) | 550(21.1%) |
| 11-50 | 1289(49.4%) | 225(8.6%) |
| 51 - 100 | 322(12.3%) | 105(4%) |
| 101-1000 | 160(6.1%) | 35(1.4%) |

Figure 11: Block Counts vs Executed Path Counts.

Since the analysis of Horspool and Ho's method collects nodes at which expression evaluation must be placed to achieve must availability of that expression at each node, the data flow variables take the form of a bit bector whose size equals the number of nodes in the control flow graph of the function being optimized. In contrast the cost-benefit analysis that we perform requires bit vectors whose size equals the number of paths through the function with non-zero execution frequency. As shown in Figure 11, the number of executed paths is significantly smaller than the number of basic blocks in a function. For example, according to Figure 11, only 14.8% of the executed functions in the SPEC95 integer benchmarks had up to five basic blocks while 64.9% of the functions had up to five paths with non-zero execution frequency. Thus our method will use significantly smaller bit vectors than Horspool and Ho's method.

Finally an alternative to speculation is to enable code motion by restructuring the control flow graph. This approach does not require hardware support for delaying or suppressing the reporting of exceptions. In general restructuring can result in exponential code growth. However, by measuring the cost of enabling the optimization in terms of the code growth, it is possible to limit optimization to the extent possible with limited amount of growth. This approach has already been applied to PRE [3], partial dead code elimination [4] and elimination of partially redundant conditional branches [5].

## 6 Concluding Remarks

We demonstrated the use of path profile information in combining the use of speculation with the PRE optimization in order to aggressively optimize frequently executed paths through a program. In conclusion, the approach we have presented is quite general and can be adapted for application to other optimizations. For example predicated execution can be used to enable code sinking past merge points resulting in the removal of dead code along frequently executed paths [8]. We have also developed analysis techniques in which redundancy and dead code removal decisions are influenced by functional unit resource availability [9]. Other applications of our approach include strength reduction along certain paths and elimination of partially redundant loads and dead stores [2].

## References


[1] T. Ball and J. Larus, "Efficient path profiling," *29th Annual IEEE/ACM International Symposium on Microarchitecture*, Paris, France, 1996.

[2] R. Bodik and R. Gupta, "Array Data-Flow Analysis for Load-Store Optimizations in Superscalar Architectures," *International Journal of Parallel Programming*, Vol. 24, No. 6, pages 481-512, 1996.

[3] R. Bodik and R. Gupta, "Partial Dead Code Elimination using Slicing Transformations," *ACM SIGPLAN Conf. on Prog. Language Design and Implementation*, pages 159-170,Las Vegas, Nevada, June 1997.

[4] R. Bodik, R. Gupta, and M.L. Soffa, "Interprocedural Conditional Branch Elimination," *ACM SIGPLAN Conf. on Prog. Language Design and Implementation*, pages 146-158, Las Vegas, Nevada, June 1997.

[5] R. Bodik, R. Gupta and M.L. Soffa, "Complete Removal of Redundant Computations," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Montreal, Canada, June 1998.

[6] F. Chow, S. Chan, R. Kennedy, S-M. Liu, R. Lo, and P. Tu, "A New Algorithm for Partial Redundancy Elimination based upon SSA Form," *ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 273-286, Las Vegas, Nevada, June 1997.

[7] J.Z. Fang, "Compiler Algorithms on If-Conversion, Speculative Predicates Assignment and Predicated Code Optimizations," *Proc. of the Ninth Workshop on Languages and Compilers for Parallel Computers*, San Jose, California, August 1996.

[8] R. Gupta, D. Berson, and J.Z. Fang, "Path Profile Guided Partial Dead Code Elimination using Predication," *International Conference on Parallel Architectures and Compilation Techniques*, pages 102-115, San Francisco, Ca, November 1997.

[9] R. Gupta, D. Berson, and J.Z. Fang, "Resource-Sensitive Profile-Directed Data Flow Analysis for Code Optimization," *The 30th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 558-568, Research Triangle Park, NC, December 1997.

[10] R.N. Horspool and H.C. Ho, "Partial Redundancy Elimination Driven by a Cost-Benefit Analysis," *8th Israeli Conf. on Computer Systems and Software Engineering*, pages 111-118, Herzliya, Israel, June 1997.

[11] J. Knoop, O. Ruthing, and B. Steffen, "Lazy Code Motion," *Proc. of Conf. on Programming Language Design and Implementation*, pages 224-234, 1992.

[12] S.A. Mahlke, et al. "Sentinel Scheduling for VLIW and Superscalar Processors," *Proc. of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992.

[13] B. Steffen, "Data Flow Analysis as Model Checking," *Proceedings TACS'91*, Sendai, Japan, Springer-Verlag, LNCS 526, pages 346-364, 1991.