Generalized Dominators [*]

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

Abstract

The notion of dominators is generalized to include multiple-vertex dominators in addition to traditional single-vertex dominators. A multiple-vertex dominator of a vertex is a group of vertices that collectively dominate the vertex. An algorithm for computing immediate multiple-vertex dominators is presented. The immediate dominator information is expressed in the form of a directed acyclic graph referred to as the DDAG. The generalized dominator set of any vertex can be computed from the DDAG. The single-vertex dominator information restricts the propagation of loop invariant statements and array bound checks out of loops. Generalized dominator information avoids these restrictions.

Keywords - control flow graph, code optimization, loop invariants, array bound checks, code hoisting.

1 Introduction

Algorithms for performing certain global code optimizations require the availability of dominator information. A vertex v in a program control flow graph (CFG) dominates another vertex w, if every path from the beginning of the CFG to vertex w contains v. The problem of computing dominators has been extensively studied and several algorithms have been developed [5, 6, 7, 10, 4]. In previous research only single-vertex dominators have been considered. The single-vertex dominators prohibit the full exploitation of optimizations. For example, consider the code segment shown in Figure 1(i). This code segment can be optimized by propagating a loop invariant subscript expression and array bound checks (shown in italics) out of the loop resulting in the code segment shown in Figure 1(ii). However, none of these optimizations can be directly performed using single-vertex dominators since loop invariants and array bound checks are typically moved out of a loop if they belong to a vertex in the loop which dominates all loop exits [3].

In this paper the notion of dominators is generalized to include sets of vertices which *collectively dominate* a given vertex. This generalization enables the optimization of code in Figure 1. The vertices B1 and B2 together dominate the loop exit and therefore these invariant expression and bound checks that are present in both B1 and B2 can be moved out of the loop.

[*]An extended version of this paper appeared in POPL'92. This work is supported in part by the National Science Foundation through a Presidential Young Investigator Award CCR-9157371 to the University of Pittsburgh.

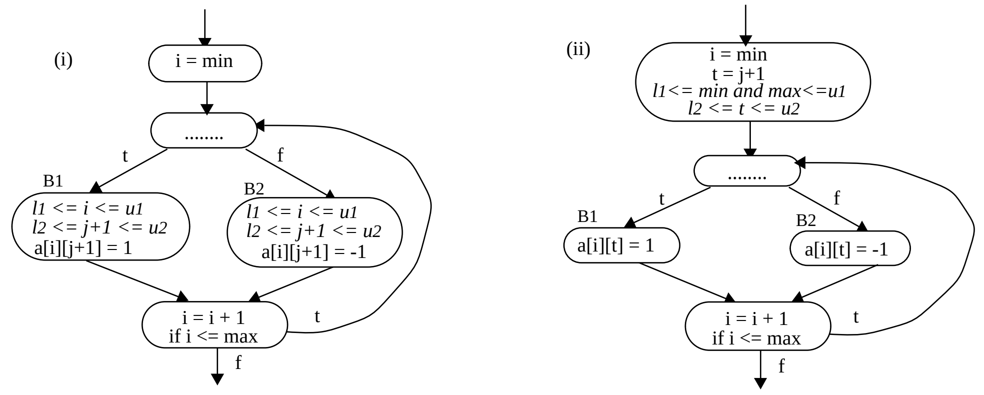


Figure 1: Propagation of Loop Invariants and Array Bound Checks.

2 Generalized Dominators

The *generalized dominator* set of a vertex v, denoted as $D(v)$, consists of sets containing single-vertex and multiple-vertex dominators of v. Therefore, each member of a generalized dominator set is a set of vertices.

Definition: A set of vertices V *dominates* vertex v iff the following two conditions are met:

1. all paths from the beginning of the program to vertex v contain some vertex $w \in V$; and
2. for each vertex $w \in V$, there is at least one path from the beginning of the program to vertex v which contains w and does not contain any other vertex in V.

In Figure 1 the set {B1,B2} dominates the loop exit since all paths from the start to the loop exit must pass through either B1 or B2. The above definition encompasses the traditional notion of single-vertex dominators since it reduces to the definition of a single-vertex dominator if the cardinality of set V is one. The second condition in the above definition leads to the following lemma.

Lemma 1: Given a set of vertices V which dominates vertex v, there does not exist any set of vertices $V' \subset V$ such that V' dominates v.

Proof: Let us assume that there is a set $V' \subset V$ which dominates vertex v. Let w be a vertex in set V which does not belong to V'. Since V dominates v there is a path from the start of the program to v which contains w and does not contain any other vertex in V. Since $V' \subset V$ this path does not pass through any vertex in V'. This contradicts our assumption that V' dominates v. Thus, there is no V' which dominates v. □

An algorithm developed by Purdom and Moore [7] computes single-vertex dominators for all the vertices in a CFG. In this algorithm the dominator set for a vertex v is computed by taking the intersection of the dominator sets of (immediate) predecessors of v. The algorithm is based upon the observation that if a vertex v is dominated by another vertex u, then u must also dominate all predecessors of vertex v. Along the same lines the following observation can be made regarding generalized dominators. If a vertex v is dominated by a set of vertices V, then for each predecessor of v, say p, V must contain a set of vertices which dominates p. This observation implies that if we consider a combination of vertices which is obtained by unioning together a dominator of each predecessor of a vertex v, then this combination must contain a set of vertices which dominates v. In theorem 1 we prove this result formally. However, in order to do so we first prove the following lemma.

Lemma 2: Given a set of vertices V which dominates vertex v, there does not exist any $V' \subset V$ which dominates a vertex $w \in V - V'$.

Proof: Let us assume that V' dominates $w \in V - V'$. All paths from the start of the program to vertex w must pass through some vertex in V'. Thus, there is no path from the start of the program to vertex v which contains w and does not contain any other vertex in V' and hence V. However, this contradicts our assumption that V dominates v. Thus, there is no V' which dominates a vertex $w \in V - V'$. $\Box$

Theorem 1: Let d denote a set $\cup_{i=1}^{|Pred(v)|} d_i$ such that $d_i \in D(p_i)$, where p_i is the i^{th} predecessor of vertex v. There exists a set $d' \subseteq d$ which dominates vertex v.

Proof: To prove the above result first we will show that d satisfies the first condition for being a dominator of vertex v. Next we show that either d also satisfies the second condition or a subset of d (say d') satisfies the two conditions specified in the definition of dominator.

In order to reach vertex v from the start of the program, we have to visit one of its predecessors. Furthermore, to arrive at a predecessor of vertex v, say p_i, we have to visit one of the vertices in d_i. By including a dominator (d_i) of each predecessor (p_i) in the set d, we have ensured that we will visit one of the vertices in d before arriving at v. This satisfies the first condition for d to be a dominator of v. At this point if d satisfies the second condition then it is a dominator of vertex v. However, if d does not satisfy the second condition then there is a subset of d (say d') which satisfies the second condition or else d dominates v by lemma 1. This set d' can be constructed from d using lemma 2. The vertices to be removed from d to obtain d' are the ones which have a dominator in d. Therefore each vertex w in d, such that a dominator of w belongs to d, is removed from d. The resulting set d' is a subset of set d which satisfies the two conditions for it to be a dominator of v. $\Box$

Based upon the above result we develop the following approach for computing the generalized dominator set of a vertex. First, the single-vertex dominators are computed using an existing algorithm [6, 7, 10]. Next, the multiple-vertex dominators are determined by considering combinations of dominators of its predecessors. To verify whether a set of vertices V, with cardinality $\mid V \mid$, dominates a vertex v, we must ensure that no subset of vertices in V dominates v. These checks require that the dominators of cardinality less than $\mid V \mid$ are known. Thus, the computation of dominators consisting of n vertices is carried out after the computation of dominators consisting of fewer than n vertices. An algorithm based upon this approach is presented in Figure 2.

3 Generalized Immediate Dominators

The computation of all dominators can result in large dominator sets. In practice only a small subset of dominators are needed for performing code optimizations. In particular, the dominator relationships among the vertices that belong to a loop are required for performing loop optimizations. Next we present an approach for avoiding the computation of unnecessary dominator information using the notion of *generalized immediate dominators*. A vertex w is an *immediate single-vertex* dominator (isdom) of

```
Algorithm Generalized Dominators ( CFG = (V,E) ) {
    $v_0$ - the start vertex of the control flow graph
    $Cardinality = 1$;
    Compute all single-vertex dominators using an existing algorithm.
    while changes to any $D(v)$ occur {
        $Cardinality$++;
        for $v \in V - \{v_0\}$ {
            for each set $d = \cup_{i=1}^{|Pred(v)|} d_i$, where $d_i \in D(p_i) - D(v)$ and
                $p_i$ is the $i^{th}$ predecessor of $v$ and $| d |= Cardinality$ {
                for each vertex $w \in d$ {
                    if $\exists$ $V \in D(w)$ and $V \subseteq d - \{w\}$ { $d = d$ - $\{w\}$ } }
                $D(v) = D(v) \cup \{d\}$; }}
    }
}
```

Figure 2: Computing Generalized Dominator Sets.

vertex v if w is dominated by all other dominators of vertex v. This information is represented in the form of a dominator tree. An *immediate multiple-vertex* dominator (imdom) of a vertex v is defined to be a subset of v's immediate predecessors which dominates v. The addition of imdoms to the dominator tree results in the *dominator directed acyclic graph* or the DDAG. The entire generalized dominator set of any vertex can be computed by traversing the DDAG. Furthermore, it is also possible to compute dominators composed only from vertices belonging to a specific loop by restricting the traversal of the DDAG to the part that represents the loop. It should be noted that the isdom of a vertex may not be a predecessor of the vertex, while the imdom of a vertex only includes vertices from the predecessor set of the vertex. Therefore if a predecessor of a vertex dominates the vertex, then the vertex does not have an imdom. However, if this is not the case the following result guarantees that the vertex has an imdom.

Corollary 1: Let $Pred(v)$ denote the predecessor set of vertex v. There exists a set $idom(v) \subseteq Pred(v)$, such that $idom(v)$ dominates v.

Proof: Each vertex dominates itself and therefore the set $Pred(v)$ is a combination of dominators of predecessors of vertex v. Thus, from Theorem 1 it follows that there is a subset of $Pred(v)$ which dominates vertex v. □

The DDAG for an example CFG is shown in Figure 3. The predecessor set of vertex 7 is {6,4} and since 6 does not dominate 4 and 4 does not dominate 6, the set {6,4} is an imdom of 7. The predecessor set of vertex 6 is {5,3} and since 3 dominates 5 the vertex 6 does not have an imdom.

An algorithm for computing all immediate dominators is shown in Figure 4. First all isdoms are computed and the dominator tree is constructed. Next imdoms of increasing cardinalities are computed and added to the DDAG. The imdom of a vertex v, $imdom(v)$, is initialized to v's predecessor set $Pred(v)$. Next we continue to remove vertices from $imdom(v)$ until it satisfies lemma 2. To ensure that a set of size $Cardinality$ satisfies lemma 2, we require multiple-vertex dominators of size less

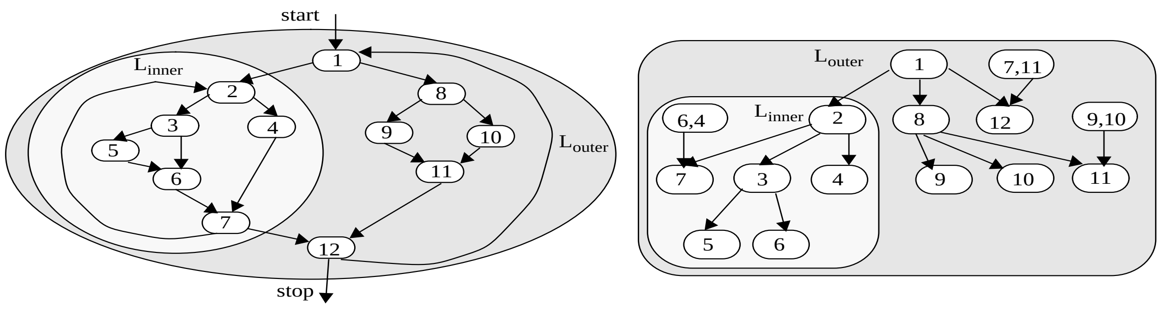


Figure 3: A Control Flow Graph and its DDAG.

than *Cardinality* for each vertex in the set. The required information is computed using the function *Expand*. The function *Expand*(c, v) computes all of the multiple-vertex dominators of vertex v with cardinality c. If a set of vertices V dominates vertex v, then additional dominators of v, with cardinality c, are generated by replacing elements in V by appropriate dominators of those elements. In particular a vertex $w \in V$ is replaced by a dominator of w, say d, if d does not dominate any vertex other than w in V. This condition is essential to ensure that the set of vertices obtained after replacing w in V by d satisfies lemma 2 and hence represents a multiple-vertex dominator of v. The imdoms of a given cardinality are computed after dominators of lower cardinalities have been computed.

To compute the imdoms the above algorithm may potentially compute multiple-vertex dominators other than imdoms. However, these dominators are a small fraction of all multiple-vertex dominators. This is because firstly this information is only computed for a subset of vertices. Secondly the non-imdoms computed have a cardinality less than *maxp*, which is the maximum number of predecessors of any vertex, while the maximum cardinality of any multiple-vertex dominator can be significantly higher than *maxp*. For a majority of the vertices in a program *maxp* is a small constant (1 or 2) and therefore the determination of the imdoms of these vertices will not require the computation of any non-imdoms. For example, in the CFG shown in Figure 3 no vertex has more than two predecessors, and therefore all imdoms are computed from single-vertex dominator information.

From the DDAG the generalized dominator set of any vertex v can be computed. This computation is merely an extension of function *Expand* described in Figure 4. The ancestors of the vertex in the DDAG clearly dominate the vertex. Additional dominators can be computed by recursively carrying out the replacement process used during function *Expand* of Figure 4. If a dominator that has been previously encountered is generated again, we stop performing replacements on this dominator since all dominators resulting from this dominator should have been previously generated. The detailed algorithm is presented in Figure 5.

The computation of the generalized dominator set for vertex 12 of the CFG in Figure 3 is shown in Figure 6(i). It should be noted that vertex 12 has multiple-vertex dominators of cardinality four although the value of *maxp* is only two. This indicates that the approach based upon immediate dominators is indeed far more efficient than the approach based upon computing all dominators.

As mentioned earlier, for loop optimizations, only the dominators of a loop exit composed of vertices

```
Algorithm BuildDominatorDAG {
    $v_0$ - the start vertex of the control flow graph;
    $Pred(v)$ - predecessor set of vertex $v$;
    $D(v)$ - the dominator set of vertex $v$;

    – Computation of all single-vertex dominators
    $D(v_0) := \{ \{v_0\} \}$
    for each $v \in V - \{v_0\}$ $\{ D(v) := \{ \{v_0\}, \{v_1\}, \{v_2\}, ..... \{v_{|V|-1}\} \} \}$
    while there is a change in any dominator set {
        for each $v \in V$ $\{ D(v) := \{ \{v\} \} \cup \cap_{p \in Pred(v)} D(p) \} \}$
    Construct dominator tree from single-vertex dominator information.

    – Computation of immediate multiple-vertex dominators
    for each vertex $v$ $\{ idom(v) = Pred(v) \}$
    if $|\ idom(v)\ |= 1$ $\{ done(v) = true \}$ else $\{ done(v) = false \}$
    $Cardinality = 1$
    while $\exists$ $v$ such that $done(v) = false$ {
        $Cardinality$++;
        for each vertex $v$ such that $done(v) = false$ {
            for each vertex $w \in idom(v)$ {
                if $\exists$ $d \subseteq idom(v) - \{w\}$, where $|\ d\ |= Cardinality - 1$ and $d \in D(w)$ {
                    $idom(v)$=$idom(v)$-$\{w\}$ }}
            if $|\ idom(v)\ |\leq Cardinality$ { – $idom(v)$ satisfies lemma 2
                Add node $idom(v)$ to the DDAG; $done(v) = true$ }}
        for each vertex $w \in idom(v)$ st $done(v) = false$ {
            $Expand(Cardinality, w)$ }}
}

Expand(c,v) {
    – compute all multiple-vertex dominators of $v$ with cardinality $c$
    for each $V \in D(v)$ {
        for each $w \in V$ {
            for each $d$ st $d \in D(w)$ and $\nexists$ $x \in V - \{w\}$ st $d \in D(x)$ {
                $V' = V \cup d$ - $\{w\}$
                if $|\ V'\ |$=$c$ $\{ D(v) = D(v) \cup \{ V' \} \} \}\}\}$
}
```

Figure 4: An Algorithm for Computing Generalized Immediate Dominators.

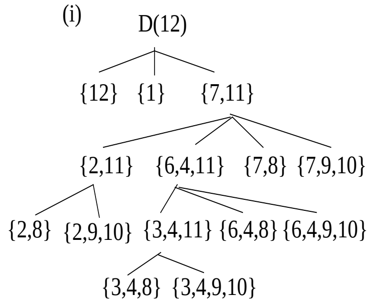
```
Algorithm ComputeAllDominators ($v$,DDAG) {
    $D(v) = \{ \{v\} \} \cup \{d$: $d$ is an ancestor of $v$ in the DDAG$\}$
    for each $V \in D(v)$ { Expand(V) } }

Expand(V) {
    for each $w \in V$ {
        for each ancestor of $w$, say $d$ {
            if $d$ is not an ancestor of a vertex in $V$-$\{w\}$ {
                $V' = V' \cup d - \{w\}$
                if $V'$ does not belong to $D(v)$ {
                    $D(v) = D(v) \cup \{ V' \}$
                    Expand($V'$) }}}}
}
```

Figure 5: Computing the Generalized Dominator Set of a Vertex.

belonging to the loop are required. This subset of information can be computed by restricting the traversal of the DDAG to the subgraph rooted at the vertex in the DAG representing the head of the loop. For example, when considering the outer loop (L_{outer}) in Figure 3, only the dominators of vertex 12 composed of vertices belonging only to L_{outer} need to be computed. During the computation of dominators for vertex 12, the inner loop can be considered as single vertex denoted as L_{inner}. As shown in Figure 6(ii) the above process reduces the size of the dominator set for vertex 12 from fourteen dominators to five dominators.

(i)



(ii)

$D(12) = \{\{1\},\{12\},\{2,8\},\{2,11\},\{7,8\},\{7,11\},\{2,9,10\},$
$\{3,4,8\},\{3,4,11\},\{6,4,8\},\{6,4,11\},\{7,9,10\},$
$\{3,4,9,10\},\{6,4,9,10\}\}$

$D'(12) = \{\{1\},\{12\},\{L_{inner},8\},\{L_{inner},11\},\{L_{inner},9,10\}\}$

Figure 6: Computing Dominators from the DDAG.

Next we analyze the run-time complexity of computing immediate multiple-vertex dominators.

Theorem 2: The time spent on computing imdoms is $O(\mid V \mid^n)$, where n is the maximum cardinality of any imdom (for example n is two for the CFG in Figure 3). The value of n is bounded by $maxp$, the maximum number of predecessors of any vertex.

Proof: The computation of imdoms involves two types of set operations. The first kind of operations are performed when we apply checks which eliminate a predecessor from the imdom set of a vertex if that predecessor is dominated by a subset of predecessors, not including the predecessor under consideration. We refer to these checks as dominator checks in this discussion. The second kind of operations are performed to compute the dominators so that the above checks can be performed. The imdoms are computed in a series of steps. In each step we identify the imdoms of cardinality higher than the cardinality of imdoms identified in the previous step. Let m be the cardinality of the imdom of vertex v. The identification of v's imdom will take $m-1$ steps. In each step dominator checks are performed. During the i^{th} step at most $i \times {}^{m}C_{m-i}$ checks are performed, where ${}^{m}C_{m-i} = m!/(m-i)!i!$. Thus, the total number of dominator checks performed during the computation of the imdom for vertex v is $\sum_{i=1}^{m-1} i \times {}^{m}C_{m-i}$ or $m(2^{m-1}-1)$. The total time spent on performing dominator checks during the computation of all imdoms is $O(n(2^{n-1}-1) \mid V \mid)$, where n is the maximum cardinality imdom. Dominators must be computed so that dominator checks can be performed. The computation of imdoms of cardinality i requires that the multiple-vertex dominators of cardinality less than i be known. Since the maximum cardinality of any imdom is n, we may have to compute all dominators of cardinality less than n. The computation of multiple-vertex dominators takes time linear in the number of such dominators. Thus, the total time spent on computing all required dominators for a single vertex is $\sum_{i=1}^{n-1} {}^{|V|}C_i$ or $O(\mid V \mid^{n-1})$. Therefore the time spent on the computation of multiple-vertex dominators for all vertices is $O(\mid V \mid^{n})$. Thus, the total time spent on computing immediate multiple-vertex dominators for all vertices is $O(n(2^{n-1}-1) \mid V \mid + \mid V \mid^{n})$. Since n is bounded by $maxp$, which is typically a small constant compared to total the number of vertices, we conclude that the worst case time complexity of our algorithm is $O(\mid V \mid^{n})$. $\square$

4 Concluding Remarks

In this paper the notion of multiple-vertex dominators was introduced and algorithms for computing multiple-vertex dominators were presented. A dual notion of generalized multiple-vertex postdominators can also be defined. The discussion on generalized postdominators can be found in [2]. Generalized dominator information can be used during loop invariant code motion and code hoisting optimizations. Dominator and postdominator information has also been used to address the problem of program coverage. In [1] Agrawal identifies groups of statements such that a test set that exercises one statement also exercises the other statements in the subset. Generalized dominator and postdominator information can be used to discover opportunities for sharing test cases among statements that fall in different subsets according to Agrawal's analysis [2]. In [8] we have developed static analysis techniques that analyze the synchronization structure of a program to uncover opportunities for distributed breakpointing through which event state can be captured following the detection of an event occurrence. Generalized postdominator information is used in identifying distributed breakpoints. Recently Sreedhar and Gao [9] have developed a new representation for flowgraph analysis called DJ-graphs. Using this representation they have derived an algorithm for computing generalized dominators with the time complexity $O(\mid E \mid \times \mid V \mid \times n)$, where $\mid E \mid$ is the number of edges in the flow graph and n is bounded by $maxp$.

References

[1] H. Agrawal, "Dominators, Super Blocks, and Program Coverage," *Proc. 21st Annual ACM Symp. on Principles of Programming Languages*, pages 25-34, Jan. 1994.

[2] R. Gupta, "Generalized Dominators and Post-Dominators," *Proc. 19th Annual ACM Symposium on Principles of Programming Languages*, Albuquerque, New Mexico, pages 246-257, Jan. 1992.

[3] R. Gupta, "Optimizing Array Bound Checks using Flow Analysis," *ACM Letters on Programming Languages and Systems*, Vol. 2, Nos. 1-4, pages 135-150, March-December 1994.

[4] D. Harel, "A Linear Time Algorithm for Finding Dominators in Flow Graphs and Related Problems," *Proc. of the 17th ACM Symposium on Theory of Computing*, pages 185-194, May 1985.

[5] M.S. Hetch and J.D. Ullman, "A Simple Algorithm for Global Data Flow Analysis of Programs," *SIAM Journal of Computing*, Vol. 4, pages 519-532, 1975.

[6] T. Lengauer and R.E. Tarjan, "A Fast Algorithm for Finding Dominators in a Flowgraph," *ACM Transactions on Programming Languages and Systems*, Vol. 1, pages 121-141, 1979.

[7] P.W. Purdom and E.F. Moore, "Immediate Predominators in a Directed Graph," *Communications of the ACM*, Vol. 15, No. 8, pages 777-778, 1972.

[8] M. Spezialetti and R. Gupta, "Debugging Distributed Programs through the Detection of Simultaneous Events," *IEEE-CS 14th International Conference on Distributed Computing Systems*, pages 634-641, Poznan, Poland, June 1994.

[9] V.C. Sreedhar, Y-F. Lee, and G.R. Gao, "DJ-Graphs and their Application to Flowgraph Analyses," McGill University, School of Computer Science, ACAPS Technical Memo 70, May 1994.

[10] R.E. Tarjan, "Finding Dominators in Directed Graphs," *SIAM Journal of Computing*, Vol. 3, No. 1, pages 62-89, 1974.