A Practical and Fast Iterative Algorithm for φ-Function Computation Using DJ Graphs

Dibyendu Das
U. Ramakrishna

ACM Transactions on Programming Languages and Systems
May 2005

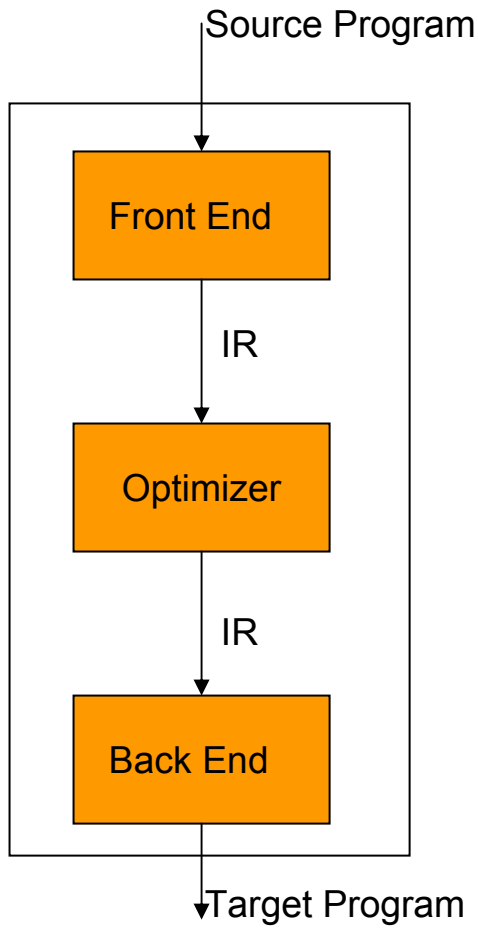
Humayun Zafar

Outline

- Introduction to a compiler.
- Static Single Assignment.
- Related Work.
- TDMSC-I
- TDMSC-II
- Implementation and Experiments.
- Conclusion

A compiler has…

- Three phases:
  - Front-End.
  - Optimizer.
  - Back-End.
- Intermediate Representation (IR) allows for more phases to be added to compilation.
- This paper concentrates on a form of IR.
  - Single Static Assignment.



Static Single-Assignment

- Adds information about both control flow and data flow to the program text.
- Encodes information about definitions and uses them in the name space of the code.
- Each distinct name is defined by a single operation in the code – hence the name.
- How to reconcile this single-assignment discipline with the effects of control flow?
  - Insert phi (φ) functions, at points where the control-flow paths meet.

A Small Loop in SSA Form

```
x ← …
y ← …
while (x < 100)
    x ← x + 1
    y ← y + 1
```

Original Code

```
x0 ← …
y0 ← …
    If (x0 >= 100) goto next
loop:   x1 ← φ(x0, x2)
        y1 ← φ(y0, y2)
        x2 ← x1 + 1
        y2 ← y1 + x2
        If (x2 < 100) goto loop
next:   x3 ← φ(x0, x2)
        y3 ← φ(y0, y2)
```

Its SSA Form

Loop Example…

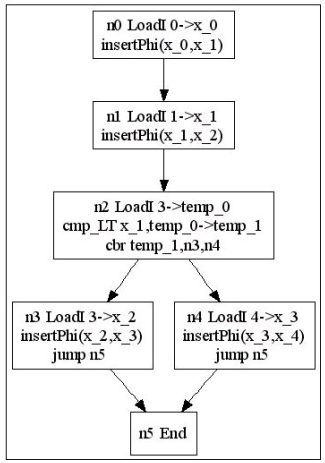
- Φ- functions are unusual.
- SSA form was designed to for use in code optimization.
- Placement of Φ-functions provides the compiler with information about the flow of values.
- Name space eliminates issues such as lifetime of a value.
  - Each value is defined in exactly one instruction.

Static Single-Assignment

- Does not have an obvious construction algorithm.
- Central to the idea of SSA is the placement of φ-functions.

- Das and Ramakrishna.
  - Propose an algorithm which computes the φ-function based on the merge set of each CFG of each function.

```
PROGRAM hello
VAR x AS INT ;
BEGIN
x := 1 ;
IF ( x LT 3 ) THEN
x := 3 ;
ELSE
x := 4 ;
END
END
```



An Example – CFG and DJ Graph

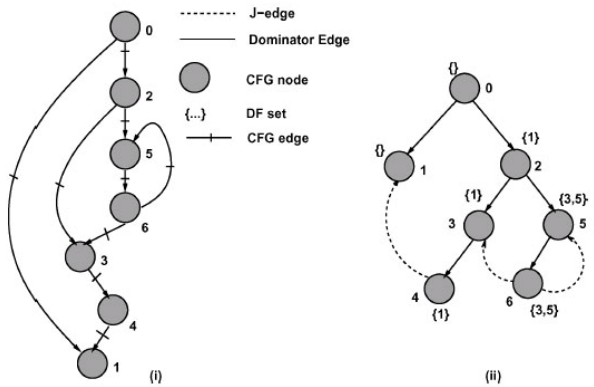


Fig. 1. An example of a CFG, its DJ graph and its DF sets.

- *J-edges.* If *e* = (*s*, *t*) but *s* does *not* strictly dominate *t*, then *e* is called a J-edge and *s* is the source and *t* the target nodes.
- *DJ Graph.* Sreedhar and Gao [1995] define a DJ graph to be a dominator tree with the J-edges added.
- *Dominance Frontier.* A node *v* is said to be in *DF*(*w*), if a predecessor of *v* is dominated by *w* but *v* is not strictly dominated by *w*.
- Three J edges in the DJ graph.
- DF sets in braces.

Existing φ-placement Algorithms

- Cytron et al. [1991]. Provide a 2-phase method in which they compute the DF for each node in the first phase, and the φ-placements in the other.
- Sreedhar and Gao [1995]. Introduce the idea of J-edges.
- Bilardi and Pingali [2003]. Compute the φ points using a mixed static and dynamic approach. Main challenge?
- Reif and Tarjan [1982]. Compute the inverse of the φ relation (φ^{-1}).
- Ramalingam [2002]. Uses loop-nesting forests for calculating the iterative dominance frontier.

Top Down Merge Set Computation-I

```
01: RequireAnotherPass = False;
02: while (in B(readth) F(irst) S(earch) order) do
03: Let n = Next Node in the BFS list
04:     for (all incoming edges to n) do
05:       Let e = Incoming edge
06:       if (e is a J-edge && e not visited) then
07:       Visit(e)
08:       Let snode = Source Node of e
09:       Let tnode = Target Node of e
10:       Let tmp = snode
11:       Let lnode = NULL
12:       while (level(tmp) ≥ level(tnode)) do
13:         Merge(tmp) = Merge(tmp) U Merge(tnode) U {tnode} 14: lnode = tmp
15:         tmp = parent(tmp) // dominator tree parent
16:       end while
17:       for (all incoming edges to lnode) do //lnode ancestor of snode
18:           Let e = Incoming edge
19:         if (e is a J-edge && e visited) then
20:           Let snode = Source Node of e
21:               if (Merge(snode)!(Subset) Merge(lnode))then // Check inconsistency
22:                   RequireAnotherPass = True
23:               end if
24:         end if
25:      end for
26:     end if
27:  end for
28: end while
29: return RequireAnotherPass
```

Complete Top Down Merge Set Computation

1: **do**

2: RequireAnotherPass = TDMSC-I(*GDJ*)

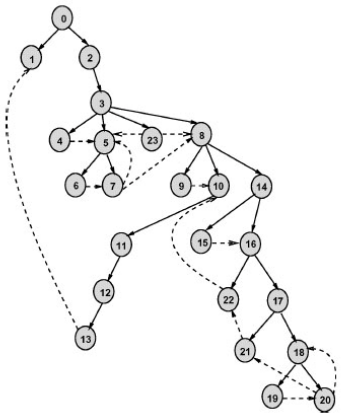
3: **while** (RequireAnotherPass)

Finding the φ Points Given an N_α

- N_α. It is the set of basic blocks in the CFG that have the initial definitions for a variable.
- Inputs: a. G_{DJ} with Merge sets computed b. N_α
- Output: Blocks augmented by φ-functions for the given variable

1: **for** (every node *n* in N_α) **do**
2: **for** (every node *n* in *Merge*(*n*)) **do**
3: Add a φ for *e* = (*n*, *n*) if not already placed
4: **end for**
5: **end for**

An Example From 254.gap



Merge(4)	={5} U Merge(5);	{5,8}
Merge(5)	={5,8} U Merge(5) U Merge(8);	{5,8}
Merge(6)	={7} U Merge(7);	{5,7,8}
Merge(7)	={5,8} U Merge(5) U Merge(8);	{5,8}
Merge(9)	={10} U Merge(10);	{10}
Merge(14)	={10} U Merge(10);	{10}
Merge(15)	={16} U Merge(16);	{10,16}
Merge(16)	={10} U Merge(10);	{10}
Merge(17)	={22} U Merge(22);	{10,22}
Merge(18)	={18,21} U Merge(18) U Merge(21);	{10,18,21,22}
Merge(19)	={20} U Merge(20);	{10,18,20,21,22}
Merge(20)	={18,21} U Merge(18) U Merge(21);	{10,18,21,22}
Merge(21)	={22} U Merge(22);	{10,22}
Merge(22)	={10} U Merge(10);	{10}
Merge(23)	={5,8} U Merge(5) U Merge(8);	{5,8}

Fig. 3. A DJ Graph from 254.gap and its merge sets.

Table I. Number of Passes for TDMSC-I

Benchmarks	No. of Runs	1-Pass Completion	2-Pass Completion	3-Pass Completion
164.gzip	826	696	130	0
175.vpr	1907	1610	293	4
176.gcc	21147	15225	5735	187
181.mcf	208	174	34	0
186.crafty	1162	793	363	6
197.parser	2800	2257	537	6
254.gap	7656	5464	2128	64
255.vortex	8542	6371	2168	3
256.bzip2	525	434	91	0
300.twolf	1859	1397	444	18
253.perlbmk	10339	7754	2481	104

- First column indicates how many times the merge algorithm was invoked.
- The other columns show how many of those completed in 1, 2 and 3 passes respectively.

Complexity of TDMSC-I

- Consists of three parts:
  - BFS.
  - Merge set computation.
  - Consistency check.
- BFS
  - Can be done in $O(|V| + |E|)$
- Merge set computation
  - Can be done in $O(h^{avg} * |J|)$
- Consistency check
  - Can be done in $O(e^{avg} * |J|)$

Overall Complexity of TDMSC-I

- Single Pass
  - $[O(|V| + |E|) + O(h^{avg} * |J|) + O(e^{avg} * |J|)]$
- For P passes the complexity of CTDMSC is
  - $[O(|V| + |E|) + O(h^{avg} * |J|) + O(e^{avg} * |J|)] * P$

Improvements to the Iterative Merge Set Computation Algorithm

```
01: RequireAnotherPass = False;
02: while (in BFS order) do
03:   Let n = Next Node in the BFS list
04:   for (all incoming edges to n) do
05:     Let e = Incoming edge
06:     if (e is a J-edge && e not visited) then
07:       Visit(e)
08:       Let snode = Source Node of e
09:       Let tnode = Target Node of e
10:       Let tmp = snode
11:       Let lnode = NULL
12:       while (level(tmp) ≥ level(tnode)) do
13:           Merge(tmp) = Merge(tmp) U    Merge(tnode) U {tnode}
14:           lnode = tmp
15:          tmp = parent(tmp) // dominator tree   parent
16:      end while
17:     for (all incoming edges to lnode) do
18:         Let e = Incoming edge
19:        if (e is a J-edge && e visited) then
20:           Let snode = Source Node of e
21:           if (Merge(snode) !(Subset) Merge(lnode)) then
22:              Let node = snode
23:              Let lnode = NULL
24:              while (level(node) ≥ level (lnode)) do
25:              // Local Correction for Shadow((snode, lnode))
26:                 Merge(node) = Merge(node) U Merge(lnode)
27:                 lnode = node
28:                 node = parent(node)
29:             end while
30:            if (An Incoming J Edge Inconsistent(lnode)) then
31:               RequireAnotherPass = True
32:            end if
33:          end if
34:      end if
35:   end for
36:  end if
37: end for
38: end while
39: return RequireAnotherPass
```

```
01: RequireAnotherPass = False;
02: while (in B(readth) F(irst) S(earch) order) do
03: Let n = Next Node in the BFS list
04:      for (all incoming edges to n) do
05:        Let e = Incoming edge
06:        if (e is a J-edge && e not visited) then
07:       Visit(e)
08:       Let snode = Source Node of e
09:       Let tnode = Target Node of e
10:       Let tmp = snode
11:       Let lnode = NULL
12:       while (level(tmp) ≥ level(tnode)) do
13:         Merge(tmp) = Merge(tmp) U Merge(tnode) U {tnode} 14: lnode =
      tmp
15:         tmp = parent(tmp) // dominator tree parent
16:       end while
17:       for (all incoming edges to lnode) do //lnode ancestor of snode
18:           Let e = Incoming edge
19:         if (e is a J-edge && e visited) then
20:            Let snode = Source Node of e
21:               if (Merge(snode)!(Subset) Merge(lnode))then // Check
      inconsistency
22:                   RequireAnotherPass = True
23:               end if
24:        end if
25:      end for
26:     end if
27:  end for
28: end while
29: return RequireAnotherPass
```

Improvement?

Table II. Number of Passes for TDMSC-II

Benchmarks	No. of Runs	1-Pass Completion	2-Pass Completion	3-Pass Completion
164.gzip	826	819	7	0
175.vpr	1907	1843	64	0
176.gcc	21147	19825	1309	3
181.mcf	208	206	2	0
186.crafty	1162	1071	91	0
197.parser	2800	2682	118	0
254.gap	7656	7427	229	0
255.vortex	8542	8496	46	0
256.bzip2	525	517	8	0
300.twolf	1859	1713	146	0
253.perlbmk	10339	9610	729	0

- **TDMSC-II** cuts down on the second pass for Spec2000 benchmarks to around 5%.
- The number of cases where a third pass is required is just 3.
- Complexity
  - An extra $O(e^{avg} * h^{avg} * |J|)$ added.

Implementation and Experiments

- The merge sets implemented as bit-vectors.
- These bit-vectors are converted to linked lists (on first use) during the φ-function placement phase.
- Used C Specint2000 and the Specfp2000 SPEC benchmarks.
- The benchmarks have been run at the highest optimization level of +O4
- Noted around 17% improvement (using **TDMSC-II**) for the Specint2000 benchmarks and around 37% improvement for the Specfp2000 benchmarks.
- Total time is the time taken to compute the merge sets and the time to compute the φ points for all given N_{α}.

Compile Time Comparisons

Table IV. Compile-Time Comparisons for Specint(fp)2000

Benchmarks	TDMSC-II (Compile Time in Secs)	CFR (Compile Time in Secs)	% Improvement
164.gzip	0.98	1.25	21.6
175.vpr	2.74	3.30	16.9
176.gcc	52.53	52.90	0.7
181.mcf	0.07	0.08	12.5
186.crafty	15.18	20.33	25.33
197.parser	2.85	6.29	64.23
254.gap	6.63	9.52	30.36
255.vortex	15.49	21.36	27.48
256.bzip2	0.08	0.09	11.11
300.twolf	4.88	6.56	25.61
253.perlbmk	21.02	25.52	17.63
Overall	122.45	147.20	17
177.mesa	3.42	4.3	20.47
179.art	0.16	0.27	40.74
183.equake	0.36	0.77	53.25
188.ammp	2.04	4.23	51.77
Overall	5.98	9.57	37

Conclusion

- Algorithm computes the merge sets of all the basic blocks statically and uses these sets to find the φ points.
- No additional data structure other than the DJ graph has been used for this purpose.