A Practical and Fast Iterative Algorithm for \( \phi \)-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

Original Code

\[
\begin{align*}
x & \leftarrow \ldots \\
y & \leftarrow \ldots \\
\text{while } (x < 100) & \\
& \quad x \leftarrow x + 1 \\
& \quad y \leftarrow y + 1
\end{align*}
\]

Its SSA Form

\[
\begin{align*}
x_0 & \leftarrow \ldots \\
y_0 & \leftarrow \ldots \\
\text{If } (x_0 \geq 100) & \text{ goto next} \\
\text{loop: } & \\
& \quad x_1 \leftarrow \varphi(x_0, x_2) \\
& \quad y_1 \leftarrow \varphi(y_0, y_2) \\
& \quad x_2 \leftarrow x_1 + 1 \\
& \quad y_2 \leftarrow y_1 + x_2 \\
& \text{If } (x_2 < 100) \text{ goto loop} \\
\text{next: } & \\
& \quad x_3 \leftarrow \varphi(x_0, x_2) \\
& \quad y_3 \leftarrow \varphi(y_0, y_2)
\end{align*}
\]
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-Assigment

- Does not have an obvious construction algorithm.
- Central to the idea of SSA is the placement of $\phi$-functions.
- Das and Ramakrishna.
  - Propose an algorithm which computes the $\phi$-function based on the merge set of each CFG of each function.
**An Example – CFG and DJ Graph**

- **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.

Fig. 1. An example of a CFG, its DJ graph and its DF sets.
Existing $\varphi$-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 $\varphi$-placements in the other.
- Bilardi and Pingali [2003]. Compute the $\varphi$ points using a mixed static and dynamic approach. Main challenge?
- Reif and Tarjan [1982]. Compute the inverse of the $\varphi$ relation ($\varphi^{-1}$).
- 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 inorder = NULL
12: while (level(tmp) ≥ level(tnode)) do
13: Merge(tmp) = Merge(tmp) U Merge(inode) U {tnode}
14: inorder = tmp // dominator tree parent
15: end while
16: for (all incoming edges to inorder) do // inorder ancestor of snode
17: if (e is a J-edge && e visited) then
18: Let snode = Source Node of e
19: if (Merge(snode)!(Subset) Merge(inorder)) then // Check inconsistency
20: RequireAnotherPass = True
21: end if
22: end if
23: end for
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 $\varphi$ Points Given an $N_\alpha$

- $N_\alpha$. 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. $N_\alpha$
- Output: Blocks augmented by $\varphi$-functions for the given variable

1: for (every node $n$ in $N_\alpha$) do
2:  for (every node $n$ in $\text{Merge}(n)$) do
3:    Add a $\varphi$ for $e = (n, n)$ if not already placed
4:  end for
5: end for
An Example From 254.gap

- 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.

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

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>No. of Runs</th>
<th>1-Pass Completion</th>
<th>2-Pass Completion</th>
<th>3-Pass Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>826</td>
<td>696</td>
<td>130</td>
<td>0</td>
</tr>
<tr>
<td>vpr</td>
<td>1907</td>
<td>1610</td>
<td>293</td>
<td>4</td>
</tr>
<tr>
<td>gcc</td>
<td>21147</td>
<td>15235</td>
<td>5725</td>
<td>187</td>
</tr>
<tr>
<td>macro</td>
<td>208</td>
<td>174</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>crafty</td>
<td>1162</td>
<td>793</td>
<td>363</td>
<td>6</td>
</tr>
<tr>
<td>parser</td>
<td>2200</td>
<td>2257</td>
<td>537</td>
<td>6</td>
</tr>
<tr>
<td>254.gap</td>
<td>7656</td>
<td>5446</td>
<td>2128</td>
<td>64</td>
</tr>
<tr>
<td>vortex</td>
<td>8542</td>
<td>6371</td>
<td>2168</td>
<td>3</td>
</tr>
<tr>
<td>hzips2</td>
<td>525</td>
<td>434</td>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td>twolf</td>
<td>1859</td>
<td>1397</td>
<td>444</td>
<td>18</td>
</tr>
<tr>
<td>perlbench</td>
<td>10339</td>
<td>7754</td>
<td>2481</td>
<td>104</td>
</tr>
</tbody>
</table>
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 $O(h_{avg} \times |J|)$

- Consistency check
  - Can be done in $O(e_{avg} \times |J|)$
Overall Complexity of TDMSC-I

- Single Pass
  \[O(|V| + |E|) + O(h_{avg} \times |J|) + O(e_{avg} \times |J|)\]

- For \(P\) passes the complexity of CTDMSC is
  \[O(|V| + |E|) + O(h_{avg} \times |J|) + O(e_{avg} \times |J|) \times P\]
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:                Inode = 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 // Check inconsistency
22:                        RequireAnotherPass = True
23:                    end if
24:                end if
25:            end for
26:        end for
27:    end while
28: return RequireAnotherPass
Improvement?

- **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 $O(e^{avg} \times h^{avg} \times |J|)$ added.
Implementation and Experiments

- The merge sets implemented as bit-vectors.
- These bit-vectors are converted to linked lists (on first use) during the $\phi$-function placement phase.
- 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 $\phi$ points for all given $N_\alpha$. 
## Compile Time Comparisons

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

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

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