Storage Assignment to Decrease Code Size

< ACM SIGPLAN PLDI’95 >

Stan Liao  Srinivas Devadas  Kurt Keutzer Steve Tjiang Albert Wang
MIT Department of EECS  Synopsys, Inc.
I. Introductions..

Processor Models..

1. Memory access can occur only indirectly via a set of address registers, AR0 through AR(k-1).
2. If an instruction uses ARi for indirect addressing, then in the same instruction ARi can be optionally post-incremented or post-decremented by one at no extra cost.
3. If an address register does not point to the desired location, it may be changed by adding or subtracting a constant, using the instructions ADAR and SBAR.
4. To initialize an address register, the LDAR instruction is used.

Notations..

*(ARi) : indirect addressing through Ari
*(ARi)+ : indirect addressing with post-increment
*(ARi)- : indirect addressing with post-decrement
Figure 1: (a) Code sequence (b) Offset assignment (c) Assembly code

Figure 2: (a) Code sequence (b) Different Offset assignment (c) Assembly code
II. SOA (Simple Offset Assignment) problem

- Assumptions..
  - A single address register
  - One-to-one mapping of variables to locations
  - The basic block has a fixed evaluation order (schedule)

- Given a code sequence C that represents a BB,
  => Unique definition of an access sequence for the block.
  => Construct Access Graph.

- Access Graph $G(V,E,W)$:
  \[ V : \forall v \in V \text{ in the graph corresponds to a unique variable.} \]
  \[ E : \text{An edge } e = \langle v_i, v_j \rangle \in E \text{ between nodes } V_i \text{ and } V_j \text{ are adjacent to each other } w(e) \text{ times in the access sequence} \]

  => In terms of access graph, the cost of an assignment is equal to the sum of the weights of all edges that do not connect variables assigned to adjacent locations.
Figure 3: (a) Access sequence  (b) Access graph

Figure 4: (a) A disjoint path cover  (b) an implied assignment with a cost of four
SOLVE-SOA(L)
{
    /* L = access sequence for basic block */
    G(V, E) ← ACCESS-GRAPH(L);
    E ← sorted list of edges in E
        in descending order of weight:
    G'(V', E') : V' ← V, E' ← φ;
    while ( |E'| < |V| - 1 and E ≠ φ ) {
        choose e ← first edge in E;
        E ← E - e;
        if ( (e does not cause a cycle in G') and
            (e does not cause any node in V' to have degree > 2) )
            add e to E';
        else
discard e;
    } /* Construct an assignment from E' */
return CONSTRUCT-ASSIGNMENT(E');
}

SOLVE-GOA(L, k)
{
    /* L = access sequence of basic block */
    /* k = number of address registers */
    H ← SOLVE-SOA(L);
    if (k == 1)
        return {H};
    P ← SELECT-VARIABLES(L);
    L₁ ← SUBSEQ(L, P);
    L₂ ← SUBSEQ(L, L - P);
    H₁ ← SOLVE-SOA(L₁);
    H₂ ← SOLVE-SOA(L₂);
    if (setup-cost + cost(H₁) + cost(H₂) > cost(H))
        return {H};
    else
        return {H₁} ∪ SOLVE-GOA(L₂, k - 1);
}

Figure 6: Heuristic Algorithm for SOA
Figure 8: Heuristic Algorithm for GOA
III. GOA(General Offset Assignment) Problem

- Additional Assumptions..
  - There is a fixed cost of introducing the use of an address register.
    : Set-up cost
  - Each address register is used to point to a disjoint subset of variables.

- Subsequence S’ of sequence S :
  The access subsequence generated by subset of variables $W \subseteq V$.

$\Rightarrow$ Given an access sequence L, the set of variables V, and the number of address registers k, find a partition of V, $\Pi = \{P_1, P_2, ..., P_m\}$, where $m \leq k$, such that the total cost of the optimal SOA of the corresponding access subsequences plus the setup costs for using m registers is minimum.
Figure 7: (a) Access sequence and graph (b) Access subsequence and graph generated by \{a,d,e\} (c) Access subsequence and graph generated by \{b,c\}
### IV. Experiments and Results

<table>
<thead>
<tr>
<th>program</th>
<th>decl. order</th>
<th>greedy SOA</th>
<th>b-b SOA</th>
<th>greedy GOA</th>
<th>b-b GOA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># inst.</td>
<td># inst.</td>
<td>% red.</td>
<td># inst.</td>
<td>% red.</td>
</tr>
<tr>
<td>chendct</td>
<td>756</td>
<td>730</td>
<td>3.5%</td>
<td>728</td>
<td>3.7%</td>
</tr>
<tr>
<td>chenidct</td>
<td>817</td>
<td>778</td>
<td>4.8%</td>
<td>776</td>
<td>5.0%</td>
</tr>
<tr>
<td>lleedct</td>
<td>893</td>
<td>842</td>
<td>5.7%</td>
<td>841</td>
<td>5.8%</td>
</tr>
<tr>
<td>lleedct</td>
<td>1017</td>
<td>949</td>
<td>6.7%</td>
<td>948</td>
<td>6.8%</td>
</tr>
<tr>
<td>jrev</td>
<td>4296</td>
<td>4070</td>
<td>5.3%</td>
<td>4057</td>
<td>5.6%</td>
</tr>
<tr>
<td>reacgif</td>
<td>648</td>
<td>599</td>
<td>7.6%</td>
<td>589</td>
<td>9.1%</td>
</tr>
<tr>
<td>autocrop</td>
<td>549</td>
<td>534</td>
<td>2.7%</td>
<td>531</td>
<td>3.3%</td>
</tr>
<tr>
<td>smooth</td>
<td>4002</td>
<td>3841</td>
<td>4.0%</td>
<td>3837</td>
<td>4.1%</td>
</tr>
<tr>
<td>huffree</td>
<td>956</td>
<td>914</td>
<td>4.4%</td>
<td>907</td>
<td>5.1%</td>
</tr>
<tr>
<td>grucrypt</td>
<td>3188</td>
<td>3063</td>
<td>3.9%</td>
<td>3065</td>
<td>3.9%</td>
</tr>
</tbody>
</table>

Table 1: Experimental results
Rainer Leupers, Peter Marwedel’s GOA algorithm & Results

![Algorithm SOLVEGOA(V, S, k)]

```
algorithm SOLVEGOA(V, S, k)
begin
   AG = (V, E, w) := access graph for S;
   L := sorted list of non-zero edges in E
      in descending order of weight;
   V₁, ..., Vₖ := Ø;
   i := 0;
   repeat
      i := i + 1;
      {u, v} := next edge in L with u, v \notin V₁ \cup ... \cup Vₖ;
      Vᵢ := {u, v};
   until (i = k) or ({u, v} = Ø);
   l := i;
   for all v \in V, v \notin V₁ \cup ... \cup Vᵢ do
      V* := the element of {V₁, ..., Vᵢ}, such that
            SOA_cost(S(V* \cup {v})) - SOA_cost(S(V*)) \rightarrow \text{min};
      V* := V* \cup {v};
   end for
   return SOLVESOA(S(V₁)) \circ ... \circ SOLVESOA(S(Vᵢ));
end algorithm
```

Fig. 4. Algorithm for General Offset Assignment

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>k</th>
<th>Liao</th>
<th>SOLVEGOA</th>
<th>gain (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td>2</td>
<td>13.14</td>
<td>11.93</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>4</td>
<td>9.30</td>
<td>5.02</td>
<td>46</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>4</td>
<td>10.40</td>
<td>7.65</td>
<td>26</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>8</td>
<td>8.14</td>
<td>8.22</td>
<td>-1</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>4</td>
<td>17.16</td>
<td>8.45</td>
<td>51</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>8</td>
<td>17.29</td>
<td>5.00</td>
<td>71</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>8</td>
<td>21.34</td>
<td>12.69</td>
<td>41</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>8</td>
<td>17.80</td>
<td>11.30</td>
<td>37</td>
</tr>
<tr>
<td>100</td>
<td>200</td>
<td>8</td>
<td>59.61</td>
<td>37.84</td>
<td>37</td>
</tr>
</tbody>
</table>

TABLE II
COMPARISON OF GOA ALGORITHMS