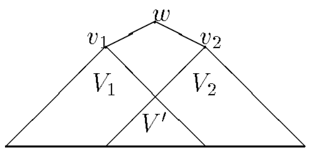
# Efficient Register Allocation for Large Basic Blocks


Christoph W. Keßler, Thomas Rauber
Computer Science Department
University of Saarbrücken, Germany
email: {kessler,rauber}@cs.uni-sb.de


We consider the NP–complete problem of generating evaluations for expression DAGs with a minimal number of registers. We restrict our attention to contiguous evaluations, because for nearly all of the DAGs derived from real application programs there exists a contiguous evaluation that is optimal w.r. to the register need. We present an algorithm that generates an optimal contiguous evaluation for a given DAG. The algorithm is very fast on the average. It generates the evaluation by splitting the DAG in trees with import and export nodes and applying a labeling scheme to the trees.

A **contiguous evaluation** of an expression DAG is the sequence of the DAG nodes in the order of a postorder traversal of the DAG such that at each binary node, one subDAG is traversed *completely* before traversing the rest of the other subDAG.




A **tree node** is either a leaf, or an inner node with all its sons being tree nodes and none of them having more than one father.
A **decision node** is a binary node which is *not* a tree node.
If we evaluate all the tree nodes in a DAG with $d$ decision nodes by the Labeling Algorithm of Sethi/Ullman, there remain at most $2^d$ different contiguous evaluations for the DAG:

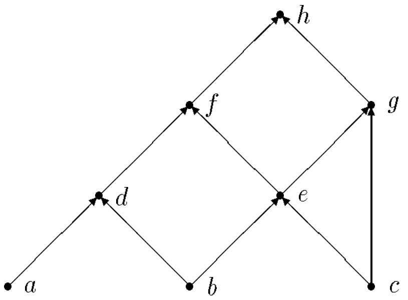
---

**algorithm** *simple*
Let $v_1, \ldots, v_d$ be the decision nodes of the DAG, and
let $\beta = (\beta_1, \ldots, \beta_d) \in \{0,1\}^d$ be a bitvector.
**forall** $2^d$ different $\beta \in \{0,1\}^d$ **do**
    start $dfs(root)$ with each $\beta$, such that for $1 \le i \le d$
      **if** $\beta_i = 0$ in the call $dfs(v_i)$,
      **then** the left son of $v_i$ is evaluated first
      **else** the right son of $v_i$ is evaluated first **fi od**

---

run time: $O(n2^d)$, with $d \le n-2$.

**First improvement**: traverse only for bitvectors that yield *different* evaluations:




For this example DAG the improved algorithm *descend* executes the evaluation steps given in the table to the right. Only 7 instead of $2^5 = 32$ contiguous evaluations are printed.

| decision nodes $v_1, v_2, \ldots, v_5$: | $h$ | $f$ | $g$ | $d$ | $e$ | |
|---|---|---|---|---|---|---|
| start at the root: preset first bit: | **0** | | ∗ | | | |
| propagate bits and asterisks to next stage: | 0 | **0** | ∗ | | ∗ | |
| all bits set: first evaluation found: | 0 | 0 | ∗ | **0** | ∗ | $A_1$ |
| | 0 | 0 | ∗ | **1** | ∗ | $A_2$ |
| 'backtrack': | 0 | **1** | ∗ | ∗ | | |
| | 0 | 1 | ∗ | ∗ | **0** | $A_3$ |
| | 0 | 1 | ∗ | ∗ | **1** | $A_4$ |
| 'backtrack': | **1** | ∗ | | ∗ | | |
| | 1 | ∗ | **0** | ∗ | | |
| | 1 | ∗ | 0 | ∗ | **0** | $A_5$ |
| | 1 | ∗ | 0 | ∗ | **1** | $A_6$ |
| 'backtrack': | 1 | ∗ | **1** | ∗ | ∗ | $A_7$ |

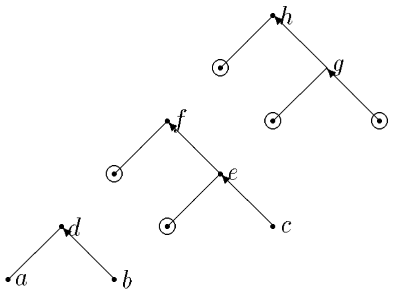
Here, $g$ is *excluded* from the decision after setting the decision bit of $h$ to 0, because the son $e$ (and $c$) is taken away from $g$ by $f$. The decision bit of $g$ is *redundant* (marked by $*$).

Let $v_1, \ldots, v_d$ be the decision nodes in reversed topological order.
Call $descend(\Theta, 1)$ where $\Theta$ is a bitvector that contains $d$ 0's.

```
function descend ( bitvector β, int pos )
while β_pos = * and pos < d do pos ← pos + 1 od
if pos ≥ d
then if β_pos = *
        then print β   // new evaluation found //
        else β_d = 0; print β;   // new evaluation found //
             β_d = 1; print β;   // new evaluation found // fi
else β_pos = 0;
        mark exclusions of nodes v_j, j ∈ {pos + 1, ..., d} through lson(v_pos) by β_j ← *;
        descend( β, pos + 1);
        β_pos = 1;
        mark exclusions of nodes v_j, j ∈ {pos + 1, ..., d} through rson(v_pos) by β_j ← *;
        descend( β, pos + 1);
end descend;
```

run time: $O(n2^{d-1})$ worst case, around $O(n2^{d/2})$ average

**Second improvement:** cut the DAG into trees and apply a modified labeling algorithm to the trees. Import and export nodes are used to account for the relationship between the different trees of the DAG.




The example DAG is split into 3 subtrees by setting $\beta_1 = 0$, $\beta_2 = 0$, $\beta_4 = 0$. The newly introduced import nodes are marked with a circle.
By evaluating the subtrees of the DAG with the modified labeling algorithm, the number of evaluations generated is reduced from 7 to 3, because there are 3 possibilities to split the DAG into trees.

**Experimental Results:**

| Source | DAG | $n$ | $d$ | $N_{simple}$ | $N_{descend}$ | $N_{descend2}$ | $T_{descend}$ | $T_{descend2}$ |
|---|---|---|---|---|---|---|---|---|
| LL 14 | second loop | 19 | 10 | 1024 | 432 | 18 | 0.1 sec. | < 0.1 sec. |
| LL 20 | inner loop | 23 | 14 | 16384 | 992 | 6 | 0.2 sec. | < 0.1 sec. |
| MDG | calc. $\cos(\theta), \sin(\theta), \ldots$ | 26 | 15 | 32768 | 192 | 96 | < 0.1 sec. | < 0.1 sec. |
| MDG | calc. forces, first part | 81 | 59 | $2^{59}$ | — | 7168 | — | 13.6 sec. |
| | subDAG of this | 65 | 45 | $2^{45}$ | — | 532 | — | 0.9 sec. |
| | subDAG of this | 52 | 35 | $2^{35}$ | 284672 | 272 | 70.2 sec. | 0.8 sec. |
| | subDAG of this | 44 | 30 | $2^{30}$ | 172032 | 72 | 42.9 sec. | 0.3 sec. |
| SPEC77 | mult. FFT analysis | 49 | 30 | $2^{30}$ | 131072 | 32768 | 20.05 sec. | 21.1 sec. |

Measurements for DAGs taken from real programs (LL = Livermore Loop Kernels; MDG = Molecular Dynamics, and SPEC77 = atmospheric flow simulation, both from the Perfect Club Benchmark Suite). Runtime for a SUN SPARC station SLC.

# References


[1] Keßler, C.W., Paul, W.J., Rauber, T.: *A Randomized Heuristic Approach to Register Allocation.* Proceedings of PLILP'91 Third International Symposium on Programming Language Implementation and Logic Programming, Aug. 26–28, 1991, Passau, Germany. Springer LNCS Vol. 528, 195–206.

[2] Keßler, C.W., Paul, W.J., Rauber, T.: *Scheduling Vector Straight Line Code on Vector Processors.* in: R. Giegerich, S.L. Graham (Ed.): Code Generation — Concepts, Tools, Techniques. Springer Workshops in Computing Series (WICS), 1992.