Fine-Grain Register Allocation Based on a Global Spill Costs Analysis

2003. 9. 25 (Thr.)

Seoul National University

Dae-Hwan Kim dhkim@capp.snu.ac.kr
Hyuk-Jae Lee hjlee@ee.snu.ac.kr

Outline

- Graph coloring register allocation
- Motivation example
- Proposed register allocation algorithm
- Allocation benefit model
- Experimental results
- Conclusions

Overview of Register Allocation

- Determines whether a live range (variable/temporary) is to be stored in a register or in memory
- Goal:
  - Store live ranges as many as possible in registers
  - Minimize the number of memory accesses (load/store instructions)
- An important compiler technique
  - The reduction of load/store instructions leads to the decrease of execution time, code size and power consumption.

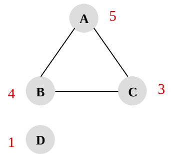
Graph-Coloring Register Allocation

- Dominant allocation paradigm (Chaitin, Brigss, …)
- Models register allocation problem as a graph coloring problem of an interference graph
- Interference graph: an undirected graph, where
  - A node: a live range
  - there is an edge between two nodes if corresponding live ranges interfere.
  - Interfering live ranges can not share the same register
- The contribution of graph coloring approach: the simplicity by abstracting each live range as a single node of an interference graph

Graph-Coloring Limitations

- What is not represented in the conventional graph coloring approach ?
  - does not specify where and how much live ranges interfere

```
A = foo1( );
B = A + 1;
foo2(A + B);
C = foo3();
foo4(C + 1);
foo5(C + 2);
foo6(B + 3);
foo7(A + 4);
D = A - B;
```



Motivation Example

- Suppose the number of registers is two.
- Graph coloring spills ‘C’ with 3 memory accesses
- Different spill cost at each reference in the flow analysis
- Spill A after (3) and before (8): 2 memory accesses

```
A = foo1( );     (1)
B = A + 1;       (2)
foo2(A + B);     (3)   store A
C = foo3();      (4)
foo4(C + 1);     (5)
foo5(C + 2);     (6)
foo6(B + 3);     (7)   load A
foo7(A + 4);     (8)
D = A - B;       (9)
```

Overview of Proposed Algorithm

- Decides whether to allocate a register or not for every reference of a variable
- When there is no free register, it determines allocation based on the allocation benefit in the reference flow
- Two stages
  - Variable allocation: variables are allocated with the number of machine registers
  - Scratch allocation: temporaries and unallocated variables are allocated

Variable Reference Flow Graph

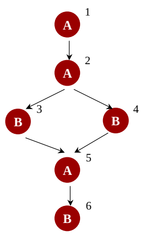
- The proposed approach constructs a varef-graph (variable reference flow graph)
- Node: a variable reference
- Edge: control flow of the program (i.e, execution order of the variable references of the program)
- The varef-graph models the execution order of the references in the program.

Varef-Gaph Example

```
(1)    A = 1;

(2)    if (A)
(3)        B = 1;
       else
(4)        B = 2;
(5)    return A + B;
```

code example

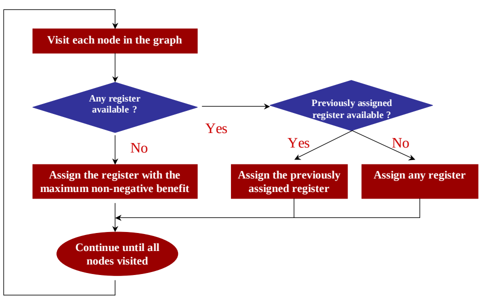


Variable reference flow graph

Allocation Algorithm

- The proposed algorithm visits each node of a varef-graph in the breadth-first order.
- When no register is free for a node, the allocator estimates the benefit and loss of register preemption for each register, and selects the register with the maximum benefit.
- If all registers have larger loss than benefit, no register is assigned to the node.
- For those nodes that are not assigned to a register, the second stage register allocation, called scratch allocation, is performed.

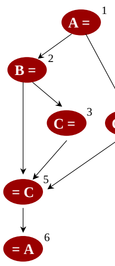
Variable Allocation Algorithm



Register Allocation Benefit

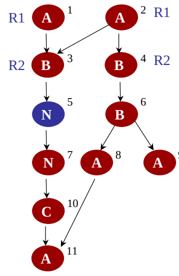
- BenefitRegAlloc(n,r) = PenaltySpill(n,r) – PenaltyPreempt(n,r)
  - n: node number, r: register number
- PenaltySpill(n,r): the total number of load/store instructions that are required if node ‘n’ is spilled.
- PenaltyPreempt(n,r): the total number of load/store instructions that are required when node ‘n’ preempts register ‘r’.

Impact Range



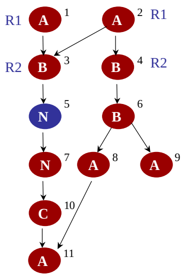
- The decision for one node affects the register allocation for another node
  - If node ‘1’ is spilled, node ‘6’ is also spilled
- Impact Range: the set of the nodes that are affected by the register allocation for a given node.
- The impact range of node ‘n’ for register ‘r’ is defined as the path from node ‘n’ to the next references of a variable that currently holds register ‘r’.

Impact Range Example (1)



- Suppose register allocation visits node ‘5’
- Suppose the number of registers is 2
- VarHold(n,r): the variable that holds register ‘r’ when the register allocation is performed for node ‘n’.
- NodeHold(n,r): the nodes for VarHold(n,r)
- VarHold(5,r1)=‘A’, NodeHold(5, r1) = {1,2}.
- NextRef(n) = {p| p is a next reference of n} .
- NextRef(1) = {11} , NextRef(2) = {8,9,11},
- NextRef({1,2}) = {8, 9,11}
- ImpactRange(5,r1) = path (5, 8) $\cup$ path (5, 9) $\cup$ path (5,11)
  = { } $\cup$ { } $\cup$ path(5,11)
  = {5, 7, 10, 11}

Impact Range Example (2)



- subpath(n,p,s) is the path(p,s) $\subset$ path(n,s).
- subpath(1, 5, 11) = path(5, 11) = {5, 7, 10, 11}
- SubPathRange (n,p) = $\cup_s$ subpath (n, p, s)
  for all s $\in$ NextRef(n)
- SubPathRange (S,p) = $\cup_n$ SubPathRange (n, p) for all n $\in$ S
- ImpactRange(n,r) = SubPathRange (NodeHold(n,r),n)
- ImpactRange(5, ‘r1’)
  = SubPathRange(NodeHold(5,r1), 5)
  = SubPathRange({1,2}, 5)
  = subpath(1, 5, 11) $\cup$ subpath(2, 5, 8)
  $\cup$ subpath(2, 5, 9) $\cup$ subpath(2, 5, 11)
  = {5, 7, 10,11 } $\cup$ { } $\cup$ { } $\cup$ {5, 7, 10.11}
  = {5, 7, 10,11}

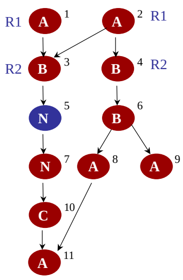
Definition of Impact Set

- Only two types of nodes in the impact range contribute to BenefitRegAlloc(n,r)

1) The node that references the same variable as node ‘n’
2) The node that references the variable that holds register ‘r’ when node ‘n’ is visited for register allocation.

- var(n): the variable that is referenced by node ‘n’.
- VarHold(n,r): the variable that holds register ‘r’ when the register allocation is performed for node ‘n’
- ImpactSet(n,r) = { m| m $\in$ ImpactRange(n,r), and
  (var(m) = var(n) or var(m) = VarHold(n,r)}

Impact Set Example

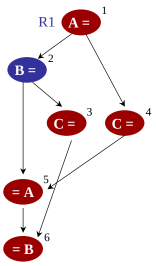


- ImpactRange(5,r1) = path (5, 8) $\cup$ path (5, 9) $\cup$ path (5,11)
  = { } $\cup$ { } $\cup$ path(5,11)
  = {5, 7, 10, 11}
- var(5) = ‘N’
- VarHold(5, ‘r1’) = ‘A’
- ImpactSet(5, ‘r1’) = { 5, 7, 11}

Benefit Model

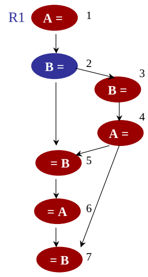
- PenaltySpill(n,r) = $\Sigma_{m \in \text{ImpactSet(n,r) and var(m)=var(n)}}$ cost(m)
- PenaltyPreempt(n,r) = $\Sigma_{m \in \text{ImpactSet(n,r) and var(m)=VarHold(n,r)}}$ cost(m)
  - n: node number, r: register number
  - var(n): the variable that is referenced by node ‘n’.
  - VarHold(n,r): the variable that holds register ‘r’ when the register allocation is performed for node ‘n’.
- BenefitRegAlloc(n,r) = PenaltySpill(n,r) – PenaltyPreempt(n,r)
  - n: node number, r: register number

Cost of a Node



- Let NodeCost(n) be the cost of the execution of ‘n’
- NodeCost(n) = 10^d where *d* is a loop depth of a node n’.
- Consider the cost of ‘2’ for ‘r1’
- ImpactSet(2, ‘r1’) = { 2, 5}
- Node ‘6’ is not in the impact set, however, it is affected by the allocation of node ‘2’
- cost(2) = NodeCost(2) + NodeCost(6)
- $\text{cost(m)}\,_{|\text{m:definition}} = \text{NodeCost(m)} + \Sigma_{\text{k} \in \text{NextRef(m), k:use, k} \notin \text{impact set}} \text{NodeCost(k)}$
- $\text{cost(m)}\,_{|\text{m:use}} = \text{NodeCost(m)} + \Sigma_{\text{k} \in \text{PrevRef(m), k} \notin \text{impact set}} \text{NodeCost(k)}$

Benefit Estimation Example



BenefitRegAlloc(2, R1)
= PenaltySpill(2, R1) - PenaltyPreempt(2,R1)

ImpactSet(2, R1) = {2, 3, 4, 5, 6}

PenaltySpill(2, R1) = cost(2) + cost(3) + cost(5)
= NodeCost(2) + NodeCost(3) +
NodeCost(5) + NodeCost(7)
= 4

PenaltyPreempt(2, R1) = cost(4) + cost(6)
= NodeCost(1) + NodeCost(4) +NodeCost(6)
= 3

BenefitRegAlloc(2, R1) = 4 –3 = 1

Scratch Allocation

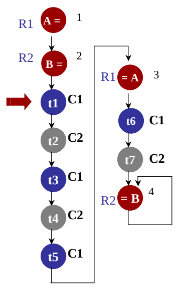
- Unallocated variables and temporaries are allocated.
- Nodes corresponding to temporaries are added to the varef-graph.
- $\text{PenaltyPreempt(s,r)} = \Sigma_{\text{m} \in \text{ ImpactRange(s,r) and var(m)=VarHold(n,r)}} \text{cost(m)}$
- $\text{PenaltySpill(s,r)} = \Sigma_{\text{m} \in \text{ ImpactRange(s,r), m} \in \text{ CLASS(s)}} \text{cost(m)}$
- If a scratch ‘s’ preempts a register ‘r’, then this register can be used for the scratch ‘s’ as well as other scratches that are in the impact range.
- However, not all the scratches in the impact range can be allocated to the same register, due to the overlapping of their live ranges.
- CLASS(s): the class that the scratch ‘s’ belongs to so that all scratches in the class can be allocated to the same register.

Derivation of Class

- All the scratches are colored with infinite colors.
- Scratches are partitioned into classes according to the assigned color.
- Example
  - two classes: {t1, t3, t5} and {t2, t4, t6}

a = t1 + t2 t5 = t3- t4 b = t5 - t6	a = C1 + C2 C1 = C1- C2 b = C1 - C2

Allocation Example (1)



Assume two registers. At node ‘t1’, runs out of registers

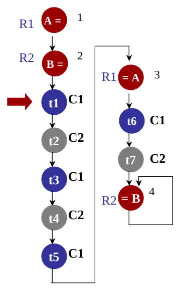
ImpactRange(t1, R1) = {t1, t2, t3, t4, t5, 3}

PenaltyPreempt(t1, R1) = cost(3)
= NodeCost(1) + NodeCost(3) = 2

PenaltySpill(t1, R1) = cost(t1) + cost(t3) + cost(t5) = 3

BenefitRegAlloc(t1, R1) = 3 –2 = 1

Allocation Example (2)



BenefitRegAlloc(t1, R1) = 3 –2 = 1

ImpactRange(t1, R2) = {t1, t2, t3, t4, t5, 3, t6, t7,4}

PenaltyPreempt(t1, R2) = cost(4)
= NodeCost(2) + NodeCost(4) = 11

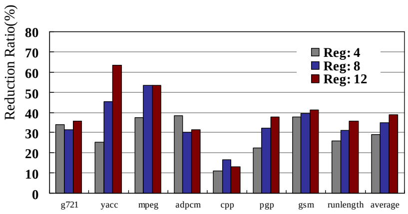
PenaltySpill(t1, R2) = NodeCost(t1) + NodeCost(t3) +
NodeCost(t5) + NodeCost(t6) = 4

BenefitRegAlloc(t1, R2) = 4–11 = -7

‘t1’ preempts register ‘R1’, and ‘t1’, ‘t3’, and ‘t5’ are assigned to ‘R1’.

Experimental Results

- Implemented in LCC targeting ARM7TDMI
- With the eight benchmarks, an average of 34.3% improvement is achieved over the graph coloring approach



Compilation Time Measurements

On average, 1.85 times larger than that for Briggs' allocator.

benchmark	Number of registers		
	4	8	12
g721	1.64	1.86	1.97
yacc	1.73	2.13	2.01
mpeg	3.28	2.77	2.79
adpcm	1.29	1.49	1.62
rep	2.21	2.00	2.17
pgp	1.42	1.75	1.67
gsm	1.49	1.24	1.10
runlength	1.34	1.41	1.93

Complexity Analysis

- The dominant complexity: the derivation of the impact range
- N: the number of nodes in the varef-graph
- The derivation of the impact range of a node for a register: O(N)
- Iterated N times for each node ;
- Total Complexity: $O(N^2)$
- In practice, the next reference of a variable is generally located close to the node, thus search spaces are localized.

Conclusions

- Improves the Briggs' allocator by an average of 34.3%
- The compilation time increase by the amount of 85%
- Time overhead is not serious considering that graph-coloring allocators run fast in practice.
- The proposed varef-graph can be used for further optimizations such as instruction scheduling