Linear Scan Register Allocation

Kevin Millikin

Register Allocation

Register Allocation

- An important compiler optimization
- Compiler: unbounded # of virtual registers
- Processor: bounded (small) # of registers (k)
- Assign virtual registers to processor registers

Liveness Analysis

A virtual register is live at a program point if it is read before being written on some path to the program's exit

Liveness analysis: compute for each program point the set of live virtual registers

Interference Graph

Construct a graph with

- vertexes all virtual registers
- edges between each pair live at the same program point

Virtual registers that interfere cannot be allocated to the same register

Graph Coloring

Assign one of k colors to each vertex, such that no adjacent pairs of vertexes have the same color

Produce a k-coloring or no such coloring

NP-complete for $k > 2$

Spilling Virtual Registers

If the interference graph is not k-colorable, spill a virtual register (assign it to memory)

Try to k-color the resulting graph, repeat.

Linear Scan

Linear Scan

Poletto, M., & Sarkar, V. (1999). Linear scan register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *21*(5), 895-913.

Drawbacks of Graph Coloring

Too slow to use in a JIT compiler

Number the Instructions

Consecutively

The algorithm works for any ordering

The specific ordering is important

Approximate Liveness

Liveness analysis: compute for each program point the set of live virtual registers

Compute for each virtual register a set of program points where it is live

Compute for each virtual register the first and last program point where it is live

A Key Insight

Live ranges are intervals

The number of live virtual registers only changes when an interval starts or ends

Sort the Live Ranges

Sort them by their start point

Scan them in order, use a greedy strategy to assign registers

Linear Scan Implementation

```
class Interval {
  int start, end, reg;
}
int active[k];  // Register number or -1.
for (Interval i in intervals) {
  // Implementation goes here.
}
```

Try to Allocate a Free Register

Loop over the active live ranges.

Tentatively assign the given live range to the first free register.

Fail if there is no free register.

Expire Old Live Ranges

Loop over the active live ranges.

Remove any that end before the given live range starts.

This frees their register.

Allocate a Blocked Register

Consider the active live ranges and also the given live range.

Spill the one with the lowest cost based on a heuristic.

If the given live range was not spilled, tentatively assign it the free register.

Putting It All Together

```
for (Interval i in intervals) {
  ExpireOldIntervals(i);
  if (!AllocateFreeRegister(i)) {
    AllocateBlockedRegister(i);
  }
}
```

Instruction Order

Poletto and Sarkar use reverse postorder

They also considered code (linear) order

There was negligible impact

Spill Cost Heuristic

Poletto and Sarkar use last end point

They considered weights based on estimated use counts

There was negligible impact

Second-chance Binpacking

Second-chance Binpacking

Traub, O., Holloway, G., & Smith, M. D. (1998, June). Quality and speed in linear-scan register allocation. In *Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation* (Vol. 17, No. 19, pp. 142-151).

Drawbacks of Linear Scan

A virtual register is spilled for its entire live range.

Lifetime holes are ignored.

Scan the Virtual Register Uses

This algorithm is somewhat different.

Scan the uses (not the live ranges).

Rewrite the program on the fly.

Spilling

A register to memory move is inserted at the spill point

Previous uses of the spilled virtual register are unaffected

Later uses are candidates for allocation when they come up

Spill Store Elimination

Track the value in each spill slot

Do not issue a store if the value is already spilled

Spill Cost Heuristic

Latest next use (remember Poletto and Sarkar used latest end point)

Weighted by loop nesting depth

Resolution

Different allocation decisions on different control-flow paths (e.g., spill/reg, reg/reg).

Resolved at control-flow joins by moves in the predecessor block.

Implemented by iterating the control-flow edges.

Sink Stores and Hoist Loads

Resolution can insert unnecessary stores.

Move stores downward and loads upward, eliminate them (replace with a move) if they meet.

Dataflow Analysis for Consistency

Spill store elimination can be inconsistent

Value assumed spilled is not spilled on all paths

Requires a dataflow analysis and insertion of stores for correctness

Linear Scan in the Context of SSA

Linear Scan in the Context of SSA

Mössenböck, H., & Pfeiffer, M. (2002). Linear scan register allocation in the context of SSA form and register constraints. In *Compiler Construction* (pp. 153-206). Springer Berlin/Heidelberg.

Static Single Assignment Form

Each virtual register assigned once.

Phi functions to resolve control flow.

Phi moves are inserted before register allocation -- partial translation out of SSA.

Liveness Analysis

Liveness analysis computes lifetime holes.

Live range is a collection of live intervals.

Joining Virtual Registers

Before register allocation, virtual registers are connected if they should share a register.

Example: phi moves

Example: x86 two-operand instructions

Inactive Intervals

In addition to active intervals (allocated to a register and live)

Inactive intervals are allocated to a register but in a lifetime hole

Expire Old Live Ranges

Active live ranges: if ended are removed,

if reached a lifetime hole are moved to inactive

Reactivate Inactive Live Ranges

Inactive live ranges: if ended are removed,

if reached the end of a lifetime hole are moved to active

Try to Allocate a Free Register

Same as linear scan, but

a register is not free if there is an inactive interval allocated to it that overlaps a given live range.

Allocate a Blocked Register

Same as linear scan, but

consider inactive live ranges (as well as active) for spilling

Notice

No splitting

No resolution

Optimized Interval Splitting

Optimized Interval Splitting

Wimmer, C., & Mössenböck, H. (2005, June). Optimized interval splitting in a linear scan register allocator. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments* (pp. 132-141). ACM.

Try to Allocate a Free Register

If the given live range intersects with one from inactive,

split the given live range and the end of the lifetime hole.

Resolution

Now, resolution is required again.

Optimal Split Positions

Second-chance binpacking split as late as possible.

Move split positions out of loops.

Move split positions to basic-block boundaries.

Register Hints

Lightweight coalescing

Rather than joining live ranges, record the register of the first as a hint to the second

Hints are preferred but can be ignored

Spill Store Elimination

As in second-chance binpacking

Linear Scan on SSA Form

Linear Scan on SSA Form

Wimmer, C., & Franz, M. (2010, April). Linear scan register allocation on SSA form. In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization* (pp. 170-179). ACM.

Resolution

Phi moves were used to translate out of SSA before register allocation

Resolution inserts moves to resolve control flow after register allocation

They can be combined by inserting all moves after register allocation

Conclusion

Conclusion

A relatively new technique in compilers (less than 15 years)

Universally adopted

Easy to implement in its simplest forms

Performance approaches graph coloring