Graph colouring register allocation for the Glasgow Haskell Compiler

Ben Lippmeier

Australian National University

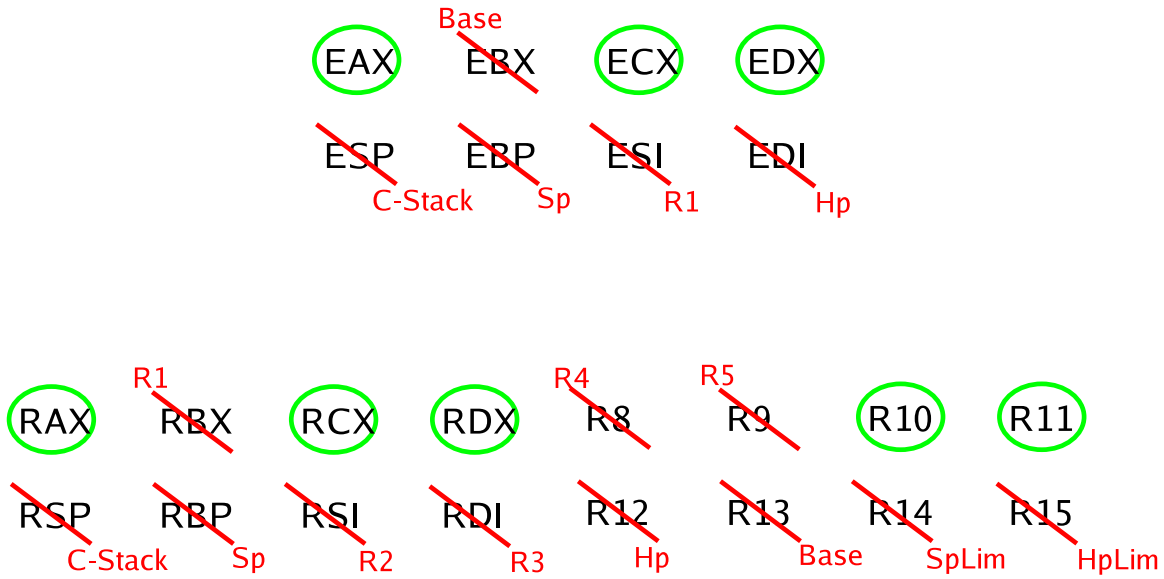
Register allocation

- GHC's native code generator (NCG) compiles `C--` code into native machine instructions.
- The code generator assumes that an unlimited set of virtual registers *(vregs)* are available.
- The real hardware has a small, fixed number of real registers *(hregs)*.
- The allocator must rewrite uses of vregs into uses of hregs, spilling to the stack if need be.

Stolen registers

Most hregs are stolen by the CPS converter for its own evil purposes.

On some architetures there aren't many hregs left to allocate to:
(**x86:** 3d/6f – **x86_64:** 5d/2f – **ppc:** 16d/26f).



The stack based nature of GHC's backend results in minimal pressure being placed on the register set (mostly).

For *lazy* code, the implementation spends the vast majority of its time copying data around memory.

```
c11h:
  movl  %esi, %vI_n11l        |b
  andl  $3, %vI_n11l          |
  cmpl  $2, %vI_n11l          |d
  jae   .Lc11i
  addl  $20, %edi
  cmpl  92(%ebx), %edi
  ja    .Lc11k
  movl  $sXd_info, -16(%edi)
  movl  12(%ebp), %vI_n11m     |b
  movl  %vI_n11m, -8(%edi)     |d
  movl  8(%ebp), %vI_n11n        |b
  movl  %vI_n11n, -4(%edi)       |d
  movl  4(%ebp), %vI_n11o          |b
  movl  %vI_n11o, (%edi)           |d
  leal  -16(%edi), %vI_n11p          |b
  movl  %vI_n11p, 8(%ebp)            |d
  movl  $base_GHCziHandle_stdout_closure, 4(%ebp)
  movl  $sZt_info, 12(%ebp)
  addl  $4, %ebp
  jmp   base_GHCziIO_a16_info
```

However, if strictness analysis, inlining and unboxing optimisations are successful the situation can be very different.

The native code for an implementation of SHA1 has vregs which live for 1700+ instructions, with 30+ live at once.

```
...                                  Fy Gp Fr xe
andl  %vI_s3Fy, %vI_n7xh             |
movl  %vI_s3Fr, %vI_n7xe             |     |b |b
andl  %vI_s3G2, %vI_n7xe             |     |  |
orl   %vI_n7xh, %vI_n7xe             |     |  |
addl  %vI_n7xf, %vI_n7xe             |     |  |
movl  %vI_s3FV, %vI_n7xi             |     |  |
shrl  $27, %vI_n7xi                  |     |  |
movl  %vI_s3FV, %vI_s3Gp             |  |b |  |
shll  $5, %vI_s3Gp                   |  |  |  |
orl   %vI_n7xi, %vI_s3Gp             |  |  |  |
addl  %vI_n7xe, %vI_s3Gp             |  |  |  |d
movl  %vI_s3Fr, %vI_n7xj             |  |  |
shrl  $2, %vI_n7xj                   |  |  |
movl  %vI_s3Fr, %vI_s3Gw             |  |  |d
shll  $30, %vI_s3Gw                  |  |
orl   %vI_n7xj, %vI_s3Gw             |  |
movl  280(%ebp), %vI_n7Ql            |  |
leal  18..(%vI_n7Ql), %vI_n7xm       |  |
movl  %vI_s3Fy, %vI_n7xl             |  |
... etc etc
```

Existing allocator

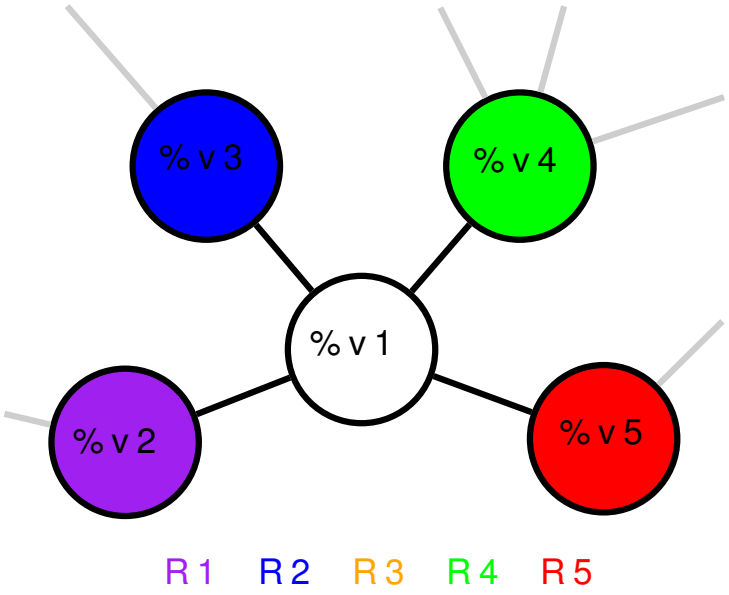
- Performs a single linear scan through the code.
- When not enough hregs are available, just spills the first one that isn't used in the current instruction.
- Handling of join points/loops requires extra code to be inserted which connects up the different vreg $\Leftrightarrow$ hreg mappings.
- No lookahead - can't see that a hreg will be clobbered in the next instruction

Graph colouring

- Build a register conflict graph.
  - Nodes represent vregs.
  - Edges show what vregs cannot be allocated the same hreg.
- Assign a colour (hreg) to each node so that adjacent nodes always have different colors.
- If no coloring can be found then insert spill code and start over.
- Patch the code with the vreg $\Leftrightarrow$ hreg mapping from the graph.
- Optimal graph colouring is NP-complete, must fall back to heuristic algorithms.

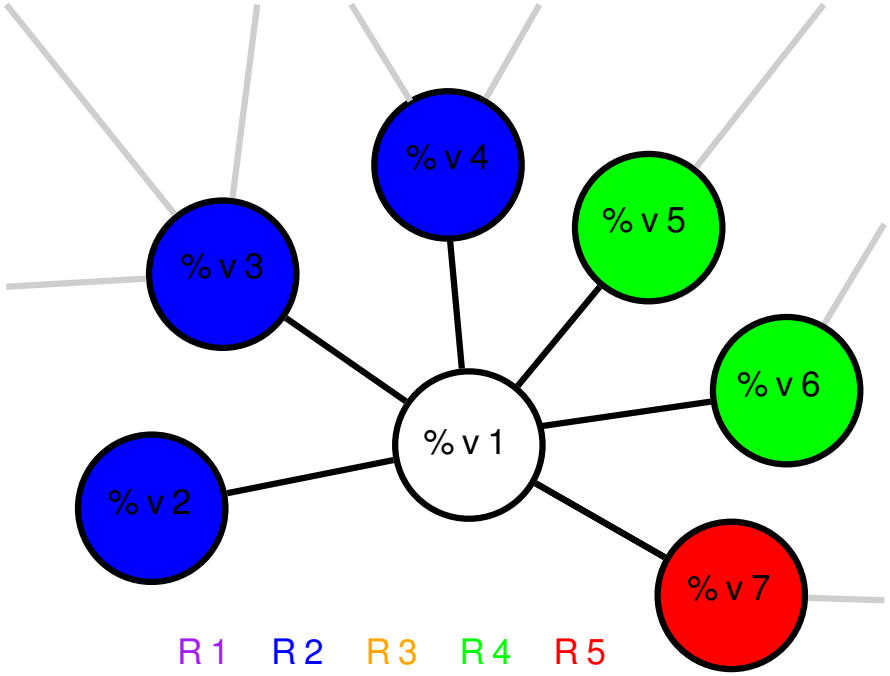
Squeese and trivial colourability

- The *squeese* of a node is the number of colours denied to it due to the colouring of its neighbours.
- If *squeese* $<$ *number of colors available*, then the node is said to be *trivially colourable*.

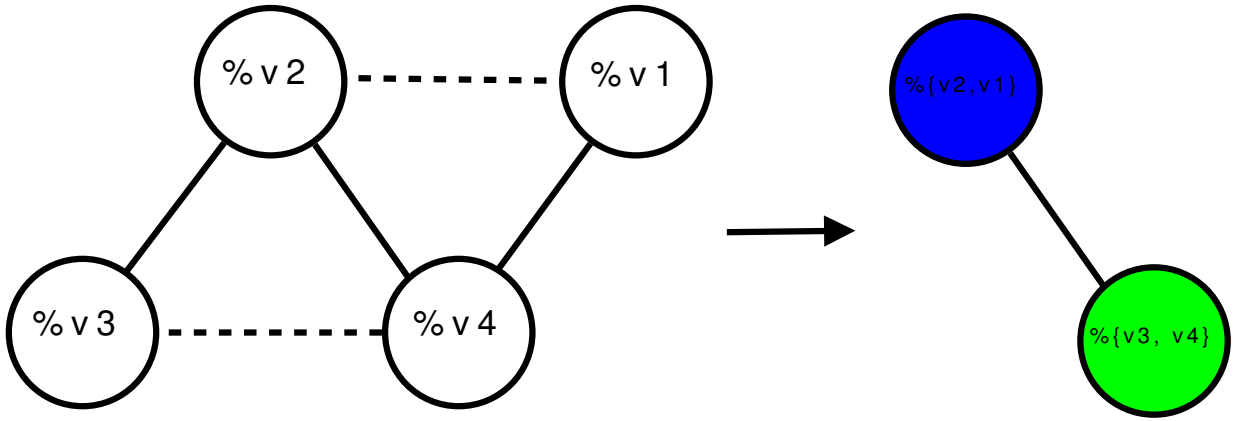


Optimistic colouring

If a node has *more* neighbours than available colors it may still be colourable - depending on the colors assigned to its neighbors.

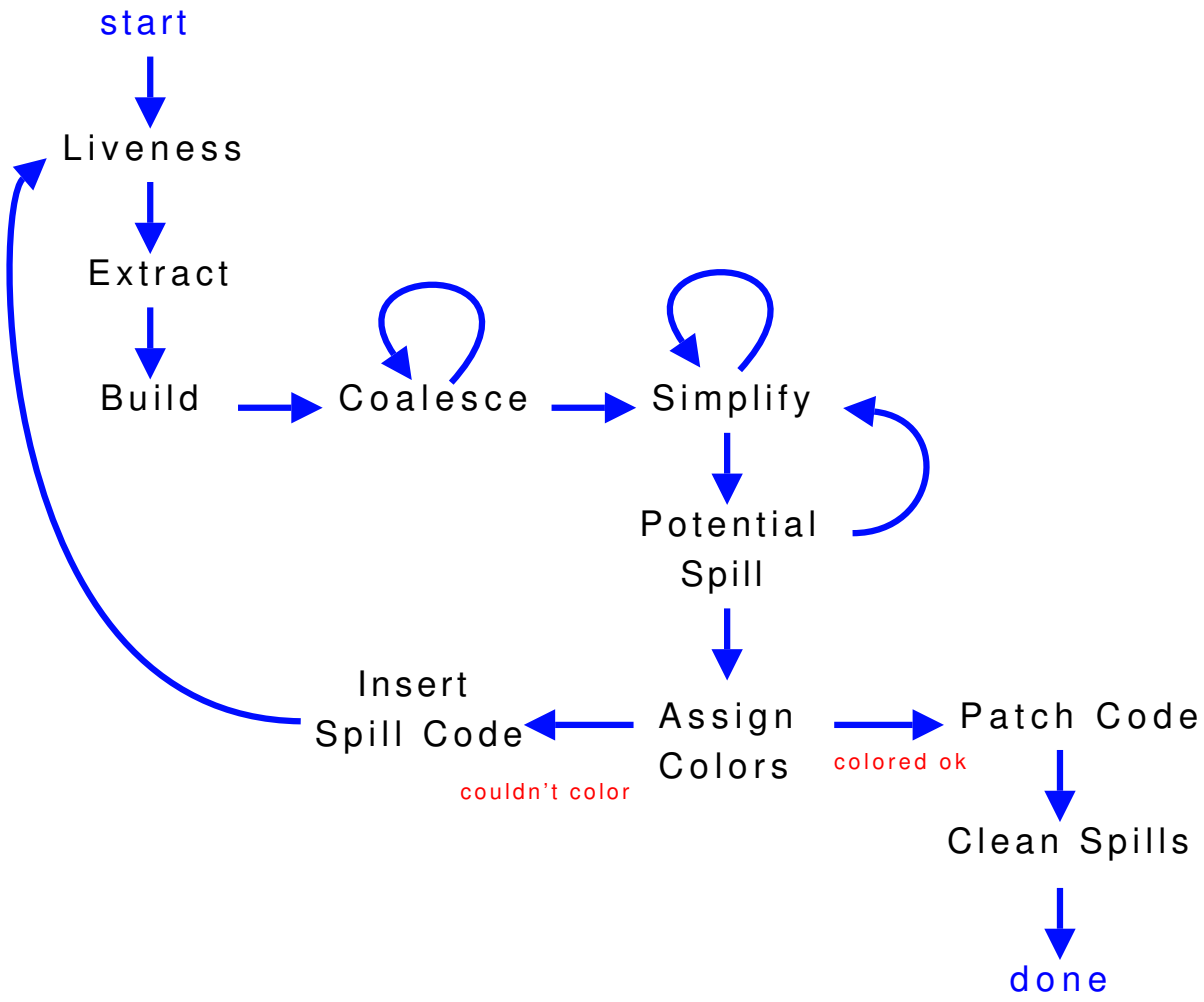


Coalescing

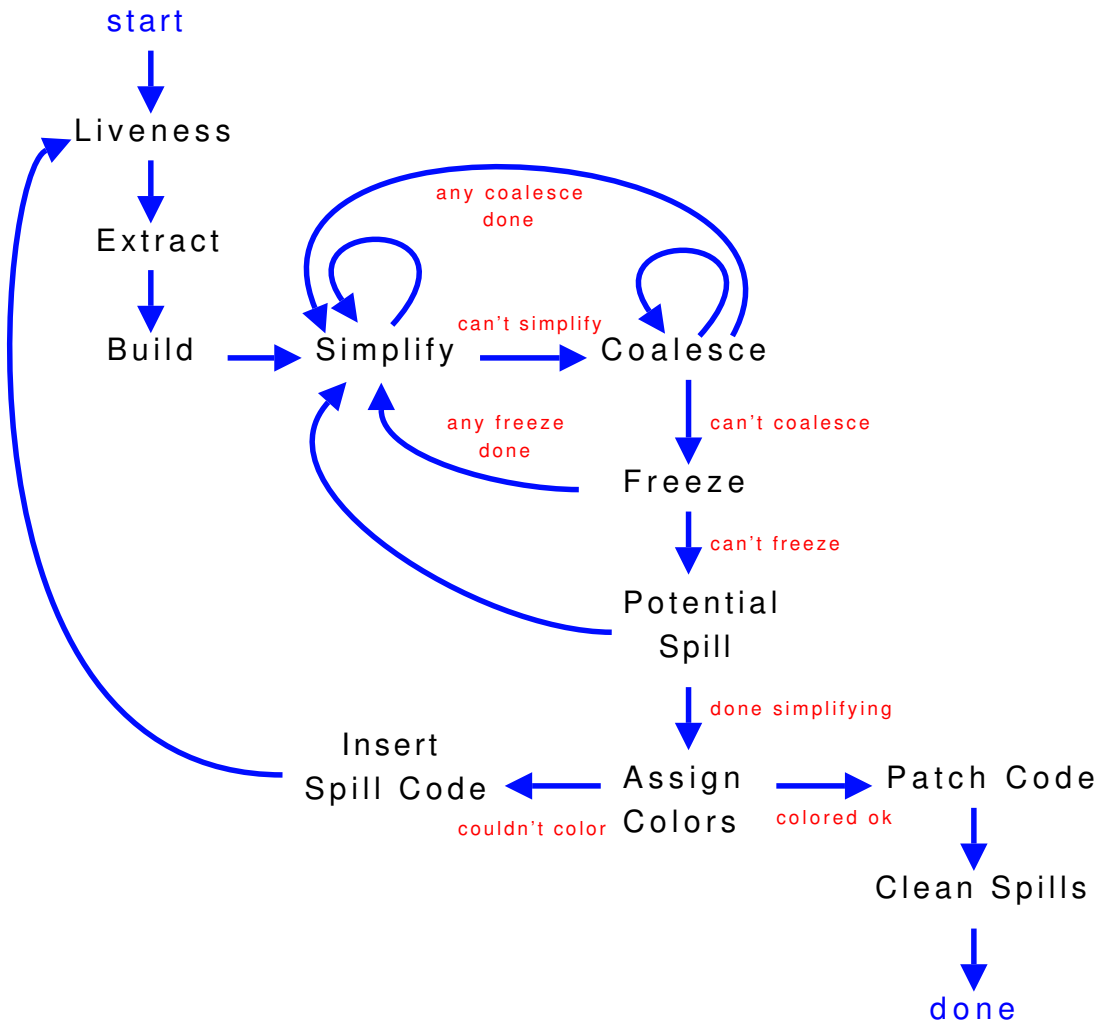



```
                    v1  v2  v3  v4
mov %v1, %v2        |d  |b      |          -mov %R1, %R1
add %v4, %v2            |       |           add %R2, %R1
mov %v3, %v4            |   |b  |          -mov %R2, %R2
mul %v2, %v4            |d  |   |           mul %R1, %R2
```

New allocator (non-iterative version)

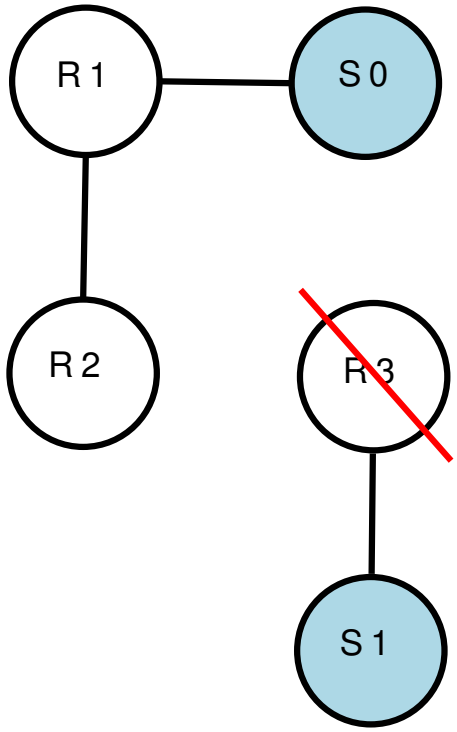


New allocator (with iterative coalescing)



Spill cleaning

```
 add     %R0,      %R1
 SPILL   %R1,      SLOT(0)
 mov     %R1,      %R2
-RELOAD  SLOT(0),  %R1
 add     %R1,      %R3
 SPILL   %R3,      SLOT(1)
-RELOAD  SLOT(0),  %R2
 mul     %R2,      %R3
```



Interface

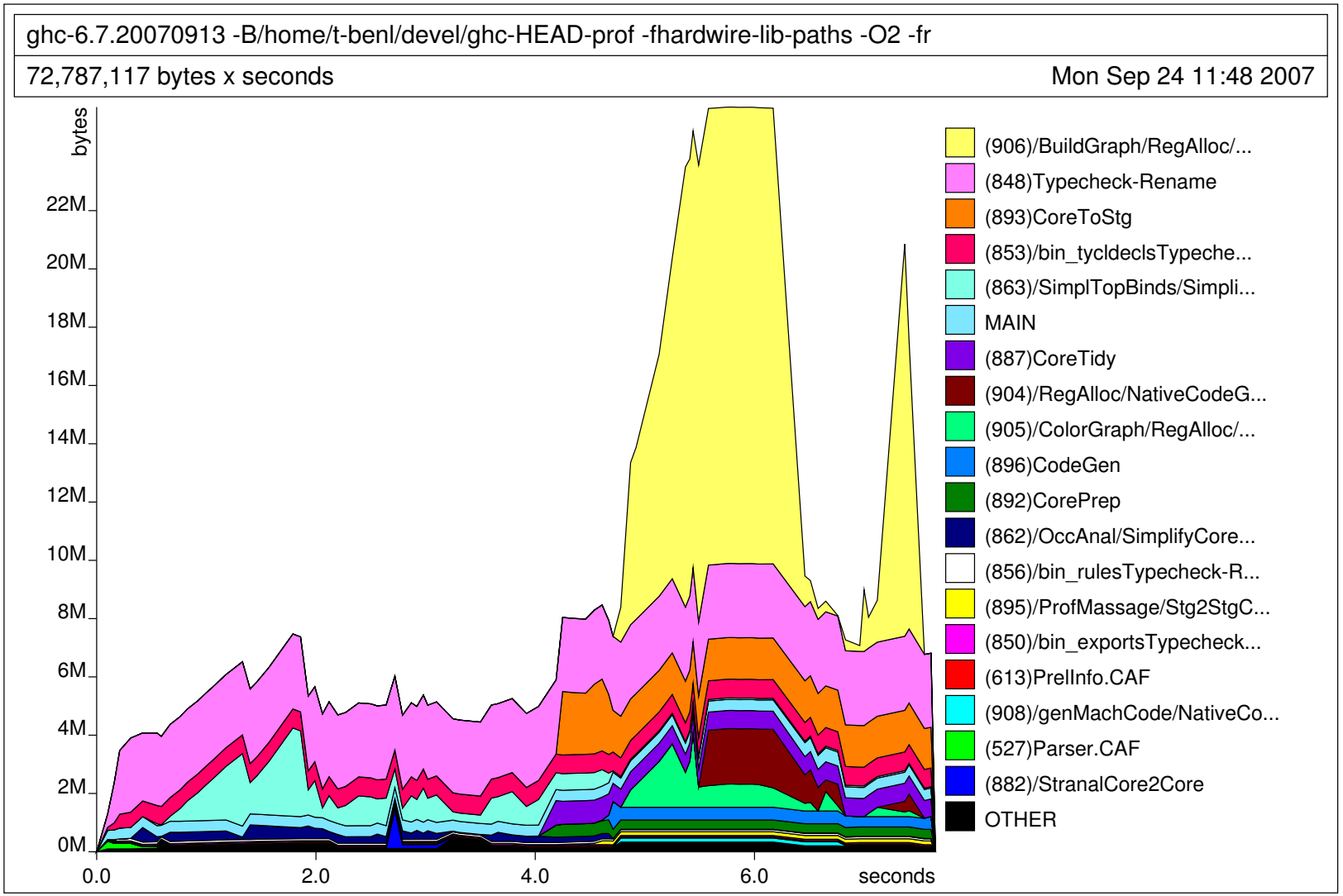
```
colorGraph
  :: ( Uniquable k, Uniquable cls, Uniquable color
     , Eq color, Eq cls, Ord k)
  => Bool -> Int
  -> UniqFM (UniqSet color)      -- class -> colors
  -> Triv   k cls color
  -> (Graph k cls color -> k)    -- potential spill
  -> Graph  k cls color

  -> ( Graph k cls color     -- colored graph
     , UniqSet k             -- uncolored nodes
     , UniqFM  k )           -- coalesce rewrite
```

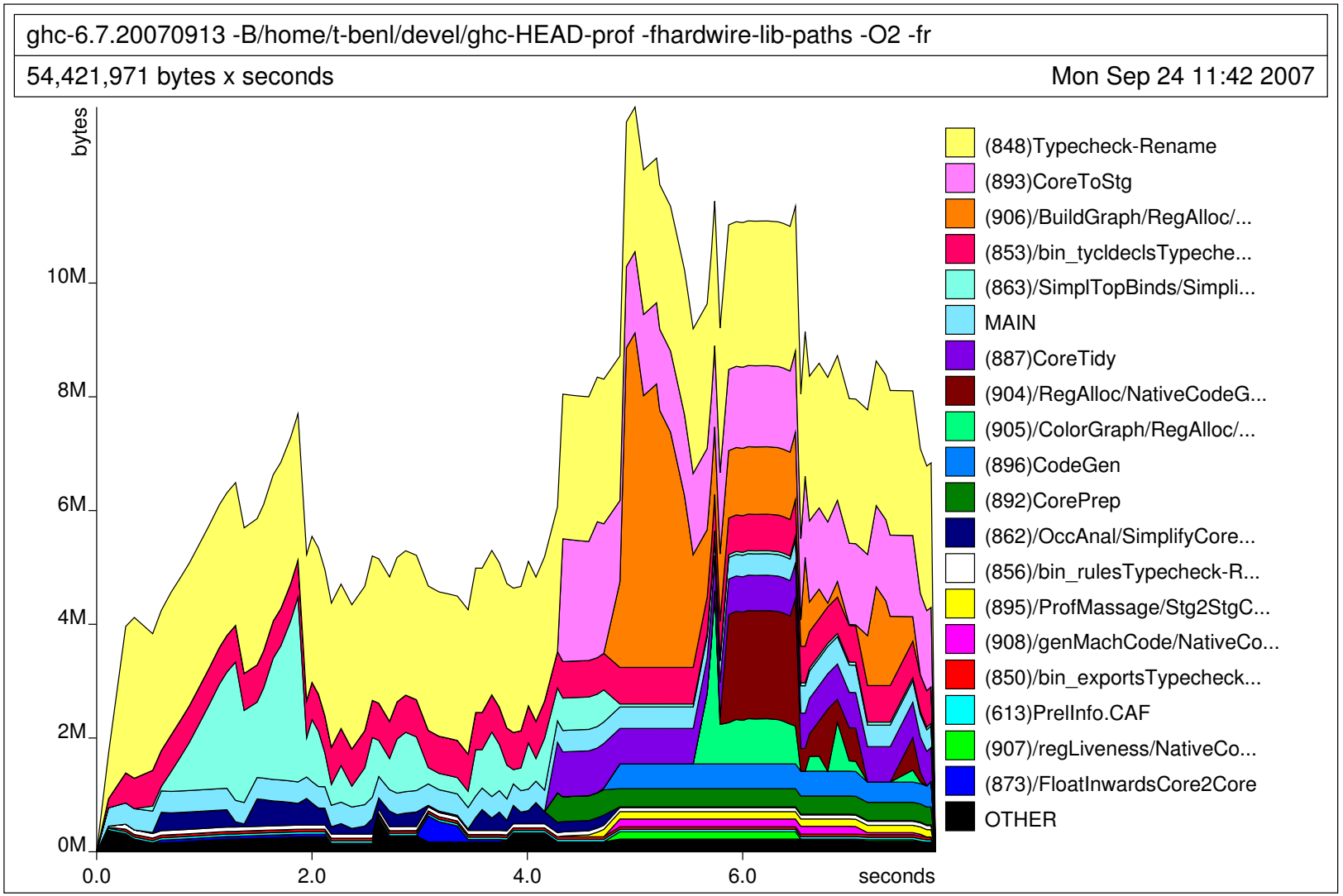
```
type Triv k cls color
     =  cls                 -- node class
     -> UniqSet k           -- neighbors
     -> UniqSet color       -- exclusions
     -> Bool
```

Runtime

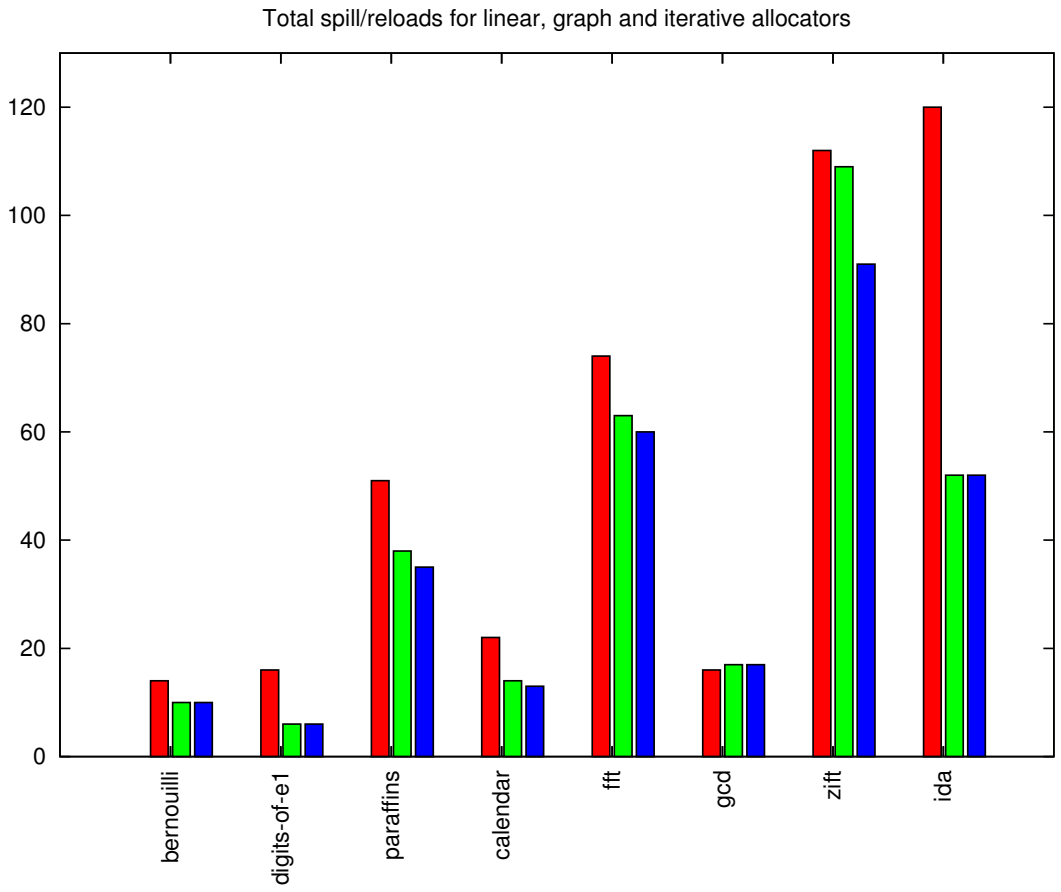
- Graph is implemented as `UniqFM`s of nodes, where the nodes contain `UniqSet`s of neighbour keys.
  - O(log n) insert and lookup.
- To check whether a node is trivially colourable we must count the number of keys in a node's conflict set.
  - nice `foldUniqSet` with unboxed math is too slow.
  - do unboxed accumulation directly over `UniqFM` data type.
- Must deep seq graph 'after' building, else graphs from different build/spill cycles will be live at once.

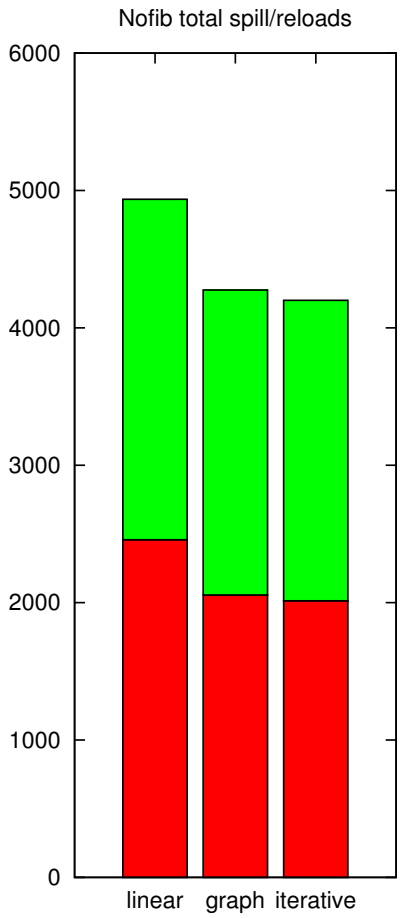
```
#define ALLOCATABLE_REGS_INTEGER 3#
#define ALLOCATABLE_REGS_DOUBLE  6#

isSqueesed :: Int# -> Int# -> UniqSet Reg -> (# Bool, Int#, Int#)
isSqueesed cI cF ufm
 = case ufm of
        NodeUFM _ _ left right
         -> case isSqueesed cI cF right of
             (# s, cI', cF' #)
                -> case s of
                        False   -> isSqueesed cI' cF' left
                        True    -> (# True, cI', cF' #)

        LeafUFM _ reg
         -> case regClass reg of
                RcInteger
                 -> case cI +# 1# of
                        cI' -> (# cI' >=# ALLOCATABLE_REGS_INTEGER, cI', cF #)

                RcDouble
                 -> case cF +# 1# of
                        cF' -> (# cF' >=# ALLOCATABLE_REGS_DOUBLE,  cI, cF' #)

        EmptyUFM
         -> (# False, cI, cF #)
```


ghc-6.7.20070913 -B/home/t-benl/devel/ghc-HEAD-prof -fhardwire-lib-paths -O2 -fr
72,787,117 bytes x seconds
Mon Sep 24 11:48 2007
bytes
22M
20M
18M
16M
14M
12M
10M
8M
6M
4M
2M
0M
0.0
2.0
4.0
6.0
seconds
(906)/BuildGraph/RegAlloc/...
(848)Typecheck-Rename
(893)CoreToStg
(853)/bin_tycldeclsTypeche...
(863)/SimplTopBinds/Simpli...
MAIN
(887)CoreTidy
(904)/RegAlloc/NativeCodeG...
(905)/ColorGraph/RegAlloc/...
(896)CodeGen
(892)CorePrep
(862)/OccAnal/SimplifyCore...
(856)/bin_rulesTypecheck-R...
(895)/ProfMassage/Stg2StgC...
(850)/bin_exportsTypecheck...
(613)PrelInfo.CAF
(908)/genMachCode/NativeCo...
(527)Parser.CAF
(882)/StranalCore2Core
OTHER


ghc-6.7.20070913 -B/home/t-benl/devel/ghc-HEAD-prof -fhardwire-lib-paths -O2 -fr
54,421,971 bytes x seconds
Mon Sep 24 11:42 2007
bytes
10M
8M
6M
4M
2M
0M
0.0
2.0
4.0
6.0
seconds
(848)Typecheck-Rename
(893)CoreToStg
(906)/BuildGraph/RegAlloc/...
(853)/bin_tycldeclsTypeche...
(863)/SimplTopBinds/Simpli...
MAIN
(887)CoreTidy
(904)/RegAlloc/NativeCodeG...
(905)/ColorGraph/RegAlloc/...
(896)CodeGen
(892)CorePrep
(862)/OccAnal/SimplifyCore...
(856)/bin_rulesTypecheck-R...
(895)/ProfMassage/Stg2StgC...
(908)/genMachCode/NativeCo...
(850)/bin_exportsTypecheck...
(613)PrelInfo.CAF
(907)/regLiveness/NativeCo...
(873)/FloatInwardsCore2Core
OTHER

Performance





Out in the Wild

- Released in GHC 6.8
- Old linear allocator is still the default for now.
  - Use `-fregs-graph` or `-fregs-iterative` to enable the new allocator.
- New CPS converter should result in more use of the register set, while freeing up some of the pinned registers - net *decrease* in register pressure?