Weaving Source Code into ThreadScope

Peter Wortmann
scpmw@leeds.ac.uk

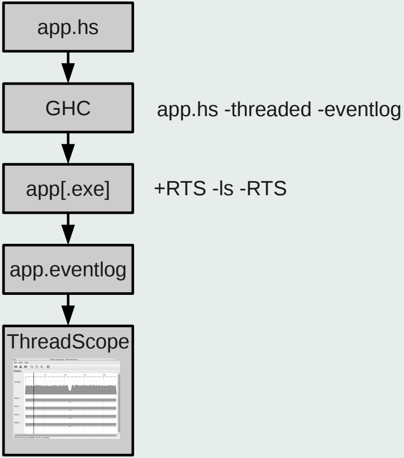
University of Leeds
Visualization and Virtual Reality Group

sponsorship by
Microsoft Research

Haskell Implementors' Workshop 2011

UNIVERSITY OF LEEDS

ThreadScope Work-Flow



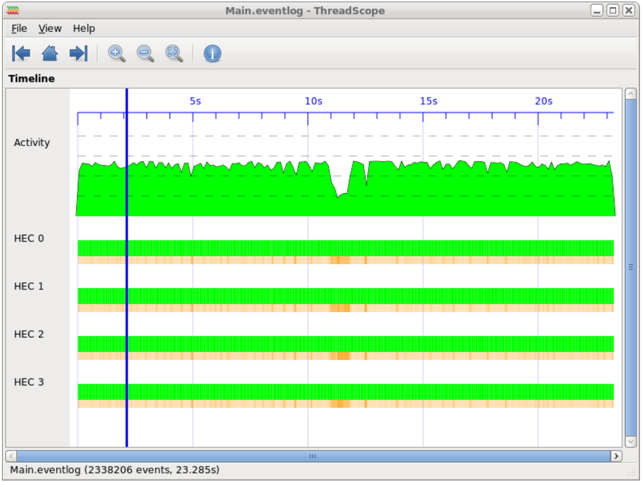
For reference:

Event-Log

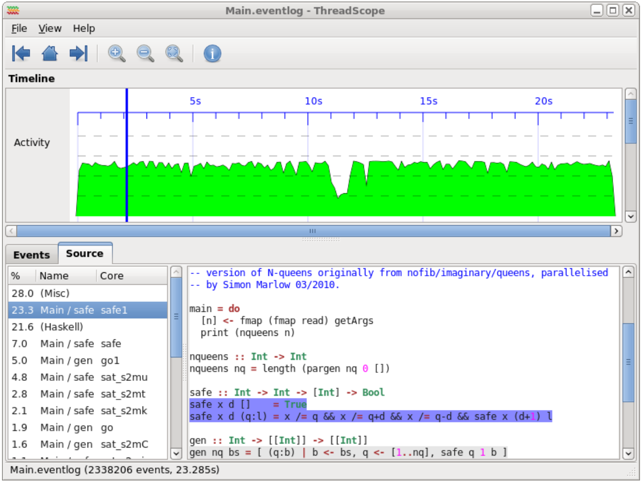
Trace of the GHC run time system.
Extensible to carry other data as required.

ThreadScope

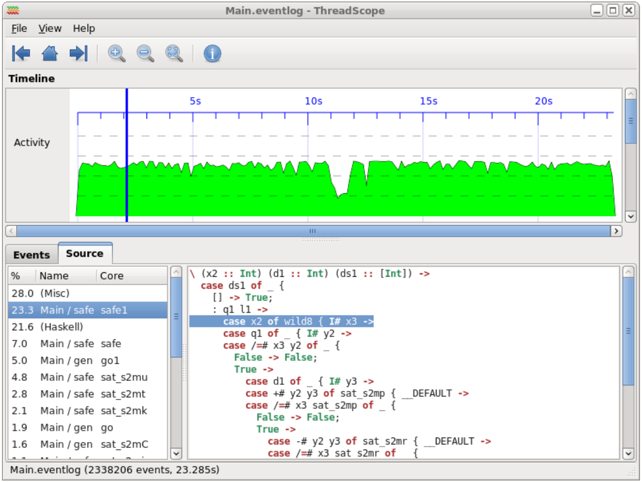
The principal visualisation tool for event-log traces



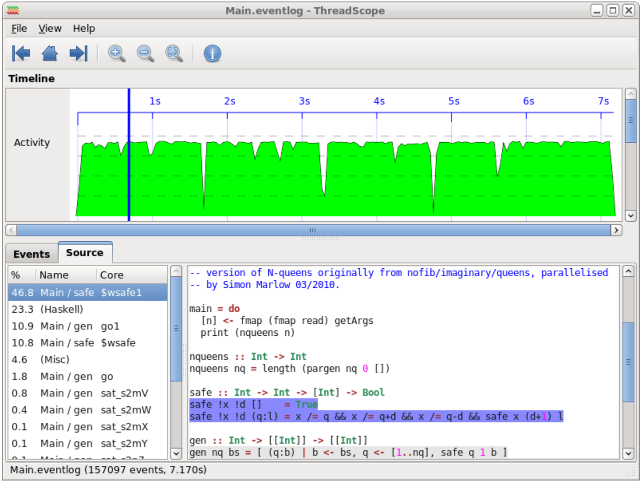
Speedup low for 4 cores. . . What is the reason?



Main worker only active 23% of the time! Not good.



Okay, this should *not* happen.



A simple strictness annotation gives 3 fold speed-up.

Design Trade-Offs

Cannot have everything at once

UNIVERSITY OF LEEDS

The Goal

Timestamped source-level profiling data.

... written out:

Accurate profiling

- Reliable performance data
- Reflect original program well
  $\Rightarrow$ Allow for optimisations!

Source Code Hints

- Helpful cost allocation
- User friendly
  (automatic in a useful way)

Good Time Resolution

Data for every point in time

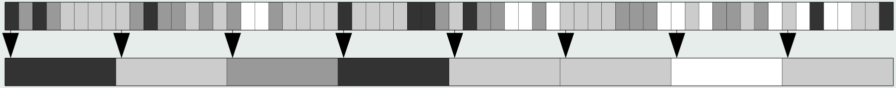
Future Proof

Multi-Core, cache misses. . .

Main Problem

Program execution is fast! ⇒ Lots of data, cannot possibly retain in full

Sampling



1. Write status info into known memory location
2. Periodically look up and save a sample

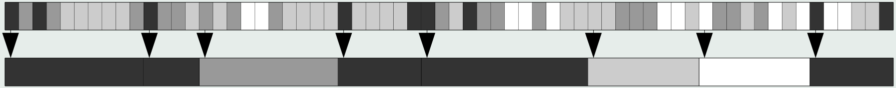
Distribution of samples expected reasonably close to "true" distribution

Bonus: Variable periods allow special sampling (e.g. **cache misses**)

Main Problem

Program execution is fast! $\Rightarrow$ Lots of data, cannot possibly retain in full

Sampling



1. Write status info into known memory location
2. Periodically look up and save a sample

Distribution of samples expected reasonably close to "true" distribution

Bonus: Variable periods allow special sampling (e.g. **cache misses**)

Hardware Support

Modern CPUs support *Hardware Performance Counters*:

- Special registers count events/statistics (cycles, branch misses...)
- Programmable so program gets interrupted on threshold

Properties:

- Very reliable performance data ("outsider" perspective)
- Fast & flexible

Operation system support spotty, though:

Linux:	PAPI & perf_events!
Windows:	(needs driver?)
Mac Os:	(undocumented?)

Portable Alternatives

This is for Microsoft Research, after all

UNIVERSITY OF LEEDS

Plain Timers

Use a simple timer for sampling

- Only by time — not what we want, strictly speaking
- Again unportable below $\sim$ 10ms?
- Harder to get to thread data

Instrument

Prefix all generated code chunks to sum up status changes in table

- Has access to thread-local state (allocations)!
- Relatively slow: $\sim$ 60% slowdown for cycle counter

Bottom Line: Support hardware counters and instrumentation.

Sampling Question

What source code executed here?

1 Cost Centres [SansomJones1997]

- Instrument program on functional level
- Restrict code transformations

⇒ Good source attribution, concerning subtly different program

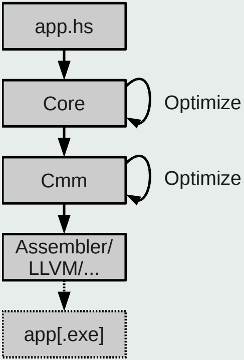
2 Our approach

- Minimal or no instrumentation – just look at instruction pointer!
- **Follow** code transformations

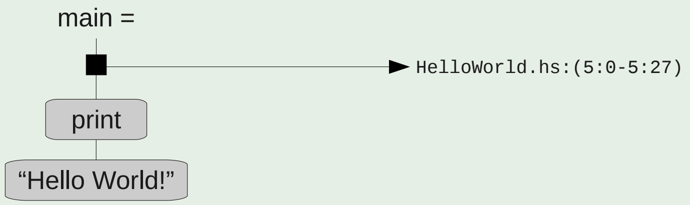
⇒ Worse source attribution on fully optimised program

GHC stages we must make transparent

1. Haskell program
2. Functional representation
   (functions, lets, cases...)
3. Imperative representation
   (procedures, blocks, instructions...)
4. Low-level assembly
5. Linked executable



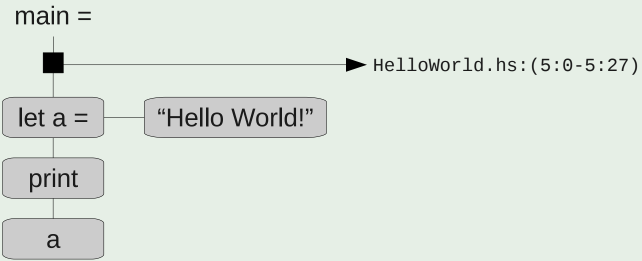
Put annotations into expression graph, update for optimisations [1]:



- Code gets separated → duplicate annotation
- Code gets (partially) removed → remove/move annotation
- Code gets integrated → *allow overlap?*

[1] Not quite the same as [SansomJones1997], [GillRunciman2007]

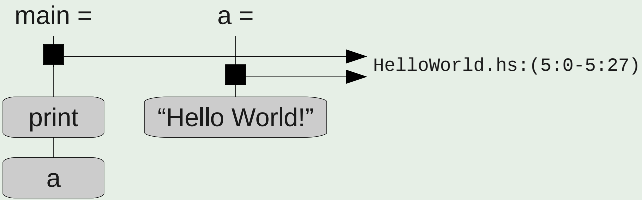
Put annotations into expression graph, update for optimisations [1]:



- Code gets separated → duplicate annotation
- Code gets (partially) removed → remove/move annotation
- Code gets integrated → *allow overlap?*

[1] Not quite the same as [SansomJones1997], [GillRunciman2007]

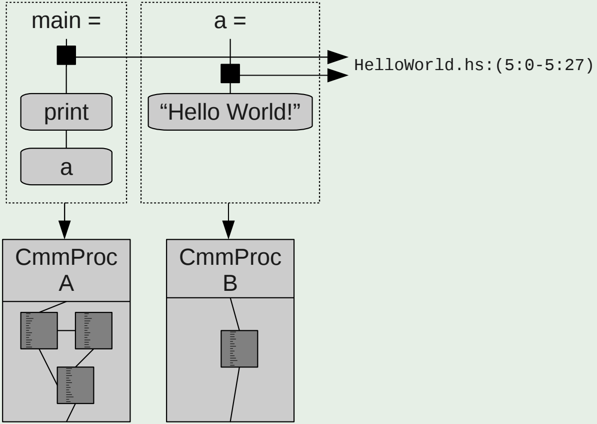
Put annotations into expression graph, update for optimisations [1]:



- Code gets separated → duplicate annotation
- Code gets (partially) removed → remove/move annotation
- Code gets integrated → *allow overlap?*

[1] Not quite the same as [SansomJones1997], [GillRunciman2007]

Generated closure code is imperative-style procedures & blocks

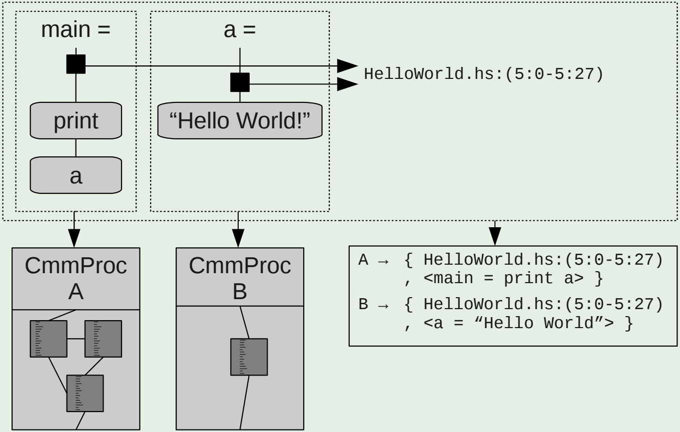


Cmm transformations only touch blocks ⇒ can separate data (retain Core!)

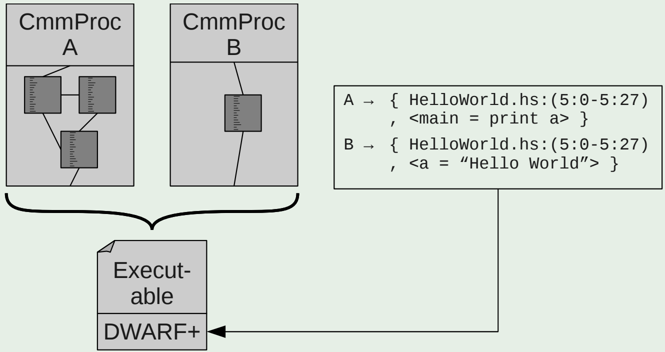
Dealing with Code Generation
... from functional to imperative

UNIVERSITY OF LEEDS

Generated closure code is imperative-style procedures & blocks



Cmm transformations only touch blocks ⇒ can separate data (retain Core!)



Linking is done by external programs (**LLVM** & **GCC**). Split debug data:

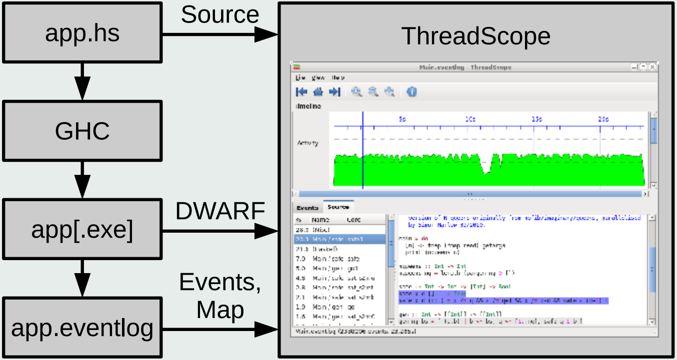
- Use C-style DWARF format where possible (will be kept consistent!)
- Put rest into binary to be prepended to event-log

Wrapping up

Tying everything together

UNIVERSITY OF LEEDS

The New Workflow



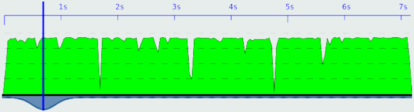
ThreadScope Visualization

What to make of the data

UNIVERSITY OF LEEDS

Weighting samples

What samples to use at point?
⇒ Weight those found nearby



Many procedures per function

Code often very splintered up
⇒ Subsume shared names/cores!

5.0	Main / gen	go1
4.8	Main / safe	sat_s2mu
2.8	Main / safe	sat_s2mt
2.1	Main / safe	sat_s2mk

```
nqueens :: Int -> Int
nqueens nq = length (pargen nq 0 [])

safe :: Int -> Int -> [Int] -> Bool
safe x d []    = True
safe x d (q:l) = x /= q && x /= q+d &&
```

Many functions per procedure

Inlining distributes responsibility
⇒ Mark all or use heuristic

```
for_each xs init op = foldl' op init xs
{-# INLINE for_each #-}

join_tree :: [Int] -> (Int -> [Int]) -> Graph
join_tree vertices adjacent
    -- For each vertex v in the dataset ..
```

Project Status — Future Work:

Profiling

Works well, a bit restricted on non-Linux

Infrastructure

Only mechanical work remains (support native codegen!)

Code Association

Roll CCs, HPC and our approach into a consistent whole

Visualization

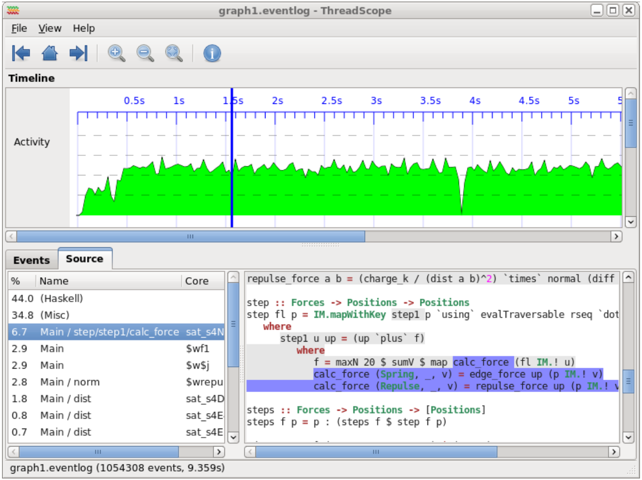
A lot of data available, analysis still relatively crude.

Thanks for listening ... Discussion?

Another Optimization Problem

True story!

UNIVERSITY OF LEEDS



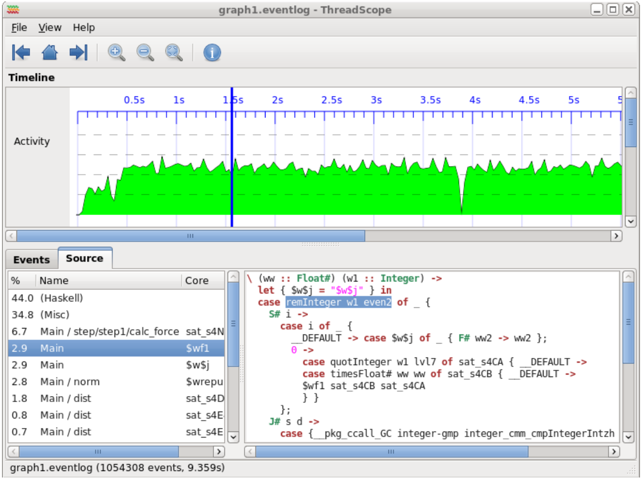
Uh, only 6.7% activity in worker!

Hm, “$wf1” and “$w$j” look suspicious. . .

Further Investigation

Strange enough that I first suspected a bug…

UNIVERSITY OF LEEDS

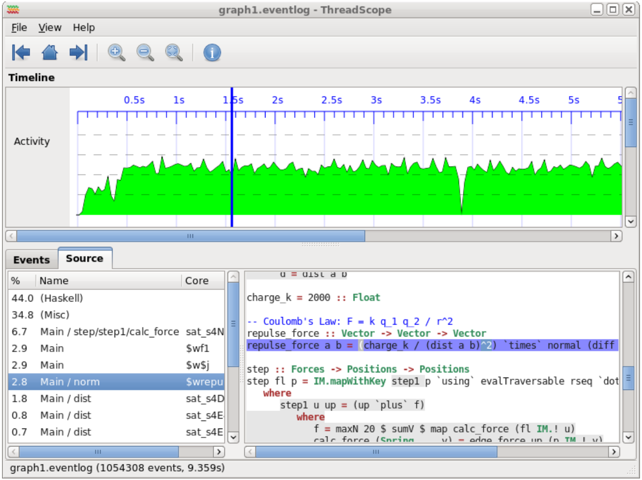


Integer arithmetic, of all things?

The program is only dealing with `Floats`!

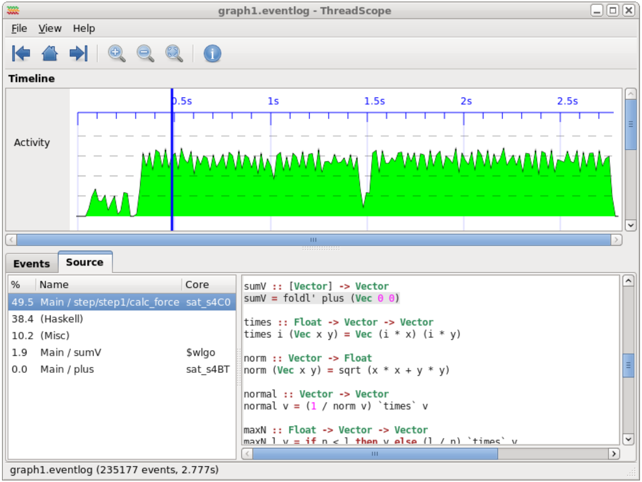
The Unexpected Villian

Small operator, large effect



Looking at the call site: Exponentiation was to blame!

(2 :: `Integer` by defaulting, (^) implemented as loop, see #5237)



Final speedup: Over 3 fold!

This page was intentionally left blank.