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
ThreadScope
What this will be about

ThreadScope Work-Flow

app.hs

GHC

app.hs -threaded -eventlog

app[.exe]

+RTS -Is -RTS

app.eventlog

ThreadScope

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
The Problem
What is happening?

Speedup low for 4 cores... What is the reason?
Main worker only active 23% of the time! Not good.
Okay, this should not happen.
Optimization Results

Much better!

A simple strictness annotation gives 3 fold speed-up.
The Goal

Timestamped source-level profiling data.

... written out:

**Accurate profiling**
- Reliable performance data
- Reflect original program well
  ⇒ 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...
Profiling
Throwing data away done right

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)
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 Performance Counters

Everything truly great must be unportable

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

Plain Timers

Use a simple timer for sampling
- Only by time — not what we want, strictly speaking
- Again unportable below $\sim 10\text{ms}$?
- 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

Peter Wortmann (Uni Leeds) Weaving Source Code into ThreadScope HaskImp’ 2011 12 / 18
Dealing with Core Optimization
... from functional to better functional

Put annotations into expression graph, update for optimisations:

```
main =
print
"Hello World!"
HelloWorld.hs:(5:0-5:27)
```

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

¹Not quite the same as [SansomJones1997], [GillRunciman2007]
Dealing with Core Optimization
... from functional to better functional

Put annotations into expression graph, update for optimisations:\(^1\):

```
main =
  print
  let a = "Hello World!"
  a
```

- Code gets separated
- Code gets (partially) removed
- Code gets integrated

\(^1\)Not quite the same as [SansomJones1997], [GillRunciman2007]
Put annotations into expression graph, update for optimisations:

- Code gets separated
- Code gets (partially) removed
- Code gets integrated

→ duplicate annotation
→ remove/move annotation
→ allow overlap?

---

1 Not quite the same as [SansomJones1997], [GillRunciman2007]
Dealing with Code Generation
... from functional to imperative

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

Generated closure code is imperative-style procedures & blocks

Main =
print
HelloWorld.hs:(5:0-5:27)
a
CmmProc
A
CmmProc
B
A ⊆ { HelloWorld.hs:(5:0-5:27)
, <main = print a> }
B ⊆ { HelloWorld.hs:(5:0-5:27)
, <a = "Hello World"> }

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

The New Workflow

- app.hs
- GHC
- app.exe
- app.eventlog

Source → ThreadScope → Events, Map
What to make of the data

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!

Many functions per procedure

Inlining distributes responsibility
⇒ Mark all or use heuristic
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!

Uh, only 6.7% activity in worker!
Hm, “$wf1” and “$w$j” look suspicious...
Further Investigation
Strange enough that I first suspected a bug...

Integer arithmetic, of all things?
The program is only dealing with Floats!
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.