Spark visualisation in ThreadScope

Duncan Coutts, Mikolaj Konarski,
Andres Löh, Nicolas Wu

Haskell Implementors' Workshop 2011

Point of this talk

About profiling of parallel Haskell programs

> *"I parallelised my program but I'm not getting the speedup I expected.*
>
> *Why not? What is going on!?"*

Point of this talk

About profiling of parallel Haskell programs

> *"I parallelised my program but I'm not getting the speedup I expected.*
>
> *Why not? What is going on!?"*

- We'll briefly go over basic parallel profiling with ThreadScope
- Main point is the new 'par spark' profiling

Context

GHC family of profiling / tracing / debugging systems

- time profiling
- heap profiling
- event tracing
- HPC tracing
- GHCi debugger

Only event tracing and HPC work for multi-core programs

Event tracing

Traces runtime events, including

- Haskell forkIO threads starting/stopping
- Garbage collector start/stop
- `traceEvent :: String -> IO ()`
- various other instantaneous and information events

GHC RTS dumps events to a log file

- low runtime overhead

Eventlog and ghc-events

Eventlog file format

- binary format
- extensible with new events
- also used by Mercury and Eden

ghc-events library used for reading `.eventlog` files

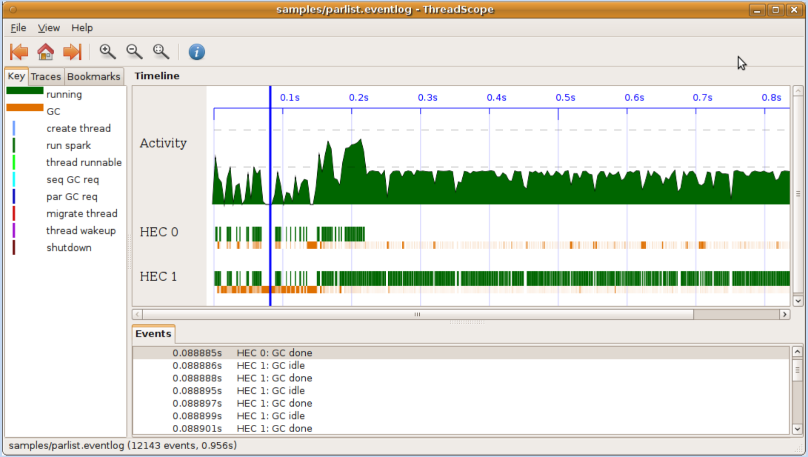
- used by ThreadScope

`ghc-events` tool with commands for

- showing event log contents
- merging event logs

ThreadScope

Viewer for `.eventlog` files



```
0.088885s    HEC 0: GC done
0.088886s    HEC 1: GC idle
0.088888s    HEC 1: GC done
0.088895s    HEC 1: GC idle
0.088897s    HEC 1: GC done
0.088899s    HEC 1: GC idle
0.088901s    HEC 1: GC done
```

samples/parlist.eventlog (12143 events, 0.956s)

Single-core uses

Limited number of single-threaded uses

- GC visualisation
- comparative tracing using `ghc-events merge`

Future potential

- could integrate time and heap profiling
- operating system events
  - useful for I/O server style apps
- distributed use cases via merging
  - time sync is tricky

Compiling, running and viewing

Compile your program

```
ghc parprog.hs -O -threaded -eventlog -rtsopts
```

Run your program

```
./parprog +RTS -N2 -ls -RTS ...
```

View the eventlog

```
threadscope parprog.eventlog
```

Eventlog generation options

In GHC 6.12–7.2

`-l`		(none)
	`s`	scheduler

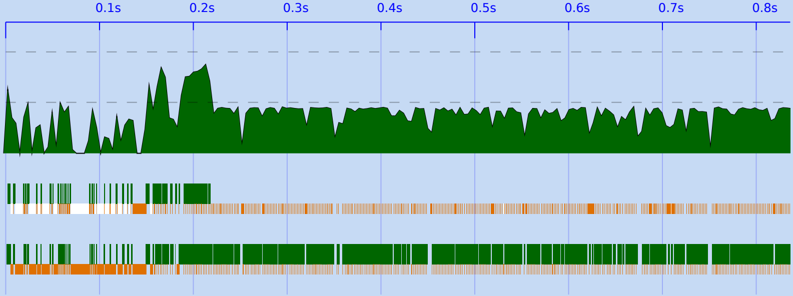
```
./parprog +RTS -ls
```

In GHC 7.4+

`-l`		(defaults)
	`s`	scheduler
	`g`	GC
	`p`	par sparks (sampled)
	`f`	par sparks (fully detailed)
	`a`	all of the above
	`-x`	remove class `x`

```
./parprog +RTS -l-ag
```

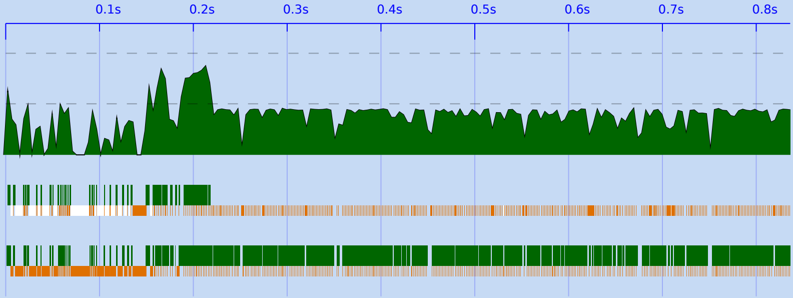
Activity plots: what they display



The activity plot shows

- combined mutator CPU usage
- runtime activity for each core
    - mutator and GC separately

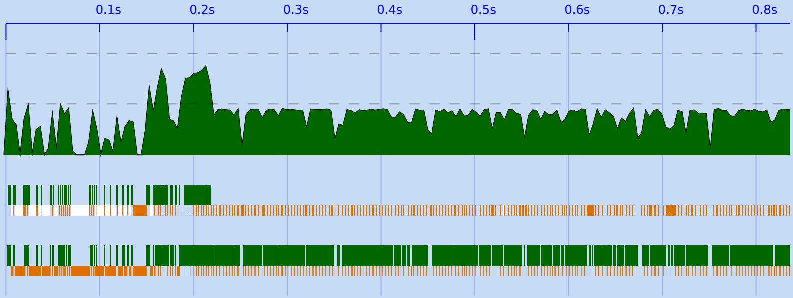
Activity plots: what we can see



The kinds of runtime behaviours we can see

- that we’re not hitting full N-CPU usage
- that one core is doing all the work
- work is badly distributed
- if there’s lots of ‘stutter’
- the interruption effect of GC, major & minor

Activity plots: what we can see



Cannot generally see why we see the behaviours we do

- sometimes seeing behaviour is enough of a hint
- can experiment, tweak and compare runs
  but basically intuition and trial and error
- at worst, just a more detailed `./parprog +RTS -s`
  i.e. wall clock, mutator and GC times

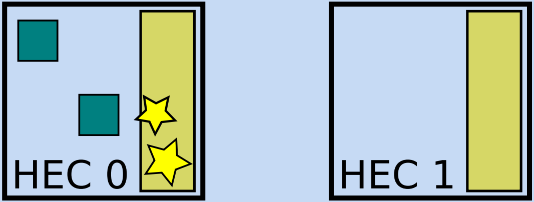
Spark profiling

Spark profiles

- an attempt to see why, not just what
- for class of parallel programs using ‘par sparks’
  - including libs built on top like strategies
- does not cover `forkIO`
  - including `Par` monad

Idea is to visualise information that is meaningful to the parallel paradigm the program is using

Par spark evaluation model

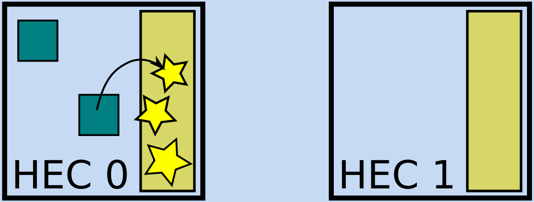


- per-core task queue

Terminology:

- a task is called a ‘spark’
- a task queue is called a ‘spark pool’

Par spark evaluation model

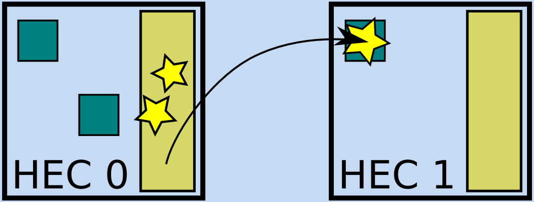


- per-core task queue
- tasks created using `par`

Terminology:

- a task is called a ‘spark’
- a task queue is called a ‘spark pool’

Par spark evaluation model



- per-core task queue
- tasks created using `` `par` ``
- tasks run on any available core

Terminology:

- a task is called a ‘spark’
- a task queue is called a ‘spark pool’
- sparks get ‘converted’, meaning evaluated

Spark life cycle

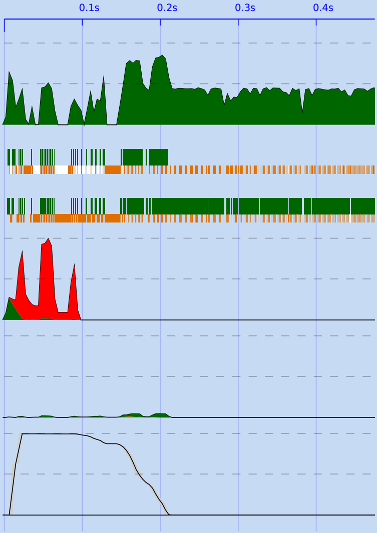
Sparks created with `` `par` ``

- but spark could be ‘dud’
- but spark pool could be full

Sparks get ‘converted’

- but could have already been evaluated
  - now points to a WHNF
- or could have been GC’d
  - note: sparks are not GC roots

Interpreting spark graphs

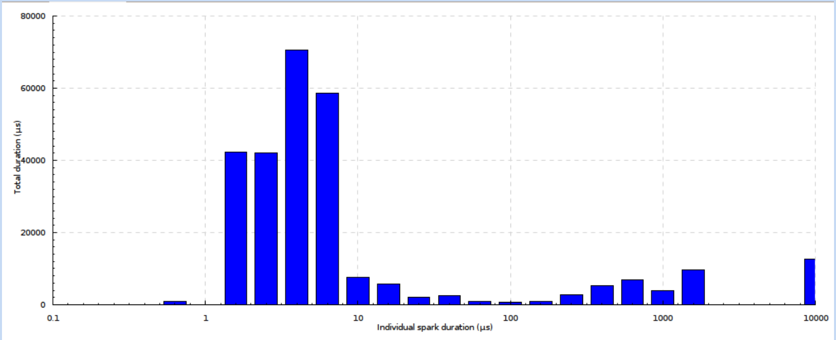


- number of sparks created (per unit time)
  - area is total number of sparks
  - green for created
  - red for overflow
  - grey for dud
- number of sparks converted (per unit time)
  - area is total number of sparks
  - green for converted
  - grey for fizzled
  - orange for GC'd
- graph of size of spark pool

Well-Typed

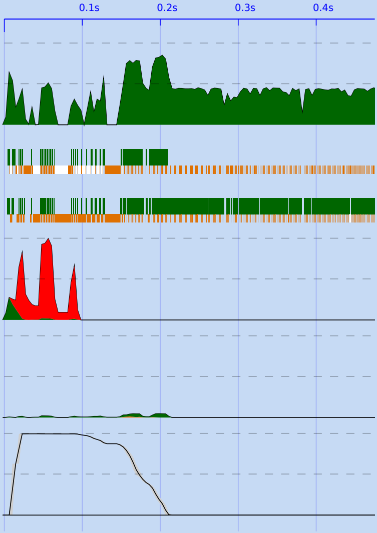
Interpreting spark graphs

Distribution of spark sizes



- total evaluation time of sparks of various sizes
- histogram bucket divisions on log scale

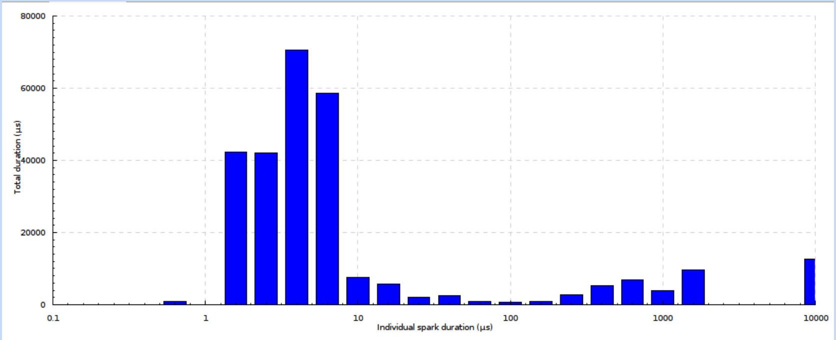
Diagnosing spark problems



Problems can we diagnose

- too few sparks
  (not enough parallelism)
  - spark pool hits empty
  - low spark creation rate
- too many sparks
  - overflow is wasted work
  - can cause catastrophic loss of parallelism

Diagnosing spark problems



More problems can we diagnose

- sparks too small
  - overheads too high
- sparks too big
  - load balancing problems

Diagnosing spark problems

More subtle cases

- too many dud sparks
- too many sparks that fizzle
- too many sparks that get GC'd

Demo

Demo!

Implementation: spark events

Sampled spark events

- RTS maintains counters of number of sparks created / converted / dud . . .
- occasionally log an event with current counters + spark pool size
- low overhead
- enough for creation / conversion / pool size graphs

Full spark events

- log an event for every spark created / converted . . .
- higher overhead
- needed to calculate spark sizes

Implementation: calculating spark size

Using full spark events we can calculate how long it takes to evaluate each spark

- sparks are evaluated by special threads
- can see when each spark is picked up
- can see when spark thread is running
- nice implementation using state machine

Future work

Future work we intend to do (more or less)

- polishing spark visualisation
    - scaling and rescaling of graphs
    - lots of little TODOs
- breakdown of spark graphs by strategy
    - labelling sparks with the strategy that generated them
- events from Haskell library code (not just RTS)
    - needed to do profiling at the level of library abstractions, e.g. Par monad
- operating system info via Linux ‘perf’ tracing system
    - to find out the reasons for blocking, e.g. syscalls and being descheduled

Feedback requested

ThreadScope is available now from hackage

- GHC HEAD (version 7.3+) needed for spark events
- `gtk-0.12.1` package works on Linux, Windows, OSX and with all recent GHC versions

We want feedback from ThreadScope users

- what is helpful, unhelpful, missing?

Feedback requested

ThreadScope is available now from hackage

- GHC HEAD (version 7.3+) needed for spark events
- `gtk-0.12.1` package works on Linux, Windows, OSX and with all recent GHC versions

We want feedback from ThreadScope users

- what is helpful, unhelpful, missing?

That's it!

Questions?