GHC on the OpenSPARC T2

Ben Lippmeier
Australian National University
FP-SYD 2009/08/20

A Slow Program

```
result :: [Integer]
result = map (ack 2) [1..200]


ack :: Integer -> Integer -> Integer
ack 0 n = n + 1
ack n 0 = ack (n-1) 1
ack n m = ack (n-1) (ack n (m - 1))
```

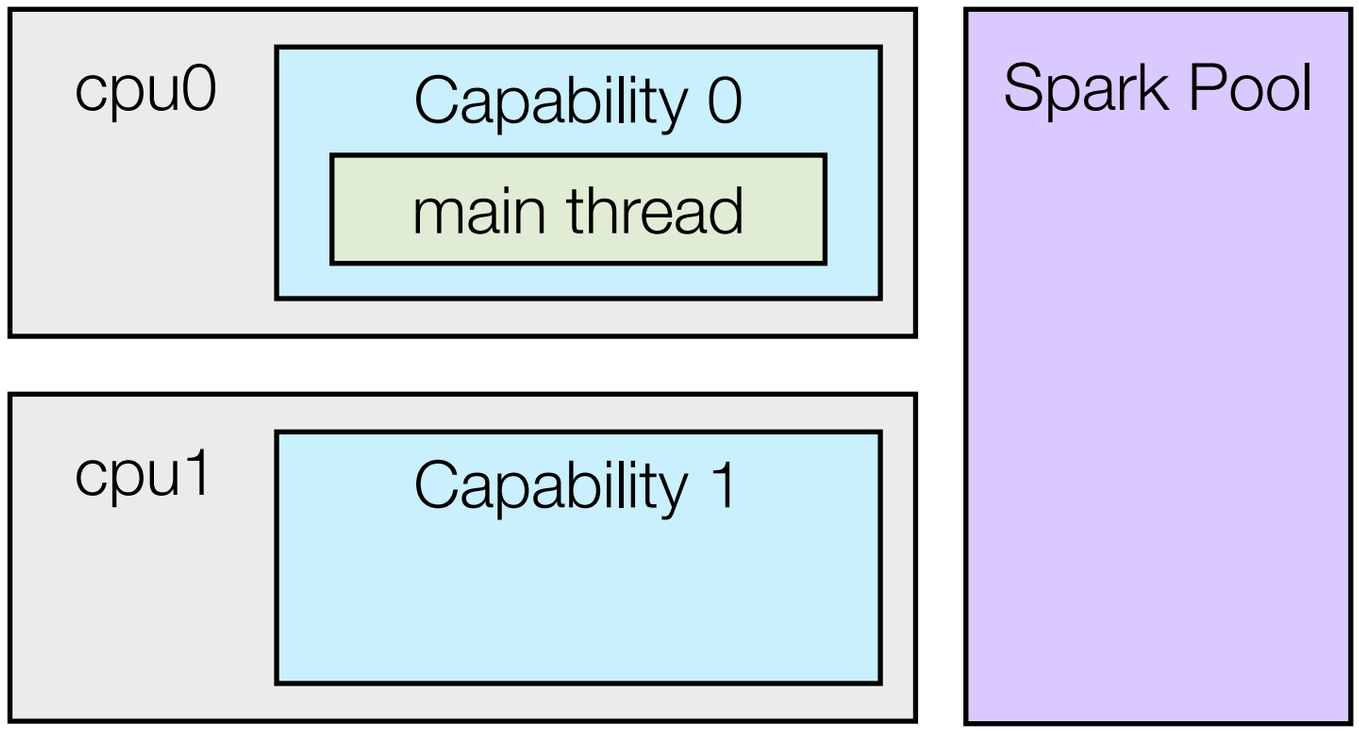
Algorithm + Strategy = Parallelism

```
import Control.Parallel.Strategies

result :: [Integer]
result = map (ack 2) [1..200]
                `using` parList rwhnf

ack :: Integer -> Integer -> Integer
ack 0 n = n + 1
ack n 0 = ack (n-1) 1
ack n m = ack (n-1) (ack n (m - 1))
```

- Add parallel combinators => program runs faster.

- All locking / scheduling / work balancing done by RTS.

GHC parallel evaluation model

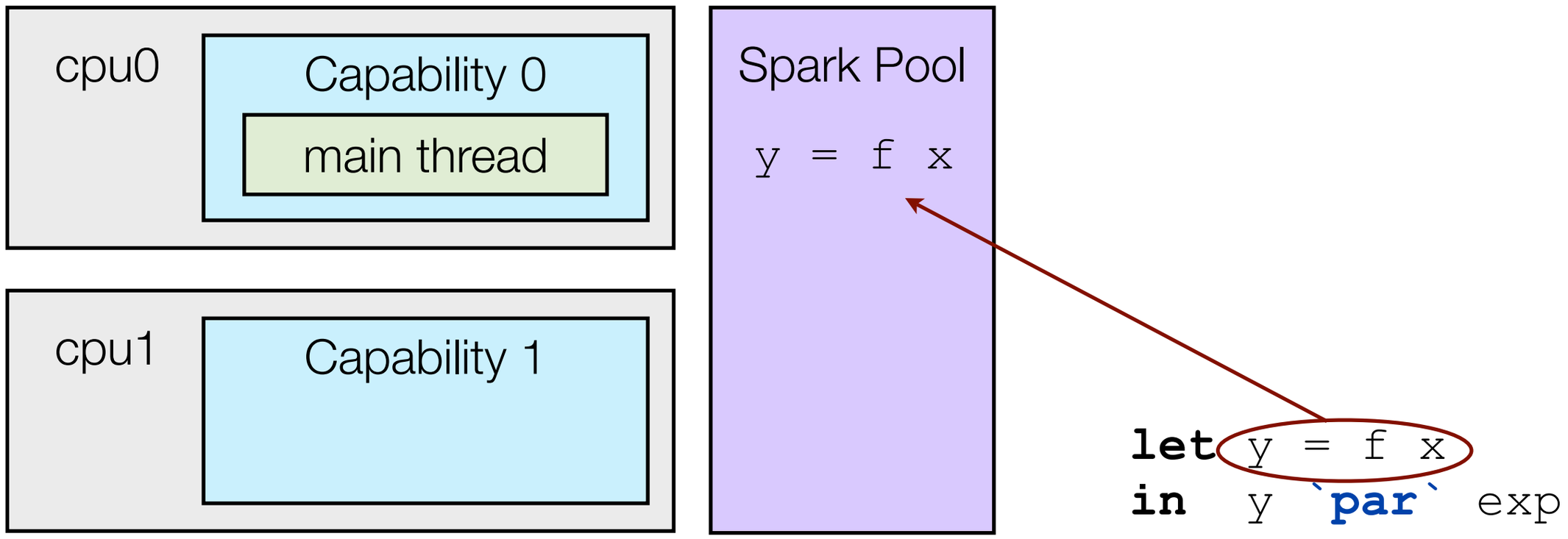



```
let y = f x
in  y `par` exp
```

- A capability is a thread of the GHC runtime system that can evaluate parts of the program. There is one capability per CPU/hardware thread.

- Capability 0 holds the main thread, which evaluates the main routine.

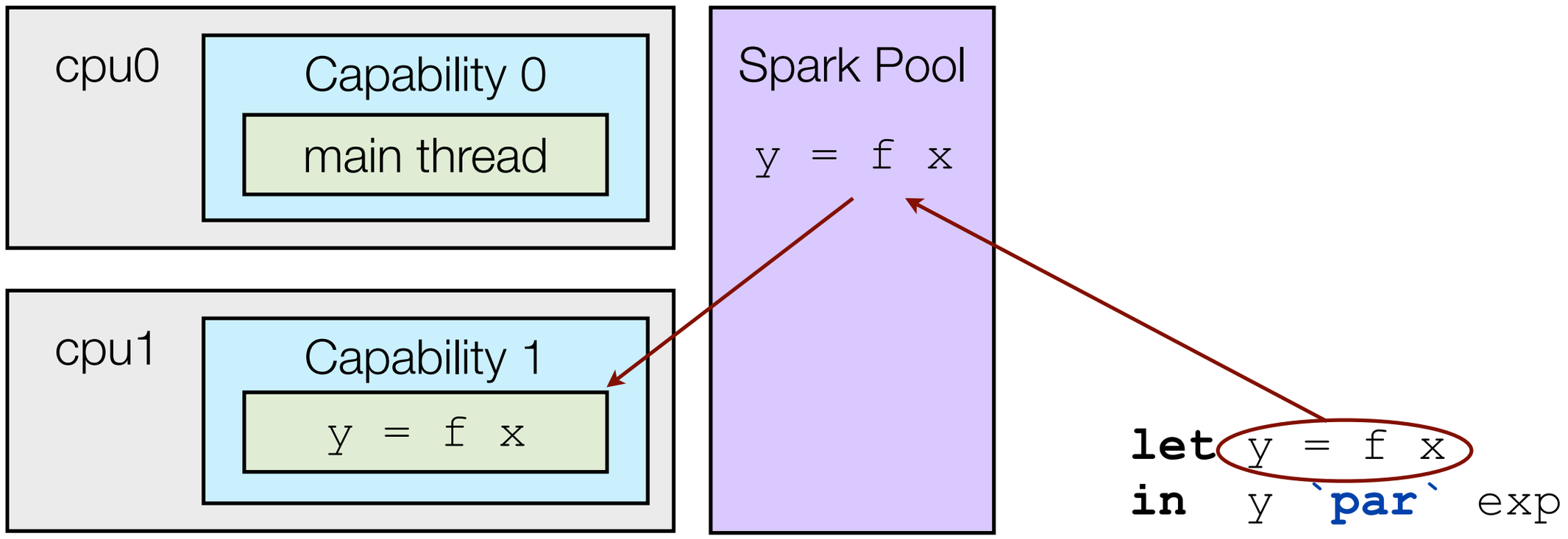
GHC parallel evaluation model



- A spark represents an expression in the program that could be potentially evaluated in parallel.

- Sparks are created with the primitive Haskell operator `par`

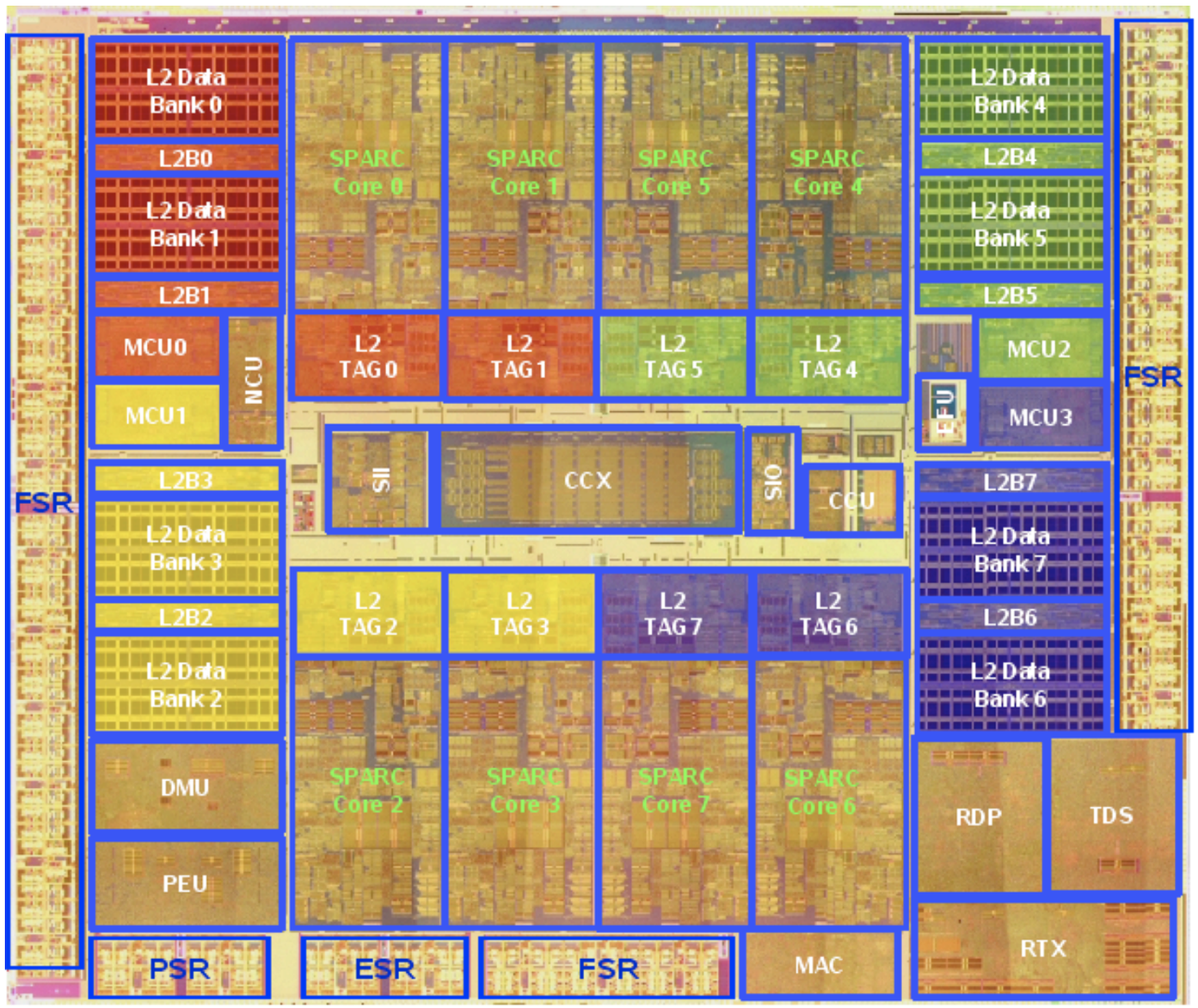
GHC parallel evaluation model



- When a Capability is idle, a spark is taken from the spark pool and made into a running thread.

- All *we* need to do is to create sparks, and specify the number of capabilities to use. Great for irregular parallelism!

The OpenSPARC T2: Released October 2007



- 8 cores / processor.
  8 threads / core.
  = 64 threads / processor

- Hardware per core:
  + 2 ALUs
  + 1 Load/Store Unit
  + 1 FP Unit

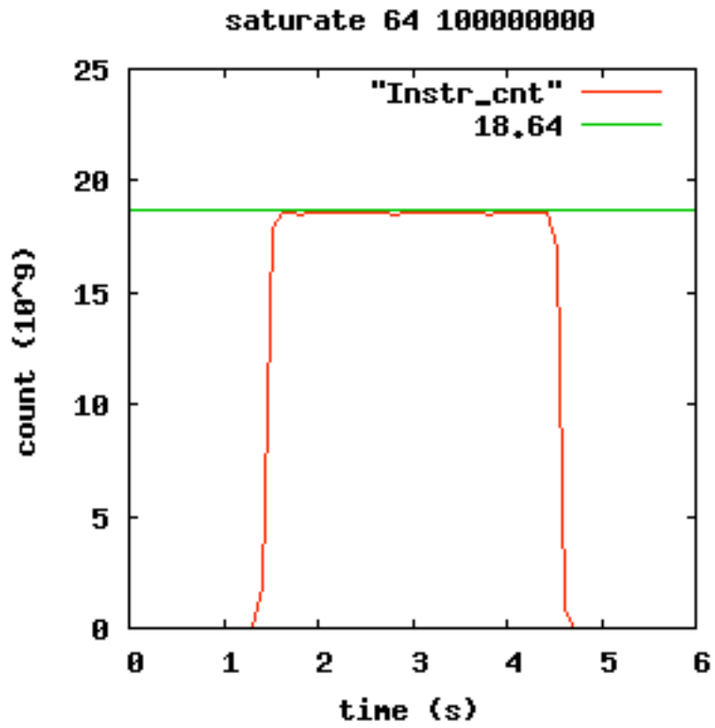
- In each cycle a core can dispatch 2 instructions.

- Threads on the same core share the same L1 cache.

OpenSPARC T2 peak issue rate (in order)

- Peak instruction issue rate:

  1165 Meg cycles / sec
  * 2 instructions / core
  * 8 cores / processor
  = 18.64 Gig Instr/s



Intel Core2 Duo peak issue rate (out of order)

- Peak instruction issue rate:

  1600 Meg cycles / sec
  * 4 instructions / core
  * 2 cores / processor
  = 12.80 Gig Instr/s

Out-of-order execution doesn't help us much...

```
sethi   %hi(s1p9_info), %g1
or      %g1,        %lo(s1p9_info), %g1
st      %g1,        [%i3-24]
ld      [%i0+8],    %g1
st      %g1,        [%i3-16]
ld      [%i0+4],    %g1
st      %g1,        [%i3-12]
st      %l2,        [%i3-8]
ld      [%i0+12],   %g1
st      %g1,        [%i3-4]
st      %l1,        [%i3]
add     %i3, -24,   %g1
st      %g1,        [%i0+12]
ld      [%i0+8],    %l1
sethi   %hi(s1rX_info), %g1
or      %g1,        %lo(s1rX_info), %g1
st      %g1,        [%i0+8]
add     %i0, 8,     %i0
and     %l1, 3,     %g1
cmp     %g1,        0
bne     .Lc1Un
```

- Lots of memory traffic
  => Lots of cache miss
- Not much Instruction Level Parallelism (ILP)

Project: Make GHC work on the OpenSPARC T2

- Project funded by Sun Microsystems.
  - Organised by Duncan Coutts, Roman Leshchinskiy, Darryl Gove.

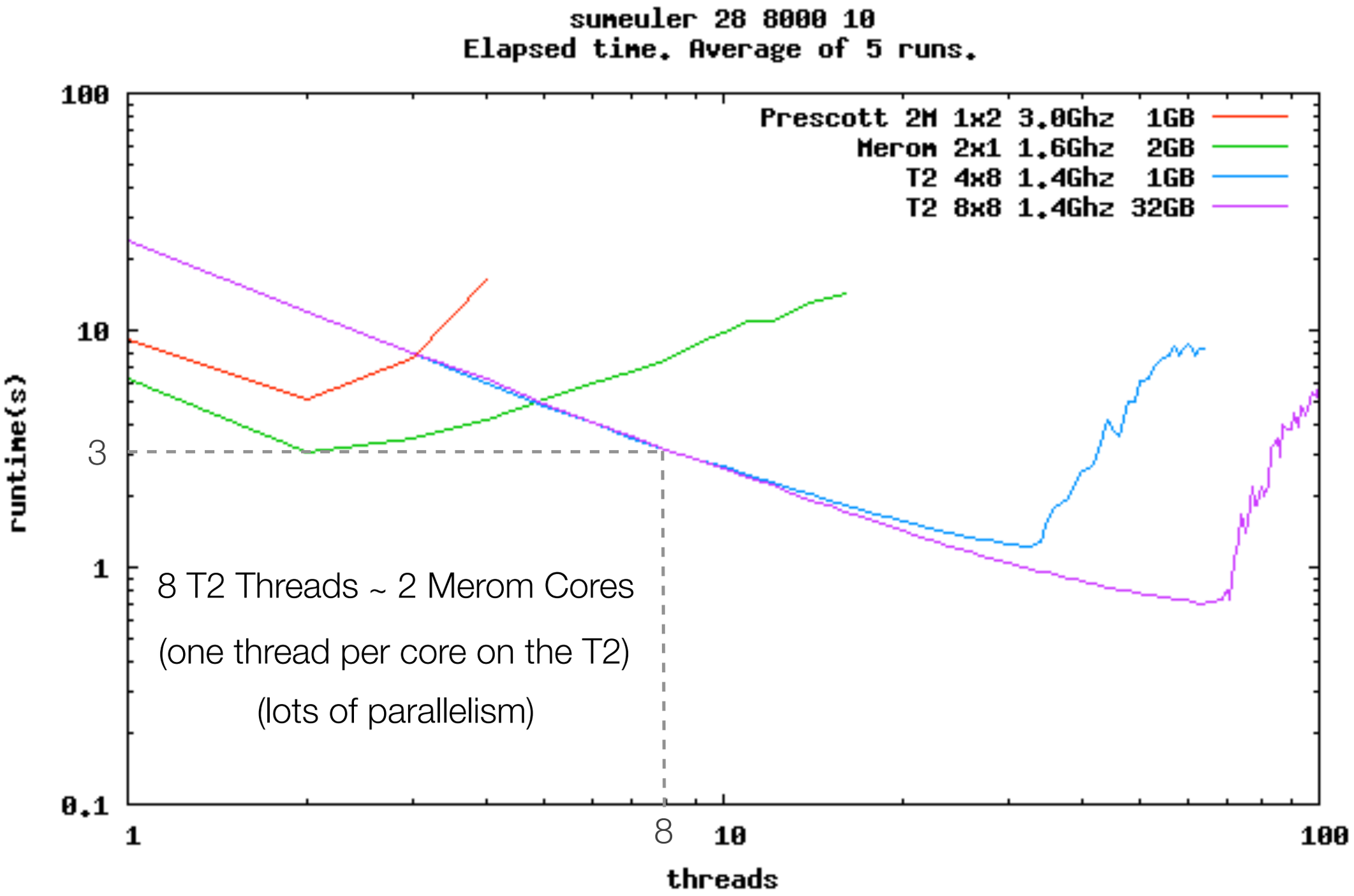
- As of 1st Jan 2009, GHC did not build at all on SPARC.

- Step1: Fix the via-C build.
  - No buildbots for SPARC.
  - Existing SPARC build was entirely community supported.

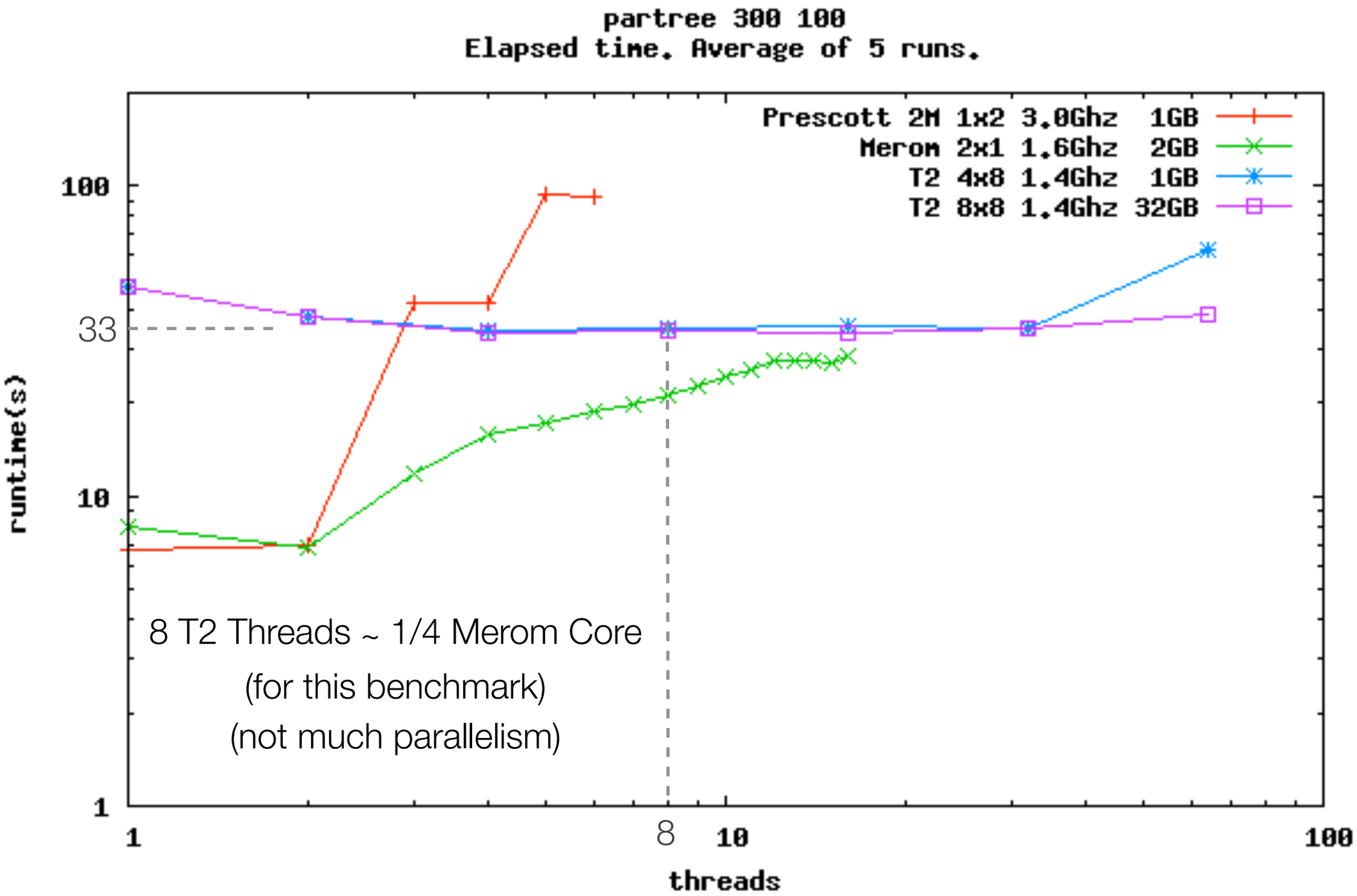
- Step2: Fix the Native Code Generator
  - SPARC NCG hadn't worked for years.
  - Badly in need of cleaning up and refactoring.

- Step 3: Benchmarking and Tuning

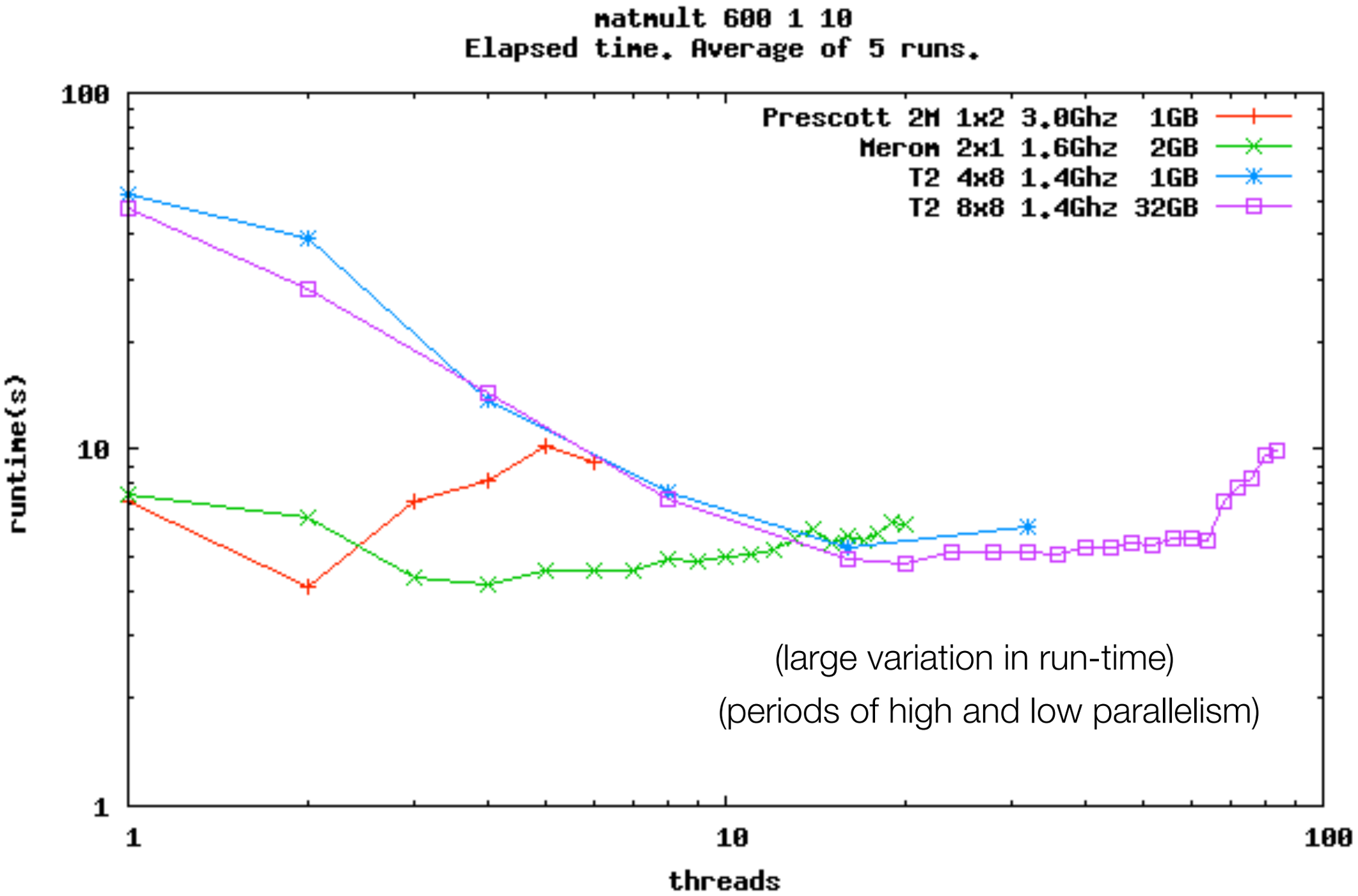
Benchmarking on the T2



Benchmarking on the T2



Benchmarking on the T2



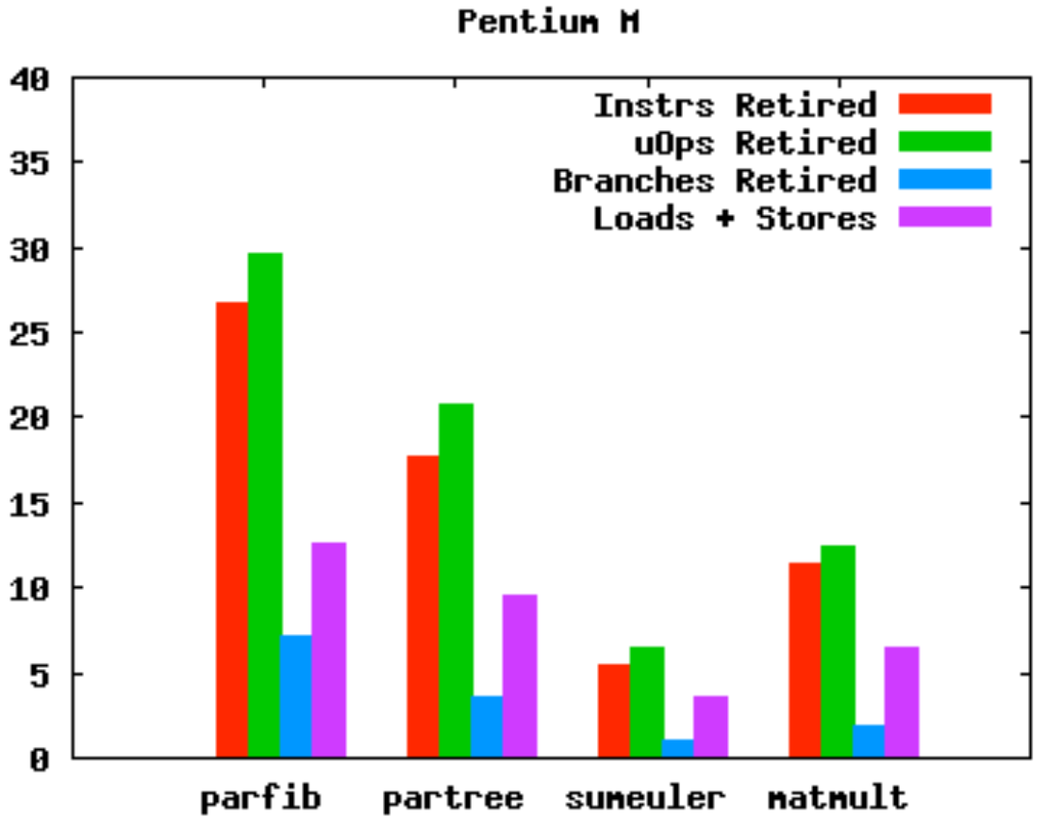
(large variation in run-time)

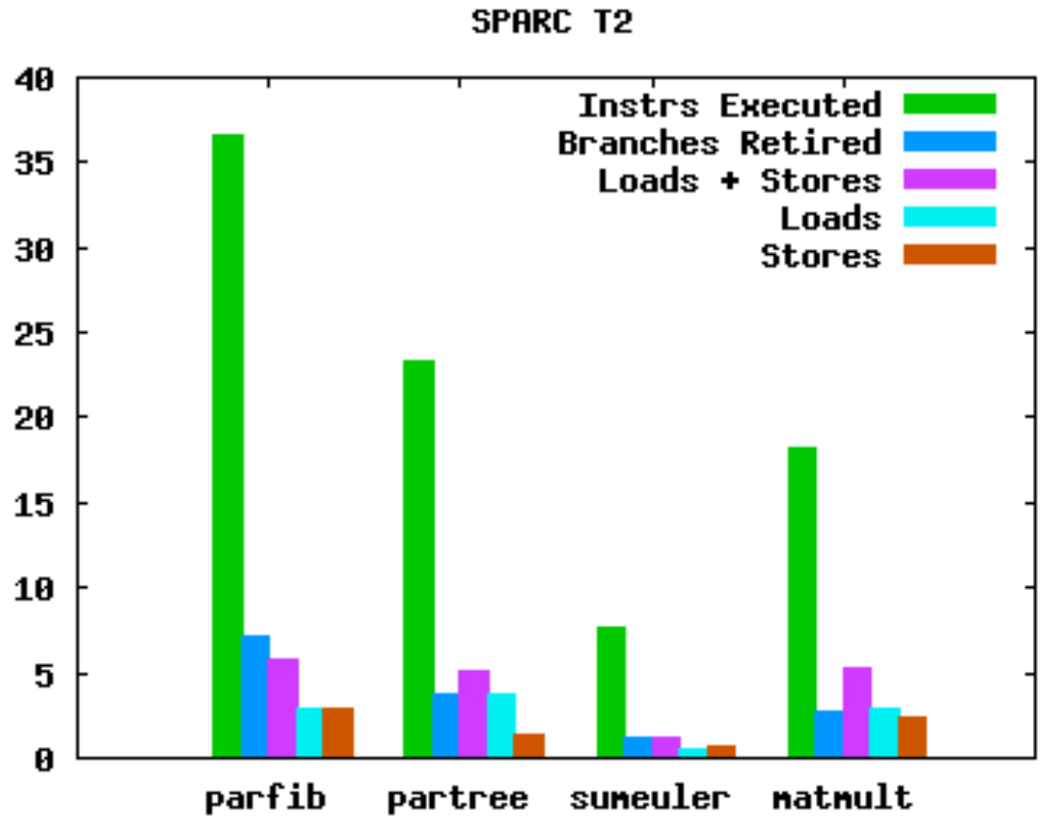
(periods of high and low parallelism)

Benchmarking Summary

If you have less than 8 threads of work,
then stay home.

Instruction counts on Pentium M *vs* SPARC T2

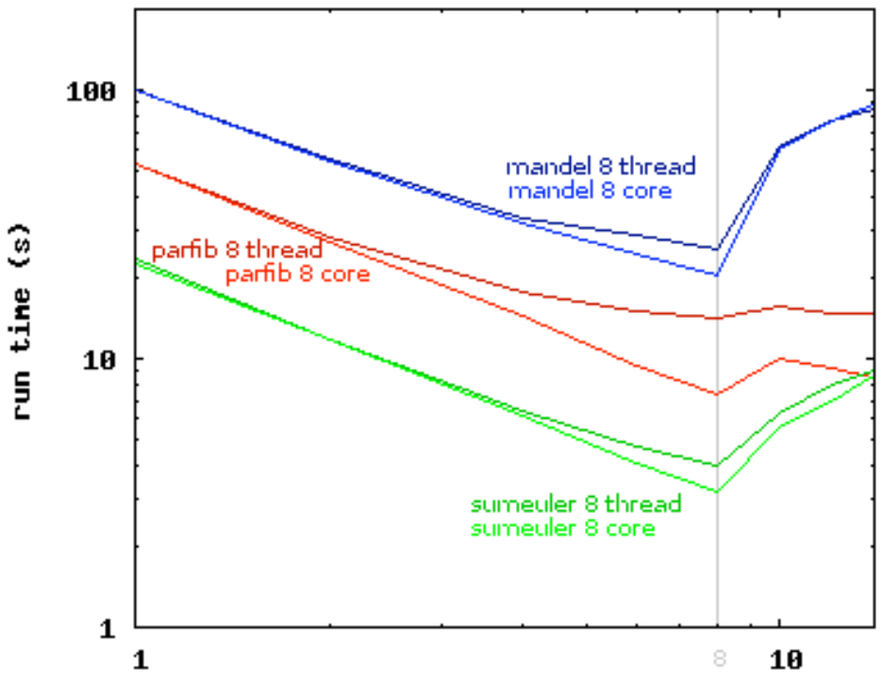




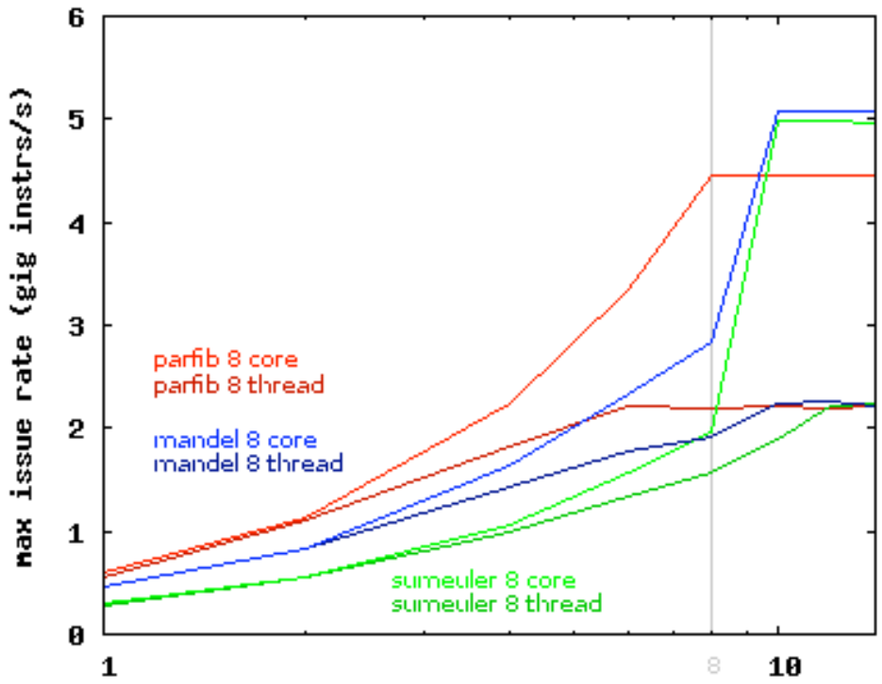

```
sethi   %hi(__stginit_base_Prelude_),%g1
or      %g1,%lo(__stginit_base_Prelude_),%g1
st      %g1,[%i0]
add     %i0,-4,%i0
sethi   %hi(__stginit_base_SystemziEnvironment_),%g1
or      %g1,%lo(__stginit_base_SystemziEnvironment_),%g1
st      %g1,[%i0]
```

Load 32bit Imm

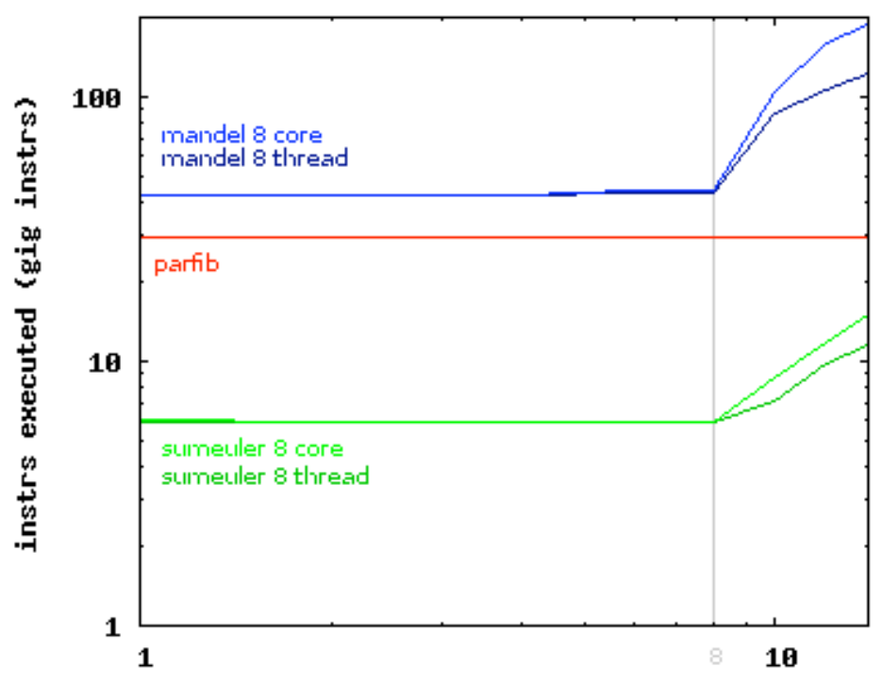
run time



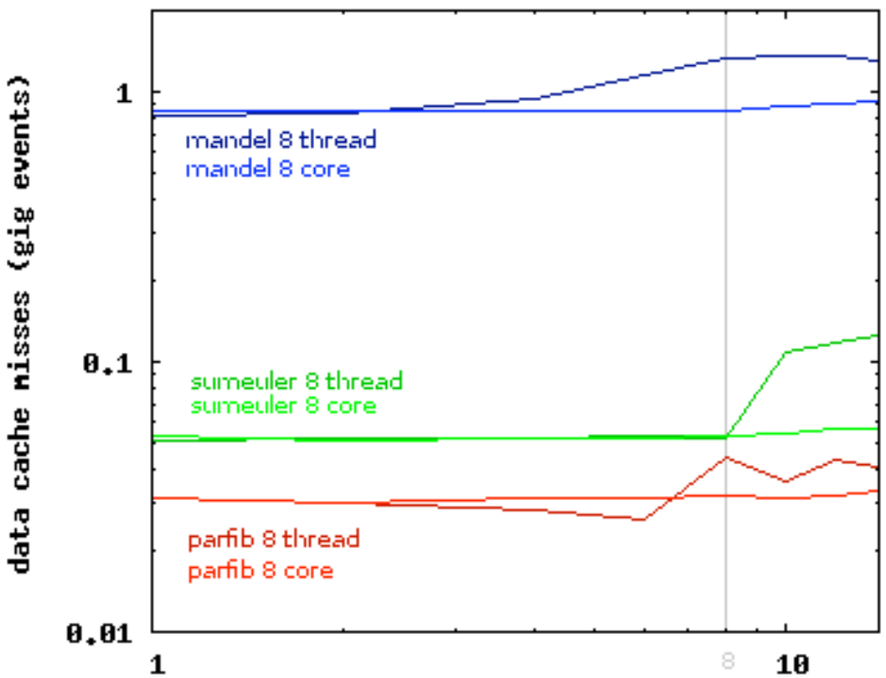
instr issue rate



total instrs executed



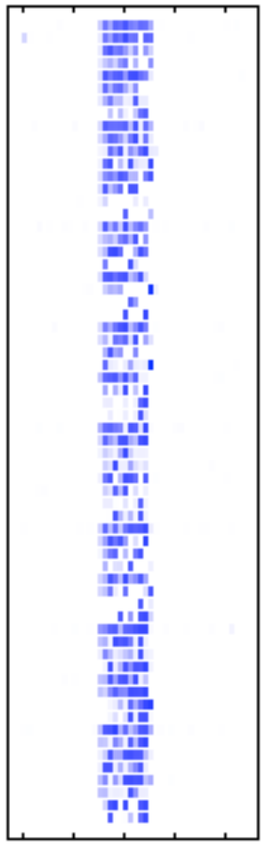
data cache misses (scaled)



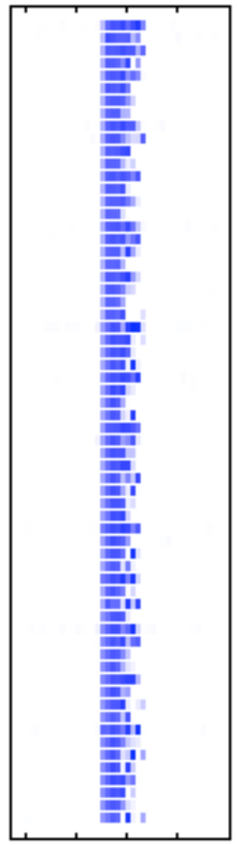
8 threads on 1 core *vs* 1 thread per core

Thread activity for sumeuler benchmark

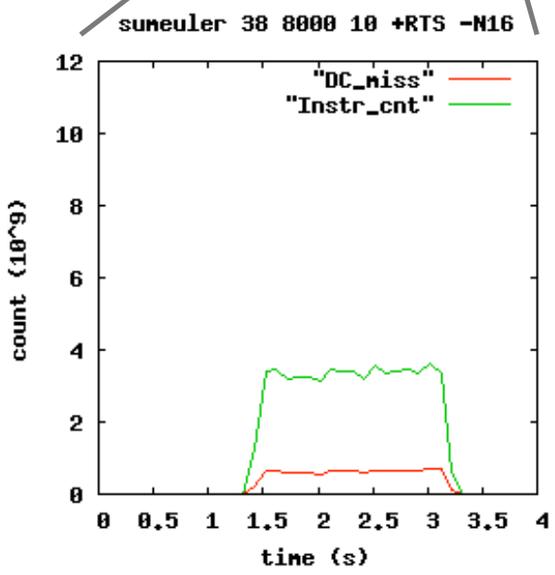
-N16

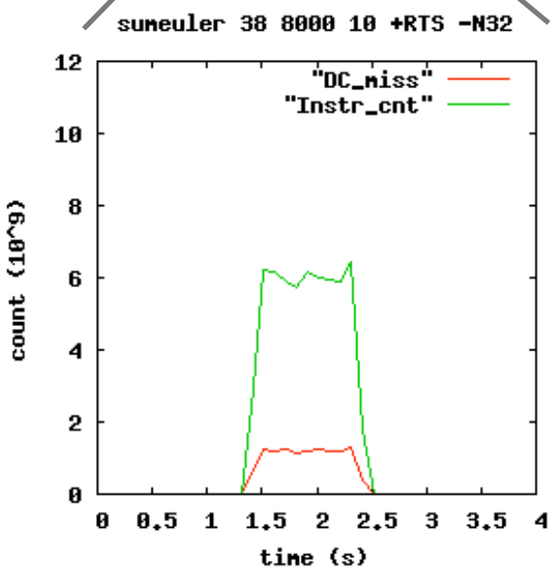


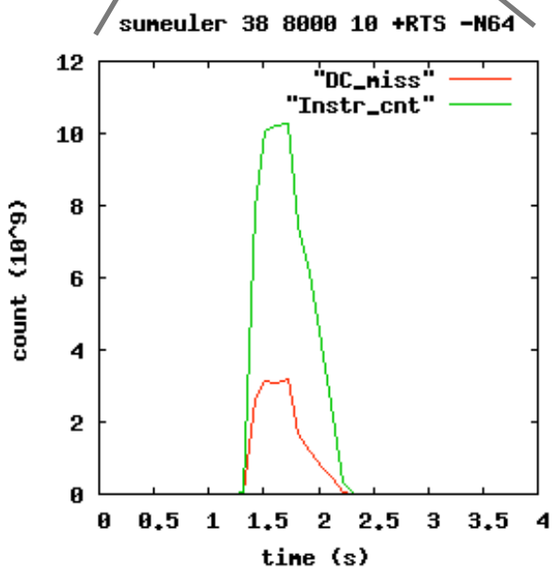
-N32



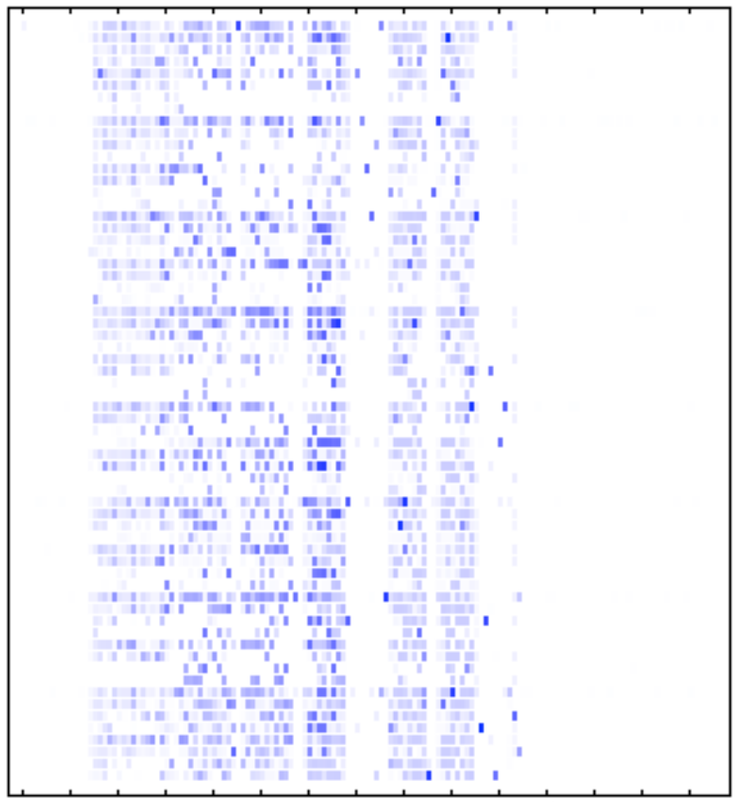
-N64



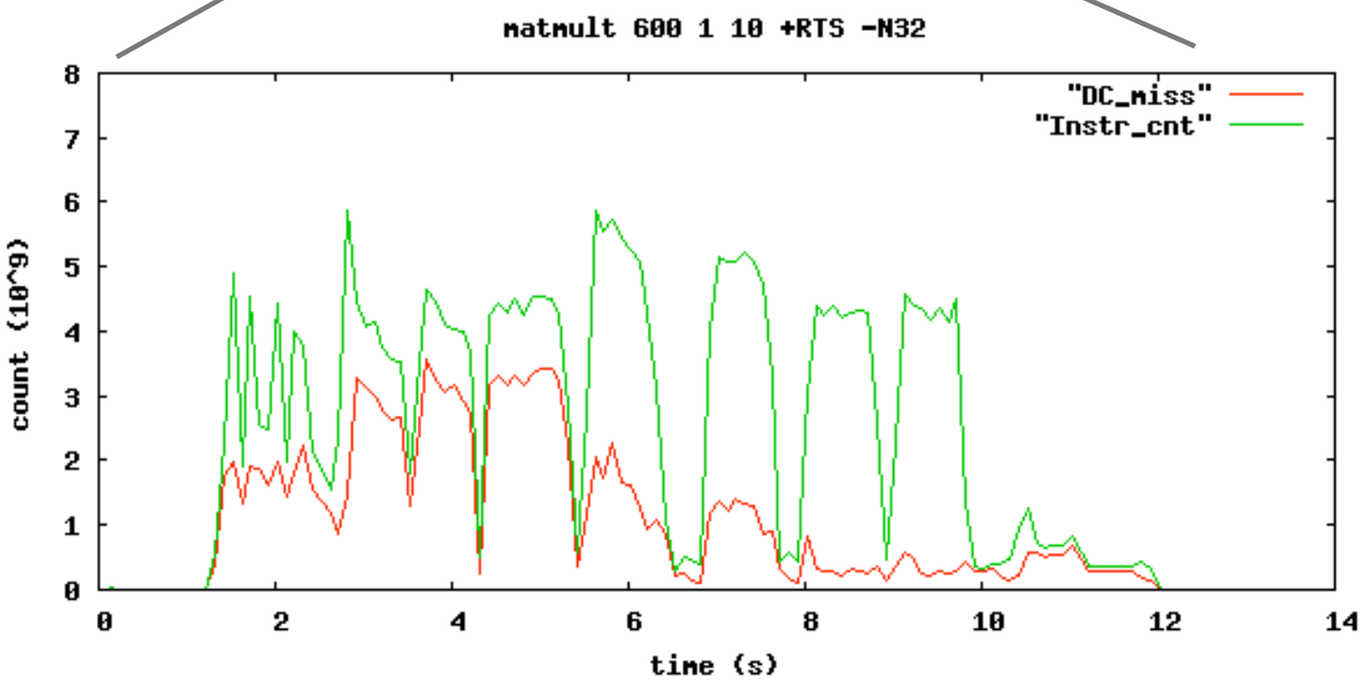




Thread activity for matmult benchmark



-N32



Distinct phases of high and low activity

Large variation run-to-run

Why??

Perhaps ThreadScope can help...

Future work

- Try to rewrite benchmarks to expose more parallelism.
  Until now we haven't been dealing with 64 hardware threads.

- Use ThreadScope to determine why we have periods of low activity in benchmarks like matmult.

- Some simple compile-time instruction reordering could help.
  The T2 core does no runtime reordering => pipeline stalls.

- Keep the build working!!