Performance Analyser
Play your game cycle by cycle

Lionel Lemarié
SCEE Technology Group

December 7th, 2002

First word

- Who am I ?

  Lionel Lemarié

  Sony Computer Entertainment Europe

  Developer Support Team

  Based in London, UK

What is this talk about ?

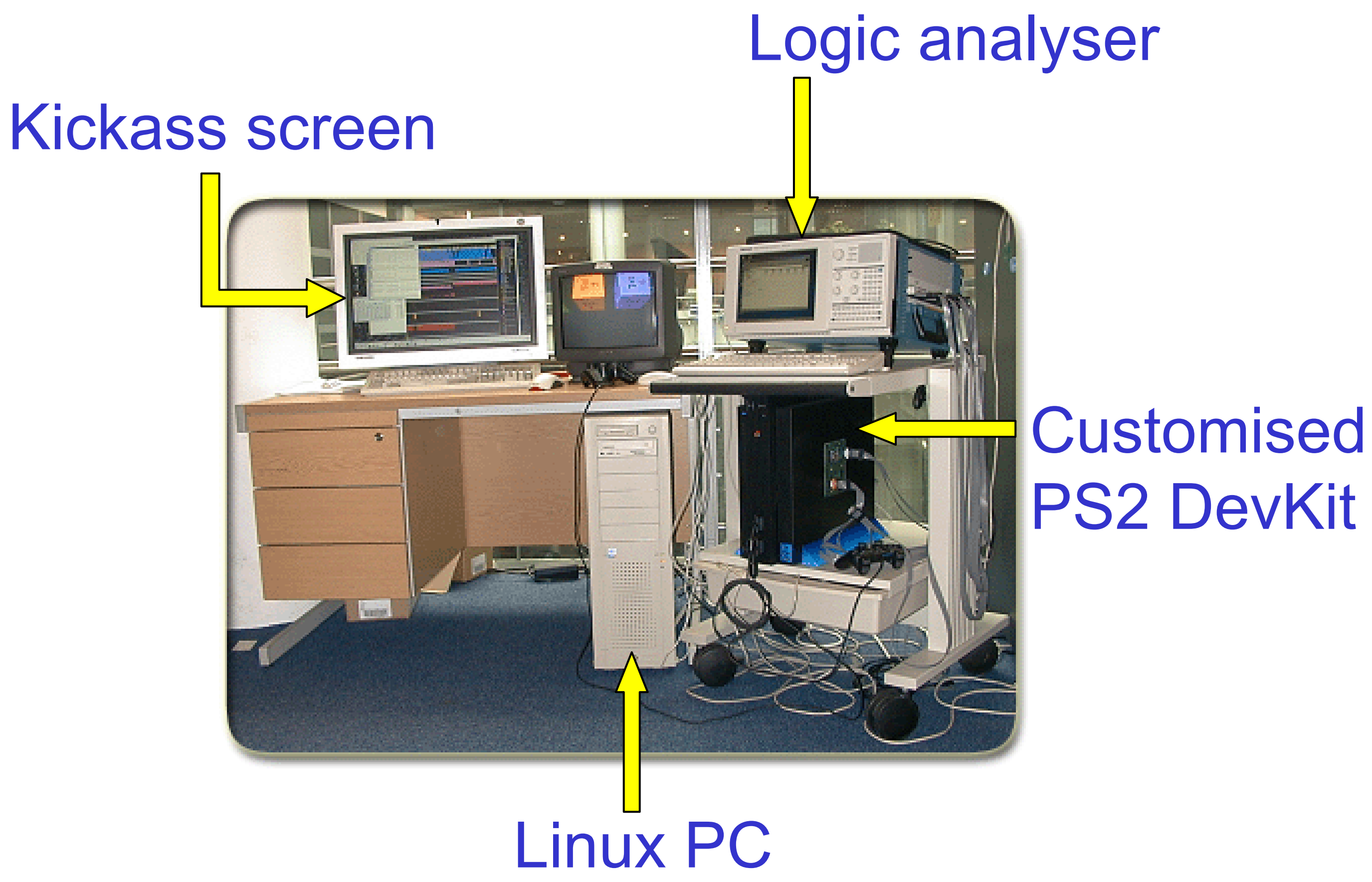
- Presenting the Performance Analyser
  - how to take advantage of this powerful benchmarking tool
- Giving a sneak preview of the upcoming kit and software
- Introducing the software
  - get the useful information quickly

Contents

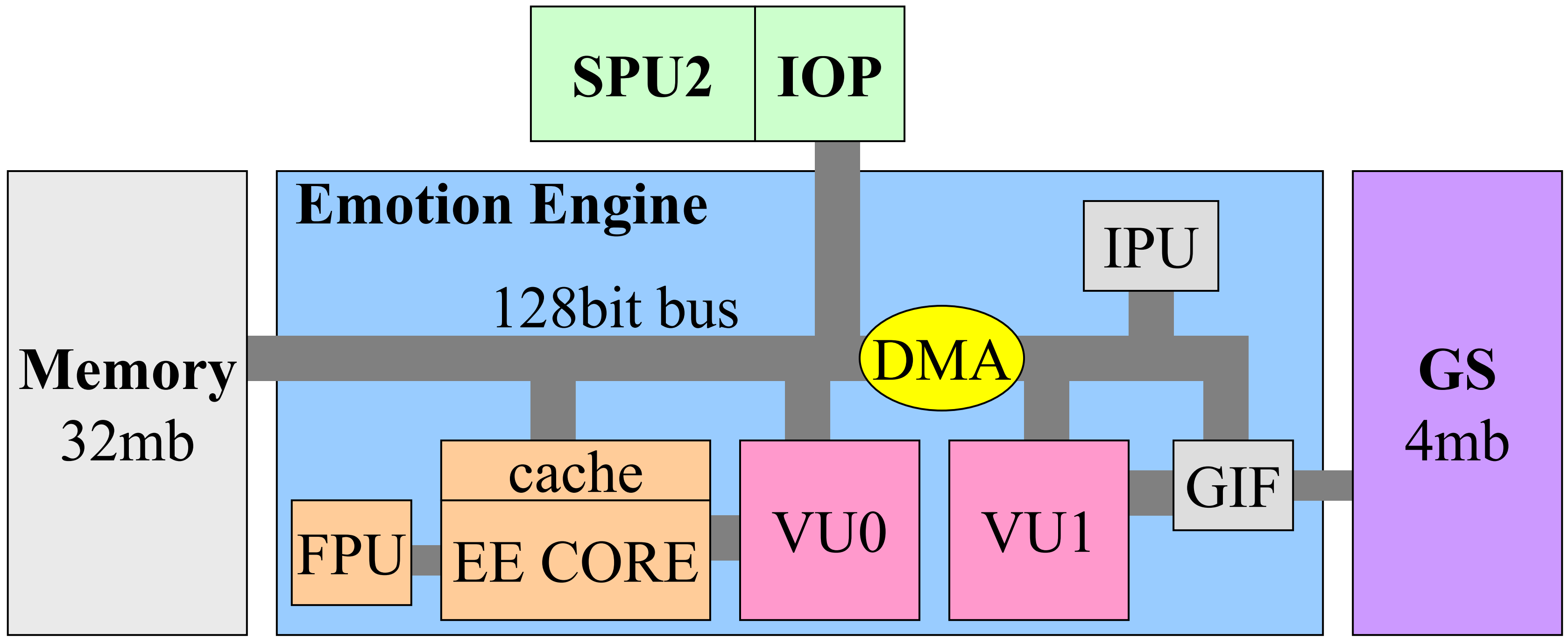
Current Performance Analyser

- Hardware - Prototype only
  - T10k
  - Logic analyser
    - grabs several frames of data
    - over a 100 signals
    - cycle accurate
  - A PC under Linux to visualise the graph

Our prototype



Reminder of PS2 architecture



It’s all about balance

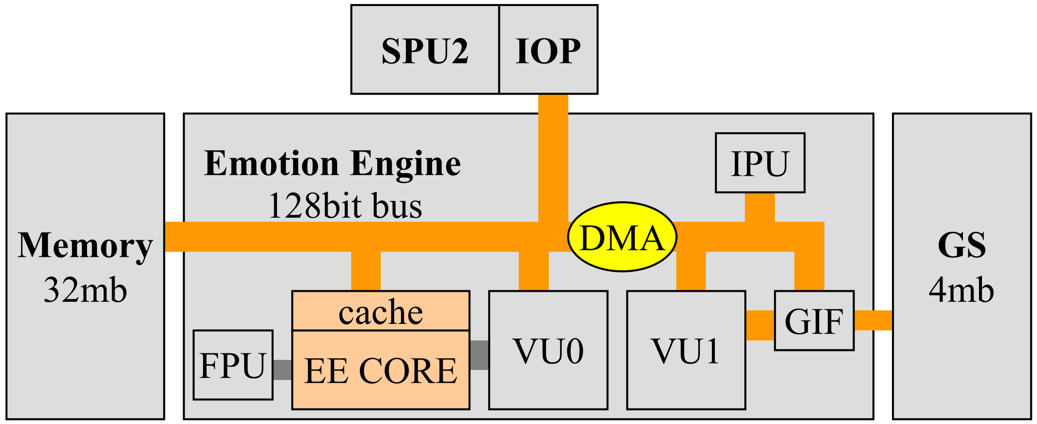
- PS2 designed for parallelism
  - EE and GS
  - CPU and DMA
    - Shared memory bus, VIF, GIF etc.
  - CPU and VUs
    - Simultaneous independent processing units
  - CPU and VU pipelines
    - 2 instructions per cycle

What the prototype doesn't do

- No data bus captured
  - internal states only
- No true address bus capture
  - most long jump captures
  - nothing else
- No IOP information at all

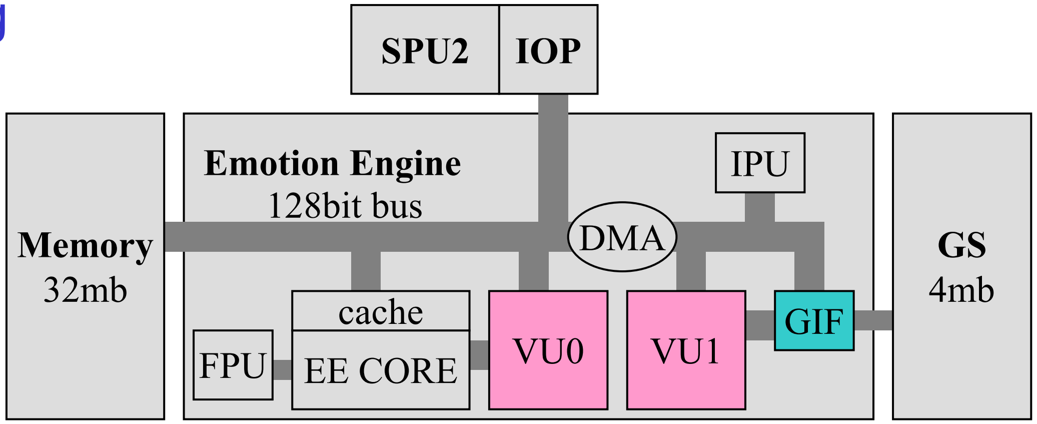
What the prototype does

- CPU signal capture
  - CPU pipeline activity
    - Single and dual issue
    - Interrupt display hack
  - Main bus
    - CPU
    - DMA



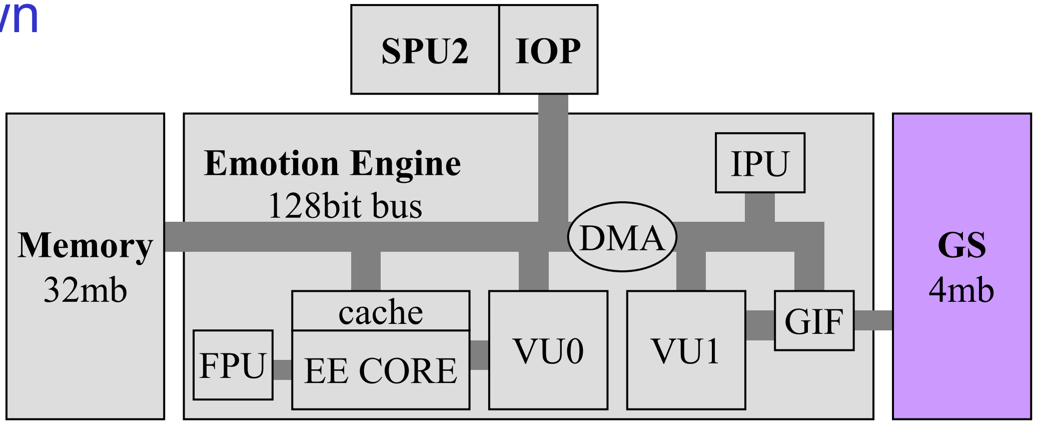
What the prototype does

- CPU signal capture
  - GIF usage
    - 3 paths
  - Vector units status
    - Running
    - XGKICK stalling



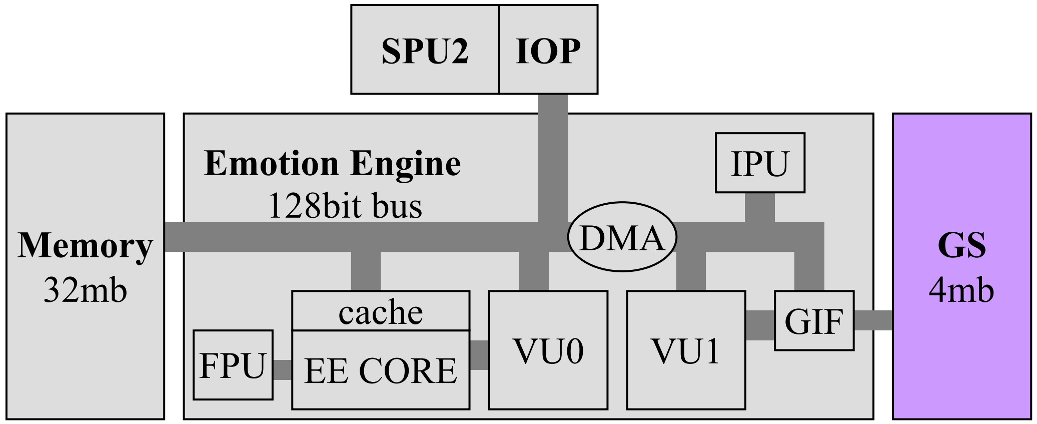
What the prototype does

- GS signal capture
  - Primitives rendered
    - Doesn't count zero-area polys
  - Pixels output
    - Half-full for textured ops
    - Only those drawn
    - Not scissored

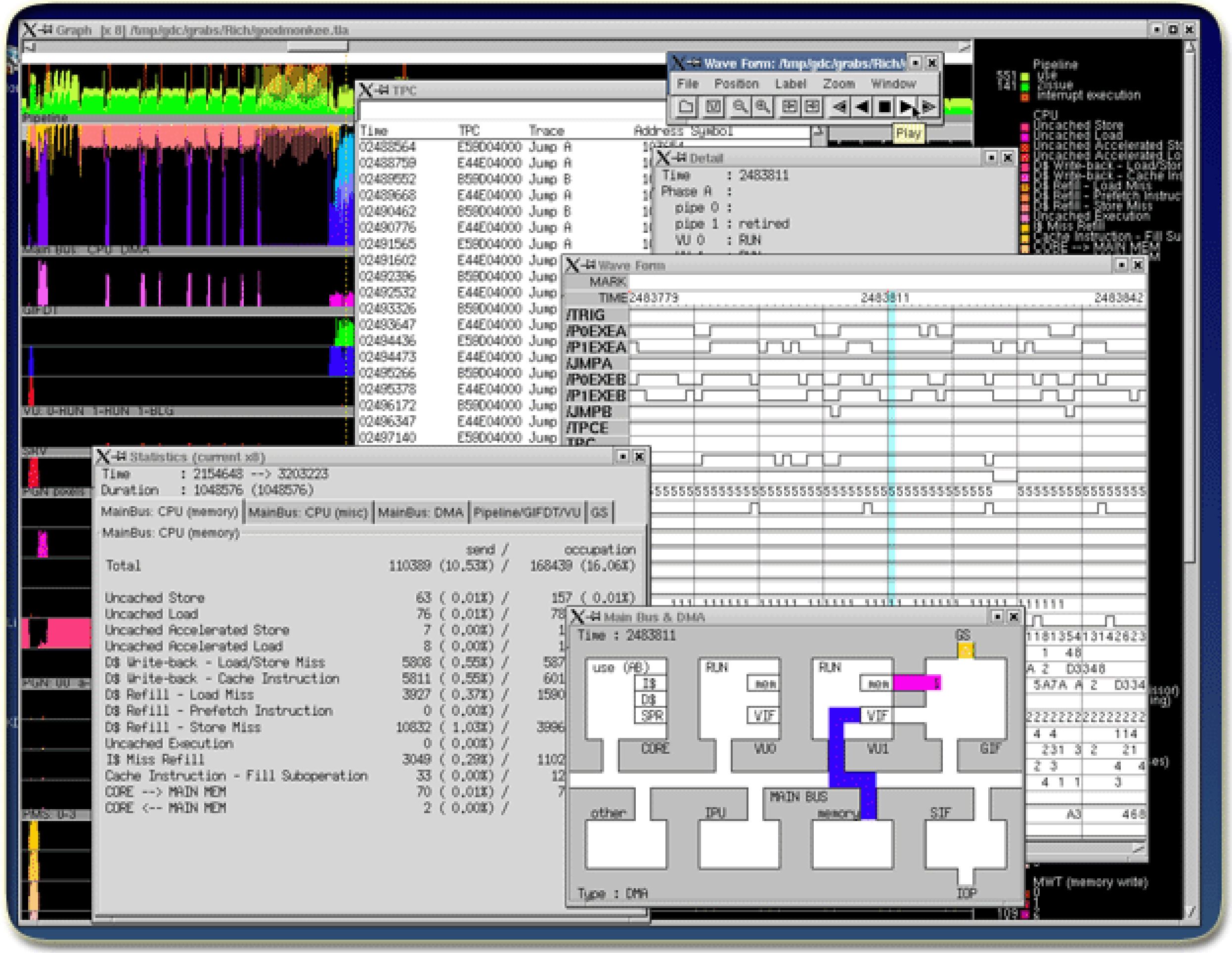


What the prototype does

- DDA void
  - Busy, but no pixels
    - Scissor, narrow poly
- Non polygonal data
- Pixel unit stalls
  - Many reasons
- GS idle time



PA software: pmon



```
TPC
Time       TPC       Trace     Address Symbol
02488564   E58D04000 Jump A
02488759   E44E04000 Jump A
02489552   B59D04000 Jump B
02489668   E44E04000 Jump A
02490462   B59D04000 Jump B
02490776   E44E04000 Jump A
02491565   E58D04000 Jump A
02491602   E44E04000 Jump
02492396   B59D04000 Jump
02492532   E44E04000 Jump
02493326   B59D04000 Jump
02493647   E44E04000 Jump
02494436   E58D04000 Jump
02494473   E44E04000 Jump
02495266   B59D04000 Jump
02495378   E44E04000 Jump
02496172   B59D04000 Jump
02496347   E44E04000 Jump
02497140   E58D04000 Jump
```

```
Detail
Time      : 2483811
Phase A   :
  pipe 0  :
  pipe 1  : retired
  VU 0    : RUN
```

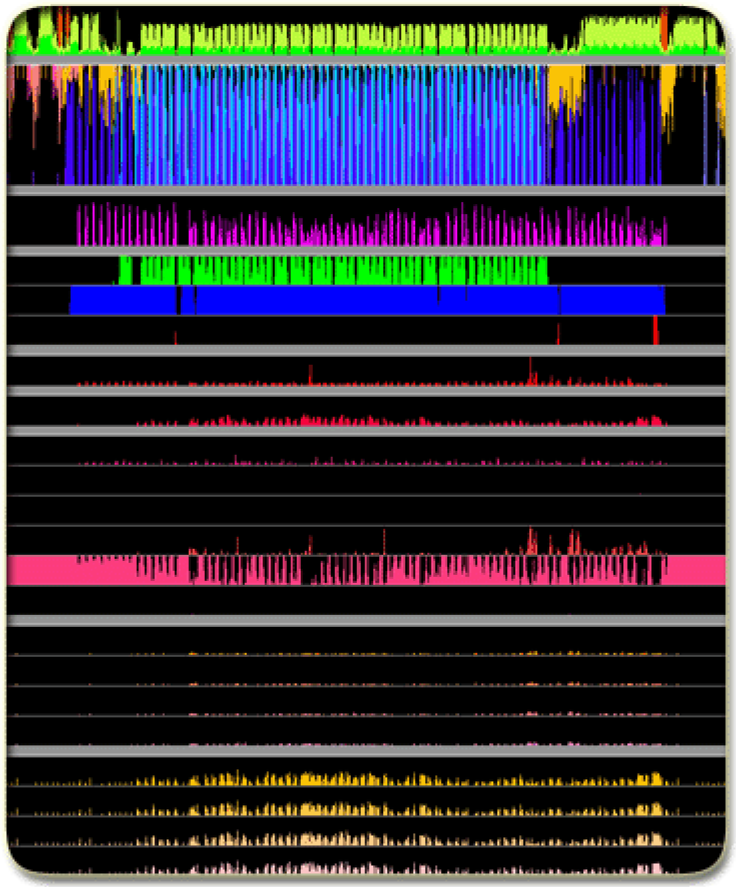
```
Statistics (current x8)
Time      : 2154648 --> 3203223
Duration  : 1048576 (1048576)
MainBus: CPU (memory) | MainBus: CPU (misc) | MainBus: DMA | Pipeline/GIFDT/VU | GS
MainBus: CPU (memory)
                                                 send /          occupation
Total                                  110389 (10.53%) /  168439 (16.06%)

Uncached Store                             63 ( 0.01%) /     157 ( 0.01%)
Uncached Load                              76 ( 0.01%) /     78
Uncached Accelerated Store                  7 ( 0.00%) /     1
Uncached Accelerated Load                   8 ( 0.00%) /     1
D$ Write-back - Load/Store Miss          5808 ( 0.55%) /    587
D$ Write-back - Cache Instruction        5811 ( 0.55%) /    601
D$ Refill - Load Miss                    3927 ( 0.37%) /   1590
D$ Refill - Prefetch Instruction            0 ( 0.00%) /
D$ Refill - Store Miss                  10832 ( 1.03%) /   3996
Uncached Execution                          0 ( 0.00%) /
I$ Miss Refill                           3049 ( 0.29%) /   1102
Cache Instruction - Fill Suboperation      33 ( 0.00%) /     12
CORE --> MAIN MEM                          70 ( 0.01%) /      7
CORE <-- MAIN MEM                           2 ( 0.00%) /
```

Software

- Linux app.
  - uses GTK
- Windows port
  - uses GTK too

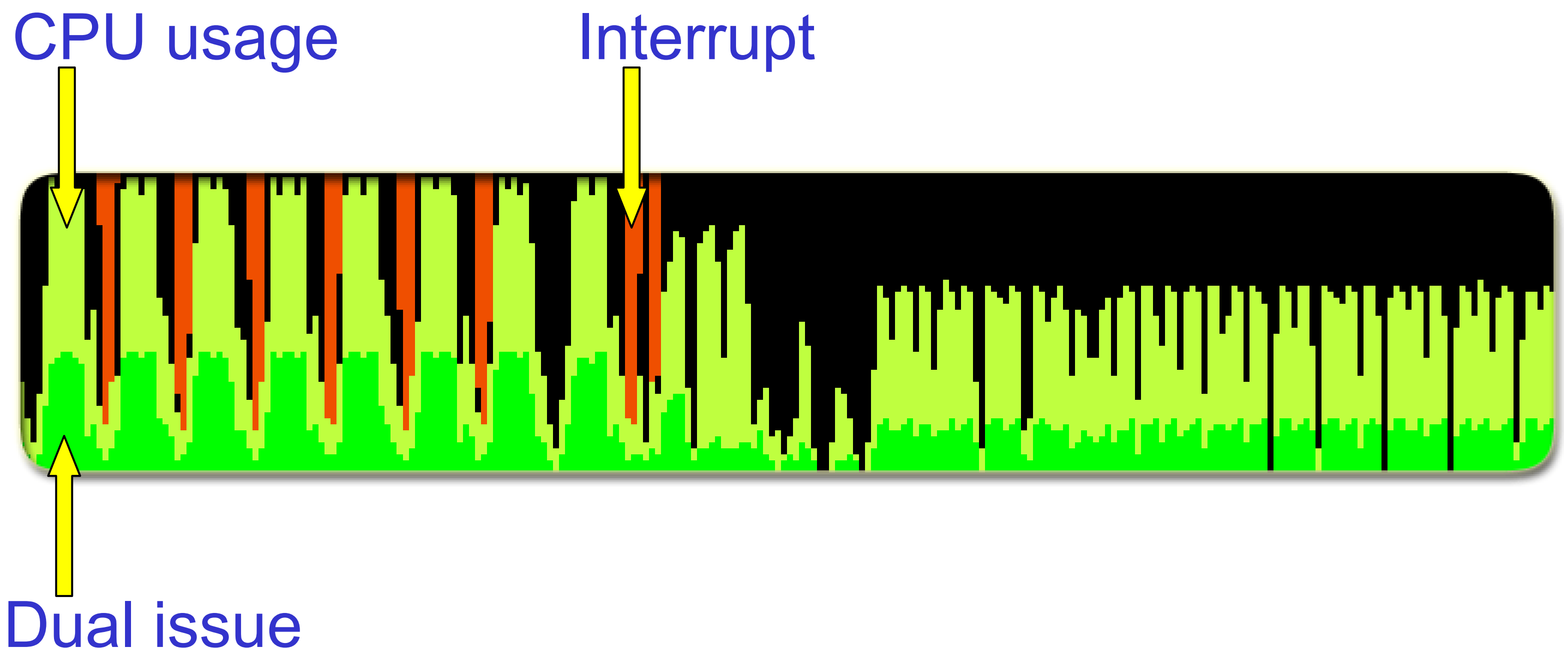
How do you read the graphs ?

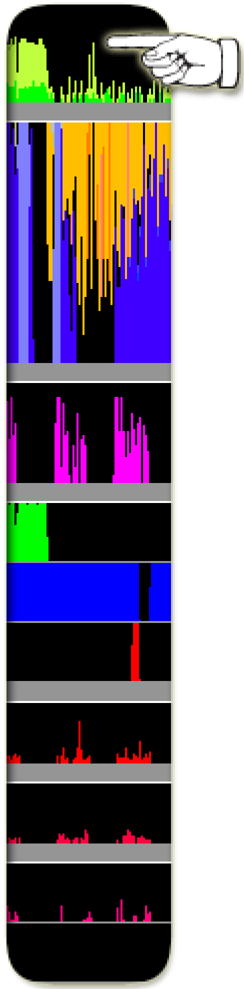
The different rows detail the different peripherals of the hardware



- You can see CPU activity, DMA transfers, VUs running or stalling, pixels being drawn, GS being idle, …

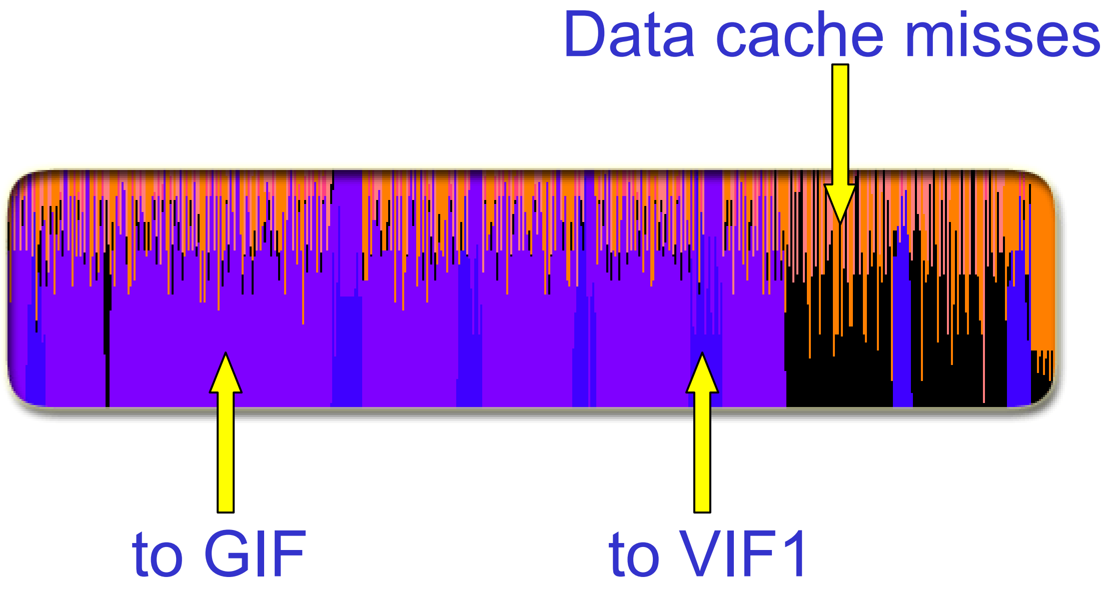
CPU usage

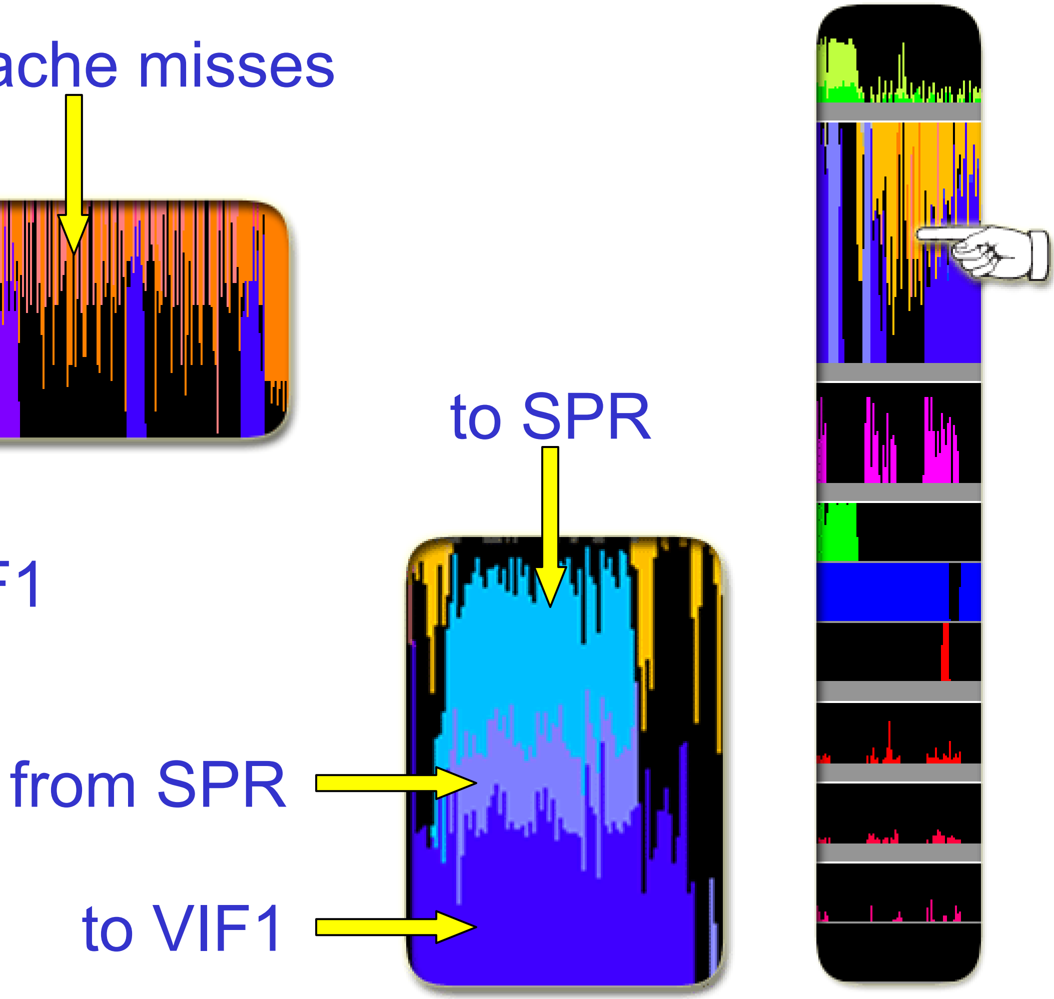




Most games are CPU bound (IA, physics), this row shows how efficiently the CPU is used

DMA transfers

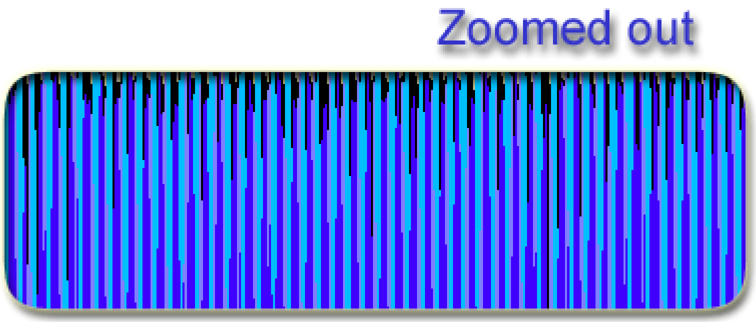


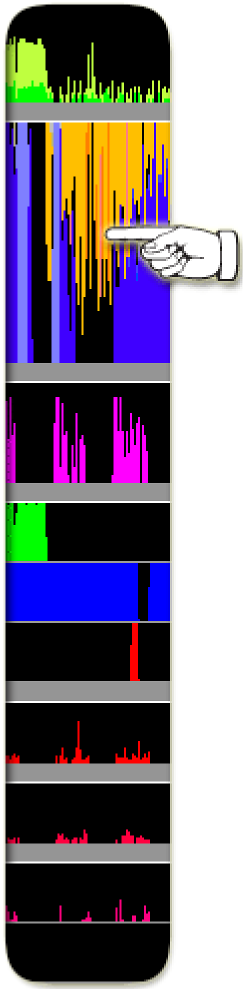


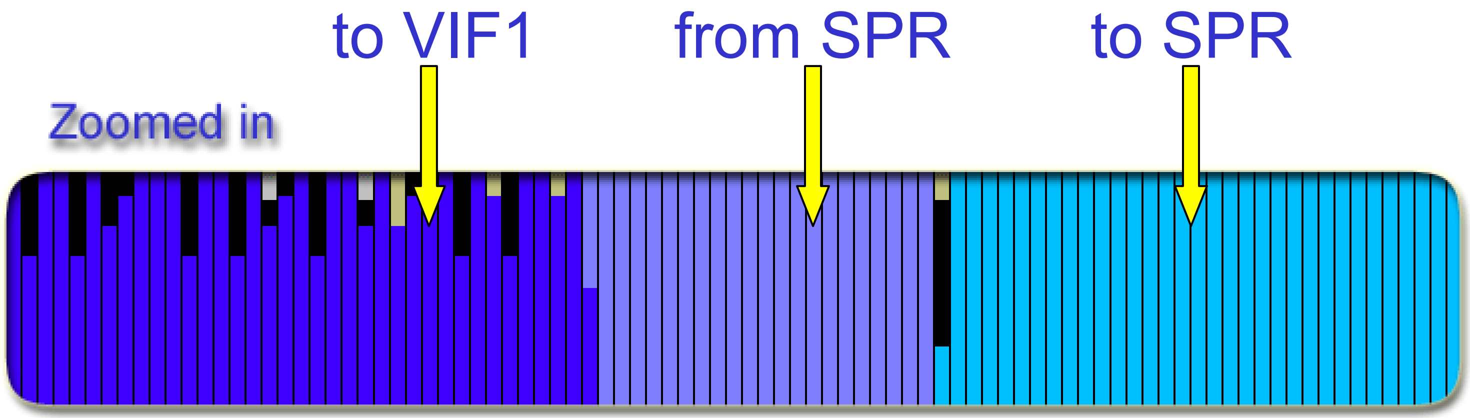
Cache misses affect
DMA performance

Short DMA transfers

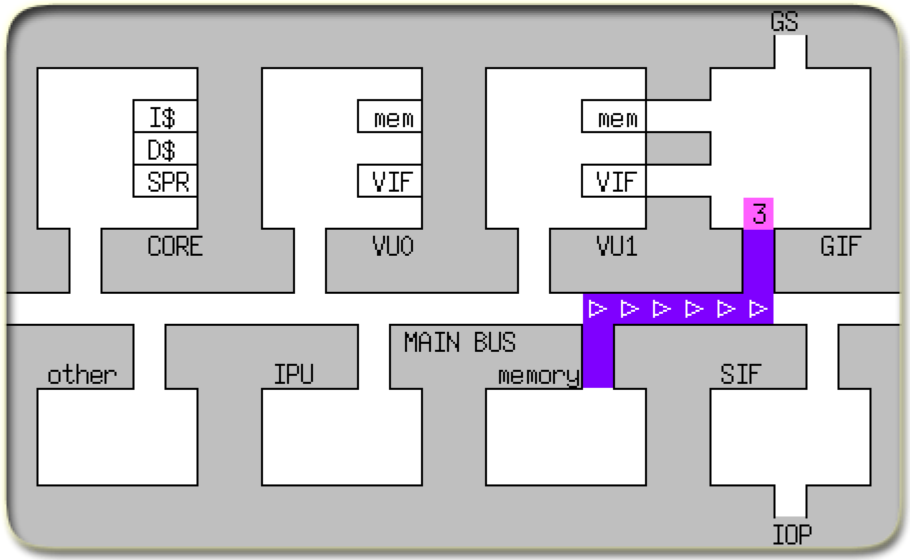
Short bursts of transfers are generally the way to go when using VU0

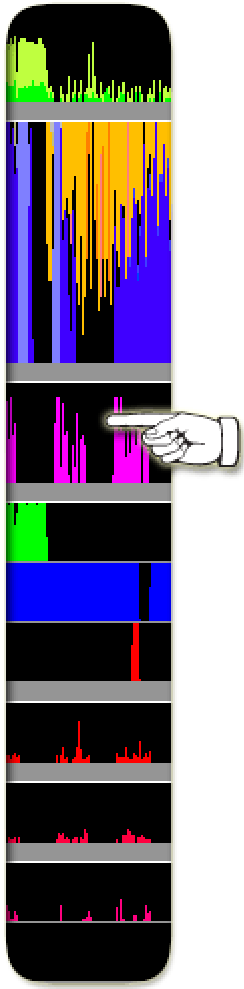




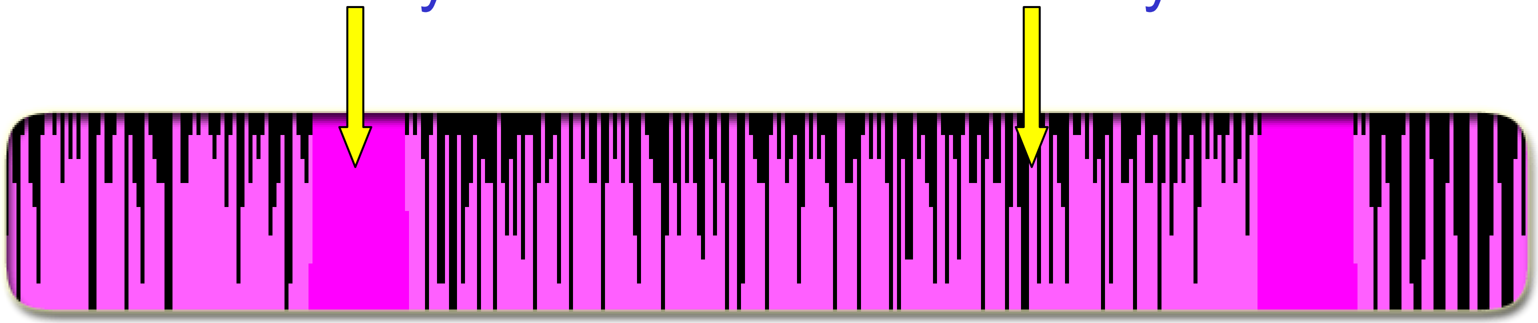


GIF traffic





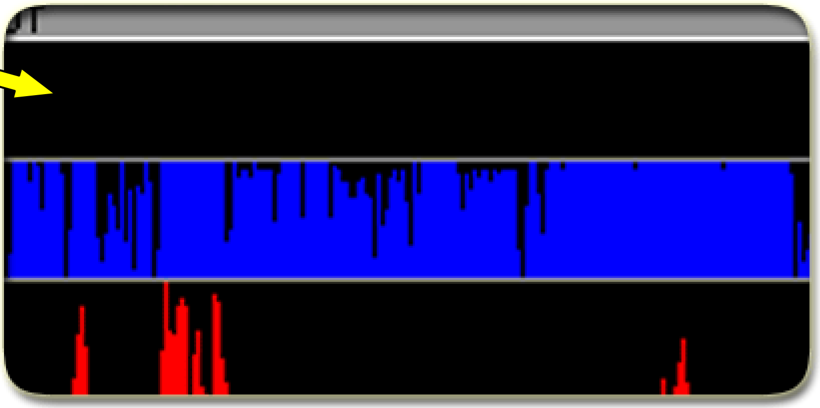
VU memory to GIF main memory to GIF



Vector Units usage

Typical: no VU0…

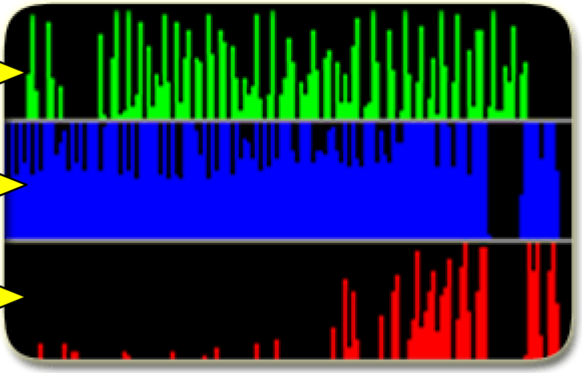
Macro mode is evil

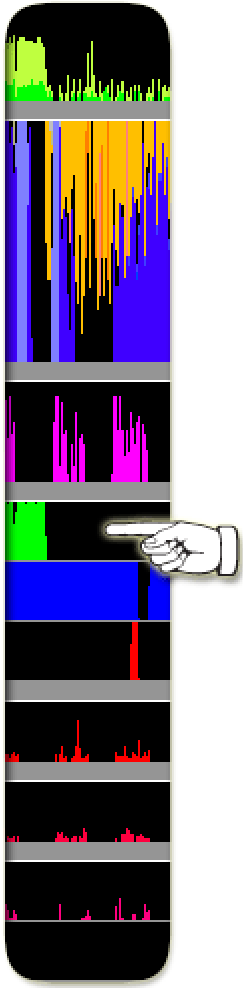


VU0 runs

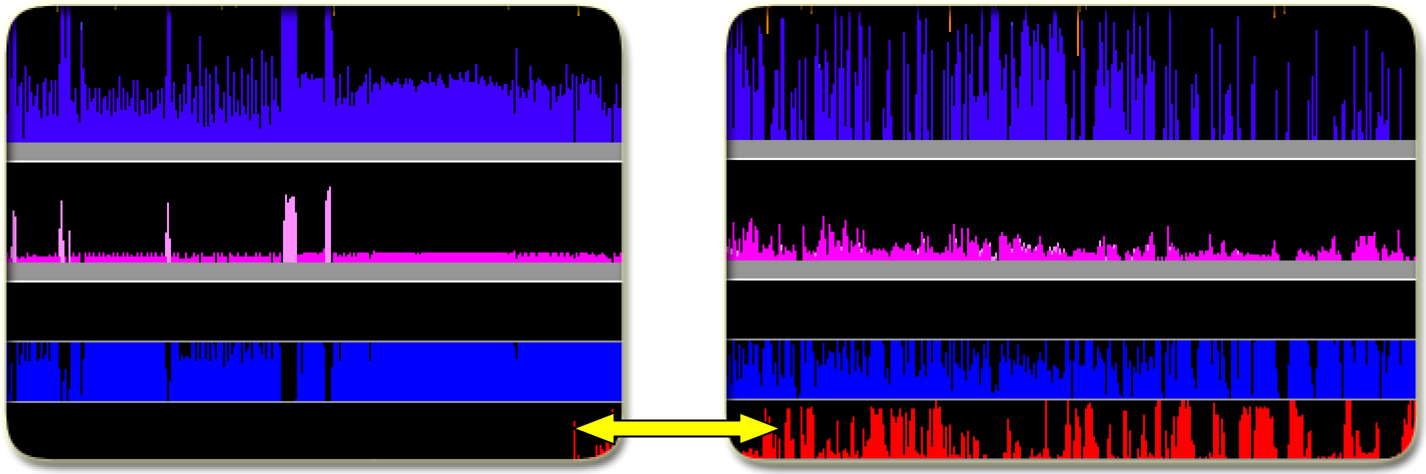
VU1 runs

VU1 stalls (xgkick)



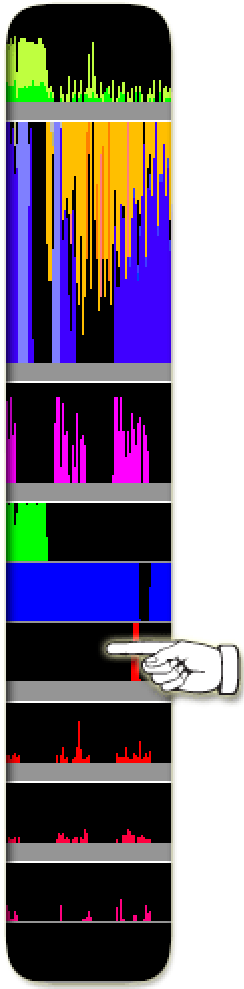


XGKICK stalling



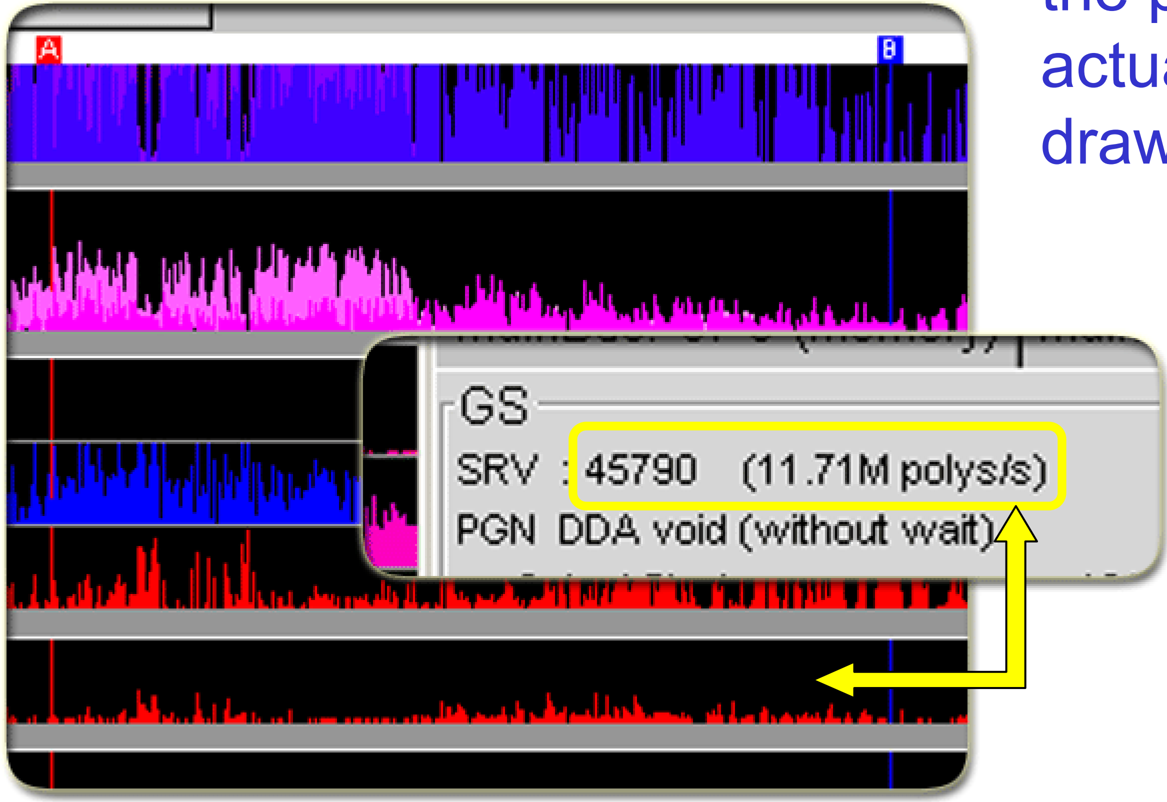
VU1 bound renderer

GS bound renderer



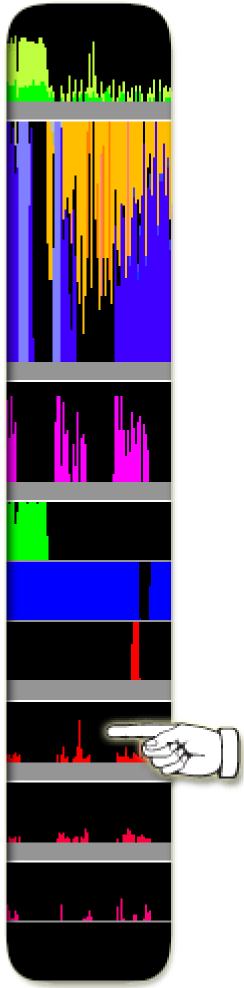
This row shows VU1 waiting for the GS to finish drawing the previous batch of primitives

Primitives being drawn



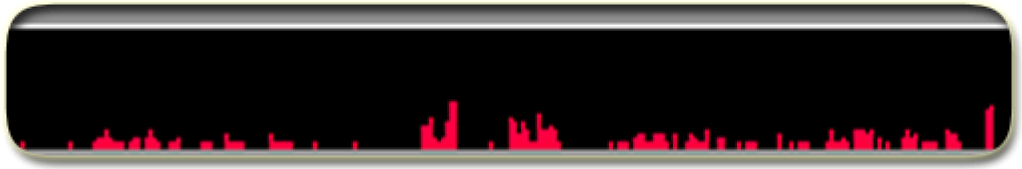
SRV : 45790 (11.71M polys/s)
PGN DDA void (without wait)

SRV row shows the primitives actually being drawn

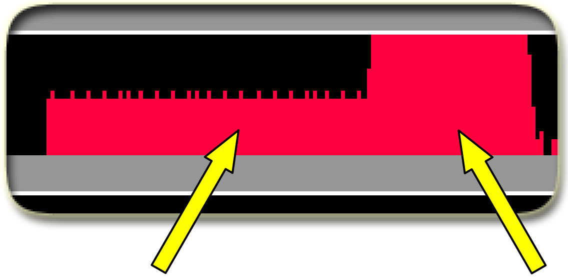


Pixels output

Typical geometry being drawn

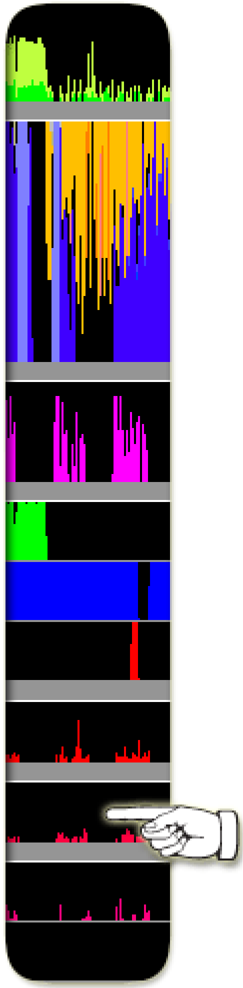


Typical fullscreen operations

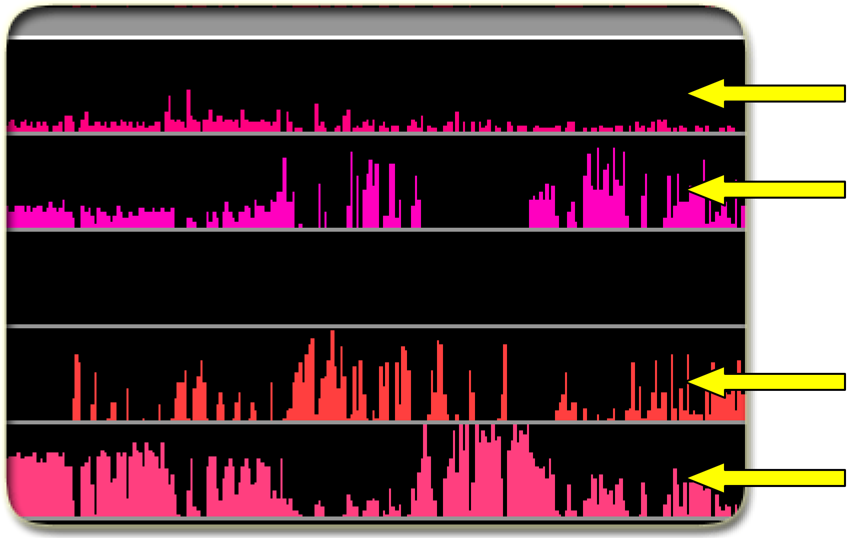


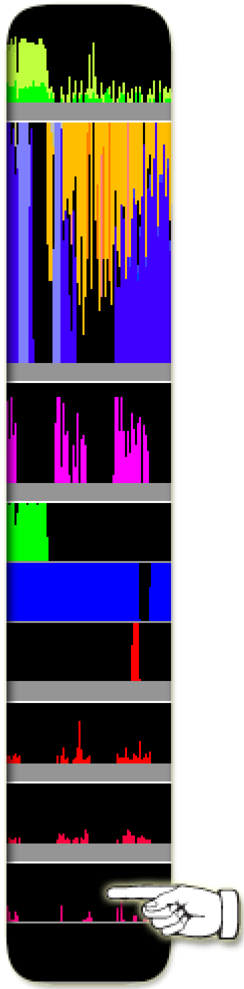
Textured

Untextured



More GS signals





You typically want less pixels undrawn (DDA void), less pixel unit stalls and less idling on the GS.

Lessons learnt

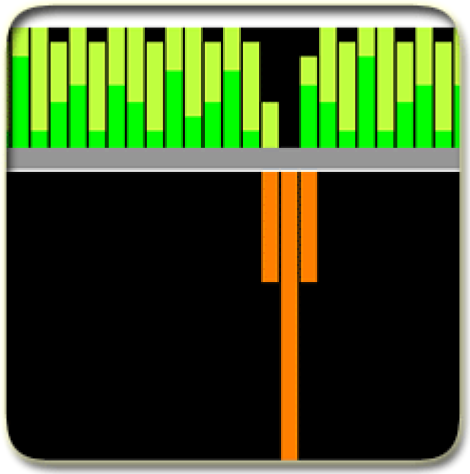
- A lot of games have been tested
- Patterns tend to show up
  - We'll look at some of them in a minute
- PA proved to be very useful in many cases
- It's never too late to PA your game
  - Even after submission
    - See what you can improve in the sequel
  - But preferably right at the beginning of development

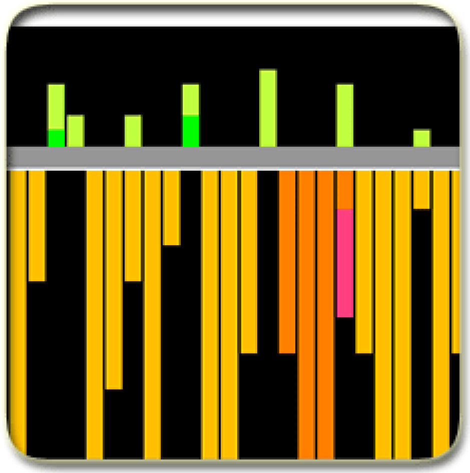
Lessons learnt (2)

- Most games are CPU bound
  - Spend more time optimising CPU code than VU1
- VU0 is largely underused
  - and it is an understatement
- DMA toSPR, processing, DMA fromSPR back to memory faster than processing from memory directly
- Some full screen operations inefficient
  - Use 32 pixels wide sprites

Typical patterns

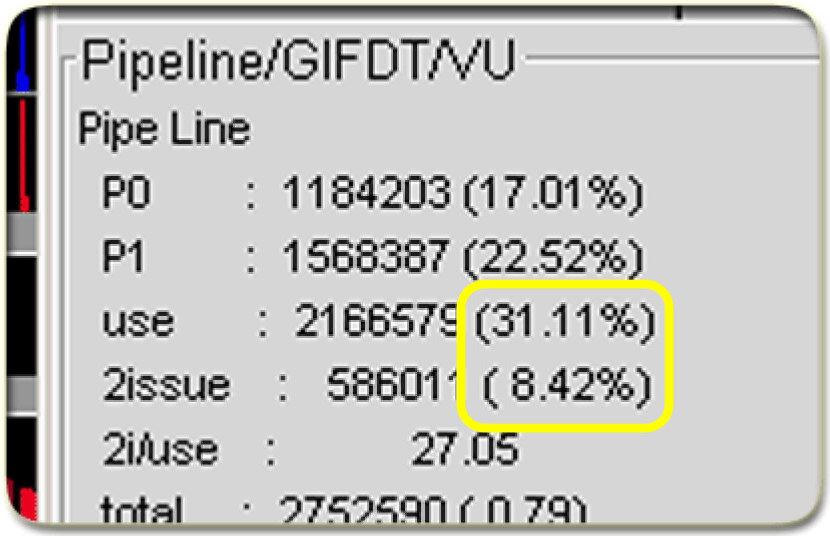
- Cache misses
  - Note that cache misses have a direct influence on CPU efficiency
  - It is a major factor on the overall performance on the EE side
- VU0, lack thereof
  - Use it, I can not say it enough
- Interrupts are expensive
  - Use with parsimony





CPU efficiency

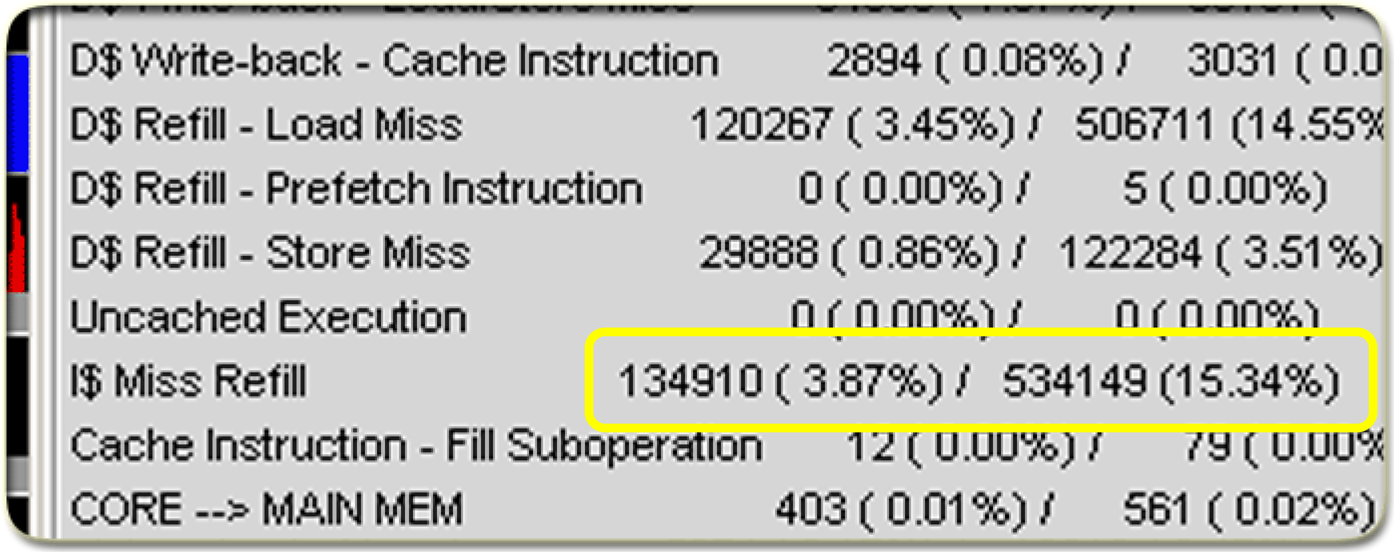
- Compilers not exactly optimal
- Use inline asm for time critical loops whenever possible
- Use the vector units
  - Especially VU0
- Use ScratchPad



```
Pipeline/GIFDT/VU
Pipe Line
 P0      : 1184203 (17.01%)
 P1      : 1568387 (22.52%)
 use     :  2166579 (31.11%)
 2issue  :   586011 ( 8.42%)
 2i/use  :         27.05
 total   : 2752590 ( 0.79)
```

Memory efficiency

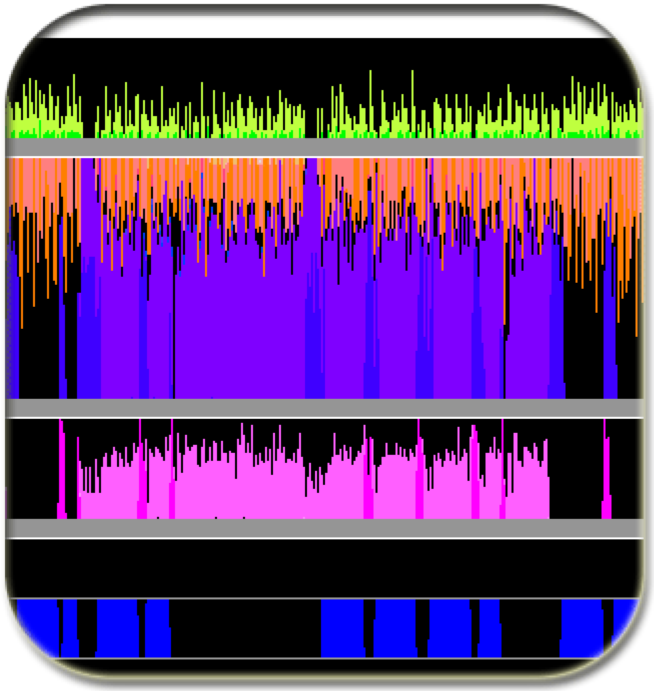
- 4% sent = 15% occupation
  - Cache misses are a killer
  - Better to use several small loops rather than one big loop that does everything



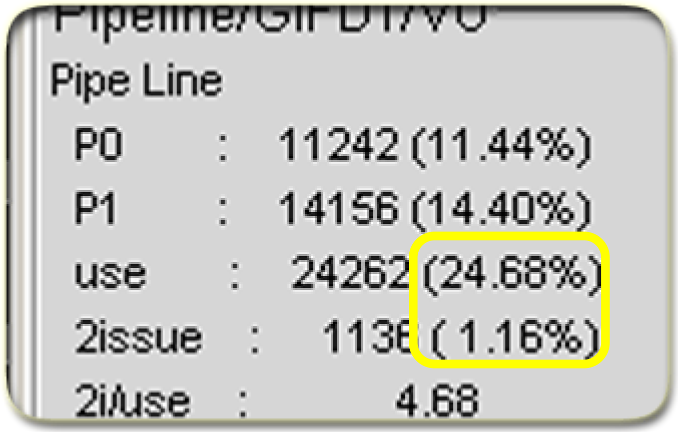
```
D$ Write-back - Cache Instruction        2894 ( 0.08%) /    3031 ( 0.0
D$ Refill - Load Miss                  120267 ( 3.45%) /  506711 (14.55%
D$ Refill - Prefetch Instruction            0 ( 0.00%) /       5 ( 0.00%)
D$ Refill - Store Miss                  29888 ( 0.86%) /  122284 ( 3.51%)
Uncached Execution                          0 ( 0.00%) /       0 ( 0.00%)
I$ Miss Refill                         134910 ( 3.87%) /  534149 (15.34%)
Cache Instruction - Fill Suboperation      12 ( 0.00%) /      79 ( 0.00%
CORE --> MAIN MEM                         403 ( 0.01%) /     561 ( 0.02%)
```

Mixing geometry and textures

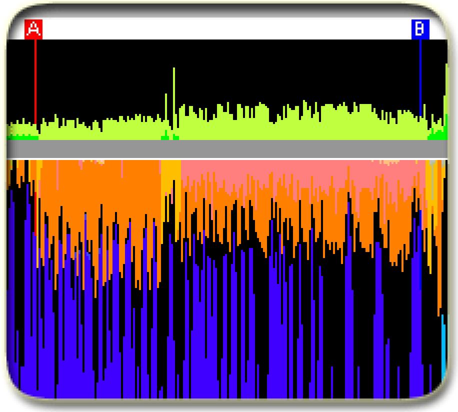
- Example of transfers interleaved
  - Cache misses affect CPU performance and DMA transfers efficiency
  - Textures are sent with low priority, so geometry data can be interleaved



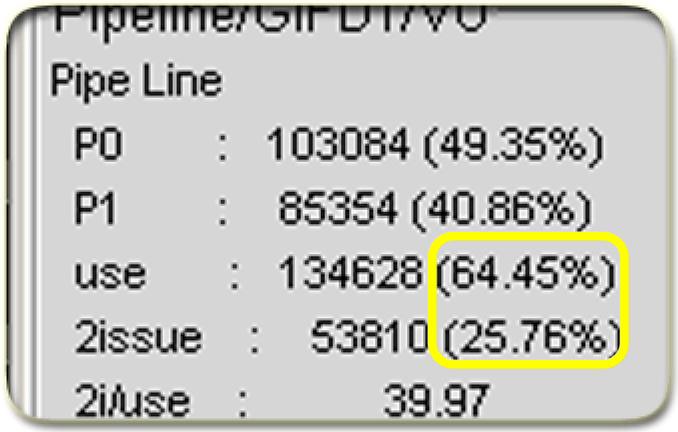
Good and bad performance



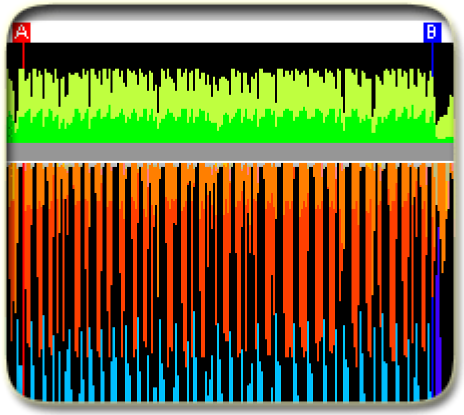
```
Pipeline/GIFDT/VU
Pipe Line
  P0      :   11242 (11.44%)
  P1      :   14156 (14.40%)
  use     :   24262 (24.68%)
  2issue  :    1136 ( 1.16%)
  2i/use  :      4.68
```



Could be better
- 25% used
- 1% dual issue



```
Pipeline/GIFDT/VU
Pipe Line
  P0      :  103084 (49.35%)
  P1      :   85354 (40.86%)
  use     :  134628 (64.45%)
  2issue  :   53810 (25.76%)
  2i/use  :     39.97
```



More like it
- 65% used
- 25% dual issue

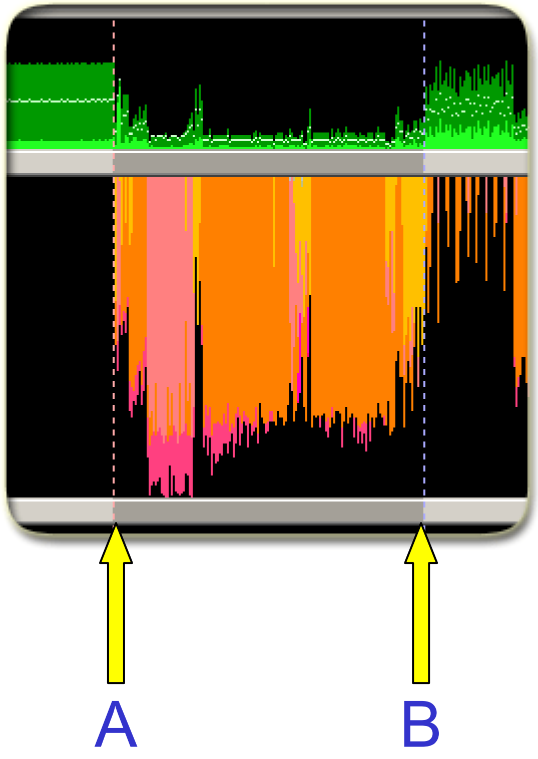
(Note the use of SPR)

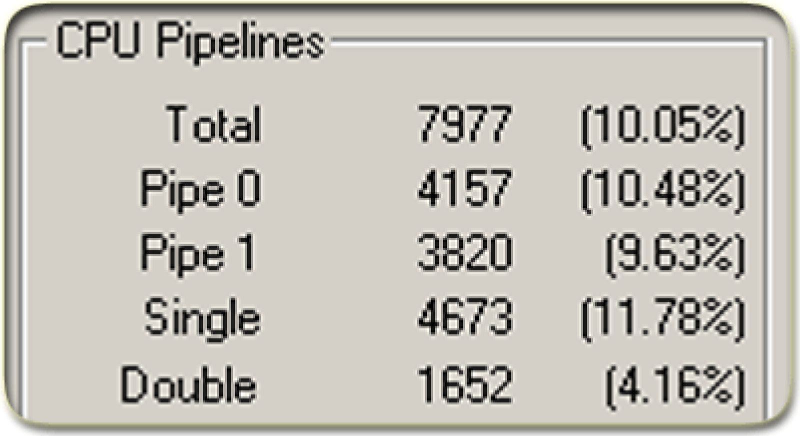
Upcoming Performance Analyser

- The hardware
  - A T15k
  - Contains a PC board to grab the data
  - Roughly the same as the prototype
    - only better
- The software
  - A Linux only version of pmon2
  - The PA will be controlled
    remotely
    from your machine through DECI2 protocol

Get the useful information quickly

- Set the A and B markers
  - You can then read the stats
  - Useful to measure the speed of a piece of code



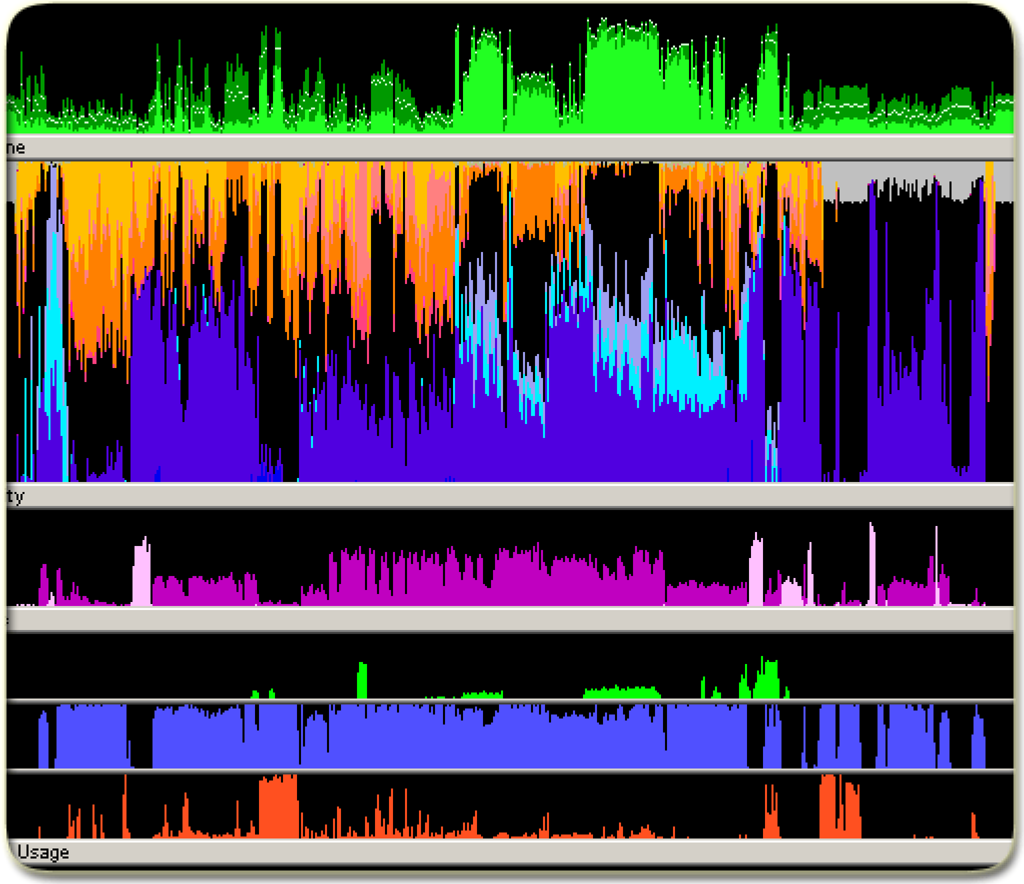


CPU Pipelines

Total	7977	(10.05%)
Pipe 0	4157	(10.48%)
Pipe 1	3820	(9.63%)
Single	4673	(11.78%)
Double	1652	(4.16%)

What you ideally want to get

- 10..20 million polys per second
- >50% CPU usage
- >80% dual issue
- It's been done !



How to get hold of the PA ?

- Available to all licensed developers
  - A matter of months
- You can already send us CDs to run through the prototype
  - Via FTP (ISOs)
  - Good old fashion airmail
- Or visit us in London
  - We have pizza

Further Information

- Meet us
  - SCEE Booth, Exhibition Stand #9
  - Hotel suite on appointments
  - Come and try your game through the PA
- Fin