Adaptive Optics - There's an app for that now, right?

The prototype ESO/ELT WFRTC

and

What COTS really means these days

Poul-Henning Kamp & Force Technology
Poul-Henning Kamp
30 years hacking Unix
20 years FreeBSD
NTP/nanokernel
Varnish HTTP cache
Lots of other stuff

Force Technology
~1k Engineers
ESO/ELT WFRTC in numbers

5004  (X,Y) pairs from WF-sensors

n*12  Sparse registration matrix

6350  Actuators

n*m  Fully populated actuator matrix

500  Hz operation rate

1  ms latency

20  µs jitter

64  Mflop/cycle

~64  Gflops
1G ether switch
10G Arista switch
2 * Dell 4-core PC
5 * Dell 48-core PC
~= € 50K
FreeBSD (UNIX)
15 lines assembler
10k lines C source
Actual performance

Using only 4 * 48 core servers
(last one became WF-simulator)

Rate 500 ... 800 Hz
Limiting factor: 1500 bytes/packet
Rate > 1kHz possible with 9k packets
Latency < 800µs —//--—

Jitter < 15µs
(Requirement driver is unclear)
How ?!

WFRTC is not a real-time job

WFRTC is a batch-job

We know when data arrives

We know our deadline

We have nothing else to do
HowTo

Stock FreeBSD kernel
    Linux can probably also do it

Disable interrupt-hogs
    USB, cron, power-mgt, etc.

Steer OS-timer interrupt to work-cycle

Enable HW-NUMA (= disable interleave)

Lock proc/threads to CPU-cores
Big picture

Reg-mtx

Reg-mtx

Reg-mtx

Reg-mtx

\(\frac{1}{4}\) Act-mtx

\(\frac{1}{4}\) Act-mtx

\(\frac{1}{4}\) Act-mtx

\(\frac{1}{4}\) Act-mtx

W0 W1 W2 W3
Big picture, more details

PLL:
Adaptation to WF rate/phase

Reg-mtx:
Sort+partition reg-matrix (1Hz)

OaM:
Operation & Maintenance
Incl: update act-matrix, params
Work distribution

Socket #0
- Core #0: Kernel
- Core #1: Main-loop, RX, TX
- Core #2-5: Reg-mtx

Socket #1-3
- Core #6-47: Act-mtx (+ filter)

Reflect HW hierarchy:

Really not a 4*12
But a 8*6 topology
Performance: (1 server ~ €6k)
Conclusion:

Easy bit:
  Math
  Straight forward matrix-operations

Hard bit:
  Timing
  Merging async events into sync work-flow

Take-home message:

  Adaptive optics runs on COTS hardware now

  2kHz & 500µs within reach
Why it works

1. Stock-trading needs fast networks
2. Climate models need massive clusters
3. Gamers wants fast graphics/SIMD/MMX (Trick#1)
4. Modern UNIX kernels
5. Trick#2: Timer-steering
Why it works: 10G Ethernet

Driven by:
  Algorithmic stock-trading paid for this
  Low and predictable latency
    Arista switch: < 500ns cut-through delay

BUT:

- Packet loss
- Packet reordering
- No end-to-end connectivity
- No end-to-end packet timing
Why it works: MPP COTS machines

Driven by:
  - Moores law running out
  - Scientific computing (climate, oil, biology)

BUT:
  - Speed at the cost of:
    - Parallelism
    - Latency
    - Multiplexing
Why it works: Faster graphics/SIMD/MMX

Driven by:
   First Person Shooter games
   Fast (and loose) physical modelling

BUT:
   - Moving towards GPU/Co-processor model
Why it works: Trick #1 MMX instructions

```asm
__asm__ __volatile__(
"\n"
"    xorps %%xmm0, %%xmm0\n"
"    .align 16, 0x90\n"
"1:\n"
"    movups (%1), %%xmm1\n"
"    movups 16(%1), %%xmm2\n"
"    movups (%2), %%xmm3\n"
"    movups 16(%2), %%xmm4\n"
"    add $32,%1\n"
"    add $32,%2\n"
"    mulps %%xmm3, %%xmm1\n"
"    addps %%xmm1, %%xmm0\n"
"    mulps %%xmm4, %%xmm2\n"
"    addps %%xmm2, %%xmm0\n"
"    decl %3\n"
"    jne 1b\n"
"    movaps %%xmm0, %0\n"
: /* outputs */
"=m" (ans)
: /* inputs */
"r" (regsens),
"r" (lhsrow),
"b" (n)
: /* clobbered */
"xmm0",
"xmm1",
"xmm2",
"xmm3",
"xmm4",
"memory"
);
```
Why it works: Modern UNIX kernels

Driven by:
  Keeping up with hardware
  Massive server-farms (Google, Facebook, etc)
  Real-Time like facilities (financial, SCADA etc)

BUT:
  "The kernel is obsolete"
  — Rob Pike (2001)
Why it works: Trick #2 Timer steering

```c
void
i8254_trick_now(double period)
{
    unsigned i;

    trick = (uint16_t)((14.318318e6/12.) * period);

    disable_intr();

    /*
    * Switch timer to single-shot, and force an interrupt
    * in a few microseconds
    */
    outb(TIMER_MODE, TIMER_SEL0 | TIMER_INTTC | TIMER_16BIT);
    outb(TIMER_CNTR0, 2);
    outb(TIMER_CNTR0, 0);

    /*
    * Wait for interrupt to happen
    */
    for (i = 0; i < 4U; i++)
        if (inb(TIMER_CNTR0) & 0x80)
            break;

    /*
    * Set timer in rategen mode
    */
    outb(TIMER_MODE, TIMER_SEL0 | TIMER_RATEGEN | TIMER_16BIT);
    outb(TIMER_CNTR0, trick & 0xff);
    outb(TIMER_CNTR0, trick >> 8);

    enable_intr();
}
```
The future?

How many servers will ESO/ELT need?
- 2013: 3 or 4
- 2020: 1 — maybe 2

Perspectives:
- More advanced control law possible?
- Lower barrier for experimental modes
- Hardware redundancy
- Very high rates/low latencies
- No custom hardware
- Cheap
The future?

Consider a joint FOSS project:
- LGPL source code (ask ESO for copy)
- Already parameterized and adaptable

Force & PHK will be happy to help
- Reasonable rates
- Will visit telescopes
Supporting material:

Packet transmit-time histogram:
Supporting material:

Packet transmit-time histogram: