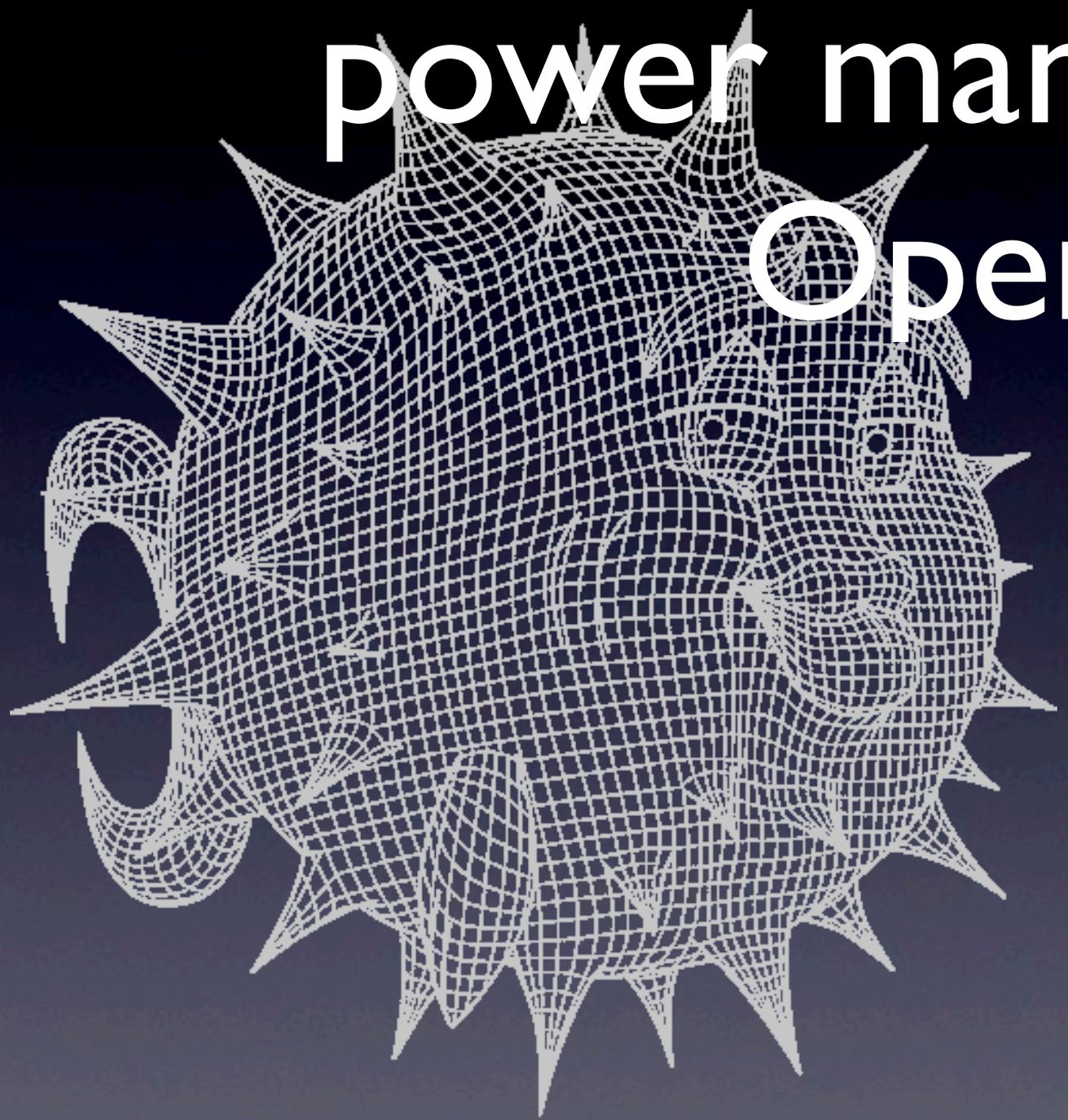
The future of processor power management in OpenBSD



Gordon Willem Klok
gwk@openbsd.org

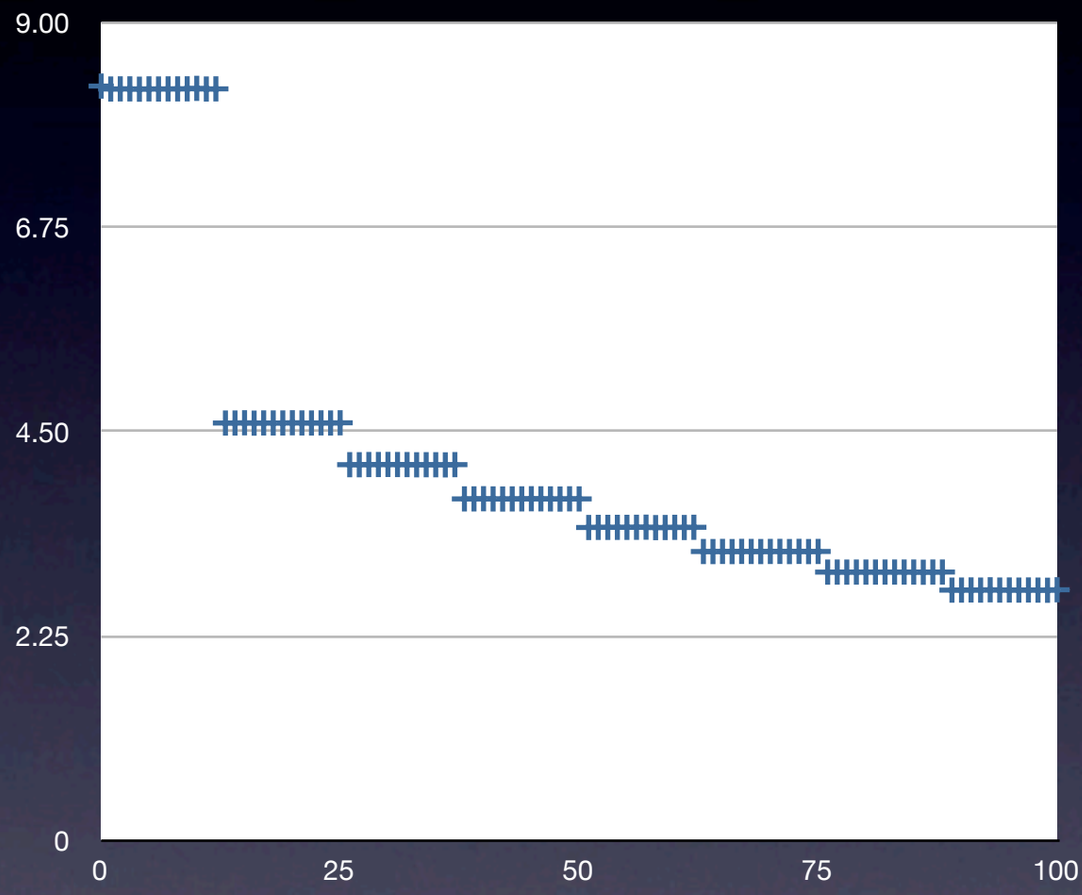
What is processor power management?

- Frequency and Voltage Scaling e.g. Intel Enhanced Speed Step (EST), AMD Powernow!/Cool`n'Quiet
- Duty cycle modulation e.g. Intel Thermal Control Circuit (TCC)
- Not limited to OpenBSD/i386 or OpenBSD/amd64 other architectures have setperf (sparc64, macppc, zaurus)

hw.setperf sysctl

- Accepts integer values between 0 and 100 inclusive.
- It is supposed to exhibit a rough correspondence to a percentage of machine performance.
- However only influences cpu power management and no processors currently provide such granularity.

hw.setperf vs md5 -tt



As you can see the behavior is similar to a step wise function

apmd

- Original purpose of this daemon was to communicate suspend and battery status information between user land/kernel
- New functionality added support for two modes -A automatic performance adjustment mode and -C or cool running mode
- Both methods use the same hw.setperf sysctl and make decisions based upon the average amount of time the cpu(s) spend in the idle process

Automatic mode

- If idle time is less than 10% or AC power is connected and the battery is more than 15% charged raise hw.setperf
- If idle time is greater than 30% and the machine is on battery it lowers hw.setperf as much as possible

Cool running

- Attempts to keep the machine in its lowest state provided idle time is above 30%
- If the machine is idle less than 10% of the time it raises hw.setperf

Going in kernel

hw.setperf mechanics

```
int
hw_sysctl(int *name, u_int namelen, void *oldp, size_t *oldlenp, void
    *newp,  size_t newlen, struct proc *p)

sys/kern/kern_sysctl.c line 619:

    case HW_SETPERF:
            if (!cpu_setperf)
                return (EOPNOTSUPP);
        err = sysctl_int(oldp, oldlenp, newp, newlen, &perflevel);
        if (err)
                return err;
        if (perflevel > 100)
                perflevel = 100;
        if (perflevel < 0)
                perflevel = 0;
        if (newp)
                return (cpu_setperf(perflevel));
        else
                return (0);
```

hw.setperf driver initialization

- Initialization of setperf drivers takes place outside of autoconf
- Very architecture specific drivers: are typically identified by CPU features
- Initialization is delayed because identification of CPU feature compatibility may not be the whole story (more on that later)

Multiple processor power mechanisms

- Some machines can have more then one method of controlling process power consumption e.g. any modern intel processor.
- So we need a mechanism to control which setperf routine will be used on these systems
- Yet a another global variable

setperf_prio

- Each driver has a priority decided by the developer
- It checks the global setperf_prio when its init routine is called.
- If the current value is greater than its value it punts.
- Other wise it proceeds with init and if everything is good it overwrites the values of the globals with its own.

i386 enhanced speed step example

```
void
est_init(const char *cpu_device, int vendor)
{
    if (setperf_prio > 3)
                return;
    /* many boring lines later */
    cpu_setperf = est_setperf;
    setperf_prio = 3;
}
```

Lockstep SMP scaling

- Create a mechanism to synchronize processor transitions
- Create wrapper function around the sysctl mechanism
- The wrapper function figures out which underlying mechanism to call.

mp_setperf example

```
# sysctl hw.setperf=0
```

???

hw_setperf

*cpu_setperf(0)

mp_setperf

*ul_setperf(0)

k8_powernow_setperf(0)

cpu_setperf
=
mp_setperf

On each cpu

Implementing lock step behavior

- A hw.setperf sysctl is received from user land, processor sets it's state to in transition.
- Sends an IPI (inter-processor interrupt) to every other processor in the system informing them that they should stop
- Each processor adjusts its state to ready,
- Initial processor has been spinning, waiting for other processors, it then signals each one to begin its transitions
- (they use shared variable to know what state to transition to)
- Initial processor performs its own transitions then spins waiting for the others to finish
- Once thats done it restores the system to the state prior to the transition.

ACPI

Advanced Configuration and Power Interface

- OpenBSD ACPI support is not based on the Intel reference code ACPICA
- Developed mostly by Jordan Hargrave and Marco Peereboom over the last two years.
- Currently used for interrupt routing, battery status, and one of the setperf drivers (suspend/resume is a work in progress)

PCT

Performance Control

- ACPI Object that when evaluated will transition the processor to a different frequency/voltage
- However ACPI does not understand CPU registers (can only address I/O ports and memory)
- All modern technologies for FVS scaling use processor model specific registers
- Completely useless on modern machines.

PSS

Performance Supported States

- ACPI object used by setperf drivers to determine the supported frequencies (more importantly the register values to get the processor into that state)
- Simple function to get the PSS states from the ACPI tree.

```
int
acpicpu_fetch_pss(struct acpicpu_pss **pss)
```

PPC

- Used by ACPI to notify the operating system that something has changed wrt the supported states.
  - e.g. machine may not support all available states when running on battery power

```
void
acpicpu_set_notify(void (*func)(struct acpicpu_pss
*, int))
```

PDC

Processor Driver Capabilities

- Previously I described the PCT object as completely useless
- In order to support legacy operating systems which only understood speedstep as a I/O register off the south bridge (which works with PCT), some vendors decided to emulate the original speedstep using SMM (system management mode) for systems supporting enhanced speedstep
- SMM makes what should be a low latency operation decidedly not so
- So ACPI incorporates the ability for the operating system to declare its capabilities as far as supporting the hardware required to do FVS and if you support the right options the BIOS will re enable the EST registers.
- This mess is only necessary on Intel chips, AMD did it right.

Problem summary

- Can only use one power management mechanism at a time.
- Deciding which one is the “right” one is difficult.
- Driver attachment and exclusion is not standardized and architecture specific.
- Are there more problems ? Of course

More problems

- CPU idle time is not updated "fast enough" modern FVS technologies can transition between states in less than 25 microseconds
- Lock step approach also has a high latency between transitions
- Processor technology marches on... split power planes and the ability to scale cores independently of each other.

Problems continue

- apmd reacts poorly to changes in load (it lags considerably and over reacts to spikes leading to poor power savings)
- Lots of cores schedular has some concept of balance (say round robin between the cores) but if your objective is saving power leaving a CPU halted is ideal.

Objectives of a new API

- Support more than a single setperf method at once
- Make the data structure per core (tie it to struct cpu_info)
- Make the bulk of the functions for managing cpu_governors machine independent

Objectives (continued)

- Split detection and initialization, provide the ability for drivers to view what other drivers are available and not register if they spot a potential conflict.

Move scaling decisions into the kernel

- Take policies such as "cool running" or automatic mode from the user.
- Make the decisions to transition between states into the kernel
- Use the new per core idle process and use better statistics to guide transition decisions (e.g. the latency of the governor, and make better accounting of idle time)

So when do I get to use it ?

- Work underway since H2K8 Hackathon in Portugal
- Expecting to complete this work during the up coming C2K9 hackathon in Edmonton.
- So hopefully we see it in OpenBSD 4.6

Questions?