Motes, nesC, and TinyOS

Gary Wong

December 9, 2003

Introduction

- System overview
- Mote hardware
- nesC language
- TinyOS operating system

System overview

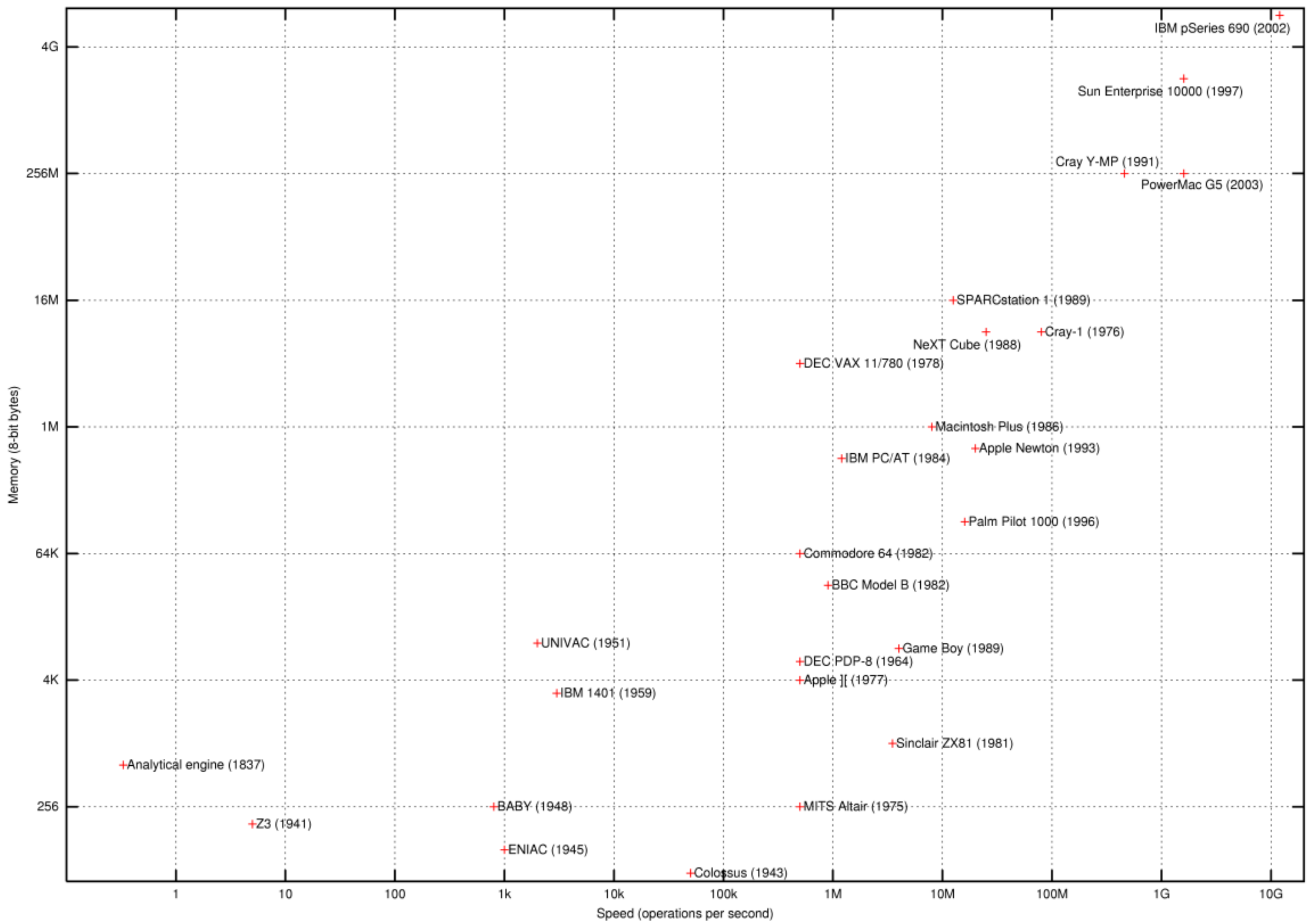
Consider an environment which requires fully autonomous operation:

- no mains power
- no wired communication
- no human intervention

What do these limitations make our computers and programs look like?

Mote hardware

- Processor: Atmel AVR ATmega128L μcontroller
  - 128KB flash ROM, 4KB RAM, 4KB E^2PROM
  - up to 8MHz
- Radio: Chipcon CC1000
  - UHF transceiver (300MHz–1GHz)
  - FSK modulation, up to 76.8kBaud
- Sensor boards


IBM pSeries 690 (2002)
Sun Enterprise 10000 (1997)
Cray Y-MP (1991)
PowerMac G5 (2003)
SPARCstation 1 (1989)
NeXT Cube (1988)
Cray-1 (1976)
DEC VAX 11/780 (1978)
Macintosh Plus (1986)
Apple Newton (1993)
IBM PC/AT (1984)
Palm Pilot 1000 (1996)
Commodore 64 (1982)
BBC Model B (1982)
UNIVAC (1951)
Game Boy (1989)
DEC PDP-8 (1964)
Apple][(1977)
IBM 1401 (1959)
Sinclair ZX81 (1981)
Analytical engine (1837)
BABY (1948)
MITS Altair (1975)
Z3 (1941)
ENIAC (1945)
Colossus (1943)
4G
256M
16M
1M
64K
4K
256
Memory (8-bit bytes)
1
10
100
1k
10k
100k
1M
10M
100M
1G
10G
Speed (operations per second)

Programming the AVR architecture

- lots of registers (32)
- RISC, load-store model
- conventional stack
- linear Harvard-style address space
- highly orthogonal instruction set

$\Rightarrow$ nice for conventional compilers

nesC language

A dialect of C:

- imperative, very C-like at the low level
- more declarative style at top level
- highly modular
- whole program compilation

nesC language

- Programs are built from **components**, which are either **modules** or **configurations**. Components provide and use **interfaces**.

- Modules implement interfaces with functions (**commands** and **events**); configurations connect interfaces together ("**wiring**").

- A program always has a top-level configuration.

- The concurrency model is based on **tasks** and **hardware events**: tasks never preempt execution, but hardware events do.

nesC language

The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages:

> *Print the words*
>
> **`hello, world`**

— Kernighan and Ritchie,
The C Programming Language (2nd edition)

But how can we write such a program in an environment with no alphanumeric I/O capability?

nesC example: HelloWorldM.nc (1)

```
module HelloWorldM {
    provides {
        interface StdControl;
    }
    uses {
        interface Timer;
        interface Leds;
    }
}
```

continues...

nesC example: HelloWorldM.nc (2)

continued...

```
implementation {
    command result_t StdControl.init() { ... }
    command result_t StdControl.start() {
        return call Timer.start( TIMER_ONE_SHOT, 1000 );
    }
    command result_t StdControl.stop() { ... }
    event result_t Timer.fired() { ... }
}
```

nesC example: HelloWorld.nc

```
configuration HelloWorld {
}
implementation {
    components Main, HelloWorldM, TimerC, LedsC;

    Main.StdControl -> HelloWorldM;
    Main.StdControl -> TimerC;

    HelloWorldM.Timer -> TimerC.Timer[ unique("Timer") ];
    HelloWorldM.Leds -> LedsC;
}
```

TinyOS operating system

TinyOS is a runtime environment for nesC programs running on Mote hardware:

- Performs some resource management.

- Selected components are linked into program at compile time.

- Written in nesC and C.

- All time-consuming commands are non-blocking.

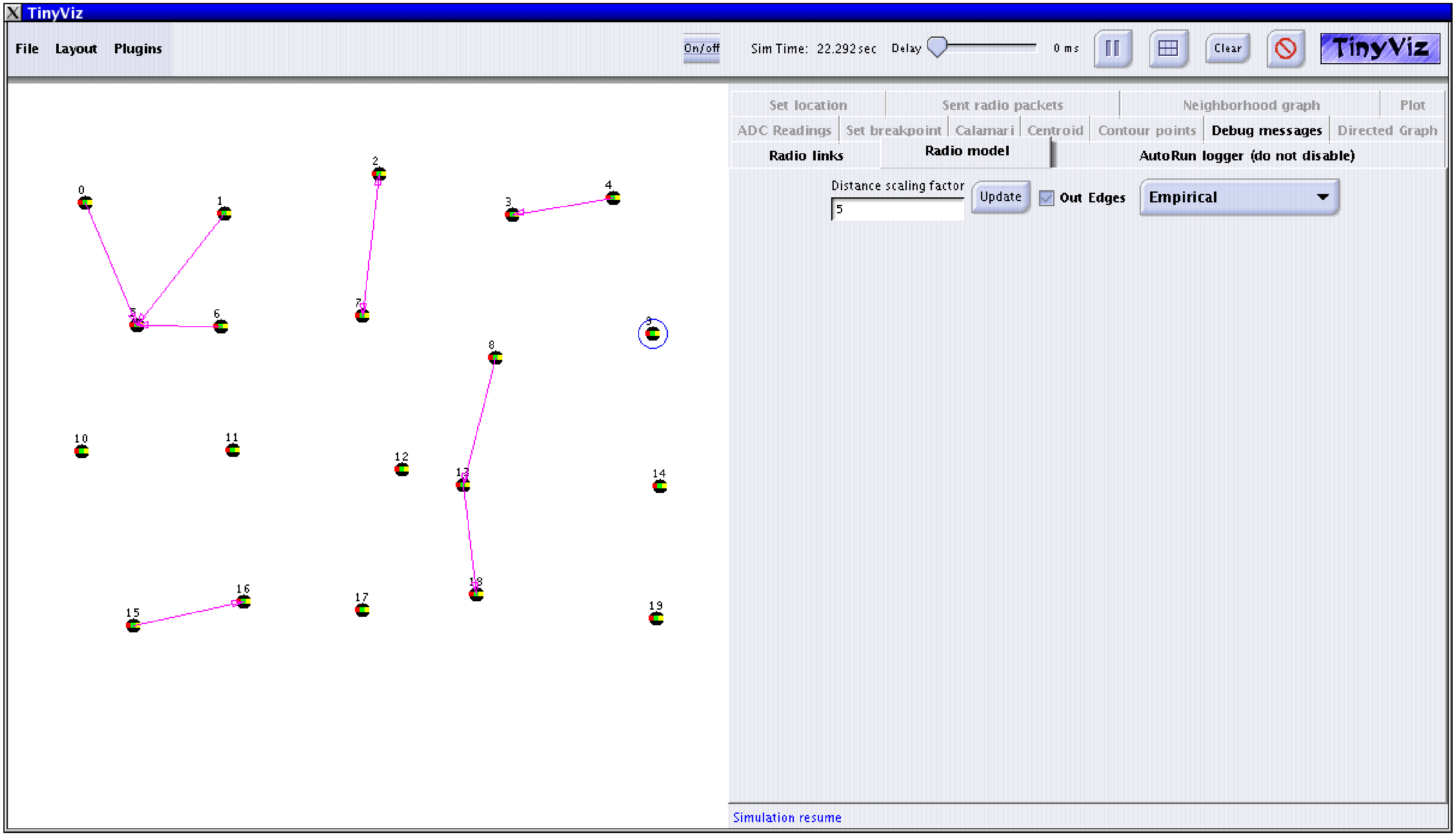
TinyOS operating system

Provided components include:

- Analogue to digital conversion
- Cryptography
- Data logging
- File system
- I^2C communication
- LED control
- Memory allocation
- Random number generation
- Routing
- Sensor board input
- Serial communication (wired and wireless)
- Timers
- Watchdog timer

TOSSIM: Tiny OS SIMulator

- nesC can compile to native binaries.
- The resulting simulator imitates a group of Motes.
- TOSSIM emulates the Mote peripheral hardware.
- Java GUI (TinyViz) connects to the simulator binary over a socket.


TinyViz
File
Layout
Plugins
On/off
Sim Time: 22.292sec
Delay
0 ms
Clear
TinyViz
Set location
Sent radio packets
Neighborhood graph
Plot
ADC Readings
Set breakpoint
Calamari
Centroid
Contour points
Debug messages
Directed Graph
Radio links
Radio model
AutoRun logger (do not disable)
Distance scaling factor
5
Update
Out Edges
Empirical
Simulation resume

Conclusion

- Mote hardware
- nesC language
- TinyOS operating system