Embedded Systems Programming

Trends of Embedded Systems (Module 2)

Yann-Hang Lee

Arizona State University

yhlee@asu.edu

(480) 727-7507

Summer 2014

Trends of RT Embedded Systems Applications

- **Wide-spreading, distributed, connected, and heterogeneous**
- **Mission and safety critical**
- **High-end consumer products**
  - cell phone, HDTV, home network, PDA, GPS, appliances
- **Quality of the products**
  - portable/reusable, reliable/dependable, interoperable, predictable (schedulable), and secured
- **Software extensive**
- **A new S-class Mercedes-Benz**
  - over 20 million lines of code
  - nearly as many ECUs as the new Airbus A380 (excluding the plane's in-flight entertainment system).

Software Complexity

- Software in Smartphones = apps + android + Linux
- Linux kernel:

Language	Code	Comment	Blank	Percentage
C	15,213,100	3,277,849	2,919,083	94.5%
Assembly	474,013	99,107	77,428	2.9%
C++	238,689	107,606	49,138	1.7%
XML	85,470	520	7,079	0.4%
Make	39,575	11,901	10,859	0.3%
Totals	16,086,836	3,504,100	3,070,580	

- Android:

Language	Code	Comment	Blank	Percentage
C	4,790,691	1,541,950	1,021,643	42.5%
C++	2,922,477	896,141	624,506	25.7%
Java	1,530,288	770,935	321,929	15.2%
HTML	955,041	51,284	220,949	7.1%
XML	524,855	75,396	34,453	3.7%
Totals	11,431,145	3,546,030	2,333,226	

(Analyses from http://www.ohloh.net/)

Building Embedded Systems

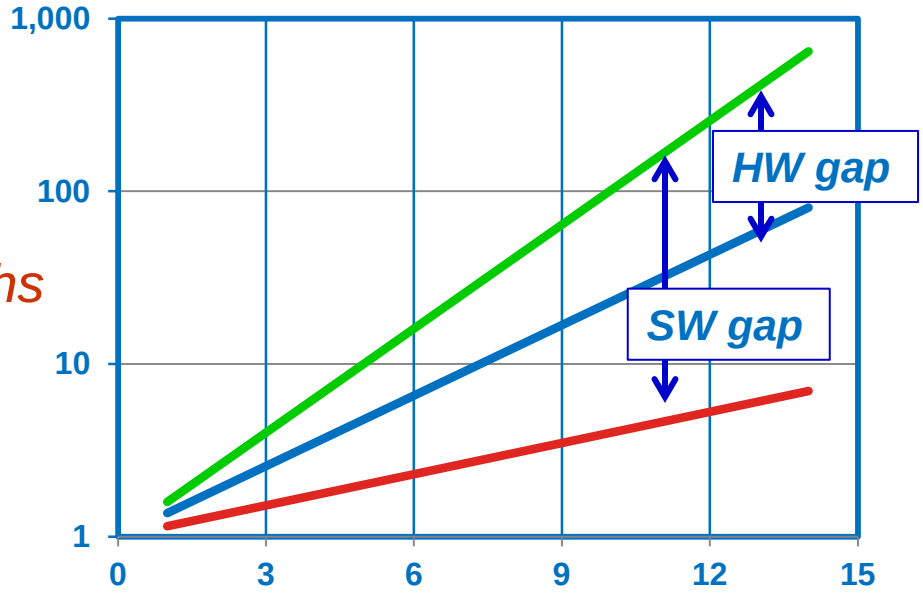
- **Advances in general-purpose computers**
  - PCs are powerful, cheap, and versatile
  - Information processing is ubiquitous
- **Thanks for the growth in productivity**
- **The design gaps**

Process technology *2x/18 months*

Hardware design *1.6x/18 months*

Software design *2x/5 years*

(International Technology Roadmap for Semiconductors, 2011)

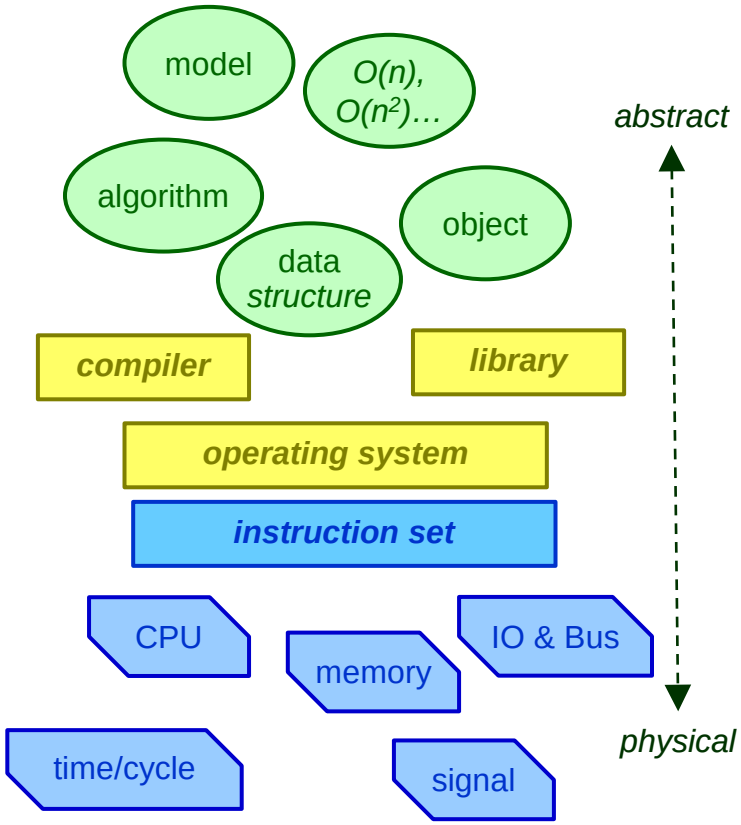


Embedded Software

- **Characteristics**
  - Concurrent, time dependent, and device/environment dependent
- **Embedded software development**
  - 80% programs in embedded system is with C/C++ and 15% in assembly
  - the same thing that has been done more than 30 years (Ada?)
- **Software complexities**
  - inherent and cannot be eliminated, i.e. algorithm, concurrency, etc.
  - accidental (due to technology or methods used), i.e. memory leaks
- **What can we do?**
  - ***abstraction (e.g. high-level languages, modeling)***
  - ***automation (e.g. compiler, code generation)***

Design for Performance and Predictability

- **A system vs. a piece of code**
- **Computer science is about abstraction, but resource is needed to run any piece of code**
  - *if (unlikely(!timer)) return -EINVAL;*
- **Benchmarking and profiling**
  - data measurement and interpretation
  - resource usage and hot spots
- **A good understanding on**
  - what hardware and software are doing together
  - space and time tradeoff
  - compiler optimization



Embedded Software Coding Practices

- **High-confidence software is difficult to validate**
- **Risk of failure is very high**
- **Software errors are systematic, not random. (i.e. *predictable and expected.)***
- **Case study in LLNL: 200 KLOC C/C++ program**
  - 285 defects found using Klocwork,
  - 56 critical (null pointer dereferences, buffer overruns)
  - 14 severe (use of freed memory)
  - 193 errors (leaks, uninitialized variables) and 22 other warnings
- **Bugs found by Coverity Scan (a source code analyzer)**

	linux 2.6	pHp 5.3	postgreSQl 9.1
Lines of code scanned	6,849,378	537,871	1,105,634
Defect Density (as of 12/31/11)	0.62	0.2	0.21
Number of outstanding defects (as of 12/31/11)	4,261	97	233
Number of defects fixed in 2011	1,283	210	78
Number of outstanding defects (as of 1/1/11)	3,457	14	247

(Coverity Scan: 2011 Open Source Integrity Report)

Coding Standards

- ***Two approaches***
  - *Use coding standards to streamline development*
  - *Use advanced static-analysis tools to find flaws early*
- ***MISRA-C/C++:* (Motor Industry Software Reliability Association)**
  - Misra C – first introduced in 1998, revised in 2004 and 2012
  - 143 rules checkable using static code analysis
  - makes development as predictable as possible, across projects or pieces of code, without errors of interpretation
  - Misra C++ in 2008 (mostly derived from JSF rules): 228 rules
- **Holzmann's "Power of Ten" and JPL coding standard**
- **Examples:**
  - check return value of non-void functions
  - name convention – dump_data_to_file()
  - no compilation warnings
  - no errors or warnings resulting from static code analyzers

Supplementary Slides

The Power of Ten

1. Restrict to simple control flow constructs.
2. Give all loops a fixed upper-bound.
3. Do not use dynamic memory allocation after initialization.
4. Limit functions to no more than 60 lines of text.
5. Use minimally two assertions per function on average.
6. Declare data objects at the smallest possible level of scope.
7. Check the return value of non-void functions, and check the validity of function parameters.
8. Limit the use of the preprocessor to file inclusion and simple macros.
9. Limit the use of pointers. Use no more than two levels of dereferencing per expression.
10. Compile with all warnings enabled, and use one or more source code analyzers.