

Systems Programming

Advanced Systems Programming (H)

Lecture 2

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow | This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Systems Programming

- What is systems programming?
- The state of the art
- Challenges and limitations
- Next steps in systems programming

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

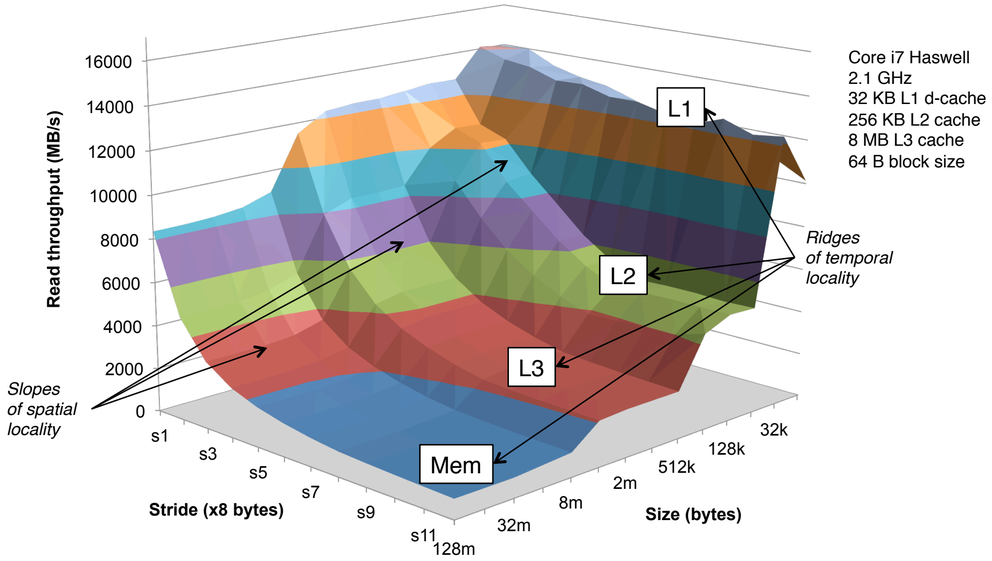
What is Systems Programming?

- Systems programs comprise infrastructure components: operating systems, device drivers, network protocols, and services
- They tend to be constrained by:
  - Memory management and data representation
  - I/O operations
  - Management of shared state

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Memory Management and Data Representation

- Predictability:
  - Timing must be bounded for real-time applications
  - Bounds on memory usage
- Data locality:
  - Cache line sharing impacts performance
  - Ensuring data is aligned and packed into cache lines for high performance
- Data representation:
  - Device drivers with fixed-layout control registers
  - Network protocol implementations must conform to specified packet layout
- Systems programming languages offer control of memory management and data representation – others languages lack such controls



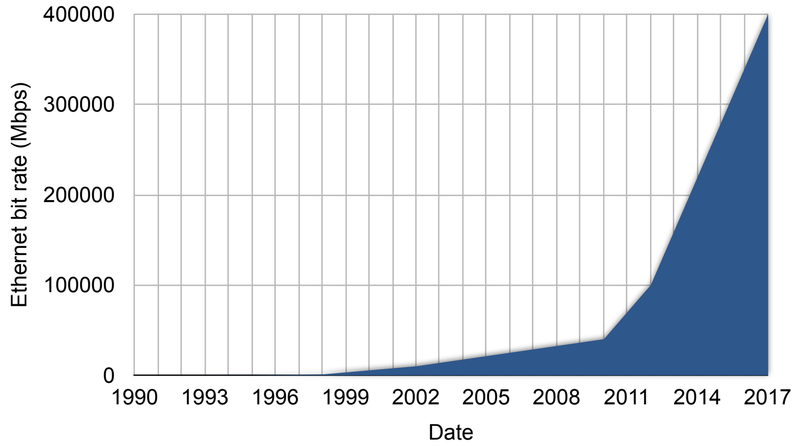
Smaller values of **stride** indicate data with better spatial locality; **size** is the total amount of data accessed

Source: Bryant & O'Hallaron, "Computer Systems: A Programmer's Perspective", 3rd Edition, Pearson, 2016, Fig. 6.41. http://csapp.cs.cmu.edu/3e/figures.html (Permission granted for lecture use with attribution)

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

I/O Operations

- Network performance increasingly a bottleneck:
  - Chart shows Ethernet bit rate over time – wireless links follow a similar curve
  - Closely tracking exponential growth over time – unlike CPU speed, where growth mostly stopped mid-2000s
  - MTU remains constant but packet rate increases; fewer cycles to process each packet



- SSD performance on a similar trend for file system access
- I/O performance of systems software critical to overall system performance
  - Device drivers, network protocol stack, file system

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

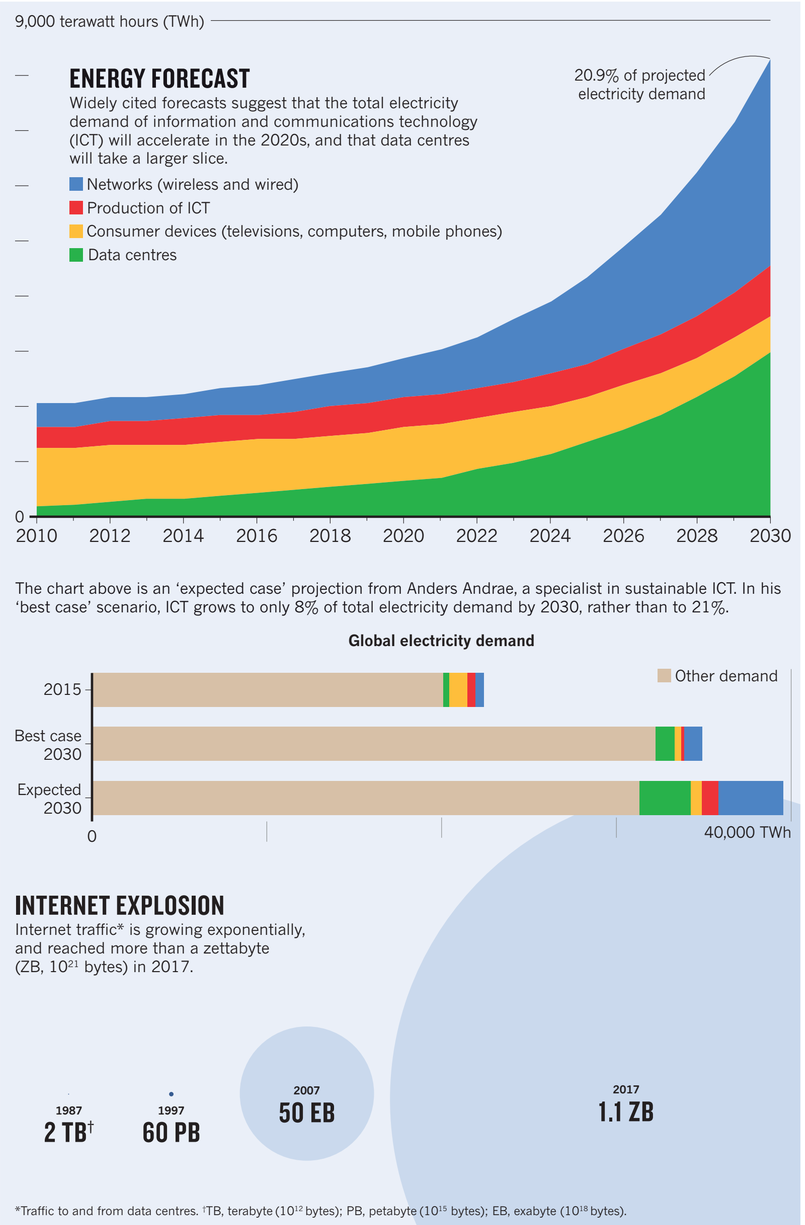
Management of Shared State

- Systems programs responsible for coordinating shared mutable state:
  - State shared across layers/between kernel and applications
    - Data structures for zero-copy networking
    - Header processing and state for the TCP stack
  - State shared between threads
    - Internal state of the kernel
    - File systems
    - Network code
  - Highly performance critical
- Systems programming languages allow sharing data between layers and/or threads – other languages disallow/discourage such sharing

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Performance

- Systems infrastructure performance fundamentally affects overall system and application performance
  - Mobile devices have limited battery life
  - Data centre efficiency and power consumption
- Systems components often the bottleneck in terms of overall performance and power efficiency
  - Simply because they're the basis on which the higher-level components depend



Source: N. Jones, "The Information Factories", Nature, v.561, p.163–166, Sep. 2018.DOI:10.1038/d41586-018-06610-y

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Systems Programming

- Systems programming languages offer low-level control of data representation, memory management, I/O, and sharing
- They are high-performance – concrete rather than high abstraction



Programming Language Challenges in Systems Codes

Why Systems Programmers Still Use C, and What to Do About It

Jonathan Shapiro, Ph.D.

Systems Research Laboratory
Department of Computer Science
Johns Hopkins University
shap@cs.jhu.edu

Abstract

There have been major advances in programming languages over the last 20 years. Given this, it seems appropriate to ask why systems programmers continue to largely ignore these languages. What are the deficiencies in the eyes of the systems programmers? How have the efforts of the programming language community been misdirected (from their perspective)? What can/should the PL community do address this?

As someone whose research straddles these areas, I was asked to give a talk at this year's PLOS workshop. What follows are my thoughts on this subject, which may or not represent those of other systems programmers.

1. Introduction

Modern programming languages such as ML [16] or Haskell [17] provide newer, stronger, and more expressive type systems than systems programming languages such as C [15, 13] or Ada [12]. Why have they been of so little interest to systems developers, and what can/should we do about it?

As the primary author of the EROS system [18] and its successor Coyotos [20], both of which are high-performance microkernels, it seems fair to characterize myself primarily as a hardcore systems programmer and security architect. However, there are skeletons in my closet. In the mid-1980s, my group at Bell Labs developed one of the first large commercial C++ applications — perhaps *the* first. My early involvement with C++ includes the first book on reusable C++ programming [21], which is either not well known or has been graciously disregarded by my colleagues.

In this audience I am tempted to plead for mercy on the grounds of youth and ignorance, but having been an active advocate of C++ for so long this entails a certain degree of *chutzpah*.[1] There is hope. Microkernel developers seem to have abandoned C++ in favor of C. The book is out of print in most countries, and no longer encourages deviant coding practices among susceptible young programmers.

A Word About BitC Brewer *et al.*'s cry that *Thirty Years is Long Enough* [6] resonates. It really *is* a bit disturbing that we are still trying to use a high-level assembly language created in the early 1970s for critical production code 35 years later. But Brewer's lament begs the question: why has no viable replacement for C emerged from the programming languages community? In trying to answer this, my group at Johns Hopkins has started work on a new programming language: BitC. In talking about this work, we have encountered a curious blindness from the PL community.

We are often asked "Why are you building BitC?" The tacit assumption seems to be that if there is nothing fundamentally new in the language it isn't interesting. The BitC goal isn't to invent a new language or any new language concepts. It is to integrate existing concepts with advances in prover technology, and reify them in a language that allows us to build stateful low-level systems codes that we can reason about in varying measure using automated tools. The feeling seems to be that everything we are doing is straightforward (read: uninteresting). Would that it were so.

Systems programming — and BitC — are fundamentally about engineering rather than programming languages. In the 1980s, when compiler writers still struggled with inadequate machine resources, engineering considerations were respected criteria for language and compiler design, and a sense of "transparency" was still regarded as important.[2] By the time I left the PL community in 1990, respect for engineering and pragmatics was fast fading, and today it is all but gone. The concrete syntax of Standard ML [16] and Haskell [17] are every bit as bad as C++. It is a curious measure of the programming language community that nobody cares. In our pursuit of type theory and semantics,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PLOS 2006, Oct. 22, 2006, San Jose, California, United States

Copyright © 2006 ACM 1-59593-577-0/10/2006. . . $5.00

[1] *Chutzpah* is best defined by example. Chutzpah is when a person murders both of their parents and then asks the court for mercy on the grounds that they are an orphan.

[2] By "transparent," I mean implementations in which the programmer has a relatively direct understanding of machine-level behavior.

J. Shapiro, "Programming language challenges in systems codes: why systems programmers still use C, and what to do about it", Workshop on Programming Languages and Operating Systems, San Jose, CA, October 2006. DOI:10.1145/1215995.1216004

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Systems Programming

- **What is systems programming?**
- The state of the art
- Challenges and limitations
- Next steps in systems programming

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow