CS263: Runtime Systems
Winter 2019 MW 9-10:15am Phelps 2510

http://www.cs.ucsb.edu/~cs263

TA: Vivek Pradhan
vivekpradhan@ucsb.edu

Prof. Chandra Krintz
Laboratory for Research on Adaptive Computing Environments (RACELab)
Computer Science Department
Harold Frank Hall (HFH) 2153
ckrintz@ucsb.edu
Past 40 Years in Technology Were Extraordinary

- Sustained exponential improvement in fundamental technologies
  - Exponential: gain in 2 years = all gains overall previous years

- 1974
  - computers used by business

- 2019
  - computer in every home
  - information everywhere, anytime
  - computer in every pocket
  - computer in every object
  - Integrated into the environment (buildings, streets, cars, infrastructure, stores, farms, ...)

Unprecedented Technology Change...
Moore’s Law

Double by shrinking transistors; smaller transistors switch faster
Reduces current used by each transistor while maintaining clock frequency & lowering fabrication cost
Moore’s law of doubling every 2 years (transistor count at constant cost) has not been true for years, but is still remarkable.
Moore’s Secret: Dennard Scaling

Design of Ion-Implanted MOSFET’s with Very Small Physical Dimensions

ROBERT H. DENNARD, MEMBER, IEEE, FRITZE H. GAENSSLEN, BWA-NIEN YU, MEMBER, IEEE, Y. LEO RIDEOUT, MEMBER, IEEE, ERNEST BASSOUS, AND ANDRE H. LEBLANC, MEMBER, IEEE

Abstract—This paper considers the design, fabrication, and characterization of very small MOSFET switching devices suitable for digital integrated circuits using dimensions of the order of 1 µm. The scaling law of MOSFET can be deduced from it. An improved small device structure is presented that uses ion implantation to provide shallow source and drain regions and a nonuniform substrate doping profile. One-dimensional models are used to predict the substrate doping profile and the corresponding threshold voltage versus source voltage characteristic. A two-dimensional current transport model is used to predict the relative degree of short-channel effects and the current-voltage characteristics. Physical parameters are extracted from the two-dimensional model and the device characteristics measured and compared with predicted values. The performance improvement expected from using these very small devices in highly integrated integrated circuits is projected.

Device or Circuit Parameter | Scaling Factor
--- | ---
Dimension, Tox, L, W | 1/k
Doping Concentration Na | k
Voltage (V) | 1/k
Current (I) | 1/k
Capacitance (eA/t ) | 1/k
Delay time/circuit (VC/I) | 1/k
Power dissipation/circuit (VI) | 1/k^2
Power density (VI/A) | 1

Historically, k ~ 1.4 (√2)

2x transistor count
40% faster
50% more efficient

[Denndard, Gaensslen, Yu, Rideout, Bassous, Leblanc, IEEE JSSC, 1974]

Power is proportional to the area (linear dimensions) of the transistor
Smaller transistors switch at higher speeds
As transistors shrank, so did voltage/current (while maintaining speed)
Dennard Scaling is Dead

Each transistor has a baseline of power (Watts)

Power density increases as transistors get smaller (Watts/cm²)

Clock frequency (4GHz) stops increasing even though more transistors are added b/c of baseline power – which increases power dissipation (heat)

Other problems: slow memory speeds and limited instruction level parallelism
That Was Fun!

What’s Next?
Traditional Sources of Improvement

Compilers

Computer Architecture

Semiconductors
New Opportunities

- Distributed Systems
- Reconfigurable Computing
- Software
State of Software

- Software is large, complex, and bloated
- Emphasis on programmer productivity, not software efficiency
- Performance improvement opportunities abound
  - Not long-term, secular trend like Moore’s Law, but still important
Large & Bloated – Ex: Linux Growth

Size (Linux 1.0 = 1)

- 1/31/1993
- 10/28/1995
- 7/24/1998
- 4/19/2001
- 1/14/2004
- 10/10/2006
- 7/6/2009
- 4/1/2012
- 12/27/2014

Lincoln Size
Moore's Law
Large & Bloated – Ex: Linux Complexity

![Number of Configuration Options Graph](image-url)
Large & Bloated – Ex: Windows Growth

Recommended Minimum Configuration (32 bit)
Transform date from SOAP message to Java object (IBM “Trade” benchmark)

268 calls
70 objects allocated
Computer Science is the Science of Abstraction
Object Bloat

Array holding 1 string

Hash set containing 3 strings

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Header</th>
<th>Pointer</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>40</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>22.0%</td>
<td>55.6%</td>
<td>11.1%</td>
<td>11.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Header</th>
<th>Pointer</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>156</td>
<td>40</td>
<td>88</td>
</tr>
<tr>
<td>22.0%</td>
<td>42.9%</td>
<td>11.0%</td>
<td>21.2%</td>
</tr>
</tbody>
</table>
What is Going On?

Developer Efficiency

Language Inefficiency

Frameworks

Systems

Abstraction
Developer Efficiency

- Time to market valued over execution efficiency
  - “First mover advantage” in competitive world
- Features more important than memory footprint or execution time
- High-level languages and rich libraries
  - Modularity and abstraction essential to develop complex applications
-Unmanaged language: statically compiled, architecture-dependent binary, streamlined runtime (C, C++, VB, asm, ObjC/Swift, Go)

-Managed: high-level, architecture-independent (portable) binary format, runtime performs translation (all others)

From: Tiobe 2019
Roughly equivalent to number of lines of code in the wild

<table>
<thead>
<tr>
<th>Jan 2019</th>
<th>Jan 2018</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>Java</td>
<td>16.904%</td>
<td>+2.69%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>C</td>
<td>13.337%</td>
<td>+2.30%</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>🟠</td>
<td>Python</td>
<td>8.294%</td>
<td>+3.62%</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>🟢</td>
<td>C++</td>
<td>8.158%</td>
<td>+2.55%</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>🟠</td>
<td>Visual Basic.NET</td>
<td>6.459%</td>
<td>+3.20%</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td>JavaScript</td>
<td>3.302%</td>
<td>-0.16%</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>🟢</td>
<td>C#</td>
<td>3.284%</td>
<td>-0.47%</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>🟠</td>
<td>PHP</td>
<td>2.680%</td>
<td>+0.15%</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>🟠</td>
<td>SQL</td>
<td>2.277%</td>
<td>+2.28%</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>🟠</td>
<td>Objective-C</td>
<td>1.781%</td>
<td>-0.08%</td>
</tr>
<tr>
<td>11</td>
<td>18</td>
<td>🟠</td>
<td>MATLAB</td>
<td>1.502%</td>
<td>-0.15%</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>🟢</td>
<td>R</td>
<td>1.331%</td>
<td>-1.22%</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>🟢</td>
<td>Perl</td>
<td>1.225%</td>
<td>-1.19%</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>🟠</td>
<td>Assembly language</td>
<td>1.196%</td>
<td>-0.86%</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>🟢</td>
<td>Swift</td>
<td>1.187%</td>
<td>-1.19%</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>🟠</td>
<td>Go</td>
<td>1.115%</td>
<td>-0.45%</td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>🟢</td>
<td>Delphi/Object Pascal</td>
<td>1.100%</td>
<td>-1.28%</td>
</tr>
<tr>
<td>18</td>
<td>11</td>
<td>🟢</td>
<td>Ruby</td>
<td>1.097%</td>
<td>-1.31%</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>🟠</td>
<td>PL/SQL</td>
<td>1.074%</td>
<td>-0.35%</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>🟢</td>
<td>Visual Basic</td>
<td>1.029%</td>
<td>-1.28%</td>
</tr>
</tbody>
</table>
BinaryInformatics 2018 Popularity:
1) Python
2) Java
3) Javascript
4) C++
5) PHP
6) C#
7) Perl

Quora 2018 Highest Paying:
1) Go. ($110K)
2) Objective-C
3) Python
4) Ruby/Rails
5) C#
6) Java
7) Swift. ($78K)

Coding Dojo Dec’17 blog post analysis of job postings on Indeed.com
Hello World!

Avg. Ticks (280ns)
Lower is better

Hello World

C Console
C Window
C# Console
C# Window
Language Implementations: Runtime Systems/VMs

- Collection of general-purpose components, libraries, frameworks
  - Java, .NET, WebSphere, ...
  - Productivity through reuse of high quality, high-level abstractions

- Flipside of generality is inefficiency
  - Appeal to widest audience by handling many scenarios
    - Bloated, complex software
    - Unused functionality “tax”
  - Not specialized to specific use
  - Cut/pasted solutions restrict true understanding and introduce bugs
Abstraction is Bad (For Performance)

- Abstraction captures functionality, obscures performance
  - Performance characteristic of implementation, not interface
  - Performance tuning destroys abstraction boundaries
- But, abstraction essential to construct large, complex systems
  - Cannot understand or predict performance of these systems
- Little work on specifying, analyzing, or modeling performance
  - Big-O notation hides too much
- Compilers and parallelism have not been able to solve the problem
Are Languages or Runtimes the Problem?

- Type safe, memory safe, modern programming languages
  - Not necessarily intrinsically expensive: MSR Singularity OS in C#

- But, some very popular languages have very poor implementations
  - Portability over performance (interpreter only)
    - Global interpreter lock in Python (cpython: https://wiki.python.org/moin/GlobalInterpreterLock)
  - Dynamic typing = Run-time checks + barrier to compiler optimization
  - Unsophisticated compilers in widely used implementations, if even available
    - Most use unoptimized interpreter for execution
  - Lack support for emerging software needs
    - High-performance, distributed computing/asynchrony, concurrency/parallelism, scalability, data-oriented computing, reliability, ...
## Matrix Multiply

<table>
<thead>
<tr>
<th>Method</th>
<th>ms</th>
<th>Python</th>
<th>Python (Jitted)</th>
<th>Java</th>
<th>In C</th>
<th>Transposed</th>
<th>tiled</th>
<th>Vectorized</th>
<th>BLAS MxM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9,298,440</td>
<td>6,145,070</td>
<td>540,891</td>
<td>348,749</td>
<td>19,564</td>
<td>17,206</td>
<td>12,887</td>
<td>6,607</td>
<td>1,680</td>
</tr>
<tr>
<td></td>
<td>1.6x</td>
<td>1.6x</td>
<td>1.2x</td>
<td>2x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11.4x</td>
<td>17.9x</td>
<td>1.4x</td>
<td>4x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.6x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27x</td>
<td>476x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>541x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>722x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1408x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5537x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles/OP</td>
<td>343</td>
<td>227</td>
<td>20</td>
<td>1/2</td>
<td>1/2</td>
<td>1/3</td>
<td>1/5</td>
<td>1/17</td>
<td></td>
</tr>
</tbody>
</table>
Increasing Performance of Software and Systems

- Managed runtime systems have become the norm
- Requires that we understand what they do, what they hide, how they work, and how they can be improved
  - **Performance implications of**
    - Object orientation
    - Typing
    - Garbage collection
- **How managed runtime system (VMs for high-level languages) work**
  - Interpretation
  - Compilation (dynamic and JIT)
  - Performance monitoring
  - Adaptive optimization
Managed runtime systems have become the norm

Requires that we understand what they do, what they hide, how they work, and how they can be improved

- Performance implications of
  - Object orientation
  - Typing
  - Garbage collection

- How managed runtime system (VMs for high-level languages) work
  - Interpretation
  - Compilation (dynamic and JIT)
  - Performance monitoring
  - Adaptive optimization
Managed runtime systems have become the norm

Requires that we understand what they do, what they hide, how they work, and how they can be improved

- **Performance implications of**
  - Object orientation
  - Typing
  - Garbage collection

- **How managed runtime system (VMs for high-level languages) work**
  - Interpretation
  - Compilation (dynamic and JIT)
  - Performance monitoring
  - Adaptive optimization
CS263 Evaluation

- 50% Homework assignments and quizzes in and out of class
  - Includes class participation; **no makeups** or date changes given
- CS190C: 50% Midterm exam
- CS263: 50% Project (2 person groups)
  - Vision statement
  - Weekly code commits (starting week 2)
    - Public github repo, documentation to build/regenerate
  - 10-12 minute in class presentation and demo the last week(s) of class
  - 5 page writeup: Problem, solution, evaluation
  - Project ideas posted on web page
- No final
Questions?

- Instructor: Chandra Krintz
  - HFH 2153
  - Office hours by appointment, skype (ckrintz), chat (ckrintz@ucsb.edu, ckrintz)

  - Lectures posted (slides and youtube)

- Class starts promptly at 9am (please be on time)
  - Will end between 10:15 and 10:45 depending on the topic
  - Assigned readings on website/schedule should be read by the class date indicated