James Larus
Microsoft Research

Workshop on Exploiting Concurrency Efficiently and Correctly
July 7, 2008

Spending Moore’s Dividend
Cramming more components onto integrated circuits

By Gordon E. Moore
Director, Research and Development Laboratories, Fairchild Semiconductor
Division of Fairchild Camera and Instrument Corp.

The future of integrated electronics is the future of electronics itself. The advantages of integration will bring about a proliferation of electronics, pushing this science into many new areas.

Integrated circuits will lead to such wonders as home computers—or even terminals connected to central computers—automated controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today.

But the biggest potential lies in the production of large systems. In telephone communications, integrated circuits in digital filters will separate channels on multiplex equipment. Integrated circuits will also switch telephone circuits and perform data processing.

Computers will be both more powerful and organized in completely different ways. For example, memories built of integrated electronics may be distributed throughout the machine instead of being concentrated in a central unit. In addition, the improved reliability made possible by integrated circuits will allow the construction of larger processing units. Machines similar to those in existence today will be built at lower costs and with faster turn-around.

Present and future

By integrated electronics, I mean all the various technologies which are referred to as microelectronics today as well as any additional ones that result in electronics functions supplied to the user as replaceable units. These technologies were first investigated in the late 1950’s. The object was to miniature electronics equipment to include increasing complex electronic functions in limited space with minimum weight. Several approaches evolved, including microassembly techniques for individual components, thin-film structures and semiconductor integrated circuits.

Each approach evolved rapidly and converged so that each borrowed techniques from another. Many researchers believe the way of the future is to be a combination of the various approaches.

The advantage of semiconductor integrated circuits are already using the improved characteristics of thin-film circuits by applying such films directly to an active semiconductor substrate. Those advocating a technology based upon films are developing sophisticated techniques for the attachment of active semiconductor devices to the passive film arrays.

Both approaches have worked well and are being used in equipment today.
Moore’s Law Enforced

1974
Intel® 8080 processor
- Initial clock speed: 2 MHz
- Number of transistors: 4,500
- Manufacturing technology: 6μ

2003
Quad-Core Intel® Xeon® processor (Penryn)
- Initial clock speed: >3 GHz
- Number of transistors: 820,000,000
- Manufacturing technology: 45nm

Intel x86 Family

Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)

18 Month Doubling
24 Month Doubling

Data from Kunle Olukotun, Herb Sutter, and Intel
More Than Processors

G. Moore, "No Exponential is Forever," Presentation ISSCC '03.
Moore’s Dividend

SPEC Integer Performance (single proc x86)
It’s Over

Industry

This talk

Diagram:
- Self Esteem vs. Time
  - Denial
  - Anger
  - Bargaining
  - Depression
  - Acceptance
Outline

- Where Moore’s Dividend was spent?
  - Software size
  - Software functionality
  - Programming complexity

- Is parallel computing a plausible successor?
- Parallel computing models
- Impact on computing
Myhrvold’s Laws, c. 1997

- 1st Law
  - Software is a gas!

- 2nd Law
  - Initial growth is rapid - like gas expanding (like browser)
  - Eventually, limited by hardware (like NT)
  - Bring any processor to its knees, just before the new model is out

- 3rd Law
  - That’s why people buy new hardware - economic motivator
  - That’s why chips get faster at same price, instead of cheaper
  - Will continue as long as there is opportunity for new software

- 4th Law
  - It’s impossible to have enough
  - New algorithms
  - New applications and new users
  - New notions of what is cool
Code Size Increases Less Than Processor Speed

Fine print: Wikipedia estimates of LoC. Does not measure code shipped to customers. SPEC normalized between SPEC95 and SPEC2000.
Where Moore’s Dividend Was Spent

- Processor performance consumed by changes in:
  - Software size
  - **Software functionality**
  - Programming complexity
IBM PC
Intel 8080 @ 4.77 Mhz
16-640 KB memory

PERQ (c 1981)
“3M” machine (1 MIPS, 1 MB, 1 megapixel)
Pervasive Improvements

- 1 bit display
- 25 lines of 80 chars (4K)
- Console
- stdio.h
- Single task, single address space
- No protection
- etc.

- 24 bit display
- 1280x1024 (64M)
- GUI
- Window system
- Multi-tasking, virtual address space
- Sophisticated security
- etc.
Recommended Windows Configurations

- Processor (SPECInt)
- Memory (MB)
- Disk (MB)
- Moore's Law

Graph showing recommendations for different Windows versions.
- Features monotonically increases
  - Office user uses 10% of features
  - Everyone uses a different 10% and 100% used
- Legacy compatibility sets floor

<table>
<thead>
<tr>
<th></th>
<th>Relative to WinXP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size Increase</td>
</tr>
<tr>
<td></td>
<td>Files</td>
</tr>
<tr>
<td>Win 2k3</td>
<td>1.43</td>
</tr>
<tr>
<td>Vista</td>
<td>1.80</td>
</tr>
</tbody>
</table>
Generalized Functionality

- Improvements are prevasive
  - Abstract model for many needs becomes less efficient
  - Generality precludes optimization
- E.g. print spooling
  - Security, notification – 1.5-4x
  - Color management, better text handling – 2x
  - Resolution
    - 300*300 dip @ 1bit → 600*600 @ 24bits (1MB → 96MB)
  - Memory latency and bandwidth
Where Moore’s Dividend Was Spent

- Processor performance consumed by changes in:
  - Software size
  - Software functionality
  - Programming complexity
Increased Abstraction

- High-level programming languages
  - Object-oriented (C++, Java, C#)
  - Interpreted (VB, Perl, Python, Ruby, etc.)
- Rich, abstract libraries
  - C++ Standard Template Library (STL)
  - Java class libraries
  - .NET platform
- Domain-specific language/systems
  - Ruby on Rails
Increased Use of C++ in Windows

Vista / Win 2003 (Mean per binary)
C/C# Minimum Application Size

<table>
<thead>
<tr>
<th></th>
<th>Working Set</th>
<th>Startup Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Debug</td>
<td>4.6x</td>
<td>18.4x</td>
</tr>
<tr>
<td>C# Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C Optimize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C# Optimize</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cost of Hello World

Avg. of 100 executions. Single proc, 3.4Ghz X86, Windows Vista SP1, Visual Studio 2008, Release build
Increased performance and memory size dulls programmers’ edge
  - Gates changed “READY” to “OK” in Altair Basic to save 5 bytes

Little understanding of processor performance models
  - Who really understands cache behavior?

Increasing reliance on compiler optimization
  - Uniformly “good” quality
  - Sometimes 10-100x off hand-written code

Performance is not an abstraction
  - Cuts across software abstractions
  - Cannot be understood locally
    - Think locally, act globally?
This is not bad!

- Increased abstraction improves productivity and enables richer functionality
- Without abstraction, modern software is beyond human conception
  - SAP Business Suite is 319 million LoC

<table>
<thead>
<tr>
<th>OS</th>
<th>MLoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Linux 7.1</td>
<td>30</td>
</tr>
<tr>
<td>Debian 3.0</td>
<td>104</td>
</tr>
<tr>
<td>Debian 4.0</td>
<td>283</td>
</tr>
<tr>
<td>Mac Os X 10.4</td>
<td>86</td>
</tr>
<tr>
<td>Windows XP</td>
<td>40</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>50</td>
</tr>
</tbody>
</table>

Source: Wikipedia.org
Software Development, c. 1950 – 2005

- Increased processor performance
- Larger, more feature-full software
- Larger development teams
- Higher-level languages and programming abstractions
- Slower programs
Software Development, RIP 2005?

- Slower programs
- Larger, more feature-full software
- Larger development teams
- High-level languages and programming abstractions
- Increased processor performance

X
Outline

- Where Moore’s Dividend was spent?
- Is parallel computing a plausible successor?
- New parallel computing models
- Impact on computing
Can Multicore Supplant Moore’s Dividend?

- Double number of cores instead of speed
- NO, at least without major innovation
  - Sequential code
  - Lack of parallel algorithms
  - Difficult programming
  - Few abstractions
Some Confusion Out There
Sequential Code

- Existing code is sequential
  - Series of decisions/actions
  - Difficult to change execution model

- Failed parallel compiler effort in ‘80s-‘90s
  - Compiler cannot change fundamental programming model

- Failed instruction-level parallelism in 90’s-00’s
  - Dynamic mechanisms cannot find more than 2–4x parallelism

- Artifact of problems & thinking
  - Not language specific ()

Harris & Singh [ICFP 07]
“In the context of sequential algorithms, it is standard practice to design more complex algorithms that outperform simpler ones (for example, by implementing a balanced tree instead of a list). For non-blocking algorithms, however, implementing more complex data structures has been prohibitively difficult.

[Herlihy, Luchangco, Moir, Scherer, PODC 2003]

Discussing a concurrent red-black tree (data structures 101).
Sadistic Homework (c. Maurice Herlihy)

Double-ended queue

No interference if ends “far enough” apart

enq(x)

enq(y)
Sadistic Homework

Double-ended queue

Interference OK if ends “close enough” together
Double-ended queue

Make sure suspended dequeueers awake as needed
You Try It ...

- One lock?
  - too conservative
- Locks at each end?
  - deadlock, too complicated, etc
- Waking blocked dequeuers?
  - harder that it looks
Solution

- Clean solution is a publishable result
  - [Michael & Scott, PODC 96]
- What kind of world are we moving to when solutions to such elementary problems are publishable?
Parallel programming is as difficult as sequential programming +
- Synchronization
- Data races
- Non-determinism
- Non-existent language and tools support
Few Parallel Abstractions

- Parallel programming models are low-level and machine-specific
  - Shared memory or message passing (~ hardware)
- Parallel programming constructs are “assembly language”
  - Thread == processor
  - Semaphore == atomic increment
  - Lock == compare & swap
- Performance models are machine-specific
- Parallel programs are low-level and machine-specific
  - Hard to port, reuse investments, develop market, or gain economies of scale
9.1. Why are GUIs Single-threaded?
In the old days, GUI applications were single-threaded and GUI events were processed from a "main event loop". Modern GUI frameworks use a model that is only slightly different: they create a dedicated event dispatch thread (EDT) for handling GUI events. Single-threaded GUI frameworks are not unique to Java; Qt, NextStep, MacOS Cocoa, X Windows, and many others are also single-threaded. This is not for lack of trying; there have been many attempts to write multithreaded GUI frameworks, but because of persistent problems with race conditions and deadlock, they all eventually arrived at the single-threaded event queue model in which a dedicated thread fetches events off a queue and dispatches them to application-defined event handlers. (AWT originally tried to support a greater degree of multithreaded access, and the decision to make Swing single-threaded was based largely on experience with AWT.)

Where Moore’s Dividend was spent?

Is parallel computing a plausible successor?

Parallel computing models
- Multiple paradigms
- TM
- Functional programming
- Message passing

Impact on computing
Multiple Programming Models

- Long-standing consensus on sequential programming model ("von Neumann")
- No consensus on parallel programming model
  - Data parallelism
  - Unstructured (thread) parallelism
  - Message passing
- Single application may use all three
- Language and tools need to support and integrate models
  - Education and training about when and how to use
Transactional Memory

- Lightweight transactions for mutual exclusion/atomicity in parallel programming
- Replace explicit synchronization with declarative specification
  - “This code should appear atomic” not “acquire this lock”
- All-or-nothing semantics helps ensure consistent program state
- Composable parallel abstraction
  - Need not be aware of abstraction’s locking discipline
Many Open Questions

- Fundamental TM issues not yet resolved
  - Strong and weak atomicity
  - IO
  - Legacy code
  - Open transactions
  - Long running transactions
- Old belief: programs written without side-effects are better suited to parallel execution
  - Backus’ 1977 Turing Lecture “Can Programming be Liberated from the von Neumann Style?”

- Functional languages carefully incorporate side-effects
  - “Mostly” functional languages, e.g., ML
  - IO Monads in Haskell

- Little evidence that functional programs expose more parallelism

- Considerable evidence that parallel functional programs are easier to write correctly
  - Less need for synchronization
  - Easier integration of high-level parallel constructs
Outline

- Where Moore’s Dividend was spent?
- Is parallel computing a plausible successor?
- Parallel computing models
- Impact on computing
Technology Drives Change

- Moore’s Law stacked the deck in favor of commodity multiprocessors
  - Specialized processors and mainframes faltered
  - Software industry was born
  - Hard to compete against 50%/yr improvement
- Parallelism will change computing landscape
- If existing applications and systems cannot use parallelism, new applications and systems will arise
  - Software + services
  - Mobile computing
Software as a Service (SaaS) / Software + Services
Embarrassingly Parallel Processing

- Even sequential applications become embarrassingly parallel when hosted
  - Few dependencies between users
- Moore’s Benefits accrue to platform owner
  - 2x processors ⇒
    - ½ servers (+ ½ power, space, cooling, etc.)
    - Or 2x service (same cost)
- Many implications for desktop platform, mobility, etc.
- Tradeoffs not entirely one-sided because of latency, bandwidth, privacy, off-line considerations; as well as capital investment, security, programming problems
Mobile is Parallel
Parallelism Reduces Energy

8-bit adder/compare
- 40MHz at 5V, area = 530 kµ²
- Base power \( P_{\text{ref}} \)

Two parallel interleaved adder/cmp units
- 20MHz at 2.9V, area = 1,800 kµ² (3.4x)
- Power = 0.36 \( P_{\text{ref}} \)

One pipelined adder/cmp unit
- 40MHz at 2.9V, area = 690 kµ² (1.3x)
- Power = 0.39 \( P_{\text{ref}} \)

Pipelined and parallel
- 20MHz at 2.0V, area = 1,961 kµ² (3.7x)
- Power = 0.2 \( P_{\text{ref}} \)

---

Heterogeneous Parallelism Really Reduces Energy

Energy Efficiency (MOPS/mW or OP/nJ)

Better

Microprocessors
DSP’s
Dedicated

3 orders of magnitude!

Single, General Purpose Processor
Multiple, Specialized Processors

Day-to-day challenges should not obscure possibility of major improvements in computing experience
  - PC (Mac, Linux, etc.) is not epitome of computing (I hope)

Focus on performance can eclipse more important qualities (reliability, robustness)
  - Wasteful to use half of processors to monitor other half?

Disruptive changes are opportunity to introduce “impossible” improvements
Plea to Architects

- Design machines to be programmed
  - Software is difficult enough to write correctly
- Don’t repeat relaxed consistency mistake

“There are some significant differences between loads and stores and interlocked operations. It’s pretty mind bending stuff. For example we have a case of memory reordering on all our platforms for a read after a write and you need explicit barriers if you do something needing this. There are also some global ordering issues that are beyond my comprehension currently.

“Declare you shared variables as volatile to prevent compiler reordering etc.

“Only use shared memory when the win is big as it’s a really big pain in the rear to get right.
Plea to Software Designers

- Design languages and write software to be correct by construction and verifiable
  - (Talk on Thursday at CAV)
Conclusions

- Parallelism is a software (not hardware) challenge
  - Breaks the cycle of innovation driving industry for 50 years
  - Difficult to use parallel hardware to solve typical computing problems
- No “silver bullets”
  - Programming language, compiler, library, tool changes are necessary, but not sufficient
  - Need to evolve together ⇒ large scale research ⇒ research $$
- Will drive major changes in software and computing industry
  - Benefits of parallel performance improvement will not accrue evenly
Views in this deck are mine, not Microsoft’s
  (Though, they should be)
Backup
Disks Too

Track, Areal, Linear Density Perspective

- Hard Disk Drive Products
- Circles = Server products
- Squares = Mobile products

- Track Density CGR = 50%
- Area Density CGR = 100%
- Linear Density CGR = 30%

Availability Year

90 92 94 96 98 2000 02 04
Si Is Destiny
Manycore