Concurrency Issues

Motivation, Problems, Directions

Reasons for Concurrency

multitasking

parallelism

coordination

performance

Moore's Law

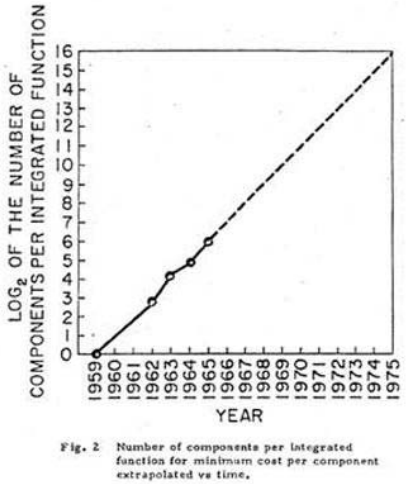
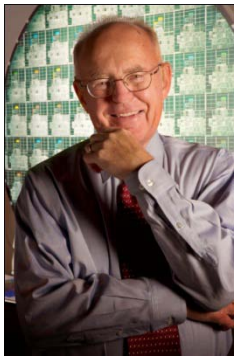
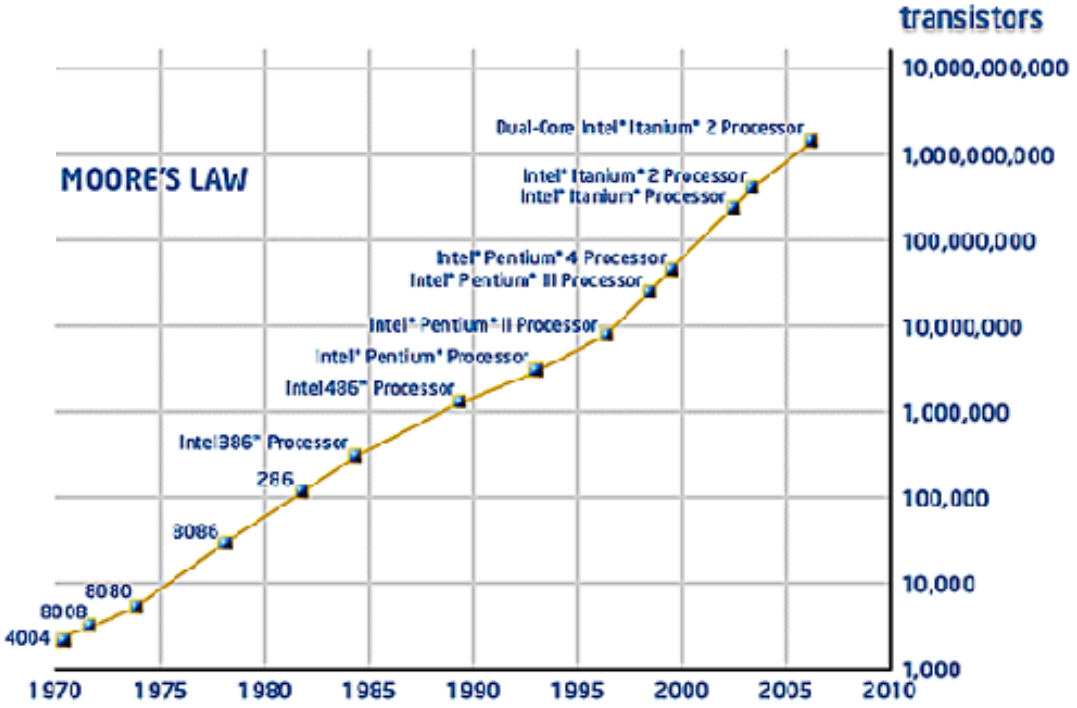


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

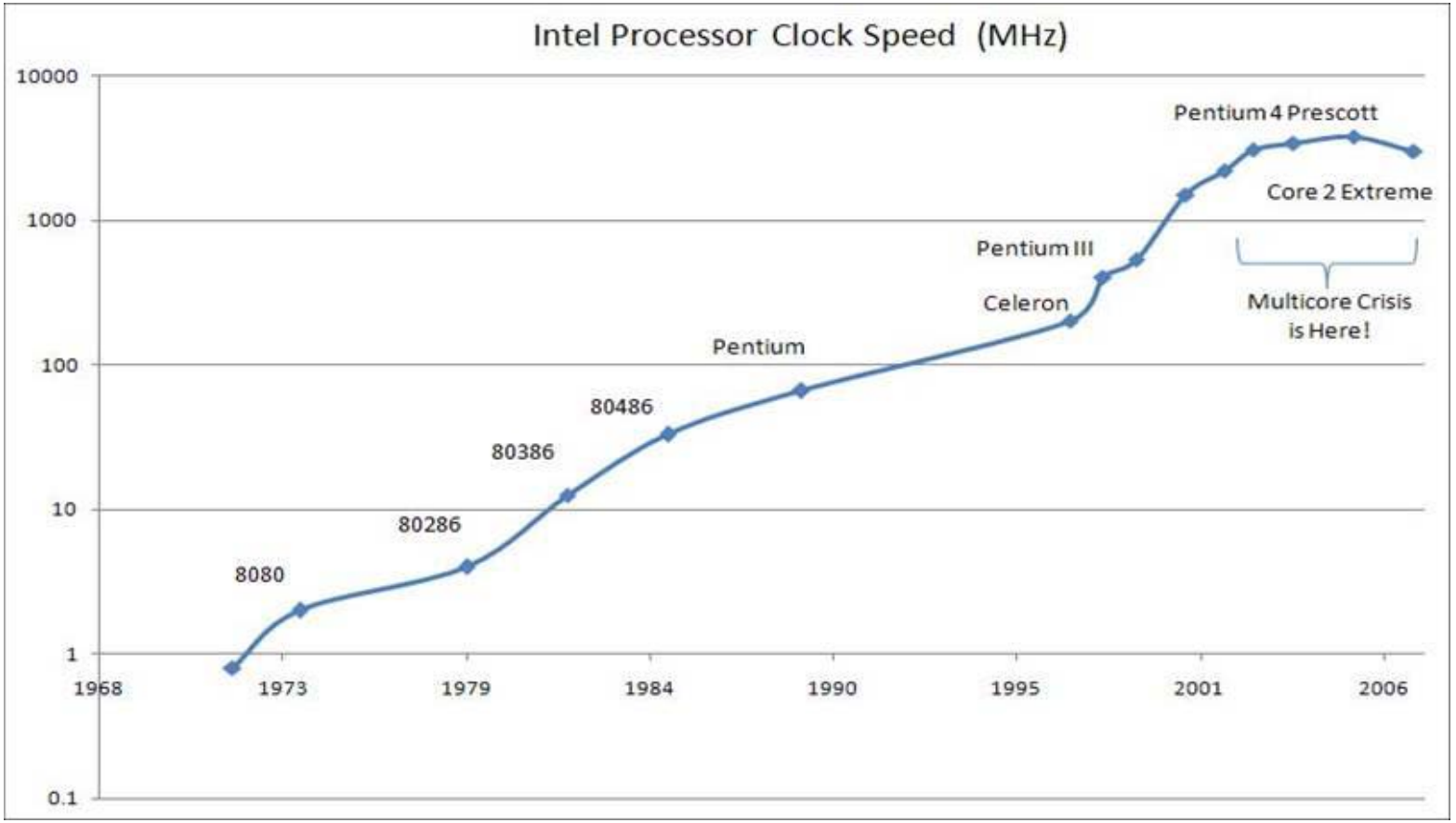


Gordon E. Moore,
Co-founder,
Intel Corporation.

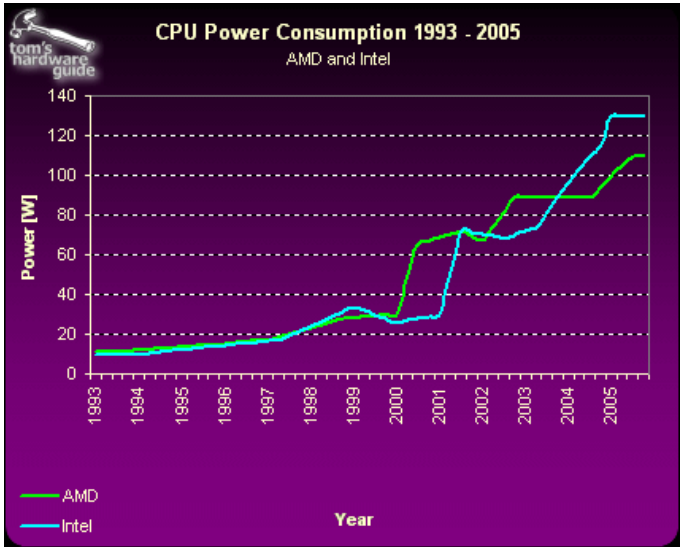
"Transistor density on integrated circuits doubles about every two years."

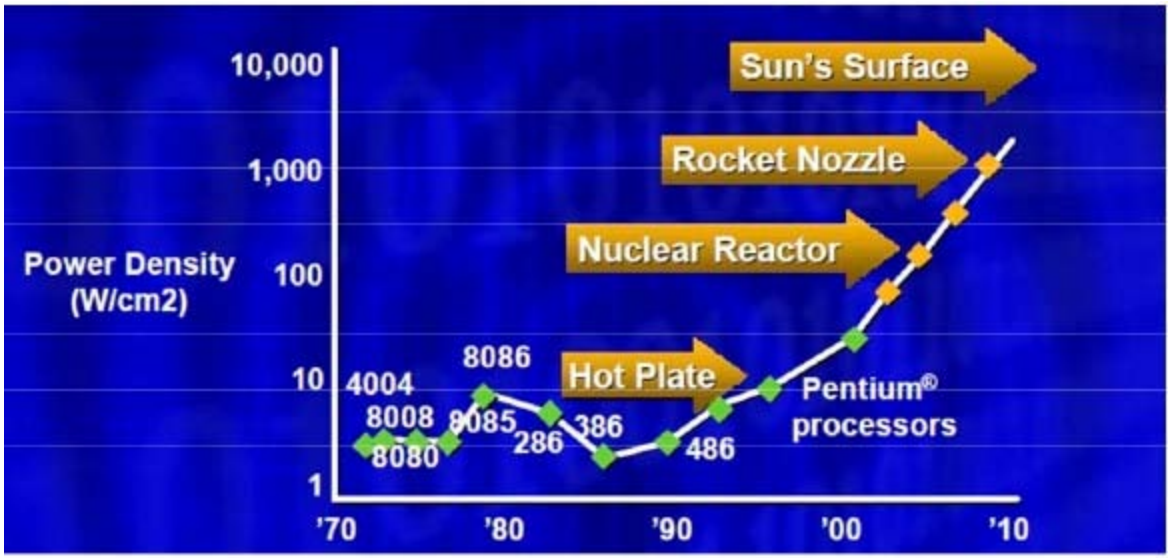


Hitting the wall...



Thermal Density



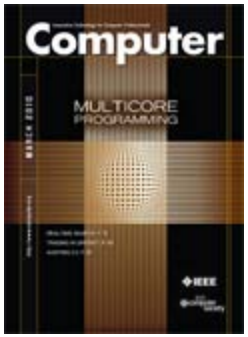


Source: Patrick Gelsinger, *Intel Developer's Forum*, Intel Corporation, 2004.

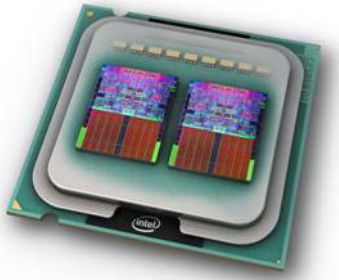
2005 (cooler alone)

1993 (CPU and cooler)

Rise of Multi-/Many- Core Technologies

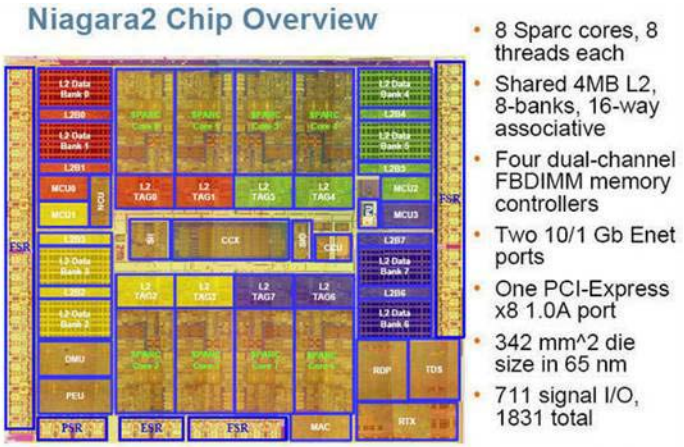


In the future, exponential growth in CPU performance will primarily be obtainable from more hardware threads and cores.

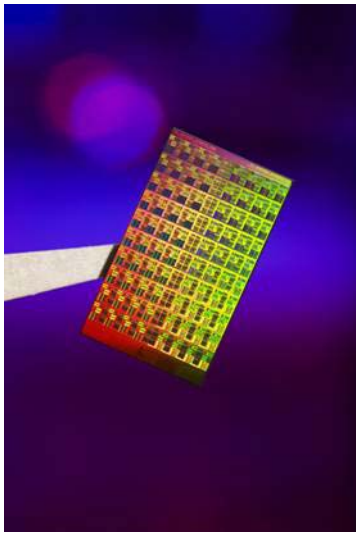


Intel: Quad Core

Sun: 8 core chip

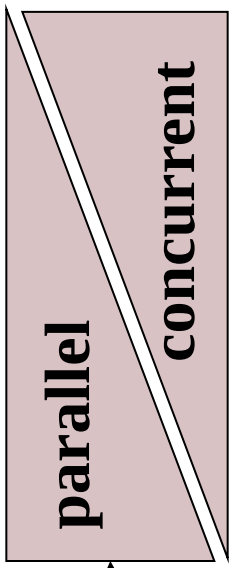


Intel: 80 core experimental system



Context

Support for concurrent and parallel programming



conform to application semantics

respect priorities of applications

no unnecessary blocking

fast context switch

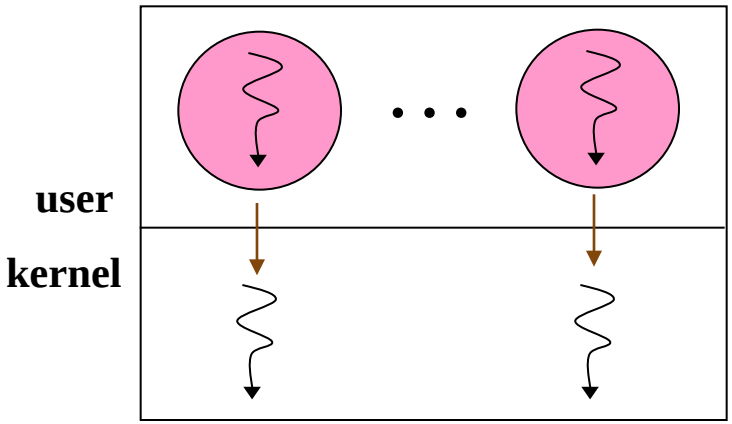
high processor utilization

relative importance

functionality

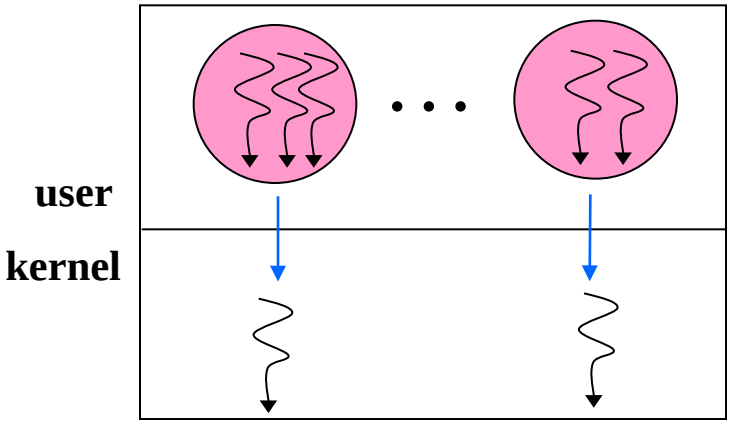
performance

"Heavyweight" Process Model



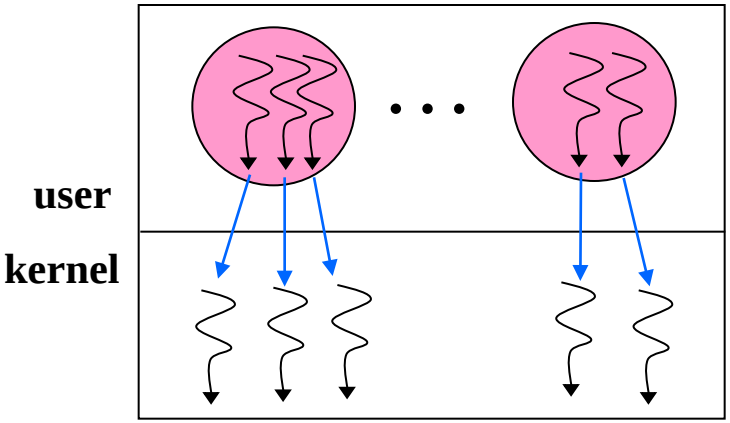
- simple, uni-threaded model
- security provided by address space boundaries
- high cost for context switch
- coarse granularity limits degree of concurrency

"Lightweight" (User-level) Threads



- thread semantics defined by application
- fast context switch time (within an order of magnitude of procedure call time)
- system scheduler unaware of user thread priorities
- unnecessary blocking (I/O, page faults, etc.)
- processor under-utilization

Kernel-level Threads

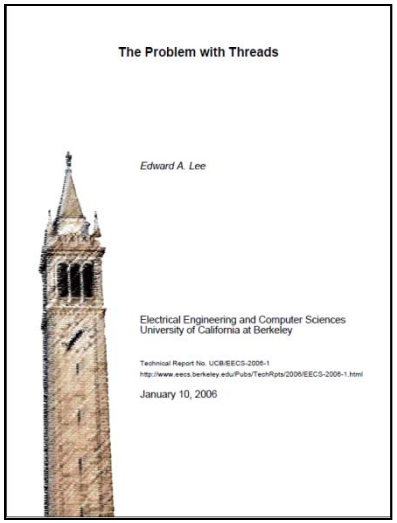


- thread semantics defined by system
- overhead incurred due to overly general implementation and cost of kernel traps for thread operations
- context switch time better than process switch time by an order of magnitude, but an order of magnitude worse than user-level threads
- system scheduler unaware of user thread state (e.g, in a critical region) leading to blocking and lower processor utilization

Threads are Bad

- Difficult to program
  - Synchronizing access to shared state
  - Deadlock
  - Hard to debug (race conditions, repeatability)
- Break abstractions
  - Modules must be designed “thread safe”
- Difficult to achieve good performance
  - simple locking lowers concurrency
  - context switching costs
- OS support inconsistent
  - semantics and tools vary across platforms/systems
- May not be right model
  - Window events do not map to threads but to events

Lee's Crticisms of Threads



- Threads are not composable
  - Inteference via shared resources
- Difficult to reason about threads
  - Everything can change between steps
- Threads are "wildly nondeterministic"
  - Requires careful "pruning" by programmer
- In practice, difficult to program correctly
  - Experience and examples

Ousterhout's conclusions

Why Threads Are A Bad Idea (for most purposes)

John Ousterhout
Sun Microsystems Laboratories

john.ousterhout@eng.sun.com
http://www.sunlabs.com/~ouster

Conclusions

- Concurrency is fundamentally hard; avoid whenever possible.
- Threads more powerful than events, but power is rarely needed.
- Threads much harder to program than events; for experts only.
- Use events as primary development tool (both GUIs and distributed systems).
- Use threads only for performance-critical kernels.

Why Threads Are A Bad Idea *September 28, 1995, slide 15*

Resilience of Threads

- Widely supported in mainstream operating systems
  - even if semantics differ
- Direct kernel/hardware support
  - via kernel threads and multi-core
  - shared address spaces
- Ability to pass complex data structures efficiently via pointers in shared memory
- Programmability
  - standard interfaces defined (e.g., POSIX)
  - construct in some languages (e.g., Java)
  - widely delolyed/understood (even if misused)

Concurrency Errors in Practice

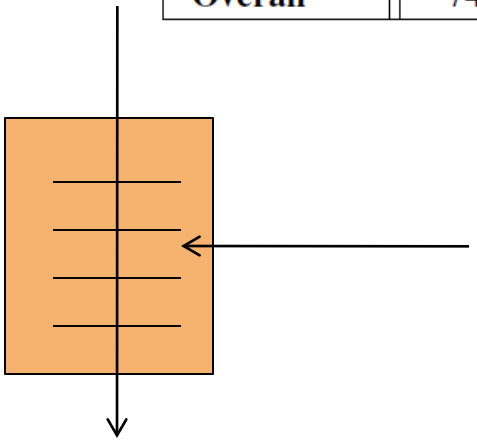
- Characterization study
  - Four large, mature, open-source systems
  - 105 randomly selected currency errors
  - Examined bug report, code, corrections
  - Classified bug patterns, manifestation, fix strategy

Application	Description	# of Bug Samples	
		Non-Deadlock	Deadlock
MySQL	Database Server	14	9
Apache	Web Server	13	4
Mozilla	Browser Suite	41	16
OpenOffice	Office Suite	6	2
Total		74	31

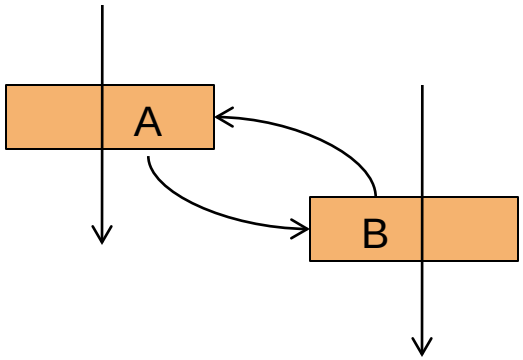
Concurrency Error Patterns

Finding (1): Most (72 out of 74) of the examined non-deadlock concurrency bugs are covered by two simple patters: *atomicity-violation* and *order-violation*.

Application	Total	Atomicity	Order	Other
MySQL	14	12	1	1
Apache	13	7	6	0
Mozilla	41	29	15	0
OpenOffice	6	3	2	1
Overall	74	51	24	2



atomicity-violation: interference with a sequence of steps intended to be performed as a unit



order-violation: failure to perform steps in the intended order

Concurrency Bug Manifestations

Finding (3): The manifestation of most (101 out of 105) examined concurrency bugs involves no more than two threads.

Non-deadlock concurrency bugs					
Application	Total	Env.	>2 threads	2 threads	1 thread
MySQL	14	1	1	12	0
Apache	13	0	0	13	0
Mozilla	41	1	0	40	0
OpenOffice	6	0	0	6	0
Overall	74	2	1	71	0

Deadlock concurrency bugs					
Application	Total	Env.	>2 threads	2 threads	1 thread
MySQL	9	0	0	5	4
Apache	4	0	0	4	0
Mozilla	16	0	1	14	1
OpenOffice	2	0	0	0	2
Overall	31	0	1	23	7

Other findings: most (66%) non-deadlock concurrency bugs involved only one variable and most (97%) of deadlock concurrency bugs involves at most two resources..

Concurrency Bug Fix Strategies

Finding (9): Adding or changing locks is *not* the major fix strategy.

Application	Total	COND	Switch	Design	Lock	Other
MySQL	14	2	0	5	4	3
Apache	13	4	2	3	4	0
Mozilla	41	13	8	9	9	2
OpenOffice	6	0	0	2	3	1
Overall	74	19	10	19	20	6

COND: Condition check
Switch: Code switch
Design: algorithm change
Lock: add or change lock

Another finding: transactional memory (TM) can help avoid many (41 or 105) concurrency bugs.

Solutions to thread problems

- New models of concurrent computation
  - MapReduce
    - Large-scale data
    - Highly distributed, massively parallel environment
  - Concurrent Collections (CnC)
    - General concurrent programming vehicle
    - Multicore architectures
- Thread-per-process models
  - Communicating Sequential Processes
  - Grace
  - Sammati