ELE 455/555
Computer System Engineering

Section 1 – Review and Foundations

Class 5 – Computer System Performance

Performance

Overview

- Eight Great Ideas in Computer Architecture
  - Design for Moore's Law
    - Integrated Circuit resources double every 18-24 months
    - More gates, parallelism, function specific blocks
  - Use Abstraction to Simplify Design
    - Show only those details required to get the job done
    - HW – transistor/gate/block/module level abstraction
    - SW – subroutine/object/program level abstraction
  - Make the Common Case Fast
    - Make the things you do the most fast
    - Manage the rare tasks

src: Patterson and Hennessy

Performance

Overview

- Eight Great Ideas in Computer Architecture – cont'd
  - Performance via Parallelism
    - Some tasks (but not all) can be separated and done in parallel leading to overall reduction in "time"
  - Performance via Pipelining
    - Subset of parallelism
    - Leverages a series of smaller tasks to accomplish a bigger goal
    - Utilizes resources across several goals
  - Performance via Prediction
    - In some situations the "likely" next step is known
    - By moving forward before you are sure – you can save time
    - Requires the penalty for being wrong to not be too large

Performance

Overview

- Eight Great Ideas in Computer Architecture – cont'd
  - Hierarchy of Memories
    - More memory is always desired
    - Large = Slow
    - Build a hierarchy of memory – fast but small → large and slow
  - Dependability via Redundancy
    - The underlying technology can fail
    - Build in circuit level, device level and system level redundancy
      - Extra row/columns in memories
      - Parity checking
      - 2 of 3 voting

Performance
Overview

- Five Classic Components of a Computer System
  - Input
    - Keyboard, Mouse, Touch Screen, Camera, Sensors, Microphone
  - Output
    - Display, Speakers, Vibrator
  - I/O
    - External Memory, Transceivers (BT, WLAN, Cellular, …),
  - Memory
    - Program and Data Storage
    - ROM, RAM, dRAM, Flash
    - Cache, Main, Removable
  - Datapath
    - Does the arithmetic and logical operations
  - Control
    - Manages the Datapath, Memory and I/O

The Datapath + Control is what we call a Processor or CPU (Central Processing Unit)

Performance

Overview

- Instruction Set Architecture (ISA)
  - Defines the HW/SW interface
  - Allows low level SW to run on multiple versions of HW (as long as they have the same ISA)
  - Typically includes:
    - List and format of Instructions
    - Register definitions
    - I/O definitions, locations and operation
  - ARM7, MIPS, AVR, HSC12, IA-64,…

Performance

Overview

- Instruction Set Architecture (ISA)
  - Accumulator
    - oldest version (not used anymore – but the term will show up occasionally
    - All arithmetic and logic operations are done with a single register – the accumulator
  - Stack
    - Became popular in the 60's and 70's – not used anymore
    - All arithmetic and logic operations were done using data push'd/pop'd on a stack
  - Memory – Memory
    - Also not used anymore
    - All arithmetic and logic operations were done directly to memory (slow)

Performance

Overview

- Instruction Set Architecture (ISA)
  - Register – Memory
    - Fixed set of registers added to the Datapath
    - Arithmetic and logic operations between registers and between registers and memory
    - HC11/HSC12, Intel 80386, AMD64
  - Register – Register
    - Also called Load-Store
    - Fixed set of registers added to the Datapath
    - Arithmetic and logic operations between registers
    - Only memory operations are Load and Store (to/from registers)
    - ARM, MIPS, Intel IA-64

Performance

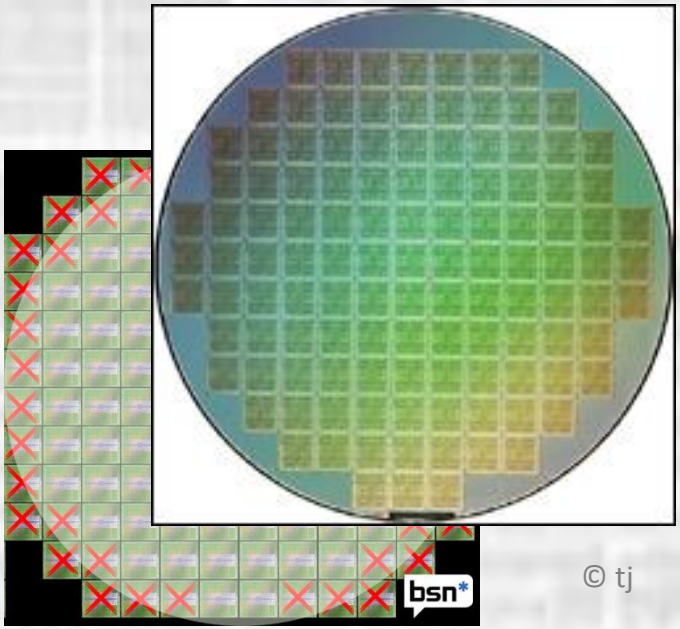
Cost

- Processor Cost Overview
  - Key Components building to part cost
    - Wafer Cost
      - 300mm wafers range from $5000/wafer (early) down to $1200/wafer (mature)
      - Cost reductions associated with process maturity
      - Wafer Yield = number of wafers that make it through the process with working transistors
    - Die Cost
      - Based on the number of full die that can fit on a wafer
    - Good Die Cost
      - Based on the number of die that are fully functional (may include redundancy)
    - Packaged Part Cost
      - Add the cost of package and packaging process
    - Good Packaged Part Cost
      - Based on the number of fully functional packaged parts (may include redundancy)
    - Margin
      - Additional $ to cover R&D, facilities, … AND profit

Performance

Cost

- Processor Cost Overview
  - Wafer Cost
    - 45nm, 300mm wafers ~ $2000/wafer
    - Typical “lot” of 25 wafers
    - Typical wafer yield of 95%
      - Losses are a combination of single wafers and whole lots
  - Die Cost
    - # of full die that will fit on a wafer
    - Various approaches to maximize die count
    - Die Cost = Wafer Cost / # of Die



Performance
Cost

- Processor Cost Overview

Apple intel Ad.mp4

Performance

Cost

- Processor Cost Overview
  - Good Die Cost
    - Based on the number of die that are fully functional
    - 2 primary yield components
      - Parametric Yield
      - Process Yield
    - Parametric Yield
      - Parts that fail to meet a performance measure
      - Typically max frequency or current drain
      - Can be mitigated by binning (have a fast version of the part and a slow version)
      - Typically 95% on digital parts
    - Process Yield
      - Dominated by defects in the manufacturing process
      - $Y = Y_o(1 + \frac{D_0 A}{\alpha})^{-\alpha}$ NB - negative binomial model
      - Y_0 – portion of area subject to defects (0.8-0.95)
      - D_0 – defect density (100defects/cm^2 (early) – 0.15defects/cm^2(mature))
      - A - die area (20mm^2 – 400mm^2)
      - α - cluster factor (10 – 20)

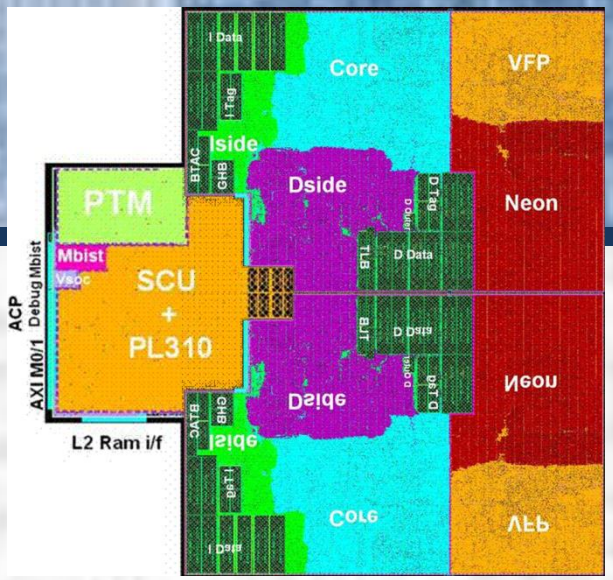
Performance
Cost

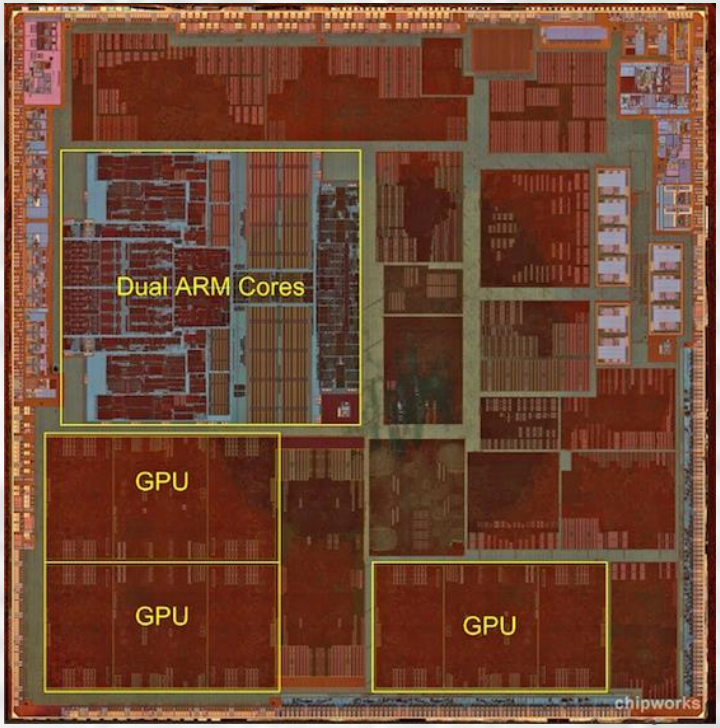
- Processor Cost Overview
  - Packaged Part Cost
    - Add the cost of package and packaging process
    - $0.20 for small simple packages
    - $2 - $4 for complex BGAs
    - $1 for POP
  - Good Packaged Part Cost
    - Based on the number of fully functional packaged parts
    - Package yield is typically 95% - 99+%
  - Margin
    - Additional $ to cover R&D, facilities, … AND profit
    - 20% for mature products
    - 50% for new products

Performance

Cost

- Processor Cost Example
  - Arm Cortex A9
    - 32kB I/D Cache
    - 26M transistors
    - 500mW @ 2GHz
    - 5mm^2 in 45nm process technology
  - Apple A5
    - Dual - Arm Cortex A9s
    - 45nm Samsung Process
    - Die size = 122mm^2
    - 1300 pin POP - BGA
  - Samsung 45nm process
    - 300mm wafers
    - D_0 = 0.25 defects / cm^2
    - α= 10
    - Y_0A5 = 0.95

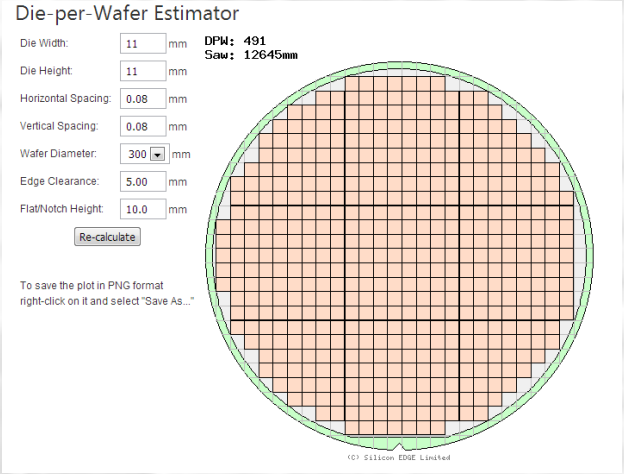




Performance

Cost

- Processor Cost Example
  - Wafer Cost
    - 300mm, 45nm → $2000 / wafer
    - Wafer yield – 95% → $2105 / wafer
  - Die Cost
    - 122mm2 → 491 die/wafer → $4.29 / die
  - Good Die Cost
    - $Y = Y_o(1 + \frac{D_0 A}{\alpha})^{-\alpha} = 0.95(1 + \frac{(0.25\ defects/cm^2)(122mm2 \times (\frac{1cm}{10mm})^2}{10})^{-10} = 0.703$
    - Defect driven die cost = $4.29/die / 0.703 = $6.10 / die
    - Parametric yield – 0.98 → $6.22 / die
  - Packaged Part Cost
    - 1300 pin – POP-BGA = $3 → 9.22 / part



Performance

Cost

- Processor Cost Example
  - Good Packaged Part Cost
    - 98% yield → 9.41 / part
  - Margin
    - If this was not an Apple design
      - Margin = 50% → Part cost = $18.82
    - Apple can cover the margin costs at the final product level
      - → Part Cost = $9.41
  - Gut feel cost before margin = $6.50

Performance
Analytics

- What do we mean by Performance
  - Two primary performance parameters
  - Speed
    - the following discussion focuses on speed
  - Power
    - For mobile devices power is critical
      - want your laptop to last 5-8 hours
      - need your cell phone to last all day
      - want your mp3 player to last all week
      - want your e-reader to last a month
    - For Servers
      - cooling can be a significant expense for server farms
      - cooling is an issue for individual server “closets”

Performance
Analytics

- Two primary speed measurements
  - Execution Time
    - How long it takes the processor to complete a task
    - Most familiar parameter to most of us
      - boot time
      - time to update the calculations on a large spreadsheet
      - time to read/write a file to disk
      - games – video updates and controller response time
    - In many cases the individual tasks are completed so fast we no longer perceive a delay
      - curser updates
      - directory traversal

Performance
Analytics

- Two primary speed measurements
  - Throughput (bandwidth)
    - How many things can be completed in a fixed amount of time
    - Differentiated from execution time when –
      - Tasks can be performed in parallel
      - Portions of a task are dependent on outside resources
        - A processor that can jump to the next task while waiting on a read from disk will have higher throughput than one that must stall during the wait
        - Both take the same execution time to perform the task requiring the read but the first will also accomplish additional tasks during the same time.

Performance
Analytics

- Two primary speed measurements
  - Improving (decreasing) execution time
    - Generally improves throughput
      - Each parallel or subtask completes faster
    - Exception: when there are not enough tasks to perform to fill the time
      - hurry-up and wait
  - Improving (increasing) throughput
    - Typically does not improve execution time
      - No one task completes any faster
    - Exception: when there are more tasks than can be completed in the allotted time queues will form
      - Assuming queue time is included in the execution time, improving throughput will improve execution time

Performance
Analytics

- Theoretical CPI or IPC
  - CPI – Clocks per Instruction
    - Number of clocks required to execute a single instruction
    - Varies by instruction
    - Varies by ISA
  - HCS12: 1to 4 clocks per instruction for most instructions
  - AVR: 1 clock per instruction for most instructions
  - Cortex A8: 0.5 clocks per instruction (dual issue)
  - Intel Core I7: 0.25 clocks per instruction (quad issue)
  - When the CPI gets below 1.0 – start to talk about IPC
    - Instructions per Clock
    - Cortex A8: IPC=2
    - Intel Core I7: IPC=4

Performance
Analytics

- Practical CPI or IPC
  - CPI – Clocks per Instruction
    - Number of clocks in a program or program segment divided by the number of instructions executed
  - Varies from theoretical
    - Cache misses
    - Instruction distribution
    - Branch prediction errors
  - Impacted by
    - Architecture
    - Program
    - Compiler
    - Memory – hierarchy, size, speed

Performance
Analytics

- Basic Calculations

$$CPU\ time\ (execuion\ time) = \frac{CPU\ clock\ cycles\ (for\ the\ task\ or\ program)}{Clock\ Rate}$$

Example 1: Task A requires 10,000 clock cycles on the ARM8 processor. How long will this task take using an 800MHz clock?

$$CPU\ time\ = \frac{10{,}000\ clock\ cycles}{800 \times 10^6\ {}^{cycles}/_{s}} = 12.5\mu s$$

Example 2: Task B takes 2us when running on your 2GHz laptop. You have the ability to modify the clock rate on your laptop. What clock rate should you use to achieve a 1.5us execution time?

$$Clock\ Cycles = 2E9\frac{cycles}{s} \times 2\mu s = 4000\ clocks$$

$$Clock\ Rate = \frac{4000\ clock\ cycles}{1.5\mu s} = 2.666\ GHz$$

Performance
Analytics

- Basic Calculations

$$CPU\ clock\ cycles = \text{Instructions for task} \times CPIav_{erage}$$

Example: A program requires 10,000 instructions on the ARM8 processor.
Assuming a CPI_{ave} = 1.2, how many clock cycles will this program take?

$$CPU\ clock\ cycles = 10{,}000 \text{ instructions} \times 1.2\ {}^{clocks}/_{instruction} = 12K\ clocks$$

Performance
Analytics

- Basic Calculations

$$CPU\ time = \frac{Instruction\ count \times CPI}{Clock\ Rate}$$

Example: Program A requires 10,000 instructions using the ARM8 processor at 1.5GHz and a CPI =1.2.
How long will it take this program to run?

$$CPU\ time = \frac{10{,}000\ instructions \times 1.2\ {clocks}/{inst}}{1.5 \times 10^9\ {clocks}/{s}} = 8\mu s$$

Performance
Analytics

- Basic Calculations
  - Amdahl's Law
    - The maximum expected improvement to an overall system when only part of the system is improved

$$CPU\ time\ after\ = \frac{Execution\ time\ affected\ by\ improvement}{Amount\ of\ improvement} + Execution\ time\ unaffected$$

Example: Cache misses represent 20% of the overall execution time of your program.
You have developed a new cache that cuts the miss penalty in half.
How much will your program speed up?

$$CPU\ time\ new\ = \frac{0.2Ys}{2} + 0.8Ys = 0.9Ys\ ,\ Y = current\ CPU\ time$$

speed up = 10%

Performance
Analytics

- Basic Calculations
  - MIPS
    - Measure of performance
    - Millions of Instructions / sec
    - Can only be used to compare processors of a common architecture
      - Different instructions → different # of instructions
      - Different CPIs → different clocks / instruction → different times
    - Can only be used to compare processors using the same program
      - Different programs on the same computer will lead to different MIPS measurements

Performance
Benchmarks

- Benchmarks
  - Groups of programs designed to:
    - Exercise the various components of the processor
    - Emulate software representative of a more random application
  - Operate at the program level
    - Can be used across various processor architectures
    - Account for clock rates, memory compliments, accelerators
  - Can be manipulated
    - Compilers can target code to the benchmark, making a given implementation appear faster than it would normally be.

Performance

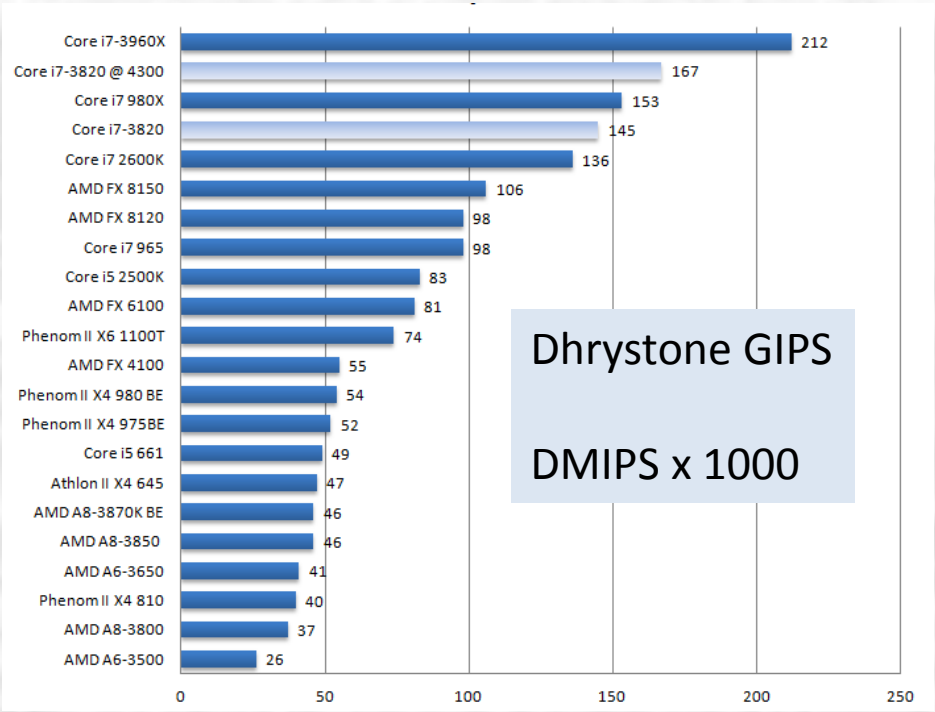
Benchmarks

- Dhrystone
  - Dhrystone
    - Number of iterations of a loop of the benchmark code per second
  - Dhrystone VAX MIPs (Dhrystone MIPS or DMIPS)
    - Compare the performance of a processor against the performance of a reference machine
    - The benchmark is calculated by measuring the number of Dhrystones per second for the system, and then dividing that figure by the number of Dhrystones per second achieved by the reference machine (VAX11/780)
    - VAX 11/780 could execute 1750 Dhrystones/s
      - 1DMIP = 1750 Dhrystones/s
  - So “100 DMIPS" means “100 Dhrystone VAX MIPS", which means 100 times faster than a VAX 11/780
  - Measuring DMIPS/MHz removes clock frequency confusion

Performance

Benchmarks

- Dhrystone
  - Limitations
    - No floating point operations
    - Easy to optimize compilers for the test



Notice – there is no clock frequency normalization here

Performance

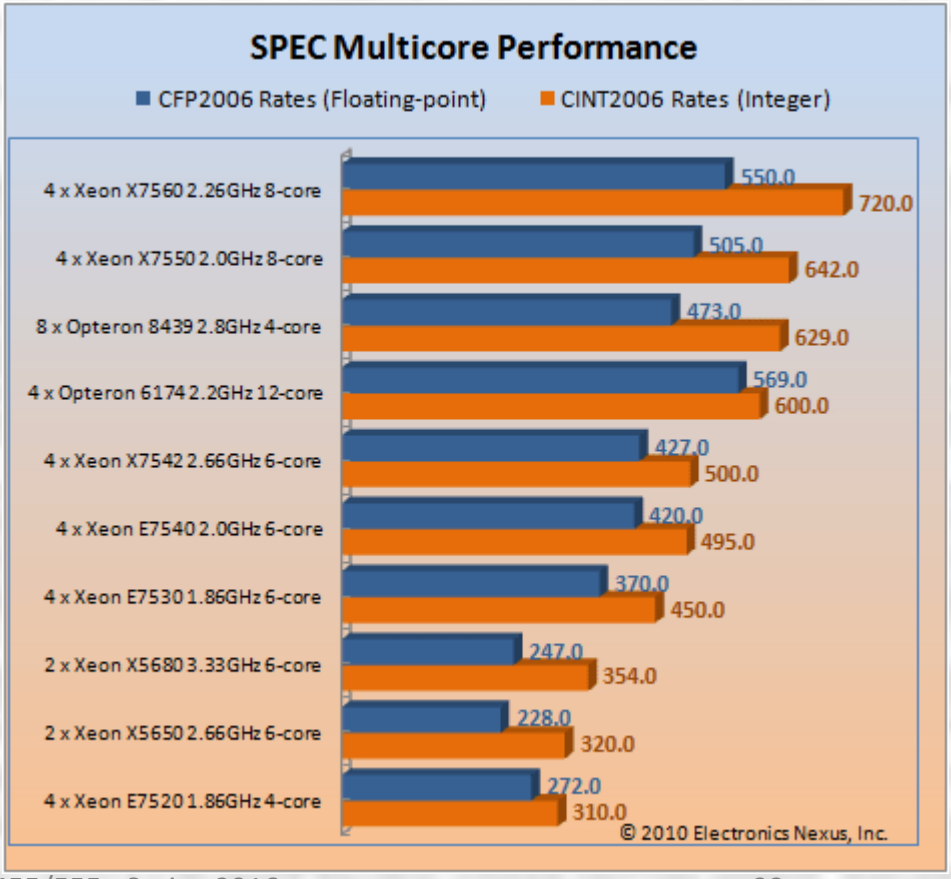
Benchmarks

- SPEC
  - Standard Performance Evaluation Corporation
  - SPEC CPU2006
    - relatively recent suite of programs (includes java)
    - CINT2006 – measure integer performance
    - CFP2006 – measures floating point performance
  - SPEC defines a baseline runtime for each component of the benchmark
  - The ratio of reference time / run time = SPECmarkxyz
  - The geometric mean of each of full set of tests is calculated
    - SPECint or SPECfp

Performance

Benchmarks

- SPEC
  - More common in larger processors



Performance

Benchmarks

- COREMARK
  - Embedded Microprocessor Benchmark Consortium (EEMBC)
  - System focused benchmarks
    - Android, browser, TCP/IP
  - Processor focused benchmarks
    - COREMARK
    - variations for automotive, entertainment, low power, multiprocessors
  - “Coremark” is a measure of the number of iterations of the benchmark code loop per second

Performance

Benchmarks

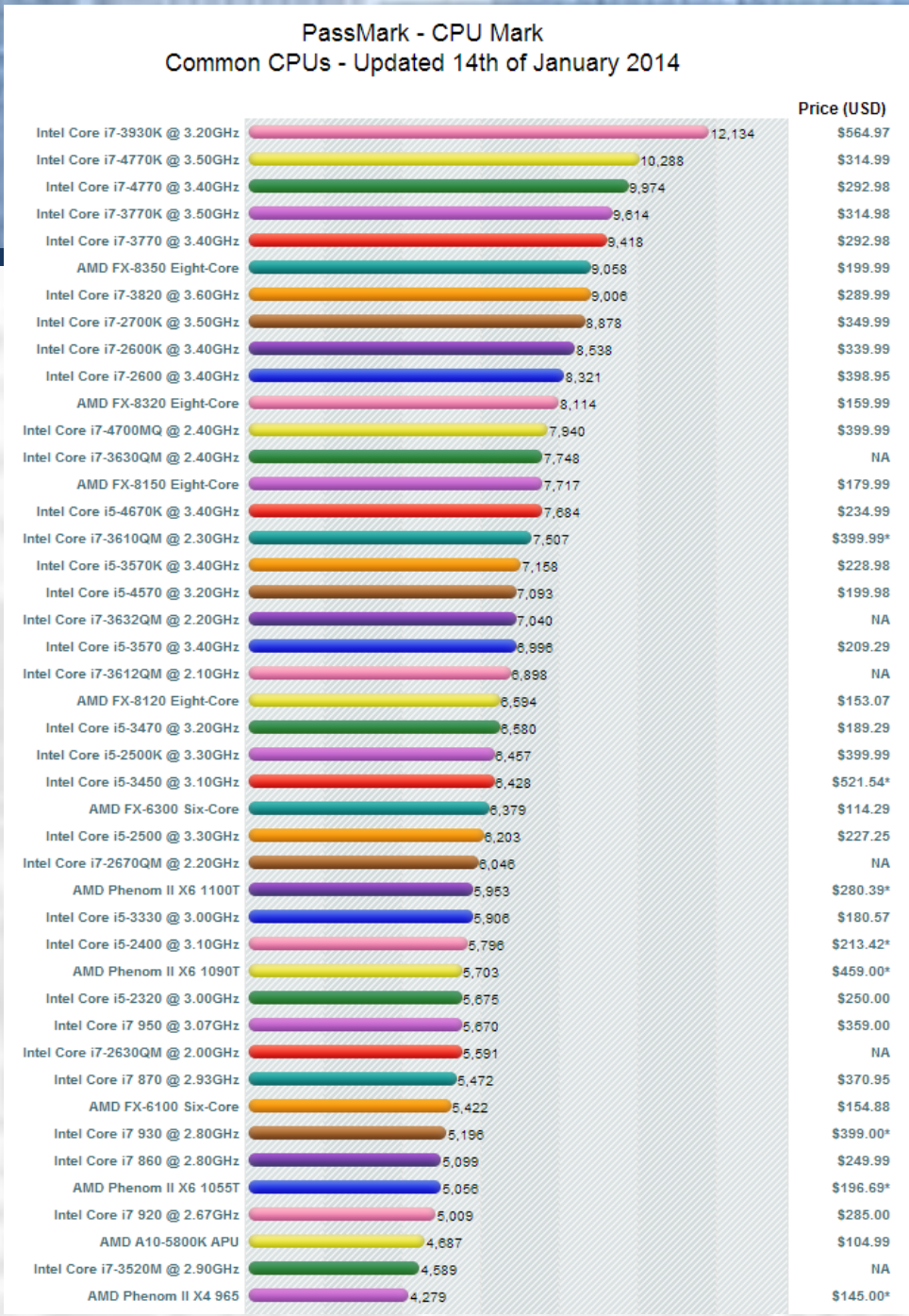
- COREMARK

ar	Processor	Compiler	Operating Speed in Mhz	CoreMark /MHz (1)	CoreMark (1)	CoreMark /Core (1)	EEMBC CERTIFIED	Parallel Execution	Comments	Date Submitted
	Intel Xeon E5-2650	GCC 4.4.6	2000	145.98	291957.48			32:PThreads	2	08/09/12
	Intel Xeon E5 2687W (2P, 16C, 32T)	GCC 4.5.3	3400	117.68	400116.70	25007.29		32:PThreads	1	07/24/13
	Intel Xeon L5640 ES (2) (Fujitsu RX300 S6)	GCC4.1.2 20080704 (Red Hat 4.1.2-46)	2266	52.33	118571.75			24:PThreads	comment	08/05/10
	Intel Core i7-3930K CPU	GCC4.4.6 20110731 (Red Hat 4.4.6-3)	3200	47.17	150962.39	25160.40		12:PThreads	2	05/18/12
	Intel Xeon L5640 ES (2) (Fujitsu RX300 S6)	GCC4.1.2 20080704 (Red Hat 4.1.2-46)	2266	40.87	92612.09			16:PThreads	comment	08/05/10
	Intel Xeon L5640 ES (2) (Fujitsu RX300 S6)	GCC4.1.2 20080704 (Red Hat 4.1.2-46)	2266	40.49	91743.12			12:PThreads	comment	08/05/10
	Intel i7-3612QE	Intel C++ 12.1	2100	39.97	83931.00	20982.75		8:PThreads	comment	01/30/13
	Intel(R) Core i7-3930K CPU	GCC4.4.6 20110731 (Red Hat 4.4.6-3)	3200	36.35	116324.16			12:PThreads	comment	05/18/12
	Intel Core i7-2760QM CPU @ 2.40GHz	GCC 4.5.3	2400	35.48	85151.68	21287.92		8:PThreads	comment	10/19/12
	Intel Core i7 2600	GCC 4.4.5	3392.236	29.35	99562.34			16:PThreads	comment	03/12/11
	Intel Core i5	GCC4.1.2 20080704 (Red Hat 4.1.2-46)	3100	16.57	51361.07			4:PThreads	comment	08/09/11
	ARM Cortex-A9 (Exynos4 Quad)	armcc 5.03-24	1400	15.89	22243.00	5560.75		4:PThreads	comment	04/16/13
	Intel Core i7 950	GCC3.4.4	3600	15.88	57163.27			8:PThreads	comment	05/13/10
	Intel Core i7 950	gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-48)	3060	15.80	48343.68			8:PThreads	comment	02/16/11
	Intel Core2 Quad Q6600	GCC4.5.2	2400	15.23	36553.96	9138.49		4:PThreads	1	03/22/11
	Intel X5450	Intel C++ - icc (ICC) 11.1 20090630	3000	15.22	45664.71			4:Fork	comment	08/10/09
	Intel Xeon E5504 ES (1) (Asus P6T WS)	GCC4.1.2 20080704 (Red Hat 4.1.2-46)	2000	15.15	30291.56			8:PThreads	comment	04/21/10
	Intel Core 2 Quad 6700	GCC4.4.1 20090725 (Red Hat	2667	14.70	39201.71	19600.86		4:PThreads	comment	08/31/09

Performance
Benchmarks

- PassMark

- **Integer Math Test**
  - The **Integer Math Test** aims to measure how fast the CPU can perform mathematical integer operations…provides a good indication of 'raw' CPU throughput.
- **Compression Test**
  - The **Compression Test** measures the speed that the CPU can compress blocks of data into smaller blocks of data without losing any of the original data… a function that is very common in software applications.
- **Prime Number Test**
  - The **Prime Number Test** aims to test how fast the CPU can search for Prime numbers …this algorithm uses loops and CPU operations that are common in computer software.
- **Encryption Test**
  - The **Encryption Test** encrypts blocks of random data using several different encryption techniques…also uses a large amount of binary data manipulation and CPU mathematical functions like 'to the power of'.
- **Floating Point Math Test**
  - The **Floating Point Math Test** performs the same operations as the Integer Math Test however with floating point numbers…these kinds of numbers are handled quite differently in the CPU compared to Integer numbers as well as being quite commonly used.
- **Multimedia Instructions**
  - The **Multimedia Instructions** measures the **SSE** capabilities of a CPU… enable blocks of data to be processed at higher speeds.
  - SSE stands for Streaming SIMD extensions.
- **String Sorting Test**
  - The **String Sorting Test** uses the **qSort algorithm** to see how fast the CPU can sort strings.
- **Physics Test**
  - The **Physics Test** uses the **Tokamak Physics Engine** to perform a benchmark of how fast the CPU can calculate the physics interactions of several hundred objects colliding.
- **Single Core Test**
  - The single core test only uses one CPU core and rates the computers performance under these conditions…many applications still only use one core so this is an important metric,



Performance

Benchmarks

- Caveats
  - Most benchmarks measure a combination of CPU/system and compiler performance
  - Significant results variation depending on cache size
    - If the benchmark fits in the cache → better results
  - All benchmarks simulate a fixed amount of code and situations
    - Most processors are subject to wide variations in code