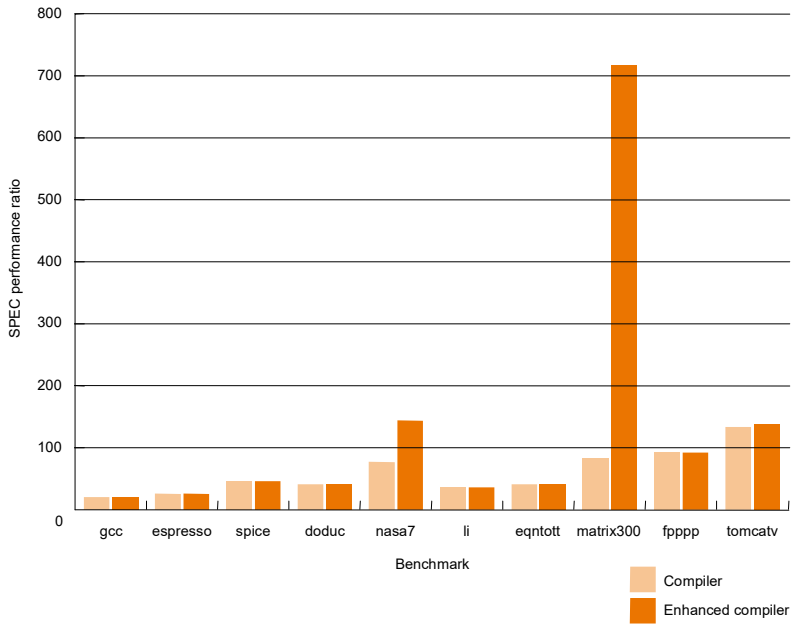
Chapter One, Again (2/2)

Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
    e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperation)
  - companies have agreed on a set of real program and inputs
  - can still be abused (Intel's "other" bug)
  - valuable indicator of performance (and compiler technology)

SPEC ‘89

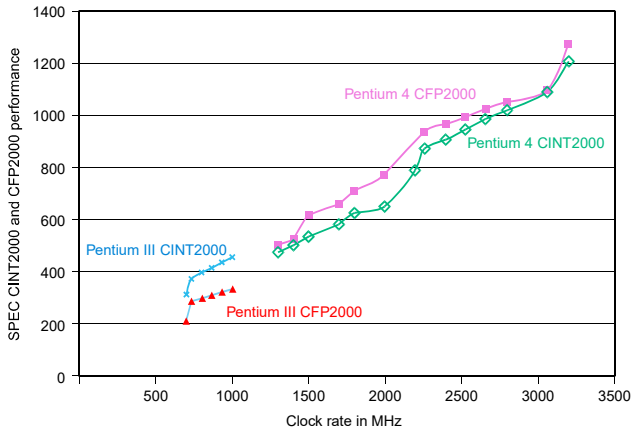
- **Compiler “enhancements” and performance**

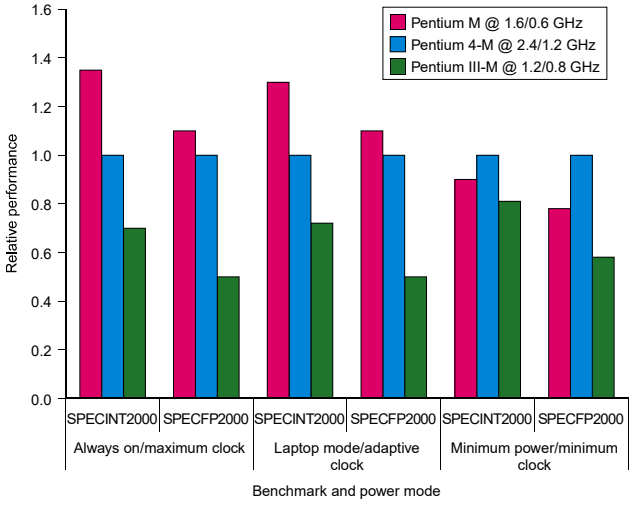


SPEC 2000

Does doubling the clock rate double the performance?

Can a machine with a slower clock rate have better performance?





SPEC CPU 2006 v1.2

CINT2006 Workloads	CFP2006 Workloads
CINT2006: C & C++ applications	**CFP2006: C, C++ & FORTRAN**
400.perlbench C PERL Programming Language	410.bwaves Fortran Fluid Dynamics
401.bzip2 C Compression	416.gamess Fortran Quantum Chemistry
403.gcc C C Compiler	433.milc C Physics: Quantum Chromodynamics
429.mcf C Combinatorial Optimization	434.zeusmp Fortran Physics/CFD
445.gobmk C Artificial Intelligence: go	435.gromacs C/Fortran Biochemistry/Molecular Dynamics
456.hmmer C Search Gene Sequence	436.cactusADM C/Fortran Physics/General Relativity
458.sjeng C Artificial Intelligence: chess	437.leslie3d Fortran Fluid Dynamics
462.libquantum C Physics: Quantum Computing	444.namd C++ Biology/Molecular Dynamics
464.h264ref C Video Compression	447.dealII C++ Finite Element Analysis
471.omnetpp C++ Discrete Event Simulation	450.soplex C++ Linear Programming, Optimization
473.astar C++ Path-finding Algorithms	453.povray C++ Image Ray-tracing
483.xalancbmk C++ XML Processing	454.calculix C/Fortran Structural Mechanics
	459.GemsFDTD Fortran Computational Electromagnetics
	465.tonto Fortran Quantum Chemistry
	470.lbm C Fluid Dynamics
	481.wrf C/Fortran Weather Prediction
	482.sphinx3 C Speech recognition

SPEC 2006 INT and FP Comparison

SPEC CPU2006 INT benchmark suite		SPEC CPU2006 FP benchmark suite	
Business applications	C, Perl programming language workloads	Chemistry and biology	Quantum chemistry
	"bzip" file compression workload		Quantum chromodynamics
	XML processing workload		Molecular dynamics
Scientific applications	Search gene sequencing	Physics	Fluid dynamics
	Quantum computing		General relativity
Problem solving applications	Video compression		Structural mechanics
	Combinatorial optimization		Electromagnetics
	Discrete event simulation		Weather prediction
	Artificial intelligence applications	Mathematics	Finite element analysis
			Linear programming
			Ray-tracing
			Speech recognition

SPEC: System Performance Evaluation Cooperative

- **The SPEC Bencmarks are the most widely used benchamrks for reporting workstation and PC performance.**

Round	CINT	CFP	Remarks
SPEC89	10		
SPEC92	6	14	
SPEC95	8	10	
SPEC2000	12	14	
SPEC2006	12	18	Memory system & compiler

- **Value reported is the SPEC ratio**
  - **CPU time of reference machine / CPU time of measured machine**

Other SPEC Benchmarks

- **JVM2008:**
  - **Measures performance of Java Virtual Machines**
- **SFS2008:**
  - **Measures performance of network file server (NFS) protocols**
- **Web2009:**
  - **Measures performance of World Wide Web applications**
- **HPC2002:**
  - **Measures performance of large, industrial applications**
- **APC (3DS MAX, Lightwave 3D, Maya, etc)**
  - **Meausre performance of graphics applications**

SPEC CPU Benchmark

- **Programs used to measure performance**
  - **Supposedly typical of actual workload**
- **Standard Performance Evaluation Corp (SPEC)**
  - **Develops benchmarks for CPU, I/O, Web, …**
- **SPEC CPU2006**
  - **Elapsed time to execute a selection of programs**
    - Negligible I/O, so focuses on CPU performance
  - **Normalize relative to reference machine**
  - **Summarize as geometric mean of performance ratios**
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

CINT2006 for AMD Opteron X4 2356

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Power Benchmarking

- **The power benchmarking of a computer is fundamentally the notion of determining how much energy the computer is consuming in order to accomplish some measure of work.**
- **The BDTI (Berkeley Design Technology Inc.), EEMBC (EDN Embedded Microprocessor Benchmark Consortium), and SPEC (Standard Performance Evaluation Corp) benchmark organizations support benchmark suites that highlight a processor's performance when performing application specific tasks.**
- **Researchers at BDTI and EEMBC are both working on how to extend their benchmark suites to measure and compare a processor's energy efficiency as opposed to power consumption when performing application specific tasks.**

Power Benchmark Strategy

- There are 3 primary areas of interest when benchmarking the characteristics of "low power" systems employing power management techniques to achieve low power goals.
  - First – actual power consumption of the system under typical user conditions, presumably under power management spectrum.
  - Second – system operability or usability under power management conditions. Its clear that one could achieve remarkable power characteristics at the cost of system performance and the response time.
  - Third – impact of power management techniques on system reliability.

- An appropriate benchmarking strategy for power managed systems must address these three areas in order to postulate an overall system figure merit, low power without sacrificing system operability or reliability. It would be one that characterizes the system power consumption while the system was carrying out some useful task.

Power Benchmarking

- The energy consumption of a system is therefore, the aggregate power dissipated by its components over time, at varying power states. In terms of time , it is the energy required to execute a given task to completion. This can be reflected at the system level as the summation of the energy requirements of each subtask and can be computed by the following expression:

$$Pt = \left(\sum^{m} Pn \right) \times Tc$$

where, Pt = "Task Energy" in watt hours, WHrs.

m = no. of power transitions occurring during the task

Pn = Segmented power dissipation during a given power state

Tc = "Task Cycle" time, time required to complete the task

SPEC Power Benchmark

- **Power consumption of server at different workload levels**
  - **Name: SPECpower_ssj2008**
  - **Performance: ssj_ops/sec**
  - **Power: Watts (Joules/sec)**

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

SPECpower_ssj2008 Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Types of Benchmarks

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks (e.g., SPEC benchmarks)	• less representative
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• does not measure memory system
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

Amdahl's Law

Execution Time After Improvement =
Execution Time Unaffected + Execution Time Affected / Amount of Improvement

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- **Example:**
  "Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

  100/4 = 20 + 80/x => x = 80/5 = 16

  How about making it 5 times faster?

  100/5 = 20 + 80/x => x = 80/0 (Impossible!)

- ***Principle: Make the common case fast***

Examples

- **Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?**

- **We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 2. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?**

Solutions

- **Speedup**
  - **T_{new} = 5 + 5/5 = 6 sec**
  - **n = T_{old}/T_{new} = 10/6 = 1.67 times faster**

- **Benchmark**
  - **100/2 = (100-x) + x/5 => 4x/5 = 50**
  - **x = 62.5**
  - **62.5% of instructions should be floating point instructions**

Remember

- Performance is specific to a particular program/s
  - Total execution time is a consistent summary of performance
- For a given architecture performance increases come from:
  - increases in clock rate (without adverse CPI effect)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count
- Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance
- You should not always believe everything you read! Read carefully!

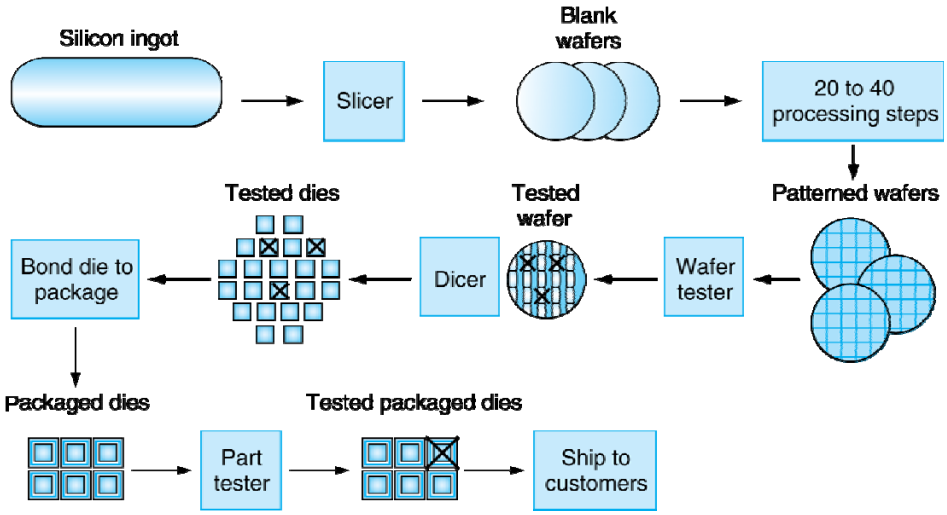
Fallacy: Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Semiconductor Technology

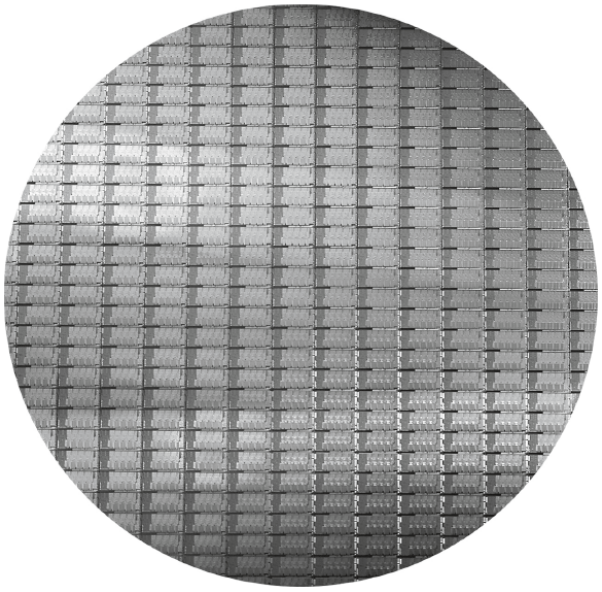
- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

Manufacturing ICs



- Yield: proportion of working dies per wafer

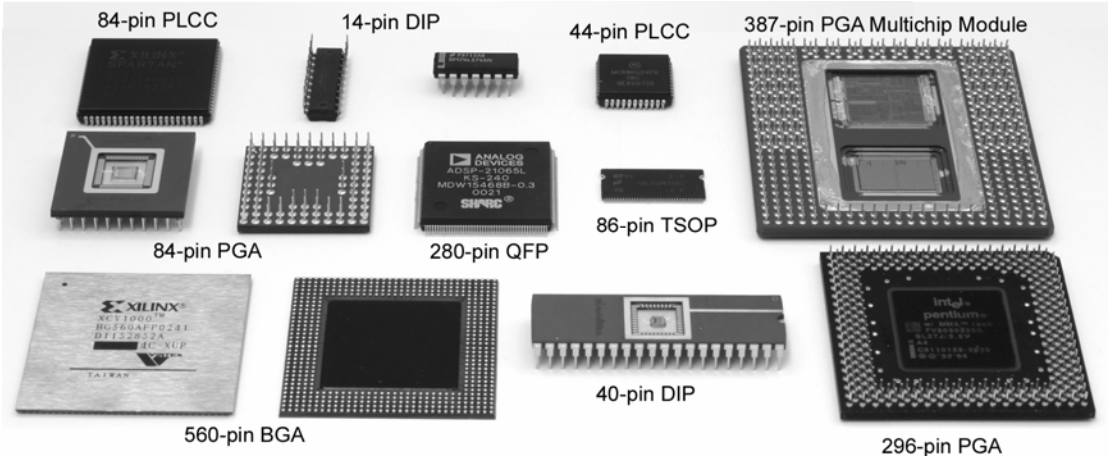
Intel Core i7 Wafer



- **300mm wafer, 280 chips, 32nm technology**
- **Each chip is 20.7 x 10.5 mm**

Package Types

- **Through-hole vs. surface mount**



Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

$$\text{Yield} = \frac{1}{(1+(\text{Defects per area} \times \text{Die area}/2))^2}$$

- **Nonlinear relation to area and defect rate**
  - **Wafer cost and area are fixed**
  - **Defect rate determined by manufacturing process**
  - **Die area determined by architecture and circuit design**