Compiler Optimizations and Benchmarking

Lecture 02

The University of
British Columbia

Overview

Last day…

- Flynn’s taxonomy – types of parallel architectures
- Dependencies – RAW forces sequence, avoid WAR+WAW using renaming
- Amdahl’s law – make the common case fast, infinite parallelism is doomed
- Gustafson’s law – if you want more parallelism, increase the problem size
- Readings – read upon compiler optimizations
- Mini-assignment – try out compiler optimizations on matrix multiply

Today…

- Optimizing your sequential program
  - Compiler optimizations
  - Measuring performance of a single program
  - Summarizing performance – **distill** set of performance results into 1 magic number
- Next week
  - A general introduction to parallel programming
  - Culler text, Chapter 2

Compiler Optimizations

Making your program go fast!

Compiler Optimizations

- Wikipedia article (lots of links)
  - Follow links for explanations and examples
- Bacon 76-page survey
  - General compiler optimizations for any CPU
- Padua 18-page Optimizations for Supercomputers
  - Describes dependences very well
  - Techniques to find parallelism in loops etc
- Assignment 0
  - Write matrix multiply
  - Try C compiler optimizations: none, -O, -O2, -O3
  - Rewrite matrix multiply with manual optimizations

Compiler Optimizations

- Gcc compiler options, -O is always safe

-O
-O1
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.
With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O turns on the following optimization flags:

-fauto-inc-dec
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fguess-branch-probability
-fif-conversion2
-fif-conversion
-fipa-pure-const
-fipa-profile
-fipa-reference
-fmerge-constants
-fsplit-wide-types
-ftree-bit-ccp
-ftree-builtin-call-dce
-ftree-ccp
-ftree-ch
-ftree-copyrename
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-sra
-ftree-pta
-ftree-ter
-funit-at-a-time

-O also turns on -fomit-frame-pointer on machines where doing so does not interfere with debugging.

Compiler Optimizations

- Gcc compiler options, -O2 is good

-O2
Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O, this option increases both compilation time and the performance of the generated code.
-O2 turns on all optimization flags specified by -O. It also turns on the following optimization flags:

-fthread-jumps
-falign-functions -falign-jumps
-falign-loops -falign-labels
-fcaller-saves
-fcrossjumping
-fcse-follow-jumps -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize
-fexpensive-optimizations
-fgcse -fgcse-lm
-finline-small-functions
-findirect-inlining
-fipa-sra
-foptimize-sibling-calls
-fpartial-inlining
-fpeephole2
-fregmove
-freorder-blocks -freorder-functions
-frerun-cse-after-loop
-fsched-interblock -fsched-spec
-fschedule-insns -fschedule-insns2
-fstrict-aliasing -fstrict-overflow
-ftree-switch-conversion
-ftree-pre
-ftree-vrp

Please note the warning under -fgcse about invoking -O2 on programs that use computed gotos.

Compiler Optimizations

- Gcc compiler options, -O3 is aggressive/unsafe

-O3
Optimize yet more. -O3 turns on all optimizations specified by -O2 and also turns on the -finline-functions, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload, -ftree-vectorize and -fipa-cp-clone options.

- Lots of gcc optimization options
  - http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
  - These change often -- check the version-specific manual

```
hydra2:~> gcc --version
```

gcc (Ubuntu 4.9.3-13ubuntu2) 4.9.3

- Ask gcc which optimizations are on/off:

```
hydra2:~> gcc -O3 -Q --help=optimizers
The following options control optimizations:
  -falign-jumps                                      [enabled]
  -falign-labels                                     [enabled]
  -falign-loops                                      [disabled]
  -fargument-alias                                   [enabled]
<etc>
```

Measuring Performance

Evaluating performance of
one and **many** programs!

Benchmarking – SPEC CPU2000
standard suite and others.

Two **Fundamental** Performance Concepts

1. **Throughput (aka bandwidth)**
   - Total amount of work done in a given time
     - Boeing 747
     - Laundromat with **many** washers & dryers
     - Important for computer data centres

2. **Response time (aka latency)**
   - Time from start to end of a given task
     - Concorde
     - One fast, modern laundry machine at home
     - Important for personal computers

Which is more important for this course?

- Mostly response time!
- Better response time → usually higher throughput
- Higher throughput != better response time

Evaluating Performance

... of **one** program!

(aka latency)

MIPS and others…

- MIPS and its relatives…
  - MIPS = Millions of Instructions Per Second
    - Aka Meaningless Indicator of Processor Speed
  - Relative MIPS (VAX 11/780 = 1 MIPS)
    - Someone got it wrong, it is actually 0.5 MIPS! (see Joel Emer)
  - Dhrystone MIPS
  - GIPS
  - MFLOPS, GFLOPS, TFLOPS
  - MOPS, GOPS
- What's wrong with these?

Detailed Performance Equation

$$\text{CPUTime} = \sum_{i} (\text{InstrCount}_i * \text{CPI}_i) * \text{CycleTime}$$

- CPUTime total run time of a program
  - Program contains mixture of instruction types

- InstrCount_i count of instructions of type ***i***
- CPI_i cycles per instruction of type ***i***
- CycleTime 1/frequency, eg 1/1GHz = 1ns

Example

InstrType$_i$	**CPI$_i$**
***i*=1**	1
***i*=2**	2
***i*=3**	3

Program:	**InstrCount$_i$**		
	***i*=1**	***i*=2**	***i*=3**
A	4	2	4
B	8	2	2

- Given
  - Same CPU with 3 different instruction types
  - CycleTime = 20ns
  - Program A, Program B with diff instruction mixes
- Find
  - Performance of A, B

Example (cont'd)

InstrType$_i$	**CPI$_i$**
***i*=1**	1
***i*=2**	2
***i*=3**	3

Program:	**InstrCount$_i$**		
	***i*=1**	***i*=2**	***i*=3**
A	4	2	4
B	8	2	2

CPUTime = $\sum_i$ (InstrCount$_i$ * CPI$_i$) * CycleTime

Program A (total 10 instructions):

= [(4*1) + (2*2) + (4*3)] * 20

= [4+4+12]*20

= 400 ns/program

Example (cont'd)

InstrType$_i$	**CPI$_i$**
***i*=1**	1
***i*=2**	2
***i*=3**	3

Program:	**InstrCount$_i$**		
	***i*=1**	***i*=2**	***i*=3**
A	4	2	4
B	8	2	2

CPUTime = $\sum_i$ (InstrCount$_i$ * CPI$_i$) * CycleTime

Program B (total 12 instructions):

= [(8*1) + (2*2) + (2*3)] * 20

= [8+4+6]*20

= 360 ns/program

Example (cont’d, final)

InstrType$_i$	**CPI$_i$**
***i*=1**	1
***i*=2**	2
***i*=3**	3

Program:	**InstrCount$_i$**		
	***i*=1**	***i*=2**	***i*=3**
A	4	2	4
B	8	2	2

Program A (total 10 instructions): 400ns
Program B (total 12 instructions): 360ns

Program B is faster!

(Intuitively, why should we expect this?)

Evaluating Performance

… of **many** programs!

First…. choose the programs!

Benchmarks

- You’re in engineering and you want to buy a computer….

… which one should you buy?

The **<u>fastest</u>** one of course!

- But you can’t trust:
  - MHz/GHz
  - MIPS
  - Or even your friend!

Benchmarks

Important: Choose A Realistic Workload

- Best solution: **Try it before you buy it!**
  - Run your program on the computer
  - Mix and match your most-frequently used programs
    - Quake 3, Google Chrome, Skype, Altera Quartus II (help!)
  - Called a ***workload***
  - Measure the CPUTime (fast stopwatch?)
  - Use **TOTAL CPUTime** as your metric (?)

- Problem: salesman doesn't want you to try it!
  - Find a new salesman!

Benchmarks

- Problem: your programs are not portable!
  - Different OS, different CPU architectures, …
  - Find a new OS ? A new program?
  - Write tiny version of your program to be portable
    - Toy Benchmarks: Sieve of Erastosthenes, Puzzle, Quicksort
    - Synthetic Benchmarks: Dhrystone (int), Whetstone (fp)
    - Computational Kernels: Livermore Loops, Linpack
- Problem: your program wasn’t tuned for this computer
  - Spend an eternity tuning it for each one?
  - Rely upon compiler?
- Benchmarking is problematic!

Benchmarks

- Compromise solution: **Let somebody else try it!**
  - They run their program
  - You trust them
  - Who? 3rd parties, eg: ZDnet? CNET? Tom's Hardware?
  - Who? Manufacturer, eg: IBM? Intel? Dell?
  - Who do ***you*** trust?

- SPEC: System Performance Evaluation Cooperative
  - Collect and distribute set of programs: a ***benchmark suite***
  - Benchmark suite represents some typical workload (who's?)
  - Founded by industry (Apollo/HP, DEC, MIPS, and Sun)
  - Note: this is a bit like buying a car from an auto-mechanic…
    "of course, it runs just fine"

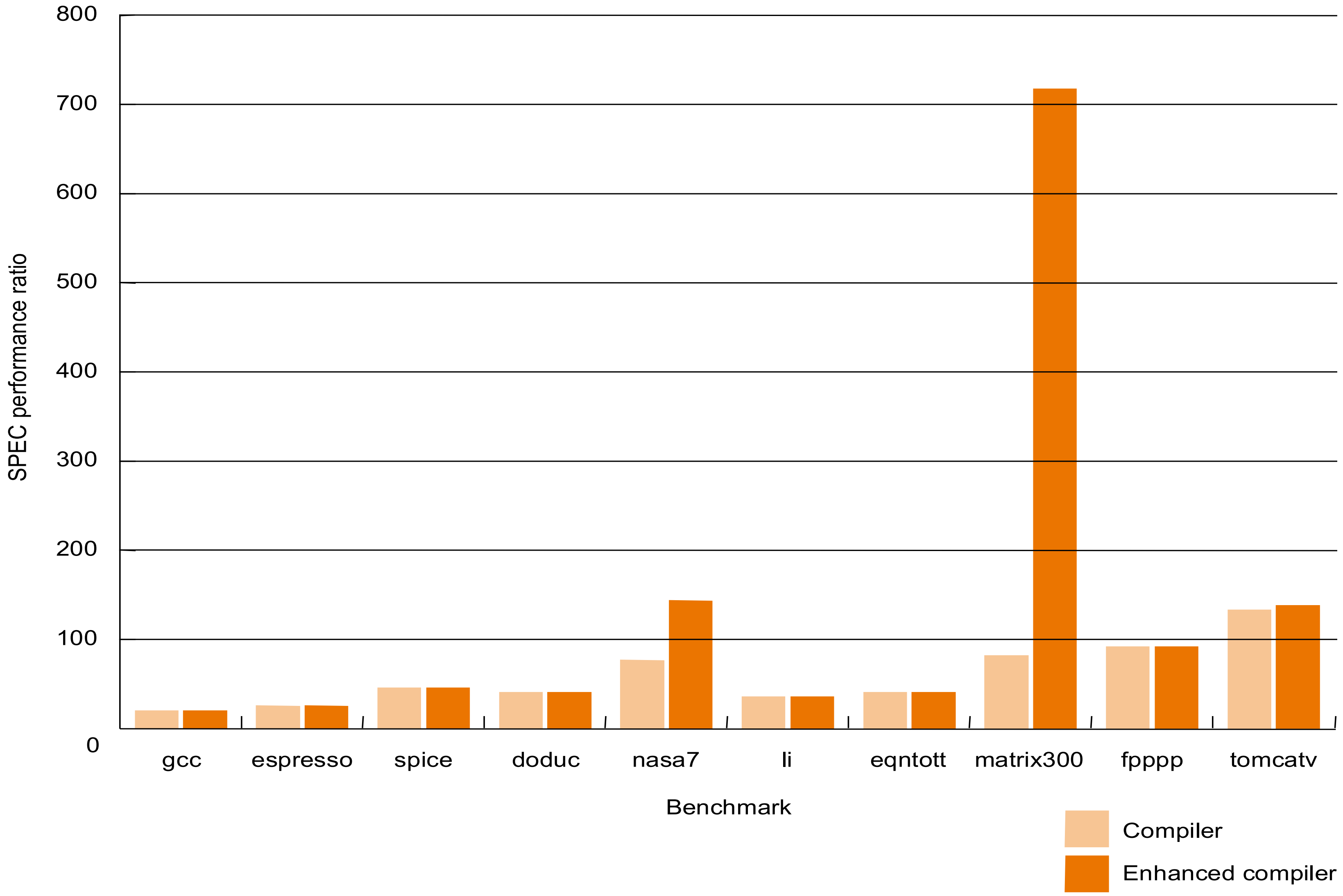
SPEC Benchmarks

- SPEC Benchmarks
  - Measure speed of a ***<u>system</u>***
  - System = CPU + memory subsystem + compiler + OS
  - Improve any of these => improved performance
  - Valuable indicator of ***<u>system</u>*** performance!

- SPEC Rules
  - Strict data reporting and collecting standards
  - Rules of gameplay (benchmarks can be abused!)
  - SPEC is the best we've got (so far…)!
  - Only possible in last 10-15 years due to portable software (C)

- SPEC Periodically Updates Benchmarks…
  - 1989, 1992, 1995, 2000, 2006
  - Eventually computers get too fast!
  - Or nature of the workload changes!
  - Or compilers get too smart!

Benchmarks – Compiler Result!



SPEC Benchmark Evolution

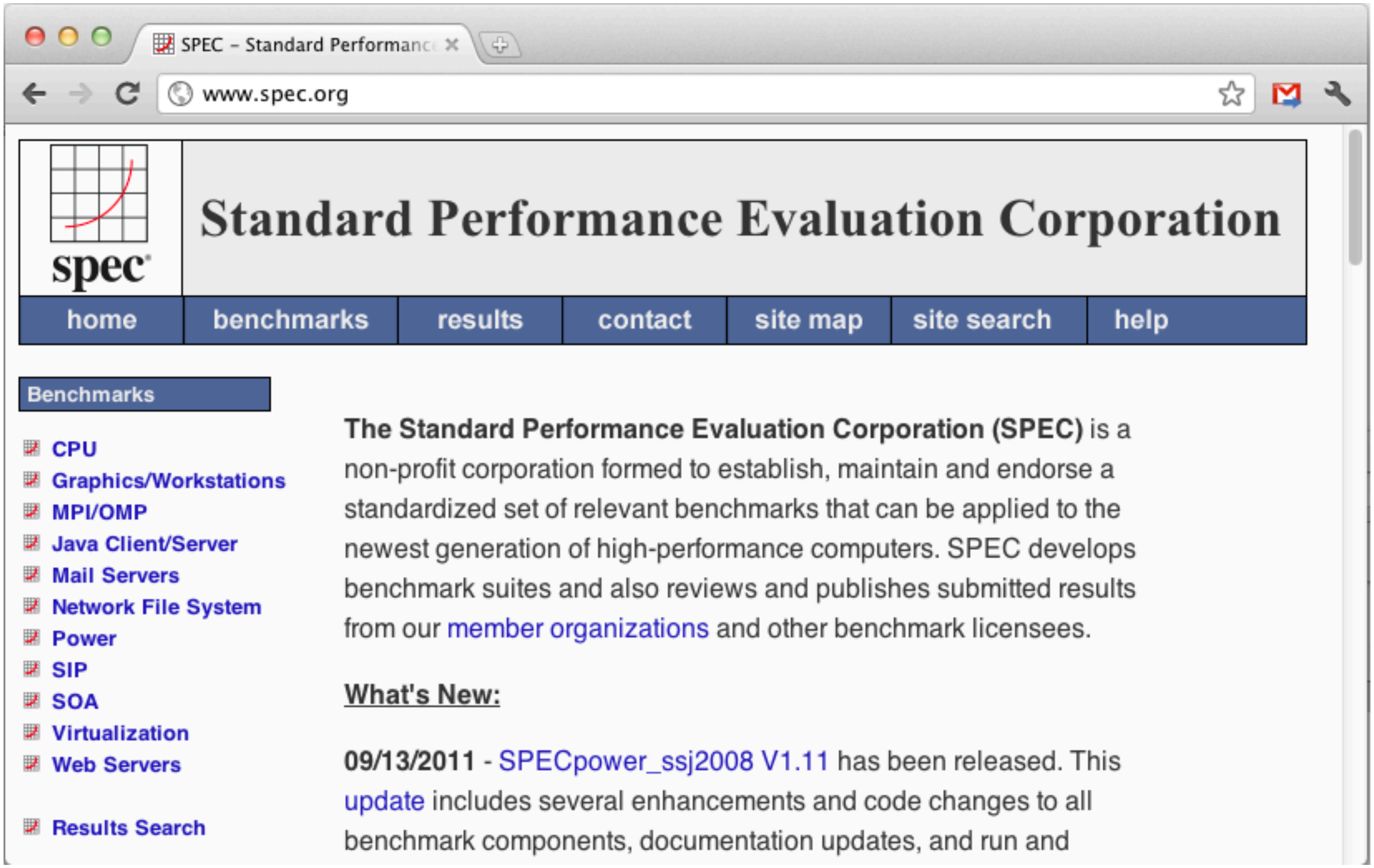
- SPEC89
  - Originally, called "SPEC"
  - 4 integer programs, 6 floating-point programs
  - One number: <u>***geometric mean***</u> of speedup relative to VAX 11/780
  - Represents a scientific workload (note – fp bias)

- SPEC92
  - 6 integer, 14 floating-point (int, fp results are <u>**always**</u> separated)
  - Eliminates matrix300 from SPEC89
  - Called CINT92, CFP92 or SPECint92, SPECfp92
  - Each number: <u>***geometric mean***</u> of speedup relative to VAX 11/780

- SPEC95
  - 8 integer, 10 floating-point
  - Two numbers: SPECint95, SPECfp95, relative to Sun 10/40

- SPEC history http://en.wikipedia.org/wiki/SPEC

SPEC CPU2000 and CPU2006



Lots of workloads! www.spec.org

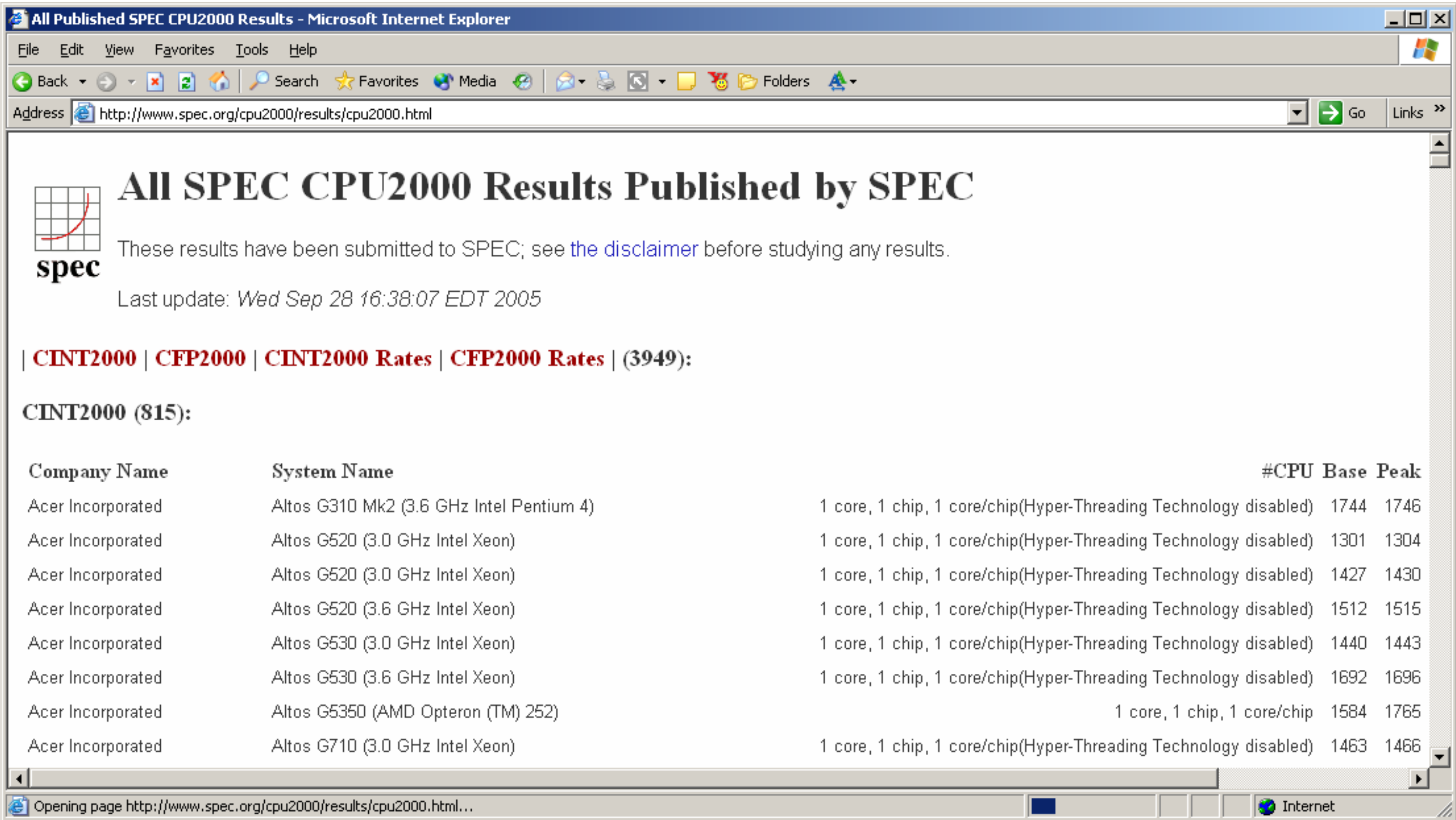
SPEC CPU2000

- Two benchmark sets
  - 12 Integer, 14 Floating-Point
- Two measurement conditions
  - Speed ("response time" or latency)
    - SPECint2000, SPECfp2000
  - Throughput
    - SPECint_rate2000, SPECfp_rate2000
- Why throughput numbers?
  - Computers with multiple CPUs (or multiple cores)
  - Computers with "virtual multiple" CPUs (eg, hyperthreading)
- How to measure throughput?
  - Run N copies of the benchmark, measure completion time
  - Convert execution time into a rate

SPEC CPU2000 Benchmarks

	INTEGER		FLOATING-POINT
Name	**Description**	**Name**	**Type**
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The GNU C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptical partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and route simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meterorology: pollutant distribution

SPECint2000 Results



All Published SPEC CPU2000 Results - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://www.spec.org/cpu2000/results/cpu2000.html

spec

All SPEC CPU2000 Results Published by SPEC

These results have been submitted to SPEC; see the disclaimer before studying any results.

Last update: *Wed Sep 28 16:38:07 EDT 2005*

| **CINT2000** | **CFP2000** | **CINT2000 Rates** | **CFP2000 Rates** | (3949):

CINT2000 (815):

Company Name	System Name	#CPU	Base	Peak
Acer Incorporated	Altos G310 Mk2 (3.6 GHz Intel Pentium 4)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1744	1746
Acer Incorporated	Altos G520 (3.0 GHz Intel Xeon)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1301	1304
Acer Incorporated	Altos G520 (3.0 GHz Intel Xeon)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1427	1430
Acer Incorporated	Altos G520 (3.6 GHz Intel Xeon)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1512	1515
Acer Incorporated	Altos G530 (3.0 GHz Intel Xeon)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1440	1443
Acer Incorporated	Altos G530 (3.6 GHz Intel Xeon)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1692	1696
Acer Incorporated	Altos G5350 (AMD Opteron (TM) 252)	1 core, 1 chip, 1 core/chip	1584	1765
Acer Incorporated	Altos G710 (3.0 GHz Intel Xeon)	1 core, 1 chip, 1 core/chip(Hyper-Threading Technology disabled)	1463	1466

Opening page http://www.spec.org/cpu2000/results/cpu2000.html...

Internet

SPEC CPU2000: Base vs Peak

- “Base” results
  - same compiler flags used for all programs, “typical user”

- “Peak” results
  - choose best compiler flags for each program, “power user”

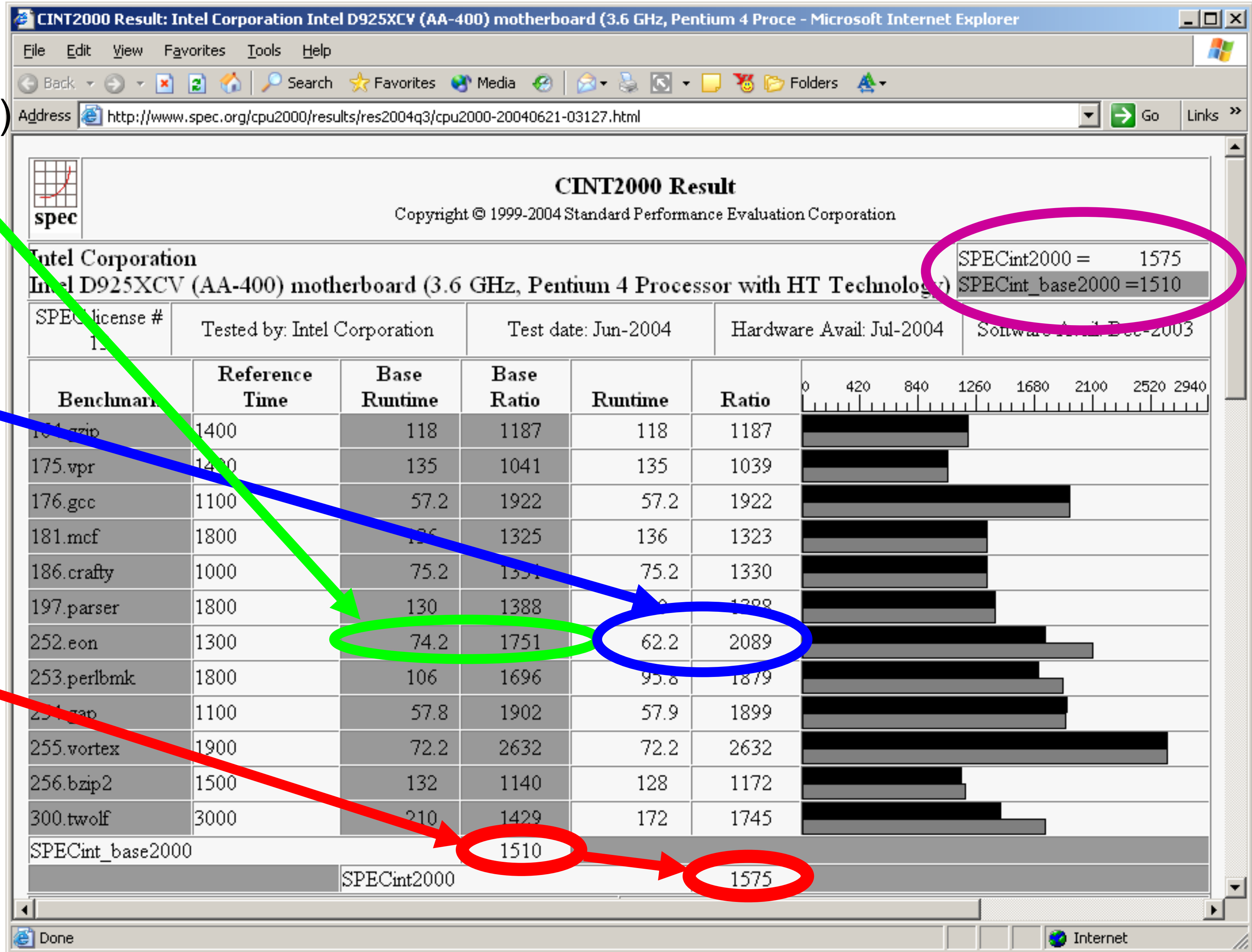
- Base, Peak numbers are normalized “percentage” results
  - Base Machine: Sun ULTRA5-10, 300MHz
  - Each program “problem size” scaled once
    - Runtime ~1000s-3000s on Base Machine
    - Takes ~40hrs to run full suite! (3 passes, CINT + CFP)
  - Base machine Performance defined to be “100%”

3.6 GHz Pentium 4

Base Ratio = 100*(1300/74.2) = 1752

Peak Ratio = 100*(1300/62.2) = 2090

Geometric Mean



Benchmark	Reference Time	Base Runtime	Base Ratio	Runtime	Ratio
164.gzip	1400	118	1187	118	1187
175.vpr	1400	135	1041	135	1039
176.gcc	1100	57.2	1922	57.2	1922
181.mcf	1800	136	1325	136	1323
186.crafty	1000	75.2	1331	75.2	1330
197.parser	1800	130	1388	130	1388
252.eon	1300	74.2	1751	62.2	2089
253.perlbmk	1800	106	1696	95.8	1879
254.gap	1100	57.8	1902	57.9	1899
255.vortex	1900	72.2	2632	72.2	2632
256.bzip2	1500	132	1140	128	1172
300.twolf	3000	210	1429	172	1745
SPECint_base2000			1510		
		SPECint2000			1575

SPEC CPU2000 Measurement

- SPECint2000base score of 1510
  - Means "15.10 times faster" than Base Machine
  - This is an average result (how was average computed?)
- Fair measurement requirement
  - Run each program an ODD number of times, report the median execution time
  - Must not create special compiler flags, eg "-spec_cpu2000"
- Reporting requirements
  - Vendor supplies results, SPEC accepts and publishes them
  - Must report which OS, compiler version, all compiler flags used
  - Must report complete system configuration

Top 20 Computer Systems
According to SPEC CPU2000 Data

Top 20 systems by processor count - Microsoft Internet Explorer

http://www.aceshardware.com/SPECmine/top.jsp

Top 20 SPEC systems

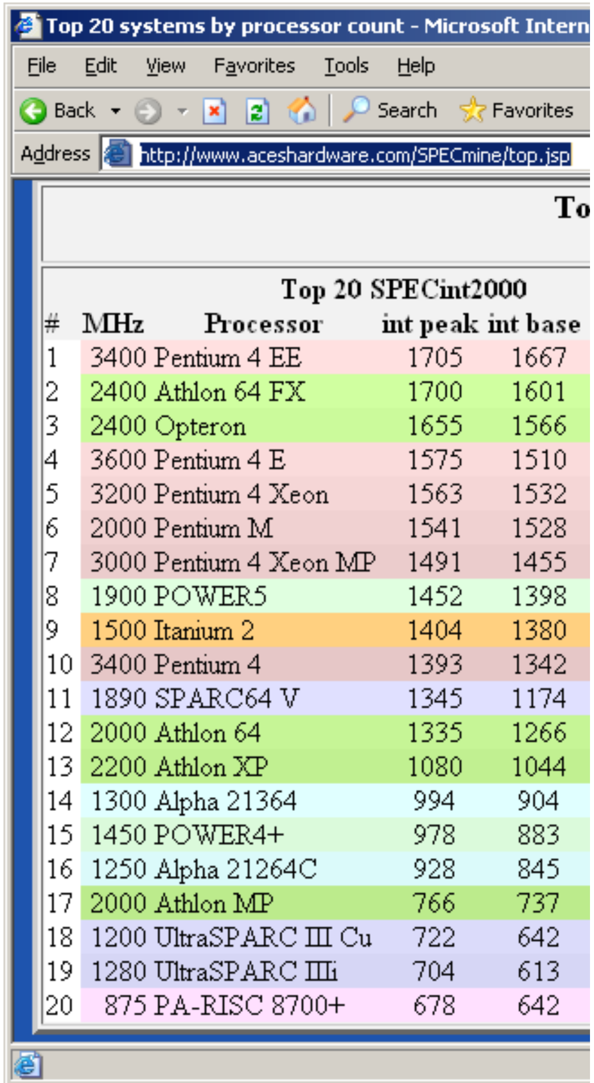
	Top 20 SPECint2000					Top 20 SPECfp2000				
#	MHz	Processor	int peak	int base	Full results	MHz	Processor	fp peak	fp base	Full results
1	2800	Athlon 64 FX	1970	1862	HTML	1900	POWER5	2796	2585	HTML
2	2800	Opteron	1956	1837	HTML	1600	Itanium 2	2712	2712	HTML
3	2260	Pentium M	1839	1812	HTML	2800	Opteron	2267	2051	HTML
4	3800	Pentium 4 E	1815	1793	HTML	2800	Athlon 64 FX	2261	2086	HTML
5	3800	Pentium 4 Xeon	1806	1799	HTML	2160	SPARC64 V	2139	1808	HTML
6	3466	Pentium 4 EE	1772	1701	HTML	3733	Pentium 4 E	2016	2013	HTML
7	2160	SPARC64 V	1594	1456	HTML	3800	Pentium 4 Xeon	1887	1861	HTML
8	1600	Itanium 2	1590	1590	HTML	3466	Pentium 4 EE	1724	1719	HTML
9	3667	Pentium 4 Xeon MP	1567	1556	HTML	3667	Pentium 4 Xeon MP	1717	1694	HTML
10	1900	POWER5	1456	1385	HTML	1300	Alpha 21364	1684	1279	HTML
11	3400	Pentium 4	1393	1342	HTML	3800	Pentium 4	1631	1633	HTML
12	2000	Athlon 64	1335	1266	HTML	2260	Pentium M	1375	1355	HTML
13	2200	Athlon XP	1080	1044	HTML	1250	Alpha 21264C	1365	1019	HTML
14	2200	PowerPC 970	1040	986	HTML	1600	UltraSPARC IIIi	1353	1200	HTML
15	1300	Alpha 21364	994	904	HTML	1450	POWER4+	1295	1221	HTML
16	1450	POWER4+	978	883	HTML	2000	Athlon 64	1250	1180	HTML
17	1250	Alpha 21264C	928	845	HTML	1200	UltraSPARC III Cu	1118	953	HTML
18	1600	UltraSPARC IIIi	845	743	HTML	2200	Athlon XP	982	873	HTML
19	2000	Athlon MP	766	737	HTML	1000	POWER4	886	843	HTML
20	1200	UltraSPARC III Cu	722	642	HTML	833	Alpha 21264B	784	643	HTML

Source: (Sept 27, 2005)
http://www.aceshardware.com/SPECmine/top.jsp

Top 20 Computer Systems

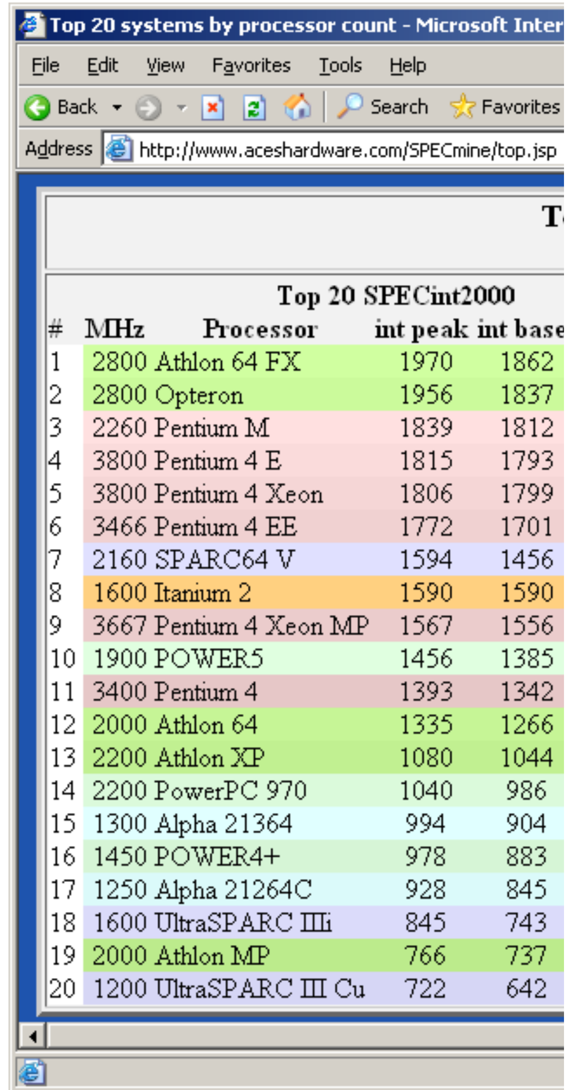
2004 2005

What's Changed?



Top 20 SPECint2000

#	MHz	Processor	int peak	int base
1	3400	Pentium 4 EE	1705	1667
2	2400	Athlon 64 FX	1700	1601
3	2400	Opteron	1655	1566
4	3600	Pentium 4 E	1575	1510
5	3200	Pentium 4 Xeon	1563	1532
6	2000	Pentium M	1541	1528
7	3000	Pentium 4 Xeon MP	1491	1455
8	1900	POWER5	1452	1398
9	1500	Itanium 2	1404	1380
10	3400	Pentium 4	1393	1342
11	1890	SPARC64 V	1345	1174
12	2000	Athlon 64	1335	1266
13	2200	Athlon XP	1080	1044
14	1300	Alpha 21364	994	904
15	1450	POWER4+	978	883
16	1250	Alpha 21264C	928	845
17	2000	Athlon MP	766	737
18	1200	UltraSPARC III Cu	722	642
19	1280	UltraSPARC IIIi	704	613
20	875	PA-RISC 8700+	678	642



Top 20 SPECint2000

#	MHz	Processor	int peak	int base
1	2800	Athlon 64 FX	1970	1862
2	2800	Opteron	1956	1837
3	2260	Pentium M	1839	1812
4	3800	Pentium 4 E	1815	1793
5	3800	Pentium 4 Xeon	1806	1799
6	3466	Pentium 4 EE	1772	1701
7	2160	SPARC64 V	1594	1456
8	1600	Itanium 2	1590	1590
9	3667	Pentium 4 Xeon MP	1567	1556
10	1900	POWER5	1456	1385
11	3400	Pentium 4	1393	1342
12	2000	Athlon 64	1335	1266
13	2200	Athlon XP	1080	1044
14	2200	PowerPC 970	1040	986
15	1300	Alpha 21364	994	904
16	1450	POWER4+	978	883
17	1250	Alpha 21264C	928	845
18	1600	UltraSPARC IIIi	845	743
19	2000	Athlon MP	766	737
20	1200	UltraSPARC III Cu	722	642

Source: (Oct, 2004) (Sept 27, 2005)

http://www.aceshardware.com/SPECmine/top.jsp

SPEC CPU2006 Benchmarks

CINT2006 contains 12 benchmarks:

400.perlbench	**C**	**PERL Programming Language**
401.bzip2	**C**	**Compression**
403.gcc	**C**	**C Compiler**
429.mcf	**C**	**Combinatorial Optimization**
445.gobmk	**C**	**Artificial Intelligence: go**
456.hmmer	**C**	**Search Gene Sequence**
458.sjeng	**C**	**Artificial Intelligence: chess**
462.libquantum	**C**	**Physics: Quantum Computing**
464.h264ref	**C**	**Video Compression**
471.omnetpp	**C++**	**Discrete Event Simulation**
473.astar	**C++**	**Path-finding Algorithms**
483.xalancbmk	**C++**	**XML Processing**

SPEC CPU2006 Benchmarks

CFP2006 contains 17 benchmarks:

410.bwaves	**Fortran**	**Fluid Dynamics**
416.gamess	**Fortran**	**Quantum Chemistry**
433.milc	**C**	**Physics: Quantum Chromodynamics**
434.zeusmp	**Fortran**	**Physics/CFD**
435.gromacs	**C/Fortran**	**Biochemistry/Molecular Dynamics**
436.cactusADM	**C/Fortran**	**Physics/General Relativity**
437.leslie3d	**Fortran**	**Fluid Dynamics**
444.namd	**C++**	**Biology/Molecular Dynamics**
447.dealII	**C++**	**Finite Element Analysis**
450.soplex	**C++**	**Linear Programming, Optimization**
453.povray	**C++**	**Image Ray-tracing**
454.calculix	**C/Fortran**	**Structural Mechanics**
459.GemsFDTD	**Fortran**	**Computational Electromagnetics**
465.tontoFortran	**Quantum**	**Chemistry**
470.lbm	**C**	**Fluid Dynamics**
481.wrf	**C/Fortran**	**Weather Prediction**
482.sphinx3	**C**	**Speech recognition**

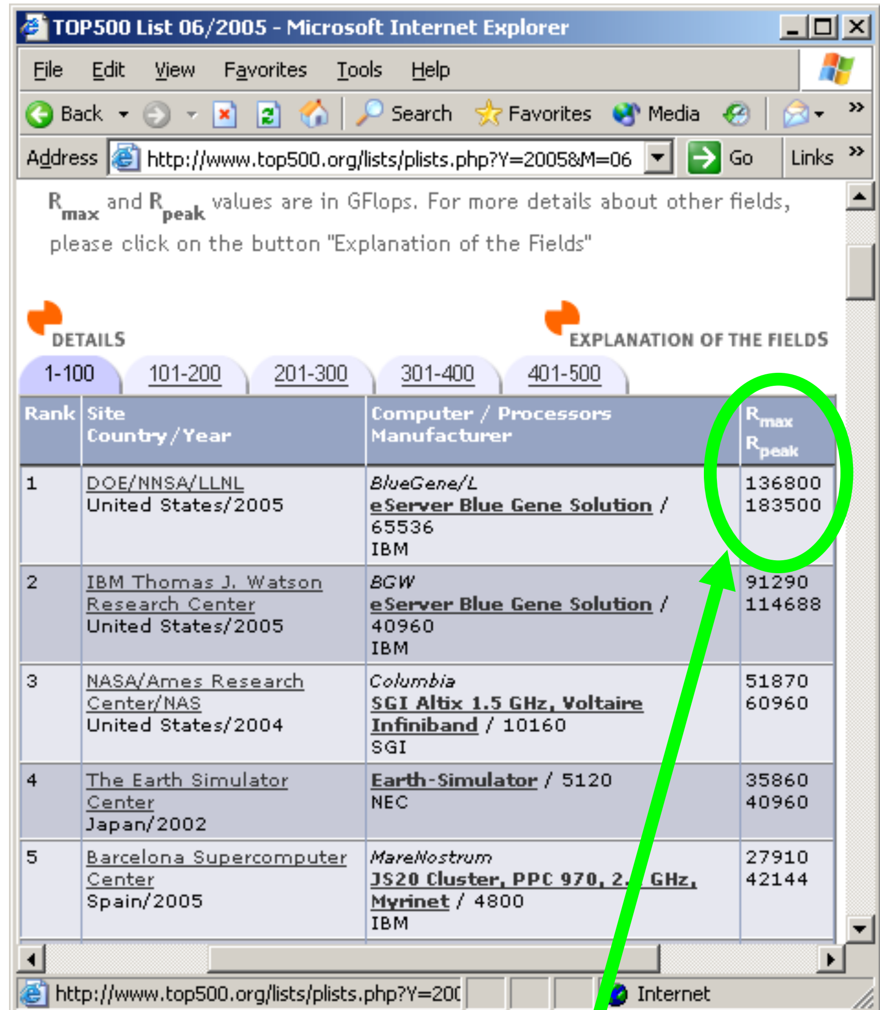
Coming Soon: New SPEC

aka “CPUv6”

Top 500 Supercomputers

http://www.top500.org

- Supercomputing
  - 1,000s of processors
  - Specialized programs
    - Nuclear simulations, weather prediction
  - Usually: floating-point operations on dense matrices
- Linpack performance



R_{max} and R_{peak} values are in GFlops. For more details about other fields, please click on the button "Explanation of the Fields"

Rank	Site Country/Year	Computer / Processors Manufacturer	R_{max} R_{peak}
1	DOE/NNSA/LLNL United States/2005	BlueGene/L eServer Blue Gene Solution / 65536 IBM	136800 183500
2	IBM Thomas J. Watson Research Center United States/2005	BGW eServer Blue Gene Solution / 40960 IBM	91290 114688
3	NASA/Ames Research Center/NAS United States/2004	Columbia SGI Altix 1.5 GHz, Voltaire Infiniband / 10160 SGI	51870 60960
4	The Earth Simulator Center Japan/2002	Earth-Simulator / 5120 NEC	35860 40960
5	Barcelona Supercomputer Center Spain/2005	MareNostrum JS20 Cluster, PPC 970, 2.2 GHz, Myrinet / 4800 IBM	27910 42144

LINPACK GFLOPS !!

Other Benchmarks

- Banking/database transactions
  - http://www.tpc.org (Transaction Processing Performance Council)
- Embedded CPUs
  - http://www.eembc.org (Embedded Microprocessor Benchmark Consortium)
- Multimedia, Network processors
  - http://cares.icsl.ucla.edu (MediaBench, NetBench) (broken link?)
- Supercomputing (Linpack)
  - http://www.netlib.org/benchmark/hpl (linear eqn solving of dense matrices)
- Reconfigurable computing
  - RAW http://cag-www.lcs.mit.edu/raw/benchmark/README.html
  - VersaBench http://cag.csail.mit.edu/versabench/
- Some toy benchmarks or misc benchmarks…
  - http://rib.cs.utk.edu/cgi-bin/catalog.pl?rh=226&term=0!0
  - http://www.netlib.org

Summary: Evaluating Performance

- Performance of one program
  - Execution time, performance equation
  - Instruction counts, CPI, clock cycle time

- Performance of many programs
  - First, choose the benchmark program(s)
    - This is the **most critical** step !!!
  - Second, compute execution time for each task/program
    - SPEC has **strict rules** about fair play…
  - Third, summarize performance
    - SPEC computes **geometric average** of normalized performance
    - We'll investigate the **rationale** behind these rules tomorrow…

Summarizing Performance

Distilling the performance results of
many programs into
one performance number!

aka .. What the Means Mean !

Summarizing Performance

- Given
  - Many benchmark programs
  - Each has different execution time
- Goal
  - How to calculate one “performance number” for all programs?
- Confusing fact
  - Problems exist with every possible summary method!
- Guiding light
  - The summary method you choose should track execution time
  - Execution time the only true measure of performance!

Summarizing Performance

Many possible summary methods…

- Arithmetic Mean of ExecutionTime (AMET)
  - Also: Weighted AMET
- Arithmetic Mean of Performance (AMP)
  - Beware **AMP != 1 / AMET**
- Harmonic Mean of Performance (HMP)
  - Note **HMP = 1 / AMET**
- Geometric Mean of Performance (GM)
  - Note **GM(X)/GM(Y) = GM(X/Y)**
- Many problems with each
  - But researchers often use “geometric mean”
  - This lecture will show you why…

Two Arithmetic Means

Two easy summary methods…

- Arithmetic Mean of ExecutionTime (AMET)
  - ET of 3 programs: 1 s, 10 s, 100 s
  - AMET = (1+10+100)/3 = 111/3 = 37 s

- Arithmetic Mean of Performance (AMP)
  - **RECALL:** Performance = 1 / ExecutionTime
  - Performance of 3 programs : $1\ s^{-1}$, $0.1\ s^{-1}$, $0.01\ s^{-1}$
  - AMP = (1+0.1+0.01)/3 = $0.37\ s^{-1}$
    - 1 / AMP = 2.7 s

- Beware

  AMET != 1 / AMP

- Leads to confusion, we'll **never** use AMP

Probing Further: Problem with Arithmetic Mean of Performance

- Beware
  - AMET != 1 / AMP

- **Cannot** use Arithmetic Mean of Performance (AMP) to summarize

- Why not??
  - Because Performance is a **rate** measurement, **"programs per second"**

- Instead, can sometimes use **Harmonic Mean of Performance** (HMP) to average rates

Harmonic Mean of Performance

Note: similar to putting R_i resistors in parallel and multiplying by N

- Harmonic Mean

$$HMP(R_i) = \frac{N}{1/R_1 + 1/R_2 + \ldots + 1/R_N}$$

- Can use **Harmonic Mean of Performance** (HMP) to average **rates**
  - But, it is correct only if "same amount of work" is done
  - What is "work" ?
    - "1 program" = 1 unit of work?
    - "1 instruction" = 1 unit of work?

- Example
  - $HMP = 3 / (1/1 + 1/0.1 + 1/0.01) = 3/111 = 0.027\ s^{-1}$

- *Harmonic Mean of Performance* tracks *Arithmetic Mean of ExecutionTime*
  - $1 / HMP = 1/0.027 = 37\ s$
  - HMP is equivalent to:
    - "Convert Performance to ET, take arithmetic mean, convert back to Performance"

- Notice

  AMET = 1 / HMP

Example 1
The Need for Arithmetic…

- You're at home and late for class….
  - Run 10km/h to your car (6 minutes, 1km)
  - Drive 100km/h to school (6 minutes, 10km)
    - Arrive early (whew!)
  - Walk 5km/h to class (6 minutes, 0.5km)

- True Average speed?
  - You travelled 11.5km in 18 min
  - 11.5km / 18min = 38.3 km/h

- Arithmetic Mean for average speed?
  (10km/h + 100km/h + 5km/h)/3 = 38.3 km/h
  - **Arithmetic Mean works**!

- Harmonic Mean for average speed?
  HM = 3 * 1/(1/10 + 1/100 + 1/5) = 3 / (0.1 + 0.01 + 0.2) = 3 / 0.31 = 9.68 km/h
  - **Harmonic Mean doesn't work**!!

Example 2
The Need for Harmony…

- You' re going to an island for a vacation….
  - Drive 100km/h to the ferry (12 minutes, 20km)
  - Boat 30km/h to the island (40 minutes, 20km)
  - Bus 60km/h to the resort (20 minutes, 20km)

- True Average speed?
  - You travelled 60km in 72 minutes
  - 60km/72min = 50 km/h

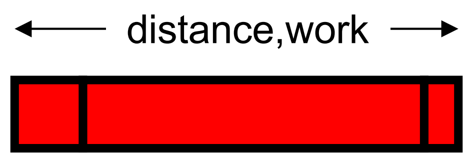
- Arithmetic Mean for average speed?
  (100km/h + 30km/h + 60km/h)/3 = 63.3 km/h
  - **Arithmetic Mean doesn' t work**!!

- Harmonic Mean for average speed?
  HM = 3 * 1/(1/100 + 1/30 + 1/60) = 3 / (0.01 + 0.033 + 0.0167) = 50 km/h
  - **Harmonic Mean works**!

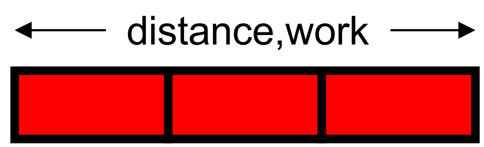
Arithmetic vs Harmonic Means

- Averaging speed → speed is a rate
  - Rate is "work per time"
  - Averaging RATES requires some care!

- What's the difference?
  - Example 1



    - AM works → each task is performed for same length of time
    - HM fails → each task performs different amount of work
      - Work can be "same distance", or "same program"



  - Example 2
    - AM fails → each task takes different length of time
    - HM works → each task performs the same amount of work

Example 3
Correctly Averaging Rates

- Slightly different numbers… note: different time, different work
  - Drive 100km/h to the ferry (6 minutes, 10km)
  - Boat 30km/h to the island (80 minutes, 40km)
  - Bus 60km/h to the resort (20 minutes, 20km)

- True Average speed? TrueAvg = TotalDist / TotalTime
  - You travelled 70km in 106 minutes
  - 70km/106min = 39.6 km/h (this is the only true answer!)

- Arithmetic Mean for average speed?
  (100km/h + 30km/h + 60km/h)/3 = 63.3 km/h
  - **Arithmetic Mean doesn't work**!!

- Harmonic Mean for average speed?
  HM = 3 * 1/(1/100 + 1/30 + 1/60) = 3 / (0.01 + 0.033 + 0.0167) = 50 km/h
  - **Harmonic Mean doesn't work**!!

- Lesson
  - Easy to miss-use AMET or HMP and get incorrect results !!

Another Problem with Arithmetic Mean

- Example
  - Three programs: A=1s, B=10000s, C=10000s
    - ArithMeanBefore = 20001 / 3 = 6667s
  - Speed up Program A from 1s to 0.01s (100x!!)
    - ArithMeanAfter = 20000.01 / 3 = 6666.67s
  - Average Speedup = Before/After = 6667/6666.67 ≈ 1.00005
    - But one program was sped up by 100 times !!!
- Arithmetic Mean sometimes hides significant gains in performance
  - Each program is not treated equally
- Proposed Solution: Weighted Arithmetic Mean
  - Make each program "equally significant" in the mean.

Weighted Arithmetic Mean

Weighted Arithmetic Mean

- How to choose the weights? Two main choices:

1. Equalized Weighting
   - Fix weights to each program to equalize ExecutionTime of each
     - Weight(A) = 10000, Weight(B) = 1, Weight(C) = 1
     - EQWeightArithMean = (10000*1 + 1*10000 + 1*10000)/3 = 10000s
   - Speed up Program A from 1s to 0.01s (100x)
     - EQWeightArithMean = (10000*0.01 + 1*10000 + 1*10000)/3 = 6700s
   - Average Speedup = Before/After = 10000/6700 ≈ 1.49
     - Change in EQWeightArithMean now reflects the big improvement to Program A
   - What is the Speedup if we improve Program B from 10000s to 100s (also 100x)?

2. Alternate "Unequal" Weighting
   - According to your Workload
   - Eg, if you run Program B 10 times more often than Programs A or C…

Simple Arithmetic Mean Example

- Arithmetic Mean Example
  - Two computers, X and Y, each run 3 programs A, B, C
    - X: 10s 20s 30s
    - Y: 1s 10s 100s
  - ArithMean(X) = (10s + 20s + 30s)/3 = 20s
  - ArithMean(Y) = (1s + 10s + 100s)/3 = 37s
- Conclusion
  - X is faster than Y

The truth!

Limits of Arithmetic Mean

- Arithmetic Mean of Normalized results?
  - Normalize to X: ExecTime(Y relative to X) = ExecTime(Y) / ExecTime(X)

- Same Example
  - Two computers, X and Y, each run 3 programs A, B, C
    - X: 10s 20s 30s
    - Y: 1s 10s 100s
  - Conclusion: X is faster than Y

The truth!

- Normalize ExecTime to X (smaller number → Y faster)
  - ArithMean (Y/X) = (0.1 + 0.5 + 3.333) / 3 = 1.311
  - AM of normalized-ET of Y wrt X = 1.311 > 1.0 so Y is slower

- Normalize ExecTime to Y (smaller number → X faster)
  - ArithMean (X/Y) = (10 + 2 + 0.3) / 3 = 4.1
  - AM of normalized-ET of X wrt Y = 4.1 > 1.0 so X is slower

Inconsistent!

Arithmetic Geometric Mean

- When to abandon Arithmetic Mean?
  - **Must abandon** if we ***normalize*** results
    - Get inconsistent answers, depends on choice of baseline
  - **Must use** Geometric Mean instead
- Geometric Mean

$$GM(R_i) = (R_1 * R_2 * \ldots * R_N)^{1/N}$$

Multiply N numbers, take N^{th} root

Note – none of the numbers can be zero !!

Geometric Mean of Normalized Numbers

- Normalized numbers…
  - **Must abandon** Arithmetic Mean (inconsistent)
  - **Must use** Geometric Mean instead
- Same example
  - Two computers, X and Y, each run 3 programs A, B, C
  - GeoMean(X) = $(10s * 20s * 30s)^{1/3} = 18.17$ s
  - GeoMean(Y) = $(1s * 10s * 100s)^{1/3} = 10$ s
  - GeoMean(X/Y) = $(10 * 2 * 0.3)^{1/3} = 1.817$
  - GeoMean(Y/X) = $(0.1 * 0.5 * 3.33)^{1/3} = 0.55 = 1 / 1.817$
  - GeoMean gives **consistent** answers: Y is faster than X

Geometric Mean Properties

- Useful Properties of Geometric Mean
  - GeoMean(X) / GeoMean(Y) = GeoMean(X/Y)
  - GeoMean(X/Y) = 1 / GeoMean(Y/X)
  - Whether we normalize ***before*** computing the mean, or normalize ***after*** computing the mean, we get the ***same answer***!
- Homework: prove it!
- Consequences
  - It doesn’t matter which machine is used for normalization, we get the same relative answers!
  - Easy to change data if we decide to normalize to a different machine

Normalizing Performance Example

	Computer X	Computer Y
Program A	1s	10s
Program B	1000s	100s
Total Time	**1001s**	**110s**

- Which is faster, computer X or Y?
  - X is 10x faster than Y for program A
  - Y is 10x faster than X for program B
  - Confusing! How to summarize?

- Total ExecutionTime says Y is 9.1x faster
  - ArithMean of ExecutionTime says same thing
- OK so far…..

Normalizing Performance Example

			Normalize to X	
	Comp X	Comp Y	**X**	**Y**
Program A	1s	10s	**1**	**10**
Program B	1000s	100s	**1**	**0.1**
ArithMean	500.5s	55s	**1**	**5.05**
GeoMean	31.6s	31.6s	**1**	**1**

- Normalize to X….
  - GeoMean says X and Y are same speed
  - ArithMean says X is 5.05x faster than Y
  - Huh???

- It gets worse
  - Try normalizing to Y… ** Beware… it will be confusing…

Normalizing Performance Example

			Normalize to X		**Normalize to Y**	
	Comp X	Comp Y	X	Y	**X**	**Y**
Program A	1s	10s	1	10	**0.1**	**1**
Program B	1000s	100s	1	0.1	**10**	**1**
ArithMean	500.5s	55s	1	5.05	**5.05**	**1**
GeoMean	31.6s	31.6s	1	1	**1**	**1**

- Normalize to X….
  - GeoMean says X and Y are same speed
  - ArithMean says X is 5.05x faster than Y

- Normalize to Y….
  - GeoMean still says X and Y are same speed
  - ArithMean says Y is 5.05x faster than X (opposite to before!)

Summarizing Summary

Lessons

1. Cannot take Arithmetic Mean of Rates
2. Cannot take ArithMean after Normalizing
3. Use HarmMean of Rates (if they do same amount of work)
4. Use GeoMean after Normalizing
5. GeoMean gives consistent answers, but may be **misleading**
   - Do a sanity check
   - AMET or TotalET sometimes more helpful (Y is 9.1x faster)
   - **Would you rather own computer X or Y ?**

	Comp X	**Comp Y**
Program A	1s	10s
Program B	1000s	100s
Total Time	**1001s**	**110s**

Understanding SPEC CPU2000

- SPEC CPU2000 == standard set of benchmark programs
  - Constantly being updated, eg SPEC2006

- Results are **Normalized** to Base Machine
  - Base Machine: Sun ULTRA5-10, 300MHz
  - Base Machine Performance defined to be “100%”

- Multiple benchmark programs
  - Report “normalized” performance result of each program
  - Summarize by computing **GEOMETRIC MEAN**
  - Example
    - SPECint2000 = 1510
    - Means “15.10 times faster” than Base Machine
    - 1510 is a geometric mean of <u>normalized</u> performance for each benchmark

- Fair measurement requirement
  - Run each program ODD # times, report median execution time
  - Why the **median** ? Why not fastest? Why not average?

Homework !

- Do Assignment 0
  - Matrix multiply, done several ways
- Start Assignment 1
  - Instead of matrix multiply, do Gaussian Elimination algorithm
  - Do both forward-elimination only
- Next Week
  - Culler text, Chapter 2, is online under “Lectures - Reading”