Benchmarking

How to Lie with Statistics*

*Darrell Huff, *How to Lie with Statistics*, Norton, New York, 1954

The only reliable way to measure performance is by running actual applications on real hardware.

If we want to compare performance across different contexts, this implies use of a benchmark.

Standard Performance Evaluation Corporation

- http://www.spec.org/
- Many benchmarks, most commonly CPU
- CPU2006 / 2000 / 95 / 92
- [published results]
- Choice of integer or floating point
- Each is a suite (12 integer, 17 floating point)
- C, C++, Fortran, statically compiled & linked

SPEC CINT 2006

Benchmark	Brief Description
400.perlbench	Based on Perl V5.8.7. The workload includes SpamAssassin, MHonArc email indexer, and specdiff
401.bzip2	Julian Seward's bzip2 version 1.0.3, modified to work in memory
403.gcc	gcc V 3.2 targeting an AMD Opteron
429.mcf	Network simplex public transport scheduler
445.gobmk	Plays the game of Go
456.hmmer	Protein sequence analysis using profile hidden Markov models
458.sjeng	Chess program that also plays several variants
462.libquantum	Simulates a quantum computer
464.h264ref	H.264/AVC video compression
471.omnetpp	OMNet++ discrete event simulator modeling an Ethernet network
473.astar	Pathfinding library for 2D maps, including A* search
483.xalancbmk	A modified version of Xalan-C++, for transforming XML

SPEC CFP2006 Part 1

Benchmark	Brief Description
410.bwaves	3D transonic viscous flow
416.gamess	Quantum chemistry
433.milc	Lattice gauge field generator
434.zeusmp	Astrophysics CFD (computational fluid dynamics)
435.gromacs	Molecular dynamics
436.cactusADM	Einstein equation solver
437.leslie3d	Large eddy CFD
444.namd	Biology molecular dynamics

SPEC CFP2006 Part 2

Benchmark	Brief Description
447.dealll	Finite element analysis
450.soplex	Simplex linear algorithm
453.povray	Ray tracing
454.calculix	Structural analysis
459.GemsFDTD	Solves 3D Maxwell equations
465.tonto	Quantum chemistry w/ OO Fortran
470.lbm	Lattice Boltzmann fluid flow simulation
481.wrf	Weather model
482.sphinx3	Speech recognition

SPEC History

H&P Fig. 1.16
Note how few persist for multiple generations

		Benchmark name by SPEC generation			
SPEC2006 benchmark description	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler	←	←	←	←	gcc
Interpreted string processing	←	←	perl		espresso
Combinatorial optimization	←	mcf			li
Block-sorting compression	←	bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

Typical CINT Summary

Company and model

spec

SPEC® CINT2006 Result

Copyright © 2007 Standard Performance Evaluation Corporation

Bull SAS NovaScale T860 E1 (Intel Xeon X5260,3.33GHz)	SPECint®2006 = 26.9 SPECint_base2006 = 22.6

CPU2006 license:	20	**Test date:**	Feb-2008
Test sponsor:	Bull SAS	**Hardware Availability:**	Feb-2008
Tested by:	NEC Corporation	**Software Availability:**	Nov-2007

Dates

Typical CINT Summary

What they quote in marketing material

spec

SPEC® CINT2006 Result

Copyright © 2007 Standard Performance Evaluation Corporation

Bull SAS NovaScale T860 E1 (Intel Xeon X5260,3.33GHz)	SPECint®2006 = 26.9 SPECint_base2006 = 22.6

CPU2006 license:	20	**Test date:**	Feb-2008
Test sponsor:	Bull SAS	**Hardware Availability:**	Feb-2008
Tested by:	NEC Corporation	**Software Availability:**	Nov-2007

Typical CINT Summary

What naive people think is more realistic

spec

SPEC® CINT2006 Result

Copyright © 2007 Standard Performance Evaluation Corporation

Bull SAS NovaScale T860 E1 (Intel Xeon X5260,3.33GHz)	SPECint®2006 = 26.9 SPECint_base2006 = 22.6

CPU2006 license:	20	**Test date:**	Feb-2008
Test sponsor:	Bull SAS	**Hardware Availability:**	Feb-2008
Tested by:	NEC Corporation	**Software Availability:**	Nov-2007

What's the difference?

Base Rules

1. No naming benchmarks or routines
2. No library substitution
3. No feedback-directed optimizations
4. Only safe optimizations
5. Same optimizations for all
6. No assertions to guide optimization

Base vs Peak

- Base sounds more realistic
- Peak is “no holds barred, anything goes”
- So why is it naive to think base is more meaningful?
- Need to look deeper

Individual Results

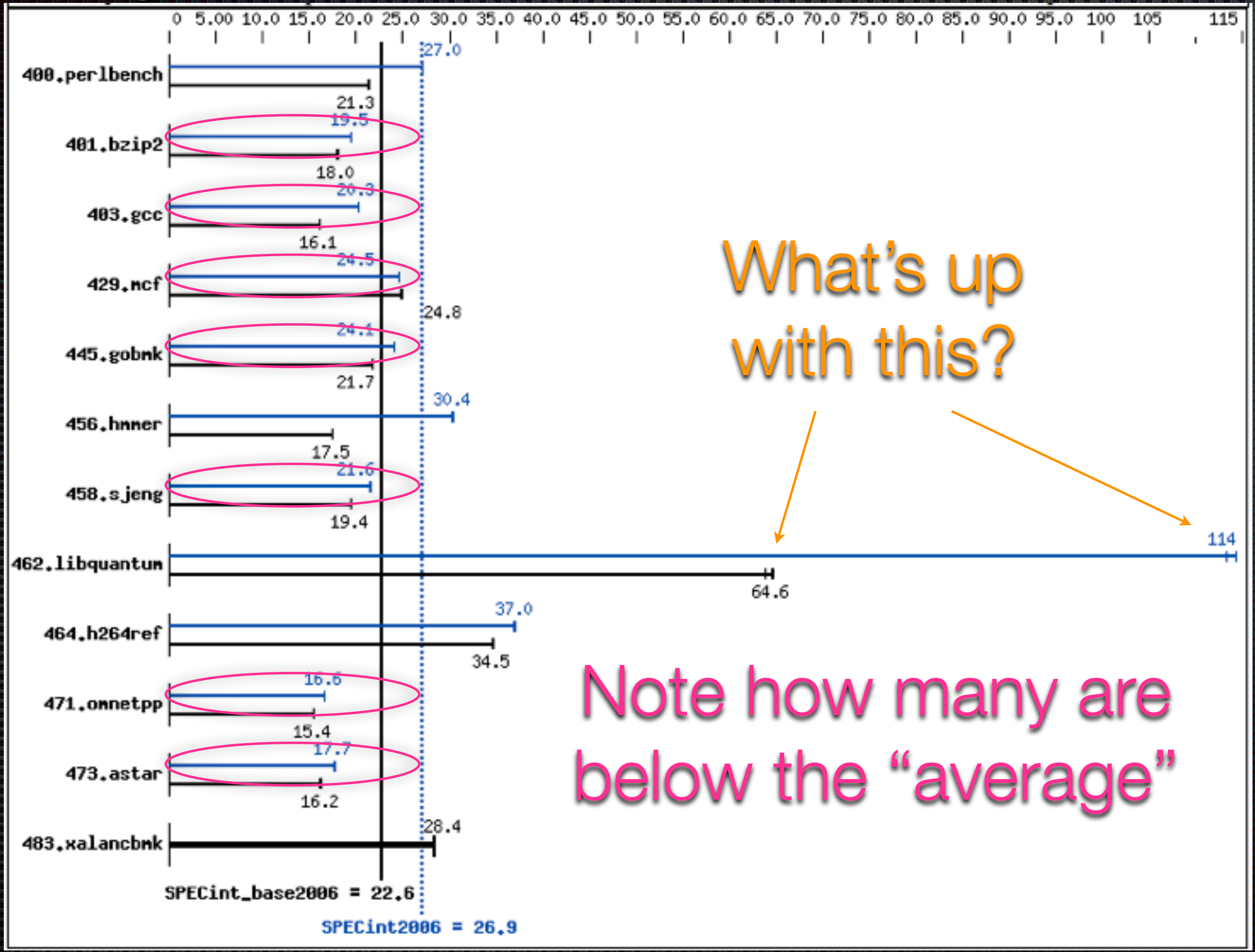
Results Table

Benchmark	Base						Peak					
	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
400.perlbench	461	21.2	**459**	**21.3**	455	21.5	**361**	**27.0**	361	27.0	361	27.0
401.bzip2	535	18.0	**536**	**18.0**	537	18.0	496	19.5	**496**	**19.5**	497	19.4
403.gcc	499	16.1	**501**	**16.1**	502	16.0	396	20.3	397	20.3	**397**	**20.3**
429.mcf	368	24.8	366	24.9	**367**	**24.8**	371	24.6	**372**	**24.5**	372	24.5
445.gobmk	**483**	**21.7**	483	21.7	482	21.8	**436**	**24.1**	436	24.1	436	24.0
456.hmmer	**534**	**17.5**	534	17.5	533	17.5	**307**	**30.4**	308	30.3	307	30.4
458.sjeng	**623**	**19.4**	624	19.4	623	19.4	**560**	**21.6**	564	21.5	559	21.7
462.libquantum	321	64.6	**321**	**64.6**	325	63.8	183	113	**181**	**114**	181	114
464.h264ref	638	34.7	641	34.5	**641**	**34.5**	**598**	**37.0**	599	37.0	597	37.1
471.omnetpp	406	15.4	**406**	**15.4**	406	15.4	376	16.6	376	16.6	**376**	**16.6**
473.astar	436	16.1	**433**	**16.2**	433	16.2	396	17.7	400	17.6	**397**	**17.7**
483.xalancbmk	**243**	**28.4**	243	28.4	244	28.3	**243**	**28.4**	243	28.4	244	28.3

Results appear in the order in which they were run. Bold underlined text indicates a median measurement.

Run each benchmark three times, divide each run by a reference time (so higher score is better), use median values to compute summary average of ratios. Sounds reasonable...

Graphically



What's up with this?

Note how many are below the "average"

How to Average?

- The usual way (arithmetic mean)

$$\bar{x} = \frac{\sum_{i=1}^{n} r_i}{n}$$

- The SPEC way (geometric mean)

$$\bar{x} = \sqrt[n]{\prod_{i=1}^{n} r_i}$$

- Both are sensitive to outliers
- A little effort to improve one benchmark yields a much better average overall

Another Average

$$\bar{x} = \frac{n}{\sum_{i=1}^{n} \frac{1}{r_i}}$$

- When averaging ratios, harmonic mean yields a value proportional to the total
  - Short-running applications have less influence on total time
- Harmonic mean is less sensitive to outliers

Example

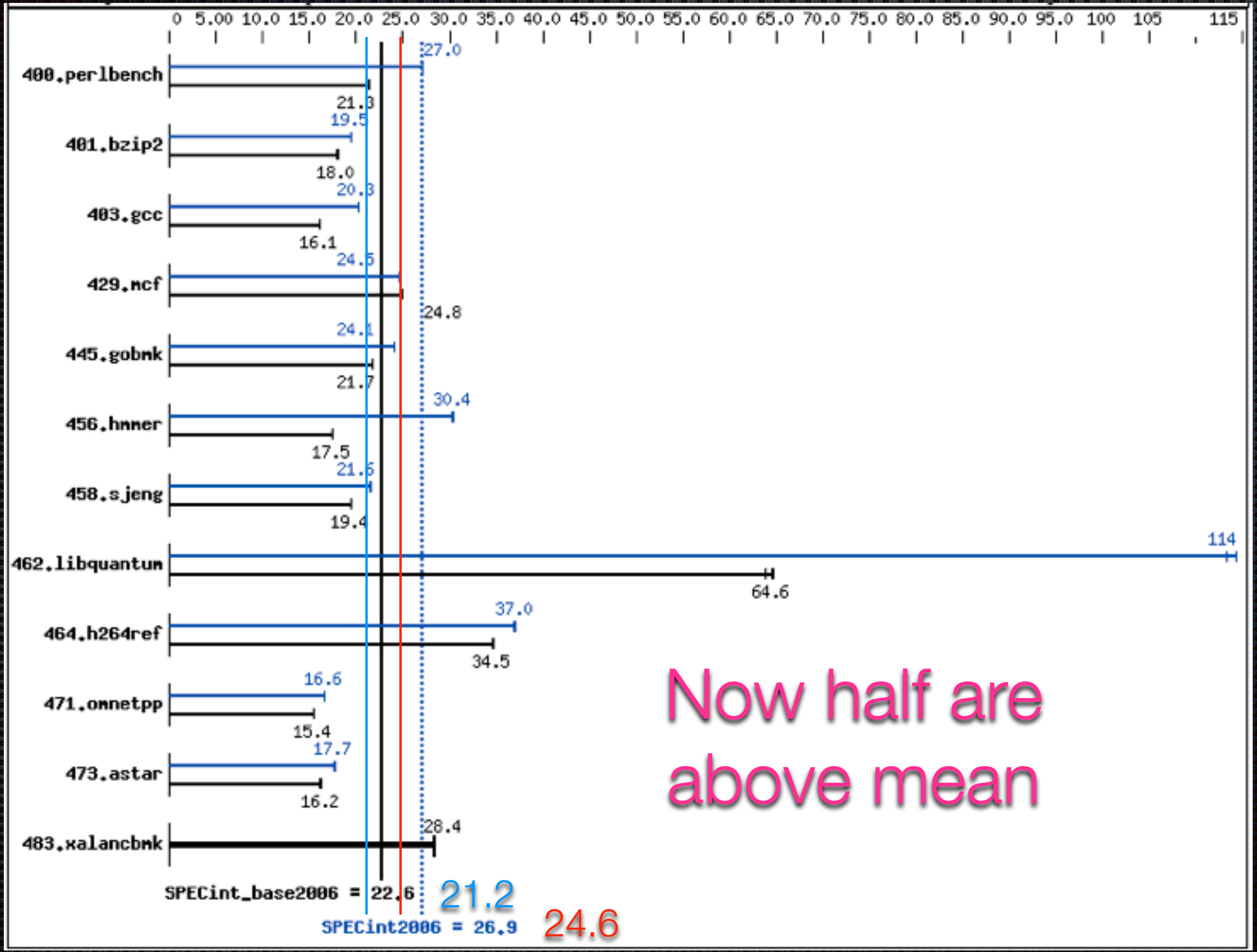
Results Table

Benchmark	Base						Peak					
	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
400.perlbench	461	21.2	**459**	**21.3**	455	21.5	**361**	**27.0**	361	27.0	361	27.0
401.bzip2	535	18.0	**536**	**18.0**	537	18.0	496	19.5	**496**	**19.5**	497	19.4
403.gcc	499	16.1	**501**	**16.1**	502	16.0	396	20.3	397	20.3	**397**	**20.3**
429.mcf	368	24.8	366	24.9	**367**	**24.8**	371	24.6	**372**	**24.5**	372	24.5
445.gobmk	**483**	**21.7**	483	21.7	482	21.8	**436**	**24.1**	436	24.1	436	24.0
456.hmmer	**534**	**17.5**	534	17.5	533	17.5	**307**	**30.4**	308	30.3	307	30.4
458.sjeng	**623**	**19.4**	624	19.4	623	19.4	**560**	**21.6**	564	21.5	559	21.7
462.libquantum	321	64.6	**321**	**64.6**	325	63.8	183	113	**181**	**114**	181	114
464.h264ref	638	34.7	641	34.5	**641**	**34.5**	**598**	**37.0**	599	37.0	597	37.1
471.omnetpp	406	15.4	**406**	**15.4**	406	15.4	376	16.6	376	16.6	**376**	**16.6**
473.astar	436	16.1	**433**	**16.2**	433	16.2	396	17.7	400	17.6	**397**	**17.7**
483.xalancbmk	**243**	**28.4**	243	28.4	244	28.3	**243**	**28.4**	243	28.4	244	28.3

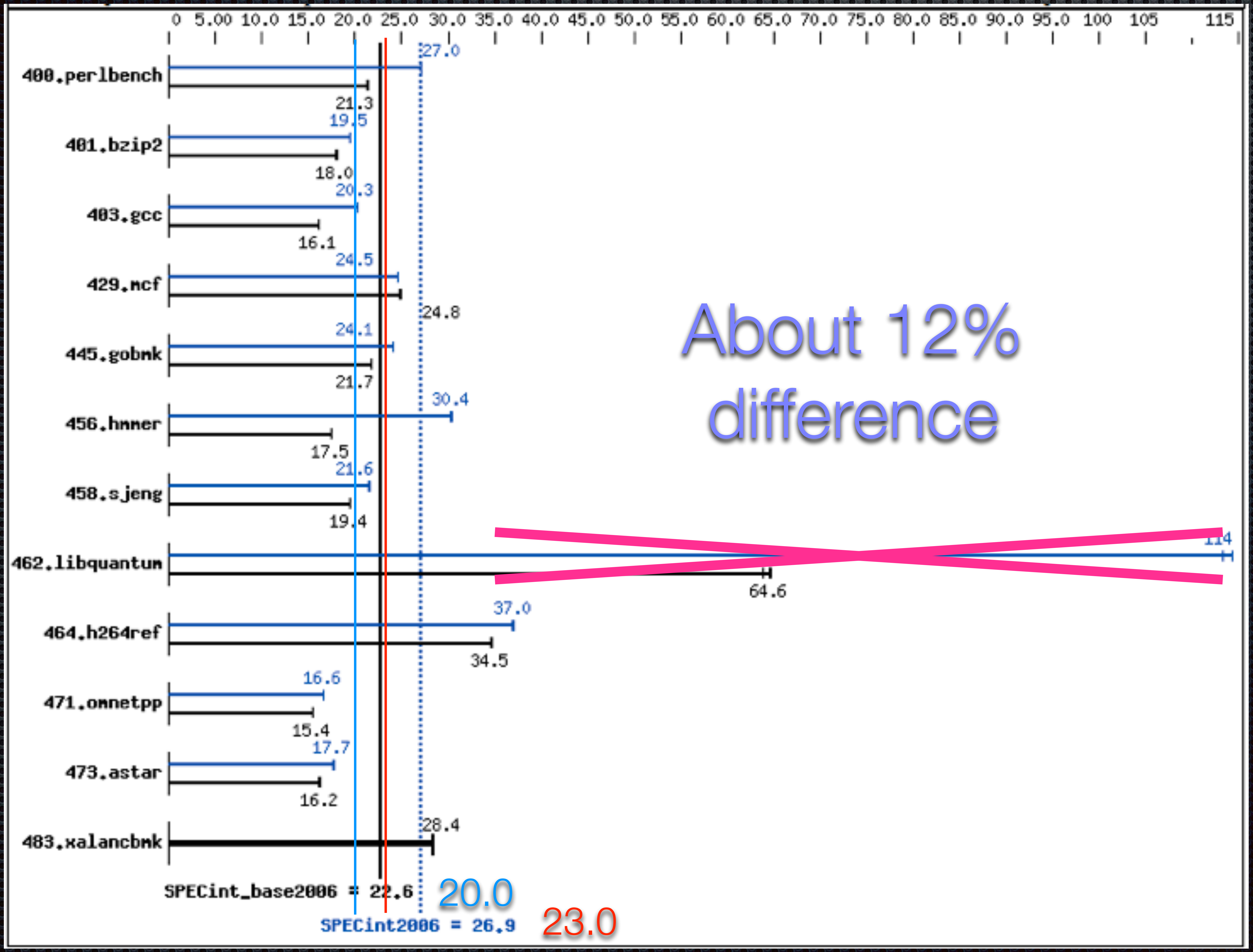
Results appear in the order in which they were run. Bold underlined text indicates a median measurement.

$$\bar{X} = \frac{n}{\sum_{i=1}^{n} \frac{1}{r_i}}$$

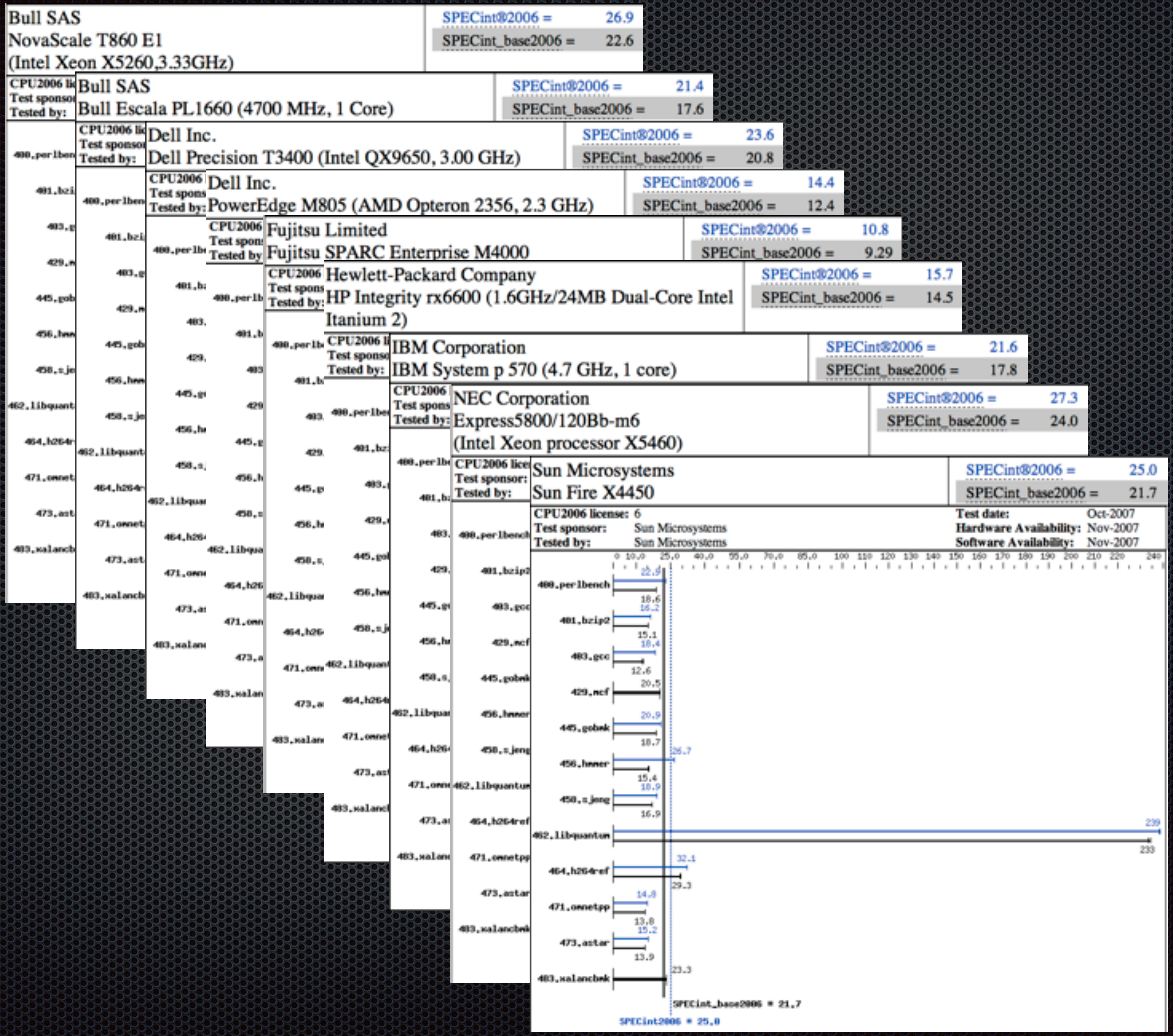
Using Harmonic Mean



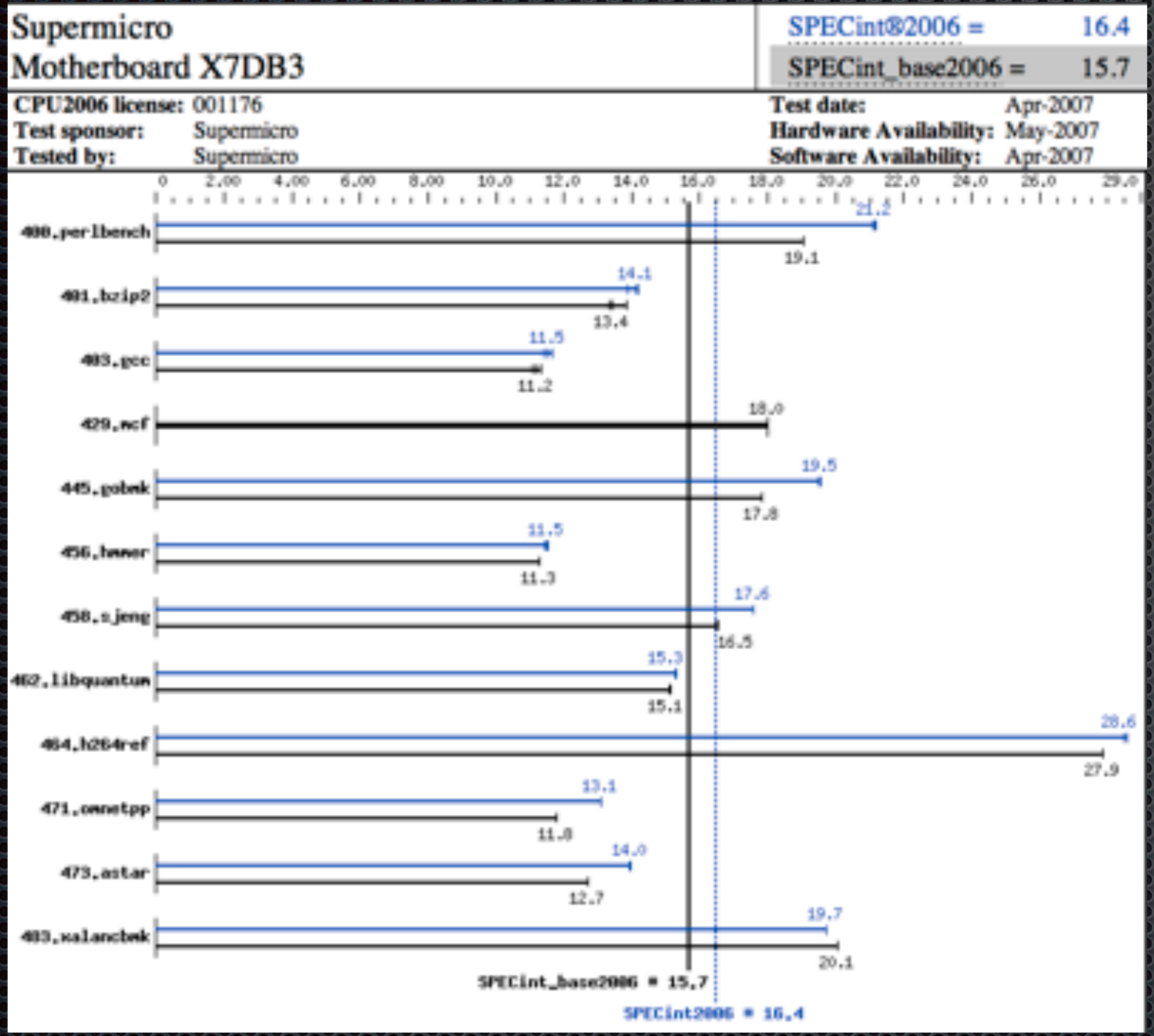
Omitting the Outlier

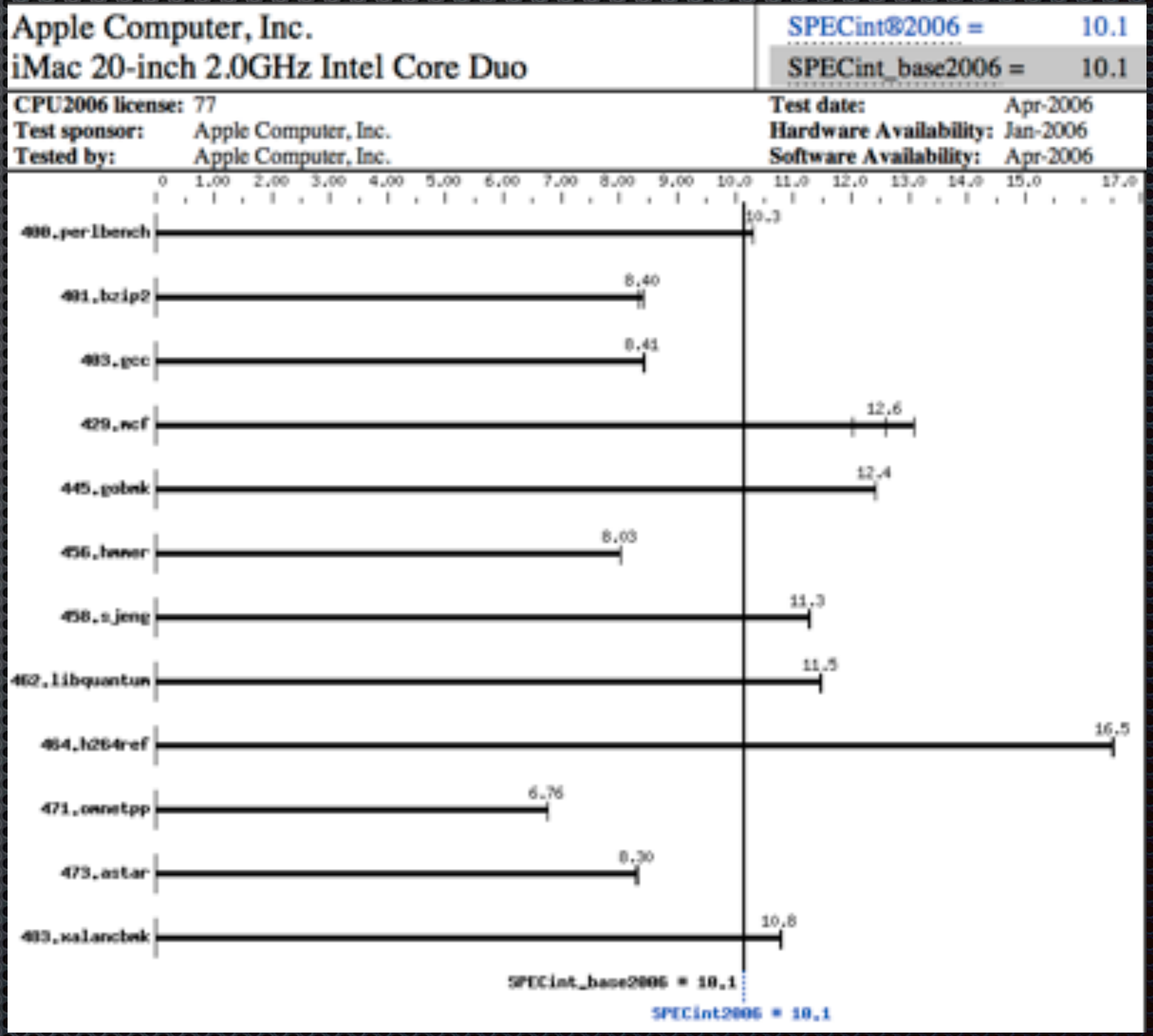


How Common is This?

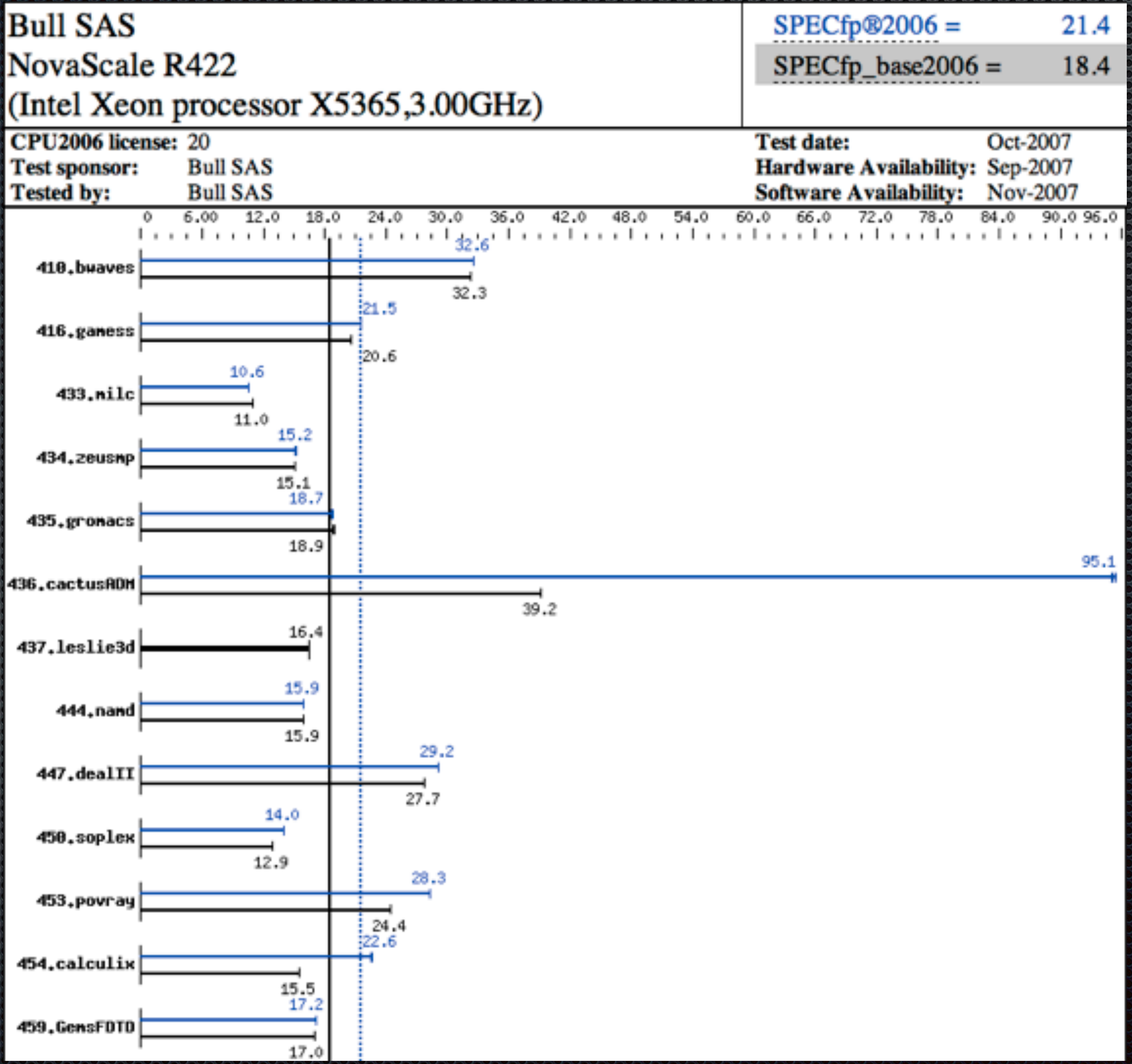


Are any Different?





How About SPEC FP?



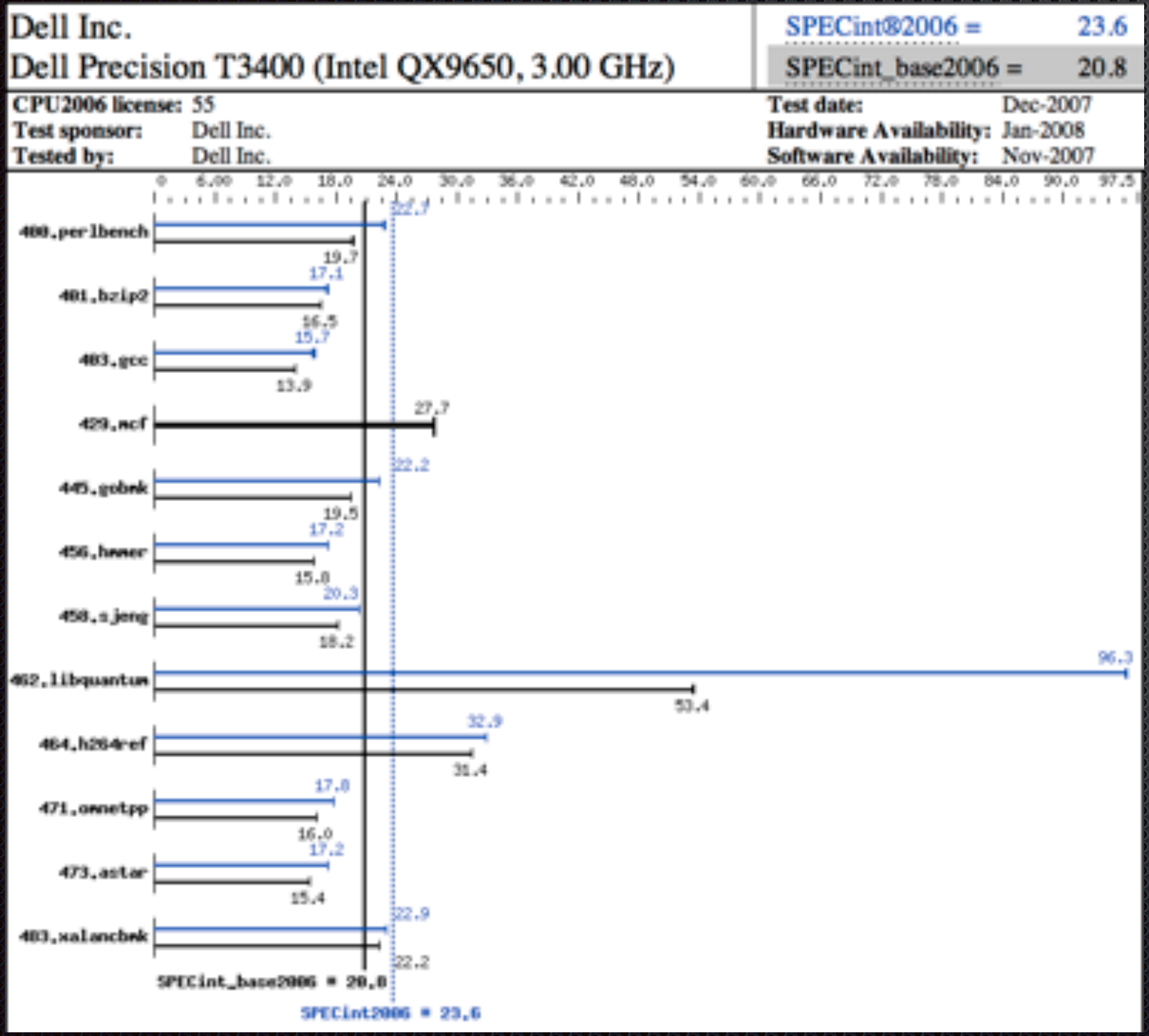
So?

If they all do it, aren’t the numbers meaningful in a relative sense?

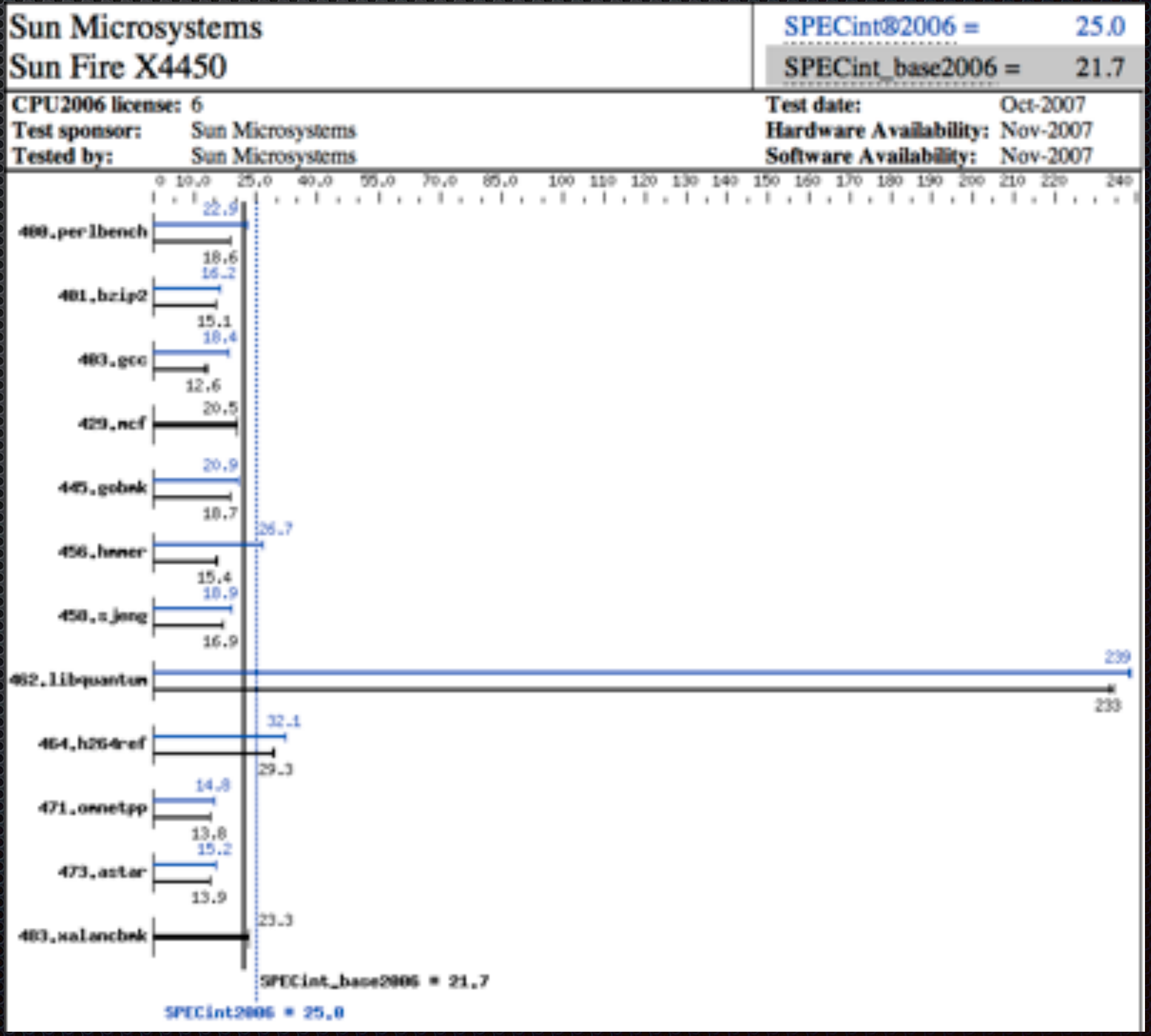
So?

Consider this example:

23.6



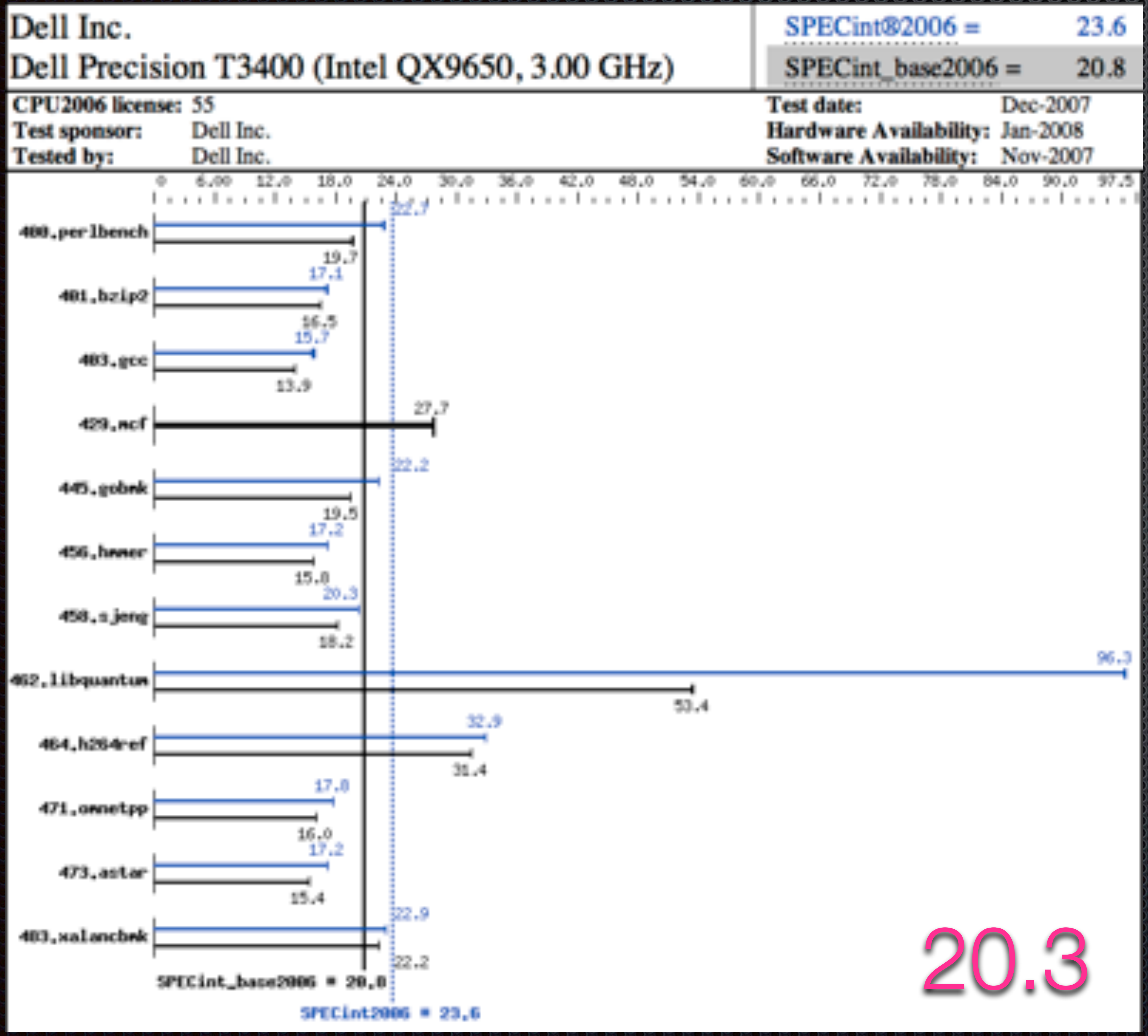
25.0



So?

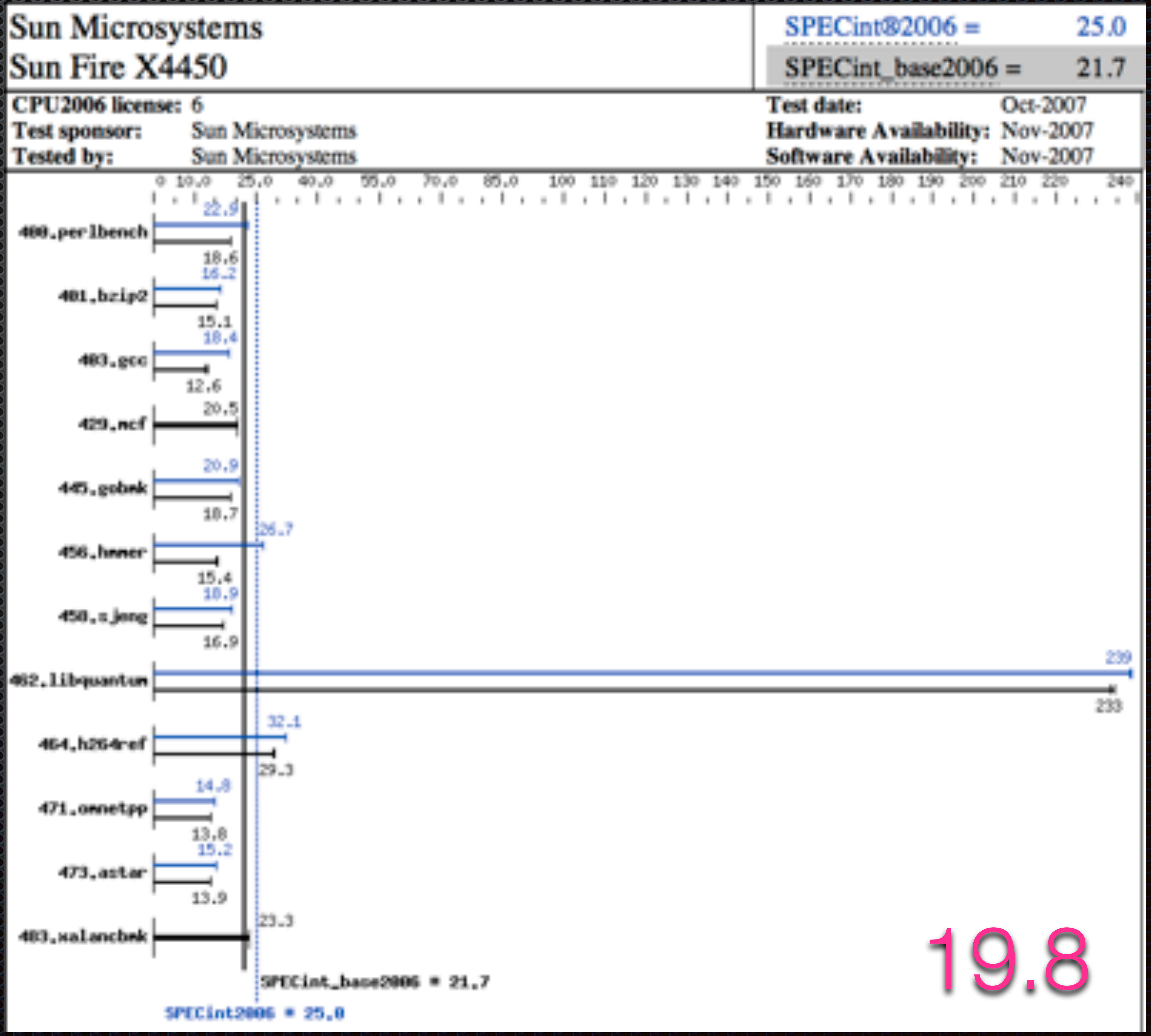
How does deleting the outlier and using the harmonic mean change the results?

23.6



20.3

25.0



19.8

"Benchmark Engineering"

- There are obvious ways to enhance performance using the SPEC CPU peak rules:
  - Profile directed feedback
  - Special libraries
  - Unsafe optimizations
  - Different optimization options
  - Assertions to guide optimization
- What else can you think of?

“Benchmark Engineering”

- Single user/diagnostic mode
- Strip-down kernel to minimum services
- Disable network interface, user I/O
- Lengthen OS quantum
- Hand pick processor board and memory
- Use fastest disk (15K RPM or SSD)
- Reformat disk with longer sectors
- Make compiler recognize benchmarks
- Turn off multithreading
- Specially cool processor chip

“Benchmark Engineering”

Commercial benchmarks report results that you are guaranteed never to exceed (or even match)

Amdahl's Law

- Gene Amdahl
- Architect for IBM 709, Stretch, 360
- Left IBM to form his own company, building IBM mainframe "clones"
- Observed that speeding up one aspect of an architecture has limited value

Amdahl's Law

- Overall Speedup =
- 1/((1-Percent affected)+ Percent affected/Speedup)
- Even if X% of a processor's performance is improved infinitely, only X amount is removed from the total
- The remaining 1-X% dominates
  - If 99% disappears, 1% remains, so at most 100X speedup

Desikan

- Validation of software simulation of architecture
- Compares real Alpha to simulations
- Identifies sources of error with microbenchmarks
- Shows results with macrobenchmarks

Simulator Error

	Alpha 21264	Initial simulator (sim-initial)		Validated simulator (sim-alpha)		SimpleScalar 3.0b (sim-outorder)	
benchmark	IPC	IPC	% error	IPC	% error	IPC	% difference
C-Ca	1.80	0.38	-498.1%	1.87	4.3%	3.17	28.2%
C-Cb	1.87	0.52	-260.4%	1.87	0.6%	3.00	37.8%
C-R	2.65	0.89	-198.4%	2.66	0.3%	3.54	25.2%
C-S1	0.56	0.81	31.2%	0.60	6.4%	0.88	36.1%
C-S2	0.85	0.82	-3.6%	0.86	2.1%	1.33	36.5%
C-S3	0.95	0.87	-8.5%	0.95	0.5%	1.64	42.2%
C-CO	1.75	0.53	-273.6%	1.74	-0.6%	2.05	3.0%
E-I	4.00	3.31	-20.9%	3.99	-0.4%	3.99	-0.4%
E-F	1.01	1.01	-0.1%	1.01	0.2%	1.01	0.2%
E-D1	1.03	1.04	0.3%	1.04	0.4%	1.04	0.4%
E-D2	2.16	2.15	-0.0%	2.15	0.0%	2.21	2.6%
E-D3	2.72	2.99	9.3%	3.07	11.5%	3.19	14.8%
E-D4	2.79	2.89	3.6%	2.80	0.3%	4.00	30.2%
E-D5	3.30	3.23	-2.1%	3.50	5.8%	4.00	17.6%
E-D6	3.11	3.31	6.1%	3.15	1.3%	4.00	22.2%
E-DM1	0.15	1.04	85.7%	0.15	-0.3%	0.15	-0.3%
M-I	2.98	2.39	-24.2%	2.99	0.6%	3.00	0.7%
M-D	1.66	1.25	-32.9%	1.66	0.4%	1.26	-31.1%
M-L2	0.36	0.34	-4.0%	0.35	-0.9%	0.55	35.6%
M-M	0.07	0.07	-8.2%	0.08	4.2%	0.07	-0.3%
M-IP	1.75	0.89	-97.9%	1.76	0.5%	1.22	-43.1%
Mean			74.7%		2.0%		19.5%

Table 2: Microbenchmark validation

Simulator Error

	gzip	vpr	gcc	parser	eon	twolf	mesa	art	equake	lucas	mean
Alpha 21264 IPC	1.53	1.02	1.04	1.18	1.21	1.10	1.57	0.48	1.02	1.57	1.05
sim-alpha IPC	1.28	0.99	0.90	0.97	1.21	1.07	1.17	0.82	0.94	1.37	1.05
% error	-22.01	-4.63	-18.07	-23.09	-0.92	-6.07	-38.37	43.04	-10.94	-14.74	18.19
sim-stripped IPC	1.07	0.74	0.84	0.89	0.96	0.84	1.04	0.82	0.83	1.44	0.92
% difference	-51.52	-44.12	-42.33	-42.01	-34.10	-42.09	-62.10	39.75	-32.71	-9.96	40.07
sim-outorder IPC	2.28	1.62	1.89	2.00	2.08	1.76	2.59	2.14	1.69	1.79	1.95
% difference	28.56	34.04	37.20	37.05	38.29	32.25	36.80	76.89	34.60	11.54	36.72

Table 3: Macrobenchmark validation

Discussion

Hill CAECW 2002

- Commercial workloads are different
- Big memory and disk
- Nondeterminism
- Benchmarks run for hours

OLTP

- Database benchmark
- Reduce size
- Zero think time
- Super-fast disk
- 10K transaction warm-up (real machine), 1K run (sim)

SPECjbb

- Transaction processing in Java
- 1.8GB heap to minimize GC
- 500MB data per warehouse
- 100K warmup, 100K run

Apache

- 10 SURGE clients per processor
- Zero think time
- 2K file repository with 50 MB
- 80K warmup, 2.5K run

Slashcode

- Dynamic web page generation
- 3K messages, 5 MB total
- 240 transactions warmup, 50 run

Barnes-Hut

- N-body Simulation
- Numerical benchmark for comparison
- 64K bodies

The Workload

Table 1. Workload properties

Workload	Memory blocks touched (64 bytes)	Unique miss PCs	L2 cache misses per 1000 instructions	Supervisor misses (% of total)	Time Spent in Kernel (% of total)
OLTP	57 MB	12136	3.0	43%	28%
SPECjbb	353 MB	8163	3.2	15%	1%
Apache	102 MB	10214	2.9	82%	84%
Slashcode	173 MB	17009	1.1	48%	43%
Barnes-Hut	16 MB	3413	0.3	16%	3%

Variation

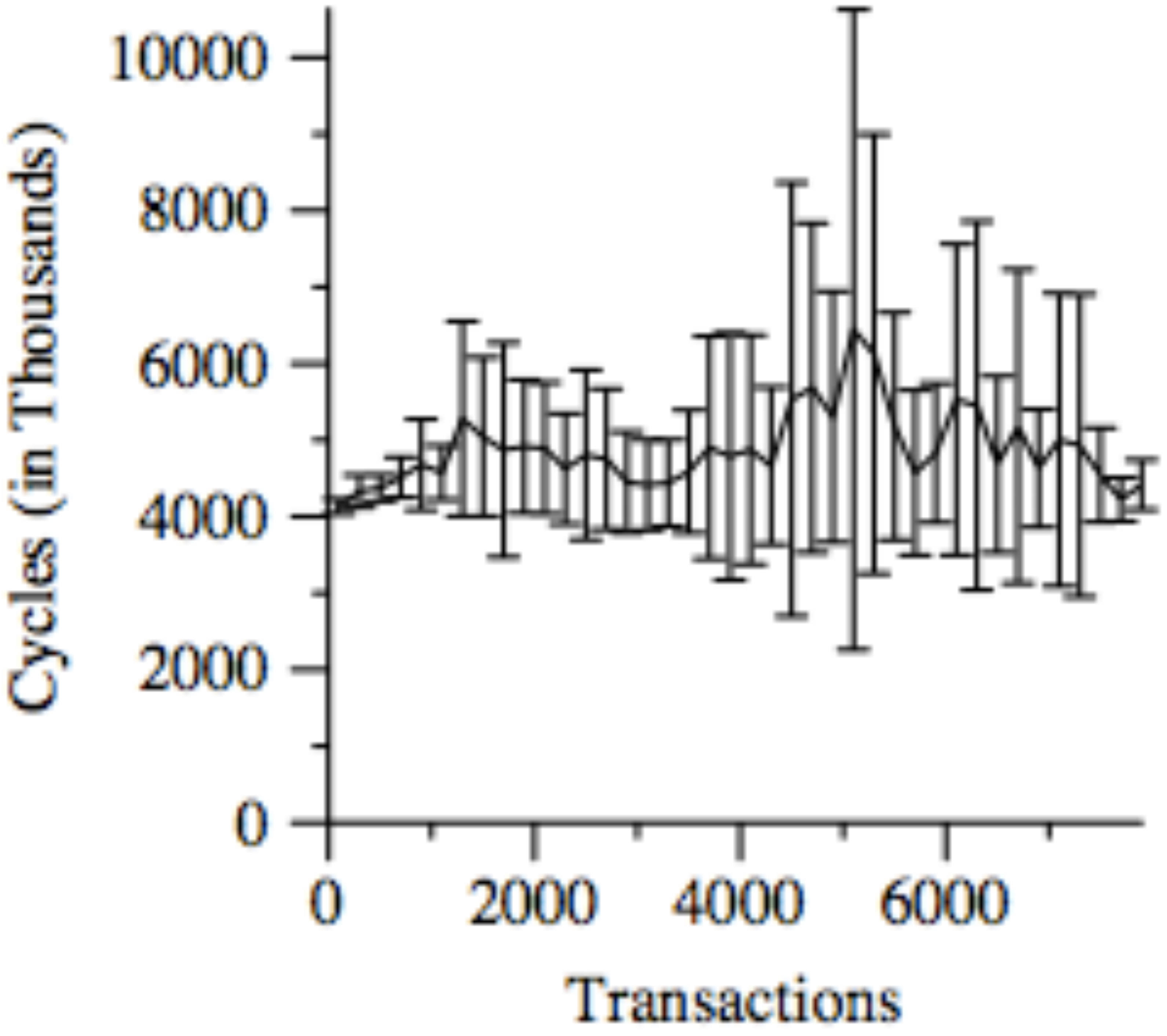


Figure 4. Cycles per transaction

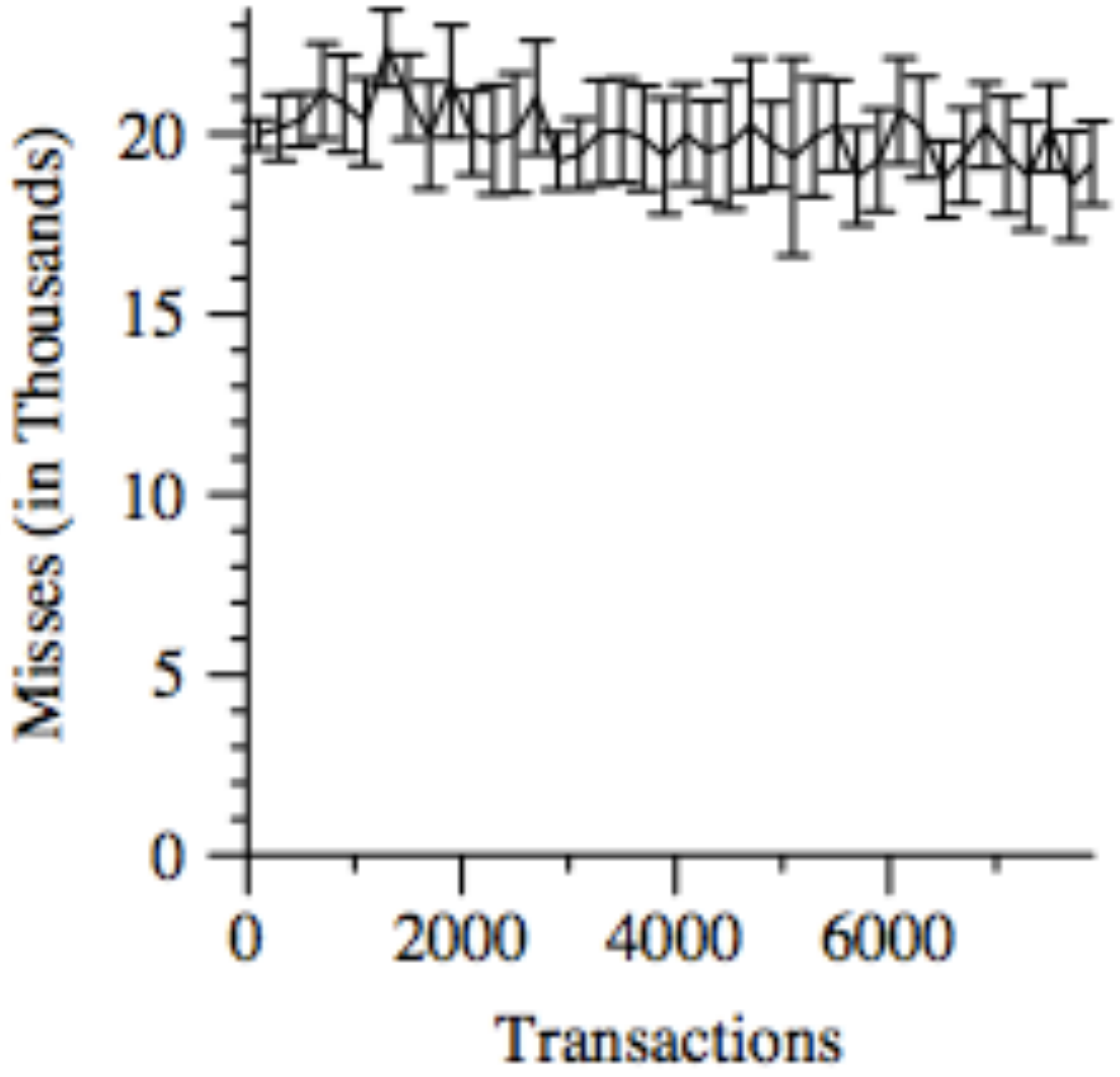


Figure 5. Misses per transaction

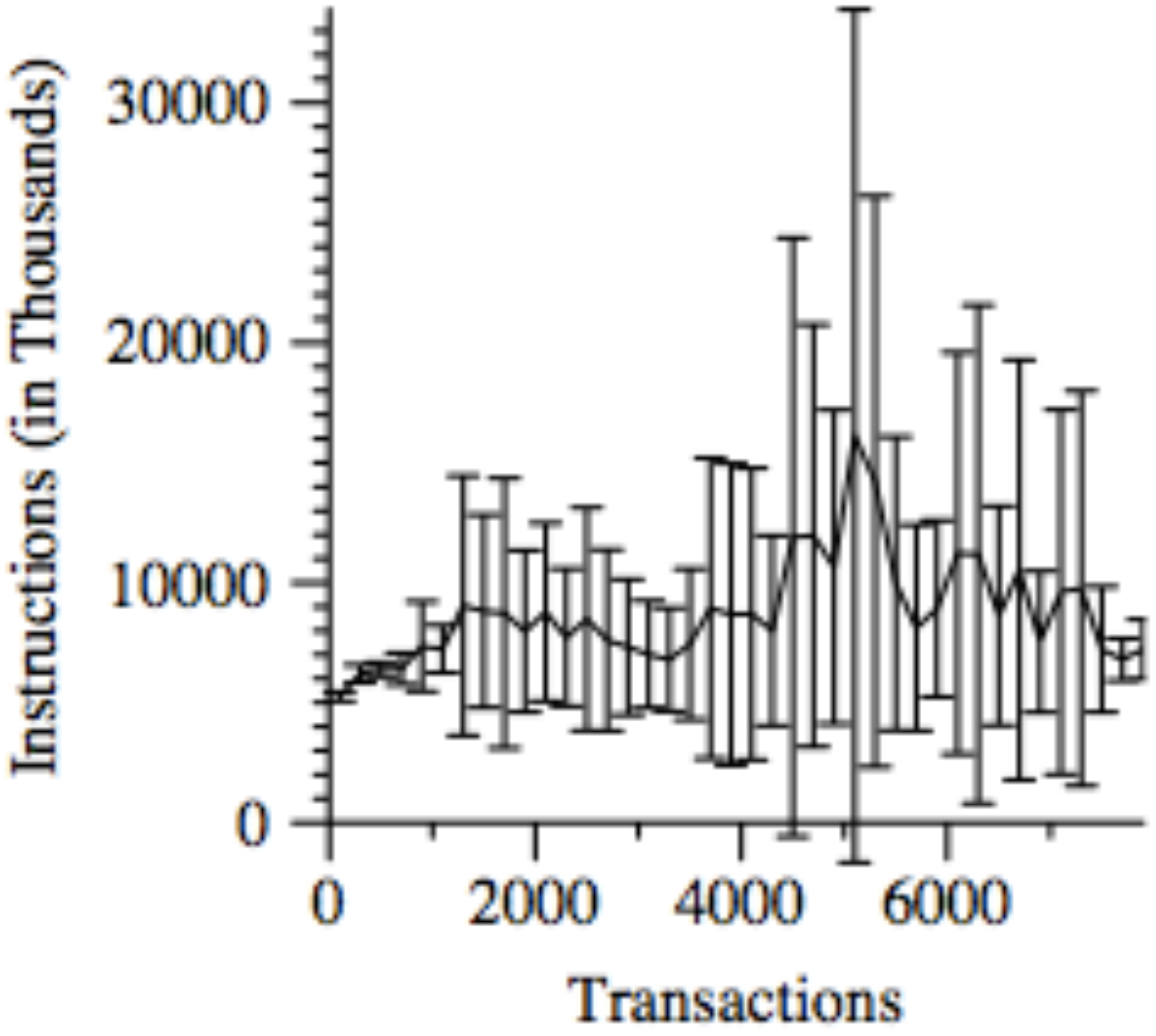


Figure 6. Instructions per transaction

Discussion