Benchmarking Computers for High Energy Physics Computing

Yang Suli

yangsuli@gmail.com

Department of Physics, Peking University, Beijing, 100871

Abstract:

Today's high energy physics requires enormous computing resources, so there is a common desire for a standard benchmarking method, which gives a standard and simple evaluation to computers' capability to do high energy physics computing. Traditionally, the industry standard benchmark suite SPEC CPU2000 is used. But this benchmark suite has been retired, and is not representative to modern computers anymore.

In this paper I will examine several aspects of CPU2006----SPEC's next generation, CPU intensive benchmark suite; compare it to real HEP applications; and try to establish a fairly accurate and convenient way to evaluate computing powers that are used to do data analysis in high energy physics. Also, I will use the established method to benchmark several machines with different configurations, so that the effect of different factors on high energy physics computing, such as the operating system and the processors' effect, can be shown.

Content:

0. Acknowledgment

I hereby wish to thank my supervisor Dr. Peter Wegner, for his support and constructive feedback at various opportunities during the work.
Some data cited in this paper is obtained by Dr. Michele Michelotto, Alejandro Iribarren and other members in the HEPIX group. The private discussion in their mail list is also very helpful. I want to give my special gratitude to them.
I would also like to thank Mr. Goetz Waschk and Mr. Stephan Wiesand for their kind help to my work.

1. Introduction

Today's high energy physics requires enormous computing resources, so there is a common desire for a standard benchmarking method, which gives a standard and simple evaluation to computers' capability to do high energy physics computing.
Traditionally, the industry standard benchmark suite SPEC CPU2000 is used. But this benchmark suite has been retired, and is not representative to modern computers anymore. CPU2006 is the next generation, industry-standardized, CPU intensive benchmark suite supported by SPEC, which gives it the potential to serve as the HEP standard benchmark. However, high energy physics experiments applications have their own characteristics. This benchmark must be adjusted to our own needs in order to properly represent computers' capability to do high energy physics computing. In this paper I will examine several aspects of CPU2006, compare it to real HEP applications, and try to establish a fairly convenient and accurate method to do high energy physics benchmarking.
The rest of this paper is organized as follows: in section 2 I give a brief introduction to the SPEC CPU2006 benchmark suite and the platforms under test. The accidental error fluctuation of the results is discussed in section 3. Then in section 4 several properties of CPU2006 are studied, corresponding adjustments are given and the standard method for HEP benchmarking is established. I also discuss the conversion from CPU2000 results to CPU2006 results in section 4. The comparison of the standard benchmark results to real HEP application performance is given in section 5. In section 6 this standard method is used to analyze operating system and processor's effect on high energy physics computing performance. Finally I conclude in section 6.

2. The SPEC CPU2006 benchmark suite and platforms for test

There has already been an agreement of using an industry standard benchmark suite, instead of using codes from experiments, to evaluate the performance of computers, though we may use it in a nonstandard way. There are several reasons for such a decision. First, having an industry standard benchmark saves us a great deal of trouble of maintaining it. We will have the benchmark venders to fix the bugs and port it to new compliers and operating systems for us. We can expect that an industry benchmark to be maintained for 5, or even 10 years. In contrast, the codes from experiments are relatively unstable, have a much shorter lifetime, and are hard to run. Second, a cluster of machines can be used for various experiments computing, e.g. a machine runs both CMS and ATLAS analysis, which makes it very difficult to choose a representative collection of codes from experiments as benchmark suite. Also, using an industry standard benchmark makes the communication with hardware venders easier.

2.1 The SPEC CPU2006 benchmark suite

SPEC CPU2006, an industry-standardized, CPU-intensive benchmark suite which provides comparative measure of compute-intensive performance across different hardware platforms, seems to be a good choice. This benchmark suite stresses a system's processor, memory subsystem and compiler, which are the key factors in high energy physics computing. Also, this benchmark suite is widely accepted in the industry world. The basic idea of this benchmark is to run several real user applications as benchmarks, including some physics applications like QCD computing; to compare the execution time of each benchmark to a reference machine (a SUN SPARC); to decide the score of each benchmark by calculating how many times faster the tested machine executes a given benchmark than the reference machine; and finally, to take the geometric mean of all the benchmarks' scores as the final result. There are two basic components of this benchmark suite: CINT2006, which tests the integer performance; and CFP2006, which tests the floating-point performance. The ALL_CPP bset is also heavily used, which will be discussed in its respective section. More technical details can be found in [1].
However, as I mentioned before, just copying the industry benchmark method is not suitable to evaluate the computing power in high energy physics world. Especially, the results in the SPEC website published by the venders are pointless to us. Because these results are obtained by carefully setting up the environment and highly optimizing the generated binary to get the highest score, which is not representative to the production runs in high energy physics.
For the above reasons, we will use the standard SPEC CPU2006 benchmark suite, but in a nonstandard way, which will be explained later.

2.2 Platforms for test

The platforms used to perform the benchmarking are listed in Table 1. They are all x86_64 machines, with typical modern configurations. Scientific Linux [2] is used because this is the dominating operating system in high energy physics computing. We use a 64-bit OS because that is how our production machines are installed. All the new hardware is 64bit capable and we're not interested in running a 32-bit OS [3]. Several typical Intel and AMD architectures are investigated in order to get an impression of modern commodity processors' performance in high energy physics.
This paper has also cited some external benchmark results in order to enlarge the statistics. Those benchmark runs are conducted in machines from CERN and INFN,

whose parameters are listed in Table 2.

2.3 Complier flags and configuration files

The complier used is gcc with tuning "-m32 –O2 –fPIC –fpthread". These are the flags mandated by the LCG board, and are supposed to be the common flags used in the production runs. We use "-m32" to produce 32-bit mode applications because some 32-bit-only resources are still used in the experiments codes, and 32-bit application on a 64-bit OS represents a usual environment.

To control the way SPEC CPU2006 benchmark suite runs, one need a config file to specify the compliers for the source codes, the optimization flags, the selection of benchmarks, and various other settings. To unify the benchmark settings so that one can get comparable results in different systems to represent typical high energy physics computing performance, the hepix group has offered a reference config file, which is used in our benchmark runs [3].

hostname	Processors	Memory(GB)	System	Compiler
hpbl1.ifh.de	Intel Xeon E5440 Quad Core * 2 2.83GHz	16	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
hpbl2.ifh.de*	Intel Xeon E5440 Quad Core * 2 2.83GHz	16	Scientific Linux 5.2 64bit (kernel 2.6.18)	gcc 4.1.2
hpbl3.ifh.de	AMD Opteron 2356 Quad Core * 2 2.30GHz	16	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
hpbl4.ifh.de	AMD Opteron 2356 Quad Core * 2 2.30GHz	16	Scientific Linux 5.2 64bit (kernel 2.6.18)	gcc 4.1.2
qftquad3.ifh.de	Intel Xeon E5450 Quad Core * 2 3.00GHz	32	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
Blade5a.ifh.de	Intel Xeon E5450 Quad Core * 2 3.00GHz	32	Scientific Linux 5.2 64bit (kernel 2.6.18	gcc 4.1.2

Tab. 1: Platforms for Test (inside DESY)

*On hpbl2, Scientific Linux 4.6 and Scientific Linux 5.2 have in turn been installed. I will state it explicitly when citing results obtained on hpbl2 using SL4.6.

hostname	Processors	Memory(GB)	System	Compiler
Povada	AMD Bacerlona Quad Core * 2 2.50GHz	16	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench01	Intel Nocona Single Core * 2 2.80GHz	2	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench02	Intel Irvingdale Single Core * 2 2.80GHz	4	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench03	AMD Opteron 275 Dual Core * 2 2.20GHz	2	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench04	Intel Woodcrest Dual Core * 2 2.66GHz	8	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench05	Intel Woodcrest Dual Core * 2 3.00GHz	8	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench06	AMD Opteron 2218 Rev.F Dual Core * 2 2.60GHz	8	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench07	Intel Clovertown Quad Core * 2 2.33GHz	16	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)
lxbench08	Intel Harpertown E5410 Quad Core * 2 2.33GHz	16	Scientific Linux 4.6 64bit (kernel 2.6.9)	gcc 3.4.6 (gcc 4.1.2 for Fortran part)

Tab. 2: Platforms for Test (outside DESY)

3. Discussion of accidental errors

Even if benchmarks are runned in machines with the same configurations, or even in the same machine, one may get results with slight differences. These accidental errors are introduced because of the differences in different batches of hardware products, the unpredictable disturbance of the runtime environment, or various other reasons. In order to correctly compare results from different benchmark runs, we must take into account the accidental error fluctuations.

To decide the accidental errors in benchmark results, benchmark runs are repeated several times in the same machine, as well as in the machines with same configurations. The comparison is showed in Table 3 and Table 4.

	RUN_1	**RUN_2**	**RUN_3**	**Standard Deviation**	**Greatest Difference (%)**
INT	61.27	61.36	61.15	0.105357	0.3%
FP	52.59	51.49	51.79	0.568624	2.1%
ALL_CPP	60.67	59.63	59.4	0.676683	2.1%

Tab. 3: benchmark results obtained in the same machine.
Machine hpbl3 (AMD Opteron 2356, SL4) is used.

	hpbl1	**hpbl2**	**Difference (%)**
INT	72.92	73.03	0.2%
FP	51.5	52.66	2.3%

Tab. 4: benchmark results obtained in machines with the same configurations.
Machine hpbl1 and hpbl2 (Intel E5440, SL4) are used.

From the data we can find that while the INT benchmark results keep relatively stable (deviations no more than 0.5%), fluctuation of FP or ALL_CPP benchmark results can get up to 2%-3%. So a difference less than 3% in benchmark results should be considered as accidental error fluctuation, which does not indicate a real inequality.

4. The standard HEP benchmarking method

4.1 Multispeed vs. Rate

SPEC offers two modes to run benchmark on multi-core machines. The SPEC speed benchmark runs a single copy of the benchmark on the machine, using only a single core, which hardly presents the production environment. The SPEC rate benchmark runs as many benchmarks as there are cores, but it calculates the results as a function of the total elapsed time. This skews the results, as a single slow core can keep the others idle until it finishes.

To overcome these problems, we decided to run multiple speed benchmarks in parallel. We launch one independent speed benchmark for each core in the system and then add up the results to come up with a total result for the system. This mimics what we do in our production environment, where we run multiple independent single-threaded applications in parallel. The graphical explanation of the difference between SPEC rate and multispeed is showed in Fig. 1 [3].

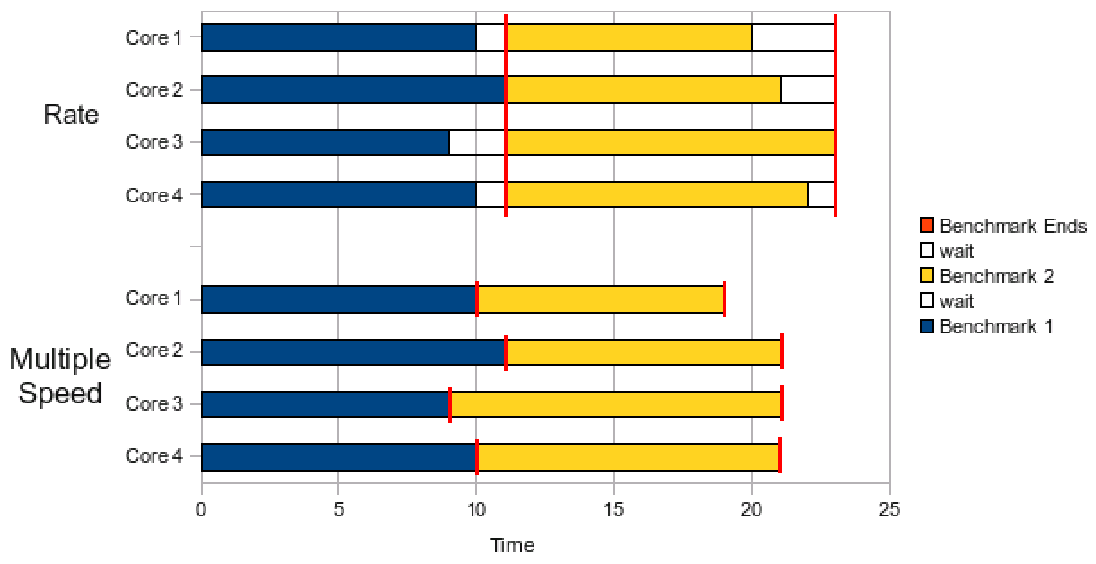


Fig.1: Graphical explanation of multi-speed mode

To understand the difference between rate and multispeed method, some tests are runned. The results are shown in Table 5 and Figure 2.

machine	Architecture	number of cores	INT32 Multispeed	INT32 RATE	FP32 Multispeed	FP32 RATE
lxbench01	Intel	2	11.06	11	9.5	9.42
lxbench02	Intel	2	10.09	10.1	7.7	7.66
lxbench03	AMD	4	28.76	27.8	25.23	23.9
lxbench04	Intel	4	36.77	36.2	27.85	27.4
lxbench05	Intel	4	39.39	38.9	29.72	29.3
lxbench06	AMD	4	31.4	32.3	27.82	27.2
hpbl1	Intel	8	72.92	69.8	51.5	48.4
hpbl2(SL4)	Intel	8	73.03	69.8	52.66	48.3
hpbl3	AMD	8	61.27	62.1	52.59	53
hpbl4	AMD	8	62.98	63.6	53.05	53

Tab. 5: Comparison of multi-speed and rate modes runs

From the results we can find that there is significant difference between Multispeed and Rate for benchmarking in Intel machines. But that's not the case for AMD machines. Also, we can see that for fewer cores in the box, the differences between the multispeed and rate are smaller. This fact can be easily understood in the way that for fewer cores, the time used for the idle cores to wait the slower one is less.

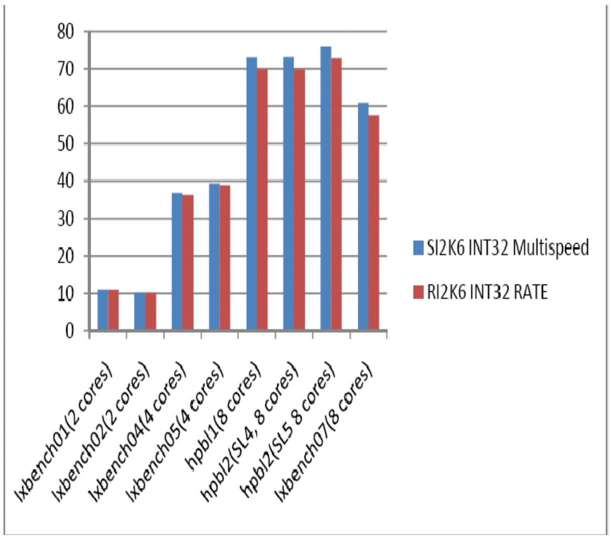


Fig. 2.a: Intel INT multispeed vs. rate

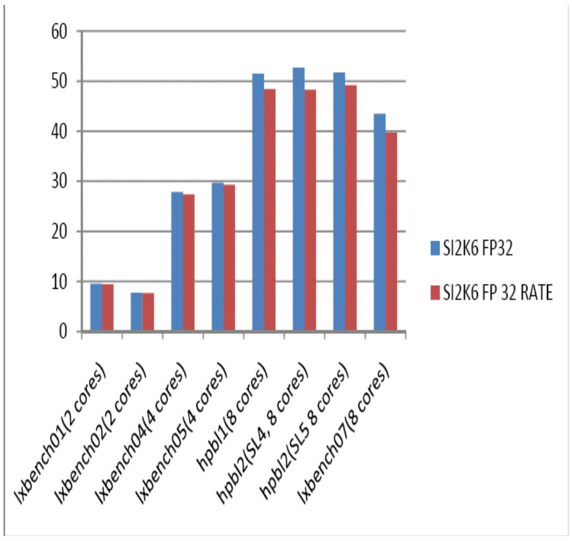


Fig. 2.b: Intel FP multispeed vs. rate

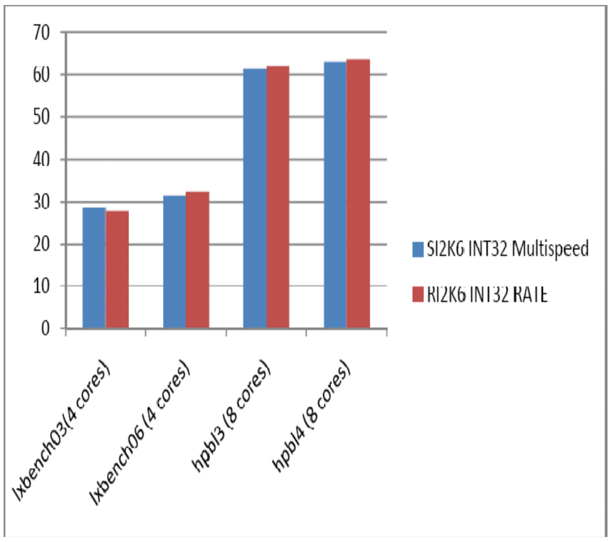


Fig. 2.c: AMD INT multispeed vs. rate

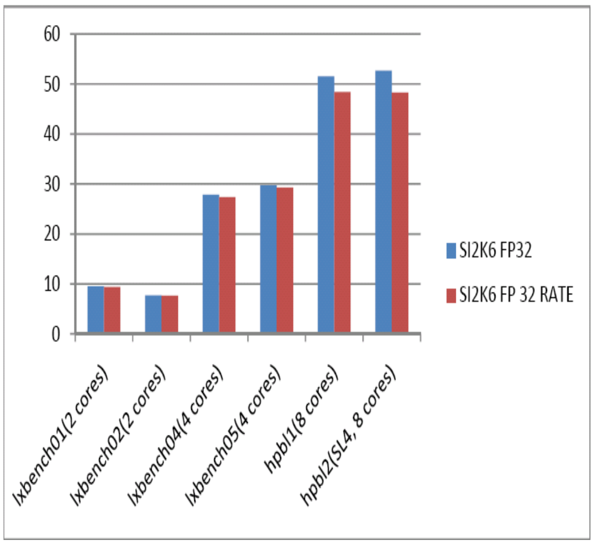


Fig 2.d AMD FP multispeed vs. rate

Fig. 2: Comparison of multi-speed and rate modes runs

4.2 The ALL_CPP bset

SPEC CPU2006 offers two basic benchmark suites: CINT2006 and CFP2006, which measure the integer and the floating-point performance, respectively. But in our production runs, both integer and floating-point computing are heavily used. That is why we introduce the ALL_CPP bset suite, which is simply all the C++ benchmarks contained in CPU2006, to measure the overall performance of high energy physics computing. There are seven benchmarks in the ALL_CPP bset, 3 of them are from CINT, while the others 4 are from CFP suite. To obtain the ALL_CPP benchmark result one must take the geometric mean of the 3 integer benchmark results, and of the 4 floating-point benchmark results, and then take an average of these two numbers. The advantages of using ALL_CPP suite are as follows:

1) By using ALL_CPP we are incorporating floating-point part as well as the integer part, and in same ratio as the HEP experiments are showing now. Low level CPU statistics shows that the ALL_CPP suite contains about 10%-14% floating-point instructions, which is in good agreement with HEP codes' roughly 10% floating-point content. In contrast, the CINT suite contains only 0.1% of floating-point instructions, which is negligible*. The measurement also shows that ALL_CPP suite is a good match of HEP applications in the sense that its loads and mispredicted branch ratio is very similar to the production environment [4].
2) Compared to the CINT and CFP suite, running ALL_CPP suite saves significant time. A simple study shows that in a machine with Intel Xeon E5430 processor (Harpertown 2.66 GHz), the total elapsed time of CINT2006 is 48334s (13:20h), while for ALL_CPP suite this time is only 19789s (5:30h). This means that ALL_CPP runs at about 2.5 times faster than CINT, and we can benchmark the same machine in 40% of the time [5].

The disadvantage is that the set of applications in CINT or CFP is carefully selected by SPEC, but with ALL_CPP the selection is pretty random: just all the C++ benchmarks. In fact, for integer computing, the C++ benchmarks tend to have a lower performance than average, while for floating-point computing the C++ benchmarks tend to have a higher performance. However, in the section 5 we will see that ALL_CPP benchmark has a better agreement to experiments than any other benchmark else, which means the advantage overcomes the disadvantage.

Concerning the method used to obtain the final results, there were two options: geometric mean and arithmetic mean of the benchmark results. In principle, geometric mean of all the 7 benchmark results is mathematically more correct, because geometric mean has the property that a certain percentage change in any one of the terms has the same effect as the same percentage change in any of the other terms. In other words, no one benchmark (or no one set of benchmarks) will become more important than any of the others in the suite. In contrast, an arithmetic mean may give unfair weight to one single benchmark if this benchmark has higher score than the others in one machine. This is why geometric mean is traditionally used by SPEC [6]. However, compared to arithmetic mean, calculating the geometric mean requires significant more effort and more complex scripts to analyze the resulting SPEC log files; that goes against our wish of "making things simple and stupid". We use a method that may be less correct in the statistical sense, but is much easier to conduct: to take the results of the INT and FP part in ALL_CPP bset, which are given directly by the SPEC program, then to calculate the arithmetic mean of these two. This method introduces two deviations from the "correct" way:

1) We calculate the arithmetic mean instead of the geometric mean.
2) Since we have 3 INT tests and 4 FP tests we should make a weighted mean instead of non-weighted mean. The method currently used gives INT part about 7% more credit than it deserves.

We prefer this method because despite all those deviations above, it still gives results very close to that given by the correct but complex way. The comparison of all kind of methods is listed in Table 6. We can find that the deviation of results of this “quick and dirty” method gets no more than 1%, not even reach the accidental error fluctuation, well below the accuracy we aim for.

machine	CINT results	CFP results	geometric mean of all benchmarks in ALL_CPP bset (A)	arithmetic mean of INT and FP part in ALL_CPP bset (B)	(B)/(A)
hpbl1	72.34	52.41	68.92	68.34	0.992
hpbl2	76.03	51.72	70.34	69.85	0.993
hpbl3	61.15	51.79	61.10	60.64	0.992
hpbl4	62.89	53.03	61.77	61.38	0.994
Padova	64.41	--	64.08	63.89	0.997
lxbench01	11.06	9.50	10.27	10.23	0.996
lxbench02	10.09	7.70	9.66	9.57	0.991
lxbench03	28.76	25.23	28.18	27.92	0.991
lxbench04	36.77	27.85	35.39	35.17	0.994
lxbench05	39.39	29.72	37.98	37.82	0.996
lxbench06	31.40	27.82	30.61	30.35	0.992
lxbench07	60.89	43.37	57.45	56.96	0.991
lxbench08	64.78	46.48	60.74	60.15	0.990

Tab. 6: Comparison of benchmark results obtained in different ways

machine	Benchmark set	V1.1 result	V1.0 result	Difference
hpbl1	int	72.92	73.59	0.9%
hpbl2	int	73.03	73.1	0.1%
hpbl3	int	61.27	62.85	2.6%
hpbl4	int	62.98	64.41	2.3%
hpbl1	fp	51.5	52.03	1.0%
hpbl2	fp	52.65	52.98	0.6%
hpbl3	fp	52.59	54.09	2.9%
hpbl4	fp	53.03	51.14	-3.6%
hpbl1	all_cpp	67.75	68.38	0.9%
hpbl2	all_cpp	69.81	69.83	0.0%
hpbl3	all_cpp	54.91	60.67	10.5%
hpbl4	all_cpp	61.92	61.37	-0.9%

Tab. 7: Comparison of V1.1 and V1.0 results

4.3 Version 1.0 vs. 1.1

The current version of SPEC 2006 is V1.1, which is the recommended version by SPEC organization. This version is an incremental update to V1.0, intended to improve compatibility, stability, documentation and ease of use, but not to change the benchmark results [6]. So the results generated by SPEC CPU2006 V1.0 should be comparable to results from V1.1, as claimed by SPEC. To confirm this I had some test runs. The results are shown in Table 7 and Figure 3.

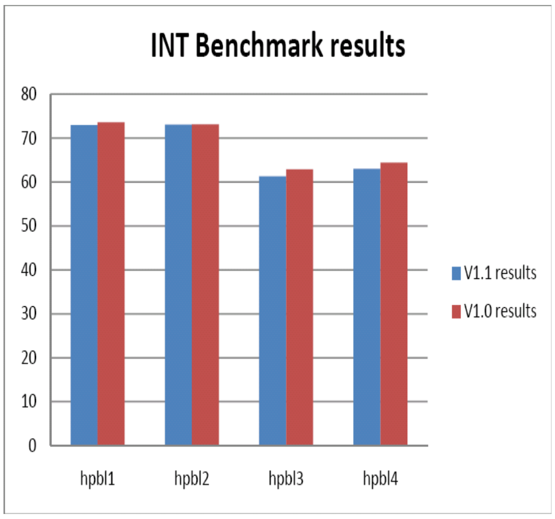


Fig. 3.a: INT results

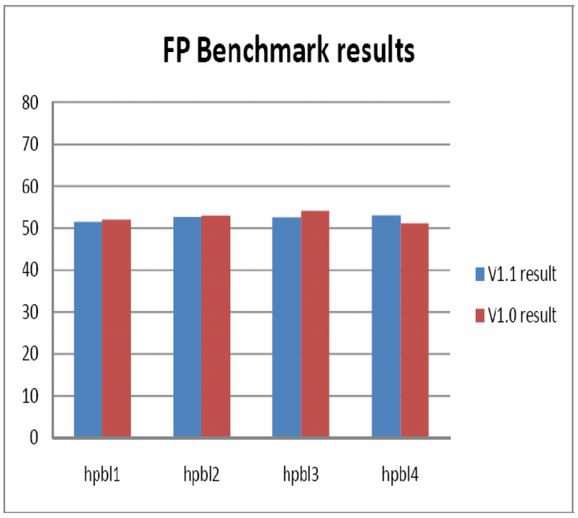


Fig. 3.b: FP results

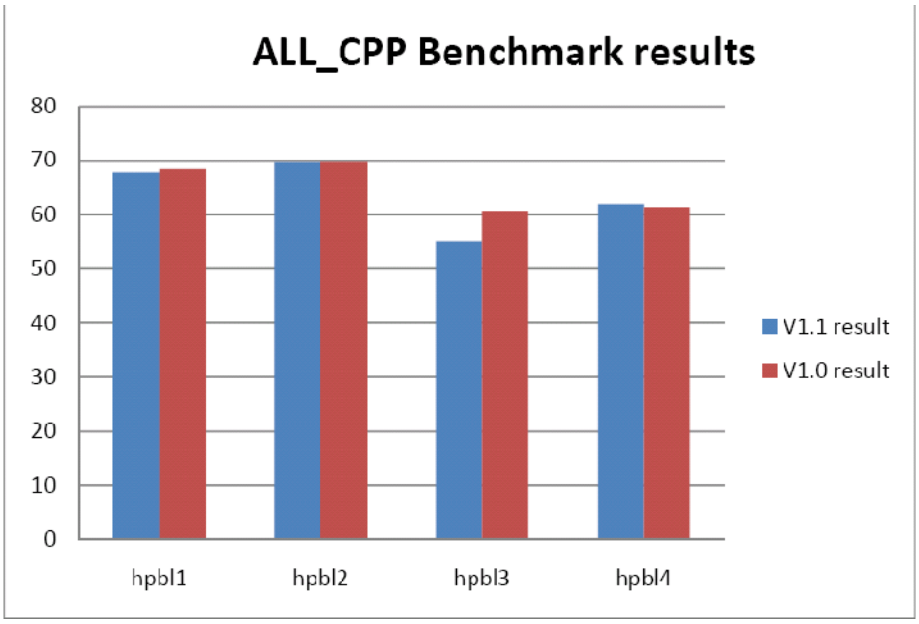


Fig. 3.c: ALL_CPP results

Fig. 3: Comparison of V1.1 and V1.0 results

From the results we can find that except for one special case, the difference between V1.1 and V1.0 results are in the range of 0%-3%, i.e. inside the fluctuation range. So we can safely say that the results from SPEC CPU2006 v1.1 and 1.0 are comparable.

4.4 Conversion from CPU2000 to CPU2006 results

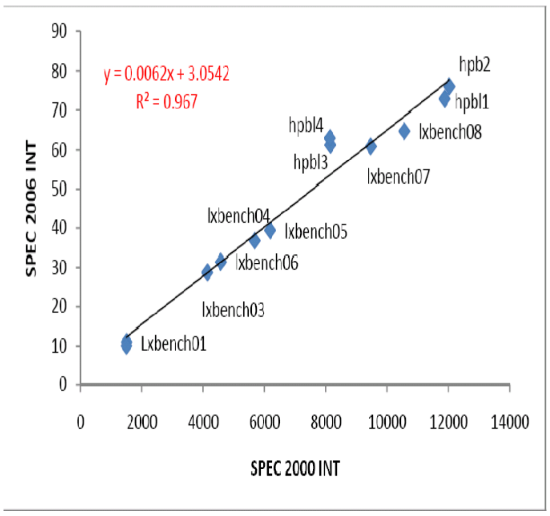
SPEC CPU2000 is the last generation of benchmark suite used to benchmark high energy physics computing power. It is no longer supported by SPEC; and the results given by SPEC CPU2000 are not very representative for current computers. However, since there are some CPU2000 INT results available for relatively older machines, it is still our interest to give a conversion from CPU2000 to CPU2006 results to make them, at least in some sense, comparable.

In table 8 I list the SPEC CPU2000 and CPU2006 results obtained in different machines. Linear fits of these data are shown in Figure 4.a and 5.a. We can find that the linear correlation between CPU2000 and CPU2006 results is significant (linear correlation coefficient $R^2 = 0.967$ or 0.942). Especially we notice that the data points of hpbl3 and hpbl4 remarkably deviate from the trend line (Notice that both abnormal values are coming for AMD machines, for whatever reason). After excluding those data points that obviously violate the linearity, one can get a much better linear fit (linear correlation coefficient R^2 gets up to more than 0.99), which are shown in Fig 4.b and 5.b. So for conversion from CPU2000 INT results to CPU2006 INT results, we have $y = 0.006x + 2.458$; where y is the CPU2006 INT benchmark value, and x is the CPU2000 INT benchmark result. For conversion from CPU2000 INT results to CPU2006 ALL_CPP results, a similar conversion method $y = 0.0056x + 3.1618$ is given. Actually in practice, a simple $y = 0.006x + 3$ would be sufficient, for both INT and ALL_CPP conversion, where y is the CPU2006 value, and x is the CPU2000 result.

MACHINE	SPEC 2000 INT	SPEC2006 INT	SPEC2006 ALL_CPP
lxbench01	1501	11.06	10.23
lxbench02	1495	10.09	9.57
lxbench03	4133	28.76	27.92
lxbench04	5675	36.77	35.17
lxbench05	6181	39.39	37.82
lxbench06	4569	31.44	30.34
lxbench07	9462	60.89	56.96
lxbench08	10556	64.78	60.15
hpbl1	11873	72.92	68.34
hpbl2	12022	76.03	69.85
hpbl3	8146	61.27	60.64
hpbl4	8133	62.98	61.38

Tab. 8: Conversion from CPU2000 to CPU2006 results

Through the above discussion we have established the standard procedure to do high energy physics benchmarking: using either SPEC CPU2006 v1.0 or v1.1, launching ALL_CPP suite in the multispeed mode, and then calculating the arithmetic mean of the results of integer part and floating-point part as the final benchmark result. The proposal of making it the official HEP benchmark method has already been submitted to the LCG board.



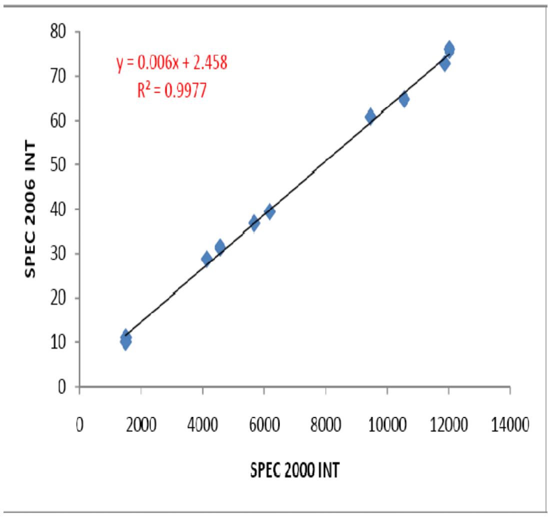
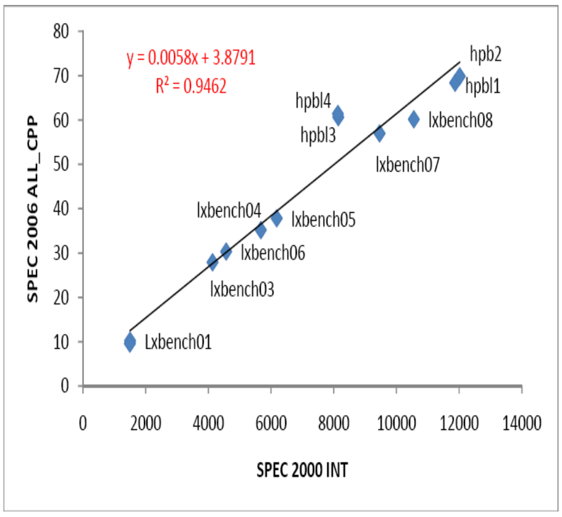


Fig. 4.a

Fig. 4.b

Fig. 4: Conversion from CINT2000 to CINT2006 results



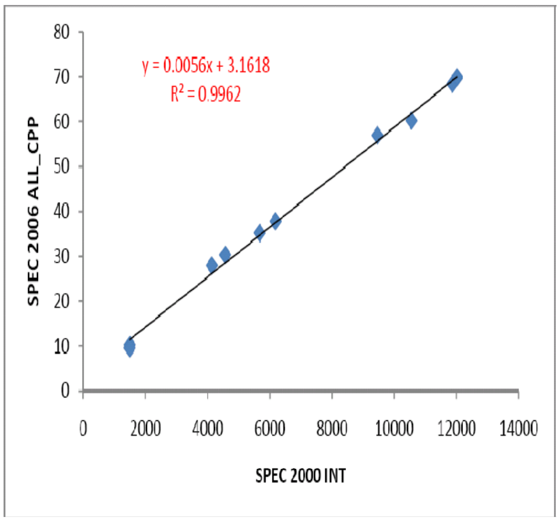


Fig. 5.a

Fig. 5.b

Fig. 5: Conversion from CINT2000 to SPEC CPU2006 ALL_CPP results

5. Comparison of SPEC benchmark results to HEP code performance

To measure the agreement of the benchmark results to real HEP application performance, three major experiments in LHC are selected: ATLAS, CMS and ALICE. The elapsed time of generation, simulation, digitization and reconstruction of typical physical processes are measured, and the Pearson product-moment correlations of the performance to the benchmark results are calculated and listed in Table 9.

Test	SPEC CINT2000	SPEC CINT2006	SPEC CFP2006	SPEC ALL_CPP
ATLAS Generation	0.645	0.651	0.693	0.743
ATLAS Simulation	0.679	0.686	0.737	0.743
ATLAS Digitization	0.726	0.729	0.76	0.771
ATLAS Reconstruction	0.691	0.706	0.752	0.822
ATLAS Total	0.685	0.692	0.743	0.751

Tab. 9.a: Correlations of ATLAS experiment codes performance to benchmark results

Physical Process	Test	SPEC CINT2000	SPEC CINT2006	SPEC CFP2006	SPEC ALL_CPP
HiggsZZ4LM190	GEN+SIM	0.983	0.988	0.986	0.997
	DIGI	0.971	0.977	0.974	0.999
	RECO	0.979	0.985	0.983	0.998
	TOTAL(SUM)	0.982	0.988	0.986	0.997
MinBias	GEN+SIM	0.982	0.988	0.986	0.997
	DIGI	0.972	0.978	0.973	0.998
	RECO	0.970	0.976	0.970	0.997
	TOTAL(SUM)	0.981	0.987	0.984	0.997
QCD_80_120	GEN+SIM	0.980	0.986	0.984	0.998
	DIGI	0.973	0.980	0.976	0.999
	RECO	0.975	0.981	0.977	0.998
	TOTAL(SUM)	0.980	0.986	0.983	0.998
SingleElectronE1000	GEN+SIM	0.983	0.989	0.988	0.996
	DIGI	0.970	0.976	0.974	0.999
	RECO	0.962	0.968	0.960	0.995
	TOTAL(SUM)	0.983	0.989	0.987	0.996
SingleMuMinusPt10	GEN+SIM	0.972	0.978	0.974	0.998
	DIGI	0.970	0.976	0.972	0.998
	RECO	0.956	0.963	0.955	0.994
	TOTAL(SUM)	0.966	0.972	0.967	0.997
SinglePiMinusE1000	GEN+SIM	0.985	0.991	0.992	0.993
	DIGI	0.968	0.974	0.969	0.998
	RECO	0.980	0.986	0.995	0.965
	TOTAL(SUM)	0.985	0.991	0.992	0.993
TTbar	GEN+SIM	0.982	0.987	0.985	0.997
	DIGI	0.974	0.980	0.975	0.998
	RECO	0.902	0.908	0.891	0.963
	TOTAL(SUM)	0.977	0.982	0.978	0.998
Total	Total	0.969	0.975	0.970	0.998

Tab. 9.b: Correlations of CMS experiment codes performance to benchmark results

Physical Process	Test	SPEC CINT2000	SPEC CINT2006	SPEC CFP2006	SPEC ALL_CPP
pp MinBias	GEN+SIM	0.974	0.981	0.980	0.999
	DIGI	0.949	0.959	0.979	0.992
	RECO	0.956	0.966	0.989	0.981
	TOTAL(SUM)	0.965	0.974	0.983	0.998
PbPb per2 8.6 - 11.2fm	GEN+SIM	0.976	0.983	0.982	0.999
	DIGI	0.754	0.752	0.682	0.733
	RECO	0.942	0.949	0.943	0.990
	TOTAL(SUM)	0.976	0.983	0.983	0.999

Tab. 9.c: Correlations of ALICE experiment codes performance to benchmark results

Tab. 9: Pearson product-moment correlation of benchmark results and experimental performance given by events/second. Data comes from machine lxbench01-lxbench07.

From the results we can find that for CMS and ALICE experiments, the ALL_CPP benchmark introduced by us gives very good agreement to real performance. For ATLAS experiments, though the agreement is not as good, ALL_CPP result still shows better correlation to real performance than any other benchmark method.

6. Multiple factors that may affect computing performance

Establishing a standard benchmark method enables us to further investigate how various factors would affect computer's performance on high energy physics computing, and how we can improve computer's performance. In the following sections I will discuss different operating systems and processors' effect on high energy physics computing performance.

6.1 Operating Systems

Operating system can affect computing performance in many ways: compliers may generate binaries with unequal efficiency; kernels may deal with memory access and process scheduling using different strategy, standard libraries may use improved algorithms, etc. We will focus on the comparison of Scientific Linux 4 (SL4) and Scientific Linux 5 (SL5), both of which are heavily used in the production environment of high energy physics. Currently, SL4 is the mainstream operating system in CERN, while SL5 is widely deployed in DESY.

The results of the comparison are shown in Table 10 and Figure 6. SL4.6 (kernel 2.6.9-67.0.20.ELsmp, gcc 3.4.6) and SL5.2 (2.6.18-92.1.6.el5, gcc 4.1.2) are used. We can find that for integer benchmarks, the performance gain of using SL5 instead of SL4 get up to 2.5%-4.5%. Given that INT benchmark results have relatively smaller fluctuations, it is clear that SL5 has a better integer performance than SL4. However, for FP benchmark results, we do not have such an unambiguous tendency. In most cases, the differences of results can be considered inside the range of fluctuations, thus one would expect a roughly equal floating-point performance of SL4 and SL5. For ALL_CPP benchmark results, again, the performance gain does not exceed the accidental error fluctuation. Since the ALL_CPP suite contains a large fraction of integer part, and SL5 has a better performance in integer computing than SL4, one may expect a performance gain of SL5 in ALL_CPP benchmark suite as well. But even if there is a difference, it's not very significant.

	machine	**SL4**	**SL5**	**SL5 on SL4 Gain**
SPEC CPU2006 INT	hpbl2	73.03	76.03	4.11%
	hpbl2	73.1	75.29	3.00%
	hpbl3/4	61.27	62.98	2.79%
	hpbl3/4	62.85	64.41	2.48%
SPEC CPU2006 FP	hpbl2	52.66	51.72	-1.79%
	hpbl2	52.98	51.23	-3.30%
	hpbl3/4	52.59	53.03	0.84%
	hpbl3/4	54.09	51.14	-5.45%
SPEC CPU2006 ALL_CPP	hpbl3/4	59.63	61.37	2.92%
	hpbl1/2	68.12	69.83	2.51%

Tab. 10: Benchmark results comparison for different operating systems. Machine hpbl1/2, hpbl3/4 have the same hardware configuration, respectively; but differ in operation system (SL4 vs. SL5). Also, both SL4 and SL5 were in turn installed in hpbl2, for comparison.

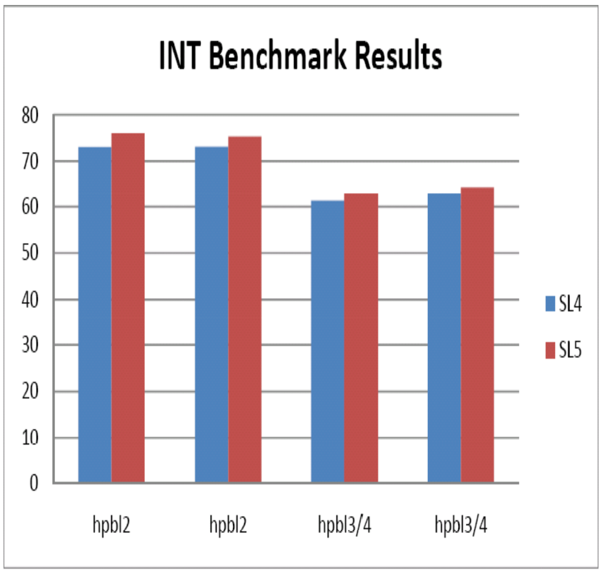


Fig. 6.a

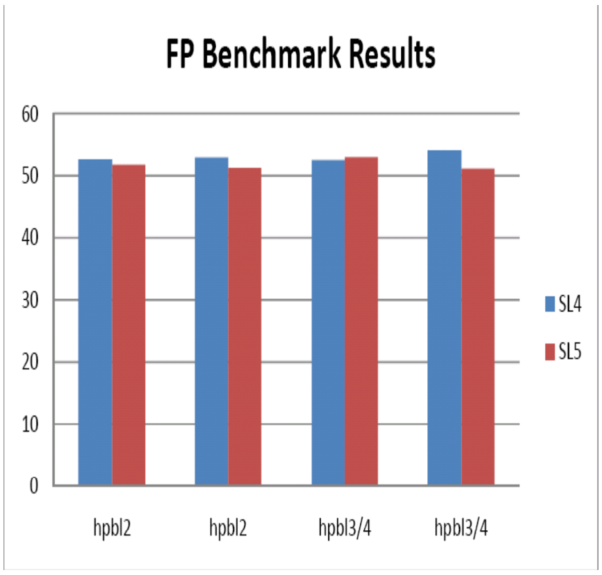


Fig. 6.b

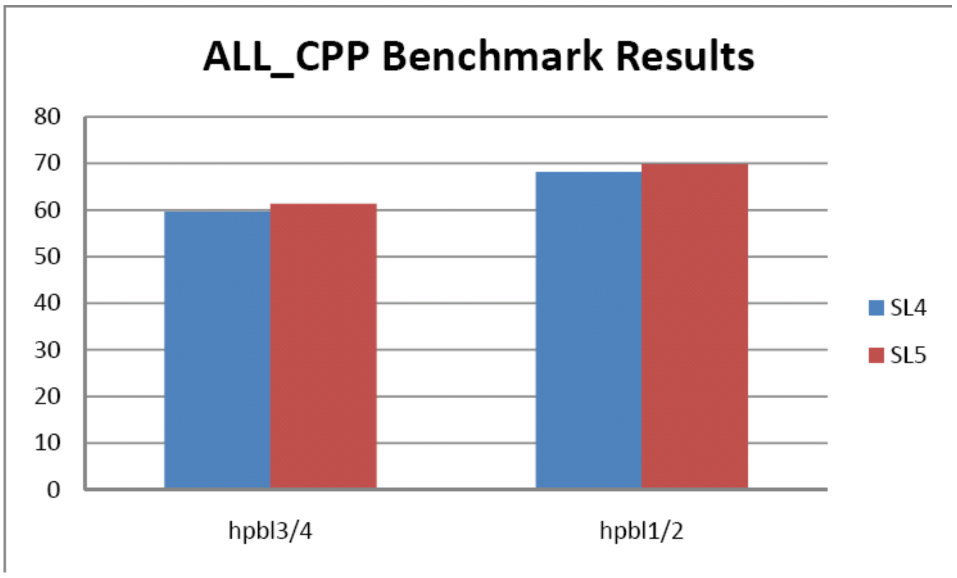


Fig. 6.c

Fig. 6 Benchmark results comparison for different operating system.

6.2 Processors

To study the effect of processors on high energy physics computing, two popular commodity processors, Intel Xeon E5440 and AMD Opteron 2356, which are believed to be representative to the modern Intel and AMD multi-core CPUs, are investigated.

Intel E5440 has a higher main frequency of 2.83 GHz, whereas AMD 2356 has a 2.30 GHz main frequency. AMD Opteron 2356 provides 128-bit floating point pipeline enhancement to improve floating-point performance, and HyperTransport technology to refine memory access strategy. Detailed technical information about these two processors can be found in Intel and AMD's official Website. The comparisons are shown in Table 11 and Figure 7.

	run	Intel E5440	AMD Opteron 2356	Difference
SPEC CPU2006 INT	1	72.92	61.27	19.0%
	2	73.59	62.85	17.1%
	3	76.03	62.98	20.7%
	4	75.29	64.41	16.9%
SPEC CPU2006 FP	1	51.5	52.59	-2.1%
	2	52.03	54.09	-3.8%
	3	51.72	53.03	-2.5%
	4	51.23	51.14	0.2%
SPEC CPU2006 ALL_CPP	1	68.38	60.67	12.7%
	2	69.83	61.37	13.8%

Table 7: Benchmark results comparison for different processors. The results are obtained from machine hpbl1/3 and hpbl2/4, which has the same software environment but different processors, respectively.

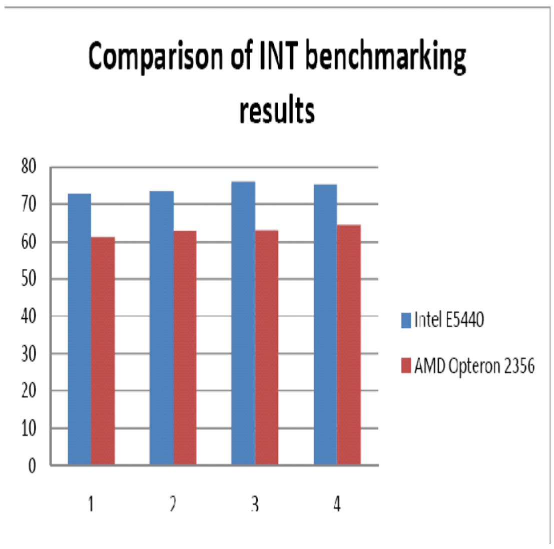


Fig. 7.a

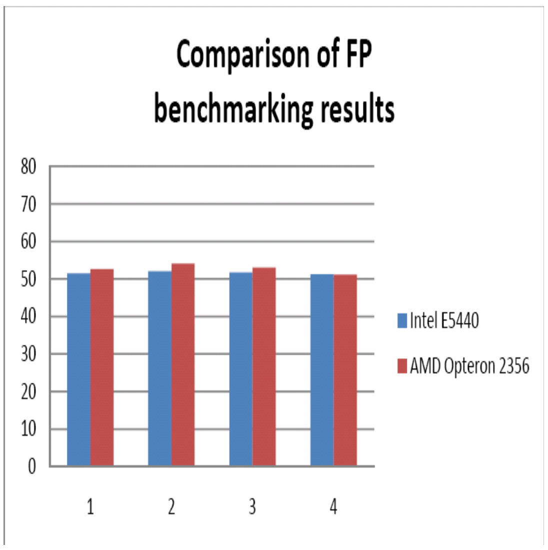


Fig. 7.b

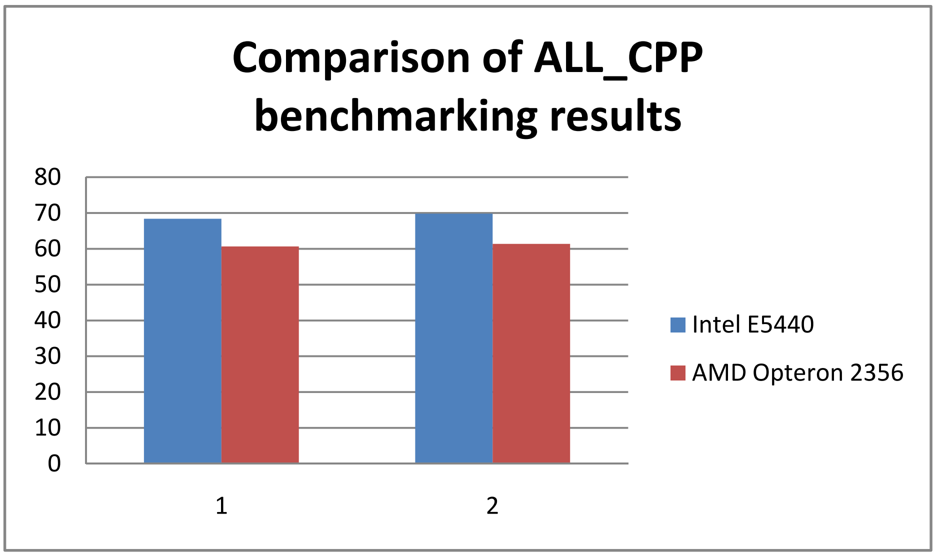


Fig. 7.c

Fig. 7: Benchmark results comparison for different processors

From the results we can find that for INT benchmark suite, Intel E5440 has a much higher score, which approximates 120% of the performance of AMD Opteron 2356. This roughly agrees with the fact that Intel E5440's main frequency exceeds AMD Opteron 2346's for 23%. As for the FP benchmark suite, despite its lower main frequency, AMD Opteron 2356 has an almost equal performance with Intel E5440, which is largely due to the floating-point pipeline enhancement. As a combination of integral and floating-point benchmarks, the ALL_CPP benchmark results fall in between of FP and INT results. Intel E5440 has a roughly 110% performance of AMD Opteron 2356.

We are now expecting the next generation of Intel processor, which is believed to be able to combine the advantage of current Intel and AMD architectures. However, it is not available yet.

7. Conclusions

In this paper we established a standard procedure to benchmark high energy physics worker nodes, and checked its agreement with experiment application performance. This benchmark method is proved to be simple and accurate. We then used this standard method to investigate the effect to performance of different operating systems and processors, and have got some interesting results.

References

[1] http://www.spec.org/cpu2006/
[2] https://www.scientificlinux.org/
[3]https://twiki.cern.ch/twiki/bin/view/FIOgroup/TsiBenchHEPSPEC
[4] https://edms.cern.ch/file/917289/1/A.Hirstius_presentation.pdf
[5] Manfred Alef, private communication
[6] John Russell Mashey, ACM SIGARCH Computer Architecture News, Vol. 32, pp. 1-14
[7] http://www.spec.org/cpu2006/Docs/changes-in-v1.1.html