Representative Benchmarks For Commercial Workloads

J. Fernández and J. M. García[1]

***Abstract* — The conventional methodology for system performance evaluation, which relies primarily on benchmarks and throughput metrics, has major limitations when analyzing the behavior and performance of computer systems under commercial workloads. The differences between workloads represented by benchmarks and commercial workloads are the main reason. We find that benchmarks have neither meaningful input/output activity nor enough interaction with the operating system, overlook concurrent users and underestimate scalability. This portrait will help us to understand the limitations of current benchmarks. Also, it could provide a possible guide for the selection of benchmarks modelling commercial workloads.**

***Keywords* — benchmarking, commercial workload, performance evaluation.**

I. Introduction

BENCHMARKS are used in computer systems research to analyze design alternatives, identify performance problems, and motivate improvements in system design. Equally important, users and manufacturers employ benchmarks to evaluate and compare computer systems. As an illustration, Linpack[8] or SPEC95[43] are commonly used to make absolute comparisons, that is, they are assumed to characterize the overall system performance. However, these benchmarks represent scientific workloads which are very different from commercial workloads [18].

Commercial workloads differ from scientific workloads in their datatypes, context switch rates, portion of the execution time spent on OS calls, and amount of loops, branch instructions and input/output operations. We have excluded desktop applications whose properties are a bit different from commercial workloads.

This paper makes two contributions. First, it provides a frame of reference for classifying most popular benchmarks and describes their essential properties. Second, it identifies the key features of commercial workloads that are not covered by those benchmarks.

J. Fernandez and J.M.Garcia are with the Department of Ingeniería y Tecnología de Computadores. University of Murcia (Spain).
email:{peinador,jmgarcia}@ditec.um.es
This work has been partially supported by the Spanish CICYT under grant TIC97-0897-C04-03.

Our analysis shows that most benchmarks has major limitations when analyzing the behavior and performance of computer systems under commercial workloads. They have neither meaningful input/output activity nor enough interaction with the operating system, overlook concurrent users and underestimate scalability.

The rest of the paper is organized as follows. In the next section the various stages of the performance evaluation process using benchmarks are delineated. Also, performance metrics, benchmark desiderable properties, a benchmark taxonomy and some related issues are explained. Section III provides an overview of the most popular benchmarks. Section IV identifies the essential features of commercial workloads. Finally, we conclude discussing the representativeness of benchmarks to evaluate systems destined to execute them.

II. Outline of the Performance Evaluation Using Benchmarks

A. Evaluation process

The proper evaluation of a system is difficult because of three reasons. First, it is seldom done systematically. Second, the goals are habitually unclear and ambiguous. Third, the process is so vague that their results are not reproducible. To avoid these problems, we need make the entire evaluation process explicit, systematic, and reproducible.

The evaluation process can be divided into the definition of the problem and the evaluation method [4]. Defining the evaluation problem implies precisely specifying four aspects of the problem: the system or subsystem under consideration as well as the variable parameters; the metrics to evaluate the system; the workload the system has to cope with; and the solution expected by the application of the evaluation method. The evaluation method could be by analytical modeling, simulation or benchmarking but, in this case, we are interested in the use of benchmarks.

B. Single figures of merit

The fundamental measurement made in any benchmark is the time to complete some specified task. The execution time will be a function of the problem size and the number of processors. All other performance

figures are derive from this basic metric. Each metric has its own uses, and gives different information about the computer and the algorithm used in the benchmark. It is important therefore to understand their properties, and the differences among them. The rest of this section looks over the most popular metrics.

The metric called **MIPS** stands for millions of instructions per second. This figure of merit, used in earlier benchmarks, is not suited for comparisons because of some disadvantages. MIPS loses nearly all its significance for comparisons across different instruction-set architectures. This became more obvious when RISC architectures appeared. Many operations performed by one CISC instruction may require several RISC instructions. Otherwise, MIPS varies for different programs executed on the same computer because depends on the instruction mix of each program. Thus, it is conditioned by the programming language and the compiler. Finally, MIPS can inversely change with performance whereas it can seem strange. To soften these inconveniences, MIPS have been redefined generating a bundle of versions: native MIPS, peak MIPS and MIPS VAX, related to VAX 11/780 [30]. In short, MIPS is a skew metric unless you know the computer CPI. Anyway, it is not an adequate metric for users.

Another popular metric, commonly reported as a benchmark result, is **MFLOPS** (millions of floating point operations per second). Clearly, MFLOPS depends on the hardware and the code. Since MFLOPS tries to measure the floating-point performance, it is not applicable outside of that environment. In any case, MFLOPS is not a reliable measure because the floating-point instruction set do not fit well among different architectures. MFLOPS is very sensitive to the mix of slow and fast operations. To solve this problem McMahon [20] proposed the utilization of normalized MFLOPS: one flop for sums, subtractions and products; four flops for divisions and square roots; and eight flops for sin, cosin and exponential. Just like MIPS, we distinguish native MFLOPS, peak MFLOPS and normalized MFLOPS [17]. Giladi [11] examines the correlation between the MIPS and the MFLOPS measures, according to Linpack, for various configurations.

Speedup is a measure of how much faster an application runs on a parallel computer. It is calculated as the ratio of serial execution time to parallel execution time. There are diverse definitions of serial and parallel execution times that result in at least three different definitions of speedup. In any case, the speedup varies with both the problem size and the number of processors.

Absolute speedup: compares parallel execution time with the fastest serial algorithm run on the fastest serial computer. In addition to the difficulty to have access the fastest serial algorithm, the faster serial computer improves continuously.

Real speedup: compares the parallel execution time with the time needed by the fastest serial algorithm for the application executing on a single node of the parallel computer. Determining real speedup can be a problem on a distributed-memory parallel computer because of the shortness of memory to execute the serial algorithm for large instances of the problem. Since all the proofs are executed on the same system, we eliminate the variation due to the changes in the fastest serial computer. By contrast, real speedup is still affected by changes in the best serial algorithm. Furthermore, using real speedup as a performance measure independent of execution time favours slow processors.

Relative speedup: uses the execution time of the parallel program when run on a single node of the parallel computer for the serial time. This variant of speedup suffers the same problems that real speedup: memory limitations and preference of slower processors. In addition, relative speedup favors code that is inefficient when run on a single processor.

According to all previous definitions, speedup cannot exceed the number or processors unless the serial version incurs some overhead. For example, if the problem size is bigger than the main memory, the secondary storage is employed. Then, there are more accesses for data residing in secondary storage what generates an additional overhead for the serial version. In this way, the speedup can be greather than the number of processors in some situations.

Efficiency is the ratio of speedup to the number of processors. This metric is an indication of the actual degree of speedup performance achieved as compared with the maximum value. Clearly, the lowest efficiency corresponds to the case of the entire program code being executed sequentially. Meanwhile, the maximum efficiency is achieved when all the processors are fully utilized throughout the execution period. For the same reasons as speedup, efficiency should not be use as an independent metric of execution time.

Finally, there are another metrics, common in the literature, derived from speedup -such as scaled speedup, sizeup, generalized speedup, scalability, issoefficiency and isospeed- that are analized by Sahni [25] and Sun [28].

C. Benchmark desiderable properties

Not all the applications are suitable for being used as benchmarks. This section depicts those attributes that are expected to have a program considered as a true benchmark.

- Comprehensibility/Intelligibility. A benchmark has to define explicitly the aspects of the computer that exercises. In addition, the results provided by the benchmark should be unambiguous, concise and easy to understand.
- Easy to use. The simplicity when executing benchmarks is a necessary requirement to accept them as a common frame for comparisons.
- Scalability. A scalable benchmark can be used for evaluating a broad range of machines with very different levels of computation power.

- Portability. Assuming we are interested in comparing heterogeneous systems, the code should be easy to adapt for every machine. The main features determining the portability of a benchmark are the programming language, the code length and the distribution of every type of operation. The selected programming language assumes the existence of a compiler for each target machine. A short benchmark simplifies the adjustment when passing from a system to another one. Several operations, such as input/output operations, are difficult to adapt for different platforms. Thus, they constitute a drawback for code portability.
- Representativeness. A benchmark is said to be representative of a user's workload if the benchmark accurately predicts performance of that workload on a range of configurations. When looking for a predictive benchmark, search for one with a workload as similar as possible to the selected one, and see how well it correlates with the relative performance of the aim workload.
- Accessibility. The availability of a benchmark is essential for accepting them as a standard. However, if code is also attainable, users could compare some results of programs which are very different among them because of possible modifications. Therefore, the benchmark user should know the version of the program being used, and the changes suffered related to the original version.
- Reproducibility. The guiding principle for reporting performance measurements should be reproducibility, that is, the ability to replicate the results. For that reason, we need an exhaustive description of both software and hardware configuration. Reproducibility is a lost feature in benchmarking. Thus, we can find many results in books, reviews and papers lacking of accurate information for knowing under what conditions the experiment was carried out.

Finally, we must note that all the mentioned properties are recommendable for benchmarks, but can be in a jam among them. For example, a portable benchmark is short and has few input/output operations. So, that benchmark won't be representative of real applications with substantial code lengths and profusion of input/output operations.

D. Benchmark taxonomy

This section is intended to provide a classification of the most common benchmarks.

1. Toy benchmarks typically has a few hundreds of code lines. Their output is known in advance. They are little meaningful. Programs like Heapsort, Hanoi, Queens, Shuffle and Sieve of Eratosthenes and many other similar to these, are examples of this item [31].
2. Microbenchmarks are programs designed to understand the basic performance characteristics of the primitive operations provided by hardware, hardware/software interface (OS and similar), communication interface or programming model. We can distinguish six types of microbenchmarks:
   - Processing microbenchmarks measure the performance of internal operations, e.g., *BOBMark* calculates the relative speed of a processor's FPU [32].
   - Local memory microbenchmarks measure latencies and bandwidths of the various levels of the memory hierarchy and determine their organization, e.g., *STREAM* is a popular program, created and maintained by McCalpin [19], that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. Saavedra entirely determine cache and TLB parameters using a memory hierarchy microbenchmark [24].
   - Input-output microbenchmarks measure the latencies and bandwidths for I/O operations, commonly disk reads and writes, for various strides and lengths, e.g., STKIO [34] generates either random or sequential I/O reference patterns over a number of drives or files to measure I/Os per second and sustained transfer rate (in MB/s).
   - Communication microbenchmarks measure the latencies and bandwidths of data communication operations, for various message sizes, e.g., *netperf*, developed by Rick Jones [35], can be used to measure the performance of many different types of networking. It provides tests for both unidirectional throughput, and end-to-end latency.
   - Synchronization microbenchmarks measure the performance of different synchronization operations.
   - OS microbenchmarks measure the overhead of some basic OS tasks such as context switching, system calls or process creation.

   Recently, it has become popular to put together collections of microbenchmarks, e.g., *lmbench* is a microbenchmark suite designed to identify and evaluate system performance bottlenecks in a global manner [21]. *lmbench* measures memory bandwidth, IPC bandwidth, cache I/O bandwidth, memory latency, signal handling cost, process creation cost, context switching and IPC latency.
3. Kernels are code fragments, extracted from real applications, that constitute significant portions of their execution times. Kernels are suitable to isolate the performance of specific features of a machine in order to explain the reasons for differences in performance of real programs. Most popular kernels, Livermore Loops [20], Nas Parallel Benchmarks [3] and Linpack [8] are explained in a later section.
4. Synthetic benchmarks exist solely to evaluate performance. They try to factorize most common

operations and operand types of real applications. Unlike kernels, synthetic benchmarks don't belong to any real application. Whetstone and Dhrystone are the most popular synthetic benchmarks exercising floating-point and integer arithmetic respectively.

5. Application benchmarks are programs commonly employed by users for realizing their jobs. They exhibit complex interactions that are present neither kernels nor synthetic benchmarks.
6. Benchmark Suites. Last years, it has become popular to measure the performance of systems with a variety of applications. These collections include benchmarks of different types but real applications predominate. Clearly, a benchmark suite is as good as their individual components. Regardless, the advantage is that the weakness of any benchmark is softened by other benchmarks, whenever the method for summarizing the performance reflects the execution time of the entire suit. Advantages and disadvantages of methods for summarizing performance are discussed in [10] and [16]. Initiatives like SPEC [7] or SPLASH [26] have achieved a great relevance.

Other classifications can be found at [1] and [16].

E. Non-CPU influences in performance

Sometimes, benchmark ratios refer only to the hardware speed. With microprocessors, this is even more reduced to the CPU speed. However, the preceding discussion makes it clear that other factors also have a great influence in overall performance.

- Memory hierarchy. As DRAM becomes denser and cheaper, machines are equiped with more memory, and so, more data are permanently keep in memory. Futhermore, memory subsystems are typically organized as cache hierarchies. Otherwise, SMPs (Symmetric Multiprocessors), where the main bottleneck is the memory system accesed by all the processors, have emerged as a serious option for commercial servers [18]. All these facts have elevated memory system as a critical performance factor.
- I/O subsystem. Commercial applications such as on-line transaction processing (OLTP) generate a significant amount of I/O activity. Moreover, for these storage intensive workloads, RAID systems have become the industry standard for providing efficient, fault-tolerant mass storage.
- Operating system (OS). Portability is favoured by the absence of OS calls in benchmarks code. However, Web servers spend on OS calls the most of their execution time (about 70-85% [22]).
- Programming Language and Compiler. The intrinsic properties of a programming language notably influence in the execution time of applications even if they are similar and generate the same results. Obviously, the maximum code optimization level attainable has an effect on the execution time.
- Libraries. Certain applications, such as numeric programs, waste most of their execution time in functions of some specialized library. The efficiency and accuracy of these libraries affects meaningfully their behavior.

F. Fallacies, pitfalls and other mistakes

In this section we show an overview of common ways to distort results when evaluating performance in order to make a computer system seems to be more powerful than it is. These points constitute a guide for artificially boosting performance.

1. Quotting only 32-bit performance results, not 64-bit results.
2. Omitting some critic details about software or hardware configuration.
3. Substracting the setup time, wasted by the initialization routines, from the overall execution time.
4. Including assembly code and other low-level language constructs into the benchmark code .
5. Scaling up the problem size with the number of processors, while analyzing performance rates versus number or processors, to increase efficiency.
6. Projecting performance results from a partial-scale system to a full-scale system.
7. Comparing your results against old unoptimized code on an obsolete system.
8. When giving MFLOPS, basing the flop operation count on an inflated parallel implementation instead of the best sequential implementation.
9. Characterizing the performance in terms of some unclear figure of merit that increases the performance in detriment of execution time.
10. Modifying the algorithm used in the parallel implementation to increase some figure of merit as MFLOPS, no matter if it requires more time to complete the execution.
11. Measuring parallel run times on a dedicated system, but measuring conventional run times in a busy environment. In short, using the best time for the parallel system and the worst time for the conventional system.
12. Assuming that kernels or synthetic benchmarks can predict the performance of real applications.

Bailey [2] explains some of these items in an humorous manner.

III. POPULAR BENCHMARKS

A. Classical benchmarks

1) Whetstone

Whetstone, developed by H.J. Curnow and B.A. Wichman as the first synthetic benchmark, was published in 1976 in Algol 60. It consists of several modules. Each

module loops around statements of some particular type and a final statement to print the results. The number of iterations of every loop assigns a weight to it in such way that program executes a multiple of one million of Whetstone instructions. Benchmark results are given as KWIPS (kilo Whetstone instructions per second) or MWIPS- (mega Whetstone instructions per second). Whetstone has a high percentage of floating-point operations and data (numeric programming), spends a lot of time in mathematical library functions, lacks of local variables and exhibits a very high code locality.

2) Dhrystone

Just like Whetstone, Dhrystone is a synthetic benchmark published by R.P. Weicker in 1984 in Ada. It represents nonnumeric programming. Whetstone is composed of twelve procedures included in one measurement loop. Benchmark results are given as Dhrystones per second, where a Dhrystone is an iteration of the measurement loop. Dhrystone doesn't contain floating-point operations and has more procedure calls, more *if* statements and fewer loops. Local variables and parameters are used more often than global variables.

3) Livermore Fortran Kernels

Livermore Fortran Kernels, also called Livermore Loops, are a set of 24 Fortran DO-loops extracted from operational codes used at the Lawrence Livermore National Laboratory. The author, McMahon [20], collected them adding statements for time measuring. They are a mixture of vectorizable and non-vectorizable loops and test rather fully the computational capabilities of the hardware, and the skill of the software in compiling efficient code, and in vectorization. The main value of the benchmark is the range of performance that it demonstrates, and in this respect it complements the limited range of loops tested in the LINPACK benchmark. The benchmark computes normalized MFLOPS for each kernel, together with various averages (arithmetic, geometric and harmonic mean) and the quartiles of the distribution.

4) PERFECT Benchmarks

The Perfect (Perfect Evaluation and Cost effective Transformations) Benchmark suite consists of thirteen application programs to obtain basic metrics about supercomputer performance [12]. Their components are programs from four application areas: fluid dynamics, chemistry and physics modeling, engineering design and signal processing. Program codes are written in Fortran77. Each program reports the CPU time, the response time and MFLOPS. Two modes of execution are allowed for the programs in the set. The first mode allows no alteration of the codes beyond those that were necessary to resolve portability problems. The second mode permits executing an optimised version reporting all modifications performed. All programs are complete applications that have a few input/output operations and perform useful calculations. The main problem of Perfect Benchmarks is the difficulty to interpret results as a matter of the complexity of applications. Data sets and size of problems have been questioned too. Perfect Benchmarks are no longer maintained.

5) Euroben

The European benchmark group was created in 1990 to analyze the performance of high-performance scientific computers. Euroben is a benchmark for the evaluation of single user performance of computers that was proposed by the European Benchmark Group [27]. This benchmark is intended to give information about the behavior of the CPU (or CPUs in a parallel environment), the influence of CPU-memory traffic and the I/O-throughput. Euroben follows a graded approach, that is, it has a modular structure making possible to extract information from simpler modules that can be used to explain more complicated behavior in the programs of the more extensive modules. The complete benchmark set contains approximately 40 programs, ranging from very small and simple ones to large application codes. For this purpose, four modules were defined. Module 1 contains programs that measure basic characteristics of the machines like speed of basic floating-point operations, memory bank conflicts, and speed and accuracy of important intrinsic functions. Module 2 represents important numerical algorithms like the solution of full and sparse linear systems of FFTs. Module 3 contains programs using more than one basic algoritm as given in Module 2. Module 4 consists of full applications of circuit simulation, computational chemistry, computational fluid dynamics, lattice-gauge simulation, seismic imaging, structural mechanics, reservoir modelling and weather simulation. The later module is similar to the PERFECT benchmark. All modules are written in Fortran77 and the precision required is at least 64 bits. Euroben Release 3.2 is available at [37].

6) GENESIS Benchmarks

GENESIS benchmarks were developed to evaluate the performance of the SUPRENUM distributed-memory computer on scientific applications [1]. The programs use the PARMACS macros for message-passing and task creation to facilitate code portability, and are written in standard Fortran77. Each program is provided both a sequential and a distributed-memory version. It is permitted to optimize the benchmark for showing the full potential of the hardware but the results with unmodified programs must be reported. The programs are classified into three levels of complexity: five synthetic benchmarks, nine kernels and six applications. The fundamental measurement made in any benchmark is the elapsed time to complete some specified task. GENESIS Release 3.0 is available at [38]. It uses PARMACS 5.1 message-passing macros for PVM 3.x.

7) PARKBENCH

PARKBENCH (PARallel Kernels and BENCHmarks) is a suite of microbenchmarks, kernels, applications and compiler benchmarks for parallel machines with explicit message-passing programs written in Fortran77 or HPF [23]. The former are divided in uniprocessor and multiprocessor

microbenchmarks and their results are given in terms of the performance parameters defined by Hockney [17]. Uniprocessor benchmarks measure timer resolution and accuracy (TICK1,TICK2), performance of basic arithmetic operations (RINF1), and memory latency and bandwidth (POLY1,POLY2). Following the methodology of Euroben, its purpose is to obtain basic architectural parameters that help us to understand the behavior of kernels and applications. Kernel benchmarks are divided into matrix kernels, Fourier transforms, partial different equation kernels (PDE Kernels) and others. Aplications are intended in the areas of climate and meteorological modeling, computational fluid dynamics (CFD), finance, moledular dynamics, plasma physics, quantum chemistry, quantum chromodynamics (QCD) and reservoir modeling. Finally, the compiler benchmarks are intended for people developing High Performance Fortran compilers to test their compiler optimizations. PARKBENCH Release 2.2.1 is available at [39]. Performance Database Server stores results from PARKBENCH and other benchmarks [40].

8) NAS Parallel Benchmarks

NAS Parallel Benchmarks (NPB) were developed at NASA Ames Research Center [3] as a set of eight programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications. Each program can be executed with different sizes (Class A for “small” size, Class B for “large” size and Class C for “very large” size) and reports the execution time. For each program, the execution time, MFLOPS and the normalized execution time, related to a Cray Y-MP (Class A) or a Cray C90 (Class B), are reported. The first version of the benchmarks, NPB 1, was defined in a "pencil and paper" way, that is, no implementation for solving the problem was prescribed; only precise descriptions of the problems were given. Vendors and others implement the detailed specifications in the NPB 1 report, using algorithms and programming models appropriate to their different machines. The results are verified by NAS and published in a periodic NAS report. NPB 1 implementations are generally proprietary. Although the idea behind this approach is attractive from the viewpoint of the originators of the benchmark, it may be difficult to discriminate between the effects of a particularly good benchmark implementation and the capabilities of the machine it is executed on. NPB 2 are MPI-based source-code implementations written and distributed by NAS. They are intended to be run with little or no tuning, and approximate the performance a typical user can expect to obtain for a portable parallel program. They complement, rather than replace NPB 1. NPB 2-serial are single processor (serial) source-code implementations derived from the NPB 2 by removing all parallelism. They are intended to be starting points for shared memory, as tests for parallelization tools, and as benchmarks for workstations and PC's (Class W). NPB Release 2.3 is available at [41].

B. Current benchmarks

1) The most reported benchmark: LINPACK

This well-known benchmark is a Fortran program (also exists a C version) for the solution of (100x100, 300x300 or 1000x1000) dense set of linear equations by Gaussian elimination. It is distributed by Jack Dongarra at the University of Tennessee [8]. The results are quoted in MFLOPS (single or double precision) and are regularly published. Most of the compute time is wasted in vectorisable DO-loops such as the DAXPY (scalar times vector plus vector) and inner products. Therefore one expects vector computers to perform well on this benchmark. The weakness of the benchmark is that it tests only a small number of vector operations, but it does include the effect of memory access and it is solving a complete (although small) real problem. Futhermore, Dongarra in [8] advises that LINPACK doesn’t pretend to be more than it is: it measures the performance of systems with respect to the solution of dense linear systems and no more. In any case, the main value of this benchmark is that the results are known for the most speed computers in the world (TOP500 Supercomputing sites [42]).

2) SPEC

SPEC (Standard Performance Evaluation Corporation) consortium was created in 1990 by a number of large workstation vendors for creating, maintaining and distributing a standardized set of relevant benchmarks and metrics for performance evaluation of modern computer systems [7]. SPEC has developed several suites for different purposes, but we are mainly interested in SPEC CPU and SPEC WEB. SPEC consortium also promotes other initiatives: SPECjvm98 for comparing Java virtual machine (JVM) client platforms; SPECsfs97 for testing NFS; SPEC GPC (Graphics Performance Characterization Group); and SPEChpc96 for high-performance computing.

SPEC CPU is a suite of benchmarks for testing CPU, memory hierarchy and compiler. The first version, SPEC CPU 92, has five integer and five floating-point applications written in C and Fortran respectively. Their results are reduced to one figure called SPECmark, that is meant to be equivalent of the factor by which the machine under consideration is faster than a DEC VAX 11/750. SPECmark is calculated as the geometric mean of the ten SPEC ratios. Each SPEC ratio is the quotient derived from dividing the SPEC reference time by a particular machine’s run time. This first version of SPEC was criticised by some weaknesses: the suite is biased in favor of double precision floating-point processing, it has minimal input/output, and most benchmarks stress neither instruction nor data cache and run too sort. The lastest version, SPEC CPU 95, appeared to alleviate these faults. SPEC95 includes eight integer and ten floating-point programs. Their input data sizes have been

augmented related to SPEC92, and their resulting metrics have been specialized. SPEC 95 distinguishes between integer and floating-point results and each of them has an aggressive (SPECint95, SPECfp95) and a conservative mode (SPECint_base95, SPECfp_base95). Each combination may be used for reporting speed or throughput (SPECint_rate95, SPECint_base_rate95, SPECfp_rate95, SPECfp_rate_base95), carrying out a number of tasks. SPEC95 calculates the ratios related to a SPARCStation 10/40 (40MHz SuperSPARC with no L2 cache). Unfortunately, these units have no clear relation to hardware and software parameters which make the interpretation difficult. Futhermore, there are some serious conceptual problems with the SPEC metrics. First, there is no clear judgment to choice the programs included in the suites CINT95 and CFP95. Second, the difference between aggressive and conservative execution depends on a fixed number of compiler options selected by the benchmark user. Third, the metrics are generated using the geometric mean of individual SPECratios.

Since SPEC95 is focused to workstations, the developers of SPEC CPU teamed up with the developers of Perfect Benchmarks to form, in 1994, the SPEC High-Performance Group (HPG) that published the SPEChpc96. SPEChpc96 benchmarks represent large industrial applications. They can measure sequential, parallel/shared memory, and parallel/message-passing architectures, can evaluate workstations as well as high-performance computer systems and come in several data set sizes [9].

SPECweb96 is a standarized performance benchmark for evaluating the performance of Web-server software. A SPECweb96 test consists of a server machine that runs the Web-server software to be tested and a number of client machines. The client machines use the SPECweb96 software to generate a workload that stresses the server software. The workload is gradually increased until the server software is saturated and the response time degrades significantly. The point at which the server is saturated is the maximum number of HTTP operations per second (the metric reported) that the Web-server software can sustain. SPECweb96 has some limitations. SPECweb96 workload was determined by looking at log files for major sites but may not be representative of all the possibilities. In fact, it uses only HTTP GET commands, but it doesn't use POST commands or CGI calls. Also, SPECweb96 supports only the HTTP 1.0 protocol. It does not support any of the HTTP 1.1 mechanisms like keepalives and persistent connections. A newer version, called SPECweb99, is under development.

Both SPEC95 and SPECweb96 are commercial distributions, including source code, for UNIX and Windows NT platforms.

3) SPLASH

The SPLASH (Stanford ParalleL Applications for SHared memory) was developed at Stanford University to facilitate the evaluation of architectures that support a cache-coherent address space [26]. It was replaced by the SPLASH-2 suite, which enhanced some applications and added others. The SPLASH-2 suite currently contains seven complete applications and five computational kernels. The programs represent various computational domains, mostly scientific and engineering applications and computer graphics. They are written in C, and use the PARMACS macros for parallelism constructs. For every application, a complete description is given. This includes the problem being solved, the principal data structures used, profile information, the structure of paralellism in the program, and some of their static and dynamics characteristics. Parallelism in some of the applications is limited by the nature and size of the input data sets. Also, the applications were written for small to medium-scale machines in mind, and may require restructuring to be run efficiently on larger multiprocessors. By these reasons, the authors believe that it is inappropriate to use SPLASH for absolute comparisons between two systems. However, the full applications of the suite are real, written in an architecture-independent way and well-documented to facilitate the comparability of results. SPLASH-2 is available at [44].

4) TPC

The Transaction Performance Council (TPC), founded in 1988, has developed several benchmarks, TPC-A, TPC-B, TPC-C and TPC-D, representing different kinds of transaction-processing and database system workloads. TPC benchmarks are specified in terms of high-level functional requirements rather than specifying any given hardware or software platform or code-level requirements. Benchmarks scale, increasing the number of users and the database size, to prevent the workload from being overwhelmed by the improvements of OLTP systems. TPC uses two metrics for reporting results: throughput of transactions per second (tps), where response time must be smaller than a threshold over 90% of the transactions, and cost/performance related to the transactions per second. Total system cost includes all hardware and software used to successfully run the benchmark, including five years of maintenance costs.

The first TPC benchmark was TPC-A that consists of a simple update-intensive transaction. Its purpose was to exercise the main features of an on-line transaction processing (OLTP) system. It modeled a bank transaction reading from a terminal, updating an account, writing an history record, and finally writing to a terminal.

TPC-B was designed to exercise the system components necessary for update-intensive database transactions, that is, TPC-B used the same TPC-A transaction type but eliminating the network and user interaction components of the TPC-A workload. The two preceding benchmarks were replaced by TPC-C and TPC-D.

TPC-C, approved in 1992, can be considered the successor of TPC-A as an OLTP benchmark. It was designed to be more realistic than TPC-A but

maintaining most of its characteristics. TPC-C is a multiuser benchmark that requires a remote terminal emulator to simulate a population of terminals. It models the activities of a wholesale supplier with distributed sales districts and supply warehouses, including customers placing orders, making payments, or making inquiries, as well as deliveries and inventory checks. TPC-C requires guaranteeing the atomicity and the integrity of multiple types of transactions of varying complexity. Database size scales with the throughput of the system. TPC-C has a more complex database structure compared to TPC-A, multiple transaction types of varying complexity, on-line and deferred execution modes, higher levels of contention on data access and update, patterns that simulate hot spots, access by primary as well as nonprimary keys, realistic requirements for full-screen terminal I/O and formatting, requirements for full transparency of data partitioning, and transaction rollbacks.

TPC-D is a benchmark for decision support systems (DSS). It models systems able to formulate of business decisions through complex queries against a database. The queries access large portions of the database unlike OLTP, and include complex operations. Also, they consume a lot of time, rarely modify the database, and have only a few concurrent users.

Projected TPC benchmarks are TPC-H, TPC-R and TPC-W. TPC-W benchmark is designed to represent any business that must market and sell a product or service over the Internet. For a more detailed description see [13] or the website of the Transaction Performance Council [45].

C. Other approaches

All the benchmarks we have mentioned up to now, follow the fixed-size speedup model [28], that is, as more computation power is available, the problem can be solved in less time. In this section we expose SLALOM that adapts to the fixed-time speedup model, and its consequence HINT which lies over a philosophy slightly different.

1) The precedent of SLALOM

SLALOM was developed in the Scalable Computing Lab at Ames Laboratory on the Iowa State University. It stands for Scalable, Language-independent, Ames Laboratory, One-minute Measurement. SLALOM, the first scalable supercomputer benchmark, was created to measure how much work a computer could do in one minute. The problem is to find the equilibrium radiation inside a box made of diffuse colored surfaces (*radiosity* method). The faces are divided into regions called *patches*, the equations that determine their coupling are set up, and are solved for red, green and blue spectral components. SLALOM also allows parallel processing to be used. Each processor can work with a subset of the scene, working asynchronously with the best available information about the rest of the scene. The SLALOM results are the number of patches calculated in one minute as well as the MFLOPS. The idea, as described in [14], is correcting some deficiencies of existing benchmarks; SLALOM is scalable and is neither language nor platform dependent. Its use as a benchmark has been superceded by the much simpler HINT. SLALOM source code can be found at [31].

2) HINT

The HINT (Hierarchical INTegration) benchmark was developed at Ames Laboratory to evaluate the performance of a wide spectrum of computers [15]. HINT fixes neither time nor problem size. It uses internal subdivision to look for rational bounds on the area in the xy plane for which x ranges from 0 to 1 and y ranges from 0 to $(1-x)/(1+x)$. It subdivides x and y ranges into an integer power of two equal subintervals and count the squares that are completely inside the area (lower bound) or completely outside the area (upper bound). HINT reports with a measurement called QUIPS (QUality Improvement Per Second) where quality is the reciprocal of the difference between the upper and lower bounds. Benchmark has unlimited scalability because quality has no mathematical upper limit, that is, the only limit is imposed by hardware (speed, memory and precision). Plotting QUIPS versus memory used, we can identify the different memory regimes of the machine. While HINT provides a graph of performance, it also has a single number measure called Net QUIPS. Net QUIPS is defined as the integral of the quality divided by the square of the time (the area under the graph). HINT can be run with any datatype (floating-point of any precision, integer of any precision and even BCD) and is available for several architectures (shared memory with pthreads, distributed-memory with MPI and several vector machines). HINT code can be accessed at [47].

IV. COMMERCIAL WORKLOADS

The behavior of commercial workloads is known to be very different from scientific workloads. This section is intended to recognize the main features of commercial workloads in order to identify what aspects of computer performance benchmarks would have to stress for being representative.

The main datatypes in commercial workloads are integers and strings, in comparison with floating-point operands in scientific workloads. This item seems not to constitute a drawback since most benchmarks take into account it.

In general, commercial workloads have high context switch rates because they have many concurrent users. Most benchmarks usually execute tasks in batch mode. Only TPC benchmarks consider the existence of various users at the same time.

Commercial workloads spend an important portion of their execution time on OS calls. OLTP applications and decision support applications spend on OS code about 20% of their execution time, while web servers spend between 70% and 85% [22]. However, most benchmarks have no OS calls.

Input/output operations have been historically considered as the primary performance bottleneck for commercial workloads. Innovations in disk subsystems and the increasing gap between processor and memory speed have brought the bottleneck near the memory hierarchy [5]. Anyway, commercial workloads have many input/output operations and most benchmarks have minimal input/output operations.

Also, commercial applications execute fewer loops and use more branch instructions than scientific applications what along with their data access patterns prevent them from use the memory system as effectively as tradicional architectures. Nevertheless, almost all the benchmarks described above are primarily based on scientific codes.

Agreeing more computation power is available, users confront systems with more complex problems. However, except for SLALOM, HINT and TPC benchmarks, extant benchmarks are based on the idea of measuring the time various computers take to complete a fixed-size task, that is, scalability is supplied by means of new versions having bigger data sets.

Characterization of specific commercial workloads such as on-line transaction processing (OLTP), decision support systems (DSS), web index searchs or web multimedia services, and their arquitectonic implications, are present-day topics under study. For a more detailed description see [5], [18], [22] and [29].

V. CONCLUSIONS

Most benchmarks are not appropriate to evaluate or compare the performance of computer systems destinated to execute commercial workloads. Their codes don't represent commercial workloads by the reasons above exposed. In addition, as more power is available, users dynamically increase the size of application programs. Meanwhile, benchmarks remain static and so benchmarks results overstate computer performance. Except for TPC and HINT, the scalability of benchmarks is another pendant issue.

In the other hand, computer researchers have long cope with the problem of systematically comparing different computers and algorithms. It is difficult or misleading to compare them using the ratio of execution times, above all when architectures, methods, precision or storage capacity are very different. Our view is that giving a graph characterizing performance, accordingly to time or memory, seems more plausible than providing a single number.

REFERENCES

[1] C.A. Addison, V.S. Getov, A.J.G. Hey, R.W. Hockney and I.C. Wolton. "The GENESIS distributed-memory benchmarks". Computer Benchmarks. J. Dongarra and W. Gentzsch eds., ed. North-Holland, pp. 257-271, 1993.

[2] D.H. Bailey. "Twelve ways to fool the masses when giving performance results". Supercomputing Review, pp. 54-55, August 1991.

[3] D.H. Bailey, E. Barszcz, L. Dagun and H.D. Simon. "NAS Parallel Benchmarks results". Computer Benchmarks. J. Dongarra and W. Gentzsch eds., ed. North-Holland, pp. 225-237, 1993.

[4] G. Böckle et al. "Structured Evaluation of Computer Systems". *IEEE Computer*, Vol. 29, No 6, pp.45-51, June 1996.

[5] L.A. Barroso, K. Gharachorloo and E. Bugnion. "Memory System Characterization of Commercial Workloads". Procc. of the 25th Int. Symposium on Computer Architecture, pp. 3-14, June 1998.

[6] D.E. Culler and J.P. Singh. Parallel Computer Architecture. A Hardware/Software Approach. Ed. Morgan Kauffman, 1999.

[7] K. Dixit. "The SPEC benchmarks". *Parallel Computing*, Vol. 17, pp. 1195-1209, 1991.

[8] J.J. Dongarra and H.A. van der Horst. "Performance of various computers using standard sparse linear equations solving techniques". Computer Benchmarks. J. Dongarra and W Gentzsch eds., ed. North-Holland, pp. 177-188, 1993.

[9] R. Eigenmann and S. Hassanzadeh. "Benchmarking with Real Industrial Applications: The SPEC High-Performance Group". *IEEE Computational Science & Engineering*, Vol. 3, No. 1, pp. 18-23, Spring 1996.

[10] R. Giladi and N. Ahituv. "SPEC as a Performance Evaluation Measure". *IEEE Computer*, Vol. 28, No. 8, pp. 33-42, August 1995.

[11] R. Giladi. "Evaluating the Mflops Measure". *IEEE Micro*, Vol. 16, No. 4, pp. 69-75, August 1996.

[12] C.M. Grassl. "Parallel Performance of applications on supercomputers". *Parallel Computing*, Vol. 17, pp. 1257-1273, 1991.

[13] J. Gray. The Benchmark Handbook for Database and Transaction Processing Systems, Second Edition. Ed. Morgan Kauffman Publishers, 1993.

[14] J.L. Gustafson, D. Rover, S. Elbert and M. Carter. "The Design of a Scalable, Fixed-Time Computer Benchmark". *Journal of Parallel and Distributed Computing* 12, pp. 388-401, 1991.

[15] J.L. Gustafson and Q.O. Snell. "HINT: A new Way To Measure Computer Performance". Proceedings of the 28th Annual Hawaii International Conference on Systems Sciences, IEEE Computer Society Press, Vol. 2, pp. 392-401, 1995.

[16] J.L. Hennessy and D.A. Patterson. Computer Architecture: A Quantitative Approach, Morgan Kaufmann, Second Edition, San Francisco, 1996.

[17] R. Hockney. "Performance parameters and benchmarking of supercomputers". Computer Benchmarks. J. Dongarra and W Gentzsch eds., ed. North-Holland, pp. 41-63, 1993.

[18] K. Keeton et al. "Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads". Procc. of the 25th Int. Symposium on Computer Architecture, pp. 15-26, June 1998.

[19] J.D. McCalpin. "Memory and Machine Balance in Current High Performance Computers". Technical Committe on Computer Architecture (TCCA) Newsletter, IEEE Computer Society, December 1995.

[20] F. McMahon. "The Livermore Fortran Kernels Test of the Numerical Performance Range". Performance Evaluation of Supercomputers. J.L. Martin ed., ed. North-Holland, pp. 143-186, 1988.

[21] L. McVoy and C. Staelin. "lmbench: Portable Tools for Performance Analysis". Procc. of the USENIX 1996 Annual Technical Conference, San Diego, January 1996.

[22] A.K. Nanda. "Multiprocessor Architecture Evaluation Using Commercial Applications". First Workshop on Computer Architecture Evaluation Using Commercial Workloads, in conjunction with HPCA-4, February 1998.

[23] PARKBENCH Committee. Public International Benchmarks for Parallel Computers, February 1994.

[24] R.H. Saavedra. "Measuring Cache and TLB Performance and Their Effect on Benchmark Run Times". *IEEE Transactions on Computers*, Vol. 44, No. 10, pp. 1223-1235, October 1995.

[25] S. Sahni and V. Thanvantri. "Performance Metrics: Keeping the Focus on Runtime". *IEEE Parallel & Distributed Technology*, Vol. 4, No. 1, pp. 43-56, Spring 1996.

[26] J. P. Singh, W. Weber and A. Gupta. “SPLASH: Stanford Parallel Applications for Shared-Memory”. *Computer Architecture News*, Vol. 20, No. 1, pp. 5-44, 1992.
[27] A. J. van der Steen. “The benchmark of the Euroben group”. *Parallel Computing*, Vol. 17, pp. 1211-1221, 1991.
[28] X. Sun and J.L. Gustafson. “Toward a better paralell performance metric”. *Parallel Computing*, Vol. 17, pp. 1093-1109, 1991.
[29] P. Trancoso and J. Torrellas. “Exploiting Caches Under Database Workloads”. First Workshop on Computer Architecture Evaluation Using Commercial Workloads, in conjunction with HPCA-4, February 1998.
[30] R.P. Weicker. “An Overview of Common Benchmarks”. *IEEE Computer*, Vol. 23, No. 12, pp. , 1990.

WEB SITES

[31] Several Benchmarks: http://ftp.sunet.se/pub2/benchmark/
[32] BOBMark: http://www.ece.orst.edu/~rose/bobmark/
[33] STREAM: http://www.cs.virginia.edu/stream/
[34] STKIO: http://www.stortek.com/StorageTek/software/freeware/
[35] Netperf: http://www.netperf.org/netperf/NetperfPage.html
[36] Perfect Club: http://www.csrd.uiuc.edu/benchmark/benchmark.html
[37] EuroBen: http://www.phys.uu.nl/~steen/
[38] GENESIS: http://www.hpcc.ecs.soton.ac.uk/RandD/genesis/genesis.html
[39] PARKBENCH: http://www.netlib.org/parkbench/
[40] Performance Database Server (PDS): http://www.netlib.org/performance/html/PDStop.html
[41] NAS: http://science.nas.nasa.gov/Software/NPB/
[42] TOP500 Supercomputing sites: http://www.top500.org/
[43] SPEC: http://www.specbench.org/
[44] SPLASH: http://www-flash.stanford.edu/apps/SPLASH/
[45] TPC: http://www.tpc.org/
[46] SLALOM: http://www.scl.ameslab.gov/scl/Projects/slalom1.html
[47] HINT: http://www.scl.ameslab.gov/Projects/HINT/