# AN INFORMATION-THEORETIC METHOD FOR ESTIMATING THE PERFORMANCE OF COMPUTER SYSTEMS

*Boris Ryabko*

Siberian State University of Telecommunications and Information Sciences,
Institute of Computational Technology of Siberian Branch of Russian
Academy of Science, Novosibirsk, Russia; boris@ryabko.net

## ABSTRACT

We consider a notion of computer capacity as a novel approach to evaluation of computer performance. Computer capacity is based on the number of different tasks that can be executed in a given time. This characteristic does not depend on any particular task and is determined only by the computer architecture. It can be easily computed at the design stage and used for optimizing architectural decisions.

## 1. INTRODUCTION

The problem of computer performance evaluation attracts much research because various aspects of performance are the key goals of any new computer design, see, e.g., [1, 2]. Simple performance metrics, such as the number of integer or floating point operations executed per second, are not adequate for complex computer architectures we face today. A more appropriate and widely used approach is to measure performance by execution time of specially developed programs called benchmarks. The main issues of benchmarking are well known, we only mention a few. First, it is very difficult, if ever possible, to find an adequate set of tasks (in fact, any two different researchers suggest quite different benchmarks). Then, when a benchmark is used at the design stage, it must be run under a simulated environment which slows down the execution in many orders of magnitude, making it difficult to test various design decisions in the time-limited production process. As a consequence, the designers reduce the lengths and the number of benchmarks, which raises the question of conformity with real applications. Quite often, benchmarking is applied to already made devices for the purposes of evaluation and comparison. Here, the benchmarks produced by a hardware manufacturer may be suspected of being specially tuned just to facilitate sales. The benchmarks suggested by independent companies are prone to be outdated when applied to technologically novel devices. All these appeal to objectivity of evaluation results. The performance figures obtained in this way may be suitable for one kind of applications but useless for another.

We suggest a completely different approach to evaluation of computer performance which allows to circumvent the difficulties outlined above. The new approach is based on calculation of the number of different tasks that can be executed in time $T$. This is quite similar to determining the channel capacity in information theory through the number of different signals that can be transmitted in a unit of time [3]. If one computer can execute, say, $10^{10}$ different tasks in one hour while another one can execute $10^{20}$ tasks, we may conclude that the latter computer is more capable in doing its work. The number of different tasks does not depend on any particular task and is determined only by the computer architecture which, in turn, is described by the instruction set, execution times of instructions, structure of pipelines and parallel processing units, memory structure and access time, and some other basic computer parameters. All these parameters can be set and adapted at the design stage to optimize the performance.

It is important to note that, generally speaking, the number of different tasks grows exponentially as a function of time. Indeed, if we have two different tasks $X$ and $Y$, each executed in time $T$, then their succession $XY$ will require $2T$, and the whole number of different tasks will grow from $N$ to about $N^2$ (not $N^2$ exactly because there are some instructions that may start before and end after the moment $T$). So we may write $N(2T) \approx N^2(T)$ and, generally, $N(kT) \approx N^k(T)$, where $N(T)$ denotes the number of task whose execution time equals $T$. This shows informally that the number of tasks grows exponentially as a function of time. Formal arguments will be presented below. So it makes sense to consider $\log N(T)$ and to deal only with exponents, which may differ for different computers.

The idea of computer capacity was first suggested in [4, 5], where it was applied to Knuth's MMIX computer [7]. In this paper, we extend the approach to modern computers that incorporate cache memory, pipelines and parallel processing units. Thus we prepare a theoretical basis for determining capacities and making comparisons against benchmarks of well-known processors of Intel x86 family which was presented in [8].

## 2. COMPUTER CAPACITY

Denote by $I = \{u_1, u_2, \ldots, u_s\}$ the instruction set of a computer (processor). An admissible sequence of instructions $X = x_1 x_2 \ldots x_t$, $x_i \in I$, seen as a process in time,

is called a computer task. The term "admissible" means that the instruction sequence $X$ can be executed up to the last element without errors in computation (so-called exceptions), such as division by zero or illegal memory reference. We consider two tasks $X$ and $Y$ as different if they differ at least in one instruction, i.e., there is an $i$ such that $x_i \neq y_i$. Notice also the difference between the computer task and the computer program. The task, as we think of it, is the flow of instructions executed by the processor. It is produced as a realization of some program. For example, if the program contains a loop which is to be iterated 100 times, the corresponding task will contain the body of the loop repeated 100 times.

Denote the execution time of instruction $x$ by $\tau(x)$. Then the execution time $\tau(X)$ of a task $X$ is given by

$$\tau(X) = \sum_{i=1}^{t} \tau(x_i).$$

The number of different tasks whose execution time equals $T$ may be written as

$$N(T) = |\{X : \tau(X) = T\}|.$$

The main performance characteristic which is essential in our approach, is the computer capacity $C(I)$ defined as

$$C(I) = \limsup_{T\to\infty} \frac{\log N(T)}{T}. \qquad (1)$$

Notice that this definition is virtually the same as the definition of channel capacity in [3], where $N(T)$ means the number of different signal sequences of duration $T$. The majority of modern computers are synchronous devices, i.e., they operate in discrete time scale determined by a clock cycle. In this case $\tau(x)$ can be measured in the number of processor cycles. It was shown in [5] that if all $\tau(x)$ are integers with the greatest common divisor 1, then the limsup in (1) equals lim and always exists.

Notice also the following thing. Let there be given two computers with identical sets of instructions $I_1$ and $I_2$ apart that the first computer is twice faster than the second one, i.e. $\tau_1(x) = \tau_2(x)/2$ for any $x \in I_1$ $(I_2)$. From definition (1) we immediately obtain that the capacity of the first computer is two times greater than that of the second one, i.e. $C(I_1) = 2\,C(I_2)$. Apparently, this equation is quite natural.

The suggested approach can be applied to multiprocessor systems. Consider a computer system that consists of $l$ processors which can operate independently. Let each $j$-th processor has an instruction set $I_j$ and can perform $N_j(T)$ tasks in time $T$. Then the total number of tasks $N(T) = N_1(T)N_2(T)\cdots N_l(T)$, and from (1) we have

$$C(\otimes_{j=1}^{l} I_j) = C(I_1) + C(I_2) + ... + C(I_l)\,, \qquad (2)$$

where $C(\otimes_{j=1}^{l} I_j)$ is the capacity of the considered multiprocessor system. In particular, the capacity of computer system with $l$ identical processors is $l$ times greater than the capacity of computer with one processor. The same arguments are relevant to distributed computer systems, or computer networks. Note that (2) is not a simple sum if the processors have some shared resources, such as shared memory. In this case the individual capacities must be diminished due to competitions for shared resources.

The definition of computer capacity is quite general, it does not restrain us from using one or other model of computer task formation. We may apply restrictions on instruction sequences, consider dependence of instruction execution times upon preceding instructions, and so on. Generally, the calculation of the limit in (1) becomes a complicated combinatorial problem. But as a first step, we can use a simple method suggested by Shannon in [3] for finding the capacity of noiseless channel where code symbols had different durations. When we use this simple method, we assume that all sequences of instructions are admissible. Clearly, by doing that we obtain an upper bound of capacity, which we denote by $\hat{C}(I)$, because the number of admissible instruction sequences $N(T)$ cannot be larger than the number of all possible sequences, denoted thus by $\hat{N}(T)$. Despite this simplification, we take proper account of the effects of caches, pipelines and parallel processing, as will be shown below. More specifically, following [3], for the instruction set $I = \{u_1, u_2, \ldots, u_s\}$ we may state that the number of all possible instruction sequences must satisfy the difference equation

$$\hat{N}(T) = \hat{N}(T-\tau_1) + \hat{N}(T-\tau_2) + \cdots + \hat{N}(T-\tau_s).$$

Here $\hat{N}(T-\tau_j)$ is the number of instruction sequences of duration $T$ ending in instruction $u_j$. It is well-known from the theory of finite dirrerences that asymptotically, as $T \to \infty$, $\hat{N}(T) = Z_0^T$, where $Z_0$ is the greatest positive root of the characteristic equation

$$Z^{-\tau(u_1)} + Z^{-\tau(u_2)} + \cdots + Z^{-\tau(u_s)} = 1. \qquad (3)$$

So from the definition of computer capacity (1) we have

$$\hat{C}(I) = \log Z_0.$$

In what follows we will estimate $\hat{C}(I)$ as a first approximation of real computer capacity, realizing that there are more complicated and more exact methods of finding $C(I)$.

Consider some examples. Let the first computer has only two instructions and execution time of each instruction is one clock cycle. So we have $I_1 = \{u_1, u_2\}, \tau(u_1) = \tau(u_2) = 1$ and the characteristic equation is $2Z^{-1} = 1$. Hence $Z_0 = 2$ and the computer capacity $C(I_1) = \log 2 = 1$ bit per cycle. Now add a third instruction with duration 2 cycles: $I_2 = \{u_1, u_2, u_3\}, \tau(u_1) = \tau(u_2) = 1$, $\tau(u_3) = 2$. The characteristic equation is $2Z^{-1} + Z^{-2} = 1$, its greatest root $Z_0 = 2.414$. The capacity $C(I_2) = 1.27$ bit per cycle, it is greater than $C(I_1)$ due to "more rich" instruction set $I_2$.

In practice, the computer instructions are often built of operation codes and operands, which may be references to internal registers, memory, or some immediate data. The key point is that to find the computer capacity we must consider the instruction set containing all operations with

all combinations of operands. Let, for example, the computer have 8 registers, $2^{16}$ memory locations, and can perform two operations op1 and op2 of the following format: (op1 reg reg) and (op2 reg mem), where reg is one of 8 registers, and mem is a reference to one of $2^{16}$ memory locations. Let op1 require 1 cycle and op2 2 cycles. Then the characteristic equation will be

$$\frac{8 \cdot 8}{Z} + \frac{8 \cdot 2^{16}}{Z^2} = 1.$$

The solution $Z_0 = 757$ and $C(I_3) = 9.56$ bits per cycle.

## 3. ENTROPY EFFICIENCY

It should be noted that to calculate the computer capacity, no probabilities or frequencies of instructions are needed. It does not mean that all the instructions are assumed to be equiprobable. In fact, the capacity is attained if the instructions appear with some "optimal" probabilities. In other words, the capacity is a maximal value which can be obtained if we use the processor instructions with certain frequencies. A connection between the computer capacity and various probabilistic models is established with the aid of the notion of entropy efficiency. There the sense of "optimal" probabilities mentioned above is clarified.

Consider the situation when computer is used for solving a particular kind of problems. For example, we use computer for solving differential equations. In this case the set of tasks to be performed is a subset of all possible tasks. We assume that the tasks of the set of interest can be modeled as realizations of a stationary and ergodic stochastic process. Let $X = x_1 x_2 x_3 \dots$ be a sequence of random variables taking values over instruction set $I$. Denote by $P_X(w)$ the probability that $x_1 x_2 \dots x_{n+1} = w$, $w \in I^{n+1}$ for any $n \geq 0$. The entropy rate is defined as usually, see, e.g., [9]:

$$h(X) = \lim_{n\to\infty} -\frac{1}{n+1} \sum_{w \in I^{n+1}} P_X(w) \log P_X(w).$$

Now the entropy efficiency, as a measure of computer performance, is defined as follows:

$$c(I, X) = h(X) / \sum_{u \in I} P_X(u)\tau(u). \tag{4}$$

In other words, $c(I, X)$ is the ratio of the entropy rate of instruction flow $X$ to the average execution time of instruction.

To motivate this definition, notice that if we take a large integer $t$ and consider all $t$-element instruction sequences $x_1 \dots x_t$, then the number of "typical" sequences will be approximately $2^{th(X)}$, whereas the total execution time of any sequence will be approximately $t \sum_{u \in I} P_X(u)\tau(u)$. (By definition of a typical sequence, the frequency of any word $w$ in it is close to the probability $P_X(w)$. The total probability of the set of all typical sequences is close to 1.) So the ratio between $\log(2^{th(X)})$ and the average execution time will be asymptotically equal to (4) if $t \to \infty$. This observation shows the relation between computer capacity (1) and entropy efficiency: the former is defined through the number of all tasks, the latter through the number of typical tasks, executed in one time unit. Another conclusion from this consideration is that

$$c(I, X) \leq C(I). \tag{5}$$

Now we shall say some words about estimation of the entropy efficiency. To do that we must observe the flow of instructions generated by the application of interest. Then we may use any method known in Information Theory to estimate the entropy of the instruction sequence and probabilities of particular instructions. Again, the simplest approach is to consider the case where all instructions are independent and identically distributed (i.i.d. sequence). In this situation the definition of entropy efficiency may be re-written in the following form:

$$\hat{c}(I, X) = -\sum_{u \in I} P_X(u) \log P_X(u) / \sum_{u \in I} P_X(u)\tau(u).$$

It can be easily checked now by direct calculation that if $P_X(u) = Z_0^{-\tau(u)}$ for all $u \in I$, where $Z_0$ is the greatest root of characteristic equation (3), then

$$\hat{c}(I, X) = \log Z_0 = \hat{C}(I),$$

i.e. the entropy efficiency reaches the computer capacity and is maximal according to (5).

## 4. COMPUTER CAPACITY IN MODERN COMPUTER ARCHITECTURES

The most essential elements of modern computer architectures that influence the capacity defined in (1) are cache memory (usually organized in several levels), parallel execution units (such as floating point unit), instruction pipelines and closely connected branch predictors, and multiple cores (including such technologies as hyperthreading). In this section, we address all these issues and show simple ways of their solution when determining computer capacity.

To assess the effect of cache memory on computer capacity we observe what happens at every time instant. Let, for example, instruction "ADD REG, MEM" is executed which adds a word in memory to a register and stores the result in the register. In our approach to estimation of capacity we assume that any register and any memory location can be accessed. Let there be $R$ registers and $M$ words in memory available. To show the main idea, consider a cache memory consisting of two levels L1 and L2 of sizes $L_1$ and $L_2$, respectively. If the address MEM hits L1 cache, let the execution time of the instruction be $\tau_{\mathrm{L1}}$. Otherwise, if the address hits L2 cache, let the execution time be $\tau_{\mathrm{L2}}$ (usually, much greater than $\tau_{\mathrm{L1}}$). If the address is not cached, let the execution time be $\tau_{\mathrm{M}}$ (usually, much more greater than $\tau_{\mathrm{L2}}$). Suppose that L1 and L2 are not exclusive, i.e. a memory location cached in L1 is also cached in L2. Then the corresponding part of characteristic equation will look like this:

$$RL_1 \frac{1}{Z^{\tau_{\mathrm{L1}}}} + R(L_2 - L_1) \frac{1}{Z^{\tau_{\mathrm{L2}}}} + R(M - L_2 - L_1) \frac{1}{Z^{\tau_{\mathrm{M}}}}.$$

If L1 and L2 are exclusive then we should not subtract $L_1$ in the second summand. All other processor instructions that operate with memory can be considered similarly.

The other issue is the presence of some units U1, U2, ... that can operate concurrently with the "main" part that performs basic operations (e.g., FPU, MMX and XMM blocks in x86 processors). Although the instructions executed by those units usually alternate with basic instructions and may have dependences, to find an upper bound on computer capacity we may consider these units as independent processors, i.e. to find their own capacities and sum them up according to (2). However, we must take into account that some units may be mutually exclusive (e.g., FPU and MMX blocks cannot operate concurently in x86 processors since they are based on one and the same register pool [10]). The solution is to consider all subsets of mutually compatible units and calculate capacities of those subsets. Then, since we are interested in an upper bound of computer capacity, we may choose the greatest capacity estimate. For example, there are two compatible subsets in x86 processors: $\mathrm{MAIN} + \mathrm{FPU} + \mathrm{XMM}$ and $\mathrm{MAIN} + \mathrm{MMX} + \mathrm{XMM}$ (obviously, there is no need to consider the subsets of smaller sizes). The subset having greater capacity determines the capacity of the whole computer.

The next architectural feature is the pipeline processing combined with branch prediction. Instruction timings provided in documentation assume that the pipeline is optimally filled, i.e., there are no empty stages and execution time is determined solely by the complexity of instruction. However, the pipeline operation is stopped when a mispredicted branch occurs. The instruction that must follow the mispredicted branch is delayed for the number of cycles, $k$, equal to the number of pipeline stages from the fetch stage to the execute stage. The next instruction is delayed for $k-1$ cycles and so on. The exact model would require to consider all $k$-element instruction sequences with any mispredicted branch. But we prefer a simpler way, sufficient for obtaining an upper bound of capacity. Assume that after any mispredicted branch we wait for $k$ cycles before the execution of next instructions. That is, the execution time of mispredicted jump instruction is increased by $k$ cycles. Since the computer capacity is defined through the number of all computer tasks, we can separately consider predicted and mispredicted jump instructions.

Finally, we address the problem of parallelism which is essential in hyper-threading and multicore technologies.

It is demonstrated there that computer performance indicators obtained through calculation of computer capacity and by benchmarks are very close to each other. So the computer capacity approach definitely can be used at the design stage when benchmarking is time-consuming or not at all possible.

## 5. REFERENCES


[1] W. Stallings, Computer Organization and Architecture: Designing for Performance. Prentice-Hall, 2009.

[2] A. S. Tanenbaum, Structured Computer Organization. Prentice Hall, 2005.

[3] C. E. Shannon, "A mathematical theory of communication," Bell Sys. Tech. J., Vol. 27, 1948, pp. 379–423, pp. 623–656.

[4] B. Ryabko, "Using information theory to study the efficiency and capacity of computers and similar devices," Proc. of the 2010 Workshop on Information Theoretic Methods in Science and Engineering (Tampere, Finland, 16-18 August 2010) .

[5] B. Ryabko , "On the efficiency and capacity of computers," Applied Mathematics Letters, v. 25, 2012, pp. 398 - 400

[6] B. Ryabko, "An information-theoretic approach to estimate the capacity of processing units," Performance Evaluation, V. 69, 2012, pp. 267–273.

[7] D. E. Knuth, The Art of Computer Programming. Vol. 1, Fascicle 1: MMIX – A RISC Computer for the New Millennium. Addison-Wesley, 2005.

[8] A. Fionov, Yu. Polyakov, and B. Ryabko, "Application of computer capacity to evaluation of Intel x86 processors," 2nd International Congress on Computer Applications and Computational Science, November 15–17, 2011, Bali, Indonesia, (Springer, Advances in Intelligent and Soft Computing, Vol. 145, 2012, pp. 99–104).

[9] T. M. Cover and J. A. Thomas. Elements of Information Theory. Wiley, 2006.

[10] Intel 64 and IA-32 Architectures Software Developers Manual Volume 1: Basic Architecture, Intel Corp., 2011.