International Journal of Theoretical Physics, Vol. 24, No. 12, 1985

Uncontrollable Computational Growth in Theoretical Physics

R. Cuykendall[1]

Received May 29, 1985

Some new results in the theory of synchronous parallel computation indicate there may be fundamentally unavoidable limitations to computing in certain kinds of large computational problems arising naturally in science and engineering. These limitations are in the naturé of uncontrolled growth (discontinuous jumps) in computation times under fixed programming schemes, and arise for computations allowing arbitrary (uniform) inputs over F^n for sufficiently large n, where F is a finite field. Instances of such discontinuity may appear, for example, in very-large-scale Monte Carlo simulations, such as those being contemplated for carrying out quantum chromodynamics (QCD) computations on lattices of substantially larger size than is now practicable. In this case, the QCD simulation may encounter abnormally (and unexplained) long run times on particular internally generated updates, resulting in distortion among time-weighted runs. The mechanism of these updates is believed to satisfy our necessary assumption of fixed encodings over uniform inputs.

1. INTRODUCTION

Conventional computation theory primarily involves possibilities, not probabilities. Certainly a great many problems are known to be much easier on average than in the worst case. It is the thesis of this paper that, on the contrary, for certain kinds of physical processes, the density of irreducibly (uncontrollably) complex computations arising from questions about these processes is very large. In a recent paper (Wolfram, 1985) it has been suggested that such computational irreducibility occurs whenever a physical system can act as a computer, and that computational reducibility may well be the exception rather than the rule. That is, most physical questions may be answerable only through uncontrollable amounts of computation.

[1]California Institute of Technology/Jet Propulsion Laboratory, Pasadena, California 91109. Present address: Department of Electrical and Computer Engineering, University of Iowa, Iowa City, Iowa 52242.

Similarly, it has been argued (Feynman, 1982) that quantum mechanics cannot be simulated by locally interconnected computers when size-growth is controlled in such a way that the number of computer elements necessary to simulate a large physical system be proportional to the space-time volume of the physical system. Our results seem to imply that this argument should hold for any (fixed) computable relation between the number of computer elements and the space-time volume.

Using a source encoding result (Fine, 1975) we show that there exists an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f not in C, where C is the set of reasonably programmable binary functions in the sense that there exists a program P which yields an approximation $f' \in \delta(f)$ to f, that P run in time no longer than τ, and that $|P| < \gamma[K_\psi(f)]$ for all preassigned recursive τ, γ, and δ, γ increasing and δ such that for some $\varepsilon < 1$ and each binary function $g \in M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions. $K_\psi(f)$ is the Kolmogorov measure of the quantity of information contained in the function f.

Next we relate lower bounds on the VLSI complexity of approximately computing certain uniform sets of finite binary functions f directly to their information content $K_\psi(f)$. This suggests a hierarchy theorem in which we show the existence of infinite sequences of finite functions f of increasing degreee of amodularity which cannot be reasonably approximated to within $\delta(f)$ by modular (VLSI-efficient) functions. We further show that if all sufficiently large M_n contain at least one function f not in C, then every (reasonably accurate) computation allowing arbitrary inputs over $\Sigma^{n \geq n_0}$ is inherently amodular. A result (Helm and Young, 1971; Constable and Hartmanis, 1971; Meyer and Fischer, 1972) on the size of programs admitting speedups provides convincing evidence that practically all large computations are amodular.

In general, one is interested in finding an efficient algorithm for solving a problem, where the notion of efficiency may involve a variety of (perhaps yet unknown) computing resources. Here though, we are concerned with the single resource time. The time requirements of an algorithm are usually expressed in terms of a single variable, the size of a problem instance, which is intended to reflect the amount of input data needed to describe the instance. This is intuitively appealing because one would expect the relative difficulty of problem instances to vary roughly with their size. The size of a problem instance is often measured in an informal way, so if time requirements are to be compared in a precise way, care must be taken to define instance size in a uniform manner.

The description of a problem instance provided as input to a computer can be viewed as a single finite string of symbols, chosen from a finite fixed alphabet. Although there are many different ways in which instances of a given problem might be described, it is assumed that each problem has associated with it a fixed encoding scheme E which maps problem instances into the strings describing them. The input length for an instance I of a problem π is defined to be the number of symbols n in the description $E(I)$ obtained from the encoding scheme for π, and it is this number n that is used as the formal measure of instance size.

The time complexity for an algorithm, defined as the largest amount of time needed by the algorithm to solve for each input length $|E(I)|$ an instance I of that length, thus depends upon the particular encoding scheme chosen. However, the standard encoding schemes used in practice always seem to differ at most polynomially from one another, so that any algorithm having polynomial time complexity under one of these encodings will have polynomial complexity under all the others. As it is difficult to imagine an encoding scheme for a naturally occurring problem that differs more than polynomially from the standard ones, there has been general agreement as to what constitutes a reasonable encoding scheme. Although what is meant by reasonable cannot be completely (satisfactorily) formalized, the generally accepted meaning includes the notions of conciseness and decodability. The intent of conciseness is that instances of a problem should be described with the natural brevity that would be used in actually specifying those instances for a computer, without any unnatural padding of the input, as such padding might expand the input length so drastically that an exponential time algorithm would be artificially converted to an algorithm with only polynomial complexity. The intent of decodability is that, given any particular component of an arbitrary instance, a polynomial time algorithm can be specified for decoding a description of that component from any given encoded instance. In other words, for a problem to be considered realistically defined (encoded), the solution which satisfies the problem parameter values specified in each instance must itself not be so extensive that it cannot be described with an expression having length bounded by a polynomial function of the input length.

2. SOURCE ENCODING

The concern in information theory with only average program (codeword) length has obscured the problem of the efficient coding of individual message sequences. A theorem due to T. Fine (1975) demonstrates that any code that can be realistically decoded must, for many sources, assign incredibly long (exceeding $\gamma[K_\psi]$) programs P to some (not very many) messages relative to the minimum possible (incompressible) codeword length K_ψ. This limitation seems to be fundamentally unavoidable even under very weak assumptions concerning the definition of a running time bound, the concept of a neighborhood set of a sequence, and the degree γ to which the length of a program approaches its minimal value K_ψ. The concern with individual messages is formulated and treated within the framework of algorithmic information theory, rather than with respect to sources for which a relative frequency characterization of uncertainty is assumed.

The ability to compress binary data in the form of a binary sequence, for example, requires a compression (encoding) function E and a reconstruction (decoding) function ψ. If $|S|$ denotes the length of the data sequence S, it is hoped that $|E(S)|$ tends to be less than $|S|$ at least for those sequences in a finite message source M. The fundamental difficulty of data compression dealt with concerns ψ rather than E, the properties of the decoder rather than those of the encoder. The nature of the conflict is such that for many message sets it is practically impossible to decode some efficiently encoded sequences (incompressible or nearly incompressible input programs) from either probabilistic, nonprobabilistic, or unknown sources. There is a similar conflict between the degree to which a sequence is compressed, and the difficulty of doing so. Details of this argument are analogous to the first problem. Furthermore, the results remain valid even when the coding requirements are relaxed so that the decoder need reconstruct only a reasonable approximation to the encoded sequence.

Specifically, if $\tau(S)$ is the running-time bound on computational effort of decoder (receiver-computer) ψ accepting codeword (program) P for message S, and $\gamma[K_\psi(S)]$ is the upper bound to acceptable codeword length $|P|$ when the shortest codeword for S has length $K_\psi(S)$, then for many message sources M there exist messages $S \in M$ such that (1) if encoder satisfies γ, then decoder violates τ; (2) if decoder satisfies τ, then encoder violates γ. These conclusions seem to be fundamentally unavoidable, and remain valid even when the decoder is allowed to reconstruct only an approximation S' in a neighborhood $\delta(S)$ of S. Compatibility of these results with those of information theory is that detailed properties of coding systems for individual messages, and not ensemble average properties, are analyzed. In a sense, concepts of Kolmogorov (Kolmogorov, 1968; Chaitin, 1975) increase resolving power of information theory for looking at individual sequences, and thus reveal obstacles to uniformly good coding systems.

The formalization of program efficiency is effected through a measure of information introduced by Kolmogorov (Kolmogorov, 1965). Let s denote a finite binary string and let x denote an infinite binary sequence. The first n bits of x are written as x^n, x_n [sometimes written $x(n)$] is the nth bit of x and $l(s)$ is the length of s. We also write x^n to denote an arbitrary finite string of length n.

Definition. A programming language φ is a partial recursive (p.r.) function on the finite strings,

$$\varphi:\{0,1\}^* \to \{0,1\}^*$$

Definition. p is a φ program for s iff $\varphi(p)=s$.

Intuitively, programming languages (computers) are thought of as mappings from programs to their outputs. The value $\varphi(p)$ is the binary string output by the computer φ when it is given the program p. If $\varphi(p)$ is undefined, this means that running the program p on φ produces an unending computation with no output.

Definition. The Kolmogorov information in x^n relative to the programming language φ is

$$\begin{aligned} K_\varphi(x^n) &= \min\{l(p)\colon \exists p\, \varphi(p)=x^n\} \\ &= \infty \text{ otherwise} \end{aligned}$$

Definition. ψ is a universal programming language iff

$$\forall\varphi\exists c\forall s[K_\psi(s)\le K_\varphi(s)+c]$$

The existence of universal programming languages is well known (Rogers, 1967). Such languages ψ have the property that for any programming language φ, there is a constant c (which depends on φ) such that the shortest programs relative to ψ never exceed the shortest programs relative to φ by more than c, independent of the string s being programmed. Thus ψ is as succinct a relative description scheme as any. Therefore we define the Kolmogorov information measure simply as K_ψ. Kolmogorov information content is often equivalently referred to as algorithmic information content, program-size complexity, Kolmogorov complexity, or descriptive complexity, and inputs (programs) p for which $K_\psi(p)\ge|p|$ are called incompressible.

It has also been found useful to study programs which are given, as a separate input, the length of the desired output, where no charge is made for the length of this second input.

Definition. The Kolmogorov conditional information in x^n is

$$\begin{aligned} K_\psi(x^n\mid n) &= \min\{l(p)\colon \exists p\psi(p,n)=x^n\} \\ &= \infty \text{ otherwise} \end{aligned}$$

The two measures express a slightly different quality of the sequence x^n in assessing its information content. The quantity $K_\psi(x^n)$ gives the minimum length of programs for x^n which must contain, in addition to the distribution of 1's and 0's in x^n, also information concerning the length n.

The integer n can generally be expected to use about length $\log_2 n$ of the binary program p for x^n. On the other hand, the quantity $K_\psi(x^n \mid n)$ gives the minimum length of a program which need not contain information on the length n, but which only determines the distribution of 1's and 0's in x^n. This distinction is dramatic at the low-information end of the scale where the information needed to determine the distribution is less than $\log_2 n$.

A more general definition of information conditioned on an arbitrary amount of *a priori* information can be formulated as follows:

Definition. The conditional (Kolmogorov) information $K_\psi(s \mid I)$ of a binary sequence s given information I represented as a finite binary sequence is given by

$$K_\psi(s \mid I) = \min\{l(p) \colon \exists p \psi(p, I) = s\}$$
$$= \infty \text{ otherwise}$$

Theorem 1 (Fine, 1975). If τ, γ, and δ are recursive functions, γ increasing and δ such that for some $\varepsilon < 1$ each sequence S has a neighborhood $\delta(S)$ containing no more than $2^{\varepsilon|S|}$ sequences, then

$$(\exists l_0)(\forall l > l_0)(\exists \xi(l) \in M_{\lceil \gamma(l)/(1-\varepsilon) \rceil})\xi(l) \in \bar{C}$$

where $M_n = \{S \colon |S| = n\}$; $\lceil x \rceil$ is the smallest integer $\geq x$; $C = \{S \colon (\exists P)\psi(P) \in \delta(S), \rho_\psi(P) < \tau(S), |P| < \gamma[K_\psi(S)]\}$ is the set of reasonably compressible source sequences in the sense that P yield a reasonable approximation $S' \in \delta(S)$ to S, that P run in time no longer than τ, and that it is moderately efficient in that for some preassigned function γ, $|P| < \gamma[K_\psi]$, where K_ψ is the length of the most efficient possible codeword; ψ is a p.r. function which can be thought of as any one of countably infinitely many universal Turing machines using the encoding function E; ρ_ψ is the running time of P on ψ and is defined iff ψ halts.

The above result thus shows that there is an infinite sequence $\{M_{\lceil \gamma(k)/1-\varepsilon \rceil}\}$ such that each $M_{\lceil \gamma(k)/1-\varepsilon \rceil}$ contains at least one sequence $\xi(k)$ not in C. This conclusion can be extended to other sequences of subsets of $\{0, 1\}^*$, but it is not known whether it extends to all sufficiently large M_n. [Based on program-size vs. speedup results (Helm and Young, 1971; Constable and Hartmanis, 1971; Meyer and Fischer, 1972) for certain functions f, we conjecture it does extend to all sufficiently large message sources M_n, but the specification of n would be noneffective.]

3. VLSI COMPUTATION

There is a close connection between the theory of information transmission over a channel and complexity of computing. The above result can be extended to binary functions which leads to the following theorem.

Definition. $M_n = \{f: |f| = n\}$ and $C = \{f: (\exists P)\psi(P) \in \delta(f),\ \rho_\psi(P) < \tau(f), |P| < \gamma[K_\psi(f)]\}$ is the set of reasonably programmable binary functions in the sense that there exists a program P which yields an approximation $f' \in \delta(f)$ to f, that P run in time no longer than τ, and that $|P| < \gamma[K_\psi(f)]$ for all preassigned recursive τ, γ, and δ, γ increasing, and δ such that for some $\varepsilon < 1$ and each binary function $g \in M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

Theorem 2. There is an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f not in C.

Proof. The table of an arbitrary binary function $f: \Sigma^{\log n} \to \Sigma^m$, where $\Sigma = \{0, 1\}$, can be regarded as a binary sequence $S \in M_{mn}$ for some message source M_{mn} by concatenating the n rows of length m comprising the table. The encoding $E(S)$ is then a program P for computing f on (universal Turing machine) ψ. By the previous theorem there is an infinite sequence $\{M_{|\gamma(k)/1-\varepsilon|}\}$ such that each $M_{|\gamma(k)/1-\varepsilon|}$ contains at least one binary function $\xi(k) = f$ not in C. ■

(The restriction to binary functions in the theorem is clearly unnecessary, as any finite function of natural numbers into natural numbers can be transformed into an equivalent function f over Σ^*.)

So far, we have established (by Theorem 2) that uncontrolled (discontinuous) growth in the program length or computation time is unavoidable when considering finite uniform computations carried out on a sequential machine. Furthermore, uncontrolled growth is unavoidable in approximating such computations. The question naturally arises if we cannot somehow avoid these difficulties through parallelism in computing. That is, could the bottleneck between memory and cpu in sequential computers cause the discontinuous jumps in complexity with problem size?

In order to compute a family of functions in which the inputs and outputs are distributed among a number of processors, information must in general be transferred between the processors. The role of such internal communication requirements in contributing to the inherent complexity of computational problems is still poorly understood. In distributed systems it can be expensive both in time and hardware to send information between processors, and in some computations the processors spend significantly more time waiting for information to be transferred than in performing actual computation. In VLSI chips the computation is distributed over the chip, and the various processing elements must communicate via wires. These wires generally occupy more space than the processors themselves, and can therefore be a more significant factor in determining cost and performance (Mead and Rem, 1979).

The information transfer required in a distributed computation is defined to be the smallest number I such that, for any values of the inputs, the computation can be accomplished with a total of at most I units of information transferred between the processors. In general, the information transfer can be regarded as a measure of the inherent modularity of the function being computed. Finding a configuration for which the inputs or outputs are evenly distributed, but which requires small information transfer, is a way of modularizing the computation. If this is not possible, we say that the function is inherently amodular. In other words, any partitioning of the computational process demands highly interacting parts. This (information transfer) amodularity leads to an area-time tradeoff for VLSI circuits.

The basic model of VLSI computation allows great generality. It allows features which certainly are not even contemplated in the near future. There are three main components: the boolean function f which is to be computed, a synchronous circuit C that computes f, and a VLSI layout V that realizes C. We assume that C is a network of wires attached to each other and to gates. The gates of C can be "and," "or," or "not" gates of arbitrary fan-in and fan-out. Such a circuit, which may have feedback, computes f provided there is an input–output schedule that describes how the inputs and outputs of f are mapped onto the input and output wires of C. It is assumed that each input arrives once and each output leaves once. (The motivation for this is that otherwise we would be allowing the circuit "free" memory.)

Definition (Lipton and Sedgewick, 1981). A VLSI layout V is a (λ, μ_1, μ_2) layout of the sequential circuit C if there is a map that assigns to each gate g (wire w) of C a closed connected region of the plane g^* (w^*) so that (1) if w is an input or output wire of gate g, then g^* intersects w^*; and (2) for each $\lambda \times \lambda$ square S of the plane, (a) at most μ_1 gates g map to regions g^* that intersect S, and (b) at most μ_2 wires w map to regions w^* that intersect S.

It is further assumed that all of the g^* and w^* lie in a convex region R, the region of the layout. The layout area A for a function f is defined as the area of the smallest region R containing a VLSI layout V of a network C that computes f in time T. No assumption is made about the location of circuit inputs or outputs or how they are assigned to gates, but we do require that circuits use all their inputs. Condition (1) simply forces electrical connections to map to topological connections (the converse is not true because multiple layers are allowed). Condition (2) is a direct result of the limits of VLSI fabrication. It ensures that any such S (e.g., a transistor) can only "see" a fixed number of gates and wires, and therefore limits the number of layers that can be used at the same point to $\mu_1 + \mu_2$.

Theorem 3. There is an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f which cannot be computed or approximated within $\delta(f)$ by any VLSI circuit in less than AT^2 (equivalently $A^2T) > \Omega[h(f)]$ or AT^2 (equivalently $A^2T) > \Omega[h(K_\psi)]$, for all recursive h, and recursive δ such that for some $\varepsilon < 1$ and each binary function $g \in M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

Proof. By Theorem 2 we know there exists at least one f not in C for each M_n. Thus either f violates the allowed running time τ on ψ, or it violates the allowed program-size bound $\gamma[K_\psi]$ for f on ψ. If τ is exceeded for any recursive τ, then since a $\tau(n)$ time bounded Turing machine simulated on n bits by a boolean circuit requires $\theta[\tau(n) \log \tau(n)]$ gates (Pippenger and Fischer, 1979; Schnorr, 1976) and any planar circuit requires at least this number of gates, the result follows (Lipton and Sedgewick, 1981; Savage, 1981) with $h = [\tau \log \tau]$. If on the other hand γ is exceeded so that any program for f takes more than $\gamma[K_\psi(f)]$ tape squares to read into storage (for example, on an off-line machine with a two-way read only input tape), more than $\gamma[K_\psi(f)]$ time steps on ψ are required just to read the input tape. This leads to precisely the same argument as before, and thus the result is established. Note that the entire input tape must be read in the computation of f, since by Theorem 2 the necessary length $|P|$ of any program for f exceeds $\gamma[K_\psi(f)]$ if running time is to remain less than $\tau(f)$, and any lesser input in computing f will violate τ. If a model of computation is assumed whereby the input is already on a working tape, either the above argument or one relating Turing machine space polynomially to planar circuit depth (Borodin, 1977) leads to the stated result. (A result requiring the layout area A of any VLSI circuit performing binary multiplication be proportional to the total number of bits input to the chip appears in the literature (Brent and Kung, 1980), and is extended (Baudet, 1981) to functions corresponding roughly to shifting or transitive operations which depend on all their inputs.) ■

The connection pointed out earlier between on-chip information flow and inherent modularity of functions ignores the problem of input, and instead assumes each processor already has its roughly equally divided share of inputs in memory. Given enough input, no interaction among the processors would in theory be required, regardless of the function being computed. Account for program length must therefore be taken in defining the concept of modularity. Intuitively, we think of a computation being modular if the total time it takes can be arbitrarily reduced by partitioning among a large enough number of processors. Total time in practice obviously includes the time required to transfer program bits into the various processor memories. Thus modularity relates to all processor interactions, not just to on-chip flows. Using this global notion of processor interaction in measuring inherent modularity of computations, we can show that finite functions abound that are highly amodular. Moreover, such functions cannot be reasonably approximated by modular functions.

Definition. If a function (computation) f requires a program inefficiency (redundancy) γ, i.e.,

$$|P| = \gamma[K_\psi(f)]$$

for f to run within time $\tau(|P|)$ on ψ, then f has amodularity degree $h = \max(\gamma, \tau)$, the greater growth rate. If h is polynomial or less, f is modular. Otherwise f is amodular.

Theorem 4 (Hierarchy theorem). There is an infinite sequence of finite functions f of arbitrarily increasing degree of amodularity which cannot be approximated to within reasonable $\delta(f)$ by functions f' having an amodularity degree lower in the sequence.

Proof. Theorem 2 establishes the existence of infinite sequences of finite functions that take longer than τ time to compute on ψ, or require input longer than $\gamma(K_\psi)$. Such sequences exist for each pair (τ, γ), where τ is recursive and γ is recursive increasing. If ψ satisfies τ, then the input violates γ. The polynomial relation between Turing machine space and circuit depth (Borodin, 1977) thus requires chip interaction $> \gamma(K_\psi)$ to within a polynomial factor. The degree of amodularity therefore grows as γ if τ is small. If on the other hand the input satisfies γ, then ψ violates τ. For complexity bounds $\Omega(\log n)$ on input of length n, it is well known (Dymond and Cook, 1980) that sequential space and reversal are both $\Omega(\log \tau)$, and that reversal is polynomially related to aggregate depth (parallel time on combinational circuits which can reuse their gates). Thus either the space-circuit depth or reversal-aggregate depth relation requires chip interaction $> \log \tau$ within a polynomial factor. For the pair (τ, γ) we then have at least one (actually many) infinite sequence of functions f having degree of amodularity greater than γ or $\log \tau$ (to within a polynomial factor). By successively choosing faster growing τ and γ we obtain infinite sequences of functions with arbitrarily increasing degrees of amodularity. As ψ is required to compute (in Theorem 2) only an approximation f' to f, subject to constraints on the reasonableness of the approximation, we can replace each f by any f' within the δ neighborhood obtaining f' sequences instead of f sequences. Clearly f' has the same degree of amodularity as f, since the running time for any such f' approximating f exceeds τ or ψ requires more than $\gamma(K_\psi)$ bits to execute f', completing the proof. ■

Actually, the situation may be much worse. Theorem 4 establishes only the sparse existence of functions which are increasingly amodular, namely one such f for approximately each $\gamma(n)$ with $n \geq n_0$. Thus the distance between $\Sigma^{\gamma(n)}$ and $\Sigma^{\gamma(n+1)}$ can be very large, and in fact becomes arbitrarily large with ever increasing γ. The high-density existence of such functions, say one for each $\gamma(n_0)+k$, $k=0, 1, \ldots$ would seem to have serious implications for very large-scale computations, e.g., solving weather prediction equations on large uniform data sets. The next two results provide conditions for a high-density hierarchy theorem.

Corollary 1. If all sufficiently large M_n contain at least one function f not in C, then every (reasonably accurate) computation allowing arbitrary inputs over $\Sigma^{n \geq n_0}$ is inherently amodular.

Proof. If the conclusion of Theorem 2 extends to all sufficiently large M_n, then by the above proof $\exists n_0$ such that there is at least one f satisfying Theorem 4 for each integer $n \geq n_0$. A computation which allows all inputs over Σ^n includes such an f as a partial computation if $n \geq n_0$, and is therefore itself amodular even when only approximated to within δ. ■

There is strong evidence that the condition of Corollary 1 holds. Note that Theorem 2 implies that in order to stay within any given recursive running time bound τ, the program P for computing some f in M_n must grow more rapidly as a function of $K_\psi(f)$ than any computable γ. And since (Fine, 1975) $K_\psi(f') > \gamma(l_0)$, where n is the smallest integer $\geq \gamma(l_0)/1-\varepsilon$, $|P|$ must grow faster than any computable function of n. Now as we decrease τ, the function f begins to look like a function that has an almost everywhere (a.e.) speedup at the expense of an ever larger program. Moreover, the faster program cannot be effectively determined from the given program, nor can we effectively compute from which point on the speedup started.

Definition. Let φ_i be the ith partial recursive function of one variable in a standard Godel numbering (Rogers, 1967) of p.r. functions. A family $\Phi_0, \Phi_1, \ldots$ of functions of one variable is called a Blum measure (Blum, 1967) on computation providing (1) domain (φ_i) = domain (Φ_i), and (2) the predicate $[\Phi_i(x) = m]$ is recursive in i, x, and m.

Definition. A p.r. function φ is speedable if for all i such that $\varphi_i = \varphi$ and for all recursive functions h there exists j such that $\varphi_j = \varphi$, and

$$[\Phi_i(x) > h(x, \Phi_j(x))] \text{ a.e.}$$

Furthermore, φ is effectively speedable if j can be recursively computed from i and an index for h.

Intuitively, if φ is speedable then for every program i for computing φ and every recursive function h there is another program j for φ which is an h speedup of the first on all but a finite number of inputs x, where j

is an h speedup of i on argument x if

$$\Phi_i(x) > h(x, \Phi_j(x))$$

The question whether for a function f with h speedup there must exist a recursive function which bounds the size of program necessary to effect the speedup was originally posed by Blum (Blum, 1971). A negative answer for effective operators slightly larger than h appears in the literature (Helm and Young, 1971), where it is shown that functions f exist which have the property that if we are given any program P for computing the function and want to pass to a program P' which computes the function much more efficiently, then we can only do so at the expense of obtaining a much larger program. In fact, the function which describes the necessary increase in the size of the more efficient program P' must grow more rapidly than any recursive function. The functions f have speedup, but not only can one not effectively find the programs P' which admit the speedup, even if one could, their complexity must increase in such a way that their size becomes totally uncontrolled. It is thus evident that the speedup property is related to Theorem 2 in the sense that certain speedable functions exhibit the discontinuous trade between their program-size and computation time predicted by the theorem. That the programs for such functions are noneffective is of course already implied by Theorem 2.

Such behavior is shown to arise for certain operator speedups $R(\tau_j)$-$(n) < \tau_i(n)$ a.e. (ordinary speedups by recursive functions $h(n, \tau_j(n)) < \tau_i(n)$ a.e. are a restricted form of operator) and is conjectured to hold for all sufficiently large operators on the basis that it should be more difficult to bound the size of programs effecting large speedups than those bounding smaller speedups. A slightly stronger result appears (Constable and Hartmanis, 1971; Meyer and Fischer, 1972), the question of extension to all total effective operators R remaining open.

Theorem 5. If the Helm–Young (1971) theorem extends to all sufficiently large operators R, then Theorem 2 extends to all sufficiently large M_n.

Proof. Helm–Young prove that for every recursive function h there is an effective operator R only slightly larger than h, and a total recursive function f which has R speedup

$$R(\tau_j)(n) < \tau_i(n)$$

but for which the size of the program necessary to effect the speedup increases more rapidly than any recursive bound. If their result holds for any operator R sufficiently large that

$$R(\tau)(n) \geq h(n+1)$$

then f requires such programs for all running times $\tau \leq \tau_j$. If the size of a program necessary to effect an R speedup cannot be effectively bounded for some R-speedable $f \in M_{n\to\infty}$, then $\forall \gamma \exists n_0$ such that its initial segments f_n of length $n \geq n_0$ require programs P_n of size $|P_n| > \gamma[K_\psi(f_n)]$, where ψ computes f therefore f_n within time $\tau_j(n)$, and Theorem 2 holds for all $\tau \geq \tau_j$ and all sufficiently large M_n. If this behavior arises for all sufficiently large R, then Theorem 2 holds additionally for all $\tau \leq \tau_j$ and sufficiently large M_n. ■

4. DISCUSSION

In view of the fact that most simple (noncomposite) functions already have been shown to have time-efficient layouts only when wire area is nearly as large as the chip, it would not have been completely unexpected to find that running time (or VLSI circuit area) for arbitrary composite functions would exceed polynomial or simple exponential limits. For example, it is shown (Lipton and Sedgewick, 1981) that certain n-input functions which are easy to compute become difficult under union or composition if each input arrives once and each output leaves once. If inputs are allowed to arrive multiple times (i.e., at many different times during the computation), this result does not hold, but if we do not allow free boundary (off chip) memory, the on-chip storage area will correspondingly increase. However, the probable abundance of well-defined functions which exceed any recursive bound on the area-time product is unexpected.

Blum (1971) gave an axiomatic characterization of some of the properties which should be possessed by a measure of computational complexity and established the existence of speedable functions—computable functions which fail to possess optimal programs in a particularly strong sense. Recursion theorists tend to like such functions, and computer scientists tend to consider such functions somewhat pathological. It is shown (Alton, 1976) that such pathology is rampant: there is a great diversity of behavior among the collections of run times of different functions which do not possess optimal programs, where such diversity is gauged by certain algebraic criteria which have computational significance. Roughly speaking, these algebraic criteria concern the ways in which various functions can be intermixed to satisfy requirements that certain functions can or cannot be computed more easily than certain other functions.

It is in fact shown (Alton, 1976) that there are enough speedable functions so that they can be responsible for embedding every countable partial order into the set of all possible complexity classes with respect to an arbitrary measure (such as time). Such diversity thus includes all countable sets of incomparable complexity classes. Suppose, for instance, that computable functions $f=\varphi_i$ and $g=\varphi_j$ are responsible for making the complexity classes C_{Φ_i} and C_{Φ_j} set-theoretically incomparable. This means that any program which computes f takes more resource on infinitely many inputs than the particular program j which computes g, and any program which computes g takes more resource on infinitely many inputs than the particlar program i which computes f. Thus f and g are easy to compute on very different sets of inputs. We therefore see that there are enough speedable functions to model (encode by indexing) the ways in which various programs relate to one another with respect to the patterns of inputs on which those programs compute rapidly. The prevalence (distribution) of uncontrolled program growth among these functions is unknown.

REFERENCES

Alton, D. A. (1976). Diversity of speed-ups and embeddability in computational complexity, *Journal of Symbolic Logic*, **41**(1), 199-214.

Baudet, G. (1981). On the area required by VLSI circuits, in *CMU Conference on VLSI Systems and Computations*, Computer Science Press, Rockville, Maryland, p. 100.

Blum, M. (1967). A machine independent theory of the complexity of recursive functions, *Journal of the Association for Computing Machinery*, **14**, 322-336.

Blum, M. (1971). On effective procedures for speeding up algorithms, *Journal of the Association for Computing Machinery*, **18**(2), 290-305.

Borodin, A. (1977). On relating time and space to size and depth, *SIAM Journal on Computing*, **6**(4), 733.

Brent, R. P., and Kung, H. T. (1980). The chip complexity of binary arithmetic, in *the 12th Annual ACM Symposium on Theory of Computing*, pp. 90-120.

Chaitin, G. (1975). A theory of program size formally identical to information theory, *Journal of the Association for Computing Machinery*, **22**, 329-340.

Constable, R., and Hartmanis, J. (1971). Complexity of formal translations and speedup results, in *Proceedings of the 3rd Annual ACM Symposium*, pp. 244-250.

Dymond, P. E., and Cook, S. A. (1980). Hardware complexity and parallel computation, in *the IEEE FOCS Conference*, pp. 360-372.

Feynman, R. P. (1982). Simulating physics with computers, *International Journal of Theoretical Physics*, **21**, 467.

Fine, T. (1975). Uniformly reasonable source encoding is often practically impossible, *IEEE Transactions on Information Theory*, **IT-21**(4), 368.

Helm, J., and Young, P. (1971). On size vs efficiency for programs admitting speed-ups, *Journal of Symbolic Logic*, **36**(1), 21-27.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information, *Problemy Peradachi Informatsii*, **1**(1), 3. (In Russian.)

Kolmogorov, A. (1968). Logical basis for information theory and probability theory, *IEEE Transactions on Information Theory* **IT-14**, 662-664.

Lipton, R. J., and Sedgewick, R. (1981). Lower bounds for VLSI, in *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 300-307.

Mead, C. A., and Rem, M. (1979). Cost and performance of VLSI computing structures, *IEEE, Journal of Solid-State Circuits*, **14**, 455-462.

Meyer, A., and Fischer, P. (1972). Computational speedup by effective operators, *Journal of Symbolic Logic*, **37**(1), 55-68.

Pippenger, N., and Fischer, M. J. (1979). Relations among complexity measures, *Journal of the Association for Computing Machinery*, **26**(2), 361-381.

Rogers, H. (1967). *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York.

Savage, J. E. (1981). Planar circuit complexity and the performance of VLSI algorithms, in *CMU Conference on VLSI Systems and Computations*, Computer Science Press, Rockville, Maryland, pp. 61-68.

Schnorr, C. P. (1976). The network complexity and the Turing machine complexity of finite functions, *Acta Informatica*, **7**, 95-107.

Wolfram, S. (1985). Undecidability and intractability in theoretical physics, *Physical Review Letters*, **54**, 735.