# A Perspective on Computer Progress in the Last Five Years

**by**

**Maurice V. Wilkes**

**Abstract:** The paper surveys recent progress under the following headings: workstations, processor design, parallelism, local area networks, UNIX, computer security, electronic mail and facsimile, and programming languages. It updates an earlier paper published in 1984 [1].

Five years ago personal computers were already enjoying a great success. The personal computer, or rather the microprocessor on which it was based, was a product of the semiconductor industry which needed to find a profitable application for VLSI at a time (1970–71) when only a few thousand transistors could be put on a chip. They started with pocket calculators, went on to logic replacement chips such as might be used in a computer terminal and, as soon as enough transistors could be put on a chip, ended by re-inventing the computer [2].

The personal computer industry developed as a rapidly expanding bubble on the side of the mainstream industry. The personal computer market was largely a new market. Nearly all established computer users continued to do their serious work on time-sharing systems, which underwent a major improvement as a result of the spectacular development that took place of low cost minicomputers. Since 1984, we have seen the development of much more powerful personal computers known as workstations. These are sufficiently powerful to engage the attention of those who had remained loyal to time-sharing, and it may be said that the bubble is now merging with the main industry. In fact, workstations may claim a dual ancestry. On the one hand they are direct descendants of the personal computer which, as I remarked above, came from the semiconductor industry. On the other hand, as regards the philosophy behind their use and the kind of user environment that they provide, they are the natural outcome of the pioneering work done at Xerox PARC with the Alto and the Dorado.

Personal computers and the earlier workstations appealed to frustrated users who felt that they had suffered enough from overloaded time-sharing systems and unsympathetic computer centre managers; these users tended to stress the independence that having a computer in their own office and under their own control gave them. This attitude is now changing. The modern workstation user relies heavily on a local area network and makes use of all the central services traditionally associated with time-sharing, namely, central filing, printing, mail,

etc., except that he is self-sufficient as regards processor cycles. Even there, however, he may make use of the network to secure extra cycles on other computers, for example on other people's workstations, when he needs them. He may even be glad to use his workstation as a terminal on a time-sharing system that is available to him on the network. Timesharing systems are no longer endemically overloaded and system managers know their job better.

The larger workstations now rival in power all but the very largest minicomputers, and can fill similar roles. It is quite common for a workstation to have attached to it several terminals to which it provides a time-sharing service. We may expect that soon the only difference between a workstation and a minicomputer will lie in the packaging and in the amount of peripheral and communication equipment that goes with it.

Quite recently we have seen operating systems at last break free from particular computers. This happened to computer languages 25 years ago and for most of that time it has been apparent that one day operating systems would do the same. The breakthrough, as I think it may well be described, has been associated with the UNIX phenomenon, a subject to which I shall return later.

**Processor Design**

A major revolution in processor design has taken place with the triumph of the RISC movement. This movement began with a paper by Patterson and Ditzel, published in 1980, in which they challenged the principles on which processor instruction sets were then designed [3]. Instead of a set of complex microprogrammed instructions it was better, they asserted, to have a reduced instruction set consisting of instructions simple enough to be hard wired. They indicated the principles on which such an instruction set should be designed. They described a single chip processor being implemented at Berkeley and for which the name RISC was originally coined. Other papers appeared describing the MIPS at Stanford and the IBM 801 [4, 5]. It appeared that the latter had been in gestation within IBM research for some time.

RISC processors are almost twice as fast as the processors they displaced. They can be designed in half the time and take much less space on the silicon chip. The gain in speed in a RISC processor comes principally from the pipelining which the particular flavour of RISC instructions makes possible. At an earlier period, pipelining had been confined to high end machines. In a RISC processor at least three instructions are in different stages of execution at the same time. For this to be possible, the high-speed memory, assisted one does not need to say by a cache, must be able to supply instructions at the required high rate. It was the coming of memories fast enough to satisfy this criterion that made the RISC movement possible. In the days of slower memories, the designer attempted to keep the

processor busy by packing as much information into each instruction as possible. However, when he had provided instructions for multiplication and division there was little further he could do. An instruction for extracting a square root would not be used often enough to affect the overall speed to any appreciable extent and the same is true of instructions for speeding up loops. It is for this reason that attempts to microprogram processors to perform well on high level languages had such limited success.

It is no accident that the RISC movement came at a moment when it had become possible to simulate a processor design sufficiently accurately to obtain a reliable estimate of its performance prior to implementation. Simulation enables a number of designs to be compared and the effect of including this or that feature to be qualitatively evaluated. Experiments with simulators soon showed that intuition is a poor guide to performance when the statistics of usage patterns are an important element in the evaluation.

The fact that all RISC processors have similar instruction sets facilitates the retargeting of compilers, since the same intermediate language suits them all. There are no quirks for the writer of the code generator to waste his time on. Perhaps partly for these reasons, when the early RISC processors were being designed, there was no formal pressure to standardize their instruction sets. The *de facto* standards that are now emerging are all the better for this. The investment in compilers remains, of course, a heavy one.

The long awaited point at which it would be possible to put a processor plus a useful amount of cache memory on a single chip was achieved some time ago. The fact that a RISC processor occupied only a small area of silicon was of great importance in making this possible sooner than would have been the case if the RISC movement had not taken place.

Progress has continued and smaller feature sizes have made it possible to increase the number of transistors that can be accommodated in a given area. At the same time improvements in process technology, particularly by way of achieving greater cleanliness, have enabled chips to become larger. In consequence, it is now possible to accommodate on a single chip a RISC processor, an integer multiplier, and a fast floating point unit, together with a cache and a memory management unit. The process will continue, and it will become possible to find room also for the other units that go to make up a complete computer, such as a display processor, peripheral controllers, network interfaces, etc. We can thus look forward to having a complete processing system on a single chip, needing only a memory bank and peripherals to make it into a complete computer of great power and very low cost. Photographs of modern high density processor chips are shown in Figures 1 and 2.

For some years, the ability of the semiconductor industry to manufacture large

chips has been steadily improving, and the largest chips on the market have increased in size at a linear rate. There appears to be no technical reason why this trend should not continue. It will be interesting to see to what extent the semiconductor industry will in fact go in for very large chips. A further trend worth noting is that more layers of metal—up to four—are now being offered. Having this number available simplifies signal routing, the control of crosstalk, and the distribution of power and clock, and thereby contributes to the performance that can be obtained.

The first set of units mentioned above, namely, the RISC processor, the integer multiplier, the fast floating point unit, and the memory managing unit are those between which it is necessary for signals to pass at high speed. There is not the same need in the case of the other units. Their purpose is to act as an interface between the high speed units and the outside world. Thus, from the point of view of achieving the highest possible operating speed, it may be said that the priority list of functional units to be accommodated on a single chip has now been exhausted or is about to be exhausted. If speed is the principal consideration, any increment of silicon space that becomes available should be used in the first place to improve the performance of these units and to provide more on-chip memory, cache or otherwise. Eventually perhaps there will be enough space to accommodate all demands. Until then, making the best possible use of silicon real estate will be a major preoccupation of the system designer. He will arrive at a different decision according to whether he is aiming at the highest possible performance, or whether cost is also a consideration.

Now that we are within sight of being able to put a complete computer on a single piece of silicon by conventional methods, it is unlikely that we will hear any more in this connection of wafer scale integration based on redundancy techniques.

Five years ago, microcomputers were based on CMOS or some similar process, and it was possible to put the whole processor on a single chip. More powerful computers used multi-chip processors based on bipolar technology. There was a tendency to believe that this state of affairs would continue. At the same time it was observed that, as the switching times of bipolar transistors became shorter, the time taken for signals to pass from one chip to another became the critical factor. The development of systems of inter-chip interconnect that would be much faster that conventional printed circuit boards was, therefore, seen as something to which high priority should be given. IBM had already led the way with their Multilayer Ceramic Multichip Module and their *Thermal Conduction Module*, radical developments based on a multi-layer ceramic structure. A much discussed proposal was to use silicon or some other material as a substrate on which wiring would be deposited and to which the chips would be attached by means of solder bumps. The difficulty was to achieve a sufficiently high density of interconnect and at the same time to arrive at a satisfactory system for distributing power and

clock. An alternative approach was to develop something more closely resembling an ordinary printed circuit board, but smaller and capable of operating at a higher speed. While this may not achieve the ultimate speed, it has proved to be well matched to the higher level of on-chip integration that has now become possible. A recently announced interconnect system based on this approach is illustrated in Figure 3.

Not everyone saw the future in the above terms. Some people thought that CMOS would carry all before it and ultimately become the dominating technology for computers of all sizes. They discounted the argument that there is an intimate connection between power and speed and that for this reason bipolar technology must survive.

Recently, a development has occurred that puts the matter in a new light. Bipolar VLSI has begun to compete with CMOS as regards packing density. It is possible to achieve almost the same feature size as with CMOS, although twice as many transistors are needed to implement a processor. We may therefore expect in the near future to see experimental processors, complete with cache memories, implemented in this technology. Some could be versions of existing CMOS architectures. It may be, therefore, that we will see no more multi-chip processors.

Whether the use of gallium arsenide will enable even faster computers to be built remains to be seen. For multi-chip computers, even with silicon, the interconnect time was becoming the dominant factor, and doubts were expressed as to whether it would ever be possible to take advantage of the even higher switching speed of gallium arsenide. The development of gallium arsenide VLSI, with total gate counts approaching those available with silicon—now being held out as a possibility—would change this.

RISC principles could be challenged if the present balance of speed between a processor and memory were significantly disturbed. Since however the cache memory, on which the memory speed as seen by the processor immediately depends, is built using technology similar to that used for the processor, this seems unlikely to happen. Within RISC principles there is some scope for evolution, for example, as to how much instruction scheduling should be put on the compiler and how much instruction level parallelism the designer should aim for. There is always the danger that false intuition will reassert itself and that designers will be tempted to put in features which, if statistically evaluated using simulation, would be seen not to enhance performance.

### Parallelism

The last five years have brought an increased understanding of the nature of parallelism and its exploitation. There is no longer the same general disposition

among computer specialists and others to believe that a breakthrough is just round the corner.

When low cost microprocessors first appeared on the horizon the problem that presented itself was how to use them to build multiprocessors of high performance. The difficulty was one of memory bandwidth. If all the microprocessors were to work out of the same memory, there would not be enough bandwidth to keep them busy. The only solution available was to provide some private memory as well as shared memory. At the 12th Annual Symposium on Computer Architecture held in June 1985 no fewer than eight such systems, differing in their exact organisation, were described. They were too awkward to program to be offered for use on general applications, but their designers hoped that they might find application in special fields, such as fault tolerance and high volume data base enquiry.

In 1983, J.R. Goodman presented the idea of the snoopy cache, as it was later called [6]. This made it possible to design a symmetric multiprocessor system in which the local memory for each processor was in the form of a cache and so invisible to the programmer. It was widely felt that such systems would give high throughput and at the same time enable parallelism to be exploited to obtain high speed. Many people saw them coming into general use as successors to the minicomputer of the period. At the symposium just cited, two systems with a snoopy cache were described, one by the author of a paper and the other by the luncheon speaker.

Experience has indeed confirmed that speed can be obtained by parallelism in a significant number of scientific problems. To do so, it is necessary to keep most of the processors busy most of the time. This requires the investment of a non-negligible amount of effort on the part of problem analysts and programmers, thereby limiting the approach to high value calculations and frequently re-used programs. As a consequence, symmetrical multi-processors have not had the overwhelming success that their enthusiasts predicted for them. They do, however, continue to have their place among the offerings of the industry and for some workloads they are cost effective in a time-sharing role. Multi-processor workstations have been built, but there are no signs at present that they will become popular. A factor in the situation is that uniprocessor systems have continued to get faster. Moreover, memory costs have come down, so that the the incentive to share memory is reduced.

The new single chip processors all feature an on-chip cache. If these are used in a multi-processor configuration with a (snoopy) external cache, a problem of coherence arises. There is no entirely satisfactory solution to this problem, certainly not a simple one. It is necessary to bring enough information off the chip to enable the behaviour of the on-chip cache to be modelled and a formal

coherence algorithm to be implemented. This we will see happening.

The value of vector hardware is now established in those scientific fields in which the use made of programs justifies a large effort in developing them. Vector hardware, however, is of little value unless the scalar performance of the machine is of the highest standard achievable, since in most cases, when vectorisation has been carried as far as possible, it is the remaining scalar parts of the program that dominate. It is worth recalling that the Cray I, which established the credibility of vector processing, was not excelled in scalar performance by any comparable machine available at the time it was introduced. The challenge to the designer is how to provide vector capability without adversely affecting the scalar performance.

The symmetrical multi-processor is based on conventional processor and programming technology and may be said to be a development within the mainstream. A variety of less conventional systems have also been built with the object of gaining speed by parallelism. Among these are the Connection Machine and the Hypercube. They all suffer, at least as severely as the symmetrical multi-processor, from load balancing problems, and in addition, are handicapped by having non-standard programming systems. The formulation, now many years ago, of the data flow principle was seen as making possible a radical attack on the problem of parallelism. Unfortunately, workers in the data flow field have little progress to report and continue to struggle with what appear to be deep fundamental problems. In spite of much interest in the area of highly parallel systems, no really new ideas have appeared for a long time. The recently acquired ability to put large numbers of gates on a chip has not stimulated, as many people hoped it would, any radically new proposals. I do not expect that any breakthrough will take place. On the positive side, there a number of scientific areas in which people are prepared to provide the effort needed to develop optimised programs for vector machines and various forms of multiprocessor, and the last few years have seen a modest increase in that number.

### Local Area Networks

The new subject of wide band local area networks (LANs) burst on the world in the late 1970's with the development of rings and the Ethernet [7, 8, 9]. These were pioneered by computer engineers using computer techniques. As soon as it became clear that wide bandwidth LANs were what the computer world had been waiting for without knowing it, other interests made a bid for the business, notably purveyors of broadband cable television systems and of telephone PBXs [10]. In the event, these interests failed to make good their claims, and we are left with Ethernets and rings.

The fundamental limitation of the Ethernet to a relatively small territorial area

has not been greatly felt, since what is available has proved to be a good match with the ordinary need. Fairly satisfactory methods of interconnecting neighbouring Ethernets have been developed. Rings can run at much higher data rates than Ethernets and do not suffer from such a severe territorial limitation; indeed the development of fibre optics may be said to have removed the latter limitation altogether. At the present time, the application niche into which rings ideally fit is that of providing a backbone communication system to which Ethernets or local rings can be attached. High speed switches, based on integrated circuits, have recently emerged as a possible basis for a high speed LAN. It is unlikely that there will be further development of the Ethernet, and designers of the higher speed LANs of the future will have to look either to rings or to switches.

We will see in the course of the next five years a great increase in the bandwidth offered by LANs and by metropolitan area networks. This can be expected to lead to a revolution in thinking about where processing power should be sited, whether on the user's desk or remotely. It will also lead to a new view of the relationship between a local hard disk on a workstation and a central file server. A sufficiently fast LAN would make it possible for a user to keep all his files on a central server, and to access them directly without incurring a performance penalty. He could then do his work on any workstation to which he could get access without the tedious operation of first copying files from his own workstation. It remains to be seen whether the use of a local disk on a workstation to hold current files will come to be seen as a temporary aberation.

We are now faced with the exciting prospect of having vastly more bandwidth than ever before available for medium and long distance communication as well as for local communication. Many of the uses to which this can be put are obvious enough, since computer communications have always suffered from bandwidth famine. The transmission of images and live video are new possibilities, hardly explored up to the present. The immediate need is to acquire some operational experience and user reactions without waiting for high performance systems to be developed. The Pandora project (see Figure 4, sponsored by Cambridge University and Olivetti Research, sets out to do this [11].

### The Phenomenon of UNIX

UNIX was developed in a research environment. When it was released, it had an immediate appeal to computer scientists in research laboratories. It was powerful and allowed them to build ingenious and effective systems. Their background enabled them to cope easily with the technical problems that the use of UNIX presented, and they did not mind working with the documentation provided which, although of lower standard than that provided with other operating systems, was in a style that appealed to them; they were quite happy to pass information by word of mouth and enjoyed being gurus.

Soon other researchers began to want UNIX in order that they might take advantage of the things that were available with it—included in the UNIX package was a valuable library of miscellaneous functions—or to use systems written in C. The growth in popularity of this language was closely bound up with the spread of UNIX, the one supporting the other.

It was was by no means an expected development that UNIX should emerge from the back room to become a widely used operating system. A major factor was undoubtedly the fact that UNIX had already been implemented on a number of different processors, so that when the world wanted a system not tied to the products of any particular vendor, no other system of comparable merit was available. To this must be added the fact that the developers of the first experimental workstations were all UNIX devotees and it was natural that, when workstations came on the market, they were equipped with UNIX. At this point, windows came into general use and, more often that not, the ordinary user now approaches UNIX through a windowing system. He is thus insulated from the UNIX command structure, which does not provide a very easy interface for the less sophisticated.

Although windowing systems may be said to have saved UNIX as far as the general user is concerned, they consume a significant proportion of the computing cycles, even of a powerful processor. It may be that the larger workstations will come to incorporate a separate processor dedicated to this function. However, in my view it is more likely that the majority of workstation users will regard the support of the environment in which they work, including the windowing system, as a good use for workstation computing cycles, and will be willing to go to central compute servers for any additional cycles they require.

At the present time UNIX exists in two major versions—AT&T and Berkeley—and in various flavours. In this respect it very much reminds one of the state of FORTRAN before the ANSI standard received general acceptance. Documentation, although improving, is still poor and a newcomer needs the help of an expert.

### Computer Security

Early computers were used in such a way as to present no security problems. F. Corbató, the designer of the CTSS, the first of the large scale time sharing systems, saw clearly that the sharing of a computer by many users simultaneously would create a need for security features. Accordingly the CTSS incorporated a password system and also had file protection. As time went on, experience showed that these security features were not as effective as had been innocently thought. The early time-sharing systems operated in university environments and the students were quick to take up the challenge.

The problem was not at first met in industry; staff after all are not like students. Later, however, when remote login and networking had become common, systems managers would sometimes find, to their consternation, that unknown strangers were getting into their systems.

There has been much discussion of computer security from a military point of view and in some ways this has confused the issues because military and business security are very different. In the military world the importance of security is taken for granted, whereas in industry money spent on security must be justified on the same terms as other expenditure. Unless their complacency is disturbed, industrial managers prefer not to know about security. Again the military face the threat of well funded professional espionage taking place outside the reign of law. In civil life laws can be enforced and there are also cultural restraints. One of the difficulties is that the criminal law in many countries is weak when it comes to computer penetration and law makers are having to be convinced of the need to strengthen it.

Recently computer networks have been attached by *worms* and this has done much to alert users to the danger they are in. A worm is a free standing program designed to exploit known or suspected security loopholes. When introduced into a computer, it systematically searches for routes by which it can penetrate other computers on the network. If successful, it implants copies of itself, and the process continues. Worms operate at high speed and, once they have made an initial penetration, operate from within the system; for these reasons, they impose a far greater threat than an unaided human intruder could do.

It is not difficult to *design* a computer operating system which, if used in a secure manner, will provide a standard of security high enough for most business purposes at the present time. The difficulty is to make sure that there are no *implementation* bugs that compromise security. If it were possible for the implementer to do everything in a safety first manner, the difficulty would be less, but in practice optimisations are essential to secure acceptable performance and it is these optimisations that are apt to breed bugs. A good example is clearing blocks on a disk when they contain information no longer required. Efficiency is promoted by postponing this operation until new information comes to be written, since the old information will then be automatically destroyed. However, if the implementer is insufficiently alert, there is a danger that, in some circumstances, the user will be able to read information in a block before he has written to it. Many mischief makers have exploited security loopholes of this kind to their advantage.

During the past decade a stupendous effort has been made to improve the security of the leading proprietary systems. This has been done in part by identifying and removing design flaws and implementation bugs, and in part by adding improved

features for such functions as access control and auditing. These systems now provide a degree of security that is good enough for normal purposes, provided that any insecure (but convenient) facilities they offer are disabled and that they are used in a secure manner. Unfortunately, neither of these things can be relied on. Many common user practices, mostly connected with the safe custody of passwords, seriously compromise security.

As far as the security of operating systems is concerned, the coming of UNIX has taken us back to square one. UNIX was designed to give maximum flexibility and power. Security was not a goal; in fact, since it conflicts with the above objectives, it may be described as a non-goal. It is not to be wondered at that UNIX is giving rise to security concerns. An organisation wishing at the present moment to set up a reasonably secure time-sharing system would be well advised not to choose UNIX as the operating system. Anyone who thinks that a few fixes will suffice to make secure the versions of UNIX currently in general use is deceiving himself. The nature of the UNIX market is such that proprietary versions for which security claims are made cannot be expected to have more than a very limited impact.

Although the patient identification and removal of design flaws and implementation bugs has yielded fruit, it would obviously be desirable if more formal and less uncertain methods could be evolved for arriving at a secure system and providing evidence that it is secure. Everyone who has thought about the subject has his own pet ideas.

The difficulty about giving a formal proof that a system is secure—along the lines of a proof that a program meets its specification—is that one is trying to prove a negative, namely that the system cannot be penetrated. Some element of inspection, as distinct from proving, would appear to be necessary and one would like to see this made as formal as possible. Since implementation bugs are a major worry, the inspection must be repeated whenever the system is transferred to another computer.

There has been some discussion of ways in which the operating system might be structured so as to facilitate, and make more systematic, the task of the inspector. One suggestion has been to restrict to an absolute minimum the amount of information that the running process can access at any given moment. This is expressed by saying that the *domain of protection* in which the process runs is small. Unfortunately, if everything is done in software, the frequent change of domain which this approach makes necessary leads to unacceptable loss of performance. Attention was accordingly directed to providing hardware support for domain switching. An early suggestion, implemented in MULTICS, was to have rings of protection. The amount of information available to a process decreased as it moved from the inner to the outer rings. Unfortunately, the

hierarchical model of protection which this implied is fundamentally flawed, and it was found that rings of protection were little improvement if any on the simple system of a privileged and an unprivileged mode.

A further idea investigated was the provision of hardware support for *capabilities*. Capabilities are simply tickets, the mere possession of which gives the right to use or access some resource. As a practical approach capabilities proved disappointing for a number of reasons, one of which was that the software required to manage the capabilities proved to be unexpectedly complex.

As a theoretical model of computer security, however, the capability model is an attractive and powerful one. For example, if certain restrictions on the passing of capabilities from one procedure to another are accepted, it is easy to verify that the running process has access to what it needs and no more. This makes it vastly easier to mount a defence against Trojan Horses and viruses. Unfortunately the advantage is a theoretical one only, since the capability model cannot be efficiently implemented. In a capability system, all the protection is enforced at run time. Some protection can be bound into a program at compile time and, indeed, programming languages do this to a certain extent. In the general security context, however, the compile time approach has not proved capable of useful development.

It is perhaps just as well that attempts to devise effective hardware support for security within the processor have not been successful. Processor development has been driven by the demand for the highest possible performance and processors with special security features would have been left behind in the race.

Until recently, most thinking about security was in terms of the design of an operating system for a large public time-sharing system and was based on the assumption that the users were not mutually trustworthy and that security barriers must be erected to prevent them from interfering with one another and with the system. Apart from the password system for logging-in, these are the only barriers that exist. If an intruder succeeds in logging-in, for example by guessing a user's password, the barriers that prevent him from then getting access to other user's accounts are exactly the barriers that are established between regular users.

The above model, however appropriate in it proper place, does not fit the many departmental timesharing systems now in operation. These serve a relatively small group of cooperating users, between whom no security barriers need exist. They can be trusted not to pry into each others files, and it does not matter very much if they do. What is needed is to make the system into a self-contained secure enclave from which strangers are firmly excluded. One way of making this possible is not to allow external users to log-in directly, but to insist that all remote connections are made from within the enclave itself.

In the most secure implementations of this principle, a dial-out capability—to the telephone system or to a computer network—is provided, but no dial-in capability. If a distant user wishes to be connected, he must send a message through an independent channel, for example by telephone, or via another computer (possibly insecure) on the net. If he can establish his credentials, then the connection is established by an operator from within the secure enclave. In more common, but less secure, implementations the user is allowed to log-in, but only for the purpose of entering a request to be automatically called back at a location known to the system. A secure enclave can comprise a local time-sharing system as described above, or it can comprise a group of workstations with file servers on a LAN. It can even consist of a single personal workstation.

The problem is essentially one of the user authenticating himself. The integrity of a secure enclave depends entirely on the enforcement of security at points of entry. Any system based solely on passwords or encryption keys is vulnerable to lack of care on the part of the users. The call back procedure has the merit that it provides information that is not provided by present day communication systems, namely, the location of the site from which the caller is operating.

The concept of the secure enclave is not appropriate in all circumstances but, where it is, it has much to commend it. It puts the onus for security on the owner of a workstation or on the person in charge of a group. He can take his own decision as to how high a degree of security is necessary, or even whether to worry about security at all. What he decides to do will have no effect, good or bad, on other enclaves. Within an enclave, an operating system like UNIX, designed for convenience rather than security, is perfectly satisfactory. Security exists—and is seen to exist—to keep strangers out, not to come between friends.

**Electronic Mail and Facsimile**

The electronic mail situation is far from satisfactory. Instead of a uniform system, we have a large number of separate networks with more or less unsatisfactory gateways between them. The addressing system is a source of great confusion to users, and often messages are returned or fail to be delivered for trivial reasons.

Meanwhile, a powerful competitor to electronic mail has arisen in the shape of facsimile, known for short as faxO Fax developed from the paper office and is compatible with it. It uses the telephone network and messages are sent to a telephone number; they are then routed to the recipient by office staff in the ordinary way. Since there is no queuing in the network, successful transmission implies receipt of the message. If the connection cannot be established, the message remains at the sending station, and the fact is obvious to all concerned. For the sake of a better name, I will refer to this as *immediately delivery.*

The triumph of fax has been made possible by the dramatic improvement that

has taken place during the last five years in the reliability and availability of the telephone network, especially the international telephone network. This improvement has been matched by an equally dramatic improvement in the performance of high speed modems.

Fax exploits the universal nature of the telephone network. The user sees one hop to the destination. The telephone company worries about routing and redundancy, and is also responsible for the maintenance and upgrading of the switches and lines. Fax automatically gives immediate delivery. Some electronic mail systems, based on a network of leased lines, also give immediate delivery, but many emphatically do not. They work on the store and forward principle. Whatever the initial specification of such a network, and the performance of a pilot system, in practice the emphasis will inevitably come to be on 'store' rather than on 'forward'. Queues will build up within the network, and at gateways to other networks. Messages will become subject to delays, which are all the more serious because they are uncertain. Usually, if gateways are involved, no positive indication is given of delivery. Messages often get lost and the sender does not know.

Fax is based on transmission of scanned images, electronic mail on the transmission of ASCII characters. The gap can be expected to close. The storage and transmission of scanned images is already engaging the attention of the computer research community. Similarly, we may expect to see fax systems in which information is sent in ASCII form when that is available, otherwise as a scanned image. Soon, perhaps, character recognition will enable scanned text to be converted to ASCII—at least in some useful cases. There is no problem about turning ASCII into a scanned image.

These developments have still to happen. At the present time fax is compatible with a paper office, but incompatible with electronic mail. It is, therefore, more acceptable to companies which are not far advanced in the use of computers than to those in which all documents exist in computer form. In some places, *fax servers*, which enable fax messages to be sent from a computer terminal, are in use, and help to mitigate the incompatibility with electronic mail.

Provided that the recipient's fax number is known, a message may be addressed to him using his ordinary name. In the case of an electronic mail message, it is necessary to know his local user identifier. For example, J Brown may be known locally as jb or jbrown. It would be a major step forward if electronic mailing systems were designed to accept ordinary names. At the receiving node, the message would be routed to the correct individual by a simple expert program which could easily deal with the common run of combinations of names and initials, and even with common mis-spellings. If the expert program failed to identify the recipient, the message would be referred to the equivalent of a mail

room clerk. The sender would not then be subjected to the irritation of having his message returned because he had mis-spelled the name. I have heard of a bank which uses an expert system of exactly the kind described to route to the appropriate branch incoming messages dealing with fund transfers.

### Programming Languages

Things move slowly in the computer language field but, over a sufficiently long period of time, it is possible to discern trends. In the 1970s, there was a vogue among system programmers for BCPL, a typeless language. This has now run its course, and system programmers appreciate some typing support. At the same time, they like a language with low level features that enable them to do things their way, rather than the compiler's way, when they want to. They continue, to have a strong preference for a lean language. At present they tend to favour C in its various versions. For applications in which flexibility is important, Lisp may be said to have gained strength as a popular programming language.

Further progress is necessary in the direction of achieving modularity. No language has so far emerged which exploits objects in a fully satisfactory manner, although C++ goes a long way. ADA was progressive in this respect, but unfortunately it is in the process of collapsing under its own great weight. ADA is an example of what can happen when an official attempt is made to orchestrate technical advances. After the experience with PL/1 and ALGOL 68, it should have been clear that the future did not lie with massively large languages. I would direct the reader's attention to Modula-3, a modest attempt to build on the appeal and success of Pascal and Modula-2 [12].

### Acknowledgments

I am grateful to many colleagues for commenting on drafts or partial drafts of this paper, especially to: J. Dion, D.A. Gaubatz, J.L. Hennessy, A. Hopper, M.A. Johnson, P.A. Karger, S.B. Lipner, R.M. Needham, P. Robinson, N.E. Wiseman. It goes without saying that I alone am responsible for the statements made and opinions expressed.

### References

1. WILKES, M.V.: 'Past, Present, and Future of the Computer Field', *Proc. IEE,* 1984, **131** Part E, pp. 106–112

2. WILKES, M.V.: 'The Past and Future Development of Personal Computers', *Computer Standards and Interfaces,* 1988, **8**, pp. 5–7

3. PATTERSON, D.A., and DITZEL, D.R.: 'The Case for the Reduced Instruction Set Computer', *Computer Architecture News,* 1980, **8** no. 6 pp. 25–

4. HENNESSY, J.L., JOUPPI, N., BASKETT, F., GROSS, T.R., and GILL, J.: 'Hardware/software Trade-offs for Increased Performance', *Proc. Symposium on Architectural Support for Programming Languages and Operating Systems,* 1982, pp. 2–11

5. RADIN, G.: 'The 801 Minicomputer', *Proc. Symposium on Architectural Support for Programming Languages and Operating Systems,* 1982, pp. 39–47

6. GOODMAN, J.R.: 'Using Cache Memory to Reduce Processor-Memory Traffic', *Proc. 10th Annual Symposium on Computer Architecture,* (1983), pp. 124–131

7. WILKES, M.V.: 'Communication Using a Digital Ring', *Proc. PACNET Conference,* 1975, pp. 47–55, Sendai, Japan

8. WILKES, M.V., and WHEELER, D.J.: 'The Cambridge Digital Communication Ring.' *Proc. Local Area Communications Network Symposium,* 1979, NBS special publication (ed. Meisner and Rosenthal) p. 47, Mitre Corp. and NBS

9. METCALFE, R.M., and BOGGS, D.R.: 'Ethernet: Distributed Packet Switching for Local Area Computer Networks', *Comm. ACM,* **19,** 1976, pp. 395–404

10. WILKES, M.V.: 'The Impact of Wide Band Local Area Communication Systems on Distributed Computing', *Computer,* 1980, **13,** no. 9, pp. 22–25, IEEE

11. HOPPER, A.: 'Pandora - An Experimental System for Multimedia Applications', *Operating System Review* 1990, **24** no. 9, pp. 19–34, ACM

12. L. CARDELLI, J. DONAHUE, M. JORDAN, B. KALSOW, and G. NELSON.: 'The Modula-3 Type System,' *Proc. 16th Annual ACM Symposium on Principles of Programming Languages,* 1989, pp. 202–212

26 September 2005

**Captions for Figures**

Figure 1. MIPS R3000 processor Chip - photograph and caption to follow

Figure 2. Layout of the INTEL i860 processor chip.

The *Integer RISC Core*, that is the basic RISC machine, occupies only a small part of the chip. There are two caches, the *Instruction Cache* and *Data Cache*. The *Floating Point* units together constitute a highly parallel, pipelined, numerical coprocessor. The chip is 1.5 cm by 1.0 cm and contains a million transistors.

Figure 3. The multi-chip unit used in the VAX 9000 System.

The chips, of various types, are carried on a High Density Chip Carrier (HSDC) which performs the same function as a printed circuit board, but is three to five times denser. There are nine layers of wiring used for signals and power. A polyimide synthetic material is used for insulation and a typical line pitch for the signal interconnect is 75 microns. This high density of wiring is obtained by using for manufacture similar equipment to that used in the semiconductor industry. The chips are gang soldered directly to the substrate using Tape Automated Bonding (TAB) without any chip holders.

One HSDC (4 in by 4 in) is able to accommodate as much logic as was accommodated in earlier VAX models on four 15 in by 12 in printed circuit boards. A power dissipation of up to 300 watts is possible with air cooling.

*Courtesy: Digital Equipment Corporation*

Figure 4. Pandora Project.

The picture on the left shows an experimental multimedia workstation installed in an office. On the right is a close up of the screen as seen by the user. This includes several X-windows. One contains a digitised video picture of a person in another office with whom the user is working. A smaller window contains a picture of the user himself; a copy of this is currently being transmitted to the remote user. Underlying the two windows is a window containing a computer program written in C.

*Courtesy: Olivetti Research Ltd*