CMOS Workstations and Servers—How Far Can Evolution and Innovation Take Us?

by

Maurice V. Wilkes
Olivetti and Oracle Research Laboratory
Cambridge, UK

In 1980 we had minicomputers and mainframes. These had once been quite distinct, but super-minicomputers were coming in and were beginning to challenge mainframes. All computers intended for serious work were based on bipolar technology. CMOS processors existed and were considered fine for personal computers, but no good for "real" computers.

There were then very few people who appreciated the potential of CMOS. To do so, one needed to believe in the scaling rules. This was not hard if one knew what they were, since they were grounded in the laws of physics. However, it was also necessary to believe in the ability of the semiconductor process industry to achieve very high densities. This was harder, since VLSI had seemed very slow in coming and it might be equally slow in developing. In the event, things went like a bomb. Stories abounded of serious problems ahead, but the industry rode right over them.

Architectural innovation was also in the air. John Cocke had done pioneering work at IBM, and Patterson and Ditzel had brought it out into the open in a paper published in 1980. In this paper, they introduced the term *Reduced Instruction Set Computer* (RISC). Once stated, RISC principles were clear enough. Instruction sets had become too complex and needed simplifying. Perhaps, as the originator of microprogramming, I was partly to blame, for microprogramming had gone to designers' heads.

RISC principles were a godsend to the designers of powerful CMOS processors, since RISC processors were twice as fast as processors with complex instruction sets, they were half the size, and they took half the time to design. It was not long before workstations based on CMOS processors with RISC instruction sets came to out-perform even the largest minicomputers. A powerful operating system, namely UNIX, was ready to hand. UNIX had already achieved widespread popularity on minicomputers and it was written in C. All that was needed to turn a RISC workstation into a useful computing tool was a C compiler. For a brief glorious moment, the instruction set of the processor did not matter. As a result, a variety of RISC processors were able to achieve a share in the market.

This was all innovation. However, it was not long before personal computers based on the x86 instruction set had evolved to the point at which their performance approached that of RISC workstations. Innovation had provided a spur for evolution.

Parallelism not a Panacea

It has always been obvious that one day we would reach a limit to the speed of hardware. Long before this point was in sight people began to discuss what we would do when we reached it. Everyone said that parallelism was the answer. We would only need to switch to multiprocessor systems and the movement towards higher and higher speeds would continue as before. They pointed to the success of optimizing compilers for uniprocessors and said that it was only a matter of time before compilers capable of optimizing programs to run efficiently on multiprocessor systems were developed. In retrospect, this appears as a naive attitude. Optimizing for a multiprocessor system can well involve major restructuring of the program and even changes to the algorithms used. These things are difficult enough for a human being and well beyond what can be expected from a compiler in the foreseeable future.

The climate of opinion is very different now. Much work has been done on the use of parallel systems and the difficulties are more fully appreciated. Computer specialists have come to understand that some applications lend themselves well to parallelism, while others are extremely resistant to it. In the past, one could be a computer expert without knowing very much about application areas. This is not true where parallel applications are concerned. Designers need all the feedback they can get from people who actually use parallel systems.

Speed versus Throughput

Computers of today normally operate in one of two roles. There are single-user computers—typified by a workstation on a user's desk—and servers. Servers are centrally located and offer a wide variety of services to all comers. These range from filing systems and databases to time-sharing systems.

By its nature, a server handles a large number of individual tasks presented by its clients. These tasks may contend for access to bulk storage, but otherwise they run quite independently of one another. For this reason, servers lend themselves very well to implementation on multiprocessor systems. Frequently these are symmetric multiprocessor systems with coherent caches and 16 or more processors. Use of a multiprocessor system does not enable a server to process individual tasks more rapidly than a single processor could do, but it does enable many more tasks to be completed within a given time. This is achieved by keeping all parts of the system fully loaded. In other words, the parallelism gives increased throughput, not increased speed.

High throughput is exactly what is needed in a server. On the other hand, an individual user with his own workstation is primarily interested in the speed at which his own application runs. Multiprocessor workstations are available, but experience with them has shown that only rarely can an individual user who writes his own programs speed up his work significantly by making use of the parallelism they offer. This is largely because the labor of casting a problem into parallel form and debugging the program is too great. The situation is entirely different if he can use ready written and debugged packages available from a software vendor and take advantage of work done by the vendors software team. For example, multi-threaded 3D graphics packages run vary well on a multiprocessor workstation.

Parallel programming tends to flourish in large organizations, especially those devoted to particular scientific subject areas, where the investment of effort needed to develop parallel programs can be afforded. In such organizations, large parallel computer systems are commonly operated as a common facility and shared by a variety of application teams all working in the same general area.

Sometimes the computers are symmetric multiprocessors, similar to those used in servers. However, they are often very large systems with hundreds or even thousands of processors in which the high-speed memory is distributed between clusters of processors. These are referred to as non-uniform memory access (NUMA) computers. Some of them are fully equal in scale and cost to the large mainframes of an earlier period. How much longer such very large shared systems will continue to be purchased remains to be seen.

The Speed of Parallel Systems

A problem with all parallel systems is how to assess the speeding up that parallelism gives. It is easy to compare the time taken for a solution using all available processors with the time on the same system with all processors but one disabled. However, one should really compare the time taken on the parallel system with the time taken on an advanced workstation based on a processor identical to those used in the parallel system. Such a workstation will be faster than the multiprocessor system working with only one processor since its memory circuits will be optimized for a single processor and there will be no switch, crossbar or otherwise, to get in the way.

Faster workstations are coming on the market all the time and, compared with them, large parallel systems will show up less well as time goes on, unless the owners of the latter can can upgrade them. It is not easy to do this for both technical and economic reasons. I would expect that, as a large shared multiprocessor system ages, users who were very satisfied with it when it was new will desert it in favor of an advanced workstation of their own. I would be interested in evidence, if any is available, that this is actually happening.

Instruction Level Parallelism

Advances in the effective speed of CMOS uniprocessors have been obtained partly by shrinkage and partly by increasing the degree of instruction level parallelism within the processor. This kind of parallelism is entirely invisible to the programmer and is thus quite different from the parallelism offered by multiprocessor systems.

The development of instruction level parallelism started with pipelining. a now long-established technique by which several instructions are in passage through the processor at any given time, each in a different stage of execution. In a simple pipelined computer the instructions are executed in the order in which they are written.

Modern superscalar processors are capable of executing instructions in parallel and even out of order when that is legitimate. In early designs, the instructions to be executed in parallel had to be identified by the compiler. In later designs, wired-in algorithms enable the compiler's efforts to be supplemented by decisions taken at run-time.

Now designs are pushing towards even more internal parallelism. They provide multiple floating-point units and even multiple integer pipelines. However, all the integer pipelines must share the same register file.

Processors vary in how many instructions they can dispatch in the same cycle and the number is increasing. The present state of the art allows four. Unfortunately, the nature of the program does not always allow advantage to be taken of even this degree of potential parallelism, and the average speeding up may be nearer two than four.

There are other ways of improving performance. Branch prediction gives improved processor performance by reducing the number of occasions on which the pipeline is disrupted as a result of a jump in the flow of control. I have been impressed by the remarkable degree of success that has been achieved with systems of branch prediction. Along with branch prediction goes speculative execution, whereby the processor can continue in the direction of the predicted branch before the outcome of the branch is known. Speculative execution puts an extra strain on the memory access system—already a factor limiting performance—since memory cycles must be devoted to non-taken branches. For this reason, there are limits to the extent to which speculative execution can be taken,

It is becoming much harder to think of architectural devices that will lead to improved performance and it seems likely that we are running out of ideas. However, there remains one device—the use of predicated instructions—that has not yet been put to large scale test.

Predicated Instructions

Predicated instructions are not in themselves new, but work carried out in recent years at the Center for High Performance Computing, University of Illinois, under the leadership of Wen-mei Hwu, has focused attention on their potential in microprocessor design and has advanced the art of writing compilers capable of exploiting it. This followed earlier work by Bob Rau and his colleagues at Cyndrome and at Hewlett Packard Laboratories.

A succinct account of the use of predicated instructions will be found in a paper by Hwu which appeared in the January 1998 issue of Computer. In the system he describes, each instruction contains bits which point to an entry in a register file which contains the value—*true* or *false*—of a predicate. This register file is separate from the main register file. Values in it can be set by the program.

A predicated instruction goes through the execution unit in the ordinary way. However, unless the predicate value is set to *true* the instruction is cut short and prevented from altering the state of the machine. Such an instruction is said to be disabled; otherwise the instruction is enabled.

Predicated instructions make it possible to compile *if then else* and similar statements without using jump instructions, thus avoiding the disruption of the pipeline that these entail. Instructions for both branches are included in the same basic block; at run time instructions for the taken branch are enabled and those for the non-taken branch are disabled. The disabled instructions take time to pass through the pipeline, but if the branch is a short one, that time is less that the time that would be lost if the pipeline were disrupted. The decision when to use predicated instructions instead of branches must be taken by the compiler. This raises non-trivial and critical issues for the compiler writer.

It appears that the use of predicated instructions have the potentiaal of exposing more instruction level parallelism than even the most advanced super-scaler architectures are capable of handling. If full advantage is to be taken of predicated instructions, means of increasing this parallelism must be sought. I return to this below.

Predicated instructions are longer than regular instructions, because of the bits needed to address the predicate file. In consequence, it is no longer possible for each instruction to be contained in 32 bits. The consequences of this are not easily evaluated and may be serious. The balance between instruction length and word length has always been a crucial, if somewhat mysterious, factor in determining the efficiency of an architecture.

The obvious way to increase the parallelism available in the processor is to provide more floating-point execution units and more integer pipelines. Since the latter must all make use of the same set of registers, the designer may feel pressure to increase the number of registers in the register file beyond the usual 32. Doing this increases the length of an instruction, but one can argue that this creates no additional problem since predicated instructions already need more bits than regular instructions. However, the additional electrical loading on the circuits may force a reduction in clock rate and thus prove counter-productive.

The last point is a general point. As processors become more complex, there is a danger that the increased complexity will defeat its own ends by forcing a reduction in the clock rate. However, it must be said that the implementers of modern chips have been surprisingly successful in keeping the clock rate up in spite of increasing complexity, and the danger is perhaps not as great as it was formerly.

Merced

You will expect me to say something about Merced, which sets out to exploit, among other things, the potential of predicated instructions. Merced is an Intel project and is stated to be the first implementation of a new instruction set architecture—IA-64—that Intel have developed jointly with Hewlett Packard. There is an air of mystery associated with Merced and, while this adds some spice, it makes it hard to comment fairly.

Intel have made at least one attempt to introduce a processor with a RISC instruction set and suitable for use in a workstation. I am thinking of the i860 processor developed around 1990. This design had good architectural features and received a first class implementation by Intel's process engineers. Unfortunately, it also had features that rendered it unattractive to designers of workstations and, as a result, it never found a place in that market.

With Merced, Intel's aim appears to be to implement a processor based on the IA-64 instruction set which will outperform RISC processors currently on the market; at the same time it will run x86 code and PA-RISC code at competitive speed. At first sight it might be thought that, as far as achieving good IA-64 performance is concerned, the Merced group have shot themselves in the foot at the outset by requiring this backward compatibility. Not enough information has been made public about how they propose to achieve it for useful comment to be made.

It is certainly true that the Merced chip will be a large and complex one. No doubt Intel are betting on this not mattering, since the transistor density will be much higher on chips of the future than on the chips of today. They had originally hoped to bring out an implementation with 0.18 micron technology ahead of competing RISC machines, but delays have occurred and it seems that this will no longer be possible.

It was a bold course to take work on predicated instructions straight from the research phase and aim to produce a competitive product, without waiting for the implementation of an experimental prototype. However, this necessarily leaves the project team somewhat exposed. They are in the unenviable position of having both hardware and software projects that must meet their deadlines.

I would expect that only those users interested in the highest level of performance will be attracted to Merced and its successors. Others will be happy with simpler chips that accept x86 code only. My impression is that the Merced team would be satisfied if they achieved a speed gain of even 30% compared with the best current workstations and mightily pleased if it were more.

Development of x86 chips will not stand still. We may expect to see advances made in their internal architecture not only by Intel's competitors, but also by Intel itself. Intel has huge resources available and no doubt has other irons in the fire apart from Merced.

To say more, one would need to know more about where Intel places Merced from the market point of view and how much their business plan has been affected by the unforeseen delays that have occurred. On the longer term, it is fair to say that even if predicated instructions prove the great success that their enthusiasts predict—something that is not to be taken for granted—the IA-64 instruction set will face most of the problems and challenges that any new instruction set must face. It is difficult to believe that all processors of the future will use predicated instructions.

The Semiconductor Outlook

The shrinkage of CMOS has long proceeded in accordance with Moore's Law. This originated in an accurate prediction of the rate at which shrinkage would take place that Gordon Moore made in 1965 with a time period of some ten years in mind. A similar rate of shrinkage has since been maintained as a result of industrial consensus.

A great surprise has been the way in which optical lithography has met the demands made on it as feature sizes have been progressively reduced. This has been achieved by using light of shorter and shorter wavelength, and developing lenses of large aperture weighing as much as 1500 lbs.

Use is now being made of light with a wavelength of 248 nanometers.* This works well for feature sizes of 0.25 micron. At a meeting held in March 1997 at the Cavendish Laboratory to mark the centennial of the electron, Gordon Moore said that it might be just possible to reach feature sizes of 0.18 micron with this wavelength, although he expected that a wavelength of 193 nanometers would have come into use by then. He added that the engineers concerned anticipated that, by using light of the latter wavelength and all the optical tricks that they could think of—including phase masks—they would be able to get down to a feature size of 0.13 micron. Note that it is possible to reach feature sizes smaller than the wavelength of the light used.

*It is customary to express wavelengths of light in nanometers and feature sizes in microns. A micron is the same as a micrometer, so that there are 1000 nanometers in a micron.

Beyond 0.13 micron innovation will be called for. It is not possible simply to use light of a shorter wavelength, since by the time a short enough wavelength has been reached, all known materials are opaque and lenses cannot therefore be made. If we move further—into the X-ray part of the spectrum—materials become transparent again, but the focusing of X-rays to make images is not possible. In spite of this, there is no reason in principle why X-rays should not be used for making chips, and the industry has put much effort into developing a practical system. However, Moore said that there were severe difficulties with the mask structures needed, and that many people, including himself, doubted whether the use of conventional X-rays would be economically viable.

Attention is now being turned to light of wavlength around 130 nanameters, which is on the border between ultraviolet and X-rays, as those terms are conventionally understood. Light of this wavelength can be called either soft X-rays or extreme ultraviolet; Moore remarked that the latter term is now preferred since X-rays have acquired a bad name. At 130 nanometers, it is just possible to form images, but by reflection instead of by refraction. The precision called for is frightening; the figuring of the surfaces, including the masks, has to be better than that of the Hubble telescope. In spite of the difficulties, this approach seemed to Moore to offer the best chance of success.

It remains to be seen what will happen. At present, shrinkage is going at a great rate, but we may well see a slowing down when the forced change to a new form of lithography takes place. In any case, we will not then be far from the point—the CMOS end-point—at which there will no longer be sufficient electrons for further shrinkage to give improved performance. This will be the end of the road for CMOS as we know it. We may expect to reach it with feature sizes of 0.1 or 0.05 micron.

There is no obvious follow-on for CMOS. Semiconductor structures, in which the presence or absence of a single electron makes the difference between a 0 and a 1, are being studied and progress is being made. However, these structures are still at the stage at which they are of interest primarily to physicists and it is hard to see practical devices emerging before 20 years. But experiment is pushing ahead of current theory and, as time goes on, there may well be surprises.

Subtle Changes in the Underlying Climate of Opinion

We are in the middle of many exciting developments. Nevertheless, I have a strong feeling that subtle changes are going on in the underlying climate of opinion.

It was the development in the late 1980s of effective benchmarks that made it possible to make comparative measurements of processor performance. Up to that time computers had been sold on the performance of the whole system, with much importance being attached to the quality of vendor support. No one enquired very much about how the various vendors' processors compared, and their differing instruction sets tended to be judged like pet dogs for their appearance and pretty points, not like racehorses for their speed.

Benchmarking came just as the increasing power of computers had made it possible to evaluate processor architectures without actually building the hardware. As a result, people began to attach importance to relatively small differences in the speed of processors. A factor of two was regarded as enormous. I think that the market will become less sensitive to differences in processor speeds of this order.

One trend may turn into a major concern. The penalty for cache misses is becoming greater as the gap in speed between cache and main memory gets wider. On a 500 Mhz Alpha workstation, a cache miss now costs 128 cycles, from miss to usage. The effect is especially felt with problems whose working sets do not fit into secondary cache. Users are beginning to find the performance of superscalar workstations disappointing on such problems. To make matters worse, more problems with large working sets are being tackled as users become more ambitious. With such problems, a cache can actually get in the way. The Cray 1 had no cache for that very good reason. It is perhaps because performance now depends to an increasing extent on the problem area that the market is beginning to emphasize clock rate as much as benchmarks when comparing processors.

I feel that the widening of the gap by a factor of four—which can be seen coming—will radically disturb the assumptions on which we work. Unless something can be done about the memory gap, we may have to revise our present attitudes fundamentally. At least, one can foresee severe headaches for workstation designers and also for benchmark designers.

A change that we would all like to see would be for more attention to be paid to the efficiency and user friendliness of the system software.

Acknowledgments

I would like to express my thanks to the following colleagues and friends who kindly read an early draft of this paper in whole or in part, and made helpful comments: Peter Robinson, Ruby Lee, Wen-mei Hwu, Don Gaubatz, Keith Diefendorf, and Lance Berc.