PARALLEL COMPUTING: THE STORY OF THE ELVES AND THE SHOEMAKER

INAUGURAL LECTURE

DELIVERED AT RHODES UNIVERSITY

on 23 March 1994

by

PROFESSOR PETER G. CLAYTON

BSc (Hons) MSc PhD (Rhodes) FICS

GRAHAMSTOWN

RHODES UNIVERSITY

1994

PARALLEL COMPUTING: THE STORY OF THE ELVES AND THE SHOEMAKER

INAUGURAL LECTURE
DELIVERED AT RHODES UNIVERSITY
on 23 March 1994

by

PROFESSOR PETER G. CLAYTON

BSc (Hons) MSc PhD (Rhodes) FICS

GRAHAMSTOWN
RHODES UNIVERSITY
1994

First published in 1994
by Rhodes University
Grahamstown
South Africa

Peter G Clayton
Parallel computing:
The story of the elves and the shoemaker

ISBN: 0-86810-274-1

Mr Vice-Chancellor, Colleagues, Ladies and Gentleman:

My first and pleasant duty is to pay tribute to my predecessors in the Department of Computer Science, all of whom have personally influenced my thinking on the subject.

The late Professor Rolf Braae of the Applied Mathematics Department had the vision to introduce Computer Science as a major subject at Rhodes University in 1970. Those of you who remember Professor Braae will probably recall him, as I do, as a warm person who had an appropriate anecdote to go with any occasion. I would like to take a few moments to recount one of his anecdotes. Before Professor Braae came to Rhodes University, he was Professor of Electrical Engineering at the University of Stellenbosch. It so happened that the Professor of Botany lived next door to him at that time, and he leaned over the fence one day to ask Rolf Braae whether he might be able to repair an electric razor. The reply he received was something like this: "As the professor of Electrical Engineering, I certainly know something about electrical appliances, and as the professor of Botany, you surely know something about gardens. So, if you would like to come around to my house, then I will repair your razor while you mow my lawn." The undergraduate computing laboratory in the Department of Computer Science has been named in memory of Professor Rolf Braae.

Professor Howard Williams was the first Professor of Computer Science when a separate Computer Science Department was established in 1980. He is now Research Director of Computer Science and Electrical Engineering at Herriot Watt University in Edinburgh, but he has maintained a personal and professional interest in Rhodes University, and will be the recipient of a DSc degree from Rhodes at a forthcoming graduation ceremony. In the latter 1970's, Howard

Williams was the only staff member at Rhodes University whose time was wholly devoted to the teaching of Computer Science, and I marvel now at the size of his teaching load at that time. When I was nearing the end of my bachelor's degree, Professor Williams told me that if I had any sense, I would resist the financial attractions of the job market (as it was then) and stay on for postgraduate study. So, he was a direct influence in one of the major career choices that brought me to where I stand tonight.

The headship of Computer Science rotated for a few years (until 1987) between the current head, Professor Patrick Terry, and Professor Denis Riordan, who is now at the Technical University of Nova Scotia. Professor Riordan was the only Rhodes Professor of Computer Science who was not a teacher of mine. Nonetheless, I learned much from him, particularly as a young member of staff in his department. Professor Terry, on the other hand, did teach me formally. The first lecture I attended at University was given by a new young lecturer who introduced himself as Dr Terry. He didn't sing at that lecture*, but still managed to keep us extremely entertained by building a mathematical model of the mating habits of penguins in the Antarctic. That was the first meeting in what for me has become a long and rewarding association, and the influence he has had over my career choices has been considerable.

I would like to use this opportunity to acknowledge the intellectual and material contribution to my work of my colleagues and students. Many of the examples that I shall refer to later in this lecture are due to them.

---

*Three weeks before this inaugural lecture was delivered, Professor Terry delivered a public lecture (to mark his receipt of the Vice-Chancellor's Distinguished Teaching Award) at which he sang to the audience.

I would also like to pay a special tribute to my family: to my parents whose guidance and example shaped my value system; to my wife, Louisa, who has been more than a support - she has been a partner in building my career; and to my two boys who have encouraged me with their enthusiasm for what I do professionally.

To introduce my topic, I would like to tell you a story.

The Elves and the Shoemaker*

There was once a shoemaker who was very honest and worked very hard; but still he could not earn enough to live upon, and at last all he had in the world was gone, except just enough leather to make one more pair of shoes. By the time he had painstakingly cut them out, the daylight was gone and he went to bed exhausted from his labours. In the morning, after he had said his prayers, he sat down to complete his work, to find, to his delight, that the shoes stood already completed upon the table. The good man did not know what to make this strange event; but, being in urgent need of a meal, he went out and sold the shoes, and, after satisfying his hunger, had money over to buy leather enough for two more pairs of shoes. In the evening, he laid the leather out apprehensively, and went to bed early, and again, when he got up in the morning, the work was already done for him. He quickly found two

*Adapted from the folk tale collected by the Brothers Grimm.

buyers, eager to reward him handsomely for his goods, and so he bought leather enough for four pairs more. Again he laid out the leather and found the work finished in the morning as before; and so it went on, night after night; and so his fortune grew exponentially with his output, until he was extremely wealthy, and more than a little curious about the mysterious source of his fortune.

One evening, he said to himself: "I will sit up and keep watch tonight, that I may see the source of this lucrative productivity increase in my business." So he left a lamp burning dimly, and hid himself in the corner of the room to observe. On the stroke of midnight, there appeared a host of little elves, who worked away furiously with their little fingers, until the job was quite finished, and the shoes stood ready for use upon the table.

Of course, what the shoemaker had stumbled across was the technique now known as parallel processing, which is the simultaneous working together of many parts for a common purpose. By himself, he was unable to complete even a single pair of shoes in one day, but the combined effort of the elves produced a huge capacity for this sort of work.

Doing things in parallel is a familiar concept to humans. At an individual level, we are designed to perform several tasks simultaneously. Even as you sit still in your seat, you are controlling the muscles of your body to maintain your balance, your eyes are focusing on images and passing them to your brain for processing, your ears are receiving signals, you are sorting them into word patterns and trying

to make semantic sense of them.

At an interpersonal level, we are used to cooperating with other humans to achieve a common goal. This form of parallel activity, combining efforts of many, is the acknowledged method of accomplishing a substantial task in a reasonable amount of time; sometimes it is the only way of completing a task at all. Indeed, throughout human problem-solving history, parallel approaches have been used to overcome complex engineering and organisational challenges. Great structures are built, large organisations are managed, and formidable enemies are conquered not by individuals, but by team work. So, in everyday life, parallel activity is the norm; individual sequential activity is the limiting exception.

Well, if the notion of parallel activity is so obviously commonplace, how does it qualify as a field of specialist endeavour? The answer to this question lies in the fairy tale abstraction with which humans tend to regard parallel domains. The story of the elves and the shoemaker provides an appealing archetype for this kind of abstraction. The problem of making shoes is simply scaled up in a linear fashion, without any consideration of the distribution of responsibilities, the coordination of different tasks, the dividing up of raw materials, the sharing of tools, or the finite size of the workspace. These details are defined away by the nature of the workers: elves are supernatural beings with magical properties.

Arthur C. Clarke, the popular science commentator and science fiction writer, once remarked that "any sufficiently advanced technology is indistinguishable from magic", which is a fine point of view for anyone except the one charged with the responsibility of creating the advanced technology. In parallel problem domains, moving from an abstract idea to a concrete implementation requires substantial effort, mainly in the area of coordinating activities that require the

cooperation of more than one participant.

When independent entities act simultaneously, all contributing to a common objective, behavioural complexity grows very quickly. For example, one entity might be capable of cycling through 6 steps in the solution of a problem. It would thus have 6 unique states. Two such entities working together would have 6^2=36 unique states corresponding to the number of permutations of their individual events. 20 such entities working on the same problem would be capable of $6^{20} \approx$ 3.7 million-billion (3.7×10^{15}) unique states. In systems of this degree of complexity, ensuring that desirable combinations of events occur timeously, and that undesirable combinations of events never occur, requires specific coordination strategies.

Just as the word *algorithm** is used to describe the set of rules and instructions to be followed in the problem solving procedure of sequential computer programs, it is also used to describe the solution to a parallel problem. (Algorithm in a computer context has roughly the same meaning as the word *recipe* does in a cooking context.) An integral part of a parallel algorithm is its coordination strategy.

The coordination strategy for every parallel application needs to be designed specifically to cater for the needs of that particular problem area. Fortunately, though, several broad classes of coordination strategy exist which are known to be useful across a range of problems. An example of such an algorithmic class in everyday life is a factory assembly line. All assembly lines comprise a *pipeline* of activities, and a conveyor belt (or similar arrangement) to

---

*After the 9th century Persian mathematician *al-Khwàrizmi*.

communicate data and state information through the system. Individual assembly lines vary in the function and duration of their activity stages, but all problems that are solved in this way need to have the common characteristic that a stream of individual items is able to be developed in stages, to produce a stream of products. The coordination strategy that simplifies the assembly line is that items move forward in lock step, and that the speed of the line is constrained by the speed of its slowest activity.

The activity of giving a lecture is an example of another algorithmic class, one commonly known as *farming*. The lecturer farms out some opinions and facts to the audience, and they assess what she has to say, in parallel. At the end of the lecture, the lecturer might gather in the responses of the audience, and tabulate them into an overall result. This would have been far more efficient than if she had approached each member of the audience individually, in sequence, with what she had to say. The coordination of individual activities in a lecture situation is achieved through the protocol that has been agreed upon before the start of the lecture; in this case it is a very simple strategy, the lecturer speaks and the audience listens. To implement this coordinated activity, all participants make use of the a shared resource, the air. This carries the sound waves from the lecturers mouth to the ears of the audience, which works well for the lecture situation, but would not necessarily work for a different class of problems.

Imagine if the lecturer were to set each member of the audience the task of swapping life stories with the person in the room whose birthday was closest to theirs. If they were all to keep their original seating positions, we might end up with the undignified situation of some people at the front of the hall having to yell at partners in the back row, while others on the left of the room attempted to shout them down to communicate with their partners on the right, and so on.

There would certainly be a lot of interference and communication breakdown. The seating arrangement would not be a satisfactory configuration for solving the problem. It might be more appropriate to allow the audience to move positions and form functional groupings, and to use the air in a localised way to whisper to each other. Alternatively, if the swapping of life stories was only a portion of the problem, and the rest of the solution benefitted in some way from the original seating arrangement, then it might make sense for members of the audience to write notes and get intermediate people to pass them on.

So, it should be noted that there is no single recipe for solving problems in parallel. The nature, organisational layout, and communications interfaces of a parallel system constrain the range of problems which can be solved using it. In terms of the examples I have offered, the coordination approach of the assembly line would not work well in a lecture situation, and vice versa.

Parallel Computing [3]

So far, I have said a fair amount about the general concept of working in parallel, but not much specifically about computing. Parallel computing is about programs that are intended for execution on many processors simultaneously. Each processing element works on a piece of the problem, and their computations contribute towards a common purpose. The analysis of parallel software is a special instance of the broader art and science of analyzing coordinated systems in general.

The most frequent use of parallel computing is to enhance the computational performance of a system. No matter how fast a computer processor might be, two of them working together should be faster, and more than two should be

even faster, and, even then, they will not be able to satisfy all current computational needs.

There are unsolved problems in science and engineering that are able to consume almost any amount of computer resources. Here are some examples:

Quantum Chemistry in which Schrodinger's equation has only been fully solved for the simplest cases;

biology and biochemistry where the analysis of DNA structures is immensely complex;

fields which produce immense amounts of data such as remote sensing;

and areas such as computational fluid dynamics, materials science, climatology, and cosmology, in which increasing the resolution of finite element modelling provides more precise results.

The appetite for computational power is not restricted to science and engineering. The amount of data required by decision makers in modern commerce and industry has mushroomed in recent years, and parallel processing has been applied to the implementation of data base searches and decision support systems. For the individual computer user, increased computing power supports more sophisticated user interfaces, which are designed to communicate in a manner more intuitive to humans. This extends the computer's accessibility to less specialised users. A current commercial example of this trend is the graphical user interface. More advanced laboratory examples are speech synthesis, handwriting recognition, and virtual reality*.

---

*A head-mounted stereo display unit, along with data gloves and tracking devices, can be applied in the computer generation of an artificial world in which the user is able to move around. This creates a virtual reality for the user. The stereo display presentation is changed as the head is moved, and the data gloves are used to manipulate computer-generated objects.

Anything that can be done in a parallel environment can also be done in a sequential environment, it just takes longer. A prominent factor which drives the pursuit of parallel computing is the need for what is known as *real time* response. The term *real time* implies a feedback situation in which a computer system produces results sufficiently promptly to allow its user or environment to take immediate action, which could take the form of further, more informed queries. For example, if a medical practitioner were to send you to see a radiologist with instructions to return in a few days time with an ultrasound scan, this would not be a real time activity. The elapsed time between consultation, scan, and follow-up consultation would decouple the activities of the scan and the diagnosis. On the other hand, if the medical practitioner were to pull out some ultrasound equipment and take a scan during the consultation, and make a decision on the basis of the scan immediately, such as to perform another scan from a slightly different angle to home in on a possible problem, we would then say that the scan was being used in real time to make a diagnosis.

The time lag which can be tolerated between a query and a result in a real time system depends entirely on the nature of the application. A computer animation, for example, whose composition can be determined by the viewer, requires that each still picture in the animation sequence should be rendered in not more than $1/25^{th}$ of a second. To give the impression of smooth motion, at least 25 separate still pictures, called frames, are required by the human eye every second. With currently available technology, this is a serious challenge, and it is one of the primary considerations in the development of virtual reality systems. One of the most important general uses of computer systems is to assist humans in extracting information from sets of data, and one of the most successful techniques for representing the information contained in complex data sets is through the use of graphical visualisation. Parallel methods are widely used in graphical

visualisation, to render pictures sufficiently quickly so as to allow the user to alter the angle of view and zoom in on features of interest. Of course, large amounts of computational power are required to produce a real time responses from any complex data, or from the application of any complex process.

However, computational speed is not the only reason for making use of parallel systems. A second physical attribute of parallel computing is its improved performance in terms of reliability. Due to inherent redundancy in the processing hardware, life critical systems or systems which cannot easily be accessed for maintenance can be designed to be robust. If one processing element malfunctions, the computation can proceed on another one. This approach is taken in the design of such systems as artificial organs which are implanted inside patients, and satellites which are placed in space.

A further attractive attribute of parallel designs, this one logical rather than physical, is that they provide a sensible model of the real world. I have already pointed to the fact that, in every day life, parallelism is the norm. When the solution to a parallel problem is formulated in terms of a sequential program, the mapping of the solution onto the sequential platform is unnatural, and therefore it is error prone, difficult to maintain, and unreliable. This is one of the reasons* why it is increasingly common for single processor computer systems to pretend to be parallel systems. A pseudo-parallel environment is provided by switching execution between entities capable of executing simultaneously. To distinguish these from genuine parallel systems, they are known as *concurrent* or *multi-tasking* systems. The UNIX operating system and the MS-Windows environment

*Other reasons for this approach to single processor system design is that it decreases the proportion of time that a processor might be idle waiting for the current task to complete a communication, and it facilitates multiple user access to the computer.

are common examples. MS-DOS, the operating system currently used on most of our personal computers, does not fall into this category, and it is the lack of this facility, more than any other, that limits its usefulness to us in the future.

The architecture of early and current electronic computers can be traced back to the concepts outlined by John von Neumann and others in a paper published in 1946 [2], in which the ideas of a serial computing machine, subsequently known as the *von Neumann Architecture*, originated. The vast majority of computers available today are in this category. When one considers the immense cost of computer processors in the 1950s, '60s and '70s, it comes as no surprise that conventional computing developed around sequential computers with single processing elements. The sheer cost of computer processors made stringing them together in parallel prohibitive.

All this changed in the early 1980s with the appearance of VLSI (very large scale integrated) microprocessing chips. The manufacturing process had an inherent economy of scale, which drove down the prices of computing devices by orders of magnitude. Personal computers appeared at the same time, also driven by the microchip technology.

By the mid 1980s, microchip technology had made it possible to construct a powerful microcomputer with memory, processor, and communications on a single device. The ability to link such devices to each other in arbitrary topologies encouraged the construction of high-speed parallel processing systems at relatively modest costs. One such computing device which is widely used as a building block for parallel systems is the *transputer* [8]. The name *transputer* has been made up from the words *TRANSistor and comPUTER*, so named because it enables whole computers to be used as discrete components to build massively

parallel computer systems, following the more conventional practice of using transistors as circuit building blocks.

Software

But there is a problem: there has been no breakthrough in software technology for parallel computing that matches the developments in microchip technology. Whereas sequential programming notations have matured over a period of forty years into a range of useful and powerful notations, parallel programming, the more complex of the two, has had only ten years of effort and experience in this direction.

The most powerful and the best understood notations in the parallel programming world are extensions of sequential programming notations, which rely on physical shared memory locations for communicating between their constituent parts. This poses a particular problem for the more affordable kinds of parallel systems, in which processors have no memory in common and all communication is done by sending messages across networks. Because of this dilemma, parallel computers have developed in two directions: the shared-memory multiprocessor and the multicomputer.

The shared-memory multiprocessor comprises a small number of processing elements connected to a common pool of memory. These computers handle information sharing quite easily, but suffer from contention which arises when accessing the common memory, which is a shared resource. To limit the amount of memory contention to a manageable level, only small numbers of processors are connected together in this way. This is usually compensated for by making the processors more powerful, and so we can think of this as a *herd of elephants* approach. Shared-memory multiprocessors are typically expensive machines.

In the multicomputer, each processing element has its own memory, and there is no shared resource to impose an upper limit on the number of processors which can be connected together. Communication between processors occurs through a high speed network. Since the number of processors is not constrained, we can achieve the same (or greater) computing power as the *herd of elephants* approach by connecting together massive numbers of less powerful processing elements. We might call this the *army of ants* approach. The transputer devices described earlier fall into this class. Multicomputers can be built to be cheaper and more powerful than shared-memory multiprocessors or traditional single-processor mainframes, but are proving to be a lot more difficult to programme. Though much effort is currently being invested in the development of effective programming notations for these computers, the problems of scaling computation up to large numbers of processors, and of communicating via message passing networks, remain challenging.

Typically, dividing up the problem into smaller problems to be solved independently is the least demanding aspect of designing a parallel algorithm, the tricky part is getting the right data to the right place at the right time, so as to keep all processing elements busy. This requires innovative coordination techniques, and a suitably large data set.

The issue of the scale of parallel activity (the large data set) is known as the Amdahl effect [1], after Gene Amdahl an early computer architect, and is a much studied area of parallel processing. Put simply, it makes the rather obvious claim that if you have many parallel participants, you will need to have a lot of work to justify their presence. The story of the Elves and the Shoemaker uses a reasonable scale of operation. It does not suggest, for example, that 10 elves could make a single shoe in 1/10 of the time that it would take one elf, it simply

states that a lot of elves could make a lot of shoes in one night.

The problem of a communications bottleneck between processors is often solved by migrating a parallel algorithm down onto dedicated VLSI circuitry. This is facilitated by the existence of programming language to silicon compilers. Where this option exists, the algorithm is the important product of the programming effort, not the prototype implementation. This is becoming a fairly common practice [5].

I will illustrate this with an everyday example. If there was a need for one to travel between one city and another more quickly, there are two ways in which this could be achieved. The first might be to find a faster means of transport; but, if the fastest feasible means of transport was already being used, then the second method would be to move the two cities closer together. This might not sound like a feasible solution in geography, but it is exactly what is done in computer circuit design. As a case in point, it is well known that the speed of the microprocessor doubles approximately every two years. This is not as a result of increasing the speed with which computer circuitry operates. Over the last 14 years, circuit switching speeds have increased by a factor of about 10, whereas overall execution speeds have increased by a factor of about 128. The additional speedup is due to the incorporation of parallel processing principles in the microcomputer itself; modern microprocessors are able to execute several instructions simultaneously in what are known as *superscalar* architectures.

Formal methods

But, of course, computer systems often have errors in them. People in the computer programming community are fond of using that very American saying "garbage in - garbage out", which is a less eloquent version of the Biblical saying

"you will reap what you sow". If you sow garbage at a computer, you will reap garbage, probably tenfold. The exceptional thing about this (as those of us who have tried to query our electricity account or our bank statement will know), is that this garbage, once it has passed through a computer, is somehow ennobled, and no one should dare criticise it.

Complex parallel computations propagate errors in complex ways, and the methods of trial and error testing, or *hacking*, that is prevalent amongst the programmers of sequential computers is hopelessly inadequate. The number of different permutations of actions that occur in parallel programs rises exponentially with the number of processes and the complexity of each process (this is called a state space explosion), and it becomes impossible to design empirical tests on millions of different execution paths. This is a sobering effect that has driven Computer Scientists to turn to their mother discipline, which is Mathematics, for a solution.

A number of mathematically rigorous notations have been developed which support formal reasoning about parallel programs. These notations are supported by computer software systems which act as workbenches to facilitate the reasoning process. Commonly used examples of such notations are:

Hoare's notation for Communicating Sequential Processes (CSP) [4],
Milner's Calculus of Communicating Systems (CCS) [6], and
extended forms of the Petri Net [7].

Bringing the precise mathematical meaning of the system to the fore with the aid of a mathematical notation yields two avenues for profit. Firstly, a formal specification acts as a single, reliable reference for all involved with system development, maintenance, and use. Secondly, it provides a secure mathematical

foundation for reasoning about properties of the program, for detecting and avoiding abstruse errors, and for verifying the correctness of the design.

Regrettably, the degree of rigour required for a detailed formal analysis is typically tedious for all but the most modest of systems, and implementors are frequently unfamiliar with the mathematical methods required. All the same, formal verification methods are no longer the exclusive property of academic experimentalists, and are being used increasingly for commercial purposes. While it is often not practical to apply formal verification methods to complete hardware or software systems, they are being employed in the development of sub-assemblies in which the complexity, novelty, or subtlety of the design justifies the effort required for formal analysis.

These formal notations are new areas of mathematics, and it is my hope that more of them will find their way into mathematics curricula so that undergraduate mathematics programmes will eventually provide as much support to the computing sciences as they currently do to the natural sciences.

Parallel computing in developing countries

As we pass through the current transition phase in our country, it is appropriate the we reassess the relevance of what we do with our resources. Parallel processing research can be an expensive pastime, and it might be argued that it is better left to wealthy Western Universities to pursue.

One of the problems in developing countries is that finance is seldom available for purchasing special purpose equipment. I have stated that VLSI technology has made the construction of high-speed parallel computers relatively cheap. The operative word here is *relatively*; there are many institutions in South Africa

which could benefit from high speed computing, but which could not afford special purpose equipment. However, most universities and businesses already have large numbers of personal computers and workstations, connected by low speed networks. The combined computing power of these networks could constitute a powerful parallel computer during periods when the computers are not being used for normal work. Computer networks are, perhaps, an even more important basis for parallelism in our kind of economy than having many processing elements packed into a single box.

However, network based parallel processing faces yet more problems. On the technical side, the networks are set up for purposes other than high speed computing, and factors such as topology and communication speed cannot be altered to suite parallel computations. On the personal side, the users of computers purchased for particular purposes are usually reluctant to have remote users reducing their response time.

At Rhodes University, we have used networks of computers for a number of parallel computing projects. The most notable has been a platform for accomplishing what is known as *adaptive* parallelism. Here the parallel program is designed to adapt to the shape and size of the available network, and to survive unreliable processing elements. If a part of the computation is being performed on a computer belonging to another party, it will regularly monitor the host computer, and will *back off* and complete the computation elsewhere when the rightful owner of that computer wishes to make use of its full resources. This system is able to run on networks of heterogeneous computers, and effectively uses up spare processor cycles during large background runs. We have had this system running on up to forty Sun workstations and IBM PCs simultaneously, and have used it for a range of parallel applications, including graphical

visualisation and parallel database searches.

Future breakthroughs for program development

In discussing some of the benefits and challenges of parallel computing, I have concentrated on what is possible right now, albeit only in the experimental laboratory in some instances. I hope I have been able to show that practical parallel computing rests upon two presumptions:

1. the availability of many low-cost, high-speed computers, that are able to share a computational load; we have these in the form of special purpose VLSI (very large scale integrated) processor devises, and in the form of networks of general purpose computers;
2. the existence of programming development tools and strategies to partition problems into smaller components and coordinate their simultaneous execution. These facilities are fairly crude at this stage in the development of economical forms of parallel computing, and finding powerful ways of managing programs which run on multicomputers is a holy grail of current research.

In the last ten years, five new computing approaches moved out of research laboratories to become land mark technologies. I have mentioned four of them in this lecture. The first is parallel computing. The second is the use of mathematically rigorous techniques, or *formal methods*, to aid in the development of software. The potential now exists for proving mathematically that programs are correct. The third is the widespread use of networks to connect individual computers into vast distributed computing facilities, and the fourth is the use of graphics to simplify human interfaces by using more pictures and less text.

The fifth development is object-orientated programming, which has not been

mentioned in this lecture. This is a rapidly maturing descendant of the structured approaches to computer program design that swept through the computer industry in the 1970s.

I believe that future breakthroughs in program development tools for parallel computing will require innovative combinations of these five technologies.

The Elves and the Shoemaker again

To end, I will return to the story of the Elves and the Shoemaker to tell you how it turned out. The ending that the Brothers Grimm put to this story was a happy ever after form of closure, a fantasy contrived to avoid upsetting the children. In the reality of human nature, the story ends somewhat differently.

> *The shoemaker soon became accustomed to the trappings of luxury, and greed overtook his better judgement. He began to look for ways to organise his workforce to make his wealth grow even faster, and decided to apply stricter control over the elves' hours of work. In the face of the shoemaker's seeming ingratitude, the elves quickly tired of their altruistic endeavours, and they escaped into another fairy tale to become mischievous little hobgoblins. With them vanished the mystical secret of parallel abstraction, which is the reason why the designers and managers of complex systems struggle today to find suitable coordination strategies, appropriate scaling mechanisms, and arbitration methods for shared resources.*
>
> *And as for the shoemaker? Well, he too aspired to a more*

prominent part in a grander story, and, having been unsuccessful at using his riches and his merry disposition to secure the role of Old King Cole, he made the claim that all the achievements in the story had been wholly as a result of his personal coordinating skills, and that he had always been in total control of all that had transpired in his workshop. Despite the ridiculousness of this claim, it was precisely the appropriate credential to land him a leading role in another fairy tale ... that of president of Bophuthatswana[*].

[*] A week before this inaugural lecture was delivered, chaos broke out in the South African homeland of Bophuthatswana over its president's refusal to allow the homeland to participate in elections about to take place in greater South Africa. With his homeland on the brink of civil war and with a totally collapsed administration, the President of Bophuthatswana announced that he was in total control, and would participate in the elections, confident of the support of his people.

References

1. Amdahl, G. (1967), *Validity of the single processor approach to achieving large scale computing capabilities*, in: AFIPS Conference Proceedings 30 (April), Thompson Books, Washington D.C., 483-485.
2. Burks, K., Goldstein, H., and von Neumann, J. (1946), *Preliminary discussion of the logical design of an electronic computing instrument*, Princeton University.
3. Carriero N. and Gelernter D. (1992), *How to write Parallel Programs: A First Course*, MIT Press, Cambridge.
4. Hoare, C.A.R. (1985), *Communicating Sequential Processes*, Prentice-Hall, Englewood Cliffs, N.J.
5. Hoare, C.A.R (1994), *Hardware and Software: The Closing Gap*, Proc. Conference on Programming Languages and System Architectures (March), Zürich, Switzerland.
6. Milner, R. (1989), *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, N.J.
7. Peterson, J.L. (1981), *Petri Net Theory and the Modelling of Systems*, Prentice-Hall, Englewood Cliffs, N.J.
8. Walker, P. (1985), *The Transputer: A Building Block for Parallel Processing*, Byte, **10**(5), 219-235.