Sharp as a Razor: A Queen's Award for the Computing Laboratory

GERAINT JONES

IN a list in a recent Saturday paper you might have chanced on the name of the University amongst the deer farmers and fertiliser manufacturers, the pet food manufacturers and the fast car companies. A Queen's Award for Technological Achievement has been conferred this year on this University's Computing Laboratory, a department hidden away in the faculty of Mathematical Sciences. This is the first time that a department of the University has achieved the distinction of a Queen's Award, and the Computing Laboratory is one of only three university research groups amongst this year's award-winners.

The award recognises a successful collaboration between the laboratory and Inmos Ltd – one-time nationally owned offspring of the white heat of Wilson's technological revolution, now a division of the pan-European microchip manufacturer SGS–Thomson. To over-simplify: they made a microprocessor and we did some mathematics; between us we made sure that the microprocessor would be right, and the great thing is that we did it together.

Since the early days of the company, OUCL and Inmos have been collaborating on the design of programming languages and computers. The laboratory has been advising Inmos on the application of relevant mathematics – the jargon is 'formal methods' – to their work on the design of a programming language and of integrated circuits. Inmos has been a wonderful source of new ideas and new applications to which to bend our techniques, and has been generally keen to try them out.

Inmos' flagship product is the 'transputer', a microprocessor with many of the parts that would normally be needed in addition built into the same single component. It is specifically intended to be good at communicating with other transputers, so is a natural component for building large parallel computers with many processors that can be working on different parts of a problem at the same time. The advantage of such machines is of course speed: the more things that happen at once, the faster they can all be completed. The drawback of such machines is that it has always been thought that they are hard to use and to reason about: too much happening at once, too unpredictable, too complicated to understand.

Fortunately the laboratory's flagship product at about the right time – some ten years ago – was Tony Hoare's Communicating Sequential Processes. This is a scheme for disciplining one's thoughts about systems of independently acting components, whether they be customers and salesmen in a shop, or processors in a parallel computer. From CSP and the concept of the transputer, and from the collaboration between Tony Hoare and David May and their respective colleagues, grew the programming language 'occam'. Yes, it is indeed named after *that* William of Occam, him of the intellectual minimalism: a distinguishing feature of the language is that it goes out of its way to be clear and simple.

Occam is the first commercially available language to take seriously the problem of programming parallel machines. Various pieces of work done in the laboratory, published both in the academic literature and as Inmos' product documentation, support reasoning about programs written in the language. Occam is a simple tool carefully crafted for a task, and having once got the intellectual tool right its every use is less taxing and easier to reason about. The intellectual leverage gained from the simplicity and elegance of occam was instrumental in the success of the work on the transputer design. As well as being used to program computers built from transputers, occam was used in the description of the components of the microprocessor itself.

Perhaps the most tangible success of the co-operation between OUCL and Inmos was the design of an arithmetic unit for the transputer. The two principal actors in this tale are Geoff Barrett and David Shepherd, both of whom came to Oxford as students on the M.Sc. in Computation. At the time the work was done Geoff was in Oxford doing his doctoral research in the Programming Research Group, and David was designing chips for Inmos. Between them they tackled the design of a floating point arithmetic unit for the T800 transputer.

Floating point arithmetic is the sort that may be familiar from electronic calculators, handling both enormous and minuscule numbers by moving – 'floating' – the decimal point to keep all the numbers a reasonable size, and keeping a separate account of where the decimal point should be. Because large scientific computers spend most of their time on just this sort of arithmetic, and because there is just that much extra book-keeping work involved in doing each operation, it is worth adding a second calculating unit to a microprocessor to do the floating point arithmetic. This additional processor – an FPU – is designed just to do floating point arithmetic, and to do it quickly.

The catch is that a circuit like the FPU is complicated and very easy to get wrong. Moreover it can be very expensive to manufacture and install many thousands of instances of a circuit before finding mistakes, as happened to a competitor of Inmos' at about this time: not only does it cost to replace the components and the computers that use them, but it causes an expensive delay in bringing products to market, and damages customers' confidence. One day a mistake may not be found until it leads to someone's death.

Unfortunately traditional engineering techniques are of very little help here.

Every possible different sum that the FPU may be asked to do is a special case, different in some way from all the others. If you are going to check that the design is correct by testing all the special cases you are in for a good long wait: much longer than the universe seems likely to last at any reasonable rate of testing. That means that any testing programme must necessarily be very incomplete.

When the T800 transputer was designed it was decided that in parallel with a conventional product development, supported by months of testing in comparison against other manufacturers' FPUs, a second team would set about a formal development of a unit which would be known to be correct by design. Because machine arithmetic has necessarily to be approximate, there is a question of what 'correct' means here: for example which way to 'round' numbers that cannot be represented exactly, and what to do when numbers get too small or too large. Gradually the computer industry is coming to agree that computer arithmetic should follow a standard laid down in a document sponsored by the Institute of Electrical and Electronics Engineers, known as IEEE Standard 754.

A big first step was to decide exactly what the Standard required. The IEEE document relies in places on the meaning of vague requirements expressed in English. Geoff Barrett's recasting of the Standard into a precise mathematical form is unambiguous and serves as the touchstone for the correctness of the T800 FPU – as well as being a precise explanation to the user of a T800 of what answer to expect from any calculation which it performs. This recast Standard was expressed in a notation called 'Z', developed by the laboratory to be used for just such precise specifications. The laboratory has been successfully using Z as a cloak under which to smuggle mathematics and mathematical techniques into many parts of the computer industry.

The microcode of the T800 FPU – the collection of detailed instructions which the FPU follows when asked to perform a sum – was documented in a restricted subset of occam. David Shepherd then demonstrated that this code was equivalent to a more naturally expressed occam program which had been shown to meet the Z specification. This involved an amount of work comparable to the size of the design, certainly altogether less work than would have been involved in exhaustive checking by testing. It is however not a trivial task, so in doing this work he used a computer program written in the laboratory and called 'the occam transformation system'. The transformation system builds on the simplicity of occam. It is able to prove essentially without human assistance that two programs – like the specification of the FPU and the implementing microcode – have the same effect.

You could hardly have made a better case for our formal methods, because the race went to the formal development. Moreover a number of errors were found in the proposed implementation that had not shown up in months of necessarily very limited testing. (Some of the discrepancies that were found by testing transpired to be mistakes in an implementation by a competitor, against which the design of the T800 was being compared.) Not only is it possible to be much more confident in the accuracy of the finished FPU, but this higher quality implementation was cheaper to design, and was completed an estimated year ahead of what would otherwise have been achievable.

The other people in Oxford working for Bill Roscoe on the occam transformation project (of which the T800 FPU work was a part) who implemented the transformation system were Michael Goldsmith and Tony Cox. With Bill and others they are now involved in a company called Formal Systems, selling just this kind of expertise from an office on the S.T.E.P. Centre on Osney Mead. Bill Roscoe and Tony Hoare are in the Programming Research Group of the Computing Laboratory. Both Geoff Barrett and David Shepherd now work for Inmos Ltd in Bristol, as does David May who is also a Visiting Professor in the department of Engineering Science. Of course many other people in both organisations were involved in this and other collaborations.

The moral of this tale is that formal methods can not only improve quality, but also the timeliness and cost of producing state-of-the-art products. Inmos can be confident that they got it right, and did so on time. The laboratory has also benefited from the opportunity to develop its work on the underlying theory into something which it has confidence will be useful.

This is a perfect example, one of many, of the benefits of a policy conceived and expressed by the late Christopher Strachey when he was the first leader of the Programming Research Group

> 'It has long been my personal view that the separation of practical and theoretical work is artificial and injurious. Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it do not have any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing. One of the central aims of the Programming Research Group as a teaching and research group has been to set up an atmosphere in which this separation cannot happen.'

Sadly, the laboratory has been so successful and has grown so rapidly that we are now spread on four sites about the science area. This work was done when we all lived in each others' pockets and everyone had the opportunity to know what anyone else was doing; now there is probably nobody who even knows who everyone else is, and there is a danger of separated groups polarising into theoreticians and practitioners. We can but hope that the Award will be of some help in attracting support to finance a new building to house the laboratory.

This article appeared in the OXFORD MAGAZINE *№ 59,*
dated FOURTH WEEK, TRINITY TERM, 1990.